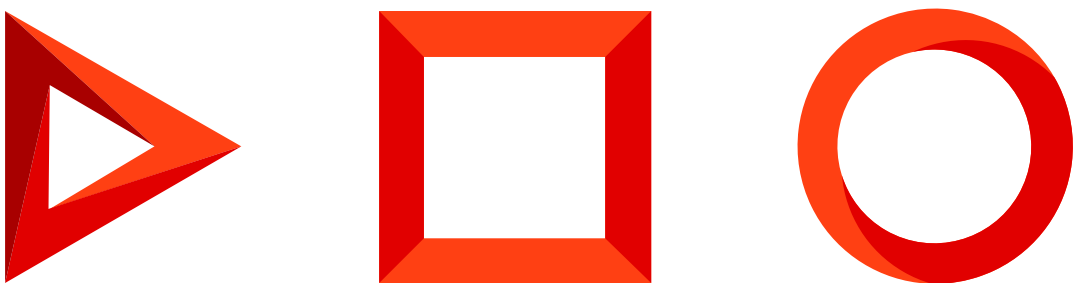
Creatio Academy

Операции с данными (back-end)

Доступ к данным через ORM

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

Доступ к данным через ORM

Сложный

Доступ к базе данных предоставляет группа классов серверного ядра приложения, перечисленные ниже. С их помощью можно выполнять весь набор CRUD-операций, учитывать права доступа текущего пользователя, помещать полученные данные в хранилища кэша.

Получение данных их базы

Select

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.

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`.

EntitySchemaQuery

Класс `Terrasoft.Core.Entities.EntitySchemaQuery` предназначен для построения запросов выборки записей из таблиц базы данных с учетом прав доступа текущего пользователя. 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.

Класс `EntitySchemaQuery` реализует механизм работы с хранилищем (кэшем Creatio либо произвольным хранилищем, определенным пользователем). При выполнении запроса `EntitySchemaQuery` данные, полученные из базы данных на сервере, помещаются в кэш. В качестве кэша запроса может выступать произвольное хранилище, которое реализует интерфейс `Terrasoft.Core.Store.ICacheStore`. По умолчанию в качестве кэша запроса `EntitySchemaQuery` выступает кэш Creatio уровня сессии (данные доступны только в сессии текущего пользователя) с локальным хранением данных.

Для запросов `EntitySchemaQuery` можно определить дополнительные настройки, которые задают параметры для постраничного вывода результатов выполнения запроса, а также параметры построения иерархического запроса. Для этого предназначен класс `Terrasoft.Core.Entities.EntitySchemaQueryOptions.`

Результатом выполнения запроса `EntitySchemaQuery` является экземпляр `Terrasoft.Nui.ServiceModel.DataContract.EntityCollection` (коллекция экземпляров класса `Terrasoft.Core.Entities.Entity`). Каждый экземпляр `Entity` в коллекции представляет собой строку набора данных, возвращаемого запросом.

Особенности EntitySchemaQuery

1) Поддержка прав доступа

Запрос на выборку данных `EntitySchemaQuery` строится таким образом, чтобы учитывать права текущего пользователя. В результирующий набор попадут только те данные, к которым текущий пользователь имеет доступ согласно его правам. Дополнительно для `EntitySchemaQuery` можно регулировать условия наложения прав на связанные таблицы, присутствующие в запросе (которые присоединяются к запросу предложением JOIN). Эти условия определяются значением свойства `JoinRightState` экземпляра `EntitySchemaQuery`.

2) Механизм кеширования

В `EntitySchemaQuery` реализован механизм работы с хранилищем (кэшем Creatio либо произвольным хранилищем, определенным пользователем). При выполнении запроса `EntitySchemaQuery` данные, полученные из базы данных на сервере, помещаются в кэш. В качестве кэша запроса может выступать произвольное хранилище, которое реализует интерфейс `ICacheStore`. По умолчанию в качестве кэша запроса `EntitySchemaQuery` выступает кэш Creatio уровня сессии (данные доступны только в сессии текущего пользователя) с локальным хранением данных. Кэш запроса определяется свойством `Cache` экземпляра `EntitySchemaQuery.` С помощью свойства `CacheItemName` задается ключ доступа к кэшу запроса (пример 4).

3) Дополнительные настройки запроса

Для запросов `EntitySchemaQuery` можно определить дополнительные настройки, которые задают параметры для постраничного вывода результатов выполнения запроса, а также параметры построения иерархического запроса. Для этого предназначен класс `EntitySchemaQueryOptions`.

Свойства класса `EntitySchemaQueryOptions`:

- `HierarchicalColumnName` — имя колонки, использующейся для построения иерархического запроса;
- `HierarchicalColumnValue` — начальное значение иерархической колонки, от которого будет строиться иерархия;
- `HierarchicalMaxDepth` — максимальный уровень вложенности иерархического запроса;
- `PageableConditionValues` — значения условий постраничного вывода;
- `PageableDirection` — направление постраничного вывода;
- `PageableRowCount` — количество записей страницы результирующего набора данных, возвращаемого запросом.

Один и тот же экземпляр `EntitySchemaQueryOptions` можно использовать для получения результатов выполнения различных запросов, передавая его в качестве параметра методу `GetEntityCollection()` соответствующего запроса (Примеры EntitySchemaQuery).

Добавление данных

Insert

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.

InsertSelect

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`.

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.

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

Изменение данных

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`.

Удаление данных

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`.

Работа с сущностью базы данных

Класс `Terrasoft.Core.Entities.Entity` предназначен для доступа к объекту, который представляет собой запись в таблице базы данных. Он также может использоваться для добавления, изменения и удаления определенных записей.

Корневая схема

Корневая схема — это схема (таблица в базе данных), относительно которой выполняется построение путей ко всем колонкам в запросе, в том числе к колонкам присоединяемых таблиц.

При построении путей к колонкам применяется принцип связей через справочные поля. Имя произвольной колонки, добавляемой в запрос, можно построить в виде цепочки взаимосвязанных звеньев, каждое из которых представляет собой "контекст" конкретной схемы, которая связывается с

предыдущей по внешнему ключу.

В общем случае формат построения имени произвольной колонки из схемы N можно представить в следующем виде:

[*Контекст схемы 1].[...*].[*Контекст схемы N*].[*Имя_колонки*]

Многопоточность при работе с базой данных

Использование нескольких потоков при работе с базой данных через `UserConnection` может привести к проблемам синхронизации старта и коммита транзакций.

Важно. Проблема возникает при работе с базой данных, даже если `DBExecutor` не используется напрямую, а используется, например, через `EntitySchemaQuery`.

Важно. Поскольку для работы с базой данных используются неуправляемые (`unmanaged`) ресурсы, то создание экземпляра `DBExecutor` необходимо оборачивать в оператор `using`. Или же явно вызывать метод `Dispose()` для освобождения ресурсов. Подробнее об использовании оператора `using` можно можно узнать из документации.

Пример неправильного использования

Ниже приведен фрагмент исходного кода, в котором `DBExecutor` используется неправильно. Нельзя выполнять вызов методов экземпляра `DBExecutor` в параллельных потоках.

```
// 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 продолжается дальше.
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,
// т.к. экземпляр DBExecutor уже используется в параллельном потоке.
using (DBExecutor dbExecutor = UserConnection.EnsureDBConnection()) {
    using (IDataReader dataReader = select.ExecuteReader(dbExecutor)) {
        while (dataReader.Read()) {
            //...
        }
    }
}
```

Пример правильного использования

Ниже приведен фрагмент исходного кода, в котором `DBExecutor` используется корректно. Вызов методов экземпляра `DBExecutor` производится последовательно, в одном потоке.

```
// 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()) {
            //...
        }
    }
}
```

Работа с PostgreSQL

Общие рекомендации:

1. Не рекомендуется использовать для создания триггеров, представлений, функций команду

`CREATE OR REPLACE`. Вместо этого используйте конструкцию `DROP … IF EXISTS` (при необходимости можно использовать команду `CASCADE`), а затем `CREATE OR REPLACE`.

2. Вместо схемы `"dbo"` используйте `"public".`
3. Учитывайте регистрозависимость системных имен, используйте кавычки ("") для имен таблиц, колонок и т.д..
4. Вместо типа `BIT` в `MS SQL` используйте в `Postgres` тип `BOOL`. Для проверки значения поля типа `BOOL` необязательно использовать конструкцию `WHERE "boolColumn" = true`, достаточно `WHERE "boolColumn"` или `WHERE NOT "boolColumn".`
5. В `Postgres` можно использовать сокращенный вид явного преобразования `::TEXT.`
6. В `Postgres` при сравнении строк учитывается регистр. Для выполнения регистронезависимого сравнения можно использовать ключевое слово `iLIKE`. Однако учитывайте, что сравнение при этом значительно более медленное, чем при использовании комбинации `UPPER+LIKE`. Кроме того у комбинации `UPPER+LIKE` менее строгие правила применимости индексов, чем у `iLIKE`.
7. Если нет какого-либо неявного приведения типов, то его можно создать с помощью команды `CREATE CAST.` Подробнее об этом читайте в документации PostgreSQL.
8. В `Postgres` нет встроенной функции `NESTLEVEL` в рекурсивных процедурах. Для хранения текущего уровня рекурсии следует создавать специальный параметр процедуры.
9. Вместо типа `SYSNAME` используйте тип `NAME`.
10. Вместо пустых `INSTEAD`-триггеров создавайте правила, например:

```
CREATE RULE RU_VwAdministrativeObjects AS
ON UPDATE TO "VwAdministrativeObjects"
DO INSTEAD NOTHING;
```

11. При выполнении команды `UPDATE` в `Postgres` не работает неявное преобразование типа `INT` в тип `BOOL`, даже при наличии соответствующего оператора `CAST`. Следует явно привести `INT` -значение к типу `BOOL`.
12. Способы форматирования строковых литералов подробно описаны на сайте документации PostgreSQL:
    - quote_ident;
    - quote_literal;
    - format.
13. Вместо `@@ROWCOUNT` используйте следующую конструкцию:

```
DECLARE rowsCount BIGINT = 0;
GET DIAGNOSTICS rowsCount = row_count;
```

14. Вместо конструкции в MS SQL

```
(CASE WHEN EXISTS (
  SELECT 1
  FROM [SysSSPEntitySchemaAccessList]
  WHERE [SysSSPEntitySchemaAccessList].[EntitySchemaUId] = [BaseSchemas].[UId]
  )
THEN 1 ELSE 0 END) AS [IsInSSPEntitySchemaAccessList]
```

в PostgreSql следует использовать конструкцию

```
EXISTS (
  SELECT 1
  FROM "SysSSPEntitySchemaAccessList"
  WHERE "EntitySchemaUId" = BaseSchema."UId"
 ) "IsInSSPEntitySchemaAccessList"
```

Полученное в результате запроса поле будет иметь тип `BOOL`.

Соответствие типов данных

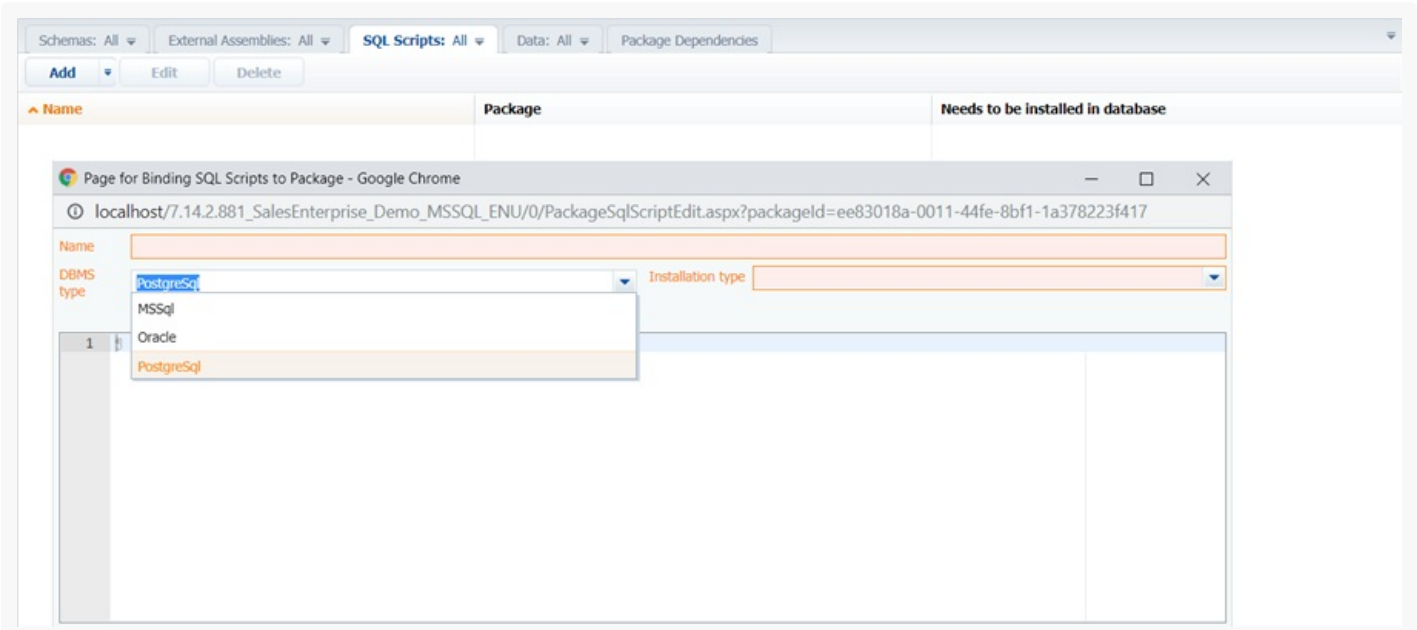
Таблица соответствия типов данных Creatio, MS SQL и PostgreSQL

Тип данных в дизайнере объекта Creatio	**Тип данных в MS SQL**	**Тип данных в PostgreSQL**
BLOB	VARBINARY	BYTEA
Boolean	BIT	BOOLEAN
Color	NVARCHAR	CHARACTER VARYING
CRC	NVARCHAR	CHARACTER VARYING
Currency	DECIMAL	NUMERIC
Date	DATE	DATE
Date/Time	DATETIME2	TIMESTAMP WITHOUT TIME ZONE
Decimal (0.00000001)	DECIMAL	NUMERIC
Decimal (0.0001)	DECIMAL	NUMERIC
Decimal (0.001)	DECIMAL	NUMERIC
Decimal (0.01)	DECIMAL	NUMERIC

Decimal (0.01)	DECIMAL	NUMERIC
Decimal (0.1)	DECIMAL	NUMERIC
Encrypted string	NVARCHAR	CHARACTER VARYING
File	VARBINARY	BYTEA
Image	VARBINARY	BYTEA
Image Link	UNIQUEIDENTIFIER	UUID
Integer	INTEGER	INTEGER
Lookup	UNIQUEIDENTIFIER	UUID
Text (250 characters)	NVARCHAR(250)	CHARACTER VARYING
Text (50 characters)	NVARCHAR(50)	CHARACTER VARYING
Text (500 characters)	NVARCHAR(500)	CHARACTER VARYING
Time	TIME	TIME WITHOUT TIME ZONE
Unique identifier	UNIQUEIDENTIFIER	UUID
Unlimited length text	NVARCHAR(MAX)	TEXT

Привязка SQL-сценария к пакету

Если в пакете привязаны SQL-сценарии, например, для MS SQL, то для работы с Postgres создайте скрипт, который выполняет те же функции, но использует синтаксис PostgreSQL. Для этого на вкладке [*SQL-сценарии*] добавьте скрипт с типом СУБД PostgreSql.



Управлять сущностями базы данных

Сложный

На заметку. Примеры 1-5, приведенные в этой статье, реализованы в веб-сервисе. Пакет с реализацией веб-сервиса прикреплен в блоке "Ресурсы".

Пример 1

Пример. Получить значение колонки схемы [*City*] с именем `Name`.

Метод `GetEntityColumnData`

```
public string GetEntityColumnData()
{
    var result = "";
    /* Создание запроса к схеме City, добавление в запрос колонки Name. */
    var esqResult = new EntitySchemaQuery(UserConnection.EntitySchemaManager, "City");
    var colName = esqResult.AddColumn("Name");
    /* Выполнение запроса к базе данных и получение объекта с заданным идентификатором. UId объе
    var entity = esqResult.GetEntity(UserConnection, new Guid("100B6B13-E8BB-DF11-B00F-001D60E93
    /* Получение значения колонки объекта. */
    result += entity.GetColumnValue(colName.Name).ToString();
    return result;
}
```

Пример 2

Пример. Получить коллекцию имен колонок схемы [*City*].

Метод `GetEntityColumns`

```
public IEnumerable<string> GetEntityColumns()
{
    /* Создание объекта строки данных схемы City (по идентификатору схемы, полученному из базы д
    var entity = new Entity(UserConnection, new Guid("5CA90B6A-93E7-4448-BEFE-AB5166EC2CFE"));
    /* Получение из базы данных объекта с заданным идентификатором. UId объекта можно получить и
    entity.FetchFromDB(new Guid("100B6B13-E8BB-DF11-B00F-001D60E938C6"),true);
    /* Получение коллекции имен колонок объекта. */
    var result = entity.GetColumnValueNames();
    return result;
}
```

Пример 3

Пример. Удалить из базы данных записи схемы [*Order*].

Метод `DeleteEntity`

```
public bool DeleteEntity()
{
    /* Создание запроса к схеме Order, добавление в запрос всех колонок схемы. */
    var esqResult = new EntitySchemaQuery(UserConnection.EntitySchemaManager, "Order");
    esqResult.AddAllSchemaColumns();
    /* Выполнение запроса к базе данных и получение объекта с заданным идентификатором. UId объе
    var entity = esqResult.GetEntity(UserConnection, new Guid("e3bfa32f-3fe9-4bae-9332-16c162c51
    /* Удаление объекта из базы данных. */
    entity.Delete();
    /* Проверка, существует ли в базе данных объект с заданным идентификатором. */
    var result = entity.ExistInDB(new Guid("e3bfa32f-3fe9-4bae-9332-16c162c51e0d"));
    return result;
}
```

Пример 4

Пример. Изменить статус заказа.

Метод `UpdateEntity`

```
public bool UpdateEntity()
{
    /* Создание запроса к схеме Order, добавление в запрос всех колонок схемы. */
    var esqResult = new EntitySchemaQuery(UserConnection.EntitySchemaManager, "Order");
    esqResult.AddAllSchemaColumns();
    /* Выполнение запроса к базе данных и получение объекта с заданным идентификатором. UId объе
    var entity = esqResult.GetEntity(UserConnection, new Guid("58be5223-715d-4b16-a5c4-e3d4ec041
    /* Создание объекта строки данных схемы OrderStatus. */
    var statusSchema = UserConnection.EntitySchemaManager.GetInstanceByName("OrderStatus");
    var newStatus = statusSchema.CreateEntity(UserConnection);
    /* Получение из базы данных объекта с заданным названием. */
    newStatus.FetchFromDB("Name", "4. Completed");
    /* Присваивает колонке StatusId новое значение. */
    entity.SetColumnValue("StatusId", newStatus.GetTypedColumnValue<Guid>("Id"));
    /* Сохранение измененного объекта в базе данных. */
    var result = entity.Save();
    return result;
}
```

Пример 5

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

Метод `UpdateEntity`

```
public bool InsertEntity(string city, string country)
{
    city = city ?? "unknown city";
    country = country ?? "unknown country";
    var citySchema = UserConnection.EntitySchemaManager.GetInstanceByName("City");
    var entity = citySchema.CreateEntity(UserConnection);
    entity.FetchFromDB("Name", city);
    /* Устанавливает для колонок объекта значения по умолчанию. */
    entity.SetDefColumnValues();
    var contryEntity = new Entity(UserConnection, new Guid("09FCE1F8-515C-4296-95CD-8CD93F79A6CF
```

```
    contryEntity.FetchFromDB("Name", country);
    /* Присваивает колонке Name переданное название города. */
    entity.SetColumnValue("Name", city);
    /* Присваивает колонке CountryId UId переданной страны. */
    entity.SetColumnValue("CountryId", contryEntity.GetTypedColumnValue<Guid>("Id"));
    var result = entity.Save();
    return result;
}
```

Пример 6

Пример. Создать контакт с именем "User01".

```
EntitySchema contactSchema = UserConnection.EntitySchemaManager.GetInstanceByName("Contact");
Entity contactEntity = contactSchema.CreateEntity(UserConnection);
contactEntity.SetDefColumnValues();
contactEntity.SetColumnValue("Name", "User01");
contactEntity.Save();
```

Пример 7

Пример. Изменить имя контакта на "User02".

```
EntitySchema entitySchema = UserConnection.EntitySchemaManager.GetInstanceByName("Contact");
Entity entity = entitySchema.CreateEntity(UserConnection);
if (!entity.FetchFromDB(some_id) {
    return false;
}
entity.SetColumnValue("Name", "User02");
return entity.Save();
```

Пример 8

Пример. Удалить контакт с именем "User02".

```
EntitySchema entitySchema = UserConnection.EntitySchemaManager.GetInstanceByName("Contact");
Entity entity = entitySchema.CreateEntity(UserConnection);
var fetchConditions = new Dictionary<string, object> {
    {"Name", "User02"}
};
if (entity.FetchFromDB(fetchConditions)) {
    entity.Delete();
}
```

Получить данные из базы данных

 Сложный

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;
}
```

Получить данные из базы данных с учетом прав пользователя

Сложный

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

Пример 1

Пример. Создать экземпляр `EntitySchemaQuery`.

Метод `CreateESQ`

```
public string CreateESQ()
{
    var result = "";
    /* Получение экземпляра менеджера схем объектов. */
    EntitySchemaManager esqManager = SystemUserConnection.EntitySchemaManager;
    /* Получение экземпляра схемы, которая будет установлена в качестве корневой для создаваемог
    var rootEntitySchema = esqManager.GetInstanceByName("City") as EntitySchema;
    /* Создание экземпляра EntitySchemaQuery, у которого в качестве корневой схемы установлена с
    var esqResult = new EntitySchemaQuery(rootEntitySchema);
    /* Добавление колонок, которые будут выбираться в результирующем запросе. */
    esqResult.AddColumn("Id");
    esqResult.AddColumn("Name");
    /* Получение экземпляра Select, ассоциированного с созданным запросом EntitySchemaQuery. */
    Select selectEsq = esqResult.GetSelectQuery(SystemUserConnection);
    /* Получение текста результирующего запроса созданного экземпляра EntitySchemaQuery. */
    result = selectEsq.GetSqlText();
    return result;
}
```

Пример 2

Пример. Создать клон экземпляра `EntitySchemaQuery`.

Метод `CreateESQClone`

```
public string CreateESQClone()
{
    var result = "";
    EntitySchemaManager esqManager = SystemUserConnection.EntitySchemaManager;
    var esqSource = new EntitySchemaQuery(esqManager, "Contact");
    esqSource.AddColumn("Id");
    esqSource.AddColumn("Name");
    /* Создание экземпляра EntitySchemaQuery, являющегося клоном экземпляра esqSource. */
    var esqClone = new EntitySchemaQuery(esqSource);
    result = esqClone.GetSelectQuery(SystemUserConnection).GetSqlText();
    return result;
}
```

Пример 3

Пример. Получить результат выполнения запроса.

Метод `GetEntitiesExample`

```
public string GetEntitiesExample()
{
    var result = "";
    /* Создание запроса к схеме City, добавление в запрос колонки Name. */
    var esqResult = new EntitySchemaQuery(UserConnection.EntitySchemaManager, "City");
    var colName = esqResult.AddColumn("Name");

    /* Выполнение запроса к базе данных и получение всей результирующей коллекции объектов. */
    var entities = esqResult.GetEntityCollection(UserConnection);
    for (int i=0; i < entities.Count; i++) {
        result += entities[i].GetColumnValue(colName.Name).ToString();
        result += "\n";
    }

    /* Выполнение запроса к базе данных и получение объекта с заданным идентификатором. */
    var entity = esqResult.GetEntity(UserConnection, new Guid("100B6B13-E8BB-DF11-B00F-001D60E93
    result += "\n";
    result += entity.GetColumnValue(colName.Name).ToString();
    return result;
}
```

Пример 4

Пример. Использовать кэш запроса `EntitySchemaQuery`.

Метод `UsingCacheExample`

```
public Collection<string> UsingCacheExample()
{
    /* Создание запроса к схеме City, добавление в запрос колонки Name. */
    var esqResult = new EntitySchemaQuery(UserConnection.EntitySchemaManager, "City");
    esqResult.AddColumn("Name");

    /* Определение ключа, под которым в кэше будут храниться результаты выполнения запроса. В ка
    esqResult.CacheItemName = "EsqResultItem";

    /* Коллекция, в которую будут помещены результаты выполнения запроса. */
    var esqCityNames = new Collection<string>();

    /* Коллекция, в которую будут помещаться закэшированные результаты выполнения запроса. */
    var cachedEsqCityNames = new Collection<string>();
```

```
    /* Выполнение запроса к базе данных и получение результирующей коллекции объектов.
    После выполнения этой операции результаты запроса будут помещены в кэш. */
    var entities = esqResult.GetEntityCollection(UserConnection);

    /* Обработка результатов выполнения запроса и заполнение коллекции esqCityNames. */
    foreach (var entity in entities)
    {
        esqCityNames.Add(entity.GetTypedColumnValue<string>("Name"));
    }

    /* Получение ссылки на кэш запроса esqResult по ключу CacheItemName в виде таблицы данных в
    var esqCacheStore = esqResult.Cache[esqResult.CacheItemName] as DataTable;

    /* Заполнение коллекции cachedEsqCityNames значениями из кэша запроса. */
    if (esqCacheStore != null)
    {
        foreach (DataRow row in esqCacheStore.Rows)
        {
            cachedEsqCityNames.Add(row[0].ToString());
        }
    }
    return cachedEsqCityNames;
}
```

Пример 5

Пример. Использовать дополнительные настройки запроса `EntitySchemaQuery`.

Метод `UsingCacheExample`

```
public Collection<string> ESQOptionsExample()
{
    /* Создание экземпляра запроса с корневой схемой City. */
    var esqCities = new EntitySchemaQuery(UserConnection.EntitySchemaManager, "City");
    esqCities.AddColumn("Name");

    /* Создание запроса с корневой схемой Country. */
    var esqCountries = new EntitySchemaQuery(UserConnection.EntitySchemaManager, "Country");
    esqCountries.AddColumn("Name");

    /* Создание экземпляра настроек для возврата запросом первых 5 строк. */
    var esqOptions = new EntitySchemaQueryOptions()
    {
```

```
        PageableDirection = PageableSelectDirection.First,
        PageableRowCount = 5,
        PageableConditionValues = new Dictionary<string, object>()
    };

    /* Получение коллекции городов, которая будет содержать первые 5 городов результирующего наб
    var cities = esqCities.GetEntityCollection(UserConnection, esqOptions);

    /* Получение коллекции стран, которая будет содержать первые 5 стран результирующего набора
    var countries = esqCountries.GetEntityCollection(UserConnection, esqOptions);
        var esqStringCollection = new Collection<string>();
    foreach (var entity in cities)
    {
        esqStringCollection.Add(entity.GetTypedColumnValue<string>("Name"));
    }
    foreach (var entity in countries)
    {
        esqStringCollection.Add(entity.GetTypedColumnValue<string>("Name"));
    }
    return esqStringCollection;
}
```

Добавить данные в базу данных

Сложный

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;
}
```

Функциональность многострочного добавления данных

Сложный

Функциональность многострочной вставки доступна на уровне класса `Insert` и ее работа определяется методом `Values()`.

При вызове метода `Values()` все последующие вызовы `Set()` попадают в новый экземпляр `ColumnsValues`. При построении запроса, если в коллекции `ColumnsValuesCollection` встречается более одного набора данных, то будет построен запрос с несколькими блоками `Values()`.

Пример

```
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 следующий 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

1. При использовании `Column.Parameter` в выражении `Set()` необходимо помнить про ограничение 2100 параметров в MS SQL.
2. Класс `Insert` не может самостоятельно разбивать запрос на несколько, если в нем находится больше параметров, чем нужно. Разбиение на несколько запросов должно быть реализовано разработчиком.

   Пример

```
IEnumerable<IEnumerable<ImportEntity>> GetImportEntitiesChunks(IEnumerable<ImportEntity> enti
                IEnumerable<ImportColumn> keyColumns) {
    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) {
            //...
        }
});
```

3. Класс `Insert()` не проводит валидацию на соответствие количества колонок и количества условий `Set()`. Например, есть результирующий SQL-запрос:

SQL-запрос

```
INSERT INTO [dbo].[Table] (Column1, Column2, Column3)
Values (1, 2), (1, 2, 3)
```

В таком случае возникнет исключение на уровне работы СУБД. Подобная валидация не лежит в зоне ответственности класса `Insert` и зависит только от разработчика.

Добавить данные в базу данных с помощью подзапросов

 Сложный

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

 Сложный

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

Сложный

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";
}
```

Примеры скриптов для MS SQL и PostgreSQL

Средний

Пример 1 (представления)

Пример. Пример SQL-скрипта, который создает представление и триггеры для добавления, изменения и удаления записей из целевой таблицы.

