В. Б. Пикулев

БАЗЫ ДАННЫХ

5. Сетевой доступ к базам данных

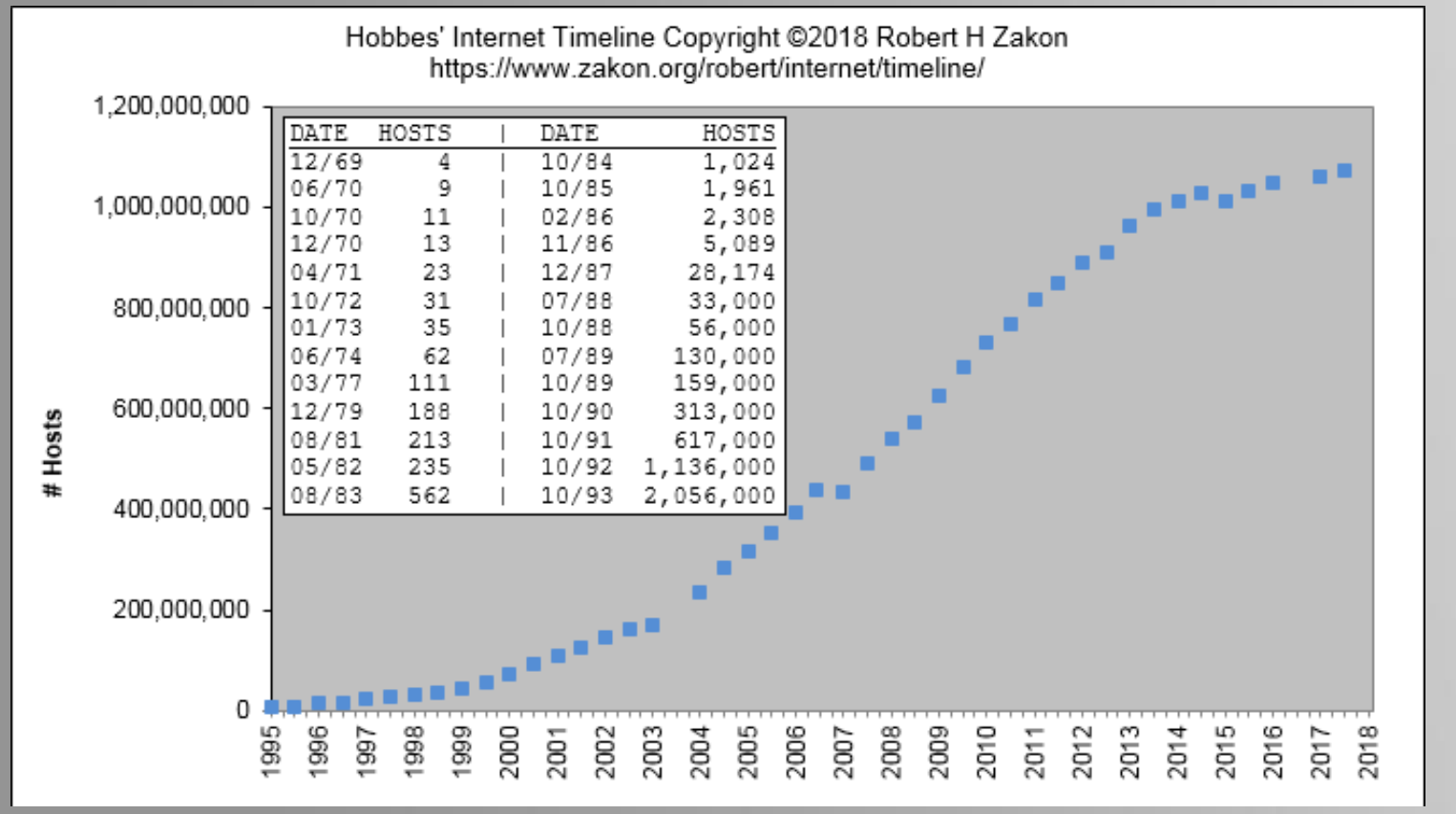
scilink.ru, 2020-21

Основные понятия

Базовым протоколом сети Интернет является протокол прикладного уровня **HTTP**.

Основным объектом манипуляции в HTTP является **ресурс**, на который указывает **URI** в запросе клиента.

Всемирная сеть **Интернет** — глобальное информационное пространство, основанное на физической сетевой инфраструктуре и протоколах открытой сети.



DATE	HOSTS	DATE	HOSTS
12/69	4	10/84	1,024
06/70	9	10/85	1,961
10/70	11	02/86	2,308
12/70	13	11/86	5,089
04/71	23	12/87	28,174
10/72	31	07/88	33,000
01/73	35	10/88	56,000
06/74	62	07/89	130,000
03/77	111	10/89	159,000
12/79	188	10/90	313,000
08/81	213	10/91	617,000
05/82	235	10/92	1,136,000
08/83	562	10/93	2,056,000

Программное обеспечение для работы с протоколом HTTP разделяется на три категории:

- Серверы – поставщики информации
- Клиенты – потребители информации
- Прокси-серверы – посредники доступа к информации

В протоколе HTTP отсутствует *сохранение состояния* – в нём не заложена возможность запоминать предыдущие запросы клиента и ответы сервера.

Протокол HTTP

Пример HTTP GET запроса от клиента:

URI

Версия протокола

GET /wiki/page HTTP/1.1
Host: ru.wikipedia.org
User-Agent: Mozilla/5.0 (X11; U; Linux i686; ru; rv:1.9b5) Gecko/2008050509 Firefox/3.0b5
Accept: text/html
Connection: close
пустая строка

Информация о клиенте

MIME-типы данных, которые может обработать клиент

Метод	*Описание*
GET	Запрос содержимого указанного ресурса или начало процесса
POST	Передача данных (в том числе загрузка файлов) на сервер
OPTIONS	Определение возможностей web-сервера

HyperText Transfer Protocol – протокол передачи гипертекста в Интернет. **HTTPS** - расширение протокола HTTP, поддерживающее шифрование

Стандартная схема HTTP-сеанса:

1. Установление TCP-соединения
2. Запрос клиента
3. Ответ сервера
4. Разрыв TCP-соединения.

Протокол HTTP

Пример ответа сервера:

HTTP/1.1 200 OK
Date: 01 Sep 2019 16:10:19 GMT
Server: py.scilink.ru
X-Powered-By: HHVM/3.18.6-dev
Last-Modified: Tue, 23 Aug 2019 18:49:34 GMT
Content-Language: ru
Content-Type: text/html; charset=UTF-8
Content-Length: 1234
Connection: close
пустая строка
[запрошенная страница в HTML]

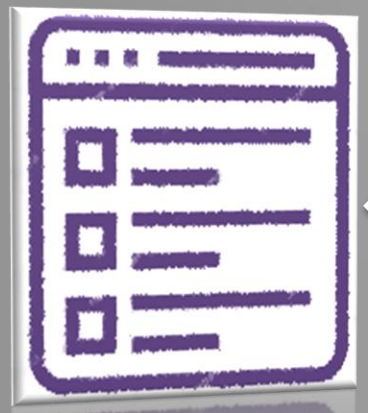
	Значение первой цифры
1	Сервер продолжает обработку запроса
2	Успешная обработка запроса клиента
3	Перенаправление запроса
4	Ошибка клиента
5	Ошибка сервера

Схема передачи данных

Web-сервер

Браузер

Web-страница



Код HTML
+ CSS
+ JavaScript
+ Расширения

- Java-апплет
- Adobe Flash
- SilverLight
- …

Код HTML
+ CSS
+
CGI-сценарий
(скрипт)
+
динамические
серверные
страницы

Java
Python
Perl
C
…

PHP
JSP
Razor
Ruby
…

База данных 1

База данных 2

Сервер баз данных

КЛИЕНТЫ

СЕРВЕРЫ

Веб-доступ к данным

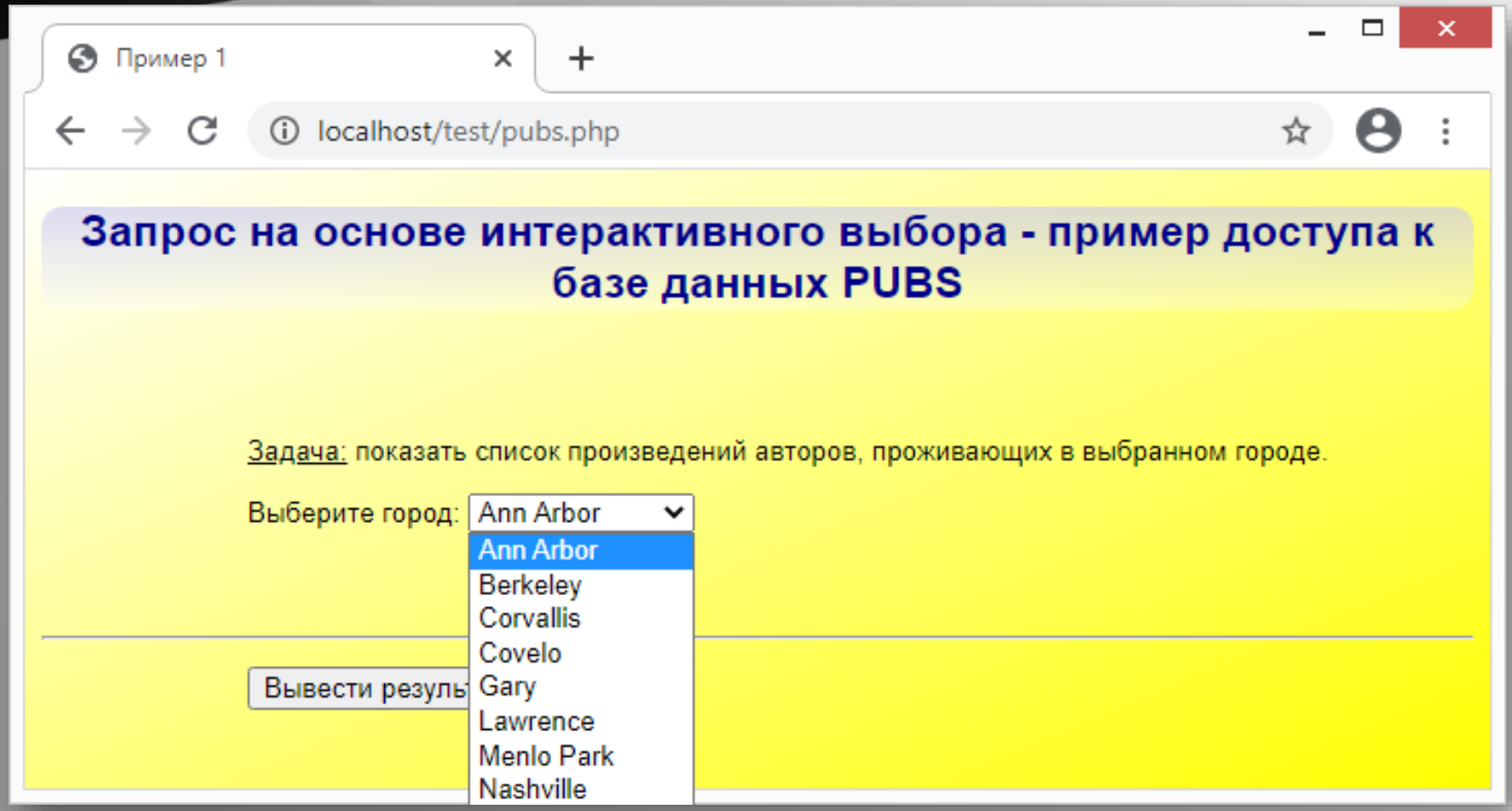
- Однократное или периодическое преобразование содержимого БД в статические документы. Используются программы-генераторы HTML-страниц
- Динамическое создание гипертекстовых документов на основе информации, запрашиваемой из базы данных, с помощью серверных языков программирования
- Динамическое создание гипертекстовых документов на основе данных из промежуточной базы данных, находящейся на web-сервере. Для взаимодействия между базами данных используется дополнительное программное обеспечение.

