Программный интерфейс C++ для NitrosBase RDF Storage

Руководство программиста

Содержание

Общие сведения

NitrosBase RDF Storage – это графовая база данных, основанная на семантических стандартах W3C включая RDF и SPARQL. NitrosBase RDF Storage специально разрабатывалось с целью объединить гибкость графовых моделей RDF и мощь высокопроизводительных баз данных.

Настоящее руководство распространяется на редакцию NitrosBase RDF Storage для работы под управлением ОС Linux, 64 bit и ОС Windows 7, ОС Windows server 2008, 64 bit.

Формат импорта-экспорта данных в NitrosBase RDF Storage: turtle/n3

Язык запросов, используемый в NitrosBase RDF Storage: SPARQL (SPARQL Query Language for RDF / W3C Recommendation 15 January 2008 - http://www.w3.org/TR/rdf-sparql-query/).

Настройка проекта на работу с NitrosBase RDF Storage

Для создания нового клиентского C++ приложения, необходимо добавить в проект файл nitrosbase_rdf.h. Он содержит C++ классы для установления соединения с базой данных, посылки запросов и получения результатов запросов.

Функции со строковыми параметрами или возвращающие строковые значения имеют две разные реализации: одну для строк в кодировке UTF8 и одну для строк в UTF16. Версии UTF8 предпочтительнее, потому что UTF8 используется для внутреннего представления. Версии UTF16 предназначены для поддержки языков программирования, содержащих строки UTF16 по умолчанию.

Замечание: Поставляемые примеры приложений уже содержат все необходимые настройки для взаимодействия с базой данных.

Установление соединения с базой данных

Чтобы получить дескриптор базы данных, вызовите функцию NBGetDatabaseUTF8 (или NBGetDatabaseUTF16). Например, такие вызовы:

```
int64_t dbh = NBGetDatabaseUTF8 ( "person", int len , "c:\\NitrosBaseRDF\\data\\person\\",
"c:\\NitrosBaseRDF" );
```

или

```
int64_t dbh = NBGetDatabaseUTF16 ( "person", int len , "c:\\NitrosBaseRDF\\data\\person\\",
"c:\\NitrosBaseRDF" );
```

вернут дескрипторы базы данных “person”.

Закрытие соединения с базой данных

Чтобы закрыть соединение с базой данных, вызовите функцию NBCloseConnect.

```
NBCloseConnect ( connecth );
```

Добавление данных

Для добавления данных в базу необходимо сначала подготовить данные, затем вызывать метод Execute. Существуют два способа подготовки данных:

1. Подготовка данных в строке
2. Подготовка данных в текстовом файле

Для добавления небольших объемов данных предпочтителен первый способ, если же объемы данных значительны – то предпочтителен второй способ. В обоих случаях данные подготавливаются в формате turtle/n3.

Подготовка данных в памяти компьютера

Пример:

```
int64_t dbh = NBGetDatabaseUTF8 ( "person", 6 , "c:\\NitrosBaseRDF\\data\\person\\",
"c:\\NitrosBaseRDF" );
int64_t connecth = NBConnect ( dbh );
string querytext =
        "<http://Person10000> type k:Person. "
        "<http://Person10000> ID \"10000\"^^xsd:integer. "
        "<http://Person10000> First_Name \"Jonny\". "
        "<http://Person10000> Last_Name \"Fisher\". "
        "<http://Person10000> Male \"0\"^^xsd:integer. ";

ExecuteQueryUTF8 ( connecth, querytext.c_str(), querytext.length(), QUERY_INSERTTRIPLES );
NBCloseConnect ( connecth );
```

или

```
int64_t dbh = NBGetDatabaseUTF16 ( L"person", 6 , L"c:\\NitrosBaseRDF\\data\\person\\",
L"c:\\NitrosBaseRDF" );
int64_t connecth = NBConnect ( dbh );
wstring querytext =
        L"<http://Person10000> type k:Person. "
        L"<http://Person10000> ID \"10000\"^^xsd:integer. "
        L"<http://Person10000> First_Name \"Jonny\". "
        L"<http://Person10000> Last_Name \"Fisher\". "
        L"<http://Person10000> Male \"0\"^^xsd:integer. ";

ExecuteQueryUTF16 ( connecth, querytext.c_str(), querytext.length(), QUERY_INSERTTRIPLES );
NBCloseConnect ( connecth );
```

Метод Execute в данном примере использует в качестве параметра строку querytext, содержащую набор троек в формате turtle/n3. Все тройки с subject равным http://Person10000 ранее не были добавлены в базу данных, поэтому они добавляются как новые данные.