MS SQL

```
-- Представление и триггеры, которые позволяют редактировать целевую таблицу
-- MSSQL
IF EXISTS (SELECT * FROM sys.views WHERE object_id = OBJECT_ID(N'[dbo].[VwSysAdminUnit]'))
DROP VIEW [dbo].[VwSysAdminUnit]
GO
CREATE VIEW [dbo].[VwSysAdminUnit]
AS
SELECT [SysAdminUnit].[Id]
    ,[SysAdminUnit].[CreatedOn]
    ,[SysAdminUnit].[CreatedById]
    ,[SysAdminUnit].[ModifiedOn]
    ,[SysAdminUnit].[ModifiedById]
    ,[SysAdminUnit].[Name]
    ,[SysAdminUnit].[Description]
    ,[SysAdminUnit].[ParentRoleId]
    ,[SysAdminUnit].[ContactId]
    ,[SysAdminUnit].[IsDirectoryEntry]
    ,[TimeZone].[Id] AS [TimeZoneId]
    ,[SysAdminUnit].[UserPassword]
    ,[SysAdminUnitType].[Id] AS [SysAdminUnitTypeId]
    ,[SysAdminUnit].[AccountId]
    ,[SysAdminUnit].[Active]
    ,[SysAdminUnit].[LoggedIn]
```

```
    ,[SysAdminUnit].[SynchronizeWithLDAP]
    ,[SysAdminUnit].[LDAPEntry]
    ,[SysAdminUnit].[LDAPEntryId]
    ,[SysAdminUnit].[LDAPEntryDN]
    ,[SysAdminUnit].[SysCultureId]
    ,[SysAdminUnit].[ProcessListeners]
    ,[SysAdminUnit].[PasswordExpireDate]
    ,[SysAdminUnit].[HomePageId]
    ,[SysAdminUnit].[ConnectionType]
    ,[ConnectionType].[Id] AS [UserConnectionTypeId]
    ,[SysAdminUnit].[ForceChangePassword]
    ,[SysAdminUnit].[DateTimeFormatId]
    ,[SysAdminUnit].[Id] as [SysAdminUnitId]
    ,[SysAdminUnit].[SessionTimeout] as [SessionTimeout]
FROM [SysAdminUnit]
INNER JOIN [SysAdminUnitType] ON [SysAdminUnitType].[Value] = [SysAdminUnit].[SysAdminUnitTypeVa
LEFT JOIN [ConnectionType] AS [ConnectionType] ON [ConnectionType].[Value] = [SysAdminUnit].[Cor
LEFT JOIN [TimeZone] AS [TimeZone] ON [TimeZone].[Code] = [SysAdminUnit].[TimeZoneId]
GO
CREATE TRIGGER [dbo].[ITR_VwSysAdminUnit_I]
ON [dbo].[VwSysAdminUnit]
    INSTEAD OF INSERT
AS
BEGIN
SET NOCOUNT ON;
INSERT INTO [SysAdminUnit](
    [Id]
    ,[CreatedOn]
    ,[CreatedById]
    ,[ModifiedOn]
    ,[ModifiedById]
    ,[Name]
    ,[Description]
    ,[ParentRoleId]
    ,[ContactId]
    ,[IsDirectoryEntry]
    ,[TimeZoneId]
    ,[UserPassword]
    ,[SysAdminUnitTypeValue]
    ,[AccountId]
    ,[Active]
    ,[LoggedIn]
    ,[SynchronizeWithLDAP]
    ,[LDAPEntry]
    ,[LDAPEntryId]
    ,[LDAPEntryDN]
    ,[SysCultureId]
    ,[ProcessListeners]
    ,[PasswordExpireDate]
```

```
    ,[HomePageId]
    ,[ConnectionType]
    ,[ForceChangePassword]
    ,[DateTimeFormatId]
    ,[SessionTimeout])
SELECT [Id]
    ,[CreatedOn]
    ,[CreatedById]
    ,[ModifiedOn]
    ,[ModifiedById]
    ,[Name]
    ,[Description]
    ,[ParentRoleId]
    ,[ContactId]
    ,[IsDirectoryEntry]
    ,(SELECT COALESCE(
        (SELECT [TimeZone].[Code] FROM [TimeZone]
            WHERE [TimeZone].[Id] = [INSERTED].[TimeZoneId]), ''))
    ,[UserPassword]
    ,ISNULL((SELECT [SysAdminUnitType].[Value] FROM [SysAdminUnitType]
        WHERE [SysAdminUnitType].[Id] = [INSERTED].[SysAdminUnitTypeId]), 4)
    ,[AccountId]
    ,[Active]
    ,ISNULL([LoggedIn], 0)
    ,[SynchronizeWithLDAP]
    ,[LDAPEntry]
    ,[LDAPEntryId]
    ,[LDAPEntryDN]
    ,[SysCultureId]
    ,[ProcessListeners]
    ,[PasswordExpireDate]
    ,[HomePageId]
    ,COALESCE([INSERTED].[ConnectionType],
        (SELECT [ConnectionType].[Value] FROM [ConnectionType]
        WHERE [ConnectionType].[Id] = [INSERTED].[UserConnectionTypeId]), 0)
    ,ISNULL([ForceChangePassword], 0)
    ,[DateTimeFormatId]
    ,[SessionTimeout]
FROM [INSERTED]
END
GO
CREATE TRIGGER [dbo].[ITR_VwSysAdminUnit_U]
ON [dbo].[VwSysAdminUnit]
    INSTEAD OF UPDATE
AS
BEGIN
SET NOCOUNT ON;
UPDATE [SysAdminUnit]
```

```
SET [SysAdminUnit].[CreatedOn] = [INSERTED].[CreatedOn]
    ,[SysAdminUnit].[CreatedById] = [INSERTED].[CreatedById]
    ,[SysAdminUnit].[ModifiedOn] =[INSERTED].[ModifiedOn]
    ,[SysAdminUnit].[ModifiedById] = [INSERTED].[ModifiedById]
    ,[SysAdminUnit].[Name] = [INSERTED].[Name]
    ,[SysAdminUnit].[Description] = [INSERTED].[Description]
    ,[SysAdminUnit].[ParentRoleId] = [INSERTED].[ParentRoleId]
    ,[SysAdminUnit].[ContactId] = [INSERTED].[ContactId]
    ,[SysAdminUnit].[IsDirectoryEntry] = [INSERTED].[IsDirectoryEntry]
    ,[SysAdminUnit].[TimeZoneId] =
        (SELECT COALESCE(
            (SELECT [TimeZone].[Code] FROM [TimeZone]
                WHERE [TimeZone].[Id] = [INSERTED].[TimeZoneId]), ''))
    ,[SysAdminUnit].[UserPassword] = [INSERTED].[UserPassword]
    ,[SysAdminUnit].[SysAdminUnitTypeValue] =
        (SELECT [SysAdminUnitType].[Value] FROM [SysAdminUnitType]
            WHERE [SysAdminUnitType].[Id] = [INSERTED].[SysAdminUnitTypeId])
    ,[SysAdminUnit].[AccountId] = [INSERTED].[AccountId]
    ,[SysAdminUnit].[Active] = [INSERTED].[Active]
    ,[SysAdminUnit].[LoggedIn] = [INSERTED].[LoggedIn]
    ,[SysAdminUnit].[SynchronizeWithLDAP] = [INSERTED].[SynchronizeWithLDAP]
    ,[SysAdminUnit].[LDAPEntry] = [INSERTED].[LDAPEntry]
    ,[SysAdminUnit].[LDAPEntryId] = [INSERTED].[LDAPEntryId]
    ,[SysAdminUnit].[LDAPEntryDN] = [INSERTED].[LDAPEntryDN]
    ,[SysAdminUnit].[SysCultureId] = [INSERTED].[SysCultureId]
    ,[SysAdminUnit].[ProcessListeners] = [INSERTED].[ProcessListeners]
    ,[SysAdminUnit].[PasswordExpireDate] = [INSERTED].[PasswordExpireDate]
    ,[SysAdminUnit].[HomePageId] = [INSERTED].[HomePageId]
    ,[SysAdminUnit].[ConnectionType] = COALESCE([INSERTED].[ConnectionType],
        (SELECT [ConnectionType].[Value] FROM [ConnectionType]
        WHERE [ConnectionType].[Id] = [INSERTED].[UserConnectionTypeId]), 0)
    ,[SysAdminUnit].[ForceChangePassword] = [INSERTED].[ForceChangePassword]
    ,[SysAdminUnit].[DateTimeFormatId] = [INSERTED].[DateTimeFormatId]
    ,[SysAdminUnit].[SessionTimeout] = [INSERTED].[SessionTimeout]
FROM [SysAdminUnit]
INNER JOIN [INSERTED] ON [SysAdminUnit].[Id] = [INSERTED].[Id]
END
GO
CREATE TRIGGER [dbo].[ITR_VwSysAdminUnit_D]
ON [dbo].[VwSysAdminUnit]
    INSTEAD OF DELETE
AS
BEGIN
SET NOCOUNT ON;
DELETE FROM [SysAdminUnit]
WHERE EXISTS(SELECT * FROM [DELETED] WHERE [SysAdminUnit].[Id] = [DELETED].[Id])
END
GO
```

Postgre SQL

```
-- Представление и триггеры, которые позволяют редактировать целевую таблицу
-- PostgreSql
DROP FUNCTION IF EXISTS "public"."ITR_VwSysLookup_IUD_Func" CASCADE;
DROP VIEW IF EXISTS "public"."VwSysLookup";

CREATE VIEW "public"."VwSysLookup" AS
SELECT "SysLookup"."Id"
    ,"SysLookup"."CreatedOn"
    ,"SysLookup"."CreatedById"
    ,"SysLookup"."ModifiedOn"
    ,"SysLookup"."ModifiedById"
    ,"SysLookup"."Name"
    ,"SysLookup"."Description"
    ,"SysLookup"."SysFolderId"
    ,"SysLookup"."SysEntitySchemaUId"
    ,"SysLookup"."SysGridPageSchemaUId"
    ,"SysLookup"."SysEditPageSchemaUId"
    ,"VwSysSchemaInfo"."SysWorkspaceId"
    ,"SysLookup"."ProcessListeners"
    ,"SysLookup"."IsSystem"
    ,"SysLookup"."IsSimple"
FROM "public"."SysLookup"
INNER JOIN "public"."VwSysSchemaInfo" ON "SysLookup"."SysEntitySchemaUId" = "VwSysSchemaInfo"."U

CREATE FUNCTION "public"."ITR_VwSysLookup_IUD_Func"() RETURNS TRIGGER AS $$
    BEGIN
        IF TG_OP = 'INSERT' THEN
            INSERT INTO "public"."SysLookup"(
                "Id"
                ,"CreatedOn"
                ,"CreatedById"
                ,"ModifiedOn"
                ,"ModifiedById"
                ,"Name"
                ,"Description"
                ,"SysFolderId"
                ,"SysEntitySchemaUId"
                ,"SysGridPageSchemaUId"
                ,"SysEditPageSchemaUId"
                ,"ProcessListeners"
                ,"IsSystem"
                ,"IsSimple")
            SELECT NEW."Id"
                ,NEW."CreatedOn"
```

```
                ,NEW."CreatedById"
                ,NEW."ModifiedOn"
                ,NEW."ModifiedById"
                ,NEW."Name"
                ,NEW."Description"
                ,NEW."SysFolderId"
                ,NEW."SysEntitySchemaUId"
                ,NEW."SysGridPageSchemaUId"
                ,NEW."SysEditPageSchemaUId"
                ,NEW."ProcessListeners"
                ,NEW."IsSystem"
                ,NEW."IsSimple";
            RETURN NEW;
        ELSIF TG_OP = 'UPDATE' THEN
            UPDATE "public"."SysLookup"
            SET "CreatedOn" = NEW."CreatedOn"
                ,"CreatedById" = NEW."CreatedById"
                ,"ModifiedOn" = NEW."ModifiedOn"
                ,"ModifiedById" = NEW."ModifiedById"
                ,"Name" = NEW."Name"
                ,"Description" = NEW."Description"
                ,"SysFolderId" = NEW."SysFolderId"
                ,"SysEntitySchemaUId" = NEW."SysEntitySchemaUId"
                ,"SysGridPageSchemaUId" = NEW."SysGridPageSchemaUId"
                ,"SysEditPageSchemaUId" = NEW."SysEditPageSchemaUId"
                ,"ProcessListeners" = NEW."ProcessListeners"
                ,"IsSystem" = NEW."IsSystem"
                ,"IsSimple" = NEW."IsSimple"
            WHERE "SysLookup"."Id" = NEW."Id";
            RETURN NEW;
        ELSIF TG_OP = 'DELETE' THEN
            DELETE FROM "public"."SysLookup" WHERE OLD."Id" = "SysLookup"."Id";
            RETURN OLD;
        END IF;
        RETURN NEW;
    END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER "ITR_VwSysLookup_IUD"
    INSTEAD OF INSERT OR UPDATE OR DELETE ON "public"."VwSysLookup"
    FOR EACH ROW EXECUTE PROCEDURE "public"."ITR_VwSysLookup_IUD_Func"();
```

Пример 2 (представления)

Пример. Пример SQL-скрипта, который иллюстрирует использование правила вместо триггера в PostgreSQL.

MS SQL

```
-- Использование rule вместо instead of триггера
-- MSSQL
IF EXISTS (SELECT * FROM sys.views WHERE object_id = OBJECT_ID(N'[dbo].[VwAdministrativeObjects]
DROP VIEW [dbo].[VwAdministrativeObjects]
GO
CREATE VIEW [dbo].[VwAdministrativeObjects]
AS
WITH
[SysSchemaAdministrationProperties] AS (
SELECT [AdministrationPropertiesAll].[Id] AS [SysSchemaId],
    max([AdministrationPropertiesAll].[AdministratedByOperations]) AS [AdministratedByOperations
    max([AdministrationPropertiesAll].[AdministratedByColumns]) AS [AdministratedByColumns],
    max([AdministrationPropertiesAll].[AdministratedByRecords]) AS [AdministratedByRecords],
    max([AdministrationPropertiesAll].[IsTrackChangesInDB]) AS [IsTrackChangesInDB]
FROM (
    SELECT [SysSchema].[Id],
            (CASE WHEN EXISTS (
                SELECT 1
                FROM [SysSchemaProperty]
                WHERE (([SysSchemaProperty].[SysSchemaId] = [SysSchema].[Id] AND [SysSchema].[Ex
                    OR [SysSchemaProperty].[SysSchemaId] = [DerivedSysSchema].[Id])
                    AND [SysSchemaProperty].[Name] = 'AdministratedByOperations'
                    AND [SysSchemaProperty].[Value] = 'True'
                    AND [SysSchemaProperty].[SysSchemaId] IS NOT NULL
                )
            THEN 1 ELSE 0 END) AS [AdministratedByOperations],
            (CASE WHEN EXISTS (
                SELECT 1
                FROM [SysSchemaProperty]
                WHERE (([SysSchemaProperty].[SysSchemaId] = [SysSchema].[Id] AND [SysSchema].[Ex
                    OR [SysSchemaProperty].[SysSchemaId] = [DerivedSysSchema].[Id])
                    AND [SysSchemaProperty].[Name] = 'AdministratedByColumns'
                    AND [SysSchemaProperty].[Value] = 'True'
                    AND [SysSchemaProperty].[SysSchemaId] IS NOT NULL
                )
            THEN 1 ELSE 0 END) AS [AdministratedByColumns],
            (CASE WHEN EXISTS (
                SELECT 1
                FROM [SysSchemaProperty]
                WHERE (([SysSchemaProperty].[SysSchemaId] = [SysSchema].[Id] AND [SysSchema].[Ex
                    OR [SysSchemaProperty].[SysSchemaId] = [DerivedSysSchema].[Id])
                    AND [SysSchemaProperty].[Name] = 'AdministratedByRecords'
                    AND [SysSchemaProperty].[Value] = 'True'
                    AND [SysSchemaProperty].[SysSchemaId] IS NOT NULL
```

```
                )
            THEN 1 ELSE 0 END) AS [AdministratedByRecords],
            (CASE WHEN EXISTS (
                SELECT 1
                FROM [SysSchemaProperty]
                WHERE (([SysSchemaProperty].[SysSchemaId] = [SysSchema].[Id] AND [SysSchema].[Ex
                    OR [SysSchemaProperty].[SysSchemaId] = [DerivedSysSchema].[Id])
                    AND [SysSchemaProperty].[Name] = 'IsTrackChangesInDB'
                    AND [SysSchemaProperty].[Value] = 'True'
                    AND [SysSchemaProperty].[SysSchemaId] IS NOT NULL
                )
            THEN 1 ELSE 0 END) AS [IsTrackChangesInDB]
    FROM [SysSchema]
    LEFT OUTER JOIN [SysSchema] AS [DerivedSysSchema] ON ([SysSchema].[Id] = [DerivedSysSchema].
    WHERE [SysSchema].[ManagerName] = 'EntitySchemaManager'
        AND [SysSchema].[ExtendParent] = 0
) AS [AdministrationPropertiesAll]
GROUP BY [AdministrationPropertiesAll].[Id]
)
SELECT [BaseSchemas].[UId] AS [Id],
    [BaseSchemas].[UId],
    [BaseSchemas].[CreatedOn],
    [BaseSchemas].[CreatedById],
    [BaseSchemas].[ModifiedOn],
    [BaseSchemas].[ModifiedById],
    [BaseSchemas].[Name],
    [VwSysSchemaExtending].[TopExtendingCaption] as Caption,
    [BaseSchemas].[Description],
    (CASE WHEN EXISTS (
        SELECT 1
        FROM [SysLookup]
        WHERE [SysLookup].[SysEntitySchemaUId] = [BaseSchemas].[UId])
        THEN 1 ELSE 0 END) AS [IsLookup],
    (CASE WHEN EXISTS (
        SELECT 1 FROM [SysModule]
        INNER JOIN [SysModuleEntity] ON [SysModuleEntity].[Id] = [SysModule].[SysModuleEntityId]
        WHERE [BaseSchemas].[UId] = [SysModuleEntity].[SysEntitySchemaUId])
        THEN 1 ELSE 0 END) AS [IsModule],
    [SysSchemaAdministrationProperties].[AdministratedByOperations],
    [SysSchemaAdministrationProperties].[AdministratedByColumns],
    [SysSchemaAdministrationProperties].[AdministratedByRecords],
    [SysSchemaAdministrationProperties].[IsTrackChangesInDB],
    [SysWorkspaceId],
    [BaseSchemas].[ProcessListeners],
    (CASE WHEN EXISTS (
        SELECT 1
        FROM [SysSSPEntitySchemaAccessList]
        WHERE [SysSSPEntitySchemaAccessList].[EntitySchemaUId] = [BaseSchemas].[UId]
        )
```

```
        THEN 1 ELSE 0 END) AS [IsInSSPEntitySchemaAccessList]
FROM  [SysSchema] as [BaseSchemas]
INNER JOIN [VwSysSchemaExtending] ON BaseSchemas.[Id] = [VwSysSchemaExtending].[BaseSchemaId]
INNER JOIN [SysPackage] on [BaseSchemas].[SysPackageId] = [SysPackage].[Id]
INNER JOIN [SysSchemaAdministrationProperties] ON [BaseSchemas].[Id] = [SysSchemaAdministrationP
GO
CREATE TRIGGER [dbo].[TRVwAdministrativeObjects_IU]
ON [dbo].[VwAdministrativeObjects]
    INSTEAD OF UPDATE
AS
BEGIN
    SET NOCOUNT ON;
    RETURN
END
GO
```

Postgre SQL

```
-- Использование rule вместо instead of триггера
-- PostgreSql
DROP VIEW IF EXISTS public."VwAdministrativeObjects";
DROP RULE IF EXISTS RU_VwAdministrativeObjects ON "VwAdministrativeObjects";

CREATE VIEW public."VwAdministrativeObjects" AS
WITH SysSchemaAdministrationProperties AS (
    SELECT AdministrationPropertiesAll.Id "SysSchemaId",
        MAX(AdministrationPropertiesAll.AdministratedByOperations) "AdministratedByOperations",
        MAX(AdministrationPropertiesAll.AdministratedByColumns) "AdministratedByColumns",
        MAX(AdministrationPropertiesAll.AdministratedByRecords) "AdministratedByRecords",
        MAX(AdministrationPropertiesAll.IsTrackChangesInDB) "IsTrackChangesInDB"
    FROM (
        SELECT ss."Id" Id
            ,(CASE WHEN EXISTS (
                SELECT 1
                FROM "SysSchemaProperty" ssp
                WHERE ((ssp."SysSchemaId" = ss."Id" AND NOT ss."ExtendParent") OR ssp."SysSchema
                    AND ssp."Name" = 'AdministratedByOperations'
                    AND ssp."Value" = 'True'
                    AND ssp."SysSchemaId" IS NOT NULL
            ) THEN 1 ELSE 0 END) AdministratedByOperations
            ,(CASE WHEN EXISTS (
                SELECT 1
                FROM "SysSchemaProperty" ssp
                WHERE ((ssp."SysSchemaId" = ss."Id" AND NOT ss."ExtendParent") OR ssp."SysSchema
                    AND ssp."Name" = 'AdministratedByColumns'
                    AND ssp."Value" = 'True'
```

```
                    AND ssp."SysSchemaId" IS NOT NULL
            ) THEN 1 ELSE 0 END) AdministratedByColumns
            ,(CASE WHEN EXISTS (
                SELECT 1
                FROM "SysSchemaProperty" ssp
                WHERE ((ssp."SysSchemaId" = ss."Id" AND NOT ss."ExtendParent") OR ssp."SysSchema
                    AND ssp."Name" = 'AdministratedByRecords'
                    AND ssp."Value" = 'True'
                    AND ssp."SysSchemaId" IS NOT NULL
            ) THEN 1 ELSE 0 END) AdministratedByRecords
            ,(CASE WHEN EXISTS (
                SELECT 1
                FROM "SysSchemaProperty" ssp WHERE ((ssp."SysSchemaId" = ss."Id" AND NOT ss."Ext
                    OR ssp."SysSchemaId" = DerivedSysSchema."Id")
                    AND ssp."Name" = 'IsTrackChangesInDB'
                    AND ssp."Value" = 'True'
                    AND ssp."SysSchemaId" IS NOT NULL
            ) THEN 1 ELSE 0 END) IsTrackChangesInDB
        FROM "SysSchema" ss
        LEFT OUTER JOIN "SysSchema" DerivedSysSchema ON (ss."Id" = DerivedSysSchema."ParentId" A
        WHERE ss."ManagerName" = 'EntitySchemaManager' AND NOT ss."ExtendParent"
    ) AdministrationPropertiesAll
    GROUP BY AdministrationPropertiesAll.Id
)
SELECT BaseSchema."UId" "Id"
    ,BaseSchema."UId"
    ,BaseSchema."CreatedOn"
    ,BaseSchema."CreatedById"
    ,BaseSchema."ModifiedOn"
    ,BaseSchema."ModifiedById"
    ,BaseSchema."Name"
    ,public."VwSysSchemaExtending"."TopExtendingCaption" "Caption"
    ,BaseSchema."Description"
    ,EXISTS (
        SELECT 1
        FROM "SysLookup"
        WHERE "SysEntitySchemaUId" = BaseSchema."UId"
    ) "IsLookup"
    ,EXISTS (
        SELECT 1
        FROM "SysModule" sm
        INNER JOIN "SysModuleEntity" sme ON sme."Id" = sm."SysModuleEntityId"
        WHERE BaseSchema."UId" = sme."SysEntitySchemaUId"
    ) "IsModule"
    ,SysSchemaAdministrationProperties."AdministratedByOperations"::BOOLEAN
    ,SysSchemaAdministrationProperties."AdministratedByColumns"::BOOLEAN
    ,SysSchemaAdministrationProperties."AdministratedByRecords"::BOOLEAN
    ,SysSchemaAdministrationProperties."IsTrackChangesInDB"::BOOLEAN
    ,"SysWorkspaceId"
```

```
    ,BaseSchema."ProcessListeners"
    ,EXISTS (
        SELECT 1
        FROM "SysSSPEntitySchemaAccessList"
        WHERE "EntitySchemaUId" = BaseSchema."UId"
    ) "IsInSSPEntitySchemaAccessList"
FROM "SysSchema" BaseSchema
INNER JOIN "VwSysSchemaExtending" ON BaseSchema."Id" = "VwSysSchemaExtending"."BaseSchemaId"
INNER JOIN "SysPackage" on BaseSchema."SysPackageId" = "SysPackage"."Id"
INNER JOIN SysSchemaAdministrationProperties ON BaseSchema."Id" = SysSchemaAdministrationPropert

CREATE RULE RU_VwAdministrativeObjects AS
    ON UPDATE TO "VwAdministrativeObjects"
    DO INSTEAD NOTHING;
```

Пример 3 (хранимые процедуры)

Пример. Пример SQL-скрипта, который создает хранимую процедуру, использующую циклы, курсоры и временные таблицы.

