В. Я. Шабашов

Организация доступа к данным из PHP приложений для различных СУБД

Учебное пособие

DirectMEDIA

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

В. Я. Шабашов

ОРГАНИЗАЦИЯ ДОСТУПА К ДАННЫМ ИЗ PHP ПРИЛОЖЕНИЙ ДЛЯ РАЗЛИЧНЫХ СУБД

Учебное пособие
по дисциплине «Web-программирование»

DirectMEDIA
Москва
Берлин
2019

УДК 004:681.3.06(075)
ББК 32.973.4я7
Ш12

Шабашов, В. Я.

Ш12 Организация доступа к данным из PHP приложений для различных СУБД : учебное пособие по дисциплине «Web-программирование» / В. Я. Шабашов — Москва ; Берлин : Директ-Медиа, 2019. — 120 с.

ISBN 978-5-4475-9888-4

Учебное пособие содержит теоретические и практические сведения для создания информационных систем в виде PHP приложений, использующих СУБД MySQL, Microsoft SQL Server, Firebird и PostgreSQL.

Учебное пособие рассмотрено и одобрено на заседании кафедры информационных систем в экономике ФГБОУ ВО «Алтайский государственный технический университет им. И. И. Ползунова» протокол № 1 от 07.09.2018 г.

Текст приводится в авторской редакции.

УДК 004:681.3.06(075)
ББК 32.973.4я7

ISBN 978-5-4475-9888-4

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время информационные системы, используемые в экономике и бизнесе, все чаще строятся в виде Web приложений. В нашей стране для создания таких систем довольно широко используются технологии основанные на PHP. Одной из причин широкого применения PHP является то, что PHP обеспечивает возможность работы с самыми различными СУБД.

В данной работе рассмотрена организация доступа к данным из приложений PHP на примере четырех СУБД: Microsoft SQL Server Express, MySQL Community, PostgreSQL и Firebird. Предложены функции, упрощающие создание информационных систем. Рассмотрен поэтапный процесс их разработки. В приложениях приведены листинги этих функций.

Эти функции можно использовать непосредственно при разработке информационных систем, а также использовать как учебное пособие для разработки собственных средств для работы с базами данных.

Выбор перечня СУБД, рассматриваемых в данной работе, определяется их широким распространением и возможностью их бесплатного применения. Правда СУБД MySQL и Microsoft SQL Server являются проприетарными, но они имеют версии Community и Express, которые дают возможность их использовать для изучения и некоммерческого применения.

1 УСТАНОВКА PHP РАСШИРЕНИЙ ДЛЯ РАБОТЫ С СУБД

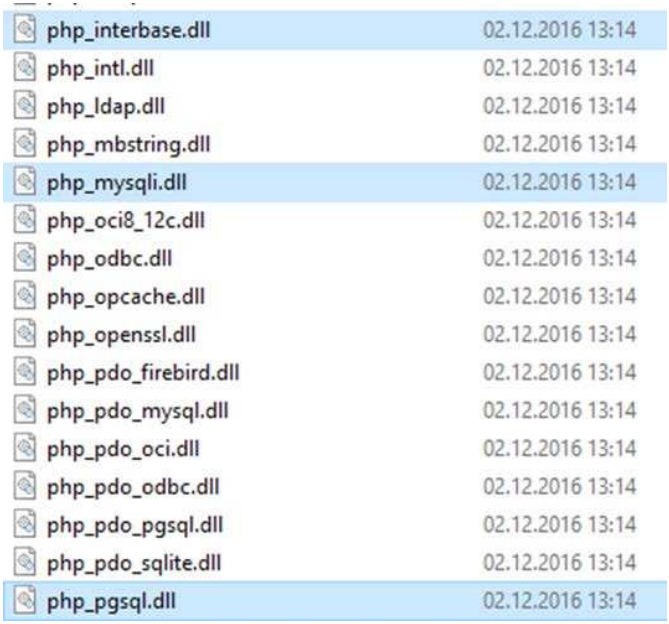
На официальном сайте PHP http://php.net/manual/ru/refs.database.php можно увидеть, что имеются расширения практически для всех наиболее распространенных СУБД (рис. 1.1).

Рис. 1.1 — Расширения для работы с СУБД

Эти расширения разработаны различными производителями и имеют свои особенности для каждой конкретной СУБД. Различия между ними определяются также конструктивными возможностями СУБД и языком инструкций для каждой из них.

Перед их использованием для разработки приложений необходимо настроить PHP таким образом, чтобы можно было выполнять действия с требуемыми СУБД. Рассмотрим, как это выполняется.

Скачав PHP с сайта http://php.net/downloads.php, можно увидеть, что в папке ext имеется расширения для работы со многими СУБД:



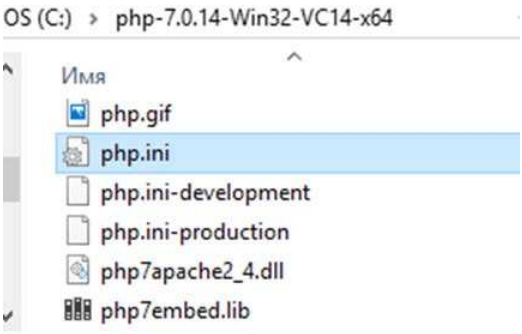
Для работы с интересующими нас СУБД имеются следующие расширения:

- php_interbase.dll для Firebird,
- php_mysqli.dll для MySQL,
- php_pgsql.dll для PostgreSQL.

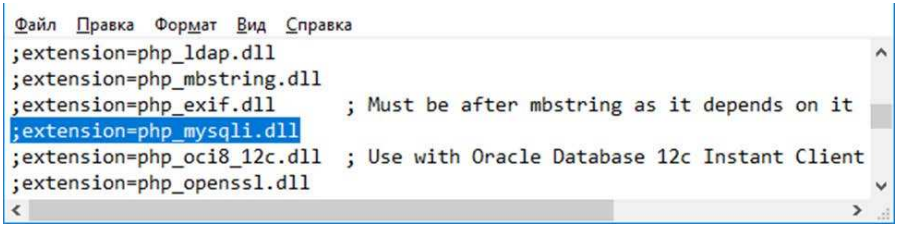
Кроме вышеперечисленных имеются еще и PDO расширения, например, php_pdo_mysql.dll. Драйверы PDO мы рассматривать не будем — они обеспечивают универсальный механизм доступа для всех СУБД, но при этом падает производительность и нет возможности воспользоваться достоинствами каждой конкретной системы управления базами данных.

Для СУБД Microsoft SQL Server расширения здесь отсутствуют, но их можно скачать с сайта фирмы Microsoft. Это будет рассмотрено позднее в этом же разделе.

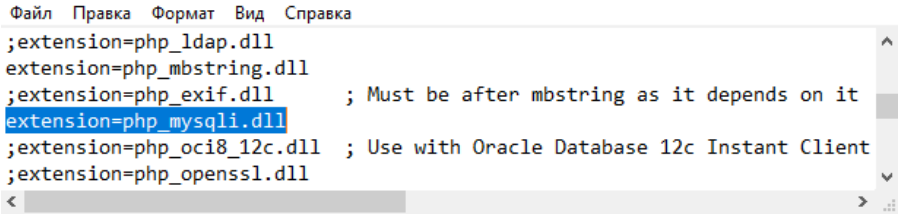
Наличие драйверов в папке ext автоматически не обеспечивает возможность работы с СУБД. Надо также внести соответствующую информацию в файл PHP инициализации php.ini:



Открыв этот файл можно увидеть, что первоначально они не активированы:



Необходимые расширения следует активировать следующим образом:



Следует обратить внимание, что, кроме этого, активировано расширение для работы со строками, состоящими из многобайтовых символов: `extension=php_mbstring.dll`

Аналогичные действия необходимо выполнить для каждой СУБД, с которой будет работать информационная система.

После изменения в массив php.ini требуется перезапустить службу, которая поддерживает WEB сервер (рис. 1.2), например, Apache.

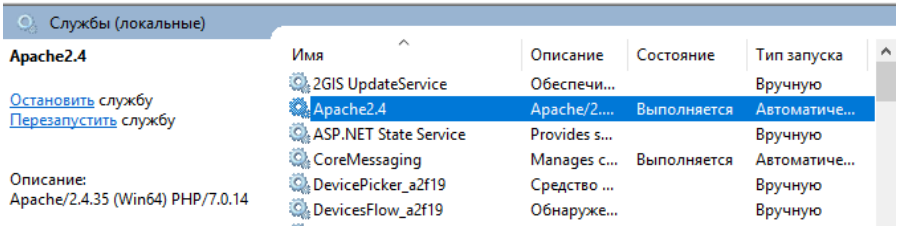


Рис. 1.2 — Перезапуск службы Apache

Для того, чтобы проверить правильность установки, надо запустить в интернет обозревателе приложение info.php, которое автоматически устанавливается при установке PHP (рис. 1.3). Если используется WEB сервер Apache, оно находится в папке `htdocs`.

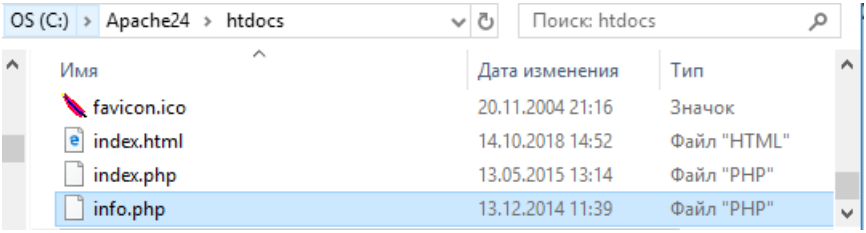


Рис. 1.3 — Приложение info.php

В сведениях о составе, установленной версии PHP должна появиться информация приблизительно такого содержания (рис. 1.4–1.7):

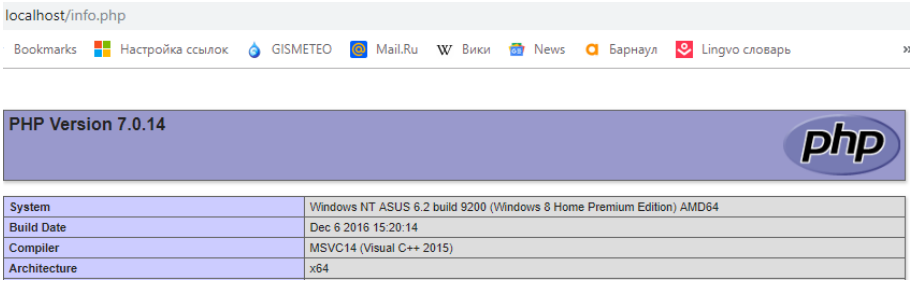


Рис. 1.4 — Заголовок, выдаваемый приложением info.php

mysqli

Mysqli Support	enabled
Client API library version	mysqlnd 5.0.12-dev - 20150407 - $Id: 241ae00989d1995ffcbbf63d579943635faf9972 $
Active Persistent Links	0
Inactive Persistent Links	0
Active Links	0

Directive	Local Value	Master Value
mysqli.allow_local_infile	On	On
mysqli.allow_persistent	On	On
mysqli.default_host	*no value*	*no value*
mysqli.default_port	3306	3306
mysqli.default_pw	*no value*	*no value*
mysqli.default_socket	*no value*	*no value*
mysqli.default_user	*no value*	*no value*
mysqli.max_links	Unlimited	Unlimited
mysqli.max_persistent	Unlimited	Unlimited
mysqli.reconnect	Off	Off
mysqli.rollback_on_cached_plink	Off	Off

Рис. 1.5 — Данные по драйверам для MySQL

interbase

Firebird/InterBase Support	dynamic
Compile-time Client Library Version	Firebird API version 25
Run-time Client Library Version	WI-V6.3.3.32900 Firebird 3.0

Directive	Local Value	Master Value
ibase.allow_persistent	On	On
ibase.dateformat	%Y-%m-%d	%Y-%m-%d
ibase.default_charset	*no value*	*no value*
ibase.default_db	*no value*	*no value*
ibase.default_password	*no value*	*no value*
ibase.default_user	*no value*	*no value*
ibase.max_links	Unlimited	Unlimited
ibase.max_persistent	Unlimited	Unlimited
ibase.timeformat	%H:%M:%S	%H:%M:%S
ibase.timestampformat	%Y-%m-%d %H:%M:%S	%Y-%m-%d %H:%M:%S

Рис. 1.6 — Данные по драйверам для Firebird

pgsql

PostgreSQL Support	enabled
PostgreSQL(libpq) Version	9.6.0
PostgreSQL(libpq)	PostgreSQL 9.6.0 (win32)
Multibyte character support	enabled
SSL support	enabled
Active Persistent Links	0
Active Links	0

Directive	Local Value	Master Value
pgsql.allow_persistent	On	On
pgsql.auto_reset_persistent	Off	Off
pgsql.ignore_notice	Off	Off
pgsql.log_notice	Off	Off
pgsql.max_links	Unlimited	Unlimited
pgsql.max_persistent	Unlimited	Unlimited

Рис. 1.7 — Данные по драйверам для PostgreSQL

Как видим, установка выполнена и можно начинать работу с этими СУБД.

Описание функций для работы с различными СУБД представлено на сайте http://php.net/manual/ru/refs.database.php «Расширения для работы с базами данных».

Для того, чтобы использовать СУБД Microsoft SQL Server, надо получить расширения, созданные разработчиками этой СУБД. Это можно сделать, обратившись к сайту Microsoft (рис. 1.8). Драйверы для СУБД Microsoft SQL Server можно получить по адресу: https://www.microsoft.com/en-us/download/details.aspx?displaylang=en&id=20098

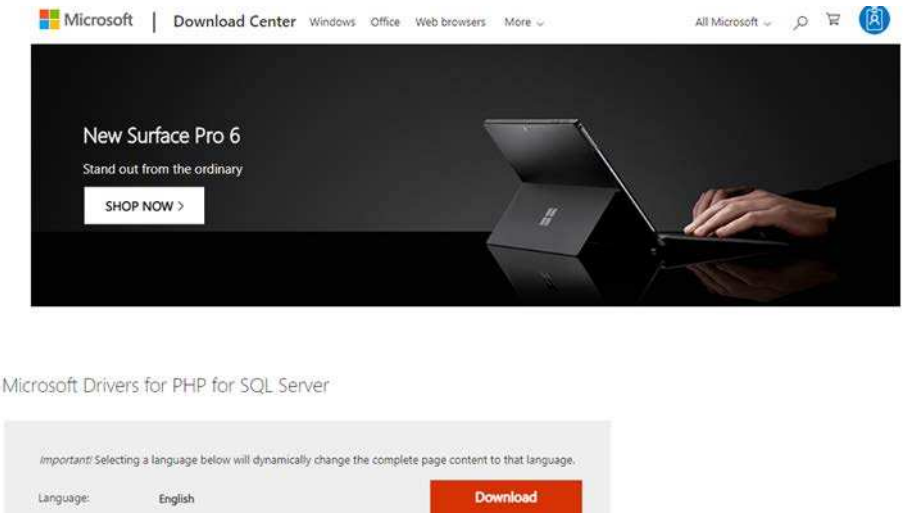


Рис. 1.8 — Начальная страница для загрузки драйверов для MS SQL Server

После нажатия на кнопку Download необходимо выбрать какой модуль вам необходим.

Для выбора необходимого модуля следует в начале посмотреть, для чего каждый из них предназначен. Эту информацию можно получить, прочитав разъяснения, которые представлены на вышеупомянутом сайте (рис. 1.9, 1.10).

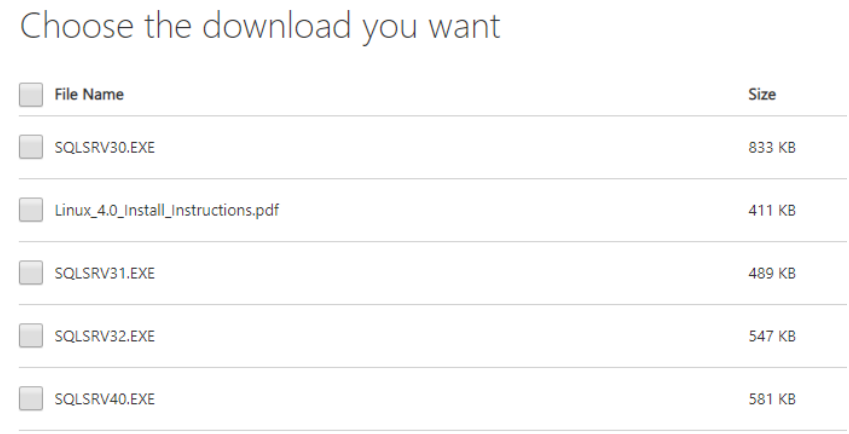
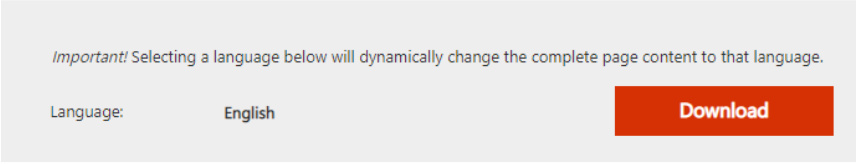


Рис. 1.9 — Перечень модулей для MS SQL Server



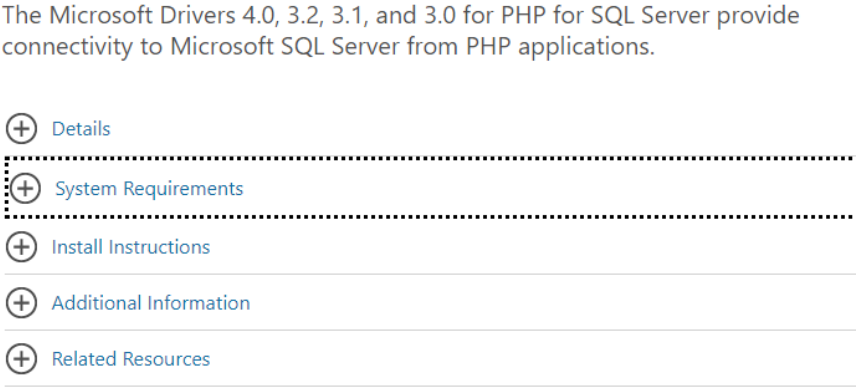


Рис. 1.10 — Дополнительная информация по скачиванию

Раскроем раздел «Системные требования» (System Requirements) (рис. 1.11).

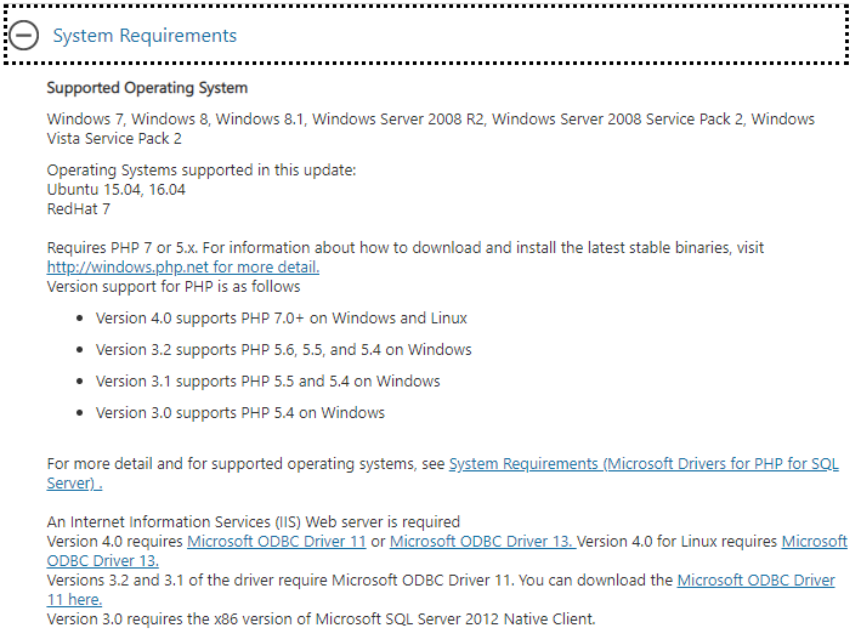


Рис. 1.11 — Системные требования

Из этого документа можно увидеть, что при использовании PHP 7.0 необходим модуль версии 4.0. Значит необходимо скачать модуль **SQLSRV40.EXE**.

Рис. 1.12 — Лицензионное соглашение

После двойного щелчка по скаченному модулю появляется окно для его установки (рис. 1.12).

Надо нажать кнопку Yes и выбрать, куда этот модуль требуется установить (рис. 1.13).

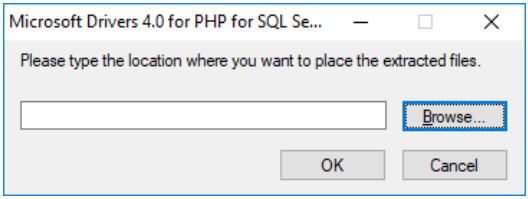


Рис. 1.13 — Выбор места для расположения скачиваемых данных

Расширения для MS Sql Server в конечном итоге необходимо установить в папку ext PHP, но в начале поместим их в какую-нибудь промежуточную папку (рис. 1.14).

