操作数据库

-- 创建数据库

CREATE DATABASE MaternalShop;

-- 使用数据库

USE MaternalShop;

操作表

/\*删除表\*/

DROP TABLE MaternalShop;

/\*清空表 不可 回滚\*/

truncate table t\_cart;

/\*清空表 可以 回滚\*/

delete from t\_student;

/\*查看表结构\*/

desc t\_student;

describe t\_student;

新建表

DROP TABLE t\_cart;

/\*购物车 表\*/

CREATE TABLE t\_cart(

/\*购物车 id\*/

cid VARCHAR(18) NOT NULL,

/\*谁的 购物车\*/

uid VARCHAR(18) NOT NULL,

/\*购物车 开始生成的 时间\*/

startDate DATETIME,

/\*购物车 结束的时间，用户进行付款完成、清空该购物车、购物车过期失效\*/

endDate DATETIME,

price DECIMAL(10,5),

/\*购物车的状态：初次生成、再次编辑、完成结算、被清空、购物车过期失效\*/

status VARCHAR(10) default '0',

/\*关联外键\*/

FOREIGN KEY(uid) REFERENCES t\_user(uid),

/\*设置 主键\*/

PRIMARY KEY(cid)

);

/\*创建表 并使用原表中的数据\*/

create table t\_off\_student

as select t\_student.sid sid, t\_student.cid cid, t\_student.credit credit

from t\_student;

/\*创建表 不使用原表中的数据\*/

create table t\_off\_student

as select t\_student.sid sid, t\_student.cid cid, t\_student.credit credit

from t\_student

where 1=2;

唯一性约束

create table t\_order(

/\*订单金额 不能 小于0\*/

amount decimal(12, 3) check(amount >=0),

/\*使用奖券 表中 要唯一\*/

hotCheck varchar(20) not null unique

);

插入数据

insert into links(name,address) values('jerichen','gdsz');

insert into t\_student

set

sid = null,

sname = '张飞',

cid = (select t\_class.cid from t\_class where t\_class.cname = '一班'),

phone = '13146008028',

pwd = '123456',

age = 24,

credit = 100

;

时间戳

-- 更多api见 http://dev.mysql.com/doc/refman/5.5/en/date-and-time-functions.html#function\_current-timestamp

insert into t\_cart values('c\_6','u\_2',DATE\_FORMAT('2016-10-17 23:00:02','%Y-%m-%d %H:%i:%s'), DATE\_FORMAT(SYSDATE(),'%Y-%m-%d %H:%i:%s'),200, 20,'first created');

基本查询语句

查询所有字段

/\*查询所有字段\*/

select \* from t\_user;

查询某几个字段

/\*查询某几个字段\*/

select uname, pwd, accountbalance from t\_user;

查询10倍的余额

/\*查询 10 倍的余额\*/

select t\_user.accountbalance \* 10 from t\_user;

取别名

/\*取 别名， 每100积分 = 1元\*/

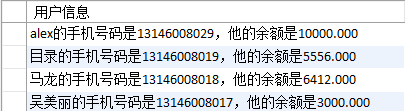
select t\_user.pointbalance/100 moey from t\_user;

select t\_user.pointbalance/100 as moey from t\_user;

select t\_user.pointbalance/100 as '积分兑换成的余额' from t\_user;

连接符 字符串拼接

select concat(t\_user.uname, '的手机号码是', t\_user.phone, '，他的余额是', t\_user.accountbalance) as '用户信息' from t\_user;



查询余额在1000以上的

select t\_user.uname from t\_user where t\_user.accountbalance > 1000;

区间查询

/\*查询 余额在[1000, 2000]的用户的 用户名\*/

select \* from t\_user where t\_user.accountbalance >=1000 and t\_user.accountbalance <= 2000;

/\*查询 名字是 A B C D 的用户\*/

select \* from t\_user where t\_user.uname in ('Alex','张三','李四','陆毅','秦始皇');

前10条数据

/\*从第一条开始， 向右找最多3条数据\*/

select \* from t\_student limit 1,3;

模糊查询

/\*查询 用户名 包含 '黄'的\*/

select \* from t\_user where t\_user.uname like '%黄%';

/\*查询 用户名 以 '丽'结尾的\*/

select \* from t\_user where t\_user.uname like '%丽';

/\*查询 用户名 以 '黄'开头的\*/

select \* from t\_user where t\_user.uname like '黄%';

/\*查询 用户名 第二个是 '三'的\*/

select \* from t\_user where t\_user.uname like '\_三%';

/\*查询 名字中 包含[ \_ ] 的\*/

select \* from t\_user where t\_user.uname like '%\\_%' escape '\\';

select \* from t\_user where t\_user.uname like '%#\_%' escape '#';

