

NITROBASE SQL V2

Описание языка SQL
ID:NBSQL2-SQL-0922

ООО «НитросДэйта Рус»
2022

Оглавление

Введение	4
Синтаксические элементы языка SQL	4
Создание таблиц.....	6
Оператор CREATE TABLE.....	6
Создание временных таблиц.....	7
Создание Edge-таблиц.....	7
Изменение структуры таблицы	8
Оператор ALTER TABLE	8
Удаление таблицы.....	9
Оператор DROP TABLE.....	9
Добавление данных в таблицу.....	9
Оператор INSERT	9
Оператор INSERT INTO ... SELECT	10
Оператор BULK INSERT	10
Модификация данных в таблице	11
Оператор UPDATE.....	11
Удаление данных из таблицы	11
Оператор DELETE.....	11
Оператор TRUNCATE TABLE	12
Оператор SELECT	12
Оператор UNION	18
Резервное копирование / восстановление	19
Резервное копирование	19
Восстановление базы из backup файла.....	19
Функции.....	20
Функция FORMAT	22
Типы данных	24
Работа с датой и временем	26
Тип DATE	26
Тип DATETIME	27

Тип DATETIME2	27
Оператор SET DATEFORMAT	28
Работа с индексами	28
Оператор CREATE INDEX.....	28
Оператор DROP INDEX.....	29
Использование индексов	29
Транзакции.....	30
Уровни изоляции транзакций	31
ПСЕВДОТАБЛИЦЫ	32
Работа со схемой данных	32
Работа с Edge-таблицами	32

ВВЕДЕНИЕ

В документе описан поддерживаемый NitroBase диалект SQL. Подавляющее большинство приведенных примеров можно протестировать, используя приложение NAdmin (см. документ «Руководство администратора»).

В примерах используется демонстрационная база данных, описанная в разделе «Демонстрационная база данных» документа «Руководство администратора».

СИНТАКСИЧЕСКИЕ ЭЛЕМЕНТЫ ЯЗЫКА SQL

В следующей таблице перечислены обозначения, которые используются в синтаксических диаграммах данного документа.

Обозначение	Комментарий
ПРОПИСНЫЕ БУКВЫ	Ключевые слова SQL.
<i>курсив</i>	Пользовательские параметры синтаксиса SQL.
полужирный	Имена баз данных типов, таблиц, столбцов, индексов, текст и т.д. должны вводиться в точном соответствии с примером.
(вертикальная черта)	Разделяет элементы синтаксиса внутри квадратных или фигурных скобок. Может быть использован только один из этих элементов.
[] (квадратные скобки)	Необязательный элемент синтаксиса.
{ } (фигурные скобки)	Обязательные элементы синтаксиса. Фигурные скобки не вводятся.
[, ...n]	Означает, что предшествующий элемент можно повторить n раз. Отдельные вхождения элемента разделяются запятыми.
[...n]	Означает, что предшествующий элемент можно повторить n раз. Отдельные вхождения элемента разделяются пробелами.
;	Признак конца SQL запроса.

Обозначение	Комментарий
<label> ::=	<p>Имя синтаксического блока. Синтаксические блоки используются для</p> <ul style="list-style-type: none"> • группирования сегментов с длинным синтаксисом или • элемента синтаксиса, который может использоваться в нескольких расположениях данной инструкции. <p>Каждое расположение, в котором может быть использован синтаксический блок, обозначается меткой, заключенной в chevrons: <label>.</p>

Ключевые слова языка SQL, например, SELECT, UPDATE и VALUES не чувствительны к регистру, например, следующие запросы эквивалентны:

```
SELECT * FROM car WHERE Year > 2011 AND Year < 2017;
select * from car where Year > 2011 and Year < 2017;
Select * From car Where Year > 2011 And Year < 2017;
```

Идентификаторы, например названия таблиц или полей также не чувствительны к регистру, например, следующие запросы эквивалентны:

```
SELECT * FROM car WHERE Year > 2011 AND Year < 2017;
SELECT * FROM CAR WHERE year > 2011 AND year < 2017;
SELECT * FROM Car WHERE YEAR > 2011 AND YEAR < 2017;
```

Комментарии поддерживаются однострочные (--) и многострочные (/* ... */), например:

```
/*
 * Два одинаковых запроса на разных языках
 * Демонстрируют преимущества графового подхода
 */
-- Запрос на SQL
SELECT car.number FROM car
  JOIN owner ON owner.fromid = car.id
  JOIN person ON owner.toid = person.id
 WHERE person.id = 'person679'
-- Запрос на Graph-SQL
SELECT c.number MATCH! (person p, p.id='person679')<-[owner]-(car c)
```

СОЗДАНИЕ ТАБЛИЦ

Оператор CREATE TABLE

NitrosBase поддерживает все классические опции и параметры оператора *CREATE TABLE*.

Синтаксис

```
CREATE TABLE tbl_name
  ( create_definition [, create_definition] ... )

create_definition:
  col_name col_data_type [NOT NULL | NULL] [DEFAULT literal]
  [IDENTITY [(seed, increment)]]
  [PRIMARY KEY] | [CONSTRAINT [symbol]] FOREIGN KEY
  [index_name] (col_name) REFERENCES tbl_name
```

Пример

```
CREATE TABLE vehicle (
  id nvarchar PRIMARY KEY,
  model nvarchar,
  year int,
);
```

Пример

```
CREATE TABLE IF NOT EXISTS car (
  id int NOT NULL,
  model nvarchar,
  year int,
  color nvarchar,
  number nvarchar,
  owner nvarchar FOREIGN KEY(owner) REFERENCES person
);
```

Поле IDENTITY

поле *identity* — это числовое поле, которое автоматически получает целое значение, когда вставляется запись. Поля *identity* могут быть объявлены как *int* или *bigint*. В таблице может быть только одно поле *identity*.

