

# Операции с данными (back-end)

Прямой доступ к данным

Версия 8.0



Эта документация предоставляется с ограничениями на использование и защищена законами об интеллектуальной собственности. За исключением случаев, прямо разрешенных в вашем лицензионном соглашении или разрешенных законом, вы не можете использовать, копировать, воспроизводить, переводить, транслировать, изменять, лицензировать, передавать, распространять, демонстрировать, выполнять, публиковать или отображать любую часть в любой форме или посредством любые значения. Обратный инжиниринг, дизассемблирование или декомпиляция этой документации, если это не требуется по закону для взаимодействия, запрещены.

Информация, содержащаяся в данном документе, может быть изменена без предварительного уведомления и не может гарантировать отсутствие ошибок. Если вы обнаружите какие-либо ошибки, сообщите нам о них в письменной форме.

# Содержание

<b>Прямой доступ к данным</b>	<b>5</b>
Получить данные из базы данных	5
Добавить данные в базу данных	6
Изменить данные в базе данных	8
Удалить данные из базы данных	9
Использовать многопоточность при работе с базой данных	9
<b>Получить данные из базы данных</b>	<b>11</b>
Пример 1	11
Пример 2	12
Пример 3	13
Пример 4	13
Пример 5	14
Пример 6	15
Пример 7	15
<b>Добавить данные в базу данных</b>	<b>16</b>
Пример 1	16
Пример 2	17
<b>Добавить данные в базу данных с помощью подзапросов</b>	<b>17</b>
Пример 1	17
Пример 2	18
<b>Изменить данные в базе данных</b>	<b>19</b>
Пример 1	19
Пример 2	19
<b>Удалить данные из базы данных</b>	<b>20</b>
Пример	20
<b>Класс Select</b>	<b>21</b>
Конструкторы	21
Свойства	21
Методы	23
<b>Класс Insert</b>	<b>32</b>
Конструкторы	33
Свойства	33
Методы	34
<b>Класс InsertSelect</b>	<b>36</b>
Конструкторы	36
Свойства	36

Методы	37
<b>Класс Update</b>	<b>39</b>
Конструкторы	39
Свойства	40
Методы	40
<b>Класс UpdateSelect</b>	<b>43</b>
Конструкторы	43
Свойства	44
Методы	44
<b>Класс Delete</b>	<b>44</b>
Конструкторы	45
Свойства	45
Методы	45
<b>Класс QueryFunction</b>	<b>48</b>
Класс QueryFunction	49
Класс AggregationQueryFunction	52
Класс IsNullQueryFunction	54
Класс CreateGuidQueryFunction	56
Класс CurrentDateTimeQueryFunction	57
Класс CoalesceQueryFunction	58
Класс DatePartQueryFunction	60
Класс DateAddQueryFunction	62
Класс DateDiffQueryFunction	64
Класс CastQueryFunction	66
Класс UpperQueryFunction	67
Класс CustomQueryFunction	69
Класс DataLengthQueryFunction	71
Класс TrimQueryFunction	72
Класс LengthQueryFunction	74
Класс SubstringQueryFunction	75
Класс ConcatQueryFunction	77
Класс WindowQueryFunction	79

# Прямой доступ к данным



**Способы доступа** к базе данных, которые предоставляют back-end компоненты ядра:

- Доступ через ORM-модель.
- Прямой доступ.

Для доступа к данным рекомендуется использовать ORM-модель, хотя прямой доступ к базе данных также реализован в back-end компонентах ядра. Выполнение запросов к базе данных через ORM-модель подробно описано в статье [Доступ к данным через ORM](#).

В этой статье будет рассмотрено выполнение запросов к базе данных через прямой доступ.

**Классы**, которые реализуют работу с данными через прямой доступ:

- `Terrasoft.Core.DB.Select` — построение запросов на получение записей из таблиц базы данных.
- `Terrasoft.Core.DB.Insert` — построение запросов на добавление записей в таблицы базы данных.
- `Terrasoft.Core.DB.InsertSelect` — построение запросов на добавление записей в таблицу базы данных на основании данных, полученных в запросах на получение записей из таблицы базы данных.
- `Terrasoft.Core.DB.Update` — построение запросов на изменение записей в таблице базы данных.
- `Terrasoft.Core.DB.UpdateSelect` — построение запросов на изменение записей в таблице базы данных на основании данных, полученных в запросах на получение записей из таблицы базы данных.
- `Terrasoft.Core.DB.Delete` — построение запросов на удаление записей в таблице базы данных.
- `Terrasoft.Core.DB.DBExecutor` — построение и выполнение сложных запросов (например, с несколькими вложенными фильтрациями, различными комбинациями join-ов и т. д.) к базе данных.

## Получить данные из базы данных

**Классы**, которые реализуют получение данных из базы данных:

- `Terrasoft.Core.DB.Select` — получение данных из базы данных через прямой доступ.
  - `Terrasoft.Core.Entities.EntitySchemaQuery` — получение данных из базы данных через ORM-модель.
- Выполнение запросов к базе данных с использованием класса `Terrasoft.Core.Entities.EntitySchemaQuery` подробно описано в статье [Доступ к данным через ORM](#).

**Назначение** класса `Terrasoft.Core.DB.Select` — построение запросов на выборку записей из таблиц базы данных. После создания и конфигурирования экземпляра класса будет построен `SELECT`-запрос к базе данных приложения. В запрос можно добавить колонки, фильтры и условия ограничений.

**Особенности** класса `Terrasoft.Core.DB.Select`:

- В результирующем запросе не учитываются права доступа текущего пользователя. Пользователь получает доступ ко всем таблицам и записям базы данных.

- В результирующем запросе не учитываются [данные из хранилища кэша](#).

**Результат выполнения запроса** — экземпляр, реализующий интерфейс `System.Data.IDataReader`, или скалярное значение соответствующего типа.

При необходимости построения запросов на выборку записей из базы данных с учетом прав доступа пользователя и данных из хранилища кэша, используйте класс `Terrasoft.Core.Entities.EntitySchemaQuery`.

## Добавить данные в базу данных

**Классы**, которые реализуют добавление данных в базу данных:

- `Terrasoft.Core.DB.Insert`.
- `Terrasoft.Core.DB.InsertSelect`.

## Массовое добавление данных

**Назначение** класса `Terrasoft.Core.DB.Insert` — построение запросов на добавление записей в таблицы базы данных. После создания и конфигурирования экземпляра класса будет построен `INSERT`-запрос к базе данных приложения.

**Результат выполнения запроса** — количество записей, которые были добавлены с помощью запроса.

Класс содержит реализацию функциональности многострочной вставки. Для этого предназначен метод `Values()`. При вызове метода `Values()` все последующие вызовы метода `Set()` попадают в новый экземпляр `ColumnsValues`. Если коллекция `ColumnsValuesCollection` содержит более одного набора данных, то будет построен запрос с несколькими блоками `Values()`.

### Пример многострочной вставки

```
new Insert(UserConnection)
    .Into("Table")
    .Values()
        .Set("Column1", Column.Parameter(1))
        .Set("Column2", Column.Parameter(1))
        .Set("Column3", Column.Parameter(1))
    .Values()
        .Set("Column1", Column.Parameter(2))
        .Set("Column2", Column.Parameter(2))
        .Set("Column3", Column.Parameter(2))
    .Values()
        .Set("Column1", Column.Parameter(3))
        .Set("Column2", Column.Parameter(3))
        .Set("Column3", Column.Parameter(3))
    .Execute();
```

В результате будет сформирован SQL-запрос.

**SQL-запрос**

```
-- Для MSSQL или PostgreSQL
INSERT INTO [dbo].[Table] (Column1, Column2, Column3)
VALUES (1, 1, 1),
       (2, 2, 2),
       (3, 3, 3)

-- Для Oracle
INSERT ALL
  into Table (column1, column2, column3) values (1, 1, 1)
  into Table (column1, column2, column3) values (2, 2, 2)
  into Table (column1, column2, column3) values (3, 3, 3)
SELECT * FROM dual
```

**Особенности использования** многострочного добавления данных:

- Ограничение количества параметров в MS SQL при использовании `Column.Parameter` в выражении `Set()` — 2100 параметров.
- При превышении допустимого количества параметров разбивку запроса на подзапросы должен выполнить разработчик, поскольку класс `Terrasoft.Core.DB.Insert` не предоставляет такую возможность.

**Пример**

```
IEnumerable<IEnumerable<ImportEntity>> GetImportEntitiesChunks(IEnumerable<ImportEntity> enti
    var entitiesList = entities.ToList();
    var columnsList = keyColumns.ToList();
    var maxParamsPerChunk = Math.Abs(MaxParametersCountPerQueryChunk / columnsList.Count + 1)
    var chunksCount = (int)Math.Ceiling(entitiesList.Count / (double)maxParamsPerChunk);
    return entitiesList.SplitOnParts(chunksCount);
}

var entitiesList = GetImportEntitiesChunks(entities, importColumns);
entitiesList.AsParallel().AsOrdered()
    .ForAll(entitiesBatch => {
        try {
            var insertQuery = GetBufferedImportEntityInsertQuery();
            foreach (var importEntity in entitiesBatch) {
                insertQuery.Values();
                SetBufferedImportEntityInsertColumnValues(importEntity, insertQuery,
                    importColumns);
                insertQuery.Set("ImportSessionId", Column.Parameter(importSessionId));
            }
            insertQuery.Execute();
        } catch (Exception e) {
```

```

        //...
    }
});

```

- Валидацию совпадения количества колонок и количества условий `Set()` должен выполнить разработчик, поскольку класс `Terrasoft.Core.DB.Insert` не предоставляет такую возможность. При несовпадении количества возникнет исключение на уровне работы СУБД.

## Добавление данных из выборки

**Назначение** класса `Terrasoft.Core.DB.InsertSelect` — построение запросов на добавление записей в таблицы базы данных на основании данных, полученных в запросах на получение записей из таблицы базы данных. То есть в качестве источника данных запроса используется экземпляр класса `Terrasoft.Core.DB.Select`. После создания и конфигурирования экземпляра класса будет построен `INSERT INTO SELECT`-запрос к базе данных приложения.

**Особенность** класса `Terrasoft.Core.DB.InsertSelect` — в результирующем запросе для добавляемых записей не учитываются права доступа текущего пользователя. Пользовательское соединение используется только для доступа к таблице базы данных.

**Результат выполнения запроса** — добавление в таблицы базы данных записей, которые были получены в `Select`-запросе.

## Изменить данные в базе данных

**Классы**, которые реализуют изменение данных в базе данных:

- `Terrasoft.Core.DB.Update`.
- `Terrasoft.Core.DB.UpdateSelect`.

## Массовое изменение данных

**Назначение** класса `Terrasoft.Core.DB.Update` — построение запросов на изменение записей в таблицах базы данных. После создания и конфигурирования экземпляра класса будет построен `UPDATE`-запрос к базе данных приложения.

## Изменение данных на основании выборки

**Назначение** класса `Terrasoft.Core.DB.UpdateSelect` — построение запросов на изменение записей в таблицах базы данных на основании данных, полученных в запросах на получение записей из таблицы базы данных. То есть в качестве источника данных запроса используется экземпляр класса `Terrasoft.Core.DB.Select`. После создания и конфигурирования экземпляра класса будет построен `UPDATE FROM`-запрос к базе данных приложения.

**Результат выполнения запроса** — изменение в таблице базы данных записей, которые были получены в `Select`-запросе.



## Удалить данные из базы данных

`Terrasoft.Core.DB.Delete` — класс, который реализует удаление данных из базы данных.

**Назначение** класса `Terrasoft.Core.DB.Delete` — построение запросов на удаление записей из таблиц базы данных. После создания и конфигурирования экземпляра класса будет построен `DELETE`-запрос к базе данных приложения.

## Использовать многопоточность при работе с базой данных

**Многопоточность** — использование нескольких параллельных потоков при отправке запросов к базе данных через `UserConnection`.

`Terrasoft.Core.DB.DBExecutor` — класс, который позволяет использовать многопоточность. Реализует построение и выполнение нескольких запросов к базе данных в одной транзакции. Одному пользователю доступен только один экземпляр `DBExecutor`. Пользователь не имеет возможности создавать новые экземпляры.

Использование многопоточности может привести к проблемам синхронизации старта и подтверждения транзакций. Проблема возникает, даже если `DBExecutor` не используется напрямую, а используется, например, через `EntitySchemaQuery`.

