创建数据库：

create database xxx(数据库名);

查看数据库名字：

show database;

删除数据库：

drop database xxx(数据库名);

使用某个数据库：

use xxx(数据库名字);

查看表结构的方法：

show xxx(表名字);--必须在某个数据库中才能使用此命令

创建一个表的及约束条件

**create** **table** **example**( *#创建表example*

student\_id **int** **primary** **key**, *#创建student\_id字段，并设为主键*

student\_name **varchar**(20), *#创建student\_name字段，并设置数据格式为20长度的varchar类型*

student\_sex **int**(5) *#创建-----------*

);

查看一个表字段的属性：

describe xxx(表名字)；

创建外键约束：

**create** **table** example\_test(

student\_id **int** **primary** **key**,

student\_name **varchar**(20),

student\_sex **int**(5),

**constraint** c\_fk **foreign** **key** (student\_id,student\_name,student\_sex)

**references** **example**(student\_id,student\_name,student\_sex)

);

索引：

普通索引：

创建普通索引：

ALTER TABLE example0 ADD INDEX index12\_name(name(20))；

唯一性索引：使用unique参数可以设置索引为唯一索引。例如id

创建唯一性索引：ALTER TABLE index14 ADD UNIQUE INDEX index14\_id(course\_id);

全文索引：使用fulltext参数可以设置全文索引，全文索引只能建立在char，varchar或text类型字段上，查询数据量较大的字符串类型的字时。

创建全文索引：ALTER TABLE index15 ADD INDEX index15\_info(info);

单列索引：在表的单个列上建立索引。单列索引可以是普通索引，也可以是唯一性索引，还可以是全文索引，只要保证该索引只对应一个字段就可以了。

创建单列索引：ALTER TABLE index 16 ADD INDEX index16\_addr(address(4));

多列索引：在表的多个字段上创建索引。该索引指向创建时对应的多个字段，可以通过这几个字段进行查询。但是，只有查询条件中使用了这些字段中第一个字段时，索引才会被使用，例如：在表中的id，name，和sex字段上建立一个多列索引，那么只有查询条件使用了id字段时，该索引才会被使用。

创建多列索引：ALTER TABLE index17 ADD INDEX index17\_na(name, address);

空间索引:使用spatial参数可以设置索引为空间索引。空间索引只能建立在空间数据类型上，这样可以提高系统获取空间数据的效率。空间数据类型包括GEOMETRY.POINT.LINESTRING.POLYGON等。目前只有mysql存储引擎支持空间索引。这类检索很少会用到

创建空间索引：ALTER TABLE index18 ADD INDEX index18\_line(line);

在已经存在的表上，可以直接为表上的一个或几个字段创建索引。

create unique|fulltext|spatial|index 索引名 on 表名（属性名[长度][asc|desc]）;

例如：creat index index7\_id on example(id);

视图：

作用：

1.使操作简单化；

视图看到的信息就是所需要了解的信息。视图可以简化对数据的操作。例如：可以为经常使用的查询定义一个视图，使用户不必为同样的操作指定条件，方便用户操作。

2.增加数据的安全性；

通过视图，用户只能查询和修改指定的数据。

3.提高表的逻辑独立性

视图可以屏蔽原有表结构变化带来的影响。

语法：

create [algorithm={undefined|merge|temptable}] view 视图名[(属性清单)]

as select语句

[with[cascaded|local]check option];

algorithm有三个选项undefined|merge|temptable。undefined表示mysql将自动选择所需要的算法，merge表示将使用视图的语句与视图定义合并起来，使得视图定义的某一部分取代语句的对应部分；temptable选项表示将视图的结果存入临时表，然后使用临时表执行语句。

cascaded是可选参数，表示更新视图时要满足所有相关视图和表的条件，该参数为默认值，local表示更新视图时要满足该视图本身的定义条件即可。

创建单表视图

例1：create view view1

as select \* from department；

例2：create view view2（name,function,location）

as select d\_name,function,address from department；

查看视图结构：DESC view1;