Подготовка данных в текстовом файле

Пример:

```
int64_t dbh = NBGetDatabaseUTF8 ( "person", 6 , "c:\\NitrosBaseRDF\\data\\person\\",
"c:\\NitrosBaseRDF" );
int64_t connecth = NBConnect ( dbh );
string filepath = "c:\\NitrosBaseRDF\\data\\person\\person.n3";
ExecuteQueryUTF8 ( connecth, filepath.c_str(), filepath.length(), QUERY_INSERTTRIPLES_FILE );
NBCloseConnect ( connecth );
```

В данном примере *filepath*– это путь к файлу, содержащему набор троек в формате *turtle/n3*. Например, файл *person.n3* в данном примере мог бы содержать следующие текстовые строки:

```
<http://Person10000> type k:Person.
<http://Person10000> ID "10000"^^xsd:integer.
<http://Person10000> First_Name "Jonny".
<http://Person10000> Last_Name "Fisher".
<http://Person10000> Male "0"^^xsd:integer.
```

Внимание!

Данная версия NitrosBase RDF Storage не поддерживает множественных литеральных значений object в тройках, имеющих одинаковые subject и predicate. Например, вы не можете одному человеку (subject) присвоить 2 разных фамилии. Поэтому, следующая конфигурация троек является некорректной:

```
<http://Person10000> Last_Name "Fisher".
<http://Person10000> Last_Name "Hunter".
```

Если же object представлен в виде URI то такая множественность возможна. Например, одна статья может иметь несколько авторов, и следующая конфигурация троек является корректной:

```
<http://Article90000> Author <http://Person12345>.
<http://Article90000> Author <http://Person12345>.
```

Такая интерпретация литеральных значений objects характерна для многих RDF баз. Мы также используем эту особенность для реализации простого и изящного алгоритма модификации данных (см. раздел "Модификация литеральных данных").

Удаление данных

В NitrosBase RDF Storage реализованы следующие способы удаления данных.

1. Удаление указанных троек.
2. Удаление данных, отобранных поисковым запросом.
3. Удаление по списку URI

Удаление указанных троек

Способ позволяет удалить указанные тройки. Тройки, которые надо удалить, задаются в формате turtle/n3. Удаление можно произвести двумя способами:

Первый способ. В этом случае набор троек в виде строки передаётся в запрос, как параметр.

```
int64_t dbh = NBGetDatabaseUTF8 ( "person", 6 , "c:\\NitrosBaseRDF\\data\\person\\",
"c:\\NitrosBaseRDF" );
int64_t connecth = NBConnect ( dbh );
string querytext =
        "<http://Person_clone19> System_Str_10p \"KKKK\". "
        "http://Person_clone19> p0link <http://Person4292>. ";
ExecuteQueryUTF8 ( connecth, querytext.c_str(), querytext.length(), QUERY_DELETETRIPLES );
NBCloseConnect ( connecth );
```

Второй способ. В этом случае набор троек находится в файле, в качестве параметра передаётся полный путь к этому файлу.

```
int64_t dbh = NBGetDatabaseUTF8 ( "person", 6 , "c:\\NitrosBaseRDF\\data\\person\\",
"c:\\NitrosBaseRDF" );
int64_t connecth = NBConnect ( dbh );
string filepath = "c:\\NitrosBaseRDF\\data\\q_person\\deletetriples.n3";
ExecuteQueryUTF8 ( connecth, filepath.c_str(), filepath.length(), QUERY_DELETETRIPLES_FILE);
NBCloseConnect ( connecth );
```

Файл *deletetriples.n3* в данном примере мог бы содержать следующие текстовые строки:

```
<http://Person_clone19> System_Str_10p "KKKK".
<http://Person_clone19> p0link <http://Person4292>.
```

Удаление данных отобранных поисковым запросом

Способ позволяет удалить все тройки, с URI, полученными в результате поискового запроса. Для этого клиент передаёт на сервер запрос на удаление троек. Запрос должен удовлетворять следующему требованию: список полей SPARQL инструкции SELECT должен содержать единственную переменную типа URI троек, которые должны быть удалены.
Пример:

```
int64_t dbh = NBGetDatabaseUTF8 ( "person", 6 , "c:\\NitrosBaseRDF\\data\\person\\",
"c:\\NitrosBaseRDF" );
int64_t connecth = NBConnect ( dbh );
string querytext =
        "SELECT ?Person "
        "WHERE  "
        "{ "
        "   ?Person ID ?ID1. "
        "   ?Person type k:Person. "
        "   FILTER(?ID1 < "6"^^xsd:integer) "
        "}";
ExecuteQueryUTF8 ( connecth, querytext.c_str(), querytext.length(),
        QUERY_DELETERECORDS_QUERY );
NBCloseConnect ( connecth );
```

В этом примере удаляются все тройки с URI полученными в результате запроса (на месте subject и object).

