Записки инженера

Доступным языком заметки по IT технологиям

Структура и синтаксис SQL (выжимка)

Данный пост был написан как памятка по основным операторам языка SQL и представляет собой “выжимку” из официальной англоязычной документации. Перед изучением структуры и синтаксиса SQL рекомендую прочесть следующие посты:

Если вы только знакомитесь с БД, то рекомендую прочесть пост:

Теоретические азы БД (введение в SQL)

Для того, чтобы использовать язык SQL необходима СУБД, которая будет его исполнять. В качестве такой СУБД, мы будем рассматривать MySQL, как наиболее популярную в web разработке. Если вам нужна помощь в установки MySQL под Windows или FreeBSD, то рекомендую прочесть пост:

Установка MySQL

После установки любой СУБД, первым делом возникает вопрос, а как собственно с ней работать? С чего и как посылать SQL запросы, как добавить или удалить пользователя, изменить пароль, сделать резервную копию и восстанавливается из неё. Ответы на такого рода

вопросы вы найдете в посте:

Как работать с MySQL?

Теперь перейдем к сути поста – “структура и синтаксис SQL”.

Если вам лень читать весь пост, то для быстрого поиска, я привел список вопросов на которые вы найдете ответы

- Какие бывают типы данных у MySQL?
- Как создать БД?
- Как удалить БД?
- Как создать таблицу?
- Как удалить таблицу?
- Как создать таблицу и указать столбцу значение по умолчанию?
- Как создать таблицу с первичным ключом?
- Как создать две таблицу, что-бы первичный ключ одной ссылался на внешней другой?
- Как переименовать таблицу?
- Как добавить в таблицу столбец?
- Как удалить в таблице столбец?
- Как изменить тип столбца в таблице?
- Как изменить значение столбца в таблице? (например прибавить единицу)
- Как изменить первичный ключ в таблице?
- Как создать процедуру?
- Как создать функцию?
- Как создать триггер?
- Как создать событие?
- Как создать представление?
- Как вставить строку в таблицу?
- Как вставить данные в таблицу из другой таблицы?
- Как произвести выборку записей из таблицы?
- Как удалить все строки в таблице?
- Как использовать транзакцию?

Язык SQL (в реализации MySQL)

Откуда я беру информацию? (при нажатие – раскрывается блок текста)

1. Типы данных в SQL

Типы данных, которыми мы будем манипулировать, могут варьироваться в зависимости от СУБД. В нашем случае в качестве СУБД мы рассматриваем **MySQL**, т.к. данная система, наиболее часто используемая под web разработку.

Типы данных MySQL можно разделить на группы:

- числовые типы с фиксированной точкой

- **TINYINT** – целое число (**+-128**), занимаемый размер 1 байт

- **SMALLINT** – «малое» целое число (**+-32768**), занимаемый размер 2 байт

- **MEDIUMINT** – «среднее» целое чило (**+-8388608**), занимаемый размер 3 байт

- **INTEGER** или **INT** – «обычное» целое число (**+-2147483648**), занимаемый размер 4 байт

- **BIGINT** – «большое» целое число (**+-9223372036854775808**), занимаемый размер 8 байт

- **DECIMAL (x,y)** – число, в котором x разрядов в числе, y знаков после запятой

- **NUMERIC (x,y)** в MySQL это тоже самое, что и DECIMAL

Если перед именем типа использовать ключевое слово **UNSIGNED**, то **величина** для числового типа будет только **положительной**, причем **максимальное положительное число** будет **больше в два раза прежнего**, т.е. диапазон сдвинется, например INT это от -2147483648 до +2147483647, а UNSIGNED INT от 0 до 4294967295 .

- числовые типы с плавающей точкой

- **REAL** – вещественное число, размером 8 байт

- **DOUBLE PRECISION** или **DOUBLE** - в MySQL такое же вещественное число, как и REAL, размер 8 байт

- **FLOAT (x)** – вещественное число, точность указывается в скобках в байтах, если ничего не указать то точность – 4 байт, можно также использовать вариант FLOAT (x,y), где x – так же точность в байтах, а y количество знаком после запятой.

При записи чисел большого размера, чем это могу позволить типы с плавающей точкой, значения будут округлены.

- строковый тип(символьный)

- **CHAR (x)** – строка с длинной x (**от 1 до 255**), если значения меньше этой длины, то оно заполняется пробелами, если больше урезается, выделяемый размер под данный тип x байт.

- **VARCHAR (x)** – строка с длинной x (**от 1 до 255**), длина строки зависит от длины значения, пробелами ничего не заполняется, если значение больше заданной длины x, то оно, как и в CHAR урезается, выделяемый размер под данный тип зависит от длины значения.

- **TINYTEXT** или **TINYBLOB** строка с максимальной длиной текста **255**, занимаемый размер зависит от длины текста (максимум 255 байт)

- **TEXT** или **BLOB** строка с максимальной длиной текста **65535**, занимаемый размер зависит от длины текста (максимум 65535 байт)

Отличие TEXT от BLOB заключается только в том, что в TEXT при сравнение не учитывается регистр (заглавная или маленькая буква).