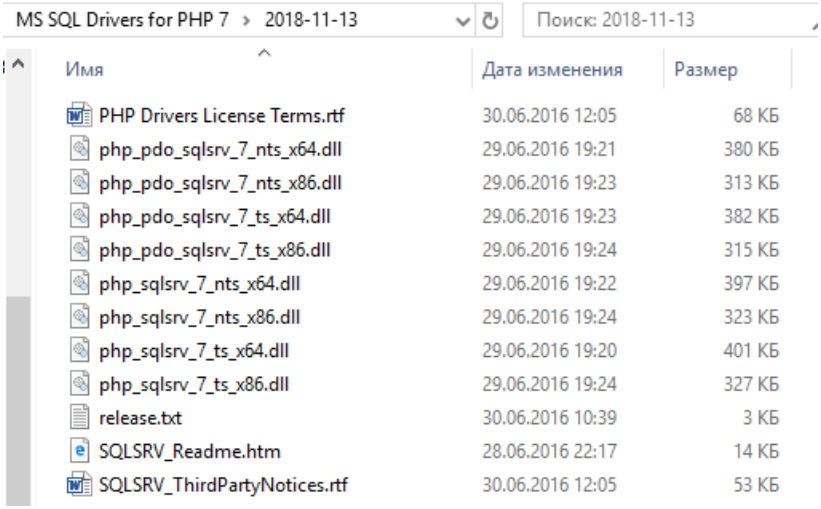


Рис. 1.14 Расширения в промежуточной папке

Как видим, архивный файл содержит большое количество модулей, в том числе имеются файл `SQLSRV_Readme.htm`, в котором содержатся пояснения. Откроем этот файл (рис. 1.15).

В связи с тем, что мы будем делать установку на 64-разрядную систему и PHP у нас работает в потоко-защищенном режиме, переместим модули `php_sqlsrv_7_ts_x64.dll` и `php_pdo_sqlsrv_7_ts_x64.dll` в папку `ext`.

Ранее я рекомендовал не применять драйверы PDO, но бывают ситуации, когда все-таки их требуется использовать, поэтому мы установили не только основной модуль, но и модуль для режима PDO.

Driver file	PHP version	Thread safe?	Use with PHP .dll
php_sqlsrv_7_nts_x86.dll php_pdo_sqlsrv_7_nts_x86.dll	7.0	no	php7.dll
php_sqlsrv_7_ts_x86.dll php_pdo_sqlsrv_7_ts_x86.dll	7.0	yes	php7ts.dll
php_sqlsrv_7_nts_x64.dll php_pdo_sqlsrv_7_nts_x64.dll	7.0	no	php7.dll
php_sqlsrv_7_ts_x64.dll php_pdo_sqlsrv_7_ts_x64.dll	7.0	yes	php7ts.dll

Рис. 1.15 — Сведения о модулях в файле SQLSRV_Readme.htm

Кроме этого, надо открыть файл php.ini и добавить в него две строчки:

```
extension=php_sqlsrv_7_ts_x64.dll
extension=php_pdo_sqlsrv_7_ts_x64.dll
```

Теперь следует перезапустить службу Apache (рис. 1.16) для того, чтобы установленные расширения начали работать.

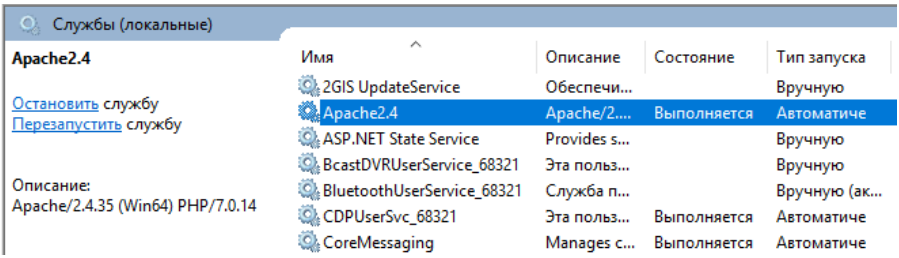


Рис. 1.16 — Перезапуск службы

Для того, чтобы проверить, что установка выполнена, запустим в интернет обозревателе приложение info.php, которое находится в папке htdocs приложения Apache (рис. 1.17).

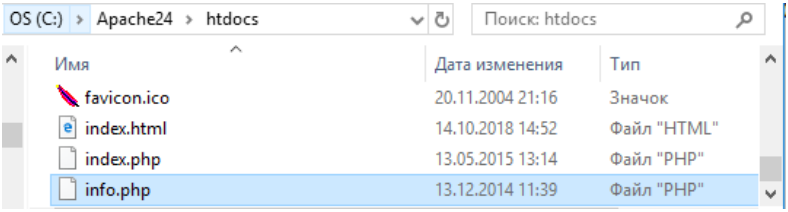


Рис. 1.17 — Запуск приложения info.php

В сведениях о составе, установленной версии PHP должна появиться информация приблизительно такого содержания (рис. 1.18).

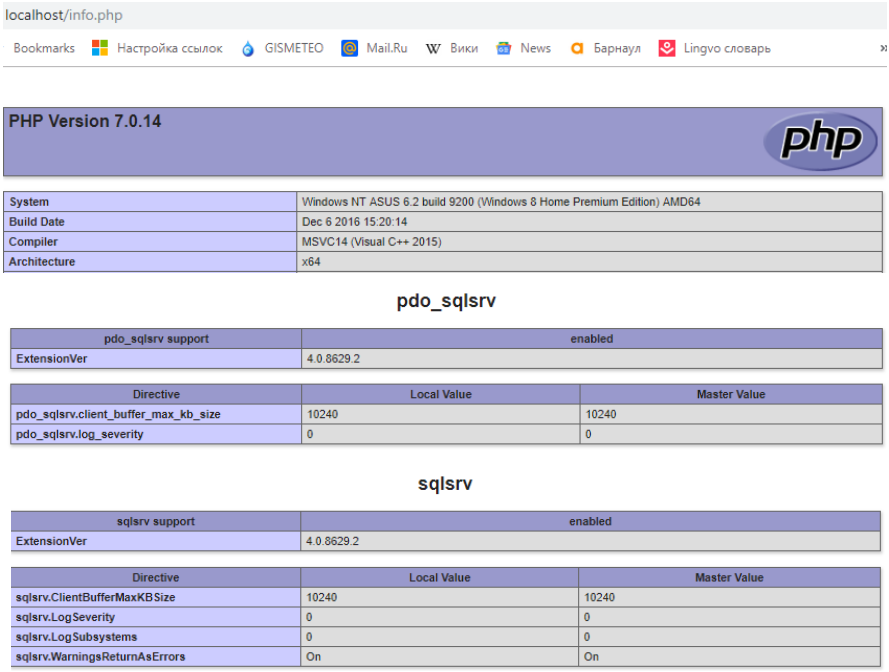


Рис. 1.18 Сведения об установленных драйверах

Как видим, установка выполнена и можно начинать работу с Microsoft SQL сервером.

Описание функций, обеспечивающих доступ к СУБД Microsoft SQL Server дано на сайте http://php.net/manual/ru/book.sqlsrv.php. Более детальное описание представлено на следующем сайте.

https://docs.microsoft.com/en-us/sql/connect/php/sqlsrv-driver-api-reference?view=sql-server-2017

2 КОНЦЕПЦИЯ УНИФИЦИРОВАННЫХ ФУНКЦИЙ ДЛЯ ДОСТУПА К ДАННЫМ

Доступ к данным в информационных системах обычно предназначен для выполнения следующих видов работ:

получение сведений из базы данных;

внесение изменений в базу данных.

Иногда требуется выполнять и другие функции, например, создавать таблицы и другие объекты, получать сведения о структуре таблиц (имена, полей типы данных и т. д.). Мы будем рассматривать только два первых вида работ, остальные действия в информационных системах выполняются достаточно редко.

Наша цель — создать унифицированные функции доступа к данным на языке PHP для различных СУБД, причем входные и выходные данные должны быть унифицированными, одинаковыми для всех СУБД. На запросы не должны накладываться ограничения, ведущие к сокращению возможностей, каждой из систем.

При работе с различными СУБД можно увидеть, что функции PHP для доступа к данным могут возвращать информацию в различном виде. Например, расширения для MySQL возвращают информацию о дате в виде текстового поля, а расширения для PostgreSQL в виде объекта типа DateTime. Поэтому определим некоторый единый тип возвращаемых данных.

Функции должны работать таким образом, чтобы можно было с небольшими затратами обеспечить миграцию от одной СУБД к другой. Для этого потребуется только заменить обращение к одной функции на другую и при необходимости в некоторых ситуациях исправить запрос. Без исправления запросов в общем случае не обойтись, так как каждая СУБД использует свой диалект языка SQL.

Например, в качестве знака конкатенации символьных строк в большинстве СУБД используется символ " + ", а в Firebird символ " || ".

Разрабатываемые функции при получении наборов данных должны обеспечивать возможность разделения информации на страницы. При обращении к функции должен задаваться SQL запрос на получение всего набора данных в целом и указываться номер страницы, а функция должна выдавать данные только по одной

странице. Такая возможность существенно упростит разработку приложения, обеспечивающего разбиение данных на страницы.

Для выполнения вышеуказанных действий для каждой СУБД создадим по две функции на языке PHP, которые назовем:

MyExecQuery и MyExecNonQuery для MySQL;

MsExecQuery и MsExecNonQuery для Microsoft SQL;

FbExecQuery и FbExecNonQuery для Firebird;

PgExecQuery и PgExecNonQuery для Postgre SQL;

Каждая из этих функций имеет два параметр — первый из них представляет ассоциативный массив, который содержит сведения, необходимые для открытия соединения с базой данных, второй параметр также является ассоциативным массивом, и содержит SQL запрос и другие сведения, необходимые для выполнения соответствующих действий в базе данных. Результатом выполнения функции будет JSON массив, который содержит сведения, полученные в результате выполнения SQL запроса, а в случае возникновения ошибок — сведения об ошибках.

Первый параметр обоих функций будет содержать следующие элементы:

`host` — имя сервера;

`port` — номер порта, является необязательным;

`dbName` — имя базы данных;

`userName` — имя пользователя;

`password` — пароль.

Например, этот параметр может выглядеть следующим образом:

```
[
    'host' => 'localhost',
    'port' => '5432',
    'dbName' => 'okved_db',
    'userName' => 'php',
    'password' => 'phpPassword',
]
```

Для открытия соединений с базой данных могут использоваться и некоторые другие элементы, например `CharacterSet`, который задает кодировку на стороне пользователя. Мы его не включили во входной параметр так как результатом работы будет JSON массив, а для его формирования необходимо чтобы данные

были представлены в кодировке UTF-8. Поэтому при открытии соединения функция будет всегда задавать кодировку UTF-8. В результате этого независимо от кодировки, определенной на стороне сервера, результат всегда будет представлен в UTF-8.

Набор параметров, используемых при соединении с базой данных, не исчерпывается теми, которые здесь перечислены. Например, при работе с СУБД Microsoft SQL Server может использоваться до тридцати различных параметров. В большинстве же случаев достаточно только вышеперечисленных.

Для функций ExecQuery второй параметр будет содержать следующие элементы:

`sqlQuery` — строка запроса;

`pageSize` — размер страницы (является необязательным);

`pageNumber` — номер страницы (является необязательным).

Сведения по страницам задаются в связи с тем, что в информационных системах выходные данные часто требуется разбивать на страницы. Каждый запрос при этом будет считывать данные только для одной страницы. Если задан `pageSize`, то должен быть задан и `pageNumber`. Нумерация страниц ведется с единицы.

Этот параметр может быть выглядеть следующим образом:

```
[
    'sqlQuery' => 'SELECT kodokved, nameokved FROM okved',
    'pageSize' => 15,
    'pageNumber' => 5
]
```

Для функций ExecNonQuery второй параметр будет содержать следующие элементы:

`sqlQuery` — строка запроса;

`newId` — признак, указывающий, что при операции добавления новой записи надо возвратить идентификатор этой записи (является необязательным, определяется только для первичных ключей представляющих из себя столбец с автоувеличением).

Этот параметр может выглядеть следующим образом:

```
[
    'sqlQuery' => "INSERT INTO okved (nameokved) " .
                  "VALUES ('Банковская деятельность')",
    'newId' => TRUE
]
```

Результатом выполнения функции типа ExecQuery является массив, назовем его условно `result`. Выходная информация должна быть передана в представлении JSON. Преобразование массива в JSON выполняется с помощью PHP функции json_encode. Эта функция работает с данными, представленными в коде UTF-8. Результирующий массив `result` содержит два элемента с индексами 0 и 1, которые также являются массивами. Назовем их условно `res` и `data`. Массив `res`, в свою очередь, при нормальном завершении работы, состоит из двух массивов `count` и `names`. Массив `count` содержит 4 элемента с индексами от 0 до 3:

0 — признак, определяющий завершение работы ("OK" — нормальное завершение, "Error" — наличие ошибок),

1 — общее количество записей с данными, которые могут быть получены с помощью заданного SQL запроса (если поддерживается разбиение на страницы, то дается общее количество строк, а не размер страницы),

2 — количество столбцов в одной записи,

3 — номер текущей страницы.

Массив `names` содержит перечень полей массива `data`.

В случае ненормального завершения работы первый элемент массива `res` содержит строковое значение "Error".

Кроме этого, массив `res` может содержать еще несколько элементов, содержащих описание ошибки. Их количество зависит от характера ошибки. Описание ошибки представляет собой группу строк, заданных в массиве `res` начиная с элемента с индексом 1.

При аварийном завершении работы массив `data` отсутствует.

Например, выходной JSON массив при ненормальном завершении работы может иметь следующий вид:

[["Error", "Ошибка при обращении к БД", "Access denied"]].

В случае нормального завершения работы массив `data` содержит информацию, которую требуется отобразить в виде таблицы. Каждая запись массива `data` является также массивом.

При нормальном завершении результирующий массив может иметь следующий вид:

[[["OK", 3, 2,1], ["id", "name"]], [[1, "ИСЭ"], [2, "ИТ"], [3, "ПМ"]]]

В приведенном примере массив `data` содержит три записи, каждая из них содержит по два поля.

Результатом выполнения запроса типа ExecNonQuery будет массив, который в случае нормального выполнения запроса, содержит три элемента: первый содержит значение "OK", второй — количество записей, которые были затронуты запросом, а третий — идентификатор вновь введенной строки. Третий параметр определяется только в том случае, если была выполнена инструкция `INSERT` и был задан входной параметр `newId` со значением "true".

Например, он может иметь следующий вид:

["OK", 1, 235].

Если при выполнении функции возникла ошибка, то первый элемент результирующего массива содержит значение "Error", второй и последующие содержат описание ошибки. Например, результирующий массив может иметь следующий вид:

["Error","Запрос не выполнен:", "Unknown column 'id1' in 'where clause'", "update ul_okved set dtend = '2017-03-01' where id1 = 3700;"]

Вышеизложенная концепция предполагает, что результат выполнения функции представлен в виде JSON массива. Это связано с тем, что в настоящее время широко используется технология AJAX, которая предполагает обращение к модулям PHP из приложения, выполняемого в интернет обозревателе и написанного на JavaScript. Приложение JavaScript хорошо интегрируется с JSON массивами, ими удобно пользоваться.

В том случае, если приложение не использует технологию AJAX и приложение пишется исключительно на языке PHP, получение данных в виде JSON не требуется. В этом случае в предлагаемых функциях достаточно изменить только последний оператор и информация будет возвращаться в виде PHP массивов. Требуется оператор `return json_encode($res);` заменить на `return $res;`.

3 ДОСТУП К СУБД MYSQL

3.1 Функции PHP для работы с СУБД MySQL

В данном разделе будут рассмотрены следующие функции:

1. mysqli_connect
2. mysqli_connect_errno
3. mysqli_connect_error
4. mysqli_close
5. mysqli_set_charset
6. mysqli_query
7. mysqli_multi_query
8. mysqli_fetch_array
9. mysqli_free_result
10. mysqli_num_fields
11. mysqli_num_rows
12. mysqli_fetch_field
13. mysqli_affected_rows
14. mysqli_insert_id

Полный перечень функций, используемых для работы СУБД MySQL можно найти на сайте http://php.net/manual/ru/book.mysqli.php.

3.1.1 Функция, обеспечивающая соединение с базой данных, имеет следующую спецификацию:

```
mysqli_connect ([ string $host =
ini_get("mysqli.default_host")

[, string $username= ini_get("mysqli.default_user")

[, string $passwd = ini_get("mysqli.default_pw")

[, string $dbname = ""

[, int $port = ini_get("mysqli.default_port")

[, string $socket =
ini_get("mysqli.default_socket") ]]]]]] )
```

Параметр `$host` определяет сервер, на котором расположена база данных. Если сервер не задан, то по умолчанию берется сервер, заданный директивой *mysqli.default_host* в файле php.ini. Заданное значение может быть именем хоста или IP-адресом.

Передача NULL или строки "localhost" этому параметру означает, что в качестве хоста будет использоваться локальная машина, на которой запущен скрипт.

Параметр `$username` определяет имя пользователя, а `$password` его пароль. Если они не указаны, то используется пользователь и пароль, определенный по умолчанию. В этом случае их значения задаются директивой *mysqli.default_user* в файле php.ini. Обычно такая возможность не используется так как с приложениями php работает много пользователей с различными правами доступа.

Если после установления соединения будет выполнен второй вызов функции с теми же аргументами, то новое соединение не будет установлено. Вместо этого функция вернёт ссылку на уже установленное соединение.

Если параметр `$username` задан, его значение будет использоваться в качестве имени базы данных по умолчанию при выполнении запросов.

Имя базы данных задается параметром `$dbname`. Если параметр задан, его значение будет использоваться в качестве имени базы данных по умолчанию при выполнении запросов.

Параметр `$port` задает номер порта для подключения к серверу MySQL.

Параметр `$socket` задает сокет или именованный пайп, который необходимо использовать.

3.1.2 Для того, чтобы определить, были ли ошибки при выполнении функции `mysql_connect()` можно использовать функцию `mysqli_connect_errno`, которая имеет следующую спецификацию:

```
int mysqli_connect_errno (void)
```

Она возвращает код ошибки последнего вызова `mysql_connect()`. В случае отсутствия ошибок возвращается 0.

3.1.3 Для того, чтобы получить текстовое описание ошибки используется функция `mysqli_connect_error`, которая имеет следующую спецификацию:

```
string mysqli_connect_error (void)
```

Если ошибка отсутствует она возвращает `NULL`.

3.1.4 Для закрытия соединения используется функция `mysqli_close`, которая имеет следующую спецификацию:

```
bool mysqli_close (mysqli $link),
```

где `$link` — идентификатор соединения, полученный с помощью `mysqli_connect()`.

Открытые непостоянные соединения MySQL и результирующие наборы автоматически удаляются сразу по окончании работы PHP скрипта. Следовательно, закрывать соединения и очищать результирующие наборы не обязательно, но рекомендуется, так как это сразу же освободит ресурсы базы данных и память, занимаемую результатами выборки, что может положительно сказаться на производительности.

3.1.5 Ввиду того, что функция `mysqli_connect()` не позволяет задать кодировку, которая будет использоваться при получении данных из базы, используется функция `mysqli_set_charset` , которая имеет спецификацию:

```
bool mysqli_set_charset (mysqli $link, string $charset)
```

Параметр `$link` определяет идентификатор соединения, полученный с помощью `mysqli_connect()`.

Параметр `charset` задает идентификатор кодировки, которую требуется установить. Например, при использовании кодировки UTF-8 надо задать значение '**`utf8`**'. Вставлять тире (**`utf-8`**) нельзя.

Функция возвращает `TRUE` в случае успешного завершения или `FALSE` при возникновении ошибки.

3.1.6 Функция, обеспечивающая выполнение запроса к базе данных, имеет следующую спецификацию:

```
mixed mysqli_query (mysqli $link, string $query [,
int $resultmode = MYSQLI_STORE_RESULT ]) ,
```

где `$link` — идентификатор соединения, полученный с помощью функции `mysqli_connect()`.

`$query` — текст запроса.

`$resultmode` — принимает значение `MYSQLI_USE_RESULT`, либо `MYSQLI_STORE_RESULT` в зависимости от требуемого поведения функции. По умолчанию используется `MYSQLI_STORE_RESULT`.

При использовании MYSQLI_USE_RESULT все последующие вызовы этой функции будут возвращать ошибку до тех пор, пока не будет вызвана функция mysqli_free_result).

Функция mysqli_query возвращает FALSE в случае неудачи. В случае нормального выполнения запросов типа SELECT, она вернет объект mysqli_result. Для остальных успешных запросов (например, INSERT, UPDATE, DELETE) эта функция возвращает TRUE.

3.1.7 Кроме mysqli_query имеется еще одна функция для выполнения запроса к базе данных, она имеет следующую спецификацию

```
bool mysqli_multi_query (mysqli $link , string $query )
```

Она запускает на выполнение один или несколько запросов, перечисленных через точку с запятой. Первый параметр определяет соединение, которое должно быть установлено до выполнения функции. Второй параметр определяет строку, содержащую запрос или группу запросов.

Функция mysqli_multi_query возвращает FALSE, если первое выражение содержит ошибку. Чтобы получить доступ к ошибкам остальных подзапросов, нужно сначала вызвать функцию mysqli_next_result().