Удаление по списку URI

Удаляет из списка все тройки с указанными URI.

Пример:

```
int64_t dbh = NBGetDatabaseUTF8 ( "person", 6 , "c:\\NitrosBaseRDF\\data\\person\\",
"c:\\NitrosBaseRDF" );
int64_t connecth = NBConnect ( dbh );
string querytext =
        "<http://Person_clone19>\n"
        "<http://Person_clone33> \n"
        "<http://Person101>";
ExecuteQueryUTF8 ( connecth, querytext.c_str(), querytext.length(),
        QUERY_DELETERECORDS_URILIST );
NBCloseConnect ( connecth );
```

В этом примере все тройки с URI из параметра querytext будут удалены.

Модификация литеральных данных

Существуют два способа модификации существующих данных:

1. Замена путем удаления и добавления троек.
2. Прямая замена данных в тройках.

Замена путем удаления и добавления троек

Пример:

```
int64_t dbh = NBGetDatabaseUTF8 ( "person", 6 , "c:\\NitrosBaseRDF\\data\\person\\",
"c:\\NitrosBaseRDF" );
int64_t connecth = NBConnect ( dbh );
string querytext = "<http://Person_clone19> System_Str_10p \"KKKK\".";
ExecuteQueryUTF8 ( connecth, querytext.c_str(), querytext.length(), QUERY_DELETETRIPLES);

querytext = "<http://Person_clone19> System_Str_10p \"JJJJ\".";
ExecuteQueryUTF8 ( connecth, querytext.c_str(), querytext.length(), QUERY_INSERTRIPLES);
NBCloseConnect ( connecth );
```

Первый вызов ExecuteQueryUTF8 с QUERY_DELETRIPLES
удаляет тройку <http: // Person_clone19> System_Str_10p "KKKK".

Второй вызов ExecuteQueryUTF8 с QUERY_INSERTRIPLES
добавляет тройку <http: // Person_clone19> System_Str_10p "JJJJ".

Прямая замена данных в тройках

Данный способ использует указанную в разделе особенность, что NitrosBase RDF Storage не поддерживает множественных литеральных значений object в тройках, имеющих одинаковые subject и predicate. Это позволяет заменить существующее значение, если добавить тройку с существующим subject и существующим predicate, но с другим значением object.

Пример:

```
int64_t dbh = NBGetDatabaseUTF8 ( "person", 6 , "c:\\NitrosBaseRDF\\data\\person\\",
"c:\\NitrosBaseRDF" );
int64_t connecth = NBConnect ( dbh );
querytext = "<http://Person_clone19> System_Str_10p \"JJJJ\".";
ExecuteQueryUTF8 ( connecth, querytext.c_str(), querytext.length(), QUERY_INSERTRIPLES);
NBCloseConnect ( connecth );
```

Поскольку существование множественных литеральных object при одинаковых subject и predicate невозможно, вызов метода Execute с параметром DATABASE_ INSERTTRIPLES_SMEM в данном случае заменит "KKKK" на "JJJJ".

Модификация ссылочных данных

Для модификации троек содержащих URI в качестве значения object необходимо сначала удалить существующую тройку, затем добавить новую.

Пример:

```
int64_t dbh = NBGetDatabaseUTF8 ( "person", 6 , "c:\\NitrosBaseRDF\\data\\person\\",
"c:\\NitrosBaseRDF" );
int64_t connecth = NBConnect ( dbh );
string querytext = "<http://Person_clone11> p0link <http://Person9505>.";
ExecuteQueryUTF8 ( connecth, querytext.c_str(), querytext.length(), QUERY_DELETETRIPLES);

querytext = "<http://Person_clone11> p0link <http://Person9506>.";
ExecuteQueryUTF8 ( connecth, querytext.c_str(), querytext.length(), QUERY_INSERTRIPLES);
NBCloseConnect ( connecth );
```

Сохранение и восстановление данных

Для того, чтобы зафиксировать состояние базы данных в некоторый момент времени, существует способ сохранения данных базы в виде набора троек в файл формата turtle/n3.