- **MEDIUMTEXT** или **MEDIUMBLO** – строка с максимальной длиной текста **16777215**, занимаемый размер зависит от длины текста (максимум 16777215 байт)

- **LONGBLOB** или **LONGTEXT** – строка с максимальной длиной текста **4294967295**, занимаемый размер зависит от длины текста (максимум 4294967295 байт)

- **ENUM ('value1','value2',…)** – символьный тип перечисления, может принимать значение из списка допустимых значений, максимум 65535 байт

- **SET (‘value1′,’value2′,…)** – символьный тип – множество, которое содержит значения указанные в скобках, максимум 64 значения, максимальный занимаемый размер 64.

- дата и время

- **DATETIME** – дата и время в формате **YYYY-MM-DD HH:MM:SS**, диапазон допустимых значений от ’1000-01-01 00:00:00′ до ’9999-12-31 23:59:59′, занимаемый объем 8 байт.

- **DATE** – дата в формате **YYYY-MM-DD**, диапазон допустимых значений от ’1000-01-01′ до ’9999-12-31′, занимаемый объем 3 байта.

- **TIMESTAMP** – дата и время в формате **YYYYMMDDHHMMSS**, занимаемый объем 4 байт.

- **TIME** – время в формате **HH:MM:SS**, диапазон допустимых значений от ’-838:59:59′ до ’838:59:59′, занимаемый объем 3 байта.

- **YEAR** – год в формате **YYYY**, занимаемый объем 1 байт.

2. Синтаксис языка SQL

Синтаксис языка SQL традиционно делят на три части:

1) Первая часть языка SQL отвечающая за определение данными (Date Definition Language или DDL), в неё входят такие действия, как:

- создать БД;
- удалить БД;
- создать таблицу;
- изменить таблицу;
- удалить таблицу;
- создать первичный/внешний ключ;
- удалить первичный/внешний ключ.

С помощью DDL задают схему БД (Что такое схема БД? – статья в википедии).

2) Вторая часть языка SQL отвечающая за манипуляцию данными (Date Manipulation Language или DML), в неё входят такие действия, как:

- вставить запись (строку) в таблицу;
- выбрать запись в таблице;
- обновить запись в таблице;
- удалить запись в таблице.

3) Третья часть языка SQL отвечающая за управлениями данными (Data Control Language или DCL), в неё входят такие действия, как:

- управление транзакциями;
- управление правами доступа.

Если кратко, то данная часть языка нужна для администрирования БД.

Мы будем рассматривать данные части SQL по очереди, начиная с 1-ой

2.1 Часть языка SQL отвечающая за определение данными (DDL)

1. Создаем БД

```
create database имя_бд;
```

2. Изменяем БД

Под изменением БД обычно подразумевается указание новой кодировки, подробнее об этом вы можете прочесть в официальной документации – ссылка)

3. Удаляем БД

```
drop database имя_бд;
```

4. Создаем таблицы

Перед тем как создавать таблицу, нужно выбрать в какой БД мы будем это делать (даже если вы только что создали БД это не означает что она выбрана как текущая), посмотреть список вы можете с помощью команды:

```
show databases;
```

А выбрать в какой БД мы будем создавать таблицы с помощью команды:

```
use имя_бд;
```

Теперь вы может посмотреть какие таблицы, есть в данной БД

```
show tables;
```

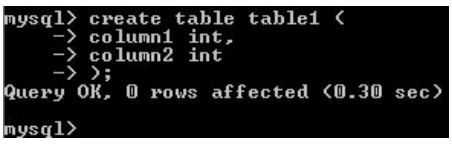
Создадим таблицу в выбранной БД

```
CREATE TABLE имя_т
```

Пример:

```
CREATE TABLE table1
```

Кстати вводить всю команду не обязательно в одну строку, можно через Enter



При создание таблицы, можно указать некоторые необязательные параметры, я не буду приводить их полный список (его вы можете посмотреть в документации – ссылка). Я продемонстрирую лишь некоторые примеры:

Пример №1, создаем таблицу **table1** с указанием, что столбец **COLUMN1** не может принимать значения **NULL**

```
CREATE TABLE table1
```

Если вы не указываете «notnull», то по умолчанию столбец будет принимать значение «NULL».

Пример №2, создаем таблицу **table1** с указанием для столбца **COLUMN1** значение по умолчанию **123**

```
CREATE TABLE table1
```

Пример №3, создаем таблицу **table1** с указанием что столбец **COLUMN1** является «**первичным ключом**»

```
CREATE TABLE table1
```

Таблица может иметь только один первичный ключ.

