



PHP_{and} PostgreSQL

Programming by Example

Copyright

PHP and PostgreSQL Programming By Example

Agus Kurniawan

1st Edition, 2016

Copyright © 2016 Agus Kurniawan

Table of Contents

[Copyright](#)

[Preface](#)

[1. Setting up Development Environment](#)

[1.1 System Environment](#)

[1.1.1 IIS Web Server on Windows Platform](#)

[1.1.2 Apache on Ubuntu and Mac](#)

[1.2 Deploying PHP on Web Server](#)

[1.2.1 IIS for Windows Platform](#)

[1.2.2 Apache for Linux and Mac Platforms](#)

[1.3 PostgreSQL Server](#)

[1.4 PostgreSQL Driver for PHP](#)

[1.5 Development Tools](#)

[2. Hello World - PHP and PostgreSQL](#)

[2.1 Connecting to PostgreSQL](#)

[2.2 Creating PHP and Connecting to PostgreSQL](#)

[2.3 Testing](#)

[3. CRUD Operations](#)

[3.1 CRUD Operations](#)

[3.2 Creating Data](#)

[3.3 Reading Data](#)

[3.4 Update Data](#)

[3.5 Deleting Data](#)

[4. Working with Image and Blob Data](#)

[4.1 Image and Blob Data](#)

[4.2 Uploading Image](#)

[4.3 Listing Image Data](#)

[5. Transaction](#)

[5.1 PHP and PostgreSQL Transaction](#)

[5.2 Demo](#)

[6. Stored Procedures](#)

[6.1 Stored Procedures](#)

[6.2 Calling Stored Procedure](#)

[6.3 Calling Stored Procedure with Parameter](#)

[6.4 Calling Stored Procedure with Returning Values](#)

[6.5 Calling Stored Procedure with Input and Output Parameters](#)

[Source Code](#)

[Contact](#)

Preface

In general people use PHP and database MySQL. This book provides alternative approach to build PHP application with database PostgreSQL. It describes how to work with PHP and PostgreSQL and illustrates their use with code examples.

Agus Kurniawan

Berlin, March 2016

1. Setting up Development Environment

This chapter explains how to setup development to develop PHP and PostgreSQL.

1.1 System Environment

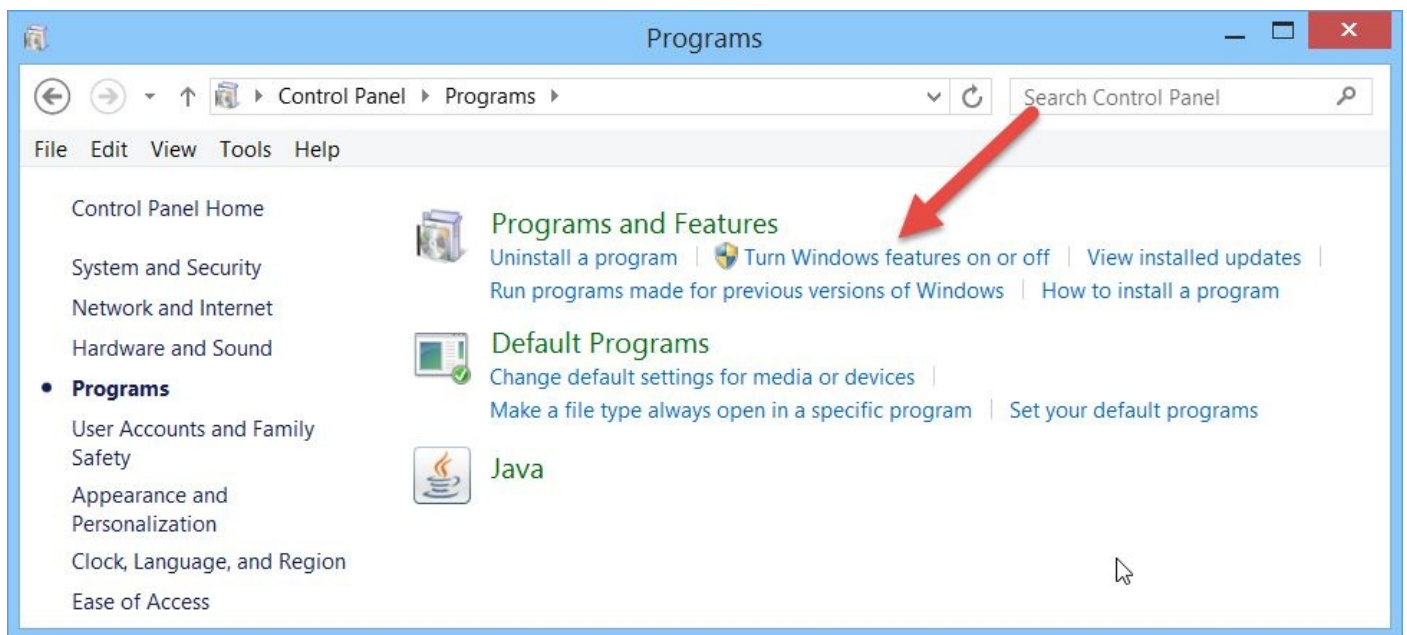
We need to setup development environment to develop PHP application with database PostgreSQL. The following is the list of development platform you can use:

- Linux
- Mac
- Windows

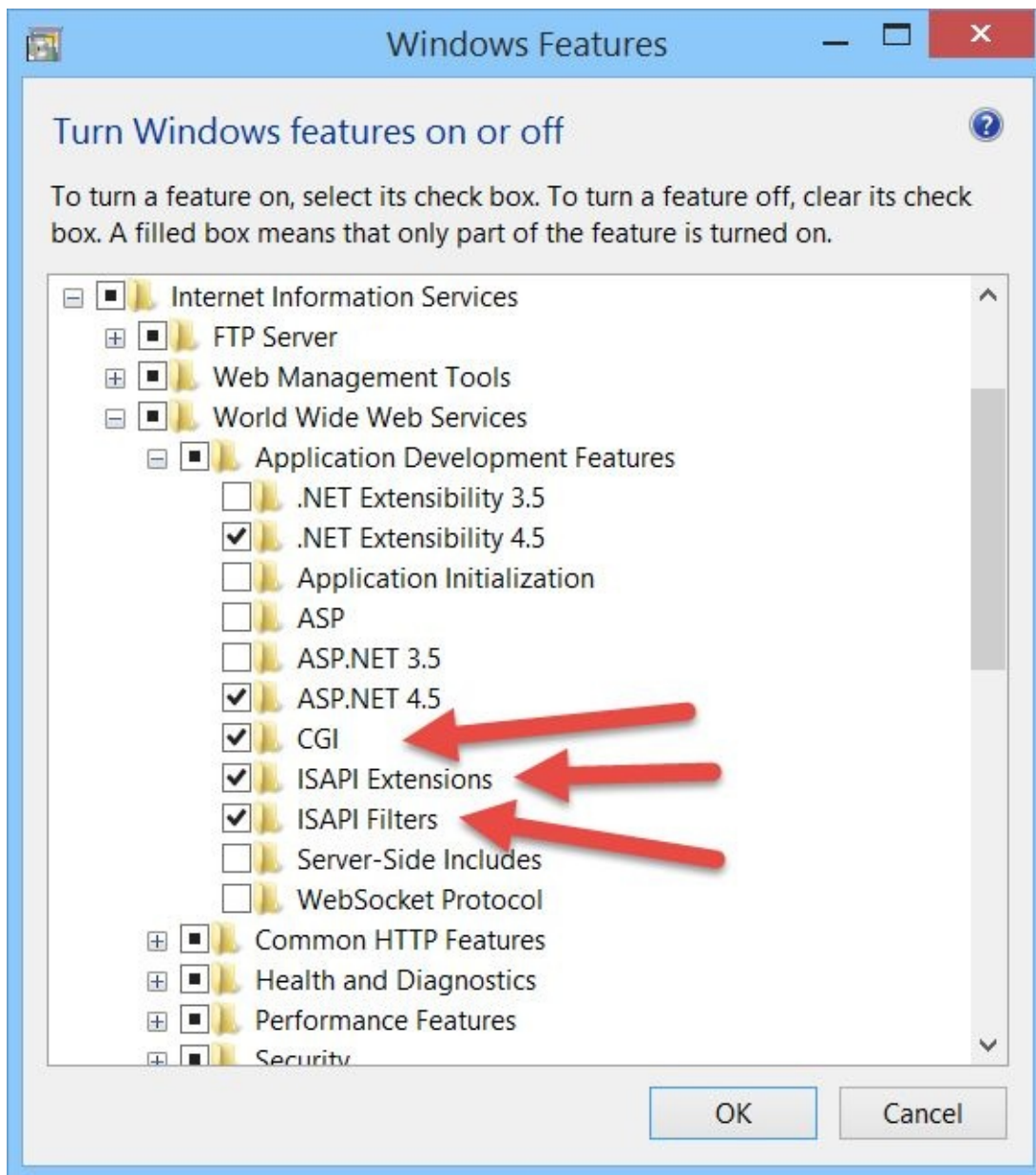
In this book, I use Ubuntu, OS X and Windows 10 for testing. I use Apache web server on Linux and Mac. Otherwise, I use IIS web server on Windows platform.

1.1.1 IIS Web Server on Windows Platform

You also must install IIS on your Windows platform. On Windows Client, you can find it on **Program and Features**. You can see it on Figure below. Then you can find **Turn Windows features on or off** (red arrow).

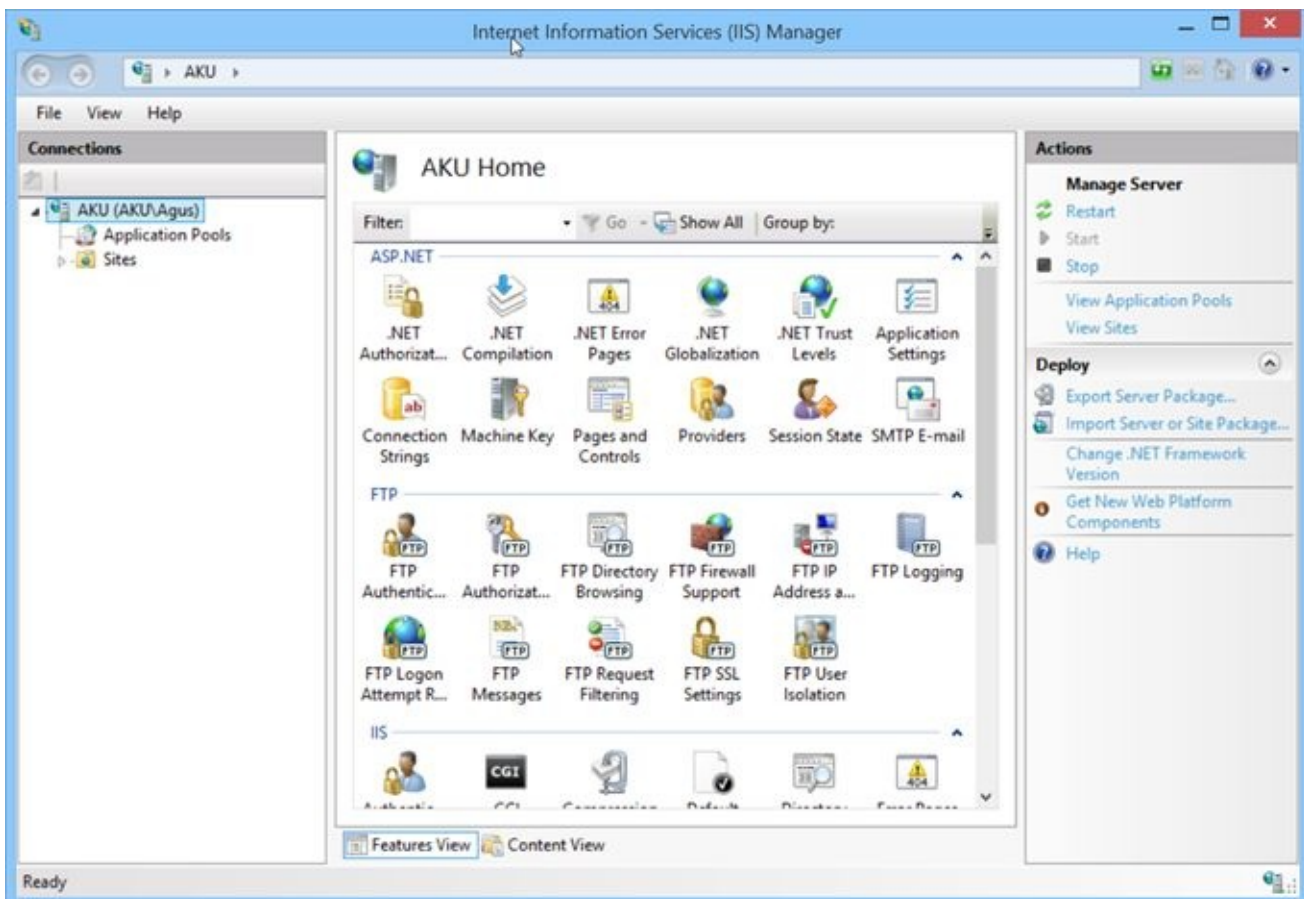


After you click **Turn Windows features on or off** you can obtain a dialog as below.

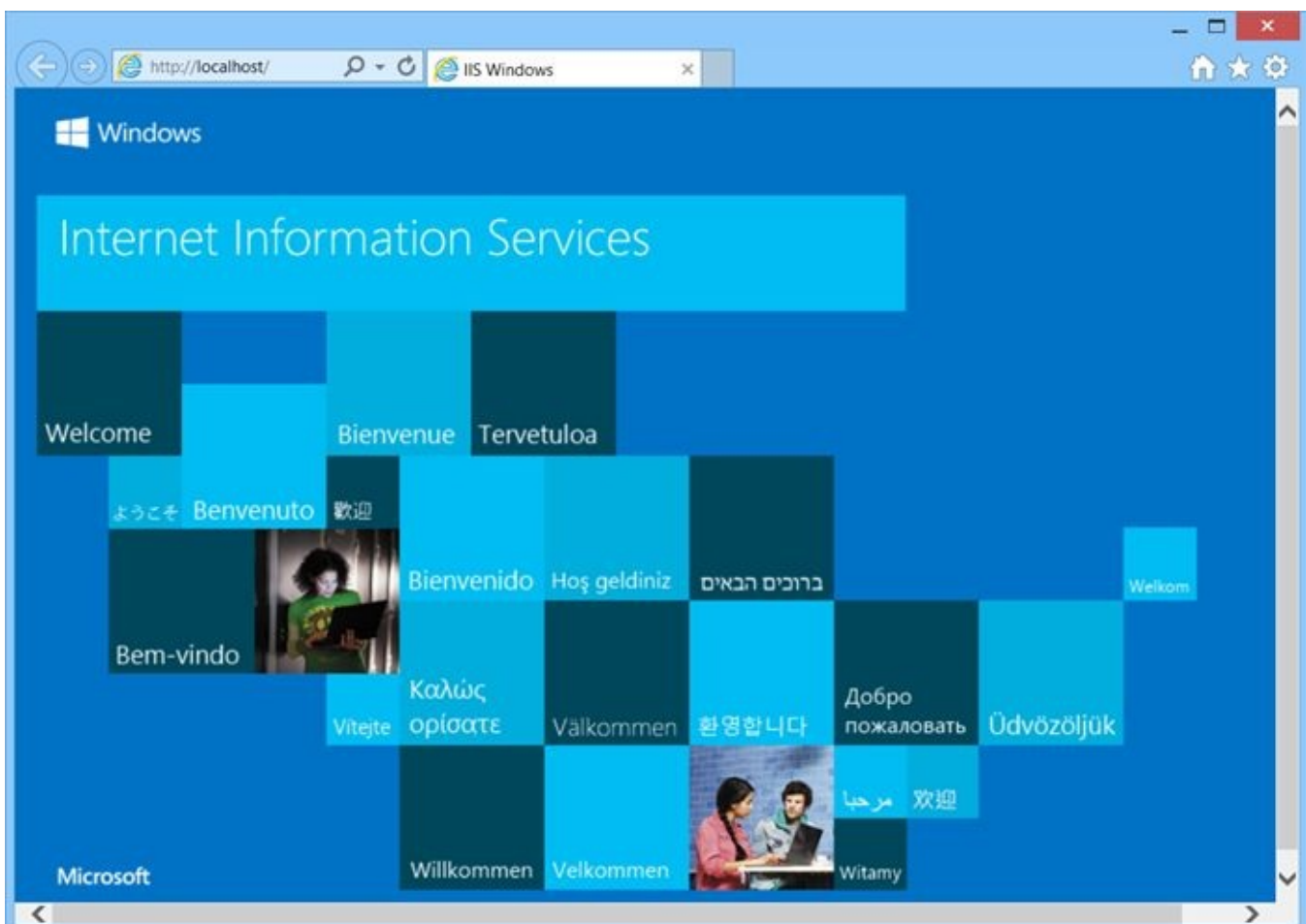


Checked **Internet Information Services** and **World Wide Web Services**. Don't forget to checked **CGI**, **ISAPI Extensions** and **ISAPI Filters**. You can checked other several IIS features. If finished, click **OK** button.

It will install all selected features. If done, you can check web server IIS on **Administrative Tools**. You can see **Internet Information Services (IIS) Manager**. Click it and then you can obtain web server IIS dialog, shown in Figure below.



You also can verify installed web server IIS by opening browser and then type <http://localhost> . If success, you can see the result as follows.