Классический пример
Объектный PHP + MySQL :

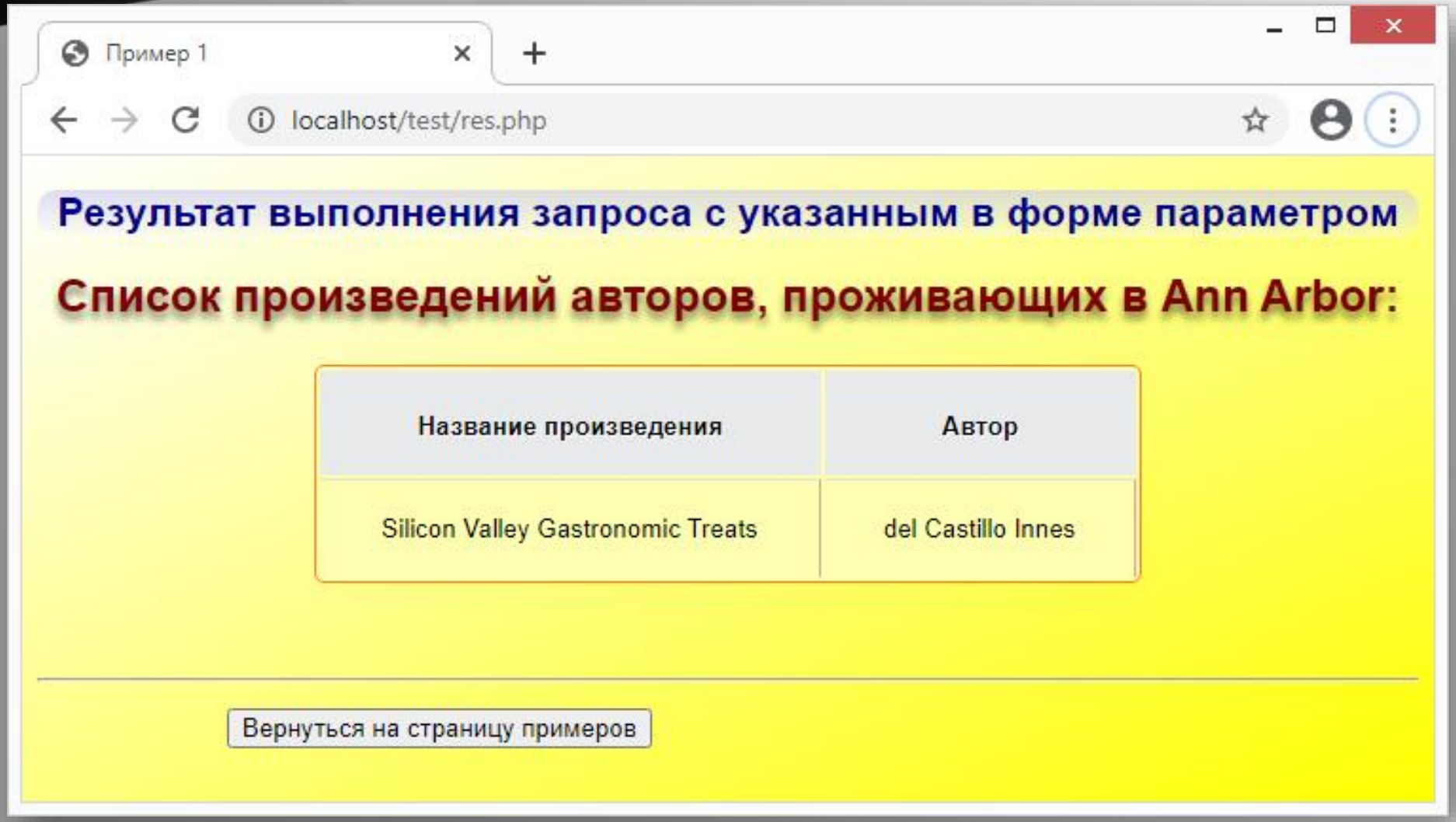
фрагмент

```php
<?php
  $conn = new mysqli('localhost', 'pikulev', 'password', 'db_pubs');
  if($conn->connect_error) die ($conn->connect_error);
  $str = "SELECT * FROM TITLES";
  $result = $conn->query($str);
  if(!$result) die ($conn->error); // если фатальная ошибка
  $rows = $result->num_rows; // узнаём число строк
  for($j=0; $j<$rows; ++$j){
    $result->data_seek($j);
    $row = $result->fetch_array(MYSQLI_ASSOC);
    // возвращает запись как одномерный ассоциативный массив
    echo 'Название: '.$row['title'].'<br>';
    echo 'Год публикации: '.$row['yearpub'].'<br>';
    echo 'ISBN: '.$row['ISBN'].'<br>';
  }
  $result->close();
  $conn->close();
?>
```

проект Pubs: выбор условия



проект Pubs: вывод результата



проект Pubs: файл pubs.php

фрагмент

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

if( !$conn )
{
    echo "Ошибка соединения с БД! ";
    die( print_r( sqlsrv_errors(), true));
}

  $sql="SELECT DISTINCT city FROM Authors ORDER BY city";
  $result = sqlsrv_query($conn, $sql);
?>
<p> Выберите город: <select name="var_town" size="1">
<?php
  while ($row = sqlsrv_fetch_array($result, SQLSRV_FETCH_ASSOC)) {
    $town = $row ['city'];
    echo "<option value='$town'>$town</option>";
  }
?>
</select></p>
```

connect.php

```php
<?php
$connectionList = array
("Database"=>"pubs",
"Uid"=>"worker",
"PWD"=>"somepassword");

$conn = sqlsrv_connect(
"localhost",
$connectionList);
?>
```

проект Pubs:
файл res.php

фрагмент

```
<?php
    include('connect.php');
    $var_town = $_POST['var_town'];
?>
<p><table width=60% class='data_table'>
<tr><th>Название произведения</th><th>Автор</th></tr>
<?php
  $sql="SELECT t.title AS title, au.au_lname AS lname, au.au_fname AS fname
        FROM (titleauthor ta INNER JOIN titles t ON ta.title_id = t.title_id)
        INNER JOIN authors au ON ta.au_id = au.au_id
        WHERE au.city LIKE '$var_town' ORDER BY t.title";
  $result = sqlsrv_query($conn, $sql);
  while ($row = sqlsrv_fetch_array($result, SQLSRV_FETCH_ASSOC)) {
    $title = $row['title'];
    $name = $row['lname']." ".$row['fname'];
    echo "<tr><td>$title</td><td>$name</td></tr>";
  }
?>
</table></p><br/><br/>
```

проект Pubs для PHP 5

```
res.php                [----] 15 L:[  1+ 0   1/ 34] *(15  /1292b) 0010 0x00A  UTF-8
<!DOCTYPE HTML>
<HTML>
<HEAD>
<meta charset="utf-8">
<link href="artistyle.css" rel="stylesheet" type="text/css" />
<TITLE>Пример 1</TITLE>
</HEAD>
<BODY>
<h2>Результат выполнения запроса с указанным в форме параметром</h2>
<?php
    include('connect.php');
    $var_town = $_POST['var_town'];
    echo "<h1>Список произведений авторов, проживающих в <b>$var_town</b>:</h1>";
?>
<p align='center'><table width=60% class='data_table'>
<tr><th>Название произведения</th><th>Автор</th></tr>
<?php
  $sql="SELECT t.title AS title, au.au_lname AS lname, au.au_fname AS fname FROM (title
  $result = mssql_query($sql);
    while ($row = mssql_fetch_array($result)) {
    $title = $row['title'];
    $name = $row['lname']." ".$row['fname'];
    echo "<tr><td>$title</td><td>$name</td></tr>";
  }
?>
</table></p><br/><br/>
<hr>
  <form name="form2" method="post" action="../index.html">
    <p><input type="submit" name="submit" value="Вернуться на страницу примеров"></p>
  </form>

</BODY>
</HTML>
```

PDO (PHP Data Object) – современный интерфейс для доступа к базам данных из PHP. Переход к PDO связан со стремлением создать сходный языковой интерфейс для различных типов баз данных.

PHP + MS SQL Server

использование PDO

```
<!DOCTYPE html>
<head>
  <title>Соединение с MSSQL</title>
</head>
<body>
  <h1>Интерфейс PDO для связи с MS SQL Server</h1>
  <?php
$dsn = 'dblib:host=172.20.195.200;dbname=Museum';
//источник данных для каждой СУБД оформляется по-своему
$user = 'pikulev'; $password = 'somepassword';
try {
  $db = new PDO($dsn, $user, $password); //создаётся объект PDO
  $sql = "SELECT fname, ftype, flength FROM museum";
  $q = $db->prepare($sql);
  $q->execute(); // так удобно делать для параметризованных запросов
  while($data = $q->fetch(PDO::FETCH_ASSOC)){
    print $data['fname'].'   '.$data['ftype'].'   '.$data['flength'].'<br>';
  }
}
catch (PDOException $e) {
  echo 'Ошибка соединения с БД!'.$e->getMessage();
}