在多个表上创建视图：

create algorithm=merge view

worker\_view1(name,department,sex,age,address)

as select name,department.d\_name,sex,2017-birthday(年数),address

from worker,department where worker.id=department.d\_id

with local check option;

查看视图的基本信息：

describe xxx(视图名)

show table status like "视图名";(like后面匹配的是字符串)

show create view xxx(视图名)

修改视图：

create or replace [algorithm={undefined|merge|temptable}]

view xxx(视图名)[(属性清单)]

as select 语句

[with [cascaded|local]check option]

alert [algorithm={undefined|merge|temptable}]

view xxx(视图名)

as select 语句

[with [cascaded|local]check option]

例1：

create or replace algorithm= temptable

view department\_view1(department,function,location)

as select d\_name,function,address from department;

例2：

alert view department\_view1(department,sex,location)

as select d\_name,sex,address from department

from department,worker where department.id=worker.id;

不能更新视图的情况：

1. 视图中包含sum().count().max().min()等函数；
2. 视图中包含union.unionall.distinct.groupby.having 等关键字
3. 常量视图：

create view view1(name,sex,address) as select name,sex,address

from worker groupby id;

1. 视图中的select包含子查询。
2. 由不可更新的视图导出的视图。
3. 创建视图时，algorithm为temptable类型
4. 视图对应的表上有没有默认值的列，而且该列没有包含在视图里

删除视图：

drop view [if exist] xxx(视图名或列表)

触发器：由事件来触发某个操作。这些事件包括insert、update、delete语句。当数据库系统执行这些事件时，就会激活触发器执行相应的操作。

基本形式：

create trigger (触发器名) before|after 触发事件

on 表名 for each row 执行语句

“触发器名”指要创建触发器的名字

“before|after”指定触发器执行的时间

“触发事件”指参数触发的条件：insert，update，delete

“表名”指触发事件操作的表名

“for each row”指任何一条记录上的操作满足触发事件都会触发该事件

“执行语句”指触发器被触发后执行的程序。

例如：

create trigger dept\_trigger before insert

on department for each row

insert into trigger\_time value(now())

创建有多个执行语句的触发器。基本形式：

create trigger (触发器名) before|after 触发事件

on 表名 for each row

begin

执行语句列表

end

查看触发器：show triggers

删除触发器：drop trigger xxx（触发器名）

正则表达式查询匹配

|  |  |
| --- | --- |
| 字符 | 含义 |
| ^ | 匹配字符串开始的部分 |
| $ | 匹配字符串结束的部分 |
| . | 代表字符串中任意一个字符，包括回车和换行 |
| [字符集合] | 匹配“字符集合”中的任何一个字符 |
| [^字符集合] | 匹配除了“字符集合”以外的任何一个字符 |
| S1|S2|S3 | 匹配S1|S2|S3中的任何一个字符 |
| \* | 代表多个该符号之前的字符，包括0和1个 |
| + | 代表多个该符号之前的字符，包括1个 |
| 字符串{n} | 字符串出现n次 |
| 字符串{m,n} | 字符串出现至少m次，至多n次 |

运算符

|  |  |  |
| --- | --- | --- |
| + | x1+x2+x3+x4 | 加法运算 |
| - | x1-x2-x3-x4 | 减法运算 |
| \* | x1\*x2\*x3\*x4 | 乘法运算 |
| / | x1/x2/ | 除法运算,返回x1/x2的商 |
| div | x1 div x2 | 除法运算,返回x1/x2的商，效果同”/” |
| % | x1%x2 | 求余运算，返回余数 |
| mod | mod(x1,x2) | 求余运算，返回余数,同% |