1.1.2 Apache on Ubuntu and Mac

On OS X , Apache already installed for you. On Ubuntu, you can install it using this command.

```
$ sudo apt-get update  
$ sudo apt-get install apache2
```

1.2 Deploying PHP on Web Server

In this section, we will deploy PHP on web servers. I use IIS and Apache web servers for testing.

Let's go.

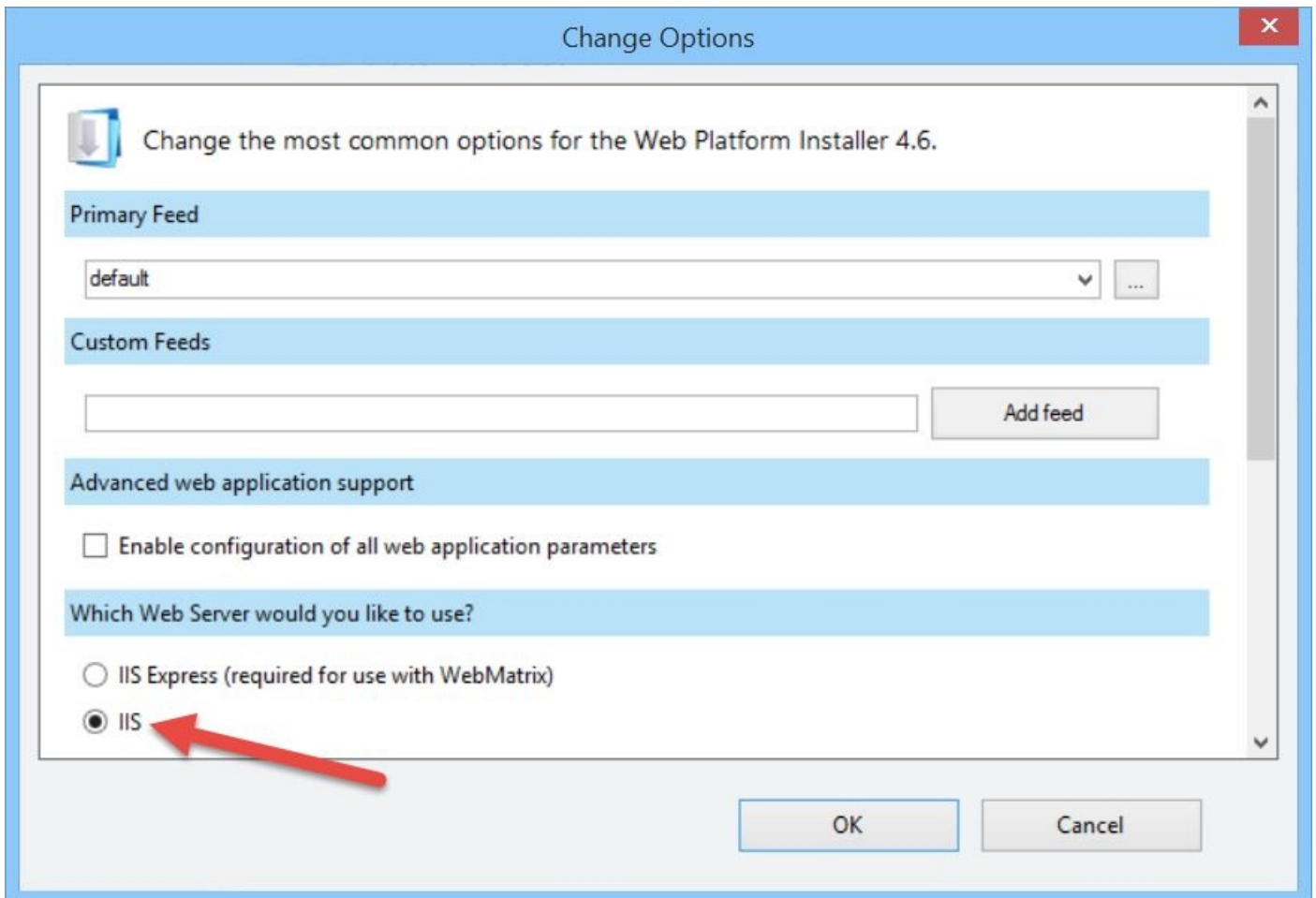
1.2.1 IIS for Windows Platform

There are two options to install PHP on IIS. You can install it via Microsoft Web Platform Installer, <http://php.iis.net/>. After downloaded, you can run it and then you obtain the following dialog.

** this dialog is a sample for Web Platform Installer 4.6.



Click Options and select IIS. If done, click **OK** button.

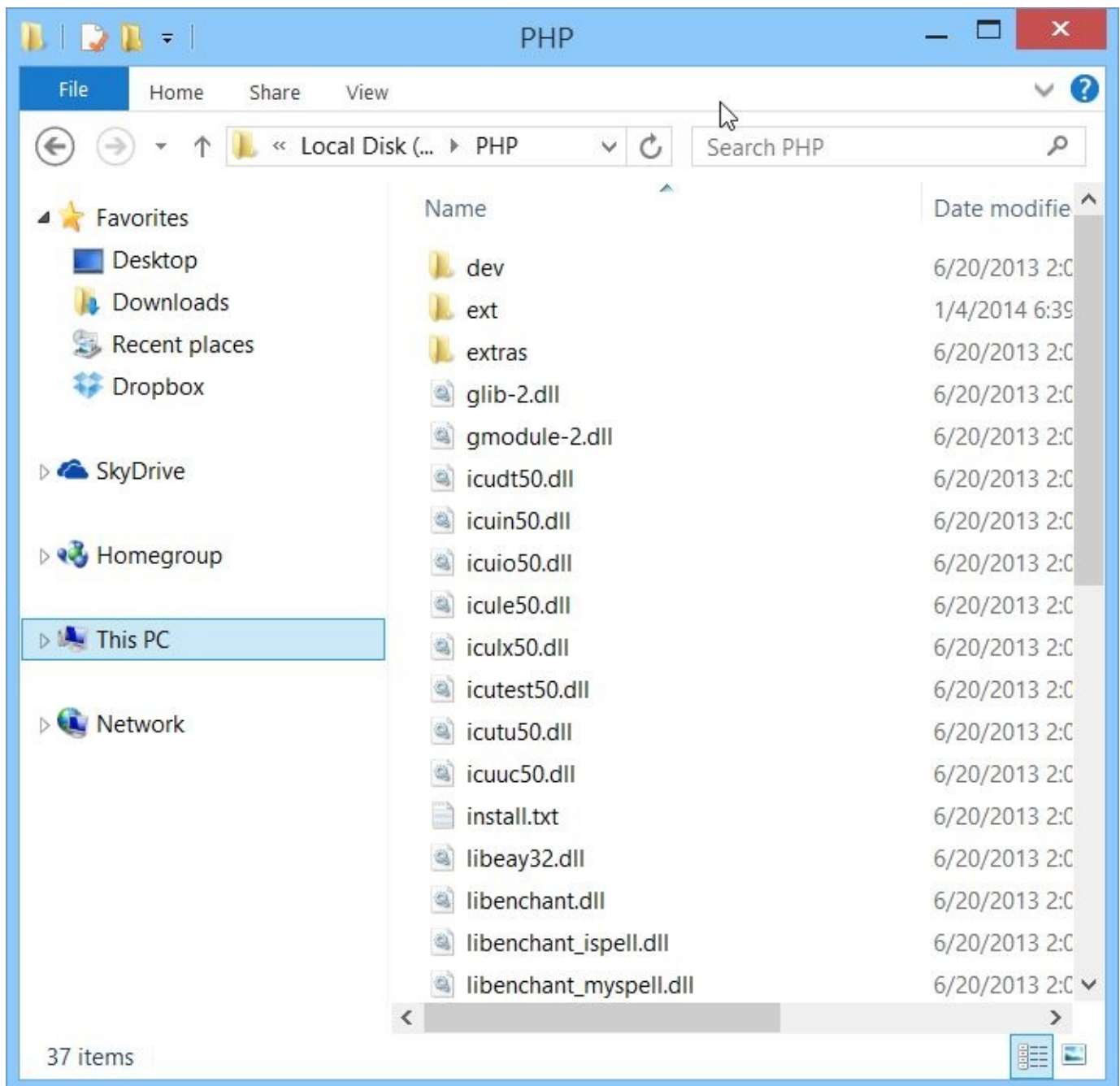


After that, click **Install** button to start installation.

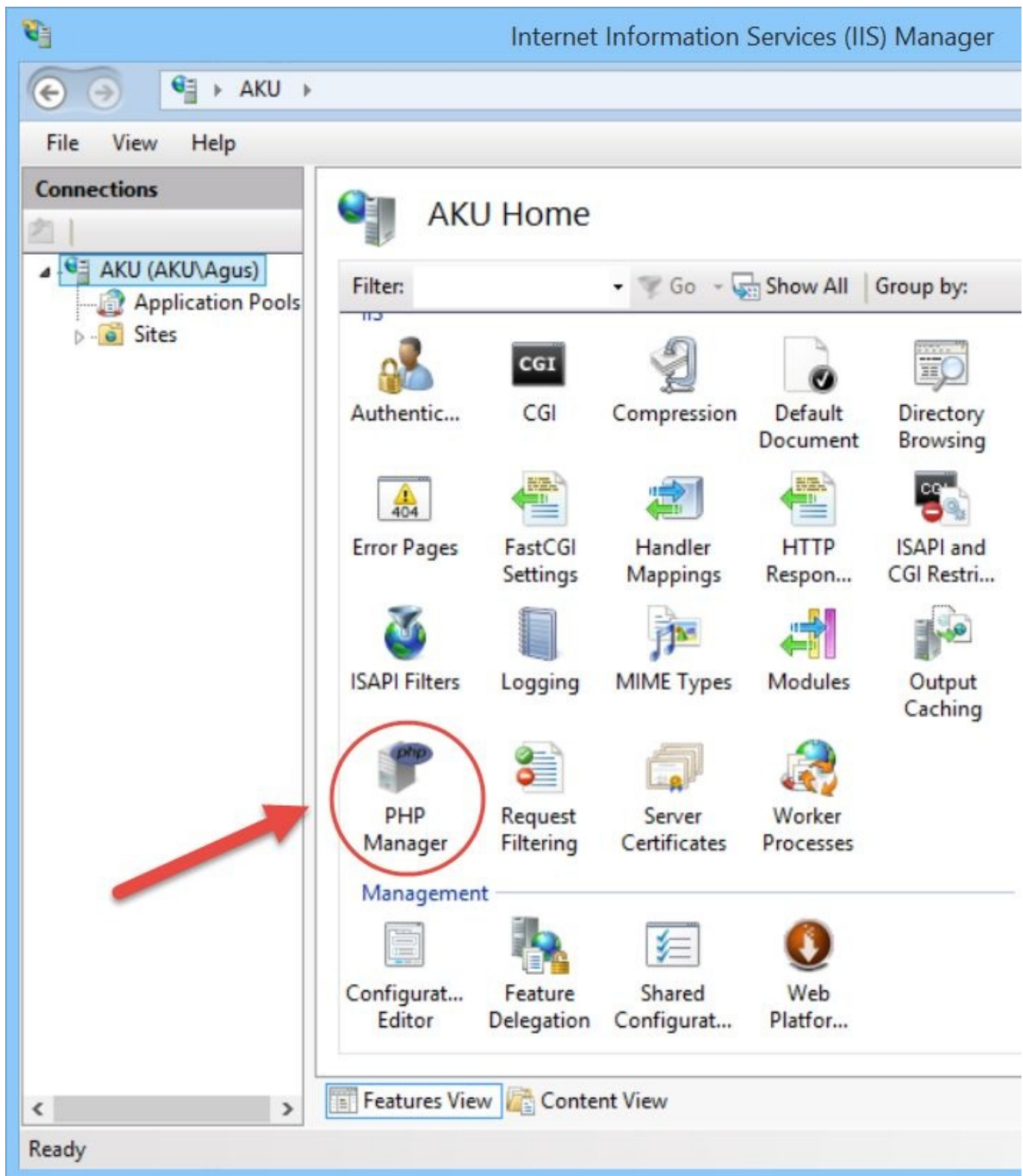
In this book, I installed PHP on IIS manually. You can install the latest PHP on web server IIS. How to install?

Firstly, we must install PHP for Windows. You can download it on <http://windows.php.net/download/> . You can download it for x86 or x64 and Thread-Safe or Non Thread-Safe. It's recommended to download Non Thread-Safe PHP. After downloaded, you obtain ZIP file (particular PHP version provides installer file). Extract it on the specific folder, for instance, C:\PHP.

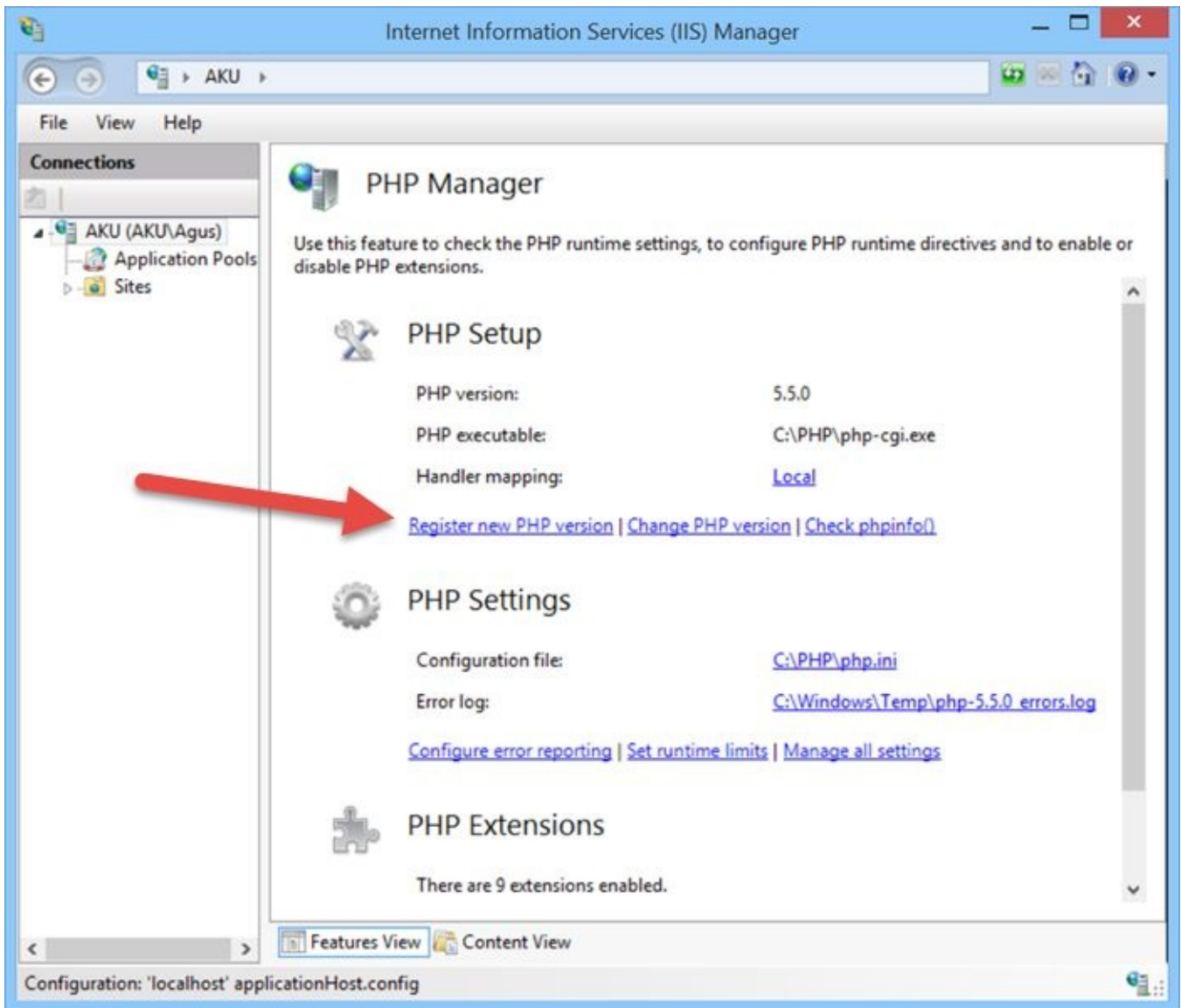
The following is a sample result of extracted PHP.



