Mysq

清屏：

**一：对数据库的操作**

**1-1修改mysql提示符：**

\D:完整的日期

\d:当前数据库

\h:服务器名称

\u:当前用户

Eg:命令：PROMPT \u@\h\d>

提示符即被修改为：root@localhost(none); 因为还没打开数据库，所以为none.

**1-2 mysql常用命令**

SELECT VERSION(); 显示当前服务器版本

SELECT NOW(); 显示当前时间

SELECT USER(); 显示当前用户

**1-3 mysql语句规范**

关键字与函数名称全部大写

数据库名称 表名称 字段名称全部小写

Sql语句分号结尾

**1-4 操作数据库**

**创建数据库**

CREATE{DATABASE } name CHARACTER SET charset\_name; 划线部分可有可无

**查看当前服务器下的数据列表**

SHOW DATEBASES 查看数据库

**修改数据库**：

ALTER DATABASE name CHARACTER SET charsetname;

**删除数据库**

DROP DATABASE name;

**二．对表的操作**

**2-1 数据类型之整型**（决定数据的存储格式）

TINYINT : 有符号值：-127~127；

无符号值：0~255；

字节：1；

SMALINT : 有符号值：-32768~32768（15）

无符号值：0到65535

字节：2

MEDIUMINT : 有符号值：（23）

无符号值：(24)

字节：3

INT : 有符号值：（31）

无符号值：（32）

字节：4

BIGINT : 有符号值：（63）

无符号值：（64）

字节：8

2-2 **浮点型**

FLOAT[(M,D)] M表示总的位数 D表示小数点之后的位数 可省去 (10的38次)

DOUBLE[(M,D)] 参数类同 (10的308次)

2-3 **日期时间**

省略········

此类型一般用数据表示

2-4 **字符型**

CHAR（M） M个字节 0到255

VARCHAR(M) 0到65535个字节

TINYTEXT 小于2的8次

TEXT 小于2的16次

MEDIUMTEXT 小于2的24次

LONGTEXT 小于2的32次

ENUM(value 1,value 2，·········) 1或2个字节，取决于枚举的个数（小于等于65535个）

SET(value 1,value 2，·········) 1，2，3，4或8个字节，取决于SET成员个数，最多64

2-5 **创建数据表**

CREATE TABLE name (列名 数据类型·········)

**查看数据列表**

SHOW TABLES [FROM 库名]

**查看数据表结构**

SHOW COLUMNS FROM name(表名)

**插入记录**：

INSERT name(表名) [(列名，列名，······)] VALUES(value,··········)

**记录查找：**

SELECT \* FROM name（表名）

2-6  **空值与非空值**（列属性的描述）

NULL; 字段值可以不写

NOT NULL; 字段值禁止为空

Eg:

Name VARCHAR(6) NOT NULL; (表构建时)

Age TINYINT UNSIGNED NULL; (允许为空/也可以不写，表示默认可为空值)

INSERT T1 VALUES( “JINYU”,NULL)（插入记录）

2-7 **自动编号**

AUTO\_INCREMENT

**自动编号必须与主键组合使用**

默认情况下，起始值为1，每次增量为1；

**主键**

PRIMARY KEY

主键约束

每张数据表只能有一个主键

主键保证记录的唯一性

主键自动为 NOT NULL;

主键的使用并不是一定需要 AUTO\_INCREMENT;

**2-8 约束**

1. **主键约束**

主键

PRIMARY KEY

主键约束

每张数据表只能有一个主键

主键保证记录的唯一性

主键自动为 NOT NULL;

主键的使用并不是一定需要 AUTO\_INCREMENT;

