Univerzitet u Beogradu

Matematički fakultet

MASTER RAD

Kreiranje Veb aplikacije za podršku SMS saobraćaju

Milica Ćurčić

1111/2011

Beograd, jun 2014.

Mentor: Dr Dušan Tošić
Matematički fakultet, Beograd

Dr Miodrag Živković
Matematički fakultet, Beograd

Članovi Komisije:

Dr Filip Marić
Matematički fakultet, Beograd

Datum odbrane: _____

REZIME

Master rad na temu "Kreiranje Veb aplikacije za podršku SMS saobraćaju" sadrži teorijski deo i deo u kome se opisuje izrada i funkcionalnost same Veb aplikacije.

Teorijski deo obuhvata opis PDO upravljača (PHP Data Objects driver), koji predstavlja pouzdan i konzistentan interfejs za pristup bazama podataka u PHP-u, a na kome se bazira funkcionisanje aplikacije.

Teorijska osnova je praktično primenjena u Veb apikaciji. Aplikacija je osmišljena tako da radi kao inteligentan sistem za praćenje slanja i dostavljenosti SMS poruka generisanih od strane sistema koji sluzi kao kapija[1] u mobilnoj telefoniji. Na osnovu izveštaja iz aplikacije vrši se niz izmena u sistemu koje bi trebalo da dovedu do povećanja procenta dostavljenih poruka.

[1]Engl. *gateway* – Softverski alat koji omogućava slanje i primanje SMS poruka preko digitalnih mobilnih telefonskih mreža.

SADRŽAJ

1. UVOD

PHP je prerastao u veoma popularan Veb programski jezik zbog svoje jednostavnosti i lakog korišćenja. Jedan od ključnih faktora njegovog uspeha je ugrađena mogućnost pristupa mnogim popularnim relacionim sistemima za upravljanje bazama podataka (RSUBP) kao što su MySQL, MSSQL, PostgreSQL, SQLite, i mnogi drugi. Danas se mnoge od postojećih i novonapravljenih Veb aplikacija povezuju sa ovim bazama podataka kako bi proizvele dinamične, RTA[2] veb aplikacije.

U ovom radu biće opisan jedan od najvažnijih dostupnih dodataka PHP-u, počevši od PHP verzije 5.0 – **PHP Data Objects**, poznatiji kao **PDO**.

PHP Data Objects (PDO) je PHP5 dodatak koji definiše laganu i doslednu biblioteku za apstrakciju pristupa bazama podataka u PHP-u.

Potrebu za alatom kakav je PDO zahtevao je veoma veliki broj sistema za upravljanje bazama podataka koji su podržani od strane PHP-a. Svaki od ovih sistema zahteva poseban dodatak koji definiše jedinstveni API[3] za izvršavanje istih zadataka, počevši od uspostavljanja konekcije pa sve do naprednih svojstava kao što su pripremljeni iskazi (Prepared Statements) i upravljanje greškama (error handling).

Činjenica da ovi API-ji nisu ujedinjeni čini tranzicju između fundamentalnih sistema za upravljanje bazama podataka teškom, često se svodi na ponovno pisanje velikog broja linija kôda, koje pak vodi do novih programskih grešaka i zahteva se dodatno vreme za otkrivanje, debagovanje i ispravku. Sa druge strane, nedostatak jedinstvene biblioteke kao što je to npr. JDBS kod Java programskog jezika, činio je PHP manje popularnim u odnosu na ostale programske jezike kod kojih je ovakva biblioteka postojala. Pošto sada takva biblioteka postoji, PHP je ponovo zadobio svoju poziciju i predstavlja platformu izbora za milione programera.

PDO omogućava jasan, jednostavan (ali moćan), ujedinjen API za rad na svim podržanim sistemima za upravljanje bazama podataka.

Treba napomenuti, međutim, da postoji nekoliko biblioteka napisanih u PHP-u koje imaju istu svrhu kao i PDO. Najpopularnije su ADOdb biblioteka i PEAR DB paket. Ključna razlika između njih i PDO je brzina. PDO je PHP dodatak napisan u jeziku C/C++ , dok su PHP biblioteke napisane u PHP interpretatorskom jeziku. Takođe, jednom kada je PDO

[2] RTA (Real-time application) – su aplikacije koje funkcionišu unutar vremenskog okvira koje korisnik doživljava kao neposredan ili aktuelan.

[3] API (Application Programming Interface) - programski interfejs za pisanje aplikacija. Označava skup komandi, funkcija i protokola kroz koje jedna aplikacija koristi drugu aplikaciju.

omogućen, on ne zahteva uključivanje izvornih fajlova u skript i njegovo ponovno distribuiranje sa aplikacijom. Ovo čini razoj same aplikacije jednostavnijim, pošto programer PHP veb-aplikacije ne mora da se brine o dodatnom softveru.

Pomoću dole navedenih drajvera implementira se PDO interfejs za različite sisteme za upravljanje bazama podataka:

Ime drajvera	Podržana baza
PDO_CUBRID	Cubrid
PDO_DBLIB	FreeTDS / Microsoft SQL Server / Sybase
PDO_IBM	IBM DB2
PDO_INFORMIX	IBM Informix Dynamic Server
PDO_MYSQL	MySQL 3.x/4.x/5.x
PDO_OCI	Oracle Call Interface
PDO_ODBC	ODBC v3 (IBM DB2, unixODBC and win32 ODBC)
PDO_PGSQL	PostgreSQL
PDO_SQLITE	SQLite 3 and SQLite 2
PDO_SQLSRV	Microsoft SQL Server / SQL Azure
PDO_4D	4D

Za dodatne informacije o svakom drajveru pojedinacno pogledati [6].

2. KORIŠĆENJE PDO

Kao što je već rečeno, PDO omogućava konekciju ka bazi podataka i predstavlja biblioteku za apstrakciju pristupa podacima. Ovo zapravo znači da PDO definiše jedinstveni interfejs za kreiranje i održavanje konekcija sa bazom podataka, izdavanje upita, citiranje parametara, dovlačenje rezultujućih skupova, rukovanje pripremljenim iskazima i upravljanje greškama. Dakle, PDO nam omogućava da, bez obzira na to koji sistem za upravljanje bazama podataka je u pitanju, koristimo iste funkcije za postavljanje upita i dopremanje podataka.

Da bismo koristili PHP-ov PDO dodatak, nije potrebna nikakva spoljašnja biblioteka. PHP podrazumevano sadrži PDO počev od verzije PHP 5.1.0. Jedino što je potrebno uraditi je omogućavanje PDO drajvera za specifičnu bazu podataka koja će biti korišćena. Detaljnije o omogućavanju PDO drajvera za specifične baze podataka videti u [6].

2.1. Primer upotrebe PDO-a u PHP fajlu

Pre nego što detaljno razmotrimo sve osobine i mogućnosti PDO drajvera, kreiraćemo jednostavnu Veb stranicu upotrebom PHP programskog jezika i njegovog PDO dodatka.

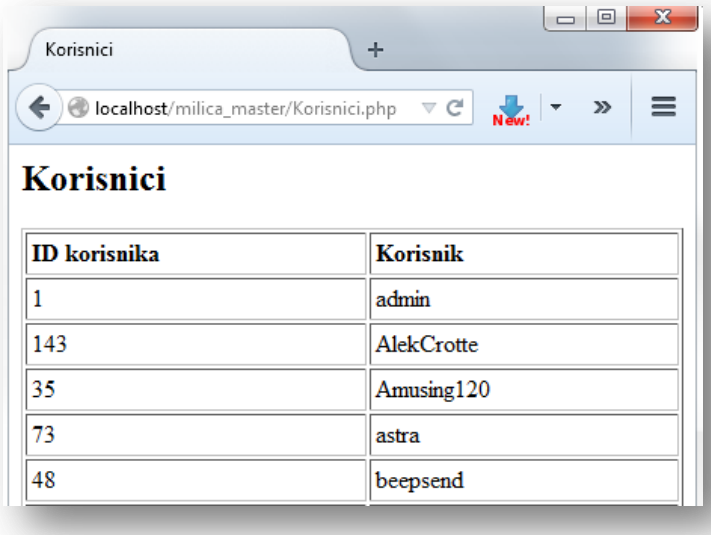
PDO je PHP-ov dodatak, pa se iz tog razloga sve linije koda koje se odnose na PDO pišu unutar PHP etiketa, za razliku od HTML etiketa koje se pišu van njih. Detaljnije o ovome može se naći u [5]. Prvo uspostavljamo konekciju sa bazom podataka (u ovom slučaju MySQL). Zatim pravimo istancu PDO klase koja predstavlja objekat konekcije.

```
<?php>

	// Konekcija sa bazom podataka
	$connStr = 'mysql:host=localhost;dbname=pdo';
	$user = 'root';
	$pass = 'root';

?>
	<html>
	<head><title><Korisnici></title></head>
	<body>
	<h1><Korisnici></h1>

<?php

	//Pravljenje objekta konekcije
	$conn = new PDO($connStr, $user, $pass);
```

Kada smo uspostavili konekciju, postavljamo upit, a potom iterativno ispisujemo na izlaz svaki iščitani red iz baze podataka.

```
	//Postavljamo upit
	$q = $conn->query("SELECT userid, username FROM emguser ORDER BY
		username");

?>

	//Pravimo tabelu
	<table width="100%" border="1" cellpadding="3">
	<tr style="font-weight: bold">
		<td>ID korisnika</td>
		<td>Korisnik</td>
	</tr>

<?php

	//Sada iterativno prolazimo kroz svaki red i ispisujemo ga na
	izlaz
	while($r = $q->fetch(PDO::FETCH_ASSOC))
		{

?>
		<tr>
			<td><?php echo $r['userid']?></td>
			<td><?phpecho $r['username']?></td>
		</tr>

<?php

		}

?>
	</table>
	</body>
	</html>
```

Pokretanjem prethodnog skripta dobićemo sledeći izlaz.



2.2. Povezivanje sa bazom podataka

Povezivanje sa bazom podataka uspostavlja se kreiranjem instance PDO klase. Ime klase je uvek PDO, nezavisno od toga koji se drajver koristi. Konstruktor prihvata **string za konekciju** - DSN[4] kojeg čine parameter za specifikovanu bazu podataka i opciono parametri za korisničko ime i lozinku (ukoliko postoje). Različiti tipovi baza podataka mogu, u manjoj ili u većoj meri, imati različite stringove za konekciju. Oni sadrže informacije kao što su ime sistema za upravljanje bazama podataka, kao i ime same baze, ali i neke druge parametre za konekciju.

Nadalje ćemo se pozabaviti metodima za konektovanje sa nekim od najpopularnijuh sistema.

Najpre razmatramo MySQL konekciju:

```
mysql_connect($host, $user, $password);
mysql_select_db($db);
```

Ovim smo uspostavili konekciju i potom biramo podrazumevanu bazu podataka za konekciju.

[4]Data Source Names, skraćeno DSN, označava skup svih informacija koje su neophodne za konekciju aplikacije sa bazom podataka.

SQLite:

```
$dbh = sqlite_open($db, 0666);
```

MSSQL:

```
mssql_connect($host, $user, $password)
```

PostgreSQL:

```
pg_connect("host=$host dbname=$db user=$user password=$password");
```

Kao što možemo da vidimo, sve ove baze podataka zahtevaju prilično različite načine za otvaranje konekcije. Ovo nije problem ako stalno koristitimo isti sistem za upravljanje bazama podataka, međutim, u slučaju da je potrebna migracija sa jednog na drugi sistem, morali bismo ponovo da pišemo svoje skriptove.

Pogledaćemo sada kako se ostvaruje konekcija korišćenjem PDO-a. Kako je PDO u potpunosti objektno orijentisan, bavićemo se **objektima konekcije**, a dalja interakcija sa bazom podataka će uključiti pozivanje različitih metoda nad ovim objektima.

U gornjim primerima postoji potreba za nečim što je slično objektima konekcije – pozivi prema `mysql_connect` ,`mssql_connect` ili `pg_connect` vraćaju link identifikatore, koji predstavljaju link ka bazi podataka, i PHP promenljive specifičnog tipa: resurs[5]. Međutim, objekti konekcije nisu korišćeni pošto API-ji ove tri baze ne zahtevaju da ih eksplicitno koristimo ako imamo samo jednu konekciju u našem skriptu. Na primer za MySQL možemo koristiti:

```
$con = mysql_connect($host, $user, $password);
mysql_select_db($db, $con);
```

ili:

```
$con = mysql_connect($host, $user, $password);
mysql_select_db($db);
```

Upotreba link identifikatora je opciona. Ukoliko se ovaj parameter izostavi, otvorice se konekcija koja je poslednja bila otvorena.

Ovo nije slučaj sa proceduralnom verzijom MySQLi (*MySQL Improved Extension*), iako većina funkcija ima isto ime i namenu kao kod MySQL. Zapravo, MySQLi, kao poboljšana verzija MySQL-a nudi dva pristupa: proceduralni interfejs, koji je sličan sa MySQL, i objektno - orijentisani interfejs.

[5]Resurs (engl. *resource*) je specijalna promenljiva koja čuva reference ka spoljašnjem resursu.

Povezivanje sa bazom korišćenjem proceduralnog interfejsa:

```
$mysqli = mysqli_connect($host, $user, $password, $db);
mysqli_connect_errno($mysqli);
```

Povezivanje sa bazom korišćenjem objektno - orijentisanog interfejsa:

```
$mysqli = new mysqli($host, $user, $password, $db);
$mysqli->connect_errno;
```

Kao što možemo da primetimo, upotreba link identifikatora je obavezna kod `mysqli_connect`, za razliku od `mysql_connect`.

Specijalno, SQLite uvek zahteva link identifikator.

Sa PDO, uvek ćemo morati eksplicitno da koristimo objekat konekcije, pošto ne postoji drugi način za konektovanje upotrebom ovog metoda.

Sve prethodne konekcije mogu se ostvariti na sledeći način:

MySQL:

```
$conn = new PDO("mysql:host=$host;dbname=$db", $user, $pass);
```

SQLite:

```
$conn = new PDO("sqlite:$db");
```

MSSQL

```
$conn = new PDO("mssql:host=$host;dbname=$db", $user, $pass);
```

PostgreSQL

```
$conn = new PDO("pgsql:host=$host dbname=$db", $user, $pass);
```

Kao što možemo da vidimo, jedini deo koji se menja ovde je prvi argument prosleđen PDO konstruktoru. Za SQLite, koji ne koristi korisničko ime i lozinku, drugi i treći argument mogu biti preskočeni.

Jednom kada je uspostavljena konekcija, instanca PDO klase se prosleđuje skriptu. Konekcija ostaje aktivna sve dok taj PDO objekat postoji. Da bi se konekcija zatvorila, potrebno je uništiti objekat tako što ćemo obrisati sve preostale reference ka njemu. Ovo možemo učiniti dodeljivanjem NULL vrednosti promenljivima koje čuvaju taj objekat.

```
$conn = null;
```

Ukoliko se ovo ne uradi eksplicitno, PHP će automatski zatvoriti konekciju kada se skript završi.