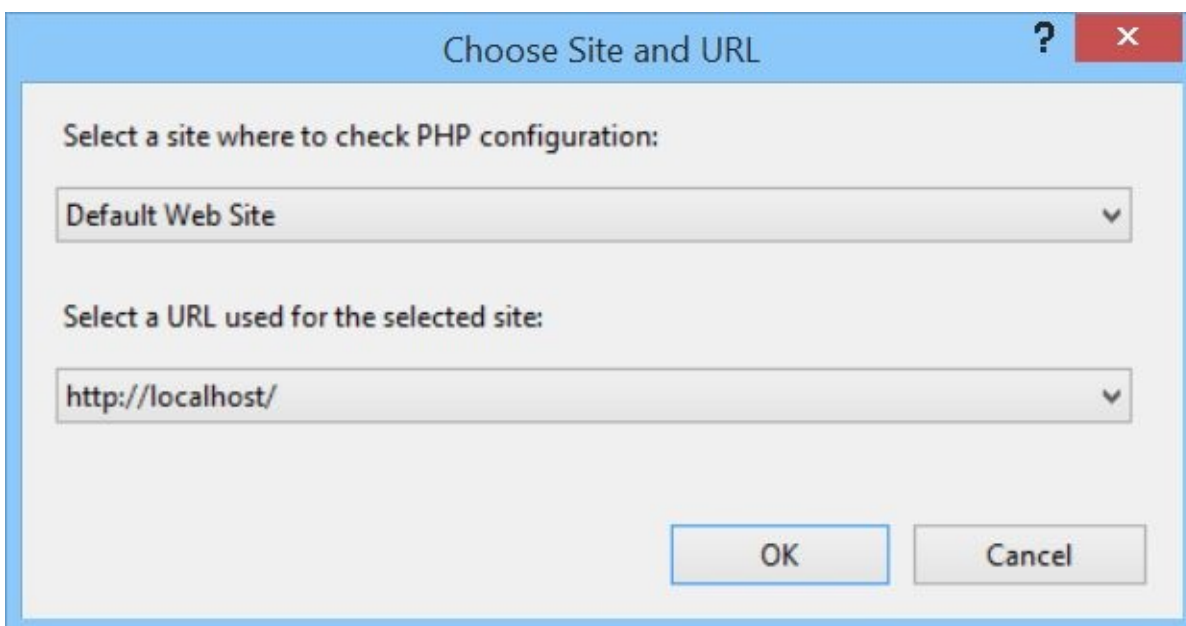
After that, we can install PHP Manager for IIS. You can download it on <https://phpmanager.codeplex.com/> . If you have install PHP Manager for IIS, you can see PHP Manager on IIS.



Double click PHP Manager. Then you can see PHP Manager view. Click **Register new PHP version**. Navigate to php-cgi.exe file location. If you have extracted PHP file on C:\PHP, you can find it on C:\PHP\php-cgi.exe .

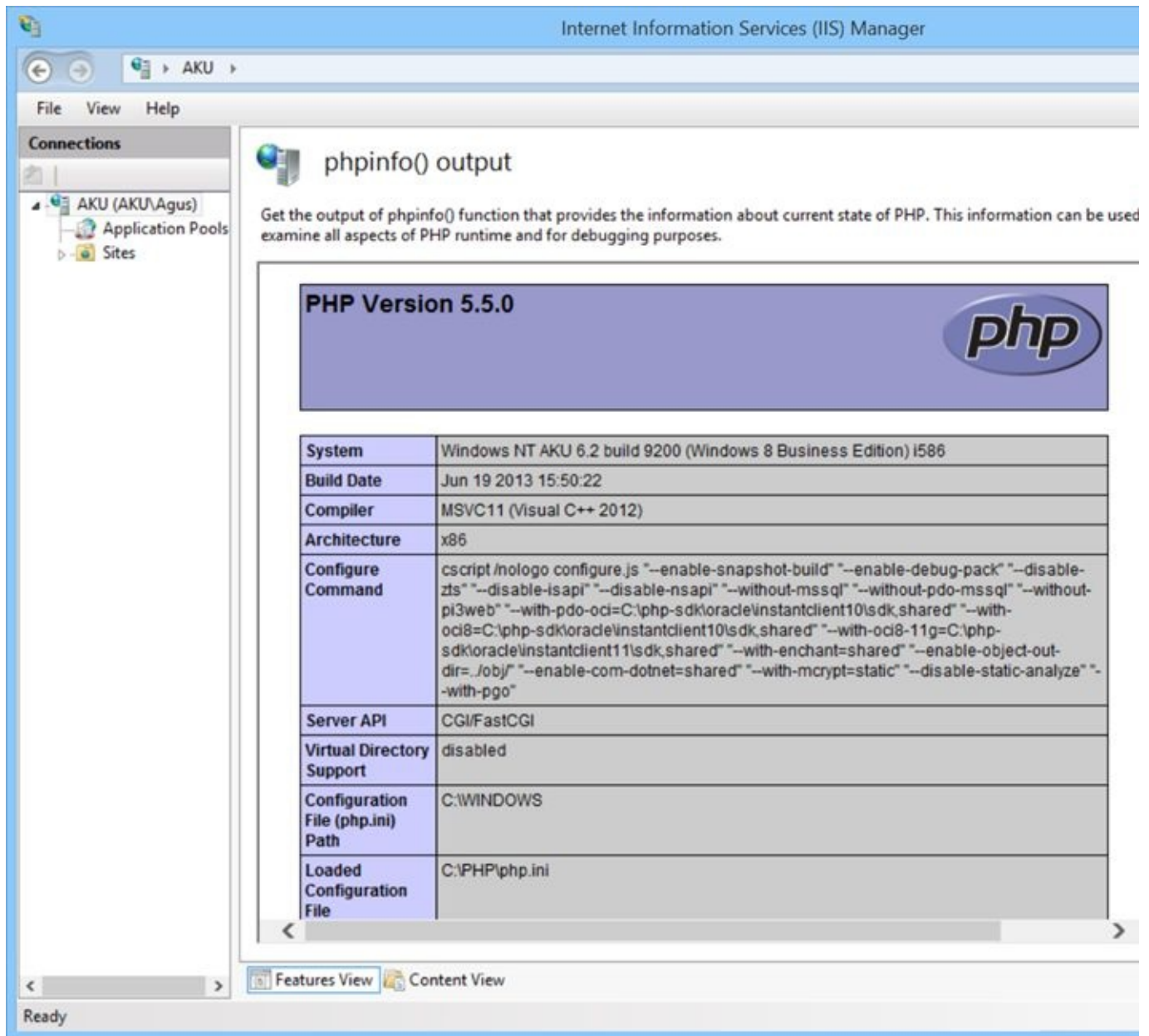


To verify, click Check phpinfo() and then you obtain a dialog as below.



Select **Default Web Site** and URL `http://localhost`. If done, click **OK** button.

If success, you can see phpinfo() output. The following is a sample output.



The screenshot shows the Internet Information Services (IIS) Manager interface. The left-hand pane displays the 'Connections' tree with 'AKU (AKU\Agus)' selected. The main content area is titled 'phpinfo() output' and includes a description: 'Get the output of phpinfo() function that provides the information about current state of PHP. This information can be used to examine all aspects of PHP runtime and for debugging purposes.' Below this is a table of PHP configuration details.

PHP Version 5.5.0	
System	Windows NT AKU 6.2 build 9200 (Windows 8 Business Edition) i586
Build Date	Jun 19 2013 15:50:22
Compiler	MSVC11 (Visual C++ 2012)
Architecture	x86
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--enable-debug-pack" "--disable-zts" "--disable-lsapi" "--disable-nsapi" "--without-mssql" "--without-pdo-mssql" "--without-pi3web" "--with-pdo-oci=C:\php-sdk\oracle\instantclient10\sdk,shared" "--with-oci8=C:\php-sdk\oracle\instantclient10\sdk,shared" "--with-oci8-11g=C:\php-sdk\oracle\instantclient11\sdk,shared" "--with-encchant=shared" "--enable-object-out-dir=.obj" "--enable-com-dotnet=shared" "--with-mcrypt=static" "--disable-static-analyze" "--with-pgo"
Server API	CGI/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	C:\WINDOWS
Loaded Configuration File	C:\PHP\php.ini

1.2.2 Apache for Linux and Mac Platforms

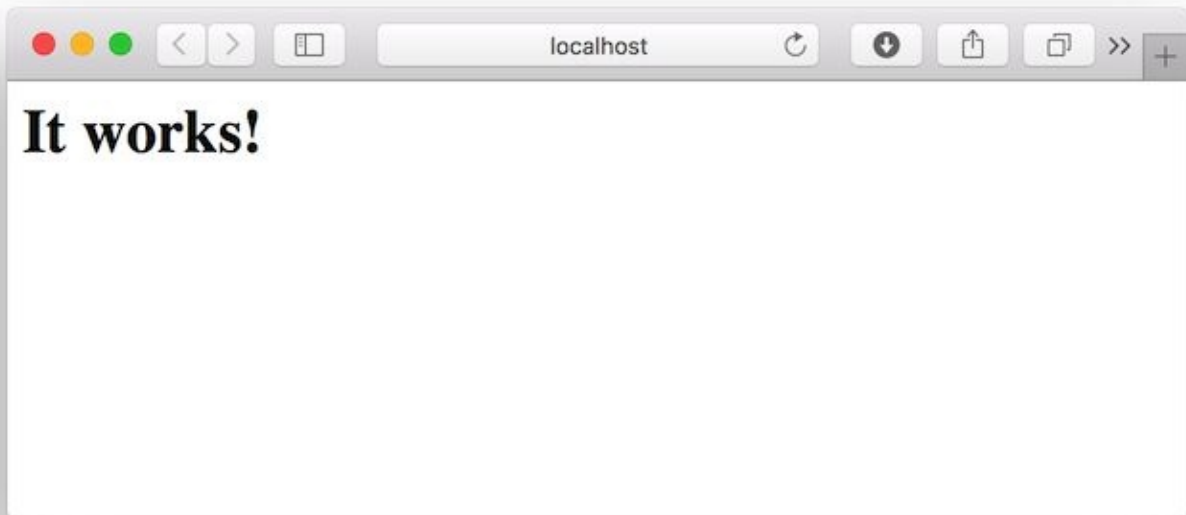
For Linux platform, you can install Apache with PHP using these commands

```
$ sudo apt-get update
$ sudo apt-get install apache2
$ sudo apt-get install php5 libapache2-mod-php5
```

For Mac, Apache has installed by default. You can enable this service using this command.

```
$ sudo apachectl start
```

Then, open a browser and navigate to <http://localhost>

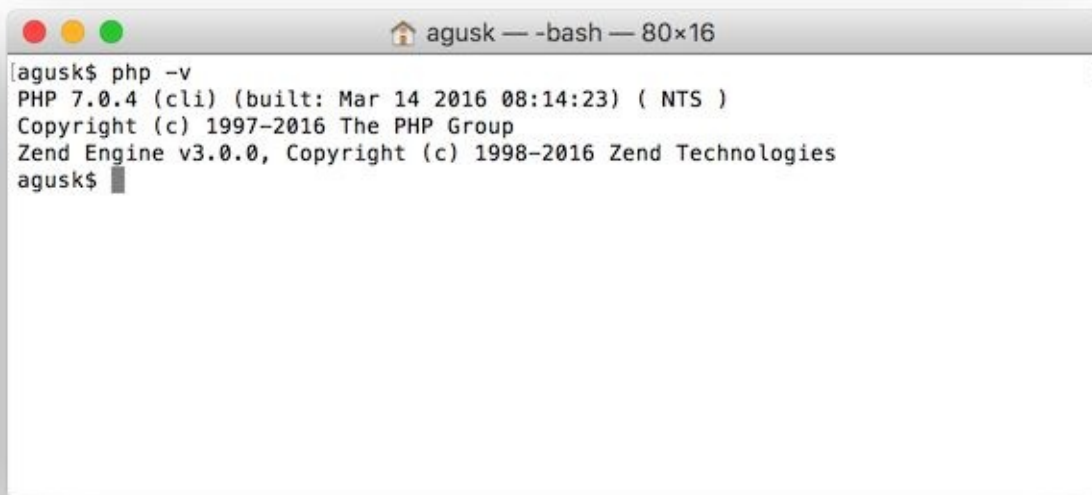


To install PHP on Mac, you can use brew. For instance, I install PHP 7 with PostgreSQL driver using these commands.

```
$ brew tap homebrew/dupes  
$ brew tap homebrew/versions  
$ brew tap homebrew/homebrew-php  
$ brew install php70 --with-postgresql
```

After installed PHP, you can verify by checking PHP version using this statement

```
$ php -v
```

A terminal window titled 'agusk — -bash — 80x16' showing the output of the command 'php -v'. The output displays 'PHP 7.0.4 (cli) (built: Mar 14 2016 08:14:23) (NTS)', 'Copyright (c) 1997-2016 The PHP Group', and 'Zend Engine v3.0.0, Copyright (c) 1998-2016 Zend Technologies'. The prompt 'agusk\$' is visible at the bottom.

```
agusk$ php -v
PHP 7.0.4 (cli) (built: Mar 14 2016 08:14:23) ( NTS )
Copyright (c) 1997-2016 The PHP Group
Zend Engine v3.0.0, Copyright (c) 1998-2016 Zend Technologies
agusk$
```

Now you can configure PHP on Apache. Firstly, backup your

```
$ cd /etc/apache2/
$ sudo cp httpd.conf httpd.conf.bak
```

Now edit httpd.conf file.

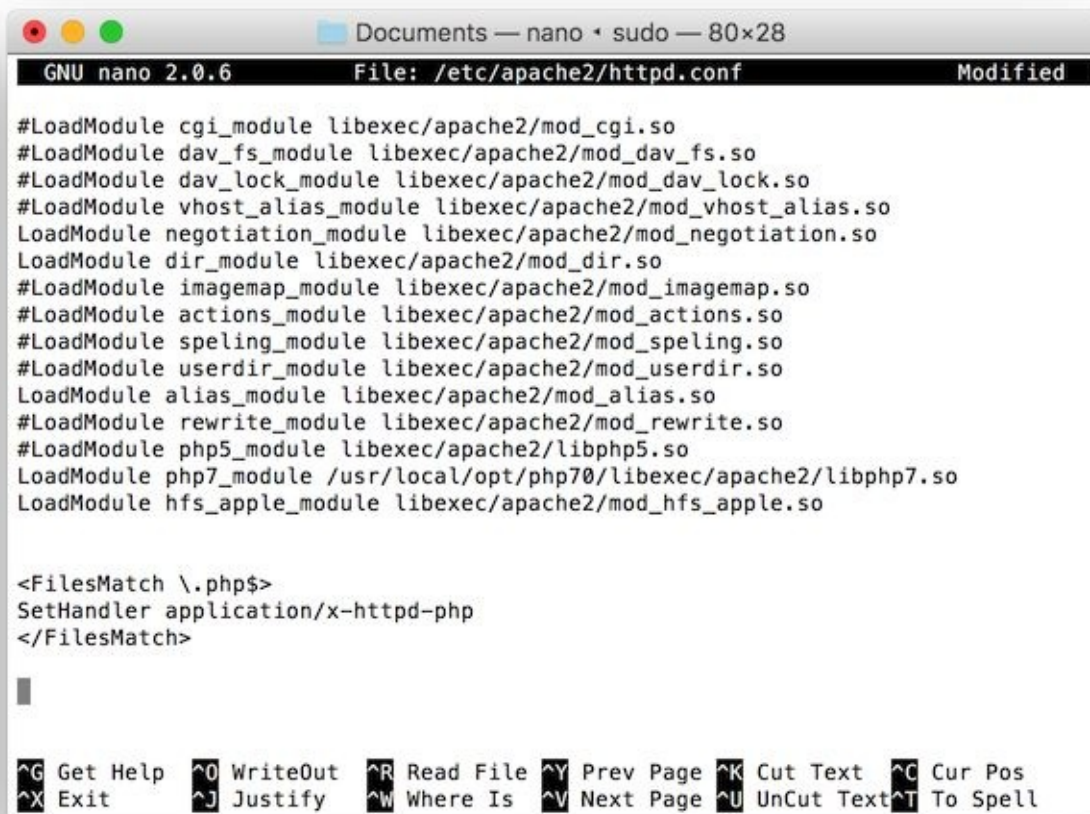
```
$ sudo nano httpd.conf
```

Disable PHP5 and enable PHP 7 as follows.

```
#disable PHP5 module
#LoadModule php5_module libexec/apache2/libphp5.so

#Enable PHP 7 module
LoadModule php7_module /usr/local/opt/php70/libexec/apache2/libphp7.so

<FilesMatch \.php$>
SetHandler application/x-httpd-php
</FilesMatch>
```



```
Documents — nano • sudo — 80x28
GNU nano 2.0.6      File: /etc/apache2/httpd.conf      Modified

#LoadModule cgi_module libexec/apache2/mod_cgi.so
#LoadModule dav_fs_module libexec/apache2/mod_dav_fs.so
#LoadModule dav_lock_module libexec/apache2/mod_dav_lock.so
#LoadModule vhost_alias_module libexec/apache2/mod_vhost_alias.so
LoadModule negotiation_module libexec/apache2/mod_negotiation.so
LoadModule dir_module libexec/apache2/mod_dir.so
#LoadModule imagemap_module libexec/apache2/mod_imagemap.so
#LoadModule actions_module libexec/apache2/mod_actions.so
#LoadModule speling_module libexec/apache2/mod_speling.so
#LoadModule userdir_module libexec/apache2/mod_userdir.so
LoadModule alias_module libexec/apache2/mod_alias.so
#LoadModule rewrite_module libexec/apache2/mod_rewrite.so
#LoadModule php5_module libexec/apache2/libphp5.so
LoadModule php7_module /usr/local/opt/php70/libexec/apache2/libphp7.so
LoadModule hfs_apple_module libexec/apache2/mod_hfs_apple.so

<FilesMatch \.php$>
SetHandler application/x-httpd-php
</FilesMatch>

^G Get Help  ^O WriteOut  ^R Read File  ^Y Prev Page  ^K Cut Text   ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is   ^V Next Page  ^U UnCut Text ^T To Spell
```

Now restart Apache.

```
$ sudo apachectl restart
```

For testing PHP, create test.php on Document root of Apache.


For Mac users, you can type this command.

```
$ sudo nano /Library/WebServer/Documents/test.php
```

In test.php, you can write this code.