/\*查询名字中 包含 美|丽 的\*/

select \* from t\_user where t\_user.uname like '%美%丽%' or t\_user.uname like '%丽%美%';

空值查询

/\*查询 余额 为空值 的用户\*/

select \* from t\_user where t\_user.accountbalance is null;

/\*查询 余额 非空 的用户\*/

select \* from t\_user where t\_user.accountbalance is not null;

排序

/\*余额 逆序\*/

select \* from t\_user where t\_user.phone like '%2%' order by t\_user.accountbalance desc;

/\*余额 正序\*/

select \* from t\_user where t\_user.phone like '%2%' order by t\_user.accountbalance asc;

/\*余额 正序，遇到相同的， 再按 手机号码 逆序\*/

select \* from t\_user where t\_user.phone like '%2%' order by t\_user.accountbalance asc, t\_user.phone desc;

去重

/\*查询所有人的 用户信息，按照余额递增，余额不能有重复项\*/

select \* , group\_concat(distinct t\_user.accountbalance) as balance from t\_user group by t\_user.accountbalance asc;

/\*查询 学生表中 出现多少个 班级\*/

select count(distinct t\_student.cid) from t\_student;

条件匹配

select \*, case t\_user.uname when 'alex' then t\_user.accountbalance \*10

end from t\_user

where t\_user.uname in ('Alex','张三','李四');

select \*, case t\_user.uname

when 'alex' then t\_user.accountbalance \*10

when '张三' then t\_user.accountbalance \* 100

when '李四' then t\_user.accountbalance \* 1000

end from t\_user

where t\_user.uname in ('Alex','张三','李四');

连表操作

简单的连表查询

select t\_user.uid, t\_user.uname, t\_cart.amount, t\_cart.usepoint from t\_user, t\_cart where t\_user.uid = t\_cart.uid;

内连接

内连接是匹配两个表中 共有的行

select t\_student.sname, t\_class.cname from t\_student inner join t\_class on t\_student.cid = t\_class.cid;

左外连接

左外连接是匹配两个表中 共有的行 + 左表中特有的行

select t\_student.sname, t\_class.cname from t\_student left join t\_class on t\_student.cid = t\_class.cid;

右外连接

右外连接是匹配两个表中 共有的行 + 右表中特有的行

select t\_student.sname, t\_class.cname from t\_student right join t\_class on t\_student.cid = t\_class.cid;

连表查询

/\*查询学生表中，学生的 平均学分 大于100的 所在班级的id 和 该班级的名称 和 该班级的平均学分，按班级分组\*/

select t\_student.cid, t\_class.cname, avg(ifnull(t\_student.credit, 0))

from t\_student join t\_class on t\_student.cid = t\_class.cid

group by t\_student.cid

having max(ifnull(t\_student.credit, 0))>120;

单行函数

字符串截取

/\*(13146008029)从 第2个 开始，向右 找4个\*/

select substring(t\_user.phone, 2,4) from t\_user where t\_user.uname = 'Alex';

获取字符串的长度

/\*获取字符串的长度\*/

select length(t\_user.uname) from t\_user where t\_user.uname = 'Alex';

获取字符在字符串中的首次位置

/\*获取 e 首次在 alex 中是第一个字母\*/

select t\_user.uname, instr(t\_user.uname, 'e') from t\_user where t\_user.uname = 'Alex';

左对齐

select lpad(t\_user.accountbalance, 10,0) from t\_user;

select lpad(t\_user.accountbalance, 10,'0') from t\_user;

右对齐

select rpad(t\_user.accountbalance, 10,'0') from t\_user;

字符删除

/\*删除首 a\*/

select t\_user.uname, trim(leading 'a' from 'alexa') from t\_user where t\_user.uname = 'Alex';

/\*删除 首位两端的 a\*/

select t\_user.uname, trim(both 'a' from 'alexalexa') from t\_user where t\_user.uname = 'Alex';

/\*删除尾部的a\*/

select t\_user.uname, trim(trailing 'a' from 'alexa') from t\_user where t\_user.uname = 'Alex';

字符替换

/\*将所有的 a 换成 z\*/

select t\_user.uname, replace('alexalexa', 'a', 'z') from t\_user where t\_user.uname = 'Alex';

判断数据为空

/\*如果 积分余额 为空， 就用'没有积分代替'\*/

select t\_user.uname, ifnull(t\_user.pointbalance, '没有积分') from t\_user;

数据精度

/\*保留2位小数， 4舍 5入\*/

select round(123.456, 2), round(123.456, 0), round(123.456, -2) from t\_user;



/\*数据 截断\*/

select truncate(123.456, 2), truncate(123.456, 0), truncate(123.456, -2) from t\_user;



求余

/\*取 余\*/

select mod(123, 2) from t\_user;