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

проект Demo

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8"> <link rel="stylesheet" href= "web.css"> <title>Тест</title>
</head>
<body>
   <h1>SELECT * FROM aircrafts</h1>
   <?php
$conn = pg_connect('host=localhost port=5432 dbname=demo user=work password=x') or die;
$query = pg_query('SELECT * FROM aircrafts') or die;
echo '<table><tr><th>Код</th><th>Модель</th><th>Дальность, км</th><tr>';
while ($row = pg_fetch_array($query)) {
  echo '<tr><td class="t1">'.$row[0].PHP_EOL.'</td><td class="t2">'.$row[1].PHP_EOL.
       '</td><td class="t3">'.$row[2].PHP_EOL.'</td></tr>';
}
echo '</table>';
pg_free_result($query);
pg_close($conn);
    ?>
</body>
</html>
```

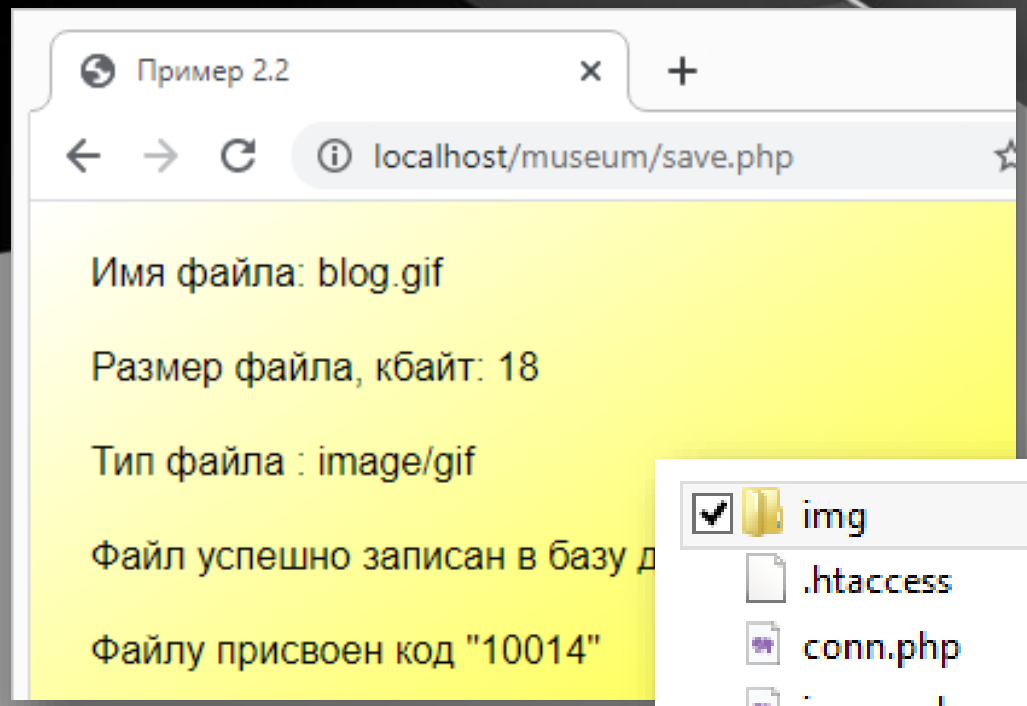
проект Demo

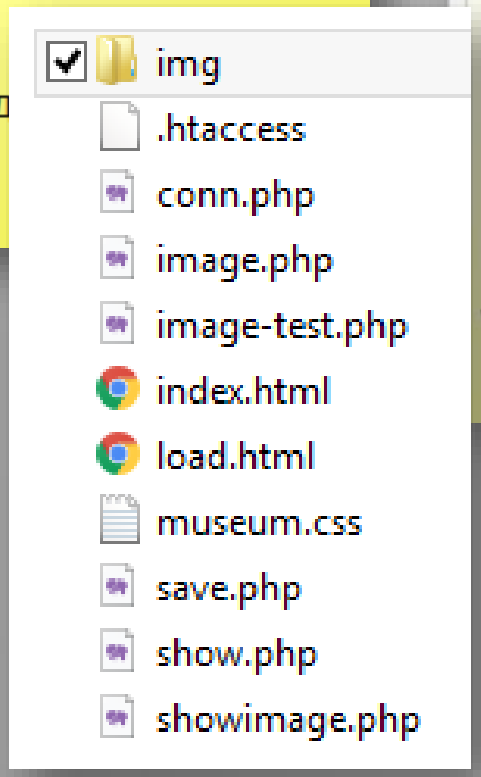
PHP + PostgreSQL

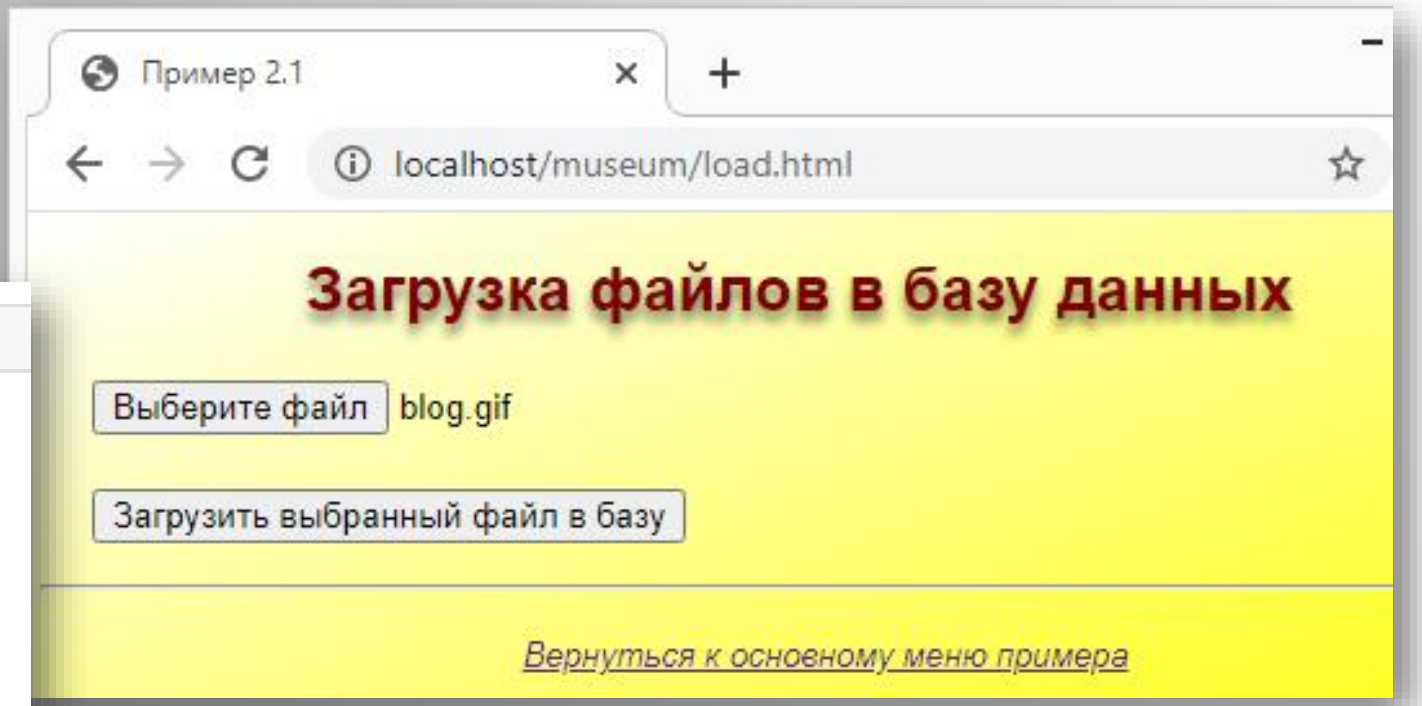
SELECT * FROM aircrafts

Код	*Модель*	*Дальность, км*
773	**Боинг 777-300**	11100
763	**Боинг 767-300**	7900
SU9	**Сухой Суперджет-100**	3000
320	**Аэробус A320-200**	5700
321	**Аэробус A321-200**	5600
319	**Аэробус A319-100**	6700
733	**Боинг 737-300**	4200
CN1	**Сессна 208 Караван**	1200
CR2	**Бомбардье CRJ-200**	2700

проект Museum: загрузка изображений





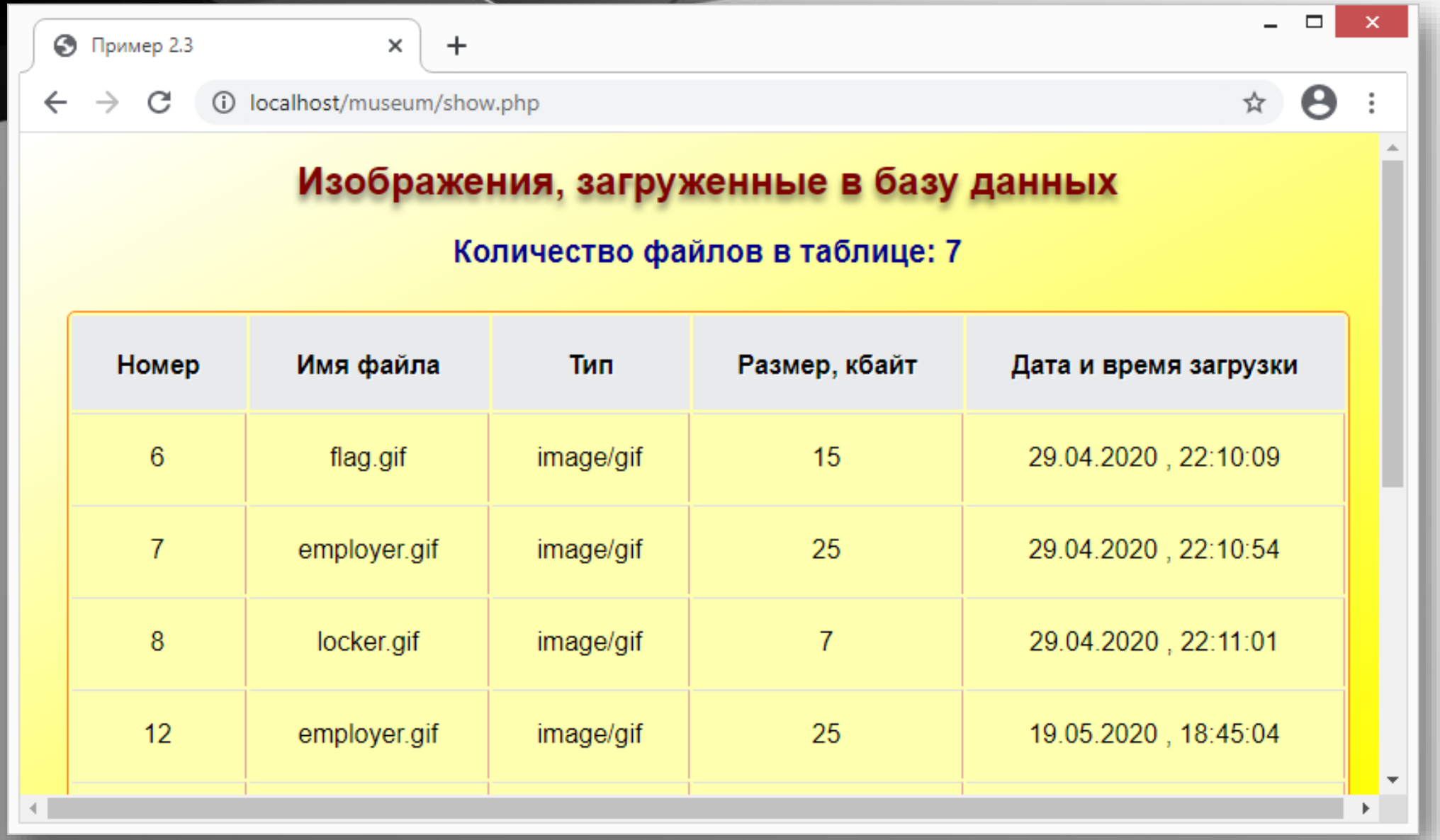


проект Museum: загрузка изображений

фрагмент файла
save.php