**Особенность** класса `Terrasoft.Core.DB.DBExecutor` — создание экземпляра `DBExecutor` необходимо оборачивать в оператор `using`. Это связано с тем, что для работы с базой данных используются неуправляемые ( `unmanaged` ) ресурсы. Также для освобождения ресурсов можно явно вызвать метод `Dispose()`. Использование оператора `using` подробно описано в официальной [документации Microsoft](#).

Транзакция начинается вызовом метода `dbExecutor.StartTransaction` и заканчивается вызовом `dbExecutor.CommitTransaction` или `dbExecutor.RollbackTransaction`. Если выполнение вышло за область видимости блока `using` и не был вызван метод `dbExecutor.CommitTransaction`, происходит автоматический откат транзакции.

**Важно.** При выполнении нескольких запросов в одной транзакции необходимо передавать `dbExecutor` в методы `Execute`, `ExecuteReader`, `ExecuteScalar`.

Ниже представлены фрагменты исходного кода с использованием `DBExecutor`. Нельзя выполнять вызов методов экземпляра `DBExecutor` в параллельных потоках.

### Пример правильного использования DBExecutor

```
/* Первое использование экземпляра DBExecutor в основном потоке. */
using (DBExecutor dbExecutor = UserConnection.EnsureDBConnection()) {
    dbExecutor.StartTransaction();
    //...
    dbExecutor.CommitTransaction();
}
//...
```

```

var select = (Select)new Select(UserConnection)
    .Column("Id")
    .From("Contact")
    .Where("Name")
    .IsEqual(Column.Parameter("Supervisor"));
/* Повторное использование экземпляра DBExecutor в основном потоке. */
using (DBExecutor dbExecutor = UserConnection.EnsureDBConnection()) {
    using (IDataReader dataReader = select.ExecuteReader(dbExecutor)) {
        while (dataReader.Read()) {
            //...
        }
    }
}

```

### Пример неправильного использования DBExecutor

```

/* Создание параллельного потока. */
var task = new Task(() => {

    /* Использование экземпляра DBExecutor в параллельном потоке. */
    using (DBExecutor dbExecutor = UserConnection.EnsureDBConnection()) {
        dbExecutor.StartTransaction();
        //...
        dbExecutor.CommitTransaction();
    }
});

/* Запуск асинхронной задачи в параллельном потоке. Выполнение программы в основном потоке продолжится. */
task.Start();
//...
var select = (Select)new Select(UserConnection)
    .Column("Id")
    .From("Contact")
    .Where("Name")
    .IsEqual(Column.Parameter("Supervisor"));

/* Использование экземпляра DBExecutor в основном потоке приведет к возникновению ошибки, поскольку экземпляр DBExecutor уже используется в параллельном потоке. */
using (DBExecutor dbExecutor = UserConnection.EnsureDBConnection()) {
    using (IDataReader dataReader = select.ExecuteReader(dbExecutor)) {
        while (dataReader.Read()) {
            //...
        }
    }
}

```

# Получить данные из базы данных



**На заметку.** Примеры, приведенные в этой статье, реализованы в веб-сервисе. Пакет с реализацией веб-сервиса прикреплен в блоке "Ресурсы".

Ниже приведен метод `CreateJson`, который используется в примерах для обработки результата запросов.

## Метод `CreateJson`

```
private string CreateJson(IDataReader dataReader)
{
    var list = new List<dynamic>();
    var cnt = dataReader.FieldCount;
    var fields = new List<string>();
    for (int i = 0; i < cnt; i++)
    {
        fields.Add(dataReader.GetName(i));
    }
    while (dataReader.Read())
    {
        dynamic exo = new System.Dynamic.ExpandoObject();
        foreach (var field in fields)
        {
            ((IDictionary<String, Object>)exo).Add(field, dataReader.GetColumnValue(field));
        }
        list.Add(exo);
    }
    return JsonConvert.SerializeObject(list);
}
```

## Пример 1

**Пример.** Выбрать определенное количество записей из требуемой таблицы (схемы объекта).

## Метод `SelectColumns`

```
public string SelectColumns(string tableName, int top)
{

```

```

top = top > 0 ? top : 1;
var result = "{}";
var select = new Select(UserConnection)
    .Top(top)
    .Column(Column.Asterisk())
    .From(tableName);
using (DBExecutor dbExecutor = UserConnection.EnsureDBConnection())
{
    using (IDataReader dataReader = select.ExecuteReader(dbExecutor))
    {
        result = CreateJson(dataReader);
    }
}
return result;
}

```

## Пример 2

**Пример.** Выбрать идентификатор, имя и дату рождения контактов, дата рождения которых позже требуемого года.

### Метод `SelectContactsYoungerThan`

```

public string SelectContactsYoungerThan(string birthYear)
{
    var result = "{}";
    var year = DateTime.ParseExact(birthYear, "yyyy", CultureInfo.InvariantCulture);
    var select = new Select(UserConnection)
        .Column("Id")
        .Column("Name")
        .Column("BirthDate")
        .From("Contact")
        .Where("BirthDate").IsGreater(Column.Parameter(year))
        .Or("BirthDate").IsNull()
        .OrderByDesc("BirthDate")
        as Select;
    using (DBExecutor dbExecutor = UserConnection.EnsureDBConnection())
    {
        using (IDataReader dataReader = select.ExecuteReader(dbExecutor))
        {
            result = CreateJson(dataReader);
        }
    }
    return result;
}

```

```
}
```

## Пример 3

**Пример.** Выбрать идентификатор, имя и дату рождения контактов, дата рождения которых позже заданного года и у которых указан контрагент.

### Метод `SelectContactsYoungerThanAndHasAccountId`

```
public string SelectContactsYoungerThanAndHasAccountId(string birthYear)
{
    var result = "{}";
    var year = DateTime.ParseExact(birthYear, "yyyy", CultureInfo.InvariantCulture);
    var select = new Select(UserConnection)
        .Column("Id")
        .Column("Name")
        .Column("BirthDate")
        .From("Contact")
        .Where()
        .OpenBlock("BirthDate").IsGreater(Column.Parameter(year))
        .Or("BirthDate").IsNull()
        .CloseBlock()
        .And("AccountId").Not().IsNull()
        .OrderByDesc("BirthDate")
        as Select;
    using (DBExecutor dbExecutor = UserConnection.EnsureDBConnection())
    {
        using (IDataReader dataReader = select.ExecuteReader(dbExecutor))
        {
            result = CreateJson(dataReader);
        }
    }
    return result;
}
```

## Пример 4

**Пример.** Выбрать идентификатор и имя всех контактов, присоединив к ним идентификаторы и названия соответствующих контрагентов.

**Метод** `SelectContactsJoinAccount`

```

public string SelectContactsJoinAccount()
{
    var result = "{}";
    var select = new Select(UserConnection)
        .Column("Contact", "Id").As("ContactId")
        .Column("Contact", "Name").As("ContactName")
        .Column("Account", "Id").As("AccountId")
        .Column("Account", "Name").As("AccountName")
        .From("Contact")
        .Join(JoinType.Inner, "Account")
        .On("Contact", "Id").IsEqual("Account", "PrimaryContactId")
        as Select;
    using (DBExecutor dbExecutor = UserConnection.EnsureDBConnection())
    {
        using (IDataReader dataReader = select.ExecuteReader(dbExecutor))
        {
            result = CreateJson(dataReader);
        }
    }
    return result;
}

```

## Пример 5

**Пример.** Выбрать идентификатор и имя контактов, являющихся основными для контрагентов.

**Метод** `SelectAccountPrimaryContacts`

```

public string SelectAccountPrimaryContacts()
{
    var result = "{}";
    var select = new Select(UserConnection)
        .Column("Id")
        .Column("Name")
        .From("Contact").As("C")
        .Where()
        .Exists(new Select(UserConnection)
            .Column("A", "PrimaryContactId")
            .From("Account").As("A")
            .Where("A", "PrimaryContactId").IsEqual("C", "Id"))
        as Select;
}

```

```

using (DBExecutor dbExecutor = UserConnection.EnsureDBConnection())
{
    using (IDataReader dataReader = select.ExecuteReader(dbExecutor))
    {
        result = CreateJson(dataReader);
    }
}
return result;
}

```

## Пример 6

**Пример.** Выбрать страны и количество городов в стране, если количество городов больше указанного.

### Метод `SelectCountriesWithCitiesCount`

```

public string SelectCountriesWithCitiesCount(int count)
{
    var result = "{}";
    var select = new Select(UserConnection)
        .Column(Func.Count("City", "Id")).As("CitiesCount")
        .Column("Country", "Name").As("CountryName")
        .From("City")
        .Join(JoinType.Inner, "Country")
        .On("City", "CountryId").IsEqual("Country", "Id")
        .GroupBy("Country", "Name")
        .Having(Func.Count("City", "Id").IsGreater(Column.Parameter(count)))
        .OrderByDesc("CitiesCount")
        as Select;
    using (DBExecutor dbExecutor = UserConnection.EnsureDBConnection())
    {
        using (IDataReader dataReader = select.ExecuteReader(dbExecutor))
        {
            result = CreateJson(dataReader);
        }
    }
    return result;
}

```

## Пример 7

**Пример.** Получить идентификатор контакта по его имени.

**Метод** `SelectCountryIdByCityName`

```
public string SelectCountryIdByCityName(string CityName)
{
    var result = "";
    var select = new Select(UserConnection)
        .Column("CountryId")
        .From("City")
        .Where("Name").IsEqual(Column.Parameter(CityName)) as Select;
    result = select.ExecuteScalar<Guid>().ToString();
    return result;
}
```

## Добавить данные в базу данных



**На заметку.** Примеры, приведенные в этой статье, реализованы в веб-сервисе. Пакет с реализацией веб-сервиса прикреплен в блоке "Ресурсы".

### Пример 1

**Пример.** Добавить контакт с указанным именем.

**Метод** `InsertContact`

```
public string InsertContact(string contactName)
{
    contactName = contactName ?? "Unknown contact";
    var ins = new Insert(UserConnection)
        .Into("Contact")
        .Set("Name", Column.Parameter(contactName));
    var affectedRows = ins.Execute();
    var result = $"Inserted new contact with name '{contactName}'. {affectedRows} rows affected"
    return result;
}
```



## Пример 2

**Пример.** Добавить город с указанным названием, привязав его к указанной стране.

**Метод** InsertCity

```
public string InsertCity(string city, string country)
{
    city = city ?? "unknown city";
    country = country ?? "unknown country";

    var ins = new Insert(UserConnection)
        .Into("City")
        .Set("Name", Column.Parameter(city))
        .Set("CountryId",
            new Select(UserConnection)
                .Top(1)
                .Column("Id")
                .From("Country")
                .Where("Name")
                .IsEqual(Column.Parameter(country)));
    var affectedRows = ins.Execute();
    var result = $"Inserted new city with name '{city}' located in '{country}'. {affectedRows} r
    return result;
}
```

# Добавить данные в базу данных с помощью подзапросов



**На заметку.** Примеры, приведенные в этой статье, реализованы в веб-сервисе. Пакет с реализацией веб-сервиса прикреплен в блоке "Ресурсы".

## Пример 1

**Пример.** Добавить контакт с указанными именем и названием контрагента.

**Метод** InsertContactWithAccount

```

public string InsertContactWithAccount(string contactName, string accountName)
{
    contactName = contactName ?? "Unknown contact";
    accountName = accountName ?? "Unknown account";

    var id = Guid.NewGuid();
    var selectQuery = new Select(UserConnection)
        .Column(Column.Parameter(contactName))
        .Column("Id")
        .From("Account")
        .Where("Name").IsEqual(Column.Parameter(accountName)) as Select;
    var insertSelectQuery = new InsertSelect(UserConnection)
        .Into("Contact")
        .Set("Name", "AccountId")
        .FromSelect(selectQuery);

    var affectedRows = insertSelectQuery.Execute();
    var result = $"Inserted new contact with name '{contactName}'" +
        $" and account '{accountName}'." +
        $" Affected {affectedRows} rows.";
    return result;
}

```

## Пример 2

**Пример.** Добавить контакт с указанным именем, связав его со всеми контрагентами.

### Метод InsertAllAccountsContact

```

public string InsertAllAccountsContact(string contactName)
{
    contactName = contactName ?? "Unknown contact";

    var id = Guid.NewGuid();
    var insertSelectQuery = new InsertSelect(UserConnection)
        .Into("Contact")
        .Set("Name", "AccountId")
        .FromSelect(
            new Select(UserConnection)
                .Column(Column.Parameter(contactName))
                .Column("Id")
                .From("Account") as Select);
}

```

```

var affectedRows = insertSelectQuery.Execute();
var result = $"Inserted {affectedRows} new contacts with name '{contactName}'";
return result;
}

```

# Изменить данные в базе данных



**На заметку.** Примеры, приведенные в этой статье, реализованы в веб-сервисе. Пакет с реализацией веб-сервиса прикреплен в блоке "Ресурсы".