MS SQL

```
-- Хранимая процедура, в которой используются циклы, курсоры, временные таблицы
-- MSSQL
IF NOT OBJECT_ID('[dbo].[tsp_ActualizeUserRoles]') IS NULL
BEGIN
    DROP PROCEDURE [dbo].[tsp_ActualizeUserRoles]
END
GO

CREATE PROCEDURE dbo.tsp_ActualizeUserRoles (@UserId uniqueidentifier)
AS
BEGIN
    SET NOCOUNT ON

    IF OBJECT_ID('tempdb..#AdminUnitListTemp') IS NOT NULL
    BEGIN
        DROP TABLE [#AdminUnitListTemp];
    END;
    CREATE TABLE [#AdminUnitListTemp] (
        [UserId] uniqueidentifier NOT NULL,
        [Id] uniqueidentifier NOT NULL,
        [Name] NVARCHAR(250) NOT NULL,
```

```
    [ParentRoleId] uniqueidentifier NULL,
    [Granted] BIT NULL
);

DECLARE @GetAdminUnitList TABLE (
    [Id] uniqueidentifier NOT NULL,
    [Name] nvarchar(260) NOT NULL,
    [ParentRoleId] uniqueidentifier NULL
);
DECLARE @NewRoles TABLE ([Id] uniqueidentifier NOT NULL);
DECLARE @OldUserRoles TABLE ([Id] uniqueidentifier NOT NULL);

DECLARE @getUserAdminUnits CURSOR;
DECLARE @SysAdminUnitRoles TABLE (
    [Id] uniqueidentifier,
    [Name] nvarchar(260),
    [ParentRoleId] uniqueidentifier
);
DECLARE @ManagersBeforeActualization TABLE ([Id] uniqueidentifier NOT NULL);
DECLARE @ManagersAfterActualization TABLE ([Id] uniqueidentifier NOT NULL);
DECLARE @StillManagers TABLE ([Id] uniqueidentifier NOT NULL);
DECLARE @NoLongerManagers TABLE ([Id] uniqueidentifier NOT NULL);
DECLARE @NewManagers TABLE ([Id] uniqueidentifier NOT NULL);
DECLARE @SysAdminUnitId uniqueidentifier;

-- Old user roles
INSERT INTO @OldUserRoles
    SELECT DISTINCT [SysAdminUnitInRole].[SysAdminUnitRoleId] [Id]
    FROM [SysAdminUnitInRole]
    WHERE [SysAdminUnitInRole].[SysAdminUnitId] = @UserId

-- Old user managers
INSERT INTO @ManagersBeforeActualization
    SELECT DISTINCT [SysUserInRole].[SysUserId] [Id]
    FROM [SysAdminUnitInRole]
    INNER JOIN [SysAdminUnit] [Roles]
        ON [SysAdminUnitInRole].[SysAdminUnitRoleId] = [Roles].[Id]
    INNER JOIN @OldUserRoles
        ON [Roles].[ParentRoleId] = [@OldUserRoles].[Id]
    INNER JOIN [SysUserInRole]
        ON [SysUserInRole].[SysRoleId] = [Roles].[Id]
    WHERE [Roles].[SysAdminUnitTypeValue] = 2

-- Get and insert new user roles
INSERT INTO @GetAdminUnitList EXEC [tsp_GetAdminUnitList] @UserId=@UserId;
INSERT INTO @NewRoles SELECT [Id] FROM @GetAdminUnitList;
DELETE FROM [SysAdminUnitInRole] WHERE [SysAdminUnitId] = @UserId;
INSERT INTO [SysAdminUnitInRole] ([SysAdminUnitId], [SysAdminUnitRoleId])
    SELECT DISTINCT @UserId, [Id] FROM @NewRoles;
```

```
-- User managers after actualization
INSERT INTO @ManagersAfterActualization
    SELECT DISTINCT
        [SysUserInRole].[SysUserId] [Id]
    FROM [SysAdminUnitInRole]
    INNER JOIN [SysAdminUnit] [Roles]
        ON [SysAdminUnitInRole].[SysAdminUnitRoleId] = [Roles].[Id]
    INNER JOIN @NewRoles NewRoles
        ON [Roles].[ParentRoleId] = NewRoles.[Id]
    INNER JOIN [SysUserInRole]
        ON [SysUserInRole].[SysRoleId] = [Roles].[Id]
    WHERE [Roles].[SysAdminUnitTypeValue] = 2;

-- New (who were not but become) user managers
INSERT INTO @NewManagers
    SELECT [Id] FROM @ManagersAfterActualization AS managersAfterActualization
        WHERE NOT EXISTS (
            SELECT NULL
            FROM @ManagersBeforeActualization AS managersBeforeActualization
            WHERE managersBeforeActualization.[Id] = managersAfterActualization.[Id]
        );

-- Add all user roles to new managers and their grantee-users, if they arent already have
SET @getUserAdminUnits = CURSOR FOR
    SELECT DISTINCT [Id] FROM (
        SELECT [Id] FROM @NewManagers
        UNION
        SELECT [GranteeSysAdminUnitId]
        FROM [SysAdminUnitGrantedRight]
        WHERE EXISTS (
            SELECT NULL FROM @NewManagers as newManagers
            WHERE [SysAdminUnitGrantedRight].[GrantorSysAdminUnitId] = newManagers.[Id]
        )
    ) Roles;

OPEN @getUserAdminUnits;
FETCH NEXT
FROM @getUserAdminUnits INTO @SysAdminUnitId;
WHILE @@FETCH_STATUS = 0
BEGIN
    INSERT INTO [SysAdminUnitInRole] ([SysAdminUnitId], [SysAdminUnitRoleId])
        SELECT DISTINCT @SysAdminUnitId, [Id]
        FROM @NewRoles AS newRoles
        WHERE NOT EXISTS (
            SELECT 1
            FROM [SysAdminUnitInRole]
            WHERE [SysAdminUnitInRole].[SysAdminUnitId] = @SysAdminUnitId
```

```
                AND [SysAdminUnitInRole].[SysAdminUnitRoleId] = newRoles.[Id]
            );
        FETCH NEXT FROM @getUserAdminUnits INTO @SysAdminUnitId;
    END;
    CLOSE @getUserAdminUnits;
    DEALLOCATE @getUserAdminUnits;

    DECLARE @isUserLostAtLeastOneRole INT = (
        SELECT COUNT(*)
        FROM @OldUserRoles AS oldUserRoles
        WHERE NOT EXISTS (
            SELECT 1
            FROM @NewRoles AS newUserRoles
            WHERE newUserRoles.[Id] = oldUserRoles.[Id]
        )
    );

    -- Still (who were and remained) user managers
    INSERT INTO @StillManagers
        SELECT  DISTINCT managersAfterActualization.[Id] AS [Id]
        FROM @ManagersAfterActualization AS managersAfterActualization
            JOIN @ManagersBeforeActualization AS managersBeforeActualization
                ON managersAfterActualization.[Id] = managersBeforeActualization.[Id];

    -- If user lost at least one role, we need to actualize all his still-managers.
    -- If not (user only gained new roles) - we just add to still-managers and their grantee-use
    IF (@isUserLostAtLeastOneRole = 0)
    BEGIN
        -- Add all new user roles to his still-managers and to their grantee-users
        SET @getUserAdminUnits = CURSOR FOR
            SELECT DISTINCT [Id] FROM (
                SELECT stillManagers.[Id] AS [Id]
                FROM @StillManagers AS stillManagers
                UNION
                SELECT [GranteeSysAdminUnitId]
                FROM [SysAdminUnitGrantedRight]
                WHERE EXISTS (
                    SELECT NULL
                    FROM @StillManagers AS stillManagers
                    WHERE stillManagers.[Id] = [GrantorSysAdminUnitId]
                )
            ) Roles;

        OPEN @getUserAdminUnits;
        FETCH NEXT
        FROM @getUserAdminUnits INTO @SysAdminUnitId;
        WHILE @@FETCH_STATUS = 0
        BEGIN
            INSERT INTO [SysAdminUnitInRole] ([SysAdminUnitId], [SysAdminUnitRoleId])
```

```
                SELECT DISTINCT @SysAdminUnitId, [Id]
                FROM @NewRoles AS newRoles
                WHERE NOT EXISTS (
                    SELECT 1
                    FROM [SysAdminUnitInRole]
                    WHERE [SysAdminUnitInRole].[SysAdminUnitId] = @SysAdminUnitId
                        AND [SysAdminUnitInRole].[SysAdminUnitRoleId] = newRoles.[Id]
                );
            FETCH NEXT FROM @getUserAdminUnits INTO @SysAdminUnitId;
        END;
        CLOSE @getUserAdminUnits;
        DEALLOCATE @getUserAdminUnits;
    END ELSE
    BEGIN
        --Actualize all roles for still-managers
        SET @getUserAdminUnits = CURSOR FOR
            SELECT DISTINCT [Id]
            FROM @StillManagers
            UNION
                SELECT [GranteeSysAdminUnitId]
                FROM [SysAdminUnitGrantedRight]
                WHERE EXISTS (
                SELECT NULL
                FROM @StillManagers AS stillManagers
                WHERE stillManagers.[Id] = [GrantorSysAdminUnitId]
                );

        OPEN @getUserAdminUnits;
        FETCH NEXT
        FROM @getUserAdminUnits INTO @SysAdminUnitId;
        WHILE @@FETCH_STATUS = 0
        BEGIN
            DELETE FROM @SysAdminUnitRoles;
            INSERT INTO @SysAdminUnitRoles
                EXEC [tsp_GetAdminUnitList] @UserId=@SysAdminUnitId;
            BEGIN TRAN;
                DELETE FROM [dbo].[SysAdminUnitInRole] WHERE SysAdminUnitId = @SysAdminUnitId;
                INSERT INTO [dbo].[SysAdminUnitInRole] (SysAdminUnitId, SysAdminUnitRoleId)
                    SELECT @SysAdminUnitId, [Id] FROM @SysAdminUnitRoles;
            COMMIT;
            FETCH NEXT
                FROM @getUserAdminUnits INTO @SysAdminUnitId;
        END;
        CLOSE @getUserAdminUnits;
        DEALLOCATE @getUserAdminUnits;
    END;

    -- No longer (who were but not remained) user managers
```

```
INSERT INTO @NoLongerManagers
    SELECT [Id] FROM @ManagersBeforeActualization as managersBeforeActualization
        WHERE NOT EXISTS (
            SELECT NULL
            FROM @ManagersAfterActualization AS managersAfterActualization
            WHERE managersAfterActualization.[Id] = managersBeforeActualization.[Id]
        );

--Actualize roles for all noLonger-managers, his grantee-users and all grantee-users of user
SET @getUserAdminUnits = CURSOR FOR
    SELECT DISTINCT [Id] FROM (
        SELECT [Id] FROM @NoLongerManagers
        UNION
        SELECT [GranteeSysAdminUnitId]
        FROM [SysAdminUnitGrantedRight]
        WHERE EXISTS (
            SELECT NULL
            FROM @NoLongerManagers AS noLongerManagers
            WHERE noLongerManagers.[Id] = [GrantorSysAdminUnitId]
        )
        UNION ALL
        SELECT GranteeSysAdminUnitId
        FROM SysAdminUnitGrantedRight
        WHERE GrantorSysAdminUnitId = @UserId
    ) Roles;

OPEN @getUserAdminUnits;
FETCH NEXT
    FROM @getUserAdminUnits INTO @SysAdminUnitId;
WHILE @@FETCH_STATUS = 0
BEGIN
    DELETE FROM @SysAdminUnitRoles;
    INSERT INTO @SysAdminUnitRoles
        EXEC [tsp_GetAdminUnitList] @UserId=@SysAdminUnitId;
    BEGIN TRAN;
        DELETE FROM [dbo].[SysAdminUnitInRole] WHERE SysAdminUnitId = @SysAdminUnitId;
        INSERT INTO [dbo].[SysAdminUnitInRole] (SysAdminUnitId, SysAdminUnitRoleId)
            SELECT @SysAdminUnitId, [Id] FROM @SysAdminUnitRoles;
    COMMIT;
    FETCH NEXT
        FROM @getUserAdminUnits INTO @SysAdminUnitId;
END;
CLOSE @getUserAdminUnits;
DEALLOCATE @getUserAdminUnits;

IF OBJECT_ID('tempdb..#AdminUnitListTemp') IS NOT NULL
BEGIN
    DROP TABLE [#AdminUnitListTemp];
END;
```

```
END;
GO
```

Postgre SQL

```
-- Хранимая процедура, в которой используются циклы, курсоры, временные таблицы
-- PostgreSql
DROP FUNCTION IF EXISTS "tsp_ActualizeUserRoles";
CREATE FUNCTION "tsp_ActualizeUserRoles"(
    UserId UUID
)
RETURNS VOID
AS $$
DECLARE
    getUserNewManagers CURSOR FOR
        SELECT DISTINCT "Id" FROM (
            SELECT "Id" FROM "NewManagers"
            UNION
            SELECT "GranteeSysAdminUnitId"
            FROM "SysAdminUnitGrantedRight"
            WHERE EXISTS (
                SELECT NULL FROM "NewManagers" as "newManagers"
                WHERE "SysAdminUnitGrantedRight"."GrantorSysAdminUnitId" = "newManagers"."Id"
            )
        ) "Roles";
    lostUserRolesCount INT;
    getUserStillManagers CURSOR FOR
        SELECT DISTINCT "stillManagers"."Id" AS "Id"
        FROM "StillManagers" AS "stillManagers"
        UNION
        SELECT "GranteeSysAdminUnitId"
        FROM "SysAdminUnitGrantedRight"
        WHERE EXISTS (
            SELECT NULL
            FROM "StillManagers" AS "stillManagers"
            WHERE "stillManagers"."Id" = "GrantorSysAdminUnitId"
        );
    getUserNoLongerManagers CURSOR FOR
        SELECT DISTINCT "Id" FROM (
            SELECT "Id"
            FROM "NoLongerManagers"
            UNION
            SELECT "GranteeSysAdminUnitId"
            FROM "SysAdminUnitGrantedRight"
            WHERE EXISTS (
                SELECT NULL
```

```
                FROM "NoLongerManagers" AS "noLongerManagers"
                WHERE "noLongerManagers"."Id" = "GrantorSysAdminUnitId"
            )
            UNION ALL
            SELECT "GranteeSysAdminUnitId"
            FROM "SysAdminUnitGrantedRight"
            WHERE "GrantorSysAdminUnitId" = UserId
        ) "Roles";
BEGIN
    DROP TABLE IF EXISTS "GetAdminUnitListTmp";
    CREATE TEMP TABLE "GetAdminUnitListTmp" (
        "Id" UUID,
        "Name" VARCHAR(250),
        "ParentRoleId" UUID
    );

    DROP TABLE IF EXISTS "SysAdminUnitRoles";
    CREATE TEMP TABLE "SysAdminUnitRoles" (
        "Id" UUID,
        "Name" VARCHAR(250),
        "ParentRoleId" UUID
    );

    -- Old user roles
    DROP TABLE IF EXISTS "OldUserRoles";
    CREATE TEMP TABLE "OldUserRoles" (
        "Id" UUID
    );
    INSERT INTO "OldUserRoles"
        SELECT DISTINCT "SysAdminUnitInRole"."SysAdminUnitRoleId" "Id"
        FROM "SysAdminUnitInRole"
        WHERE "SysAdminUnitInRole"."SysAdminUnitId" = UserId;

    -- Old user managers
    DROP TABLE IF EXISTS "ManagersBeforeActualization";
    CREATE TEMP TABLE "ManagersBeforeActualization" (
        "Id" UUID
    );
    INSERT INTO "ManagersBeforeActualization"
        SELECT DISTINCT "SysUserInRole"."SysUserId" "Id"
        FROM "SysAdminUnitInRole"
        INNER JOIN "SysAdminUnit" "Roles"
            ON "SysAdminUnitInRole"."SysAdminUnitRoleId" = "Roles"."Id"
        INNER JOIN "OldUserRoles"
            ON "Roles"."ParentRoleId" = "OldUserRoles"."Id"
        INNER JOIN "SysUserInRole"
            ON "SysUserInRole"."SysRoleId" = "Roles"."Id"
        WHERE "Roles"."SysAdminUnitTypeValue" = 2;
```

```
-- Get and insert new user roles
DROP TABLE IF EXISTS "GetAdminUnitList";
CREATE TEMP TABLE "GetAdminUnitList" (
    "Id" UUID,
    "Name" VARCHAR(250),
    "ParentRoleId" UUID
);
DROP TABLE IF EXISTS "NewRoles";
CREATE TEMP TABLE "NewRoles" (
    "Id" UUID
);
INSERT INTO "GetAdminUnitList" SELECT * FROM "tsp_GetAdminUnitList"(UserId);
INSERT INTO "NewRoles" SELECT "Id" FROM "GetAdminUnitList";
DELETE FROM "SysAdminUnitInRole" WHERE "SysAdminUnitId" = UserId;
INSERT INTO "SysAdminUnitInRole" ("SysAdminUnitId", "SysAdminUnitRoleId")
    SELECT DISTINCT UserId, "Id" FROM "NewRoles";

-- User managers after actualization
DROP TABLE IF EXISTS "ManagersAfterActualization";
CREATE TEMP TABLE "ManagersAfterActualization" (
    "Id" UUID
);
INSERT INTO "ManagersAfterActualization"
    SELECT DISTINCT
        "SysUserInRole"."SysUserId" "Id"
    FROM "SysAdminUnitInRole"
    INNER JOIN "SysAdminUnit" "Roles"
        ON "SysAdminUnitInRole"."SysAdminUnitRoleId" = "Roles"."Id"
    INNER JOIN "NewRoles" "NewRoles"
        ON "Roles"."ParentRoleId" = "NewRoles"."Id"
    INNER JOIN "SysUserInRole"
        ON "SysUserInRole"."SysRoleId" = "Roles"."Id"
    WHERE "Roles"."SysAdminUnitTypeValue" = 2;

-- New (who were not but become) user managers
DROP TABLE IF EXISTS "NewManagers";
CREATE TEMP TABLE "NewManagers" (
    "Id" UUID
);
INSERT INTO "NewManagers"
    SELECT "Id" FROM "ManagersAfterActualization" AS "managersAfterActualization"
        WHERE NOT EXISTS (
            SELECT NULL
            FROM "ManagersBeforeActualization" AS "managersBeforeActualization"
            WHERE "managersBeforeActualization"."Id" = "managersAfterActualization"."Id"
        );

-- Add all user roles to new managers and their grantee-users, if they arent already have
```

```
    FOR UserNewManager IN getUserNewManagers LOOP
        EXIT WHEN UserNewManager = NULL;
        INSERT INTO "SysAdminUnitInRole" ("SysAdminUnitId", "SysAdminUnitRoleId")
            SELECT DISTINCT UserNewManager."Id", "Id"
            FROM "NewRoles" AS "newRoles"
            WHERE NOT EXISTS (
                SELECT 1
                FROM "SysAdminUnitInRole"
                WHERE "SysAdminUnitInRole"."SysAdminUnitId" = UserNewManager."Id"
                AND "SysAdminUnitInRole"."SysAdminUnitRoleId" = "newRoles"."Id"
            );
    END LOOP;

    SELECT COUNT(*) INTO lostUserRolesCount
    FROM "OldUserRoles" AS "oldUserRoles"
    WHERE NOT EXISTS (
        SELECT 1
        FROM "NewRoles" AS "newUserRoles"
        WHERE "newUserRoles"."Id" = "oldUserRoles"."Id"
    );

    -- Still (who were and remained) user managers
    DROP TABLE IF EXISTS "StillManagers";
    CREATE TEMP TABLE "StillManagers" (
        "Id" UUID
    );
    INSERT INTO "StillManagers"
        SELECT DISTINCT "managersAfterActualization"."Id" AS "Id"
        FROM "ManagersAfterActualization" AS "managersAfterActualization"
            JOIN "ManagersBeforeActualization" AS "managersBeforeActualization"
                ON "managersAfterActualization"."Id" = "managersBeforeActualization"."Id";

    -- If user lost at least one role, we need to actualize all his still-managers.
    -- If not (user only gained new roles) - we just add to still-managers and their grantee-use
    IF lostUserRolesCount = 0 THEN

        -- Add all new user roles to his still-managers and to their grantee-users
        FOR UserStillManager IN getUserStillManagers LOOP
            EXIT WHEN UserStillManager = NULL;
            INSERT INTO "SysAdminUnitInRole" ("SysAdminUnitId", "SysAdminUnitRoleId")
                SELECT DISTINCT UserStillManager."Id", "Id"
                FROM "NewRoles" AS "newRoles"
                WHERE NOT EXISTS (
                    SELECT 1
                    FROM "SysAdminUnitInRole"
                    WHERE "SysAdminUnitInRole"."SysAdminUnitId" = UserStillManager."Id"
                        AND "SysAdminUnitInRole"."SysAdminUnitRoleId" = "newRoles"."Id"
                );
        END LOOP;
```

```
    ELSE

        --Actualize all roles for still-managers
        FOR UserStillManager IN getUserStillManagers LOOP
            EXIT WHEN UserStillManager = NULL;
            DELETE FROM "SysAdminUnitRoles";
            INSERT INTO "SysAdminUnitRoles"
                SELECT * FROM "tsp_GetAdminUnitList"(UserStillManager."Id");
                DELETE FROM "SysAdminUnitInRole" WHERE "SysAdminUnitId" = UserStillManager."Id";
                INSERT INTO "SysAdminUnitInRole" ("SysAdminUnitId", "SysAdminUnitRoleId")
                    SELECT UserStillManager."Id", "Id" FROM "SysAdminUnitRoles";
        END LOOP;
    END IF;

    -- No longer (who were but not remained) user managers
    DROP TABLE IF EXISTS "NoLongerManagers";
    CREATE TEMP TABLE "NoLongerManagers" (
        "Id" UUID
    );
    INSERT INTO "NoLongerManagers"
        SELECT "Id" FROM "ManagersBeforeActualization" AS "managersBeforeActualization"
            WHERE NOT EXISTS (
                SELECT NULL
                FROM "ManagersAfterActualization" AS "managersAfterActualization"
                WHERE "managersAfterActualization"."Id" = "managersBeforeActualization"."Id"
            );


    -- Actualize roles for all noLonger-managers, his grantee-users and all grantee-users of use
    FOR UserNoLongerManager IN getUserNoLongerManagers LOOP
        EXIT WHEN UserNoLongerManager = NULL;
        DELETE FROM "SysAdminUnitRoles";
        INSERT INTO "SysAdminUnitRoles"
            SELECT * FROM "tsp_GetAdminUnitList"(UserNoLongerManager."Id");
            DELETE FROM "SysAdminUnitInRole"
                WHERE "SysAdminUnitId" = UserNoLongerManager."Id";
            INSERT INTO "SysAdminUnitInRole" ("SysAdminUnitId", "SysAdminUnitRoleId")
                SELECT UserNoLongerManager."Id", "Id" FROM "SysAdminUnitRoles";
    END LOOP;

    DROP TABLE IF EXISTS "GetAdminUnitListTmp";
END;
$$ LANGUAGE plpgsql;
```

Пример 4 (хранимые процедуры)

Пример. Пример рекурсивной хранимой процедуры, которая возвращает таблицу и в которой используется `PERFORM`.

MS SQL

```
-- Рекурсивная хранимая процедура, которая возвращает таблицу и в которой используется PERFORM:
-- MSSQL
IF NOT OBJECT_ID('[dbo].[tsp_GetAdminUnitList]') IS NULL
BEGIN
    DROP PROCEDURE [dbo].[tsp_GetAdminUnitList];
END;
GO

CREATE PROCEDURE dbo.tsp_GetAdminUnitList (
    @UserId uniqueidentifier, @Granted BIT = 0
)
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @StartNestedLevel INT;

    IF object_id('tempdb..#AdminUnitList') IS NULL
    BEGIN
        CREATE TABLE [#AdminUnitList]
        (
            [Id] uniqueidentifier NOT NULL,
            [Name] NVARCHAR(250) NULL,
            [ParentRoleId] uniqueidentifier NULL,
            [Granted] BIT NULL,
            Level INT NOT NULL
        );
        SET @StartNestedLevel = @@NESTLEVEL;
    END;

    DECLARE @ConnectionType INT = (SELECT [ConnectionType] FROM SysAdminUnit WHERE [Id] = @UserI

    -- #AdminUnitListTemp should be created in tsp_ActualizeUserRoles or in tsp_ActualizeAdminUr
    DECLARE @IsAdminUnitListTempExists BIT = OBJECT_ID('tempdb..#AdminUnitListTemp');

    IF (@IsAdminUnitListTempExists IS NULL)
    BEGIN
        WITH
        [MainSelect] AS (
            SELECT
                [Id] [Id],
```

```
            [Name] [Name],
            [ParentRoleId] [ParentRoleId]
        FROM
            [dbo].[SysAdminUnit]
        WHERE
            ([SysAdminUnitTypeValue] <= 4 OR [SysAdminUnitTypeValue] = 6)
        AND [ConnectionType] = @ConnectionType
        UNION ALL
        SELECT
            [Id] [Id],
            [Name] [Name],
            [ParentRoleId] [ParentRoleId]
        FROM
            [dbo].[SysAdminUnit]
        WHERE
            [Id] = @UserId),
    [ChiefUnitsSelect] AS (
        (
            SELECT
                [Chief].[ParentRoleId] [Id]
            FROM
                [dbo].[SysUserInRole] userInRole
                INNER JOIN [dbo].[SysAdminUnit] sau ON (sau.[Id] = userInRole.[SysUserId])
                INNER JOIN [dbo].[SysAdminUnit] [Chief] ON ([Chief].[Id] = userInRole.[SysRo
            WHERE
                sau.[Id] = @UserId AND NOT (userInRole.[SysRoleId] IS NULL) AND [Chief].[Sys
            UNION ALL
            SELECT
                [Chief].[ParentRoleId] [Id]
            FROM
                [dbo].[SysAdminUnit] [Chief]
            WHERE
                [Chief].[Id] = @UserId AND [Chief].[SysAdminUnitTypeValue] = 2
        )
        UNION ALL
        SELECT
            sau.[Id]
        FROM
            [ChiefUnitsSelect]
            INNER JOIN [dbo].[SysAdminUnit] sau ON (sau.[ParentRoleId] = [ChiefUnitsSelect].
        WHERE
            sau.[SysAdminUnitTypeValue] < 4
    ),
    [HierarchicalSelect] AS (
        SELECT
            [Id],
            [Name],
            [ParentRoleId],
            0 [Level]
```

```
            FROM
                [MainSelect] [SelectStartLevel]
            WHERE
                [Id] IN (
                    SELECT
                        userInRole.[SysRoleId]
                    FROM
                        [dbo].[SysUserInRole] userInRole
                        INNER JOIN [dbo].[SysAdminUnit] sau ON (sau.[Id] = userInRole.[SysUserId
                    WHERE
                        sau.[Id] = @UserId
                    UNION ALL
                    SELECT [Id] FROM [ChiefUnitsSelect]
                    UNION ALL
                    SELECT
                        [Id]
                    FROM
                        [dbo].[SysAdminUnit]
                    WHERE
                        ([ParentRoleId] IS NULL OR [Id] = @UserId)
                        AND [SysAdminUnitTypeValue] < 4
                    UNION ALL
                    SELECT
                        [FuncRoleId]
                    FROM
                        [dbo].[SysFuncRoleInOrgRole]
                    WHERE
                        [SysFuncRoleInOrgRole].[OrgRoleId] = @UserId
                    )
            UNION ALL
            SELECT
                [SelectPriorLevel].[Id],
                [SelectPriorLevel].[Name],
                [SelectPriorLevel].[ParentRoleId],
                [Level] + 1 level
            FROM
                [MainSelect] [SelectPriorLevel]
                INNER JOIN [HierarchicalSelect] hierSelect ON (hierSelect.[ParentRoleId] = [Sele
        ),
        [FuncRoleHierarchicalSelect] AS (
            SELECT
                [Id],
                [Name],
                [ParentRoleId],
                0 [Level]
            FROM
                [MainSelect] [StartLevel]
            WHERE EXISTS (
```

```
                SELECT NULL
                FROM [dbo].[SysFuncRoleInOrgRole] funcRoleInOrgRole
                    INNER JOIN [HierarchicalSelect] hierSelect ON funcRoleInOrgRole.[OrgRoleId]
                WHERE funcRoleInOrgRole.[FuncRoleId] = [StartLevel].[Id]
            )
            UNION ALL
            SELECT
                [PriorLevel].[Id],
                [PriorLevel].[Name],
                [PriorLevel].[ParentRoleId],
                [Level] + 1 level
            FROM
                [MainSelect] [PriorLevel]
                INNER JOIN [FuncRoleHierarchicalSelect] funcRoleHierSelect ON (funcRoleHierSelec
        ),
        [DependentUserSelect] AS (
            SELECT
                mainSelect.[Id] [Id],
                mainSelect.[Name] [Name],
                mainSelect.[ParentRoleId] [ParentRoleId],
                0 [Level]
            FROM
                [MainSelect] mainSelect
            INNER JOIN [SysUserInRole] userInRole
                ON mainSelect.[Id] = userInRole.[SysUserId]
            INNER JOIN [ChiefUnitsSelect] [AllUnits]
                ON [AllUnits].[Id] = userInRole.[SysRoleId]
            WHERE
                NOT EXISTS (
                        SELECT
                            [UserUnits].[Id]
                        FROM [ChiefUnitsSelect] [UserUnits]
                        INNER JOIN [SysUserInRole] [UserInRole]
                            ON [UserUnits].[Id] = [UserInRole].[SysRoleId]
                        INNER JOIN [SysAdminUnit] sau
                            ON sau.[Id] = [UserUnits].[Id]
                        WHERE sau.[SysAdminUnitTypeValue] = 2
                            AND [UserInRole].[SysUserId] = @UserId
                            AND [UserUnits].[Id] = [AllUnits].[Id])
        )
        INSERT INTO [#AdminUnitList] ([Id], [Name], [ParentRoleId], [Granted], [Level])
        SELECT DISTINCT
            [Id],
            [Name],
            [ParentRoleId],
            @Granted,
            @@NESTLEVEL
        FROM
            (
```

```
                SELECT
                    [Id],
                    [Name],
                    [ParentRoleId]
                FROM
                    [HierarchicalSelect]
                UNION ALL
                SELECT
                    [Id],
                    [Name],
                    [ParentRoleId]
                FROM
                    [dbo].[SysAdminUnit]
                WHERE
                    [Id] = @UserId
            UNION ALL
                SELECT
                    [Id],
                    [Name],
                    [ParentRoleId]
                FROM
                    [FuncRoleHierarchicalSelect]
            UNION ALL
                SELECT
                    [Id],
                    [Name],
                    [ParentRoleId]
                FROM
            [DependentUserSelect]
        ) [AdminUnitList];
    END ELSE
    BEGIN
        DECLARE @alreadyGotRolesForThisUser bit = 0;

        IF (@IsAdminUnitListTempExists = 1)
        BEGIN
            SET @alreadyGotRolesForThisUser = (SELECT CAST( CASE WHEN EXISTS(SELECT 1 FROM [#Adm
                    WHERE [UserId] = @UserId
                    )
                    THEN 1
                    ELSE 0
                END
            AS BIT));
        END;

        IF (@alreadyGotRolesForThisUser = 1)
        BEGIN
            INSERT INTO [#AdminUnitList] ([Id], [Name], [ParentRoleId], [Granted], [Level])
```

```
                SELECT DISTINCT
                    [Id],
                    [Name],
                    [ParentRoleId],
                    @Granted,
                    @@NESTLEVEL
                FROM [#AdminUnitListTemp] WHERE UserId = @UserId;
        END ELSE
        BEGIN
            WITH
            [MainSelect] AS (
                SELECT
                    [Id] [Id],
                    [Name] [Name],
                    [ParentRoleId] [ParentRoleId]
                FROM
                    [dbo].[SysAdminUnit]
                WHERE
                    ([SysAdminUnitTypeValue] <= 4 OR [SysAdminUnitTypeValue] = 6)
                AND [ConnectionType] = @ConnectionType
                UNION ALL
                SELECT
                    [Id] [Id],
                    [Name] [Name],
                    [ParentRoleId] [ParentRoleId]
                FROM
                    [dbo].[SysAdminUnit]
                WHERE
                    [Id] = @UserId),
            [ChiefUnitsSelect] AS (
                (
                    SELECT
                        [Chief].[ParentRoleId] [Id]
                    FROM
                        [dbo].[SysUserInRole] sysUserInRole
                        INNER JOIN [dbo].[SysAdminUnit] sau ON (sau.[Id] = sysUserInRole.[SysUse
                        INNER JOIN [dbo].[SysAdminUnit] [Chief] ON ([Chief].[Id] = sysUserInRole
                    WHERE
                        sau.[Id] = @UserId AND NOT (sysUserInRole.[SysRoleId] IS NULL) AND [Chie
                    UNION ALL
                    SELECT
                        [Chief].[ParentRoleId] [Id]
                    FROM
                        [dbo].[SysAdminUnit] [Chief]
                    WHERE
                        [Chief].[Id] = @UserId AND [Chief].[SysAdminUnitTypeValue] = 2
                )
                UNION ALL
                SELECT
```

```
                sau.[Id]
            FROM
                [ChiefUnitsSelect] ChiefUnitsSelect
                INNER JOIN [dbo].[SysAdminUnit] sau ON (sau.[ParentRoleId] = [ChiefUnitsSele
            WHERE
                sau.[SysAdminUnitTypeValue] < 4
        ),
        [HierarchicalSelect] AS (
            SELECT
                [Id],
                [Name],
                [ParentRoleId],
                0 [Level]
            FROM
                [MainSelect] [SelectStartLevel]
            WHERE EXISTS (
                SELECT NULL
                FROM (
                    SELECT [SysUserInRole].[SysRoleId] AS RoleId
                    FROM [dbo].[SysUserInRole]
                        INNER JOIN [dbo].[SysAdminUnit] ON ([SysAdminUnit].[Id] = [SysUserIr
                    WHERE [SysAdminUnit].[Id] = @UserId

                    UNION ALL

                    SELECT [Id] AS RoleId
                    FROM [ChiefUnitsSelect]

                    UNION ALL

                    SELECT [Id] AS RoleId
                    FROM [dbo].[SysAdminUnit]
                    WHERE ([ParentRoleId] IS NULL OR [Id] = @UserId)
                        AND [SysAdminUnitTypeValue] < 4

                    UNION ALL

                    SELECT [FuncRoleId] AS RoleId
                    FROM [dbo].[SysFuncRoleInOrgRole]
                    WHERE [SysFuncRoleInOrgRole].[OrgRoleId] = @UserId
                ) AS Roles
                WHERE Roles.RoleId = [SelectStartLevel].[Id]

            )
            UNION ALL
            SELECT
                [SelectPriorLevel].[Id],
                [SelectPriorLevel].[Name],
```

```
            [SelectPriorLevel].[ParentRoleId],
            [Level] + 1 level
        FROM
            [MainSelect] [SelectPriorLevel]
            INNER JOIN [HierarchicalSelect] hierSelect ON (hierSelect.[ParentRoleId] = [
    ),
    [FuncRoleHierarchicalSelect] AS (
        SELECT
            [Id],
            [Name],
            [ParentRoleId],
            0 [Level]
        FROM
            [MainSelect] [StartLevel]
        WHERE EXISTS (
            SELECT NULL
            FROM [dbo].[SysFuncRoleInOrgRole] funcRoleInOrgRole
                INNER JOIN [HierarchicalSelect] hierSelect ON funcRoleInOrgRole.[OrgRole
            WHERE funcRoleInOrgRole.[FuncRoleId] = [StartLevel].[Id]
        )
        UNION ALL
        SELECT
            [PriorLevel].[Id],
            [PriorLevel].[Name],
            [PriorLevel].[ParentRoleId],
            [Level] + 1
        FROM
            [MainSelect] [PriorLevel]
            INNER JOIN [FuncRoleHierarchicalSelect] funcRolesHierSelect ON (funcRolesHie
    ),
    [DependentUserSelect] AS (
        SELECT
            [MainSelect].[Id] [Id],
            [MainSelect].[Name] [Name],
            [MainSelect].[ParentRoleId] [ParentRoleId],
            0 [Level]
        FROM
            [MainSelect]
        INNER JOIN [SysUserInRole] sysUserInRole
            ON [MainSelect].[Id] = sysUserInRole.[SysUserId]
        INNER JOIN [ChiefUnitsSelect] [AllUnits]
            ON [AllUnits].[Id] = sysUserInRole.[SysRoleId]
        WHERE
            NOT EXISTS (
                    SELECT
                        [UserUnits].[Id]
                    FROM [ChiefUnitsSelect] [UserUnits]
                    INNER JOIN [SysUserInRole] [UserInRole]
                        ON [UserUnits].[Id] = [UserInRole].[SysRoleId]
```

```
                    INNER JOIN [SysAdminUnit] sau
                        ON sau.[Id] = [UserUnits].[Id]
                    WHERE sau.[SysAdminUnitTypeValue] = 2
                        AND [UserInRole].[SysUserId] = @UserId
                        AND [UserUnits].[Id] = [AllUnits].[Id])
            )
            INSERT INTO #AdminUnitListTemp ([UserId], [Id], [Name], [ParentRoleId], [Granted])
            SELECT DISTINCT
                @UserId,
                [Id],
                [Name],
                [ParentRoleId],
                @Granted
            FROM
                (
                    SELECT
                        [Id],
                        [Name],
                        [ParentRoleId]
                    FROM
                        [HierarchicalSelect]
                    UNION ALL
                    SELECT
                        [Id],
                        [Name],
                        [ParentRoleId]
                    FROM
                        [dbo].[SysAdminUnit]
                    WHERE
                        [Id] = @UserId
                UNION ALL
                    SELECT
                        [Id],
                        [Name],
                        [ParentRoleId]
                    FROM
                        [FuncRoleHierarchicalSelect]
                UNION ALL
                    SELECT
                        [Id],
                        [Name],
                        [ParentRoleId]
                    FROM
                [DependentUserSelect]
                ) [AdminUnitList];

            INSERT INTO [#AdminUnitList] ([Id], [Name], [ParentRoleId], [Granted], [Level])
            SELECT DISTINCT
```

```
                [Id],
                [Name],
                [ParentRoleId],
                @Granted,
                @@NESTLEVEL
            FROM [#AdminUnitListTemp] WHERE UserId = @UserId;
        END;
    END;

    DECLARE @DependentUserId uniqueidentifier;
    DECLARE @DependentUsersList CURSOR;
    SET @DependentUsersList = CURSOR FOR
        SELECT
            [#AdminUnitList].[Id]
        FROM
            [#AdminUnitList]
        INNER JOIN [SysAdminUnit] ON [#AdminUnitList].[Id] = [SysAdminUnit].[Id]
        WHERE
            [SysAdminUnit].[SysAdminUnitTypeValue] = 4 AND [#AdminUnitList].[Id] <> @UserId
            AND [#AdminUnitList].[Granted] <> 1 AND [#AdminUnitList].[Level] >= @@NESTLEVEL;
    OPEN @DependentUsersList;
    FETCH NEXT
    FROM @DependentUsersList INTO @DependentUserId;
    WHILE @@FETCH_STATUS = 0
    BEGIN
        EXEC [tsp_GetAdminUnitList] @UserId=@DependentUserId, @Granted=1;
    FETCH NEXT
    FROM @DependentUsersList INTO @DependentUserId;
    END;
    CLOSE @DependentUsersList;
    DEALLOCATE @DependentUsersList;

    DECLARE @GrantorSysAdminUnitId uniqueidentifier;
    DECLARE @getGrantorSysAdminUnitList CURSOR;
    SET @getGrantorSysAdminUnitList = CURSOR FOR
        SELECT
            [GrantorSysAdminUnitId]
        FROM
            [dbo].[SysAdminUnitGrantedRight]
        WHERE
            [GranteeSysAdminUnitId] = @UserId
            AND NOT EXISTS(SELECT * FROM [#AdminUnitList] WHERE [Id] = @UserId AND [Granted] = 1
    OPEN @getGrantorSysAdminUnitList;
    FETCH NEXT
    FROM @getGrantorSysAdminUnitList INTO @GrantorSysAdminUnitId;
    WHILE @@FETCH_STATUS = 0
    BEGIN
        EXEC [tsp_GetAdminUnitList] @UserId=@GrantorSysAdminUnitId, @Granted=1;
    FETCH NEXT
```

```
    FROM @getGrantorSysAdminUnitList INTO @GrantorSysAdminUnitId;
    END;
    CLOSE @getGrantorSysAdminUnitList;
    DEALLOCATE @getGrantorSysAdminUnitList;

    IF @@NESTLEVEL = @StartNestedLevel
    BEGIN
        WITH QQ ([Id], [Name], [ParentRoleId], SysAdminUnitTypeValue) as (
            SELECT DISTINCT adminUnitList.[Id],
                adminUnitList.[Name],
                adminUnitList.[ParentRoleId],
                sau.SysAdminUnitTypeValue
            FROM [#AdminUnitList] adminUnitList
            INNER JOIN SysAdminUnit sau on sau.Id = adminUnitList.[Id]
        )
        SELECT [Id], [Name], [ParentRoleId] FROM QQ
        ORDER BY SysAdminUnitTypeValue DESC;
    END;
END;
GO
```