Пример:

Для Windows:

```
int64_t dbh = NBGetDatabaseUTF8 ( "person", 6 , "c:\\NitrosBaseRDF\\data\\person\\",
"c:\\NitrosBaseRDF" );
int64_t connecth = NBConnect ( dbh );
string n3file = "c:\\NitrosBaseRDF\\data\\n3files\\person_19_05_2014.n3";
ExecuteQueryUTF8 ( connecth, n3file.c_str(), n3file.length(), DATABASE_SAVE2TURTLEN3 );
NBCloseConnect ( connecth );
```

Для Linux:

```
int64_t dbh = NBGetDatabaseUTF8 ( "person" );
int64_t connecth = NBConnect ( dbh );
string n3file = "data/n3files/person_19_05_2014.n3";
ExecuteQueryUTF8 ( connecth, n3file.c_str(), n3file.length(), DATABASE_SAVE2TURTLEN3 );
NBCloseConnect ( connecth );
```

В данном примере состояние базы данных на 19 мая 2014 года будет сохранено в виде файла троек ‘person_19_05_2014.n3’.

Чтобы восстановить состояние базы данных на этот момент (например, из-за сбоя и порчи данных), нужно заново создать базу данных на основе файла троек ‘person_19_05_2014.n3’

Получение данных с помощью SPARQL

Выполнение запроса

Метод Execute класса CClientQuery отдает SPARQL запрос на выполнение серверу базы данных.

Пример:

```
int64_t dbh = NBGetDatabaseUTF8 ( "person", 6 , "c:\\NitrosBaseRDF\\data\\person\\",
"c:\\NitrosBaseRDF" );
int64_t connecth = NBConnect ( dbh );
string querytext = "SELECT ?uri ?Male ?ID WHERE { ?uri ID ?ID. ?uri Male ?Male.}";
ExecuteQueryUTF8 ( connecth, querytext.c_str(), querytext.length(), QUERY_SPARQL );
NBCloseConnect ( connecth );
```

Постраничное выполнение запроса

Функция ExecuteQueryUTF8 с параметром QUERY_SPARQL посылает запрос SPARQL для выполнения сервером и извлекает страницу из набора результатов.

Пример:

```
int64_t dbh = NBGetDatabaseUTF8 ( "person", 6 , "c:\\NitrosBaseRDF\\data\\person\\",
"c:\\NitrosBaseRDF" );
int64_t connecth = NBConnect ( dbh );
string querytext = "SELECT ?uri ?Male ?ID WHERE { ?uri ID ?ID. ?uri Male ?Male.}";
int count = ExecutePagingQueryUTF8 ( connecth, querytext.c_str(), querytext.length(), from, to,
afterto);
NBCloseConnect ( connecth );
```

Получение количества полей, имен и типов

Для получения количества полей, имен и типов предоставляются следующие функции:

```
int             GetFieldCount ( int64_t connecth )
char *          GetFieldNameUTF8 ( int64_t connecth, int fieldnum, int * len )
wchar_t *       GetFieldNameUTF16 ( int64_t connecth, int fieldnum, int * len )
CLIENT_DATA_TYPE GetFieldType ( int64_t connecth, int fieldnum )
```

Пример:

```
int fcount = GetFieldCount ( connecth );
for ( int i = 0; i < fcount; i++ )
{
        int len;
        char * fieldname = GetFieldName (connecth, i, &len );
        CLIENT_DATA_TYPE fieldtype = GetFieldType (connecth, i );
        //… do something
```

```
}
```

Получение результата запроса

Обычно после выполнения запроса выполняется итерация по всем полученным записям. В следующем примере показано суммирование всех значений третьего поля (обратите внимание, что индексирование поля начинается с 0):

```
int sum = 0;
while ( ReadRecord ( connecth ) )
{
  int * v = GetFieldInt ( connecth, 2 );
  if ( v != NULL )
    sum += *v;
}
```

После вызова ReadRecord для получения значений полей для текущей записи можно использовать следующие функции:

```
int *        GetFieldInt ( int64_t connecth, int fieldnum )
int *        GetFieldInt64 ( int64_t connecth, int fieldnum ) // для получения значений sum(?x)
double *     GetFieldDouble ( int64_t connecth, int fieldnum )
char *       GetFieldDateTimeUTF8 ( int64_t connecth, int fieldnum,   int * len )
wchar_t *    GetFieldDateTimeUTF16 ( int64_t connecth, int fieldnum, int * len )
char *       GetFieldStringUTF8 ( int64_t connecth, int fieldnum,     int * len )
wchar_t *    GetFieldStringUTF16 ( int64_t connecth, int fieldnum,  int * len )
```