3.1.8 Для возвращения одной строки из результирующего набора, полученного в результате выполнения запроса используется функция mysqli_fetch_array, которая имеет следующую спецификацию:

```
mixed mysqli_fetch_array ( mysqli_result $result
[, int $resulttype = MYSQLI_BOTH ] ),
```

где $result — идентификатор, получаемый при выполнении запроса к базе данных с помощью mysqli_query().

$resulttype — необязательный параметр, принимающий значение константы. Он указывает на тип массива, который будет получен в результате выполнения функции mysqli_fetch_array. Возможные значения параметра: MYSQLI_ASSOC — ассоциативный массив, MYSQLI_NUM — числовой массив или MYSQLI_BOTH — оба типа массива.

Функция $mysqli_fetch_array() используется для последовательного извлечения строк из результирующего набора, полученного при выполнении запроса к базе данных. Обычно она используется в цикле. После того как все строки будут извлечены, она вернет не массив, а значение NULL.

3.1.9 Для освобождения памяти от результирующего набора, полученного при обращении к базе данных, используется функция, которая имеет следующую спецификацию:

```
void mysqli_stmt_free_result (mysqli_result $result)
```

где $result — идентификатор, получаемый при выполнении запроса к базе данных с помощью mysqli_query().

3.1.10 Для получения количества полей, результирующего набора данных используется функция

```
int mysqli_num_fields (mysqli_result $result),
```

где $result — идентификатор, получаемый при выполнении запроса к базе данных с помощью mysqli_query().

3.1.11 Для получения количества строк результирующего набора данных используется функция

```
int mysqli_num_rows ( mysqli_result $result),
```

где $result — идентификатор, получаемый при выполнении запроса к базе данных с помощью mysqli_query().

3.1.12 Для получения метаинфомации, содержащей различные сведения о структуре данных результирующего набора, используется функция, спецификация которой имеет следующий вид:

```
array mysqli_fetch_fields (mysqli_result $result )
```

где $result — идентификатор, получаемый при выполнении запроса к базе данных с помощью mysqli_query().

Функция mysqli_fetch_fields возвращает массив объектов, содержащих метаданные полей или FALSE, если нет доступных столбцов.

В следующей таблицы описаны свойства объекта.

Свойство	Описание
name	Имя столбца
orgname	Исходное имя столбца, если у него есть псевдоним
table	Имя таблицы, которой принадлежит столбец (если столбец не является вычисляемым)
orgtable	Исходное имя таблицы, если есть псевдоним
max_length	Максимальная ширина поля результирующего набора (в символах).
length	Длина поля в байтах, как она задана при определении таблицы. Обратите внимание, что данная величина (в байтах) может отличаться от величины в символах, указанной в определении поля таблицы, так как в разных кодировках один символ может записываться несколькими байтами. Например, поле VARCHAR(10) в кодировке UTF-8 вернет длину 20 = 10 символов * 2 байта на символ, а для кодировки LATIN1 - длину 10, так как в этой кодировке один символ занимает один байт.
charsetnr	Числовой идентификатор кодировки.
flags	Целое число, представляющее битовые флаги для поля.
type	Тип данных поля
decimals	Число знаков после запятой (для целочисленных полей)

Как видим, можно получить самую разнообразную информацию. Большинство представленных свойств являются интуитивно понятными, рассмотрим только свойство flags, которое может принимать следующие значения:

```
NOT_NULL_FLAG = 1
PRI_KEY_FLAG = 2
UNIQUE_KEY_FLAG = 4
BLOB_FLAG = 16
UNSIGNED_FLAG = 32
ZEROFILL_FLAG = 64
BINARY_FLAG = 128
ENUM_FLAG = 256
AUTO_INCREMENT_FLAG = 512
TIMESTAMP_FLAG = 1024
```

```
SET_FLAG = 2048
NUM_FLAG = 32768
PART_KEY_FLAG = 16384
GROUP_FLAG = 32768
UNIQUE_FLAG = 65536
```

Параметр `flags` дает самую разнообразную информацию о характере столбца. Например, если нас интересует, является ли столбец первичным ключом и столбцом с автоувеличением, это можно сделать следующим образом:

```php
$meta = $mysqli_result_object->fetch_field();
if ($meta->flags & 4) {
    echo 'Первичный ключ';
}
if ($meta->flags & 512) {
    echo 'Столбец с автоувеличением';
}
```

3.1.13 При выполнении запросов типа `INSERT`, `UPDATE` или `DELETE` часто требуется выяснить — сколько строк таблицы были охвачены инструкцией такого типа (сколько добавлено, сколько изменено или сколько удалено).

Это можно выполнить с помощью функции, имеющей следующую спецификацию:

```
int mysqli_affected_rows (mysqli $link) ,
```

где `$link` — идентификатор соединения, полученный с помощью функции `mysqli_connect()`.

Функция возвращает целое число. Если оно большее нуля, то это количество затронутых или полученных строк. Ноль означает, что запросом вида `UPDATE` не обновлено ни одной записи, или что ни одна строка не соответствует условию `WHERE` в запросе. Значение -1 указывает на то, что запрос вернул ошибку.

3.1.14 При вводе новой строки при помощи инструкции `INSERT` в таблицу, в которой первичный ключ является столбцом с автоувеличением, часто необходимо знать значение первичного ключа для вновь введенной строки. Эту информацию можно получить с помощью функции, имеющей следующую спецификацию:

```
mixed mysqli_insert_id ( mysqli $link ),
```

где `$link` — идентификатор соединения, полученный с помощью функции `mysqli_connect()`.

Функция `mysqli_insert_id()` возвращает идентификатор, генерируемый запросом (обычно `INSERT`) к таблице, которая содержит колонку с атрибутом `AUTO_INCREMENT`. Если последний запрос не был `INSERT` или в модифицируемой таблице отсутствует колонка с атрибутом `AUTO_INCREMENT`, данная функция вернет ноль.

3.2 Создание вспомогательных функций для MySQL

Функции MyExecNonQuery и MyExecQuery используют следующие вспомогательные функции:

- `parenthesisLevel`,
- `inQuotation`,
- `fromPosition`.

3.2.1 Первая функция определяет уровень вложенности в круглые скобки. Она используется для того, чтобы определить для любой заданной позиции SQL запроса, входит ли она в состав основного запроса или в состав вложенного запроса.

```
function parenthesisLevel($s, $fromPos)
{
    $level = 0;
    for ($i = 6; $i < $fromPos; $i++)
    {
        if (substr($s, $i, 1) == '(') {
                $level++;
        }
        if (substr($s, $i, 1) == ')') {
                $level--;
        }
    }
    return $level;
}
```

Здесь `$s` — строка SQL запроса, а `$fromPos` — номер позиции в запросе (целое число).

3.2.2 Вторая функция определяет для заданной позиции SQL запроса, входит ли она в состав текстового литерала, то есть содержится ли она внутри одинарных кавычек. Эта функция используется для поиска служебных слов в запросе. Например, если найдено слово `FROM` надо определить не является ли это слово частью текстового литерала.

```
function inQuotation($s, $fromPos)
{
    $inQuot = 0;
    for ($i = 6; $i < $fromPos; $i++)
    {
        if (substr($s, $i, 1) == "'") {
                $inQuot = $inQuot ^ 1;
        }
    }
    return $inQuot;
}
```

Здесь `$s` — строка SQL запроса, а `$fromPos` — номер позиции в запросе (целое число).

3.2.3 Третья функция служит для поиска слово `FROM` в основном запросе. Если в запросе в предложении `SELECT` имеются вложенные запросы, то слово `FROM` вложенного запроса будет отброшено и поиск продолжится до получения адреса слово `FROM` основного запроса.

```
function fromPosition($s, $from)
{
    $ar = str_word_count($s, 2, '_');
    foreach ($ar as $key => $value)
    {
        if (strtolower($value) != $from) {
            continue;
        }
        if (parenthesisLevel($s, $key) == 0 && inQuotation($s, $key) == 0){
            return $key;
        }
    }
    return FALSE;
}
```

Здесь `$s` — строка SQL запроса, а `$from` — строка фрагмента запроса, обычно это текстовый литерал `'from'`.

Искомый фрагмент кода надо задавать в нижнем регистре. В строке запроса искомый контекст может находиться как в верхнем, так и в нижнем регистре.

3.3 Создание функции MyExecNonQuery

3.3.1 Функция MyExecNonQuery вначале формирует переменные, необходимые для установления соединения с базой данных.

```
function MyExecNonQuery($connectionInfo, $Query)
{
    $host = $connectionInfo["host"];
    $port = $connectionInfo["port"];
    $dbName = $connectionInfo["dbName"];
    $userName = $connectionInfo["userName"];
    $password = $connectionInfo["password"];
    $q = trim($Query["sqlQuery"]);
    if (substr($q, strlen($q) - 1) != ';') {
        $q .= ';';
    }
    $res = array();
```

Ввиду того, что параметры этой функции не имеют полного соответствия с параметрами функции `mysqli_connect` делаются некоторые преобразования.

Заданная с помощью параметров инструкция SQL очищается от пробелов в начале и в конце строки, а также при отсутствии в конце точки с запятой дополняется этим символом. Необходимо отметить, что допускается в качестве параметра задавать строку, которая содержит несколько инструкций, разделенных между собой символом ";".

Объединять вместе несколько инструкций не всегда является целесообразным. Например, если объединить несколько инструкций `INSERT` и задать параметр `newId = TRUE`, то будет возвращено значение идентификатора введенной записи только для последней выполненной инструкции.

Определяемый массив `$res` будет содержать результаты выполнения инструкции.

3.3.2 Далее создается соединение и определяется количество инструкций в строке SQL запроса.

```
$link = mysqli_connect($host, $userName, $password, $dbName, $port);
if (mysqli_connect_errno($link)) {
    $res[0] = "Error";
    $res[1] = "Host: " . $host;
    $res[2] = "Login: " . $userName;
    $res[3] = "Password: " . $password;
    $res[4] = "DbName: " . $dbName;
    $res[5] = "Соединение не установлено: " . mysqli_connect_error();
    goto end;
}
mysqli_set_charset($link, 'utf8');  // utf8 обязательна для json_encode
$qCount = count(explode(';', $q)) - 1; // количество запросов в одной строке
```

Здесь также задается кодировка, которая будет использоваться на стороне клиента с помощью функции `mysqli_set_charset()`. В результате этого, независимо от того какая кодировка используется на сервере, выходные данные будут представлены в UTF-8.

В случае возникновения ошибки при создании соединения в результирующий массив будут записаны параметры, которые использовались при обращении к функции, а также информация о характере ошибок.

3.3.3 Выполнение запроса реализуется с помощью функции `mysqli_multi_query`. Она запускает на выполнение один или несколько запросов, перечисленных через точку с запятой.

Обращение к этой функции выполняется следующим образом:

```
$result = mysqli_multi_query($link, $q);
if (!$result) {
    $res[0] = "Error";
    $res[1] = "Запрос не выполнен :<br>" . mysqli_error($link);
    $res[2] = $q;
    mysqli_free_result($result);
    goto end;
}
```

В случае возникновения ошибки в результирующий массив записываются сведения об ошибке, а также выполняемый SQL запрос.

3.3.4 Если запрос выполнен без ошибок формируется результирующий массив и преобразовывается в JSON массив.

```
$res[0] = "OK";
$res[1] = mysqli_affected_rows($link);
if ($qCount > $res[1]) {
    $res[1] = $qCount;  // В строке несколько запросов
}
$res[2] = mysqli_insert_id($link); //Если нет поля identity, то будет
end:
    if ($link) {
        mysqli_close($link);
    }
return json_encode($res);
```

Извлечение идентификатора вновь введенной записи выполняется с помощью функции `mysqli_insert_id()`, при этом проверка, нужно ли это значение, не выполняется. В случае, если оно не имеет смысла для выполняемой инструкции в массив `$res` будет записываться ноль.

3.3.5 Проверка работы функции ***MyExecNonQuery***:

Добавление новой записи.

```
$conInfo =
        [
            'host' => 'localhost',
            'port' => 3306,
            'dbName' => 'okved_db',
            'userName' => 'php',
            'password' => '*********'
        ];
$Query =
        [
            'sqlQuery' => "INSERT INTO ul_okved (idul, kodokved, main, " .
                          "dtstart) VALUES ('6', '501', 0, '2018-10-10');",
            'newId' => TRUE,
        ];

$res = MyExecNonQuery($conInfo, $Query);
echo $res;
```

Результат: ["OK",1,331] — инструкция выполнена, одна запись добавлена и ее идентификатор равен 331.

Корректировка данных.

```
$Query =
[
    'sqlQuery' => "update ul_okved set dtend = '2017-03-01' where id = 327;" .
        "update ul_okved set dtend = '2017-12-01' where id = 328;",
];
$res = MyExecNonQuery($conInfo, $Query);
echo $res;
```

Результат: ["OK",2,0] — выполнено две инструкции, обновлены две записи.

Удаление данных:

```
$Query =
[
    'sqlQuery' => "DELETE FROM ul_okved WHERE id >= 331;",
];
$res = MyExecNonQuery($conInfo, $Query);
echo $res;
```

Результат: ["OK",1,0] — выполнено удаление одной записи.

3.4 Создание функции MyExecQuery

3.4.1 Функция MyExecQuery вначале получает переменные, необходимые для установления соединения с базой данных.

```
function MyExecQuery($connectionInfo, $Query)
{
    $host = $connectionInfo["host"];
    $port = $connectionInfo["port"];
    $dbName = $connectionInfo["dbName"];
    $userName = $connectionInfo["userName"];
    $password = $connectionInfo["password"];

    $query = $Query["sqlQuery"];
    $pSize = $Query["pageSize"];
    $pNum = $Query["pageNumber"];
```

Из второго параметра функции извлекаются сведения о запросе, и записываются в соответствующие переменные.

3.4.2 Далее объявляются массивы, в которых будет формироваться результаты работы функции и устанавливается соединение с базой данных.

```
$res = array();
$data = array();
$link = mysqli_connect($host, $userName, $password, $dbName, $port);
if (mysqli_connect_errno($link))
{
    $res[0] = "Error";
    $res[1] = "Host: " . $host;
    $res[2] = "Login: " . $userName;
    $res[3] = "Password: " . $password;
    $res[4] = "DbName: " . $dbName;
    $res[5] = "Port: " . $port;
    $res[6] = "Соединение не установлено: " . mysqli_connect_error();
    goto end;
}
mysqli_set_charset($link, 'utf8');  // UTF8 обязательна для json_encode
```

Ввиду того, что функция `mysqli_connect()` не позволяет при установлении соединения задавать кодировку на стороне клиента, это выполняется с помощью функции `mysqli_set_charset()`.

3.4.3 Теперь начинается работа со страницами. Она выполняется только в том случае, если задана длина страницы и ее значение превышает 0.

Сначала необходимо определить общее количество строк, которые мог бы возвратить запрос при отсутствии разбиения на страницы.

```
if ($pSize > 0)  // работа со страницами
{
    $sql = $query;
    $start = fromPosition($sql, 'select') + 6;
    $end = fromPosition($sql, 'from');
    $s = substr($sql, 0, $start) . ' count(*) ' . substr($sql, $end);
```

Функция `fromPosition` используется для того, чтобы определить позицию в запросе, с которой начинается слово `select` и слово `from`. Функция работает правильно независимо от регистра, используемого при написании ключевых слов. Если в предложении `select` используется текстовый литерал, содержащий ключевое

слово, или вложенный запрос эта функция отработает правильно, и она найдет адрес ключевого слова `from` правильно.

После определения позиций ключевых слов содержимое предложения `select` удаляется и вместо этого записывается предложение, которое определяет количество строк.

3.4.4 Далее запускается запрос, который определяет количество строк `$rowCount`.

```
$result = mysqli_query($link, $s);
if (!$result) {
    $res[0] = "Error";
    $res[1] = "Запрос не выполнен :<br>" . mysqli_error($link);
    $res[2] = $sql;
    $res[3] = $s;
    goto end;
}
$line = mysqli_fetch_array($result, MYSQLI_NUM);
$rowCount = $line[0];
mysqli_free_result($result);
```

Для извлечения необходимых данных из результатов запроса используется функция `mysqli_fetch_array`.

Полученная величина количества строк заносится в переменную `$rowCount` и результат запроса очищается.

3.4.5 Теперь необходимо определить переменную `$pMin`, которая должна содержать номер строки, с которой надо извлекать информацию для заданной страницы

Ввиду того, что при работе с данными при получении следующей страницы, другие пользователи могут удалить или добавить записи, а также при обращении к функции номер страницы номер страници может быть задан ошибочно, приложением которое использует эта функция, выполняется некоторая обработка, которая ведет к тому, чтобы не возникали различные ошибки при извлечении данных. Так, например, если указан номер страницы, который находится за пределами диапазона, определяемого количество строк, номер извлекаемой страницы может быть откорректирован и будет извлечена последняя страница.

```
        $min = $pSize * ($pNum - 1);
        if ($rowCount > 0) {
            while ($min >= $rowCount) {
                $min -= $pSize;
                $pNum--;
            }
            if ($min <= 0) {
                $min = 0;
                $pNum = 1;
            }
        }
        else {
            $min = 0;
            $pNum = 1;
        }
```

3.4.6 После этого в исходный запрос необходимо включить предложение `limit`, которое определяет начальную позицию и количество считываемых записей.

```
    $ind = strrpos($query, ';');
    if ($ind !== FALSE) {
        $query = substr ($query, 0, $ind);
    }
    $query = $query . ' limit ' . $min . ', ' . $pSize . ';';
}   // конец работы со страницами
```

3.4.7 Извлечение данных из базы выполняется с помощью функции `mysqli_query()`.

```
$result = mysqli_query($link, $query);
if (!$result) {
    $res[0] = "Error";
    $res[1] = "Запрос не выполнен :<br>" . mysqli_error($link)
    $res[2] = $query;
    goto end;
}
$fieldCount = mysqli_num_fields($result);
if ($pSize == 0) {
    $rowCount = mysqli_num_rows($result);
}
```

Здесь же определяется количество столбцов результирующего набора данных, а также количество строк результирующего набора

(при отсутствии режима разбиения на страницы). Для этого используется функции `mysqli_num_fields()` и `mysqli_num_rows()`.

3.4.8 Результат работы функции MyExecQuery должен содержать наименования столбцов полученного набора данных. Для этого используется функция `mysqli_fetch_field`. Результатом работы этой функции будет ассоциативный массив, содержащий различную метаинформацию: имена полей, названия таблиц для каждого поля (результирующий набор может содержать данные из нескольких таблиц), типы данных, длину текстовых полей, признак первичного ключа, признак столбца с автоувеличением, признак допустимости значения `null` в столбце. Следует отметить, что набор функций для MySql является единственным (из рассматриваемых СУБД), который имеет возможность извлекать такой объем мета данных.

Извлеченные данные о наименовании столбцов будут записаны в массив `$names`.

```
$names = array();
for ($i = 0; $i < $fieldCount; $i++) {
    $info = mysqli_fetch_field($result);
    $names[$i] = $info->orgname;
    if ($names[$i] == '') {
        $names[$i] = $info->name;
    }
}
```

Здесь `orgname` — исходное имя столбца, если у него есть псевдоним, а `name` — имя столбца или псевдоним. То есть, если был запрос `'select idul as id from Ul'` будет извлечено имя `idul`. Если был запрос `'select idul from Ul'` также будет извлечено имя `idul`. Если в предложении `select` используется звездочка, все названия полей будут благополучно извлечены. Если имя поля в запросе задается вместе с именем таблицы, будет извлечено только имя поля без названия таблицы.

3.4.9 Теперь будет сформирован массив `$res`, который содержит информацию о результатах работы функции: признак нормального завершения, количество строк, количество столбцов и номер страницы, названия столбцов, а также массив `$data`,

содержащий результирующий набор данных в виде массива с числовыми индексами.

```
$res[0] = array("OK", $rowCount, $fieldCount, $pNum);
$res[1] = $names;
$i = 0;
while ($line = mysqli_fetch_array($result, MYSQLI_NUM)) {
    $data[$i++] = $line;
}
```

Если при обращении к функции MyExecQuery не будет указан размер страницы и ее номер, в массиве `$res` номер страницы будет иметь значение `null`.

3.4.10 После этого освобождается ресурс, закрывается соединение, и функция возвращает результирующий JSON массив.

```
        mysqli_free_result($result);
        end:
        if ($link) {
            mysqli_close($link);
        }
        return  json_encode([$res, $data]);
    }
```

3.4.11 Пример использования функции:

```
$conInfo =
        [
            'host' => 'localhost',
            'port' => 3306,
            'dbName' => 'okved_db',
            'userName' => 'php',
            'password' => '******'
        ];
$Query =
        [
            'sqlQuery' => 'select * from ul_okved',
            'pageSize' => 8,
            'pageNumber' => 1
        ];
$res = MyExecQuery($conInfo, $Query);
echo $res;
```

Результат работы:

[[["OK","61",6,1],["id","idul","kodokved","main","dtstart","dtend"]],
[["37","2","001","1","2004-12-18","2017-03-01"],["38","2","301","0",
"2009-12-12","2010-10-26"],["39","3","001","1","2010-10-25","2011-
10-25"],["40","3","101","1","2010-10-25",null],["41","3","301","0",
"2009-11-25","2010-10-16"],["42","1","001","0","2007-11-15","2016-
03-19"],["43","1","102","1","2010-11-16",null],["44","2","102","0",
"2007-11-16",null]]]

4 ДОСТУП К СУБД MICROSOFT SQL SERVER

4.1 Функции PHP для доступа к Microsoft SQL

В данном разделе будут рассмотрены следующие функции:

1. sqlsrv_connect
2. sqlsrv_errors
3. sqlsrv_close
4. sqlsrv_query
5. sqlsrv_free_stmt
6. sqlsrv_fetch_array
7. sqlsrv_num_fields
8. sqlsrv_num_rows
9. sqlsrv_field_metadata
10. sqlsrv_rows_affected

Полный перечень функций, используемых для работы с СУБД Microsoft SQL Server, дан на сайте http://php.net/manual/ru/book.sqlsrv.php, а также на сайте https://docs.microsoft.com/ru-ru/sql/connect/php/microsoft-php-driver-for-sql-server?view=sql-server-2017 "Драйверы Microsoft SQL Server для PHP".