Postgre SQL

```
-- Рекурсивная хранимая процедура, которая возвращает таблицу и в которой используется PERFORM:
-- PostgreSql
DROP FUNCTION IF EXISTS "tsp_GetAdminUnitList";
CREATE FUNCTION "tsp_GetAdminUnitList"(
    UserId UUID,
    IsGranted BOOLEAN = FALSE,
    NestLevel INT = 0
)
RETURNS TABLE (
    "Id" UUID,
    "Name" VARCHAR(250),
    "ParentRoleId" UUID
)
AS $$
DECLARE
    ConnectionType INT;
    IsAdminUnitListTempExists BOOLEAN = FALSE;
    DependentUserId UUID;
    DependentUsersList CURSOR FOR
        SELECT
            "AdminUnitList"."Id"
        FROM
            "AdminUnitList"
```

```
        INNER JOIN "SysAdminUnit" ON "AdminUnitList"."Id" = "SysAdminUnit"."Id"
        WHERE
            "SysAdminUnit"."SysAdminUnitTypeValue" = 4
            AND "AdminUnitList"."Id" <> UserId
            AND "AdminUnitList"."Granted" = FALSE
            AND "AdminUnitList"."Level" >= NestLevel;
    GrantorSysAdminUnitId UUID;
    GetGrantorSysAdminUnitList CURSOR FOR
        SELECT
            "GrantorSysAdminUnitId" AS "Id"
        FROM
            "SysAdminUnitGrantedRight"
        WHERE
            "GranteeSysAdminUnitId" = UserId
            AND NOT EXISTS (
                SELECT *
                FROM "AdminUnitList"
                WHERE "AdminUnitList"."Id" = UserId
                    AND "AdminUnitList"."Granted" = TRUE
                    AND "AdminUnitList"."Level" < NestLevel
            );
    ParentRoleId UUID = NULL;
BEGIN

    IF NestLevel = 0 THEN
        CREATE TEMPORARY TABLE IF NOT EXISTS "AdminUnitList" (
            "Id" UUID,
            "Name" VARCHAR(250),
            "ParentRoleId" UUID,
            "Granted" BOOLEAN,
            "Level" INT
        );
        TRUNCATE TABLE "AdminUnitList";
    END IF;

    SELECT "ConnectionType" INTO ConnectionType FROM "SysAdminUnit" WHERE "SysAdminUnit"."Id" =

    WITH RECURSIVE "MainSelect" AS (
        SELECT
            "SysAdminUnit"."Id" "Id",
            "SysAdminUnit"."Name" "Name",
            "SysAdminUnit"."ParentRoleId" "ParentRoleId"
        FROM
            "SysAdminUnit"
        WHERE
            ("SysAdminUnitTypeValue" <= 4 OR "SysAdminUnitTypeValue" = 6)
        AND "ConnectionType" = ConnectionType
        UNION ALL
        SELECT
```

```
            "SysAdminUnit"."Id" "Id",
            "SysAdminUnit"."Name" "Name",
            "SysAdminUnit"."ParentRoleId" "ParentRoleId"
        FROM
            "SysAdminUnit"
        WHERE
            "SysAdminUnit"."Id" = UserId),
        "ChiefUnitsSelect" AS (
            SELECT
                "chief"."ParentRoleId" "Id"
            FROM
                "SysUserInRole" AS "userInRole"
                INNER JOIN "SysAdminUnit" AS "sau" ON ("sau"."Id" = "userInRole"."SysUserId")
                INNER JOIN "SysAdminUnit" AS "chief" ON ("chief"."Id" = "userInRole"."SysRoleId"
            WHERE
                "sau"."Id" = UserId
                AND "userInRole"."SysRoleId" IS NOT NULL
                AND "chief"."SysAdminUnitTypeValue" = 2
            UNION ALL
            SELECT
                "chief"."ParentRoleId" "Id"
            FROM
                "SysAdminUnit" "chief"
            WHERE
                "chief"."Id" = UserId AND "chief"."SysAdminUnitTypeValue" = 2
            UNION ALL
            SELECT
                "sau"."Id"
            FROM
                "ChiefUnitsSelect"
                INNER JOIN "SysAdminUnit" "sau" ON ("sau"."ParentRoleId" = "ChiefUnitsSelect"."I
            WHERE
                "sau"."SysAdminUnitTypeValue" < 4
        ),
        "HierarchicalSelect" AS (
            SELECT
                "SelectStartLevel"."Id",
                "SelectStartLevel"."Name",
                "SelectStartLevel"."ParentRoleId",
                0 "Level"
            FROM
                "MainSelect" "SelectStartLevel"
            WHERE
                "SelectStartLevel"."Id" IN (
                    SELECT
                        "userInRole"."SysRoleId"
                    FROM
                        "SysUserInRole" AS "userInRole"
```

```
                    INNER JOIN "SysAdminUnit" AS "sau" ON ("sau"."Id" = "userInRole"."SysUse
                WHERE
                    "sau"."Id" = UserId
                UNION ALL
                SELECT "ChiefUnitsSelect"."Id"
                FROM "ChiefUnitsSelect"
                UNION ALL
                SELECT
                    "SysAdminUnit"."Id"
                FROM
                    "SysAdminUnit"
                WHERE
                    ("SysAdminUnit"."ParentRoleId" IS NULL OR "SysAdminUnit"."Id" = UserId)
                    AND "SysAdminUnitTypeValue" < 4
                UNION ALL
                SELECT
                    "FuncRoleId"
                FROM
                    "SysFuncRoleInOrgRole"
                WHERE
                    "SysFuncRoleInOrgRole"."OrgRoleId" = UserId
                )
            UNION ALL
            SELECT
                "SelectPriorLevel"."Id",
                "SelectPriorLevel"."Name",
                "SelectPriorLevel"."ParentRoleId",
                "Level" + 1 "level"
            FROM
                "MainSelect" "SelectPriorLevel"
                INNER JOIN "HierarchicalSelect" AS "hierSelect" ON ("hierSelect"."ParentRoleId"
        ),
        "FuncRoleHierarchicalSelect" AS (
            SELECT
                "StartLevel"."Id",
                "StartLevel"."Name",
                "StartLevel"."ParentRoleId",
                0 "Level"
            FROM
                "MainSelect" "StartLevel"
            WHERE EXISTS (
                SELECT NULL
                FROM "SysFuncRoleInOrgRole" AS "funcRoleInOrgRole"
                    INNER JOIN "HierarchicalSelect" AS "hierSelect" ON "funcRoleInOrgRole"."OrgR
                WHERE "funcRoleInOrgRole"."FuncRoleId" = "StartLevel"."Id"
            )
            UNION ALL
            SELECT
                "PriorLevel"."Id",
```

```
                "PriorLevel"."Name",
                "PriorLevel"."ParentRoleId",
                "Level" + 1 "level"
            FROM
                "MainSelect" "PriorLevel"
                INNER JOIN "FuncRoleHierarchicalSelect" AS "funcRoleHierSelect" ON ("funcRoleHie
        ),
        "DependentUserSelect" AS (
            SELECT
                "mainSelect"."Id" "Id",
                "mainSelect"."Name" "Name",
                "mainSelect"."ParentRoleId" "ParentRoleId",
                0 "Level"
            FROM
                "MainSelect" AS "mainSelect"
            INNER JOIN "SysUserInRole" AS "userInRole"
                ON "mainSelect"."Id" = "userInRole"."SysUserId"
            INNER JOIN "ChiefUnitsSelect" AS "AllUnits"
                ON "AllUnits"."Id" = "userInRole"."SysRoleId"
            WHERE
                NOT EXISTS (
                        SELECT
                            "UserUnits"."Id"
                        FROM "ChiefUnitsSelect" AS "UserUnits"
                        INNER JOIN "SysUserInRole" AS "UserInRole"
                            ON "UserUnits"."Id" = "UserInRole"."SysRoleId"
                        INNER JOIN "SysAdminUnit" AS "sau"
                            ON "sau"."Id" = "UserUnits"."Id"
                        WHERE "sau"."SysAdminUnitTypeValue" = 2
                            AND "UserInRole"."SysUserId" = UserId
                            AND "UserUnits"."Id" = "AllUnits"."Id")
        )
        INSERT INTO "AdminUnitList" ("Id", "Name", "ParentRoleId", "Granted", "Level")
        SELECT DISTINCT
            "AdminUnitList"."Id",
            "AdminUnitList"."Name",
            "AdminUnitList"."ParentRoleId",
            IsGranted,
            NestLevel
        FROM (
            SELECT
                "HierarchicalSelect"."Id",
                "HierarchicalSelect"."Name",
                "HierarchicalSelect"."ParentRoleId"
            FROM "HierarchicalSelect"
            UNION ALL
            SELECT
                "SysAdminUnit"."Id",
```

```
                "SysAdminUnit"."Name",
                "SysAdminUnit"."ParentRoleId"
            FROM "SysAdminUnit"
            WHERE
                "SysAdminUnit"."Id" = UserId
            UNION ALL
            SELECT
                "FuncRoleHierarchicalSelect"."Id",
                "FuncRoleHierarchicalSelect"."Name",
                "FuncRoleHierarchicalSelect"."ParentRoleId"
            FROM "FuncRoleHierarchicalSelect"
            UNION ALL
            SELECT
                "DependentUserSelect"."Id",
                "DependentUserSelect"."Name",
                "DependentUserSelect"."ParentRoleId"
            FROM "DependentUserSelect"
        ) AS "AdminUnitList";

    DependentUsersList := 'DependentUsersList' || NestLevel ;
    FOR DependentUser IN DependentUsersList LOOP
        EXIT WHEN DependentUser = NULL;
        DependentUserId = DependentUser."Id";
        PERFORM "tsp_GetAdminUnitList"(DependentUserId, 1, NestLevel + 1);
    END LOOP;

    GetGrantorSysAdminUnitList := 'GetGrantorSysAdminUnitList' || NestLevel ;
    FOR GrantorSysAdminUnit IN GetGrantorSysAdminUnitList LOOP
        EXIT WHEN GrantorSysAdminUnit = NULL;
        GrantorSysAdminUnitId = GrantorSysAdminUnit."Id";
        PERFORM "tsp_GetAdminUnitList"(GrantorSysAdminUnitId, 1, NestLevel + 1);
    END LOOP;

    IF NestLevel = 0 THEN
        RETURN QUERY
        SELECT "QQ"."Id",
                "QQ"."Name",
                "QQ"."ParentRoleId"
        FROM (
            SELECT DISTINCT
                "AdminUnitList"."Id",
                "AdminUnitList"."Name",
                "AdminUnitList"."ParentRoleId",
                "sau"."SysAdminUnitTypeValue"
            FROM "AdminUnitList"
            INNER JOIN "SysAdminUnit" AS "sau" ON "sau"."Id" = "AdminUnitList"."Id") AS "QQ"
        ORDER BY "QQ"."SysAdminUnitTypeValue" DESC;
    END IF;
```

```
END;
$$ LANGUAGE plpgsql;
```

Пример 5 (хранимые процедуры)

Пример. Пример хранимой процедуры, в которой используется обработка исключений и выполнение кастомного скрипта.

MS SQL

```
-- Хранимая процедура, в которой используется обработка исключений и выполнение кастомного скрип
-- MSSQL

IF  EXISTS (SELECT * FROM sys.objects WHERE object_id = OBJECT_ID(N'[dbo].[tsp_CanConvertData]')
DROP PROCEDURE [dbo].[tsp_CanConvertData]
GO
CREATE PROCEDURE [dbo].[tsp_CanConvertData]
    @EntitySchemaName SYSNAME,
    @SourceColumnName SYSNAME,
    @NewColumnDataType SYSNAME,
    @Result BIT OUT
AS
BEGIN
    SET NOCOUNT ON

    SET @Result = 0

    DECLARE @sql NVARCHAR(MAX)
    DECLARE @unicodeCharLength INT = 2
    DECLARE @dataTypeName SYSNAME
    DECLARE @dataTypeSize INT
    DECLARE @dataTypePrecision INT

    SELECT
        @dataTypeName = UPPER(DATA_TYPE),
        @dataTypeSize =
        CASE
            WHEN CHARACTER_MAXIMUM_LENGTH IS NULL THEN NUMERIC_PRECISION
            ELSE CHARACTER_MAXIMUM_LENGTH
        END,
        @dataTypePrecision = ISNULL(NUMERIC_SCALE, 0)
    FROM INFORMATION_SCHEMA.COLUMNS
    WHERE TABLE_NAME = @EntitySchemaName
```

```
    AND COLUMN_NAME = @SourceColumnName

    IF (@dataTypeName IS NULL)
    BEGIN
        RETURN
    END

    DECLARE @newDataTypeName SYSNAME
    DECLARE @newDataTypeSize INT
    DECLARE @newDataTypePrecision INT
    DEClARE @i INT
    DECLARE @newDataTypeSizeDefinition NVARCHAR(MAX)

    SET @i = CHARINDEX('(', @NewColumnDataType)
    IF (@i = 0)
    BEGIN
        SET @newDataTypeName = @NewColumnDataType
        SET @newDataTypeSize = 0
        SET @newDataTypePrecision = 0
    END ELSE
    BEGIN
        SET @newDataTypeName = UPPER(LTRIM(RTRIM(SUBSTRING(@NewColumnDataType, 1, @i - 1))))
        SET @newDataTypeSizeDefinition = LTRIM(RTRIM(SUBSTRING(@NewColumnDataType, @i + 1,
            LEN(@NewColumnDataType))))
        SET @i = CHARINDEX(')', @newDataTypeSizeDefinition)
        IF (@i > 0)
        BEGIN
            SET @newDataTypeSizeDefinition = LTRIM(RTRIM(SUBSTRING(@newDataTypeSizeDefinition, 1
        END
        SET @i = CHARINDEX(',', @newDataTypeSizeDefinition)
        IF (@i > 0)
        BEGIN
            SET @newDataTypeSize = CAST(LTRIM(RTRIM(SUBSTRING(@newDataTypeSizeDefinition, 1, @i
            SET @newDataTypePrecision = CAST(LTRIM(RTRIM(SUBSTRING(@newDataTypeSizeDefinition, @
                LEN(@newDataTypeSizeDefinition)))) AS INT)
        END ELSE
        BEGIN
            SET @newDataTypePrecision = 0
            IF (UPPER(@newDataTypeSizeDefinition) = 'MAX')
            BEGIN
                SET @newDataTypeSize = -1
            END ELSE
            BEGIN
                SET @newDataTypeSize = CAST(@newDataTypeSizeDefinition AS INT)
            END
        END
    END

    DECLARE @ImplicitDataConvertTable TABLE (
```

```
        SourceDataType SYSNAME,
        DestinationDataType SYSNAME
    )
    INSERT INTO @ImplicitDataConvertTable
    SELECT 'INT', 'INT'
    UNION ALL
    SELECT 'INT', 'BIT'
    UNION ALL
    SELECT 'INT', 'DECIMAL'
    UNION ALL
    SELECT 'INT', 'VARCHAR'
    UNION ALL
    SELECT 'INT', 'NVARCHAR'
    UNION ALL
    SELECT 'INT', 'VARBINARY'
    UNION ALL
    SELECT 'BIT', 'BIT'
    UNION ALL
    SELECT 'BIT', 'INT'
    UNION ALL
    SELECT 'BIT', 'DECIMAL'
    UNION ALL
    SELECT 'BIT', 'VARCHAR'
    UNION ALL
    SELECT 'BIT', 'NVARCHAR'
    UNION ALL
    SELECT 'BIT', 'VARBINARY'
    UNION ALL
    SELECT 'DECIMAL', 'BIT'
    UNION ALL
    SELECT 'UNIQUEIDENTIFIER', 'UNIQUEIDENTIFIER'
    UNION ALL
    SELECT 'UNIQUEIDENTIFIER', 'VARBINARY'
    UNION ALL
    SELECT 'VARCHAR', 'INT'
    UNION ALL
    SELECT 'VARCHAR', 'BIT'
    UNION ALL
    SELECT 'VARCHAR', 'UNIQUEIDENTIFIER'
    UNION ALL
    SELECT 'DATETIME2', 'DATETIME2'
    UNION ALL
    SELECT 'DATETIME2', 'DATE'
    UNION ALL
    SELECT 'DATETIME2', 'TIME'
    UNION ALL
    SELECT 'DATETIME2', 'VARCHAR'
    UNION ALL
```

```
    SELECT 'DATE', 'DATE'
    UNION ALL
    SELECT 'DATE', 'DATETIME2'
    UNION ALL
    SELECT 'DATE', 'VARCHAR'
    UNION ALL
    SELECT 'DATE', 'NVARCHAR'
    UNION ALL
    SELECT 'TIME', 'TIME'
    UNION ALL
    SELECT 'TIME', 'DATETIME2'
    UNION ALL
    SELECT 'TIME', 'VARCHAR'
    UNION ALL
    SELECT 'TIME', 'NVARCHAR'
    UNION ALL
    SELECT 'VARBINARY', 'INT'
    UNION ALL
    SELECT 'VARBINARY', 'BIT'
    UNION ALL
    SELECT 'VARBINARY', 'UNIQUEIDENTIFIER'

    IF EXISTS(SELECT * FROM @ImplicitDataConvertTable
        WHERE SourceDataType = @dataTypeName AND DestinationDataType = @newDataTypeName)
    BEGIN
        SET @Result = 1
        RETURN
    END

    DECLARE @ImplicitDataOverflowConvertTable TABLE (
        SourceDataType SYSNAME,
        DestinationDataType SYSNAME
    )
    INSERT INTO @ImplicitDataOverflowConvertTable
    SELECT 'DECIMAL', 'INT'
    UNION ALL
    SELECT 'DECIMAL', 'DECIMAL'
    UNION ALL
    SELECT 'DECIMAL', 'VARCHAR'
    UNION ALL
    SELECT 'DECIMAL', 'NVARCHAR'
    UNION ALL
    SELECT 'DECIMAL', 'VARBINARY'
    UNION ALL
    SELECT 'UNIQUEIDENTIFIER', 'VARCHAR'
    UNION ALL
    SELECT 'UNIQUEIDENTIFIER', 'NVARCHAR'
    UNION ALL
    SELECT 'VARCHAR', 'INT'
```

```
    UNION ALL
    SELECT 'VARCHAR', 'BIT'
    UNION ALL
    SELECT 'VARCHAR', 'DECIMAL'
    UNION ALL
    SELECT 'VARCHAR', 'VARCHAR'
    UNION ALL
    SELECT 'VARCHAR', 'NVARCHAR'
    UNION ALL
    SELECT 'NVARCHAR', 'INT'
    UNION ALL
    SELECT 'NVARCHAR', 'BIT'
    UNION ALL
    SELECT 'NVARCHAR', 'DECIMAL'
    UNION ALL
    SELECT 'NVARCHAR', 'VARCHAR'
    UNION ALL
    SELECT 'NVARCHAR', 'NVARCHAR'
    UNION ALL
    SELECT 'VARBINARY', 'VARCHAR'
    UNION ALL
    SELECT 'VARBINARY', 'NVARCHAR'
    UNION ALL
    SELECT 'VARBINARY', 'VARBINARY'

    IF EXISTS(SELECT * FROM @ImplicitDataOverflowConvertTable
        WHERE SourceDataType = @dataTypeName AND DestinationDataType = @newDataTypeName)
    BEGIN
        SET @sql = N'IF EXISTS(SELECT * FROM [' + @EntitySchemaName + ']) SET @Result = 0 ELSE S
        EXEC sp_executesql @sql, N'@Result BIT OUT', @Result = @Result OUT

        IF (@Result = 1)
        BEGIN
            RETURN
        END
        BEGIN TRY
            IF (@dataTypeName = 'DECIMAL' AND @newDataTypeName = 'INT') OR
                (@dataTypeName = 'DECIMAL' AND @newDataTypeName = 'VARCHAR') OR
                (@dataTypeName = 'DECIMAL' AND @newDataTypeName = 'NVARCHAR') OR
                (@dataTypeName = 'DECIMAL' AND @newDataTypeName = 'VARBINARY') OR
                (@dataTypeName = 'DECIMAL' AND @newDataTypeName = 'DECIMAL')
            BEGIN
                DECLARE @cnt INT
                DECLARE @ConvertDescription NVARCHAR(MAX)
                SET @ConvertDescription = 'CONVERT(' + @NewColumnDataType +', [' + @SourceColumr
                SET @sql = N'IF EXISTS(SELECT * FROM [' + @EntitySchemaName + '] WHERE ' +
                @ConvertDescription + ' = ' + @ConvertDescription + ') SET @cnt = 1 ELSE SET @cr
                EXEC sp_executesql @sql, N'@cnt INT OUT', @cnt = @cnt OUT
```

```
                SET @Result = 1
            END ELSE
            BEGIN
                DECLARE @dl INT
                SET @sql = N'SELECT @dl = MAX(DATALENGTH([' + @SourceColumnName + ])) ' +
                'FROM [' + @EntitySchemaName + ']'
                EXEC sp_executesql @sql, N'@dl INT OUT', @dl = @dl OUT
                IF (@newDataTypeName IN ('VARCHAR', 'NVARCHAR', 'VARBINARY') AND @newDataTypeSiz
                BEGIN
                    SET @Result = 1
                END ELSE
                IF (@dl <= @newDataTypeSize OR (
                    @newDataTypeName IN ('NVARCHAR', 'NCHAR') AND (@dl / @unicodeCharLength) <=
                BEGIN
                    SET @Result = 1
                END ELSE
                BEGIN
                    SET @Result = 0
                END
            END
        END TRY
        BEGIN CATCH
            SET @Result = 0
        END CATCH
    END ELSE
    BEGIN
        SET @Result = 0
    END
END
GO
```