Пример получения и вывода данных

В следующем примере выполняется запрос SPARQL и его результаты выводятся на консоль, чтобы продемонстрировать использование функций из этой главы:

```
NBConnectUTF8
ExecuteQueryUTF8
ExecutePagingQueryUTF8
ReadRecord
GetFieldCount
GetFieldNameUTF8
GetFieldType
GetFieldInt
GetFieldInt64
GetFieldDouble
GetFieldDateTimeUTF8
GetFieldStringUTF8
NBCloseConnect
try
{
      string querytext = "select ?x ?y ?z where{ ?x ?y ?z. }";
      ExecuteQueryUTF8 ( connecth, querytext.c_str(), querytext.length(), QUERY_SPARQL );
      while ( ReadRecord ( connecth ) )
      {
            int fcount = GetFieldCount ( connecth );
            for ( int i = 0; i < fcount; i++ )
            {
```

```
                char * fname = GetFieldNameUTF8 ( connecth, i, NULL );
                switch ( GetFieldType ( connecth, i ) )
                {
                    case CLIENT_DATA_INT: {
                        int * v = GetFieldInt ( connecth, i );
                        if(v)
                            cout << fname << " : " << *v << endl;
                    }
                    break;
                    case CLIENT_DATA_INT64: {
                        int64_t * v = GetFieldInt64 ( connecth, i );
                        if(v)
                            cout << fname << " : " << *v << endl;
                    }
                    break;
                    case CLIENT_DATA_DOUBLE: {
                        double * v = GetFieldDouble ( connecth, i );
                        if(v)
                            cout << fname << " : " << *v << endl;
                    }
                    break;
                    case CLIENT_DATA_DATETIME: {
                        char * v = GetFieldDateTimeUTF8 ( connecth, i, NULL );
                        if(v)
                            cout << fname << " : " << v << endl;
                    }
                    break;
                    case CLIENT_DATA_URI:
                    case CLIENT_DATA_STRING: {
                        char * v = GetFieldStringUTF8 ( connecth, i, NULL );
                        if(v)
                            cout << fname << " : " << v << endl;
                    }
                    break;
                }
            }
        }
        NBCloseConnect ( connecth );
}
catch ( const char *error )
{
    cout << "Error : " << error << endl;
}
```

Список функций

Функция	Назначение
NBConnectToDbUTF8 NBConnectToDbUTF16	Подключиться к базе данных с именем dbname и вернуть дескриптор соединения.
NBConnectOpen	Открыть соединение с базой данных.
NBCloseConnect	Закрыть соединение с базой данных.
ExecuteQueryUTF8 ExecuteQueryUTF16	Выполнить запрос к базе данных.
ExecutePagingQueryUTF8 ExecutePagingQueryUTF16	Выполнить постраничный запрос к базе данных.
ReadRecord	Прочитать следующую запись из результата
GetFieldCount	Вернуть количество полей в результате
GetFieldNameUTF8 GetFieldNameUTF16	Вернуть имя требуемого поля в текущем результате.
GetFieldType	Вернуть тип требуемого поля в текущем результате.
GetFieldInt	Вернуть значение поля int в текущем результате.
GetFieldInt64	Вернуть значение поля int64 в текущем результате.
GetFieldDouble	Вернуть значение поля double в текущем результате.
GetFieldDateTimeUTF8 GetFieldDateTimeUTF16	Вернуть значение поля datetime в текущем результате
GetFieldStringUTF8 GetFieldStringUTF16	Вернуть значение поля string в текущем результате.

Справочник функций

NBGetDatabaseUTF8

Возвращает дескриптор базы данных с именем dbname

Вызов:

int64_t NBGetDatabaseUTF8 (const char * dbname, int len, const char * dbfolder, const char * nbserverfolder)

Параметры:

dbname – имя базы данных в 8-битной кодировке.

len – длина имени базы данных.

dbfolder – Путь к папке базы данных.

nbserverfolder – путь к папке модуля сервера базы данных nitrosbase (и dll).