Второй сайт содержит более подробную информацию.

4.1.1 Функция PHP, обеспечивающая соединение с базой данных, имеет следующую спецификацию:

```
resource sqlsrv_connect (string $serverName [, array $connectionInfo])
```

Первый параметр `$serverName` определяет имя сервера и имя экземпляра, они разделены обратной косой чертой. Обычно этот параметр имеет следующий вид: `'asus\sqlexpress'`. Если используется экземпляр по умолчанию, то задается только имя сервера, например, `'asus'`.

Второй параметр `$connectionInfo` определяет информацию, необходимую для открытия соединения. Он является ассоциативным массивом, который задает свойства соединения. Количество элементов этого массива достигает тридцати различных параметров, рассмотрим основные из них:

`Database` — имя базы данных;
`UID` — имя пользователя;
`PWD` — пароль;
`CharacterSet` — кодировка на стороне пользователя.
Этот массив может выглядеть следующим образом:

```
$connectionInfo = array('Database'=>'dbName',
'UID'=>'userName', 'PWD'=>'password');
```

СУБД Microsoft SQL Server допускает два вида аутентификации: серверную и аутентификацию Windows (когда для доступа к СУБД используется учетная запись Windows). При Windows аутентификации имя и пароль не задаются.

Полное описание всех параметров соединения представлено на сайте:
https://docs.microsoft.com/en-us/sql/connect/php/connection-options?view= sql-server-2017

4.1.2 Для определения нормального завершения процесса создания соединения и получения сведений об ошибках используется функция `sqlsrv_errors`, спецификация которой имеет следующий вид:

```
mixed sqlsrv_errors ([ int $errorsOrWarnings ] ) ,
```

где `$errorsOrWarnings` — предопределенная константа, которая может принимать одно из значений, содержащихся в следующей таблице.

Значение	Описание
SQLSRV_ERR_ALL	Возвращаются ошибки и предупреждения, созданные при последнем вызове функции `sqlsrv`.
SQLSRV_ERR_ERRORS	Возвращаются ошибки, созданные при последнем вызове функции `sqlsrv`.
SQLSRV_ERR_WARNINGS	Возвращаются предупреждения, созданные при последнем вызове функции `sqlsrv`.

Если значение параметра не указано, возвращаются ошибки и предупреждения, созданные при последнем вызове функции sqlsrv.

Функция sqlsrv_errors возвращает информацию об ошибках и предупреждениях, возникающих при выполнении различных операций (не только операций создания соединений). При возникновении ошибок или предупреждений в последней выполненной операции функция возвращает массив, содержащих информацию об ошибках и предупреждениях. Если последняя выполненная операция завершилась нормально, функция возвращает NULL.

Следующая таблица описывает структуру возвращаемого ассоциативного массива.

Ключ	Описание
SQLSTATE	Ошибки, порождаемые драйвером ODBC, возвращают SQLSTATE созданный драйвером ODBC. Ошибки порождаемые драйвером PHP возвращают SQLSTATE IMSSP. Предупреждения, порождаемые драйвером PHP возвращают SQLSTATE 01SSP.
code	Ошибки, порождаемые SQL сервером, возвращают код ошибки SQL сервера. Ошибки, порождаемые ODBC драйверами — коды ошибок ODBC. Ошибки порождаемые Microsoft драйверами PHP возвращают свои собственный коды ошибок.
message	Текстовое описание ошибки.

Как видим, информация имеет сложную структуру. Обычно не имеет смысла извлекать все данные. Если необходимо получить только сообщения об ошибках, то это можно сделать следующим образом: `sqlsrv_errors(SQLSRV_ERR_ERRORS)['message'].`

Если извлечь информацию об ошибках оператором `sqlsrv_errors()[SQLSRV_ERR_ERRORS]['message'],` результат будет таким же, как и в предыдущем случае. Здесь вернется массив, содержащий три массива. Из массива верхнего уровня выберется только элемент с индексом SQLSRV_ERR_ERRORS.

4.1.3 Закрытие соединения выполняется с помощью функции `sqlsrv_close`, которая имеет следующую спецификацию:

```
bool sqlsrv_close (resource $conn)
```

При закрытии соединения освобождается ресурс, после закрытия его использовать нельзя.

4.1.4 Выполнение запроса к базе данных производится с помощью функции `sqlsrv_query`, спецификация которой имеет следующий вид:

```
mixed sqlsrv_query (resource $conn, string $sql
[, array $params [, array $options]])
```

Параметры этой функции имеют следующий вид:

`$conn` — ресурс, определяющий соединение, которое должно быть создана заранее;

`$sql` — строка, содержащая одну или несколько инструкций SQL;

`$params` — массив, определяющий параметры запроса.

Параметры запроса дают возможность обеспечить более безопасное выполнение инструкции и упрощают процесс формирования запроса. Необходимо отметить, что возможность работы с параметрами обеспечивает только СУБД Microsoft SQL Server. Остальные рассматриваемые СУБД такой возможности не имеют.

Описание этих параметров, а также опций запроса `options` даны по адресу:

http://php.net/manual/ru/function.sqlsrv-query.php

4.1.5 Освобождение ресурса, созданного функцией `sqlsrv_query` выполняется функцией `sqlsrv_free_stmt`, спецификация которой имеет следующий вид:

```
bool sqlsrv_free_stmt (resource $stmt) ,
```

где `$stmt` — оператор освобождаемого ресурса.

После выполнения этой функции освобожденный ресурс использовать нельзя. Если `$stmt` равен *`null`*, ошибка не возникает.

4.1.6 Получение очередной строки результирующего набора в виде массива после выполнения запроса производится с помощью

функции sqlsrv_fetch_array, которая имеет следующую спецификацию:

```
array sqlsrv_fetch_array (resource $stmt [, int $fetchType [, int $row [, int $offset]]]) ,
```

где $stmt — ресурс, создаваемый функцией sqlsrv_query,

$fetchType — предопределенная константа, определяющая тип возвращаемого массива, возможные значения — это константы: SQLSRV_FETCH_ASSOC (массив с ассоциативными индексами), SQLSRV_FETCH_NUMERIC (массив с числовыми индексами), SQLSRV_FETCH_BOTH (оба типа массива — используется по умолчанию).

В данной работе используется SQLSRV_FETCH_NUMERIC так как он обеспечивает минимизацию объема информации и максимальную скорость доступа.

Остальные параметры в работе не используются, их описание можно найти на сайте: http://php.net/manual/ru/function.sqlsrv-fetch-array.php.

4.1.7 Для получения количества полей активного результирующего набора данных используется функция

```
int sqlsrv_num_fields (sqlsrv_result $result) ,
```

где $result — идентификатор, получаемый при выполнении запроса к базе данных с помощью sqlsrv_query().

4.1.8 Для получения количества строк активного результирующего набора данных используется функция

```
int sqlsrv_num_rows ( sqlsrv result $result) ,
```

где $result — идентификатор, получаемый при выполнении запроса к базе данных с помощью sqlsrv_query().

4.1.9 Для извлечения метаданных используется инструкция sqlsrv_field_metadata , спецификация которой имеет следующий вид:

```
mixed sqlsrv_field_metadata(resource $stmt),
```

где $stmt — ресурс, создаваемый функцией sqlsrv_query.

Функция sqlsrv_field_metadata возвращает массивов или значение FALSE. Массив содержит один подмассив для каждого поля в результирующем наборе. Каждый подмассив имеет ключи, которые описаны в следующей таблице. Если при извлечении метаданных полей возникает ошибка, возвращается значение FALSE.

Key	Описание
Имя	Имя столбца, которому соответствует поле.
Тип	Числовое значение, соответствующее типу SQL.
Размер	Число символов для полей символьного типа (char(n), varchar(n), nchar(n), nvarchar(n), XML). Число байтов для полей двоичного типа (binary(n), varbinary(n), UDT). Значение NULL для других типов данных SQL Server.
Точность	Точность для типов переменной точности (real, numeric, decimal, datetime2, datetimeoffset и time). Значение NULL — для других типов данных SQL Server.
Масштаб	Масштаб для типов переменного масштаба (numeric, decimal, datetime2, datetimeoffset и time). Значение NULL — для других типов данных SQL Server.
Допускает значения NULL	Значение перечисления, указывающее, что столбец допускает значение NULL (**SQLSRV_NULLABLE_YES**), столбец не допускает значение NULL (**SQLSRV_NULLABLE_NO**), либо неизвестно, допускает ли столбец значение NULL (**SQLSRV_NULLABLE_UNKNOWN**).

Следующая таблица содержит дополнительные сведения о ключах для каждого подмассива (дополнительные сведения об этих типах можно получить в документации по Microsoft SQL Server).

Тип данных SQL Server	Тип	Размер
bigint	SQL_BIGINT (-5)	8
binary	SQL_BINARY (-2)	$0 < n < 8000$ [1]
bit	SQL_BIT (-7)	
char	SQL_CHAR (1)	$0 < n < 8000$ [1]

Окончание таблицы

Тип данных SQL Server	Тип	Размер
data	SQL_TYPE_DATE (91)	
datetime	SQL_TYPE_TIMESTAMP (93)	
datetime2	SQL_TYPE_TIMESTAMP (93)	
datetimeoffset	SQL_SS_TIMESTAMPOFFSET (-155)	
decimal	SQL_DECIMAL (3)	
float	SQL_FLOAT (6)	
image	SQL_LONGVARBINARY (-4)	2 GB
int	SQL_INTEGER (4)	
money	SQL_DECIMAL (3)	
nchar	SQL_WCHAR (-8)	$0 < n < 4000$ [1]
ntext	SQL_WLONGVARCHAR (-10)	1 ГБ
numeric	SQL_NUMERIC (2)	
nvarchar	SQL_WVARCHAR (-9)	$0 < n < 4000$ [1]
real	SQL_REAL (7)	
smalldatetime	SQL_TYPE_TIMESTAMP (93)	
smallint	SQL_SMALLINT (5)	2 байта
smallmoney	SQL_DECIMAL (3)	
text	SQL_LONGVARCHAR (-1)	2 GB
time	SQL_SS_TIME2 (-154)	
timestamp	SQL_BINARY (-2)	8 байт
tinyint	SQL_TINYINT (-6)	1 байт
uniqueidentifier	SQL_GUID (-11)	16
varbinary	SQL_VARBINARY (-3)	$0 < n < 8000$ [1]
varchar	SQL_VARCHAR (12)	$0 < n < 8000$ [1]
xml	SQL_SS_XML (-152)	0

4.1.10 При выполнении запросов типа `INSERT`, `UPDATE` или `DELETE` часто требуется выяснить — сколько строк таблицы были охвачены инструкцией такого типа (сколько добавлено, сколько изменено или сколько удалено).

Это можно выполнить с помощью функции, имеющей следующую спецификацию:

```
int sqlsrv_rows_affected (resource $stmt)
```

Функция возвращает целое число, указывающее количество строк, измененных при выполнении последней инструкции. Если никакие строки не были изменены, возвращается нуль. Если данные о количестве измененных строк недоступны, возвращается минус единица. Если при получении количества измененных строк произошла ошибка, возвращается значение `FALSE`.

4.2 Создание вспомогательных функций для Microsoft SQL

Функции MsExecNonQuery и MsExecQuery используют следующие вспомогательные функции:

- `typeOfColumn`
- `parenthesisLevel`,
- `inQuotation`,
- `fromPosition`.

4.2.1 Функция `typeOfColumn` определяет тип колонки результирующего набора, создаваемого при выполнении запроса. Функция PHP для получения метаинформации выдает данные в виде целой константы, кроме этого, типы Microsoft SQL сервера более разнообразны, чем это требуется для наших функций, например, там имеется несколько целых типов (bit, int, bigint и т.д), несколько строковых типов (char, nchar, varchar, nvarchar и т.д.).

```
function typeOfColumn($it)
{
    if ($it === -5 || $it === -7 || $it === 4 || $it === 5 || $it === -6) {
        return 'int';
    }
    if ($it === 91) {
        return 'date';
    }
    if ($it === 1 || $it === 12 || $it === -9 || $it === -1 || $it === -10) {
        return 'string';
    }
    if ($it === 3 || $it === 6 || $it === 2 || $it === 7) {
        return 'number';
    }
    return 'other';
}
```

Параметр `$it` — это целое, определяющие тип колонки.

4.2.2 Следующая функция определяет уровень вложенности в круглые скобки. Она используется для того, чтобы определить для любой заданной позиции SQL запроса, входит ли она в состав основного запроса или в состав вложенного запроса.

```
function parenthesisLevel($s, $fromPos)
{
    $level = 0;
    for ($i = 6; $i < $fromPos; $i++)
    {
        if (substr($s, $i, 1) == '(') {
                $level++;
        }
        if (substr($s, $i, 1) == ')') {
                $level--;
        }
    }
    return $level;
}
```

Здесь `$s` — строка SQL запроса, а `$fromPos` — номер позиции в запросе (целое число).

4.2.3 Остальные вспомогательные функции описаны в 3.2.

4.3 Создание функции MsExecNonQuery

4.3.1 Функция MsExecNonQuery вначале получает параметры, необходимые для установления соединения с базой данных.

```
function MsExecNonQuery($connectionInfo, $Query)
{
    $host = $connectionInfo["host"];
    if (isset($connectionInfo["port"]) && $connectionInfo["port"] != "") {
        $host .= ", " . $connectionInfo["port"];
    }
    $dbName = $connectionInfo["dbName"];
    $userName = $connectionInfo["userName"];
    $password = $connectionInfo["password"];

    $q = trim($Query["sqlQuery"]);
    if (substr($q, strlen($q) - 1) != ';') {
            $q .= ';';
    }
```

Так как параметры этой функции не имеют полного соответствия с параметрами функции `sqlsrv_connect` приходится выполнять некоторые преобразования.

Заданная с помощью параметров инструкция SQL очищается от пробелов в начале и в конце строки, а также при отсутствии в конце точки с запятой дополняется этим символом. Допускается в качестве параметра задавать строку, которая содержит несколько инструкций, разделенных между собой символом ";".

Объединять вместе несколько инструкций не всегда является целесообразным. Например, если объединить несколько инструкций `INSERT` и задать параметр `newId = TRUE`, то будет возвращено значение идентификатора введенной записи только для последней выполненной инструкции.

4.3.2 В отличие от функций для СУБД MySQL здесь отсутствует специальная функция по извлечению идентификатора для вновь созданной записи (это необходимо, если первичным ключом является поле с автоувеличением). В связи с этим, запрос дополняется еще одним запросом, использующим системную функцию `@Identity` для получения нового значения ключа. Подготовительные действия по получению вновь созданного первичного ключа выглядят следующим образом:

```
if (isset($Query["newId"])) {
    $newId = $Query["newId"];
}
else {
    $newId = FALSE;
}
if ($newId) {
    $q .= 'SELECT @@Identity;';
}
$res = array();
```

Для получения нового первичного ключа используется стандартная функция `$$Identity`, которая будет выполнена вместе с SQL инструкцией по вводу новой строки.

Определяемый здесь массив `$res` будет содержать результаты выполнения инструкции.

4.3.3 Далее создается соединение.

```
$link = sqlsrv_connect($host, ['UID'=>$userName,
                               'PWD'=>$password,
                               'Database' => $dbName,
                               'CharacterSet' => 'UTF-8']);
if (!$link) {                          // UTF-8 обязательна для json_encode
    $res[0] = "Error";
    $res[1] = "Host: " . $host;
    $res[2] = "Login: " . $userName ;
    $res[3] = "Password: " . $password;
    $res[4] = "DbName: ". $dbName;
    $res[5] = 'Соединение не установлено: \n' .
                    sqlsrv_errors()[SQLSRV_ERR_ERRORS]['message'];
    goto end;
}
```

Здесь также задается кодировка, которая будет использоваться на стороне клиента. В результате этого, независимо от того какая кодировка используется на сервере, выходные данные будут представлены в UTF-8.

В случае возникновения ошибки при создании соединения, в результирующий массив будут записаны параметры, которые использовались при обращении к функции, а также информация о характере ошибок.

4.3.4 Обращение к функции `sqlsrv_query` выполним следующим образом:

```
$result = sqlsrv_query($link, $q);
if (!$result) {
    $res[0] = "Error";
    $res[1] = "Запрос не выполнен :<br>" .
                sqlsrv_errors()[SQLSRV_ERR_ERRORS]['message'];
    $res[2] = $q;
    goto end;
}
```

В случае возникновения ошибки в результирующий массив записываются сведения об ошибке, а также текст выполняемого SQL запрос.

4.3.5 Если запрос выполнен без ошибок, формируется результирующий массив и преобразовывается в JSON массив. Результат, полученный функцией, очищается с помощью `sqlsrv_free_stmt`.

```
    $res[0] = "OK";
    $res[1] = sqlsrv_rows_affected($result);
    if ($newId && (strtolower(substr($q, 0, 6)) == 'insert')) {
        sqlsrv_next_result($result);
        $res[2] = (int)sqlsrv_fetch_array($result, SQLSRV_FETCH_NUMERIC)[0];
    }
     else {
        while(sqlsrv_next_result($result)) {
          $res[1] += sqlsrv_rows_affected($result);
        }
    }
    sqlsrv_free_stmt($result);
    end:
    if ($link) {
        sqlsrv_close($link);
    }
    return json_encode($res);
}
```

Извлечение идентификатора вновь введенной записи выполняется с помощью функции `sqlsrv_next_result`. При этом, если не задан параметр `newId` и выполняется операция, отличная от insert, функция `sqlsrv_next_result` не выполняется.

4.3.6 Проверка работы функции ***MsExecNonQuery***:

Добавление новой записи.

```
$conInfo =
        [
            'host' => 'localhost',
            'dbName' => 'okved_db',
            'userName' => 'php',
            'password' => '********'
        ];
$Query =
[
    'sqlQuery' => "INSERT INTO ul_okved (idul, kodokved, main, "
                . "dtstart) VALUES ('6', '501', 0, '2018-10-10');",
    'newId' => TRUE
];
```

Результат: ["OK",1,3030] — инструкция выполнена, одна запись добавлена и ее идентификатор равен 3030.

Корректировка данных.

```
$Query =
[
  'sqlQuery' => "update ul_okved set dtend = '2017-03-01' where id = 3004;"
              . "update ul_okved set dtend = '2017-03-01' where id = 3030;",
];
$res = MsExecNonQuery($conInfo, $Query);
echo $res;
```

Результат: ["OK",2] — выполнено две инструкции, обновлены две записи.

Удаление данных:

```
$Query =
[
    'sqlQuery' => "DELETE FROM ul_okved WHERE id >= 3028",
];
$res = MsExecNonQuery($conInfo, $Query);
echo $res;
```

Результат: ["OK",1] — выполнено удаление одной записи.

4.4 Создание функции MsExecQuery

4.4.1 Функция MsExecQuery вначале получает параметры, необходимые для установления соединения с базой данных.

```
function MsExecQuery($connectionInfo, $Query)
{
    $host = $connectionInfo["host"];
    if (isset($connectionInfo["port"]) && $connectionInfo["port"] != "") {
        $host .= ", " . $connectionInfo["port"];
    }
    $dbName = $connectionInfo["dbName"];
    $userName = $connectionInfo["userName"];
    $password = $connectionInfo["password"];

    $query = $Query["sqlQuery"];
    $pSize = $Query["pageSize"];
    $pNum = $Query["pageNumber"];
```

Здесь, также, из второго параметра функции извлекаются данные и записываются в соответствующие переменные.

4.4.2 Далее объявляются массивы, в которых будет формироваться результаты работы функции и устанавливается соединение с базой данных.

```
$res = array();
$data = array();
$link = sqlsrv_connect($host, ['UID'=>$userName,
                               'PWD'=>$password,
                               'Database' => $dbName,
                               'CharacterSet' => 'UTF-8']);
if (!$link)                          // UTF-8 обязательна для json_encode
{
    $res[0] = "Error";
    $res[1] = "Host: " . $host;
    $res[2] = "Login: " . $userName ;
    $res[3] = "Password: " . $password;
    $res[4] = "DbName: ". $dbName;
    $res[5] = 'Соединение не установлено: \n'
                    . sqlsrv_errors()[SQLSRV_ERR_ERRORS]['message'];
    goto end;
}
```

4.4.3 После этого начинается работа по определению количества строк, которое возвращает запрос (без учета разбиения на страницы).