Postgre SQL

```
-- Хранимая процедура, в которой используется обработка исключений и выполнение кастомного скрип
-- PostgreSql
DROP FUNCTION IF EXISTS public."tsp_CanConvertData" CASCADE;
CREATE FUNCTION public."tsp_CanConvertData"(
    EntitySchemaName NAME,
    SourceColumnName NAME,
    NewColumnDataType NAME,
    CanConvert OUT BOOLEAN)
AS $BODY$
DECLARE
    dataTypeName NAME;
    newDataTypeName NAME;
    newDataTypeSize INTEGER;
    countRow INTEGER;
```

```
    dataLength INTEGER;
    convertDescription TEXT;
    unicodeCharLength INTEGER = 2;
    sqlQuery TEXT;
    castQuery TEXT;
BEGIN
    CanConvert = FALSE;
    dataTypeName = (
        SELECT UPPER(data_type) FROM information_schema.columns
        WHERE table_name = EntitySchemaName AND column_name = SourceColumnName);
    IF dataTypeName IS NULL THEN
        RETURN;
    END IF;

    SELECT "fn_ParseDataType".DataTypeName, "fn_ParseDataType".DataTypeSize
    FROM public."fn_ParseDataType"(NewColumnDataType)
    INTO newDataTypeName, newDataTypeSize;

    DROP TABLE IF EXISTS "NotConvertTable";
    CREATE TEMP TABLE "NotConvertTable" (
        SourceDataType NAME,
        DestinationDataType NAME
    );
    INSERT INTO "NotConvertTable" VALUES
        ('INTEGER', 'UUID'),
        ('INTEGER', 'TIMESTAMP WITHOUT TIME ZONE'),
        ('INTEGER', 'DATE'),
        ('INTEGER', 'TIME WITHOUT TIME ZONE'),
        ('NUMERIC', 'UUID'),
        ('NUMERIC', 'TIMESTAMP WITHOUT TIME ZONE'),
        ('NUMERIC', 'DATE'),
        ('NUMERIC', 'TIME WITHOUT TIME ZONE'),
        ('BOOLEAN', 'UUID'),
        ('BOOLEAN', 'TIMESTAMP WITHOUT TIME ZONE'),
        ('BOOLEAN', 'DATE'),
        ('BOOLEAN', 'TIME WITHOUT TIME ZONE'),
        ('UUID', 'INTEGER'),
        ('UUID', 'NUMERIC'),
        ('UUID', 'BOOLEAN'),
        ('UUID', 'TIMESTAMP WITHOUT TIME ZONE'),
        ('UUID', 'DATE'),
        ('UUID', 'TIME WITHOUT TIME ZONE'),
        ('TIMESTAMP WITHOUT TIME ZONE', 'INTEGER'),
        ('TIMESTAMP WITHOUT TIME ZONE', 'NUMERIC'),
        ('TIMESTAMP WITHOUT TIME ZONE', 'BOOLEAN'),
        ('TIMESTAMP WITHOUT TIME ZONE', 'UUID'),
        ('DATE', 'INTEGER'),
        ('DATE', 'NUMERIC'),
```

```
        ('DATE', 'BOOLEAN'),
        ('DATE', 'UUID'),
        ('DATE', 'TIME WITHOUT TIME ZONE'),
        ('TIME WITHOUT TIME ZONE', 'INTEGER'),
        ('TIME WITHOUT TIME ZONE', 'NUMERIC'),
        ('TIME WITHOUT TIME ZONE', 'BOOLEAN'),
        ('TIME WITHOUT TIME ZONE', 'UUID'),
        ('TIME WITHOUT TIME ZONE', 'DATE');
    IF EXISTS(SELECT SourceDataType, DestinationDataType FROM "NotConvertTable"
        WHERE SourceDataType = dataTypeName AND DestinationDataType = newDataTypeName) THEN
        RETURN;
    END IF;

    DROP TABLE IF EXISTS ImplicitDataConvertTable;
    CREATE TEMP TABLE ImplicitDataConvertTable (
        SourceDataType NAME,
        DestinationDataType NAME
    );
    INSERT INTO ImplicitDataConvertTable VALUES
        ('INTEGER', 'INTEGER'),
        ('INTEGER', 'NUMERIC'),
        ('INTEGER', 'BOOLEAN'),
        ('INTEGER', 'CHARACTER VARYING'),
        ('INTEGER', 'TEXT'),
        ('NUMERIC', 'CHARACTER VARYING'),
        ('NUMERIC', 'TEXT'),
        ('BOOLEAN', 'INTEGER'),
        ('BOOLEAN', 'BOOLEAN'),
        ('BOOLEAN', 'CHARACTER VARYING'),
        ('BOOLEAN', 'TEXT'),
        ('CHARACTER VARYING', 'TEXT'),
        ('CHARACTER VARYING', 'BYTEA'),
        ('TEXT', 'TEXT'),
        ('TEXT', 'BYTEA'),
        ('BYTEA', 'BYTEA'),
        ('UUID', 'CHARACTER VARYING'),
        ('UUID', 'TEXT'),
        ('UUID', 'UUID'),
        ('TIMESTAMP WITHOUT TIME ZONE', 'CHARACTER VARYING'),
        ('TIMESTAMP WITHOUT TIME ZONE', 'TEXT'),
        ('TIMESTAMP WITHOUT TIME ZONE', 'TIMESTAMP WITHOUT TIME ZONE'),
        ('DATE', 'CHARACTER VARYING'),
        ('DATE', 'TEXT'),
        ('DATE', 'TIMESTAMP WITHOUT TIME ZONE'),
        ('DATE', 'DATE'),
        ('TIME WITHOUT TIME ZONE', 'CHARACTER VARYING'),
        ('TIME WITHOUT TIME ZONE', 'TEXT'),
        ('TIME WITHOUT TIME ZONE', 'TIMESTAMP WITHOUT TIME ZONE'),
        ('TIME WITHOUT TIME ZONE', 'TIME WITHOUT TIME ZONE'),
```

```
        ('TIMESTAMP WITHOUT TIME ZONE', 'DATE'),
        ('TIMESTAMP WITHOUT TIME ZONE', 'TIME WITHOUT TIME ZONE'),
        ('INTEGER', 'BYTEA'),
        ('NUMERIC', 'BOOLEAN'),
        ('NUMERIC', 'BYTEA'),
        ('BOOLEAN', 'NUMERIC'),
        ('BOOLEAN', 'BYTEA'),
        ('UUID', 'BYTEA'),
        ('TIMESTAMP WITHOUT TIME ZONE', 'BYTEA'),
        ('DATE', 'BYTEA'),
        ('TIME WITHOUT TIME ZONE', 'BYTEA'),
        ('NUMERIC', 'INTEGER'),
        ('NUMERIC', 'NUMERIC');
    IF EXISTS(SELECT SourceDataType, DestinationDataType FROM ImplicitDataConvertTable
        WHERE SourceDataType = dataTypeName AND DestinationDataType = newDataTypeName) THEN
        CanConvert = TRUE;
        RETURN;
    END IF;

    EXECUTE FORMAT('SELECT count(*) FROM %1$I', EntitySchemaName) INTO countRow;
    CanConvert = (countRow = 0);
    IF CanConvert THEN
        RETURN;
    END IF;

    DROP TABLE IF EXISTS "ExplicitDataConvertTable";
    CREATE TEMP TABLE "ExplicitDataConvertTable" (
        SourceDataType NAME,
        DestinationDataType NAME
    );
    INSERT INTO "ExplicitDataConvertTable" VALUES
        ('CHARACTER VARYING', 'INTEGER'),
        ('CHARACTER VARYING', 'NUMERIC'),
        ('CHARACTER VARYING', 'BOOLEAN'),
        ('CHARACTER VARYING', 'UUID'),
        ('CHARACTER VARYING', 'TIMESTAMP WITHOUT TIME ZONE'),
        ('CHARACTER VARYING', 'DATE'),
        ('CHARACTER VARYING', 'TIME WITHOUT TIME ZONE'),
        ('TEXT', 'INTEGER'),
        ('TEXT', 'NUMERIC'),
        ('TEXT', 'BOOLEAN'),
        ('TEXT', 'UUID'),
        ('TEXT', 'TIMESTAMP WITHOUT TIME ZONE'),
        ('TEXT', 'DATE'),
        ('TEXT', 'TIME WITHOUT TIME ZONE'),
        ('BYTEA', 'INTEGER'),
        ('BYTEA', 'NUMERIC'),
        ('BYTEA', 'BOOLEAN'),
```

```
        ('BYTEA', 'UUID'),
        ('BYTEA', 'TIMESTAMP WITHOUT TIME ZONE'),
        ('BYTEA', 'DATE'),
        ('BYTEA', 'TEXT'),
        ('BYTEA', 'TIME WITHOUT TIME ZONE'),
        ('NUMERIC', 'BOOLEAN');
    IF EXISTS(SELECT SourceDataType, DestinationDataType FROM "ExplicitDataConvertTable"
        WHERE SourceDataType = dataTypeName AND DestinationDataType = newDataTypeName) THEN
        castQuery = FORMAT('CAST(%1$I%3$s AS %2$s)', SourceColumnName, NewColumnDataType,
            CASE
                WHEN dataTypeName = 'BYTEA' THEN '::TEXT'
                WHEN dataTypeName = 'NUMERIC' THEN '::INTEGER'
                ELSE ''
            END);
        sqlQuery = FORMAT('SELECT COUNT(*) FROM %1$I WHERE %2$s = %2$s',
            EntitySchemaName, castQuery);
        BEGIN
            EXECUTE sqlQuery;
            CanConvert = TRUE;
        EXCEPTION WHEN OTHERS THEN
            CanConvert = FALSE;
        END;
        RETURN;
    END IF;

    DROP TABLE IF EXISTS "ImplicitDataOverflowConvertTable";
    CREATE TEMP TABLE "ImplicitDataOverflowConvertTable" (
        SourceDataType NAME,
        DestinationDataType NAME
    );
    INSERT INTO "ImplicitDataOverflowConvertTable" VALUES
        ('CHARACTER VARYING', 'CHARACTER VARYING'),
        ('TEXT', 'CHARACTER VARYING'),
        ('BYTEA', 'CHARACTER VARYING');
    IF EXISTS(SELECT SourceDataType, DestinationDataType FROM "ImplicitDataOverflowConvertTable"
        WHERE SourceDataType = dataTypeName AND DestinationDataType = newDataTypeName) THEN
        EXECUTE FORMAT('SELECT count(*) FROM %1$I', EntitySchemaName) INTO countRow;
        CanConvert = (countRow = 0);
        IF CanConvert THEN
            RETURN;
        END IF;
        BEGIN
            EXECUTE FORMAT('SELECT MAX(PG_COLUMN_SIZE(%1$I)) FROM %2$I', SourceColumnName, Entit
            INTO dataLength;
            IF (dataLength <= newDataTypeSize) THEN
                CanConvert = TRUE;
            ELSE
                CanConvert = FALSE;
            END IF;
```

```
        EXCEPTION WHEN OTHERS THEN
            CanConvert = FALSE;
        END;
    END IF;
END;
$BODY$
LANGUAGE 'plpgsql';
```

Пример 6 (функции)

Пример. Пример функции.

MS SQL

```
-- Функция
-- MSSQL
IF  EXISTS (SELECT * FROM sys.objects
        WHERE object_id = OBJECT_ID(N'[dbo].[fn_IsGuid]') AND type = N'FN')
DROP FUNCTION [dbo].[fn_IsGuid]
GO

CREATE FUNCTION [dbo].[fn_IsGuid] (
    @ValidateValue NVARCHAR(MAX))
RETURNS BIT
AS
BEGIN
    DECLARE @hasLeftBraces BIT
    IF @ValidateValue LIKE '{%'
    BEGIN
        SET @ValidateValue = SUBSTRING(@ValidateValue, 2, LEN(@ValidateValue) - 1)
        SET @hasLeftBraces = 1
    END ELSE
    BEGIN
        SET @hasLeftBraces = 0
    END
    DECLARE @hasRightBraces BIT
    IF @ValidateValue LIKE '%}'
    BEGIN
        SET @ValidateValue = SUBSTRING(@ValidateValue, 1, LEN(@ValidateValue) - 1)
        SET @hasRightBraces = 1
    END ELSE
    BEGIN
        SET @hasRightBraces = 0
```

```
    END
    DECLARE @Result BIT
    IF @ValidateValue LIKE '[0-9a-fA-F][0-9a-fA-F][0-9a-fA-F][0-9a-fA-F][0-9a-fA-F][0-9a-fA-F][0
    BEGIN
        SET @Result = 1
    END ELSE
    BEGIN
        SET @Result = 0
    END
    IF @hasLeftBraces = @hasRightBraces
    BEGIN
        RETURN @Result
    END ELSE
    BEGIN
        SET @Result = 0
    END
    RETURN @Result
END
GO
```

Postgre SQL

```
-- Функция
-- PostgreSql
DROP FUNCTION IF EXISTS "public"."fn_IsGuid";
CREATE OR REPLACE FUNCTION public."fn_IsGuid"(ValidateValue IN VARCHAR) RETURNS BOOLEAN AS $$
    BEGIN
        IF (regexp_matches(ValidateValue, '^\{?[0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9
            RETURN TRUE;
        ELSE
            RETURN FALSE;
        END IF;
    END;
$$ LANGUAGE plpgsql;
```

Класс Entity C#

Сложный

Пространство имен `Terrasoft.Core.Entities`.

Класс `Terrasoft.Core.Entities.Entity` предназначен для доступа к объекту, который представляет собой запись в таблице базы данных.

На заметку. Полный перечень методов и свойств класса `Entity`, 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`.

`Entity(UserConnection userConnection, Guid schemaUId)`

Создает новый экземпляр класса `Entity` для заданного пользовательского подключения `UserConnection` и схемы заданной идентификатором `schemaUId`.

`Entity(Entity source)`

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

Свойства

`ChangeType EntityChangeType`

Тип изменения состояния объекта (добавлен, изменен, удален, без изменений).

`EntitySchemaManager EntitySchemaManager`

Экземпляр менеджера схемы объекта.

`EntitySchemaManagerName string`

Имя менеджера схемы объекта.

`HasColumnValues bool`

Определяет, имеет ли объект хотя бы одну колонку.

`HierarchyColumnValue Guid`

Значение колонки связи с родительской записью для иерархических объектов.

`InstanceUId` `Guid`

Идентификатор экземпляра объекта.

`IsDeletedFromDB` `bool`

Определяет, удален ли объект из базы данных.

`IsInColumnValueChanged` `bool`

Определяет, выполняется ли обработка события `ColumnValueChanged`.

`IsInColumnValueChanging` `bool`

Определяет, выполняется ли обработка события `ColumnValueChanging`.

`IsInDefColumnValuesSet` `bool`

Определяет, выполняется ли обработка события `DefColumnValuesSet`.

`IsInDeleted` `bool`

Определяет, выполняется ли обработка события `Deleted`.

`IsInDeleting` `bool`

Определяет, выполняется ли обработка события `Deleting`.

`IsInInserted` `bool`

Определяет, выполняется ли обработка события `Inserted`.

`IsInInserting` `bool`

Определяет, выполняется ли обработка события `Inserting`.

`IsInLoaded` `bool`

Определяет, выполняется ли обработка события `Loaded`.

`IsInLoading` `bool`

Определяет, выполняется ли обработка события `Loading`.

`IsInSaved` `bool`

Определяет, выполняется ли обработка события `Saved`.

`IsInSaveError` `bool`

Определяет, выполняется ли обработка события `SaveError`.

`IsInSaving` `bool`

Определяет, выполняется ли обработка события `Saving`.

`IsInUpdated` `bool`

Определяет, выполняется ли обработка события `Updated`.

`IsInUpdating` `bool`

Определяет, выполняется ли обработка события `Updating`.

`IsInValidating` `bool`

Определяет, выполняется ли обработка события `Validating`.

`IsSchemaInitialized` `bool`

Определяет, является ли схема объекта проинициализированной.

`LicOperationPrefix` `string`

Префикс лицензируемой операции.

`LoadState` `EntityLoadState`

Состояние загрузки объекта.

`PrimaryColumnValue` `Guid`

Идентификатор первичной колонки.

`PrimaryDisplayColumnValue` `string`

Значение для отображения первичной колонки.

`Process` `Process`

Встроенный процесс объекта.

`Schema` `EntitySchema`

Экземпляр схемы объекта.

`SchemaName` `string`

Имя схемы объекта.

`StoringState` `StoringObjectState`

Состояние объекта (изменен, добавлен, удален, без изменений).

`UseAdminRights` `bool`

Определяет, будут ли учитываться права при вставке, обновлении, удалении и получении данных.

`UseDefRights` `bool`

Определяет, использовать ли права по умолчанию на объект.

`UseLazyLoad` `bool`

Определяет, использовать ли ленивую первоначальную загрузку данных объекта.

`UserConnection` `UserConnection`

Пользовательское подключение.

`ValidationMessages` `EntityValidationMessageCollection`

Коллекция сообщений, выводимых при проверке объекта.

`ValueListSchemaManager` `ValueListSchemaManager`

Экземпляр менеджера перечислений объекта.

`ValueListSchemaManagerName` `string`

Имя менеджера перечислений объекта.

Методы

```
void AddDefRights()
void AddDefRights(Guid primaryColumnValue)
void AddDefRights(IEnumerable<Guid> primaryColumnValues)
```

Для данного объекта устанавливает права по умолчанию.

Параметры

`primaryColumnValue`	Идентификатор значения права доступа.
`primaryColumnValues`	Массив идентификаторов значений прав доступа.

```
virtual object Clone()
```

Создает клон текущего экземпляра `Entity`.

```
Insert CreateInsert(bool skipLookupColumnValues)
```

Создает запрос на добавление данных в базу.

Параметры

`skipLookupColumnValues`	Признак добавления данных с учетом справочных колонок. По умолчанию установлено значение `false`.

```
Update CreateUpdate(bool skipLookupColumnValues)
```

Создает запрос на обновление данных в базе.

Параметры

`skipLookupColumnValues`	Параметр, определяющий необходимость добавления в базу данных колонок типа справочник. Если параметр равен `true`, то колонки типа справочник не будут добавлены в базу. Значение по умолчанию — `false`.

```
virtual bool Delete()
virtual bool Delete(object keyValue)
```

Удаляет из базы данных запись объекта.

Параметры

keyValue	Значение ключевого поля.

```
bool DeleteWithCancelProcess()
```

Удаляет из базы данных запись объекта и отменяет запущенный процесс.

```
static Entity DeserializeFromJson(UserConnection userConnection, string jsonValue)
```

Создает объект типа `Entity`, используя пользовательское подключение `userConnection`, и заполняет значения его полей из указанной строки формата JSON `jsonValue`.

Параметры

jsonValue	Строка формата JSON.
userConnection	Пользовательское подключение.

```
bool ExistInDB(EntitySchemaColumn conditionColumn, object conditionValue)
bool ExistInDB(string conditionColumnName, object conditionValue)
bool ExistInDB(object keyValue)
bool ExistInDB(Dictionary<string,object> conditions)
```

Определяет, существует ли в базе данных запись, отвечающая заданному условию запроса `conditionValue` к заданной колонке схемы объекта `conditionColumn` либо с заданным первичным ключом `keyValue`.

Параметры

conditionColumn	Колонка, для которой задается условие выборки.
conditionColumnName	Название колонки, для которой задается условие выборки.
conditionValue	Значение колонки условия для выбираемых данных.
conditions	Набор условий фильтрации выборки записей объекта.
keyValue	Значение ключевого поля.

```
bool FetchFromDB(EntitySchemaColumn conditionColumn, object conditionValue, bool useDisplayValue
bool FetchFromDB(string conditionColumnName, object conditionValue, bool useDisplayValues)
bool FetchFromDB(object keyValue, bool useDisplayValues)
bool FetchFromDB(Dictionary<string,object> conditions, bool useDisplayValues)
bool FetchFromDB(EntitySchemaColumn conditionColumn, object conditionValue, IEnumerable<EntitySc
bool FetchFromDB(string conditionColumnName, object conditionValue, IEnumerable<string>columnNam
```

По заданному условию загружает объект из базы данных.

Параметры

`conditionColumn`	Колонка, для которой задается условие выборки.
`conditionColumnName`	Название колонки, для которой задается условие выборки.
`conditionValue`	Значение колонки условия для выбираемых данных.
`columnsToFetch`	Список колонок, которые будут выбраны.
`columnNamesToFetch`	Список названий колонок, которые будут выбраны.
`conditions`	Набор условий фильтрации выборки записей объекта.
`keyValue`	Значение ключевого поля.
`useDisplayValues`	Признак получения в запросе первичных отображаемых значений. Если параметр равен `true`, в запросе будут возвращены первичные отображаемые значения.

```
bool FetchPrimaryColumnFromDB(object keyValue)
```

По заданному условию `keyValue` загружает из базы данных объект с первичной колонкой.

Параметры

`keyValue`	Значение ключевого поля.

```
bool FetchPrimaryInfoFromDB(EntitySchemaColumn conditionColumn, object conditionValue)
bool FetchPrimaryInfoFromDB(string conditionColumnName, object conditionValue)
```

По заданному условию загружает из базы данных объект с первичными колонками, включая колонку, первичную для отображения.

Параметры

conditionColumn	Колонка, для которой задается условие выборки.
conditionColumnName	Название колонки, для которой задается условие выборки.
conditionValue	Значение колонки условия для выбираемых данных.

```
byte[] GetBytesValue(string valueName)
```

Возвращает значение заданной колонки объекта в виде массива байт.

Параметры

valueName	Имя колонки объекта.

```
IEnumerable<EntityColumnValue> GetChangedColumnValues()
```

Возвращает коллекцию имен колонок объекта, которые были изменены.

```
string GetColumnDisplayValue(EntitySchemaColumn column)
```

Возвращает значение для отображения свойства объекта, соответствующее заданной колонке схемы объекта.

Параметры

column	Определенная колонка схемы объекта.

```
object GetColumnOldValue(string valueName)
object GetColumnOldValue(EntitySchemaColumn column)
```

Возвращает предыдущее значение заданного свойства объекта.

Параметры

column	Определенная колонка схемы объекта.
valueName	Имя колонки объекта.

```
virtual object GetColumnValue(string valueName)
virtual object GetColumnValue(EntitySchemaColumn column)
```

Возвращает значение колонки объекта с заданным именем, соответствующее переданной колонке

схемы объекта.

Параметры

`column`	Определенная колонка схемы объекта.
`valueName`	Имя колонки объекта.

```
IEnumerable<string> GetColumnValueNames()
```

Возвращает коллекцию имен колонок объекта.

```
virtual bool GetIsColumnValueLoaded(string valueName)
bool GetIsColumnValueLoaded(EntitySchemaColumn column)
```

Возвращает признак, определяющий, загружено ли заданное свойство объекта.

Параметры

`column`	Определенная колонка схемы объекта.
`valueName`	Имя колонки объекта.

```
virtual MemoryStream GetStreamValue(string valueName)
```

Возвращает преобразованное в экземпляр типа `System.IO.MemoryStream` значение переданной колонки схемы объекта.

Параметры

`valueName`	Имя колонки объекта.

```
TResult GetTypedColumnValue<TResult>(EntitySchemaColumn column)
```

Возвращает типизированное значение свойства объекта, соответствующее заданной колонке схемы объекта.

Параметры

`column`	Определенная колонка схемы объекта.

```
TResult GetTypedOldColumnValue<TResult>(string valueName)
```

```
TResult GetTypedOldColumnValue<TResult>(EntitySchemaColumn column)
```

Возвращает типизированное предыдущее значение свойства объекта, соответствующее заданной колонке схемы объекта.

Параметры

`column`	Определенная колонка схемы объекта.
`valueName`	Имя колонки объекта.

```
virtual bool InsertToDB(bool skipLookupColumnValues, bool validateRequired)
```

Добавляет запись текущего объекта в базу данных.

Параметры

`skipLookupColumnValues`	Параметр, определяющий необходимость добавления в базу данных колонок типа справочник. Если параметр равен `true`, то колонки типа справочник не будут добавлены в базу. Значение по умолчанию — `false`.
`validateRequired`	Параметр, определяющий необходимость проверки заполнения обязательных значений. Значение по умолчанию — `true`.

```
bool IsColumnValueLoaded(string valueName)
bool IsColumnValueLoaded(EntitySchemaColumn column)
```

Определяет, загружено ли значение свойства объекта с заданным именем.

Параметры

`column`	Определенная колонка схемы объекта.
`valueName`	Имя колонки объекта.

```
virtual bool Load(DataRow dataRow)
virtual bool Load(DataRow dataRow, Dictionary<string,string> columnMap)
virtual bool Load(IDataReader dataReader)
virtual bool Load(IDataReader dataReader, IDictionary<string,string> columnMap)
virtual bool Load(object dataSource)
virtual bool Load(object dataSource, IDictionary<string,string> columnMap)
```

Заполняет объект переданными данными.

Параметры

columnMap	Свойства объекта, заполняемые данными.
dataRow	Экземпляр `System.Data.DataRow`, из которого загружаются данные в объект.
dataReader	Экземпляр `System.Data.IDataReader`, из которого загружаются данные.
dataSource	Экземпляр `System.Object`, из которого загружаются данные.

```
void LoadColumnValue(string columnValueName, IDataReader dataReader, int fieldIndex, int binaryP
void LoadColumnValue(string columnValueName, IDataReader dataReader, int fieldIndex)
void LoadColumnValue(string columnValueName, object value)
void LoadColumnValue(EntitySchemaColumn column, object value)
```

Для свойства с заданным именем загружает его значение из переданного экземпляра.

Параметры

binaryPackageSize	Размер загружаемого значения.
column	Колонка схемы объекта.
columnValueName	Имя свойства объекта.
dataReader	Экземпляр `System.Data.IDataReader`, из которого загружается значение свойства.
fieldIndex	Индекс загружаемого из `System.Data.IDataReader` поля.
value	Загружаемое значение свойства.

```
static Entity Read(UserConnection userConnection, DataReader dataReader)
```

Возвращает значение текущего свойства типа `Entity` из потока ввода.

Параметры

dataReader	Экземпляр `System.Data.IDataReader`, из которого загружается значение свойства.
userConnection	Пользовательское подключение.

```
void ReadData(DataReader reader)
void ReadData(DataReader reader, EntitySchema schema)
```

Считывает данные из схемы объекта в заданный объект типа `System.Data.IDataReader`.

Параметры

`reader`	Экземпляр `System.Data.IDataReader`, в который загружаются данные схемы объекта.
`schema`	Схема объекта.

```
void ResetColumnValues()
```

Для всех свойств объекта отменяет изменения.

```
void ResetOldColumnValues()
```

Для всех свойств объекта отменяет изменения, устанавливая предыдущее значение.

```
bool Save(bool validateRequired)
```

Сохраняет объект в базе данных.

Параметры

`validateRequired`	Определяет необходимость проверки заполнения обязательных значений. Значение по умолчанию — `true`.

```
static string SerializeToJson(Entity entity)
```

Преобразует объект `entity` в строку формата `JSON`.

Параметры

`entity`	Экземпляр `Entity`.

```
virtual void SetBytesValue(string valueName, byte[] streamBytes)
```

Устанавливает для заданного свойства объекта переданное значение типа `System.Byte`.

Параметры

streamBytes	Значение типа `System.Byte`, которое устанавливается в заданную колонку объекта.
valueName	Имя колонки объекта.

```
bool SetColumnBothValues(EntitySchemaColumn column, object value, string displayValue)
bool SetColumnBothValues(string columnValueName, object value, string displayColumnValueName, st
```

Устанавливает свойству объекта, соответствующему заданной колонке схемы, переданные значение `value` и значение для отображения `displayValue`.

Параметры

column	Колонка схемы объекта.
displayValue	Загружаемое значение для отображения.
displayColumnValueName	Имя колонки, содержащей значение для отображения.
value	Загружаемое значение колонки.

```
bool SetColumnValue(string valueName, object value)
bool SetColumnValue(EntitySchemaColumn column, object value)
```

Устанавливает заданной колонке схемы переданное значение `value`.

Параметры

column	Колонка схемы объекта.
value	Загружаемое значение колонки.
valueName	Имя колонки объекта.

```
void SeddefColumnValue(string columnValueName, object defValue)
void SeddefColumnValue(string columnValueName)
```

Устанавливает значение по умолчанию свойству с заданным именем.

Параметры

columnValueName	Имя колонки объекта.
defValue	Значение по умолчанию.

```
void SeddefColumnValues()
```

Для всех свойств объекта устанавливает значения по умолчанию.

```
bool SetStreamValue(string valueName, Stream value)
```

Устанавливает для заданного свойства объекта переданное значение типа `System.IO.Stream`.

Параметры

value	Загружаемое значение колонки.
valueName	Имя колонки объекта.

```
virtual bool UpdateInDB(bool validateRequired)
```

Обновляет запись объекта в базе данных.

Параметры

validateRequired	Определяет необходимость проверки заполнения обязательных значений. Значение по умолчанию — `true`.

```
bool Validate()
```

Проверяет заполнение обязательных полей.

```
static void Write(DataWriter dataWriter, Entity entity, string propertyName)
static void Write(DataWriter dataWriter, Entity entity, string propertyName, bool couldConvertFc
```

Осуществляет запись значения типа `Entity` в поток вывода с заданными именем.

Параметры

couldConvertForXml	Разрешить преобразование для xml-сериализации.
dataWriter	Экземпляр класса `Terrasoft.Common.DataWriter`, предоставляющий методы последовательной записи значений в поток вывода.
entity	Значение для записи типа `Entity`.
propertyName	Имя объекта.

```
void Write(DataWriter dataWriter, string propertyName)
```

Осуществляет запись данных в поток вывода с заданным именем.

Параметры

dataWriter	Экземпляр класса `Terrasoft.Common.DataWriter`, предоставляющий методы последовательной записи значений в поток вывода.
propertyName	Имя свойства.

```
void WriteData(DataWriter writer)
void WriteData(DataWriter writer, EntitySchema schema)
```

Осуществляет запись в поток вывода для указанной либо текущей схемы объекта.

Параметры

schema	Схема объекта.
writer	Экземпляр класса `Terrasoft.Common.DataWriter`, предоставляющий методы последовательной записи значений в поток вывода.

События

```
event EventHandler<EntityColumnAfterEventArgs> ColumnValueChanged
```

Обработчик события, возникающего после изменения значения колонки объекта.

Обработчик события получает аргумент типа `EntityColumnAfterEventArgs`.

Свойства `EntityColumnAfterEventArgs` предоставляющие сведения, относящиеся к событию:

- `ColumnValueName`;
- `DisplayColumnValueName`.

`event EventHandler<EntityColumnBeforeEventArgs> ColumnValueChanging`

Обработчик события, возникающего перед изменением значения колонки объекта.

Обработчик события получает аргумент типа `EntityColumnBeforeEventArgs`.

Свойства `EntityColumnBeforeEventArgs` предоставляющие сведения, относящиеся к событию:

- `ColumnStreamValue`.
- `ColumnValue`.
- `ColumnValueName`.
- `DisplayColumnValue`.
- `DisplayColumnValueName`.

`event EventHandler<EventArgs> DefColumnValuesSet`

Обработчик события, возникающего после установки значений по умолчанию полей объекта.

`event EventHandler<EntityAfterEventArgs> Deleted`

Обработчик события, возникающего после удаления объекта.