Mnoge Veb aplikacije mogu imati korist od pravljenja **perzistentne konekcije** (engl. *persistant connection*) sa SQL serverom. Perzistetna konekcija se ne zatvara po završetku skripta, već se čuva u keš memoriji i ponovo se koristi kada drugi skript zahteva konekciju upotrebom istih parametara. Keširanje perzistentne konekcije omogućava izbegavanje uspostavljanja nove konekcije svaki put kada skript treba da komunicira sa bazom podataka.

```
<?php
$dbh = new PDO('mysql:host=localhost;dbname=db', $user, $pass, array(
    PDO::ATTR_PERSISTENT => true
));
?>
```

Primetimo samo da, ukoliko želimo da koristimo perzistentnu konekciju, neohodno je podešavanje `PDO::ATTR_PERSISTENT` u nizu opcija za drajver koji se prosleđuje PDO konstruktoru. Ukoliko se ovaj atribut podesi upotrebom `PDO::setAttribute()` nakon instalacije objekta, drajver neće koristiti perzistentnu konekciju.

2.3. Stringovi za konekciju

PDO koristi takozvane stringove za konekciju koji omogućavaju PDO konstruktoru da odabere odgovarajuće upravljače i omogući sledećem metodu da ga pozove. Ovi stringovi za konekciju ili DSN-ovi su drugačiji za svaki sistem za upravljanje bazama podataka i to su jedine stvari koje ćemo morati da menjamo.

Ako razvijamo veliku aplikaciju, koja će moći da radi sa različitim bazama podataka, onda ovaj string za konekciju (zajedno sa korisničkim imenom i lozinkom za konekciju) može da se definiše u konfiguracionom fajlu i kasnije koristi na sledeći način (ako pretpostavimo da je konfiguracioni fajl sličan sa php.ini[6])

```
$config = parse_ini_file($pathToConfigFile);
$conn = new PDO($config['db.conn'],
config['db.user'],$config['db.pass']);
```

Tada bi konfiguracioni fajl izgledao ovako:

```
db.conn = "mysql:host=location;dbname=test"
```

[6]Php.ini je konfiguracioni fajl u kome se čuvaju globalna podešavanja za PHP.

```
db.user = "johns"
db.pass = "mypassphase"
```

Prefiks u promenljivoj db.conn (do znaka dve tačke) uvek čuva ime PDO drajvera. Pošto ne moramo da koristimo različite funkcije za kreiranje konekcije sa PDO, ovaj prefix nam govori koji interni drajver bi trebalo da bude korišćen. Ostatak stringa se parsira od strane drajvera kako bi se dalje inicirala konekcija. U ovom primeru prosleđujemo ime baze podataka, kao I domena sa kojeg se pokreće server.

Osim standardnih parametara, PDO drajver razume sledeće, dodatne, parametre:

- host – ime domena odakle se server pokreće
- port – broj porta za server baze podataka
- db_name – ime baze podataka
- unix_socket – MySQL UNIX soket (umesto host i/ili port)

Prednost korišćenja PDO-a, nasuprot korišćenju tradicionalih metoda u kreiranju konekcije ka bazi podataka, je to što ne moramo da menjamo svoj kôd ukoliko menjamo sistem za upravljanje bazama podataka. String za konekciju može biti definisan u konfiguracionom fajlu i taj fajl biva procesiran od strane naše aplikacije. Ukoliko postoji potreba za promenom baze podataka, sve što je potrebno da izmenimo je konfiguracioni fajl, a ostatak koda će ostati nepromenjen.

2.4. Izvršavanje SQL upita, rukovanje parametrima i rezultujućim skupom

PDO objekat, sa kojim smo se upoznali u prethodnom primeru, poseduje sve metode koje su potrebne za uniformno izvršavanje upita, bez obzira na to koji sistem za upravljanje bazama podataka se koristi.

Razmotrićemo jednostavan primer upita:

```
SELECT DISTINCT username FROM emguser ORDER BY username;
```

Ranije smo morali da pozivamo različite funkcije u zavisnosti od toga koja je baza podataka u pitanju:

```
// Sačuvaćemo SQL upit kao jednu promenljivu
$sql = 'SELECT DISTINCT username FROM emguser ORDER BY username;

// Sada ćemo pretpostaviti da je MySQL baza u pitanju:
mysql_connect('localhost', 'username', 'password');
mysql_select_db('emg');
```

```
$q = mysql_query($sql);

// Za SQLite bismo uradili sledeće:
$dbh = sqlite_open('/path/to/emg.ldb', 0666);
$q = sqlite_query($sql, $dbh);

// MSSQL:
mssql_connect('localhost', 'username', 'password');
$q = mssql_query($sql);

// PostgreSQL:
pg_connect("host=localhost dbname=emg user= username
password=password");
$q = pg_query($sql);
```

Ali sada, korišćenjem PDO, mi možemo uraditi sledeće:

```
$sql = 'SELECT DISTINCT username FROM emguser ORDER BY username;
$conn = new PDO($connStr, 'username', 'password');
$q = $conn->query($sql);
```

Kao što možemo da primetimo, upotreba PDO nije previše drugačija od tradicionalnih metoda rukovanja upitima. Ovde je takođe bitno napomenuti da se pozivom `$conn->query($sql)` vraća drugi objekat klase PDOStatement, za razliku od poziva `mysql_query()`, `sqlite_query()`, `mssql_query()` i `pg_query()`, koji vraćaju PHP promenljivu tipa resurs.

Sada ćemo postaviti nešto komplikovaniji upit od prethodnog:

```
SELECT count(id) FROM cdr where s_date > '2013/03/31',
```

a pre nego što pređemo dalje, smestićemo datum u promenljivu `$date`.

Upit bez korišćenja PDO izgledao bi ovako:

```
$date = '2013/03/31';
```

MySQL:

```
$m = mysql_real_escape_string($date);
$q = mysql_query("SELECT count(id) FROM cdr where s_date > '$m'");
```

SQLite:

```
$m = sqlite_escape_string($date);
$q = sqlite_query("SELECT count(id) FROM cdr where s_date > '$m'",
$dbh);
```

MSSQL:

```
$m = mssql_escape($date);
$q = mssql_query("SELECT count(id) FROM cdr where s_date > '$m'");
```

PostgreSQL:

```
$m = pg_escape_string($date);
$q = pg_query("SELECT count(id) FROM cdr where s_date > '$m'");
```

PDO klasa definiše jedinstveni metod za **citiranje parametara** (engl. *quoting parameters),* koji se kao takvi mogu bezbedno koristiti u upitima. Pitanja sigurnosti, kao što su SQL upad (SQL injection) će biti razmotrena nešto kasnije. Ovaj metod će obaviti jedan veoma uredan posao; automatski će dodati navodnike ukoliko to bude bilo potrebno:

```
$m = $conn->quote($date);
$q = $conn->query("SELECT count(id) FROM cdr where s_date > '$m'");
```

Uočavamo da PDO omogućava korišćenje istih obrazaca koji su bili korišćeni ranije, ali su imena svih metoda ujedinjena.

Sada možemo videti kakav će biti rezultat upita koje smo koristili. Pošto će upit iz poslednjeg primera vraćati uvek samo jedan red, upotrebićemo primer koji vraća više od jednog reda:

```
$sql = "SELECT DISTINCT username FROM emguser ORDER BY username"
```

MySQL:

```
$q = mysql_query($sql);
while($r = mysql_fetch_assoc($q))
	{
		echo $r['make'], "\n";
	}
```

SQLite:

```
$q = sqlite_query($dbh, $sql);
while($r = sqlite_fetch_array($q, SQLITE_ASSOC))
	{
		echo $r['make'], "\n";
	}
```

MSSQL:

```
$q = mssql_query($sql);
while($r = mssql_fetch_assoc($q))
	{
		echo $r['make'], "\n";
	}
```

PostgreSQL:

```
$q = pg_query($sql);
while($r = pg_fetch_assoc($q))
	{
		echo $r['make'], "\n";
	}
```

Uočavamo da je ideja ista, ali moramo da koristimo različita imena funkcija. Takođe, možemo da primetimo da SQLite zahteva još jedan parameter ukoliko želimo da dobijemo rezultat identičan onom koji se dobija korišćenjem MySQL, MSSQL i PostgreSQL (naravno,

ovo možemo izostaviti, ali u tom slučaju rezultujući redovi bi sadržali, kako indeksirana imena kolona, tako i numerički indeksirane elemente.)

Prilikom korišćenja PDO-a nije bitno koja je baza podataka po sredi - metodi za prikupljanje redova su isti za sve baze podataka. Dakle, prethodni kôd bi trebalo napisati za PDO na sledeći način:

```
$q = $conn->query("SELECT DISTINCT username FROM emguser ORDER BY
username");
while($r = $q->fetch(PDO::FETCH_ASSOC))
	{
		echo $r['make'], "\n";
	}
```

Ništa nije drugačije od onoga što bi se dogodilo ranije. Ono što bi trebalo istaći ovde je da je eksplicitno navedena `PDO::FETCH_ASSOC` konstanta, pošto je podrazumevani režim za dovlačenje podataka `PDO::FETCH_BOTH`, koji doprema rezultujuće redove kao nizove indeksirane po imenu kolone i po rednom broju kolone.

2.5. PDO iskazi i skupovi rezultata

PHP PDO ima dve glavne klase, **PDO** i **PDOStatement**. PDOklasa se koristi za pravljenje konekcije i rukovanje upitima, dok se PDOStatement klasa koristiza petlje u rezultujućem skupu podataka.

Sada ćemo razmotriti PDOStatement klasu. Glavna namena ove klase je da obezbedi interfejs za rezultujući skup podataka. Instance PDOStatement klase dobijamo pozivom `PDO::query()` metoda. PDO nudi nekoliko **režima za dovlačenje podataka**[7]. Ova klasa, takođe, može da obezbedi dodatne informacije o rezulujućem skupu podataka u obliku dvodimenzionalnog niza.

[7]Režimi za dopremanje podataka su tzv.fetch modovi.

Počećemo sa razmatranjem nekih od režima za dovlačenje podataka.

- **PDO::FETCH_ASSOC** - Najkorišćeniji režim; vraća niz indeksiran po imenu kolone.
- **PDO::FETCH_BOTH** - Podrazumevana operacija PDOStatement objekta; vraća niz indeksiran po indeksu (koji je tipa integer) i imenu kolone.
- **PDO::FETCH_NUM** - Korišćenjem ovog režima zathevamo samo intiger-indeksiran niz.
- **PDO::FETCH_OBJ** - Omogućava dovlačenje redova iz baze kao objekata. Vraća anoniman objekat sa imenima karakteristika koje odgovaraju imenima kolona.
  U ovom slučaju pozivom `PDO::fetch()` metoda vraća se instanca interne klase `stdClass`, zajedno sa njenim karakteristikama koje su dodate vrednostima u svakom redu. To se dešava u sledećem kôdu:

```
$q = $conn->query('SELECT * FROM operators ORDER BY country');
$r = $q->fetch(PDO::FETCH_OBJ);
var_dump($r);

//izlaz je:
object(stdClass)[3]
public 'ID' =>string '41220' (length=5)
public 'mcc' =>string '412' (length=3)
public 'mnc' =>string '20' (length=2)
public 'mccmnc' =>string '41220' (length=5)
public 'country' =>string 'Algerie' (length=11)
public 'operator' =>string'Orascom Telecom Algerie Spa
        (Djezzy)'(length=36)
public 'price' =>string '0.006' (length=5)
public 'comment' =>string '' (length=0)
public 'margin' =>string '15' (length=2)
```

`PDO::Statemnt` klasa takođe omogućava da se režim za dovlačenje podataka podesi jednom za sve sledeće pozive `fetch()` metoda. Ovo se obavlja korišćenjem `PDOStatement::setFetchMode()` metoda koji prihvata bilo koju od `PDO::FETCH_ASSOC`, `PDO::FETCH_BOTH`, `PDO::FETCH_NUM` i `PDO::FETCH_OBJ` konstanti.

Možemo primetiti da SQLite, MySQL, MSSQL i PostgreSQL PHP ekstenzije nude sličnu funkcionalnost. I zaista, mi možemo koristiti bilo koju od `mysql_fetch_row()`, `mysql_fetch_assoc()`, `mysql_fetch_array()` ili `mysql_fetch_object()` funkcija da postignemo isti efekat. To je zbog toga što PDO omogućava korišćenje tri dodatna režima za dovlačenje podataka. Ova tri režima mogu biti podešena korišćenjem `PDOStatement::setFetchMode()` poziva, a to su:

- **PDO::FETCH_COLUMN** - Omogućava nam da damo instrukcije PDOStatement objektu tako da vrati traženu kolonu svakog reda. U ovom slučaju `PDO::fetch()` će vratiti

skalanu vrednost. Kolone imaju brojeve počevši od 0. Sledeći deo koda ilustruje prethodno:

```
$q = $conn->query('SELECT * FROM operators ORDER BY country');
$q->setFetchMode(PDO::FETCH_COLUMN, 5);
while($r = $q->fetch())
   {
         var_dump[8]($r);
   }

//izlaz je:
string ' AMX Argentina S.A Claro ' (length=25)
string 'AMX Argentina (Claro)' (length=21)
string 'AMX Argentina (Claro)' (length=21)
string 'Telefonica Argentina (Movistar Argentina' (length=40)
string 'Orange (Orange Armenia CJSC)' (length=28)
```

Ovo otkriva da poziv `$q->fetch()` zaista vraća skalarne vrednosti (a ne nizove).

- **`PDO::FETCH_INTO`** – Koristi se menjanje instance nekog objekta. Izmenićemo prethodni kôd da izgleda ovako:

```
$q = $conn->query('SELECT * FROM operators ORDER BY country');
$r = new stdClass();
$q->setFetchMode(PDO::FETCH_INTO, $r);
while($q->fetch())
   {
         var_dump($r);
   }

//tada ce izlaz biti
object(stdClass)[3]
public 'ID' =>string '60302' (length=5)
public 'mcc' =>string '603' (length=3)
public 'mnc' =>string '02' (length=2)
public 'mccmnc' =>string '60302' (length=5)
public 'country' =>string 'Algeria' (length=7)
public 'operator' =>string 'Orascom Telecom Algerie Spa (Djezzy)'
(length=36)
public 'price' =>string '0.007' (length=5)
public 'comment' =>string '' (length=0)
public 'margin' =>string '15' (length=2)
object(stdClass)[3]
public 'ID' =>string '603' (length=3)
public 'mcc' =>string '603' (length=3)
public 'mnc' =>string '' (length=0)
public 'mccmnc' =>string '603' (length=3)
public 'country' =>string 'Algeria' (length=7)
public 'operator' =>string 'Default' (length=7)
public 'price' =>string '0.024' (length=5)
```

[8]PHP-ova *var_dump* funkcija prikazuje struktuirane informacije o promenljivama koje ukljucuju tip promenljive i njenu vrednost.

```
public 'comment' =>string 'Do not cover Wataniya Nedjma (60303)'
(length=36)
public 'margin' =>string '15' (length=2)
```

Promenljivoj `$r` nije dodeljena vrednost unutar `while` petlje, a to je zapravo izlaz iz `while` petlje. Promenljiva `$r` je povezana sa ovim metodom pozivom `$q->setFetchMode()` pre same petlje.