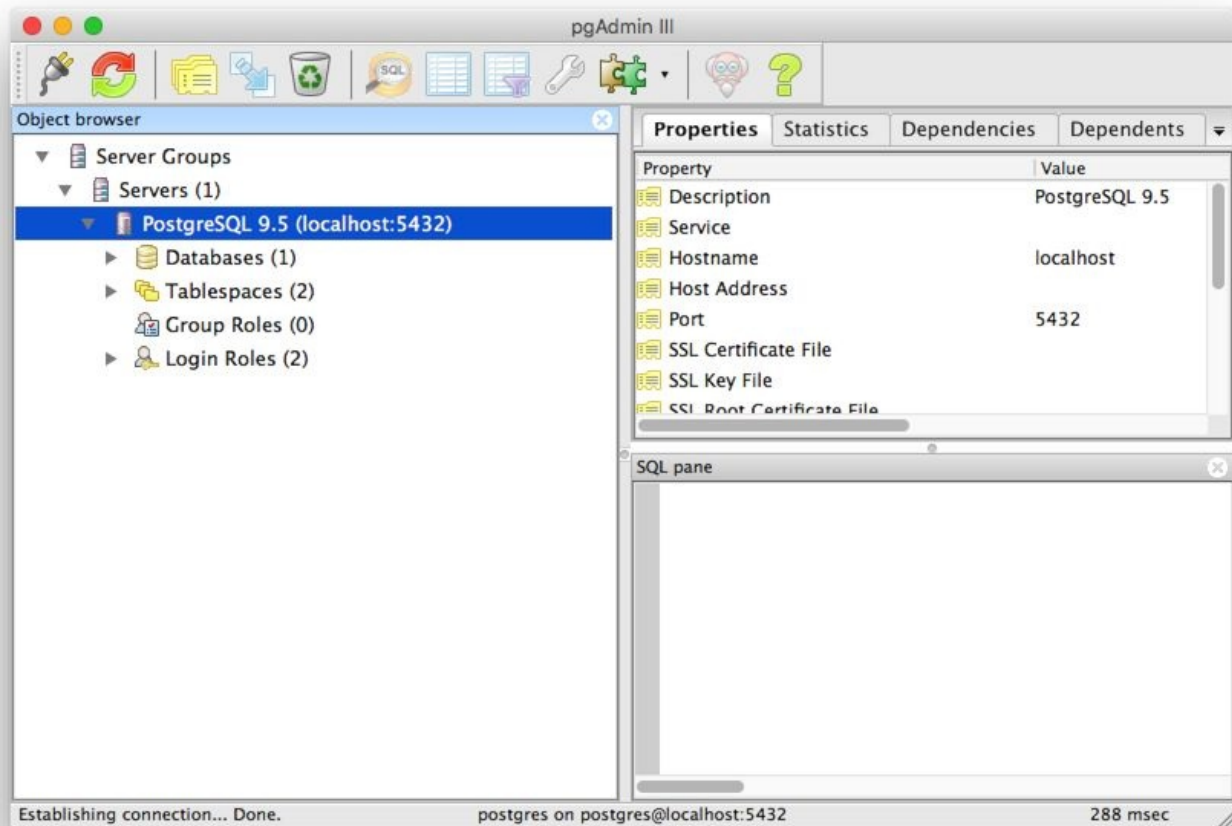
```
<?php phpinfo();
```

Now you can open a browser and navigate to `http://<server>/test.php`

Version 7.0.4	
	
	Darwin akumbp.local 15.4.0 Darwin Kernel Version 15.4.0: Fri Feb 26 22:08:05 PST 2016; root:xnu-3248.40.184~3/RELEASE_X86_64 x86_64
ate	Mar 24 2016 06:56:14
ure Command	'./configure' '--prefix=/usr/local/Cellar/php70/7.0.4' '--localstatedir=/usr/local/var' '--sysconfdir=/usr/local/etc/php/7.0' '--with-config-file-path=/usr/local/etc/php/7.0' '--with-config-file-scan-dir=/usr/local/etc/php/7.0/conf.d' '--mandir=/usr/local/Cellar/php70/7.0.4/share/man' '--enable-bcmath' '--enable-calendar' '--enable-dba' '--enable-exif' '--enable-ftp' '--enable-gd-native-ttf' '--enable-mbregex' '--enable-mbstring' '--enable-shmop' '--enable-soap' '--enable-sockets' '--enable-sysvmsg' '--enable-sysvsem' '--enable-sysvshm' '--enable-wddx' '--enable-zip' '--with-freetype-dir=/usr/local/opt/freetype' '--with-gd' '--with-gettext=/usr/local/opt/gettext' '--with-iconv-dir=/usr' '--with-icu-dir=/usr/local/opt/icu4c' '--with-jpeg-dir=/usr/local/opt/jpeg' '--with-kerberos=/usr' '--with-libedit' '--with-mhash' '--with-ndbm=/usr' '--with-pdo-odbc=unixODBC,/usr/local/opt/unixodbc' '--with-png-dir=/usr/local/opt/libpng' '--with-unixODBC=/usr/local/opt/unixodbc' '--with-xmlrpc' '--with-zlib=/usr' '--with-readline=/usr/local/opt/readline' '--without-gmp' '--without-snmp' '--with-libxml-dir=/usr/local/opt/libxml2' '--with-apxs2=/usr/sbin/apxs' '--libexecdir=/usr/local/Cellar/php70/7.0.4/libexec' '--with-bz2=/usr' '--disable-debug' '--with-openssl=/usr/local/opt/openssl' '--enable-fpm' '--with-fpm-user=_www' '--with-fpm-group=_www' '--with-curl' '--with-xsl=/usr' '--with-ldap' '--with-ldap-sasl=/usr' '--with-mysql-sock=/tmp/mysql.sock' '--with-mysqli=mysqlnd' '--with-mysql=mysqlnd' '--with-pdo-mysql=mysqlnd' '--disable-opcache' '--enable-pcntl' '--without-pear' '--with-pgsql=/usr/local/opt/postgresql' '--with-pdo-pgsql=/usr/local/opt/postgresql' '--enable-dtrace' '--disable-phpdbg' '--enable-zend-signals' 'CC=clang' 'CXX=clang++'
API	Apache 2.0 Handler
Directory Support	disabled
uration File (php.ini) Path	/usr/local/etc/php/7.0
Configuration File	/usr/local/etc/php/7.0/php.ini
is dir for additional .ini files	/usr/local/etc/php/7.0/conf.d
nal .ini files parsed	/usr/local/etc/php/7.0/conf.d/ext-pdo_pgsql.ini
	20151012

1.3 PostgreSQL Server

You can download and install PostgreSQL server on <http://www.postgresql.org/download/> . Don't forget to install pgAdmin tool to manage our database. You can use commandline or any database tool.



1.4 PostgreSQL Driver for PHP

You need PostgreSQL driver for PHP. We can install it on Ubuntu using this command.

```
$ sudo apt-get install postgresql postgresql-contrib phpPgadmin
```

On Mac, we can install it via brew.

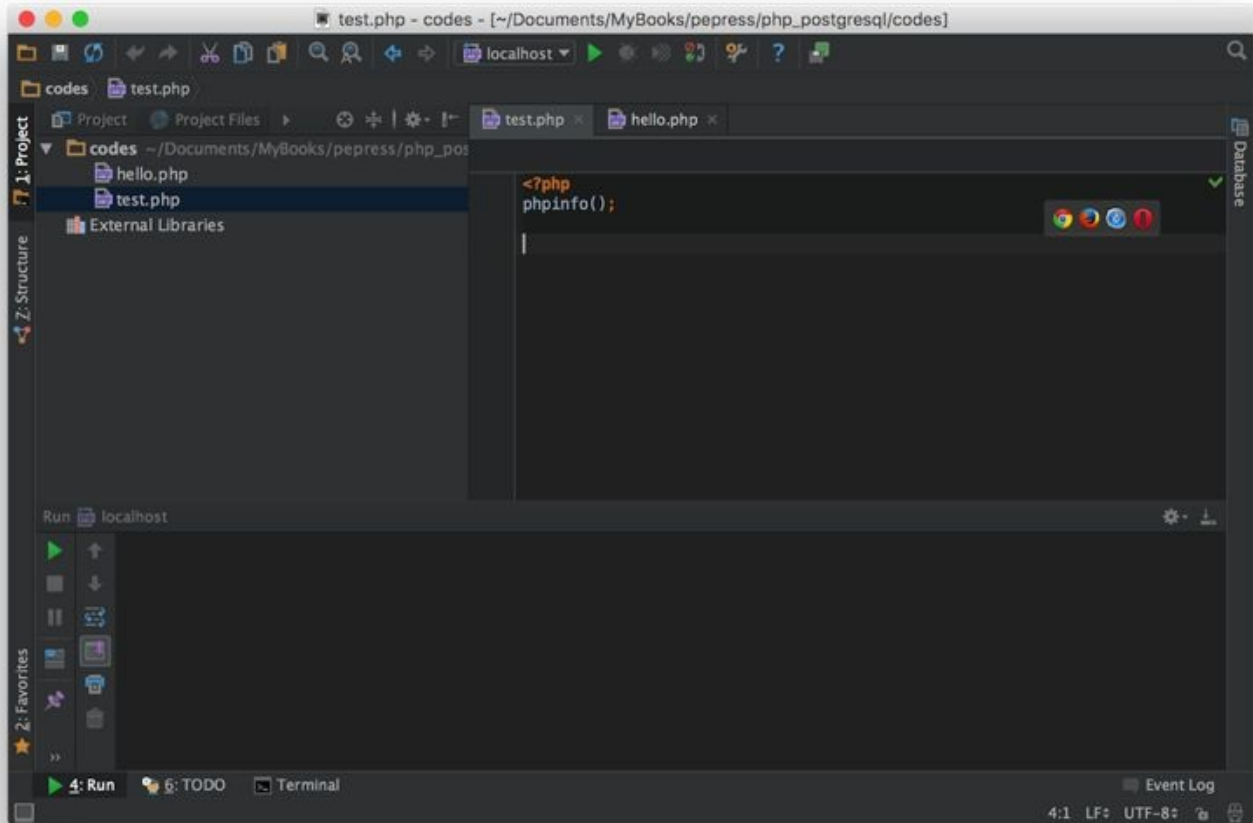
```
$ brew tap homebrew/dupes  
$ brew tap homebrew/versions  
$ brew tap homebrew/homebrew-php  
$ brew install php70 --with-postgresql
```

On Windows, a driver file for PostgreSQL is include. Then, add it using PHP Manager on IIS so this module is loaded by PHP.

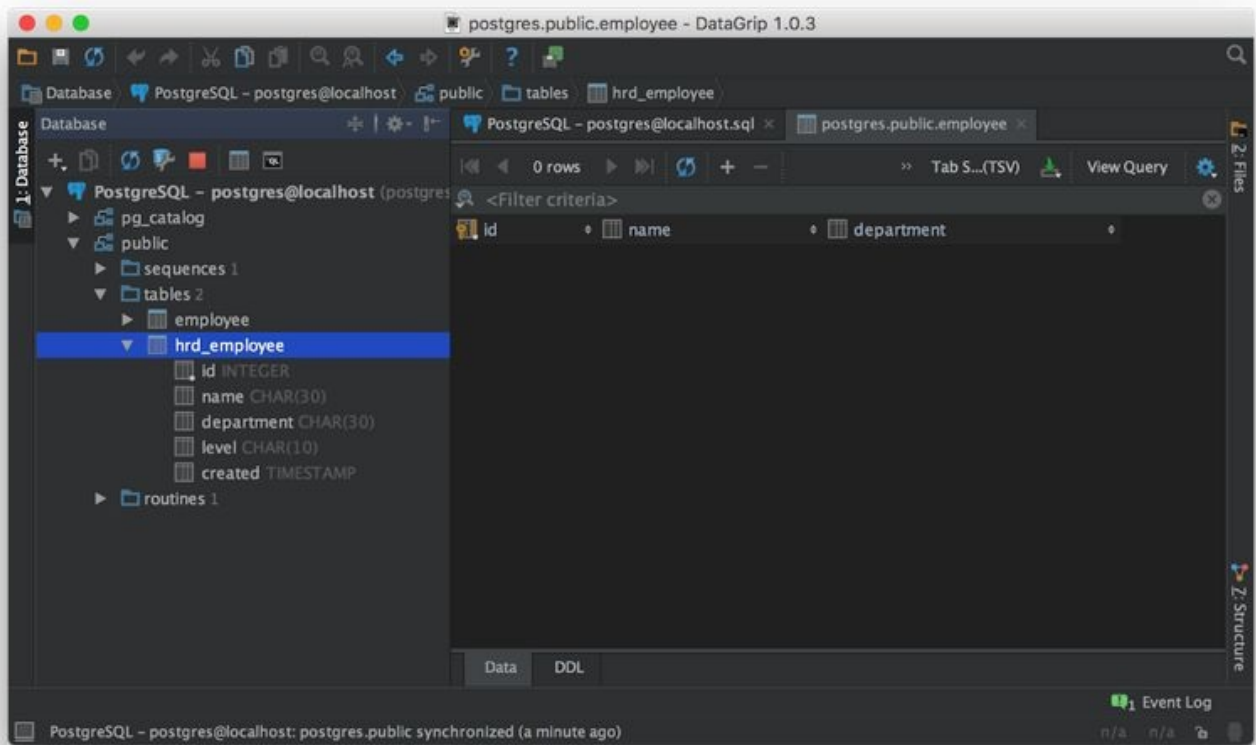
1.5 Development Tools

Basically you can use any editor to develop PHP. You can find a list of PHP editor on http://en.wikipedia.org/wiki/List_of_PHP_editors . I used PHP WebStorm, <http://www.jetbrains.com/phpstorm/> , and IntelliJ IDEA, <http://www.jetbrains.com/idea/> . They provide 30-days trial.

The following is a sample of PhpStorm.



I also use DataGrip to manage, <https://www.jetbrains.com/datagrip/> , PostgreSQL server.



2. Hello World - PHP and PostgreSQL

This chapter explains how to connect PostgreSQL Server from PHP.

2.1 Connecting to PostgreSQL

After we setup PHP and PostgreSQL driver on IIS and Apache, now we can test to connect PostgreSQL Server. We can use **pg_connect()** to connect database PostgreSQL Server and **pg_close()** to close connection.

2.2 Creating PHP and Connecting to PostgreSQL

In this section, we create PHP file, called **hello.php**. The objective is to connect PostgreSQL via database authentication.

Firstly, we define server, username, password and database.

```
<?php

$dbServer = "(local)";
$username = "dbtester";
$password = "dbtester";
$database = "postgres";
```

Please change \$username and \$password values.

Then we connect to PostgreSQL using **pg_connect()** by passing server information.

```
// Connecting, selecting database
$dbconn = pg_connect("host=$dbServer dbname=$database user=$username pas
or die('Could not connect: ' . pg_last_error());

$stat = pg_connection_status($dbconn);
if ($stat === PGSQL_CONNECTION_OK) {
    echo 'Connection status ok';
} else {
    echo 'Connection status bad';
}

// Closing connection
pg_close($dbconn);
```

At the end of code, we close connection using **pg_close()**.

Save all code.

2.3 Testing

Deploy file hello.php into web server. I testing built-in web server from PhpStorm tool. Open browser and type URL of hello.php file.

Here is a sample of running PHP application.



3. CRUD Operations

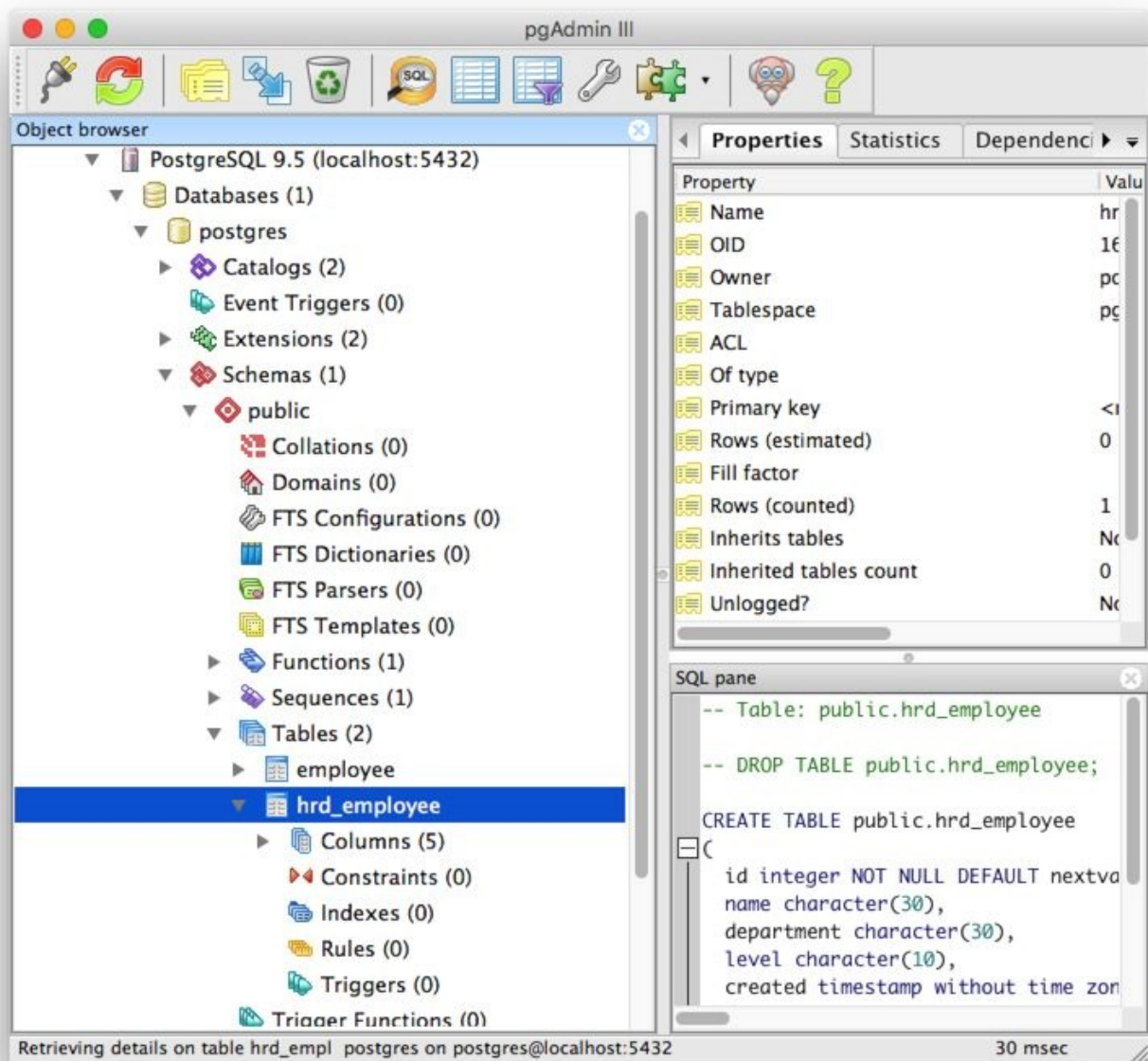
This chapter explains how to manipulate data PostgreSQL using PHP.