Обработчик события получает аргумент типа `EntityAfterEventArgs`.

Свойства `EntityAfterEventArgs` предоставляющие сведения, относящиеся к событию:

- `ModifiedColumnValues`.
- `PrimaryColumnValue`.

`event EventHandler<EntityBeforeEventArgs> Deleting`

Обработчик события, возникающего перед удалением объекта.

Обработчик события получает аргумент типа `EntityBeforeEventArgs`.

Свойства `EntityBeforeEventArgs` предоставляющие сведения, относящиеся к событию:

- `AdditionalCondition`.
- `IsCanceled`.
- `KeyValue`.

`event EventHandler<EntityAfterEventArgs> Inserted`

Обработчик события, возникающего после вставки объекта.

Обработчик события получает аргумент типа `EntityAfterEventArgs`.

Свойства `EntityAfterEventArgs` предоставляющие сведения, относящиеся к событию:

- `ModifiedColumnValues`.
- `PrimaryColumnValue`.

`event EventHandler<EntityBeforeEventArgs> Inserting`

Обработчик события, возникающего перед вставкой объекта.

Обработчик события получает аргумент типа `EntityBeforeEventArgs`.

Свойства `EntityBeforeEventArgs` предоставляющие сведения, относящиеся к событию:

- `AdditionalCondition`.
- `IsCanceled`.
- `KeyValue`.

`event EventHandler<EntityAfterLoadEventArgs> Loaded`

Обработчик события, возникающего после загрузки объекта.

Обработчик события получает аргумент типа `EntityAfterLoadEventArgs`.

Свойства `EntityAfterLoadEventArgs` предоставляющие сведения, относящиеся к событию:

- `ColumnMap`.
- `DataSource`.

`event EventHandler<EntityBeforeLoadEventArgs> Loading`

Обработчик события, возникающего перед загрузкой объекта.

Обработчик события получает аргумент типа `EntityBeforeLoadEventArgs`.

Свойства `EntityBeforeLoadEventArgs` предоставляющие сведения, относящиеся к событию:

- `ColumnMap`.
- `DataSource`.
- `IsCanceled`.

`event EventHandler<EntityAfterEventArgs> Saved`

Обработчик события, возникающего после сохранения объекта.

Обработчик события получает аргумент типа `EntityAfterEventArgs`.

Свойства `EntityAfterEventArgs` предоставляющие сведения, относящиеся к событию:

- `ModifiedColumnValues`.
- `PrimaryColumnValue`.

`event EventHandler<EntitySaveErrorEventArgs> SaveError`

Обработчик события, возникающего при ошибке сохранения объекта.

Обработчик события получает аргумент типа `EntitySaveErrorEventArgs`.

Свойства `EntitySaveErrorEventArgs` предоставляющие сведения, относящиеся к событию:

- `Exception`.
- `IsHandled`.

`event EventHandler<EntityBeforeEventArgs> Saving`

Обработчик события, возникающего перед сохранением объекта.

Обработчик события получает аргумент типа `EntityBeforeEventArgs`.

Свойства `EntityBeforeEventArgs` предоставляющие сведения, относящиеся к событию:

- `AdditionalCondition`.
- `IsCanceled`.
- `KeyValue`.

`event EventHandler<EntityAfterEventArgs> Updated`

Обработчик события, возникающего после обновления объекта.

Обработчик события получает аргумент типа `EntityAfterEventArgs`.

Свойства `EntityAfterEventArgs` предоставляющие сведения, относящиеся к событию:

- `ModifiedColumnValues`.
- `PrimaryColumnValue`.

`event EventHandler<EntityBeforeEventArgs>Updating`

Обработчик события, возникающего перед обновлением объекта.

Обработчик события получает аргумент типа `EntityBeforeEventArgs`.

Свойства `EntityBeforeEventArgs` предоставляющие сведения, относящиеся к событию:

- `AdditionalCondition`.
- `IsCanceled`.
- `KeyValue`.

`event EventHandler<EntityValidationEventArgs> Validating`

Обработчик события, возникающего при проверке объекта.

Обработчик события получает аргумент типа `EntityValidationEventArgs`.

Свойства `EntityValidationEventArgs` предоставляющие сведения, относящиеся к событию:

- `Messages`.

Класс Select C#

Сложный

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 постраничного 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

`dbExecutor`	Экземпляр `DBExecutor`, который используется для выполнения запроса.

`IDataReader ExecuteReader(DBExecutor dbExecutor, CommandBehavior behavior)`

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

Параметры

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

`void ExecuteReader(ExecuteReaderReadMethod readMethod)`

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

Параметры

`readMethod`	Метод делегата `ExecuteReaderReadMethod`.

`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 `DBExecutor`, 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 `Case`.
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.

Класс EntitySchemaQuery C#

Сложный

Класс EntitySchemaQuery C#

Пространство имен `Terrasoft.Core.Entities`.

Класс `Terrasoft.Core.Entities.EntitySchemaQuery` предназначен для построения запросов выборки записей из таблиц базы данных с учетом прав доступа текущего пользователя. 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.

На заметку. Полный перечень методов и свойств класса `EntitySchemaQuery`, его родительских классов, а также реализуемых им интерфейсов, можно найти в Библиотеке .NET классов.

Конструкторы

```
public EntitySchemaQuery(EntitySchema rootSchema)
```

Создает экземпляр класса, в котором в качестве корневой схемы устанавливается переданный экземпляр `EntitySchema`. В качестве менеджера схем устанавливается менеджер переданного экземпляра корневой схемы.

```
public EntitySchemaQuery(EntitySchemaManager entitySchemaManager, string sourceSchemaName)
```

Создает экземпляр класса с указанным `EntitySchemaManager` и корневой схемы с именем, переданным в

качестве аргумента.

`public EntitySchemaQuery(EntitySchemaQuery source)`

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

Свойства

`Cache` `Terrasoft.Core.Store.ICacheStore`

Кэш запроса.

`CacheItemName` `string`

Имя элемента кэша.

`CanReadUncommitedData` `bool`

Определяет, попадут ли в результаты запроса данные, для которых не завершена транзакция.

`Caption` `Terrasoft.Common.LocalizableString`

Заголовок.

`ChunkSize` `int`

Количество строк запроса в одном чанке.

`Columns` `Terrasoft.Core.Entities.EntitySchemaQueryColumnCollection`

Коллекция колонок текущего запроса к схеме объекта.

`DataValueTypeManager` `DataValueTypeManager`

Менеджер значений типов данных.

`EntitySchemaManager` `Terrasoft.Core.Entities.EntitySchemaManager`

Менеджер схем объектов.

`Filters` `Terrasoft.Core.Entities.EntitySchemaQueryFilterCollection`

Коллекция фильтров текущего запроса к схеме объекта.

`HideSecurityValue` `bool`

Определяет, будут ли скрыты значения зашифрованных колонок.

`IgnoreDisplayValues` `bool`

Определяет, будут ли в запросе использоваться отображаемые значения колонок.

`IsDistinct` `bool`

Определяет, убирать ли дубли в результирующем наборе данных.

`IsInherited` `bool`

Определяет, является ли запрос унаследованным.

`JoinRightState` `QueryJoinRightLevel`

Определяет условие наложения прав при использовании связанных таблиц, если схема администрируется по записям.

`Manager` `Terrasoft.Core.IManager`

Менеджер схем.

`ManagerItem` `Terrasoft.Core.IManagerItem`

Элемент менеджера.

`Name` `string`

Имя.

`ParentCollection` `Terrasoft.Core.Entities.EntitySchemaQueryCollection`

Коллекция запросов, которой принадлежит текущий запрос к схеме объекта.

`ParentEntitySchema` `Terrasoft.Core.Entities.EntitySchema`

Родительская схема запроса.

`PrimaryQueryColumn` `Terrasoft.Core.Entities.EntitySchemaQueryColumn`

Колонка, созданная по первичной колонке корневой схемы. Заполняется при первом обращении.

`QueryOptimize` `bool`

Разрешает использование оптимизации запроса.

`RootSchema` `Terrasoft.Core.Entities.EntitySchema`

Корневая схема.

`RowCount` `int`

Количество строк, возвращаемых запросом.

`SchemaAliasPrefix` `string`

Префикс, используемый для создания псевдонимов схем.

`SkipRowCount` `int`

Количество строк, которые необходимо пропустить при возврате результата запроса.

`UseAdminRights` `bool`

Определяет будут ли учитываться права при построении запроса получения данных.

`UseLocalization` `bool`

Определяет, будут ли использоваться локализованные данные.

`UseOffsetFetchPaging` `bool`

Определяет возможность постраничного возврата результата запроса.

`UseRecordDeactivation` `bool`

Определяет, будут ли данные исключены из фильтрации.

Методы

`void AddAllSchemaColumns(bool skipSystemColumns)`

В коллекцию колонок текущего запроса к схеме объекта добавляет все колонки корневой схемы.

```
EntitySchemaQueryColumn AddColumn(string columnPath, AggregationTypeStrict aggregationType, out
void AddColumn(EntitySchemaQueryColumn queryColumn)
EntitySchemaQueryColumn AddColumn(string columnPath)
EntitySchemaQueryColumn AddColumn(EntitySchemaQueryFunction function)
EntitySchemaQueryColumn AddColumn(object parameterValue, DataValueType parameterDataValueType)
EntitySchemaQueryColumn AddColumn(EntitySchemaQuery subQuery)
```

Создает и добавляет колонку в текущий запрос к схеме объекта.

Параметры

`columnPath`	Путь к колонке схемы относительно корневой схемы.
`aggregationType`	Тип агрегирующей функции. В качестве параметра передаются значения перечисления типов агрегирующей функции `Terrasoft.Common.AggregationTypeStrict`.
`subQuery`	Ссылка на созданный подзапрос, помещенный в колонку.
`queryColumn`	Экземпляр `EntitySchemaQueryColumn`, добавляемый в коллекцию колонок текущего запроса.
`function`	Экземпляр функции `EntitySchemaQueryFunction`.
`parameterValue`	Значение параметра, добавляемого в запрос в качестве колонки.
`parameterDataValueType`	Тип значения параметра, добавляемого в запрос в качестве колонки.

```
void ClearCache()
```

Очищает кэш текущего запроса.

```
static void ClearDefCache(string cacheItemName)
```

Удаляет из кэша запроса элемент с заданным именем `cacheItemName`.

```
object Clone()
```

Создает клон текущего экземпляра `EntitySchemaQuery`.

```
EntitySchemaQueryExpression CreateAggregationEntitySchemaExpression(string leftExprColumnPath, A
```

Возвращает выражение агрегирующей функции с заданным типом агрегации из перечисления `Terrasoft.Common.AggregationTypeStrict` для колонки, расположенной по заданному пути `leftExprColumnPath`.

```
static EntitySchemaQueryExpression CreateParameterExpression(object parameterValue)
static EntitySchemaQueryExpression CreateParameterExpression(object parameterValue, DataValueTyp
static EntitySchemaQueryExpression CreateParameterExpression(object parameterValue, string displ
```

Создает выражение для параметра запроса.

Параметры

`parameterValue`	Значение параметра.
`valueType`	Тип значения параметра.
`displayValue`	Значение для отображения параметра.

```
static IEnumerable CreateParameterExpressions(DataValueType valueType, params object[] parameter
static IEnumerable CreateParameterExpressions(DataValueType valueType, IEnumerable<object> param
```

Создает коллекцию выражений для параметров запроса с определенным типом данных `DataValueType`.

```
static EntitySchemaQueryExpression CreateSchemaColumnExpression(EntitySchemaQuery parentQuery, E
static EntitySchemaQueryExpression CreateSchemaColumnExpression(EntitySchema rootSchema, string
EntitySchemaQueryExpression CreateSchemaColumnExpression(string columnPath, bool useCoalesceFunc
```

Возвращает выражение колонки схемы объекта.

Параметры

parentQuery	Запрос к схеме объекта, для которого создается выражение колонки.
rootSchema	Корневая схема.
columnPath	Путь к колонке относительно корневой схемы.
useCoalesceFunctionFor MultiLookup	Признак, использовать ли для колонки типа справочник функцию `COALESCE`. Необязательный параметр, по умолчанию равен `true`.
useDisplayValue	Признак, использовать ли для колонки значение для отображения. Необязательный параметр, по умолчанию равен `false`.

```
Enumerable CreateSchemaColumnExpressions(params string[] columnPaths)
IEnumerable CreateSchemaColumnExpressions(IEnumerable columnPaths, bool useCoalesceFunctionForMu
```

Возвращает коллекцию выражений колонок запроса к схеме объекта по заданной коллекции путей к колонкам `columnPaths`.

```
IEnumerable CreateSchemaColumnExpressionsWithoutCoalesce(params string[] columnPaths)
```

Возвращает коллекцию выражений колонок запроса к схеме объекта по заданному массиву путей к колонкам. При этом, если колонка имеет тип множественный справочник, к ее значениям не применяется функция `COALESCE`.

```
static EntitySchemaQueryExpression CreateSchemaColumnQueryExpression(string columnPath, EntitySc
static EntitySchemaQueryExpression CreateSchemaColumnQueryExpression(string columnPath, EntitySc
```

Возвращает выражение запроса к схеме объекта по заданным пути к колонке, корневой схеме и экземпляру колонки схемы. При этом для колонки можно определить, какой тип ее значения использовать в выражении — хранимое значение или значение для отображения.

```
EntitySchemaQueryExpression CreateSubEntitySchemaExpression(string leftExprColumnPath)
```

Возвращает выражение подзапроса к схеме объекта для колонки, расположенной по заданному пути `leftExprColumnPath`.

```
EntitySchemaAggregationQueryFunction CreateAggregationFunction(AggregationTypeStrict aggregatior
```

Возвращает экземпляр агрегирующей функции `EntitySchemaAggregationQueryFunction` с заданным типом агрегации из перечисления `Terrasoft.Common.AggregationTypeStrict` для колонки по указанному пути относительно корневой схемы `columnPath`.

```
EntitySchemaCaseNotNullQueryFunction CreateCaseNotNullFunction(params EntitySchemaCaseNotNullQue
```

Возвращает экземпляр `CASE`-функции `EntitySchemaCaseNotNullQueryFunction` для заданного массива выражений условий `EntitySchemaCaseNotNullQueryFunctionWhenItem`.

```
EntitySchemaCaseNotNullQueryFunctionWhenItem CreateCaseNotNullQueryFunctionWhenItem(string whenC
```

Возвращает экземпляр выражения для sql-конструкции вида `WHEN <Выражение_1> IS NOT NULL THEN <Выражение_2>`.

Параметры

whenColumnPath	Путь к колонке, содержащей выражение предложения `WHEN`.
thenParameterValue	Путь к колонке, содержащей выражение предложения `THEN`.

```
EntitySchemaCastQueryFunction CreateCastFunction(string columnPath, DBDataValueType castType)
```

Возвращает экземпляр `CAST`-функции `EntitySchemaCastQueryFunction` для выражения колонки, расположенной по заданному пути относительно корневой схемы `columnPath`, и указанным целевым типом данных `DBDataValueType`.

```
EntitySchemaCoalesceQueryFunction CreateCoalesceFunction(params string[] columnPaths)
static EntitySchemaCoalesceQueryFunction CreateCoalesceFunction(EntitySchemaQuery parentQuery, E
static EntitySchemaCoalesceQueryFunction CreateCoalesceFunction(EntitySchema rootSchema, params
```

Возвращает экземпляр функции, возвращающей первое не равное `null` выражение из списка аргументов, для заданных колонок.

Параметры

columnPaths	Массив путей к колонкам относительно корневой схемы.
parentQuery	Запрос к схеме объекта, для которого создается экземпляр функции.
rootSchema	Корневая схема.

```
EntitySchemaConcatQueryFunction CreateConcatFunction(params EntitySchemaQueryExpression[] expres
```

Возвращает экземпляр функции для формирования строки, являющейся результатом объединения строковых значений аргументов функции для заданного массива выражений `EntitySchemaQueryExpression`.

`EntitySchemaDatePartQueryFunction CreateDatePartFunction(EntitySchemaDatePartQueryFunctionInterv`

Возвращает экземпляр `DATEPART`-функции `EntitySchemaDatePartQueryFunction`, определяющей интервал даты, заданный перечислением `EntitySchemaDatePartQueryFunctionInterval` (месяц, день, час, год, день недели...), для значения колонки, расположенной по указанному пути относительно корневой схемы.

`EntitySchemaDatePartQueryFunction CreateDayFunction(string columnPath)`

Возвращает экземпляр функции `EntitySchemaDatePartQueryFunction`, определяющей интервал даты [*День*] для значения колонки, расположенной по указанному пути относительно корневой схемы.

`EntitySchemaDatePartQueryFunction CreateHourFunction(string columnPath)`

Возвращает экземпляр функции `EntitySchemaDatePartQueryFunction`, возвращающей часть даты [*Час*] для значения колонки, расположенной по указанному пути относительно корневой схемы.

`EntitySchemaDatePartQueryFunction CreateHourMinuteFunction(string columnPath)`

Возвращает экземпляр функции `EntitySchemaDatePartQueryFunction`, возвращающей часть даты [*Минута*] для значения колонки, расположенной по указанному пути относительно корневой схемы.

`EntitySchemaDatePartQueryFunction CreateMonthFunction(string columnPath)`

Возвращает экземпляр функции `EntitySchemaDatePartQueryFunction`, возвращающей часть даты [*Месяц*] для значения колонки, расположенной по указанному пути относительно корневой схемы.

`EntitySchemaDatePartQueryFunction CreateWeekdayFunction(string columnPath)`

Возвращает экземпляр функции `EntitySchemaDatePartQueryFunction`, возвращающей часть даты [*День недели*] для значения колонки, расположенной по указанному пути относительно корневой схемы.

`EntitySchemaDatePartQueryFunction CreateWeekFunction(string columnPath)`

Возвращает экземпляр функции `EntitySchemaDatePartQueryFunction`, возвращающей часть даты [*Неделя*] для значения колонки, расположенной по указанному пути относительно корневой схемы.

`EntitySchemaDatePartQueryFunction CreateYearFunction(string columnPath)`

Возвращает экземпляр функции `EntitySchemaDatePartQueryFunction`, возвращающей часть даты [*Год*] для значения колонки, расположенной по указанному пути относительно корневой схемы.

`EntitySchemaIsNullQueryFunction CreateIsNullFunction(string checkColumnPath, string replacementC`

Возвращает экземпляр функции `EntitySchemaIsNullQueryFunction` для колонок с проверяемым и замещающим значениями, которые расположены по заданным путям относительно корневой схемы.

```
EntitySchemaLengthQueryFunction CreateLengthFunction(string columnPath)
EntitySchemaLengthQueryFunction CreateLengthFunction(params EntitySchemaQueryExpression[] expres
```

Создание экземпляра функции `LEN` (функция для возврата длины выражения) для выражения колонки по заданному пути относительно корневой схемы или для заданного массива выражений.

```
EntitySchemaTrimQueryFunction CreateTrimFunction(string columnPath)
EntitySchemaTrimQueryFunction CreateTrimFunction(params EntitySchemaQueryExpression[] expressior
```

Возвращает экземпляр функции `TRIM` (функция для удаления начальных и конечных пробелов из выражения) для выражения колонки по заданному пути относительно корневой схемы или для заданного массива выражений.

```
EntitySchemaUpperQueryFunction CreateUpperFunction(string columnPath)
```

Возвращает экземпляр функции `EntitySchemaUpperQueryFunction`, для преобразования символов выражения аргумента к верхнему регистру, для выражения колонки по заданному пути относительно корневой схемы.

```
EntitySchemaCurrenddateQueryFunction CreateCurrenddateFunction()
```

Возвращает экземпляр функции `EntitySchemaCurrenddateQueryFunction`, определяющей текущую дату.

```
EntitySchemaCurrenddateTimeQueryFunction CreateCurrenddateTimeFunction()
```

Возвращает экземпляр функции `EntitySchemaCurrenddateTimeQueryFunction`, определяющей текущие дату и время.

```
EntitySchemaCurrentTimeQueryFunction CreateCurrentTimeFunction()
```

Возвращает экземпляр функции `EntitySchemaCurrentTimeQueryFunction`, определяющей текущее время.

```
EntitySchemaCurrentUserAccountQueryFunction CreateCurrentUserAccountFunction()
```

Возвращает экземпляр функции `EntitySchemaCurrentUserAccountQueryFunction`, определяющей идентификатор контрагента текущего пользователя.

```
EntitySchemaCurrentUserContactQueryFunction CreateCurrentUserContactFunction()
```

Возвращает экземпляр функции `EntitySchemaCurrentUserContactQueryFunction`, определяющей

идентификатор контакта текущего пользователя.

```
EntitySchemaCurrentUserQueryFunction CreateCurrentUserFunction()
```

Возвращает экземпляр функции `EntitySchemaCurrentUserQueryFunction`, определяющей текущего пользователя.

```
EntitySchemaQueryFilter CreateExistsFilter(string rightExpressionColumnPath)
```

Создает фильтр сравнения типа [*Существует по заданному условию*] и устанавливает в качестве проверяемого значения выражение колонки, расположенной по пути `rightExpressionColumnPath`.

```
IEntitySchemaQueryFilterItem CreateFilter(FilterComparisonType comparisonType, string leftExpres
IEntitySchemaQueryFilterItem CreateFilter(FilterComparisonType comparisonType, string leftExpres
IEntitySchemaQueryFilterItem CreateFilter(FilterComparisonType comparisonType, string leftExpres
IEntitySchemaQueryFilterItem CreateFilter(FilterComparisonType comparisonType, EntitySchemaQuery
IEntitySchemaQueryFilterItem CreateFilter(FilterComparisonType comparisonType, EntitySchemaQuery
IEntitySchemaQueryFilterItem CreateFilter(FilterComparisonType comparisonType, EntitySchemaQuery
IEntitySchemaQueryFilterItem CreateFilter(FilterComparisonType comparisonType, string leftExpres
IEntitySchemaQueryFilterItem CreateFilter(FilterComparisonType comparisonType, string leftExpres
IEntitySchemaQueryFilterItem CreateFilter(FilterComparisonType comparisonType, string leftExpres
IEntitySchemaQueryFilterItem CreateFilter(FilterComparisonType comparisonType, string leftExpres
EntitySchemaQueryFilter CreateFilter(FilterComparisonType comparisonType, string leftExprColumnP
EntitySchemaQueryFilter CreateFilter(FilterComparisonType comparisonType, string leftExprColumnP
EntitySchemaQueryFilter CreateFilter(FilterComparisonType comparisonType, string leftExprColumnP
EntitySchemaQueryFilter CreateFilter(FilterComparisonType comparisonType, string leftExprColumnP
EntitySchemaQueryFilter CreateFilter(FilterComparisonType comparisonType, string leftExprColumnP
IEntitySchemaQueryFilterItem CreateFilter(FilterComparisonType comparisonType, string leftExprCo
IEntitySchemaQueryFilterItem CreateFilter(FilterComparisonType comparisonType, string leftExprCo
```

Создает фильтр запроса для выборки записей по определенным условиям.

Параметры

comparisonType	Тип сравнения из перечисления `Terrasoft.Core.Entities.FilterComparisonType`.
leftExpressionColumnPath	Путь к колонке, содержащей выражение левой части фильтра.
leftExpression	Выражение в левой части фильтра.
leftExprAggregationType	Тип агрегирующей функции.
leftExprSubQuery	Параметр, в котором возвращается подзапрос для выражения в левой части фильтра (если он не равен `null`) либо подзапрос для первого выражения в правой части фильтра (если выражение левой части фильтра равно `null`).
rightExpressionColumnPaths	Массив путей к колонкам, содержащим выражения правой части фильтра.
rightExpression	Выражение в правой части фильтра.
rightExpressionValue	Экземпляр функции выражения в правой части фильтра (тип параметра `EntitySchemaQueryFunction`) или выражение подзапроса в правой части фильтра (тип параметра `EntitySchemaQuery`).
rightValue	Значение, которое обрабатывается макросом в правой части фильтра.
rightExprParameterValue	Значение параметра, к которому применяется агрегирующая функция в правой части фильтра.
macrosType	Тип макроса из перечисления `Terrasoft.Core.Entities.EntitySchemaQueryMacrosType`.
daysCount	Значение, к которому применяется макрос в правой части фильтра. Необязательный параметр, по умолчанию равен 0.

```
IEntitySchemaQueryFilterItem CreateFilterWithParameters(FilterComparisonType comparisonType, boo
IEntitySchemaQueryFilterItem CreateFilterWithParameters(FilterComparisonType comparisonType, str
IEntitySchemaQueryFilterItem CreateFilterWithParameters(FilterComparisonType comparisonType, str
static IEntitySchemaQueryFilterItem CreateFilterWithParameters(EntitySchemaQuery parentQuery, Er
static IEntitySchemaQueryFilterItem CreateFilterWithParameters(EntitySchema rootSchema, FilterCo
```

Создает параметризированный фильтр для выборки записей по определенным условиям.

Параметры

`parentQuery`	Родительский запрос, для которого создается фильтр.
`rootSchema`	Корневая схема.
`comparisonType`	Тип сравнения из перечисления `Terrasoft.Core.Entities.FilterComparisonType`.
`useDisplayValue`	Признак типа значения колонки, которое используется в фильтре: `true` - значение для отображения; `false` - хранимое значение.
`leftExpressionColumnPath`	Путь к колонке, содержащей выражение левой части фильтра.
`rightExpressionParameterValues`	Коллекция выражений параметров в правой части фильтра.

`IEntitySchemaQueryFilterItem CreateIsNotNullFilter(string leftExpressionColumnPath)`

Создает фильтр сравнения типа [*Не является null в базе данных*], устанавливая в качестве проверяемого значения выражение колонки, расположенной по указанному в параметре `leftExpressionColumnPath` пути.

`IEntitySchemaQueryFilterItem CreateIsNullFilter(string leftExpressionColumnPath)`

Создает фильтр сравнения типа [*Является null в базе данных*], устанавливая в качестве условия проверки выражение колонки, расположенной по указанному в параметре `leftExpressionColumnPath` пути.

`EntitySchemaQueryFilter CreateNotExistsFilter(string rightExpressionColumnPath)`

Создает фильтр сравнения типа [*Не существует по заданному условию*] и устанавливает в качестве проверяемого значения выражение колонки, расположенной по пути `rightExpressionColumnPath`.

`DataTable GetDataTable(UserConnection userConnection)`

Возвращает результат выполнения текущего запроса к схеме объекта в виде таблицы данных в памяти, используя пользовательское подключение `UserConnection`.

`static int GetDayOfWeekNumber(UserConnection userConnection, DayOfWeek dayOfWeek)`

Возвращает порядковый номер дня недели для объекта `System.DayOfWeek` с учетом региональных

настроек.

```
Entity GetEntity(UserConnection userConnection, object primaryColumnValue)
```

Возвращает экземпляр Entity по первичному ключу `primaryColumnValue`, используя пользовательское подключение `UserConnection`.

```
EntityCollection GetEntityCollection(UserConnection userConnection, EntitySchemaQueryOptions opt
EntityCollection GetEntityCollection(UserConnection userConnection)
```

Возвращает коллекцию экземпляров `Entity`, представляющих результаты выполнения текущего запроса, используя пользовательское подключение `UserConnection` и заданные дополнительные настройки запроса `EntitySchemaQueryOptions`.

```
EntitySchema GetSchema()
```

Возвращает экземпляр схемы объекта `EntitySchema` текущего экземпляра `EntitySchemaQuery`.

```
Select GetSelectQuery(UserConnection userConnection)
Select GetSelectQuery(UserConnection userConnection, EntitySchemaQueryOptions options)
```

Возвращает экземпляр запроса на выборку данных, используя пользовательское подключение `UserConnection` и заданные дополнительные настройки запроса `EntitySchemaQueryOptions`.

```
EntitySchemaQueryColumnCollection GetSummaryColumns()
EntitySchemaQueryColumnCollection GetSummaryColumns(IEnumerable<string> columnNames)
```

Возвращает коллекцию выражений колонок запроса, для которых вычисляются итоговые значения.

```
Entity GetSummaryEntity(UserConnection userConnection, EntitySchemaQueryColumnCollection summary
Entity GetSummaryEntity(UserConnection userConnection)
Entity GetSummaryEntity(UserConnection userConnection, IEnumerable<string> columnNames)
Entity GetSummaryEntity(UserConnection userConnection, params string[] columnNames)
```

Возвращает экземпляр `Entity` для результата, возвращаемого запросом на выборку итоговых значений.

Параметры

userConnection	Пользовательское подключение.
summaryColumns	Коллекция колонок запроса, для которых выбираются итоговые значения.
columnNames	Коллекция имен колонок.

```
Select GetSummarySelectQuery(UserConnection userConnection, EntitySchemaQueryColumnCollection su
Select GetSummarySelectQuery(UserConnection userConnection)
Select GetSummarySelectQuery(UserConnection userConnection, IEnumerable<string> columnNames)
Select GetSummarySelectQuery(UserConnection userConnection, params string[] columnNames)
```

Строит запрос на выборку итоговых значений для заданной коллекции колонок текущего экземпляра `EntitySchemaQuery`.

Параметры

userConnection	Пользовательское подключение.
summaryColumns	Коллекция колонок запроса, для которых выбираются итоговые значения.
columnNames	Коллекция имен колонок.

```
T GetTypedColumnValue(Entity entity, string columnName)
```

Возвращает типизированное значение колонки с именем `columnName` из переданного экземпляра `Entity`.

```
void LoadDataTableData(UserConnection userConnection, DataTable dataTable)
void LoadDataTableData(UserConnection userConnection, DataTable dataTable, EntitySchemaQueryOpti
```

Загружает результат выполнения текущего запроса к схеме объекта в объект `System.Data.DataTable`, используя пользовательское подключение `UserConnection` и заданные дополнительные настройки запроса `EntitySchemaQueryOptions`.

```
void RemoveColumn(string columnName)
```

Удаляет колонку с именем `columnName` из коллекции колонок текущего запроса.

```
void ResetSchema()
```

Очищает схему текущего экземпляра `EntitySchemaQuery`.