Значения поля *identity* генерируются автоматически для каждой вставляемой строки на основе параметров *seed* (начальное значение) и *increment* (приращение) столбца *identity*. По умолчанию *seed* = 1, *increment* = 1. Если вы хотите изменить значения параметров, необходимо указать оба значения.

Пример.

```
CREATE TABLE Engine
(
    EngineID int identity(10,5) not null,
    EngineName varchar(50) not null,
    EngineDesc varchar(100) not null
);
INSERT INTO Engine (EngineName, EngineDesc) VALUES
("Bentley", "The perfect car in the world");
INSERT INTO Engine (EngineName, EngineDesc) VALUES
("Renault", "The OK car");

SELECT * from Engine;
```

Результатом оператора SELECT выше будет

EngineID	EngineName	EngineDesc
10	Bentley	The perfect car
15	Renault	The OK car

Создание временных таблиц

Если имя таблицы начинается со знака '#', то создается временная таблица. Временные таблицы существуют на протяжении текущей сессии. Если необходимо удалить таблицу до завершения сессии базы данных, то для этой таблицы следует выполнить запрос DROP TABLE.

Пример

```
CREATE TABLE #temp1 (
    p_id varchar PRIMARY KEY,
    name varchar,
    lastname varchar,
);
```

Создание Edge-таблиц

Edge-таблица - специальная таблица для представления графовой связи (связь типа «многие-ко-многим»). Edge-таблица может интерпретироваться как промежуточная таблица в обычной реляционной базе данных, имеющая два фиксированных поля: *fromid* и *toid*.

Синтаксис

```
CREATE TABLE link_name AS EDGE from_tablename to_tablename;
```

Примеры

Edge-таблица *owner* связывает (обычные) таблицу *car* и таблицу *person* (см. раздел «Демонстрационная база данных» документа «Руководство администратора»).

```
CREATE TABLE owner AS EDGE car person;
```

После создания, такая таблица может использоваться, например, в запросе *SELECT*:

```
SELECT p.id, name, lastname, age, c.id, c.model
FROM person p
  JOIN owner o ON p.id = o.toid
  JOIN car c ON c.id = o.fromid
WHERE age > 20 AND model = 'Toyota';
```

То же самое может быть сделано и более простым запросом (подробнее см. раздел «Работа с Edge-таблицами»):

```
SELECT p.id, name, lastname, age, c.id, c.model
FROM person p
  JOIN car c ON p.id = c.owner
WHERE age > 20 AND model = 'Toyota';
```

ИЗМЕНЕНИЕ СТРУКТУРЫ ТАБЛИЦЫ

Оператор ALTER TABLE

Синтаксис

```
ALTER TABLE tbl_name
  <alter_specification> [,...]

<alter_specification> ::=
  | ADD [COLUMN] <column_definition>,...
  | {ALTER|MODIFY} [COLUMN] column_definition,...
  | DROP [COLUMN [if exists]] col_name

<column_definition> ::=
  col_name data_type
  [NOT NULL | NULL] [DEFAULT literal]
  | [CONSTRAINT [symbol]] FOREIGN KEY
  [index_name] (col_name) REFERENCES tbl_name
```

Пример.

Переименование столбцов

```
ALTER TABLE person ADD car_id int, nameday DATE
```

УДАЛЕНИЕ ТАБЛИЦЫ

Оператор DROP TABLE

Синтаксис

```
DROP TABLE [IF EXISTS] tbl_name
```

tbl_name - имя таблицы или Edge-таблицы

Пример

```
DROP TABLE person;  
CREATE TABLE person(id int PRIMARY KEY, value varchar);
```

Если таблица *person* существует, она будет удалена, затем будет создана новая таблица *person*. Если таблица *person* не существует, то оператор *DROP TABLE* вернет ошибку и оператор *CREATE TABLE* выполнен не будет.

```
DROP TABLE IF EXISTS person;  
CREATE TABLE person(id int PRIMARY KEY, value varchar);
```

Если таблица *person* существует, она будет удалена, затем в любом случае будет создана новая таблица *person*.

ДОБАВЛЕНИЕ ДАННЫХ В ТАБЛИЦУ

Оператор INSERT

Синтаксис

```
INSERT INTO tbl_name  
[(col_name,...)]  
VALUES (value,...)
```

Пример

```
insert into person values
(1, 'John', 'Goodman', 30, 'Boston', 65000, 3.62, '2022/07/28 00:00:00:00')
```

Пример

```
INSERT INTO person
(id, name, lastname, age, city)
VALUES
('person5000', 'John', 'Tester', 30, 'Lisbon');
```

Оператор INSERT INTO ... SELECT ...

Синтаксис

```
INSERT INTO tbl_name
[(col_name,...)]
SELECT ...
```

Пример

```
INSERT INTO young_person
(id, name, lastname, age, city)
SELECT
id, name, lastname, age, city
FROM person
WHERE age < 30
```

Оператор BULK INSERT

Синтаксис

```
BULK INSERT table_name (col_name,...)
FROM 'file_path'
[WITH ([FIRSTROW = number,] [FIELDTERMINATOR = 'character'])]
```

Параметр *FIRSTROW* указывает, с какой строки файлов начинать импорт (нумерация строк начинаются с 1). Например, для того чтобы пропустить заголовок (пропустить одну первую строку), нужно указать *FIRSTROW = 2*.