1. **唯一约束**

UNIQUE KEY

唯一约束可以保证记录的唯一性

唯一约束的字段可以为空值（NULL）

每张数据表可以存在多个唯一约束

1. **默认约束**

默认值

当插入记录时，如果没有明确的为字段赋值，则自动赋予默认值

DEFAULT 默认值

**三． 表的操作**

**3-1 约束**

约束保证数据的完整性和一致性

约束分为表级约束和列级约束

约束类型：  
 NOT NULL 非空约束

PRIMARY KEY 主键约束

UNIQUE KEY 唯一约束（不重复）

DEFAULT 默认约束

FOREGIN KEY 外键约束

**3-2 外键约束**

保持数据一致性

实现一对一或一对多的关系

要求：

父表和子表必须使用相同的存储引擎，禁止使用临时表

数据表的存储引擎只能为InnoDB

外键和参照列必须具有相似的数据类型，其中数字的长度，或是否有符号位必须相同，而字符的长度可以不同。

外键列和参照列必须创建索引，参照列无索引则MYSQL会自动创建索引，而外键列无索引，MYSQL则不会自动创建索引。

**查看存储引擎的方式：(表的详细信息)**

1. 创建数据表
2. **SHOW CREATE 表name;**

**主键会自动创建索引**

**查看是否具有索引：  
 SHOW INDEXES FROM name(表名) (表格形式)**

**SHOW INDEXES FROM name(表名) \g (网格形式)**

**3-3 外键约束的参照操作**

1. CASCADE 从父表删除或更新且自动删除或更新子表中匹配的行
2. SET NULL 从父表删除或更新行，并设置子表中的外键列为NULL，使用此选项必须保证子表没有指定的NOT NULL.
3. RESTRICT 拒绝对父表的删除或更新操作
4. NO ACTION 标准sql的关键字，在mysql中与RESTRICT相同

**实际开发中很少使用这种物理外键约束（引擎）**

**3-4 列级约束：**对一个数据列的约束

列级约束既可以在列声明时定义，也可以在声明后定义

**表级约束：**对列数大于或等于2的约束

表级约束只能在定义后声明

(NOT NULL 和 DEFAULT 没有表级约束)

**3-5** **修改数据表**

1. 添加单列

**ALTER TABLE name(表名) ADD name1(列名) 列定义 [FIRST] AFTER name2(列名)**

1. 添加多列

**ALTER TABLE name(表名) ADD （name1(列名) 列定义,··········）**

**3-6 删除列**

**ALTER TABLE name(表名) DROP name(列名)**

**删除多列**

**ALTER TABLE name(表名) DROP name1(列名) ， DROP name2(列名)··········**

**3-7添加主键约束**

**ALTER TABLE name（表名） ADD PRIMARY KEY (name1列名)**

**注意：name1不存在有空值得数据**

**删除主键约束**

**ALTER TABLE name(表名) DROP PRIMARY KEY**

**3-8添加外键约束**

**ALTER TABLE name(表名) ADD FOREGIN KEY (列名) REFERENCES name(表名) (列名)**

**删除外键约束**

**ALTER TABLE name(表名) DROP FOREIGN KEY NAME(外键约束名)**

**要删除外键约束，某一行则必须有 ON DELETE CASCADE;**

**3-9删除唯一约束**

**ALTER TABLE name(表名) DROP INDEX (列名)**

**3-10添加删除默认约束**

**ALTER TABLE name ALTER name(列名) {SET DEFAULT 默认值| DROP DEFAULT}**

**3-11修改列定义**

1. **ALTER TABLE NAME(表名) MODIFY NAME（列名) 列定义**

**[FIRST | AFTER 列名]**

1. **ALTER TABLE NAME(表名) CHANGE OLDNAME(列名) NEWNAME(新列名) 定义 [FIRST | AFTER NAME(列名) ]**

**(先增加主键，才能再加上auto\_increment);**

**3-12 修改数据表名**

**ALTER TABLE 表名 TO 新表名**

**四．数据表的操作**

**4-1 插入记录**

**INSERT 表名[（列名，········）] VALUES(expr | default)，（）·············**

**里面也可以写表达式**

**如果该列是自动编号类型，可以写NULL | DEFAULT，也是会自动编号的**

**如果该列是可以默认空值的，那么可以写 NULL | DEFAULT**

**4-2插入记录（每次只能插入一次）**

**INSERT 表名 SET 列名 ={expr | DEFAULT },···············**

**4-3 单表更新(更改语句中有错误的地方)**

**UPDATE 表名 SET 列名 ={expr | DEFAULT }, 列名={expr | DEFAULT }·········[WHERE 条件]**

**如果省略WHERE条件，则那列会全部更新**

**4-4 单表删除**

**DELETE FROM 表名 [WHERE 条件]**

**省去条件，删除所有的行。**

**存在WHERE 删除某行，或几行。**

4-5 查询表达式

SELECT expr;