比较运算符：

|  |  |  |
| --- | --- | --- |
| = | x1 = x2 | 判断x1是否等于x2 |
| <>或!= | x1 <> x2或x1 != x2 | 判断x1是否不等于x2 |
| <=> | x1 <=> x2 | 判断x1是否等于x2 |
| > | x1 > x2 | 判断x1大于x2 |
| >= | x1 >= x2 | 判断x1大于等于x2 |
| < | x1 < x2 | 判断x1小于x2 |
| <= | x1 <= x2 | 判断x1小于等于x2 |
| is null | x1 is null | 判断x1是否为null |
| is not null | x1 is not null | 判断x1是否不为null |
| between and | x1 between m and n | 判断x1的值是否在m和n之间 |
| in | x1 in (值1，值2……) | 判断x1的值是否在值1到……之间 |
| like | x1 like 表达式 | 判断x1是否匹配表达式 |
| regexp | x1 regexp 正则表达式 | 判断x1是否匹配正则表达式 |

逻辑运算符：

|  |  |
| --- | --- |
| && 或 and | 与 |
| || 或 or | 或 |
| ! 或 not | 非 |
| xor | 亦或 |

位运算符：

|  |  |
| --- | --- |
| & | 按位与 |
| | | 按位或 |
| ~ | 按位取反 |
| ^ | 按位亦或 |
| << | 按位左移 |
| >> | 按位右移 |

1. 按位与

进行该运算时，数据库系统会先将十进制的数转换为二进制的数，然后对应操作数的每个二进制位上进行与运算。1和1相与得1，与0相与得0运算完成后再将二进制数变回十进制数

1. 按位或

将操作数化为二进制数后，每位都进行或运算。1和任何数进行或运算的结果都是1,0与0或运算结果为0

1. 按位取反

将操作数化为二进制数后，每位都进行取反运算。1取反后变成0，0取反后变成1

1. 按位亦或

将操作数化为二进制后，每位都进行或运算。相同的数异或之后结果是0，不同数异或后结果是1

1. 按位左移

“m<<n”表示m的二进制数向左移n位，右边补上n个0

1. 按位右移

运算符优先级：

|  |  |
| --- | --- |
| 1 | ! |
| 2 | ~ |
| 3 | ^ |
| 4 | \*,/,div,%,mod |
| 5 | +,- |
| 6 | <<,>> |
| 7 | & |
| 8 | | |
| 9 | =,<=>,<,>=,!=,<>,in,is null,like,regexp |
| 10 | between and,case,when,then,else |
| 11 | not |
| 12 | &&,and |
| 13 | ||,or,xor |
| 14 | != |

sql函数：

|  |  |
| --- | --- |
| ABS(X) | 返回X的绝对值 |
| CEIL(X),CEILING(X) | 返回大于或等于x的最小整数 |
| FLOOR(X) | 返回小于或等于x的最大整数 |
| RAND() | 返回0~1的随机数 |
| RAND(X) | 返回0~1的随机数，x值相同时返回随机数相同 |
| SIGN(X) | 返回x的符号，x是负数，0，正数分别返回-1,0,1 |
| PI() | 返回圆周率 |
| TRUNCATE(X,Y) | 返回数值x保留到小数点后y位的值 |
| ROUND(X) | 返回离x最近的整数 |
| ROUND(X,Y) | 保留x小数点后y位的值，但截断时要进行四舍五入 |
| POW(X,Y),POWER(X,Y) | 返回x的y次方 |
| SQRT(X) | 返回x的平方根 |
| EXP(X) | 返回e的x次方 |
| MOD(X,Y) | 返回x除以y以后的余数 |
| LOG(X) | 返回自然对数（以e为底的对数） |
| LOG10(X) | 将角度转换为弧度 |
| RADIANS(X) | 将弧度转换为角度 |
| SIN(X) | 求正弦值 |
| ASIN(X) | 求反正弦值 |
| COS(X) | 求余弦值 |
| ACOS(X) | 求反余弦值 |
| TAN(X) | 求正切值 |
| ATAN(X),ATAN2(X,Y) | 求反正切值 |
| COT(X) | 求余切值 |

存储过程和函数

存储过程基本形式：

create procedure sp\_name(存储过程名字) ([proc\_parameter[,...]](参数列表))