Кстати, столбцу с ключом можно указать, что бы его значение при каждой новой записи увеличивались на один (т.е. инкрементировались).

```
CREATE TABLE table1
```

Теперь каждая строка в столбце column1 будет нумероваться автоматически с 1

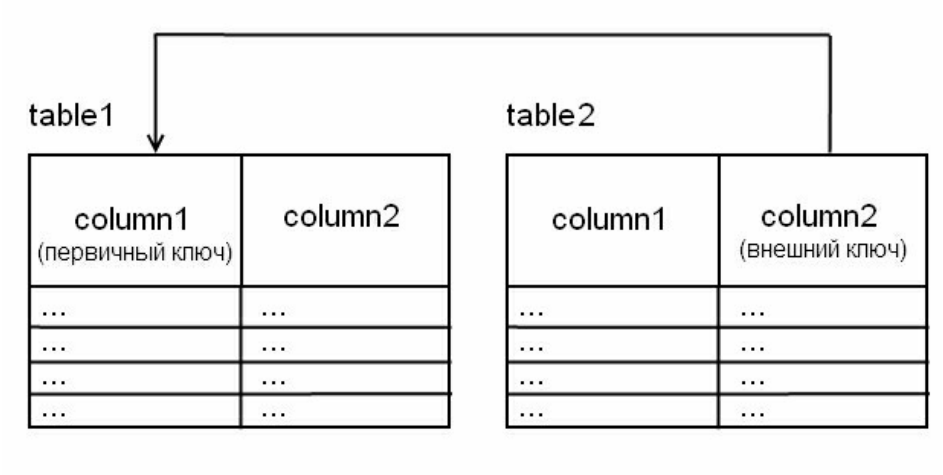
Пример №4, создаем таблицу **table1** с тремя столбцами **COLUMN1, COLUMN2, COLUMN3** и указываем, что столбец **COLUMN1** и **COLUMN2** являются составным первичным ключом.

```
CREATE TABLE table1
```

и посложнее..

Пример №5. создаем две таблицы **table1** и **table2**, с двумя колонками **COLUMN1** и **COLUMN2**. Укажем в таблице **table1** что столбец **COLUMN1** будет **первичным ключом**, а в таблице **table2**, что столбец **COLUMN2** будет **внешним ключом**, причем **ссылающийся на первичный ключ COLUMN1** таблицы **table1**.

Визуально представить это можно так:




```
CREATE TABLE table1
CREATE TABLE table2
```

5. Изменяем таблицы (ALTER)

Синтаксис команды ALTER для таблиц позволяет сделать достаточно много различных изменений для уже созданных таблиц, почитать поподробнее об этом вы сможете в официальной документации – ссылка). Я продемонстрирую лишь некоторые типовые примеры:

Пример №1, переименовываем таблицу table1 на table2

```
ALTER TABLE table1
```

Пример №2, добавляем в таблицу table1 столбец «name_new_column» с типом int

```
ALTER TABLE table1
```

Пример №3, добавляем в таблицу table1 столбец «name_new_column» с типом int, причем столбец не может принимать значение NULL

```
ALTER TABLE table1
```

Пример №4, добавляем в таблицу table1 столбец «name_new_column» с типом int, причем столбец по умолчанию имеет значение «1»;

```
ALTER TABLE table1
```

Пример №5, удаляем из таблицы table1 столбец name_new_column

```
ALTER TABLE table1
```

Пример №6, указываем что столбец name_new_column таблицы table1 будет иметь тип BIGINT;

```
ALTER TABLE  table1
```

Пример №7, указываем что столбец name_new_column таблицы table1 может (не может) принимать значение NULL

```
ALTER TABLE  table1
ALTER TABLE  table1
```

(если вы хотите, что бы тип столбца остался прежним укажите его текущий тип)

Пример №8, указываем что столбец name_new_column таблицы table1 имеет значение по умолчанию «1»

```
ALTER TABLE  table1
```

(если вы хотите, что бы тип столбца остался прежним укажите его текущий тип)

Пример №9, указываем в таблице table1, что первичный ключ теперь это столбец name_new_column

```
ALTER TABLE  table1
```

(если у вас уже есть первичный ключ в таблице, то его необходимо удалить:

ALTER TABLE table1 DROP PRIMARY KEY)

6. Удаляем таблицу

```
DROP TABLE имя_табл
```

P.S. В MySQL помимо БД и таблиц есть и другие сущности, знать о которых полезно. Я не буду подробно расписывать использование каждой из них, дам лишь общее описание с примером и ссылку на англоязычную официальную документацию.

Итак, в MySQL помимо БД и таблиц еще есть:

Процедуры (PROCEDURE) – это некий **SQL код**, который мы можем использовать многократно. Процедуры в MySQL похожи на «обыкновенные» процедуры языков высокого уровня. Они также могут иметь ветвления (if, case) в своем теле и иметь входные и выходные данные-параметры. Пример, создадим процедуру **test()**:

Выделяем БД, для которой мы хотим создать процедуру

```
use имя_БД;
```

Создаем процедуру **test** – которая выводит список таблиц имя_БД

```
delimiter //
create  procedure test ()
begin
show tables;
end//
delimiter ;
```

Где

delimiter – нужен для удобства объявления процедуры, первая строка «delimiter //», означает, что символ окончания строки будет «//», это дает нам возможность писать «;» в теле процедуры (между begin и end). Последняя строка «delimiter ;», говорит что символ «;» как и раньше будет означать окончания строки.

create procedure – ключевое слово, обозначающие что будет создаваться процедура

test – имя процедуры

() – список параметров процедуры через запятую, параметры могут быть входные, тогда перед именем параметра пишется IN, выходные OUT и входные/выходные одновременно, тогда – INOUT. Пример параметров:

(IN count int) – где count имя параметра, a int его тип.

begin – начало SQL кода процедуры, end конец кода

show tables – код процедуры, вывести список таблиц

После того как мы создали процедуру, мы её можем вызывать с помощью команды **call**:

```
call test();
```

В результате процедура выведет список таблиц.