Путь к файлу как правило указывается абсолютный. Возможно также, хотя и не удобно в большинстве случаев, указывать путь относительно файла сервера NitroBase. Например, если исполняемый файл сервера *nbserver.exe* находится в каталоге *c:/nitrobase/bin*, то относительный путь *../data/csv/person.csv* будет раскрыт как абсолютный путь *c:/nitrobase/data/csv/person.csv*.

Пример

```
BULK INSERT car (id, model, year, color, number)
FROM 'C:/MyApp/data/csv/car.csv'
WITH (FIRSTROW = 2, FIELDTERMINATOR = ',');
```

МОДИФИКАЦИЯ ДАННЫХ В ТАБЛИЦЕ

Оператор UPDATE

Синтаксис

```
UPDATE table_reference
  SET col_name = value,...
  [WHERE where_condition]

UPDATE table_name_or_alias
  SET col_name = value,...
  FROM tablename [alias] | join_expression
  [WHERE where_condition]
```

Синтаксис выражений FROM и WHERE тот же, что и в запросе SELECT.

Пример

```
UPDATE person SET name = 'LAMAR' WHERE name = 'Lamar';
```

УДАЛЕНИЕ ДАННЫХ ИЗ ТАБЛИЦЫ

Оператор DELETE

Синтаксис

```
DELETE FROM tbl_name
  [WHERE where_condition]

DELETE FROM table_name_or_alias FROM join_expression
  [WHERE where_condition]
```

Синтаксис выражений *FROM* и *WHERE* тот же, что и в запросе *SELECT*.

При удалении записи будут автоматически удалены все ссылки на нее, а также будут удалены записи из Edge-таблиц, в которых удаляемая запись была в поле *fromid* или *toid*

Примеры

```
DELETE FROM person WHERE name = 'Livia';
```

удаление с условием на связанные таблицы

```
DELETE FROM p
FROM person p join car c ON p.id = c.owner
WHERE c.model = 'Toyota';
```

Оператор TRUNCATE TABLE

Удаляет все строки из таблицы. Оператор TRUNCATE TABLE аналогичен оператору DELETE без предложения WHERE. TRUNCATE TABLE работает быстрее и использует меньше системных ресурсов.

Синтаксис

```
TRUNCATE TABLE tbl_name
```

Пример

```
TRUNCATE TABLE #temptable1;
```

ОПЕРАТОР SELECT

Синтаксис

```
SELECT [ ALL | DISTINCT ]
  [ TOP ( expression ) ]
  <select_list>
  [ FROM { <table_source> } [ ,...n ] ]
  [ WHERE <search_condition> ]
  [ GROUP BY <group_by_list> ]
  [ HAVING <search_condition> ]
  [ ORDER BY <order_by_list> ]
  [ OFFSET <offset_expression> ]

<select_list> ::=
{
  *
  | { table_name | table_alias }. *
  | {
      [ { table_name | table_alias }. ] { column_name }
      | expression
      [ [ AS ] column_alias ]
    }
  | column_alias = expression
```

```

} [ ,...n ]

/*
[ FROM { <table_source> } [ ,...n ] ]
*/
??? как описать использование генератора который просил Морозов
??? разобраться с rowset_function

<table_source> ::=
{
  table_name [ AS ] table_alias ]
  | rowset_function [ [ AS ] table_alias ]
  | derived_table [ [ AS ] table_alias ]
  | <joined_table>
}
<joined_table> ::=
{
  <table_source> <join_type> <table_source> ON <search_condition>
  | <table_source> CROSS JOIN <table_source>
  | [ ( ] <joined_table> [ ) ]
}

<join_type> ::=
[ { INNER | { { LEFT | RIGHT | FULL } [ OUTER ] } } [ <join_hint> ] ]
JOIN

<group_by_list> ::=
{
  [ { table_name | table_alias }. ] { column_name }
  | expression
  | column_number
} [ ,...n ]

<order_by_list> ::=
{
  [ { table_name | table_alias }. ] { column_name }
  | expression
  | column_number
  [ ASC | DESC ]
} [ ,...n ]

```

Select выражение

Select выражение – это часть оператора, расположенная между ключевыми словами *SELECT* и *FROM*. Она может содержать:

Простой список полей:

```
SELECT name, lastname FROM person;
```

Все поля:

```
SELECT * FROM person
SELECT p.*, c.model FROM person p JOIN car c ON p.id = c.owner;
```

AS:

```
SELECT COUNT(*) AS c FROM person;
```

Простые выражения и функции:

```
SELECT upper(name) AS uname, age*2 AS dbl FROM person;
```

Агрегирующие функции:

```
SELECT COUNT(age), COUNT_BIG(age), MIN(age), MAX(age), AVG(age) FROM person;
```

ALL | DISTINCT:

```
SELECT ALL name, lastname FROM person WHERE name LIKE 'Vanda';
SELECT DISTINCT name, lastname FROM person WHERE name LIKE 'Vanda';;
```

Замечание. Ключевое слово ALL как правило является избыточным, поскольку оно является случаем по умолчанию. То есть, вместо *SELECT ALL name ...* можно писать *SELECT name ...*