每一个表达式表示想要的一列，必须有至少一个

每个列之间用英文逗号分开

星号\* 表示所有列。(列名.\*) 可以表示命名表的所有列

查询表达式可以使用 [AS]alias\_name为其赋予别名

别名可用于GROUP BY ,ORDER BY 或HAVING 子句

4-6

WHERE

条件表达式

对记录进行过滤，如果没有指定WHERE子句，则显示所有记录。

在WHERE表达式中，可以使用MYSQL支持的函数运算

4-7GROUP BY

查询结果分组

[GROP BY{列名 | position} [ASC | DESC], ··········]

ASC:升序（默认的）

DESC：降序

4-8 HAVING

分组条件：

[HAVING 条件（一般用函数）]

4-9 ORDER BY

对查询结果进行排序

[ORDER BY {col\_name | position} [ASC | DESC]········]

4-10 LIMIT

限制查询结果返回的数量

[LIMIT { [offset,] row\_count | row\_count OFFSET offset]

Eg:

SELECT \* FROM T5 LIMIT 2,2;

表示显示两行，从第三行开始

SELECT \* FROM T5 2;

表示显示开头两行

4-11 INSET

插入记录

INSERT 表名 SET 列名={expr | DEFAULT}

此方法可以使用子查询（SubQuery）

Eg:

INSERT T1（name） SELECT name FROM T2 WHERE AGE>=30;

将表2中的name字段添加进入表1中，且是满足年龄大于等于30 的。

四． 数据表操作

5-1 子查询

设置客户端的编码显示方式 ：SET NAMES GBK/UTF8;

子查询：嵌套在查询的内部，且必须始终出现在圆括号内

Eg:

DISTINCT、GROUPE BY 、 ORDER BY 、 LIMIT,函数等

子查询的外查询可以是：

SELECT , INSERT, UPDATE, SET, DO

子查询的返回值：

标量，一行，一列，或子查询

5-2使用比较运算符的子查询

比较运算符

=，>,<,>=,<>,!=,<=>

语法结构

**Operand comparison\_operator UBQUERY**

**SELECT ROUND(AVG(GOODS\_PRICE),2) FROM TB\_GOODS;**

**对平均值四舍五入取小数点后两位**

**AVG,MAX,MIN、COUNT、SUM为聚合函数**

ANY和SOME为大于其中一个就可以

ALL就是大于所有的

**用ANY SOME ALL 修饰的比较运算**

**Operand comparison\_operator ANY(SUBQUERY);**

**Operand comparison\_operator SOME (SUBQUERY);**

**Operand comparison\_operator ALL (SUBQUERY);**

5-3 使用[not] in子查询

语法结构：

Operand comparsion\_operrator [not] in (subquery)

=ANY 运算符与 IN等效

!=All或<>ALL运算符与NOT IN等效

**5-4 使用[NOT] EXISTS 将返回TRUE; 否则为FALSE(很少用)**

INSERT ··········· SELECT FROM

将查询结果写入数据表

5-6 多表更新（相当于非物理外键（适时更新））

1. INSERT ··········· SELECT FROM

将查询结果写入数据表

1. UPDATE 表参照关系 SET 列名={expr | DEFAULT}

[,列名={expr | default ],··········[WHERE 条件]

1. 语法结构

table\_reference

{[INNER | CROSS] JOIN | {LEFT | RIGHT} [OUTER] JOIN}

table\_reference

ON conditional\_expr

1. 连接类型（减少内存）

INNER JOIN

在mysql中，JOIN，CROSS JOIN和INNER JOIN是等价的

LEFT[OUTER] JOIN ,左外连接

RIGHT [ OUTER] JOIN,右外连接

5\_7多表更新一步到位（相当于非物理外键（适时更新））

创建数据表的同时，将查询结果写入数据表

CREATE TABLE [IF NOT EXISTS] TBL\_NAME

[CREATE\_DEFINITION,········]

Select\_statement

-- 通过CREATE...SELECT来创建数据表并且同时写入记录

-- SELECT brand\_name FROM tdb\_goods GROUP BY brand\_name;

CREATE TABLE tdb\_goods\_brands (

brand\_id SMALLINT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

brand\_name VARCHAR(40) NOT NULL

) SELECT brand\_name FROM tdb\_goods GROUP BY brand\_name;

