NITROSBASE SQL V2

Программный интерфейс C/C++
ID:NBSQL2-CPP-1728

ООО «НитросДэйта Рус»
2022

Оглавление

НИЗКОУРОВНЕВЫЙ C/C++ API

Данный документ описывает работу программиста C/C++ с базой данных NitrosBase через низкоуровневый API. Основные шаги:

- Соединение с сервером базы данных
- Посылка запроса на сервер
- Получение данных от сервера
- Отключение от сервера

Дополнительные разделы документации:

- Множественные результаты запросов (multiple result sets)
- Чтение больших строк и BLOB полей
- Подготовленные (параметризованные) запросы
- Режим работы bulk insert
- Обработка ошибок
- Получение информации о схеме базы данных

ФАЙЛОВАЯ СТРУКТУРА УСТАНОВЛЕННОЙ СИСТЕМЫ

После установки NitrosBase на сервер, файлы, необходимые для работы из языковых сред находятся в следующих каталогах:

- *bin* содержит файлы *nbserver.exe* – файл сервера *NitrosBase*, *nbclient.dll* и *nbclient.lib* библиотеки для обеспечения работы приложений через API.
- *Include* содержит *nitrosbase.h* – заголовочный файл C/C++.
- samples.c++ содержит файлы примеров C/C++. Часть из них цитируется в данном документе.
- *data* содержит демонстрационную базу данных, к которой обращаются приложения C/C++ в процессе своей работы.
- *samples.data* содержит backup демонстрационной базы данных.

ПРОСТОЙ СЦЕНАРИЙ РАБОТЫ С СУБД NITROSBASE

Исходный код данного примера содержится в каталоге *samples.c++/simple-query*.

```
NB_HANDLE connection = nb_connect( u"127.0.0.1", 3020, u"TESTUSER", u"1234" );
check_error( connection );

nb_execute_sql( connection, u"SELECT TOP 10 * FROM person" );
check_error( connection );

while ( nb_fetch_row( connection ) == NB_OK )
{
    std::cout << "\n-----------------------------\n\n";
```

```
    NBValue name, v;
    int fieldcount = nb_field_count( connection );
    for ( int i = 0; i < fieldcount; i++ )
    {
        nb_field_name_utf8( connection, i, &name );
        nb_field_value_utf8( connection, i, &v );
        std::cout << name << ": " << v << std::endl;
    }
}
```

Многие примеры используют функции с окончанием _utf8 для удобства вывода в терминал в Linux и Windows. Основные функции NitrosBase API предполагают работу со строками в формате UTF-16. Но также имеются дублирующие функции с окончанием _utf8. Например, функция *nb_field_name*() работает со строками в формате UTF-16, а функция *nb_field_name_utf8()*, работает со строками в формате UTF-8. Эти функции осуществляют перевод строковых параметров из UTF-8 в UTF-16, а также перевод из UTF-16 в UTF-8 для возвращаемых значений. Поскольку внутренний формат строк в NitrosBase UTF-16, стоит отдавать предпочтение функциям, принимающим и возвращающим значения в формате UTF-16 (чтобы избежать лишней траты времени на перекодирование строк).

Соединение с сервером базы данных

Клиент открывает соединение с базой данных используя функцию *nb_connect*. При успешном соединении с сервером возвращается дескриптор соединения, однозначно идентифицирующий клиентскую сессию. Этот дескриптор в дальнейшем используется во всех вызовах API.

Для соединения с базой данных необходимо указать IP адрес, номер порта, имя пользователя и пароль. Каждая база NitrosBase работает как отдельный сервер и имеет уникальное сочетание IP адреса и номера порта. Если на одном сервере работают несколько баз данных, то они должны иметь разные номера портов.

Соединение с базой данных позволяет последовательно выполнять SQL и Graph-SQL запросы и получать результаты их работы. Для параллельного выполнения нескольких запросов и параллельного чтения их результатов, необходимо получить дополнительные соединения.

Получение дополнительного соединения

Для получения дополнительных соединений клиент использует функцию nb_clone_connect (функция не требует повторной передачи имени пользователя и пароля и работает быстрее).

Закрытие соединения с сервером базы данных

Функция nb_disconnect закрывает соединение и освобождает ресурсы. Данные функции не используют пул соединений. Пул соединений может быть реализован поверх этих функций.