- **`PDO::FETCH_CLASS`** - koristi se za vraćanje objekata neke naznačene klase. Za svaki red istanca ove klase će biti kreirana sa imenovanim osobinama i dodeljenim vrednostima iz kolona rezultujućeg skupa. Primetimo samo da klasa ne mora imati deklarisane ove osobine pošto PHP dozvoljava kreiranje osobina objekata u trenutku pokretanja koda. Primer:

```
$q = $conn->query('SELECT * FROM operators ORDER BY country'firstName');
          $q->setFetchMode(PDO::FETCH_CLASS, stdClass);
while($r = $q->fetch())
{
      var_dump($r);
}
```

Izlaz iz ovog primeraće biti sličan kao u prethodnom primeru. Takođe, ovaj režim za dovlačenje podataka dozvoljava kreiranje istanci prosleđivanjem niza parametara konstruktoru:

```
$q->setFetchMode(PDO::FETCH_CLASS, MojaKlasa, array(1, 2, 3));
```

Preporuka je da se koristi `PDOStatement::setFetchMode()` zbog toga što je pogodniji i lakši za održavanje, a ima i više svojstava. Svaki od ovih režima je koristan u određenoj situaciji.

2.6. Dopremanje rezultujućeg skupa metapodataka

Kao što smo videli u prethodnom odeljku, `PDOStatemnt` klasa nam dozvoljava da iz baze dovučemo neke informacije o podacima koji su sadržani u rezultujućem skupu. Ove informacije se zovu **metapodaci** (engl. *metadata*). Neki od metapodataka o rezultujućem skupu su broj kolona i broj redova koje taj skup sadrži.

Jedan od metoda koji može biti od koristi je `PDOStatement::columnCount()`, koji vraća broj kolona u rezultujućem skupu. Ovo je korisno prilikom izvršavanja proizvoljih upita. Može se koristiti na sledeći način:

```
$q = $conn->query("SELECT o.id, o.country, o.operator, p.*
                   FROM operators o, prefix p
                   WHERE o.id = p.prefix
                   ORDER BY title");
                   var_dump($q->columnCount());
```

Nažalost, PDO trenutno ne omogućava da se u rezultatu dobiju nazivi tabela ili njihovih kolona. Ova funkcionalnost je korisna ako skript koristi dve ili više spojenih tabela. U takvim slučajevima moguće je dobiti ime tabele za svaku kolonu pomoću numeričkog indeksa. Međutim, upotrebom aliasa u davanju naziva kolonama, eliminiše se potreba za ovom funkcionalnošću:

```
$q = $conn->
query("SELECT o.id AS operatorId, o.country, o.operator, p.*
                   FROM operators o, prefix p
                   WHERE o.id = p.prefix
                   ORDER BY title");
                   var_dump($q->columnCount());
```

Korišćenjem odgovarajućih aliasa (koji sadrze ime tabele) može se tačno znati koja kolona pripada kojoj tabeli.

Najkorišćeniji metapodatak je broj redova koje neki skup sadrži. Možemo da koristimo brojač redova obeležavanjem velikih rezultujućih skupova. Ali, ukoliko radimo sa bazom podataka čije tabele imaju ogroman broj redova, biće nam potreban neki alat kojim bismo dobili broj redova za svaku tabelu i obeležili je radi jednostavnijeg pretraživanja.

Tradicionalno, koristili bismo `mysql_num_rows()`, `sqlite_num_rows()`, `mssql_num_rows()` ili `pg_num_rows()` funkciju[9], da bismo dobili ukupan broj redova koji vraća upit. U PDO, metod odgovoran za dobijanje broja redova je `PDO::Statement::rowCount()`. Ako pokrenemo sledeći kôd:

```
$q = $conn->query("SELECT * FROM operators");
$q->setFetchMode(PDO::FETCH_ASSOC);
var_dump($q->rowCount());
```

videćemo da PDO vraća 0 kako za MySQL tako i za SQLite. Ovo se dešava zbog toga što PDO funkcioniše drugačije od tradicionalnih dodataka bazi. Šta se zapravo desilo? Ako je poslednja SQL naredba izvršena od strane PDOStatement klase `SELECT` naredba, neki sistemi za upravljanje bazama podataka mogu vratiti broj redova koje vraća ta naredba.

[9]U zavisnosti od toga koji sistem za upravljanje bazom podataka koristimo.

Međitim, ovakav način nije garantovan za sve baze i ne bi trebalo da bude pouzdan za prenosive i složene aplikacije. MySQL, kao i SQLite, ne podržavalju ovu funkcionalnost i to je razlog zašto je vrednost koja je vraćena 0. Zbog toga ćemo koristiti drugi način kako bismo došli do ovog metapodatka.

Složene, dinamičke Veb aplikacije, koje se pokreću nad bazom podataka, se u mnogome razlikuju jedne od drugih pošto je njihova složenost diktirana njihovom namenom, mada, skoro sve imaju neke zajedničke karakteristike. Jedna od ovih karakteristika je sposobnost obeleževanja velikih rezultujućih skupova radi poboljšanja kvaliteta korišćenja i bržeg učitavanja stranice aplikacije.

Ispravno obeležavanje zahteva računanje ukupnog broja redova dobijenih iz baze, broj stranica (ovo je konfiguraciona opcija), kao i broj aktuelne strane. Na osnovu ovih podataka, lakše je računati početni opseg rezultata i prikazati samo određeni podskup redova.

Kao što smo već videli, `PDOStatement::rowCount()` metod ne vraća tačan broj redova za neki upit. Razlog za to je što sistemi za upravljanje bazama podataka zapravo ne znaju taj broj sve dok se i poslednji red iz upita ne dopremi iz baze. Razlog zašto funkcija `mysql_num_rows()` (i njoj slične funkcije) vraćaju broj redova je taj što ona prethodno učita čitav skup rezultata u memoriju.

Iako ovo možda izgleda pogodno, ovakav način nije preporučljiv. Ukoliko upit vraća 20 redova, tada skript može obezbediti potrebnu memoriju. Ali, ako upit vraća nekoliko stotina hiljada redova, kao što je slučaj sa sajtovima koji imaju ogroman saobraćaj, server može ostati bez resursa jer se sve čuva u memoriji.

Jedina logična mera (i jedina opcija dostupna sa PDO-om) je da sama baza broji svoje redove pod određenim instrukcijama. Nezavisno od toga koliko je upit složen, može se izmeniti tako da koristi `SQL count()` funkciju za vraćanje broja redova.

Razmotrićemo sada neke upite:

```
SELECT DISTINCT
      e.userid,
      e.username
FROM
      emguser e
      RIGHT JOIN cdr c
      ON e.username = c.username
WHERE
      DATE(cdr.s_date) = DATE_SUB(date(now()),INTERVAL 1 DAY)
ORDER BY e.username
```

Sada ćemo izmeniti upit tako da vraća broj redova:

```
SELECT
      COUNT(*)
FROM
      emguser e
      RIGHT JOIN cdr c
      ON e.username = c.username
WHERE
      DATE(cdr.s_date) = DATE_SUB(date(now()),INTERVAL 1 DAY)
```

Primetimo da je razlika prvog i drugog upita u tome što je lista kolona zamenjena sa `COUNT(*)` i uklonjena je `order by` klauzula. Imajući ovo na umu, možemo napisati funkciju koja će prihvatati string koji sadrzi SQL upit za izvršavanje i vratiti broj redova koje taj upit vraća.

```
/**
* Funkcija vraca broj redova upita
* parameter je SQL upit
* izbacuje PDOException u slucaju da se upit ne moze izvrsiti
*/
function getRowCount($sql)
      {
            global $conn;
            $sql = trim($sql);
            $sql = preg_replace('~^SELECT\s.*\sFROM~s', 'SELECT
                COUNT(*) FROM',$sql);
            $sql = preg_replace('~ORDER\s+BY.*?$~sD', '', $sql);
            $stmt = $conn->query($sql);
            $r = $stmt->fetchColumn(0);
            $stmt->closeCursor();
            return $r;
      }
```

PHP-ova funkcija *preg_replace* ima tri argumenta: *obrazac* (regularni izraz), *zamena* i *string* (koji se pretražuje). Preg_replace izvršava pretragu stringa korišćenjem regularnih izraza i vrši supstituciju obrasca zamenom.

Upotrebom `getRowCount()` funkcije izbegavamo korišćenje baferisanih upita, što znatno ubrzava skriptove.

2.7. Prenosivost SQL koda izmedju različitih sistema upotrebom PDO-a

PDO je (apstraktna) biblioteka za konekciju sa bazom podataka i, kao takva, ne može osigurati da će neki kôd raditi za svaki relacioni sistem za upravljanje bazama podataka koji podržava. Ovo će se desiti jedino ako je SQL kôd prenosiv (portabilan). Na primer, MySQL proširuje SQL sintaksu ovakvom formom naredbe za ubacivanje podataka u bazu:

```
INSERT INTO mytable SET x=1, y='two';
```

Ovakav SQL kôd nije prenosiv, pošto drugi sistemi za upravljane bazama podataka ne razumeju ovakav način primene naredbe za ubacivanje podataka. Više o sintaksi MySQL sistema za upravljanje bazama podataka pronaći u [4].

Da bismo bili sigurni da će naša naredba za ubacivanje podataka biti izvršena na različitim sistemima za upravljanje bazama podataka, trebalo bi prethodni kôd zameniti sledećim:

```
INSERT INTO mytable(x, y) VALUES(1, 'two');
```

Ovo je samo jedan od primera nekompatibilnosti koje mogu narasti prilikom upotrebe PDO-a. Kreiranje sheme baze podataka i SQL koda prenosivim je jedino što može osigurati da će kôd biti odgovarajući (kompatibilan) sa ostalim sistemima za upravljanjem bazama podataka.

3. PRIPREMLJENI ISKAZI

3.1. Rukovanje pripremljenim iskazima

U prethodnim odeljcima su izložene neke elementarne stvari o PDO-u i mogli smo da primetimo da većina tih funkcionalnosti podseća na tradicionalne dodatke koji se koriste prilikom konekcije sa bazom podataka. Jedina funkcionalnost koja se više razlikuje su izuzeci (exceptions), ali čak i to može biti slično tradicionalnom načinu za rukovanje greškama. O izuzecima će biti reči u odeljku *4. Rukovanje Greškama.*

Sada ćemo se posvetiti potpuno novom konceptu koji nije bio prisutan u PHP-u pre: **pripremeljeni iskazi** (engl. *prepared statements*), koji mogu dodatno pojednostaviti kôd, a dovesti do boljih performasi.

Pripremljeni iskaz je šablon za izvršavanje jednog ili više SQL upita nad bazom podataka. Ideja koja stoji iza pripremljenih iskaza je sledeća: upotrebom upita koji imaju istu sintaksu, a različite vrednosti, mnogo je brže pre-procesiranje sintakse jednom, a potom izvršavanje iskaza više puta korišćenjem različitih parametara.

PHP-ovi dodaci za MySQL i za SQLite ne nude funkcionalnost koja podržava pripremljene iskaze, dok PostgreSQL i MySLQi nude, ali ne uključuju sve opcije koje nudi PDO (uprkos mogućem objektno orijentisanom stilu). Za detalje pogledati [1]. No upotrebom PDO-a ovo je dostupno svim tipovima sistema za upravljanje bazama podataka.

Pogledajmo najpre kako pripremljeni iskazi funkcionišu, a potom ćemo razmotriti i neke složenije upite.

Prilikom razvoja interaktivnih, dinamičkih aplikacija zasnovanih na bazama podataka, pre ili kasnije javlja se potreba za uzimanjem korisničkog unosa i njegovog prosleđivanja bazi podataka kao dela upita. U sledećem primeru koristi se PHP-ova globalna promenljiva $_REQUEST u kojoj se čuvaju parametri poslati sa klijentske strane. Eksplicitnim kastovanjem u celobrojne promenljive osiguravamo se da će promenljive biti istog tipa i da će se njihovo poređenje obaviti bez grešaka.

```
// Pretpostavimo da se u promenljivima startYear i endYear
// cuvaju godine:
$sy = (int)$_REQUEST['startYear'];
$ey = (int)$_REQUEST['endYear'];

if($ey < $sy)
      {
            // budimo sigurni da je $sy manje od $ey
```

```
                $tmp = $ey;
                $ey = $sy;
                $sy = $tmp;
        }
$sql = "SELECT * FROM cdr WHERE YEAR(s_date)>= $sy AND YEAR(s_date) <=
$ey";
```

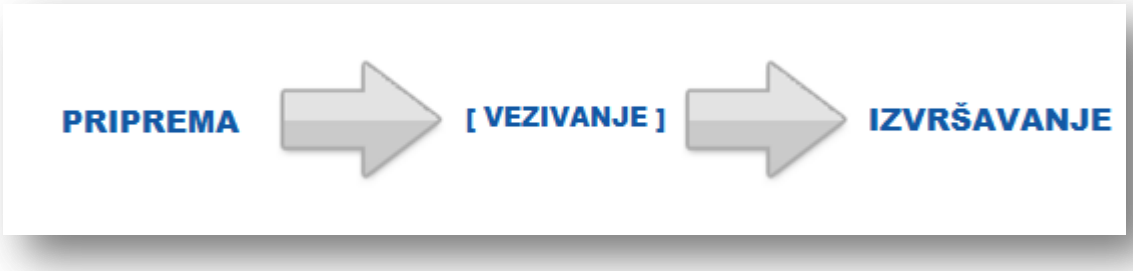
U ovom primeru upit zavisi od dve promenljive, koje su deo rezultujućeg SQL-a. Odgovarajući pripremljeni iskaz u PDO-u bi izgledao ovako:

```
$sql = 'SELECT * FROM cdr WHERE YEAR(s_date) >= ? AND YEAR(s_date) <=
?';
```

Zamenili smo imena promenljivih sa ?[10] u telu upita. Sada je potrebno da ovaj upit izvršimo.

```
// Pretpostavimo da konekcija vec postoji
// $sy i $ey su promenljive
$sql = 'SELECT * FROM cdr WHERE YEAR(s_date) >= ? AND YEAR(s_date) <=
?';
$stmt = $conn->prepare($sql);
$stmt->execute(array($sy, $ey));
```

Ove nam tri linije koda govore da su pripremljeni iskazi objekti (sa klasom PDOStatement). Oni su kreirani korišćenjem `PDO::prepare()` metoda koji prihvata SQL izjavu sa znakovima pitanja kao svoj paramater.



Nakon ovoga, pripremljene iskaze je potrebno **izvršiti** (engl. *execute*) kako bismo dobili rezultate upita pozivom metoda `PDOStatement::execute()`. Kao što primer pokazuje, pozivamo ovaj metod upotrebom niza koji čuva promenljive koje će zameniti znakove pitanja. Primetimo kako se redosled promenjljivih u nizu slaže sa redosledom znakova pitanja u promenljivoj `$sql`. Očigledno, broj elemenata u nizu mora biti isti kao i broj znakova pitanja u upitu.

Možemo primetiti da nismo sačuvali rezultat poziva `PDOStatement::execute()` metoda u nekoj promenljivoj. To je zbog toga što je sam iskaz, kao objekat, korišćen za pristup rezultatima upita, tako da možemo kompletirati primer na sledeći način:

[10]Znakovi pitanja (?) se u pripremljenim iskazima nazivaju čuvarima mesta– Placeholders.

```
// Pretpostavimo da se u promenljivima startYear and endYear
// cuvaju godine:
$sy = (int)$_REQUEST['startYear'];
$ey = (int)$_REQUEST['endYear'];
if($ey < $sy)
      {
            $tmp = $ey;
            $ey = $sy;
            $sy = $tmp;
      }
$sql = 'SELECT * FROM cdr WHERE YEAR(s_date) >= ? AND YEAR(s_date) <=
?';
$stmt = $conn->prepare($sql);
$stmt->execute(array($sy, $ey));

// sada ponavljamo nad rezultatom posto smo dobili
// $stmt u pozivu PDO::query()
while($r = $stmt->fetch(PDO::FETCH_ASSOC))
      {
            Echo "$r[make] $r[model] $r[year]\n";
      }
```

Kao što kompletiran primer pokazuje, pozivamo metod `PDOStatement::fetch()` sve dok ne vrati netačnu vrednost (false) i u tom trenutku se petlja prekida.