В большинстве случаев запрос на изменение данных должен содержать условие `where`, которое уточняет какие именно записи необходимо изменить. Если не указать условие `where`, то будут изменены все записи.

## Пример 1

**Пример.** Изменить имя контакта.

**Метод** `ChangeContactName`

```

public string ChangeContactName(string oldName, string newName)
{
    var update = new Update(UserConnection, "Contact")
        .Set("Name", Column.Parameter(newName))
        .Where ("Name").IsEqual(Column.Parameter(oldName));
    var cnt = update.Execute();
    return $"Contacts {oldName} changed to {newName}. {cnt} rows affected.";
}

```

## Пример 2

**Пример.** Для всех существующих контактов поменять пользователя, изменившего запись, на указанного.

**Метод** `ChangeAllContactModifiedBy`

```
public string ChangeAllContactModifiedBy(string Name)
{
    var update = new Update(UserConnection, "Contact")
        .Set("ModifiedById",
            new Select(UserConnection).Top(1)
                .Column("Id")
                .From("Contact")
                .Where("Name").IsEqual(Column.Parameter(Name)));
    var cnt = update.Execute();
    return $"All contacts are changed by {Name} now. {cnt} rows affected.";
}
```

Условие `where` относится к запросу `Select`. Запрос `Update` не содержит условия `where`, поскольку необходимо изменить все записи.

## Удалить данные из базы данных



**На заметку.** Примеры, приведенные в этой статье, реализованы в веб-сервисе. Пакет с реализацией веб-сервиса прикреплен в блоке "Ресурсы".

В большинстве случаев запрос на удаление данных должен содержать условие `where`, которое уточняет какие именно записи необходимо удалить. Если не указать условие `where`, то будут удалены все записи.

## Пример

**Пример.** Удалить контакт с указанным именем.

### Метод DeleteContacts

```
public string DeleteContacts(string name)
{
    var delete = new Delete(UserConnection)
        .From("Contact")
        .Where("Name").IsEqual(Column.Parameter(name));
    var cnt = delete.Execute();
    return $"Contacts with name {name} were deleted. {cnt} rows affected";
}
```

# Класс Select C#



Пространство имен `Terrasoft.Core.DB`.

Класс `Terrasoft.Core.DB.Select` предназначен для построения запросов выборки записей из таблиц базы данных. В результате создания и конфигурирования экземпляра этого класса будет построен запрос в базу данных приложения в виде SQL-выражения `SELECT`. В запрос можно добавить требуемые колонки, фильтры и условия ограничений. Результаты выполнения запроса возвращаются в виде экземпляра, реализующего интерфейс `System.Data.IDataReader`, либо скалярного значения требуемого типа.

**Важно.** При работе с классом `Select` не учитываются права доступа пользователя, использующего текущее соединение. Доступны абсолютно все записи из базы данных приложения. Также не учитываются данные, помещенные в [хранилище кэша](#). Если необходимы дополнительные возможности по управлению правами доступа и работе с хранилищем кэша Creatio, следует использовать класс `EntitySchemaQuery`.

**На заметку.** Полный перечень методов и свойств класса `Select`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

## Конструкторы

`Select(UserConnection userConnection)`

Создает экземпляр класса с указанным `UserConnection`.

`Select(UserConnection userConnection, CancellationToken cancellationToken)`

Создает экземпляр класса с указанным `UserConnection` и токеном отмены [выполнения управляемого потока](#).

`Select(Select source)`

Создает экземпляр класса, являющийся клоном экземпляра, переданного в качестве аргумента.

## Свойства

`UserConnection Terrasoft.Core.UserConnection`

Пользовательское подключение, используемое при выполнении запроса.

`RowCount int`

Количество записей, которые вернет запрос после выполнения.

---

`Parameters Terrasoft.Core.DB.QueryParameterCollection`

Коллекция параметров запроса.

---

`HasParameters bool`

Определяет наличие параметров у запроса.

---

`BuildParametersAsValue bool`

Определяет, добавлять ли параметры запроса в текст запроса как значения.

---

`Joins Terrasoft.Core.DB.JoinCollection`

Коллекция выражений `Join` в запросе.

---

`HasJoins bool`

Определяет наличие выражений `Join` в запросе.

---

`Condition Terrasoft.Core.DB.QueryCondition`

Условие выражения `Where` запроса.

---

`HasCondition bool`

Определяет наличие выражения `Where` в запросе.

---

`HavingCondition Terrasoft.Core.DB.QueryCondition`

Условие выражения `Having` запроса.

---

`HasHavingCondition bool`

Определяет наличие выражения `Having` в запросе.

---

`OrderByItems Terrasoft.Core.DB.OrderByItemCollection`

Коллекция выражений, по которым выполняется сортировка результатов запроса.

---

`HasOrderByItems` `bool`

Определяет наличие условий сортировки результатов запроса.

---

`GroupByItems` `Terrasoft.Core.DB.QueryColumnExpressionCollection`

Коллекция выражений, по которым выполняется группировка результатов запроса.

---

`HasGroupByItems` `bool`

Определяет наличие условий группировки результатов запроса.

---

`IsDistinct` `bool`

Определяет, должен ли запрос возвращать только уникальные записи.

---

`Columns` `Terrasoft.Core.DB.QueryColumnExpressionCollection`

Коллекция выражений колонок запроса.

---

`OffsetFetchPaging` `bool`

Определяет возможность分页ного возврата результата запроса.

---

`RowsOffset` `int`

Количество строк, которые необходимо пропустить при возврате результата запроса.

---

`QueryKind` `Terrasoft.Common.QueryKind`

Тип запроса (см. статью [Настройка отдельного пула запросов](#)).

---

## Методы

---

`void ResetCachedSqlText()`

Очищает закэшированный текст запроса.

---

`QueryParameterCollection GetUsingParameters()`

Возвращает коллекцию параметров, используемых запросом.

---

---

```
void ResetParameters()
```

Очищает коллекцию параметров запроса.

---

```
QueryParameterCollection InitializeParameters()
```

Инициализирует коллекцию параметров запроса.

---

```
IDataReader ExecuteReader(DBExecutor dbExecutor)
```

Выполняет запрос, используя экземпляр `DBExecutor`. Возвращает объект, реализующий интерфейс `IDataReader`.

#### Параметры

<code>dbExecutor</code>	Экземпляр <code>DBExecutor</code> , который используется для выполнения запроса.
-------------------------	--

---

```
IDataReader ExecuteReader(DBExecutor dbExecutor, CommandBehavior behavior)
```

Выполняет запрос, используя экземпляр `DBExecutor`. Возвращает объект, реализующий интерфейс `IDataReader`.

#### Параметры

<code>behavior</code>	Предоставляет описание результатов запроса и их эффект на базу данных.
<code>dbExecutor</code>	Экземпляр <code>DBExecutor</code> , который используется для выполнения запроса.

---

```
void ExecuteReader(ExecuteReaderReadMethod readMethod)
```

Выполняет запрос, вызывая переданный метод делегата `ExecuteReaderReadMethod` для каждой записи результирующего набора.

#### Параметры

<code>readMethod</code>	Метод делегата <code>ExecuteReaderReadMethod</code> .
-------------------------	---

---

```
TResult ExecuteScalar<TResult>()
```



Выполняет запрос. Возвращает типизированный первый столбец первой записи результирующего набора.

```
TResult ExecuteScalar<TResult>(DBExecutor dbExecutor)
```

Выполняет запрос, используя экземпляр `DBExecutor`. Возвращает типизированный первый столбец первой записи результирующего набора.

#### Параметры

dbExecutor	Экземпляр <code>DBExecutor</code> , который используется для выполнения запроса.
------------	--

```
Select Distinct()
```

Добавляет к SQL-запросу ключевое слово `DISTINCT`. Исключает дублирование записей в результирующем наборе. Возвращает экземпляр запроса.

```
Select Top(int rowCount)
```

Устанавливает количество записей, возвращаемых в результирующем наборе. При этом меняется значение свойства `RowCount`. Возвращает экземпляр запроса.

#### Параметры

rowCount	Количество первых записей результирующего набора.
----------	---

```
Select As(string alias)
```

Добавляет указанный в аргументе псевдоним для последнего выражения запроса. Возвращает экземпляр запроса.

#### Параметры

alias	Псевдоним выражения запроса.
-------	------------------------------

```
Select Column(string sourceColumnAlias)
```

```
Select Column(string sourceAlias, string sourceColumnAlias)
```

```
Select Column(Select subSelect)
```

```
Select Column(Query subSelectQuery)
```

```
Select Column(QueryCase queryCase)
```

```
Select Column(QueryParameter queryParameter)
```

```
Select Column(QueryColumnExpression columnExpression)
```

Добавляет выражение, подзапрос или параметр в коллекцию выражений колонок запроса. Возвращает экземпляр запроса.

#### Параметры

sourceColumnAlias	Псевдоним колонки, для которой добавляется выражение.
sourceAlias	Псевдоним источника, из которого добавляется выражение колонки.
subSelect	Добавляемый подзапрос выборки данных.
subSelectQuery	Добавляемый подзапрос.
queryCase	Добавляемое выражение для оператора <code>Case</code> .
queryParameter	Добавляемый параметр запроса.
columnExpression	Выражение, для результатов которого добавляется условие.

```
Select From(string schemaName)
Select From(Select subSelect)
Select From(Query subSelectQuery)
Select From(QuerySourceExpression sourceExpression)
```

Добавляет в запрос источник данных. Возвращает экземпляр запроса.

#### Параметры

schemaName	Название схемы.
subSelect	Подзапрос выборки, результаты которого становятся источником данных для текущего запроса.
subSelectQuery	Подзапрос, результаты которого становятся источником данных для текущего запроса.
sourceExpression	Выражение источника данных запроса.

```
Join Join(JoinType joinType, string schemaName)
Join Join(JoinType joinType, Select subSelect)
Join Join(JoinType joinType, Query subSelectQuery)
Join Join(JoinType joinType, QuerySourceExpression sourceExpression)
```

Присоединяет к текущему запросу схему, подзапрос или выражение.

## Параметры

joinType	Тип присоединения.
schemaName	Название присоединяемой схемы.
subSelect	Присоединяемый подзапрос выборки данных.
subSelectQuery	Присоединяемый подзапрос.
sourceExpression	Присоединяемое выражение.

## Возможные значения ( `Terrasoft.Core.DB.JoinType` )

Inner	Внутреннее соединение.
LeftOuter	Левое внешнее соединение.
RightOuter	Правое внешнее соединение.
FullOuter	Полное соединение.
Cross	Перекрестное соединение.

```

QueryCondition Where()
QueryCondition Where(string sourceColumnAlias)
QueryCondition Where(string sourceAlias, string sourceColumnAlias)
QueryCondition Where(Select subSelect)
QueryCondition Where(Query subSelectQuery)
QueryCondition Where(QueryColumnExpression columnExpression)
Query Where(QueryCondition condition)

```

Добавляет к текущему запросу начальное условие.

## Параметры

sourceColumnAlias	Псевдоним колонки, для которой добавляется условие.
sourceAlias	Псевдоним источника.
subSelect	Подзапрос выборки данных, для результатов которого добавляется условие.
subSelectQuery	Подзапрос, для результатов которого добавляется условие.
columnExpression	Выражение, для результатов которого добавляется условие.
condition	Условие запроса.

```

QueryCondition And()
QueryCondition And(string sourceColumnAlias)
QueryCondition And(string sourceAlias, string sourceColumnAlias)
QueryCondition And(Select subSelect)
QueryCondition And(Query subSelectQuery)
QueryCondition And(QueryParameter parameter)
QueryCondition And(QueryColumnExpression columnExpression)
Query And(QueryCondition condition)

```

К текущему условию запроса добавляет условие (предикат), используя логическую операцию И.