Удалить процедуру мы можем с помощью команды **drop**:

```
drop procedure test;
```

P.S. Посмотреть список уже созданных процедур вы можете с помощью phpMyAdmin, зайдя на вкладку «Процедуры»

Для более подробного ознакомления с процедурами читайте в англоязычной официальной документации

Функции (FUNCTION) – похоже на процедуру, это некий **SQL код**, который мы хотим использовать многократно. Отличие функции от процедуры заключается в том, что функция возвращает **не больше одного параметра** и что функция вызывается **по имени, без «call»**. Поэтому функцию можно использовать в SQL запросах. Пример, создадим функцию **test2()**;

Сначала выделим БД, для которой мы хотим создать процедуру

```
use имя_БД;
```

И создадим функцию **test2(var char(20))** – которая выводит контрольную сумму по алгоритму **CRC32** от переменной с именем **var**:

```
delimiter //
create function test2 (var
returns bigint
begin
return CRC32(var);
end //
delimiter;
```

Где

create function – ключевое слово, обозначающее что будет создаваться функция

test2 – имя функции

(var char(20))– список параметров функции, параметры в отличие от процедуры могут быть только входные, var – имя параметра, char(20) его тип. Если параметров несколько, то между ними ставиться запятая.

returns bigint – returns ключевое слово, после которого указывается тип возвращаемого результата функции, в нашем случае bigint («большой int»)

begin – начало SQL кода процедуры, end конец кода

return – ключевое слово после которого указывается что должна вернуть функция, в нашем случае должен быть возвращен результат работы функции CRC32 от переменной var. В коде функции всегда должен быть указан return.

После того как мы создали функцию, мы её можем вызывать с помощью оператора выбора – **SELECT**:

```
SELECT test2();
```

В результате функция выведет **CRC32 код от параметра var**

Удаляется функция с помощью команды

```
drop function test2;
```

P.S. Посмотреть список уже созданных функций вы можете с помощью phpMyAdmin, зайдя на вкладку «Процедуры» (функции это процедуры с типом – Function)

Для более подробного ознакомления с процедурами читайте в англоязычной официальной документации

Триггеры (TRIGGER) – это некий кусок **SQL кода**, который **относиться к таблице** и **выполняется каждый раз когда происходит то или иное событие в данной таблице**. События может быть три:

- **INSERT** (добавление чего-то в таблицу);
- **UPDATE** (изменение чего-то в таблице);
- **DELETE** (удаление чего-то в таблице).

Помимо этого можно указать когда срабатывать триггеру «до события» – **BEFORE** или «после» – **AFTER**. «До события» означает, что

событие произошло, но действия с таблицей еще не произведены.

Пример создадим триггер **test_t** для таблицы **table1**:

Сначала выделим БД, с которой мы будем работать

```
use имя_БД;
```

Пусть в данной БД будет таблица **table1** с двумя столбцами **value** и **password**

```
CREATE TABLE table1
```

Создадим триггер **test_t** – который будет **срабатывать при добавление новых строк в таблицу**, задача триггера **хешировать значение в столбце password**.

```
delimiter //
create trigger test_t
before insert on table1
for each row
begin
set NEW.password=md5
end //
delimiter ;
```

Где

create trigger – ключевое слово, обозначающие что будет создаваться триггер

test_t – имя триггера

before insert on table1 – означает что триггер будет срабатывать до добавления новых записей в таблицу table1

for each row – для каждой строки

begin – начало SQL кода процедуры, end конец кода

set NEW.password=md5(NEW.password); – SQL код триггера, он хеширует значения столбца password.

После того как мы создали триггер, мы можем его проверить добавив новую строку в таблицу **table1**.

Что бы добавить новую строку надо вести следующий SQL код:

```
insert into table1 values (
```

(если данный код возвращает ошибку, попробуйте изменить кавычки)

Проверим содержимое таблицы, введем следующий SQL код:

```
select * from table1;
```

Результат:

```
mysql> select * from table1;
+-------+----------------------------------+
| value | password                         |
+-------+----------------------------------+
| Petr  | e735ab5dfcccb3166a5b568d2ed5b5dc |
+-------+----------------------------------+
1 row in set (0.00 sec)
```

Значение **password** говорит нам о том, что **триггер сработал**.

Удаляется триггер с помощью команды

```
drop trigger test_t;
```

P.S. Посмотреть список уже созданных триггеров вы можете с помощью phpMyAdmin, зайдя на вкладку «Триггеры»

Для более подробного ознакомления с процедурами читайте в англоязычной официальной документации

События (EVENT) – это некий кусок SQL кода, который выполняется в определенное время (или через определенный временной промежуток). События в MySQL это некий планировщик, как crontab в Linux. Пример, создадим событие test_e:

Сначала выделим БД, с которой мы будем работать

```
use имя_БД;
```

Пусть в данной БД будет таблица **table2** с двумя столбцами **date** и **value**

```
CREATE TABLE table2
```

Создадим событие **test_e**, которое **добавляет каждую минуту в таблицу table2 новую запись** с текущем временем и текстом «Testing event»

```
delimiter //
create event test_e
on schedule every 1 minu
do
begin
insert into table2 values(
end//
delimiter ;
```

Где

create trigger – ключевое слово, обозначающие что будет создаваться событие

test_e – имя события

on schedule every 1 minute – означает что событие будет срабатывать каждую минуту

begin – начало SQL кода процедуры, end конец кода

insert into table2 values(now(),'Testing event'); – SQL код события, он добавляет новую запись с текущем временем и текстом «Testing event» в таблицу table2.