```php
$connectionStr = 'host=localhost port=5432 dbname=museum user=work password=somepassword';

$imageBody = file_get_contents($_FILES['myfile']['tmp_name']); // читаем файл из временного каталога
$edata = pg_escape_bytea($imageBody);
// соединяемся с базой данных
$conn = pg_connect($connectionStr) or die("<p>Ошибка подключения к серверу баз данных!\n</p>");
$sqlString = "INSERT INTO museum (fname, ftype, img, flength)
                VALUES ('$filename', '$ftype', '{$edata}', $flength)";
$result = pg_query($sqlString);
if ($result) {
   echo '<p>Файл успешно записан в базу данных!</p>';
   $sqlString = "SELECT id FROM museum WHERE fname='$filename' ORDER BY wdate DESC LIMIT 1";
   $result = pg_query($conn, $sqlString);
   $row = pg_fetch_array($result);
   echo '<p>Файлу присвоен код "' . $row['id'] . '"</p>';
} else {
   echo '<p>Произошла ошибка при записи файла в базу данных!</p>';
}
pg_free_result($result);
pg_close($conn);
```

проект Museum: таблица на экране



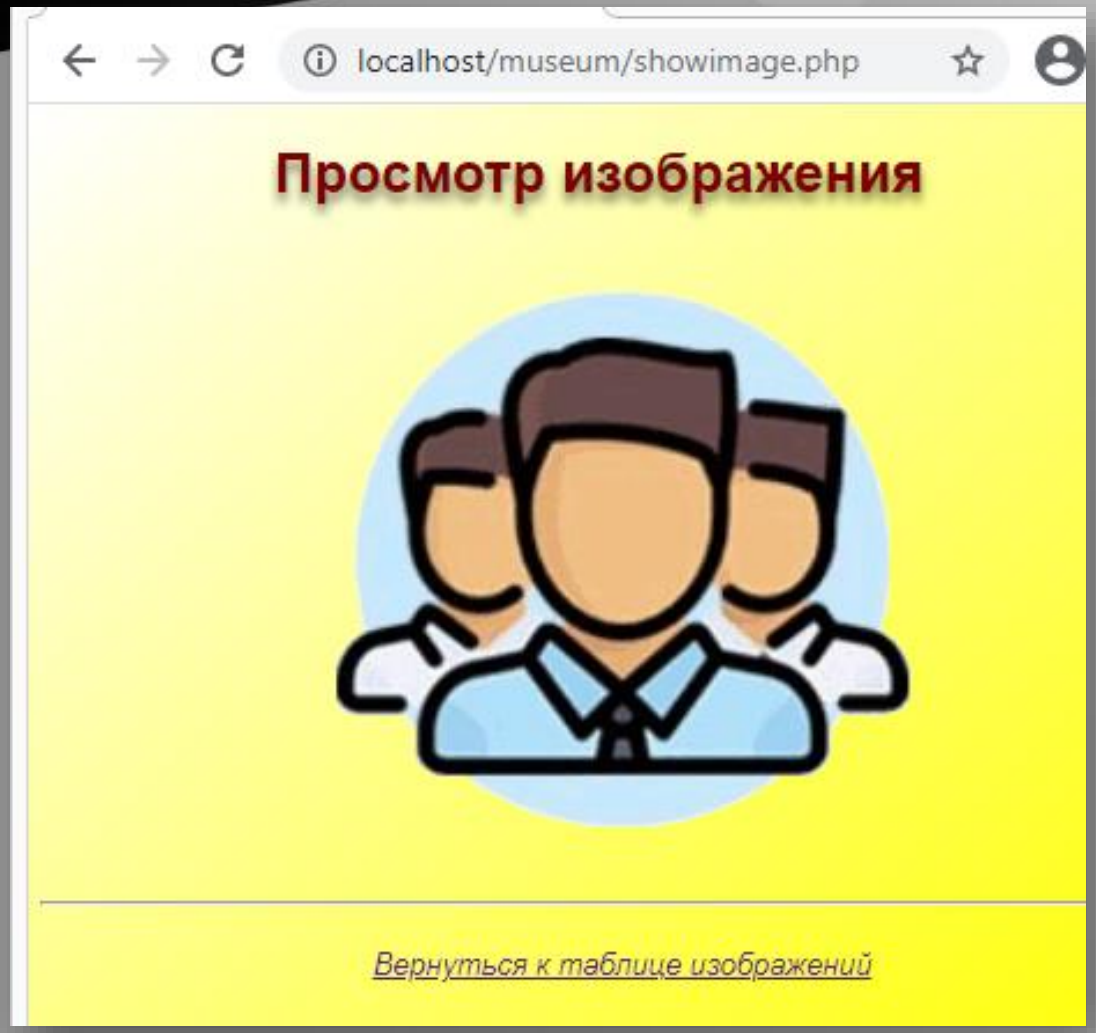
проект Museum: таблица на экране

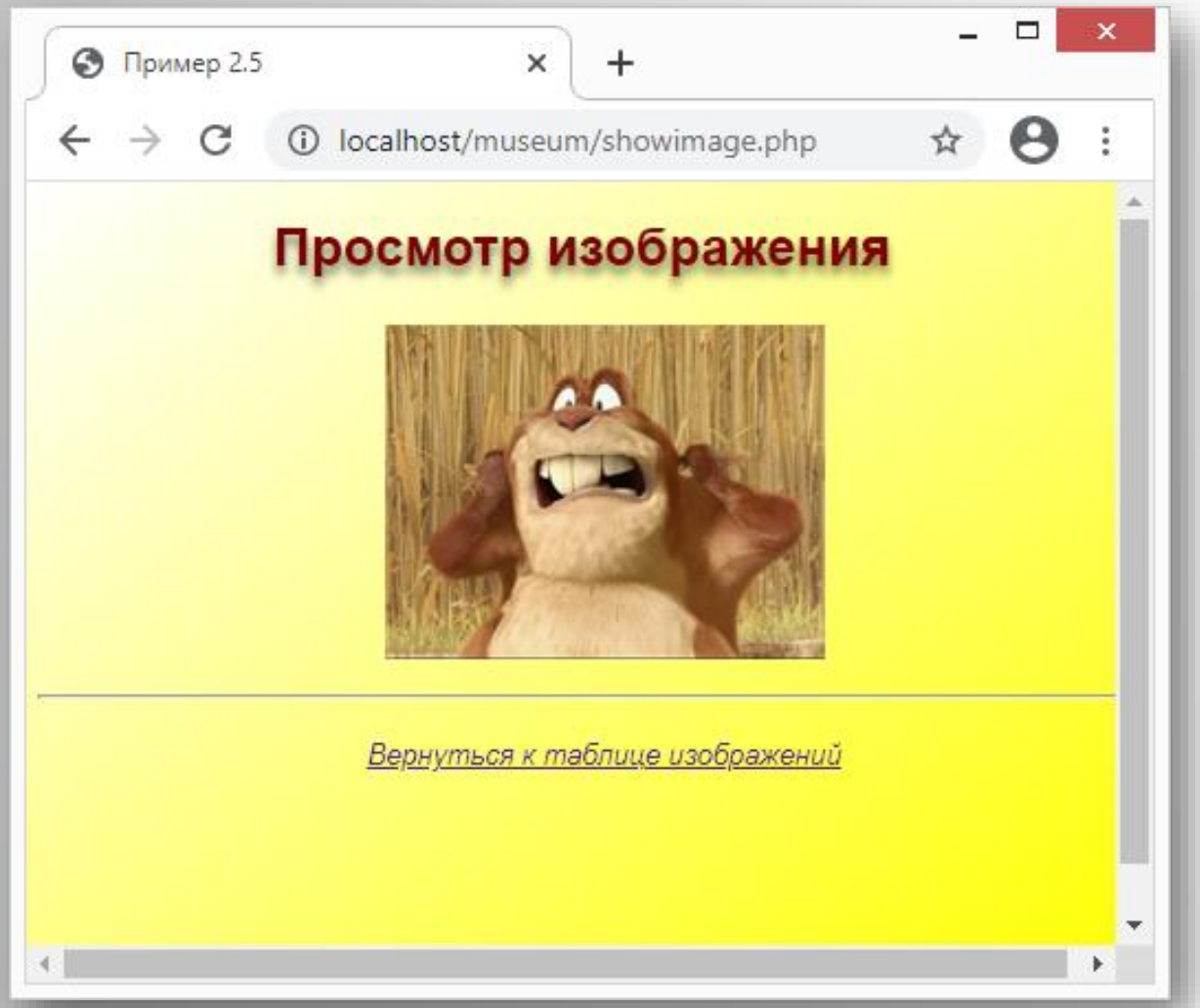
фрагмент

PHP + PostgreSQL