TOP

```
SELECT TOP 1000 * FROM car;
```

Вложенный *SELECT*:

```
SELECT
  name,
  (SELECT model FROM car WHERE p.id = owner)
  AS modelname
FROM person p;
```

Конструкцию CASE...WHEN:

```
SELECT name,
CASE name
  WHEN 'Lamar' THEN 'L'
  WHEN 'Mercedez' THEN 'M'
  ELSE 'U' END AS n
FROM person;

SELECT id,
CASE WHEN age < 30 THEN 1
ELSE 2 END as age_category,
name, lastname, income
from person
```

Условие поиска

Эта часть оператора *SELECT* может содержать:

Простые операторы сравнения и логические операторы:

```
SELECT * FROM person WHERE age > 20 AND name = 'Lamar';
```

Оператор LIKE, выполняющий сопоставление с образцом для строковых типов:

```
SELECT * FROM person WHERE name LIKE '%amar%';
SELECT * FROM person WHERE name LIKE 'L[ai]%';
```

Оператор IN или NOT IN:

```
SELECT name, age FROM person
WHERE age IN (20, 22, 24) AND name IN ('Lamar', 'Susan');

SELECT * FROM customers
WHERE ctype IN (
  SELECT distinct otype
  FROM orders
  WHERE odate >= '2020-03-03'
);
```

Проверка на NULL:

```
SELECT * FROM person WHERE age IS NULL;
SELECT * FROM person WHERE age IS NOT NULL;
```

Вложенный запрос с условиями EXISTS | NOT EXISTS

```
SELECT * FROM person WHERE age > 40
AND EXISTS( SELECT * FROM car WHERE car.owner = person.id );
```

Другие операторы и функции (список операторов и функций см. далее):

```
SELECT * FROM person WHERE age*2 - income/20 > 0 AND UPPER(name) = 'LAMAR';
```

Выражения FROM и JOIN

Выражение GROUP BY

Простая группировка по одному полю:

```
SELECT COUNT(dbl), AVG(dbl), MIN(dbl), MAX(dbl)
FROM person
GROUP BY city;
```

Группировка по нескольким полям:

```
SELECT COUNT(dbl), AVG(dbl), MIN(dbl), MAX(dbl)
FROM person
GROUP BY city, name;
```

Выражение HAVING

Фильтрация результатов агрегации с использованием HAVING:

```
SELECT city, AVG(dbl) AS avg_dbl
FROM person
GROUP BY city
HAVING (avg_dbl <= 0.5);
```

В агрегирующих функциях возможно использовать различные выражения и *DISTINCT*:

```
SELECT AVG(income*dbl) AS avg_expr FROM person

SELECT AVG(
  CASE WHEN name = 'Lamar' THEN 1
        WHEN name = 'Mercedez' THEN 2
        ELSE 0 end
) AS avg_expr
FROM person'

SELECT city, COUNT(distinct id) FROM person GROUP BY city;
```

Если выражение *GROUP BY* опущено, но присутствуют агрегирующие функции, весь результат трактуется как одна группа.

Выражение ORDER BY

Простая сортировка по одному полю:

```
SELECT name, lastname, age
FROM person
ORDER BY name;
```

Сортировка по нескольким полям и с указанием порядка сортировки:

```
SELECT name, lastname, age
FROM person
ORDER BY name, lastname ASC, age DESC;
```

Сортировка по номеру колонки (номер колонки в выражении select начинается с 1):

```
SELECT name, lastname, age
FROM person
ORDER BY 1, lastname ASC, 3 DESC;
```

Сортировка по выражению:

```
SELECT name, lastname, age
FROM person
ORDER income/age, name, lastname;
```

Выражения LIMIT и OFFSET

LIMIT и *OFFSET* служат для ограничения количества записей в результате:

Вывести только 100 записей:

```
SELECT * FROM person LIMIT 100;
```

Вывести 10 записей, начиная с 100-й (нумерация записей начинается с нуля — то есть пропустить первые 100 записей):

```
SELECT * FROM person ORDER BY id LIMIT 10 OFFSET 100;
```

Замечание. Поскольку в SQL в общем случае порядок получения данных не определен, то в примере выше без конструкции *ORDER BY* мы получили бы неопределенный результат.

Операторы

операторы сравнения: $>$, $>=$, $<$, $<=$, $=$, $<>$, $!=$, *IS NULL*, *IS NOT NULL*, *IN*

```
SELECT * FROM person WHERE age IS NULL;
```

оператор *LIKE*, выполняющий сопоставление с образцом для строковых типов.

Наряду с % (любая последовательность символов) и _ (любой один символ) в шаблонах разрешается использовать заключенные между "[" и "]" перечисления символов. При задании перечислений разрешается использовать ^ (отрицание) и - (диапазон). Для экранирования спецсимволов следует заключать их в квадратные скобки.

```
SELECT * FROM person WHERE name LIKE '%amar%';  
SELECT * FROM person WHERE name LIKE 'L[ai]%';
```

Логические операторы: *AND (&&)*, *OR (/ /)*, *NOT*

```
SELECT * FROM person  
WHERE age > 50 AND (name = 'Lamar' OR lastname = 'Wurdeman');
```

Арифметические операторы: $+$, $-$, $*$, $/$

```
SELECT age*2 FROM person WHERE age*100/income > 3;
```

ОПЕРАТОР UNION

Синтаксис

```
SELECT ...  
UNION [ALL]  
SELECT ...  
[UNION [ALL] SELECT ...]
```

Пример

```
SELECT id, name, lastname, city FROM person WHERE id = 'person22'  
UNION  
SELECT id, name, lastname, city FROM person WHERE id = 'person33'  
UNION  
SELECT id, name, lastname, city FROM person WHERE id = 'person55'
```

РЕЗЕРВНОЕ КОПИРОВАНИЕ / ВОССТАНОВЛЕНИЕ

Резервное копирование

Для резервного копирования можно использовать CLI утилиту `nbase.exe`. Для этого необходимо вызвать команду, например,

```
$ nbase -c backup -p 3023 -b "C:\snt0"
```

В этом случае необходимо указать порт и абсолютный путь к backup каталогу.