3.1 CRUD Operations

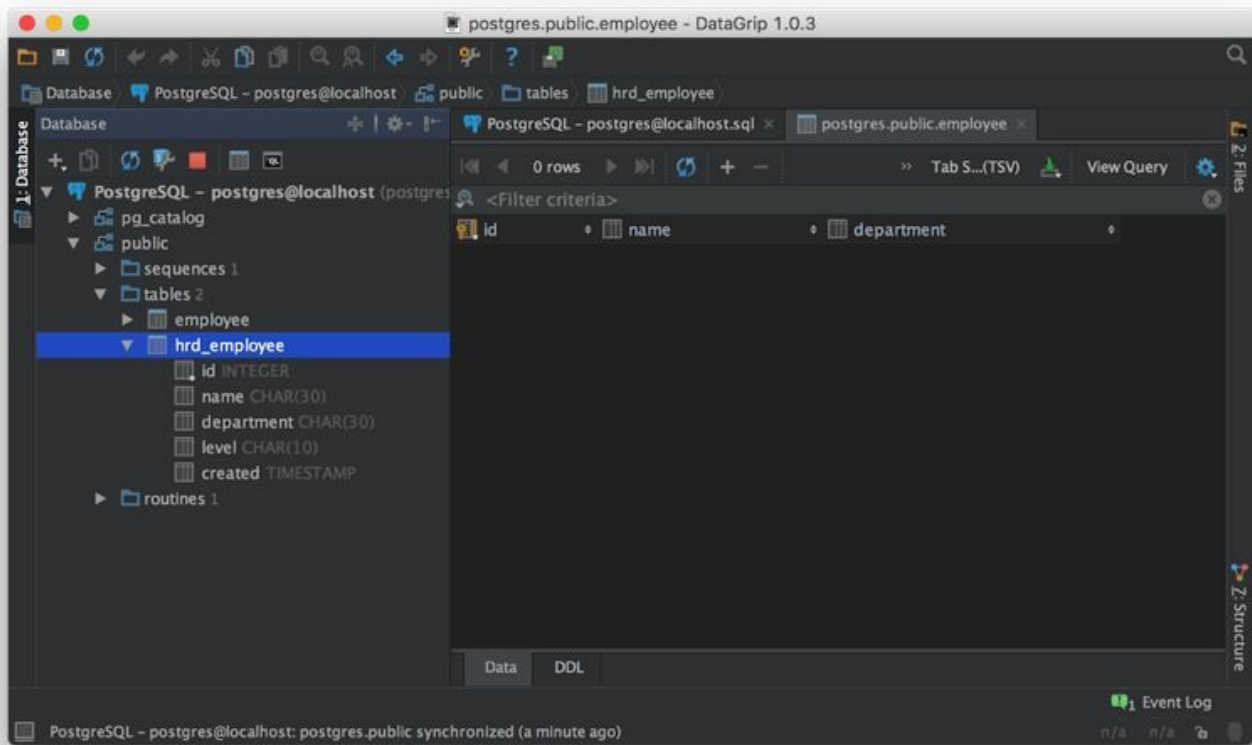
CRUD (Create, Read, Update, Delete) operations are basic data manipulation for database. In this chapter we will develop PHP application to execute CRUD operations. We use existing database, postgres. Then, create a table, called **hrd_employee**. The following is a query for creating the table.

```
CREATE TABLE public.hrd_employee
(
  id SERIAL PRIMARY KEY,
  name character(30),
  department character(30),
  level character(10),
  created timestamp without time zone
)
WITH (
  OIDS=FALSE
);
ALTER TABLE public.hrd_employee
  OWNER TO postgres;
```

You can run it using pgAdmin. You also can run it using other tool, for instance, DataGrip from JetBrains, <https://www.jetbrains.com/datagrip/> .



A form of DataGrip.



For database connection configuration, I created a PHP file, called **dbinfo.php**. The following is a content of dbinfo.php.

```
<?php

$dbServer = "localhost";
$username = "user_db";
$password = "user_pass";
$database = "postgres";

$connString = "host=$dbServer dbname=$database user=$username password=$
```

Please change database parameter values based on your configuration.

You can modify this database configuration. The next step is to create a PHP class, called **CRUDDemo**. This class provides several functions to execute CRUD operations. Let's start. Create PHP file, called **CRUDDemo.php**. Then write this script.

```
<?php

include('dbinfo.php');

class CRUDDemo {
    private $msg = "";

    public function getMsg()
    {
```

```
        return $this->msg;  
    }  
}
```

Depend on your **dbinfo.php** location, you can change its path.

We will add some functions to execute CRUD operations on next section.

3.2 Creating Data

In this scenario, we create new employee and try to retrieve inserted ID. PostgreSQL driver provides function **RETURNING** to retrieve ID from inserted query SQL statement. We will use SQL statement to insert data but we use parameters to prevent SQL injection.

We use function **pg_fetch_row()** to obtain ID value from query **RETURNING**. At the end of code, don't forget to release our usage resources and close connection using **pg_close()**.

Now we add function **createEmployee()** to create new employee. Then, write this code.

```
<?php
include('dbinfo.php');

class CRUDDemo {
    private $msg = "";

    public function getMsg()
    {
        return $this->msg;
    }
    public function createEmployee($name,$department,$level) {

        $id = -1;
        try {

            global $connString;

            $conn = pg_connect($connString);
            if( $conn === false )
            {
                $this->msg = "Unable to connect PostgreSQL Server.";
                return $id;
            }

            // insert data
            $query = "insert into hrd_employee(name,department,level,cre
            $params = array(&$name,&$department,&$level);

            $res = pg_query_params($conn,$query,$params);
            $row = pg_fetch_row($res);
            $id = $row[0];

            pg_close($conn);

        } catch (Exception $e) {
            $this->msg = $e->getMessage();
            $id = -1;
        }

        return $id;
    }
}
```

```
}  
  
}
```

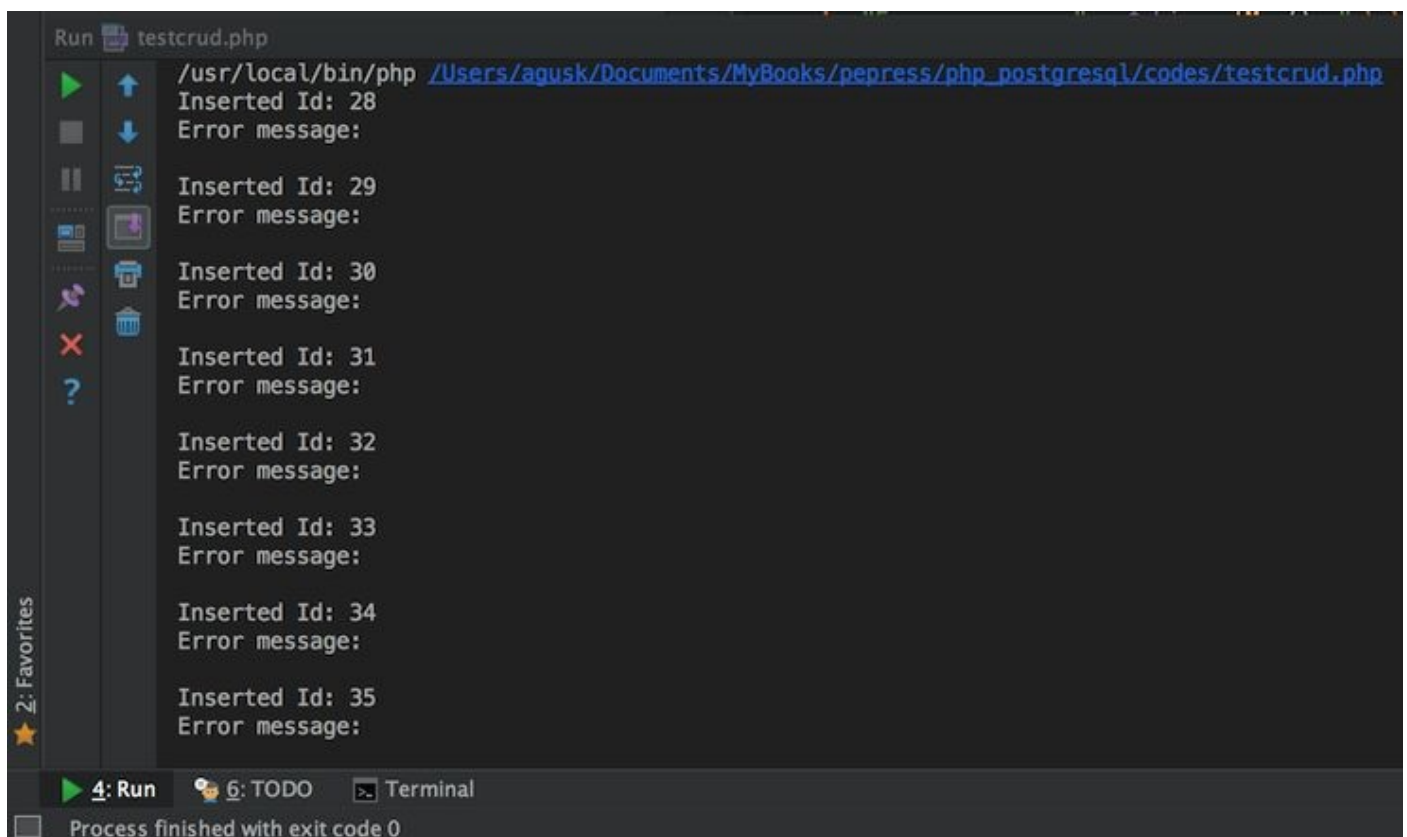
Save this code.

To test this class, we create PHP file, called **testcrud.php**. After that, we try to insert 10 data.

```
<?php  
  
include('CRUDDemo.php');  
  
$obj = new CRUDDemo();  
  
// insert 10 data  
for($i=1;$i<=10;$i++) {  
    $id = $obj->createEmployee("person " . $i,"deperment " . $i,"A".$i);  
    echo "Inserted Id: " . $id . " \n";  
    echo "Error message: " . $obj->getMsg() . " \n\n";  
}
```

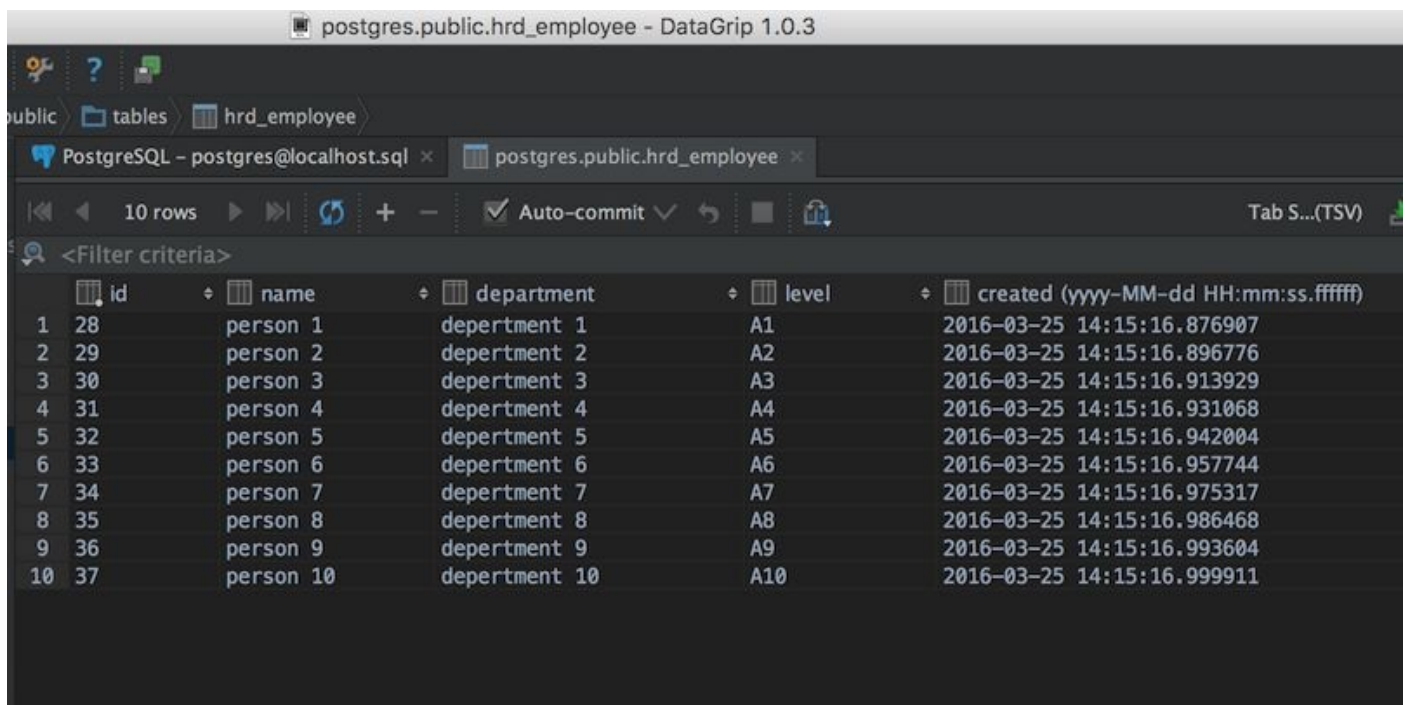
Save the code and try to run **testcrud.php**.

The following is sample running test on IntelliJ IDEA.



```
Run testcrud.php  
/usr/local/bin/php /Users/agusk/Documents/MyBooks/pepress/php_postgresql/codes/testcrud.php  
Inserted Id: 28  
Error message:  
  
Inserted Id: 29  
Error message:  
  
Inserted Id: 30  
Error message:  
  
Inserted Id: 31  
Error message:  
  
Inserted Id: 32  
Error message:  
  
Inserted Id: 33  
Error message:  
  
Inserted Id: 34  
Error message:  
  
Inserted Id: 35  
Error message:  
  
Process finished with exit code 0
```

To verify success or not, we can check database. If success, we can see 10 new data on table hrd_employee. Here is a sample of 10 data from table **hrd_employee** on PostgreSQL.



postgres.public.hrd_employee - DataGrip 1.0.3

public > tables > hrd_employee

PostgreSQL - postgres@localhost.sql x postgres.public.hrd_employee x

10 rows | Auto-commit | Tab S...(TSV)

<Filter criteria>

	id	name	department	level	created (yyyy-MM-dd HH:mm:ss.ffffff)
1	28	person 1	department 1	A1	2016-03-25 14:15:16.876907
2	29	person 2	department 2	A2	2016-03-25 14:15:16.896776
3	30	person 3	department 3	A3	2016-03-25 14:15:16.913929
4	31	person 4	department 4	A4	2016-03-25 14:15:16.931068
5	32	person 5	department 5	A5	2016-03-25 14:15:16.942004
6	33	person 6	department 6	A6	2016-03-25 14:15:16.957744
7	34	person 7	department 7	A7	2016-03-25 14:15:16.975317
8	35	person 8	department 8	A8	2016-03-25 14:15:16.986468
9	36	person 9	department 9	A9	2016-03-25 14:15:16.993604
10	37	person 10	department 10	A10	2016-03-25 14:15:16.999911

3.3 Reading Data

After inserted data, we read data. The following is a list of reading data step:

- Open connection using **pg_connect()**
- Create query with SELET statement
- Execute query using **pg_query()**
- Looping to read data using **pg_fetch_row()**
- Close connection using **pg_close()**

Create a function called **readEmployeeList()** on **CRUDDemo.php** and write this code:

```
<?php
include('dbinfo.php');

class CRUDDemo {
    private $msg = "";

    public function getMsg()
    {
        return $this->msg;
    }
    public function readEmployeeList() {

        $data = array();
        try {

            global $connString;

            $conn = pg_connect($connString);
            if( $conn === false )
            {
                $this->msg = "Unable to connect PostgreSQL Server.";
                return $data;
            }

            $query = "select id,name,department,level,created from hrd_e
            $result = pg_query($conn,$query);

            while($row = pg_fetch_row($result))
            {
                array_push($data, array("id"=>$row[0], "name"=>$row[1],
                    "department"=>$row[2], "level"=>$row[3], "created"=>$r

            }

            pg_close($conn);

        } catch (Exception $e) {
            $this->msg = $e->getMessage();
        }
    }
}
```

```
        return $data;
    }
}
```

You can see that we define **\$data** to hold data from database PostgreSQL.

