--关系型数据库

--mysql修改密码

1、登录mysql：mysql -uroot -p密码

2、使用mysql数据库：user mysql

3、修改密码:alter user '用户名'@'localhost' idenitfied with mysql_native_password by '新密码';

4、重启mysql验证：exit;

--终端操作数据库

1、查询服务器中的所有数据库

show databases;

2、如何选中一个数据库进行操作

use 数据库名（如：use mysql）

查看当前数据库下的表：show tables;

查询表结构：describe 表;

sql中的查询语句：select * from 表;

3、创建数据表

```
create table 表名（
    数据字段 数据类型,
    数据字段 数据类型）；
```

查看表结构：describe 表名;

4、往表中添加数据

```
insert into 表名
    values('字段','字段');
```

5、从表中删除数据

delete from 表名 where 字段= ‘值’ ；

如：delete from user where name='张三';

6、在表中修改数据

update 表名 set 字段= ‘修改值’ where 字段= ‘值’ ;

如：update user set name='张三' where id=2;

--数据类型

MySQL可以分为三类：数值、日期/时间和字符串(字符)类型。

数值类型

类型	大小	范围（有符号）	范围（无符号）	用途
TINYINT	1 byte	(-128，127)	(0，255)	小整数值
SMALLINT	2 bytes	(-32 768，32 767)	(0，65 535)	大整数值
MEDIUMINT	3 bytes	(-8 388 608，8 388 607)	(0，16 777 215)	大整数值
INT或INTEGER	4 bytes	(-2 147 483 648，2 147 483 647)	(0，4 294 967 295)	大整数值
BIGINT	8 bytes	(-9,223,372,036,854,775,808，9 223 372 036 854 775 807)	(0，18 446 744 073 709 551 615)	极大整数值
FLOAT	4 bytes	(-3.402 823 466 E+38，-1.175 494 351 E-38)，0，(1.175 494 351 E-38，3.402 823 466 351 E+38)	0，(1.175 494 351 E-38，3.402 823 466 E+38)	单精度浮点数值
DOUBLE	8 bytes	(-1.797 693 134 862 315 7 E+308，-2.225 073 858 507 201 4 E-308)，0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308)	0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308)	双精度浮点数值
DECIMAL	对DECIMAL(M,D)，如果M>D，为M+2否则为D+2	依赖于M和D的值	依赖于M和D的值	小数值

日期类型

类型	大小 (bytes)	范围	格式	用途
DATE	3	1000-01-01/9999-12-31	YYYY-MM-DD	日期值
TIME	3	'-838:59:59'/'838:59:59'	HH:MM:SS	时间值或持续时间
YEAR	1	1901/2155	YYYY	年份值
DATETIME	8	1000-01-01 00:00:00/9999-12-31 23:59:59	YYYY-MM-DD HH:MM:SS	混合日期和时间值
TIMESTAMP	4	1970-01-01 00:00:00/2038 结束时间是第 **2147483647** 秒，北京时间 **2038-1-19 11:14:07**，格林尼治时间 2038年1月19日 凌晨 03:14:07	YYYYMMDD HHMMSS	混合日期和时间值，时间戳

字符串类型

类型	大小	用途
CHAR	0-255 bytes	定长字符串
VARCHAR	0-65535 bytes	变长字符串
TINYBLOB	0-255 bytes	不超过 255 个字符的二进制字符串
TINYTEXT	0-255 bytes	短文本字符串
BLOB	0-65 535 bytes	二进制形式的长文本数据
TEXT	0-65 535 bytes	长文本数据
MEDIUMBLOB	0-16 777 215 bytes	二进制形式的中等长度文本数据
MEDIUMTEXT	0-16 777 215 bytes	中等长度文本数据
LONGBLOB	0-4 294 967 295 bytes	二进制形式的极大文本数据
LONGTEXT	0-4 294 967 295 bytes	极大文本数据

--mysql链表约束

主键约束：

它能够唯一确定一张表的一条记录，通过对某一字段添加约束，使该字段不重复不为空。

如：create table user(

-> id int primary key, 其中primary key就是添加主键约束

-> name varchar(20),

-> age int, 这样数据库当中相同的id就只能有一个

-> birthday date);

联合主键

如：create table user2(

-> id int,

-> name varchar(20), 只要主键值加起来不重复，就可以添加

-> birthday date,

-> primary key(id,name));

自增约束

如： create table user3(

-> id int primary key auto_increment, 这里指定id为自增主键，可以不指定id的值

-> name varchar(20),

-> birthday date);

创建表时忘记添加主键约束

alter table 表名 add primary key(字段);

删除主键约束

alter table 表名 drop primary key;

修改主键约束

alter table 表名 modify 字段 类型 primary key;

唯一约束

alter table 表名 add unique(字段);

删除唯一约束

alter table 表名 drop index 字段；(多条约束时，删除最先一个)

modify添加

alter table 表名 modify 字段 字段类型 unique;