Пример: открытие и закрытие соединения с базой данных

```
NB_HANDLE connection = nb_connect( u"127.0.0.1", 3020, u"TESTUSER", u"1234" );
check_error( connection );

nb_execute_sql( connection, u"SELECT * FROM person", 20 );
check_error( connection );
```

```
// ... (обработка результата запроса описана ниже)

nb_disconnect( connection );
```

Пример: получение дополнительного соединения с базой данных

```
NB_HANDLE newconnection = nb_clone_connect( connection );
check_error( connection );

nb_execute_sql( newconnection, u"SELECT * FROM person", 20 );
check_error( connection );

// ... (process query result (see below))

nb_disconnect( newconnection );
```

nb_connect

```
NB_HANDLE nb_connect (
    const char16_t * host,
    int sockport,
    const char16_t * user,
    const char16_t * password
);
```

Используется для первого соединения с сервером NitrosBase.

Параметры:

- host - имя узла сервера NitrosBase
- sockport - TCP-порт сервера NitrosBase
- user - пользователь сервера NitrosBase
- password - пароль пользователя сервера NitrosBase

Возвращаемое значение:

- в случае успеха - дескриптор соединения
- в случае неудачи - NB_INVALID_HANDLE

nb_clone_connect

```
NB_HANDLE nb_clone_connect ( NB_HANDLE connection );
```

Используется для быстрого получения дополнительных соединений.

Параметры:

- connection - дескриптор соединения, полученный в результате вызова функции nb_connect

Возвращаемое значение:

- в случае успеха - дескриптор дополнительного соединения
- в случае неудачи - NB_INVALID_HANDLE

nb_disconnect

```
NB_ERROR_TYPE nb_disconnect ( NB_HANDLE connection );
```

Закрывает соединение и освобождает ресурсы.

Параметры:

- connection - дескриптор соединения

Возвращаемое значение:

- в случае успеха - NB_OK
- в случае неудачи - код ошибки

ВЫПОЛНЕНИЕ ЗАПРОСОВ

Основная функция для выполнения запросов - nb_execute_sql. Она посылает запрос на сервер и дожидается ответа.

После каждого запроса можно получить дополнительную информацию о результате с помощью функции nb_get_change_info (в случае асинхронного запроса это можно сделать после получения ответа). Для всех запросов функция nb_get_change_info возвращает тип запроса. Для запросов insert/update/delete функция nb_get_change_info возвращает количество модифицированных записей.

Большая часть запросов возвращает результат в виде набора записей (например, SELECT). Функции чтения записей, получение имен и значений полей и т.д. описаны в разделе чтение-записей.

Пример: выполнение запроса

```
int type, count;

nb_execute_sql( connection, u"INSERT INTO t1 (1,2,3)", 22 );
check_error( connection );

nb_get_change_info( connection, &type, &count );
```

nb_execute_sql

```
NB_ERROR_TYPE nb_execute_sql (
    NB_HANDLE connection,
    const char16_t * query,
    size_t length
);
```

Выполняет SQL-запрос на сервере.

Параметры:

- connection — дескриптор соединения с сервером
- query — текст запроса
- length — длина текста запроса

Возвращаемое значение:

- в случае успеха - NB_OK
- в случае неудачи - код ошибки

nb_get_change_info

Функция позволяет получить информацию об изменениях после выполнения запроса.

```
NB_ERROR_TYPE nb_get_change_info (
    NB_HANDLE connection,
    int *querytype,
    int *changecount
);
```

Функция устанавливает параметры `querytype` - тип запроса и `changecount` - количество модифицированных записей.

Параметры:

- connection — дескриптор соединения
- querytype — параметр, получающий указатель на переменную, содержащую тип запроса (см. enum *SQL_QUERY*_TYPE ниже в этом разделе)
- changecount — параметр, получающий указатель на переменную, содержащую количество модифицированных записей для запросов *insert*/*update*/*delete*

Возвращаемое значение:

- в случае успеха - NB_OK
- в случае неудачи - код ошибки

Тип запроса:

```
enum SQL_QUERY_TYPE
{
    SQL_QUERY_NONE    = 0,
    SQL_QUERY_SELECT  = 1,
    SQL_QUERY_INSERT  = 2,
    SQL_QUERY_UPDATE  = 3,
    SQL_QUERY_DELETE  = 4,
    SQL_QUERY_TRANSACTION  = 5,
    SQL_QUERY_ANOTHER  = 6
};
```