Результат можно посмотреть с помощью следующей SQL команды:

```
select * from table2;
```

Пример результата, через 2 минуты

```
mysql> select * from table2;
+---------------------+---------------+
| date                | value         |
+---------------------+---------------+
| 2014-03-31 15:11:27 | Testing event |
| 2014-03-31 15:12:27 | Testing event |
+---------------------+---------------+
2 rows in set (0.00 sec)
```

Удаляется процедура с помощью команды

```
drop event test_e;
```

P.S. Посмотреть список уже созданных событий вы можете с помощью phpMyAdmin, зайдя на вкладку «События»

Для более подробного ознакомления с событиями читайте в а[англоязычной официальной документации]

Представление (VIEW) – это некая **виртуальная таблица**, которая формируется **из реальных таблиц БД**. За то, каким образом она формируется, отвечает **SQL код представления**.

Представления удобно использовать для **ограничения пользователей к столбцам и строкам таблиц**, достаточно пользователю разрешить работать с представлением и запретить работать с таблицей, ведь представление для пользователя это такая же таблица.

Также представления удобно использовать, когда надо **«забетонировать» некий сложный SQL запрос**, который объединяет много таблиц. Это позволит нам забыть его и работать уже с представлением как с результирующей таблицей.

Помимо этого, представление удобно для **разделения логики хранения данных в БД** и **приложения использующую данную БД** (например php скрипт), в этом смысле, представление выполняет роль некой прослойки, можно сказать «интерфейса». Мы можем изменить всю структуру БД и создать представления, формирующие соответствующие таблицы и ПО работающее с данной БД не заменит разницы.

Пример, создадим представление **test_v (c1,c2,c3)**:

Сначала выделим БД, с которой мы будем работать

```
use имя_БД;
```

Пусть в данной БД будет таблицы **table3_1** и **table3_2** с двумя столбцами **column1** и **column2**

```
CREATE TABLE table3
```

Пусть в таблице **table3_1** содержатся следующие три записи **(2.4), (3,4), (1,6)**

```
INSERT INTO table3_1
```

А в таблице **table3_2** следующие две записи **(3,5),(2,1)**

```
INSERT INTO table3_1
```

Создадим примитивное представление **test_e** с тремя столбцами **c1,c2 и c3,** которое объединит таблицу **table3_1** и **table3_2** (объединим таблицы по принципу «естественного соединение», что это такое смотри в посте «теоретические азы БД»)

```
create view test_v (c1, c2
as
select table3_1.*, table3_
```

Результат можно посмотреть, если мы обратимся к представлению

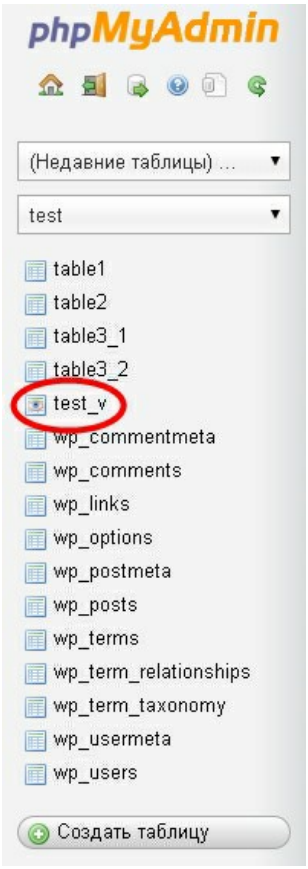
```
select * from test_v;
```

```
mysql> select * from test_v;
+------+------+------+
| c1   | c2   | c3   |
+------+------+------+
|    2 |    4 |    1 |
|    3 |    4 |    5 |
+------+------+------+
2 rows in set (0.03 sec)
```

Удаляется представление с помощью команды

```
drop view test_v;
```

P.S. Посмотреть список уже имеющихся представлений вы можете с помощью phpMyAdmin, представления располагаются вместе с таблицами в левом столбце, отличить их от таблиц можно с помощью пиктограммы «глаз».



Для более подробного ознакомления с представлениями читайте в англоязычной официальной документации

Индексы (INDEX) – …, про индексы могу сказать одно, что индекс это столбец таблицы и что индексы нужны для ускорения поиска строк. Примеры использования индексов я привести не смогу, т.к. сам их мало использовал.

Для более подробного ознакомления с индексами читайте в англоязычной официальной документации.

2.2 Часть языка SQL отвечающая за манипуляцию данными (DML)

Напомню, в DML входят такие действия, как:

- вставить запись (строку) в таблицу;
- выбрать запись в таблице;
- обновить запись в таблице;
- удалить запись в таблице.