`void ResetSelectQuery()`

Очищает запрос на выборку для текущего запроса к схеме объекта.

`void SetLocalizationCultureId(System.Guid cultureId)`

Устанавливает идентификатор локальной культуры.

Класс Insert C#

Сложный

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

`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.

```
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 экземпляр `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#

Сложный

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

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

Класс UpdateSelect C#

Сложный

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 EntityMapper C#

Сложный

Класс `Terrasoft.Configuration.EntityMapper` — это утилитный класс конфигурации, который находится в пакете [*FinAppLending*] продукта Lending. `EntityMapper` позволяет сопоставлять данные одной сущности (`Entity`) с другой по правилам, определенным в конфигурационном файле. Использование подхода сопоставления данных разных сущностей позволяет избежать появления однообразного кода.

В продукте Lending существует два объекта, содержащих одинаковые колонки. Это объекты [*Физ. лицо*

] ([*Contact*]) и [*Анкета*] ([*AppForm*]). Также существует несколько деталей, относящихся к объекту [*Физ. лицо*] ([*Contact*]) и имеющих похожие детали, относящиеся к [*Анкета*] ([*AppForm*]). Очевидно, что при заполнении анкеты должна быть возможность по колонке [*Id*] объекта [*Физ. лицо*] ([*Contact*]) получить список всех его колонок и значений, а также список нужных деталей с их колонками и значениями, и сопоставить эти данные с данными, связанными с анкетой. После этого можно автоматически заполнить поля анкеты сопоставленными данными. Таким образом можно существенно уменьшить затраты на ручной ввод одинаковых данных.

Идея сопоставления данных разных сущностей реализована в следующих классах:

- `EntityMapper` — реализует логику сопоставления.
- `EntityResult` — определяет в каком виде вернется сопоставленная сущность.
- `MapConfig` — представляет набор правил для сопоставления.
- `DetailMapConfig` — используется для установки списка правил сопоставления деталей и связанных с ними сущностей.
- `RelationEntityMapConfig` — содержит правила для сопоставления связанных сущностей.
- `EntityFilterMap` — представляет из себя фильтр для запроса в базу данных.

Класс EntityMapper C#

Пространство имен `Terrasoft.Configuration`.

Класс реализует логику сопоставления.

Методы

```
virtual EntityResult GetMappedEntity(Guid recId, MapConfig config)
```

Возвращает сопоставленные данные для двух объектов `Entity`.

Параметры

`recId`	`GUID` записи в базе данных.
`config`	Экземпляр класса `MapConfig`, представляющий из себя набор правил сопоставления.

```
virtual Dictionary<string, object> GetColumnsValues(Guid recordId, MapConfig config, Dictionary<
```

Получает из базы данных главную сущность и сопоставляет ее колонки и значения по правилам, указанным в объекте `config`.

Параметры

`recordId`	`GUID` записи в базе данных.
`config`	Экземпляр класса `MapConfig`, представляющий из себя набор правил сопоставления.
`result`	Словарь колонок и их значений уже сопоставленной сущности.

`virtual Dictionary<string, object> GetRelationEntityColumnsValues(List<RelationEntityMapConfig>`

Получает из базы данных связанные сущности и сопоставляет их с основными сущностями.

Параметры

`relations`	Список правил для получения связанных записей.
`dictionaryToMerge`	Словарь с колонками и их значениями.
`columnName`	Название родительской колонки.
`entitylookup`	Объект, содержащий название и Id записи в базе.

Класс EntityResult C#

Пространство имен `Terrasoft.Configuration`.

Класс определяет в каком виде вернется сопоставленная сущность.

Свойства

`Columns Dictionary<string, object>`

Словарь с названиями колонок основной сущности и их значениями.

`Details Dictionary<string, List<Dictionary<string, object>>>`

Словарь названий деталей со списком их колонок и значений.

Класс MapConfig C#

Пространство имен `Terrasoft.Configuration`.

Класс представляет набор правил для сопоставления.

Свойства

`SourceEntityName` `string`

Название сущности в базе данных.

`Columns` `Dictionary<string, object>`

Словарь с названиями колонок одной сущности и сопоставляемыми колонками другой сущности.

`DetailsConfig` `List<DetailMapConfig>`

Список конфигурационных объектов с правилами для деталей.

`CleanDetails` `List<string>`

Список названий деталей для очистки их значений.

`RelationEntities` `List<RelationEntityMapConfig>`

Список конфигурационных объектов с правилами сопоставления связанных записей с главной сущностью.

Класс DetailMapConfig C#

Пространство имен `Terrasoft.Configuration`.

Класс используется для установки списка правил сопоставления деталей и связанных с ними сущностей.

Свойства

`DetailName` `string`

Название детали (для обеспечения уникальности экземпляра детали).

`SourceEntityName` `string`

Название сущности в базе данных.

`Columns` `Dictionary<string, object>`

Словарь с названиями колонок одной сущности и сопоставляемыми колонками другой сущности.

`Filters` `List<EntityFilterMap>`

Список конфигурационных объектов с правилами фильтрации для более точных выборок из базы данных.

`RelationEntities` `List<RelationEntityMapConfig>`

Список конфигурационных объектов с правилами сопоставления связанных записей с главной сущностью.

Класс RelationEntityMapConfig C#

Пространство имен `Terrasoft.Configuration`.

Класс содержит правила для сопоставления связанных сущностей.

Свойства

`ParentColumnName` `string`

Название родительской колонки, при нахождении которой будет срабатывать логика получения и сопоставления данных по сущности.

`SourceEntityName` `string`

Название сущности в базе данных.

`Columns` `Dictionary<string, object>`

Словарь с названиями колонок одной сущности и сопоставляемыми колонками другой сущности.

`Filters` `List<EntityFilterMap>`

Список конфигурационных объектов с правилами фильтрации для более точных выборок из базы данных.

`RelationEntities` `List<RelationEntityMapConfig>`

Список конфигурационных объектов с правилами сопоставления связанных записей с главной сущностью.

Класс EntityFilterMap C#

Пространство имен `Terrasoft.Configuration`.

Класс представляет из себя фильтр для запроса в базу данных.

Свойства

`ColumnName` `string`

Название колонки, при нахождении которой будет срабатывать логика фильтрации.

`Value` `object`

Значение, с которым необходимо сравнение.

Класс QueryFunction C#

Сложный

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 построителя запросов `DBEngine`.

Параметры

`sb`	Экземпляр `StringBuilder`, с помощью которого формируется текст запроса.
`dbEngine`	Экземпляр построителя 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 expressior`

Инициализирует новый экземпляр `AggregationQueryFunction` с заданным типом агрегирующей функции для указанного выражения колонки.

Параметры

`aggregationType`	Тип агрегирующей функции.
`expression`	Выражение колонки, к которому применяется агрегирующая функция.

`AggregationQueryFunction(AggregationTypeStrict aggregationType, IQueryColumnExpressionConvertibl`

Инициализирует новый экземпляр `AggregationQueryFunction` с заданным типом агрегирующей функции для указанного выражения колонки.

Параметры

`aggregationType`	Тип агрегирующей функции.
`expression`	Выражение колонки, к которому применяется агрегирующая функция.

`AggregationQueryFunction(AggregationQueryFunction source)`

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

Параметры

`source`	Агрегирующая функция выражения `AggregationQueryFunction`, 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 построителя запроса `DBEngine`.

Параметры

`sb`	Экземпляр `StringBuilder`, с помощью которого формируется текст запроса.
`dbEngine`	Экземпляр построителя 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

`checkExpression`	Выражение, которое проверяется на равенство `null`.
`replacementExpression`	Выражение, которое возвращается функцией, если `checkExpression` равно `null`.

```
IsNullQueryFunction(IsNullQueryFunction source)
```

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

Параметры

`source`	Агрегирующая функция выражения `IsNullQueryFunction`, 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 построителя запросов `DBEngine`.

Параметры

sb	Экземпляр `StringBuilder`, с помощью которого формируется текст запроса.
dbEngine	Экземпляр построителя 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

```
CreateGuidQueryFunction()
```

Инициализирует новый экземпляр `CreateGuidQueryFunction`.

```
CreateGuidQueryFunction(CreateGuidQueryFunction source)
```

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

Параметры

source	Функция `CreateGuidQueryFunction`, 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 `StringBuilder`, с помощью которого формируется текст запроса.
dbEngine	Экземпляр построителя запроса к базе данных.

```
override object Clone()
```

Создает клон текущего экземпляра `CreateGuidQueryFunction`.

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 `CurrentDateTimeQueryFunction`, 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 `StringBuilder`, 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 `CoalesceQueryFunction`, 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 `StringBuilder`, 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

`interval`	Часть даты.
`expression`	Выражение колонки типа `Дата/Время`.

`DatePartQueryFunction(DatePartQueryFunction source)`

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

Параметры

`source`	Функция `DatePartQueryFunction`, 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 `StringBuilder`, 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

`interval`	Часть даты, к которой добавляется временной промежуток.
`number`	Значение, которое добавляется к `interval`.
`expression`	Выражение колонки, содержащей исходную дату.

`DateAddQueryFunction(DateAddQueryFunction source)`

Инициализирует экземпляр `DateAddQueryFunction`, являющийся клоном переданной функции.

Параметры

`source`	Экземпляр функции `DateAddQueryFunction`, 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 `StringBuilder`, 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 `DateDiffQueryFunction`, 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 построителя запросов `DBEngine`.

Параметры

`sb`	Экземпляр `StringBuilder`, 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 `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 `CastQueryFunction`, 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 построителя запросов `DBEngine`.

Параметры

`sb`	Экземпляр `StringBuilder`, 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 `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

`expression`	Выражение колонки запроса.

```
UpperQueryFunction(UpperQueryFunction source)
```

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

Параметры

source	Функция `UpperQueryFunction`, 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 построителя запросов `DBEngine`.

Параметры

sb	Экземпляр `StringBuilder`, 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 `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 `CustomQueryFunction`, 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 `StringBuilder`, 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 `DataLengthQueryFunction`, 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 `StringBuilder`, 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 C#

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 `TrimQueryFunction`, 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 `StringBuilder`, 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 `LengthQueryFunction`, 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 `StringBuilder`, 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 `SubstringQueryFunction`, 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 построителя запросов `DBEngine`.

Параметры

`sb`	Экземпляр `StringBuilder`, 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 `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

`expressions`	Коллекция выражений колонок запроса.

`ConcatQueryFunction(ConcatQueryFunction source)`

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

Параметры

`source`	Функция `ConcatQueryFunction`, 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

`sb`	Экземпляр `StringBuilder`, 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 `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

`innerFunction`	Вложенная функция.

`WindowQueryFunction(QueryFunction innerFunction, QueryColumnExpression partitionByExpression = r`

Реализует функцию SQL окна.

Параметры

`innerFunction`	Вложенная функция.
`partitionByExpression`	Выражение для разделения запроса.
`orderByExpression`	Выражение для сортировки запроса.

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

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

Параметры

`source`	Функция `WindowQueryFunction`, 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

`sb`	Экземпляр `StringBuilder`, 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 `WindowQueryFunction`.

Класс EntitySchemaQueryFunction C#

Сложный

Класс `Terrasoft.Core.Entities.EntitySchemaQueryFunction` реализует функцию выражения.

Идея функции выражения реализована в следующих классах:

- `EntitySchemaQueryFunction` — базовый класс функции выражения запроса к схеме объекта.
- `EntitySchemaAggregationQueryFunction` — реализует агрегирующую функцию выражения.
- `EntitySchemaIsNullQueryFunction` — заменяет значения `null` замещающим выражением.
- `EntitySchemaCoalesceQueryFunction` — возвращает первое выражение из списка аргументов, не равное `null`.
- `EntitySchemaCaseNotNullQueryFunctionWhenItem` — класс, описывающий выражение условия sql-оператора `CASE`.
- `EntitySchemaCaseNotNullQueryFunctionWhenItems` — коллекция выражений условий sql-оператора `CASE`.
- EntitySchemaStartOfCurrentHourQueryFunction(EntitySchemaQuery parentQuery,

EntitySchemaQueryExpression expression, int offset = 0) : this(parentQuery, offset)

- `EntitySchemaCaseNotNullQueryFunction` — возвращает одно из множества возможных значений в зависимости от указанных условий.
- `EntitySchemaSystemValueQueryFunction` — возвращает выражение системного значения.
- `EntitySchemaCurrentDateTimeQueryFunction` — реализует функцию выражения текущей даты и времени.
- `EntitySchemaBaseCurrentDateQueryFunction` — базовый класс функции выражения для базовой даты.
- `EntitySchemaCurrentDateQueryFunction` — реализует функцию выражения текущей даты.
- `EntitySchemaDateToCurrentYearQueryFunction` — реализует функцию выражения даты начала текущей недели.
- `EntitySchemaStartOfCurrentWeekQueryFunction` — реализует функцию, которая конвертирует выражение даты в такую же дату текущего года.
- `EntitySchemaStartOfCurrentMonthQueryFunction` — реализует функцию выражения даты начала текущего месяца.
- `EntitySchemaStartOfCurrentQuarterQueryFunction` — реализует функцию выражения даты начала текущего квартала.
- `EntitySchemaStartOfCurrentHalfYearQueryFunction` — реализует функцию выражения даты начала текущего полугодия.
- `EntitySchemaStartOfCurrentYearQueryFunction` — реализует функцию выражения даты начала текущего года.
- `EntitySchemaBaseCurrentDateTimeQueryFunction` — базовый класс функции выражения базовых даты и времени.
- `EntitySchemaStartOfCurrentHourQueryFunction` — реализует функцию выражения начала текущего часа.
- `EntitySchemaCurrentTimeQueryFunction` — реализует функцию выражения текущего времени.
- `EntitySchemaCurrentUserQueryFunction` — реализует функцию выражения текущего пользователя.
- `EntitySchemaCurrentUserContactQueryFunction` — реализует функцию контакта текущего пользователя.
- `EntitySchemaCurrentUserAccountQueryFunction` — реализует функцию выражения контрагента текущего пользователя.
- `EntitySchemaDatePartQueryFunction` — реализует функцию запроса для части даты.
- `EntitySchemaUpperQueryFunction` — преобразовывает символы выражения аргумента к верхнему регистру.
- `EntitySchemaCastQueryFunction` — приводит выражение аргумента к заданному типу данных.
- `EntitySchemaTrimQueryFunction` — удаляет начальные и конечные пробелы из выражения.
- `EntitySchemaLengthQueryFunction` — возвращает длину выражения.
- `EntitySchemaConcatQueryFunction` — формирует строку, которая является результатом объединения строковых значений аргументов функции.
- `EntitySchemaWindowQueryFunction` — реализует функцию SQL окна.

Класс EntitySchemaQueryFunction C#

Пространство имен `Terrasoft.Core.Entities`.

Базовый класс функции выражения запроса к схеме объекта.

На заметку. Полный перечень методов класса `EntitySchemaQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации ".NET библиотеки классов ядра платформы".

Методы

`abstract QueryColumnExpression CreateQueryColumnExpression(DBSecurityEngine dbSecurityEngine)`

Возвращает выражение колонки запроса для текущей функции, сформированное с учетом заданных прав доступа.

Параметры

`dbSecurityEngine`	Объект `Terrasoft.Core.DB.DBSecurityEngine`, определяющий права доступа.

`abstract DataValueType GetResultDataValueType(DataValueTypeManager dataValueTypeManager)`

Возвращает тип данных возвращаемого функцией результата, используя переданный менеджер типов данных.

Параметры

`dataValueTypeManager`	Менеджер типов данных.

`abstract bool GetIsSupportepataValueType(DataValueType dataValueType)`

Определяет, имеет ли возвращаемый функцией результат указанный тип данных.

Параметры

`dataValueType`	Тип данных.

`abstract string GetCaption()`

Возвращает заголовок функции выражения.

```
virtual EntitySchemaQueryExpressionCollection GetArguments()
```

Возвращает коллекцию выражений аргументов функции.

```
void CheckIsSupportepataValueType(DataValueType dataValueType)
```

Проверяет, имеет ли возвращаемый функцией результат указанный тип данных. В противном случае генерируется исключение.

Параметры

`dataValueType`	Тип данных.

Класс EntitySchemaAggregationQueryFunction C#

Пространство имен `Terrasoft.Core.Entities`.

Класс реализует агрегирующую функцию выражения.

На заметку. Полный перечень методов и свойств класса `EntitySchemaAggregationQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации ".NET библиотеки классов ядра платформы".

Конструкторы

```
EntitySchemaAggregationQueryFunction(EntitySchemaQuery parentQuery)
```

Инициализирует экземпляр `EntitySchemaAggregationQueryFunction` заданного типа агрегирующей функции для заданного запроса к схеме объекта.

Параметры

`parentQuery`	Запрос к схеме объекта, которому принадлежит функция.

```
EntitySchemaAggregationQueryFunction(AggregationTypeStrict aggregationType, EntitySchemaQuery pa
```

Инициализирует экземпляр `EntitySchemaAggregationQueryFunction` заданного типа агрегирующей функции для заданного запроса к схеме объекта.

Параметры

aggregationType	Тип агрегирующей функции.
parentQuery	Запрос к схеме объекта, которому принадлежит функция.

`EntitySchemaAggregationQueryFunction(AggregationTypeStrict aggregationType, EntitySchemaQueryExp`

Инициализирует новый экземпляр `EntitySchemaAggregationQueryFunction` для заданных типа агрегирующей функции, выражения и запроса к схеме объекта.

Параметры

aggregationType	Тип агрегирующей функции.
expression	Выражение запроса.
parentQuery	Запрос к схеме объекта, которому принадлежит функция.

`EntitySchemaAggregationQueryFunction(EntitySchemaAggregationQueryFunction source)`

Инициализирует новый экземпляр `EntitySchemaAggregationQueryFunction`, являющийся клоном переданного экземпляра агрегирующей функции выражения.

Параметры

source	Экземпляр агрегирующей функции выражения, клон которой создается.

Свойства

`QueryAlias` `string`

Псевдоним функции в sql-запросе.

`AggregationType` `AggregationTypeStrict`

Тип агрегирующей функции.

`AggregationEvalType` `AggregationEvalType`

Область применения агрегирующей функции.

`Expression` `EntitySchemaQueryExpression`

Выражение аргумента агрегирующей функции.

Методы

`override void WriteMetaData(DataWriter writer)`

Выполняет сериализацию агрегирующей функции, используя заданный экземпляр `Terrasoft.Common.DataWriter`.

Параметры

`writer`	Экземпляр `Terrasoft.Common.DataWriter`, с помощью которого выполняется сериализация.

`override QueryColumnExpression CreateQueryColumnExpression(DBSecurityEngine dbSecurityEngine)`

Возвращает выражение колонки запроса для агрегирующей функции, сформированное с учетом заданных прав доступа.

Параметры

`dbSecurityEngine`	Объект `Terrasoft.Core.DB.DBSecurityEngine`, определяющий права доступа.

`override EntitySchemaQueryExpressionCollection GetArguments()`

Возвращает коллекцию выражений аргументов агрегирующей функции.

`override DataValueType GetResultDataValueType(DataValueTypeManager dataValueTypeManager)`

Возвращает тип данных возвращаемого агрегирующей функцией результата, используя заданный менеджер типов данных.

Параметры

`dataValueTypeManager`	Менеджер типов данных.

`override bool GetIsSupportepataValueType(DataValueType dataValueType)`

Определяет, имеет ли возвращаемый агрегирующей функцией результат указанный тип данных.

Параметры

`dataValueType`	Тип данных.

`override string GetCaption()`

Возвращает заголовок функции выражения.

`override object Clone()`

Создает клон текущего экземпляра `EntitySchemaAggregationQueryFunction`.

`EntitySchemaAggregationQueryFunction All()`

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

`EntitySchemaAggregationQueryFunction Distinct()`

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

Класс EntitySchemaIsNullQueryFunction C#

Пространство имен `Terrasoft.Core.Entities`.

Класс заменяет значения `null` замещающим выражением.

На заметку. Полный перечень методов и свойств класса `EntitySchemaIsNullQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации ".NET библиотеки классов ядра платформы".

Конструкторы

`EntitySchemaIsNullQueryFunction(EntitySchemaQuery parentQuery)`

Инициализирует экземпляр `EntitySchemaIsNullQueryFunction` для заданного запроса к схеме объекта.

Параметры

`parentQuery`	Запрос к схеме объекта, которому принадлежит функция.

`EntitySchemaIsNullQueryFunction(EntitySchemaQuery parentQuery, EntitySchemaQueryExpression check`

Инициализирует новый экземпляр `EntitySchemaIsNullQueryFunction` для заданных запроса к схеме объекта, проверяемого выражения и замещающего выражения.

Параметры

`parentQuery`	Запрос к схеме объекта, которому принадлежит функция.
`checkExpression`	Выражение, которое проверяется на равенство `null`.
`replacementExpression`	Выражение, которое возвращается функцией, если `checkExpression` равно `null`.

`EntitySchemaIsNullQueryFunction(EntitySchemaIsNullQueryFunction source)`

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

Параметры

`source`	Экземпляр функции `EntitySchemaIsNullQueryFunction`, клон которой создается.

Свойства

`QueryAlias string`

Псевдоним функции в sql-запросе.

`CheckExpression EntitySchemaQueryExpression`

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

`ReplacementExpression EntitySchemaQueryExpression`

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

Методы

`override void WriteMetaData(DataWriter writer)`

Выполняет сериализацию функции выражения, используя переданный экземпляр `DataWriter`.

Параметры

`writer`	Экземпляр `DataWriter`, с помощью которого выполняется сериализация функции выражения.

`override QueryColumnExpression CreateQueryColumnExpression(DBSecurityEngine dbSecurityEngine)`

Возвращает выражение колонки запроса для текущей функции, сформированное с учетом заданных прав доступа.

Параметры

`dbSecurityEngine`	Объект `Terrasoft.Core.DB.DBSecurityEngine`, определяющий права доступа.

`override EntitySchemaQueryExpressionCollection GetArguments()`

Возвращает коллекцию выражений аргументов функции.

`override DataValueType GetResultDataValueType(DataValueTypeManager dataValueTypeManager)`

Возвращает тип данных возвращаемого функцией результата, используя переданный менеджер типов данных.

Параметры

`dataValueTypeManager`	Менеджер типов данных.

Класс EntitySchemaCoalesceQueryFunction C#

Пространство имен `Terrasoft.Core.Entities`.

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

На заметку. Полный перечень методов и свойств класса `EntitySchemaCoalesceQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации ".NET библиотеки классов ядра платформы".

Конструкторы

`EntitySchemaCoalesceQueryFunction(EntitySchemaQuery parentQuery)`

Инициализирует новый экземпляр `EntitySchemaCoalesceQueryFunction` для заданного запроса к схеме объекта.

Параметры

`aggregationType`	Тип агрегирующей функции.
`parentQuery`	Запрос к схеме объекта, которому принадлежит функция.

`EntitySchemaCoalesceQueryFunction(EntitySchemaCoalesceQueryFunction source)`

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

Параметры

`source`	Функция `EntitySchemaCoalesceQueryFunction`, клон которой создается.

Свойства

`QueryAlias string`

Псевдоним функции в sql-запросе.

`Expressions EntitySchemaQueryExpressionCollection`

Коллекция выражений аргументов функции.

`HasExpressions bool`

Признак, определяющий наличие хотя бы одного элемента в коллекции выражений аргументов функции.

Методы

`override bool GetIsSupportepataValueType(DataValueType dataValueType)`

Определяет, имеет ли возвращаемый функцией результат указанный тип данных.

Параметры

`dataValueType`	Тип данных.

Класс EntitySchemaCaseNotNullQueryFunctionWhenItem C#

Пространство имен `Terrasoft.Core.Entities`.

Класс, описывающий выражение условия sql-оператора `CASE`.

На заметку. Полный перечень методов класса `EntitySchemaCaseNotNullQueryFunctionWhenItem`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации ".NET библиотеки классов ядра платформы".

Конструкторы

`EntitySchemaCaseNotNullQueryFunctionWhenItem()`

Инициализирует новый экземпляр `EntitySchemaCaseNotNullQueryFunctionWhenItem`.

`EntitySchemaCaseNotNullQueryFunctionWhenItem(EntitySchemaQueryExpression whenExpression, EntityS`

Инициализирует экземпляр `EntitySchemaCaseNotNullQueryFunctionWhenItem` для заданных выражений предложений `WHEN` и `THEN`.

Параметры

`whenExpression`	Выражение предложения `WHEN` условия.
`thenExpression`	Выражение предложения `THEN` условия.

`EntitySchemaCaseNotNullQueryFunctionWhenItem(EntitySchemaCaseNotNullQueryFunctionWhenItem source`

Инициализирует экземпляр `EntitySchemaCaseNotNullQueryFunctionWhenItem`, являющийся клоном переданной функции.

Параметры

`source`	Функция `EntitySchemaCaseNotNullQueryFunctionWhenItem`, клон которой создается.

Свойства

`WhenExpression` `EntitySchemaQueryExpression`

Выражение предложения `WHEN`.

`ThenExpression` `EntitySchemaQueryExpression`

Выражение предложения `THEN`.

Класс EntitySchemaCaseNotNullQueryFunctionWhenItems C#

Пространство имен `Terrasoft.Core.Entities`.

Класс реализует коллекцию выражений условий sql-оператора `CASE`.

На заметку. Полный перечень методов класса `EntitySchemaCaseNotNullQueryFunctionWhenItems`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации ".NET библиотеки классов ядра платформы".

Конструкторы

`EntitySchemaCaseNotNullQueryFunctionWhenItems()`

Инициализирует экземпляр `EntitySchemaCaseNotNullQueryFunctionWhenItems`.

`EntitySchemaCaseNotNullQueryFunctionWhenItems(EntitySchemaCaseNotNullQueryFunctionWhenItems sour`

Инициализирует новый экземпляр `EntitySchemaCaseNotNullQueryFunctionWhenItems`, являющийся клоном клоном переданной коллекции условий.

Параметры

`source`	Коллекция условий, клон которой создается.

Класс EntitySchemaCaseNotNullQueryFunction C#

Пространство имен `Terrasoft.Core.Entities`.

Класс возвращает одно из множества возможных значений в зависимости от указанных условий.

На заметку. Полный перечень методов и свойств класса `EntitySchemaCaseNotNullQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации ".NET библиотеки классов ядра платформы".

Конструкторы

`CurrentDateTimeQueryFunction()`

Инициализирует новый экземпляр `CurrentDateTimeQueryFunction`.

`EntitySchemaCaseNotNullQueryFunction(EntitySchemaQuery parentQuery)`

Инициализирует новый экземпляр `EntitySchemaCaseNotNullQueryFunction` для заданного запроса к схеме объекта.

Параметры

`parentQuery`	Запрос к схеме объекта, которому принадлежит функция.

`EntitySchemaCaseNotNullQueryFunction(EntitySchemaCaseNotNullQueryFunction source)`

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

Параметры

`source`	Функция `EntitySchemaCaseNotNullQueryFunction`, клон которой создается.

Свойства

`QueryAlias string`

Псевдоним функции в sql-запросе.

`WhenItems EntitySchemaCaseNotNullQueryFunctionWhenItems`

Коллекция условий функции выражения.

`HasWhenItems bool`

Признак, имеет ли функция хотя бы одно условие.

`ElseExpression EntitySchemaQueryExpression`

Выражение предложения `ELSE`.

Методы

`void SpecifyQueryAlias(string queryAlias)`

Определяет для текущей функции выражения заданный псевдоним в результирующем sql-запросе.

Параметры

`queryAlias`	Псевдоним, определяемый для текущей функции.

Класс EntitySchemaSystemValueQueryFunction C#

Пространство имен `Terrasoft.Core.Entities`.

Класс возвращает выражение системного значения.

На заметку. Полный перечень методов и свойств класса `EntitySchemaSystemValueQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации ".NET библиотеки классов ядра платформы".

Свойства

`QueryAlias string`

Псевдоним функции в sql-запросе.

`SystemValueName string`

Имя системного значения.

Класс EntitySchemaCurrentDateTimeQueryFunction C#

Пространство имен `Terrasoft.Core.Entities`.

Класс реализует функцию выражения текущей даты и времени.

На заметку. Полный перечень методов и свойств класса `EntitySchemaCurrentDateTimeQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации ".NET библиотеки классов ядра платформы".

Конструкторы

`EntitySchemaCurrentDateTimeQueryFunction(EntitySchemaQuery parentQuery)`

Инициализирует экземпляр `EntitySchemaCurrentDateTimeQueryFunction` для заданного запроса к схеме объекта.

Параметры

`parentQuery`	Запрос к схеме объекта, которому принадлежит функция.

`EntitySchemaCurrentDateTimeQueryFunction(EntitySchemaCurrentDateTimeQueryFunction source)`

Инициализирует экземпляр `EntitySchemaCurrentDateTimeQueryFunction`, являющийся клоном переданной функции.

Параметры

`source`	Экземпляр функции `EntitySchemaCurrentDateTimeQueryFunction`, клон которой создается.

Свойства

`SystemValueName string`

Имя системного значения.

Методы

`override string GetCaption()`

Возвращает заголовок функции выражения.

`override object Clone()`

Создает клон текущего экземпляра `EntitySchemaCurrentDateTimeQueryFunction`.

Класс EntitySchemaBaseCurrentDateQueryFunction C#

Пространство имен `Terrasoft.Core.Entities`.

Базовый класс функции выражения для базовой даты.

На заметку. Полный перечень методов и свойств класса `EntitySchemaBaseCurrentDateQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации ".NET библиотеки классов ядра платформы".

Свойства

`SystemValueName string`

Имя системного значения.

`Offset` `int`

Смещение.

Класс EntitySchemaCurrentDateQueryFunction C#

Пространство имен `Terrasoft.Core.Entities`.

Класс реализует функцию выражения текущей даты.

На заметку. Полный перечень методов и свойств класса `EntitySchemaCurrentDateQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации ".NET библиотеки классов ядра платформы".

Конструкторы

```
EntitySchemaCurrentDateQueryFunction(EntitySchemaQuery parentQuery, int offset = 0) : this(parer
EntitySchemaCurrentDateQueryFunction(EntitySchemaQuery parentQuery, EntitySchemaQueryExpression
```

Инициализирует экземпляр `EntitySchemaCurrentDateQueryFunction` с указанным смещением относительно базовой даты для заданного запроса к схеме объекта.

Параметры

`parentQuery`	Запрос к схеме объекта, которому принадлежит функция.
`offset`	Смещение в днях относительно контрольной даты. Значение по умолчанию - `0`.
`expression`	Выражение запроса.

