

**……DDL……**

数据库查询

show databases;

show create database <dbname>;

数据库创建

create database <dbname>;

create database if not exists <dbname>;

create database <dbname> character set gbk;

修改数据库字符集

alter database <dbname> character set gbk;

删除数据库

drop database <dbname>;

drop database if exists <dbname>;

使用数据库

use <dbname>;

查看数据表

show tables;

desc <tableName>;

删除数据表

drop table <tableName>;

drop table if exists <tableName>;

创建数据表

create table students(name varchar(20), stu\_num char(4) not null unique);

create table if not exists students(name varchar(20), stu\_num char(4) not null unique);

修改表名

alter table <tableName> rename to <newTableName>;

修改数据表字符集

alter table <tableName> character set gbk;

添加字段

alter table <tableName> add <fieldName> varchar(8);

修改字段名和该字段的数据类型

alter table <tableName> change <FieldName> <NewFieldName> <type>;

修改字段类型

alter table <tableName> modify <FieldName> <type> [first/after id]; modify可以改type和null

删除字段

alter table <tableName> drop <FieldName>

MySQL数据类型有三大类:数值类型，字符类型，日期类型

数值类型：

tinyint 1byte

smallint 2byte

mediumint 3byte

int 4byte

bigint 8byte

float 4byte

double 8byte

decimal 示例：decimal(4,2) 总共有4位，小数占两位。

字符类型：

char 0~255字节 定长字符串，最多存储255个字符，如char(255)

varchar 0~65535字节 可变长度字符串

tinyblob 0~255字节 存储二进制字符串

blob 0~65535字节

mediumblob 0~1677215字节

longblob 0~4294967295

tinytext 0~255字节

text 0~65535字节 存储文本数据（字符串）

mediumtext 0~1677215字节

longtext 0~4294967295

日期类型：

date 存储年月日，比如：2022-09-13

time 存储时分秒，比如：11：12：13

year 存储年

datetime 存储年月日时分秒,比如：2022-09-13 11：12：13

timestamp 存储年月日时分秒，比如：20220913 111213

字段约束（是对字段的约束）

非空约束(not null)

唯一约束(unique)

主键约束(primary key)

外键约束(foreign key)

注：主键不一定必须是int类型，也可以是其他类型

创建表时将某个字段声明成主键

create table if not exists stu(id int primary key, age char(8) not null, qq char(11) unique);

create table if not exists stu(id int, age char(8) not null, qq char(11) unique, primary key(id));

取消某个字段的主键约束

alter table stu drop primary key;

表创建出来后将某个字段声明成主键

alter table stu modify id int primary key;

alter table stu add primary key(id);

定义主键自动增长（只有int类型能自动增长，如果最后一条记录主键为3，并且被删除了，下一次添加新的记录时不会从3开始，从4开始）

create table person(id int primary key auto\_increment);

联合主键（将数据表中的多列组合在一起作为主键) 当然我们不使用联合主键，单独创建一列作为主键也行

create table person(age int, id int, primary key(age, id));

将字段的unique属性修改成非unique属性

show create table <tableName>;查看约束名称

alter table <tablename> drop index <约束名称>;

alter table <tablename> add unique(id);

**……DML……**

向数据表中指定的列添加数据

insert into <tableName>(Field1, Field2, Field3) values(value1, value2, value3);值和字段的位置要一一对应，字段位置随意。字段可以不用写全，但not null的一定要写上

insert into <tableName> value(value1, value2......);当给全部字段添加数据的时候可以这样写，值和字段的位置要一一对应。

删除数据表中满足条件的记录

delete from <tableName> 删除表中的所有记录

delete from <tableName> where conditions 删除表中满足条件的记录

例如:delete from stus where stu\_num='20223435'; delete from stus where age>20;

