数据库的概念

基本概念

数据库简而言之可以看做电子化的文件柜，存储电子文件的处所，用户可以对文件中的数据进行增删改查。

分类

关系型数据库：MySQL

MySQL由Oracle开发，所使用的SQL语言是访问数据库最常用的标准化语言

非关系形数据库 NoSQL

创建数据表

登录：

```
mysql -uroot -p123456
```

查询有哪些数据库：

```
show database
```

选定数据库：

```
use db_name
```

查询数据库记录：

```
select * from t_name
```

退出数据库：

```
exit
```

创建数据库：

```
create database db_name;
```

显示数据表：

```
use db_name;
show tables;
```

创建数据表：

```
use db_name;
create table user (
name VARCHAR(20),
age VARCHAR(20),
species VARCHAR(20),
sex char(1),
birth DATE
);
```

查看数据表的创建结构：

```
DESCRIBE t_name;
```

表中添加数据：

```
insert into db_name
values ("keran","21","human",'1',"1994-04-04");
```

数据类型

MySQL支持多种类型，大致可以分为三类：数值、日期/时间和字符串(字符)类型。

数值类型

MySQL支持所有标准SQL数值数据类型。

这些类型包括严格数值数据类型(INTEGER、SMALLINT、DECIMAL和NUMERIC)，以及近似数值数据类型(FLOAT、REAL和DOUBLE PRECISION)。

关键字INT是INTEGER的同义词，关键字DEC是DECIMAL的同义词。

BIT数据类型保存位字段值，并且支持MyISAM、MEMORY、InnoDB和BDB表。

作为SQL标准的扩展，MySQL也支持整数类型TINYINT、MEDIUMINT和BIGINT。下面的表显示了需要的每个整数类型的存储和范围。

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

类型	大小	范围（有符号）	范围（无符号）	用途
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

日期和时间类型

表示时间值的日期和时间类型为DATETIME、DATE、TIMESTAMP、TIME和YEAR。

每个时间类型有一个有效值范围和一个"零"值，当指定不合法的MySQL不能表示的值时使用"零"值。

TIMESTAMP类型有专有的自动更新特性，将在后面描述。

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	YYYYMMDD HHMMSS	混合日期和时间值，时间戳

字符串类型

字符串类型指CHAR、VARCHAR、BINARY、VARBINARY、BLOB、TEXT、ENUM和SET。该节描述了这些类型如何工作以及如何在查询中使用这些类型。

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

注意：char(n) 和 varchar(n) 中括号中 n 代表字符的个数，并不代表字节个数，比如 CHAR(30) 就可以存储 30 个字符。

CHAR 和 VARCHAR 类型类似，但它们保存和检索的方式不同。它们的最大长度和是否尾部空格被保留等方面也不同。在存储或检索过程中不进行大小写转换。

BINARY 和 VARBINARY 类似于 CHAR 和 VARCHAR，不同的是它们包含二进制字符串而不要非二进制字符串。也就是说，它们包含字节字符串而不是字符字符串。这说明它们没有字符集，并且排序和比较基于列值字节的数值值。

BLOB 是一个二进制大对象，可以容纳可变数量的数据。有 4 种 BLOB 类型：TINYBLOB、BLOB、MEDIUMBLOB 和 LONGBLOB。它们区别在于可容纳存储范围不同。

有 4 种 TEXT 类型：TINYTEXT、TEXT、MEDIUMTEXT 和 LONGTEXT。对应的这 4 种 BLOB 类型，可存储的最大长度不同，可根据实际情况选择。

数据表的增删改查

表中添加数据：

```
insert into db_name;
values ("keran","21","human",'1',"1994-04-04");
```

表中删除数据：

```
delete from db_name where name="keran";
```

表中修改数据：

```
update t_name set age="29" where name="keran";
```

表中查询数据：

```
select * from t_name where name="keran";
```

SQL约束

主键约束

能过唯一确定一张表中的一条数据，通过给某个字段添加主键约束，可以使得字段不重复且不为空

```
create table user1(
    id int primary key,
    name varchar(20)
);
insert into user1 values(1,"张三");
```