```
EntitySchemaCurrentDateQueryFunction(EntitySchemaCurrentDateQueryFunction source)
```

Инициализирует экземпляр `EntitySchemaCurrentDateQueryFunction`, являющийся клоном переданной функции.

Параметры

`source`	Экземпляр функции `EntitySchemaCurrentDateQueryFunction`, клон которой создается.

Методы

`override string GetCaption()`

Возвращает заголовок функции выражения.

`override object Clone()`

Создает клон текущего экземпляра `EntitySchemaCurrentDateQueryFunction`.

Класс EntitySchemaDateToCurrentYearQueryFunction C#

Пространство имен `Terrasoft.Core.Entities`.

Класс реализует функцию, которая конвертирует выражение даты в такую же дату текущего года.

На заметку. Полный перечень методов и свойств класса `EntitySchemaDateToCurrentYearQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации ".NET библиотеки классов ядра платформы".

Конструкторы

`EntitySchemaDateToCurrentYearQueryFunction(EntitySchemaQuery parentQuery)`

Инициализирует новый экземпляр `EntitySchemaDateToCurrentYearQueryFunction` для заданного запроса к схеме объекта.

Параметры

`parentQuery`	Запрос к схеме объекта, которому принадлежит функция.

`EntitySchemaDateToCurrentYearQueryFunction(EntitySchemaQuery parentQuery, EntitySchemaQueryExpre`

Инициализирует новый экземпляр `EntitySchemaDateToCurrentYearQueryFunction` для заданного запроса к схеме объекта и переданного выражения даты.

Параметры

`parentQuery`	Запрос к схеме объекта, которому принадлежит функция.
`expression`	Выражение запроса.

`EntitySchemaDateToCurrentYearQueryFunction(EntitySchemaDateToCurrentYearQueryFunction source)`

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

Параметры

source	Функция `EntitySchemaDateToCurrentYearQueryFunction`, клон которой создается.

Свойства

`QueryAlias` `string`

Псевдоним функции в sql-запросе.

`Expression` `EntitySchemaQueryExpression`

Выражение аргументов функции.

Класс EntitySchemaStartOfCurrentWeekQueryFunction C#

Пространство имен `Terrasoft.Core.Entities`.

Класс реализует функцию выражения текущей даты.

На заметку. Полный перечень методов и свойств класса `EntitySchemaStartOfCurrentWeekQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации ".NET библиотеки классов ядра платформы".

Конструкторы

`EntitySchemaStartOfCurrentWeekQueryFunction(EntitySchemaQuery parentQuery, int offset = 0) : thi`
`EntitySchemaStartOfCurrentWeekQueryFunction(EntitySchemaQuery parentQuery, EntitySchemaQueryExpr`

Инициализирует экземпляр `EntitySchemaStartOfCurrentWeekQueryFunction` с указанным смещением относительно базовой даты для заданного запроса к схеме объекта.

Параметры

`parentQuery`	Запрос к схеме объекта, которому принадлежит функция.
`offset`	Смещение в днях относительно контрольной даты. Значение по умолчанию - `0`.
`expression`	Выражение запроса.

`EntitySchemaStartOfCurrentWeekQueryFunction(EntitySchemaStartOfCurrentWeekQueryFunction source)`

Инициализирует экземпляр `EntitySchemaStartOfCurrentWeekQueryFunction`, являющийся клоном переданной функции выражения.

Параметры

`source`	Экземпляр функции `EntitySchemaStartOfCurrentWeekQueryFunction`, клон которой создается.

Методы

`override string GetCaption()`

Возвращает заголовок функции выражения.

`override object Clone()`

Создает клон текущего экземпляра `EntitySchemaStartOfCurrentWeekQueryFunction`.

Класс EntitySchemaStartOfCurrentMonthQueryFunction C#

Пространство имен `Terrasoft.Core.Entities`.

Класс реализует функцию выражения даты начала текущего месяца.

На заметку. Полный перечень методов и свойств класса `EntitySchemaStartOfCurrentMonthQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации ".NET библиотеки классов ядра платформы".

Конструкторы

```
EntitySchemaStartOfCurrentMonthQueryFunction(EntitySchemaQuery parentQuery, int offset = 0) : th
EntitySchemaStartOfCurrentMonthQueryFunction(EntitySchemaQuery parentQuery, EntitySchemaQueryExp
```

Инициализирует экземпляр `EntitySchemaStartOfCurrentMonthQueryFunction` с указанным смещением относительно базовой даты для заданного запроса к схеме объекта.

Параметры

`parentQuery`	Запрос к схеме объекта, которому принадлежит функция.
`offset`	Смещение в днях относительно контрольной даты. Значение по умолчанию - `0`.
`expression`	Выражение запроса.

```
EntitySchemaStartOfCurrentMonthQueryFunction(EntitySchemaStartOfCurrentMonthQueryFunction source
```

Инициализирует экземпляр `EntitySchemaStartOfCurrentMonthQueryFunction`, являющийся клоном переданной функции выражения.

Параметры

`source`	Экземпляр функции `EntitySchemaStartOfCurrentMonthQueryFunction`, клон которой создается.

Методы

```
override string GetCaption()
```

Возвращает заголовок функции выражения.

```
override object Clone()
```

Создает клон текущего экземпляра `EntitySchemaStartOfCurrentMonthQueryFunction`.

Класс EntitySchemaStartOfCurrentQuarterQueryFunction C#

Пространство имен `Terrasoft.Core.Entities`.

Класс реализует функцию выражения даты начала текущего месяца.

На заметку. Полный перечень методов и свойств класса `EntitySchemaStartOfCurrentQuarterQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации ".NET библиотеки классов ядра платформы".

Конструкторы

```
EntitySchemaStartOfCurrentQuarterQueryFunction(EntitySchemaQuery parentQuery, int offset = 0) :
EntitySchemaStartOfCurrentQuarterQueryFunction(EntitySchemaQuery parentQuery, EntitySchemaQueryE
```

Инициализирует экземпляр `EntitySchemaStartOfCurrentQuarterQueryFunction` с указанным смещением относительно базовой даты для заданного запроса к схеме объекта.

Параметры

`parentQuery`	Запрос к схеме объекта, которому принадлежит функция.
`offset`	Смещение в днях относительно контрольной даты. Значение по умолчанию - `0`.
`expression`	Выражение запроса

```
EntitySchemaStartOfCurrentQuarterQueryFunction(EntitySchemaStartOfCurrentQuarterQueryFunction sc
```

Инициализирует экземпляр `EntitySchemaStartOfCurrentQuarterQueryFunction`, являющийся клоном переданной функции выражения.

Параметры

`source`	Экземпляр функции `EntitySchemaStartOfCurrentQuarterQueryFunction`, клон которой создается.

Методы

```
override string GetCaption()
```

Возвращает заголовок функции выражения.

```
override object Clone()
```

Создает клон текущего экземпляра `EntitySchemaStartOfCurrentQuarterQueryFunction`.

Класс EntitySchemaStartOfCurrentHalfYearQueryFunction C#

Пространство имен `Terrasoft.Core.Entities`.

Класс реализует функцию выражения даты начала текущего полугодия.

На заметку. Полный перечень методов и свойств класса

`EntitySchemaStartOfCurrentHalfYearQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации ".NET библиотеки классов ядра платформы".

Конструкторы

```
EntitySchemaStartOfCurrentHalfYearQueryFunction(EntitySchemaQuery parentQuery, int offset = 0) :
EntitySchemaStartOfCurrentHalfYearQueryFunction(EntitySchemaQuery parentQuery, EntitySchemaQuery
```

Инициализирует экземпляр `EntitySchemaStartOfCurrentHalfYearQueryFunction` с указанным смещением относительно базовой даты для заданного запроса к схеме объекта.

Параметры

`parentQuery`	Запрос к схеме объекта, которому принадлежит функция.
`offset`	Смещение в днях относительно контрольной даты. Значение по умолчанию - `0`.
`expression`	Выражение запроса.

```
EntitySchemaStartOfCurrentHalfYearQueryFunction(EntitySchemaStartOfCurrentHalfYearQueryFunction
```

Инициализирует экземпляр `EntitySchemaStartOfCurrentHalfYearQueryFunction`, являющийся клоном переданной функции выражения.

Параметры

`source`	Экземпляр функции `EntitySchemaStartOfCurrentHalfYearQueryFunction`, клон которой создается.

Методы

```
override string GetCaption()
```

Возвращает заголовок функции выражения.

```
override object Clone()
```

Создает клон текущего экземпляра `EntitySchemaStartOfCurrentHalfYearQueryFunction`.

Класс EntitySchemaStartOfCurrentYearQueryFunction C#

Пространство имен `Terrasoft.Core.Entities`.

Класс реализует функцию выражения даты начала текущего года.

На заметку. Полный перечень методов и свойств класса `EntitySchemaStartOfCurrentYearQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации ".NET библиотеки классов ядра платформы".

Конструкторы

```
EntitySchemaStartOfCurrentYearQueryFunction(EntitySchemaQuery parentQuery, int offset = 0) : thi
EntitySchemaStartOfCurrentYearQueryFunction(EntitySchemaQuery parentQuery, EntitySchemaQueryExpr
```

Инициализирует экземпляр `EntitySchemaStartOfCurrentYearQueryFunction` с указанным смещением относительно базовой даты для заданного запроса к схеме объекта.

Параметры

`parentQuery`	Запрос к схеме объекта, которому принадлежит функция.
`offset`	Смещение в днях относительно контрольной даты. Значение по умолчанию - `0`.
`expression`	Выражение запроса.

```
EntitySchemaStartOfCurrentYearQueryFunction(EntitySchemaStartOfCurrentYearQueryFunction source)
```

Инициализирует экземпляр `EntitySchemaStartOfCurrentYearQueryFunction`, являющийся клоном переданной функции выражения.

Параметры

`source`	Экземпляр функции `EntitySchemaStartOfCurrentYearQueryFunction`, клон которой создается.

Методы

```
override string GetCaption()
```

Возвращает заголовок функции выражения.

```
override object Clone()
```

Создает клон текущего экземпляра `EntitySchemaStartOfCurrentHalfYearQueryFunction`.

Класс EntitySchemaBaseCurrentDateTimeQueryFunction C#

Пространство имен `Terrasoft.Core.Entities`.

Базовый класс функции выражения базовых даты и времени.

На заметку. Полный перечень методов и свойств класса `EntitySchemaBaseCurrentDateTimeQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации ".NET библиотеки классов ядра платформы".

Свойства

`SystemValueName string`

Имя системного значения.

Класс EntitySchemaStartOfCurrentHourQueryFunction C#

Пространство имен `Terrasoft.Core.Entities`.

Класс реализует функцию выражения начала текущего часа.

На заметку. Полный перечень методов и свойств класса `EntitySchemaStartOfCurrentHourQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации ".NET библиотеки классов ядра платформы".

Конструкторы

`EntitySchemaStartOfCurrentHourQueryFunction(EntitySchemaQuery parentQuery, int offset = 0) : bas`

Инициализирует экземпляр `EntitySchemaStartOfCurrentHourQueryFunction`, который является частью `parentQuery` и указан `offset` относительно базовой даты.

Параметры

`parentQuery`	Экземпляр `EntitySchemaQuery`.
`offset`	Смещение в часах относительно базовой даты.

`EntitySchemaStartOfCurrentHourQueryFunction(EntitySchemaQuery parentQuery, EntitySchemaQueryExpr`

Инициализирует экземпляр `EntitySchemaStartOfCurrentHourQueryFunction`, который является частью `parentQuery`, имеет указанные аргументы `expression` и `offset` относительно базовой даты.

Параметры

`parentQuery`	Экземпляр `EntitySchemaQuery`.
`expression`	Выражение аргумента функции.
`offset`	Смещение в часах относительно базовой даты.

```
EntitySchemaStartOfCurrentHourQueryFunction(EntitySchemaStartOfCurrentHourQueryFunction source)
```

Инициализирует экземпляр `EntitySchemaStartOfCurrentHourQueryFunction`, являющийся клоном переданной функции выражения.

Параметры

`source`	Экземпляр функции `EntitySchemaStartOfCurrentHourQueryFunction`, клон которой создается.

Методы

```
override string GetCaption()
```

Возвращает заголовок функции выражения.

```
override object Clone()
```

Создает клон текущего экземпляра `EntitySchemaStartOfCurrentHourQueryFunction`.

Класс EntitySchemaCurrentTimeQueryFunction C#

Пространство имен `Terrasoft.Core.Entities`.

Класс реализует функцию выражения текущего времени.

На заметку. Полный перечень методов и свойств класса `EntitySchemaCurrentTimeQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации ".NET библиотеки классов ядра платформы".

Конструкторы

```
EntitySchemaCurrentTimeQueryFunction(EntitySchemaQuery parentQuery) : base(parentQuery)
```

Инициализирует новый экземпляр `EntitySchemaCurrentTimeQueryFunction` для заданного запроса к схеме объекта.

Параметры

`parentQuery`	Запрос к схеме объекта, которому принадлежит функция.

```
EntitySchemaCurrentTimeQueryFunction(EntitySchemaCurrentTimeQueryFunction source) : base(source)
```

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

Параметры

`source`	Функция `EntitySchemaCurrentTimeQueryFunction`, клон которой создается.

Свойства

```
SystemValueName string
```

Имя системного значения.

Методы

```
override string GetCaption()
```

Возвращает заголовок функции выражения.

```
override object Clone()
```

Создает клон текущего экземпляра `EntitySchemaCurrentTimeQueryFunction`.

Класс EntitySchemaCurrentUserQueryFunction C#

Пространство имен `Terrasoft.Core.Entities`.

Класс реализует функцию выражения текущего пользователя.

На заметку. Полный перечень методов и свойств класса `EntitySchemaCurrentUserQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации ".NET

библиотеки классов ядра платформы".

Конструкторы

`EntitySchemaCurrentUserQueryFunction(EntitySchemaQuery parentQuery) : base(parentQuery)`

Инициализирует новый экземпляр `EntitySchemaCurrentUserQueryFunction` для заданного запроса к схеме объекта.

Параметры

`parentQuery`	Запрос к схеме объекта, которому принадлежит функция.

`EntitySchemaCurrentUserQueryFunction(EntitySchemaCurrentUserQueryFunction source) : base(source)`

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

Параметры

`source`	Функция `EntitySchemaCurrentUserQueryFunction`, клон которой создается.

Свойства

`SystemValueName string`

Имя системного значения.

Методы

`override string GetCaption()`

Возвращает заголовок функции выражения.

`override object Clone()`

Создает клон текущего экземпляра `EntitySchemaCurrentUserQueryFunction`.

Класс EntitySchemaCurrentUserContactQueryFunction C#

Пространство имен `Terrasoft.Core.Entities`.

Класс реализует функцию выражения контакта текущего пользователя.

На заметку. Полный перечень методов и свойств класса `EntitySchemaCurrentUserContactQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации ".NET библиотеки классов ядра платформы".

Конструкторы

`EntitySchemaCurrentUserContactQueryFunction(EntitySchemaQuery parentQuery) : base(parentQuery)`

Инициализирует новый экземпляр `EntitySchemaCurrentUserContactQueryFunction` для заданного запроса к схеме объекта.

Параметры

`parentQuery`	Запрос к схеме объекта, которому принадлежит функция.

`EntitySchemaCurrentUserContactQueryFunction(EntitySchemaCurrentUserContactQueryFunction source)`

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

Параметры

`source`	Функция `EntitySchemaCurrentUserContactQueryFunction`, клон которой создается.

Свойства

`SystemValueName string`

Имя системного значения.

Методы

`override string GetCaption()`

Возвращает заголовок функции выражения.

`override object Clone()`

Создает клон текущего экземпляра `EntitySchemaCurrentUserContactQueryFunction`.

Класс EntitySchemaCurrentUserAccountQueryFunction C#

Пространство имен `Terrasoft.Core.Entities`.

Класс реализует функцию выражения контрагента текущего пользователя.

На заметку. Полный перечень методов и свойств класса `EntitySchemaCurrentUserAccountQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации ".NET библиотеки классов ядра платформы".

Конструкторы

`EntitySchemaCurrentUserAccountQueryFunction(EntitySchemaQuery parentQuery)`

Инициализирует новый экземпляр `EntitySchemaCurrentUserAccountQueryFunction` для заданного запроса к схеме объекта.

Параметры

`parentQuery`	Запрос к схеме объекта, которому принадлежит функция.

`EntitySchemaCurrentUserAccountQueryFunction(EntitySchemaCurrentUserAccountQueryFunction source)`

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

Параметры

`source`	Функция `EntitySchemaCurrentUserAccountQueryFunction`, клон которой создается.

Свойства

`SystemValueName string`

Имя системного значения.

Класс EntitySchemaDatePartQueryFunction C#

Пространство имен `Terrasoft.Core.Entities`.

Класс реализует функцию запроса для части даты.

На заметку. Полный перечень методов и свойств класса `EntitySchemaDatePartQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации ".NET библиотеки классов ядра платформы".

Конструкторы

`EntitySchemaDatePartQueryFunction(EntitySchemaQuery parentQuery) : base(parentQuery)`

Инициализирует новый экземпляр `EntitySchemaDatePartQueryFunction` для заданного запроса к схеме объекта.

Параметры

`parentQuery`	Экземпляр `EntitySchemaQuery`.

`EntitySchemaDatePartQueryFunction(EntitySchemaQuery parentQuery, EntitySchemaDatePartQueryFuncti`

Инициализирует новый экземпляр `EntitySchemaDatePartQueryFunction`, который является частью `parentQuery` с указанной частью даты `interval` для запроса к схеме сущности и выражению запроса `expression`.

Параметры

`parentQuery`	Экземпляр `EntitySchemaQuery`.
`interval`	Часть даты.
`expression`	Выражение запроса.

`EntitySchemaDatePartQueryFunction(EntitySchemaDatePartQueryFunction source) : base(source)`

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

Параметры

`source`	Функция `EntitySchemaDatePartQueryFunction`, клон которой создается.

Свойства

`QueryAlias string`

Псевдоним функции в sql-запросе.

`EntitySchemaDatePartQueryFunctionInterval Interval`

Часть даты, возвращаемая функцией.

`EntitySchemaQueryExpression Expression`

Выражение аргумента функции.

Методы

`override void WriteMetaData(DataWriter writer)`

Выполняет сериализацию функции, используя заданный экземпляр `Terrasoft.Common.DataWriter`.

Параметры

`writer`	Экземпляр `Terrasoft.Common.DataWriter`, с помощью которого выполняется сериализация.

`override QueryColumnExpression CreateQueryColumnExpression(DBSecurityEngine dbSecurityEngine)`

Возвращает выражение колонки запроса для текущей функции, сформированное с учетом заданных прав доступа.

Параметры

`dbSecurityEngine`	Объект `Terrasoft.Core.DB.DBSecurityEngine`, определяющий права доступа.

`override DataValueType GetResultDataValueType(DataValueTypeManager dataValueTypeManager)`

Возвращает тип данных возвращаемого функцией результата, используя переданный менеджер типов данных.

Параметры

`dataValueTypeManager`	Менеджер типов данных.

`override bool GetIsSupportedDataValueType(DataValueType dataValueType)`

Определяет, имеет ли возвращаемый функцией результат указанный тип данных.

Параметры

`dataValueType`	Тип данных.

`override string GetCaption()`

Возвращает заголовок функции выражения.

`override EntitySchemaQueryExpressionCollection GetArguments()`

Возвращает коллекцию выражений аргументов функции.

`override object Clone()`

Создает клон текущего экземпляра `EntitySchemaUpperQueryFunction`.

Класс EntitySchemaUpperQueryFunction C#

Пространство имен `Terrasoft.Core.Entities`.

Класс преобразовывает символы выражения аргумента к верхнему регистру.

На заметку. Полный перечень методов и свойств класса `EntitySchemaUpperQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации ".NET библиотеки классов ядра платформы".

Конструкторы

`EntitySchemaUpperQueryFunction(EntitySchemaQuery parentQuery)`

Инициализирует новый экземпляр `EntitySchemaUpperQueryFunction` для заданного запроса к схеме объекта.

Параметры

`parentQuery`	Запрос к схеме объекта, которому принадлежит функция.

`EntitySchemaUpperQueryFunction(EntitySchemaQuery parentQuery, EntitySchemaQueryExpression expres`

Инициализирует новый экземпляр `EntitySchemaUpperQueryFunction` для заданного запроса к схеме объекта и переданного выражения даты.

Параметры

`parentQuery`	Запрос к схеме объекта, которому принадлежит функция.
`expression`	Выражение запроса.

`EntitySchemaUpperQueryFunction(EntitySchemaUpperQueryFunction source)`

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

Параметры

`source`	Функция `EntitySchemaUpperQueryFunction`, клон которой создается.

Свойства

`QueryAlias` `string`

Псевдоним функции в sql-запросе.

`Expression` `EntitySchemaQueryExpression`

Выражение аргументов функции.

Класс EntitySchemaCastQueryFunction C#

Пространство имен `Terrasoft.Core.Entities`.

Класс приводит выражение аргумента к заданному типу данных.

На заметку. Полный перечень методов и свойств класса `EntitySchemaCastQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации ".NET библиотеки классов ядра платформы".

Конструкторы

`EntitySchemaCastQueryFunction(EntitySchemaQuery parentQuery, DBDataValueType castType)`

Инициализирует новый экземпляр `EntitySchemaCastQueryFunction` для заданного запроса к схеме объекта с указанным целевым типом данных.

Параметры

`parentQuery`	Запрос к схеме объекта, которому принадлежит функция.
`castType`	Целевой тип данных.

`EntitySchemaCastQueryFunction(EntitySchemaQuery parentQuery, EntitySchemaQueryExpression express`

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

Параметры

`parentQuery`	Запрос к схеме объекта, которому принадлежит функция.
`expression`	Выражение запроса.
`castType`	Целевой тип данных.

`EntitySchemaCastQueryFunction(EntitySchemaCastQueryFunction source)`

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

Параметры

`source`	Функция `EntitySchemaCastQueryFunction`, клон которой создается.

Свойства

`QueryAlias` `string`

Псевдоним функции в sql-запросе.

`Expression` `EntitySchemaQueryExpression`

Выражение аргумента функции.

`CastType DBDataValueType`

Целевой тип данных.

Класс EntitySchemaTrimQueryFunction C#

Пространство имен `Terrasoft.Core.Entities`.

Класс удаляет начальные и конечные пробелы из выражения.

На заметку. Полный перечень методов и свойств класса `EntitySchemaTrimQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации ".NET библиотеки классов ядра платформы".

Конструкторы

`EntitySchemaTrimQueryFunction(EntitySchemaQuery parentQuery)`

Инициализирует новый экземпляр `EntitySchemaTrimQueryFunction` для заданного запроса к схеме объекта.

Параметры

`parentQuery`	Запрос к схеме объекта, которому принадлежит функция.

`EntitySchemaTrimQueryFunction(EntitySchemaQuery parentQuery, EntitySchemaQueryExpression express`

Инициализирует новый экземпляр `EntitySchemaTrimQueryFunction` для заданного запроса к схеме объекта и переданного выражения даты.

Параметры

`parentQuery`	Запрос к схеме объекта, которому принадлежит функция.
`expression`	Выражение запроса.

`EntitySchemaTrimQueryFunction(EntitySchemaTrimQueryFunction source)`

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

Параметры

`source`	Функция `EntitySchemaTrimQueryFunction`, клон которой создается.

Свойства

`QueryAlias` `string`

Псевдоним функции в sql-запросе.

`Expression` `EntitySchemaQueryExpression`

Выражение аргументов функции.

Класс EntitySchemaLengthQueryFunction C#

Пространство имен `Terrasoft.Core.Entities`.

Класс возвращает длину выражения.

На заметку. Полный перечень методов и свойств класса `EntitySchemaLengthQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации ".NET библиотеки классов ядра платформы".

Конструкторы

`EntitySchemaLengthQueryFunction(EntitySchemaQuery parentQuery)`

Инициализирует новый экземпляр `EntitySchemaLengthQueryFunction` для заданного запроса к схеме объекта.

Параметры

`parentQuery`	Запрос к схеме объекта, которому принадлежит функция.

`EntitySchemaLengthQueryFunction(EntitySchemaQuery parentQuery, EntitySchemaQueryExpression expre`

Инициализирует новый экземпляр `EntitySchemaLengthQueryFunction` для заданного запроса к схеме объекта и переданного выражения даты.

Параметры

`parentQuery`	Запрос к схеме объекта, которому принадлежит функция.
`expression`	Выражение запроса.

`EntitySchemaLengthQueryFunction(EntitySchemaLengthQueryFunction source)`

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

Параметры

`source`	Функция `EntitySchemaLengthQueryFunction`, клон которой создается.

Свойства

`QueryAlias` `string`

Псевдоним функции в sql-запросе.

`Expression` `EntitySchemaQueryExpression`

Выражение аргументов функции.

Класс EntitySchemaConcatQueryFunction C#

Пространство имен `Terrasoft.Core.Entities`.

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

На заметку. Полный перечень методов и свойств класса `EntitySchemaConcatQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации ".NET библиотеки классов ядра платформы".

Конструкторы

`EntitySchemaConcatQueryFunction(EntitySchemaQuery parentQuery)`

Инициализирует новый экземпляр `EntitySchemaConcatQueryFunction` для заданного запроса к схеме объекта.

Параметры

`parentQuery`	Запрос к схеме объекта, которому принадлежит функция.

`EntitySchemaConcatQueryFunction(EntitySchemaQuery parentQuery, EntitySchemaQueryExpression[] exp`

Инициализирует новый экземпляр `EntitySchemaConcatQueryFunction` для заданных массива выражений и запроса к схеме объекта.

Параметры

`parentQuery`	Запрос к схеме объекта, которому принадлежит функция.
`expressions`	Массив выражений.

`EntitySchemaConcatQueryFunction(EntitySchemaConcatQueryFunction source)`

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

Параметры

`source`	Функция `EntitySchemaConcatQueryFunction`, клон которой создается.

Свойства

`QueryAlias` `string`

Псевдоним функции в sql-запросе.

`Expressions` `EntitySchemaQueryExpressionCollection`

Коллекция выражений аргументов функции.

`HasExpressions` `bool`

Признак, определяющий наличие хотя бы одного элемента в коллекции выражений аргументов функции.

Класс EntitySchemaWindowQueryFunction C#

Пространство имен `Terrasoft.Core.Entities`.

Класс реализует функцию SQL окна.

На заметку. Полный перечень методов и свойств класса `EntitySchemaWindowQueryFunction`, его родительских классов, а также реализуемых им интерфейсов, можно найти в документации ".NET библиотеки классов ядра платформы".

Конструкторы

`EntitySchemaWindowQueryFunction(EntitySchemaQuery parentQuery)`

Инициализирует новый экземпляр `EntitySchemaWindowQueryFunction` для заданного запроса к схеме объекта.

Параметры

`parentQuery`	Запрос к схеме объекта, которому принадлежит функция.

`EntitySchemaWindowQueryFunction(EntitySchemaQueryExpression function, EntitySchemaQuery esq)`

Инициализирует новый экземпляр `EntitySchemaWindowQueryFunction` для заданного запроса к схеме объекта.

Параметры

`function`	Вложенная функция запроса.
`esq`	Запрос к схеме объекта.

`EntitySchemaWindowQueryFunction(EntitySchemaQueryExpression function, EntitySchemaQuery esq, Ent`

Инициализирует новый экземпляр `EntitySchemaWindowQueryFunction` для заданного запроса к схеме объекта.

Параметры

`function`	Вложенная функция запроса.
`parentQuery`	Запрос к схеме объекта, которому принадлежит функция.
`partitionBy`	Выражение для разделения запроса.
`orderBy`	Выражение для сортировки запроса.

`EntitySchemaWindowQueryFunction(EntitySchemaQueryFunction source)`

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

Параметры

source	Функция `EntitySchemaQueryFunction`, клон которой создается.

`EntitySchemaWindowQueryFunction(EntitySchemaWindowQueryFunction source)`

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

Параметры

source	Функция `EntitySchemaWindowQueryFunction`, клон которой создается.

Свойства

`QueryAlias` `string`

Псевдоним функции в sql-запросе.

`InnerFunction` `EntitySchemaQueryExpression`

Функция для применения.

`PartitionByExpression` `EntitySchemaQueryExpression`

Разделение по пунктам.

`OrderByExpression` `EntitySchemaQueryExpression`

Сортировать по пункту.