В переменную `$sql` записывается копия запроса и, если в запросе есть предложение `order`, оно удаляется из копии запроса.

```
$sql = $query;
$order = fromPosition($sql, 'order');
if ($order > 0) {
    $sql = substr ($sql, 0, $order);
}
```

4.4.4 Далее определяются позиции в запросе, с которых начинаются предложения `select` и `from`.

```
$start = stripos($sql, 'select') + 6;
$end = fromPosition($sql, 'from');
if (!$end)
{
    $res[0] = "Error";
    $res[1] = "Запрос не может быть выполнен";
    $res[2] = "SQL: " . $sql;
    goto end;
}
```

Функция `fromPosition` используется для того, чтобы определить позицию в запросе, с которой начинается слово `select` и слово `from`. Функция работает правильно, независимо от регистра, используемого при написании ключевых слов. Если в предложении `select` используется текстовый литерал, содержащий ключевое слово, или вложенный запрос эта функция отработает правильно, и она найдет адрес ключевого слова `from` правильно.

4.4.5 После определения позиций ключевых слов содержимое предложения `select` удаляется и вместо этого записывается предложение, которое определяет количество строк. После этого запускается запрос, который определяет количество строк.

```
$s = substr($sql, 0, $start) . ' count(*) ' . substr($sql, $end);
$result = sqlsrv_query($link, $s);
if (!$result) {
    $res[0] = "Error";
    $res[1] = "Запрос не выполнен :<br>" . sqlsrv_errors()[SQLSRV_ERR_ERRORS]
    $res[2] = "Исходный SQL: " . $sql;
    $res[3] = "Измененный SQL: " . $s;
    goto end;
}
$line = sqlsrv_fetch_array($result, SQLSRV_FETCH_NUMERIC);
$rowCount = $line[0];
sqlsrv_free_stmt($result);
```

Для извлечения необходимых данных из результатов запроса используется функция `sqlsrv_fetch_array`.

Полученная величина количества строк заносится в переменную `$rowCount` и результат запроса очищается.

4.4.6 Теперь выполняются действия, связанные с обработкой страниц. Определяется переменная `$pMin`, которая должна содержать номер строки, с которой надо извлекать информацию для заданной страницы

Ввиду того, что при работе с данными при получении следующей страницы, другими пользователями могут быть удалены записи или добавлены, а также могут быть ошибочно задан номер страницы, приложением которое использует эта функция, выполняется некоторая обработка, которая ведет к тому, чтобы не возникали различные ошибки при извлечении данных. Так, например,

если указан номер страницы, который находится за пределами диапазона, определяемого количество строк, номер извлекаемой страницы будет откорректирован.

```
if ($pSize > 0)
{  // работа со страницами
    $min = $pSize * ($pNum - 1);
    if ($rowCount > 0) {
        while ($min >= $rowCount)
        {
            $min -= $pSize;
            $pNum--;
        }
        if ($min <= 0) {
            $min = 0;
            $pNum = 1;
        }
    }
    else
    {
        $min = 0;
        $pNum = 1;
    }
```

4.4.7 Далее необходимо в предложение `order` включить опции, которые задают начальную позицию и количество считываемых строк.

```
    $ind = strrpos($query, ';');//OFFSET 0 ROWS FETCH NEXT 10 ROWS ONLY
    if ($ind !== FALSE) {
        $query = substr ($query, 0, $ind);
    }
    if (fromPosition($query, 'order') > 0) {
        $query = $query . ' OFFSET ' . $min . ' ROWS FETCH NEXT '
                                    . $pSize . ' ROWS ONLY;';
    }
    else {
        $query = $query . ' ORDER BY 1 OFFSET ' . $min
                . ' ROWS FETCH NEXT ' . $pSize . ' ROWS ONLY;';
    }
}   // конец работы со страницами
```

4.4.8 Извлечение данных из базы выполняется с помощью функции `sqlsrv_query`.

```
$result = sqlsrv_query($link, $query);
if (!$result) {
    $res[0] = "Error";
    $res[1] = "Запрос не выполнен :<br>"
            . sqlsrv_errors()[SQLSRV_ERR_ERRORS]['message'];
    $res[2] = $query;
    goto end;
}
```

4.4.9 Теперь определяется количество столбцов результирующего набора данных (для этого используется функция `sqlsrv_num_fields`) и выполняются действия по извлечению наименований столбцов результирующего набора и типов данных. Здесь же формируется массив `$res` с итоговыми данными.

```
$fieldCount = sqlsrv_num_fields($result);
//   Получение мета информации
$names = array();
$info = array();
$infoAr = sqlsrv_field_metadata($result);
for ($i = 0; $i < $fieldCount; $i++)
{
    $names[$i] = $infoAr[$i]['Name'];
    $types[$i] = typeOfColumn($infoAr[$i]['Type']);
}
$res[0] = array("OK", $rowCount, $fieldCount, $pNum);
$res[1] = $names;
```

Для получения метаинформации используется функция `sqlsrv_field_metadata()`. Результатом работы этой функции является ассоциативный массив, который содержит метаинформацию по каждому столбцу. Здесь извлекаются сведения о названии столбцов и об их типе. Тип представлен как целое число, поэтому он преобразовывается к необходимому для нас виду с помощью функции `typeOfColumn()`.

4.4.10 Далее извлекаются сведения из результирующего набора и формируется массив с данными.

```
$i = 0;
while ($line = sqlsrv_fetch_array($result, SQLSRV_FETCH_NUMERIC)) {
    $t = $line;
    for ($k = 0; $k < $fieldCount; $k++)
    {
        if ($types[$k] === 'date' && $t[$k] !== NULL)
            $t[$k] = date_format($t[$k], 'Y-m-d');
    }
    $data[$i++] = $t;
}
```

Теперь сформирован массив `$res`, который содержит информацию о результатах работы функции: признак нормального завершения, количество строк, количество столбцов и номер страницы, названия столбцов, а также массив `$data`, содержащий результирующий набор данных в виде массива с числовыми индексами.

4.4.11 После этого освобождается ресурс, закрывается соединение, и функция возвращает результирующий JSON массив.

```
    sqlsrv_free_stmt($result);
    end:
    if ($link) {
        sqlsrv_close($link);
    }
    return  json_encode([$res, $data]);
}
```

4.4.12 Пример использования функции:

```
$conInfo =
        [
            'host' => 'asus',
            'port' => '',
            'dbName' => 'okved_db',
            'userName' => 'php',
            'password' => '*******'
        ];
```

```
$Query =
        [
            'sqlQuery' => 'select * from ul_okved',
            'pageSize' => 8,
            'pageNumber' => 1
        ];
$res = MsExecQuery($conInfo, $Query);
echo $res;
```

Результат работы:

[[["OK",14,6,1],["id","idul","kodokved","main","dtstart","dtend"]],
[[1,1,"101",1,"2012-12-21",null],[2,2,"002",1,"2012-12-12",null],
[4,3,"301.01",1,"2017-11-02",null],[5,4,"301.01",1,"2017-11-01",null],
[6,6,"101",1,"2017-11-08","2018-11-10"],[7,3,"301.02",0,"2016-11-
01",null],[1004,5,"101",1,"2017-12-07",null],[1005,5,"502",0,"2017-12-
08",null]]]

5 ДОСТУП К СУБД FIREBIRD

5.1 Функции PHP для доступа к Firebird

В данном разделе будут рассмотрены следующие функции:

1. ibase_connect
2. ibase_errmsg
3. ibase_close
4. ibase_query
5. ibase_fetch_row
6. ibase_gen_id
7. ibase_free_result
8. ibase_num_fields
9. ibase_affected_rows
10. ibase_field_info

Полный перечень функций, используемых для работы с СУБД Firebird, дан на сайте http://php.net/manual/ru/book.ibase.php.

5.1.1 Функция PHP, обеспечивающая соединение с базой данных, имеет следующую спецификацию:

```
resource ibase_connect ([ string $database [, string $username [, string $password [, string $charset [, int $buffers [, int $dialect[, string $role [, int $sync ]]]]]]])
```

Первый параметр `$database` определяет имя файла базы данных на сервере, на котором она находится, например, `'D:\FireBird\OKVED_BASE.FDB'`. Если выполняется работа с удаленным сервером, то перед этим должно быть задано имя сервера или имя сервера и порт, например, `'hostname:'` или `'hostname/3050:'`.

Параметр `$username` определяет имя пользователя, а `$password` его пароль. Если они не указаны, то используется пользователь и пароль, определенные по умолчанию. В этом случае их значения задаются директивами *ibase.default_user* и *ibase.default_password* в файле `php.ini`. Обычно такая возможность не используется так как с приложениями php работает много пользователей с различными правами доступа.

Параметр $charset определяет кодировку на стороне пользователя.

Остальные параметры, как правило, не используются, их описание можно найти на сайте: http://php.net/manual/ru/function.ibase-connect.php.

5.1.2 Функция ibase_errmsg() возвращает сообщение об ошибке, возникающей при установлении соединения или при выполнении запроса.

5.1.3 Закрытие соединения выполняется с помощью функции ibase_close, которая имеет следующую спецификацию:

```
bool ibase_close ([ resource $connection_id = NULL])
```

где $connection_id — идентификатор соединения, при его отсутствии подразумевается идентификатор последнего открытого соединения.

Функция возвращает TRUE в случае успешного завершения или FALSE в случае возникновения ошибки.

При закрытии соединения освобождается ресурс, после закрытия его использовать нельзя.

5.1.4 Для выполнения запросов к базе данных используется функция ibase_query(), спецификация которой имеет следующий вид:

```
resource ibase_query ([ resource $link_identifier],
string $query)
```

Параметры этой функции имеют следующий вид:

$link_identifier — ресурс, определяющий соединение, которое должно быть создана заранее;

$query — строка, содержащая одну инструкцию SQL.

При возникновении ошибки в процессе выполнения запроса функция возвращает FALSE. Если запрос содержит инструкцию SELECT, возвращает результирующий набор. Если запрос не предусматривает возвращения результирующего набора, возвращает TRUE.

5.1.5 Для получения строки результирующего набора используется функция ibase_fetch_row(), спецификация которой имеет следующий вид:

```
array ibase_fetch_row (resource $result_identifier [,
int $fetch_flag = 0])
```

Она извлекает одну строку данных из результирующего набора. При последующих обращениях возвращается либо следующая строка, либо FALSE, когда строк больше нет.

`$result_identifier` — ресурс, формируемый функцией `ibase_query`.

`$fetch_flag` — является комбинацией констант `IBASE_TEXT` и `UNIXTIME`. Константа `IBASE_TEXT` используется при работе с `BLOB` полями. Если задана `IBASE_TEXT`, функция возвращает `BLOB` контент, в противном случае `BLOB` идентификатор. Применение константы `UNIXTIME` обеспечивает возращение значения даты и времени в виде объекта `Unix timestamps`, отсутствие константы ведет к возвращению даты и времени в виде текстовой строки.

5.1.6 Для работы с генератором, который обычно используется для получения нового значения первичного ключа, используется функция, спецификация которой имеет следующий вид:

```
mixed ibase_gen_id (string $generator
                    [, int $increment = 1
                    [, resource $link_identifier = NULL]]),
```

где `$generator` – имя генератора,

`$increment` – величина увеличения (по умолчанию 1),

`$link_identifier` — ресурс, создаваемый при обращении к функции `ibase_connect`.

Функция `ibase_gen_id` формирует очередное целое значение с использование генератора, заданного первым параметром.

5.1.7 Для освобождения памяти от результирующего набора, полученного при обращении к базе данных, используется функция, которая имеет следующую спецификацию:

```
bool ibase_free_result (resource $result_identifier),
```

где `$result_identifier` — идентификатор, получаемый при выполнении запроса к базе данных с помощью `ibase_query()`.

Эта функция возвращает `TRUE` в случае успешного завершения или `FALSE` в случае возникновения ошибки.

5.1.8 Для получения количества полей, результирующего набора данных используется функция

```
int ibase_num_fields (resource $result_id),
```

где `$result_id` — идентификатор, получаемый при выполнении запроса к базе данных с помощью `ibase_query()`.

5.1.9 При выполнении запросов типа `INSERT`, `UPDATE` или `DELETE` часто требуется выяснить — сколько строк таблицы были охвачены инструкцией такого типа (сколько добавлено, сколько изменено или сколько удалено).

Это можно выполнить с помощью функции, имеющей следующую спецификацию:

```
int ibase_affected_rows ([ resource $link_identifier ] ),
```

где `$link_identifier` — идентификатор соединения, полученный с помощью функции `ibase_connect()`.

Функция возвращает целое число. Если оно большее нуля, то это количество затронутых или полученных строк. Ноль означает, что запросом вида `UPDATE` не обновлено ни одной записи, или что ни одна строка не соответствует условию `WHERE` в запросе. Значение -1 указывает на то, что запрос вернул ошибку.

5.1.10 Для получения метаинформации используется функция `ibase_field_info`, которая имеет следующую спецификацию:

```
array ibase_field_info (resource $result_identifier,
int $field_number)
```

где `$result_identifier` — идентификатор, получаемый при выполнении запроса к базе данных с помощью `ibase_query()`,

`$field_number` — номер поля результирующего набора (отсчет начинается с 0).

Функция возвращает ассоциативный массив, содержащий метаинформацию для заданного столбца.

Можно использовать следующие ключи ассоциативного массива:

name — имя поля,

alias — псевдоним поля (если не задан то равен name),

relation — имя таблицы,

length — длина поля,
type — тип поля.

Тип представлен текстовым полем, совпадающим с наименование типа, принятым в СУБД. Данные представлены в верхнем регистре. Например, 'VARCHAR', 'INTEGER', 'DATE' и т.д.

5.2 Создание вспомогательных функций для Firebird

Функции MyExecNonQuery и MyExecQuery используют следующие вспомогательные функции:

- `parenthesisLevel`,
- `inQuotation`,
- `fromPosition`,
- `tableName`,
- `getNewId`.

5.2.1 Функции `parenthesisLevel`, `inQuotation` и `fromPosition` описаны в разделе 3.2.

5.2.2 Функция `tableName` определяет имя таблицы путем извлечения данных из запроса. Обращение к ней выполняется только в том случае если используется инструкция insert и необходимо использовать генератор для определения нового значения для столбца с автоувеличением.

```
function tableName($q)
{
    $ar = explode(' ', $q);
    for ($i = 1; $i < count($ar); $i++)
    {
        if (strtolower($ar[$i]) != 'into') {
            continue;
        }
        for ($j = $i + 1; $j < count($ar); $j++)
        {
            if ($ar[$j] == null) {
                continue;
            }
            return $ar[$j];
        }
    }
    return '';
}
```

5.2.3 Функция `getNewId` используется только в том случае, если выполняется добавление новой записи с помощью инструкции `INSERT`, и необходимо определить новое значение столбца с автоувеличением. Она определяет новое значение для идентификатора добавляемой строки и преобразует запрос таким образом, чтобы в нем присутствовало имя первичного ключа в перечне вводимых полей, а также значение первичного ключа в перечне `VALUES`.

Например, если исходная строка имеет вид

```
"INSERT INTO ul_okved (IDUL, KODOKVED, MAIN, DTSTART, DTEND) VALUES ('6', '501', 0, '2018-10-10', null);"
```

строка запроса после преобразования будет такой

```
"INSERT INTO ul_okved (ID, IDUL, KODOKVED, MAIN, DTSTART, DTEND) VALUES (331, '6', '501', 0, '2018-10-10', null);"
```

```
function getNewId($link, &$q)
{
    $tName = strtoupper(tableName($q)); // Получение названия таблицы
    $q1 = 'SELECT "RDB$FIELD_NAME", "RDB$GENERATOR_NAME" ' .
          'FROM "RDB$RELATION_FIELDS" ' .
          ' WHERE  NOT "RDB$GENERATOR_NAME" IS NULL AND "RDB$RELATION_NAME" = ';
    $q1 = $q1 . "'" . $tName . "';";
    $r = ibase_query($link, $q1);
    if (!$r) {
        $res[0] = "Error";
        $res[1] = "Запрос не выполнен :<br>" . ibase_errmsg();
        $res[2] = $q1;
        ibase_free_result($r);
        return FALSE;
    }

    $line = ibase_fetch_row($r); // имя поля и имя генератора
    $fName = $line[0]; // определение имени генератора
    $id = ibase_gen_id(trim($line[1]), 1, $link); // значение идентификатор для
    $sk1 = strpos($q, '(');
    $sk2 = strpos($q, '(', $sk1 + 1);
    $q2 = substr($q, 0, $sk1 + 1) . $fName . ',' .
                  substr($q, $sk1 + 1, $sk2 - $sk1) . $id . ',' .
                  substr($q, $sk2 + 1);  // преобразовани запроса
    $q = $q2;
    ibase_free_result($r);
    return $id;
}
```

Здесь сначала определяется имя таблицы, а затем выполняется запрос, который извлекает из системных таблиц имя колонки с автоувеличением и имя генератора для этого поля.

Запрос выполняется с помощью функций `ibase_query()`, а информация извлекается с помощью `ibase_fetch_row()`.

Затем с помощью функции `ibase_gen_id()` формируется новое значение для столбца с автоувеличеним. Хотя, если говорить строго, в Firebird нет понятия столбца с автоувеличеним (как, например, в MySQL), он у нас реализован с помощью функции FbExecNonQuery, которая использует функцию getNewId. При работе с Firebird требуется указать для такого столбца конкретное значение в инструкции SELECT, и это значение определяется функцией getNewId с помощью генератора (это объект, который можно создать и сохранить в СУБД Firebird).

5.3 Создание функции FbExecNonQuery

5.3.1 Функция FbExecNonQuery вначале получает параметры, необходимые для установления соединения с базой данных.

```
function FbExecNonQuery($connectionInfo, $Query)
{
    $host = $connectionInfo["host"];
    if (isset($connectionInfo["port"]) && $connectionInfo["port"] != "") {
        $host .= '/' . $connectionInfo["port"];
    }
    $dbName = $connectionInfo["dbName"];
    $userName = $connectionInfo["userName"];
    $password = $connectionInfo["password"];

    $q = trim($Query["sqlQuery"]);
    if (substr($q, strlen($q) - 1) != ';') {
            $q .= ';';
    }
    if (isset($Query["newId"])) {
        $newId = $Query["newId"];
    }
    else {
        $newId = FALSE;
    }
```

Так как параметры этой функции не имеют полного соответствия с параметрами функции `ibase_connect` приходится выполнять некоторые преобразования.

Заданная с помощью параметров инструкция SQL очищается от пробелов в начале и в конце строки, а также при отсутствии в конце точки с запятой дополняется этим символом. Допускается в качестве параметра задавать строку, которая содержит несколько инструкций типа `update`, разделенных между собой символом ";".

В отличие от других СУБД, Firebird не обеспечивает выполнение несколько инструкций в одном запросе. Эта возможность реализуется самой функцией FbExecNonQuery.

Объединять вместе несколько инструкций не всегда является целесообразным. Например, если объединить несколько инструкций insert и задать параметр newId = true, то будет возвращено значение идентификатора введенной записи только для последней выполненной инструкции.

5.3.2 Далее определяется массив результатов и создается.

```
$res = array();
$link = ibase_connect($host . ":" . $dbName, $userName, $password, 'UTF8');
if (!$link) {                          // UTF8 обязательна для json_encode
    $res[0] = "Error";
    $res[1] = "Host: " . $host;
    $res[2] = "Login: " . $userName;
    $res[3] = "Password: " . $password;
    $res[4] = "DbName: " . $dbName;
    $res[5] = "Соединение не установлено: " . ibase_errmsg();
    goto end;
}
```

Здесь задается кодировка, которая будет использоваться на стороне клиента. В результате этого, независимо от того какая кодировка используется на сервере выходные данные будут представлены в UTF-8.

В случае возникновения ошибки при создании соединения в результирующий массив будут записаны параметры, которые использовались при обращении к функции, а также информация о характере ошибок.

5.3.3 После этого выполняются действия по получению значения первичного ключа для колонки с автоувеличением.

```
$id = 0;
if ($newId && (strtolower(substr($q, 0, 6)) == 'insert')) {
    $id = getNewId($link, $q);
    if ($id == FALSE) {
        goto end;
    }
}
```

В отличие от других СУБД эти действия имеют достаточно сложный характер. Это связано с тем, что реализация колонок с автоувеличением в Firebird не предусмотрена. Поэтому эта возможность реализуется путем обращения к генератору для извлечения очередного значения, которое затем добавляется к запросу. Это выполняется с помощью функции newId. После обращения к этой функции преобразовывается сам запрос.

Для того, чтобы эти действия были возможными, необходимо при создании таблицы создать также генератор для соответствующего столбца. Генератор в Firebird — это объект, который содержит некоторое текущее целое значение, которое может изменяться с помощью системной функции СУБД `GEN_ID`.

5.3.4 СУБД Firebird не обеспечивает возможность выполнения нескольких инструкций в одном запросе. Поэтому далее функция FbExecNonQuery извлекает в цикле инструкции типа `update` из запроса и выполняет их.

```
$start = 0;
$rAf = 0;
while ($start >= 0) // Если строка содержит несколько операторов
{
    $stNew = stripos($q, ';UPDATE', $start);
    if ($stNew == FALSE) {
        $qu = substr($q, $start);
        $start = -1;
    }
    else {
        $qu = substr($q, $start, $stNew - $start + 1);
        $start = $stNew + 1;
    }
```

```
        $result = ibase_query($link, $qu);
        if (!$result) {
            $res[0] = "Error";
            $res[1] = "Запрос не выполнен :<br>" . ibase_errmsg();
            $res[2] = $qu;
            ibase_free_result($r);
            goto end;
        }
        $rAf += ibase_affected_rows($link);
        ibase_free_result($r);
    }
```

В случае возникновения ошибки в результирующий массив записываются сведения об ошибке, а также выполняемый SQL запрос.