Пример

```
int main()
{
    // Variables to set in nb_get_change_info()
    int querytype = 0, changecount = 0;

    NB_HANDLE connection = nb_connect(u"127.0.0.1", 3020, u"TESTUSER", u"1234");

    nb_execute_sql(connection, u"SELECT TOP 5 dbl FROM person");

    nb_get_change_info(connection, &querytype, &changecount);
    std::cout << "\n  querytype: " << str_query_type(querytype);
    std::cout << "\nchangecount: " << changecount << " records affected\n";

    nb_execute_sql(connection, u"UPDATE person set dbl = 0 WHERE name = 'Vanda'");

    nb_get_change_info(connection, &querytype, &changecount);
    std::cout << "\n  querytype: " << str_query_type(querytype);
    std::cout << "\nchangecount: " << changecount << " records affected\n";

    nb_disconnect(connection);
    return 0;
}
```

Исходный код примера содержится в каталоге *samples.c++/get-change-info*

ЧТЕНИЕ РЕЗУЛЬТАТА

Результат работы SELECT запроса можно интерпретировать как множество записей с набором полей. Имена полей и их типы идентичны для всех записей. Для получения очередной записи из результирующего множества используется функция nb_fetch_row.

Для работы с полями записи используются следующие функции:

- nb_field_value - получение конкретного значения поля текущей записи

- nb_field_count - получение количества полей
- nb_field_name - получение имени поля по номеру поля
- nb_field_type - получение типа поля по номеру поля (см. enum NB_DATA_TYPE)

nb_fetch_row

```
NB_ERROR_TYPE nb_fetch_row ( NB_HANDLE connection );
```

Получение следующей записи из результирующего набора данных, возвращенного сервером в ответ на запрос.

Параметры:

- connection — дескриптор соединения с сервером

Возвращаемое значение:

- в случае успеха - NB_OK
- в случае неудачи - код ошибки

nb_field_value

```
NB_ERROR_TYPE nb_field_value (
    NB_HANDLE connection,
    int fieldnum,
    NBValue * v
);
```

Возвращает значение поля в записи, полученной последним вызовом nb_fetch_row.

Параметры:

- connection — дескриптор соединения с сервером
- fieldnum — номер поля, начиная с 0
- v — указатель на структуру NBValue, в которую будет передан результат (см. описание функции nb_field_name)

Структура NBValue определена следующим образом:

```
struct NBValue
{
    NB_DATA_TYPE type;
    union
    {
        int32_t intv;
        int64_t int64v;
        double dbl;
        struct
        {
```

```
            char *str;
            int32_t len;
            uint64_t blobnextpart;
            int blobfield;
        };
    };
    bool null;
};
```

Возвращаемое значение:

- в случае успеха - NB_OK (кроме того, функция заполняет структуру NBValue, на которую ссылается указатель v)
- в случае неудачи - код ошибки

nb_field_count

```
int nb_field_count ( NB_HANDLE connection );
```

Возвращает число полей в записи, полученной с помощью последнего вызова nb_fetch_row.

Параметры:

- connection — дескриптор соединения с сервером

Возвращаемое значение:

- в случае успеха - число полей
- в случае неудачи - -1

nb_field_name

```
NB_ERROR_TYPE nb_field_name (
    NB_HANDLE connection,
    int fieldnum, NBValue * name
);
```

Устанавливает параметр *name* - имя поля в записи, полученной последним вызовом nb_fetch_row.

Параметры:

- connection — дескриптор соединения с сервером
- fieldnum — номер поля, начиная с 0
- name — указатель на структуру NBValue, в которую будет передан результат (см. описание функции nb_field_value)

Возвращаемое значение:

- в случае успеха - NB_OK (кроме того, функция заполняет структуру NBValue, на которую ссылается указатель name)
- в случае неудачи - код ошибки

nb_field_type

```
NB_DATA_TYPE nb_field_type (
    NB_HANDLE connection,
    int fieldnum
);
```

Возвращает тип поля в записи, полученной последним вызовом nb_fetch_row.

Параметры:

- connection — дескриптор соединения с сервером
- fieldnum — номер поля, начиная с 0

Возвращаемое значение:

- В случае успеха - одно из следующих значений (см. enum MB_DATA_TYPE):
  MB_DATA_STRING,
  MB_DATA_INT,
  MB_DATA_INT64,
  MB_DATA_DOUBLE,
  MB_DATA_DATETIME,
  MB_DATA_BOOL,
  MB_DATA_DATE,
  MB_DATA_URI,
  MB_DATA_BINARY,
  MB_DATA_U16STRING,
  MB_DATA_ROWVERSION,
  MB_DATA_DECIMAL,
- В случае неудачи - NB_DATA_NONE