Либо можно использовать утилиту администрирования NBaseAdmin (см. документ "Руководство администратора"). В утилите администрирования можно произвести резервное копирование с помощью пользовательского интерфейса, либо же выбрав базу данных, в окне запросов набрать SQL запрос.

```
BACKUP DATABASE TO Path
```

Где *Path* – абсолютный путь.

Пример

```
BACKUP DATABASE TO 'c:/data/sntest1/backup'
```

В результате выполнения этой команды backup копия текущей базы данных сохраняется в каталог `'c:/data/sntest1/backup'`.

Восстановление базы из backup файла

Для восстановления базы данных можно использовать как CLI утилиту `nbase.exe`, так и NBaseAdmin (см. документ "NitrosBase Admin Tool").

В случае использования `nbase.exe`, следует выполнить, например, следующую команду:

```
$ nbase sntest2 -c restore -p 3023 -b "C:\sntest2.2022-07-26.12-57-17"
```

При использовании NBaseAdmin следует пользоваться средствами пользовательского интерфейса. Подробнее см. документ "Руководство администратора"

ФУНКЦИИ

Функция	Описание
<i>ABS</i>	Возвращает абсолютное значение (модуль) числа
<i>ACOS</i>	Возвращает арккосинус действительного числа
<i>ASCII</i>	Возвращает ASCII-код символа
<i>ASIN</i>	Возвращает арксинус действительного числа
<i>ATAN</i>	Возвращает арктангенс действительного числа
<i>CAST</i>	Выполняет преобразование типов
<i>CEIL</i>	Наименьшее целое, большее переданного аргумента или равное ему
<i>CHR</i>	Возвращает символ с соответствующим ASCII-кодом
<i>CONCAT</i>	Соединение (конкатенация) двух или более строк
<i>COS</i>	Возвращает косинус числа
<i>COT</i>	Возвращает котангенс числа
<i>DATEPART</i>	Возвращает целочисленное значение указанной части даты
<i>DATETIME2FROMPARTS</i>	Возвращает datetime2, построенное из указанных параметров: <i>year, month, day, hour, minute, seconds, fractions, precision</i>
<i>DIV</i>	Целочисленное деление, возвращается целое — результат деления с остатком первого аргумента на второй
<i>EXP</i>	Возвращает основание натуральных логарифмов <i>e</i> , возведенное в степень, переданную как аргумент
<i>FORMAT</i>	См раздел «Функция FORMAT» ниже.
<i>ISNULL</i>	Проверяет выражение на NULL и возвращает указанное значение если NULL иначе возвращает значение выражения

<i>LEFT, STRLEFT</i>	Возвращает начальную подстроку исходной строки (первый аргумент) с определенной длиной (второй аргумент)
<i>LEN, LENGTH</i>	Возвращает длину (количество байтов) строки
<i>LN, LOG</i>	Возвращает натуральный логарифм числа
<i>LOG10</i>	Возвращает логарифм по основанию 10
<i>LOG2</i>	Возвращает логарифм по основанию 2
<i>LOWER</i>	Преобразует символы строки в нижний регистр
<i>LPAD</i>	Дополняет слева первую строку второй до достижения определенной длины
<i>MOD</i>	Остаток от целочисленного деления первого аргумента на второй
<i>MONTH</i>	Возвращает месяц даты
<i>PI</i>	Возвращает значение числа пи
<i>POWER</i>	Возводит число (первый аргумент) в степень (второй аргумент)
<i>POS, POSITION</i>	Возвращает позицию первого вхождения второй строки в первую
<i>RAND</i>	Возвращает случайное число большее 0 и меньше 1
<i>REPEAT</i>	Сцепляет строку саму с собой определенное число раз; строка — первый аргумент, количество повторений — второй
<i>REPLACE</i>	Заменяет в исходной строке (первый аргумент) все вхождения одной строки (второй аргумент) другой (третий аргумент)
<i>REVERSE</i>	Обращает строку
<i>ROUND</i>	Округляет число до указанного количества десятичных разрядов
<i>STRLEFT</i>	Начальная подстрока исходной строки (первый аргумент) определенной длины (второй аргумент)
<i>STRRIGHT</i>	Конечная подстрока исходной строки (первый аргумент) определенной длины (второй аргумент)

<i>RPAD</i>	Дополняет справа первую строку второй до достижения определенной длины
<i>REGEX</i>	Проверка совпадения строки с образцом, задаваемым регулярным выражением
<i>RIGHT, STRRIGHT</i>	Возвращает начальную подстроку исходной строки (первый аргумент) с определенной длиной (второй аргумент)
<i>SIGN</i>	Возвращает знак числа: -1 для отрицательных, 0 для 0, 1 для положительных
<i>SIN</i>	Возвращает синус числа
<i>SQRT</i>	Возвращает квадратный корень числа
<i>SUBSTR, SUBSTRING</i>	Выделяет подстроку из строки
<i>TAN</i>	Возвращает тангенс числа
<i>TRIM</i>	Удаляет начальные и конечные пробелы из строки
<i>TRUNCATE</i>	Усекает число до указанного числа знаков после запятой
<i>UPPER</i>	Преобразует переданную строку в верхний регистр
<i>YEAR</i>	Возвращает год переданной даты

Функция **FORMAT**

Возвращает строковое значение в указанном формате. Выполняет форматирование даты, времени и чисел.