Naravno, zamena znakova pitanja konkretnim vrednostima nije jedina mogućnost koju imaju pripremljeni iskazi. Njihova moć leži u mogućnosti da se izvršava onoliko puta koliko je to potrebno. Ovo zapravo znači da možemo pozivati `PDOStatement::execute()` metod onoliko puta koliko nam je potrebno i svaki put mu možemo proslediti druge vrednosti.

```
$sql = 'SELECT * FROM cdr WHERE YEAR(s_date)>=? AND YEAR(s_date)<=?';
$stmt = $conn->prepare($sql);
// Prvo izvrsavanje skripta
$stmt->execute(array(2013, 2014));
$newMessages = $stmt->fetchAll(PDO::FETCH_ASSOC);
// Ponovno izvrsavanje istog skripta, ali se ovoga puta
// prosledjuju druge promenljive
$stmt->execute(array(2010, 2012));
$olderMessages = $stmt->fetchAll(PDO::FETCH_ASSOC);
// Prikazivanje
echo 'Imamo', count($newMessages), ' porukaiz perioda 2005-2007';
print_r($newMessages);
echo 'Imamo', count($olderMessages), ' poruka iz perioda 2000-2004';
print_r($olderMessages);
```

Pripremljeni iskazi imaju tendenciju da se izvršavaju brže od `PDO::query()` metoda, pošto se drajveri baze podataka optimizuju samo jednim pozivom `PDO::prepare()` metoda. Još jedna prednost korišćenja pripremljenih iskaza je što ne moramo da koristimo navodnike za parametre koje prosleđujemo u pozivu `PDOStatement::execute()`.

U našem primeru smo koristili eksplicitno kastovanje parametra u celobrojnu promenljivu, ali smo mogli da uradimo i sledeće:

```
$sql = 'SELECT * FROM cdr WHERE provider=?';
$stmt = $conn->prepare($sql);
$stmt->execute(array($_REQUEST['Provider']));
```

Ovde će se pripremljeni iskaz pobrinuti za odgovarajuće citiranje koje je učinjeno pre izvršenja upita.

Razmotrićemo i jedan primer menjanja baze podataka. Pretpostavimo da se informacije koje je potrebno dodati nalaze u sledećem PHP nizu:

```
$users = array(
array(
      'username' => 'Alexander',
      'password' => 'pass1',
      'throughput' => '100',
      'charge_balance' => '50',
      'desc' => 'Alexandre is standard user'),

$users = array(
array(
      'username' => 'TelePro',
      'password' => 'pass2',
      'throughput' => '1000',
      'charge_balance' => '2000',
      'desc' => 'TelePro is premium user'));
```

Ovo je dvodimenzionalni niz kroz koji ćemo proći iterativno koristeći `foreach` petlju da bismo sve podatke ubacili u bazu.

```
foreach($users as $user)
      {
            $conn->query(
            'INSERT INTO emguser(
                  username,
                  password,
                  throughput,
                  charge_balance,
                  desc)
            VALUES(' .
                  $conn->quote($user ['username']) .
                  ',' . $conn->quote($user ['password']) .
                  ',' . $conn->quote$user ['throughput'])')' .
                  ',' . $conn->quote$user ['charge_balance'])')' .
                  ',' . $conn->quote$user ['desc'])')' .
            );
      }
```

Kao što se vidi iz prethodnog primera, za svaku iteraciju ovaj SQL iskaz je iznova kreiran za svaki element niza, a takođe se moramo pozabaviti i citiranjem parametara.

Koristeći pripremljene iskaze, SQL upit možemo kreirati samo jednom i izvršavati ga onoliko puta koliko nam je to potrebno, prosleđivanjem različitih parametara. Kôd iz prethodnog primera izgledaće ovako:

```
$stmt = $conn->prepare('
INSERT INTO emguser(
      username,
      password,
      throughput,
      charge_balance,
      desc)
VALUES(?, ?, ?, ?, ?)');
foreach($authors as $author)
      {
            $stmt->execute(
            array(
                  $users['username']),
                  $users['password'],
                  $users['throughput'],
                  $users['charge_balance'],
                  $users['decs']));
      }
```

Uočavamo da je pripremljeni iskaz prvo pripremljen, pozivom metoda `PDO::prepare()`. Ovaj metod prihvata string koji sadrži SQL komandu gde su vrednosti, koje se menjaju, zamenjene znakovima pitanja. Poziv vraća objekat klase `PDOStatement`. Potom, u petlji pozivamo `execute()` metod umesto `PDO::query()` metoda.

`PDOStatement::execute()` metod prihvata niz vrednosti, koje su ubačene u SQL upit umesto znakova pitanja. Broj i redosled elemenata mora se poklapati sa znakovima pitanja u upitu koji je prosleđen `PDO::Prepare()` metodu. Ovde nismo koristili `PDO::quote()` u kodu zbog toga što se PDO pobrinuo za odgovarajuće citiranje parametara.

3.2. Pozicioni i imenovani čuvari mesta

U prethodnim primerima su korišćeni znakovi pitanja koji određuju pozicije za vrednosti u pripremljenim iskazima. Iz tog razloga ovi znakovi pitanja se nazivaju **pozicioni čuvari mesta** (engl.*placeholders*). Prilikom njihovog korišćenja mora se paziti na odgovarajući redosled elemenata u nizu koji se prosleđuju `PDOStatement::execute()` metodu. Iako se brzo ispisuju, mogu biti veoma teški za praćenje grešaka, posebno ako se promene kolone u upitu. Da bismo se zaštitili od potencijalnih grešaka, koristimo

takozvane **imenovane čuvare mesta** (engl. *named placeholders*), koji se sastoje od opisnih imena ispred kojih se koriste dve tačke.

Prethodni primer izmenićemo tako da koristi imenovane čuvare mesta:

```
$stmt = $conn->prepare(
'INSERT INTO emguser(
      username,
      password,
      throughput,
      charge_balance,
      desc)'.
'VALUES(
      :username,
      :password,
      :throughput,
      :balance,
      :desc)');
foreach($authors as $author)
      {
            $stmt->execute(
            array(
                  ':username'=> $users['username'],
                  ':password'=> $author['password'],
                  ':throughput' => $author['throughput'],
                  ':balance' => $author['balance'],
                  ':desc' => $author['desc'])
                  )
      );
```

Zakovi pitanja zamenjeni su imenovanim čuvarima mesta, a potom je u pozivu `PDOStatement::execute()` obezbeđen asocijativni niz, gde ključ odgovara imenovanom čuvaru mesta, a vrednost podacima koje želimo da ubacimo u bazu.

Kod imenovanih čuvara mesta redosled elemenata u nizu nije značajan, jedino što je važno je veza između odgovarajućih elemenata u nizu parova oblika ključ-vrednost. Sledeći kôd bi dao isti rezultat kao i prethodni:

```
foreach($authors as $author)
      {
            $stmt->execute(
            array(
                  ':throughput' => $author['throughput'],
                  ':password' => $author['password'],
                  ':balance' => $author['balance'],
                  ':username' => $users['username'],
                  ':desc' => $author['desc'])
                  )
      );
```

Ukoliko prosledimo niz vrednosti `PDOStatement::execute()` metodu koji se ne poklapa sa brojem čuvara mesta u upitu, ili prosledimo asocijativni niz iskazu koji koristi

imenovane čuvare mesta (i obratno), ovo će se tretirati kao greška i prouzrokovati izbacivanje nekog izuzetka.

Još jednu stvar je važno napomenuti o upotrebi čuvara mesta. One se ne mogu upotrebiti kao deo vrednosti koje se prosleđuju bazi podataka. Sledeći primer ilustruje jedan slučaj nepravilnog korišćenja:

```
$stmt = $conn->prepare("SELECT * FROM emguser WHERE username
LIKE '%?%'");
$stmt->execute(array($_GET['name']));
```

Umesto ovoga, ispravno bi bilo koristiti čuvare mesta u pozivu metoda, a ne u samom SQL upitu:

```
$stmt = $conn->prepare("SELECT * FROM emguser WHERE username
LIKE ?");
$stmt->execute(array('%' . $_GET['name'] . '%'));
```

3.3. Pripremljeni iskazi i uvezane vrednosti

Prethodni primeri koriste takozvane **neuvezane iskaze** (engl. *unbound values*). To znači da se vrednosti za upit dopremaju u obliku niza koji je prosleđen `PDOStatement::execute()` metodu. PDO podržava i **uvezane iskaze** (engl. *bound values*) gde je moguće eksplicitno uvezati trenutnu vrednost ili promenljivu sa imenovanim ili pozicionim čuvarima mesta.

Da bi se neka trenutna vrednost uvezala sa iskazom, koristi se `PDOStatement::bindValue()` metod. Ovaj metod prihvata identifikator čuvara mesta i njegovu vrednost. Identifikator čuvara mesta je indeks znakova pitanja kod pozicionih čuvara mesta ili naziv imenovanog čuvara mesta kod imenovanih čuvara mesta. Dalje menjamo primer sa pozicionim čuvarima mesta tako da koristi uvezane vrednosti na sledeći način:

```
$stmt = $conn->prepare(
'INSERT INTO emguser(
      username,
      password,
      throughput,
      charge_balance,
      desc)');

foreach($users as $user)
      {
            $stmt->bindValue(1, $user['username']);
            $stmt->bindValue(2, $user['password']);
            $stmt->bindValue(3, $user['throughput']);
```

```
                $stmt->bindValue(4, $user['charge_balance']);
                $stmt->bindValue(5, $user['desc']);
                $stmt->execute();}
```

Za imenovane čuvare mesta:

```
$stmt = $conn->prepare(
'INSERT INTO emguser(
      username,
      password,
      throughput,
      charge_balance,
      desc)'.
'VALUES(
      :username,
      :password,
      :throughput,
      :balance,
      :desc)');
foreach($users as $user)
      {
            $stmt->bindValue(username, $user['username']);
            $stmt->bindValue(password, $user['password']);
            $stmt->bindValue(throughput, $user['throughput']);
            $stmt->bindValue(charge_balance, $user['charge_balance']);
            $stmt->bindValue(desc, $user[‘desc’]);
            $stmt->execute();
      }
```

U oba slučaja nismo prosledili ništa u pozivu `PDOStatement::execute()`. I ponovo, ako ne uvežemo vrednosti sa svakim čuvarom mesta, poziv `PDOStatement::execute()` neće biti uspešan i dovešće do izbacivanja izuzetka.

PDO takođe može da uveže kolone rezultujućeg skupa sa PHP promenljivima za `SELECT` upite. Ove promenljive će biti zamenjene odgovarajućim vrednostima iz kolona svakim pozivom `PDOStatement::fetch()`. Ovo je alternativan način dovlačenja rezultujućeg skupa podataka u obliku niza, nasuprot ranije objašnjenom metodu gde su podaci bili dovlačeni preko objekata. Primer:

```
$stmt = $conn->prepare('SELECT username, charge_balance FROM emguser');
$stmt->execute();
$stmt->bindColumn(1, $username);
$stmt->bindColumn(2, $charge_balance);
while($stmt->fetch(PDO::FETCH_BOUND))
      {
            echo "$username, $charge_balance <br>";
      }
```

Ovo će prikazati sve korisnike iz tabele. Promenljive su uvezane pozivom `PDOStatement::bindColumn()` metoda, koji očekuje da prvi parameter bude indeks kolone u rezultatu ili ime kolone, a drugi parameter je vrednost koju treba promeniti.

Primetimo da prilikom upotrebe uvezanih kolona, `PDOStatement::fetch()` metod bi trebalo da bude pozvan pomoću `PDO::FETCH_BOUND`, ili bi sve trebalo prethodno podesiti pozivom `PDOStatement::setFetchMode (PDO::FETCH_BOUND)`. Takođe, poziv metoda `PDOStatement::bindColumn()` mora biti nakon poziva `PDOStatement::execute()` metoda kako bi PDO znao koliko kolona će biti u rezultatu.

3.4. Zaštita od PHP SQL neželjenih upada u bazu podataka upotrebom PDO pripremljenih iskaza

Do **SQL neželjenog upada** (engl. *SQL Injection*) može doći u aplikacijama koje prihvataju podatke sa ulaza koji nisu prethodno validirani. Zbog toga je bezbednost veoma vazan deo svake softverske aplikacije. SQL neželjeni upad trenutno je jedan od najčešćih napada na aplikacije, a potencijalno osetljive kategorije su sve aplikacije koje koriste baze podataka.

Cilj napadača je da upotrebom neispravnog koda ugrozi sistem što može dovesti do krađe osetljivih podatka, brisanja podataka ili unošenja neželjenih podataka u bazu.

Postoje dve osnovne vrste napada. **Napad prvog reda** se dešava kada napadač unese neispravan string koji izaziva trenutno izvršenje izmenjenog koda. Na ovaj način napadač odmah dobija željene informacije. **Napad drugog reda** je kada napadač ubacuje određene podatke koji će ostati u bazi podataka, ali do napada ne dolazi odmah, već će on biti izazvan nekom drugom akcijom.

SQL neželjeni upad ne mora da bude problem jer postoji nekoliko veoma dobrih načina da se ovakve neželjene akcije izbegnu. PDO omogućava zaštitu od SQL upada upotrebom pripremljenih iskaza.

Kao što je rečeno u prethodnom poglavlju, pripremljeni iskazi koriste čuvare mesta za podatke, a ne same podatke u upitima. SQL upadi se dešavaju u slučajevima kada napadač uspe da uključi dodatne informacije u sam iskaz. Upotrebom pripremljenih iskaza, izbegava se rizik od neželjenog ubacivanja podataka u bazu.

Čitava ideja sprečavanja napada upotrebom pripremljenih iskaza se zasniva na sledećem – upit i podaci se šalju odvojeno SQL serveru. Osnova problema kod SQL neželjenih upada u bazu leži u mešanju koda i podataka.

Dodavanjem promenljivih u SQL upit činimo ga dinamičnim:

```
$username = '1' or '1' = '1

SELECT *
FROM emguser
WHERE username = $username
```

Do neželjenih akcija dolazi dodavanjem podataka direktno u program čime oni postaju njegov deo. Kod pripremljenih iskaza program se ne menja, ostaje netaknut i kao takav prvo se prosleđuje serveru. Tek u drugom zahtevu šaljemo podatke.

```
$sql ='SELECT *from emguser WHERE username= ?'
$query = $db->prepare($sql);

$query->execute(array($data));
```

Do ovog trenutka upit je već izvršen i podaci ga ne mogu izmeniti ili mu naneti bilo kakvu štetu. Za sve dodatne informacije o SQL upadima pogledati [9].