1. Вставляем запись (строку) в таблицу (INSERT … VALUES)

```
INSERT INTO имя_табл
```

В данном примере, мы вставляем в таблицу «**имя_таблицы**» строку со столбцами «**столбец1**» и «**столбец3**». Значение для «**столбец1**» равняется «**значение1**», а для «**столбец3**» – «**значение2**». Пустые столбцы в данной строке равняются **NULL** или значению по умолчанию – **DEFAULT** (определяется свойствами таблицы).

Например, если применить данный SQL к следующей таблице:

столбец1	столбец2	столбец3
по умолчанию NULL	по умолчанию NULL	по умолчанию NULL

То результат будет следующий:

столбец1	столбец2	столбец3
значение1	по умолчанию NULL	значение2

2. Вставляем запись в таблицу из другой таблицы (INSERT … SELECT)

```
INSERT INTO имя_таб
```

После «…» вы можете использовать любой запрос на выборку к любой таблицы БД (кроме этой).

(Как работать с запросом на выборку, т.е. с SELECT … FROM, будет сказано ниже, пункт 3.)

Пример, нужно в **первый** и **второй** столбец **таблицы1** вставить значение **первого** и **третьего** столбца **таблицы2** (из определенной строки).

Таблицу «куда надо вставить», назовем её **table1**:

A	B
по умолчанию NULL	по умолчанию NULL

Таблицу «из которой надо получить», назовем **table2**;

A	B	C
a3	b1	c1
a2	b1	c4
a1	b1	c2

Повторю, что нам нужно сделать, но уже в терминах этих таблиц: «нужно в столбец **A** и **B** таблицы **table1** вставить значение столбца **A** и **C** таблицы **table2**, из строки с значением **A=a2**;»

```
INSERT INTO table1(A
```

Результат:

A	B
a2	c4

3. Выбираем записи из таблицы (SELECT …FROM)

Операция выборки из таблицы – самая распространённая, т.е. SELECT … FROM – самый часто используемый оператор.

Разберем как работать с SELECT … FROM на типовом примере, пусть есть таблица, назовем её **table3**

A	B	C
a3	b1	c1
a2	b1	c4
a1	b1	c2

Тогда следующий SQL код:

```
SELECT A,B FROM tat
```

Вернет следующую таблицу:

A	B
a3	b1

Данная выборка, самый распространённый вариант использования **SELECT … FROM**, вы часто с этим будете встречаться.

Если говорить в рамках реляционной алгебры (читайте в посте «азы реляционных БД»), то:

SELECT – отвечает за операцию «проекция» (выбор столбцов). Если указанно «A,B», то, это значит, что будет производиться проекция результирующей таблицы по столбцам A и B (т.е. эти столбцы останутся, а остальные нет).

FROM – отвечает за операцию «декартово произведение». Если указана одна таблица (table3 как в нашем случае), то ничего производиться не будет, значит «взять как есть». Если несколько через запятую, то результат FROM-а будет их декартово произведение.

Говоря просто, декартово произведение двух отношений это перебор кортежей первого отношения с каждым кортежем второго отношения.

WHERE – отвечает за операцию «селекция» (выборка строк). Если указанно A=a3, то, это значит, что селекция будет производиться при условие что A=a3, т.е. будет выбрана та строка, которая удовлетворяет данному условию.

Я рекомендую, читать данный SQL запрос нужно с **FROM**, далее **WHERE**, затем **SELECT**, пример для данного запроса:

Взять «как есть» таблицу **table3** (FROM), произвести с ней **селекцию** при условие что **A=a3** (WHERE), затем с этим результатом произвести **проекцию** по столбцам **A и B**.

Пример выше всего лишь типовой, оператор **SELECT … FROM** имеет много необязательных параметров, я не буду приводить их полный список, почитать о них вы можете на официальной англоязычной документации – ссылка.

Пример №1, самый простой, нужно взять всю таблицу **table1**;

```
SELECT * FROM table1
```

Эта самая короткая запись оператора SELECT ... FROM

Пример №2, нужно взять таблицу **table1** со столбцами **A,B**, причем повторяющиеся записи в результате не показывать.

```
SELECT DISTINCT A,B
```

Пример №3, нужно взять таблицу **table1** со столбцами **A,B**, причем с теми записями в которых значение столбца A начинаются с «Теле»

```
SELECT A,B FROM tab
```

Пример №4, нужно взять таблицу **table1** со столбцами **A,B**, причем с теми записями в которых значение столбца **A** начинаются и заканчивается определенным символами, например: 8 (965)xxx-xx01.

```
SELECT A,B FROM tab
```

Пример №5, нужно взять таблицу **table1** со столбцами **A,B** и со всеми записями, причем результат нужно отсортировать по столбцу **A** по алфавиту (или по возрастанию если численный тип)

```
SELECT A,B FROM tab
```

Если по убыванию – SELECT A,B FROM table1 ORDER BY A DESC;

Если нужно произвести выборку строки, т.е. использовать where то order by пишется после, пример: select A,B from table1 where A=a1 and A=a2 and A=a3 order by A;

Пример №6, есть таблица **table1** с столбцами (номер_заказа, ФИО) и таблица **table2** с столбцами (номер_заказа, дата). Нужно произвести естественное объединение данных двух таблиц (что это такое? – читайте в "основах реляционных БД").

```
SELECT * FROM table1
```

или

```
SELECT * FROM table1
```

(в документации сказано что «NATURAL JOIN» и «INEER JOIN с USING» эквивалентны)

В теле SELECT можно использовать встроенные функции MySQL (их список можно посмотреть тут)

Пример №7, пусть есть таблица **table1**, с тремя столбцами символьного типа.