-- 通过tdb\_goods\_brands数据表来更新tdb\_goods数据表(错误)

UPDATE tdb\_goods INNER JOIN tdb\_goods\_brands ON brand\_name = brand\_name

SET brand\_name = brand\_id;

-- Column 'brand\_name' in field list is ambigous

-- 正确

UPDATE tdb\_goods AS g INNER JOIN tdb\_goods\_brands AS b ON g.brand\_name = b.brand\_name

SET g.brand\_name = b.brand\_id;

-- 查看tdb\_goods的数据表结构

DESC tdb\_goods;

**5-8 连接**

**Mysql在select 语句,多表更新，多表删除语句中支持JOIN操作**

**语法结构：** table\_reference

{[INNER | CROSS] JOIN | {LEFT | RIGHT} [OUTER] JOIN}

table\_reference

ON conditional\_expr

数据表参照

table\_reference

tbi\_name[[AS] alias] | table\_subquery [AS] alias

数据表可以使用tbi\_name AS alias

或tbi\_name alias——name赋予别名

Table\_subquery可以作为子查询使用在FROM子句中

这样的子查询必须为其赋予别名

5-9 连接类型

INNER JOIN 内链接

在mysql中，JOIN，CROSS JOIN和INNER JOIN是等价的

**显示左表和右表符合连接条件的记录（公共的部分）**

LEFT[OUTER] JOIN ,左外连接

显示左表的全部记录及右表符号连接条件的记录

RIGHT [ OUTER] JOIN,右外连接

显示右表的全部记录及左表符合连接条件的记录

5-10 连接条件

使用ON关键字来设定连接条件，也可以使用WHERE来代替

通常使用ON关键字来设定连接条件

使用WHERE关键字进行结果集记录的过滤

***触发器***：

[所有分类](http://www.open-open.com/lib/list/all)  >  [数据库相关](http://www.open-open.com/lib/list/90)  >  [数据库服务器](http://www.open-open.com/lib/list/91)  >  [MySQL](http://www.open-open.com/lib/list/128)

**MySQL 触发器例子（两张表同步增加和删除）**

[MySQL](http://www.open-open.com/lib/tag/MySQL)   2014-12-21 19:21:42 发布

|  |  |  |  |
| --- | --- | --- | --- |
| **您的评价**: |  | 0.0 |  |

     0收藏

其中old表示tab2（被动触发），new表示tab1（主动触发，外部应用程序在此表里执行insert语句）

**例子1：**  
**创建两个表，目的是在一个表里添加一条记录，另一个表也添加一条记录：**  
DROP TABLE IF EXISTS tab1;  
CREATE TABLE tab1(  
tab1\_id varchar(11)  
);

DROP TABLE IF EXISTS tab2;  
CREATE TABLE tab2(  
tab2\_id varchar(11)  
);

创建触发器:t\_afterinsert\_on\_tab1  
作用：增加tab1表记录后自动将记录增加到tab2表中  
delimiter ||   
DROP TRIGGER IF EXISTS t\_afterinsert\_on\_tab1 ||  
CREATE TRIGGER t\_afterinsert\_on\_tab1   
AFTER INSERT ON tab1  
FOR EACH ROW  
BEGIN  
insert into tab2(tab2\_id) values(new.tab1\_id);  
END||  
delimiter ;

测试：  
INSERT INTO tab1(tab1\_id) values('0001');

查看结果：  
SELECT \* FROM tab1;  
SELECT \* FROM tab2;

**例子2：**  
**创建两个表，目的是在一个表里删除一条记录，另一个表也删除一条记录：**  
delimiter ||   
DROP TRIGGER IF EXISTS t\_afterdelete\_on\_tab1||  
CREATE TRIGGER t\_afterdelete\_on\_tab1  
AFTER DELETE ON tab1  
FOR EACH ROW  
BEGIN  
delete from tab2 where tab2\_id=old.tab1\_id;  
END||

测试：  
DELETE FROM tab1 WHERE tab1\_id='0001';

看看结果  
SELECT \* FROM tab1;  
SELECT \* FROM tab2;