Пример: выполнение запроса и чтение данных

```
NB_HANDLE connection = nb_connect( u"127.0.0.1", 3020, u"TESTUSER", u"1234" );
check_error( connection );

nb_execute_sql( connection, u"SELECT TOP 10 * FROM person" );
check_error( connection );

while ( nb_fetch_row( connection ) == NB_OK )
{
    std::cout << "\n-----------------------------\n\n";
```

```
    NBValue name, v;
    int fieldcount = nb_field_count( connection );
    for ( int i = 0; i < fieldcount; i++ )
    {
        nb_field_name_utf8( connection, i, &name );
        nb_field_value_utf8( connection, i, &v );
        std::cout << name << ": " << v << std::endl;
    }
}
check_error( connection );

nb_disconnect( connection );
```

Исходный код примера содержится в каталоге *samples.c++/simple-query*

МНОЖЕСТВЕННЫЕ РЕЗУЛЬТАТЫ ЗАПРОСОВ

Строка запроса для функции nb_execute_sql может содержать несколько SQL запросов. Для получения ответов на множество SQL запросов используется функция nb_nextresult. Обработав результат одного запроса, нужно вызвать функцию nb_nextresult, которая возвращает true, если еще есть результаты.

nb_nextresult

```
bool nb_nextresult( NB_HANDLE connection )
```

Проверяет есть ли результат следующего запроса.

Параметры:

- connection — дескриптор соединения с сервером

Возвращаемое значение:

- true - если есть результат следующего запроса
- false - больше нет результатов

Пример: получение результатов от нескольких запросов

```
NB_HANDLE connection = nb_connect( u"127.0.0.1", 3020, u"TESTUSER", u"1234" );
check_error( connection );

nb_execute_sql( connection, uR"(
    SELECT TOP 10 * FROM person
    BEGIN TRAN
    UPDATE person SET age = age + 1 WHERE id = 'person1'
    ROLLBACK TRAN
    )" );
```

```
for ( int i = 0;; i++ )
{
    std::cout << "\nQuery part " << i << std::endl;
    check_error( connection );
    int type, count;
    nb_get_change_info( connection, &type, &count );

    if ( type == SQL_QUERY_SELECT )
    {
        while ( nb_fetch_row( connection ) == NB_OK )
        {
            std::cout << "\n-----------------------------\n\n";

            NBValue name, v;
            int fieldcount = nb_field_count( connection );
            for ( int i = 0; i < fieldcount; i++ )
            {
                nb_field_name_utf8( connection, i, &name );
                nb_field_value_utf8( connection, i, &v );
                std::cout << name << ": " << v << std::endl;
            }
        }
    }
    else if( type == SQL_QUERY_INSERT
          || type == SQL_QUERY_UPDATE
          || type == SQL_QUERY_DELETE )
    {
        std::cout << "Affected rows: " << count << std::endl;
    }
    if ( !nb_nextresult( connection ) )
        break;
};
```

Исходный код примера содержится в каталоге samples.c++/multiple-result-sets

ЧТЕНИЕ БОЛЬШИХ СТРОК И BLOB ПОЛЕЙ

При обработке результата запроса, значения полей текущей записи читаются с помощью функции nb_field_value, которая заполняет структуру NBValue. Структура NBValue содержит тип поля, его значение для разных типов и пометку null, для отображения SQL NULL значений.

Для строковых полей и полей типа VARBINARY структура NBValue содержит:

- str - указатель на строку
- len - длина строки в байтах
- blobnextpart - дополнительная информация для получения длинных строк.
  Если blobnextpart == 0 то вернулась обычная строка, иначе строка может быть получена порциями с помощью функции nb_blob_nextpart

nb_blob_nextpart

```
NB_ERROR_TYPE nb_blob_nextpart(
    NB_HANDLE connection,
    NBValue * value
);
```

Получает очередную часть строки. Если функция возвращает NB_OK, то value->str содержит указатель на часть строки и value->len содержит ее длину в байтах.

Параметры:

- connection - дескриптор соединения
- value - указатель на структуру NBValue, полученную после вызова функции *nb_field_value (см. описание функции nb_field_value)

Возвращаемое значение:

- NB_OK - получен очередная часть длинной строки
- NB_NO_DATA - больше нет частей (на предыдущем шаге получена последняя часть строки)
- код ошибки - в случае ошибки

Пример: чтение большой строки

```
NB_HANDLE connection = nb_connect( u"127.0.0.1", 3021, u"TESTUSER", u"1234" );
check_error( connection );

prepare_testdb( connection );

nb_execute_sql( connection, u"SELECT TOP 10 doctext FROM documents" );
check_error( connection );

while ( nb_fetch_row( connection ) == NB_OK )
{
    NBValue v;
    nb_field_value_utf8( connection, 1, &v );
    if ( v.null )
        std::cout << "NULL";
    else if ( v.blobnextpart == 0 )
        std::cout.write( v.str, v.len );
    else while ( nb_blob_nextpart_utf8( connection, &v ) == NB_OK )
    {
        std::cout.write( v.str, v.len );
    }
    std::cout << "\n----------------------------\n\n";
}
check_error( connection );

nb_disconnect( connection );
```