Пример:

```
int64_t dbh = NBGetDatabaseUTF8 ( "person", 6 , "c:\\NitrosBaseRDF\\data\\person\\", "c:\\NitrosBaseRDF" );
```

NBGetDatabaseUTF16

Returns a handle for the database named dbname

Вызов:

int64_t NBGetDatabaseUTF16 (const wchar_t * dbname, int len, const wchar_t * dbfolder, const wchar_t * nbserverfolder)

Параметры:

dbname – имя базы данных в 16-битной кодировке.

len – длина имени базы данных.

dbfolder – путь к папке базы данных.

nbserverfolder – путь к папке модуля сервера базы данных nitrosbase (и dll).

Пример:

```
int64_t dbh = NBGetDatabaseUTF16 ( L"person", 6 , L"c:\\NitrosBaseRDF\\data\\person\\", L"c:\\NitrosBaseRDF" );
```

NBConnect

Подключается к базе данных и возвращает дескриптор соединения

Вызов:

Int64_t NBConnect (int64_t databasehandle)

Параметры:

databasehandle – дескриптор базы данных, полученный с помощью NBGetDatabaseUTF8 или NBGetDatabaseUTF16.

Возвращаемое значение:

дескриптор подключения к базе данных. А Возвращаемое значение of 0 indicates a connection error.

Пример:

```
int64_t dbh = NBGetDatabaseUTF8 ( "person", 6 , "c:\\NitrosBaseRDF\\data\\person\\",
"c:\\NitrosBaseRDF" );
int64_t connecth = NBConnect ( dbh );
```

NBCloseConnect

Закрывает соединение с базой данных

Вызов:

void NBCloseConnect (int64_t connect, bool hardclose = false)

Параметры:

Connect – дескриптор подключения к базе данных.

hardclose – режим закрытия: true просто закроет соединение, false освободит его ресурсы и поместит его в пул соединений, где его можно будет повторно использовать при последующем вызове NBConnect

Пример:

```
NBCloseConnect ( connecth );
```

ExecuteQueryUTF8

Выполняет запрос к базе данных.

Вызов:

void ExecuteQueryUTF8 (int64_t connecth, const char * query, int length,
CLIENT_QUERY_TYPE qtype = QUERY_SPARQL)

Параметры:

connecth – дескриптор подключения к базе данных;

query – 8bit-encoded string that, depending on the type, consists either of query text, a set of triples, file name, and so on;

length – длина запроса;

qtype – query type, one of the following:

```
enum CLIENT_QUERY_TYPE
{
	QUERY_SPARQL,
	QUERY_INSERTTRIPLES,
	QUERY_DELETETRIPLES,
	QUERY_INSERTTRIPLES_FILE,
	QUERY_DELETETRIPLES_FILE,
	QUERY_DELETERECORDS_QUERY,
	QUERY_DELETERECORDS_URILIST,
	QUERY_SAVE2TURTLEN3
};
```

Пример:

```
int64_t dbh = NBGetDatabaseUTF8 ( "person", 6 , "c:\\NitrosBaseRDF\\data\\person\\",
"c:\\NitrosBaseRDF" );
int64_t connecth = NBConnect ( dbh );
string querytext = "SELECT ?uri ?Male ?ID WHERE { ?uri ID ?ID. ?uri Male ?Male.}";
ExecuteQueryUTF8 ( connecth, querytext.c_str(), querytext.length(), QUERY_SPARQL );
NBCloseConnect ( connecth );
```

ExecuteQueryUTF16

Выполняет запрос к базе данных.

Вызов:

void ExecuteQueryUTF16 (int64_t connecth, const wchar_t * query, int length, CLIENT_QUERY_TYPE qtype = QUERY_SPARQL)

Параметры:

connecth – дескриптор подключения к базе данных;

query – 16-битная строка, которая, в зависимости от типа, состоит из текста запроса, набора троек, имени файла и т. д.;

length – длина запроса;

qtype – тип запроса, один из следующих:

```
enum CLIENT_QUERY_TYPE
{
	QUERY_SPARQL,
	QUERY_INSERTTRIPLES,
	QUERY_DELETETRIPLES,
	QUERY_INSERTTRIPLES_FILE,
	QUERY_DELETETRIPLES_FILE,
	QUERY_DELETERECORDS_QUERY,
	QUERY_DELETERECORDS_URILIST,
	QUERY_SAVE2TURTLEN3
```

```
};
```

ExecutePagingQueryUTF8

Выполняет запрос к базе данных с разбиением на страницы.