修改数据表中的数据

update <tableName> set stu\_name='李四', stu\_tel=1985454512, stu\_age=19 where stu\_num='20221234';

update <tableName> set stu\_name='李四';没有where子句，则这列都会被改成李四

**......DQL......**

查询指定字段

select Field1, Field2,Field3,...... from <tableName> [where conditions];

select \* from <tableName> [where conditions];查询所有字段

where子句:（筛选出满足条件的记录）

where子句使用的关系运算符有：= > < >= <= between and（闭区间） != <>小于大于（等价于!=）

where子句还可以用逻辑运算符连接and or not not：select \* from stus where stu\_age not between 19 and 25;

like子句：（用来模糊查询)

select \* from stus where stu\_name like %o%;

select \* from stus where stu\_name like o%;

select \* from stus where stu\_name like %o;

select \* from stus where stu\_name like \_o%;

注：%表示任意多个字符，0个也行。\_表示一个字符。

对查询结果的处理

1、显示指定要查询的列

select Field1, Field2,Field3,...... from stus;

2、对某一列进行运算之后再显示出来

select stu\_name, 2022-stu\_age from stus;

3、给字段取显示时的别名

select stu\_name as 姓名, 2022-stu\_age as 出生年月 from stus;

4、消除重复行(distinct)

select distinct stu\_age from stus;

5、对查询出的结果按某个或某些字段排序

单字段排序

select \* from stus [where] order by Field1 desc; asc升序（默认） desc降序

多字段排序

select \* from stus [where] order by Field1 desc, Field2 asc;

6、聚合函数

select count(stu\_num) from stus [where]; 查询满足条件的记录中，指定列的总数

select max(stu\_age) from stus [where]; 查询满足条件的记录中，指定列的最大值

select min(stu\_age) from stus [where]; 查询满足条件的记录中，指定列的最小值

select sum(stu\_age) from stus [where]; 查询满足条件的记录中，指定列的总和

select avg(stu\_age) from stus [where]; 查询满足条件的记录中，指定列的平均值

7、日期函数

日期类型给值的时候要以字符串的形式给:'yyyy-MM-dd hh:mm:ss' '2021-09-03 14:03:07'

now()获取当前系统时间insert into stus(date) values(now());

sysdate()获取当前系统时间insert into stus(date) values(sysdate());

还有两个很少用：curtime() curdate()

8、字符串函数

concat()字符串拼接，参数可以是字符串，也可以是字段（字段类型不一定必须是字符类型）

select concat('hello', 'world',.....); select concat(stu\_name, stu\_date) from stus;

upper()字符串转大写，参数可以是字符串，也可以是字符串类型的字段

lower()字符串转大写，参数可以是字符串，也可以是字符串类型的字段

substring(Field, start,len) 从指定列中截取部分显示，start开始截取len字节，start从1开始

分组查询

select 分组字段/聚合函数 from 表名 [where] group by 分组列名 分组字段一般是分组列名，因为分组字段相同的数据肯定是会分为一组，这时候其他字段该显示哪个值呢？

分组查询：

select stu\_age, count(stu\_age) from stus [where] group by Field [having 条件] [order by Field]; 比如having stu\_num > 20

执行顺序：where先从数据库查询记录，然后group by对查询的记录进行分组，然后having对分组后的数据进行筛选，最后执行order by

没有group by就不能有having，因此having是不能单独使用的，必须配合group by使用

where是选出符合条件的记录，having是选出符合条件的组

分页查询：

select \* from stus limit 0, 3;

select \* from stus limit (pageNum-1)\*pageSize, pageSize; 分页查询通用公式

关联关系：

1、一对一关联：（有两种解决方案）

主键关联：将两张表的主键关联起来，两个主键相同的数据互为对应关系

外键约束：在一张表中另起一个字段作为外键与另一张表的主键相关联，因为是一对一关联，所以外键要声明成unique属性

2、一对多和多对一

一对多和多对一都是一样的，只是方向反过来而已。解决方案：两个都是在多的一方添加外键与一的一端的主键（不是主键也行，但一般都是用主键）进行关联

3、多对多关联

解决方案：额外创建一张关系表来维护多对多关联，在关系表中定义两个外键，分别与两个数据表的主键进行关联。

如何添加外键约束：

将一个字段添加外键约束与另外一个数据表的主键关联后，这个字段中添加的数据必须是另外那个表关联了的那个字段（一般是主键）中有的数据，并且外键与关联的那个字段的数据类型和长度必须一致