### Параметры

sourceColumnAlias	Псевдоним колонки, для которой добавляется предикат.
sourceAlias	Псевдоним источника.
subSelect	Подзапрос выборки данных, используемый в качестве предиката.
subSelectQuery	Подзапрос, используемый в качестве предиката.
parameter	Параметр, для которого добавляется предикат.
columnExpression	Выражение, используемое в качестве предиката.
condition	Условие запроса.

```

QueryCondition Or()
QueryCondition Or(string sourceColumnAlias)
QueryCondition Or(string sourceAlias, string sourceColumnAlias)

```

```

QueryCondition Or(Select subSelect)
QueryCondition Or(Query subSelectQuery)
QueryCondition Or(QueryParameter parameter)
QueryCondition Or(QueryColumnExpression columnExpression)
Query Or(QueryCondition condition)

```

К текущему условию запроса добавляет условие (предикат), используя логическую операцию ИЛИ.

### Параметры

sourceColumnAlias	Псевдоним колонки, для которой добавляется предикат.
sourceAlias	Псевдоним источника.
subSelect	Подзапрос на выборку данных, используемый в качестве предиката.
subSelectQuery	Подзапрос, используемый в качестве предиката.
parameter	Параметр, для которого добавляется предикат.
columnExpression	Выражение, используемое в качестве предиката.
condition	Условие запроса.

```

Query OrderBy(OrderDirectionStrict direction, string sourceColumnAlias)
Query OrderBy(OrderDirectionStrict direction, string sourceAlias, string sourceColumnAlias)
Query OrderBy(OrderDirectionStrict direction, QueryFunction queryFunction)
Query OrderBy(OrderDirectionStrict direction, Select subSelect)
Query OrderBy(OrderDirectionStrict direction, Query subSelectQuery)
Query OrderBy(OrderDirectionStrict direction, QueryColumnExpression columnExpression)

```

Выполняет сортировку результатов запроса. Возвращает экземпляр запроса.

### Параметры

direction	Порядок сортировки.
sourceColumnAlias	Псевдоним колонки, по которой выполняется сортировка.
sourceAlias	Псевдоним источника.
queryFunction	Функция, значение которой используется в качестве ключа сортировки.
subSelect	Подзапрос на выборку данных, результаты которого используются в качестве ключа сортировки.
subSelectQuery	Подзапрос, результаты которого используются в качестве ключа сортировки.
columnExpression	Выражение, результаты которого используются в качестве ключа сортировки.

```

Query OrderByAsc(string sourceColumnAlias)
Query OrderByAsc(string sourceAlias, string sourceColumnAlias)
Query OrderByAsc(Select subSelect)
Query OrderByAsc(Query subSelectQuery)
Query OrderByAsc(QueryColumnExpression columnExpression)

```

Выполняет сортировку результатов запроса в порядке возрастания. Возвращает экземпляр запроса.

### Параметры

sourceColumnAlias	Псевдоним колонки, по которой выполняется сортировка.
sourceAlias	Псевдоним источника.
subSelect	Подзапрос на выборку данных, результаты которого используются в качестве ключа сортировки.
subSelectQuery	Подзапрос, результаты которого используются в качестве ключа сортировки.
columnExpression	Выражение, результаты которого используются в качестве ключа сортировки.

```

Query OrderByDesc(string sourceColumnAlias)
Query OrderByDesc(string sourceAlias, string sourceColumnAlias)
Query OrderByDesc(Select subSelect)

```

```
Query OrderByDesc(Query subSelectQuery)
```

```
Query OrderByDesc(QueryColumnExpression columnExpression)
```

Выполняет сортировку результатов запроса в порядке убывания. Возвращает экземпляр запроса.

#### Параметры

sourceColumnAlias	Псевдоним колонки, по которой выполняется сортировка.
sourceAlias	Псевдоним источника.
subSelect	Подзапрос выборки, результаты которого используются в качестве ключа сортировки.
subSelectQuery	Подзапрос, результаты которого используются в качестве ключа сортировки.
columnExpression	Выражение, результаты которого используются в качестве ключа сортировки.

```
Query GroupBy(string sourceColumnAlias)
```

```
Query GroupBy(string sourceAlias, string sourceColumnAlias)
```

```
Query GroupBy(QueryColumnExpression columnExpression)
```

Выполняет группировку результатов запроса. Возвращает экземпляр запроса.

#### Параметры

sourceColumnAlias	Псевдоним колонки, по которой выполняется группировка.
sourceAlias	Псевдоним источника.
columnExpression	Выражение, результаты которого используются в качестве ключа группировки.

```
QueryCondition Having()
```

```
QueryCondition Having(string sourceColumnAlias)
```

```
QueryCondition Having(string sourceAlias, string sourceColumnAlias)
```

```
QueryCondition Having(Select subSelect)
```

```
QueryCondition Having(Query subSelectQuery)
```

```
QueryCondition Having(QueryParameter parameter)
```

```
QueryCondition Having(QueryColumnExpression columnExpression)
```

Добавляет в текущий запрос групповое условие. Возвращает экземпляр

`Terrasoft.Core.DB.QueryCondition`, представляющий групповое условие для параметра запроса.

### Параметры

sourceColumnAlias	Псевдоним колонки, по которой добавляется групповое условие.
sourceAlias	Псевдоним источника.
subSelect	Подзапрос выборки, для результатов которого добавляется групповое условие.
subSelectQuery	Подзапрос, для результатов которого добавляется групповое условие.
parameter	Параметр запроса, для которого добавляется групповое условие.
columnExpression	Выражение, используемое в качестве предиката.

Query Union(Select unionSelect)

Query Union(Query unionSelectQuery)

Объединяет результаты переданного запроса с результатами текущего запроса, исключая дубликаты из результирующего набора.

### Параметры

unionSelect	Подзапрос выборки.
unionSelectQuery	Подзапрос.

Query UnionAll(Select unionSelect)

Query UnionAll(Query unionSelectQuery)

Объединяет результаты переданного запроса с результатами текущего запроса. Дубликаты из результирующего набора не исключаются.

### Параметры

unionSelect	Подзапрос выборки.
unionSelectQuery	Подзапрос.





Пространство имен `Terrasoft.Core.DB` .

Класс `Terrasoft.Core.DB.Insert` предназначен для построения запросов на добавление записей в таблицы базы данных Creatio. В результате создания и конфигурирования экземпляра этого класса будет построен запрос в базу данных приложения в виде SQL-выражения `INSERT` . В результате выполнения запроса возвращается количество задействованных запросом записей.

**Важно.** При работе с классом `Insert` на добавленные записи не применяются права доступа по умолчанию. Пользовательское соединение используется только для доступа к таблице базы данных.

**На заметку.** Полный перечень методов и свойств класса `Insert` , его родительских классов, а также реализуемых им интерфейсов, можно найти в [Библиотеке .NET классов](#).

## Конструкторы

`Entity(UserConnection userConnection)`

Создает новый экземпляр класса `Entity` для заданного пользовательского подключения `UserConnection` .

`Insert(UserConnection userConnection)`

Создает экземпляр класса с указанным `UserConnection` .

`Insert(Insert source)`

Создает экземпляр класса, являющийся клоном экземпляра, переданного в качестве аргумента.

## Свойства

`UserConnection Terrasoft.Core.UserConnection`

Пользовательское подключение, используемое при запросе.

`Source Terrasoft.Core.DB.ModifyQuerySource`

Источник данных

`Parameters Terrasoft.Core.DB.QueryParameterCollection`

Коллекция параметров запроса.

---

`HasParameters` `bool`

Определяет, имеет ли запрос параметры.

---

`BuildParametersAsValue` `bool`

Определяет, добавлять ли параметры запроса в текст запроса как значения.

---

`ColumnValues` `Terrasoft.Core.DB.ModifyQueryColumnValueCollection`

Коллекция значений колонок запроса.

---

`ColumnValuesCollection` `List<ModifyQueryColumnValueCollection>`

Коллекция значений колонок для множественного добавления записей.

## Методы

---

`void ResetCachedSqlText()`

Очищает кэшированный текст запроса.

---

`QueryParameterCollection` `GetUsingParameters()`

Возвращает коллекцию параметров, используемых запросом.

---

`void ResetParameters()`

Очищает коллекцию параметров запроса.

---

`void SetParameterValue(string name, object value)`

Устанавливает значение для параметра запроса.

## Параметры

<code>name</code>	Название параметра.
<code>value</code>	Значение.

```
void InitializeParameters()
```

Инициализирует коллекцию параметров запроса.

```
int Execute()
```

Выполняет запрос. Возвращает количество задействованных запросом записей.

```
int Execute(DBExecutor dbExecutor)
```

Выполняет запрос, используя экземпляра `DBExecutor`. Возвращает количество задействованных запросом записей.

```
Insert Into(string schemaName)
```

```
Insert Into(ModifyQuerySource source)
```

Добавляет в текущий запрос источник данных.

#### Параметры

schemaName	Название схемы.
source	Источник данных.

```
Insert Set(string sourceColumnAlias, Select subSelect)
```

```
Insert Set(string sourceColumnAlias, Query subSelectQuery)
```

```
Insert Set(string sourceColumnAlias, QueryColumnExpression columnExpression)
```

```
Insert Set(string sourceColumnAlias, QueryParameter parameter)
```

Добавляет в текущий запрос предложение `SET` для присвоения колонке переданного выражения или параметра. Возвращает текущий экземпляра `Insert`.

#### Параметры

sourceColumnAlias	Псевдоним колонки.
subSelect	Подзапрос на выборку.
subSelectQuery	Подзапрос.
columnExpression	Выражение колонки.
parameter	Параметр запроса.

Insert Values()

Инициализирует значения для множественного добавления записей.

## Класс InsertSelect C#

 Сложный

Пространство имен `Terrasoft.Core.DB`.

Класс `Terrasoft.Core.DB.InsertSelect` предназначен для построения запросов на добавление записей в таблицы базы данных Creatio. При этом в качестве источника добавляемых данных используется экземпляр класса `Terrasoft.Core.DB.Select`. В результате создания и конфигурирования экземпляра `Terrasoft.Core.DB.InsertSelect` будет построен запрос базу данных приложения в виде SQL-выражения `INSERT INTO SELECT`.

**Важно.** При работе с классом `InsertSelect` на добавленные записи не применяются права доступа по умолчанию. К таким записям не применены вообще никакие права приложения (по операциям на объект, по записям, по колонкам). Пользовательское соединение используется только для доступа к таблице базы данных.

**На заметку.** После выполнения запроса `InsertSelect` в базу данных будет добавлено столько записей, сколько вернется в его подзапросе `Select`.

**На заметку.** Полный перечень методов и свойств класса `InsertSelect`, его родительских классов, а также реализуемых им интерфейсов, можно найти в [Библиотеке .NET классов](#).

## Конструкторы

`InsertSelect(UserConnection userConnection)`

Создает экземпляр класса с указанным `UserConnection`.

`InsertSelect(InsertSelect source)`

Создает экземпляр класса, являющийся клоном экземпляра, переданного в качестве аргумента.

## Свойства

`UserConnection` `Terrasoft.Core.UserConnection`

Пользовательское подключение, используемое при запросе.

---

Source Terrasoft.Core.DB.ModifyQuerySource

Источник данных

---

Parameters Terrasoft.Core.DB.QueryParameterCollection

Коллекция параметров запроса.

---

HasParameters bool

Определяет, имеет ли запрос параметры.

---

BuildParametersAsValue bool

Определяет, добавлять ли параметры запроса в текст запроса как значения.

---

Columns Terrasoft.Core.DB.ModifyQueryColumnValueCollection

Коллекция значений колонок запроса.

---

Select Terrasoft.Core.DB.Select

Используемый в запросе экземпляр `Terrasoft.Core.DB.Select`.

## Методы

---

void ResetCachedSqlText()

Очищает кэшированный текст запроса.

---

QueryParameterCollection GetUsingParameters()

Возвращает коллекцию параметров, используемых запросом.

---

void ResetParameters()

Очищает коллекцию параметров запроса.

---

void SetParameterValue(string name, object value)

Устанавливает значение для параметра запроса.

## Параметры

name	Название параметра.
value	Значение.

```
void InitializeParameters()
```

Инициализирует коллекцию параметров запроса.

```
int Execute()
```

Выполняет запрос. Возвращает количество задействованных запросом записей.

```
int Execute(DBExecutor dbExecutor)
```

Выполняет запрос, используя экземпляр `DBExecutor`. Возвращает количество задействованных запросом записей.

```
InsertSelect Into(string schemaName)
```

```
InsertSelect Into(ModifyQuerySource source)
```

Добавляет в текущий запрос источник данных.

