Документация по командам MySQL

Синтаксис SQL

Синтаксис SQL (Structured Query Language) определяет правила и структуру запросов к базе данных. Он состоит из различных элементов, которые объединяются в определенном порядке для формирования корректного запроса. Вот основные правила и элементы синтаксиса SQL:

1. Ключевые слова

SQL содержит набор ключевых слов, которые определяют тип операции, выполняемой с базой данных (например, SELECT, INSERT, UPDATE, DELETE, CREATE, ALTER, DROP и т.д.).

2. Идентификаторы

Идентификаторы используются для именования таблиц, столбцов, индексов, представлений и других объектов базы данных. Они могут содержать буквы, цифры и знак подчеркивания, но должны начинаться с буквы.

3. Таблицы и столбцы

Таблицы представляют собой основные структуры данных в базе данных, а столбцы определяют типы данных, которые хранятся в таблицах.

4. Выражения и операторы

SQL поддерживает различные операторы, такие как арифметические операторы (+, -, *, /), операторы сравнения (=, <, >, <=, >=, !=), логические операторы (AND, OR, NOT) и другие.

5. Условия

Условия используются для фильтрации данных при выполнении запроса. Они могут включать операторы сравнения, логические операторы и функции.

6. Функции

SQL предоставляет множество встроенных функций для выполнения различных операций над данными, таких как агрегатные функции (COUNT, SUM, AVG, MIN, MAX), функции преобразования данных (CONVERT, CAST), функции работы с датами и другие.

7. Комментарии

Комментарии используются для документирования запросов и игнорируются при выполнении. В SQL комментарии могут быть однострочными (начинаются с двойного дефиса --) или многострочными (заключаются в /**/).

Пример запроса

```sql
-- Это комментарий к запросу
SELECT column1, column2
FROM table_name
WHERE condition = 'value'
GROUP BY column1
HAVING COUNT(column2) > 10;
```

Правила

- Запросы SQL чувствительны к регистру.
- Каждая инструкция SQL должна заканчиваться точкой с запятой (😉.
- Операторы и ключевые слова могут быть разделены пробелами или переводами строки.
- Строковые значения обычно заключаются в одинарные кавычки (' '), но некоторые СУБД также поддерживают двойные кавычки (" ").
- При указании идентификаторов или строковых значений, содержащих пробелы или специальные символы, их следует заключать в кавычки и использовать экранирование при необходимости.
- Некоторые ключевые слова могут быть зарезервированными и не могут использоваться в качестве идентификаторов без экранирования.
- Вложенные запросы должны быть корректно синтаксически закрыты и оформлены.

Это основные правила и элементы синтаксиса SQL, которые помогают составить правильные и эффективные запросы к базе данных.

SELECT

Команда **SELECT** используется для выбора данных из одной или нескольких таблиц базы данных. Она позволяет указать столбцы, которые необходимо извлечь, и условия для фильтрации данных.

```sql
SELECT column1, column2 FROM table_name WHERE condition;
```

DISTINCT

Команда **DISTINCT** применяется вместе с **SELECT** для выбора уникальных значений из указанных столбцов.

```sql
SELECT DISTINCT column1, column2 FROM table_name;
```

FROM

Команда **FROM** указывает таблицу или таблицы, из которых происходит выборка данных.

```sql
SELECT column1, column2 FROM table_name;
```

JOIN

Команда **JOIN** объединяет строки из двух и более таблиц на основе условия, указанного после **ON**.

```
SELECT column1, column2 FROM table1 JOIN table2 ON table1.column =
table2.column;
```

LEFT JOIN

Команда **LEFT JOIN** объединяет строки из левой таблицы с соответствующими строками правой таблицы. Если нет соответствия, возвращается NULL.

```
SELECT column1, column2 FROM table1 LEFT JOIN table2 ON table1.column =
table2.column;
```

RIGHT JOIN

Команда **RIGHT JOIN** работает аналогично **LEFT JOIN**, но возвращает все строки из правой таблицы, даже если нет соответствия в левой.

```
SELECT column1, column2 FROM table1 RIGHT JOIN table2 ON table1.column =
table2.column;
```

ON

Ключевое слово **ON** используется с командой **JOIN** для указания условия объединения таблиц.

```
SELECT column1, column2 FROM table1 JOIN table2 ON table1.column =
table2.column;
```

WHERE

Команда **WHERE** применяется для фильтрации результатов запроса на основе указанных условий.

```
SELECT column1, column2 FROM table_name WHERE condition;
```