Save this code.

Now we test function **readEmployeeList()**. Open file testcrud.php and add code as below.

```
<?php


include('CRUDDemo.php');

$obj = new CRUDDemo();

// read data
$list = $obj->readEmployeeList();
if($list) {
    foreach($list as $item) {
        $content = "";
        foreach($item as $key => $value) {
            $content = $content . $key . ": " . $value . "\n";
        }
        echo $content . "\n";
    }
}
```

Save this code.

You can execute file **testcrud.php**. The following is a sample of output.

Run  testcrud.php

```
/usr/local/bin/php /Users/agusk/Documents/MyBooks/pepress/php_postgresql/codes/testcrud.php
```

id: 28

```
name: person 1
```

department: deperment 1

level: A1

created: 2016-03-25 14:15:16.876907

id: 29

```
name: person 2
```

department: deperment 2

level: A2

created: 2016-03-25 14:15:16.896776

id: 30

```
name: person 3
```

department: deperment 3

level: A3

created: 2016-03-25 14:15:16.913929

id: 31

name: person 4

department: deperment 4

level: A4

created: 2016-03-25 14:15:16.931068

★ 2: Favorites

▶ 4: Run

6: TODO

 Terminal

3.4 Update Data

On this section, we will update data on particular ID. We use UPDATE statement and pass updated data.

Create a function called **updateEmployee()** and write this code.

```
<?php
include('dbinfo.php');

class CRUDDemo {
    private $msg = "";

    public function getMsg()
    {
        return $this->msg;
    }
    public function updateEmployee($id,$name,$department,$level) {

        $success = -1;
        try {

            global $connString;

            $conn = pg_connect($connString);
            if( $conn === false )
            {
                $this->msg = "Unable to connect PostgreSQL Server.";
                return $success;
            }

            $query = "update hrd_employee set name=$1,department=$2,level=$3 where id=$4";
            $params = array(&$name,&$department,&$level,&$id);

            $res = pg_query_params($conn,$query,$params);
            if($res === FALSE) {
                $this->msg = "Error in executing query.";
                return $success;
            }
            $num_rows=pg_affected_rows($res);
            $success = $num_rows;
            $this->msg = "";

            pg_close($conn);

        } catch (Exception $e) {
            $this->msg = $e->getMessage();
            $success = -1;
        }

        return $success;
    }
}
```

```
}
```

Save this code.

Now we work on file **testcrud.php** to call **updateEmployee()**. In this testing, we update employee with ID 30. You may change employee ID.

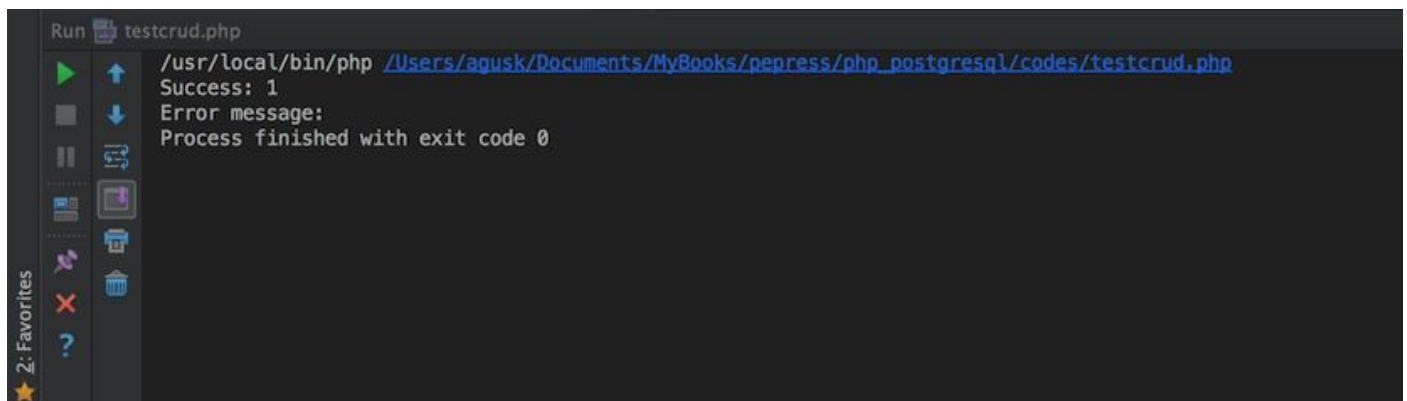
```
<?php

include('CRUDDemo.php');

$obj = new CRUDDemo();

// update
// update data with id = 30. Change it with own id data
$success = $obj->updateEmployee(30, "update-employee", "update-department");
echo "Success: " . $success . "\n";
echo "Error message: " . $obj->getMsg();
```

Save this code. Run file **testcrud.php**.



```
Run testcrud.php
/usr/local/bin/php /Users/agusk/Documents/MyBooks/pepress/php_postgresql/codes/testcrud.php
Success: 1
Error message:
Process finished with exit code 0
```

If we open table **hrd_employee**, you can find updated data on data with ID = 30.

public > tables > hrd_employee

PostgreSQL - postgres@localhost.sql × postgres.public.hrd_employee ×

10 rows | Auto-commit ✓ | Tab S...(TSV)

<Filter criteria>

	id	name	department	level	created (yyyy-MM-dd HH:mm:ss.ffffff)
1	28	person 1	department 1	A1	2016-03-25 14:15:16.876907
2	29	person 2	department 2	A2	2016-03-25 14:15:16.896776
3	31	person 4	department 4	A4	2016-03-25 14:15:16.931068
4	32	person 5	department 5	A5	2016-03-25 14:15:16.942004
5	33	person 6	department 6	A6	2016-03-25 14:15:16.957744
6	34	person 7	department 7	A7	2016-03-25 14:15:16.975317
7	35	person 8	department 8	A8	2016-03-25 14:15:16.986468
8	36	person 9	department 9	A9	2016-03-25 14:15:16.993604
9	27	person 10	department 10	A10	2016-03-25 14:15:16.000011
10	30	update-employee	update-department	u-level	2016-03-25 14:15:16.913929

3.5 Deleting Data

To delete data, we use SQL statement using DELETE FROM and pass employee ID. Create a function called **deleteEmployee()** and write this code.

```
<?php
include('dbinfo.php');

class CRUDDemo {
    private $msg = "";

    public function getMsg()
    {
        return $this->msg;
    }
    public function deleteEmployee($id) {

        $success = -1;
        try {

            global $connString;

            $conn = pg_connect($connString);
            if( $conn === false )
            {
                $this->msg = "Unable to connect PostgreSQL Server.";
                return $success;
            }

            $query = "delete from hrd_employee where id=$1";
            $params = array(&$id);

            $res = pg_query_params($conn,$query,$params);
            if($res === FALSE) {
                $this->msg = "Error in executing query.";
                return $success;
            }
            $num_rows=pg_affected_rows($res);
            $success = $num_rows;
            $this->msg = "";

            pg_close($conn);

        } catch (Exception $e) {
            $this->msg = $e->getMessage();
            $success = -1;
        }

        return $success;
    }
}
```

Save this code.

Now we test this function on file **testcrud.php**. We delete data with ID = 30. You can change this ID.

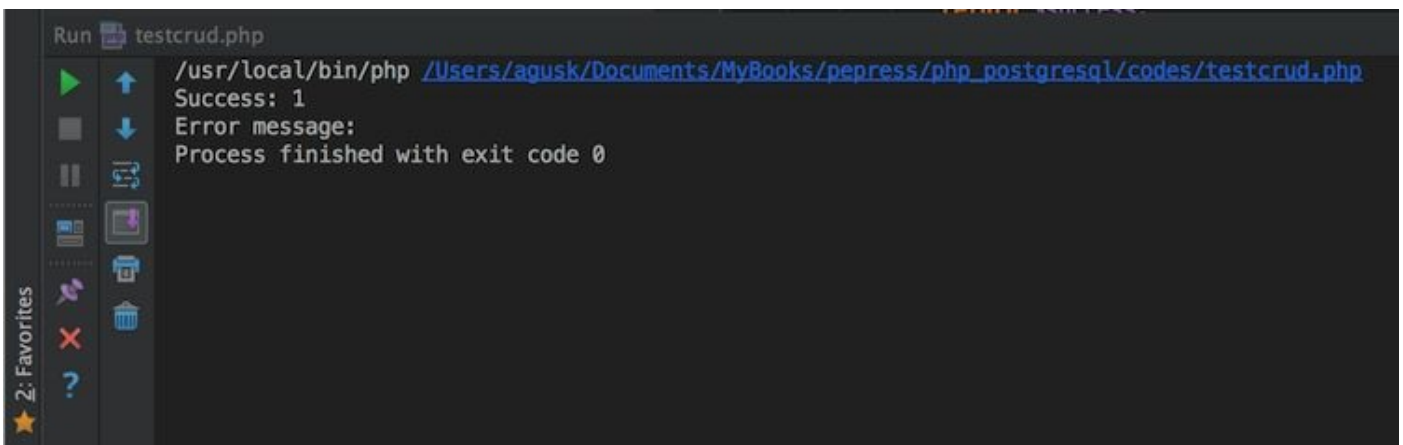
```
<?php

include('CRUDDemo.php');

$obj = new CRUDDemo();

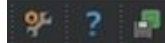
// delete
// delete data with id = 30. Please change it
$success = $obj->deleteEmployee(30);
echo "Success: " . $success . "\n";
echo "Error message: " . $obj->getMsg();
```

Save this code and run **testcrud.php**.



```
Run testcrud.php
/usr/local/bin/php /Users/agusk/Documents/MyBooks/pepress/php_postgresql/codes/testcrud.php
Success: 1
Error message:
Process finished with exit code 0
```

You can verify by opening table **hrd_employee**. Data with ID = 30 should not show.



public > tables > hrd_employee

PostgreSQL - postgres@localhost.sql × postgres.public.hrd_employee ×

9 rows | Refresh | + | - | Auto-commit | Tab S...(TSV)

<Filter criteria>

	id	name	department	level	created (yyyy-MM-dd HH:mm:ss.ffffff)
1	28	person 1	department 1	A1	2016-03-25 14:15:16.876907
2	29	person 2	department 2	A2	2016-03-25 14:15:16.896776
3	31	person 4	department 4	A4	2016-03-25 14:15:16.931068
4	32	person 5	department 5	A5	2016-03-25 14:15:16.942004
5	33	person 6	department 6	A6	2016-03-25 14:15:16.957744
6	34	person 7	department 7	A7	2016-03-25 14:15:16.975317
7	35	person 8	department 8	A8	2016-03-25 14:15:16.986468
8	36	person 9	department 9	A9	2016-03-25 14:15:16.993604
9	37	person 10	department 10	A10	2016-03-25 14:15:16.999911

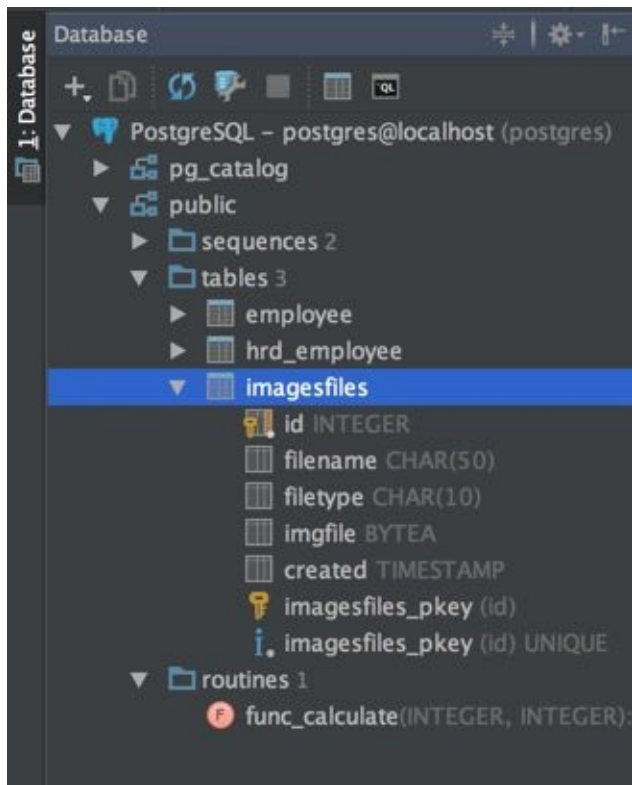
4. Working with Image and Blob Data

This chapter explains how to work with image and blob data on PostgreSQL and manipulate them using PHP.

4.1 Image and Blob Data

PostgreSQL provides data type for BYTEA which can store image file and binary type data.

In this chapter, we will work with image file on PostgreSQL. The following is table schema, called **ImagesFiles**.



The following is a script for creating imagefiles table.

```
CREATE TABLE public.imagesfiles
(
    id SERIAL PRIMARY KEY,
    filename character(50),
    filetype character(10),
    imgfile BYTEA,
    created timestamp without time zone
)
WITH (
    OIDS=FALSE
);
ALTER TABLE public.imagesfiles
OWNER TO postgres;
```

filetype is image type of uploading file. This is important while we want to display the image. See section 4.3 for further explanation.

The next action we create two PHP application, uploading image and listing image data.

4.2 Uploading Image

In this section, we upload image using traditional way: file html. After user choose image file and then click **Submit** button, PHP file will process uploading file and store it into database.

Firstly, we create **uploadimage.html** and write this script.

```
<!DOCTYPE html>
<html>
<head>
    <title>PHP and PostgreSQL Programming by Example</title>
</head>
<body>
<h1>Uploading Image file</h1>

<p><br/></p>
<p>Upload image file (JPG/PNG/TIF/GIF) with file size maximum 500KB.</p>

<form action="upload_process.php" method="post"
    enctype="multipart/form-data">
    <label for="file">Image File (JPG/PNG/TIF/GIF):</label>
    <input type="file" name="file" id="file"><br/><br/>
    <input type="submit" name="submit" value="Submit">
</form>

</body>
</html>
```

You can see we use form multipart/form-data and input file to upload image file. The form action is POST and navigate to **upload_process.php**.

Now we create **upload_process.php** to process uploading file. We validate file input for gif, tif, jpg and png and restrict file size below 500 KB. Let's write this code into **upload_process.php**.

```
<?php

include('dbinfo.php');

$allowedExts = array("gif", "tif", "jpg", "png");
$filename = $_FILES["file"]["name"];
$temp = explode(".", $filename);
$fileExt = end($temp);
$imageType = $_FILES["file"]["type"];

if ((( $imageType == "image/gif" ) || ( $imageType == "image/jpg" ) || ( $imageType == "image/tif" ) || ( $imageType == "image/png" ) )
    && ( $_FILES["file"]["size"] < 500000 )
    && in_array($fileExt, $allowedExts) ) {
```

```

$file_name = $_FILES['file']['tmp_name'];

// open connection to database server
global $connString;
$conn = pg_connect($connString);
if( $conn === false )
{
    echo "Unable to connect PostgreSQL Server.";
    return;
}

// query data to insert image file
$query = "insert into imagesfiles(filename, filetype, imgfile, created)

$img = fopen($file_name, 'r') or die("cannot read image\n");
$data = fread($img, filesize($file_name));
$img_data = pg_escape_bytea($data);
fclose($img);

$params = array(&$filename, &$imageType, &$img_data);

if(!(pg_query_params($conn, $query, $params)))
{
    echo "Error in executing query.<br>";
    pg_close($conn);
    return;
}

pg_close($conn);
echo "Uploaded file was success <br>";

}else {
    echo "Invalid file <br>";
}

```

Basically it likes normal inserting data but for field imgfile (image data type) we pass a content of image file using **pg_escape_bytea()** into query parameter.

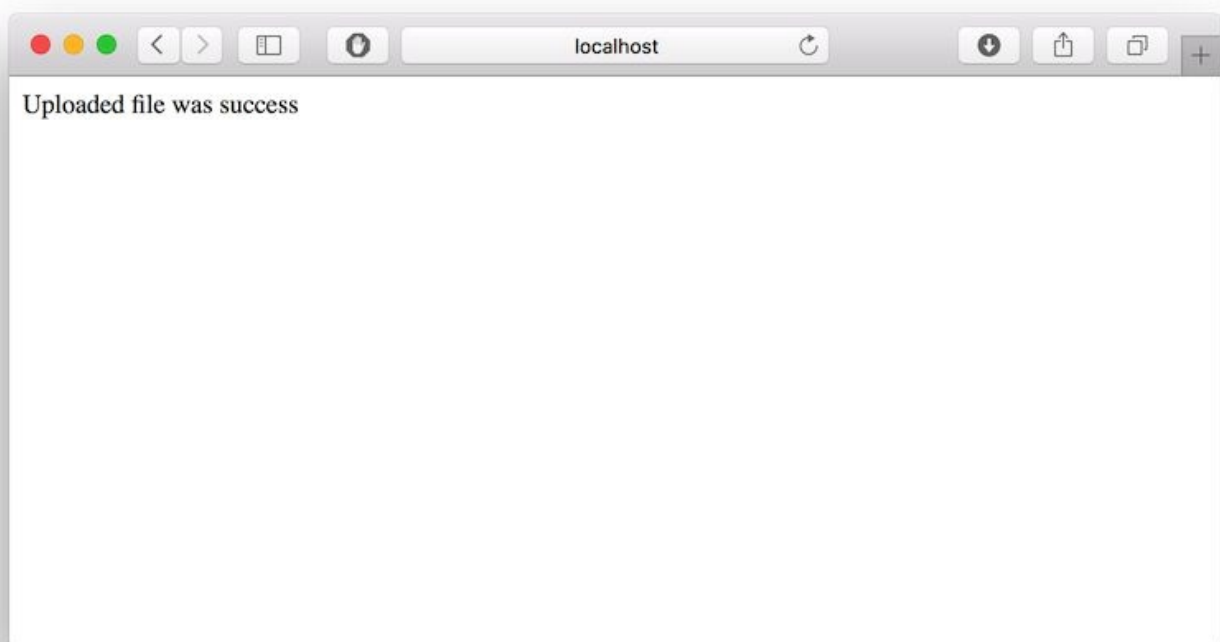
dbinfo.php is a database configuration file. You can use the same content from section 3.1.