ПОДГОТОВЛЕННЫЕ ЗАПРОСЫ

Подготовленные (параметризованные) запросы могут использоваться для оптимизации быстродействия. Запрос подготавливается с помощью функции nb_prepare_query, которая подготавливает запрос и возвращает handle запроса. Строка запроса, передаваемая в функцию nb_prepare_query, может содержать вопросительные знаки, вместо которых при выполнении запроса будут подставлены конкретные значения параметров

Заранее подготовленный запрос выполняется с помощью нескольких этапов (вызовов функций).

1. Вызовом nb_start_execute_prep - начинаем посылку запроса на сервер.
2. С помощью функций nb_write_param_int, nb_write_param_str и т.д. пишем аргументы функции (значения параметров).
3. С помощью функции nb_execute_prep инициируем выполнение запроса на сервере

Длинные строки можно писать частями

- nb_start_param_blob - начинает запись длинной строки
- nb_write_blob_part - пишет очередную часть строки
- nb_end_param_blob - завершает запись длинной строки

Если подготовленный функцией nb_prepare_query запрос более не нужен, то надо освободить ресурсы с помощью функции nb_close_prepared_query.

nb_prepare_query

```
NB_ERROR_TYPE nb_prepare_query(
    NB_HANDLE connection,
    const char16_t  *querystring,
    size_t length,
    NB_HANDLE * queryh
);
```

Подготавливает параметризованный запрос.

Параметры:

- connection - дескриптор соединения
- querystring - строка запроса, содержащая знаки вопросов на месте параметров, в формате utf-16
- length - длина строки запроса в количестве двухбайтовых знаков
- queryh - функция возвращает handle запроса по указателю

Возвращаемое значение:

- в случае успеха - NB_OK
- в случае неудачи - код ошибки

nb_execute_prep

```
NB_ERROR_TYPE nb_execute_prep( NB_HANDLE connection );
```

Инициирует выполнение запроса на сервере.

Параметры:

- connection - дескриптор соединения

Возвращаемое значение:

- в случае успеха - NB_OK
- в случае неудачи - код ошибки

nb_write_param_...

```
NB_ERROR_TYPE nb_write_param_int( NB_HANDLE connection, int value );
NB_ERROR_TYPE nb_write_param_int64( NB_HANDLE connection, int64_t value );
NB_ERROR_TYPE nb_write_param_double( NB_HANDLE connection, double value );
NB_ERROR_TYPE nb_write_param_str( NB_HANDLE connection, char16_t * str, int len );
NB_ERROR_TYPE nb_write_param_str_utf8( NB_HANDLE connection, char * str8, int len );
NB_ERROR_TYPE nb_write_param_binary( NB_HANDLE connection, void * ptr, int len );
```

Запись значений параметров запроса.

Параметры:

- connection - дескриптор соединения
- value - значение параметра простого типа int, int64_t, double
- str, len - указатель на строку в формате utf-16 и длина строки
- str8, len - указатель на строку в формате utf-8 и длина строки
- ptr, len - указатель на область памяти и длина области памяти в байтах

Возвращаемое значение:

- в случае успеха - NB_OK
- в случае неудачи - код ошибки

nb_start_param_blob

```
NB_ERROR_TYPE nb_start_param_blob(
    NB_HANDLE connection,
    NB_DATA_TYPE datatype
);
```

Начинает запись длинной строки или области.

Параметры:

- connection - дескриптор соединения
- datatype - тип параметра. Может принимать значения
  NB_DATA_STRING
  NB_DATA_U16STRING
  NB_DATA_BINARY

Возвращаемое значение:

- в случае успеха - NB_OK
- в случае неудачи - код ошибки

nb_write_blob_part

```
NB_ERROR_TYPE nb_write_blob_part(
    NB_HANDLE connection,
    void * ptr, int len
);
```

Запись части длинной строки.

Параметры:

- connection - дескриптор соединения
- ptr, len - указатель на область памяти и длина области памяти в байтах

Возвращаемое значение:

- в случае успеха - NB_OK
- в случае неудачи - код ошибки

nb_end_param_blob

```
NB_ERROR_TYPE nb_end_param_blob( NB_HANDLE connection );
```

Завершает запись длинной строки или области.