Если запрос выполнен без ошибок формируется результирующий массив и преобразовывается в JSON массив. Результат, полученный функцией, очищается с помощью `ibase_free_result`.

5.3.5 Далее формируется результирующий массив.

```
        $res[0] = "OK";
        $res[1] = $rAf;
        $res[2] = $id;
        end:
        if ($link) {
            ibase_close($link);
        }
        return  json_encode($res);
    }
```

5.3.6 Проверка работы функции ***FbExecNonQuery***:

Добавление новой записи.

```
$conInfo =
        [
            'host' => 'localhost',
            'port' => 3050,
            'dbName' => 'D:\FireBird\OKVED_BASE.FDB',
            'userName' => 'sysdba',
            'password' => '********'
        ];
$Query =
[
  'sqlQuery' => "INSERT INTO  ul_okved (IDUL, KODOKVED, MAIN, DTSTART, DTEND) "
                        . ". VALUES ('6', '501', 0, '2018-10-10', null);",
];
```

Результат: ["OK",1,320] — инструкция выполнена, одна запись добавлена и ее идентификатор равен 320.

Корректировка данных.

```
$Query =
[
    'sqlQuery' => "UPDATE ul_okved SET DTEND = '2020-11-01' WHERE ID = 306;"
          . "UPDATE ul_okved SET DTEND = '2019-11-01' WHERE ID = 307;",
];

$res = FbExecNonQuery($conInfo, $Query);
echo $res;
```

Результат: ["OK",2,0]– выполнено две инструкции, обновлены две записи.

Удаление данных:

```
$Query =
[
    'sqlQuery' => "DELETE FROM ul_okved WHERE id >= 320"
];

$res = FbExecNonQuery($conInfo, $Query);
echo $res;
```

Результат: ["OK",1,0]– выполнено удаление одной записи.

5.4 Создание функции FbExecQuery

5.4.1 Функция FbExecNonQuery вначале получает параметры, необходимые для установления соединения с базой данных.

```
function FbExecQuery($connectionInfo, $Query)
{
    $host = $connectionInfo["host"];
    if (isset($connectionInfo["port"]) && $connectionInfo["port"] != "") {
        $host .= '/' . $connectionInfo["port"];
    }
    $dbName = $connectionInfo["dbName"];
    $userName = $connectionInfo["userName"];
    $password = $connectionInfo["password"];
    $query = $Query["sqlQuery"];
    $pSize = $Query["pageSize"];
    $pNum = $Query["pageNumber"];
```

Здесь, также, из второго параметра функции извлекаются данные и записываются в соответствующие переменные.

5.4.2 Далее объявляются массивы, в которых будет формироваться результаты работы функции и устанавливается соединение с базой данных.

```
$res = array();
$data = array();
$link = ibase_connect($host . ":" . $dbName, $userName, $password, 'UTF8');
if (!$link) {                                                    // UTF8 обязательна д
    $res[0] = "Error";
    $res[1] = "Host: " . $host;
    $res[2] = "Login: " . $userName;
    $res[3] = "Password: " . $password;
    $res[4] = "DbName: " . $dbName;
    $res[5] = "Соединение не установлено: " . ibase_errmsg();
    goto end;
}
```

5.4.3 После этого начинается действия по определению количества строк, возвращаемых этим запросом (без учета разбиения на страницы).

В переменную `$sql` записывается копия запроса и, если в запросе есть предложение `order`, оно удаляется из копии запроса.

```
$sql = $query;
$sqll = strtolower($sql);
$start = stripos($sqll, 'order by');
$s = $sql;
if ($start !== FALSE) {
    $s = substr($sql, 0, $start);
}
```

5.4.4 Теперь определяются позиции в запросе, с которых начинаются предложения `select` и `from`.

```
$start = stripos($sql, 'select') + 6;
$end = fromPosition($sql, 'from');
if (!$end)
{
    $res[0] = "Error";
    $res[1] = "Запрос не может быть выполнен";
    $res[2] = "SQL: " . $sql;
    goto end;
}
```

Функция `fromPosition` используется для того, чтобы определить позицию в запросе, с которой начинается слово `select` и

слово `from`. Функция работает правильно, независимо от регистра, используемого при написании ключевых слов. Если в предложении `select` используется текстовый литерал, содержащий ключевое слово, или вложенный запрос эта функция отработает правильно и она найдет адрес ключевого слова `from` правильно.

5.4.5 После определения позиций ключевых слов содержимое предложения `select` удаляется и вместо этого записывается предложение, которое определяет количество строк. После этого запускается запрос, который определяет количество строк.

```
$sl = substr($s, 0, $start) . ' count(*) ' . substr($s, $end);
$r = ibase_query($link, $sl);
if (!$r) {
    $res[0] = "Error";
    $res[1] = "Запрос не выполнен :<br>" . ibase_errmsg();
    $res[2] = $sql;
    $res[3] = $s;
    goto end;
}
$line = ibase_fetch_row($r);
$rowCount = $line[0];
ibase_free_result($r);
```

Для извлечения необходимых данных из результатов запроса использовалась функция `ibase_fetch_row`, которая возвращает строку содержащую только одно значение — количество строк. Полученная величина заносится в переменную `$rowCount` и результат запроса очищается.

5.4.6 Теперь выполняются действия, связанные с обработкой страниц. Определяется переменная `$pMin`, которая должна содержать номер строки, с которой надо извлекать информацию для заданной страницы.

Ввиду того, что при работе с данными при получении следующей страницы, могут быть удалены записи или добавлены, а также могут быть ошибки при определении номера страницы, приложением которое использует эта функция, выполняется некоторая обработка, которая ведет к тому, чтобы не возникали различные ошибочные ситуации при извлечении данных. Так, например, если указан номер страницы, который находится за пределами диапазона,

определяемого количество строк, номер извлекаемой страницы может быть откорректирован.

5.4.7 В результате этих действий формируется запрос для извлечения данных только для одной страницы. В этом запросе используются предложения `FIRST` и `SKIP`, которые определяют количество считываемых строк и номер начальной строки.

```
if ($pSize > 0)
{  // работа со страницами
    $min = $pSize * ($pNum - 1);
    if ($rowCount > 0)
    {
        while ($min >= $rowCount) {
            $min -= $pSize;
            $pNum--;
        }
        if ($min <= 0) {
            $min = 0;
            $pNum = 1;
        }
    }
    else {
        $min = 0;
        $pNum = 1;
    }

    $query = substr($query, 0, $start) . ' FIRST ' . $pSize . ' SKIP ' . $min
}   // конец работы со страницами
```

5.4.8 Извлечение данных из базы выполняется с помощью функции `ibase_query`.

```
$result = ibase_query($link, $query);
if (!$result) {
    $res[0] = "Error";
    $res[1] = "Запрос не выполнен :<br>" . ibase_errmsg();
    $res[2] = $query;
    goto end;
}
```

5.4.9 Теперь определяется количество столбцов результирующего набора данных с помощью функции `ibase_num_fields`, и выполняются действия по извлечению наименований столбцов результирующего набора.

```
$fieldCount = ibase_num_fields($result);
//   Получение мета информации
$names = array();

for ($i = 0; $i < $fieldCount; $i++)
{
    $info = ibase_field_info($result, $i);
    $names[$i] = $info['name'];
}
```

Для получения названий столбцов используется функция `ibase_field_info`.

5.4.10 Далее формируется массив `$res`, содержащий результаты работы, и извлекаются данные из базы.

```
$res[0] = array("OK", $rowCount, $fieldCount, $pNum);
$res[1] = $names;
$i = 0;
while ($line = ibase_fetch_row($result)) {
    $data[$i++] = $line;
}
ibase_free_result($result);
```

Полученный массив `$res` содержит информацию о результатах работы функции: признак нормального завершения, количество строк, количество столбцов и номер страницы, названия столбцов, а массив `$data` — набор данных, полученных в результате выполнения запроса в виде массива с числовыми индексами.

5.4.11 После этого освобождается ресурс, закрывается соединение, и функция возвращает результирующий JSON массив.

```
    ibase_free_result($result);
    end:
    if ($link) {
        ibase_close($link);
    }
    return  json_encode([$res, $data]);
}
```

5.4.12 Пример использования функции:

```
$conInfo =
        [
            'host' => 'localhost',
            'port' => 3050,
            'dbName' => 'D:\FireBird\OKVED_BASE.FDB',
            'userName' => 'PHP',
            'password' => '*****'
        ];
$Query =
        [
            'sqlQuery' => 'select * from ul_okved',
            'pageSize' => 10,
            'pageNumber' => 2
        ];
$res = FbExecQuery($conInfo, $Query);
echo $res;
```

Результат работы:

[[["OK",72,6,2],["ID","IDUL","KODOKVED","MAIN","DTSTART","DTEND"]],[[49,2,"101",0,"2010-12-27","2010-12-27"],[50,3,"001",1,"2010-12-28","2012-11-30"],[51,3,"303",0,"2013-09-18","2013-10-24"],[55,3,"502",0,"2013-12-01",null],[56,3,"503",1,"2012-12-31",null],[99,4,"502",1,"2015-07-09",null],[103,5,"102",1,"2015-07-31","2016-09-26"],[104,5,"601",0,"2015-09-27",null],[105,5,"502",0,"2015-07-03",null],[107,6,"002",1,"2015-07-12",null]]]

6 ДОСТУП К СУБД POSTRESQL

6.1 Функции PHP для доступа к PostreSQL

В данном разделе будут описаны следующие функции:

1. pg_connect
2. pg_last_error
3. pg_close
4. pg_query
5. pg_result_error
6. pg_fetch_result
7. pg_fetch_array
8. pg_free_result
9. pg_num_fields
10. pg_num_rows
11. pg_affected_rows
12. pg_field_name

Полный перечень функций, используемых для работы с СУБД PostreSQL, дан на сайте http://php.net/manual/ru/book.pgsql.php.

6.1.1 Функция PHP, обеспечивающая соединение с базой данных, имеет следующую спецификацию:

```
resource pg_connect (string $connection_string [,int $connect_type])
```

Первый параметр `$connection_string` определяет строку соединения.

При повторном вызове функции `pg_connect()` с теми же значениями параметров в `$connection_string` функция вернет существующее подключение. Чтобы принудительно создать новое соединение, необходимо передать строку подключения функции `PGSQL_CONNECT_FORCE_NEW` в качестве параметра `$connect_type`.

Строка `$connection_string` может быть пустой строкой или содержать несколько параметров, разделенных пробелами. Каждый параметр указывается как `keyword = value`. Пробелы вокруг знака "равно" необязательны. Пустые строки в качестве значения или значения, содержащие пробелы отделяются одинарными кавычками, как например, `keyword = 'a value'`.

Список основных ключевых слов строки соединения:
host, port, dbname, user, password, connect_timeout, где

host — имя сервера,

port — номер порта,

dbname — имя базы данных,

user — имя пользователя,

password — пароль,

connect_timeout — время (в секундах), отведенное для установки соединения, если за это время создать соединение не удалось, попытка установить соединение прекращается.

Например, строка соединения может выглядеть следующим образом:

```
'host=localhost port=5432 dbname=okved_db user=php password=pass'
```

Необходимо обратить внимание, что в отличие от большинства других СУБД, PostgreSQL требует, чтобы в строке соединения разделителем параметров был пробел (в других серверах используется точка с запятой). Между знаком "=" и ключевым словом и значением можно ставить пробелы, например, 'host = localhost'.

Параметр $connect_type может содержать константу.

Если в качестве $connect_type используется константа PGSQL_CONNECT_FORCE_NEW, будет создаваться новое подключение, даже если $connection_string идентична строке существующего подключения.

Если используется константа PGSQL_CONNECT_ASYNC, то соединение устанавливается асинхронным. Состояние соединения можно проверить с помощью функций pg_connect_poll() или pg_connection_status().

6.1.2 Для получения сообщения об ошибке, возникшей в процессе создания соединения, используется функция pg_last_error, которая имеет следующую спецификацию:

```
string pg_last_error ([ resource $connection ] ),
```

где $connection — ресурс подключения к базе данных PostgreSQL. Если параметр $connection не задан, будет использо-

вано подключение по умолчанию - последнее соединение, открытое функцией pg_connect().

Функция pg_last_error возвращает строку, содержащую сообщение о последней ошибке, произошедшей на соединении $connection, либо FALSE в случае ошибки.

6.1.3 Закрытие соединения выполняется с помощью функции sqlsrv_close, которая имеет следующую спецификацию:

```
bool pg_close ([ resource $connection]),
```

где $connection — ресурс соединения с базой данных PostgreSQL. В случае, если $connection не задан, будет закрыто последнее открытое соединение открытое.

При закрытии соединения освобождается ресурс, после закрытия его использовать нельзя.

6.1.4 Выполнение запроса к базе данных производится с помощью функции pg_query, спецификация которой имеет следующий вид:

```
resource pg_query ([ resource $connection], string $query)
```

В случае ошибки функция возвращает FALSE, детали ошибки можно получить с помощью функции pg_last_error() если соединение с базой данных не нарушено.

Параметры этой функции имеют следующий вид:

$connection — ресурс соединения с базой данных PostgreSQL. Если не передать параметр $connection, используется соединение по умолчанию. Соединение по умолчанию — это последнее созданное соединение.

$query — одна или несколько инструкций SQL для выполнения. Если передано несколько инструкций они автоматически выполняются как одна транзакция если явно не указаны команды BEGIN/COMMIT внутри выражения. Тем не менее, использовать несколько транзакций в одном вызове функции не рекомендуется.

В случае удачного выполнения функции возвращается ресурс результата запроса, а в случае возникновения ошибки возвращается FALSE.

6.1.5 Для возвращения сообщения об ошибке, возникшей при выполнении запроса, используется функция pg_result_error, которая имеет следующую спецификацию:

```
string pg_result_error ( resource $result ),
```

где $result — ресурс результата запроса PostgreSQL, возвращенный pg_query(), а также другими функциям, создающими ресурс результата.

Функция возвращает строку (string). Если нет ошибки возвращает пустую строку. Если же есть ошибка, не связанная с параметром $result, то возвращается FALSE.

6.1.6 Получение значения столбца из одной строки результата выполняется с помощью функции pg_fetch_result, которая имеет следующую спецификацию:

```
string pg_fetch_result (resource $result , int $row ,
mixed $field ) ,
```

где $result — ресурс результата запроса PostgreSQL, возвращаемый функцией pg_query().

$row — номер выбираемой из результата запроса строки. Нумерация начинается с нуля. Если аргумент опущен, берется следующая по очереди строка.

$field — имя или номер поля выбираемого значения. Поля нумеруются с нуля.

6.1.7 Получение результирующего набора в виде массива после выполнения запроса производится с помощью функции pg_fetch_array, которая имеет следующую спецификацию:

```
array pg_fetch_array (resource $result [, int $row [,
int $result_type = PGSQL_BOTH]]) ,
```

где $result — номер ресурс результата запроса PostgreSQL, возвращенный функцией pg_query().

$row — номер строки в ресурсе $result для выборки. Строки пронумерованы с 0 по возрастанию. Если параметр опущен или передан NULL будет выбрана следующая строка.

$result_type — необязательный параметр для управления типом индексации возвращаемого массива. Параметр $result_type является обязательным и может принимать следующие значения: PGSQL_ASSOC, PGSQL_NUM и PGSQL_BOTH. При указании PGSQL_NUM, функция pg_fetch_array() вернет массив с числовыми индексами, в случае PGSQL_ASSOC вернет только ассоциативные индексы, а в случае PGSQL_BOTH (используется по умолчанию) - числовые и ассоциативные индексы.

Функция pg_fetch_array() возвращает FALSE, если **$row** выходит за рамки количества строк в выборке, или отсутствия строк, а также в случае любой другой ошибки.

6.1.8 Для очистки результатов запроса и освобождения памяти используется функция pg_free_result, которая имеет следующую спецификацию:

```
bool pg_free_result ( resource $result ),
```

где $result — ресурс результата запроса PostgreSQL, возвращенный pg_query(), а также другими функциям, создающими ресурс результата.

Функция возвращает TRUE в случае успешного завершения или FALSE в случае возникновения ошибки.

Вызывать эту функцию следует только в случае нехватки памяти при выполнении скрипта. В любом случае память будет освобождена автоматически по окончании работы скрипта.

6.1.9 Для получения количества полей, результирующего набора данных используется функция

```
int pg_num_fields (resource $result),
```

где $result — идентификатор, получаемый при выполнении запроса к базе данных с помощью pg_query().

6.1.10 Для получения количества строк результирующего набора данных используется функция

```
int pg_num_rows (resource $result),
```

где $result — идентификатор, получаемый при выполнении запроса к базе данных с помощью pg_query().

6.1.11 Функция pg_affected_rows, определяющая количество записей, затронутых выполненными инструкциями обновления, имеет следующую спецификацию:

```
int pg_affected_rows (resource $result),
```

где $result — результат запроса к PostgreSQL, значение типа "ресурс", возвращаемое функцией pg_query().

Если ни одна запись не была затронута, функция вернет 0.

6.1.12 Для получения метаинформации используется следующие функции: pg_field_is_null, pg_field_name, pg_field_num, pg_field_prtlen, pg_field_size, pg_field_table, pg_field_type_oid, pg_field_type.

Рассмотрим функцию pg_field_name, которая извлекает имена полей. Она имеет спецификацию:

```
string pg_field_name (resource $result, int
$field_number),
```

где $result — ресурс результата запроса PostgreSQL, возвращаемый функцией pg_query(), и некоторыми другими функциями.

$field_number — номер поля, начиная с нуля.

Возвращаемое значение содержит наименование поля, либо FALSE в случае ошибки.

Обращение к другим функциям выполняется аналогично. Подробно эти функции описаны на сайте http://php.net/manual/ru/book.pgsql.php.

6.2 Создание вспомогательных функций для PostreSQL

Функции PgExecNonQuery и PgExecQuery используют следующие вспомогательные функции:

- parenthesisLevel,
- inQuotation,
- fromPosition.

Они описаны в разделе 3.2.

6.3 Создание функции PgExecNonQuery

6.3.1 Функция PgExecNonQuery вначале получает параметры, необходимые для установления соединения с базой данных.

```
function PgExecNonQuery($connectionInfo, $Query)
{

    $host = $connectionInfo["host"];
    if (isset($connectionInfo["port"]) && $connectionInfo["port"] != "") {
        $port = ' port=' . $connectionInfo["port"];
    }
    $dbName = $connectionInfo["dbName"];
    $userName = $connectionInfo["userName"];
    $password = $connectionInfo["password"];

    $query = $Query["sqlQuery"];
    $q = trim($Query["sqlQuery"]);
    if (substr($q, strlen($q) - 1) != ';') {
            $q .= ';';
    }
```

Так как параметры этой функции не имеют полного соответствия с параметрами функции `sqlsrv_connect` приходится выполнять некоторые преобразования.

Заданная с помощью параметров инструкция SQL очищается от пробелов в начале и в конце строки, а также при отсутствии в конце точки с запятой дополняется этим символом. Допускается в качестве параметра задавать строку, которая содержит несколько инструкций, разделенных между собой символом ";".

Объединять вместе несколько инструкций не всегда является целесообразным. Например, если объединить несколько инструкций `INSERT` и задать параметр `newId = true`, то будет возвращено значение идентификатора введенной записи только для последней выполненной инструкции.

6.3.2 В отличие от функций для СУБД MySQL здесь отсутствует функция по извлечению идентификатора для вновь созданной записи (это необходимо, если первичным ключом является поле с автоувеличением. Подготовительные действия по получению вновь созданного первичного ключа выглядят следующим образом:

```
if (isset($Query["newId"])) {
    $newId = $Query["newId"];
}
else {
    $newId = FALSE;
}
if ($newId && (strtolower(substr($q, 0, 6)) == 'insert')) {
    $q .= 'SELECT LASTVAL();'; // извлекает последнее id
}
$res = array();
```

Для получения нового первичного ключа используется стандартная функция `LASTVAL()`, которая будет выполнена вместе с SQL инструкцией по вводу новой строки.

Определяемый здесь массив `$res` будет содержать результаты выполнения инструкции.

6.3.3 Далее создается соединение.

```
$link = pg_connect("host=" . $host . " dbname=" . $dbName .
        ' client_encoding=UTF8' . $port . " user=" .
        $userName . " password=" . $password); // UTF8 обязательна для
if (!$link) {
    $res[0] = "Error";
    $res[1] = "Host: " . $host . ' ' . $port;
    $res[2] = "Login: " . $userName ;
    $res[3] = "Password: " . $password;
    $res[4] = "DbName: ". $dbName;
    $res[5] = "Связь с базой данных не установлена " . pg_last_error();
    goto end;
}
```

Здесь также задается кодировка, которая будет использоваться на стороне клиента. В результате этого, независимо от того какая кодировка используется на сервере выходные данные будут представлены в UTF-8.