If done, save the code. Try to run.

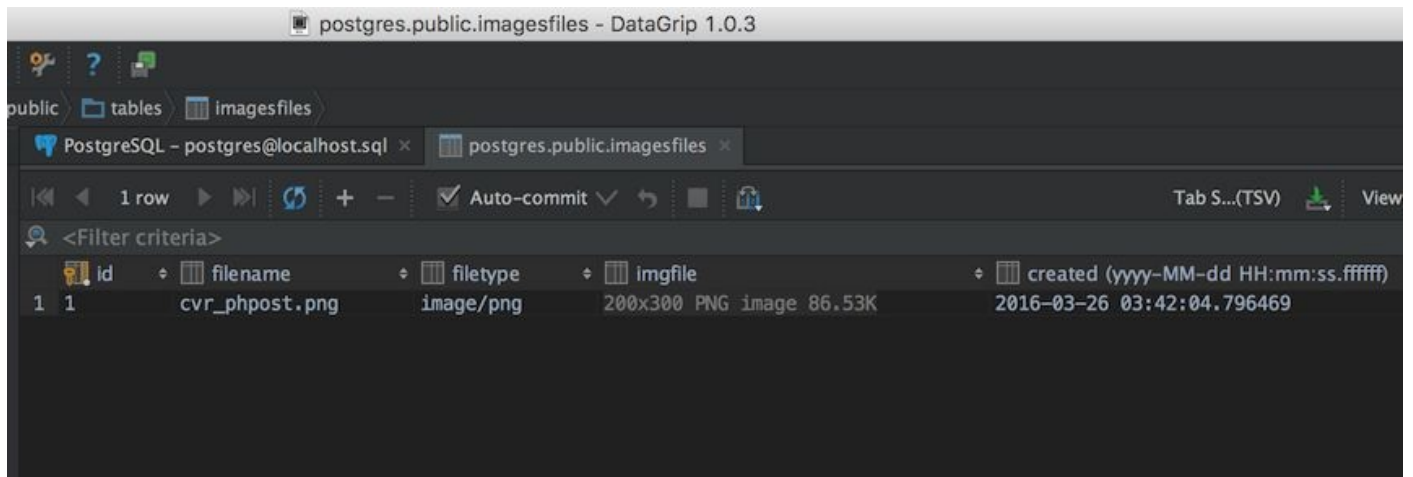
Open file **uploadingimage.html** on browser. Choose image file and then click Submit button.



If success, we will obtain response as follows.



To verify uploading file, you can check table **imagesfiles**. You can find this uploaded file into database.



The screenshot shows the DataGrip 1.0.3 interface. The top bar indicates the current database is 'postgres.public.imagesfiles'. Below the toolbar, the breadcrumb navigation shows 'public' > 'tables' > 'Imagesfiles'. The main area displays a table with the following columns: 'id', 'filename', 'filetype', 'imgfile', and 'created (yyyy-MM-dd HH:mm:ss.ffffff)'. A single row is visible with the following data: '1', 'cvr_phpost.png', 'image/png', '200x300 PNG image 86.53K', and '2016-03-26 03:42:04.796469'.

id	filename	filetype	imgfile	created (yyyy-MM-dd HH:mm:ss.ffffff)
1	cvr_phpost.png	image/png	200x300 PNG image 86.53K	2016-03-26 03:42:04.796469

Try to upload another image file for testing on listing image files.

4.3 Listing Image Data

Now we build PHP application to show a list of image data. As we know, HTML provides tag `img` (``) to display an image. We use the following format to display the image:

```
<img src='data:image/jpeg;base64,<base64_of_imagefile>' width=200px height=200px />
```

Depend on you image type, we can change data value, for instance PNG image file, to be `image/png`. `<base64_of_imagefile>` is base64 value of image. We can use `base64_encode()` to convert image to base64. After obtained image data from database, we should call `pg_unescape_bytea()` function.

Now create a PHP file, called `list_images.php`, and write this code.

```
<!DOCTYPE html>
<html>
<head>
    <title>PHP and PostgreSQL Programming by Example</title>
</head>
<body>
<h1>List of Image file</h1>

<p><br/></p>

<p>Retrieving image from database.</p>
<br/>

<?php

include('dbinfo.php');

global $connString;
$conn = pg_connect($connString);
if ($conn === false) {
    echo "Unable to connect PostgreSQL.<br>";
    return;
}

// query data to insert image file
$query = "select id,filename,filetype,imgfile,created from imagesfiles";

$result = pg_query($conn, $query);
while ($row = pg_fetch_row($result)) {
    echo "<div>";
    $imgType = $row[2];
    $imgfile = pg_unescape_bytea($row[3]);
    $img = base64_encode($imgfile);

    echo "<p>ID:" . $row[0] . "</p><p>" . $row[1] . "<p>" . $row[4] .
        "</p><p><img src='data:". $imgType. ";base64,". $img. "' width="
    echo "</div>";
}
```

```
}  
  
pg_close($conn);  
  
?>  
  
</body>  
</html>
```

Save the code.

Now open file **list_images.php** on browser. If data is available, you can see the result output, shown in Figure below.



You can modify to enhance website UI using CSS or JavaScript.

5. Transaction

This chapter explains how to work with transaction on PostgreSQL using PHP.

5.1 PHP and PostgreSQL Transaction

PostgreSQL driver provides transaction operations via query. We use BEGIN query to start a transaction and COMMIT to complete the transaction. Otherwise, we can use ROLLBACK to restore our previous transaction.

How to use these functions? Let's write PHP application.

5.2 Demo

To illustrate transaction demo, we create a simple PHP application. We insert 5 data into database. We use the same code from chapter 3. If one of them is failed we will call rollback transaction.

Let's build. Create PHP file, called **transaction_demo.php**, and write the following code:

```
<?php

include('dbinfo.php');

// connect to postgresql
global $connString;

$conn = pg_connect($connString);
if( $conn === false )
{
    echo "Unable to connect PostgreSQL Server.";
    return;
}
echo "Connected to PostgreSQL Server.<br>";

// start transaction
pg_query("BEGIN") or die("Could not start transaction\n");

echo "Inserting data.<br>";
for($i=1;$i<=5;$i++) {

    $name = "person-".$i;
    $department = "finance";
    $level = "B";
    $query = "insert into hrd_employee(name,department,level,created) va";
    $params = array(&$name,&$department,&$level);

    $res = pg_query_params($conn,$query,$params);
    if($res === false)
    {
        echo "Error in executing query.<br>";
        echo "Rollback transaction.<br>";

        pg_query("ROLLBACK") or die("Transaction rollback failed\n");;
        pg_close($conn);
        return;
    }

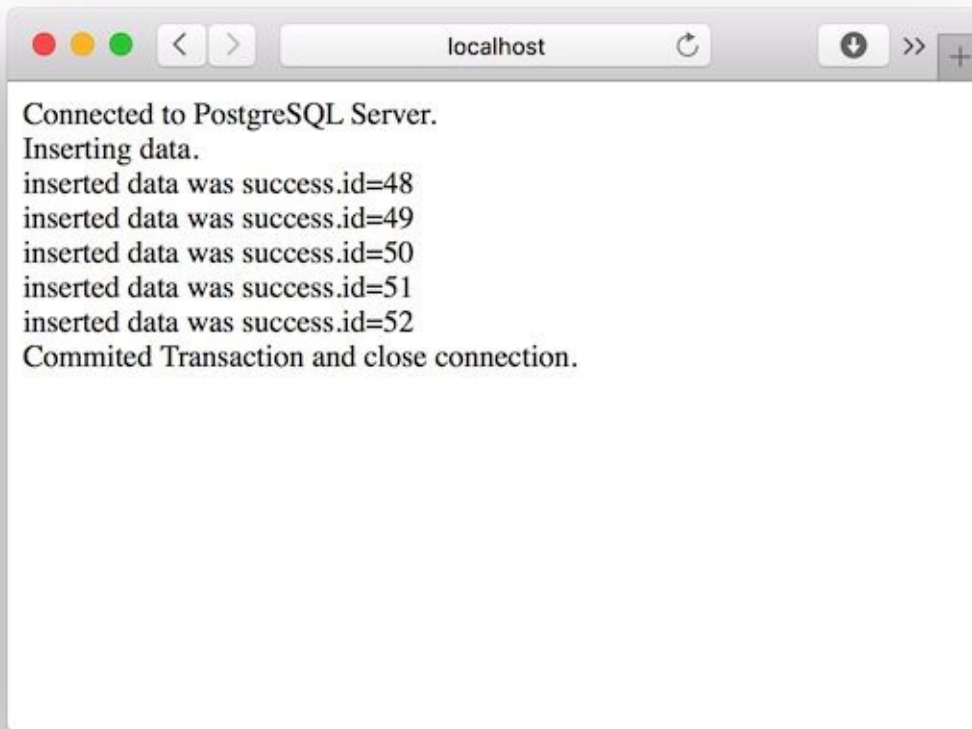
    $row = pg_fetch_row($res);
    $id = $row[0];
    echo "inserted data was success.id=".$id."<br>";

}
```

```
pg_query("COMMIT") or die("Transaction commit failed\n");  
pg_close($conn);  
echo "Committed Transaction and close connection.<br>";
```

Save this code and run it.

Open file **transaction_demo.php** on browser. If success, you can see the response as follows.



You can verify inserted data to database PostgreSQL.

public > tables > hrd_employee					
PostgreSQL - postgres@localhost.sql × postgres.public.imagesfiles × postgres.public.hrd_employee ×					
24 rows Auto-commit Tab S...(TSV) View					
<Filter criteria>					
	id	name	department	level	created (yyyy-MM-dd HH:mm:ss.ffffff)
7	35	person 8	deperment 8	A8	2016-03-25 14:15:16.986468
8	36	person 9	deperment 9	A9	2016-03-25 14:15:16.993604
9	37	person 10	deperment 10	A10	2016-03-25 14:15:16.999911
10	38	person-1	finance	B	2016-03-26 04:42:18.799703
11	39	person-2	finance	B	2016-03-26 04:42:18.799703
12	40	person-3	finance	B	2016-03-26 04:42:18.799703
13	41	person-4	finance	B	2016-03-26 04:42:18.799703
14	42	person-5	finance	B	2016-03-26 04:42:18.799703
15	43	person-1	finance	B	2016-03-26 04:42:52.573706
16	44	person-2	finance	B	2016-03-26 04:42:52.573706
17	45	person-3	finance	B	2016-03-26 04:42:52.573706
18	46	person-4	finance	B	2016-03-26 04:42:52.573706
19	47	person-5	finance	B	2016-03-26 04:42:52.573706
20	48	person-1	finance	B	2016-03-26 04:43:35.389129
21	49	person-2	finance	B	2016-03-26 04:43:35.389129
22	50	person-3	finance	B	2016-03-26 04:43:35.389129
23	51	person-4	finance	B	2016-03-26 04:43:35.389129
24	52	person-5	finance	B	2016-03-26 04:43:35.389129
Data DDL					

6. Stored Procedures

This chapter explains how to work with Stored Procedures on PostgreSQL and call them from PHP.

6.1 Stored Procedures

A stored procedure is a subroutine that access a relational database system. Stored procedure can be accessed by application. In this section, I will show you how to create a stored procedure in PostgreSQL and then call it from PHP.

There are three Stored Procedure models:

- Stored Procedure without parameter. It can be return value or not
- Stored Procedure with input parameter
- Stored Procedure with input and output parameters

These models will be implemented on next sections. Firstly, we create PHP class, called **SPDemo**. Then, write this skeleton code:

```
<?php

include( 'dbinfo.php' );

class SPDemo {

    private $msg = "";

    public function getMsg()
    {
        return $this->msg;
    }
}
```

Save this code as **SPDemo.php**.

We will add several functions to access Stored Procedure. In chapter, we use **hrd_employee** table from postgres database.

6.2 Calling Stored Procedure

To call Stored Procedure from PHP with a cursor, we do the following steps:

- Open connection to PostgreSQL using `pg_connect()`
- Do a transaction `BEGIN` and `COMMIT`
- Inside a transaction, we call `pg_query_params()` to call stored procedure

Let's start.

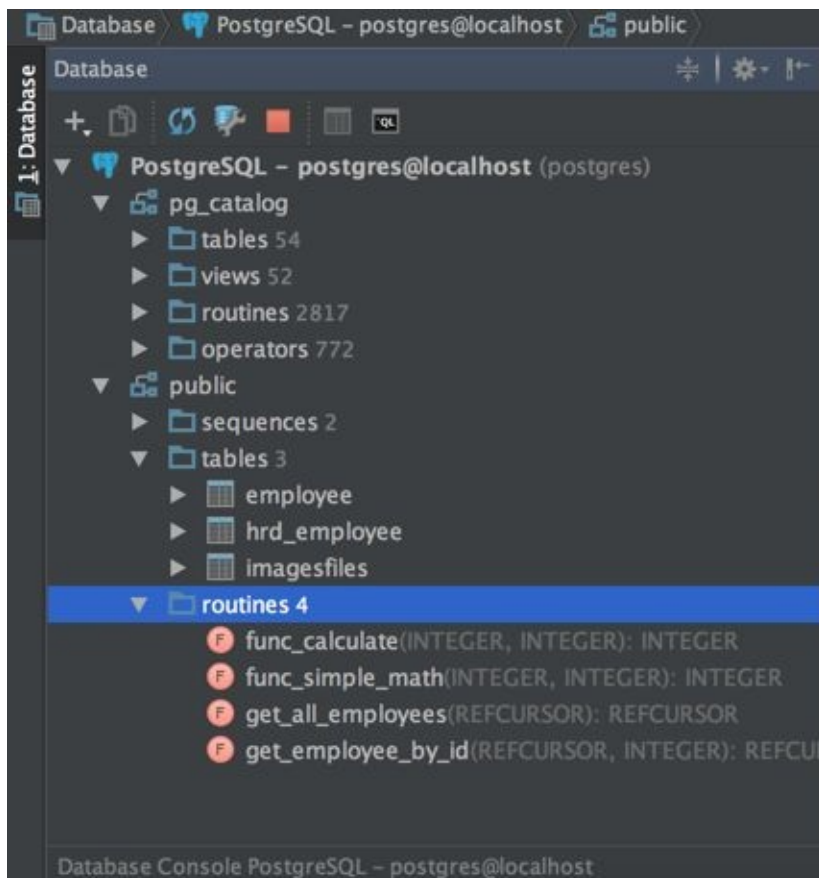
Firstly, we create Stored Procedure, call **get all employees**. Write this SQL statement.

```
DROP FUNCTION public.get_all_employees(refcursor);

CREATE OR REPLACE FUNCTION get_all_employees(ref refcursor) RETURNS refc
BEGIN
    OPEN ref FOR SELECT id,name,department,level,created FROM hrd_employee
    RETURN ref;
END;
$$ LANGUAGE plpgsql;
```

This stored procedure call a query to retrieve data on `hrd_employee` table. It returns a cursor.

Run this script to PostgreSQL database. If success, you can see this Stored Procedure on database.



Now we create new function, called **getAllEmployess()** in SPDemo.php and write this code.

```
<?php

include('dbinfo.php');

class SPDemo {

    private $msg = "";

    public function getMsg()
    {
        return $this->msg;
    }

    public function getAllEmployees() {
        $data = array();
        try {

            global $connString;

            $conn = pg_connect($connString);
            if( $conn === false )
            {
                $this->msg = "Unable to connect PostgreSQL Server.";
                return $data;
            }
        }
    }
}
```

```

        $cursor_name = "crp";
        pg_query($conn, "BEGIN");
        $query = "select * from get_all_employees($1);";

        $params = array(&$cursor_name);
        $res = pg_query_params($conn, $query, $params);
        if($res === false)
        {
            $this->msg = "Error in executing query.";
            return $data;
        }

        $res = pg_query($conn, "FETCH ALL FROM ".$cursor_name.";");
        if($res === false)
        {
            $this->msg = "Error in executing query.";
            return $data;
        }
        while($row = pg_fetch_row($res))
        {
            array_push($data, array("id"=>$row[0], "name"=>$row[1],
                                    "department"=>$row[2], "level"=>$row[3], "created"=>$r

        }

        pg_query($conn, "COMMIT");
        pg_close($conn);

    } catch (Exception $e) {
        $this->msg = $e->getMessage();
    }

    return $data;
}
}

```

Save this code.