## Параметры

schemaName	Название схемы.
source	Источник данных.

```
InsertSelect Set(IEnumerable<string> sourceColumnAliases)
```

```
InsertSelect Set(params string[] sourceColumnAliases)
```

```
InsertSelect Set(IEnumerable<ModifyQueryColumn> columns)
```

```
InsertSelect Set(params ModifyQueryColumn[] columns)
```

Добавляет в текущий запрос набор колонок, в которые будут добавлены значения с помощью подзапроса. Возвращает текущий экземпляр `InsertSelect`.

## Параметры

sourceColumnAliases	Коллекция или массив параметров метода, содержащие псевдонимы колонок.
columns	Коллекция или массив параметров метода, содержащие экземпляры колонок.

```
InsertSelect FromSelect(Select subSelect)
InsertSelect FromSelect(Query subSelectQuery)
```

Добавляет в текущий запрос предложение `SELECT`.

### Параметры

subSelect	Подзапрос на выборку.
subSelectQuery	Подзапрос.

## Класс Update C#



Пространство имен `Terrasoft.Core.DB`.

Класс `Terrasoft.Core.DB.Update` предназначен для построения запросов на изменение записей в таблице базы данных Creatio. В результате создания и конфигурирования экземпляра этого класса будет построен запрос базу данных приложения в виде SQL-выражения `UPDATE`.

**На заметку.** Полный перечень методов и свойств класса `Update`, его родительских классов, а также реализуемых им интерфейсов, можно найти в [Библиотеке .NET классов](#).

## Конструкторы

```
Update(UserConnection userConnection)
```

Создает экземпляр класса, используя `UserConnection`.

```
Update(UserConnection userConnection, string schemaName)
```

Создает экземпляр класса для схемы с указанным названием, используя `UserConnection`.

```
Update(UserConnection userConnection, ModifyQuerySource source)
```

Создает экземпляр класса для указанного источника данных, используя `UserConnection`.

---

`Update(Insert source)`

Создает экземпляр класса, являющийся клоном экземпляра, переданного в качестве аргумента.

## Свойства

`UserConnection Terrasoft.Core.UserConnection`

Пользовательское подключение, используемое при выполнении запроса.

---

`Condition Terrasoft.Core.DB.QueryCondition`

Условие выражения `Where` запроса.

---

`HasCondition bool`

Определяет наличие выражения `Where` в запросе.

---

`Source Terrasoft.Core.DB.ModifyQuerySource`

Источник данных запроса.

---

`ColumnValues Terrasoft.Core.DB.ModifyQueryColumnValueCollection`

Коллекция значений колонок запроса.

## Методы

`void ResetCachedSqlText()`

Очищает закэшированный текст запроса.

---

`QueryParameterCollection GetUsingParameters()`

Возвращает коллекцию параметров, используемых запросом.

---

`int Execute()`

Выполняет запрос. Возвращает количество задействованных запросом записей.

---



```
int Execute(DBExecutor dbExecutor)
```

Выполняет запрос, используя экземпляр `DBExecutor`. Возвращает количество задействованных запросом записей.

```
QueryCondition Where()
QueryCondition Where(string sourceColumnAlias)
QueryCondition Where(string sourceAlias, string sourceColumnAlias)
QueryCondition Where(Select subSelect)
QueryCondition Where(Query subSelectQuery)
QueryCondition Where(QueryColumnExpression columnExpression)
Query Where(QueryCondition condition)
```

Добавляет к текущему запросу начальное условие.

#### Параметры

sourceColumnAlias	Псевдоним колонки, для которой добавляется условие.
sourceAlias	Псевдоним источника.
subSelect	Подзапрос выборки данных, для результатов которого добавляется условие.
subSelectQuery	Подзапрос, для результатов которого добавляется условие.
columnExpression	Выражение, для результатов которого добавляется условие.
condition	Условие запроса.

```
QueryCondition And()
QueryCondition And(string sourceColumnAlias)
QueryCondition And(string sourceAlias, string sourceColumnAlias)
QueryCondition And(Select subSelect)
QueryCondition And(Query subSelectQuery)
QueryCondition And(QueryParameter parameter)
QueryCondition And(QueryColumnExpression columnExpression)
Query And(QueryCondition condition)
```

К текущему условию запроса добавляет условие (предикат), используя логическую операцию И.

#### Параметры

sourceColumnAlias	Псевдоним колонки, для которой добавляется предикат.
sourceAlias	Псевдоним источника.
subSelect	Подзапрос выборки данных, используемый в качестве предиката.
subSelectQuery	Подзапрос, используемый в качестве предиката.
parameter	Параметр, для которого добавляется предикат.
columnExpression	Выражение, используемое в качестве предиката.
condition	Условие запроса.

```

QueryCondition Or()
QueryCondition Or(string sourceColumnAlias)
QueryCondition Or(string sourceAlias, string sourceColumnAlias)
QueryCondition Or(Select subSelect)
QueryCondition Or(Query subSelectQuery)
QueryCondition Or(QueryParameter parameter)
QueryCondition Or(QueryColumnExpression columnExpression)
Query Or(QueryCondition condition)

```

К текущему условию запроса добавляет условие (предикат), используя логическую операцию ИЛИ.

### Параметры

sourceColumnAlias	Псевдоним колонки, для которой добавляется предикат.
sourceAlias	Псевдоним источника.
subSelect	Подзапрос на выборку данных, используемый в качестве предиката.
subSelectQuery	Подзапрос, используемый в качестве предиката.
parameter	Параметр, для которого добавляется предикат.
columnExpression	Выражение, используемое в качестве предиката.
condition	Условие запроса.

```
Update Set(string sourceColumnAlias, Select subSelect)
```

```
Update Set(string sourceColumnAlias, Query subSelectQuery)
Update Set(string sourceColumnAlias, QueryColumnExpression columnExpression)
Update Set(string sourceColumnAlias, QueryParameter parameter)
```

Добавляет в текущий запрос предложение `SET` для присвоения колонке переданного выражения или параметра. Возвращает текущий экземпляр `Update`.

### Параметры

<code>sourceColumnAlias</code>	Псевдоним колонки.
<code>subSelect</code>	Подзапрос на выборку.
<code>subSelectQuery</code>	Подзапрос.
<code>columnExpression</code>	Выражение колонки.
<code>parameter</code>	Параметр запроса.

## Класс UpdateSelect C#



Пространство имен `Terrasoft.Core.DB`.

Класс `Terrasoft.Core.DB.UpdateSelect` предназначен для построения запросов на изменение записей в таблице базы данных Creatio. При этом в качестве источника добавляемых данных используется экземпляр класса `Terrasoft.Core.DB.Select`. В результате создания и конфигурирования экземпляра этого класса будет построен запрос в базу данных приложения в виде SQL-выражения `UPDATE FROM`.

**На заметку.** Полный перечень методов и свойств класса `UpdateSelect`, его родительских классов, а также реализуемых им интерфейсов, можно найти в [Библиотеке .NET классов](#).

## Конструкторы

```
public UpdateSelect(UserConnection userConnection, string schemaName, string alias)
```

Создает экземпляр класса для схемы с указанным названием, используя `UserConnection`.

### Параметры

userConnection	Пользовательское подключение, используемое при запросе.
schemaName	Название схемы.
alias	Псевдоним таблицы.

## Свойства

SourceAlias `string`

Псевдоним таблицы данных, в которую вносятся изменения.

SourceExpression `Terrasoft.Core.DB.QuerySourceExpression`

Выражение для `SELECT`-запроса.

## Методы

`public UpdateSelect From(string schemaName, string alias)`

Добавляет к запросу `FROM`-выражение.

### Параметры

schemaName	Название таблицы, в которую вносятся изменения.
alias	Псевдоним таблицы, в которую вносятся изменения.

`public UpdateSelect Set(string sourceColumnAlias, QueryColumnExpression columnExpression)`

Добавляет к запросу `SET`-выражение для колонки.

### Параметры

sourceColumnAlias	Псевдоним колонки.
columnExpression	Выражение, содержащее значение колонки.

# Класс Delete C#



Сложный

Пространство имен `Terrasoft.Core.DB` .

Класс `Terrasoft.Core.DB.Delete` предназначен для построения запросов на удаление записей в таблице базы данных Creatio. В результате создания и конфигурирования экземпляра этого класса будет построен запрос базу данных приложения в виде SQL-выражения `DELETE` .

**На заметку.** Полный перечень методов и свойств класса `Delete` , его родительских классов, а также реализуемых им интерфейсов, можно найти в [Библиотеке .NET классов](#).

## Конструкторы

`Delete(UserConnection userConnection)`

Создает экземпляр класса, используя `UserConnection` .

`Delete>Delete source)`

Создает экземпляр класса, являющийся клоном экземпляра, переданного в качестве аргумента.

## Свойства

`UserConnection Terrasoft.Core.UserConnection`

Пользовательское подключение, используемое при выполнении запроса.

`Condition Terrasoft.Core.DB.QueryCondition`

Условие выражения `Where` запроса.

`HasCondition bool`

Определяет наличие выражения `Where` в запросе.

`Source Terrasoft.Core.DB.ModifyQuerySource`

Источник данных запроса.

## Методы

`void ResetCachedSqlText()`

Очищает закэшированный текст запроса.

---

```
QueryParameterCollection GetUsingParameters()
```

Возвращает коллекцию параметров, используемых запросом.

---

```
int Execute()
```

Выполняет запрос. Возвращает количество задействованных запросом записей.

---

```
int Execute(DBExecutor dbExecutor)
```

Выполняет запрос, используя экземпляр `DBExecutor`. Возвращает количество задействованных запросом записей.

---

```
QueryCondition Where()
```

```
QueryCondition Where(string sourceColumnAlias)
```

```
QueryCondition Where(string sourceAlias, string sourceColumnAlias)
```

```
QueryCondition Where(Select subSelect)
```

```
QueryCondition Where(Query subSelectQuery)
```

```
QueryCondition Where(QueryColumnExpression columnExpression)
```

```
Query Where(QueryCondition condition)
```

Добавляет к текущему запросу начальное условие.

### Параметры

sourceColumnAlias	Псевдоним колонки, для которой добавляется условие.
sourceAlias	Псевдоним источника.
subSelect	Подзапрос выборки данных, для результатов которого добавляется условие.
subSelectQuery	Подзапрос, для результатов которого добавляется условие.
columnExpression	Выражение, для результатов которого добавляется условие.
condition	Условие запроса.

---

```
QueryCondition And()
```

```
QueryCondition And(string sourceColumnAlias)
```

```
QueryCondition And(string sourceAlias, string sourceColumnAlias)
```

```
QueryCondition And(Select subSelect)
```

```
QueryCondition And(Query subSelectQuery)
```

```
QueryCondition And(QueryParameter parameter)
```

```
QueryCondition And(QueryColumnExpression columnExpression)
```

```
Query And(QueryCondition condition)
```

К текущему условию запроса добавляет условие (предикат), используя логическую операцию И.

#### Параметры

sourceColumnAlias	Псевдоним колонки, для которой добавляется предикат.
sourceAlias	Псевдоним источника.
subSelect	Подзапрос выборки данных, используемый в качестве предиката.
subSelectQuery	Подзапрос, используемый в качестве предиката.
parameter	Параметр, для которого добавляется предикат.
columnExpression	Выражение, используемое в качестве предиката.
condition	Условие запроса.

```
QueryCondition Or()
```

```
QueryCondition Or(string sourceColumnAlias)
```

```
QueryCondition Or(string sourceAlias, string sourceColumnAlias)
```

```
QueryCondition Or(Select subSelect)
```

```
QueryCondition Or(Query subSelectQuery)
```

```
QueryCondition Or(QueryParameter parameter)
```

```
QueryCondition Or(QueryColumnExpression columnExpression)
```

```
Query Or(QueryCondition condition)
```

К текущему условию запроса добавляет условие (предикат), используя логическую операцию ИЛИ.

#### Параметры

sourceColumnAlias	Псевдоним колонки, для которой добавляется предикат.
sourceAlias	Псевдоним источника.
subSelect	Подзапрос на выборку данных, используемый в качестве предиката.
subSelectQuery	Подзапрос, используемый в качестве предиката.
parameter	Параметр, для которого добавляется предикат.
columnExpression	Выражение, используемое в качестве предиката.
condition	Условие запроса.

```
Delete From(string schemaName)
```

```
Delete From((ModifyQuerySource source)
```

Добавляет в текущий запрос источник данных. Возвращает текущий экземпляр `Delete` .