Нужно взять таблицу **table1** с первым столбцом **A** и новым столбцом **newB**, который является результат склейки (объединения) столбцов **B** и **C**

```
SELECT A,CONCAT(B
```

*Ключевое слово «**as**» – означает что «то что слева (т.е. CONCAT(B,C)) считать как столбец newB».*

Пример №8, пусть есть таблица **table1**, с произвольным количеством столбцов, нужно узнать количество записей

```
SELECT COUNT(*) FR
```

4. Обновляем записи в таблице (UPDATE)

```
```

```
UPDATE имя_таблицы
```

Условие в WHERE нужно для выбора строки, в которой по данному столбцу нужно сделать изменения.

Если вам нужно изменить значение для всех строк данного столбца, то WHERE писать не надо:

```
UPDATE имя_таблицы
```

В условие может быть включен подзапрос, на пример:

Произведем изменение (прибавим единицу) к столбцу **column1** таблицы **table1** по тем строкам, которые есть в наличие в таблице **table2** в столбце **column1**

```
UPDATE table1 SET col
```

Где

Подзапрос (SELECT column1 FROM table2) – выводит список значений столбца **column1** таблицы **table2**

IN – условие, проверяет вхождение значений column1 в список (результат подзапроса)

И еще один простой примерчик…

Пример, прибавляем ко всем значениям столбца +1

```
UPDATE имя_таблицы
```

5. Удаляем записи в таблице (DELETE)

```
DELETE FROM имя_та
```

Условие в WHERE нужно для выбора строки, которую нужно удалить

Если вам нужно удалить все строки в данной таблице, то WHERE писать не надо:

```
DELETE FROM имя_та
```

В условие может быть включен подзапрос, на пример:

Удалим строки в таблице **table1** если значение в столбце **column1** повторяется со значением столбца **column2** – таблицы **table2**;

```
DELETE FROM table1 '
```

Где

Подзапрос (SELECT column2 FROM table2) – выводит список значений столбца **column2** таблицы **table2**

IN – условие, проверяет вхождение значений column1 в список (результат подзапроса)

3. Часть языка SQL отвечающая за управлениями данными (DataControlLanguage – DCL)

Напомню, в DCL входят такие действия, как:

- управление транзакциями;
- управление правами доступа.

Если кратко, то данная часть языка нужна для администрирования БД.

1. Управление транзакциями (START TRANSACTION …)

Транзакция это **некий SQL запрос**, который может быть в**ыполнен или полностью или вообще никак**. Решать выполнять SQL запрос или нет, зависит от вас (от вашего скрипта выполняющий SQL запрос). Механизм транзакций удобно использовать, когда вам нужны **гарантии при выполнение особо важных манипуляций с базой данных**, вы сначала выполняете SQL код, смотрите результат и решаете приводить его в окончательное действие или откатить обратно.

Как пример необходимости использования транзакции, часто приводиться случай с двумя таблицам в БД не которого банка. Допустим нам нужно вставить в «**вторую**» таблицу некоторые данные из «**первой**» таблицы, после чего эти данные в «**второй**» таблице нужно обнулить. Таким образом, мы переводим деньги с одного счета на другой. Что будет, если некоторая часть SQL запроса выполниться, после чего произойдет аппаратно-програмный сбой, а остальная часть нет? Возникнет неправильная ситуация, когда деньги переведены, но они останутся на «первом» счете. Вот для исключения таких ситуаций используется транзакция.

Транзакции поддерживают не все таблицы, например с транзакциями могут работать только таблицы типа **InnoDB** и **BDB**

По умолчанию в **MySQL**, все действия с таблицами сразу же сохраняются, т.е. «автоматически» транзакции не используются. Для использования транзакции, её необходимо объявлять с помощью оператора **START TRANSACTION** … .

```
START TRANSACTIO
SQL_запрос;
COMMIT или ROLLB
```

Оператор **COMMIT** или **ROLLBACK** используется в зависимости от удачного или не удачного выполнения SQL_запроса.

Если запрос выполнен **удачно** (не было сбоев и MySQL не вернул ошибку), то его необходимо завершить, для того чтобы все действия пришли в силу, то используем – **COMMIT**

Если запрос **не был выполнен удачно** (MySQL вернул ошибку), то его необходимо «откатить», для этого используем – **ROLLBACK**

Таким образом, транзакции позволяют выполнять SQL_запрос или полностью или вообще не как. Это поможет нам избежать ошибок, когда часть SQL запроса будет выполнена, а часть из сбоя нет. Транзакции это «**или все**» или «**ничего**».

Вызывать **COMMIT** или **ROLLBACK** нужно в зависимости от условия выполнение SQL запроса. Данное условие обычно реализуется в php скриптах.

2. Управление правами доступа

Элементарные операции по созданию и удалению аккаунтов пользователей и назначением им прав, вы можете посмотреть в разделе в посте «Как работать с MySQL?».