其中old表示tab2（被动触发），new表示tab1（主动触发，外部应用程序在此表里执行insert语句）

**例子1：**  
**创建两个表，目的是在一个表里添加一条记录，另一个表也添加一条记录：**  
DROP TABLE IF EXISTS tab1;  
CREATE TABLE tab1(  
tab1\_id varchar(11)  
);

DROP TABLE IF EXISTS tab2;  
CREATE TABLE tab2(  
tab2\_id varchar(11)  
);

创建触发器:t\_afterinsert\_on\_tab1  
作用：增加tab1表记录后自动将记录增加到tab2表中  
delimiter ||   
DROP TRIGGER IF EXISTS t\_afterinsert\_on\_tab1 ||  
CREATE TRIGGER t\_afterinsert\_on\_tab1   
AFTER INSERT ON tab1  
FOR EACH ROW  
BEGIN  
insert into tab2(tab2\_id) values(new.tab1\_id);  
END||  
delimiter ;

测试：  
INSERT INTO tab1(tab1\_id) values('0001');

查看结果：  
SELECT \* FROM tab1;  
SELECT \* FROM tab2;

**例子2：**  
**创建两个表，目的是在一个表里删除一条记录，另一个表也删除一条记录：**  
delimiter ||   
DROP TRIGGER IF EXISTS t\_afterdelete\_on\_tab1||  
CREATE TRIGGER t\_afterdelete\_on\_tab1  
AFTER DELETE ON tab1  
FOR EACH ROW  
BEGIN  
delete from tab2 where tab2\_id=old.tab1\_id;  
END||

测试：  
DELETE FROM tab1 WHERE tab1\_id='0001';

看看结果  
SELECT \* FROM tab1;  
SELECT \* FROM tab2;

**创建触发器**  
在MySQL中，创建触发器语法如下：

复制代码 代码如下:

CREATE TRIGGER trigger\_name  
trigger\_time  
trigger\_event ON tbl\_name  
FOR EACH ROW  
trigger\_stmt

其中：

trigger\_name：标识触发器名称，用户自行指定；  
trigger\_time：标识触发时机，取值为 BEFORE 或 AFTER；  
trigger\_event：标识触发事件，取值为 INSERT、UPDATE 或 DELETE；  
tbl\_name：标识建立触发器的表名，即在哪张表上建立触发器；  
trigger\_stmt：触发器程序体，可以是一句SQL语句，或者用 BEGIN 和 END 包含的多条语句。

由此可见，可以建立6种触发器，即：BEFORE INSERT、BEFORE UPDATE、BEFORE DELETE、AFTER INSERT、AFTER UPDATE、AFTER DELETE。

另外有一个限制是不能同时在一个表上建立2个相同类型的触发器，因此在一个表上最多建立6个触发器。

**trigger\_event 详解**  
MySQL 除了对 INSERT、UPDATE、DELETE 基本操作进行定义外，还定义了 LOAD DATA 和 REPLACE 语句，这两种语句也能引起上述6中类型的触发器的触发。

LOAD DATA 语句用于将一个文件装入到一个数据表中，相当与一系列的 INSERT 操作。

REPLACE 语句一般来说和 INSERT 语句很像，只是在表中有 primary key 或 unique 索引时，如果插入的数据和原来 primary key 或 unique 索引一致时，会先删除原来的数据，然后增加一条新数据，也就是说，一条 REPLACE 语句有时候等价于一条。

INSERT 语句，有时候等价于一条 DELETE 语句加上一条 INSERT 语句。

INSERT 型触发器：插入某一行时激活触发器，可能通过 INSERT、LOAD DATA、REPLACE 语句触发；  
UPDATE 型触发器：更改某一行时激活触发器，可能通过 UPDATE 语句触发；  
DELETE 型触发器：删除某一行时激活触发器，可能通过 DELETE、REPLACE 语句触发。

**BEGIN … END 详解**  
在MySQL中，BEGIN … END 语句的语法为：

BEGIN  
[statement\_list]  
END  
其中，statement\_list 代表一个或多个语句的列表，列表内的每条语句都必须用分号（;）来结尾。  
而在MySQL中，分号是语句结束的标识符，遇到分号表示该段语句已经结束，MySQL可以开始执行了。因此，解释器遇到statement\_list 中的分号后就开始执行，然后会报出错误，因为没有找到和 BEGIN 匹配的 END。