Вызов:

int ExecutePagingQueryUTF8 (int64_t connecth, const char * query, int length, int from, int to, int afterto)

Параметры:

connecth – дескриптор подключения к базе данных.

query – 8-битная строка

length – длина запроса.

from – первая запись страницы.

to – последняя запись страницы.

afterto – одно из следующих значений:

afterto = MAX_INT — запросить точное количество записей, возвращаемых этим запросом;

afterto > 0 — запросить количество записей после окончания страницы;

afterto = 0 – игнорировать.

Пример:

```
int64_t dbh = NBGetDatabaseUTF8 ( "person", 6 , "c:\\NitrosBaseRDF\\data\\person\\",
"c:\\NitrosBaseRDF" );
int64_t connecth = NBConnect ( dbh );
string querytext = "SELECT ?uri ?Male ?ID WHERE { ?uri ID ?ID. ?uri Male ?Male.}";
int count = ExecutePagingQueryUTF8 ( connecth, querytext.c_str(), querytext.length(), 0, 99, 1 );
NBCloseConnect ( connecth );
```

В этом примере нас интересует страница записей, начинающаяся с 0 и заканчивающаяся на 99, и общее количество записей после этой страницы, если таковые имеются.

ExecutePagingQueryUTF16

Выполняет запрос к базе данных с разбиением на страницы.

Вызов:

int ExecutePagingQueryUTF16 (int64_t connecth, const wchar_t * query, int length, int from, int to, int afterto)

Параметры:

connecth – дескриптор подключения к базе данных;

query – 16-битная строка;

length – длина запроса;

from – первая запись страницы;

to – последняя запись страницы;

afterto – одно из следующих значений

afterto = MAX_INT – запросить точное количество записей, возвращаемых этим запросом;

afterto > 0 – запросить количество записей после окончания страницы;

afterto = 0 – игнорировать.

ReadRecord

Читает следующую запись результата запроса

Вызов:

bool ReadRecord (int64_t connecth)

Параметры:

connecth – дескриптор подключения к базе данных.

Возвращаемое значение:

истина, если есть еще строки, ложь в противном случае.

Пример:

```
int sum = 0;
while ( ReadRecord ( connecth ) )
{
   int * v = GetFieldInt ( connecth, 2 );
   if ( v != NULL )
      sum += *v;
}
```

GetFieldCount

Возвращает количество полей в текущем результате.

Вызов:

int GetFieldCount (int64_t connecth)

Параметры:

connecth – дескриптор подключения к базе данных.

Пример:

```
Int fcount = GetFieldCount ( connecth );
```

GetFieldNameUTF8

Возвращает 8-битное имя требуемого поля в текущем результате.

Вызов:

char * GetFieldNameUTF8 (int64_t connecth, int fieldnum, int * len)

Параметры:

connecth – дескриптор подключения к базе данных;

fieldnum – индекс поля;

len – указатель на int для хранения длины возвращаемого имени; может иметь значение NULL, если длина не требуется.

Пример:

```
int len;
char * fname = GetFieldNameUTF8 ( connecth, i, &len );
```

GetFieldNameUTF16

Возвращает 16-битное имя назначенного поля в текущем наборе результатов.

Вызов:

wchar_t * GetFieldNameUTF16 (int64_t connecth, int fieldnum, int * len)

Параметры:

connecth – дескриптор подключения к базе данных;

fieldnum – индекс поля;

len – указатель на int для хранения длины возвращаемого имени; может иметь значение NULL, если длина не требуется.

Пример:

```
int len;
wchar_t * fname = GetFieldNameUTF16 ( connecth, i, &len );
```

GetFieldType

Возвращает тип указанного поля в текущем результате как одно из следующих значений:

```
enum CLIENT_DATA_TYPE
{
        CLIENT_DATA_STRING,
        CLIENT_DATA_INT,
        CLIENT_DATA_INT64,
        CLIENT_DATA_DOUBLE,
        CLIENT_DATA_DATETIME,
        CLIENT_DATA_URI,
        CLIENT_DATA_VARTYPE
};
```

Вызов

CLIENT_DATA_TYPE GetFieldType(int64_t connecth, int fieldnum)

Параметры:

connecth – дескриптор подключения к базе данных;

fieldnum – индекс поля;

Пример:

```
CLIENT_DATA_TYPE fieldtype = GetFieldType ( connecth, fieldnum );
```

GetFieldInt

Возвращает значение поля int в текущем результате.

Вызов:

Int* GetFieldInt (int64_t connecth, int fieldnum)

Параметры:

connecth – дескриптор подключения к базе данных;

fieldnum – индекс поля.

Возвращаемое значение:

Returns a pointer to an int if field is not NULL; returns NULL otherwise.