Параметры:

- connection - дескриптор соединения

Возвращаемое значение:

- в случае успеха - NB_OK
- в случае неудачи - код ошибки

nb_close_prepared_query

```
NB_ERROR_TYPE nb_close_prepared_query(
    NB_HANDLE connection,
    DB_HANDLE queryh
);
```

Освобождает память и другие ресурсы параметризованного запроса. С этого момента нельзя использовать старое значение queryh.

Параметры:

- connection - дескриптор соединения
- queryh - handle запроса

Возвращаемое значение:

- в случае успеха - NB_OK
- в случае неудачи - код ошибки

Пример: подготовка и выполнение подготовленного запроса

```
NB_HANDLE connection = nb_connect( u"127.0.0.1", 3020, u"TESTUSER", u"1234" );
check_error( connection );

//prepare query
NB_HANDLE queyhandle;
nb_prepare_query( connection, u"select * from person where age > ? and age < ? and
name = ?", &queyhandle );
check_error( connection );

//run prepared query
nb_start_execute_prep( connection, queyhandle );

nb_write_param_int( connection, 29 );
nb_write_param_int( connection, 31 );
//nb_write_param_str( connection, (char16_t *)u"Livia", 5 );
nb_start_param_blob( connection, NB_DATA_U16STRING );
nb_write_blob_part( connection, (void *)u"Liv", 6 );
nb_write_blob_part( connection, (void *)u"ia", 4 );
nb_end_param_blob( connection );

nb_execute_prep( connection );

check_error( connection );

while ( nb_fetch_row( connection ) == NB_OK )
{
    std::cout << "\n-----------------------------\n\n";

    NBValue name, v;
    int fieldcount = nb_field_count( connection );
    for ( int i = 0; i < fieldcount; i++ )
```

```
    {
        nb_field_name_utf8( connection, i, &name );
        nb_field_value_utf8( connection, i, &v );
        std::cout << name << ": " << v << std::endl;
    }
}
check_error( connection );

nb_disconnect( connection );
```

BULK INSERT

Режим bulk insert позволяет ускорить массовое добавление и обновление данных.

Перед массовым добавлением записей можно включить режим с помощью функции nb_start_bulk. По завершении массовой модификации данных надо вызывать функцию nb_stop_bulk. Между функциями nb_start_bulk и nb_stop_bulk можно выполнять подготовленные параметризованные запросы.

При выполнении каждого из этих запросов в режиме bulk insert функции nb_execute_sql не ждут ответа от сервера, и, соответственно, мы не можем знать, что очередной запрос выполнился с ошибкой. Ошибки можно проверять после выполнения некоторой порции запросов. check_error в этом режиме дожидается ответа от сервера на выполнение предыдущей порции запросов.

nb_start_bulk

```
NB_ERROR_TYPE nb_start_bulk( NB_HANDLE connection );
```

Включает режим bulk insert.

Параметры:

- connection - дескриптор соединения

Возвращаемое значение:

- в случае успеха - NB_OK
- в случае неудачи - код ошибки

nb_stop_bulk

```
NB_ERROR_TYPE nb_stop_bulk( NB_HANDLE connection );
```

Завершает режим bulk insert.

Параметры:

- connection - дескриптор соединения

Возвращаемое значение:

- в случае успеха - NB_OK
- в случае неудачи - код ошибки

Пример: добавление множества записей

```
void PrintInsertedRowsCount( NB_HANDLE connection )
{
    nb_get_bulk_info( connection );
    check_error( connection );
    int type, count;
    nb_get_change_info( connection, &type, &count );
    std::cout << "  Count = " << count << '\r';
}

int main()
{
    NB_HANDLE connection = nb_connect( u"127.0.0.1", 3020, u"TESTUSER", u"1234" );
    check_error( connection );

    nb_execute_sql( connection, u"DROP TABLE IF EXISTS Table1; CREATE TABLE Table1 (
Id INT primary key, Value INT )" );

    NB_HANDLE queryhandle;
    nb_prepare_query( connection, u"INSERT INTO Table1 VALUES(?,?)", &queryhandle );
    check_error( connection );

    nb_start_bulk( connection );

    for ( int i = 0; i < 1'000'000; i++ )
    {
        nb_start_execute_prep( connection, queryhandle );
        nb_write_param_int( connection, i );
        nb_write_param_int( connection, i );
        nb_execute_prep( connection );
        if ( ( i % 10000 ) == 9999 )
        {
            PrintInsertedRowsCount( connection );
        }
    }

    PrintInsertedRowsCount( connection );
    std::cout << std::endl;

    nb_stop_bulk( connection );

    nb_execute_sql( connection, u"SELECT TOP 10 * FROM Table1" );
    check_error( connection );

    while ( nb_fetch_row( connection ) == NB_OK )
    {
        std::cout << "\n-----------------------------\n\n";
```

```
        NBValue name, v;
        int fieldcount = nb_field_count( connection );
        for ( int i = 0; i < fieldcount; i++ )
        {
            nb_field_name_utf8( connection, i, &name );
            nb_field_value_utf8( connection, i, &v );
            std::cout << name << ": " << v << std::endl;
        }
    }
    check_error( connection );

    nb_disconnect( connection );
    return 0;
}
```