非空约束（修饰的字段不能为空）

如：create table user3(

-> id int ,

-> name varchar(20) not null,

-> birthday date);

如：alter table 表名 change column 字段 字段 字段类型 not null;

默认约束(插入字段值的时候没有传值，自动添加默认值)

如：create table user5(

-> id int,

-> name varchar(20),

-> age int default 21);　　这里给了id一个默认值21

外键约束（涉及两个表，主表和副表）

如：**主表** class

create table class(

-> id int primary key,

-> name varchar(20));

副表 student

create table student(

-> id int primary key,

-> name varchar(20),

-> class_id int,

-> foreign key (class_id) references class(id));

主表中没有的数据，在副表中是不能创建的

主表中被引用的数据是不能被删除

--数据库的三大设计范式

第一范式（表中的所有字段都是不可分割的原子值，即字段拆分的越小对某些操作越好）

例：create table user(
id int primary key,
name varchar(20),
address varchar(20));

第一范式：create table user(
id int primary key,
name varchar(20),
country varchar()20,
province varchar(20),
city varchar(20),
specific varchar(20));

第二范式（必须满足第一范式，除主键外，其他字段完全依赖主键）

例：create table myorder(
order_ id int primary key,
product_ id int,
customer_ id int);

第二范式：create table product(
id int primary key,
name varchar(20));

create table customer(
id int primary key,
name varchar(20));

第三范式（除开主键外地其余字段之间不能有传递依赖关系）

例：create table myorder(
order_ id int primary key,
product_ id int,
customer_ id int
customer_ phone varchar(15));

第三范式：create table myorder(
order_ id int primary key,
product_ id int,
customer_ id int);

create table customer(
id int primary key,
name varchar(20)

phone varchar(15));

--查询练习

新建学生表：mysql> create table student(
-> sno varchar(20) primary key, 学号
-> snmae varchar(20) not null, 姓名
-> ssex varchar(10) not null, 性别
-> sbirthday datetime, 出生日期
-> class varchar(20)); 班级

新建教师表：mysql> create table teacher(
-> tno varchar(20) primary key, 教师编号
-> tname varchar(20) not null, 姓名
-> tsex varchar(10) not null, 性别
-> tbirthday datetime, 出生日期
-> prof varchar(20) not null, 职称
-> depart varchar(20) not null); 所在部门

新建课程表：mysql> create table course(
-> cno varchar(20) primary key,
-> cname varchar(20) not null,
-> tno varchar(20) not null,
-> foreign key(tno) references teacher(tno));

新建成绩表：mysql> create table score(
-> sno varchar(20) not null,
-> cno varchar(20) not null,
-> degree decimal,
-> foreign key(sno) references student(sno),
-> foreign key(cno) references course(cno),
-> primary key(sno,cno));

查询student表中所有记录

```
select * from student;
```

查询student表中sname，ssex，class的所有记录

```
select sname,ssex,class from student;
```

查询教师所有不重复的depart

```
select distinct depart from teacher;
```

查询score表中在60-80的所有记录

```
select * from score where degree between 60 and 80;
```

或者

```
select * from score where degree > 60 and degree < 80;
```

查询score表中成绩为70或80或90的所有记录

```
select * from score where degree in(70,80,90);
```

查询student中1班或者性别为女的同学

```
select * from student where class='1' or ssex='女';
```

以class倒叙查询student表中的所有记录

```
select * from student order by class desc;(asc升序,desc降序，默认升序)
```

以cno升序、degree降序查询score表的所有记录。

```
select * from score order by cno asc,degree desc;
```

查询1班人数

```
select  count(*) from student where class='1';
```

查询score表中最高分的学生学号和课程号。（子查询或者排序）

```
select sno,cno from score where degree=(select max(degree) from score);
```

查询每门课的平均成绩

单门课程：

```
select avg(degree) from score where cno='001';
```

全部查询：

```
select cno,avg(degree) from score group by cno;
```

查询score表中有三位同学并以1开头的课程的平均分

```
select cno,avg(degree)  from score group by cno
having count(*)>=3 and cno like '1%';
```

查询分数大于70小于90的sno列

```
select sno,degree from score
where degree between 70 and 90;
```

查询所有学生的sname,cno和degree列

```
select sname,cno,degree
from student,score
where student.sno=score.sno;
```

查询所有学生的sno，cname和degree列

```
select sno,cname,degree
from course,score
where course.cno=score.cno;
```

查询所有学生的sname，cname，sno和degree列

select sname,cname,student.sno,degree 多表查询时，共用字段要加上表名

from student,course,score

where student.sno=score.sno

and course.cno=score.cno;

查询1班学生每门课的平均分

select cno,avg(degree)

from score

where sno in (select sno from student where class='1') 将查询到的学生当做条件

group by cno;

查询选修101课程分数高于2001同学003课程分数的所有同学记录

select * from score

where cno='101' and

degree>(select degree from score where sno='2001' and cno='003');