#### Параметры

schemaName	Название схемы (таблицы, представления).
source	Источник данных.

## Класс QueryFunction C#



Класс `Terrasoft.Core.DB.QueryFunction` реализует функцию выражения.

Функция выражения реализована в следующих классах:

- `QueryFunction` — базовый класс функции выражения.
- `AggregationQueryFunction` — реализует агрегирующую функцию выражения.
- `IsNullQueryFunction` — заменяет значения `null` замещающим выражением.
- `CreateGuidQueryFunction` — реализует функцию выражения нового идентификатора.
- `CurrentDateTimeQueryFunction` — реализует функцию выражения текущей даты и времени.
- `CoalesceQueryFunction` — возвращает первое выражение из списка аргументов, не равное `null` .
- `DatePartQueryFunction` — реализует функцию выражения части значения типа `Дата/Время` .
- `DateAddQueryFunction` — реализует функцию выражения даты, полученной путем добавления указанного промежутка времени к заданной дате.



- `DateDiffQueryFunction` — реализует функцию выражения разницы дат, полученного путем вычитания заданных дат.
- `CastQueryFunction` — приводит выражение аргумента к заданному типу данных.
- `UpperQueryFunction` — преобразовывает символы выражения аргумента в верхний регистр.
- `CustomQueryFunction` — реализует пользовательскую функцию.
- `DataLengthQueryFunction` — определяет число байтов, использованных для представления выражения.
- `TrimQueryFunction` — удаляет начальные и конечные пробелы из выражения.
- `LengthQueryFunction` — возвращает длину выражения.
- `SubstringQueryFunction` — получает часть строки.
- `ConcatQueryFunction` — формирует строку, которая является результатом объединения строковых значений аргументов функции.
- `WindowQueryFunction` — реализует функцию SQL окна.

## Класс QueryFunction C#

Пространство имен `Terrasoft.Core.DB`.

Базовый класс функции выражения.

**На заметку.** Полный перечень методов класса `QueryFunction`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

## Методы

```
static QueryColumnExpression Negate(QueryFunction operand)
```

Возвращает выражение отрицания значения переданной функции.

### Параметры

operand	Функция выражения.
---------	--------------------

```
static QueryColumnExpression operator -(QueryFunction operand)
```

Перегрузка оператора отрицания переданной функции выражения.

### Параметры

operand	Функция выражения.
---------	--------------------

```
static QueryColumnExpression Add(QueryFunction leftOperand, QueryFunction rightOperand)
```

Возвращает выражение арифметического сложения переданных функций выражения.

#### Параметры

leftOperand	Левый операнд в операции сложения.
rightOperand	Правый операнд в операции сложения.

```
static QueryColumnExpression operator +(QueryFunction leftOperand, QueryFunction rightOperand)
```

Перегрузка оператора сложения двух функций выражений.

#### Параметры

leftOperand	Левый операнд в операции сложения.
rightOperand	Правый операнд в операции сложения.

```
static QueryColumnExpression Subtract(QueryFunction leftOperand, QueryFunction rightOperand)
```

Возвращает выражение вычитания правой функции выражения из левой.

#### Параметры

leftOperand	Левый операнд в операции вычитания.
rightOperand	Правый операнд в операции вычитания.

```
static QueryColumnExpression operator -(QueryFunction leftOperand, QueryFunction rightOperand)
```

Перегрузка оператора вычитания правой функции выражения из левой.

#### Параметры

leftOperand	Левый операнд в операции вычитания.
rightOperand	Правый операнд в операции вычитания.

```
static QueryColumnExpression Multiply(QueryFunction leftOperand, QueryFunction rightOperand)
```

Возвращает выражение умножения переданных функций выражений.

### Параметры

leftOperand	Левый операнд в операции умножения.
rightOperand	Правый операнд в операции умножения.

```
static QueryColumnExpression operator *(QueryFunction leftOperand, QueryFunction rightOperand)
```

Перегрузка оператора умножения двух функций выражений.

### Параметры

leftOperand	Левый операнд в операции умножения.
rightOperand	Правый операнд в операции умножения.

```
static QueryColumnExpression Divide(QueryFunction leftOperand, QueryFunction rightOperand)
```

Возвращает выражение деления левой функции выражения на правую.

### Параметры

leftOperand	Левый операнд в операции деления.
rightOperand	Правый операнд в операции деления.

```
static QueryColumnExpression operator /(QueryFunction leftOperand, QueryFunction rightOperand)
```

Перегрузка оператора деления функций выражений.

### Параметры

leftOperand	Левый операнд в операции деления.
rightOperand	Правый операнд в операции деления.

```
abstract object Clone()
```

Создает копию текущего экземпляра `QueryFunction`.

```
abstract void BuildSqlText(StringBuilder sb, DBEngine dbEngine)
```

Формирует текст запроса с использованием переданных экземпляра `StringBuilder` и строителя запросов `DBEngine`.

#### Параметры

sb	Экземпляр <code>StringBuilder</code> , с помощью которого формируется текст запроса.
dbEngine	Экземпляр строителя запросов к базе данных.

```
virtual void AddUsingParameters(QueryParameterCollection resultParameters)
```

Добавляет переданную коллекцию параметров в аргументы функции.

#### Параметры

resultParameters	Коллекция параметров запроса, которые добавляются в аргументы функции.
------------------	--

```
QueryColumnExpressionCollection GetQueryColumnExpressions()
```

Возвращает коллекцию выражений колонки запроса для текущей функции запроса.

```
QueryColumnExpression GetQueryColumnExpression()
```

Возвращает выражение колонки запроса для текущей функции запроса.

## Класс AggregationQueryFunction C#

Пространство имен `Terrasoft.Core.DB`.

Класс реализует агрегирующую функцию выражения.

**На заметку.** Полный перечень методов и свойств класса `AggregationQueryFunction`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

## Конструкторы

```
AggregationQueryFunction()
```

Инициализирует новый экземпляр `AggregationQueryFunction`.

`AggregationQueryFunction(AggregationTypeStrict aggregationType, QueryColumnExpression expression)`  
Инициализирует новый экземпляр `AggregationQueryFunction` с заданным типом агрегирующей функции для указанного выражения колонки.

Параметры

<code>aggregationType</code>	Тип агрегирующей функции.
<code>expression</code>	Выражение колонки, к которому применяется агрегирующая функция.

`AggregationQueryFunction(AggregationTypeStrict aggregationType, IQueryColumnExpressionConvertible expression)`  
Инициализирует новый экземпляр `AggregationQueryFunction` с заданным типом агрегирующей функции для указанного выражения колонки.

Параметры

<code>aggregationType</code>	Тип агрегирующей функции.
<code>expression</code>	Выражение колонки, к которому применяется агрегирующая функция.

`AggregationQueryFunction(AggregationQueryFunction source)`  
Инициализирует новый экземпляр `AggregationQueryFunction`, являющийся клоном переданной агрегирующей функции выражения.

Параметры

<code>source</code>	Агрегирующая функция выражения <code>AggregationQueryFunction</code> , клон которой создается.
---------------------	--

Свойства

`AggregationType` `AggregationTypeStrict`  
Тип агрегирующей функции.

`AggregationEvalType` `AggregationEvalType`  
Область применения агрегирующей функции.

Expression QueryColumnExpression

Выражение аргумента функции.

## Методы

```
override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)
```

Формирует текст запроса с использованием заданных экземпляров `StringBuilder` и строителя запроса `DBEngine`.

### Параметры

sb	Экземпляр <code>StringBuilder</code> , с помощью которого формируется текст запроса.
dbEngine	Экземпляр строителя запросов к базе данных.

```
override void AddUsingParameters(QueryParameterCollection resultParameters)
```

Добавляет переданную коллекцию параметров в аргументы функции.

### Параметры

resultParameters	Коллекция параметров запроса, которые добавляются в аргументы функции.
------------------	--

```
override object Clone()
```

Создает клон текущего экземпляра `AggregationQueryFunction`.

```
AggregationQueryFunction All()
```

Устанавливает для текущей агрегирующей функции область применения [ *Ко всем значениям* ].

```
AggregationQueryFunction Distinct()
```

Устанавливает для текущей агрегирующей функции область применения [ *К уникальным значениям* ].

## Класс IsNullQueryFunction C#

Пространство имен `Terrasoft.Core.DB`.

Класс заменяет значения `null` замещающим выражением.

**На заметку.** Полный перечень методов и свойств класса `IsNullQueryFunction`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

## Конструкторы

`IsNullQueryFunction()`

Инициализирует новый экземпляр `IsNullQueryFunction`.

`IsNullQueryFunction(QueryColumnExpression checkExpression, QueryColumnExpression replacementExpr)`  
`IsNullQueryFunction(IQueryColumnExpressionConvertible checkExpression, IQueryColumnExpressionCor`

Инициализирует новый экземпляр `IsNullQueryFunction` для заданных проверяемого выражения и замещающего выражения.

### Параметры

<code>checkExpression</code>	Выражение, которое проверяется на равенство <code>null</code> .
<code>replacementExpression</code>	Выражение, которое возвращается функцией, если <code>checkExpression</code> равно <code>null</code> .

`IsNullQueryFunction(IsNullQueryFunction source)`

Инициализирует новый экземпляр `IsNullQueryFunction`, являющийся клоном переданной функции выражения.

### Параметры

<code>source</code>	Агрегирующая функция выражения <code>IsNullQueryFunction</code> , клон которой создается.
---------------------	---

## Свойства

`CheckExpression` `QueryColumnExpression`

Выражение аргумента функции, которое проверяется на равенство значению `null`.

ReplacementExpression QueryColumnExpression

Выражение аргумента функции, которое возвращается, если проверяемое выражение равно `null`.

## Методы

```
override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)
```

Формирует текст запроса с использованием заданных экземпляров `StringBuilder` и строителя запросов `DBEngine`.

### Параметры

sb	Экземпляр <code>StringBuilder</code> , с помощью которого формируется текст запроса.
dbEngine	Экземпляр строителя запросов к базе данных.

```
override void AddUsingParameters(QueryParameterCollection resultParameters)
```

Добавляет переданную коллекцию параметров в аргументы функции.

### Параметры

resultParameters	Коллекция параметров запроса, которые добавляются в аргументы функции.
------------------	--

```
override object Clone()
```

Создает клон текущего экземпляра `IsNullQueryFunction`.

## Класс CreateGuidQueryFunction C#

Пространство имен `Terrasoft.Core.DB`.

Класс реализует функцию выражения нового идентификатора.

**На заметку.** Полный перечень методов класса `CreateGuidQueryFunction`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

## Конструкторы



CreateGuidIdQueryFunction()

Инициализирует новый экземпляр `CreateGuidIdQueryFunction`.

CreateGuidIdQueryFunction(CreateGuidIdQueryFunction source)

Инициализирует новый экземпляр `CreateGuidIdQueryFunction`, являющийся клоном переданной функции.

#### Параметры

source	Функция <code>CreateGuidIdQueryFunction</code> , клон которой создается.
--------	--

## Методы

override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)

Формирует текст запроса с использованием заданных экземпляров `StringBuilder` и построителя запросов `DBEngine`.

#### Параметры

sb	Экземпляр <code>StringBuilder</code> , с помощью которого формируется текст запроса.
dbEngine	Экземпляр построителя запроса к базе данных.

override object Clone()

Создает клон текущего экземпляра `CreateGuidIdQueryFunction`.

## Класс CurrentDateTimeQueryFunction C#

Пространство имен `Terrasoft.Core.DB`.

Класс реализует функцию выражения текущей даты и времени.

**На заметку.** Полный перечень методов класса `CurrentDateTimeQueryFunction`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

## Конструкторы

```
CurrentDateTimeQueryFunction()
```

Инициализирует новый экземпляр `CurrentDateTimeQueryFunction`.

```
CurrentDateTimeQueryFunction(CurrentDateTimeQueryFunction source)
```

Инициализирует новый экземпляр `CurrentDateTimeQueryFunction`, являющийся клоном переданной функции.

#### Параметры

source	Функция <code>CurrentDateTimeQueryFunction</code> , клон которой создается.
--------	---

## Методы

```
override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)
```

Формирует текст запроса с использованием заданных экземпляров `StringBuilder` и построителя запросов `DBEngine`.

#### Параметры

sb	Экземпляр <code>StringBuilder</code> , с помощью которого формируется текст запроса.
dbEngine	Экземпляр построителя запроса к базе данных.

```
override object Clone()
```

Создает клон текущего экземпляра `CurrentDateTimeQueryFunction`.