这时就会用到 DELIMITER 命令（DELIMITER 是定界符，分隔符的意思），它是一条命令，不需要语句结束标识，语法为：  
DELIMITER new\_delemiter  
new\_delemiter 可以设为1个或多个长度的符号，默认的是分号（;），我们可以把它修改为其他符号，如$：  
DELIMITER $  
在这之后的语句，以分号结束，解释器不会有什么反应，只有遇到了$，才认为是语句结束。注意，使用完之后，我们还应该记得把它给修改回来。

一个完整的创建触发器示例  
假设系统中有两个表：  
班级表 class(班级号 classID, 班内学生数 stuCount)  
学生表 student(学号 stuID, 所属班级号 classID)  
要创建触发器来使班级表中的班内学生数随着学生的添加自动更新，代码如下：

复制代码 代码如下:

DELIMITER $  
create trigger tri\_stuInsert after insert  
on student for each row  
begin  
declare c int;  
set c = (select stuCount from class where classID=new.classID);  
update class set stuCount = c + 1 where classID = new.classID;  
end$  
DELIMITER ;

**变量详解**  
MySQL 中使用 DECLARE 来定义一局部变量，该变量只能在 BEGIN … END 复合语句中使用，并且应该定义在复合语句的开头，

即其它语句之前，语法如下：

DECLARE var\_name[,...] type [DEFAULT value]  
其中：  
var\_name 为变量名称，同 SQL 语句一样，变量名不区分大小写；type 为 MySQL 支持的任何数据类型；可以同时定义多个同类型的变量，用逗号隔开；变量初始值为 NULL，如果需要，可以使用 DEFAULT 子句提供默认值，值可以被指定为一个表达式。

对变量赋值采用 SET 语句，语法为：

SET var\_name = expr [,var\_name = expr] ...

**NEW 与 OLD 详解**

上述示例中使用了NEW关键字，和 MS SQL Server 中的 INSERTED 和 DELETED 类似，MySQL 中定义了 NEW 和 OLD，用来表示

触发器的所在表中，触发了触发器的那一行数据。  
具体地：  
在 INSERT 型触发器中，NEW 用来表示将要（BEFORE）或已经（AFTER）插入的新数据；  
在 UPDATE 型触发器中，OLD 用来表示将要或已经被修改的原数据，NEW 用来表示将要或已经修改为的新数据；  
在 DELETE 型触发器中，OLD 用来表示将要或已经被删除的原数据；  
使用方法： NEW.columnName （columnName 为相应数据表某一列名）  
另外，OLD 是只读的，而 NEW 则可以在触发器中使用 SET 赋值，这样不会再次触发触发器，造成循环调用（如每插入一个学生前，都在其学号前加“2013”）。

**查看触发器**

和查看数据库（show databases;）查看表格（show tables;）一样，查看触发器的语法如下：

SHOW TRIGGERS [FROM schema\_name];  
其中，schema\_name 即 Schema 的名称，在 MySQL 中 Schema 和 Database 是一样的，也就是说，可以指定数据库名，这样就