3.5. Rukovajne velikim objektima

Ponekad se javlja potreba za skladištenjem **velikih objekata** (engl. *large objects - LOB*) u bazu podataka. Velikim se obično smatraju objekti koji su većiod 4KB. Veliki objekti mogu biti znakovnog tipa (engl. *character large object - CLOB*) ili binarnog tipa (engl. *binary large object - BLOB*). Kada se veliki objekat preuzme iz rezultujućeg skupa, može se tretirati kao string. Medjutim, ovakav način rukovanja velikom objektima može opteretiti PHP server više nego što je to neophodno. PDO dozvoljava rad sa velikim objektima upotrebom `PDO::PARAM_LOB` u toku pozivanja `PDOStatemnt::bindParam()` metoda ili `PDOStatemnt::bindColumn()` metoda.

Sledeći primer vezuje veliki objekat za promenljivu `$lob` i potom je šalje na izlaz korišćenjem `fpassthru()`. U ovom primeru ilustruje se prikazivanje slike iz baze podataka:

```
<?php
$db = new PDO("mysql:host=$host;dbname=$db", $user, $pass);
$stmt = $db->prepare("select type, lob from images where id=?");
$stmt->execute(array($_GET['id']));
$stmt->bindColumn(1, $type, PDO::PARAM_STR, 256);
$stmt->bindColumn(2, $lob, PDO::PARAM_LOB);
$stmt->fetch(PDO::FETCH_BOUND);

header("Tip sadrzaja: $type");
```

```
fpassthru($lob);
?>
```

`PDOStatemnt::bindColumn()` uvezuje kolonu sa PHP promenljivom (koja je dobijena sa ulaza upotrebom `$_GET`metoda). Uvezivanje se može vršiti prosleđivanjem imena kolone ili njenog rednog broja.

Za više informacija o velikom objektima videti [6].

4. RUKOVANJE GREŠKAMA

4.1. Uopšteno o greškama

Prethodni primeri ne obezbeđuju nikakvu proveru postojanja greške, tako da nisu korisni za prave aplikacije. **Rukovanje greškama** (engl. *error handling*) predstavlja bitan aspekt svih veb aplikacija. Ne samo da će one obavestiti korisnike o stanju greške, već će ograničiti štetu ukoliko se greška detektuje čim se dogodi.

Ako radimo sa bazom podataka, trebalo bi da proverimo da li postoje greške prilikom otvaranja konekcije ka bazi podataka, prikom odabira baze podataka i posle pozivanja svakog upita. U mnogim veb aplikacijama, međutim, samo je potrebno prikazati poruku sa greškom kada nešto nije kako treba (bez zalaženja u detalje vezano za grešku, što bi moglo da otkrije neke osetljive informacije). Međutim, kada se debaguje greška, javlja se potreba za najdetaljnijim mogućim informacijama o grešci kako bi debagovanje moglo da se obavi u što kraćem roku.

Jedan pojednostavljen slučaj bi bio da se prekine izvršavanje skripta i prikaže poruka o grešci . U zavisnosti od baze podataka, kôd bi mogao da izgleda ovako:

```
//SQLite:
$dbh = sqlite_open('/path/to/emg.ldb', 0666)
      or die('Error opening SQLite database: ' .
sqlite_error_string(sqlite_last_error($dbh)));
$q = sqlite_query("SELECT DISTINCT username
                   FROM emguser
                   ORDER BY username ",$dbh)
      or die('Could not execute query because: ' .
sqlite_error_string(sqlite_last_error($dbh)));

//PostgreSQL:
pg_connect("host=localhost dbname=cars user=boss password=password")
      or die('Could not connect to
PostgreSQL: . pg_last_error());
$q = pg_query("SELECT DISTINCT username
            FROM emguser
            ORDER BY username ")
      or die('Could not execute query because: ' . pg_last_error());
```

Kao što možemo da vidimo, rukovanje greškama počinje da se razlikuje za SQLite u poređenju sa PosgreSQL.

Većina aplikacija ima veoma jednostavnu strategiju u rukovanju greškama. Kada se greška dogodi, skript se prekida i prikazuje se greška. Grešku bi trebalo upisati u logove[11], a programeri bi povremeno trebalo da ih proveravaju.

Pre nego što pogledamo kako da implementiramo istu strategiju za rukovanje greškama korišćenjem PDO, pogledajmo najpre koji su najčešći uzroci grešaka u veb aplikacijama koje koriste baze podataka:

- Pad servera ili preopterećenje servera; jedna od poznatih grešaka ovog tipa je greška "previše konekcija"
- Nepravilno konfigurisanje aplikacije, koje se može desiti ako koristimo pogrešan string za konekciju, veoma česta greška kod premeštanja aplikacije sa jednog domenskog servera na drugi
- Nepravilna validacija korisnika, koja vodi do neispravnog SQL-a, a samim tim i neuspešnog upita
- Ubacivanje zapisa u bazu sa duplim primarnim ključem ili jedinstvenom indeksvrednošću
- Sintaksne greške u SQL iskazima.

Da bi se napravila dobra strategija u rukovanju greškama, potrebno je izanalizirati gde se sve greška može javiti. Greške se mogu javiti prilikom postavljnja svakog upita nad bazom, iako je ovo jako često, razmotrićemo ovakvu situaciju. Pre nego što to odradimo, pozabavićemo se pojedinačno mogućim izvorima grešaka i definisati strategiju kako upravljati njima.

4.1.1. Pad server ili preopterećenje servera

Ovo se dešava ukoliko je server veoma zauzet, tako da ne može primiti više dolazećih konekcija. Na primer, ukoliko je u pozadini pokrenut skript za izmenu podataka u bazi kome je potrebno dosta vremena da se izvrši. Ovo dovodi do toga da ne postoji mogućnost dobijanja bilo kakvih podataka iz baze i onda se moraju preduzeti određene akcije.

Ako PDO konstruktor ne uspe da napravi konekciju, prikazujemo stranicu sa porukom koja će korisniku dati upozorenje da zahtev ne može biti ispunjen u ovom trenutku i da bi trebalo pokušati kasnije. Takođe, ovakva greška bi trebalo da bude upisana u log jer može zahtevati trenutnu pažnju. (Dobra ideja bi bila slanje imejl poruke administartoru o ovoj grešci.)

[11] Upotrebom `error_log()` metoda u log fajl se upisuju informacije o greškama. Log fajl je fajl u koji računarski sistem upisuje informacije o svojim aktivnostima.

Ova greška se obično manifestuje pre nego što je konekcija uopšte i uspostavljena (prilikom pozivanja PDO konstruktora). Međutim, postoji mali rizik da se ona desi nakon konekcije (prilikom poziva PDO ili PDOStatement objekta kada server više nije dostupan). U ovakvim situacijama, reakcija će biti ista – predstaviti korisniku poruku o nastaloj grešci.

4.1.2. Nepravilno konfigurisanje aplikacije

Ova greška nastaje jedino u slučaju migracije aplikacije na drugi domenski server za koji su drugačiji detalji potrebni za pristup bazi podataka. Dakle, ovakva greška se ne javlja prilikom redovnog izvršavanja aplikacije.

4.1.3. Nepravilna validacija podataka

Ovakva greška je veoma povezana sa SQL neželjenim ubacivajem podataka u bazu (SQL injection). Prilikom razvijanja bilo koje aplikacije moraju se preduzeti potrebne mere za validaciju i filtriranje svih korisničkih unosa. Ovakva greška vodi do dve glavne posledice: ili se upit neće ni izvršiti zbog nepravilnog SQL izraza ili dolazi do upisa neželjenih podataka u bazu i njena sigurnost se dovodi u pitanje. Iako se posledice za oba problema razlikuju, u oba slučaja prevencija se vrši na isti način.

Razmotrićemo sledeću situaciju. Prihvatamo numeričku vrednost sa ulaza i ubacujemo je u bazu podataka. Videćemo kako se korišćenjem oskudno definisanog skripta javljaju grešake i ceo sistem dovodi u rizik.

Formular koji skript procesira, prihvatiće dve promenljive: `$_REQUEST['userid']`, `$_REQUEST['throughput']`. Ukoliko ne postoji validacija ovih vrednosti, finalni kôd će izgledati ovako:

```
$userid = $_REQUEST['userid'];
$throughput = $_REQUEST['throughput'];
$sql = "UPDATE emguser SET throughput = $throughput WHERE
      userid=$userid ";
$conn->query($sql);
```

Ukoliko ostavimo polje *throughput* prazno u formularu, finalni SQL upit bi bio:

```
UPDATE emguser SET throughput = WHERE userid=1
```

Ovaj SQL upit je pogrešno napisan i dovešće do sintaksne greške. Zbog toga je neophodno da budemo sigurni da će svaka promenljiva čuvati podatke odgovarajućeg tipa. Ukoliko nisu odgovarajućeg tipa, trebalo bi ponovo prikazati formular sa odgovarajućom porukom o grešci.

Pogledajmo sada kako bi jedan napadač mogao da iskoristi ovo i obriše sadržaj čitave tabele. Da bi to postigao, dovoljno je uneti sledeće:

```
100; DELETE FROM emguser;
```

Ovo pretvara jedan skript u tri upita:

```
UPDATE emguser SET throughput = 100; DELETE FROM emguser;
WHERE userid = 1;
```

Naravno, treći skript je neispravan, ali prvi i drugi će se izvršiti i server će prijaviti grešku. Da bismo izbegli ovaj problem, dovoljno je da koristimo jednostavnu validaciju da bismo bili sigurni da će svaka promenljiva biti u formatu koji očekujemo. Međutim, ako imamo tekstualna polja koja sadrže proizvoljne karaktere, vrednosti polja se moraju proveriti pre kreiranja samog SQL upita.

4.1.4. Ubacivanje zapisa u bazu sa dupliranim primarnim ključem ili jedinstvenom vrednošću indeksa

Ovo se dešava ako pomoću aplikacije pokušamo da ubacimo red u bazu sa primarnim ključem ili jedinstvenim indeksom koji već postoje u bazi podataka. Ovo se može rešiti kreiranjem unikatnog indeksa nad kolonom tabele. Svaki pokušaj ubacivanja iste vrednostu u kolonu nad kojom je kreiran indeks generisaće grešku. Greška se sada može uhvatiti i primeniti reakcija u skladu sa njom.

4.1.5. Sintaksne greške u SQL upitima

Ova greška se javlja usled nedovoljnog testiranja aplikacije. Dobra aplikacija ne sme sadržati ovakve greške i tim koji je kreira ima odgovornost da testira svaku moguću situaciju i uveri se da će svaki SQL upit biti bez sintaksnih grešaka.

Ukoliko ipak dođe do ovakvog tipa greške, tada se ona hvata izuzetkom i prikazuje poruka o fatalnoj grešci. Za više detalja videti [7].

Nakon pregleda mogućih izvora grešaka, možemo preći na izučavanje kako se pomoću PDO-a rukuje odgovarajućim greškama.

4.2. Tipovi rukovanja greškama u PDO-u

Podrazumevani mod kod PDO-a je tzv. **tihi režim za rukovanje greškama** (engl. *silent error handling mode*). Ovo znači da bilo koja grešaka, koja se javi prilikom poziva metoda PDO ili PDOStatement klasa, prolaze neprijavljeno. Sa ovim režimom, bilo bi neophodno pozivati `PDO::errorInfo()`, `PDO::errorCode()`, `PDOStatement::errorInfo()`, ili `PDOStatement::errorCode()` svaki put kada se greška desi kako bismo bili sigurni da li je do greške zaista došlo. Možemo da primetimo da je ovaj režim sličan tradicionalnom načinu pristupa bazi podataka – obično, kod poziva `mysql_errno()` i `mysql_error()` (ili neke ekvivalentne funkcije kod drugih sistema za upravljanje bazom podataka), nakon poziva funkcija koje mogu izazvati grešku, nakon konektovanja sa bazom podataka i nakon pokretanja nekog upita.

Pored tihog režima postoji i **režim upozorenja** (engl. *warning mode*)**.** Ovde će se PDO ponašati identično tradicionalnom pristupu. Bilo koja greška, koja se dogodi prilikom komunikacije sa bazom podataka, dovešće do `E_WARNING` greške. U zavisnosti od konfiguracije, poruka o grešci se može prikazati ili upisati u log fajl.

Na posletku, PDO uvodi moderan način rukovanja greškama prilikom konekcije sa bazom podataka – upotrebom **izuzetaka**. Bilo koji neuspešan poziv PDO ili PDOStatement metoda izbaciće izuzetak.

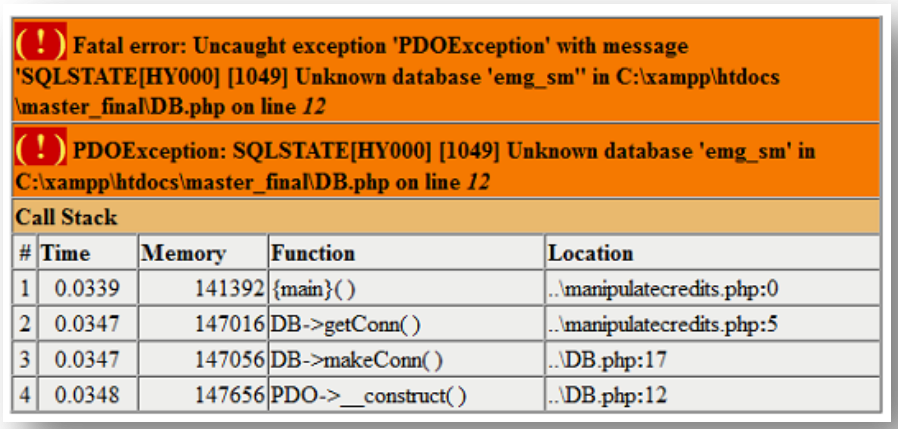
Kao što smo već napomenuli, PDO koristi tihi režim za rukovanje greškama kao podrazumevani. Prebacivanje na željeni režim za upravljanje greškama postižemo pozivom `PDO::setAttribute()` metoda. Svaki od režima je specifikovan sledećim konstantama koje su definisane u PDO klasi:

- `PDO::ERRMODE_SILENT` – *tiha* strategija.
- `PDO::ERRMODE_WARNING` – strategija *upozorenja*.
- `PDO::ERRMODE_EXCEPTION` – koristi *izuzetke.*

Za podešavanje željenog režima, potrebno je podesiti `PDO::ATTR_ERRMODE` atribut na sledeći način:

```
$conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

Da bismo videli kako PDO izbacuje neki izuzetak, izmenićemo php file za konekciju dodavanjem prethodnog iskaza. Izmenićemo string za konekciju tako da pokazuje na nepostojeću bazu podataka. Javlja se sledeća greška:



Fatal error: Uncaught exception 'PDOException' with message 'SQLSTATE[HY000] [1049] Unknown database 'emg_sm'' in C:\xampp\htdocs\master_final\DB.php on line *12*

PDOException: SQLSTATE[HY000] [1049] Unknown database 'emg_sm' in C:\xampp\htdocs\master_final\DB.php on line *12*

Call Stack

#	Time	Memory	Function	Location
1	0.0339	141392	{main}( )	..\manipulatecredits.php:0
2	0.0347	147016	DB->getConn( )	..\manipulatecredits.php:5
3	0.0347	147056	DB->makeConn( )	..\DB.php:17
4	0.0348	147656	PDO->__construct( )	..\DB.php:12

Ovo je PHP-ova podrazumevana reakcija na neuhvaćen izuzetak – smatraju se fatalnim greškama i izvršavanje programa se prekida. Poruka o grešci otkriva klasu izuzetka, PDOException, opis greške i neke informacije o debagovanju, uključujući ime i broj linije u kodu koja je prouzrokovala izuzetak. Primetimo, da ukoliko želimo da testiramo SQLite, navođenje nepostojeće baze podataka možda neće raditi pošto će se baza kreirati u slučaju daveć ne postoji. Za potrebe testiranja pogodnije je promeniti `$connStr` promeljivu tako da sadrži neki nedozvoljeni karakter u imenu baze.