[characteristic ...(代码内容，以begin和end开始和结束)]routine\_body

[in(表示输入参数)|out(表示输出参数)|inout(表示可以输入也可输出)]param\_name(指定存储过程的参数名称) type(指定存储过程的参数类型)

例：创建一个名为num\_from\_employee。代码如下

DELIMITER &&(修改结束符号为&&)

CREATE PROCEDURE num\_from\_employee(存储过程名字)(IN(输入) emp\_id(参数名称) INT(数据类型),OUT(输出) count\_num(参数名称) INT(数据类型))

READS SQL DATA(读取数据)

BEGIN

SELECT COUNT(\*) INTO count\_num

FROM employee

WHERE d\_id = emp\_id;

END(存储过程的主体部分)

&&

DELIMITER;(结束符修改回;)

创建存储函数基本形式：

create function sp\_name(存储函数名字) ([proc\_parameter[,...]](参数列表))

returns type(返回值的类型)

[characteristic ...(代码内容，以begin和end开始和结束)]routine\_body

例：创建名为name\_from\_employee的存储函数。

DELIMITER &&

CREATE FUNCTION name\_form\_employee (emp\_id int)

returns varchar(20)

begin

return (

select name

from employee

where num = emp\_id;)

END

&&

DELIMITER;

变量：declare赋值变量，作用的范围是begin…end程序段

方法1：

declare my\_sql int defualt 10;//定义变量my\_sql，数据类型为int型，默认值为10

set my\_sql = 30;//为变量my\_sql赋值30

方法2：

select d\_id into my\_sql

from employee where id=2;

--从employee表中查询id为2的记录，并将该记录的d\_id赋值给变量my\_sql

定义条件：

declare condition\_name condition for condition\_value

condition\_value:

SQLSTATE[VALUE]sqlstate\_value|mysql\_error\_code

定义“ERROR 1146(42S02)”这个错误，名称为can\_not\_find。可以用两种不同的方法来定义：

//方法一：使用sqlstate\_value

DECLARE can\_not\_find CONDITION FOR SQLSTATE ‘42S02’(这个值是sqlstate\_value);

//方法二：使用mysql\_error\_code

DECLARE can\_not\_find CONDITION FOR 1146(这个值是mysql\_error\_code);

定义处理程序：

DECLARE handler\_type(错误的处理方式) HANDLER FOR condition\_value(错误类型)[,...] sp\_statement(存储过程或函数的执行语句)

handler\_type:

CONTINUE(遇到错误不进行处理，继续向下执行)|EXIT(遇到错误，马上退出)|UNDO(遇到错误撤回之前的操作)(handler\_type的三种取值方式)

condition\_value:

SQLSTATE[VALUE] sqlstate\_value|condition\_name(定义的条件名) |SQLWARNING(以01开头的错误) |NOT FOUND(以02开头的sqlstate\_value的值) |SQLEXCEPTION(没有被其他错误定义的错误类型)|mysql\_error\_code(错误类型)

下面是定义处理程序的几种方式，代码如下：

//方法一：捕获sqlstate\_value

DECLARE CONTINUE HANDLER FOR SQLSTATE ‘42S02’SET @info=’CAN NOT FOUND’;

//方法二：捕获mysq\_error\_code

DECLARE CONTINUE HANDLER FOR 1146 SET @info=’CAN NOT FOUND’;

//方法三：先定义条件，然后调用

DECLARE can\_not\_find CONDITION FOR 1146;//先为错误定义名称

DECLARE CONTINUE HANDLER FOR can\_not\_find SET @info=’CAN NOT FIND’;

//方法四：使用SQLWARNING

DECLARE EXIT HANDLER FOR SQLWARNING SET @info=’ERROR’;

//方法五：使用NOT FOUND

DECLARE EXIT HANDLER FOR NOT FOUND SET @info=’CAN NOT FIND’;

//方法六：使用SQLEXCEPTION

DECLARE EXIT HANDLER FOR SQLEXCEPTION SET @info=’ERROR’;