Синтаксис

```
FORMAT (value, format [, culture])
```

Параметры

- Value* Выражение поддерживаемого типа данных для форматирования
- Format* Шаблон формата
- Culture* Языковые и региональные параметры. Необязательный строковый аргумент, в текущей версии NitroBase игнорируется.

Для значений даты и времени используются следующие шаблоны в форматной строке:

Шаблон	Комментарий
d	День месяца (1..31) одной или двумя цифрами.
dd	День месяца (1..31) двумя цифрами. Для значений меньших 10, добавляется лидирующий 0.
M	Номер месяца (1..12) одной или двумя цифрами.
MM	Номер месяца (1..12) двумя цифрами. Для значений меньших 10 добавляется лидирующий 0.
yy	Год двумя цифрами. При необходимости добавляется лидирующий 0.
yyyy	Год четырьмя цифрами. При необходимости дополняется лидирующими нулями до требуемого числа разрядов.
h	Час суток в 12-часовом представлении (1..12) одной или двумя цифрами.
hh	Час суток в 12-часовом представлении (1..12) двумя цифрами. При необходимости добавляется лидирующий ноль.
H	Час суток в 24-часовом представлении (0..23) одной или двумя цифрами.
HH	Час суток в 24-часовом представлении (0..23) двумя цифрами. При необходимости добавляется лидирующий ноль.
m	минута часа (0..59) одной или двумя цифрами.
mm	минута часа (0..59) двумя цифрами. При необходимости добавляется лидирующий ноль.
s	секунда (0..59) одной или двумя цифрами.
ss	секунда (0..59) двумя цифрами. При необходимости добавляется лидирующий ноль.
t	маркер AM/PM одним символом. То есть, AM представляется как A, а PM как P.
tt	маркер AM/PM двумя символами.

Примеры

```
SELECT FORMAT(CAST('0021-01-02 13:03:02' AS DATETIME), 'd.MM.yyyy hh:mm:ss tt')
SELECT FORMAT(CAST('0000.00.00 07:35:00' AS DATETIME), N'hh.mm');
SELECT FORMAT(CAST('2018-01-01 01:00:00' AS datetime2), N'hh:mm tt');
SELECT FORMAT(CAST('2018-01-01 01:00:00' AS datetime2), N'hh:mm t');
SELECT FORMAT(123456789, '###-##-####') AS [Custom Number];
```

ТИПЫ ДАННЫХ

NitrosBase поддерживает следующие типы данных в запросах:

Тип	Псевдонимы	Описание
<i>BIT</i>	<i>BOOL</i>	0 или 1, аналог булевого типа
<i>INT</i>	<i>INTEGER, TINYINT</i>	4-байтовое целочисленное значение от <i>-2 147 483 648</i> до <i>2 147 483 647</i>
<i>BIGINT</i>	—	8-байтовое целочисленное значение от <i>-9 223 372 036 854 775 808</i> до <i>9 223 372 036 854 775 807</i>
<i>CHAR(size)</i>	<i>VARCHAR(size), NVARCHAR(size), TEXT</i>	Строка, длина строки ограничивается указанным в скобках значением
<i>REAL(s, d)</i>	<i>DOUBLE(size, d), FLOAT(size, d)</i>	8-байтовое с плавающей точкой, диапазон точности от <i>1.7E-308</i> до <i>1.7E+308</i> . Значения в круглых скобках игнорируются

Тип	Псевдонимы	Описание
<i>DECIMAL(p, s)</i>	<i>NUMERIC(p, s)</i>	Число с фиксированной точностью. Занимает от 5 до 17 байт в зависимости от количества чисел после запятой. Принимает параметры p (precision) и s (scale). Параметр p - максимальное количество цифр, которые может хранить число - должен находиться в диапазоне от 1 до 38. По умолчанию 18. Параметр s - максимальное количество цифр после запятой - должен находиться в диапазоне от 0 до значения параметра p. По умолчанию оно 0. См. пример ниже.
<i>DATE</i>	—	См. раздел «Работа с датой и временем»
<i>DATETIME</i>	—	См. раздел «Работа с датой и временем»
<i>DATETIME2(prec)</i>	—	См. раздел «Работа с датой и временем»
<i>VARBINARY(size)</i>	—	Бинарные данные. Пример строкового представления: <i>0x0000000A4</i>
<i>ROWVERSION</i>	<i>TIMESTAMP</i>	Тип данных, который представляет собой генерируемые автоматически уникальные двоичные числа. Этот тип данных используется как правило для фиксации версии записи. Представляет собой увеличивающееся при каждом изменении записи двоичное число размером 8 байт.