## Класс CoalesceQueryFunction C#

Пространство имен `Terrasoft.Core.DB`.

Класс возвращает первое выражение из списка аргументов, не равное `null`.

**На заметку.** Полный перечень методов и свойств класса `CoalesceQueryFunction`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

## Конструкторы

```
CoalesceQueryFunction()
```

Инициализирует новый экземпляр `CoalesceQueryFunction`.

```
CoalesceQueryFunction(CoalesceQueryFunction source)
```

Инициализирует новый экземпляр `CoalesceQueryFunction`, являющийся клоном переданной функции.

#### Параметры

source	Функция <code>CoalesceQueryFunction</code> , клон которой создается.
--------	--

```
CoalesceQueryFunction(QueryColumnExpressionCollection expressions)
```

Инициализирует новый экземпляр `CoalesceQueryFunction` для переданной коллекции выражений колонок.

#### Параметры

expressions	Коллекция выражений колонок запроса.
-------------	--------------------------------------

```
CoalesceQueryFunction(QueryColumnExpression[] expressions)
```

```
CoalesceQueryFunction(IQueryColumnExpressionConvertible[] expressions)
```

Инициализирует новый экземпляр `CoalesceQueryFunction` для переданного массива выражений колонок.

#### Параметры

expressions	Массив выражений колонок запроса.
-------------	-----------------------------------

## Свойства

Expressions `QueryColumnExpressionCollection`

Коллекция выражений аргументов функции.

## Методы

```
override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)
```

Формирует текст запроса с использованием заданных экземпляров `StringBuilder` и построителя запросов `DBEngine`.

### Параметры

sb	Экземпляр <code>StringBuilder</code> , с помощью которого формируется текст запроса.
dbEngine	Экземпляр построителя запроса к базе данных.

```
override object Clone()
```

Создает клон текущего экземпляра `CoalesceQueryFunction`.

```
override void AddUsingParameters(QueryParameterCollection resultParameters)
```

Добавляет заданные параметры в коллекцию.

### Параметры

resultParameters	Коллекция параметров запроса, которые добавляются в аргументы функции.
------------------	--

## Класс DatePartQueryFunction C#

Пространство имен `Terrasoft.Core.DB`.

Класс реализует функцию выражения части значения типа `Дата/Время`.

**На заметку.** Полный перечень методов и свойств класса `DatePartQueryFunction`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

## Конструкторы

```
DatePartQueryFunction()
```

Инициализирует новый экземпляр `DatePartQueryFunction`.

```
DatePartQueryFunction(DatePartQueryFunctionInterval interval, QueryColumnExpression expression)
DatePartQueryFunction(DatePartQueryFunctionInterval interval, IQueryColumnExpressionConvertible
```

Инициализирует новый экземпляр `DatePartQueryFunction` с заданным выражением колонки типа `Дата/Время` и указанной частью даты.

### Параметры

<code>interval</code>	Часть даты.
<code>expression</code>	Выражение колонки типа <code>Дата/Время</code> .

```
DatePartQueryFunction(DatePartQueryFunction source)
```

Инициализирует новый экземпляр `DatePartQueryFunction`, являющийся клоном переданной функции.

### Параметры

<code>source</code>	Функция <code>DatePartQueryFunction</code> , клон которой создается.
---------------------	--

## Свойства

```
Expression QueryColumnExpression
```

Выражение аргумента функции.

```
Interval DatePartQueryFunctionInterval
```

Часть даты, возвращаемая функцией.

```
UseUtcOffset bool
```

Использование смещения всеобщего скоординированного времени (UTC) относительно заданного местного времени.

```
UtcOffset int?
```

Смещение всеобщего скоординированного времени (UTC).

## Методы

```
override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)
```

Формирует текст запроса с использованием заданных экземпляров `StringBuilder` и построителя запросов `DBEngine`.

### Параметры

sb	Экземпляр <code>StringBuilder</code> , с помощью которого формируется текст запроса.
dbEngine	Экземпляр построителя запроса к базе данных.

```
override void AddUsingParameters(QueryParameterCollection resultParameters)
```

Добавляет заданные параметры в коллекцию.

#### Параметры

resultParameters	Коллекция параметров запроса, которые добавляются в аргументы функции.
------------------	--

```
override object Clone()
```

Создает клон текущего экземпляра `DatePartQueryFunction`.

## Класс DateAddQueryFunction C#

Пространство имен `Terrasoft.Core.DB`.

Класс реализует функцию выражения даты, полученной путем добавления указанного промежутка времени к заданной дате.

**На заметку.** Полный перечень методов и свойств класса `DateAddQueryFunction`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

## Конструкторы

```
DateAddQueryFunction()
```

Инициализирует новый экземпляр `DateAddQueryFunction`.

```
DateAddQueryFunction(DatePartQueryFunctionInterval interval, int number, QueryColumnExpression e
DateAddQueryFunction(DatePartQueryFunctionInterval interval, IQueryColumnExpressionConvertible r
DateAddQueryFunction(DatePartQueryFunctionInterval interval, int number, IQueryColumnExpressionC
```

Инициализирует экземпляр `DateAddQueryFunction` с заданными параметрами.

#### Параметры

<code>interval</code>	Часть даты, к которой добавляется временной промежуток.
<code>number</code>	Значение, которое добавляется к <code>interval</code> .
<code>expression</code>	Выражение колонки, содержащей исходную дату.

`DateAddQueryFunction(DateAddQueryFunction source)`

Инициализирует экземпляр `DateAddQueryFunction`, являющийся клоном переданной функции.

### Параметры

<code>source</code>	Экземпляр функции <code>DateAddQueryFunction</code> , клон которой создается.
---------------------	---

## Свойства

`Expression QueryColumnExpression`

Выражение колонки, содержащей исходную дату.

`Interval DatePartQueryFunctionInterval`

Часть даты, к которой добавляется временной промежуток.

`Number int`

Добавляемый временной промежуток.

`NumberExpression QueryColumnExpression`

Выражение, содержащие добавляемый временной промежуток.

## Методы

`override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)`

Формирует текст запроса с использованием заданных экземпляров `StringBuilder` и построителя запросов `DBEngine`.

### Параметры

sb	Экземпляр <code>StringBuilder</code> , с помощью которого формируется текст запроса.
dbEngine	Экземпляр построителя запроса к базе данных.

```
override void AddUsingParameters(QueryParameterCollection resultParameters)
```

Добавляет заданные параметры в коллекцию.

#### Параметры

resultParameters	Коллекция параметров запроса, которые добавляются в аргументы функции.
------------------	--

```
override object Clone()
```

Создает клон текущего экземпляра `DateAddQueryFunction`.

## Класс DateDiffQueryFunction C#

Пространство имен `Terrasoft.Core.DB`.

Класс реализует функцию выражения разницы дат, полученного путем вычитания заданных дат.

**На заметку.** Полный перечень методов и свойств класса `DateDiffQueryFunction`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

## Конструкторы

```
DateDiffQueryFunction(DateDiffQueryFunctionInterval interval, QueryColumnExpression startDateExp  
DateDiffQueryFunction(DateDiffQueryFunctionInterval interval, IQueryColumnExpressionConvertible
```

Инициализирует экземпляр `DateDiffQueryFunction` с заданными параметрами.

#### Параметры

interval	Единица измерения разницы дат.
startDateExpression	Выражение колонки, содержащей начальную дату.
endDateExpression	Выражение колонки, содержащей конечную дату.



```
DateDiffQueryFunction(DateDiffQueryFunction source)
```

Инициализирует экземпляр `DateDiffQueryFunction`, являющийся клоном переданной функции.

#### Параметры

source	Экземпляр функции <code>DateDiffQueryFunction</code> , клон которой создается.
--------	--

## Свойства

```
StartDateExpression QueryColumnExpression
```

Выражение колонки, содержащей начальную дату.

```
EndDateExpression QueryColumnExpression
```

Выражение колонки, содержащей конечную дату.

```
Interval DateDiffQueryFunctionInterval
```

Единица измерения разницы дат, возвращаемая функцией.

## Методы

```
override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)
```

Формирует текст запроса с использованием заданных экземпляров `StringBuilder` и строителя запросов `DBEngine`.

#### Параметры

sb	Экземпляр <code>StringBuilder</code> , с помощью которого формируется текст запроса.
dbEngine	Экземпляр строителя запроса к базе данных.

```
override void AddUsingParameters(QueryParameterCollection resultParameters)
```

Добавляет заданные параметры в коллекцию.

#### Параметры

resultParameters

Коллекция параметров запроса, которые добавляются в аргументы функции.

```
override object Clone()
```

Создает клон текущего экземпляра `DateDiffQueryFunction`.

## Класс CastQueryFunction C#

Пространство имен `Terrasoft.Core.DB`.

Класс приводит выражение аргумента к заданному типу данных.

**На заметку.** Полный перечень методов и свойств класса `CastQueryFunction`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

## Конструкторы

```
CastQueryFunction(QueryColumnExpression expression, DBDataValueType castType)
```

```
CastQueryFunction(IQueryColumnExpressionConvertible expression, DBDataValueType castType)
```

Инициализирует новый экземпляр `CastQueryFunction` с заданными выражением колонки и целевым типом данных.

### Параметры

expression	Выражение колонки запроса.
castType	Целевой тип данных.

```
CastQueryFunction(CastQueryFunction source)
```

Инициализирует новый экземпляр `CastQueryFunction`, являющийся клоном переданной функции.

### Параметры

source	Функция <code>CastQueryFunction</code> , клон которой создается.
--------	--

## Свойства

```
Expression QueryColumnExpression
```

Выражение аргумента функции.

CastType DBDataValueType

Целевой тип данных.

## Методы

```
override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)
```

Формирует текст запроса с использованием заданных экземпляров `StringBuilder` и строителя запросов `DBEngine`.

### Параметры

sb	Экземпляр <code>StringBuilder</code> , с помощью которого формируется текст запроса.
dbEngine	Экземпляр строителя запроса к базе данных.

```
override void AddUsingParameters(QueryParameterCollection resultParameters)
```

Добавляет заданные параметры в коллекцию.

### Параметры

resultParameters	Коллекция параметров запроса, которые добавляются в аргументы функции.
------------------	--

```
override object Clone()
```

Создает клон текущего экземпляра `CastQueryFunction`.

## Класс UpperQueryFunction C#

Пространство имен `Terrasoft.Core.DB`.

Класс преобразовывает символы выражения аргумента в верхний регистр.

**На заметку.** Полный перечень методов и свойств класса `UpperQueryFunction`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

## Конструкторы

---

```
UpperQueryFunction()
```

Инициализирует новый экземпляр `UpperQueryFunction` .

---

```
UpperQueryFunction(QueryColumnExpression expression)
```

```
UpperQueryFunction(IQueryColumnExpressionConvertible expression)
```

Инициализирует новый экземпляр `UpperQueryFunction` для заданного выражения колонки.

#### Параметры

<code>expression</code>	Выражение колонки запроса.
-------------------------	----------------------------

---

```
UpperQueryFunction(UpperQueryFunction source)
```

Инициализирует новый экземпляр `UpperQueryFunction` , являющийся клоном переданной функции.

#### Параметры

<code>source</code>	Функция <code>UpperQueryFunction</code> , клон которой создается.
---------------------	---

## Свойства

---

`Expression`

Выражение аргумента функции.

## Методы

---

```
override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)
```

Формирует текст запроса с использованием заданных экземпляров `StringBuilder` и строителя запросов `DBEngine` .

#### Параметры

<code>sb</code>	Экземпляр <code>StringBuilder</code> , с помощью которого формируется текст запроса.
<code>dbEngine</code>	Экземпляр строителя запроса к базе данных.

```
override void AddUsingParameters(QueryParameterCollection resultParameters)
```

Добавляет заданные параметры в коллекцию.

#### Параметры

resultParameters	Коллекция параметров запроса, которые добавляются в аргументы функции.
------------------	--

```
override object Clone()
```

Создает клон текущего экземпляра `UpperQueryFunction`.

## Класс CustomQueryFunction C#

Пространство имен `Terrasoft.Core.DB`.

Класс реализует пользовательскую функцию.

**На заметку.** Полный перечень методов и свойств класса `CustomQueryFunction`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

## Конструкторы

```
CustomQueryFunction()
```

Инициализирует новый экземпляр `CustomQueryFunction`.

```
CustomQueryFunction(string functionName, QueryColumnExpressionCollection expressions)
```

Инициализирует новый экземпляр `CustomQueryFunction` для заданной функции и переданной коллекции выражений колонок.