Подробное описание по управлению правами доступа вы можете найти в официальной англоязычной документации – тут

Вам будет интересно:

Буду признателен если вы поделитесь данным постом

Ваш комментарий

Имя *

Почта (скрыта) *

Сайт

Отправить

Ответ в цифрах

× четыре = тридцать шесть

Подписаться и читать нас можно в:

-
- Ок

Популярные посты

- Методика верстки шаблона под OpenCart (прочитали **52 262 раз**)
- Создаем модуль в OpenCart (прочитали **36 032 раз**)
- Добавляем новую страницу (схему) в OpenCart (прочитали **29 245 раз**)
- Переменные CMS OpenCart (прочитали **24 632 раз**)
- Отправить xml / json / jsonp методом POST / GET с помощью PHP / JavaScript, обработка и прием (прочитали **24 358 раз**)
- Структура OpenCart (прочитали **18 096 раз**)
- Методика верстки шаблона под OpenCart ч.2 (прочитали **16 042 раз**)
- Установка и удаление программ в FreeBSD, коллекция портов (ports), установка пакетов (tbz) (прочитали **12 904 раз**)
- Как примонтировать USB флешку (flash) (прочитали **10 833 раз**)
- Каталог (витрина) в 1С-Битрикс (прочитали **9 539 раз**)

Посты

- 1С-Битрикс
  - Методика верстки шаблона под 1С-Битрикс
  - Полезные мелочи в 1с-Битрикс
  - Формируем структуру сайта 1С-Битрикс и наполняем его содержимым
  - Как перенести шаблон сайта на другой хостинг с установленным 1С-Битрикс?
  - Корзина в 1С-Битрикс
  - Вертикальное меню «Производители» в 1С-Битрикс
  - Каталог (витрина) в 1С-Битрикс
  - Последние поступления в 1С-Битрикс
  - Слайдер в 1С-Битрикс
  - Специальное предложение в 1С-Битрикс
  - Подписка по e-mail в 1С-Битрикс
  - Форма поиска в 1С-Битрикс
  - Включаемые области в 1С-Битрикс
  - Вертикальное меню в 1С-Битрикс
  - Горизонтальное меню в 1С-Битрикс
  - Интеграция компонентов в 1С-Битрикс
  - Подбор компонентов для шаблона в 1С-Битрикс
  - Создаем скелет шаблона в 1С-Битрикс
  - Как установить 1С-Битрикс?
  - Обзор 1С-Битрикс, принцип работы
- Joomla
  - Забыли пароль администратора в joomla? (востановить/сбросить пароль joomla)

- OpenCart
- Создаем модуль в OpenCart 2.x (OCMOD)
- Методика верстки шаблона под OpenCart
- Методика верстки шаблона под OpenCart ч.2
- Как изменить логику формирование цены в OpenCart
- Динамическая цена товара в OpenCart
- Создаем модуль в OpenCart
- Структура OpenCart
- OpenCart, прикручиваем модель к контроллеру
- OpenCart, прикручиваем представление к контроллеру
- Добавляем новую страницу (схему) в OpenCart
- Opencart, добавляем настройки в панель администратора
- Переменные CMS OpenCart

- PHP, JavaScipt, HTML, CSS, SQL
- Структура и синтаксис SQL (выжимка)
- Теоретические азы БД (введение в SQL)
- ООП в PHP (общие понятия)
- Структура и синтаксис PHP (выжимка)
- Отправить xml / json / jsonp методом POST / GET с помощью PHP / JavaScript, обработка и прием
- Отлавливаем POST/GET запрос
- Наследование, каскадирование и специфичность CSS
- Структура и синтаксис JavaScript (выжимка)
- Внутренняя оптимизация сайта (SEO)

- WordPress
- Бэкап (резервное копирования) сайта на wordpress
- Как русифицировать WordPress
- Выравнивание текста по ширине wordpress
- Перенос сайта wordpress на другой хостинг
- Создание сайта на WordPress ч.1
- Создание сайта на WordPress ч.2
- Загрузка плагинов и тем в WordPress по ftp
- Установка WordPress по ssh

- Администрирование Linux/Unix/Windows
- SSH-доступ по ключу (FreeBSD)
- Как работать с MySQL?
- Полезные фрагменты shell скриптов (FreeBSD)
- Как примонтировать USB флешку (flash)
- Установка и удаление программ в FreeBSD, коллекция портов (ports), установка пакетов (tbz)
- Как сбросить пароль root на FreeBSD
- FTP сервер на FreeBSD
- Памятка – команды Linux и Unix
- Как запаролить директорию в Apache
- Основы безопасности lighttpd

- Ближе к железу ...
- Ассемблер для AVR (быстрый старт)
- Работаем с UART на AVR
- Понятия UART, RS-232 и COM
- Работаем с LCD дисплеем на основе микроконтроллера – HD44780 (ч.2)
- Работаем с LCD дисплеем на основе микроконтроллера – HD44780 (ч.1)
- Микроконтроллер – подключаем PS/2 клавиатуру
- Интерфейс SPI
- Микроконтроллер – работаем с SD картой без файловой системы
- Как подключить SD карту к микроконтроллеру? (примитивный способ)
- Микроконтроллер – работаем на SD карте с FAT16 на низком уровне
- Описание файловой системы FAT16
- Из .sre в .bin