Now we call function **getAllEmployees()**. Create file PHP **testsp.php** and write this code:

```

<?php

include( 'SPDemo.php' );

// call stored procedure
$obj = new SPDemo();
$data = $obj->getAllEmployees();
if($data) {
    foreach($data as $item) {
        $content = "";
    }
}

```

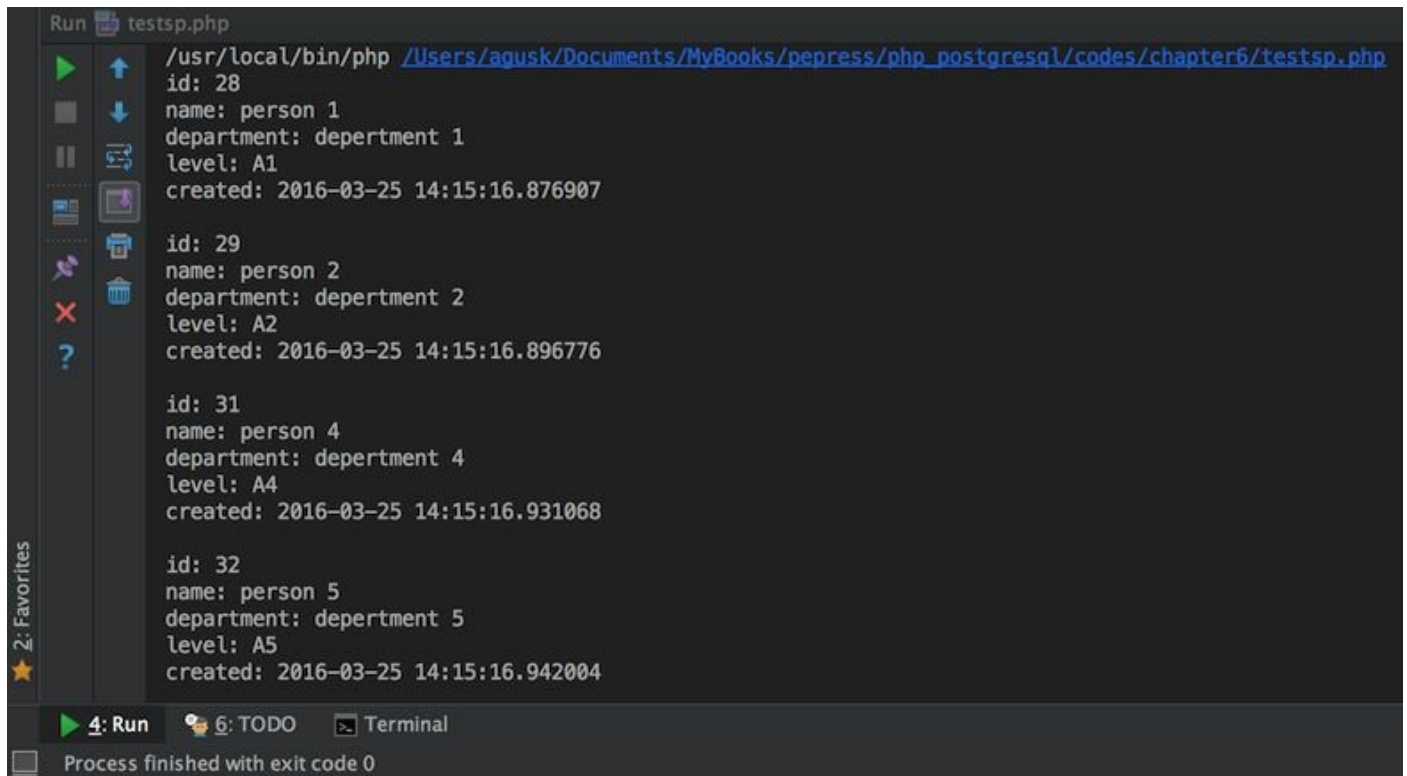
```

        foreach($item as $key => $value) {
            $content = $content . $key . ":" . strval($value) . "\n";
        }
        echo $content . "\n";
    }
}

```

Save this code.

You can execute file **testsp.php**. Here is a sample output.



Run testsp.php

```

/usr/local/bin/php /Users/agusk/Documents/MyBooks/pepress/php_postgresql/codes/chapter6/testsp.php
id: 28
name: person 1
department: department 1
level: A1
created: 2016-03-25 14:15:16.876907

id: 29
name: person 2
department: department 2
level: A2
created: 2016-03-25 14:15:16.896776

id: 31
name: person 4
department: department 4
level: A4
created: 2016-03-25 14:15:16.931068

id: 32
name: person 5
department: department 5
level: A5
created: 2016-03-25 14:15:16.942004

```

4: Run 6: TODO Terminal

Process finished with exit code 0

6.3 Calling Stored Procedure with Parameter

On previous section, we call Stored Procedure without parameter. Now we will call Stored Procedure with parameter. We create a Stored Procedure, called **get_employee_by_id**. Then write this script.

```
DROP FUNCTION public.get_employee_by_id(refcursor,INT);

CREATE OR REPLACE FUNCTION get_employee_by_id(ref refcursor,empId INT) R
BEGIN
    OPEN ref FOR SELECT id,name,department,level,created FROM hrd_employee
    RETURN ref;
END;
$$ LANGUAGE plpgsql;
```

Run this script to postgres database on PostgreSQL.

On PHP file, **SPDemo.php**, we create a function **getEmployeesById()** to call stored procedure and write this code.

```
<?php

include('dbinfo.php');

class SPDemo {

    private $msg = "";

    public function getMsg()
    {
        return $this->msg;
    }

    public function getEmployeesById($id) {
        $data = array();
        try {

            global $connString;

            $conn = pg_connect($connString);
            if( $conn === false )
            {
                $this->msg = "Unable to connect PostgreSQL Server.";
                return $data;
            }

            $cursor_name = "gtm";
            pg_query($conn,"BEGIN");
            $query = "select * from get_employee_by_id($1,$2)";

            $params = array(&$cursor_name,&$id);
```

```

        $res = pg_query_params($conn,$query,$params);
        if($res === false)
        {
            $this->msg = "Error in executing query.";
            return $data;
        }
        $res = pg_query($conn, "FETCH ALL FROM ".$cursor_name.";");
        if($res === false)
        {
            $this->msg = "Error in executing query.";
            return $data;
        }
        while($row = pg_fetch_row($res))
        {
            array_push($data, array("id"=>$row[0], "name"=>$row[1],
                                    "department"=>$row[2], "level"=>$row[3], "created"=>$r

        }

        pg_query($conn, "COMMIT");
        pg_close($conn);

    } catch (Exception $e) {
        $this->msg = $e->getMessage();
    }

    return $data;
}
}

```

Now we call function **getEmployeesById()**. Open file PHP testsp.php and add this code:

```

<?php

include('SPDemo.php');

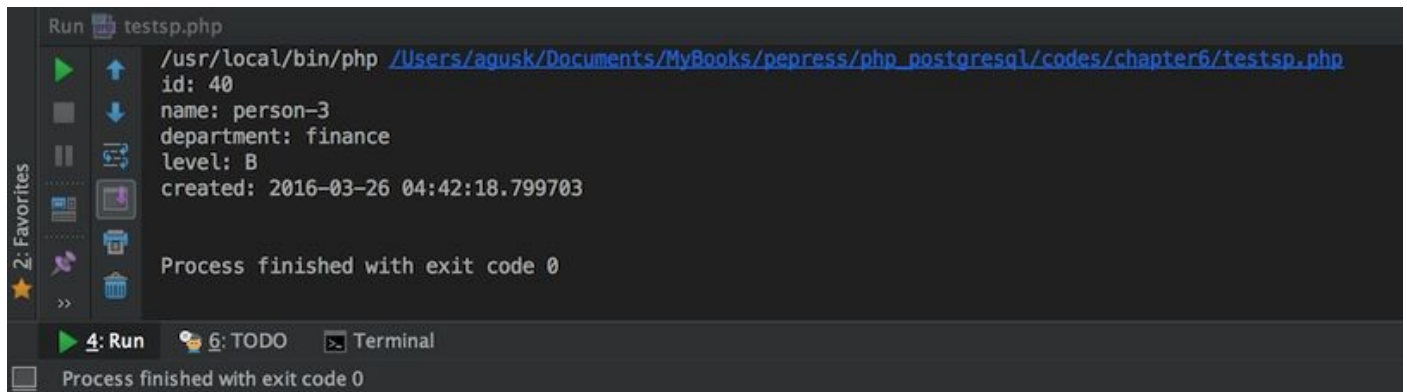
$obj = new SPDemo();

// call stored procedure with parameter
$data = $obj->getEmployeesById(40);
if($data) {
    foreach($data as $item) {
        $content = "";
        foreach($item as $key => $value) {
            $content = $content . $key . ": " . strval($value) . "\n";
        }
        echo $content . "\n";
    }
}
}

```

We pass Id=40 to function. You can change this value based on your data.

Save this code and run it. Here is a sample output.



```
Run testsp.php
/usr/local/bin/php /Users/agusk/Documents/MyBooks/pepress/php_postgresql/codes/chapter6/testsp.php
id: 40
name: person-3
department: finance
level: B
created: 2016-03-26 04:42:18.799703

Process finished with exit code 0
```

4: Run 6: TODO Terminal

Process finished with exit code 0

6.4 Calling Stored Procedure with Returning Values

In this section, we create a stored procedure which returns value. For instance, we have stored procedure as follows.

```
DROP FUNCTION public.func_simple_math(INT,INT);

CREATE OR REPLACE FUNCTION public.func_simple_math(var_a INT, var_b INT)
    RETURNS INT
AS
$BODY$
DECLARE
    c INT;
BEGIN
    c = var_a + var_b;
    RETURN c;
END;
$BODY$
LANGUAGE plpgsql VOLATILE;
```

Run this script on PostgreSQL.

Now we add **perform_math()** in SPDemo.php file to call our stored procedure.

```
<?php

public function perform_math($var_a,$var_b) {
    $ret = -1;
    try {

        global $connString;

        $conn = pg_connect($connString);
        if( $conn === false )
        {
            $this->msg = "Unable to connect PostgreSQL Server.";
            return $ret;
        }

        $query = "select func_simple_math($1,$2)";

        $params = array(&$var_a,&$var_b);
        $res = pg_query_params($conn,$query,$params);
        if($res === false)
        {
            $this->msg = "Error in executing query.";
            return $ret;
        }
        if($row = pg_fetch_row($res))
        {
            $ret = $row[0];
        }
    }
}
```

```

    }

    pg_close($conn);

} catch (Exception $e) {
    $this->msg = $e->getMessage();
}

return $ret;
}

```

We use SELECT to execute a stored procedure.

Furthermore, we work on testsp.php to call **perform_math()** function.

```

<?php

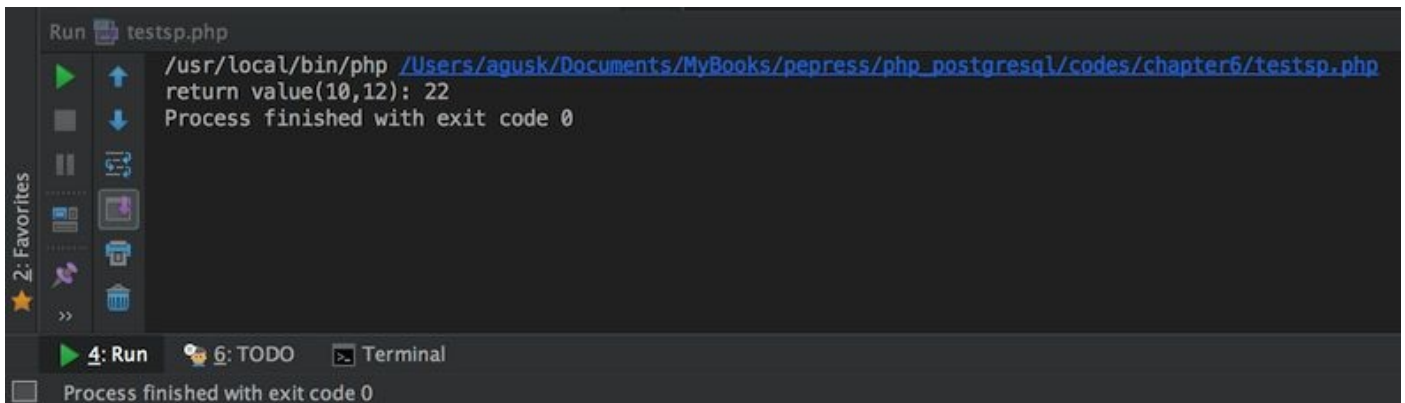
include('SPDemo.php');

// call stored procedure
$obj = new SPDemo();

///// call stored procedure with input parameter and return value
$val = $obj->perform_math(10,12);
echo "return value(10,12): " . $val;

```

Program output:



The screenshot shows an IDE window titled 'Run testsp.php'. The terminal output displays the command path, the execution of the script, and the result: 'return value(10,12): 22'. Below the output, it states 'Process finished with exit code 0'. The IDE interface includes a sidebar with 'Favorites' and a bottom status bar showing '4: Run', '6: TODO', and 'Terminal' tabs.

```

Run testsp.php
/usr/local/bin/php /Users/agusk/Documents/MyBooks/pepress/php_postgresql/codes/chapter6/testsp.php
return value(10,12): 22
Process finished with exit code 0

```


6.5 Calling Stored Procedure with Input and Output Parameters

We also can call Stored Procedure with input and output parameters. Firstly, we create a Stored Procedure, called **func_compute**. Write this code.

```
DROP FUNCTION public.func_compute(INT,OUT INT);

CREATE OR REPLACE FUNCTION public.func_compute(var_a INT, OUT val_ret INT)
    RETURNS INT
AS
$BODY$
BEGIN
    val_ret = var_a * 3;
END;
$BODY$
LANGUAGE plpgsql VOLATILE;
```

You can see we define **val_ret** parameter as **OUT**.

Back to PHP file, **SPDemo.php**, we create a function **compute()** and write this code.

```
<?php

public function compute($var_a) {
    $ret = -1;
    try {

        global $connString;

        $conn = pg_connect($connString);
        if( $conn === false )
        {
            $this->msg = "Unable to connect PostgreSQL Server.";
            return $ret;
        }

        $query = "select val_ret from func_compute($1);";

        $params = array(&$var_a);
        $res = pg_query_params($conn,$query,$params);
        if($res === false)
        {
            $this->msg = "Error in executing query.";
            return $ret;
        }
        if($row = pg_fetch_row($res))
        {
            $ret = $row[0];
        }
    }
}
```

```

        pg_close($conn);

    } catch (Exception $e) {
        $this->msg = $e->getMessage();
    }

    return $ret;
}

```

Now we test this function on **testsp.php**. Add this code.

```

<?php

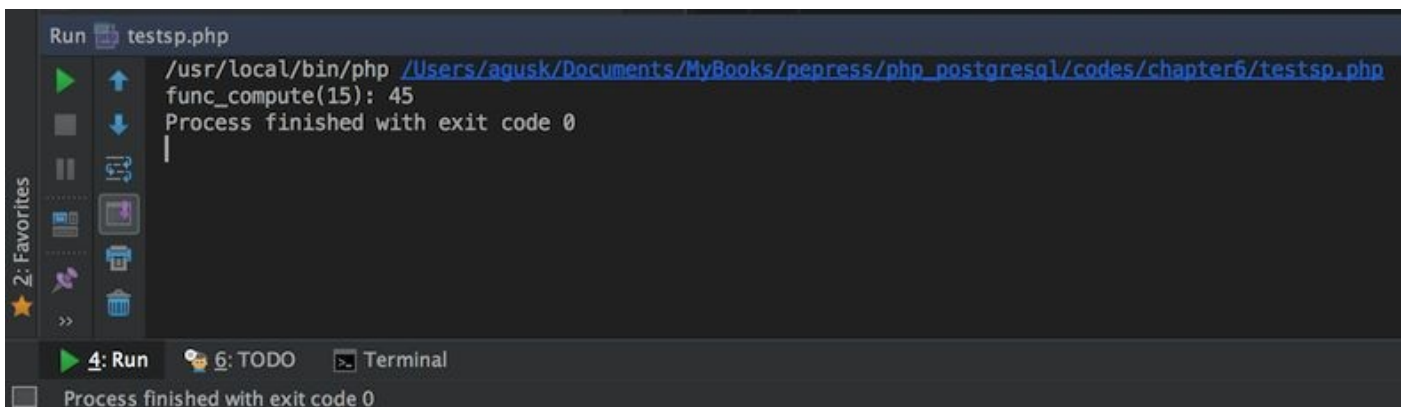
include('SPDemo.php');

// call stored procedure
$obj = new SPDemo();

///// call stored procedure with input and output parameters
$val = $obj->compute(15);
echo "func_compute(15): " . $val;

```

Save this code and run **testsp.php**. The following is a sample output.



The screenshot shows an IDE window titled 'Run testsp.php'. The command executed is `/usr/local/bin/php /Users/agusk/Documents/MyBooks/pepress/php_postgresql/codes/chapter6/testsp.php`. The output displayed is `func_compute(15): 45` followed by `Process finished with exit code 0`. The IDE interface includes a 'Favorites' sidebar on the left and a bottom status bar showing '4: Run', '6: TODO', and 'Terminal'.

Source Code

You can download source code for this book on

<http://www.aguskurniawan.net/book/phppostgres261.zip>

Contact

If you have question related to this book, please contact me at aguskur@hotmail.com . My blog: <http://blog.aguskurniawan.net> .