```
$connStr = 'sqlite:/path/to/pdo*.db';
```

Pojaviće se greška slična onoj iz prethodnog primera, uz preciziranje uzroka i lokacije greške u izvornom kôdu.

4.3. Definisanje funkcije za upravljanje greškama

Ako znamo sa sigurnošću da će određeni iskaz ili segment koda da izbaci neki izuzetak, možemo taj deo koda da ubacimo u `try…catch` blok kako bismo sprečili prikazivanje podrazumevanih poruka sa greškama i učinili stranicu sa greškom prilagodljivijom za korisnika. U ovome nam mnogo može pomoći funkcija koja prikazuje poruku i omogućava izlaz iz aplikacije. Pošto postoji potreba da se ova funkcija poziva iz

različitih skriptova, najbolje mesto za čuvanje ovakve funkcije je common.inc.php[12] fajl. Ova funkcija će raditi sledeće:

- Prikazuje poruku o grešci. Koristi `htmlspecialchars`[13] funkciju i obrađuje tekst pomoću `nl2br`[14] funkcije kako bi se mogle koristiti poruke koje sadrže veći broj redova.

Pored ovoga, moraćemo da izmenimo segment koda koji kreira objekat konekcije u common.inc.php kako bismo uhvatili mogući izuzetak. Fajl će izgledati ovako:

```
<?php

// String za konkciju, korisnicko ime i lozinka
$connStr = 'mysql:host=localhost;dbname=pdo';
$user = 'root';
$pass = 'root';

function showError($message)
	{
		echo "<h2>Error</h2>";
		echo nl2br(htmlspecialchars($message));
		exit();
	}

// Kreiranje objekta konekcije
try
	{
		$conn = new PDO($connStr, $user, $pass);
		$conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
	}

catch(PDOException $e)
	{
		showError("Doslo je do greske. Molimo pokusajte sa zahtevom
			kasnije\n" . $e->getMessage());
	}
?>
```

Kao što smo videli ranije, delove koda koji potencijalno mogu izbaciti neki izuzetak, stavljamo u `try…catch` blok. Međutim, u nekim retkim slučajevima, mogu se javiti neočekivani izuzeci.

[12] Proceduralna datoteka koja sadrži zajedničke informacije koje su potrebne svim delovima aplikacije. Izvorno se nalazi na lokaciji /phpDocumentor/common.inc.php

[13] `htmlspecialchars` je PHP-ova bibliotečka funkcija koja konvertuje specijalne karaktere u HTML kodove.

[14] `nl2br` je PHP-ova bibliotečka funkcija koja konvertuje sve karaktere za novi red u `<br>` etikete.

Da bismo ispravno upravljali ovakvom situacijom, potrebno je da napravimo globalnu funkciju koja će rukovati greškama ili da svaki deo koda koji poziva PDO ili PDOStatement klasu stavimo u `try…catch`blok.

Sada ćemo pogledati kako možemo kreirati funkciju. Ovo je jednostavniji pristup jer ne zahteva menjanje mnogo koda.

```
function exceptionHandler($e)
      {
            showError("Zao nam je, sajt je u fazi izrade.\n" .
            $e->getMessage());
      }
// Podesavanje globalnog upravljača greškama
set_exception_handler('exceptionHandler');
```

Ovu funkciju stavljamo u common.inc.php. Uvek je dobra ideja imati podrazumevani globalni upravljač greškama, jer neželjeni izuzeci mogu prikazati osetljive podatke ukljucujući i podatke za konekciju sa bazom podataka. Podrazumevani upravljač bi trebalo da upiše grešku u log i da obavesti osobe koje rade na održavanju aplikacije o greškama.

5. NAPREDNO KORIŠĆENJE PDO-A

5.1. Podešavanje atributa za konekciju i dobijanje njihovih vrednosti

Sada, kada su razmotrene neke osnovne karakteristike PDO-a, možemo se osvrnuti i na neke naprednije funkcionalnosti.

Već smo ranije pominjali podešavanje atributa za konekciju u odeljku za upravljanje greškama i izveštavanju o njima. Atributi za konekciju omogućavaju kontrolu određenih aspekata konekcije, kao i dobijanje podataka o imenu drajvera i njegovoj verziji.

- Jedan od načina je označavanje niza parova koji se sastoje iz imena atributa i vrednosti u PDO konstruktoru
- Drugi način je pozivanjem `PDO::setAttribute()` metoda, koji prihvata dva parametra: ime atributa i vrednost atributa.

U PDO-u, atributi i njihove vrednosti su definisani kao konstante u PDO klasi kao što je to u sledećem pozivu:

```
$conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

Poziv uključuje dve takve konstante `PDO::ATTR_ERRMODE` i `PDO::ERRMODE_EXCEPTION`.

Za dobijanje vrednosti nekog atributa koristi se `PDO::getAttribute()` metod. Ovaj metod prihvata jedan parameter, ime atributa i vraća vrednost atributa. Na primer, sledeći kôd će izbaciti izuzetak:

```
if($conn->getAttribute(PDO::ATTR_ERRMODE) == PDO::ERRMODE_EXCEPTION)
      {
            echo 'Exception';
      }
```

Pogledajmo sada koji sve atributi za konekciju postoje u PDO.

- **`PDO::ATR_CASE`**. Ovaj atribut kontroliše izgled imena kolona koje su vraćene `PDO::Statement::fetch()` metodom. Koristan je u slučajevima kada je režim za dovlačenje podataka `PDO::FETCH_ASSOC` ili `PDO::FETCH_BOTH` (pošto se kolone

ovim metodom vrate kao niz koji sadrži imena kolona indeksiranih po njihovom imenu). Ovaj atribut može imati jednu od sledeće tri vrednosti: `PDO::CASE_LOWER`, `PDO::CASE_NATURAL` i `PDO::CASE_UPPER`. Zavisno od ove vrednosti, imena kolona će biti zapisana malim slovima, ostavljena bez ikakvih promena, ili zapisana velikim slovima, kao što će sledeći skript:

```
$conn->setAttribute(PDO::ATTR_CASE, PDO::CASE_UPPER);
$stmt = $conn->query("SELECT * FROM operators LIMIT 1");
$r = $stmt->fetch(PDO::FETCH_ASSOC);
$stmt->closeCursor();
var_dump($r);
```

dati izlaz:

```
array(size=9)
  'ID' =>string '131' (length=3)
  'MCC' =>string '131' (length=3)
  'MNC' =>string '' (length=0)
  'MCCMNC' =>string '131' (length=3)
  'COUNTRY' =>string 'USA' (length=3)
  'OPERATOR' =>string 'Default' (length=7)
  'PRICE' =>string '0.024' (length=5)
  'COMMENT' =>string '' (length=0)
  'MARGIN' =>string '15' (length=2)
```

Uobičajena praksa je da se imena kolona ne menjaju pa je iz tog razloga i najčešća upotreba `PDO::CASE_NATURAL`.

- **`PDO::ATTR_ORACLE_NULLS`:** Ovaj atribut je, bez obzira na svoje ime, primenljiv na svim sistemima za upravljanje bazom podataka, ne samo za Oracle. On kontroliše kako se vrednosti NULL i prazni stringovi prosleđuju u PHP-u. Moguće vrednosti su `PDO::NULL_NATURAL` (ne dolazi do bilo kakve transformacije), `PDO::NULL_EMPTY_STRING` (prazni stringovi se zamenjuju PHP-ovovom NULL vrednošću) i `PDO::NULL_TO_STRING` (SQL NULL vrednosti se konvertuju u prazan string u PHP-u).
  Sledećim primerom je ilustrovano kako se ovaj atribut koristi:

```
$conn->setAttribute(PDO::ATTR_ORACLE_NULLS, PDO::NULL_TO_STRING);
$stmt = $conn->query("SELECT * FROM emguser WHERE route IS
                  NULL LIMIT 1");
$r = $stmt->fetch(PDO::FETCH_ASSOC);
$stmt->closeCursor();
var_dump($r);
```

Izlaz je:

```
array(size=20)
  'userid' =>string '1' (length=1)
  'username' =>string 'admin' (length=5)
  'password' =>string 'pass' (length=6)
```

```
'maxsessions' =>string '' (length=0)
'throughput' =>string '10' (length=2)
'lastlogin' =>string '2014-05-10 08:07:51' (length=19)
'lastfailedlogin' =>string '' (length=0)
'lastip' =>string '2130706433' (length=10)
'allowpostpaid' =>string '' (length=0)
'usergroup' =>string '' (length=0)
'fullname' =>string '' (length=0)
'email' =>string '' (length=0)
'route' =>string '' (length=0)
'routing' =>string '/etc/emg/routes/global' (length=22)
'company' =>string '' (length=0)
'phone' =>string '' (length=0)
'extra1' =>string 'http://127.0.0.1' (length=16)
'extra2' =>string '' (length=0)
'extra3' =>string '' (length=0)
'charge_balance' =>string '70.0000' (length=7)
```

Uočavamo da je vrednost za određena polja označena kao string, a ne kao NULL (što bi bio slučaj da nije korišćen`PDO::ATTR_ORACLE_NULLS` atribut).

- **`PDO::ATTR_ERRORMODE`**: Ovim atributom se podešava režim za izveštavanje o konekciji. Prihvata tri vrednosti:
    - `PDO::ERRORMODE_SILENT`: ne dolazi do akcije, a kôd greške je dostupan preko: `PDO::errorCode()` i `PDO::errorInfo()` metoda (ili njihovih ekvivalenata u PDOStatement klasi). Ovo je podrazumevana vrednost.
    - `PDO::ERROR_WARNINIG`: Isto kao i u prethodnom slučaju, nema akcije, ali greška će biti uzdignuta na E_WARNING nivo.
    - `PDO::ERRORMODE_EXCEPTION` će podesiti kodove grešaka (kao sa `PDO::ERROR_SILENT`) i izuzetak klase PDOException će biti izbačen.

  Prethodna tri atributa su za upisivanje i iščitavanje (read/write). Postoje takođe atributi samo za iščitavanje, dostupni upotrebom `PDO::getAttribute()`metoda. Ovi atributi vraćaju string vrednosti (za razliku od konstanti definisanih u PDO klasi).
- **`PDO::ATTR_DRIVER_NAME`**: Vraća ime osnovnog drajvera baze podataka.

```
echo $conn->getAttribute(PDO::ATTR_DRIVER_NAME);
(vraca mysql ako koristimo taj sistem)
```

- **`PDO::ATTR_CLIENT_VERSION`:** Vraća ime osnovne verzije klijentske biblioteke za bazu podataka.
- **`PDO_ATTR_SERVER_VERSION`:** Vraća verziju server baze podataka za koji je aplikacija konektovana.

5.2. MySQL baferisani upiti

Ukoliko radimo samo sa MySQL bazom podataka, postoji mogućnost korišćenja MySQL-ovog PDO drajver režima za baferisane upite. Kada je uspostavljena konekcija sa režimom za baferisane upite, čitav rezultujući skup za svaku `SELECT` naredbu je, pre nego što je prosleđen aplikaciji, učitan u memoriju. Ovo nam daje jednu korist – možemo koristiti `PDOStatement::rowcount()` metod za određivanje broja redova koje sadrži rezultujući skup.

Za prebacivanje PDO-a u MySQL režim za baferisane upite, potrebno je da se označi `PDO::MYSQL_ATTR_USE_BUFFERED_QUERY` atribut za konekciju, kao u sledećem primeru:

```
$conn = new PDO($connStr, $user, $pass);
$conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
$conn->setAttribute(PDO::MYSQL_ATTR_USE_BUFFERED_QUERY, 1);
$q = $conn->query("SELECT * FROM operators”);
echo $q->rowCount();
```

Izlaz će biti broj vraćenih redova.

Važno je napomenuti da se ovaj atribut primenjuje samo za MySQL baze podataka i nije portabilan između ostalih sistema. Još je bitno istaći da su baferisani upiti, koji vraćaju velike rezultujuće skupove, jako skupi u pogledu resursa i da ih zbog toga treba izbegavati. Ukoliko želimo da koristimo baferisane upite, moramo biti sigurni da su onemogućeni prilikom pokretanja upita koji vraćaju ogroman broj redova. Ovo možemo učiniti na sledeći način:

```
$conn->setAttribute(PDO::MYSQL_ATTR_USE_BUFFERED_QUERY, 0);
```

Ukoliko želimo da proverimo da li su MySQL baferisani upiti omogućeni,to možemo učiniti na sledeći način:

```
$conn->getAttribute(PDO::MYSQL_ATTR_USE_BUFFERED_QUERY);
```

5.3. Konekcija sa bazom upotrebom fajla za konfiguraciju konekcije i podešavanje php.ini fajla

Ranije je bilo reči o stringovima za konekciju i tada smo videli da string za konekciju počinje imenom drajvera iza koga stoji znak tačka-zarez. PDO takođe podržava i fajlove za konfiguraciju – fajl koji sadrži string za konekciju.

```
mysql:host=localhost;dbname=pdo
ili
sqlite:/www/hosts/localhost/pdo.db
```

Tada u PDO konstruktoru, navodimo putanju do konfiguracionog fajla ili URL, ne samo string za konekciju:

```
uri:./pdo.dsn
```

PDO će iščitati ovaj fajl i koristiti string za konekciju koji je smešten u njemu. Prednost korišćena ovog metoda je što se ne mora navesti samo lokalni fajl već bilo koji URL, tako da i udaljeni fajlovi mogu biti uključeni. Sa druge strane, ako fajl nije propisno zaštićen od pristupa svih korisnika putem Veba, tada se može desiti da pouzdane informacije procure, tako da je neophodno voditi računa o sigurnosti prilikom upotrebe ovog metoda.

Još jedan od načina za navođenje stringa za konekciju je u php.ini fajlu. Na primer, možemo definisati direktive u php.ini fajlu.

```
pdo.dsn.mysql = mysql:host=localhost;dbname=pdo
pdo.dsn.sqlite = sqlite:/www/hosts/localhost/pdo.db
```

Tada je moguće proslediti *mysql* ili *sqlite* stringove PDO konstruktoru umesto čitavog stringa za konekciju za mysql i sqlite, redom:

```
$conn = new PDO('mysql', $user, $pass);
$conn = new PDO('sqlite', $user, $pass);
```

5.4. Transakcije

PDO API takođe standardizuje metode za upravljanje transakcijama. Nakon uspešnog kreiranja PDO konekcije, podešen je **autocommit režim**. Ovo znači da je za svaki sistem za upravljanje bazama podataka koji podržava transakcije, svaki upit smešten u jednu implicitnu transakciju. Za sisteme koji ne podržavaju transakcije, svaki upit se izvršava onako kako je zapisan u skriptu. Tipično, strategija upravljanja transakcijama izgleda ovako:

1. Početak transakcije.
2. Smeštanje koda vezanog za bazu podataka u `try…catch`blok.
3. Deo koda vezan za bazu (u `try…catch` bloku) bi trebalo da izvrši promene posle završetka svih update naredbi (commit naredba).
4. `Catch` blok bi trebalo da poništi efekte transakcije (rollback naredba).