不必先“USE database\_name;”了。

**删除触发器**

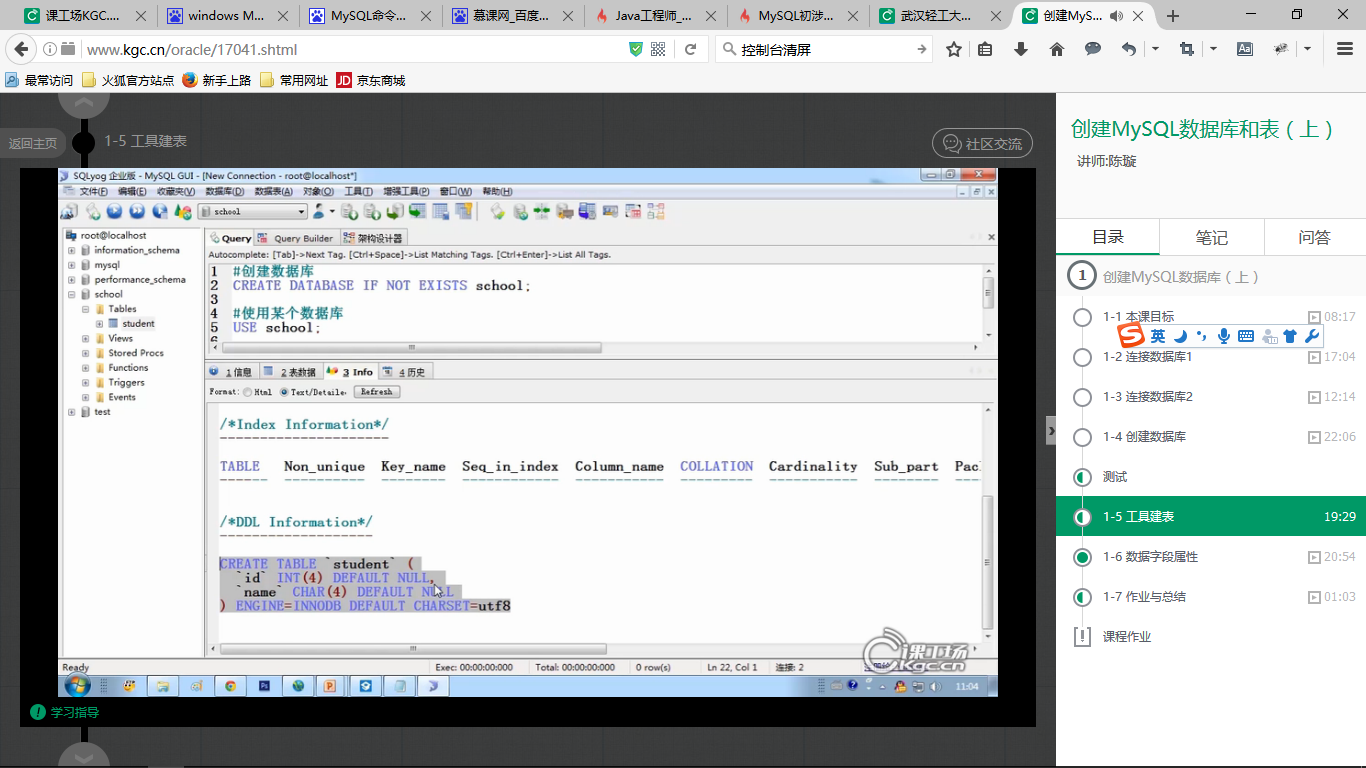
和删除数据库、删除表格一样，删除触发器的语法如下：

DROP TRIGGER [IF EXISTS] [schema\_name.]trigger\_name

**触发器的执行顺序**

我们建立的数据库一般都是 InnoDB 数据库，其上建立的表是事务性表，也就是事务安全的。这时，若SQL语句或触发器执行失败，MySQL 会回滚事务，有：

①如果 BEFORE 触发器执行失败，SQL 无法正确执行。  
②SQL 执行失败时，AFTER 型触发器不会触发。  
③AFTER 类型的触发器执行失败，SQL 会回滚。



## QL CHECK 约束

CHECK 约束用于限制列中的值的范围。

如果对单个列定义 CHECK 约束，那么该列只允许特定的值。

如果对一个表定义 CHECK 约束，那么此约束会在特定的列中对值进行限制。

## SQL CHECK Constraint on CREATE TABLE

下面的 SQL 在 "Persons" 表创建时为 "Id\_P" 列创建 CHECK 约束。CHECK 约束规定 "Id\_P" 列必须只包含大于 0 的整数。

### My SQL:

CREATE TABLE Persons

(

Id\_P int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255),

CHECK (Id\_P>0)

)

### SQL Server / Oracle / MS Access:

CREATE TABLE Persons

(

Id\_P int NOT NULL CHECK (Id\_P>0),

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255)

)

如果需要命名 CHECK 约束，以及为多个列定义 CHECK 约束，请使用下面的 SQL 语法：

### MySQL / SQL Server / Oracle / MS Access:

CREATE TABLE Persons

(

Id\_P int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255),

CONSTRAINT chk\_Person CHECK (Id\_P>0 AND City='Sandnes')

)

Dcsn

**项目数据库中的表：**

private String username;

private String password;

private String sex;

private String birthday;

private String number;

private String classs;

private String password1;

private String remarks;

private String system;

private String course;