Исходный код примера содержится в каталоге samples.c++/bulk-insert-api

ПОЛУЧЕНИЕ ИНФОРМАЦИИ О СХЕМЕ БАЗЫ ДАННЫХ

Для получения информации о схеме базы данных используются функции:

- nb_get_tableslist - получение списка таблиц
- nb_get_tableschema - получение списка полей для выбранной таблицы
- nb_get_indexschema - получение информации о всех индексах базы данных

nb_get_tableslist

```
NB_ERROR_TYPE nb_get_tableslist ( NB_HANDLE connection);
```

Возвращает список таблиц.

Параметры:

- connection — дескриптор соединения с сервером

Возвращаемое значение:

- в случае успеха - NB_OK
- в случае неудачи - код ошибки

После вызова этой функции следует вызывать nb_fetch_row и т. д., как если бы был выполнен обычный запрос с помощью nb_execute_sql. Результат выполнения запроса содержит два поля:

- name — имя таблицы
  type — тип таблицы: 2 — обычная таблица; 3 — Edge-таблица.

nb_get_tableschema

```
NB_ERROR_TYPE nb_get_tableschema (
    NB_HANDLE connection,
    const char* table,
    size_t length
);
```

Возвращает список полей для указанной таблицы (см. также описание псевдотвблицы INFORMATION_SCHEMA.COLUMNS).

Параметры:

- connection — дескриптор соединения с сервером
- table — название таблицы
- length — длина значения аргумента table

После вызова этой функции следует вызывать nb_fetch_row и т. д., как если бы был выполнен обычный запрос с помощью nb_execute_sql. Результат выполнения запроса содержит пять полей:

- name — имя поля
- type — тип поля:
  1 — string,
  2 — int,
  3 — bigint,
  4 — double,
  5 — datetime,
  6 — bool,
  7 — date,
  8 — является primary key либо foreign key (в текущей версии строка)
- subtype — уточняет сведения в поле type:
  0 — обычное поле,
  1 — primary key,
  2 — foreign key
- linktable — название таблицы, на которую ссылается поле foreign key
- nullable – может принимать значение NULL

nb_get_indexschema

```
NB_ERROR_TYPE nb_get_indexschema( NB_HANDLE connection );
```

Возвращает JSON, содержащий информацию обо всех индексах в базе данных.

Параметры:

- connection — дескриптор соединения с сервером

После вызова этой функции следует вызывать nb_fetch_row и т. д., как если бы был выполнен обычный запрос с помощью nb_execute_sql.

Результат выполнения запроса содержит одну запись с одним полем, содержащим JSON.

Пример JSON

```
[
  {

    "name" : "p_age",
    "table" : "person",
    "fields" : [ "age" ]
  },
  {

    "name" : "p_city",
    "table" : "person",
    "fields" : [ "city" ]
  },
  {

    "name" : "c_model",
    "table" : "car",
    "fields" : [ "model" ]
  }
]
```

Пример: получение схемы базы данных

```
NB_HANDLE connection = nb_connect( u"127.0.0.1", 3020, u"TESTUSER", u"1234" );
check_error( connection );

prepare_testdb( connection );

std::cout << "List of db tables\n";
nb_get_tableslist( connection );
print_query_result( connection );

std::cout << "List of fields of table 'person'\n";
nb_get_tableschema( connection, u"person", 7 );
print_query_result( connection );

std::cout << "List of indicies'\n";
nb_get_indexschema( connection );
print_query_result( connection );

nb_disconnect( connection );
```

ВСПОМОГАТЕЛЬНЫЕ ФУНКЦИИ

nb_getoption

nb_setoption

```
int  nb_getoption( NB_HANDLE connection, NB_CONN_OPTIONS option );
void nb_setoption( NB_HANDLE connection, NB_CONN_OPTIONS option, int value );
```

Параметры:

- connection — дескриптор соединения с сервером
- option – опция управления;
- value – параметр управления.