Naravno, samo kodom koji menja bazu podataka (ažuriranje, ubacivanje, brisanje), kao i kodom koji može da naruši integritet baze podataka, bi trebalo da se rukuje u transakcijama. Ako se bilo šta loše dogodi tokom transakcije, podaci u bazi neće biti izmenjeni i njihoj integritet je očuvan.

PDO nudi tri metoda za rukovanje transakcijama: `PDO::beginTransaction()` koji započinje transakciju, `PDO::commit()` koji izvršava promene napravljene posle poziva `PDO::beginTransaction()`, i `PDO::rollback()` koji poništava sve promene od trenutka kada je transakcija započeta.

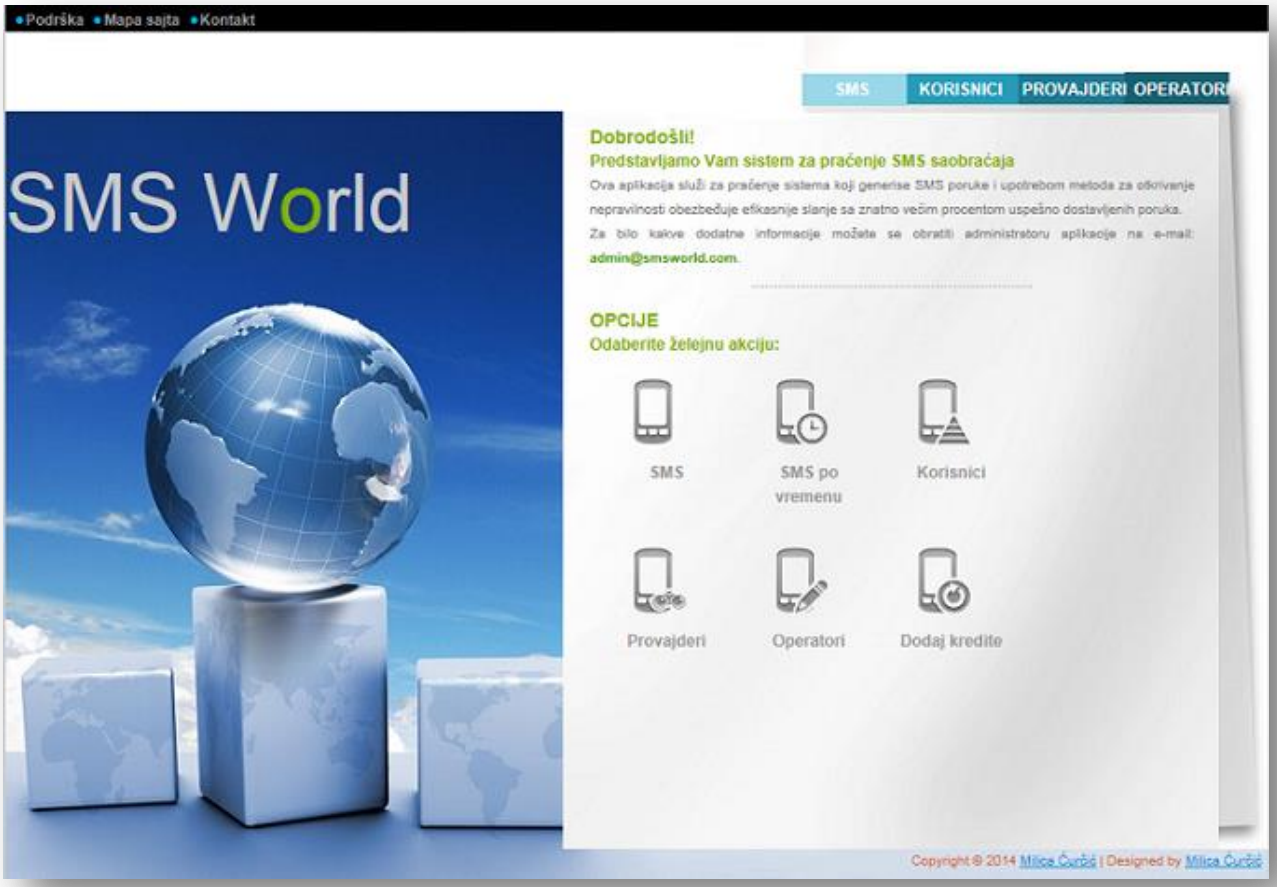
`PDO::beginTransaction()` metod ne prihvata nijedan parameter i vraća vrednost logičkog tipa (boolean) u zavisnosti od toga da li je transakcija uspešno započeta. U slučaju da se poziv ovog metoda završi neuspešno, PDO će izbaciti izuzetak. Slično, `PDO::rollback()` metod će izbaciti izuzetak ukoliko nema aktivne transakcije. Isto će se desiti ukoliko se poziv `PDO::commit()` metoda obavi pre poziva `PDO::beginTransaction()`. (Naravno, režim za upravljanje greškama mora biti podešen na `PDO::ERRMODE_EXCEPTION` kako bi izuzeci mogli biti izbačeni).

Važno je napomenuti da ne bi trebalo koristiti direktne upite za kontrolu transakcija prilikom upotrebe PDO-a. To zapravo znači da ne bi trebalo koristiti upite kao što su `BEGIN TRANSACTION`, `COMMIT`, `ROLLBACK` u `PDO::query()` metodu. U suprotnom, ponašanje ova tri metoda neće biti konzistentno.

6. APLIKACIJA ZA PRAĆENJE SMS SAOBRAĆAJA

Aplikacija je pisana u PHP programskom jeziku, verzija 5.4.7; sistem za upravljanje bazom podataka je MySQL Server verija 5.5.28;server je Apache 2.4 Handler Apache Lounge, razvojno okruženje NetBeans IDE 7.3. Korišćena je i JavaScript JQuery biblioteka.

Aplikacija je nastala za potrebe praćenja sistema koji služi kao posrednik između korisnika i operatera prilikom slanja SMS poruka.



Pre nego što vidimo kako se koristi aplikacija za praćenje sistema, pogledajmo najpre, ukratko, kako funkcioniše sam sistem koji aplikacija nagleda.

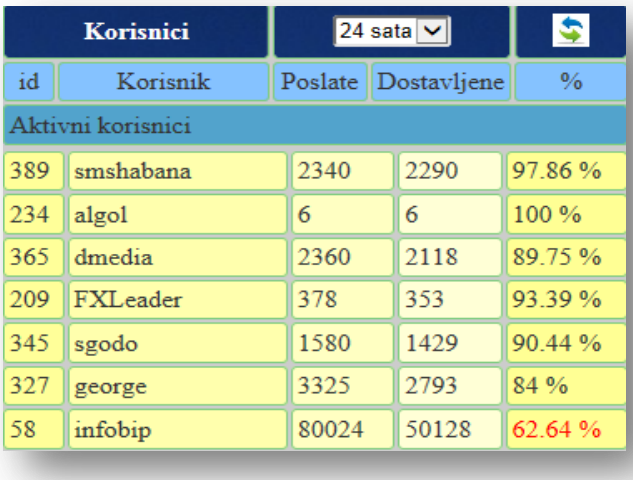
Svaka poruka poslata od strane korisnika upisuje se u bazu podataka. Odmah nakon slanja, poruka ima status “neodređen”, sve dok od provajdera ne dobije povratnu informaciju. Tada se njen status menja u “dostavljena” ukoliko je poruka uspešno stigla na

svoju destinaciju, a u suprotnom status se menja u neki od “nepoznat”, “neuspešan”, “u toku”, “obrisan”, “istekao”, “odbijen” i drugi.

6.1. Funkcionalnost aplikacije

Na osnovu podataka iz baze aplikacija za praćenje je u mogućnosti da otkrije koliko je uspešno slanje po različitim kategorijama.

Prva od kategorija koju aplikacija prati je uspešnost u dostavljanu poruka za svakog aktivnog korisnika:



Na osnovu procenta koji se prikazuje, imamo uvid o tome koliko je slanje uspešno/neuspešno po korisniku. Opcije za odabir različitog perioda za praćenje nam pružaju više mogućnosti za otkrivanje da li je problem samo trenutan ili ne. Ukoliko se ustanovi da je neuspešno slanje povezano samo sa određenim korisnikom(cima), tim koji ovo prati može da kontaktira korisnika(ke) kako bi se utvrdilo da li se slanje obavlja na ispravan način.

Sledeća kategorija se odnosi na provajdere:

Provajderi	5 min		
Provajder	Poslate	Dostavljene	%
beepsend	8624	7474	86.67 %
hlr-beepsend	3	3	100 %
http_in_01	1	0	0 %
routem	40979	25901	63.21 %

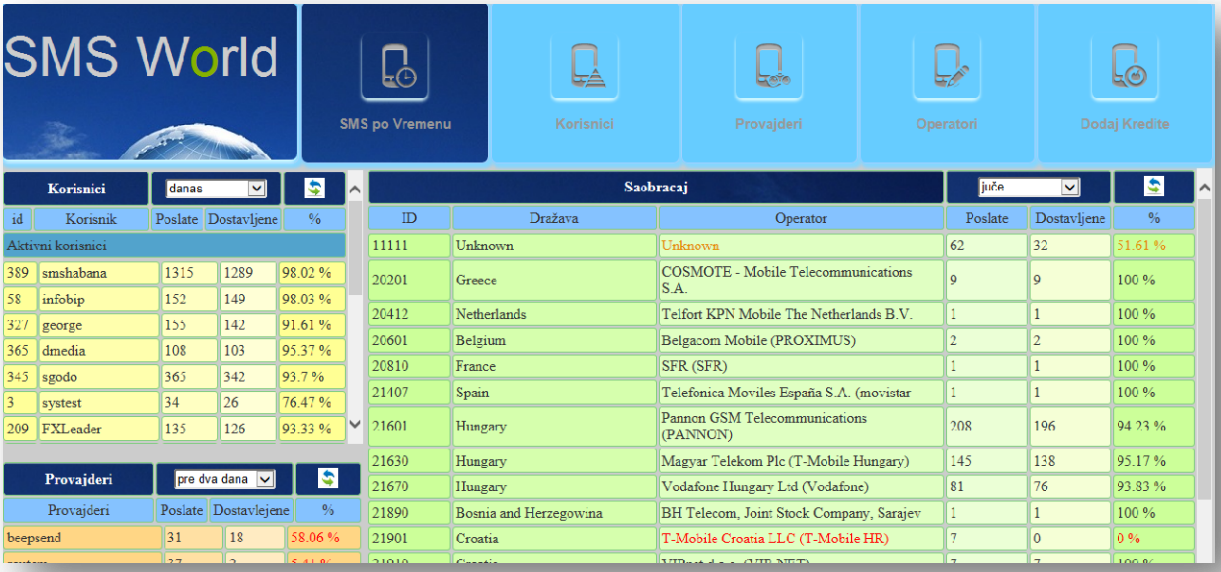
Aplikacija pruža informacije koliko je uspešna svaka ruta i na osnovu toga se može kontaktirati provajder kako bi se ruta poboljšala ili eventualno usmeriti saobraćaj na neku rutu koja ima veći procenat dostavljenih poruka.

Nadgledanje operatera se takođe može vršiti upotrebom aplikacije:

Saobraćaj			15 min		
ID	Država	Operator	Poslate	Dostavljene	%
20201	Greece	COSMOTE - Mobile Telecommunications S.A.	24	24	100 %
20205	Greece	Vodafone-Panafon (vodafone)	20	19	95 %
20210	Greece	Wind Hellas Telecommunications S.A. (WIN	1	1	100 %
20412	Netherlands	Telfort KPN Mobile The Netherlands B.V.	2	2	100 %
20416	Netherlands	T-Mobile Netherlands B.V. (T-Mobile NL)	1	1	100 %
20601	Belgium	Belgacom Mobile (PROXIMUS)	2	2	100 %
20800	France	Orange France	2	2	100 %
20810	France	SFR (SFR)	1	1	100 %
21403	Spain	France Telecom España SA (Orange)	1	1	100 %
21407	Spain	Telefonica Moviles España S.A. (movistar	3	3	100 %
21601	Hungary	Pannon GSM Telecommunications (PANNON)	2583	2305	89.24 %
21630	Hungary	Magyar Telekom Plc (T-Mobile Hungary)	420	373	88.81 %
21670	Hungary	Vodafone Hungary Ltd (Vodafone)	269	255	94.8 %
21805	Bosnia and Herzegowina	RS Telecommunications JSC Banja Luka (m	7	7	100 %

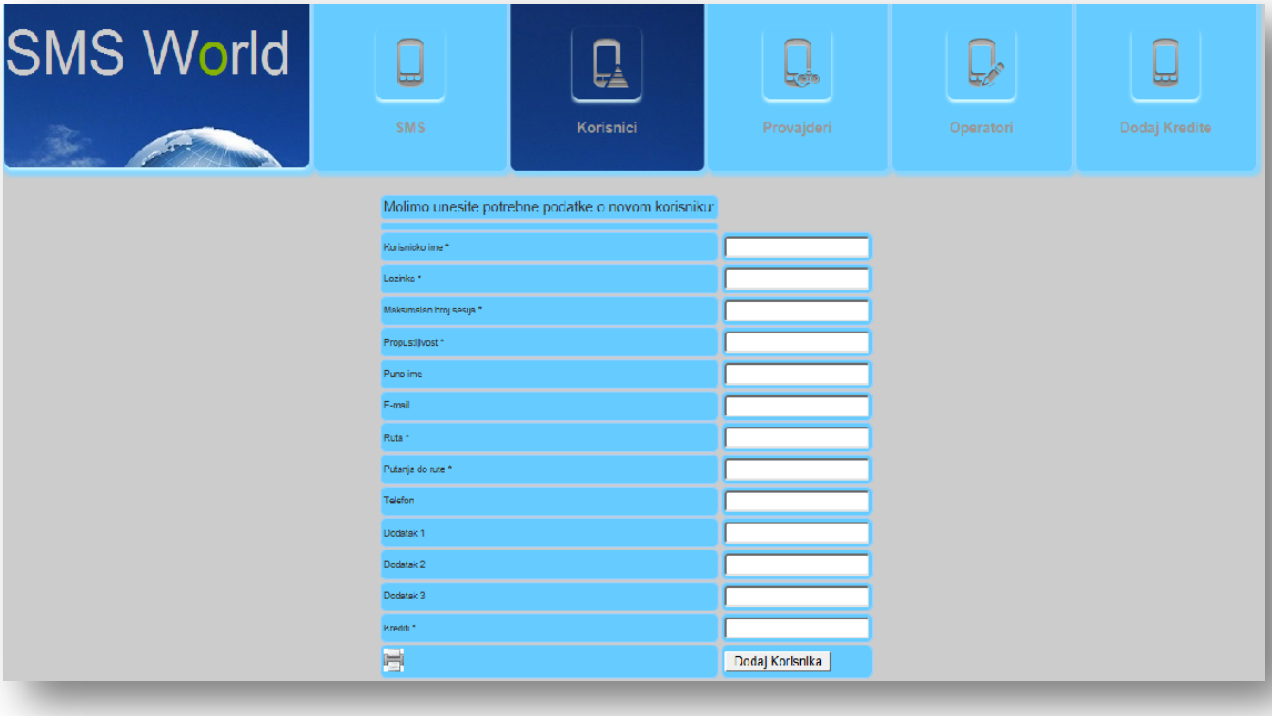
Upoređivanjem sa prethodnim informacijama, koje imamo o korisnicima I provajderima, može se preciznije locirati izvor problema, a potom i primeniti određene mere radi postizanja što boljih rezultata.