创建两张表的时候要先创建没有外键约束的那张表:create table classes(class\_id int primary key auto\_increment, class\_name varchar(10) not null);

在创建表时指定外键约束：create table stus(name varchar(10) not null, cid int, constraint FK\_STUS\_CLASSES foreign key(cid) references classes(class\_id));

在创建表后指定外键约束：alter table stus add constraint FK\_STUS\_CLASSES foreign key(cid) references classes(class\_id);

删除外键约束：alter table stus drop foreign key FK\_STUS\_CLASSES;

级联：

学生信息表中的cid关联着班级表的主键，因此不能修改或删除班级表中主键的值

那么要怎样修改或删除呢，下面有两种解决方案。

1、将cid中对应着主键中要修改的值的那些值改为null，然后修改主键中的值，然后再把cid中改为null的地方与主键中的值同步

2、用级联：设置级联修改和级联删除：

删除原来的外键alter table stus drop foreign key FK\_STUS\_CLASSES

重新添加外键alter table stus add constraint FK\_STUS\_CLASSES foreign key(cid) references classes(class\_id) ON UPDATE CASCADE ON DELETE CASCADE;

这样修改和删除班级表中主键的值，学生表也会跟着变动了

连接查询：（从多张表中查询数据）

笛卡尔积：假设有两个集合A和B，那么使集合A中的每条数据依次关联到B的所有数据

内连接：（只显示有关联的部分）

select ... from stus inner join classes；这样就会产生笛卡尔积，查询出来的很多数据就会没有意义。怎么解决呢，那就要添加条件，只留下有意义的数据，如下

select ... from stus inner join classes where stus.cid = classes.class\_id；用where的话就是先生成笛卡尔积再筛选，效率就低了

select ... from stus inner join classes on stus.cid = classes.class\_id [where]；用on是直接筛选，效率会更高

左连接：（将左表全部显示出来，右表和左表关联的部分就显示，没有关联的部分就显示null）

select ... from stus left join classes on stus.cid = classes.class\_id [where]；

右连接：（将右表全部显示出来，左边和右表关联的部分就显示，没有关联的部分就显示null）

select ... from stus right join classes on stus.cid = classes.class\_id [where]；

子查询：（即第二次的查询是基于第一次查询的结果进行的）

案例一：select \* from stus where cid = (select class\_id from classes where class\_name = 'java2105');

如果cid后面查出来的不是一个值而是多个怎么办，这时如下案例二，结果是多条的时候不能用=而是用in

案例二：select \* from stus where cid in (select class\_id from classes where class\_name like 'java%');

案列三：多条件查询，有两种方式

select \* from stus where cid=1 and gender = '男';

select \* from (select \* from stus where cid = 1) t where t.gender = '男 '; 可以将第一次查询出来的结果看作一张表再查询，不过这张表必须有名字，所以把它命名成t

上面的t.gender，指定字段的时候可以“表名.字段名”

SQL执行过程：navicat--->将SQL指令发送到数据库--->数据库接受SQL指令--->SQL引擎编译并执行SQL指令--->将执行结果返回--->显示查阅结果

存储过程：能过将完成特定功能的SQL指令集进行封装，编译后存储在数据库服务器上，并为之取一个名字，客户端可以通过名字直接调用这个SQL指令集，获取执行结果。

优点：只需要通过名字调用就行，所以减少了网络开销。因为传输的是名字所以也防止了拦截恶意篡改SQL指令。无需重复编译，提高了效率。存储过程支持流程控制语句，可以实现更复杂的业务逻辑。