Функция nb_*getoption* возвращает текущее значение для специфицированной параметром *option* опции управления некоторыми режимами работы базы данных.

Функция nb_setoption через параметр *value* устанавливает значение для специфицированной параметром *option* опции управления режимами работы базы данных.

Параметр *option* может принимать следующие значения:

- NB_OPT_TIMEOUT – установленный в системе таймаут ожидания ответа на запрос
- NB_OPT_AVGINTASDOUBLE – тип возвращаемого функцией *avg* значения при целом аргументе.
  Если NB_OPT_AVGINTASDOUBLE = 0 то avg(int_val) возвращает целое значение;
  Если NB_OPT_AVGINTASDOUBLE = 1 то avg(int_val) возвращает значение типа double

Пример

```
int timeout = nb_getoption(connection, NB_OPT_TIMEOUT);
nb_setoption(connection, NB_OPT_TIMEOUT, int timeout * 2);
nb_setoption(connection, NB_OPT_AVGINTASDOUBLE, 1); // treat AVG(int) as double
```

ОБРАБОТКА ОШИБОК

Большинство функций API возвращает значение типа NB_ERROR_TYPE (см. ниже). Возвращаемое значение NB_OK означает успешное выполнение функции. В случае неуспешного выполнения функции возвращается класс ошибки и детальное описание ошибки можно получить с помощью функции nb_err_text или nb_err_text_utf8.

Для большинства функций можно не использовать возвращаемое значение функции. Вместо этого можно после вызова функции получить код ошибки с помощью функции nb_errno. В примерах в этом документе используется функция check_error, которая вызывает функцию nb_errno для проверки кода ошибки и если код соответствует ошибке, то выводит текст ошибки и завершает тестовое приложение.

```
// печать сообщения об ошибке, полученной от базы данных
void check_error( NB_HANDLE connection )
{
    if ( nb_errno( connection ) )
    {
        cout << "ERROR: " << nb_errno( connection )
          << ": " << nb_err_text_utf8( connection )
          << endl;
        exit( 1 );
    }
}
```

Классы ошибок

Классы ошибок определены в файле mambase.h (enum MB_ERROR_TYPE) и принимают значения:

NB_OK - успешное завершение
NB_NO_DATA - нет данных при ответе на запрос
NB_PLAN
NB_ERROR_FILE - ошибка при работе с файлами
NB_ERROR_MEM - нехватка памяти
NB_ERROR_ARGS - некорректное сочетание значений аргументов функции
NB_ERROR_CPPFUNC - ошибки при вызове стандартных C/C++ функций
NB_ERROR_QUERY - ошибки в SQL запросе или другие ошибки при обработке запроса
NB_ERROR_CONNECT - ошибки во время установления соединения с базой данных
NB_ERROR_CONNECTION_LOST – потеряно соединение с базой данных
NB_ERROR_TRANSACTION_CONFLICT - конфликт при выполнении транзакции
NB_ERROR_FUNCTION_ORDER
NB_ERROR_COMPUTING
NB_ERROR_DATA_CONVERT
NB_ERROR_UNIQUE_OR_KEY
NB_ERROR_ANOTHER - прочие ошибки

nb_errno

```
NB_ERROR_TYPE nb_errno ( NB_HANDLE connection );
```

Возвращает код последней ошибки, ассоциированной с соединением.

Параметры:

- connection — дескриптор соединения с сервером

Возвращаемое значение:

- NB_OK если нет ошибки
- класс ошибки в случае ошибки

nb_err_text

```
NB_EXTERN const char16_t * nb_err_text(
    NB_HANDLE connection,
    int * len = nullptr
);

NB_EXTERN const char * nb_err_text_utf8(
    NB_HANDLE connection,
    int * len = nullptr
);
```

Возвращает текст последней ошибки, ассоциированной с соединением.

Параметры:

- connection — дескриптор соединения с сервером
- len — возвращаемое значение, указатель на переменную типа int. Возвращается длинна строки, если указатель не равен nullptr

Возвращаемое значение

- текст ошибки или пустая строка

ПОСТАВЛЯЕМЫЕ ПРИМЕРЫ

При установке NitrosBase также устанавливаются примеры, демонстрирующие использование NitrosBase C/C++ API (см. каталог samples.c++).

Вы можете создавать клиентское приложение модифицируя примеры. При создании собственного приложения в Microsoft Visual Studio следует настроить каталоги библиотек и include файлов добавив в свойства проекта пути nitrosbase/include и nitrosbase/bin.