游标：查询语句可能查询出多条记录，在存储过程和存储函数中，使用游标来逐条读取查询结果集中的记录。游标的使用包括声明游标，打开游标，使用游标和关闭游标。游标必须在处理程序之前声明，并且声明在变量和条件之后。

声明游标：声明一个名为cur\_employee的游标

DECLARE cur\_employee CURSOR FOR SELECT name,age FROM employee;

打开游标：打开一个名为cur\_employee的游标

OPEN cur\_employe;

使用游标：使用一个名为cur\_employe的游标，将查询出来的数据存入emp\_name和emp\_age这两个变量中

FETCH cur\_employe INOT emp\_name，emp\_age;

关闭游标：

CLOSE cur\_employe;

流程控制：存储过程和存储函数中，可以只用流程控制来控制语句的执行。

IF语句：

例如：

IF age>20 THEN SET @count1 = @count1+1;

ELSE age = 20 THEN @count2=@count2+1;

ELSE @count3=@count3+1;

END IF;

IF search\_conditon THEN statement\_list

[ELSEIF search\_condition THEN statement\_list]...

[ELSE statement\_list]

END IF

CASE 语句

CASE case\_value

WHEN when\_value THEN statement\_list

[WHEN when\_value THEN statement\_list]…

[ELSE statement\_list]

END CASE

CASE

WHEN search\_condition THEN statement\_list

[WHEN search\_condition THEN statement\_list]…

[ELSE statement\_list]

END CASE

CASE age

WHEN 20 THEN SET @count1=@count1+1;

ELSE SET @count2=@count2+1;

END CASE;

CASE

WHERE age=20 THEN SET @count1=@count1+1;

ELSE SET @count2=@count2+1;

END CASE;

3．LOOP 语句

[begin\_label:]LOOP

statement\_list

END LOOP[end\_label]

add\_num:LOOP

SET @count=@count+1;

END LOOP add\_num;

4．LEAVE 语句

跳出循环控制

LEAVE label

add\_num:LOOP

SET @count=@count+1;

LEAVE add\_num;

END LOOP add\_num;

5．ITERATE 语句：跳出本次循环，执行下一次循环

ITERATE label

add\_num:LOOP

SET @count=@count+1;

IF @count=100 THEN LEAVE add\_num;

ELSEIF MOD(@count,3)=0 THEN ITERATE add\_num;

SELECT \* FROM employee;

END LOOP add\_num;

REPEAT 语句：有条件循环，满足条件退出循环

[begin\_label:]REPEAT

statement\_list

UNTIL search\_condition

END REPEAT[end\_label]

REPEAT

SET @count=@count+1;

UNTIL @count=100;

END REPEAT;

7．WHILE 语句

[begin\_label:]WHILE search\_condition DO

statement\_list

END REPEAT[end\_label]

WHILE @count<100 DO

SET @count=@count+1;

END WHILE;

调用存储过程和函数

存储过程是通过call语句来调用的。而存储函数的使用方法与mysql内部函数的使用方法是一致的。执行存储过程和存储函数需要有execute权限。execute权限的信息存储在information\_schema数据库下面的user\_privileges表中

调用存储过程的基本形式：

call sp\_name([parameter[,...]]);

例如：定义一个存储过程，然后调用这个存储过程。代码如下

//创建存储过程

DELIMITER &&

CREATE PROCEDURE num\_from\_employee(IN emp\_id INT,OUT count\_num INT)

READS SQL DATA

BEGIN

SELECT COUNT(\*) INTO count\_num

FROM employee

WHERE d\_id = emp\_id;

END &&

DELIMITER ;

//调用存储过程

CALL num\_from\_employee(1002,@n)

//查询返回的结果

SELECT @n;

调用存储函数：存储函数的使用方法和mysql内部函数的使用方法是一样的。区别在与存储函数是用户自己定义的，而内部函数是mysql的开发者定义的。

例如：

//创建存储函数

DELIMITER &&

CREATE FUNCTION name\_from\_employee(emp\_id INT)