Пример:

```
int * fieldvalue = GetFieldInt ( connecth, fieldnum );
```

GetFieldInt64

Возвращает значение поля int64_t. Тип int64_t не поддерживается как тип в базе данных, но используется для значений агрегатной функции суммы в запросах SPARQL.

Вызов:

int64_t* GetFieldInt64 (int64_t connecth, int fieldnum)

Параметры:

connecth – дескриптор подключения к базе данных;

fieldnum – индекс поля.

Возвращаемое значение:

Возвращает указатель на int64_t, если поле не равно NULL; в противном случае возвращает NULL.

Пример:

```
int64_t * fieldvalue = GetFieldInt64 ( connecth, fieldnum );
```

GetFieldDouble

Возвращает значение поля типа double. Тип double поддерживается как тип в базе данных, а также используется для значений агрегатной функции avg в запросах SPARQL.

Вызов:

double* GetFieldDouble (int64_t connecth, int fieldnum)

Параметры:

connecth – дескриптор подключения к базе данных;

fieldnum – индекс поля.

Возвращаемое значение:

Возвращает указатель на двойное значение, если поле не равно NULL; в противном случае возвращает NULL.

Пример:

```
double * fieldvalue = GetFieldInt64 ( connecth, fieldnum );
```

GetFieldDateTimeUTF8

Возвращает значение поля типа datetime.

Вызов

char * GetFieldDateTimeUTF8 (int64_t connecth, int fieldnum, int * len)

Параметры:

connecth – дескриптор подключения к базе данных;

fieldnum – индекс поля;

len – указатель на int для хранения длины возвращаемого значения; может иметь значение NULL, если длина не требуется.

Возвращаемое значение:

Возвращает указатель на 8-битную строку, если поле не NULL; иначе возвращает NULL..

Пример:

```
int len;
char * fieldvalue = GetFieldDateTimeUTF8 ( connecth, fieldnum, &len );
```

GetFieldDateTimeUTF16

Возвращает значение поля типа datetime.

Вызов:

wchar_t * GetFieldDatetimeUTF16 (int64_t connecth, int fieldnum, int* len)

Параметры:

connecth – дескриптор подключения к базе данных;

fieldnum – индекс поля;

len – указатель на int для хранения длины возвращаемого значения; может иметь значение NULL, если длина не требуется.

Возвращаемое значение:

Возвращает указатель на строку с 16-битной кодировкой, если поле не равно NULL; в противном случае возвращает NULL.

Пример:

```
int len;
wchar * fieldvalue = GetFieldDateTimeUTF16 ( connecth, fieldnum, &len );
```

GetFieldStringUTF8

Возвращает значение поля строкового типа.

Вызов:

char * GetFieldStringeUTF8 (int64_t connecth, int fieldnum, int* len)

Параметры:

connecth – дескриптор подключения к базе данных;

fieldnum – индекс поля;

len – указатель на int для хранения длины возвращаемого значения; может иметь значение NULL, если длина не требуется.

Возвращаемое значение:

Возвращает указатель на 8-битную строку, если поле не NULL; иначе возвращает NULL.

Пример:

```
int len;
char * fieldvalue = GetFieldStringUTF8 ( connecth, fieldnum, &len );
```

GetFieldStringUTF16

Возвращает значение поля строкового типа.

Вызов:

wchar_t * GetFieldStringUTF16 (int64_t connecth, int fieldnum, int* len)

Параметры:

connecth – дескриптор подключения к базе данных;

fieldnum – индекс поля;

len – указатель на int для хранения длины возвращаемого значения; может иметь значение NULL, если длина не требуется.

Возвращаемое значение:

Возвращает указатель на 16-битную строку, если поле не NULL; иначе возвращает NULL.

Пример:

```
int len;
wchar_t * fieldvalue = GetFieldStringUTF16 ( connecth, fieldnum, &len );
```

Примеры приложений

В папке C:\NitrosBaseRDF\Samples (для Windows) или в папке samples (для Linux) находятся примеры приложений, использующих API NitrosBase RDF Storage:

Sample1 – пример простейшего приложения. Консольное приложение соединяется с базой *person*, выполняет простой SPARQL запрос, и выводит результат запроса в консоль.

Sample2 – пример модификации данных (добавление, удаление и изменение данных).

Sample3 – пример приложения для демонстрации разбиения на страницы. Консольное приложение, которое подключается к базе данных *person*, выполняет простой запрос SPARQL и распечатывает результаты на нескольких страницах.