查询和学号1001,2001同学同年出生的所有学生的sno，sname和sbirthday列

select * from student where year(sbirthday) in

(select year(sbirthday) from student where sno in (1001,2001));

查询教师“王大锤”任课的学生成绩(多层嵌套查询)

select * from score

where cno=(select cno from course

where tno=(select tno from teacher where tname='王大锤'));

查询某课程人数大于4人的教师姓名

select tname from teacher where tno in(

select tno from course where cno in(

select cno from score group by cno having count(*)>4));

查询“高一”与“高二”组不同职称的老师的tname和prof

select * from teacher where depart='高一组'

and prof not in(select prof from teacher where depart='高二组') 排除不属于的

union 求并集

select * from teacher where depart='高二组'

and prof not in(select prof from teacher where depart='高一组');

查询001课程中成绩比003课程任意成绩高的学生的cno,sno，degree，并按degree降序排序

```
select cno,sno,degree from score
where cno='001' and
degree>any(select degree from score where cno='003')
order by degree desc;
```

查询001课程中成绩比003课程任意成绩高的学生的cno,sno，degree

```
select cno,sno,degree from score
where cno='001' and
degree>all(select degree from score where cno='003');
```

查询所有学生和教师的name，sex和birthday

```
select tname as 姓名,tsex as 性别,tbirthday as 生日 from teacher
union
select sname,ssex,sbirthday from student;
```

查询比该课程平均成绩低的同学成绩

这里将score中的数据复制了一份，分为a，b。将a表中的数据按课程号去和b表中的平均值

进行比较

```
select * from score a where
degree < (select avg(degree) from score b where a.cno=b.cno);
```

查询所有任课教师（课程表中有课程）的tname和drpart

```
select * from teacher where tno in (select tno from course);
```

查询至少有两名男生的班号

```
select class from student where ssex='男' group by class having count(*)>1;
```

查询student中不姓王的同学

```
select * from student where sname not like '王%';
```

查询student表中的每个学生的姓名和年龄（当前年份-出生年份）

```
select year(now());                    查询当前年份
select sname,year(now())-year(sbirthday) as 年龄 from student;
```

新建一个grade表查询所有同学的sno，cno，rank列

```
select sno,cno,grade from score,grade where degree between low and upp;
```

--四种连接查询

内连接（inner join 或 join）

select * from person inner join card on person.cardId=card.id;

外连接

左连接（left join 或 left outer join）

会把左边所有数据取出来，右边表数据如果有相当就显示出来，如果没有就显示null

select * from person left join card on person.cardId=card.id;

右连接（right join 或 right outer join）

会把右边所有数据取出来，左边表数据如果有相当就显示出来，如果没有就显示null

select * from person right join card on person.cardId=card.id;

完全外连接（full join 或 full outer join）

mysql不支持fulljoin，效果和union等同

--mysql事务

一个最小的不可分割的工作单元，事务能够保证一个业务的完整性

多条sql语句，可能会有同时成功的要求，要么就是同事失败

mysql如何开启事务（默认开启）

select @@autocommit;

开启事务，我们去执行一个sql语句时，效果会立即体现出来，且不能回滚。

事务回滚：撤销sql语句的执行效果

rollback;

设置mysql自动提交为false可以进行事务回滚

set autocommit=0;

手动提交（commit;）

在自动提交的前提下，begin或者start transaction都可以帮我们开启一个事务，确认无误，再commit手动提交

事务的四大特征

A　原子性：事务是最小的单位，不可以在分割

C　一致性：事务要求，同一事务的sql语句，必须同时成功或者同时失败

I　隔离性：事务1 和 事务2 之间是有隔离性的

D　持久性：事务一旦结束，就不可返回。

事务的隔离性：

1、read uncommitted；　　读未提交的

如果有事务a 和事务b

a 事务对数据进行操作，在操作过程中，事务没有被提交，但b 可以看到 a操作的结果

查看数据库的隔离等级(mysql 8.0)

select @@global.transaction_isolation;　　系统级别

select @@transaction_isolation;　　会话级别

修改级别等级

set flobal transaction isolation leven read committed;

如果两个地方在没有提交的时候都可以读取事务，这就是（脏读）,在读取后可能会出现

事务回滚的情况。

脏读：一个事务读取到了另一个事务没有提交的数据。（实际开发中是不允许的）

2、read committed;　　读已提交的

虽然可以读取已提交的数据，但如果在事务1 第二次读取时提交数据就会出现前后读取

数据不一致，这就是不可重复读现象

3、repeatable read；　　可以重复读（默认级别）

事务a 和事务b 同时操作一张表，事务a 提交的数据也不能被事务b 读到，就可以造成幻

读

4、serializable；　　串行化

同一个表被一个事务操作的时候，其他事务的写操作是不可进行的，将会进入排队等待

状态，当事务操作结束的时候（commit），写入操作才会执行，长时间等待会出现超时

现象。

隔离性越高，性能最差，串行化性能最差！！！

read uncommitted > read committed > repeatable read > serializable