Osim globalnog pregleda slanja u protekih 24 sata (ili nekom manjem vremenskom periodu), u aplikaciji postoji mogućnost praćenja svega prethodnog i za neki raniji vremenski period.



Prelaskom na link “Korisnici”, otvara se mogućnost detaljnijeg praćenja saobraćaja za odabranog korisnika:

Kao i za dodavanje novog korisnika u bazu podataka:



Slično kao za korisnike, u aplikaciji postoji mogućnost praćenja svakog provajdera ili operatera pojedinačno, kao i dodavanje novih podataka u bazu vezano za sve prethodne entitete.

Slanjem poruka, korisnicima se umanjuju krediti koje poseduju. Da bi se uvećao iznos kredita, može se koristiti stranica aplikacije “Izmeni kredite”, koja nam to omogućava. Potrebno je samo odabrati željenog korisnika iz padajuće liste (unos nije moguć da bismo izbegli odabir nepostojećeg korisnika). Sada postoje tri opcije: prikazivanje trenutnog stanja kredita za odabranog korisnika, prikazivanje pregleda dodavanja/oduzimanja kredita ili izmena iznosa odabranog korisnika. Za izmenu iznosa potrebno je odabrati jednu od dve akcije, dodavanje ili oduzimanje kredita, uneti iznos i opciono upisati komentar.

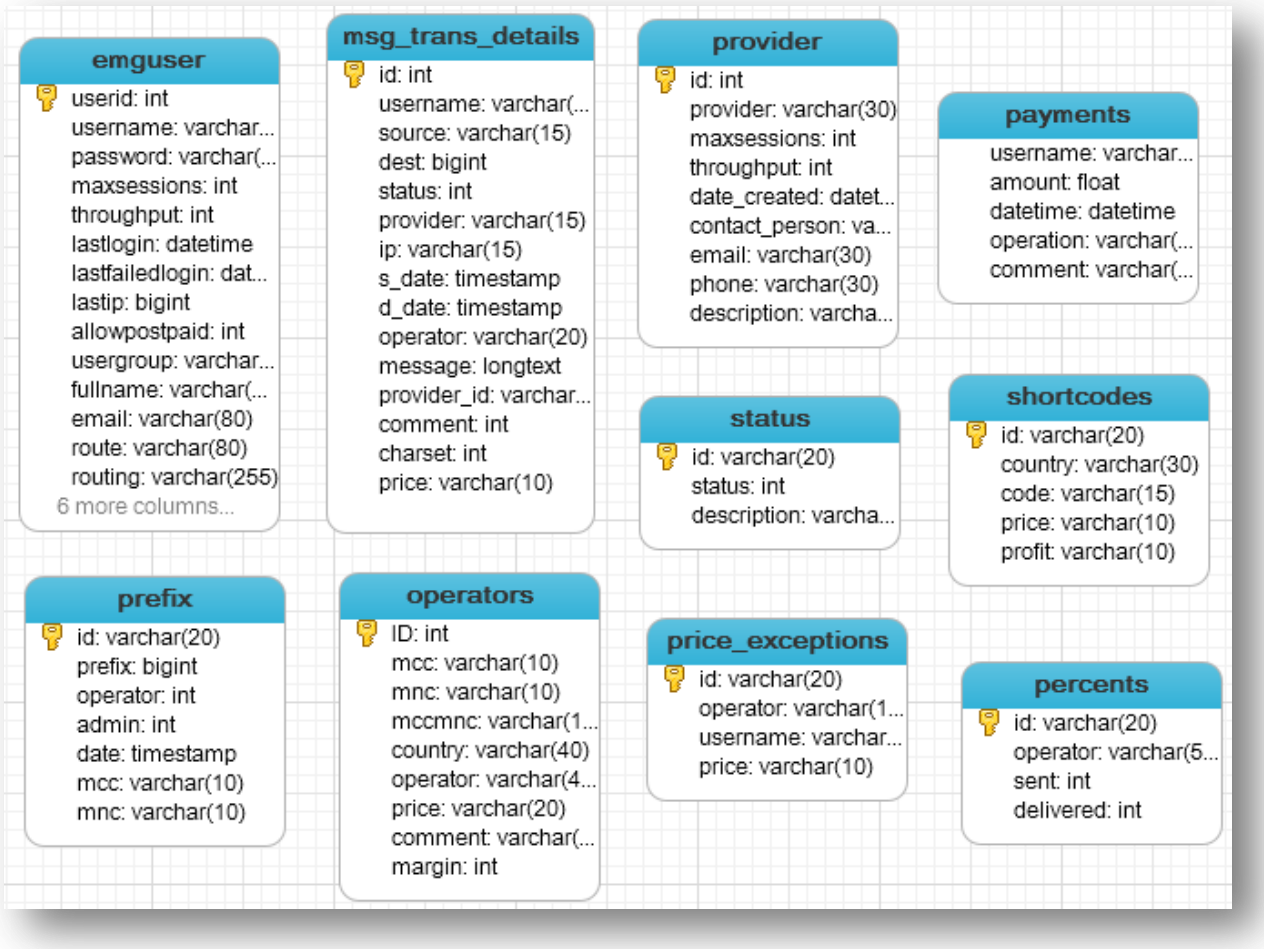
Ukoliko korišćenjem aplikacije, tim koji je zadužen za praćenje i nadgledanje, otkrije izvestan broj nedostavljenih poruka, biće mu potrebne neke dodatne informacije o njima. Aplikacija pruža mogućnost dobijanja i ovakvih informacija. Na stranici “SMS info” dovoljno je uneti redni broj poruke, koji se moze pronaći na bilo kojoj od stranica koje pružaju globalni pregled po kategorijama. Nakon toga, prikazuju se svi traženi podaci koji se mogu dalje koristi u odeđene svrhe.

Pomenućemo još da se u čitavoj aplikaciju koristi AJAX[15] funkcionalnost za ažuriranje pojedinačnih delova stranice upotrebom *JavaScript*-ove biblioteke *JQuery*. Za više detalja o *JavaScript*-u videti [3].

6.2. Model baze podataka

Osnovni element čitave baze podataka, a samim tim i aplikacije, je sms poruka. Korisnik šalje poruku na određeni broj, koji pripada nekom mobilnom operateru, a potom se poruka na destinacioni broj prosleđuje posredsvom provajdera.

U sledećem dijagramu prikazane su tabele iz baze podataka sa odgovarajućim atributima.



[15] AJAX je grupa međusobno povezanih Veb razvojnih tehnika koje se koriste na klijentskoj strani u cilju pravljenja asinhronih aplikacija.

Baza emg_sms se sastoji iz sledećih tabela:

- emguser – ova tabela sadrži podatke o svim korisnicima
- msg_trans_details – sadrži podatke o svim poslatim sms porukama
- operators – ova tabela čuva informacije o svim mobilnim operaterima
- payments – ovde se nalaze informacije o svim promenama stanja kredita korisnika
- percents – u ovoj tabeli se nalaze podaci o dostavljanu poruka izraženi u procentima
- prefix –sadrži neke dodatne inforamije o operaterima
- price_exceptions – ukoliko se cena za ordeđenu destinaciju i određenog korisnika razlikuje od podrazumevanih cena, podaci o tome su smešteni u ovoj tabeli
- provider – sadrži informacije o provajderima
- shortcodes – ukoliko neki korisnik ima pravo na dodatnu uslugu u smislu slanja poruka na kratke brojeve, u ovoj tabeli su smeštene sve informacije vezane za to
- status – u ovoj tabeli se nalazi lista svih statusa koje određena poruka može da ima od trenutka slanja do trenutka dostavljanja poruke na željenu destinaciju

7. ZAKLJUČAK

U ovom radu upoznali smo se sa PDO drajverom i ukazali na neke smernice za njegovo korišćenje prilikom razvoja dinamičnih aplikacija koje se pokreću nad bazama podataka korišćenjem PHP5 programskog jezika. Takođe smo videli kako PDO može biti efikasno korišćen u eliminisanju razlika između različitih API-a tradicionalnih sistema za upravljaje bazama podataka kao i za stvaranje jasnijeg i prenosivijeg kôda.

PDO drajver je korišćen u izradi aplikacije jer je ispunjavao skoro sve kriterijume koje je aplikacija postavljala. Pošto se komunikacija odvija sa bazom podataka čije tabele imaju ogroman broj redova, bio je neophodan alat koji svojom brzinom ne bi ugrozio performanse, a PDO je to ispunio. Takođe, svojim pripremljenim iskazima PDO je omogućio neuporedivo efikasniju komunikaciju između baze podataka i aplikacije. PDO omogućava sprečavanje neželjenih unosa u bazu, što je takođe bio jedan od bitnih uslova. Kako aplikacija ima tendenciju da izlaskom na tržište bude u rukama većeg broja korisnika koji svoje sisteme baziraju na različitim sistemima za upravljanje bazama podataka, PDO je ovde pravi izbor, jer pruža mogućnost jednostavog prelaska sa jednog sistema na drugi.

Dalje unapređenje aplikacije se može vršiti u više pravaca među kojima su:

- Dodavanje logovanja, što bi omogućilo da se korisniku sa pravima administratora dodele veća prava za razliku od ostalih korisnika aplikacije.
- Implementacija inteligentnog sistema koji bi,na osnovu podataka i informacija koje pruža aplikacija, mogao sam da vrši određene izmene u sistemu i na taj način obezbedi njegov efikasniji rad.
- Automatsko generisanje poruka i obaveštenja u slučaju javljanja nekog problema u sistemu za slanje poruka, a koji je aplikacija u stanju da detektuje.

LITERATURA

[1]Dennis Popel, *Learning PHP Data Objects*, First published September 2007 by Packt Publishing Ltd. Birmingham, UK

[2]Timothy Boronczyk, Elizabeth Naramore, Jason Gerner, Yann Le Scouarnec, Jeremy Stolz, *Beginning PHP 6, Apache, MySQL 6 Web Development*, ISBN: 978-0-470-39114-3, Paperback, January 2009

[3]Robin Nixon , *Learning PHP, MySQL, and JavaScript*, Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, United States of America, July 2009: First Edition.

[4]Luke Welling, Laura Thomson, *PHP and MySQL® Web Development, Fourth Edition*, Published by Associate Publisher, Pearson Education, Inc. 800 East 96th Street, Indianapolis, IN 46240 USA, First Printing: September 2008

[5]Gary B. Shelly, Denise M. Woods, *HTML, XHTML, and CSS Complete, Sixth Edition*, Published by Course Technology, 20 Channel Center Street, Boston, MA 02210, USA

[6] *http://www.php.net*:
http://www.php.net/manual/en/book.pdo.php,
http://www.php.net/manual/en/intro.pdo.php,
http://www.php.net/manual/en/pdo.connections.php,
http://www.php.net/manual/en/pdo.transactions.php,
http://www.php.net/manual/en/pdo.prepared-statements.php,
http://www.php.net/manual/en/pdo.error-handling.php,
http://www.php.net/manual/en/class.pdo.php

[7]*http://wiki.hashphp.org/PDO_Tutorial_for_MySQL_Developers*:
http://wiki.hashphp.org/PDO_Tutorial_for_MySQL_Developers#Why_use_PDO.3F
http://wiki.hashphp.org/PDO_Tutorial_for_MySQL_Developers#Connecting_to_MySQL
http://wiki.hashphp.org/PDO_Tutorial_for_MySQL_Developers#Error_Handling
http://wiki.hashphp.org/PDO_Tutorial_for_MySQL_Developers#Running_Simple_Select_Statements
http://wiki.hashphp.org/PDO_Tutorial_for_MySQL_Developers#Fetch_Modes
http://wiki.hashphp.org/PDO_Tutorial_for_MySQL_Developers#Getting_Row_Count
http://wiki.hashphp.org/PDO_Tutorial_for_MySQL_Developers#Getting_the_Last_Insert_Id
http://wiki.hashphp.org/PDO_Tutorial_for_MySQL_Developers#Running_Simple_INSERT.2C_UPDATE.2C_or_DELETE_statements
http://wiki.hashphp.org/PDO_Tutorial_for_MySQL_Developers#Running_Statements_With_Parameters

http://wiki.hashphp.org/PDO_Tutorial_for_MySQL_Developers#Named_Placeholders
http://wiki.hashphp.org/PDO_Tutorial_for_MySQL_Developers#INSERT.2C_DELETE.2C_UPDATE_Prepared_Queries

[8]*http://www.phpeveryday.com/articles/PHP-Data-Object:*
http://www.phpeveryday.com/articles/PDO-Activation-PHP-Data-Objects-Extension-P545.html
http://www.phpeveryday.com/articles/PDO-Connecting-Use-PHP-Data-Object-P546.html
http://www.phpeveryday.com/articles/PDO-Portable-Connection-to-Database-P547.html
http://www.phpeveryday.com/articles/PDO-Posibble-Fetch-Mode-P549.html
http://www.phpeveryday.com/articles/PDO-Error-Handling-P550.html
http://www.phpeveryday.com/articles/PDO-Prepared-Statement-P551.html
http://www.phpeveryday.com/articles/PDO-Positional-and-Named-Placeholders-P552.html
http://www.phpeveryday.com/articles/PDO-Insert-and-Update-Statement-Use-Prepared-Statement-P553.html
http://www.phpeveryday.com/articles/PDO-Prepared-Statement-and-Bound-Values-P554.html
http://www.phpeveryday.com/articles/PDO-Working-With-BLOBs-P555.html
http://www.phpeveryday.com/articles/PDO-Alternative-Retrieve-BLOB-Data-P556.html
http://www.phpeveryday.com/articles/PDO-Setting-Connection-Attributes-P557.html
http://www.phpeveryday.com/articles/PDO-Error-Mode-Attributes-P558.html
http://www.phpeveryday.com/articles/PDO-Improve-Performance-with-Persistent-Connection-P559.html

[9]*http://stackoverflow.com:*
http://stackoverflow.com/questions/16414287/pdo-error-handling
http://stackoverflow.com/questions/1179874/pdo-bindparam-versus-bindvalue
http://stackoverflow.com/questions/60174/how-can-i-prevent-sql-injection
http://stackoverflow.com/questions/13569/mysqli-or-pdo-what-are-the-pros-and-cons
http://stackoverflow.com/questions/17354412/prepared-statements-escape-variables
http://stackoverflow.com/questions/2117019/how-to-correctly-free-a-pdo-instance
http://stackoverflow.com/questions/8263371/how-prepared-statements-can-protect-from-sql-injection-attacks
http://stackoverflow.com/questions/12655325/pdo-get-row-count-before-limit-applied
http://stackoverflow.com/questions/10220333/pdo-transaction-on-one-update
http://stackoverflow.com/questions/7641088/whats-the-limitations-of-pdo-named-arguments
http://stackoverflow.com/questions/17454003/switching-from-mysql-to-pdo
http://stackoverflow.com/questions/20930136/pdo-inside-a-function

http://stackoverflow.com/questions/7542579/pdo-security-and-html-special-chars
http://stackoverflow.com/questions/7542579/pdo-security-and-html-special-chars
http://stackoverflow.com/questions/16345446/dynamic-bindvalue
http://stackoverflow.com/questions/14349257/pdo-connection-error