求平均值

/\*查询学生表中，学生的 平均学分， 没有学分的数据 不算在内\*/

select avg(t\_student.credit)from t\_student;

/\*查询学生表中，学生的 平均学分， 没有学分的 按0 来算\*/

select avg(ifnull(t\_student.credit, 0))from t\_student;

/\*查询学生表中，学生的 平均学分，按班级进行分组 \*/

select t\_student.cid, avg(ifnull(t\_student.credit, 0))from t\_student group by t\_student.cid;

/\*查询学生表中，学生的 平均学分 大于100的 所在班级的id 和 该班级的平均学分，按班级分组\*/

select t\_student.cid, avg(ifnull(t\_student.credit, 0))

from t\_student

group by t\_student.cid

having avg(ifnull(t\_student.credit, 0))>100;

最值

/\*查询学生表中，学生的 最高学分 大于120的 所在班级的id 和 该班级的平均学分，按班级分组\*/

select t\_student.cid, avg(ifnull(t\_student.credit, 0))

from t\_student

group by t\_student.cid

having max(ifnull(t\_student.credit, 0))>120;

差值

/\*查询学生表中，学生的学分差值，按班级分组\*/

select t\_student.cid, max(ifnull(t\_student.credit, 0)) - min(ifnull(t\_student.credit, 0))

from t\_student

group by t\_student.cid ;

事务

/\*想使用 事务 必须先 开启事务\*/

start transaction;

/\*做回滚操作 必须提交才会生效\*/

rollback;

commit;

保存点

start transaction;

update t\_student

set t\_student.sname = '李白2' where t\_student.sid = 's\_6';

savepoint A;

update t\_student

set t\_student.sname = '李白3' where t\_student.sid = 's\_6';

savepoint B;

rollback to savepoint B;

rollback to savepoint A;

rollback ;

commit;

视图

控制数据访问，简化查询，避免重复访问相同的数据

创建视图

create view v\_student

as

select t\_student.sid, t\_student.sname, t\_student.credit, t\_student.phone, t\_student.age

from t\_student;

修改视图数据

update v\_student

set v\_student.sname = '李大白'

where v\_student.sid = 's\_6';

序列

为多个用户提供唯一的数值的，数据库对象；

自动提供唯一的数值、共享对象、将序列值装入内存提高访问效率；

自定义函数

delimiter

delimiter是自定义分隔符，默认分隔符是 ; 遇到 ; sql语句会生效，执行命令

delimiter $

create function shorten (txt varchar(255), length int)

returns varchar(255)

begin

if isnull(txt) then

return '';

elseif length<15 then

return left(txt, length);

else

if char\_length(txt) <=length then

return txt;

else

return concat(left(txt, length-10), '...', right(txt, 5));

end if;

end if;

end;

$

delimiter ;

存储过程（Stored Procedure）

是一组为了完成特定功能的SQL语句集，是利用SQL Server所提供的Transact-SQL语言所编写的程序。经编译后存储在数据库中,是在MySQL服务器中存储和执行的，可以减少客户端和服务器端的数据传输。存储过程是数据库中的一个重要对象，用户通过指定存储过程的名字并给出参数（如果该存储过程带有参数）来执行它。存储过程是由流控制和SQL语句书写的过程，这个过程经编译和优化后存储在数据库服务器中，存储过程可由应用程序通过一个调用来执行，而且允许用户声明变量 。同时，存储过程可以接收和输出参数、返回执行存储过程的状态值，也可以嵌套调用。

为什么要使用存储过程：

存储过程是已经被认证的技术！

存储过程会使系统运行更快！

存储过程是可复用的组件！它是数据库逻辑而不是应用程序。

存储过程将被保存！

存储过程的优点：

存储过程只在创造时进行编译，以后每次执行存储过程都不需再重新编译，而一般SQL语句每执行一次就编译一次,所以使用存储过程可提高数据库执行速度。

当对数据库进行复杂操作时（如对多个表进行Update、Insert、Query、Delete时），可将此复杂操作用存储过程封装起来与数据库提供的事务处理结合一起使用。

存储过程可以重复使用，可减少数据库开发人员的工作量。

安全性高，可设定只有某此用户才具有对指定存储过程的使用权。

指明子程序使用sql语句的限制

contains sql表示子程序包含sql语句，但不包含读或写数据的语句；

no sql表示子程序中不包含sql语句；

reads sql data表示子程序中包含读数据的语句；

modifies sql data表示子程序中包含写数据的语句。

默认情况下，系统会指定为contains sql。

存储过程

一个存储过程是一个可编程的函数，它在数据库中创建并保存。它可以有SQL语句和一些特殊的控制结构组成。当希望在不同的应用程序或平台上执行相同的函数，或者封装特定功能时，存储过程是非常有用的。数据库中的存储过程可以看做是对编程中面向对象方法的模拟。它允许控制数据的访问方式。