缺点：存储过程根据不同的数据库产品编译，当数据迁移时需要重新编写SQL指令。存储过程受限于数据库产品，如果需要高性能的优化会成为一个问题。在互联网项目中，如果需要数据库的高（连接）并发访问，使用存储过程会增加数据库连接的时间（因为我们将复杂业务交给了数据库去处理）

创建存储过程：

create procedure test1(IN a int, IN b int, OUT c int)

begin

set c = a + b;

end;

调用存储过程：

set @m = 0; 定义变量@m

call test1(3,2,@m); 调用存储过程，将传出值赋给@m

select @m from dual; 查看变量@m的值，dual是一张系统表，定义的用户变量都存储在这里，所以要from dual

用户变量：相当于C语言的全局变量，存储在dual里

局部变量：存储过程内部使用的变量，就相当于C语言的局部变量

定义用户变量：set @m = 1; 变量名要@开头

定义局部变量：declare x int default 0; 意思是定义变量x，int类型，初值为0

无论是用户变量还是局部变量都是用set关键字修改值，如set @m = 20; set x = 10;

在存储过程中如何将查询的结果赋值给变量：

create procedure test(out a int)

begin

select count(stu\_num) into a from stus;//将查询到的学生数量赋值给a,用into关键字

end;

存储过程的参数：

输入参数in:不用解释了

输出参数out：不用解释了

输入输出参数inout:既可以当输入参数，也可以当输出参数。尽量少用，因为代码的可读性会变低。

存储过程中的分支语句和循环语句（在存储过程中使用，也就是在函数体内使用）：

if语句：

if a = 1 then

SQL语句

end if;

if-else语句：

if a = 1 then

SQL语句;

else

SQL语句;

end if;

case语句：

case a

when 1 then

SQL语句;

when 2 then

SQL语句;

else

SQL语句;

end case;

while循环：

while i<num do

insert into classes(Field1, Field2, Field3,...) values(CONCAT('java',i), ...);//CONCAT实现字符串的拼接，结果是'java1', 'java2', 'java3'.......

set i = i+1;

end while;

repeat循环：

repeat

SQL语句;

set i = i+1;

until i>num

end repeat;

loop循环：

myloop:loop

SQL语句;

set i = i+1;

if i>num then

leave myloop;

end if;

end loop;

存储过程的管理：

存储过程是隶属于某个数据库的，跟其他的数据库不共享。

查询存储过程：

show procedure status;查看所有数据库的所有存储过程

show procedure status where db='db\_test';查看某个数据库的所有存储过程

show create procedure db\_test.pro\_test;查看某个存储过程的创建细节

修改存储过程的特征：

alter procedure pro\_test 特征1 [特征2 特征3 特征4];

删除存储过程：

drop procedure pro\_test;

值得注意的是，删除数据库对象用drop, 删除数据表中的数据用delete

存储过程中的游标

游标可以用来一次取出查询出的结果集中的每条数据，相当于迭代器

第一步：声明游标，将select查询到的记录的集合存到游标中，游标最开始指向第一条记录

declare mycursor cursor for select book\_name, book\_author, book\_price from books;

第二步:打开游标

open mycursor;

第三步：提取游标当前指向的记录依次对应到下面三个变量里，提取之后，游标自动下移，这一步放在循环里执行

FETCH mycursor INTO name, author, price;

第四步：跳出循环的时候在循环外面关闭游标

close mycursor;

触发器

触发器是一种特殊的存储过程，它和存储过程一样是能够完成特定功能存储在MySql服务器上的Sql片段，但触发器无需手动调用，当执行DML操作时会自动调用

只有执行insert/delete/update操作才能触发触发器的执行

创建触发器：

create trigger tri\_test1

before|after 在操作之前还是之后？

insert|delete|update on <tableName> 是做哪个操作的时候触发？

for each row 每操作一条记录就触发,即使是一条SQL语句操作的两条记录也触发两次，如insert into stus（Filed1, Filed2, Filed3) values(1, "小明", "..."), (2, "小华", "...");

begin 只有一条SQL语句的时候可以不写begin和end

end;