В случае возникновения ошибки при создании соединения в результирующий массив будут записаны параметры, которые использовались при обращении к функции, а также информация о характере ошибок.

6.3.4 Обращение к функции `pg_query` выполним следующим образом:

```
$qCount = 0;
if ($newId && (strtolower(substr($q, 0, 6)) == 'insert')) {
    $qCount = count(explode(';', $query)) - 1; // количество запросов
}
$result = pg_query($link, $q);
if (!$result) {
    $res[0] = "Error";
    $res[1] = "Запрос не выполнен :<br>" . pg_last_error($link);
    $res[2] = $query;
    goto end;
}
```

В случае возникновения ошибки в результирующий массив записываются сведения об ошибке, а также выполняемый SQL запрос.

6.3.5 Если запрос выполнен без ошибок формируется результирующий массив и преобразовывается в JSON массив.

```
    $res[0] = "OK";
    $res[1] = pg_affected_rows($result);
    if ($qCount > $res[1]) {
        $res[1] = $qCount;  // В строке несколько запросов
    }
    $res[2] = pg_fetch_result($result, 0);
    end:
    if ($link) {
        pg_close ($link);
    }
    return json_encode($res);
}
```

Извлечение идентификатора вновь введенной записи выполняется с помощью функции `pg_fetch_result`.

6.3.6 Проверка работы функции ***PgExecNonQuery***:

Добавление новой записи.

```
$conInfo =
        [
            'host' => 'localhost',
            'dbName' => 'okved_db',
            'port' => 5432,
            'userName' => 'php',
            'password' => '********'
        ];
```

```
$Query =
[
  'sqlQuery' => "INSERT INTO ul_okved (idul, kodokved, main, dtstart) "
              . "VALUES (3, '301', false, '2018-10-10');"
];
$res = PgExecNonQuery($conInfo, $Query);
echo $res;
```

Результат: ["OK",1,"123"] — инструкция выполнена, одна запись добавлена и ее идентификатор равен 123.

Корректировка данных.

```
$Query =
[
   'sqlQuery' => "UPDATE ul_okved SET dtend = '2017-03-14' WHERE id = 119;"
];
$res = PgExecNonQuery($conInfo, $Query);
echo $res;
```

Результат: ["OK",1,false] — выполнена одна инструкция, обновлена одна запись.

Удаление данных:

```
$Query =
[
   'sqlQuery' => "DELETE FROM ul_okved WHERE id >= 123"
];
$res = PgExecNonQuery($conInfo, $Query);
echo $res;
```

Результат: ["OK",1,false] — выполнено удаление одной записи.

6.4 Создание функции PgExecQuery

6.4.1 Функция PgExecNonQuery вначале получает параметры, необходимые для установления соединения с базой данных.

```
function PgExecQuery($connectionInfo, $Query)
{
    $host = $connectionInfo["host"];
    if (isset($connectionInfo["port"]) && $connectionInfo["port"] != ""){
        $port = ' port=' . $connectionInfo["port"];
    }
    else {
        $port = '';
    }
```

```
$dbName = $connectionInfo["dbName"];
$userName = $connectionInfo["userName"];
$password = $connectionInfo["password"];
$query = $Query["sqlQuery"];
$pSize = $Query["pageSize"];
$pNum = $Query["pageNumber"];
```

Здесь, также, из второго параметра функции извлекаются данные и записываются в соответствующие переменные.

6.4.2 Далее объявляются массивы, в которых будет формироваться результаты работы функции и устанавливается соединение с базой данных.

```
$res = array();
$data = array();
$link = pg_connect("host=" . $host . " dbname=" . $dbName
        . ' client_encoding=UTF8'  // UTF8 обязательна для json_encode
        . $port . " user=" . $userName . " password=" . $password);
if (!$link) {
    $res[0] = "Error";
    $res[1] = "Host: " . $host;
    $res[2] = "Port: " . $port;
    $res[3] = "Login: " . $userName ;
    $res[4] = "Password: " . $password;
    $res[5] = "DbName: ". $dbName;
    $res[6] = "Связь с базой данных не установлена " . pg_last_error();
    goto end;
}
```

6.4.3 После этого начинаются действия по определению количества строк, возвращаемых этим запросом (без учета разбиения на страницы).

В переменную `$sql` записывается копия запроса и, если в запросе есть предложение `order`, оно удаляется из копии запроса.

```
if ($pSize > 0)  // работа со страницами  ///////
{
    $sql = $query;
    $start = fromPosition($sql, 'order');
    $s = $sql;
    if ($start !== FALSE) {
        $s = substr($sql, 0, $start);
    }
```

Функция `fromPosition` используется для того, чтобы определить позицию в запросе, с которой начинается слово `select` и

слово from. Функция работает правильно, независимо от регистра, используемого при написании ключевых слов. Если в предложении select используется текстовый литерал, содержащий ключевое слово или вложенный запрос, эта функция найдет адрес ключевого слова from для основного запроса.

6.4.4 Теперь определяются позиции в запросе, с которых начинаются предложения select и from. После определения позиций ключевых слов содержимое предложения select удаляется и вместо этого записывается предложение, которое определяет количество строк.

```
$start = strpos($sql, 'select') + 6;
$end = strpos($sql, 'from');
$sl = substr($s, 0, $start) . ' count(*) ' . substr($s, $end);
```

6.4.5 Далее запускается запрос, который определяет количество строк.

```
$rl = pg_query($link, $sl);
if (!$rl) {
    $res[0] = "Error";
    $res[1] = "Запрос по подсчету строк не выполнен :<br>"
                                    . pg_result_error($link);
    $res[2] = "Подсчет строк: " . $sl;
    $res[3] = "Исходный запрос: " . $query;
    goto end;
}
$line = pg_fetch_array($rl, 0, PGSQL_NUM);
$rowCount = (int)$line[0]; // $line[0] возвращает строковое значение
pg_free_result($rl);
```

Для извлечения необходимых данных из результатов запроса использовалась функция pg_fetch_array, которая возвращает массив, состоящий из одной строки, содержащей только одно значение — количество строк. Полученная величина заносится в переменную $rowCount и результат запроса очищается.

6.4.6 Теперь выполняются действия, связанные с обработкой страниц. Определяется переменная $pMin, которая должна содержать номер строки, с которой надо извлекать информацию для заданной страницы.

Ввиду того, что при работе с данными при получении следующей страницы, могут быть удалены записи или добавлены, а также

могут быть ошибки при определении номера страницы, приложением которое использует эта функция, выполняется некоторая обработка, которая ведет к тому, чтобы не возникали различные ошибочные ситуации при извлечении данных. Так, например, если указан номер страницы, который находится за пределами диапазона, определяемого количество строк, номер извлекаемой страницы может быть откорректирован.

В результате этих действий формируется запрос для извлечения данных только для одной страницы. В этом запросе используются предложения `limit` и `offset`, которые определяют количество считываемых строк и номер начальной строки.

```
    $min = $pSize * ($pNum - 1);
    if ($rowCount > 0)
    {
        while ($min >= $rowCount) {
            $min -= $pSize;
            $pNum--;
        }
        if ($min <= 0) {
            $min = 0;
            $pNum = 1;
        }
    }
    else {
        $min = 0;
        $pNum = 1;
    }
    $ind = strpos($query, ';');
    if ($ind !== FALSE)
        $query = substr ($query, 0, $ind);
    $query = $query . ' limit ' . $pSize . ' offset ' . $min . ';';
}   // конец работы со страницами  ///////
```

6.4.7 Извлечение данных из базы выполняется с помощью функции `pg_query`.

```
$result = pg_query($link, $query);
if (!$result)
{
    $res[0] = "Error";
    $res[1] = "Запрос не выполнен :<br>" . pg_result_error($link);
    $res[2] = $query;
    goto end;
}
```

6.4.8 Теперь извлекаются данные из базы. Сформированный массив `$data` содержит набор данных, полученных в результате выполнения запроса в виде массива с числовыми индексами.

С помощью функции `pg_num_rows` определяется количествок строк в данных (если нет разбиения на страницы). Определяется количество столбцов результирующего набора данных с помощью функции `pg_num_fields`, и выполняются действия по извлечению наименований столбцов результирующего набора (при этом используется функция `pg_field_name`).

```
$names = array();
$i = 0;
while ($row = pg_fetch_array($result, null, PGSQL_NUM)) {
    $data[$i++] = $row;
}
if ($pSize == 0) {  // работа без страниц
    $rowCount = pg_num_rows($result);
}
$fieldCount = pg_num_fields($result);
for ($i = 0; $i < $fieldCount; $i++)
{
    $names[$i] = pg_field_name($result, $i);
}
```

6.4.9 Далее формируется массив $res, содержащий результаты работы.

```
$res[0] = array("OK", $rowCount, $fieldCount, $pNum);
$res[1] = $names;
```

Полученный массив $res содержит информацию о результатах работы функции: признак нормального завершения, количество строк, количество столбцов и номер страницы, названия столбцов.

6.4.10 После этого освобождается ресурс, закрывается соединение, и функция возвращает результирующий JSON массив.

```
    pg_free_result($result);
    end:
    if ($link) {
        pg_close ($link);
    }
    return  json_encode([$res, $data]);
}
```

6.4.11 Пример использования функции:

```
$conInfo =
        [
            'host' => 'localhost',
            'port' => 5432,
            'dbName' => 'okved_db',
            'userName' => 'php',
            'password' => '******'
        ];
$Query =
        [
            'sqlQuery' => 'select * from ul_okved',
            'pageSize' => 10,
            'pageNumber' => 1
        ];
$res = PgExecQuery($conInfo, $Query);
echo $res;
```

Результат работы:

[[["OK",10,6,1],["id","idul","kodokved","dtstart","dtend","main"]],[["38","2","301","2009-12-12","2010-10-26","f"],["39","3","001","2010-10-25","2011-10-25","t"],["40","3","101","2010-10-25",null,"t"],["41","3","301","2009-11-25","2010-10-16","t"],["43","1","102","2010-11-16","2015-08-25","t"],["44","2","102","2007-11-16",null,"t"],["118","1","502","2018-10-11",null,"f"],["42","1","002","2007-11-15","2016-03-18","t"],["37","2","001","2004-12-18","2017-03-04","f"],["119","3","301","2017-12-12","2017-03-14","f"]]

СПИСОК ЛИТЕРАТУРЫ

1. Улучшенный модуль MySQL (MySQL Improved) [Электронный ресурс] / Режим доступа: http://php.net/manual/ru/book.mysqli.php

2. Драйвер СУБД Microsoft SQL Server для PHP [Электронный ресурс] / Режим доступа: http://php.net/manual/ru/book.sqlsrv.php

3. Firebird/InterBase [Электронный ресурс] / Режим доступа: http://php.net/manual/ru/book.ibase.php

4. PostgreSQL [Электронный ресурс] / Режим доступа: http://php.net/manual/ru/book.pgsql.php

5. Драйверы Microsoft SQL Server для PHP [Электронный ресурс] / Режим доступа: https://docs.microsoft.com/ru-ru/sql/connect/php/microsoft-php-driver-for-sql-server?view=sql-server-2017

ПРИЛОЖЕНИЕ 1
"ЛИСТИНГ ФУНКЦИЙ ДЛЯ MYSQL"

```
function parenthesisLevel($s, $fromPos)
/*
        * Author: Vl.Shabashov
*/
{
    $level = 0;
    for ($i = 6; $i < $fromPos; $i++)
    {
        if (substr($s, $i, 1) == '(') {
                $level++;
        }
        if (substr($s, $i, 1) == ')') {
                $level--;
        }
    }
    return $level;
}

function inQuotation($s, $fromPos)
/*
        * Author: Vl.Shabashov
*/
{
    $inQuot = 0;
    for ($i = 6; $i < $fromPos; $i++)
    {
        if (substr($s, $i, 1) == "'") {
                $inQuot = $inQuot ^ 1;
        }
    }
    return $inQuot;
}

function fromPosition($s, $from)
/*
        * Author: Vl.Shabashov
*/
{
```

```
    $ar = str_word_count($s, 2, '_');
    foreach ($ar as $key => $value)
    {
        if (strtolower($value) != $from) {
            continue;
        }
        if (parenthesisLevel($s, $key) == 0 && inQuota-
tion($s, $key) == 0) {
            return $key;
        }
    }
    return FALSE;
}

function MyExecQuery($connectionInfo, $Query)
/*
       * Author: Vl.Shabashov
*/
{
    $host = $connectionInfo["host"];
    $port = $connectionInfo["port"];
    $dbName = $connectionInfo["dbName"];
    $userName = $connectionInfo["userName"];
    $password = $connectionInfo["password"];

    $query = $Query["sqlQuery"];
    $pSize = $Query["pageSize"];
    $pNum = $Query["pageNumber"];

    $res = array();
    $data = array();
    $link = mysqli_connect($host, $userName, $password,
$dbName, $port);
    if (mysqli_connect_errno($link))
    {
        $res[0] = "Error";
        $res[1] = "Host: " . $host;
        $res[2] = "Login: " . $userName;
        $res[3] = "Password: " . $password;
        $res[4] = "DbName: " . $dbName;
        $res[5] = "Port: " . $port;
        $res[6] = "Соединение не установлено: " .
```

```
mysqli_connect_error();
        goto end;
    }
    mysqli_set_charset($link, 'utf8');  // UTF8 обязатель-
на для json_encode

    if ($pSize > 0)  // работа со страницами
    {
        $sql = $query;
        $start = fromPosition($sql, 'select') + 6;
        $end = fromPosition($sql, 'from');
        $s = substr($sql, 0, $start) . ' count(*) ' . sub-
str($sql, $end);

        $result = mysqli_query($link, $s);
        if (!$result) {
            $res[0] = "Error";
            $res[1]  =  "Запрос  не  выполнен  :<br>"  .
mysqli_error($link);
            $res[2] = $sql;
            $res[3] = $s;
            goto end;
        }
        $line = mysqli_fetch_array($result, MYSQLI_NUM);
        $rowCount = $line[0];
        mysqli_free_result($result);

        $min = $pSize * ($pNum - 1);
        if ($rowCount > 0) {
            while ($min >= $rowCount) {
                $min -= $pSize;
                $pNum--;
            }
            if ($min <= 0) {
                $min = 0;
                $pNum = 1;
            }
        }
        else {
            $min = 0;
            $pNum = 1;
        }
```

```
        $ind = strrpos($query, ';');
        if ($ind !== FALSE) {
            $query = substr ($query, 0, $ind);
        }
        $query = $query . ' limit ' . $min . ', ' . $pSize
. ';';
     }   // конец работы со страницами

    $result = mysqli_query($link, $query);
    if (!$result) {
        $res[0] = "Error";
        $res[1]   =   "Запрос   не   выполнен   :<br>"   .
mysqli_error($link);
        $res[2] = $query;
        goto end;
    }
    $fieldCount = mysqli_num_fields($result);
    if ($pSize == 0) {
        $rowCount = mysqli_num_rows($result);
    }
    $names = array();
    for ($i = 0; $i < $fieldCount; $i++) {
        $info = mysqli_fetch_field($result);
        $names[$i] = $info->orgname;
        if ($names[$i] == '') {
            $names[$i] = $info->name;
        }
    }

    $res[0] = array("OK", $rowCount, $fieldCount, $pNum);
    $res[1] = $names;
    $i = 0;
    while     ($line     =     mysqli_fetch_array($result,
MYSQLI_NUM)) {
        $data[$i++] = $line;
    }
    mysqli_free_result($result);
    end:
    if ($link) {
        mysqli_close($link);
    }
    return  json_encode([$res, $data]);
}
```

```
function MyExecNonQuery($connectionInfo, $Query)
/*
        * Author: Vl.Shabashov
*/
{
    $host = $connectionInfo["host"];
    if (isset($connectionInfo["port"]) && $connectionIn-
fo["port"] != "") {
        $host .= ":" . $connectionInfo["port"];
    }
    $dbName = $connectionInfo["dbName"];
    $userName = $connectionInfo["userName"];
    $password = $connectionInfo["password"];
    $q = trim($Query["sqlQuery"]);
    if (substr($q, strlen($q) - 1) != ';') {
        $q .= ';';
    }
    $res = array();
    $link = mysqli_connect($host, $userName, $password,
$dbName);
    if (mysqli_connect_errno($link)) {
        $res[0] = "Error";
        $res[1] = "Host: " . $host;
        $res[2] = "Login: " . $userName;
        $res[3] = "Password: " . $password;
        $res[4] = "DbName: " . $dbName;
        $res[5] = "Соединение не установлено: " .
mysqli_connect_error();
        goto end;
    }
    mysqli_set_charset($link, 'utf8'); // utf8 обязатель-
на для json_encode
    $qCount = count(explode(';', $q)) - 1; // количество
запросов в одной строке

    $result = mysqli_multi_query($link, $q);
    if (!$result) {
        $res[0] = "Error";
        $res[1] = "Запрос не выполнен :<br>" .
```

```
mysqli_error($link);
        $res[2] = $q;
        mysqli_free_result($result);
        goto end;
    }

    $res[0] = "OK";
    $res[1] = mysqli_affected_rows($link);
    if ($qCount > $res[1]) {
        $res[1] = $qCount;  // В строке несколько запросов
    }
    $res[2]  =  mysqli_insert_id($link);  //Если  нет  поля
identity, то будет 0
    mysqli_free_result($result);
    end:
        if ($link) {
            mysqli_close($link);
        }
    return json_encode($res);
}
```