存储过程通常有以下优点：

(1).存储过程增强了SQL语言的功能和灵活性。存储过程可以用流控制语句编写，有很强的灵活性，可以完成复杂的判断和较复杂的运算。

(2).存储过程允许标准组件是编程。存储过程被创建后，可以在程序中被多次调用，而不必重新编写该存储过程的SQL语句。而且数据库专业人员可以随时对存储过程进行修改，对应用程序源代码毫无影响。

(3).存储过程能实现较快的执行速度。如果某一操作包含大量的Transaction-SQL代码或分别被多次执行，那么存储过程要比批处理的执行速度快很多。因为存储过程是预编译的。在首次运行一个存储过程时查询，优化器对其进行分析优化，并且给出最终被存储在系统表中的执行计划。而批处理的Transaction-SQL语句在每次运行时都要进行编译和优化，速度相对要慢一些。

(4).存储过程能过减少网络流量。针对同一个数据库对象的操作（如查询、修改），如果这一操作所涉及的Transaction-SQL语句被组织程存储过程，那么当在客户计算机上调用该存储过程时，网络中传送的只是该调用语句，从而大大增加了网络流量并降低了网络负载。

(5).存储过程可被作为一种安全机制来充分利用。系统管理员通过执行某一存储过程的权限进行限制，能够实现对相应的数据的访问权限的限制，避免了非授权用户对数据的访问，保证了数据的安全。

没有参数

delimiter $

create procedure all\_cname()

begin

select cname from t\_class;

end;

$

delimiter ;

call all\_cname;



输入参数

delimiter $

create procedure selectByClsssName(in cname varchar(8))

begin

select \* from t\_student where cid = (

select t\_class.cid from t\_class where t\_class.cname = cname

);

end;

$

delimiter ;

call selectByClsssName ('一班');

输入输出参数

delimiter $

/\*通过 班级名称 获取 班级 id\*/

create procedure selectClassIdByClassName(in cname varchar(8) , out cid int)

begin

declare tmpCid int;

select t\_class.cid into tmpCid from t\_class where t\_class.cname = cname;

set cid = tmpCid;

end

$

delimiter ;

set @tmpCid = 1;

call selectClassIdByClassName('一班', @tmpCid);

insert into t\_student

set

sid = null,

sname = '王云',

cid = @tmpCid,

phone = '13146008028',

pwd = '123456',

age = 24,

credit = 100

;

set @tmpCid = 1;

call selectClassIdByClassName('一班', @tmpCid);

insert into t\_student values(null, '梁家辉', @tmpCid, '13146008029', '123456', 24, 120);

select \* from t\_student;

删除存储过程

drop procedure selectClassIdByClassName;

常见问题

时间 和 日期 是可以进行算数运算的；

凡是null 参与的运算，结果仍是null，空值不等于0；

为什么要使用外键？

如果关联表中没有对应的value，本次插入操作视为无效，不会造成脏数据。

主键和外键的作用

(1)插入非空值时，如果主键表中没有这个值，则不能插入。

(2)更新时，不能改为主键表中没有的值。

(3)删除主键表记录时，你可以在建外键时选定外键记录一起级联删除还是拒绝删除。

(4)更新主键记录时，同样有级联更新和拒绝执行的选择。

简而言之，SQL的主键和外键就是起约束作用。

engine=innodb;

存储引擎是innodb。nnoDB 是 MySQL 上第一个提供外键约束的数据存储引擎，除了提供事务处理外，InnoDB 还支持行锁，

提供和 Oracle 一样的一致性的不加锁读取，能增加并发读的用户数量并提高性能，不会增加锁的数量。

InnoDB 的设计目标是处理大容量数据时最大化性能，它的 CPU 利用率是其他所有基于磁盘的关系数据库引擎中最有效率的。

InnoDB 是一套放在 MySQL 后台的完整数据库系统，InnoDB 有它自己的缓冲池，能缓冲数据和索引，InnoDB 还把数据和索引存放在表空间里面，

可能包含好几个文件，这和 MyISAM 表完全不同，在 MyISAM 中，表被存放在单独的文件中，

InnoDB 表的大小只受限于操作系统文件的大小，一般为 2GB。

MySQL常见错误

Error Code: 1175. You are using safe update mode and you tried to update a table without a WHERE that uses a KEY column To disable safe mode, toggle the option in Preferences -> SQL Editor and reconnect.

解答： SET SQL\_SAFE\_UPDATES = 0;

where 和 having

如果sql语句中用到了组函数，需要将where 换成 having，因为having慢于组函数，where快于组函数