```php
<?php
  $conn = pg_connect($connectionStr) or die("<p>Ошибка подключения к серверу баз данных!\n</p>");
  $sqlString = "SELECT id, fname, ftype, flength, wdate, comment FROM museum ORDER BY id";
  $result = pg_query($sqlString);
  $count = pg_num_rows($result);
  echo '<h3>Количество файлов в таблице: '.$count.'</h3>';
  if ($count>0) {
    echo '<div><table><tr><th>Номер</th><th>Имя файла</th><th>Тип</th><th>Размер, кбайт</th>
        <th>Дата и время загрузки</th></tr>';
    while ($row = pg_fetch_array($result)) {
            echo '<tr><td>'.$row['id'].'</td>';
            echo '<td>'.$row['fname'].'</td>';
            echo '<td>'.$row['ftype'].'</td>';
            echo '<td>'.$row['flength'].'</td>';
            echo '<td>'.$row['wdate'].'</td></tr>';
    }
    echo '</table></div><br/><hr>';
  }
  pg_free_result($result);
  pg_close($conn);
?>
```

проект Museum: вывод изображения на экран





проект Museum: вывод изображения (1)


```
 <!DOCTYPE html>
<!--
  Внешняя форма для просмотра рисунка, id передаётся коду, генерирующему изображение
-->
<html>
    <head>
        <meta charset="utf-8">
        <link href="museum.css" rel="stylesheet" type="text/css"/>
    </head>
    <body>
        <h1>Просмотр изображения</h1>
        <p class="info"><?php
        $sel_id = $_POST['sel_id'];
        echo '<img src="image.php?sel_id='.$sel_id.'">';
        ?>
        </p><hr>
        <p class="info"><a href="show.php">Вернуться к таблице изображений</a></p>
    </body>
</html>
```

проект Museum: вывод изображения (2)

image.php


```php
<?php
    include_once 'conn.php';
    $sel_id = $_GET['sel_id'];
    $conn = pg_connect($connectionStr);
    $sqlString = "SELECT ftype, img FROM museum WHERE id=".$sel_id;
    $result = pg_query($sqlString);
    $row = pg_fetch_array($result);
    if ($row) {
        header('Content-type: '.$row['ftype']);
        echo pg_unescape_bytea($row['img']);
    } else {
        header('Content-type: image/jpeg');
        echo file_get_contents("img/broken.jpg"); // дежурное изображение
    }
?>
```

Java-сервлет – это Java-программа, которая выполняется на стороне сервера.

Использование CGI

Пример сервлета:
Java-код, в который внедрён синтаксис HTML

```
public void doGet(request, response)
{
PrintWriter out = response.getWriter();
String name =
    request.getParameter(uName);
out.println("<html><body>");
out.println("Username:" + name);
out.println("</body></html>");
}
```

Спецификация CGI (Common Gateway Interface) описывает формат и правила обмена данными между программным обеспечением web-сервера и запускаемой программой. Для инициирования CGI необходимо, чтобы в запрашиваемом URL был указан путь до запускаемой программы. Web-сервер исполняет эту программу, передает ей входные параметры и возвращает результаты ее работы, как результат обработки запроса, клиенту. CGI - программой может являться любая программа локальной операционной системы сервера - в двоичном виде или в виде программы для интерпретатора.

Java-сервлет

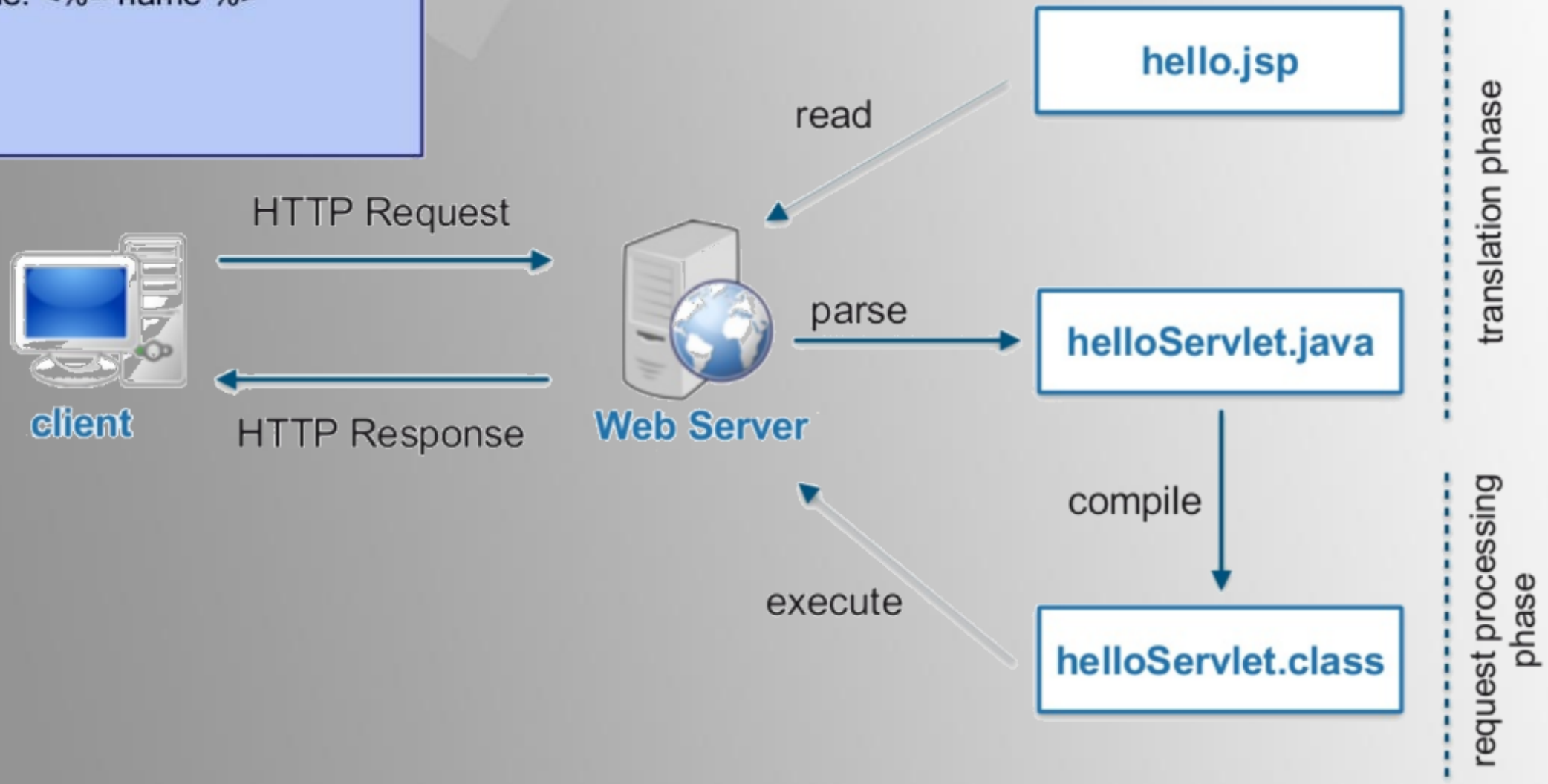
простой пример

```
@WebServlet(name = "Servlet1", urlPatterns = {"/Servlet1"})
public class Servlet1 extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.print("<h1>Я Самый Простой Сервлет!</h1>");
    }


    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        super.doPost(request, response);
    }
}
```

Динамические web-страницы

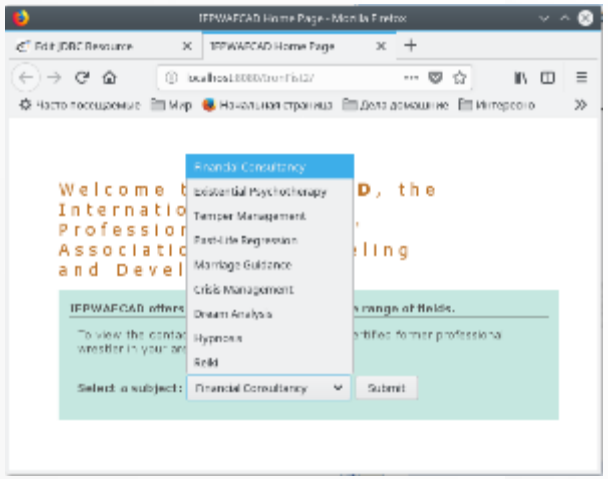
на языке Java

```
<html>
<body>
<% String name =
   request.getParameter(uName); %>

Username: <%= name %>

</body>
</html>
```



Пример : фрагмент

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql"%>
...
<body>
        <sql:query var="subjects" dataSource="jdbc/Iron2Data">
            SELECT subject_id, name FROM Subject
        </sql:query>
...
        <form action="response.jsp">
            <strong>Select a subject:</strong>
                <select name="subject_id">
                <c:forEach var="row" items="${subjects.rows}">
                    <option value="${row.subject_id}">${row.name}</option>
                </c:forEach>
            </select>
        <input type="submit" value="Submit" name="Submit" />
        </form>
...
</body>
...
```



Проблемы Интернет-доступа к базам данных

Код внутри динамической серверной страницы script.php:

```
$id = $_REQUEST['id'];
$res = mysqli_query("SELECT date, text
FROM news WHERE id_news = ".$id);
```

Типичный вид GET-запроса к этой странице:

```
http://example.org/script.php?id=15
```

Подмена GET-запроса:

```
http://example.org/script.php?id=-1
+OR+1=1
```

Изменённый запрос, который будет исполнять сервер баз данных:

```
SELECT date, text FROM news
WHERE id_news =-1 OR 1=1
```

Внедрение SQL-кода (**SQL injection**) — один из способов взлома сайтов и программ, работающих с базами данных, основанный на внедрении в запрос непредусмотренного разработчиками SQL-кода. Это может дать возможность атакующему выполнить произвольный запрос к базе данных (например, прочитать содержимое любых таблиц, удалить, изменить или добавить данные), получить возможность чтения и/или записи локальных файлов, и даже выполнения произвольных команд на атакуемом сервере.

Проще всего получить такую возможность хакер может, изменив необычным образом входные данные в web-формах, служащих источниками данных для динамических серверных страниц.