ПРИЛОЖЕНИЕ 2
"ЛИСТИНГ ФУНКЦИЙ
ДЛЯ MICROSOFT SQL SERVER "

```
function typeOfColumn($it)

/*
        * Author: Vl.Shabashov
*/{
    if ($it === -5 || $it === -7 || $it === 4 || $it === 5
|| $it === -6)
        return 'int';
    if ($it === 91)
        return 'date';
    if ($it === 1 || $it === 12 || $it === -9 || $it === -
1 || $it === -10)
        return 'string';
    if ($it === 3 || $it === 6 || $it === 2 || $it === 7)
        return 'number';
    return 'other';
}

function parenthesisLevel($s, $fromPos)

/*
        * Author: Vl.Shabashov
*/{
    $level = 0;
    for ($i = 6; $i < $fromPos; $i++)
    {
        if (substr($s, $i, 1) == '(') {
                $level++;
        }
        if (substr($s, $i, 1) == ')') {
                $level--;
        }
    }
    return $level;
}
```

```
function inQuotation($s, $fromPos)

/*
        * Author: Vl.Shabashov
*/{
    $inQuot = 0;
    for ($i = 6; $i < $fromPos; $i++)
    {
        if (substr($s, $i, 1) == "'") {
                $inQuot = $inQuot ^ 1;
        }
    }
    return $inQuot;
}


function fromPosition($s, $from)
/*
        * Author: Vl.Shabashov
*/
{
    $ar = str_word_count($s, 2, '_');
    foreach ($ar as $key => $value)
    {
        if (strtolower($value) != $from) {
            continue;
        }
        if (parenthesisLevel($s, $key) == 0 && inQuota-
tion($s, $key) == 0){
            return $key;
        }
    }
    return FALSE;
}

function MsExecQuery($connectionInfo, $Query)
/*
        * Author: Vl.Shabashov
*/
{
    $host = $connectionInfo["host"];
    if (isset($connectionInfo["port"]) && $connectionIn-
```

```
fo["port"] != "") {
        $host .= ", " . $connectionInfo["port"];
    }
    $dbName = $connectionInfo["dbName"];
    $userName = $connectionInfo["userName"];
    $password = $connectionInfo["password"];

    $query = $Query["sqlQuery"];
    $pSize = $Query["pageSize"];
    $pNum = $Query["pageNumber"];

    $res = array();
    $data = array();
    $link = sqlsrv_connect($host, ['UID'=>$userName,

'PWD'=>$password,
                                                'Database'   =>
$dbName,
                                                'CharacterSet'
=> 'UTF-8']);
    if (!$link)                          // UTF-8 обязательна
для json_encode
    {
        $res[0] = "Error";
        $res[1] = "Host: " . $host;
        $res[2] = "Login: " . $userName ;
        $res[3] = "Password: " . $password;
        $res[4] = "DbName: ". $dbName;
        $res[5] = 'Соединение не установлено: \n'
                        .
sqlsrv_errors(SQLSRV_ERR_ERRORS)['message'];
        goto end;
    }

    $sql = $query;
    $order = fromPosition($sql, 'order');
    if ($order > 0) {
        $sql = substr ($sql, 0, $order);
    }
    $start = stripos($sql, 'select') + 6;
    $end = fromPosition($sql, 'from');
    if (!$end)
```

```
    {
        $res[0] = "Error";
        $res[1] = "Запрос не может быть выполнен";
        $res[2] = "SQL: " . $sql;
        goto end;
    }
    $s = substr($sql, 0, $start) . ' count(*) ' . sub-
str($sql, $end);
    $result = sqlsrv_query($link, $s);
    if (!$result) {
        $res[0] = "Error";
        $res[1]   =   "Запрос   не   выполнен   :<br>"   .
sqlsrv_errors(SQLSRV_ERR_ERRORS)['message'];
        $res[2] = "Исходный SQL: " . $sql;
        $res[3] = "Измененный SQL: " . $s;
        goto end;
    }
    $line           =           sqlsrv_fetch_array($result,
SQLSRV_FETCH_NUMERIC);
    $rowCount = $line[0];
    sqlsrv_free_stmt($result);

    if ($pSize > 0)
    {  // работа со страницами
        $min = $pSize * ($pNum - 1);
        if ($rowCount > 0) {
            while ($min >= $rowCount)
            {
                $min -= $pSize;
                $pNum--;
            }
            if ($min <= 0) {
                $min = 0;
                $pNum = 1;
            }
        }
        else
        {
            $min = 0;
            $pNum = 1;
        }
        $ind = strrpos($query, ';');//OFFSET 0 ROWS FETCH
```

```
NEXT 10 ROWS ONLY
        if ($ind !== FALSE) {
            $query = substr ($query, 0, $ind);
        }
        if (fromPosition($query, 'order') > 0) {
            $query = $query . ' OFFSET ' . $min . ' ROWS
FETCH NEXT '
                                        .  $pSize  .  '  ROWS
ONLY;';
        }
        else {
            $query = $query . ' ORDER BY 1 OFFSET ' . $min
                    . ' ROWS FETCH NEXT ' . $pSize . '
ROWS ONLY;';
        }
     }   // конец работы со страницами

    $result = sqlsrv_query($link, $query);
    if (!$result) {
        $res[0] = "Error";
        $res[1] = "Запрос не выполнен :<br>"
                  .
sqlsrv_errors(SQLSRV_ERR_ERRORS)['message'];
        $res[2] = $query;
        goto end;
    }

    $fieldCount = sqlsrv_num_fields($result);
    //   Получение мета информации
    $names = array();
    $info = array();
    $infoAr = sqlsrv_field_metadata($result);
    for ($i = 0; $i < $fieldCount; $i++)
    {
        $names[$i] = $infoAr[$i]['Name'];
        $types[$i] = typeOfColumn($infoAr[$i]['Type']);
    }
    $res[0] = array("OK", $rowCount, $fieldCount, $pNum);
    $res[1] = $names;

    $i = 0;
    while     ($line     =     sqlsrv_fetch_array($result,
```

```
SQLSRV_FETCH_NUMERIC)) {
        $t = $line;
        for ($k = 0; $k < $fieldCount; $k++)
        {
            if ($types[$k] === 'date' && $t[$k] !== NULL)
                $t[$k] = date_format($t[$k], 'Y-m-d');
        }
        $data[$i++] = $t;
    }
    sqlsrv_free_stmt($result);
    end:
    if ($link) {
        sqlsrv_close($link);
    }
    return  json_encode([$res, $data]);
}



function MsExecNonQuery($connectionInfo, $Query)
/*
       * Author: Vl.Shabashov
*/
{
    $host = $connectionInfo["host"];
    if  (isset($connectionInfo["port"])  &&  $connectionIn-
fo["port"] != "") {
        $host .= ", " . $connectionInfo["port"];
    }
    $dbName = $connectionInfo["dbName"];
    $userName = $connectionInfo["userName"];
    $password = $connectionInfo["password"];

    $q = trim($Query["sqlQuery"]);
    if (substr($q, strlen($q) - 1) != ';') {
            $q .= ';';
    }
    if (isset($Query["newId"])) {
        $newId = $Query["newId"];
    }
    else {
        $newId = FALSE;
```

```
    }
    if ($newId) {
        $q .= 'SELECT @@Identity;';
    }
    $res = array();
    $link = sqlsrv_connect($host, ['UID'=>$userName,

'PWD'=>$password,
                                                'Database'   =>
$dbName,
                                                'CharacterSet'
=> 'UTF-8']);
    if (!$link) {                           // UTF-8 обяза-
тельна для json_encode
        $res[0] = "Error";
        $res[1] = "Host: " . $host;
        $res[2] = "Login: " . $userName ;
        $res[3] = "Password: " . $password;
        $res[4] = "DbName: ". $dbName;
        $res[5] = 'Соединение не установлено: \n' .

sqlsrv_errors(SQLSRV_ERR_ERRORS)['message'];
        goto end;
    }
    $result = sqlsrv_query($link, $q);
    if (!$result) {
        $res[0] = "Error";
        $res[1] = "Запрос не выполнен :<br>" .

sqlsrv_errors(SQLSRV_ERR_ERRORS)['message'];
        $res[2] = $q;
        sqlsrv_free_stmt($result);
        goto end;
    }
    $res[0] = "OK";
    $res[1] = sqlsrv_rows_affected($result);
    if  ($newId  &&  (strtolower(substr($q,  0,  6))  ==
'insert')) {
        sqlsrv_next_result($result);
        $res[2]     =     (int)sqlsrv_fetch_array($result,
SQLSRV_FETCH_NUMERIC)[0];
    }
```

```
     else {
        while(sqlsrv_next_result($result)) {
          $res[1] += sqlsrv_rows_affected($result);
        }
    }
    sqlsrv_free_stmt($result);
    end:
    if ($link) {
        sqlsrv_close($link);
    }
    return json_encode($res);
}
```

ПРИЛОЖЕНИЕ 3
"ЛИСТИНГ ФУНКЦИЙ ДЛЯ FIREBIRD"

```
function parenthesisLevel($s, $fromPos)
/*
        * Author: Vl.Shabashov
*/
{
    $level = 0;
    for ($i = 6; $i < $fromPos; $i++)
    {
        if (substr($s, $i, 1) == '(') {
                $level++;
        }
        if (substr($s, $i, 1) == ')') {
                $level--;
        }
    }
    return $level;
}

function inQuotation($s, $fromPos)
/*
        * Author: Vl.Shabashov
*/
{
    $inQuot = 0;
    for ($i = 6; $i < $fromPos; $i++)
    {
        if (substr($s, $i, 1) == "'") {
            $inQuot = $inQuot ^ 1;
        }
    }
    return $inQuot;
}

function fromPosition($s, $from)
/*
        * Author: Vl.Shabashov
*/
{
```

```
    $ar = str_word_count($s, 2, '_');
    foreach ($ar as $key => $value)
    {
        if (strtolower($value) != $from) {
            continue;
        }
        if (parenthesisLevel($s, $key) == 0 && inQuota-
tion($s, $key) == 0) {
            return $key;
        }
    }
    return FALSE;
}

function FbExecQuery($connectionInfo, $Query)
/*
       * Author: Vl.Shabashov
*/
{
    $host = $connectionInfo["host"];
    if (isset($connectionInfo["port"]) && $connectionIn-
fo["port"] != "") {
        $host .= '/' . $connectionInfo["port"];
    }
    $dbName = $connectionInfo["dbName"];
    $userName = $connectionInfo["userName"];
    $password = $connectionInfo["password"];
    $query = $Query["sqlQuery"];
    $pSize = $Query["pageSize"];
    $pNum = $Query["pageNumber"];

    $res = array();
    $data = array();
    $link = ibase_connect($host . ":" . $dbName,
$userName, $password, 'UTF8');
    if (!$link) {
// UTF8 обязательна для json_encode
        $res[0] = "Error";
        $res[1] = "Host: " . $host;
        $res[2] = "Login: " . $userName;
        $res[3] = "Password: " . $password;
        $res[4] = "DbName: " . $dbName;
```

```
        $res[5] = "Соединение не установлено: " .
ibase_errmsg();
        goto end;
    }

    $sql = $query;
    $sql1 = strtolower($sql);
    $start = stripos($sql1, 'order by');
    $s = $sql;
    if ($start !== FALSE) {
        $s = substr($sql, 0, $start);
    }
    $start = stripos($sql1, 'select') + 6;
    $end = fromPosition($sql1, 'from');
    if (!$end) {
        $res[0] = "Error";
        $res[1] = "Запрос не может быть выполнен";
        $res[2] = "SQL: " . $sql;
        goto end;
    }
    $s1 = substr($s, 0, $start) . ' count(*) ' . sub-
str($s, $end);
    $r = ibase_query($link, $s1);
    if (!$r) {
        $res[0] = "Error";
        $res[1] = "Запрос не выполнен :<br>" .
ibase_errmsg();
        $res[2] = $sql;
        $res[3] = $s;
        goto end;
    }
    $line = ibase_fetch_row($r);
    $rowCount = $line[0];
    ibase_free_result($r);

    if ($pSize > 0)
    {  // работа со страницами
        $min = $pSize * ($pNum - 1);
        if ($rowCount > 0)
        {
            while ($min >= $rowCount) {
                $min -= $pSize;
```

```
                $pNum--;
            }
            if ($min <= 0) {
                $min = 0;
                $pNum = 1;
            }
        }
        else {
            $min = 0;
            $pNum = 1;
        }

        $query = substr($query, 0, $start) . ' FIRST ' .
$pSize . ' SKIP ' . $min . substr($query, $start);
    }   // конец работы со страницами

    $result = ibase_query($link, $query);
    if (!$result) {
        $res[0] = "Error";
        $res[1]   =   "Запрос   не   выполнен   :<br>"   .
ibase_errmsg();
        $res[2] = $query;
        goto end;
    }

    $fieldCount = ibase_num_fields($result);
    //   Получение мета информации
    $names = array();
    for ($i = 0; $i < $fieldCount; $i++)
    {
        $info = ibase_field_info($result, $i);
        $names[$i] = $info['name'];
    }

    $res[0] = array("OK", $rowCount, $fieldCount, $pNum);
    $res[1] = $names;
    $i = 0;
    while ($line = ibase_fetch_row($result)) {
        $data[$i++] = $line;
    }

    ibase_free_result($result);
```

```
    end:
    if ($link) {
        ibase_close($link);
    }
    return  json_encode([$res, $data]);
}



function tableName($q)
/*
       * Author: Vl.Shabashov
*/
{
    $ar = explode(' ', $q);
    for ($i = 1; $i < count($ar); $i++)
    {
        if (strtolower($ar[$i]) != 'into') {
            continue;
        }
        for ($j = $i + 1; $j < count($ar); $j++)
        {
            if ($ar[$j] == null) {
                continue;
            }
            return $ar[$j];
        }
    }
    return '';
}

function getNewId($link, &$q)
/*
       * Author: Vl.Shabashov
*/
{
    $tName   =   strtoupper(tableName($q));   //   Получение
названия таблицы
    $q1 = 'SELECT "RDB$FIELD_NAME", "RDB$GENERATOR_NAME" '
.
          'FROM "RDB$RELATION_FIELDS" ' .
         ' WHERE   NOT  "RDB$GENERATOR_NAME"  IS  NULL  AND
```

```
"RDB$RELATION_NAME" = ';
    $q1 = $q1 . "'" . $tName . "';";
    $r = ibase_query($link, $q1);
    if (!$r) {
        $res[0] = "Error";
        $res[1]   =   "Запрос   не   выполнен   :<br>"   .
ibase_errmsg();
        $res[2] = $q1;
        ibase_free_result($r);
        return FALSE;
    }
    $line = ibase_fetch_row($r); // имя поля и имя генера-
тора
    $fName = $line[0]; // определение имени генератора
    $id = ibase_gen_id(trim($line[1]), 1, $link); // зна-
чение идентификатор для новой строки
    $sk1 = strpos($q, '(');
    $sk2 = strpos($q, '(', $sk1 + 1);
    $q2 = substr($q, 0, $sk1 + 1) . $fName . ',' .
                    substr($q,  $sk1  +  1,  $sk2  -  $sk1)  .
$id . ',' .
                    substr($q, $sk2 + 1);  // преобразова-
ни запроса
    $q = $q2;
    ibase_free_result($r);
    return $id;
}



function FbExecNonQuery($connectionInfo, $Query)
/*
       * Author: Vl.Shabashov
*/
{
    $host = $connectionInfo["host"];
    if  (isset($connectionInfo["port"])  &&  $connectionIn-
fo["port"] != "") {
        $host .= '/' . $connectionInfo["port"];
    }
    $dbName = $connectionInfo["dbName"];
    $userName = $connectionInfo["userName"];
```

```
    $password = $connectionInfo["password"];

    $q = trim($Query["sqlQuery"]);
    if (substr($q, strlen($q) - 1) != ';') {
            $q .= ';';
    }
    if (isset($Query["newId"])) {
        $newId = $Query["newId"];
    }
    else {
        $newId = FALSE;
    }
    $res = array();
    $link = ibase_connect($host . ":" . $dbName,
$userName, $password, 'UTF8');
    if (!$link) {                          // UTF8 обязатель-
на для json_encode
        $res[0] = "Error";
        $res[1] = "Host: " . $host;
        $res[2] = "Login: " . $userName;
        $res[3] = "Password: " . $password;
        $res[4] = "DbName: " . $dbName;
        $res[5] = "Соединение не установлено: " .
ibase_errmsg();
        goto end;
    }

    $id = 0;
    if ($newId && (strtolower(substr($q, 0, 6)) ==
'insert')) {
        $id = getNewId($link, $q);
        if ($id == FALSE) {
            goto end;
        }
    }

    $start = 0;
    $rAf = 0;
    while ($start >= 0) // Если строка содержит несколько
операторов
    {
        $stNew = stripos($q, ';UPDATE', $start);
```

```
        if ($stNew == FALSE) {
            $qu = substr($q, $start);
            $start = -1;
        }
        else {
            $qu = substr($q, $start, $stNew - $start + 1);
            $start = $stNew + 1;
        }
        $result = ibase_query($link, $qu);
        if (!$result) {
            $res[0] = "Error";
            $res[1]  =  "Запрос  не  выполнен  :<br>"  .
ibase_errmsg();
            $res[2] = $qu;
            ibase_free_result($r);
            goto end;
        }
        $rAf += ibase_affected_rows($link);
        ibase_free_result($r);
    }
    $res[0] = "OK";
    $res[1] = $rAf;
    $res[2] = $id;
    end:
    if ($link) {
        ibase_close($link);
    }
    return  json_encode($res);
}
```

ПРИЛОЖЕНИЕ 4
"ЛИСТИНГ ФУНКЦИЙ ДЛЯ POSTGRESQL"

```
function parenthesisLevel($s, $fromPos)
/*
        * Author: Vl.Shabashov
*/
{
    $level = 0;
    for ($i = 6; $i < $fromPos; $i++)
    {
        if (substr($s, $i, 1) == '(')
                $level++;
        if (substr($s, $i, 1) == ')')
                $level--;
    }
    return $level;
}

function inQuotation($s, $fromPos)
/*
        * Author: Vl.Shabashov
*/
{
    $inQuot = 0;
    for ($i = 6; $i < $fromPos; $i++)
    {
        if (substr($s, $i, 1) == "'")
                $inQuot = $inQuot ^ 1;
    }
    return $inQuot;
}

function fromPosition($s, $from)
/*
        * Author: Vl.Shabashov
*/
{
    $ar = str_word_count($s, 2, '_');
    foreach ($ar as $key => $value)
    {
```

```
        if (strtolower($value) != $from)
            continue;
        if (parenthesisLevel($s, $key) == 0 && inQuotation($s, $key) == 0)
            return $key;
    }
    return FALSE;
}



function PgExecQuery($connectionInfo, $Query)
/*
       * Author: Vl.Shabashov
*/
{
    $host = $connectionInfo["host"];
    if (isset($connectionInfo["port"]) && $connectionInfo["port"] != "") {
        $port = ' port=' . $connectionInfo["port"];
    }
    else {
        $port = '';
    }
    $userName = $connectionInfo["userName"];
    $password = $connectionInfo["password"];
    $query = $Query["sqlQuery"];
    $pSize = $Query["pageSize"];
    $pNum = $Query["pageNumber"];

    $res = array();
    $data = array();
    $link = pg_connect("host=" . $host . " dbname=" . $dbName
            . ' client_encoding=UTF8'  // UTF8 обязательна для json_encode
            . $port . " user=" . $userName . " password=" . $password);
    if (!$link) {
        $res[0] = "Error";
        $res[1] = "Host: " . $host;
```

```
        $res[2] = "Port: " . $port;
        $res[3] = "Login: " . $userName ;
        $res[4] = "Password: " . $password;
        $res[5] = "DbName: ". $dbName;
        $res[6] = "Связь с базой данных не установлена " .
pg_last_error();
        goto end;
    }

    if ($pSize > 0)  // работа со страницами  ///////
    {
        $sql = $query;
        $start = fromPosition($sql, 'order');
        $s = $sql;
        if ($start !== FALSE) {
            $s = substr($sql, 0, $start);
        }
        $start = strpos($sql, 'select') + 6;
        $end = strpos($sql, 'from');
        $s1 = substr($s, 0, $start) . ' count(*) ' . sub-
str($s, $end);

        $r1 = pg_query($link, $s1);
        if (!$r1) {
            $res[0] = "Error";
            $res[1] = "Запрос по подсчету строк не выпол-
нен :<br>"
                                        .
pg_result_error($link);
            $res[2] = "Подсчет строк: " . $s1;
            $res[3] = "Исходный запрос: " . $query;
            goto end;
        }
        $line = pg_fetch_array($r1, 0, PGSQL_NUM);
        $rowCount = (int)$line[0]; // $line[0] возвращает
строковое значение
        pg_free_result($r1);

        $min = $pSize * ($pNum - 1);
        if ($rowCount > 0)
        {
            while ($min >= $rowCount) {
```

```
                $min -= $pSize;
                $pNum--;
            }
            if ($min <= 0) {
                $min = 0;
                $pNum = 1;
            }
        }
        else {
            $min = 0;
            $pNum = 1;
        }
        $ind = strpos($query, ';');
        if ($ind !== FALSE)
            $query = substr ($query, 0, $ind);
        $query = $query . ' limit ' . $pSize . ' offset '
. $min . ';';
     }   // конец работы со страницами  ///////

    $result = pg_query($link, $query);
    if (!$result)
    {
        $res[0] = "Error";
        $res[1]   =   "Запрос   не   выполнен   :<br>"   .
pg_result_error($link);
        $res[2] = $query;
        goto end;
    }

    $names = array();
    $i = 0;
    while    ($row    =    pg_fetch_array($result,    null,
PGSQL_NUM)) {
        $data[$i++] = $row;
    }
    if ($pSize == 0) {  // работа без страниц
        $rowCount = pg_num_rows($result);
    }
    $fieldCount = pg_num_fields($result);
    for ($i = 0; $i < $fieldCount; $i++)
    {
        $names[$i] = pg_field_name($result, $i);
```

```
    }

    $res[0] = array("OK", $rowCount, $fieldCount, $pNum);
    $res[1] = $names;
    pg_free_result($result);
    end:
    if ($link) {
        pg_close ($link);
    }
    return json_encode([$res, $data]);
}



function PgExecNonQuery($connectionInfo, $Query)
/*
        * Author: Vl.Shabashov
*/
{
    $host = $connectionInfo["host"];
    if  (isset($connectionInfo["port"])  &&  $connectionIn-
fo["port"] != "") {
        $port = ' port=' . $connectionInfo["port"];
    }
    else {
        $port = '';
    }

    $dbName = $connectionInfo["dbName"];
    $userName = $connectionInfo["userName"];
    $password = $connectionInfo["password"];

    $query = $Query["sqlQuery"];
    $q = trim($Query["sqlQuery"]);
    if (substr($q, strlen($q) - 1) != ';') {
            $q .= ';';
    }
    if (isset($Query["newId"])) {
        $newId = $Query["newId"];
    }
    else {
        $newId = FALSE;
```

```
    }
    if ($newId && (strtolower(substr($q, 0, 6)) ==
'insert')) {
        $q .= 'SELECT LASTVAL();'; // извлекает последнее
id
    }
    $res = array();
    $link = pg_connect("host=" . $host . " dbname=" .
$dbName .
            ' client_encoding=UTF8' . $port . " user=" .
            $userName . " password=" . $password); // UTF8
обязательна для json_encode
    if (!$link) {
        $res[0] = "Error";
        $res[1] = "Host: " . $host . ' ' . $port;
        $res[2] = "Login: " . $userName ;
        $res[3] = "Password: " . $password;
        $res[4] = "DbName: ". $dbName;
        $res[5] = "Связь с базой данных не установлена " .
pg_last_error();
        goto end;
    }
    $qCount = 0;
    if ($newId && (strtolower(substr($q, 0, 6)) ==
'insert')) {
        $qCount = count(explode(';', $query)) - 1; // ко-
личество запросов в одной строке
    }
    $result = pg_query($link, $q);
    if (!$result) {
        $res[0] = "Error";
        $res[1] = "Запрос не выполнен :<br>" .
pg_last_error($link);
        $res[2] = $query;
        goto end;
    }

    $res[0] = "OK";
    $res[1] = pg_affected_rows($result);
    if ($qCount > $res[1]) {
        $res[1] = $qCount;  // В строке несколько запросов
    }
```

```
    $res[2] = pg_fetch_result($result, 0);
    end:
    if ($link) {
        pg_close ($link);
    }
    return json_encode($res);
}
```

Шабашов Владимир Яковлевич

Организация доступа к данным из PHP приложений для различных СУБД

Учебное пособие по дисциплине «Web-программирование»

Ответственный редактор *А. Иванова*
Верстальщик *Е. Семенова*

Издательство «Директ-Медиа»
117342, Москва, ул. Обручева, 34/63, стр. 1
Тел/факс + 7 (495) 334-72-11
E-mail: manager@directmedia.ru
www.biblioclub.ru

Отпечатано в ООО «ПАК ХАУС»
142172, г. Москва, г. Щербинка,
ул. Космонавтов, д.16