Пример 1

Тип *DECIMAL* предназначен для хранения чисел с фиксированной точкой. Он гарантирует заданную точность, в отличие от типов с плавающей точкой (*REAL*, *DOUBLE*, *FLOAT*), в которых накапливается ошибка.

```
CREATE TABLE bill(id_ INT, float_ FLOAT, dec_ DECIMAL(10,2));
INSERT INTO bill(id_, float_, dec_) VALUES(1, 0, 0);
UPDATE bill SET float_=float_+0.01, dec_=dec_+0.01 WHERE id_=1;
SELECT * FROM bill;
```

Данная последовательность запросов прибавляет на каждом шаге по 0.01 к переменным типа: *FLOAT* и *DECIMAL*. Результирующий *SELECT* запрос вернет такие данные:

Id_	Float_	Dec_
1	0.100000000000000007	0.1

Пример 2

Демонстрация работы с ROWVERSION.

```
-- Создать пустую таблицу MyTest
CREATE TABLE MyTest (myKey int PRIMARY KEY, myValue int, RV ROWVERSION);

-- Вставить 2 записи в таблицу.
INSERT INTO MyTest (myKey, myValue) VALUES (1, 0);
INSERT INTO MyTest (myKey, myValue) VALUES (2, 0);

-- После вставки поле RV стало равно:
-- для первой записи RV = 0x000000000000000001
-- для второй записи RV = 0x000000000000000002

-- Модифицировать таблицу
update MyTest set myValue = myValue + 1

-- После модификации поле RV стало равно
-- для первой записи RV = 0x000000000000000003
-- для второй записи RV = 0x000000000000000004
```

РАБОТА С ДАТОЙ И ВРЕМЕНЕМ

Тип DATE

Даты от 0001-01-01 (1 января 0001 года) до 9999-12-31 (31 декабря 9999 года).

Поддерживаемые строковые форматы значений:

- YYYY-MM-DD
- YYYY/MM/DD
- YYYY.MM.DD

Пример

```
CREATE TABLE tb (
    fld_date DATE,
    comment VARCHAR (50)
);/**/
INSERT INTO tb VALUES('2022-01-20', 'Date with -')
INSERT INTO tb VALUES('2022/01/20', 'Date with /')
INSERT INTO tb VALUES('2022.01.20', 'Date with .')
```

Тип DATETIME

Дата и время суток, где дата — в том же диапазоне, что и в *DATE*.

Поддерживаемые строковые форматы значений:

```
YYYY-MM-DD hh:mm:ss.nnnnnnnn
YYYY/MM/DD hh:mm:ss.nnnnnnnn
YYYY.MM.DD hh:mm:ss.nnnnnnnn
```

Максимальное количество цифр в дробной части секунд равно 7.

Пример

```
CREATE TABLE tb (
    fld_datetime DATETIME,
    comment VARCHAR (50)
);
INSERT INTO tb VALUES ('2022-01-20 12:37:23.1234567', 'Datetime with -')
INSERT INTO tb VALUES ('2022/01/20 12:37:23.1234567', 'Datetime with /')
INSERT INTO tb VALUES ('2022.01.20 12:37:23.1234567', 'Datetime with .')
```

Тип DATETIME2

Тип DATETIME2 практически идентичен DATETIME за исключением указания в качестве параметра количества знаков в дробной части секунд.

Синтаксис

```
DATETIME2(Precision)
```

Где *Precision* – целое - максимальное количество знаков в дробной части секунд. Максимальное значение параметра *Precision* равно 7. Если количество цифр в дробной части секунд выше заданной точности, происходит округление до заданного в параметре *Precision*.

Пример

```
CREATE TABLE tb (
    fld_datetime2 DATETIME2(4),
    comment VARCHAR (50)
);
INSERT INTO tb VALUES ('2022-01-20 12:37:23.12', 'Datetime2 with -')
INSERT INTO tb VALUES ('2022/01/20 12:37:23.1234', 'Datetime2 with /')
INSERT INTO tb VALUES ('2022.01.20 12:37:23.123456', 'Datetime2 with .')

select * from tb
```

Результат

```
2022-01-20 12:37:23.1200    Datetime2 with -
2022-01-20 12:37:23.1234    Datetime2 with /
2022-01-20 12:37:23.1235    Datetime2 with .
```

Оператор SET DATEFORMAT

Определяет порядок следования чисел в дате.

Синтаксис

```
SET DATEFORMAT mdy | dmy | ymd | ydm | myd | dym
```

По умолчанию DATEFORMAT = ymd.

Пример

```
CREATE TABLE tb (
    fld_datetime DATETIME,
    comment VARCHAR(50)
);

INSERT INTO tb VALUES('2022-01-20 12:37:23', 'Date format is ymd')
SET DATEFORMAT dmy
INSERT INTO tb VALUES('20-01-2022 12:37:23', 'Date format is dmy')
SET DATEFORMAT mdy
INSERT INTO tb VALUES('01-20-2022 12:37:23', 'Date format is mdy')
```

Результат

```
2022-01-20 12:37:23.1200    Date format is ymd
2022-01-20 12:37:23.1234    Date format is dmy
2022-01-20 12:37:23.1235    Date format is mdy
```

РАБОТА С ИНДЕКСАМИ

Оператор CREATE INDEX

Синтаксис

```
CREATE [UNIQUE] INDEX index_name ON tbl_name (col_name,...)
```

Пример

```
CREATE INDEX p_ndx_age ON person (age);
```

Оператор DROP INDEX

Синтаксис

```
DROP INDEX index_name [ON tbl_name]
```

Пример

```
DROP INDEX p_ndx_age;
```

Использование индексов

Предположим, для таблицы `person` (см. раздел «Демонстрационная база данных» документа «Руководство администратора») мы создаем следующие индексы:

```
CREATE INDEX p_age ON person (age)
CREATE INDEX p_name ON person (name)
```

Тогда при вызове следующих операторов `SELECT` будет использоваться индекс `p_age`.

```
SELECT * FROM person WHERE age = 22
SELECT * FROM person WHERE age >= 22 and age < 24
```

А в следующем запросе используются два индекса: `p_age` и `p_name`

```
SELECT * FROM person WHERE age = 22 and name = 'Lamar'
```

Возможно задавать индексы по нескольким полям, например,

```
CREATE INDEX p_name_lastname ON person (name, lastname)
SELECT * FROM person WHERE name = 'Lamar' and lastname = 'Tanon'
```

При обработке простых запросов эффективное использование индексов рекомендуется только в тех случаях, когда условие, для обработки которого может использоваться индекс, выделяет менее 6% записей.