学生：

Create table stu(

id int unsigned primary key auto\_increment,

username varchar(28) not null,

password varchar(8) not null,

sex varchar(12) not null default “man”,

birthday varchar(12) not null,

number varchar(10) not null ,

classs varchar(4) not null,

password1 varchar(8) not null,

remarks varchar(2000) not null,

system varchar(30) not null,

course varchar(28) not null

);

private String username;

private String password;

private String sex;

private String birthday;

private String number;

private String background;

private String password1;

private String remarks;

private String education;

private String course;

private String system;

老师:

Create table tea(

id int unsigned primary key auto\_increment,

username varchar(28) not null,

password varchar(8) not null,

sex varchar(12) not null default “man”,

birthday varchar(12) not null,

number varchar(10) not null ,

background varchar(4) not null,

password1 varchar(8) not null,

remarks varchar(2000) not null,

education varchar(6) not null,

system varchar(30) not null,

course varchar(28) not null

);

学号表：

Create table num(

id int unsigned primary key auto\_increment,

username varchar(28) not null,

password varchar(8) not null,

number varchar(10) not null ,

password1 varchar(8) not null,

system varchar(30) not null,

course varchar(28) not null

);

老师登记分数表：

**项目：project：**

学生1：

Create table stu(

id int unsigned primary key auto\_increment,

username varchar(28) not null,

sex varchar(8) not null default “man”,

birthday varchar(12) not null,

number varchar(10) not null ,

remarks varchar(2000) not null,

system varchar(30) not null,

sum varchar(4) not null

);

课程信息表：

Create table course(

Course\_number VARCHAR(3) primary key ,

Course\_name VArCHAR(16) not null,

Course\_time VARCHAR(8),

Course\_hour VARCHAR(3) ,

Course\_fen VARCHAR(7)

);

课程成绩表：

Create table chengji(

number varchar(12) primary key ,

Course\_number VArCHAR(3) not null,

chengji VARCHAR(3),

Course\_fen VARCHAR(7)

);

系统登录表：

Create table deng(

number varchar(10) primary key ,

password VArCHAR(8) not null

);

Create trigger [Check chengji] ON stu

For delete

AS

Delete from chengji

Where stu in(select stu from delete);

触发器：

delimiter ||   
DROP TRIGGER IF EXISTS afterinsert\_on\_stu||  
CREATE TRIGGER afterinsert\_on\_stu  
AFTER insert ON stu  
FOR EACH ROW  
BEGIN  
insert into guan(username,number) values(new.username,new.number);

END||

delimiter；

AFTER INSERT ON user1 FOR EACH ROW BEGIN insert into user2(ue\_name,ue\_phone,ue\_password) values(new.name,new.phone,new.password); END;EOF;

CREATE TRIGGER t\_afterdelete\_on\_user1 AFTER DELETE ON user1 FOR EACH ROW BEGIN delete from user2 where id=old.id; END;EOF; ; }}?></span>

触发器2

**delimiter ||   
DROP TRIGGER IF EXISTS afterdelete\_on\_stu ||  
CREATE TRIGGER afterdelete\_on\_stu  
AFTER delete ON stu  
FOR EACH ROW  
BEGIN  
delete from guan where number=old.number;**

**END||**

**delimiter；**

触发器3：

delimiter ||   
DROP TRIGGER IF EXISTS afterdelete\_on\_stu1||  
CREATE TRIGGER afterdelete\_on\_stu1  
AFTER delete ON stu  
FOR EACH ROW  
BEGIN  
delete from chengji where number=old.number;

END||

delimiter；

管理员表

Create table admin(

Username varchar(3) primary key,

Password varchar(3) unique key

);

师生登录表

Create table num1(

Username varchar(10) primary key,

Password varchar(8) unique key

);

学生登录管理：

Create table guan(

username varchar(10) ,

number varchar(10) not null unique key,

Password varchar(8)

);

老师表：

Create table tea(

id int unsigned primary key auto\_increment,

username varchar(28) not null,

sex varchar(8) not null default “man”,

birthday varchar(12) not null,

number varchar(10) not null ,

remarks varchar(2000) not null,

system varchar(30) not null,

password varchar(8) not null,

xueli varchar(6) not null

);