Проблемы Интернет-доступа к базам данных

SQL-иньекции

Продолжаем развивать атаку:

```
SELECT date, text FROM news WHERE id_news = -1
UNION SELECT username, password FROM admin
-- более чем два столбца в нужной таблице можно объединить

SELECT date, text FROM news WHERE id_news = 12;
INSERT INTO admin (username, password) VALUES ('HaCkEr', 'qwert')

SELECT date, text FROM news WHERE id_news = -1
UNION SELECT 1, LOAD_FILE('settings.php')
```

И дальше уже можно совсем не церемониться:

```
create table t(a varchar(500));
load data infile '/etc/passwd' into table t;
select a from t;
```

если поддерживаются мультизапросы

Ещё один
типичный пример

Проблемы Интернет-доступа к базам данных

SQL-иньекции

Способ передачи параметра в php-файл:

```
http://example.org/script.php?search_text=MyTestWord
```

Фрагмент этого php-файла:

```
$search_text = $_REQUEST['search_text'];
$res = mysqli_query("SELECT id_news, news_date, news_caption,
news_text, news_id_author FROM news WHERE news_caption
LIKE('%$search_text%')");
```

Введён текст в GET-запрос так, чтобы получился новый SQL-запрос:

```
SELECT id_news, news_date, news_caption, news_text, news_id_author
FROM news WHERE news_caption LIKE('%') and (news_id_author='1%')
```

http://example.org/script.php?id=1+ORDER+BY+100
ERROR 1054 (42S22): Unknown column '100' in 'order clause'

Можно начинать лечить!

?param=1/*%00*/union+select+1
?param=뻿 or 1=1;

Способы борьбы с проблемами

SQL-иньекции

- **Экранировка всех специальных символов (кавычек, апострофов прежде всего):**

```
addslashes($str);
mysql_real_escape_string($str);
```

- **Проверка типа вводимых параметров (если не строковые):**

```
settype($f, 'integer');
$id = (int) $_GET['id'];
is_numeric(n);
```

- **Усечение длины строки:**

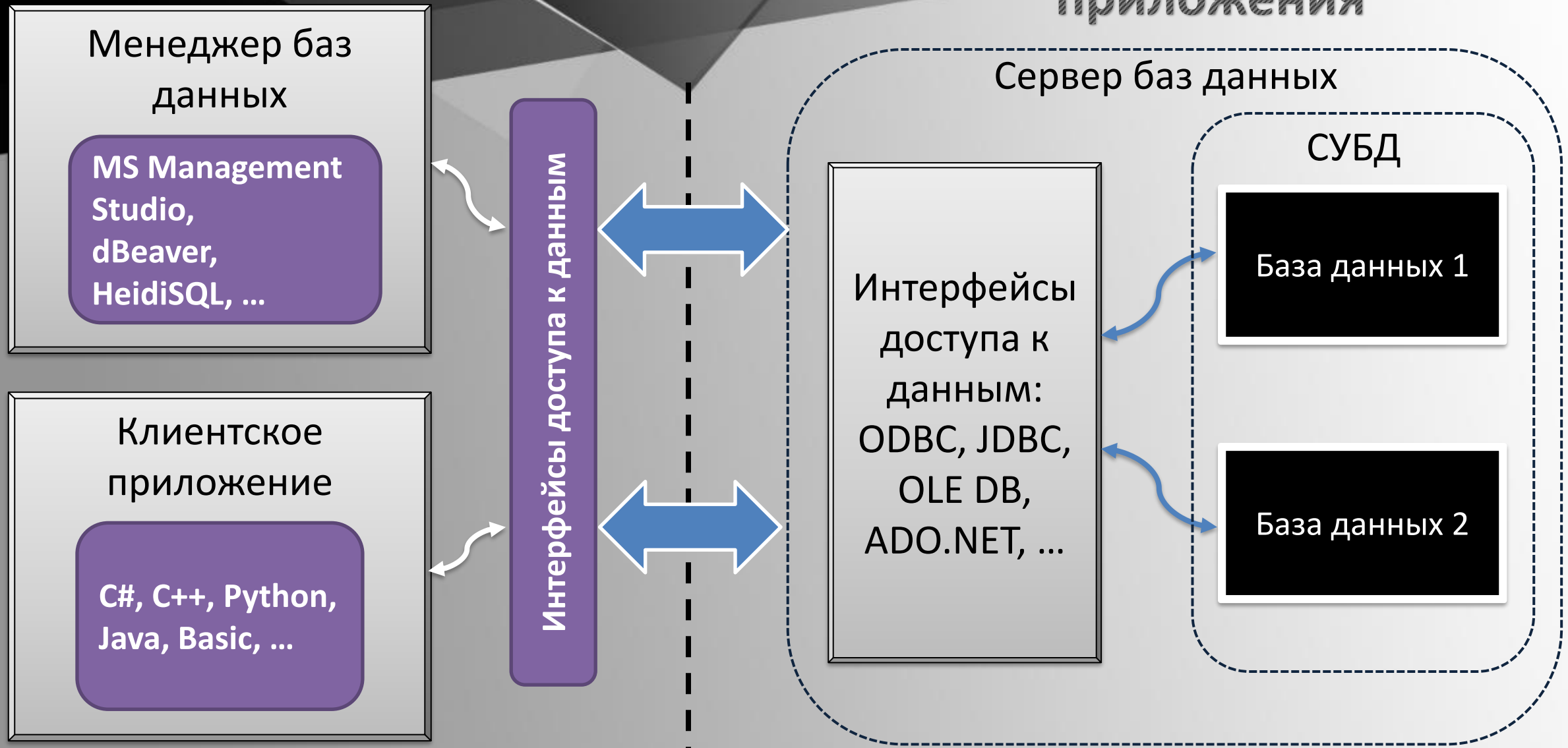
```
trim($str)
```

- **Использование параметризованных запросов и хранимых процедур с параметрами:**

```
$st = $db->prepare('update tbl set parameter = ? where id = ?');
$st->bind_param('si', $name, $id);
$st->execute();
```

- **Ограничение прав учётной записи, от имени которой выполняется доступ к базе данных**.

Клиент-серверные приложения



На Python
без web-сервера

demo.py

```
import psycopg2
conn = psycopg2.connect(
host='172.20.180.147', port='5432', database='demo',
user='demo', password='somepassword')
cur = conn.cursor()
cur.execute('SELECT * FROM aircrafts')
rows = cur.fetchall()
for row in rows:
    print(row[0],'\t| ',row[1],'\t| ',row[2])
conn.close()
```

```
PROBLEMS   OUTPUT   TERMINAL   DEBUG CONSOLE

C:\Users\Vitaly>python c:/Users/Vitaly/Documents/
773     |  Боинг 777-300          |  11100
763     |  Боинг 767-300          |  7900
SU9     |  Сухой Суперджет-100    |  3000
320     |  Аэробус A320-200       |  5700
321     |  Аэробус A321-200       |  5600
319     |  Аэробус A319-100       |  6700
733     |  Боинг 737-300          |  4200
CN1     |  Сессна 208 Караван     |  1200
CR2     |  Бомбардье CRJ-200      |  2700

C:\Users\Vitaly>
```

модуль SQLAlchemy

Первый путь – выполнение нативного SQL-запроса

```
statement = """INSERT INTO films (title, director, year)
VALUES ('Doctor Strange', 'Scott Derrickson', '2016')"""
conn.execute(statement)
```

Второй путь – построение запроса с помощью SQL Alchemy

```
statement = films.insert().values(title="Doctor Strange",
 director="Scott Derrickson", year="2016")
conn.execute(statement)
```

Третий путь – ORM SQL Alchemy

```
doctor_strange = Film("Doctor Strange",
"Scott Derrickson", "2016")
db_session.add(doctor_strange)
```

SQLAlchemy — это программная библиотека на языке Python для работы с реляционными СУБД. Её использование выгодно тогда, когда требуется синхронизация объектов Python и записей реляционной базы данных. SQLAlchemy позволяет описывать структуры баз данных и способы взаимодействия с ними на языке Python **без использования языка SQL**.

ORM (object-relational mapping) — технология сопоставления классов в объектно-ориентированном языке со структурами реляционной базы данных. При этом обычно создаётся «виртуальная объектная база данных».

Этап 1. Создание структуры базы данных

```
# упрощаем доступ к методам sqlalchemy

from sqlalchemy import create_engine
from sqlalchemy import Column, ForeignKey, Integer, String
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship
```

```
Base = declarative_base()

class Genre(Base):
    __tablename__ = 'genres'

    id = Column(Integer, primary_key=True)
    name = Column(String(250), nullable=False)

class Book(Base):
    __tablename__ = 'books'

    id = Column(Integer, primary_key=True)
    title = Column(String(250), nullable=False)
    author = Column(String(250), nullable=False)
    yearpub = Column(Integer)
    genre_id = Column(Integer, ForeignKey('genres.id'))
    genre = relationship("Genre")

ds = "postgresql://user:somepassword@scilink.ru:5432/alchemy"
db = create_engine(ds)
Base.metadata.create_all(db)
```

информации, указываемой при описании классов.

Внешний ключ между таблицами указывается с помощью метода *ForeignKey*, а связь между объектами создаётся с помощью функции *relationship*.