AND

Оператор **AND** используется в команде **WHERE** для объединения нескольких условий. Он требует, чтобы оба условия были истинными.

```
SELECT column1, column2 FROM table_name WHERE condition1 AND condition2;
```

OR

Оператор **OR** также используется в команде **WHERE** для объединения условий, но требует, чтобы хотя бы одно из условий было истинным.

```
SELECT column1, column2 FROM table_name WHERE condition1 OR condition2;
```

AS

Ключевое слово **AS** используется для присвоения псевдонима столбцу или таблице в результирующем наборе данных.

```
SELECT column1 AS alias1, column2 AS alias2 FROM table_name;
```

COUNT

Функция **COUNT** используется для подсчета числа строк в результирующем наборе данных.

```
SELECT COUNT(column1) FROM table_name;
```

GROUP BY

Команда **GROUP BY** используется для группировки строк на основе значений столбца. К общим данным применяются агрегатные функции, такие как **COUNT**, **SUM**, **AVG** и другие.

```
SELECT column1, COUNT(column2) FROM table_name GROUP BY column1;
```

HAVING

Команда **HAVING** применяется после **GROUP BY** для фильтрации групп, основываясь на агрегатных функциях.

```
SELECT column1, COUNT(column2) FROM table_name GROUP BY column1 HAVING
COUNT(column2) > 10;
```

NOT IN

Оператор **NOT IN** используется в команде **WHERE** для исключения строк, соответствующих заданным значениям.

```
SELECT column1 FROM table_name WHERE column1 NOT IN (value1, value2,
value3);
```

CREATE TABLE

Команда **CREATE TABLE** используется для создания новой таблицы в базе данных.

```
CREATE TABLE table_name (
    column1 INT,
    column2 VARCHAR(255),
    PRIMARY KEY (column1)
);
```

IF NOT EXISTS

Команда **IF NOT EXISTS** применяется с **CREATE TABLE** для предотвращения создания таблицы, если она уже существует.

```
CREATE TABLE IF NOT EXISTS table_name (
    column1 INT,
    column2 VARCHAR(255),
    PRIMARY KEY (column1)
);
```

INT

Тип данных **INT** используется для хранения целочисленных значений.

```
CREATE TABLE table_name (
    column1 INT,
    column2 INT
);
```

PRIMARY KEY

Ключевое слово **PRIMARY KEY** используется для указания первичного ключа в таблице.

```
CREATE TABLE table_name (
    column1 INT PRIMARY KEY,
```

```
    column2 VARCHAR(255)
);
```

AUTO_INCREMENT

Ключевое слово **AUTO_INCREMENT** используется для автоматического увеличения значения первичного ключа при вставке новой строки.

```
CREATE TABLE table_name (
    id INT PRIMARY KEY AUTO_INCREMENT,
    column1 VARCHAR(255)
);
```

VARCHAR

Тип данных **VARCHAR** используется для хранения строк переменной длины.

```
CREATE TABLE table_name (
    column1 VARCHAR(255),
    column2 VARCHAR(100)
);
```

NOT NULL

Ключевое слово **NOT NULL** указывает, что столбец не может содержать NULL.

```
CREATE TABLE table_name (
    column1 INT NOT NULL,
    column2 VARCHAR(255) NOT NULL
);
```

CONSTRAINT

Ключевое слово **CONSTRAINT** используется для определения ограничений на столбцы таблицы, такие как уникальность или внешний ключ.

```
CREATE TABLE table1 (
    column1 INT PRIMARY KEY,
    column2 VARCHAR(255)
);

CREATE TABLE table2

 (
```

```
    column3 INT,
    column4 VARCHAR(255),
    CONSTRAINT fk_column3 FOREIGN KEY (column3) REFERENCES table1(column1)
);
```

FOREIGN KEY

Ключевое слово **FOREIGN KEY** используется для создания внешнего ключа, связывающего столбец с другой таблицей.

```
CREATE TABLE table1 (
    column1 INT PRIMARY KEY,
    column2 VARCHAR(255)
);

CREATE TABLE table2 (
    column3 INT,
    column4 VARCHAR(255),
    CONSTRAINT fk_column3 FOREIGN KEY (column3) REFERENCES table1(column1)
);
```

REFERENCES

Ключевое слово **REFERENCES** указывает на таблицу и столбец, на который ссылается внешний ключ.

```
CREATE TABLE table1 (
    column1 INT PRIMARY KEY,
    column2 VARCHAR(255)
);

CREATE TABLE table2 (
    column3 INT,
    column4 VARCHAR(255),
    CONSTRAINT fk_column3 FOREIGN KEY (column3) REFERENCES table1(column1)
);
```