RETURNS VARCHAR(20)//设置存储函数返回的类型

BEGIN

RETURN (SELECT name FROM employee WHERE num=emp\_id);

END &&

DELIMITER ;

//调用存储函数

SELECT name\_from\_employee(3);

查看存储过程和存储函数的状态：SHOW STATUS

基本形式如下：

SHOW {PROCEDURE|FUNCTION}STATUS[LIKE ’pattern’];

其中，“PROCEDURE”表示查询存储过程，“FUNCTION”表示查询存储函数。LIKE ’pattern’参数用来匹配存储过程或函数的名称。

例如：

SHOW PROCEDURE STATUS LIKE ‘num\_from\_employee’ \G

查看存储过程和存储函数的定义：SHOW CREATE

基本形式：

SHOW CREATE {PROCEDURE|FUNCTION} sp\_name;

SHOW CREATE PROCEDURE num\_from\_employee \G

也可以通过查看information\_schema数据库下Routines表来查看存储过程和函数的信息。

修改存储过程和函数

基本形式如下：

ALTER {PROCEDURE|FUNCTION} sp\_name[characteristic ...]

characteristic:

{CONTAINS SQL|NO SQL|READS SQL DATA|MONIFIES SQL DATA}

|SQL SECURITY{DEFINER(创建者)|INVOKER(调用者)}//执行权限

|COMMENT ‘string’//注释

例如：有一个存储过程：

SELECT \* FROM information\_schema.Routines WHERE

ROUTINE\_NAME = ‘num\_from\_employee’ \G

下面修改存储过程num\_from\_employee的定义，将读写权限改为MODIFIES SQL DATA,并指明调用者可以执行：

ALTER PROCEDURE num\_from\_employee

MODIFIES SQL DATA

SQL SECURITY INVOKER;

SELECT SPECIFIC,SQL\_DATA\_ACCESS,SECURITY\_TYPE FROM information\_schema,Routines

WHERE ROUTINE\_NAME=‘num\_from\_employee’;

下面修改存储函数name\_from-employee的定义，将读写权限修改为READS SQL DATA,并加上注释信息‘FIND NAME’。代码如下：

ALTER FUNCTION name\_from\_employee

READS SQL DATA

COMMENT ‘FIND NAME’;

SELECT SPECIFIC\_NAME,SQL\_DATA\_ACCESS,ROUTINE\_COMMENT

FROM information\_schema,Routines WHERE ROUTINE\_NAME=‘name\_from\_employee’;

删除存储过程和函数：

DROP {PROCEDURE|FUNCTION} sp\_name;

实例：在food表上创建名为food\_price\_count的存储过程。存储过程有三个参数。输入参数为price\_info1和price\_info2，输出参数为count。存储过程的作用是查询food表中食品单价高于price\_info1且低于price\_info2的食品种数，然后由count参数来输出，并计算满足条件的单价的总和。

DELIMITER &&

CREATE PROCEDURE food\_price\_count(IN price\_info1 FLOAT,IN price\_info2 FLOAT,OUT count INT)

READS SQL DATA

BEGIN

DECLARE temp FLOAT;

DECLARE match\_price CURSOR FOR SELECT price FROM food;

DECLARE EXIT HANDLER FOR NOT FOUND CLOSE match\_price;

SET @sum = 0;

SELECT COUNT(\*) INTO count FROM food

WHERE price>price\_info1 AND price<price\_info2;

OPEN match\_price;

REPEAT

FETCH match\_price INTO temp;

IF temp>price\_info1 AND temp<price\_info2

THEN SET @sum = @sum+temp;

END IF;

UNTIL 0 END REPEAT;//直到光标等于0，结束循环

CLOSE match\_price;

END &&

DELIMITER ;

mysql性能优化

分析查询语句：explain/describe(desc)

不走索引的查询

Like 以 %开头的丌走

Or 两边的列有一个没有建立索引丌走索引

多列索引第一个字段没有使用丌走索引