查看触发器：show triggers;

删除触发器：drop trigger tri\_test1;

NEW：在触发器中用于获取insert操作添加的记录的某字段的值，update操作修改后的记录的某字段的值，比如new.stu\_name表示添加或修改的记录的stu\_name字段的值

OLD：在触发器中用于获取delete操作删除前的记录的某字段的值，update操作修改前的记录的某字段的值，比如OLD.stu\_name表示删除或修改前的记录的stu\_name字段的值

视图

视图是由数据库中一张表或多张表根据特定条件查询出的数据组成的一张虚拟表

创建视图：

create view view\_test

as

select \* from stus where stu\_gender='男';

当视图的查询来自一张表的时候，对视图数据的增删改也会影响到原表，即视图的增删改会同步原表。

当视图的查询来自多张表的时候，对视图的增删改是不允许的。

查询视图：desc view\_name;

修改视图：

create or replace view view\_test

as

select \* from stus where stu\_gender='女';

alter view view\_test

as

select \* from stus where stu\_gender='男';

删除视图：drop view view\_test;

索引

索引是将数据表中某一列或某几列的数据全部提取出来构造成一个B+树，这样查询效率就大大提高了，避免了全表扫描。

索引的分类：

主键索引：在数据表的主键字段创建的索引

唯一索引：在数据表的唯一字段创建的索引

普通索引：在数据表的普通字段创建的索引

组合索引：两个及以上的字段联合起来创建的索引

在创建数据表时，将字段声明为主键，会自动在主键字段创建主键索引。

在创建数据表时，将字段声明为唯一，会自动在唯一字段创建唯一索引。

查看索引

show indexes from <tableName>

创建唯一索引：（唯一索引的列的值不能重复，要唯一）

create unique index index\_test1 on stus(tid);

创建普通索引：（不要求列的值的唯一性）

create index index\_test2 on stus(tid);

创建组合索引：

create index index\_test3 on stus(tid, name);

索引创建完后无需调用，当根据创建索引的列进行数据查询的时候，会自动使用索引，组合索引也是当用创建索引的所有字段进行查询时才会触发。

查看索引

show create table <tableName>\G;

show indexes from <tableName>\G;

show keys from <tableName>\G;

删除索引

drop index index\_test3 on stus;

事务

我们把完成特定业务的多条数据库的DML操作看成一个整体，称之为事务

事务的特性（ACID特性）

原子性：事务中的多个DML操作，要么同时执行成功，要么同时执行失败。

一致性：事务执行前和执行后数据都是一致的。比如有十本书，借了一本书的总数还是十

隔离性：多个并行执行的事务互不影响。

持久性：事务完成之后，对数据的操作是永久的。

开启事务（关闭自动提交，变成手动提交）

start transaction;

事务回滚（如果执行到一半出现了问题，那就删除连接缓存中的操作）

rollback;

提交事务（将连接缓存中的操作写入数据文件）

commit;

事务隔离级别

读未提交：（read uncommitted)T2读取T1执行但未提交的数据，可能会导致出现脏读。

读已提交：(read committed)T2只能读取T2已经提交的数据，避免了脏读，但可能会导致不可重复读（虚读）。

可重复读：(repeatable read)T2执行第一次查询之后，在事务结束之前其他事务不能修改对应的数据，避免了不可重复读，但可能会导致幻读

串行化：(serializable)同时只允许一个事务对数据表进行操作，避免了所有问题。

mysql数据库默认的隔离级别为可重复读

查看隔离级别：select @@transaction\_isolation;

设置mysql默认隔离级别：set session transaction isolation level read committed;