#### Параметры

functionName	Имя функции.
expressions	Коллекция выражений колонок запроса.

```
CustomQueryFunction(string functionName, QueryColumnExpression[] expressions)
```

```
CustomQueryFunction(string functionName, IQueryColumnExpressionConvertible[] expressions)
```

Инициализирует новый экземпляр `CustomQueryFunction` для заданной функции и переданного массива выражений колонок.

#### Параметры

functionName	Имя функции.
expressions	Массив выражений колонок запроса.

```
CustomQueryFunction(CustomQueryFunction source)
```

Инициализирует новый экземпляр `CustomQueryFunction`, являющийся клоном переданной функции.

#### Параметры

source	Функция <code>CustomQueryFunction</code> , клон которой создается.
--------	--

## Свойства

Expressions `QueryColumnExpressionCollection`

Коллекция выражений аргументов функции.

FunctionName `string`

Имя функции.

## Методы

```
override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)
```

Формирует текст запроса с использованием заданных экземпляров `StringBuilder` и построителя запросов `DBEngine`.

#### Параметры

sb	Экземпляр <code>StringBuilder</code> , с помощью которого формируется текст запроса.
dbEngine	Экземпляр построителя запроса к базе данных.

```
override void AddUsingParameters(QueryParameterCollection resultParameters)
```

Добавляет заданные параметры в коллекцию.

#### Параметры

resultParameters	Коллекция параметров запроса, которые добавляются в аргументы функции.
------------------	--

`override object Clone()`

Создает клон текущего экземпляра `CustomQueryFunction`.

## Класс DataLengthQueryFunction C#

Пространство имен `Terrasoft.Core.DB`.

Класс определяет число байтов, использованных для представления выражения.

**На заметку.** Полный перечень методов и свойств класса `DataLengthQueryFunction`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

## Конструкторы

`DataLengthQueryFunction()`

Инициализирует новый экземпляр `DataLengthQueryFunction`.

`DataLengthQueryFunction(QueryColumnExpression expression)`

Инициализирует новый экземпляр `DataLengthQueryFunction` для заданного выражения колонки.

#### Параметры

expression	Выражение колонки запроса.
------------	----------------------------

`DataLengthQueryFunction(IQueryColumnExpressionConvertible columnNameExpression)`

Инициализирует новый экземпляр `DataLengthQueryFunction` для заданного выражения колонки.

#### Параметры

columnNameExpression	Выражение колонки запроса.
----------------------	----------------------------

```
DataLengthQueryFunction(DataLengthQueryFunction source)
```

Инициализирует новый экземпляр `DataLengthQueryFunction`, являющийся клоном переданной функции.

#### Параметры

source	Функция <code>DataLengthQueryFunction</code> , клон которой создается.
--------	--

## Свойства

```
Expression QueryColumnExpression
```

Выражение аргумента функции.

## Методы

```
override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)
```

Формирует текст запроса с использованием заданных экземпляров `StringBuilder` и построителя запросов `DBEngine`.

#### Параметры

sb	Экземпляр <code>StringBuilder</code> , с помощью которого формируется текст запроса.
dbEngine	Экземпляр построителя запроса к базе данных.

```
override void AddUsingParameters(QueryParameterCollection resultParameters)
```

Добавляет в аргументы функции переданную коллекцию параметров.

#### Параметры

resultParameters	Коллекция параметров запроса, которые добавляются в аргументы функции.
------------------	--

```
override object Clone()
```

Создает клон текущего экземпляра `DataLengthQueryFunction`.

## Класс TrimQueryFunction



Пространство имен `Terrasoft.Core.DB` .

Класс удаляет начальные и конечные пробелы из выражения.

**На заметку.** Полный перечень методов и свойств класса `TrimQueryFunction` , его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

## Конструкторы

```
TrimQueryFunction(QueryColumnExpression expression)
TrimQueryFunction(IQueryColumnExpressionConvertible expression)
```

Инициализирует новый экземпляр `TrimQueryFunction` для заданного выражения колонки.

### Параметры

expression	Выражение колонки запроса.
------------	----------------------------

```
TrimQueryFunction(TrimQueryFunction source)
```

Инициализирует новый экземпляр `TrimQueryFunction` , являющийся клоном переданной функции.

### Параметры

source	Функция <code>TrimQueryFunction</code> , клон которой создается.
--------	--

## Свойства

```
Expression QueryColumnExpression
```

Выражение аргумента функции.

## Методы

```
override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)
```

Формирует текст запроса с использованием заданных экземпляров `StringBuilder` и построителя запросов `DBEngine` .

### Параметры

sb	Экземпляр <code>StringBuilder</code> , с помощью которого формируется текст запроса.
dbEngine	Экземпляр построителя запроса к базе данных.

```
override void AddUsingParameters(QueryParameterCollection resultParameters)
```

Добавляет в аргументы функции переданную коллекцию параметров.

#### Параметры

resultParameters	Коллекция параметров запроса, которые добавляются в аргументы функции.
------------------	--

```
override object Clone()
```

Создает клон текущего экземпляра `TrimQueryFunction`.

## Класс LengthQueryFunction C#

Пространство имен `Terrasoft.Core.DB`.

Класс возвращает длину выражения.

**На заметку.** Полный перечень методов и свойств класса `LengthQueryFunction`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

## Конструкторы

```
LengthQueryFunction()
```

Инициализирует новый экземпляр `LengthQueryFunction`.

```
LengthQueryFunction(QueryColumnExpression expression)
```

```
LengthQueryFunction(IQueryColumnExpressionConvertible expression)
```

Инициализирует новый экземпляр `LengthQueryFunction` для заданного выражения колонки.

#### Параметры

expression	Выражение колонки запроса.
------------	----------------------------

```
LengthQueryFunction(LengthQueryFunction source)
```

Инициализирует новый экземпляр `LengthQueryFunction`, являющийся клоном переданной функции.

#### Параметры

source	Функция <code>LengthQueryFunction</code> , клон которой создается.
--------	--

## Свойства

```
Expression QueryColumnExpression
```

Выражение аргумента функции.

## Методы

```
override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)
```

Формирует текст запроса с использованием заданных экземпляров `StringBuilder` и построителя запросов `DBEngine`.

#### Параметры

sb	Экземпляр <code>StringBuilder</code> , с помощью которого формируется текст запроса.
dbEngine	Экземпляр построителя запроса к базе данных.

```
override void AddUsingParameters(QueryParameterCollection resultParameters)
```

Добавляет в аргументы функции переданную коллекцию параметров.

#### Параметры

resultParameters	Коллекция параметров запроса, которые добавляются в аргументы функции.
------------------	--

```
override object Clone()
```

Создает клон текущего экземпляра `LengthQueryFunction`.

## Класс SubstringQueryFunction C#

Пространство имен `Terrasoft.Core.DB` .

Класс получает часть строки.

**На заметку.** Полный перечень методов и свойств класса `SubstringQueryFunction` , его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

## Конструкторы

```
SubstringQueryFunction(QueryColumnExpression expression, int start, int length)
SubstringQueryFunction(IQueryColumnExpressionConvertible expression, int start, int length)
```

Инициализирует новый экземпляр `SubstringQueryFunction` для заданного выражения колонки, начальной позиции и длины подстроки.

### Параметры

expression	Выражение колонки запроса.
start	Начальная позиция подстроки.
length	Длина подстроки.

```
SubstringQueryFunction(SubstringQueryFunction source)
```

Инициализирует новый экземпляр `SubstringQueryFunction` , являющийся клоном переданной функции.

### Параметры

source	Функция <code>SubstringQueryFunction</code> , клон которой создается.
--------	---

## Свойства

```
Expression QueryColumnExpression
```

Выражение аргумента функции.

```
StartExpression QueryColumnExpression
```

Начальная позиция подстроки.

```
LengthExpression QueryColumnExpression
```

Длина подстроки.

## Методы

```
override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)
```

Формирует текст запроса с использованием заданных экземпляров `StringBuilder` и строителя запросов `DBEngine`.

### Параметры

sb	Экземпляр <code>StringBuilder</code> , с помощью которого формируется текст запроса.
dbEngine	Экземпляр строителя запроса к базе данных.

```
override void AddUsingParameters(QueryParameterCollection resultParameters)
```

Добавляет в аргументы функции переданную коллекцию параметров.

### Параметры

resultParameters	Коллекция параметров запроса, которые добавляются в аргументы функции.
------------------	--

```
override object Clone()
```

Создает клон текущего экземпляра `SubstringQueryFunction`.

## Класс ConcatQueryFunction C#

Пространство имен `Terrasoft.Core.DB`.

Класс формирует строку, которая является результатом объединения строковых значений аргументов функции.

**На заметку.** Полный перечень методов и свойств класса `ConcatQueryFunction`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

## Конструкторы

```
ConcatQueryFunction(QueryColumnExpressionCollection expressions)
```

Инициализирует новый экземпляр `ConcatQueryFunction` для переданной коллекции выражений.

#### Параметры

<code>expressions</code>	Коллекция выражений колонок запроса.
--------------------------	--------------------------------------

```
ConcatQueryFunction(ConcatQueryFunction source)
```

Инициализирует новый экземпляр `ConcatQueryFunction`, являющийся клоном переданной функции.

#### Параметры

<code>source</code>	Функция <code>ConcatQueryFunction</code> , клон которой создается.
---------------------	--

## Свойства

```
Expressions QueryColumnExpressionCollection
```

Коллекция выражений аргументов функции.

## Методы

```
override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)
```

Формирует текст запроса с использованием заданных экземпляров `StringBuilder` и построителя запросов `DBEngine`.

#### Параметры

<code>sb</code>	Экземпляр <code>StringBuilder</code> , с помощью которого формируется текст запроса.
<code>dbEngine</code>	Экземпляр построителя запроса к базе данных.

```
override void AddUsingParameters(QueryParameterCollection resultParameters)
```

Добавляет в аргументы функции переданную коллекцию параметров.

#### Параметры

<code>resultParameters</code>	Коллекция параметров запроса, которые добавляются в аргументы функции.
-------------------------------	--

```
override object Clone()
```

Создает клон текущего экземпляра `ConcatQueryFunction`.

## Класс WindowQueryFunction C#

Пространство имен `Terrasoft.Core.DB`.

Класс реализует функцию SQL окна.

**На заметку.** Полный перечень методов и свойств класса `WindowQueryFunction`, его родительских классов, а также реализуемых им интерфейсов можно найти в [Библиотеке .NET классов](#).

## Конструкторы

```
WindowQueryFunction(QueryFunction innerFunction)
```

Реализует функцию SQL окна.

### Параметры

<code>innerFunction</code>	Вложенная функция.
----------------------------	--------------------

```
WindowQueryFunction(QueryFunction innerFunction, QueryColumnExpression partitionByExpression = r
```

Реализует функцию SQL окна.

### Параметры

<code>innerFunction</code>	Вложенная функция.
<code>partitionByExpression</code>	Выражение для разделения запроса.
<code>orderByExpression</code>	Выражение для сортировки запроса.

```
WindowQueryFunction(WindowQueryFunction source) : this( source.InnerFunction, source.PartitionBy
```

Инициализирует новый экземпляр `WindowQueryFunction`, являющийся клоном переданной функции.

### Параметры

<code>source</code>	Функция <code>WindowQueryFunction</code> , клон которой создается.
---------------------	--

## Свойства

`InnerFunction QueryFunction`

Функция для применения.

`PartitionByExpression QueryColumnExpression`

Разделение по пунктам.

`OrderByExpression QueryColumnExpression`

Сортировать по пункту.

## Методы

`override void BuildSqlText(StringBuilder sb, DBEngine dbEngine)`

Формирует текст запроса с использованием заданных экземпляров `StringBuilder` и построителя запросов `DBEngine`.

### Параметры

<code>sb</code>	Экземпляр <code>StringBuilder</code> , с помощью которого формируется текст запроса.
<code>dbEngine</code>	Экземпляр построителя запроса к базе данных.

`override void AddUsingParameters(QueryParameterCollection resultParameters)`

Добавляет в аргументы функции переданную коллекцию параметров.

### Параметры

<code>resultParameters</code>	Коллекция параметров запроса, которые добавляются в аргументы функции.
-------------------------------	--

`override object Clone()`

Создает клон текущего экземпляра `WindowQueryFunction`.