С помощью create_engine соединяемся с конкретной базой данных и проводим все необходимые транзакции

pg_mydb_show.py | pg_mydb_create.py

C: > Users > Vitaly > Documents > Education > Courses > Базы данных > Лекции > SQLAlchemy > pg_mydb_create.py > Genre

```python
class Genre(Base):
    __tablename__ = 'genres'

    id = Column(Integer, primary_key=True)
    name = Column(String(250), nullable=False)

    def __repr__(self):
        return " жанр: '{}'"\
            .format(self.name)

class Book(Base):
    __tablename__ = 'books'

    id = Column(Integer, primary_key=True)
    title = Column(String(250), nullable=False)
    author = Column(String(250), nullable=False)
    yearpub = Column(Integer)
    genre_id = Column(Integer, ForeignKey('genres.id'))
    genre = relationship("Genre")

    def __repr__(self):
        return " Книга: '{}', автор {}, {} года издания,{} "\
            .format(self.title, self.author, self.yearpub, self.genre)
```

ВЫХОДНЫЕ ДАННЫЕ | ТЕРМИНАЛ | КОНСОЛЬ ОТЛАДКИ | ПРОБЛЕМЫ

Python

```
None
 Книга: 'Техану', автор Ле Гуин У, 2013 года издания, жанр: 'Фэнтези'
```

Этап 2. Ввод данных в таблицы

pg_mydb_insert.py

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from pg_mydb_create import Book, Genre
```

```
ds = "postgresql://..."
db = create_engine(ds)

Session = sessionmaker(db)
s = Session()
# вводим жанры
s.add_all([
    Genre(name='Детская литература'),
    Genre(name='Научная литература'),
    Genre(name='Классические произведения'),
    Genre(name='Фэнтези'),
])

# вводим книги

s.add( Book(
    title="Сказка о рыбаке и рыбке",
    author="Пушкин А",
    yearpub=1981,
    genre_id=1)
)
```

Сессии формируют основу для обмена данными с базой

```
da
    'yearpub' : 2013,
    'genre_id': 4
    },{
    'title'   : "Властелин колец",
    'author'  : "Толкин Д",
    'yearpub' : 2001,
    'genre_id': 4
})

for d in data:
    book = Book(**d)
    s.add(book)

s.commit()
s.close()
```

"Двойная звёздочка" распаковывает словарь в набор отдельных аргументов.

*Только после выполнения **commit()** произойдёт запись данных в базу (транзакция)*

Этап 3.1 Запросы на просмотр базы данных

pg_mydb_show.py

SQL Alchemy

```
from sqlalchemy import create_engine, and_, or_
fr
fr
```

```
Сделано.
Просматриваем отдельные записи
 Книга: 'Сказка о рыбаке и рыбке', автор Пушкин А, 1981 года издания, жанр: 'Детская литература'
 Книга: 'Техану', автор Ле Гуин У, 2013 года издания, жанр: 'Фэнтези'
None
 Книга: 'Техану', автор Ле Гуин У, 2013 года издания, жанр: 'Фэнтези'
```

```
ds
db

Session = sessionmaker(db)
s = Session()

print('Просматриваем отдельные записи')

print(s.query(Book).first())
print(s.query(Book).filter_by(title='Техану').first())
print(s.query(Book).filter(Book.author.like('%Струг%')).first()
)
r = s.query(Book).filter(
        or_(Book.yearpub<=1990, Book.yearpub>2010)
).order_by(Book.title.desc()).limit(1).first()
print(r)
```

Запросы создаются посредством использования функции ***query()*** для сессии. Эта функция берет переменное число аргументов, которыми может быть любая комбинация классов и дескрипторов, созданных с помощью классов. Результаты запроса возвращаются в виде кортежей.

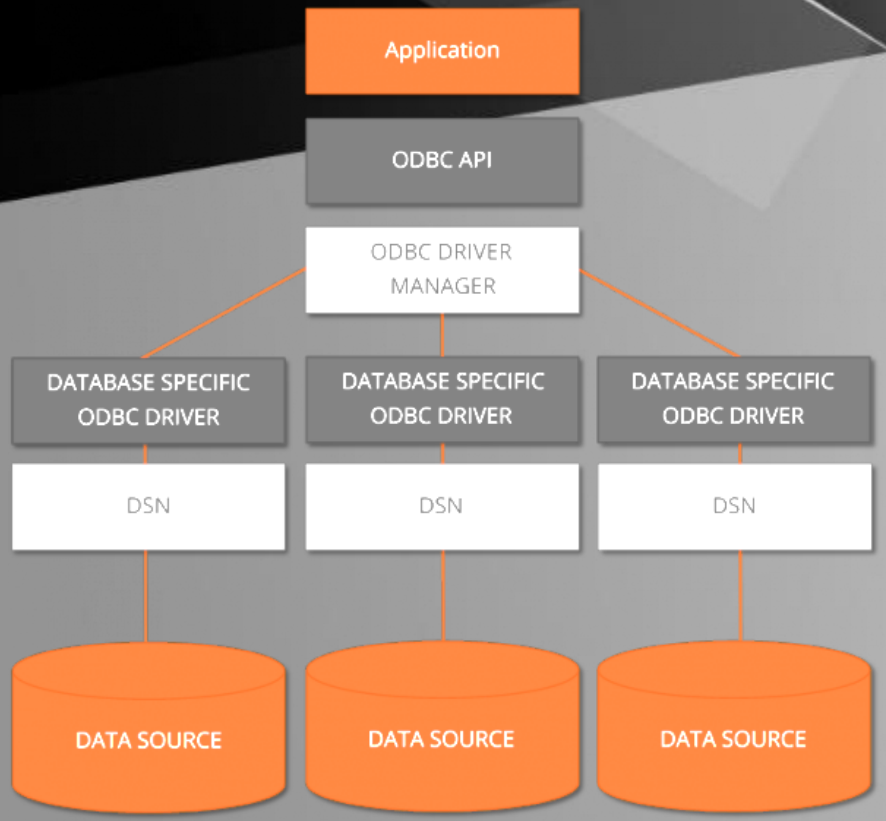
Этап 3.2 Запросы на просмотр базы данных

pg_mydb_show.py

```
print('\nПросматриваем наборы записей')
res = s.query(Book).filter(Book.yearpub.between(2000,2010)).order_by(Book.title)
for b in res:
    print(b)

print('Фильтр по связанным таблицам')
res = s.query(Book).join(Genre)\
    .filter(Genre.name.like('Фэнтези')).order_by(Book.title)
for b in res:
    print(b)
print()