Для простых сравнений можно явно потребовать использования предварительно созданного индекса. Указание использовать индекс — ключевое слово INDEX после WHERE:

```
SELECT name, lastname, city, model
FROM person p JOIN car c ON c.owner = p.id
WHERE INDEX name = 'Lamar';
```

Индексы также используются при обработке запросов, содержащих выражение JOIN (см. раздел “Выражения FROM и JOIN”).

ТРАНЗАКЦИИ

Транзакция является важнейшим понятием в теории и практике баз данных. Транзакция объединяет некоторую совокупность SQL запросов в единую атомарную сущность. Транзакция начинается с директивы BEGIN TRANSACTION и заканчивается либо директивой COMMIT TRANSACTION, если все запросы на модификацию данных, находящиеся внутри транзакции выполнены успешно, либо директивой ROLLBACK TRANSACTION. Если все изменения внутри транзакции принимаются (COMMIT TRANSACTION), то они фиксируются в базе данных как окончательные. Если в результате выполнения транзакции происходит хотя бы одна ошибка, то срабатывает директива ROLLBACK TRANSACTION и должна быть произведена отмена (выполнен откат), все изменения данных будут отменены.

Синтаксис:

```
SET TRANSACTION ISOLATION LEVEL
{ READ UNCOMMITTED
| READ COMMITTED
| REPEATABLE READ
| SNAPSHOT
| SERIALIZABLE
}
```

Пример

```
BEGIN TRANSACTION;
DELETE FROM Candidates WHERE CandidateID = 13;
UPDATE JobPositions SET...
UPDATE Managers SET...
COMMIT TRANSACTION;
```

Уровни изоляции транзакций

NitrosBase поддерживает все базовые уровни изоляции транзакций:

Уровень изоляции	Комментарий
READ UNCOMMITTED	Низший уровень изоляции. Если несколько параллельных транзакций пытаются изменить одну и ту же строку таблицы, то в окончательном варианте строка будет иметь значение, определенное всем набором успешно выполненных транзакций. При этом возможно считывание не только логически несогласованных данных, но и данных, изменения которых ещё не зафиксированы.
READ COMMITTED	На этом уровне обеспечивается защита от чернового, «грязного» чтения, тем не менее, в процессе работы одной транзакции другая может быть успешно завершена и сделанные ею изменения зафиксированы. В итоге первая транзакция будет работать с другим набором данных.
REPEATABLE READ	Уровень, при котором читающая транзакция «не видит» изменения данных, которые были ею ранее прочитаны. При этом никакая другая транзакция не может изменять данные, читаемые текущей транзакцией, пока та не окончена.
SNAPSHOT	На этом уровне транзакция видит то состояние данных, которое было зафиксировано до её запуска, а также изменения, внесённые ею самой, то есть ведёт себя так, как будто получила при запуске моментальный снимок данных БД и работает с ним.
SERIALIZABLE	Самый высокий уровень изолированности; транзакции полностью изолируются друг от друга, каждая выполняется так, как будто параллельных транзакций не существует.

Установка уровня изоляции транзакций устанавливается директивой SET TRANSACTION ISOLATION LEVEL.

Пример:

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
```

ПСЕВДОТАБЛИЦЫ

Работа со схемой данных

Доступ к информации о схеме данных нужен как правило приложениям. При работе с данными из NVAdmin объекты схемы данных доступны непосредственно из пользовательского интерфейса. Подробную информацию о функциях доступа к схеме данных, а также примеры работы с объектами схемы данных можно получить, в разделе «Получение информации о схеме базы данных» документа "Программный интерфейс C/C++".

Работа с Edge-таблицами

На примере операций с данными из демонстрационной базы данных (см. раздел «Демонстрационная база данных» документа «Руководство администратора»).

Создание таблицы:

```
CREATE TABLE friends AS EDGE person person;
```

Заполнение таблицы:

```
INSERT INTO friends VALUES ( 'person22', 'person1022' );  
INSERT INTO friends VALUES ( 'person22', 'person1023' );
```

Получение данных:

```
SELECT p1.id, p2.id  
FROM person p1  
JOIN friends f ON p1.id = f.fromid  
JOIN person p2 ON f.toid = p2.id  
WHERE p1.id = 'person22';
```

Обновление данных:

```
UPDATE friends SET fromid = 'person7', toid = 'person67'  
WHERE fromid = 'person7' AND toid = 'person1007'  
  
UPDATE car SET owner = 'person23' WHERE id = 'car4022',
```

Удаление данных:

```
DELETE FROM friends WHERE fromid = 'person22' AND toid = 'person1022';  
DELETE FROM owner WHERE toid = 'person22',  
DELETE FROM owner WHERE fromid = 'car4021',
```