s.close()
```

Таким образом, SQLAlchemy является мощным инструментом языка Python, который уменьшает время разработки программных продуктов с доступом к базам данных, реализуя объектно-реляционное отображения и поддерживая наследование. Однако не всегда такой подход может оказаться эффективным, тогда на помощь приходит классический язык SQL.

Интерфейс ODBC



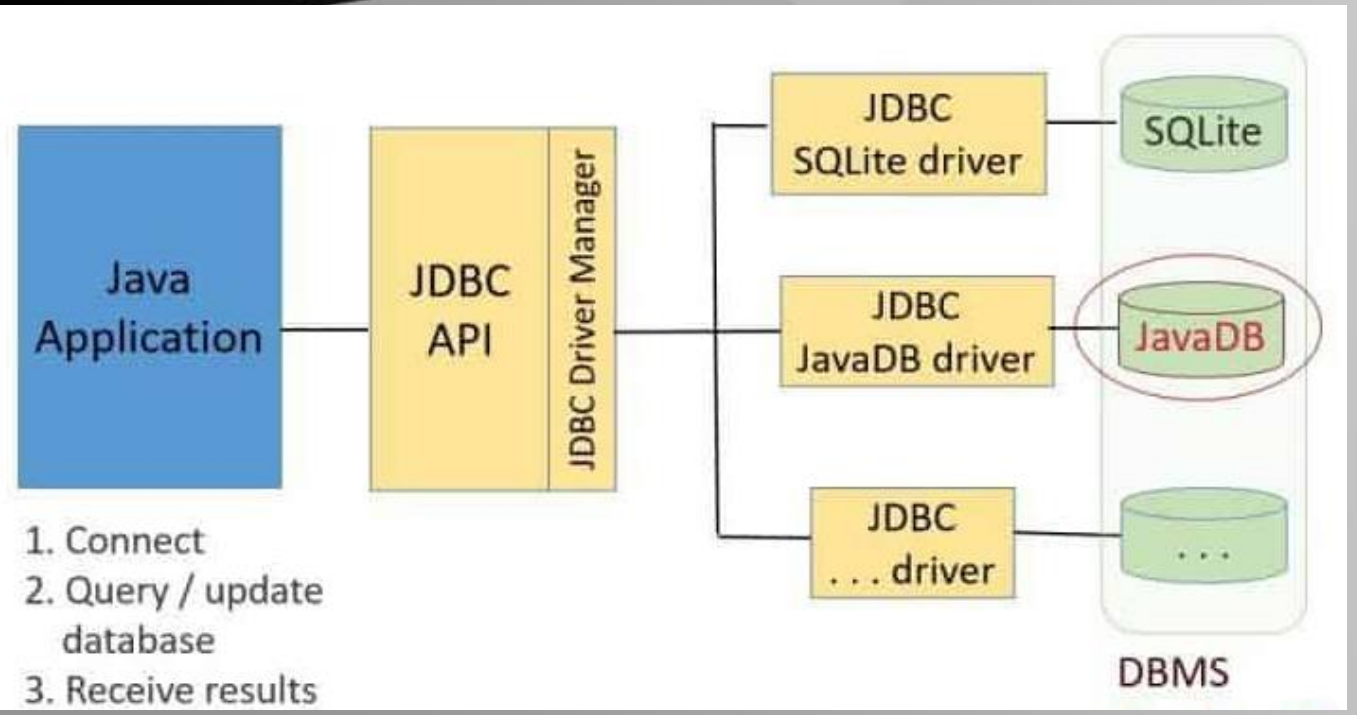
Функционал интерфейса ODBC:

- библиотека вызовов функций
- стандартный синтаксис SQL
- стандартные типы данных SQL
- стандартный протокол соединения с БД
- стандартные коды ошибок

Стандарт ODBC (Open Database Connectivity) был внедрён в 1992 г. компанией Microsoft на основе более ранних разработок от других компаний и организаций как свободный интерфейс доступа к базам данных. Наличие подобного стандарта позволяет любому приложению на клиентском компьютере получать доступ к любой базе данных на сервере с помощью языка SQL. ODBC предоставляет унифицированные средства взаимодействия прикладной клиентской программы с сервером баз данных. Интерфейс ODBC является многопотоковым, низкоуровневым, процедурным и независимым от операционной платформы. Для взаимодействия с базой данных приложение-клиент вызывает функции интерфейса, которые реализованы в специальных модулях, называемых ODBC-драйверами.

Источниками данных могут быть: реляционная БД, хранилище данных, БД индексно-последовательного метода доступа (ISAM), электронная таблица Microsoft Excel, текстовый файл, NoSQL БД (MongoDB) и т. п.

Интерфейс JDBC

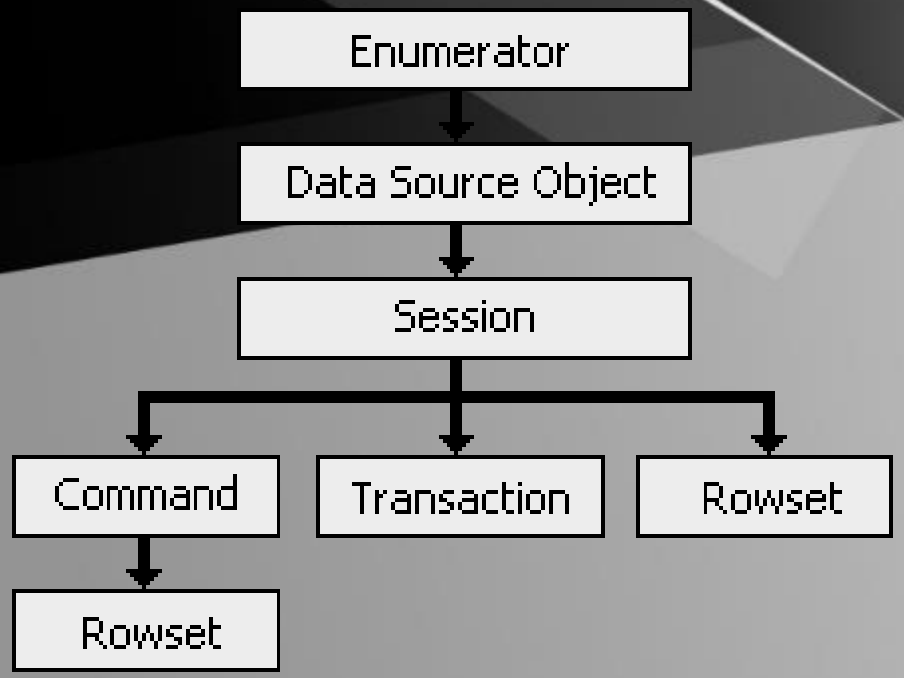


При реализации доступа через JDBC может использоваться мост JDBC-ODBC, реализующий доступ к базе данных через ODBC-драйверы.

JDBC (Java Database Connectivity) – интерфейс прикладного программирования, разработанный исключительно для использования в программах на языке Java, т.е. для виртуальных Java-машин. Разработан компанией Sun Microsystems в 1997 г. Является объектно-ориентированным интерфейсом, работающим под любыми операционными платформами, на которых может быть установлена виртуальная машина Java.

Интерфейс JDBC, как и ODBC, основан на концепции драйверов, позволяющих получать соединение с базой данных по специально описанному URL. Загрузившись, драйвер сам регистрирует себя и вызывается автоматически, когда программа требует URL, содержащий необходимый протокол. То есть, нет необходимости отдельно устанавливать эти драйверы на каждой клиентской машине (в отличие от ODBC-драйверов).

Интерфейс OLE DB



OLE DB (Object Linking and Embedding, Database) является проприетарной разработкой Microsoft и задумывался как альтернатива свободному интерфейсу ODBC. OLE DB представляет собой набор COM-интерфейсов (Component Object Model), которые предоставляют приложению-клиенту унифицированный доступ к различным источникам данных. OLE DB провайдеры, как и все COM-компоненты, регистрируются в реестре Windows.

OLE DB отделяет хранилище данных от приложения, которое должно иметь доступ к нему через набор абстракций, состоящий из источника данных (DataSource), сессии (Session), команды (Command) и набора строк (Rowset). В качестве данных могут выступать базы данных, простые документы, таблицы Excel и любые другие источники данных.

При реализации доступа к БД посредством OLE DB провайдера сначала следует создать объект данных и установить соединение с базой данных. Далее необходимо создать объект "сеанс". И только потом можно создавать результирующий набор. Объектная модель OLE DB включает объект Transaction. Применение OLE DB позволяет поддерживать простые, вложенные и распределенные транзакции.

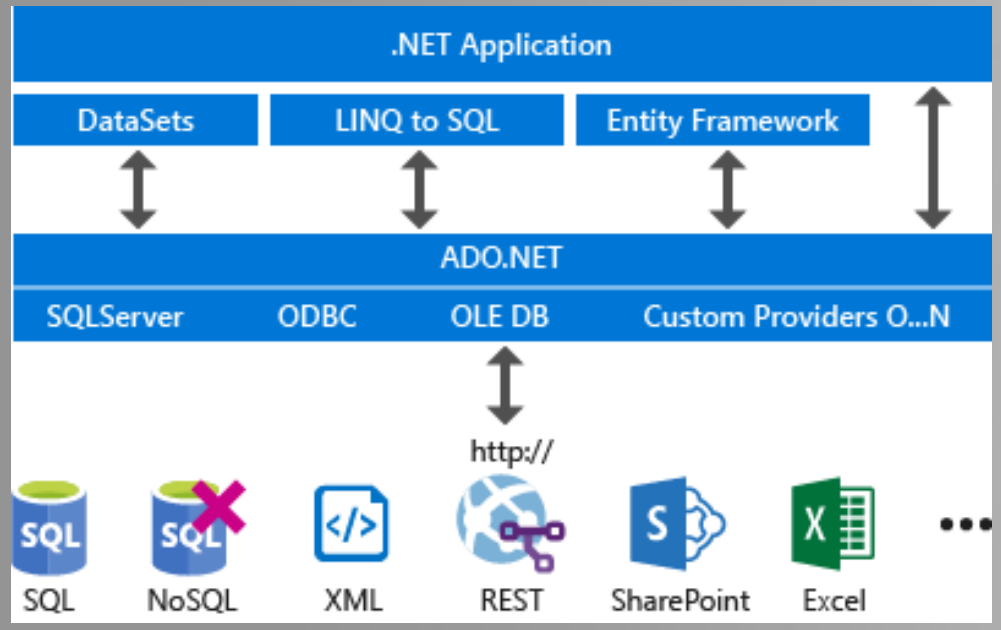
Интерфейс ADO.NET

ADO.NET

Технология ADO.NET (ActiveX Data Object для .NET Framework, Microsoft) представляет собой набор классов для приложений, разрабатываемых на базе библиотеки .NET Framework, через которые можно отправлять запросы к базам данных, устанавливать подключения, получать ответ от базы данных и производить ряд других операций.

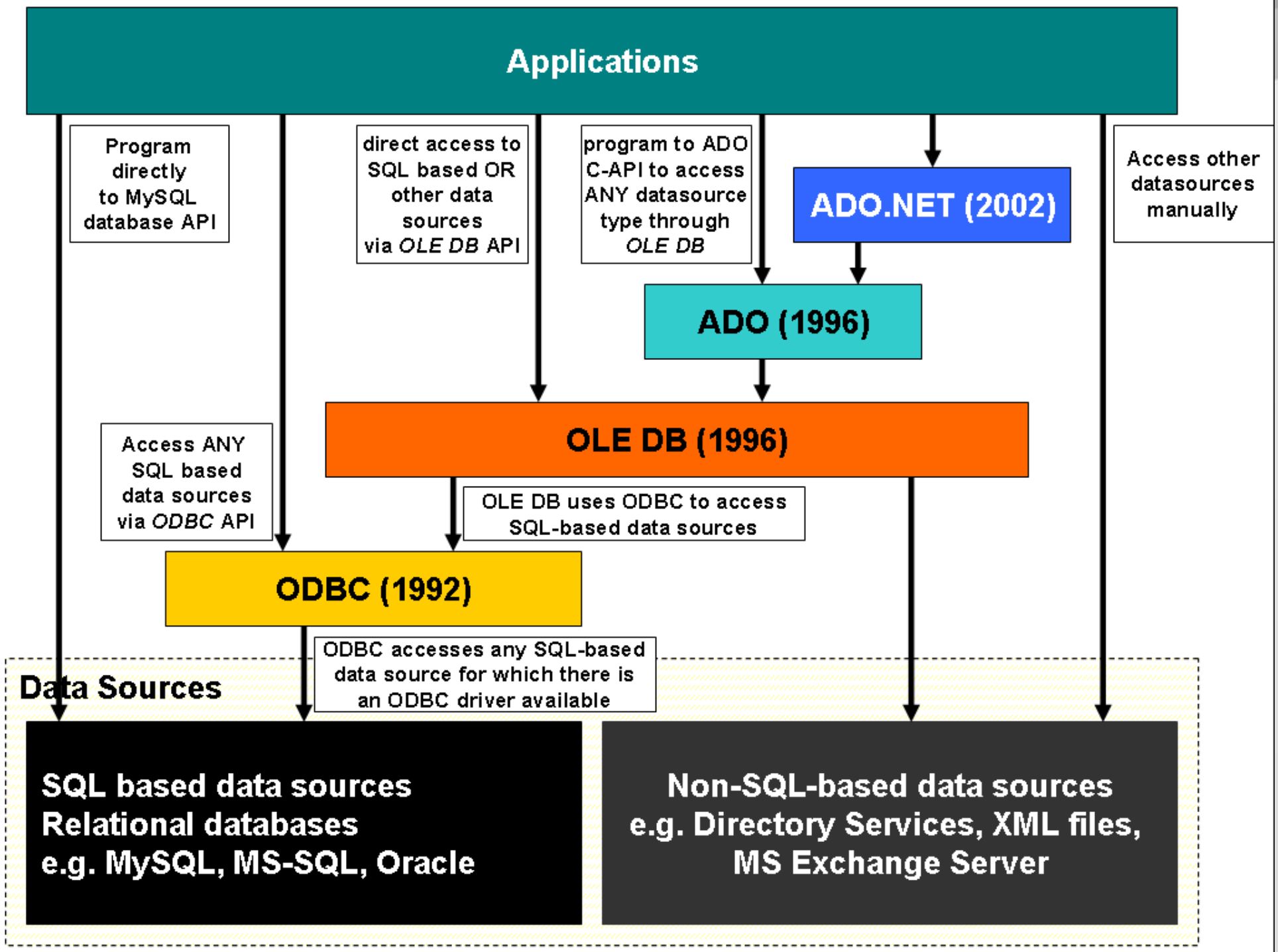
Поставщик данных .NET состоит из четырех основных компонентов:

- **Connection** — для связи с источником данных;
- **Command** — выполняет команды над источником данных;
- **DataReader** — читает данные из источника данных;
- **DataAdapter** — читает данные из источника данных и использует их для заполнения объекта DataSet.



Объекты DataSet представляют локальные копии взаимосвязанных таблиц данных, каждая из которых содержит набор строк и столбцов. Объекты DataSet позволяют вызывающей сборке (например, программе, выполняющейся на локальном компьютере) работать с содержимым DataSet, изменять его, не требуя подключения к источнику данных, и отправлять обратно блоки измененных данных для обработки с помощью соответствующего адаптера данных.

Интерфейсы баз данных



ADO.NET предоставляет самый прямой способ доступа к данным в .NET Framework. Для абстракции более высокого уровня, которая позволяет приложениям работать с концептуальной моделью, а не с базовой моделью хранения, следует использовать ADO.NET **Entity Framework**.

Спасибо за понимание!