联合主键

```
create table user1(
    id int,
    name varchar(20),
    PRIMARY key(id,name)
);
insert into user1 values(1,"zhangsan");
insert into user1 values(2,"lisi");
```

联合主键中的字段都不可以为空；只要整体不重复即可。

当忘记创建主键或者要删除逐渐时，

```
alter table user add primary key(id);
alter table user drop primary key(id);
```

自增约束

```
create table user2(
    id int PRIMARY key AUTO_INCREMENT,
    name varchar(20)
);
insert into user2(name) values("zhangsan");
insert into user2(name) values("lisi");
insert into user2(name) values("lisi");
SELECT * from user2;
```

自增约束管控id自动增长，和主键约束共同使用。

唯一约束

被约束的字段不可以重复，可以为空

```
create table user3(
    id int,
    name varchar(20) UNIQUE,
);
insert into user3 values(1,"zhangsan");
insert into user3 values(2,"");
```

联合唯一中的字段只要整体不重复即可。

非空约束

被约束的字段不可以为空

```
create table user4(
    int id not null,
    name varchar(20)
)
```

默认约束

当对被约束字段没有传值，以默认字段赋值,传值则以传入值为准。

```
create table user5(
    int id,
    int age default 10
)
```

外键约束

涉及两个表，主表和副表。

副表中不可以设定主表中不存在的值。

副表不可以删除主表的元素

```
create table class(
    id int PRIMARY key,
    name varchar(20)
);

create table student(
    id int primary key,
    name varchar(20),
    class_id int,
    foreign key(class_id) references class(id)
);

insert into  class values(1,"class_1");
insert into  class values(2,"class_2");
insert into  class values(3,"class_3");
insert into  class values(4,"class_4");

insert into  student values(1001,"zhangsan",1);
insert into  student values(1002,"lisi",1);
```

```
insert into  student values(1003,"wangwu",4);
```

数据库设计范式

第一范式

数据表中的所有字段都是不可以分割的原子。

例如地址可以拆分成国家-省份-城市-乡镇

字段值还可以进行进一步拆分的就不满足第一范式。

范式设计的越详细，对于某些操作可能更好，但是不一定是好处。

第二范式

必须在满足第一范式的前提下，除了主键外的每一列必须都要依赖主键，如果不依赖主键，只可能发生在联合主键的情况下。一般拆表来解决该问题。

```
create table morder(
    order_id int primary key,
    product_id int,
    customer_id int
);
create table product(
    id int primary key,
    name varchar(20)
);
create table customer(
    id int primary key,
    name varchar(20)
);
```

第三范式

必须必须要满足第二范式，除主键外的其他字段不可以有依赖关系。

实验

准备数据库数据

```

use test;
drop table if exists student;
drop table if exists teacher;
drop table if exists course;
drop table if exists score;
create table student(
    student_id VARCHAR(20) primary key,
    student_name varchar(20) not null,
    student_gender varchar(10) not null,
```

```
    student_birth datetime,
    class varchar(20)
)DEFAULT CHARSET=utf8;

insert into student VALUES ("101","曾华","男","1997-01-09","9003");
insert into student VALUES ("102","贾玲","女","1977-11-19","9003");
insert into student VALUES ("103","花花","男","1997-01-21","9013");
insert into student VALUES ("104","小童","女","1987-01-09","9013");
insert into student VALUES ("105","沈腾","男","1987-03-11","9003");

 create table teacher(
    teacher_id varchar(20) primary key,
    teacher_name varchar(20) not null,
    teacher_gender varchar(10) not null,
    teacher_birth datetime,
    teacher_title varchar(20) not null,
    dept varchar(20) not null
 )CHARSET=utf8;

 insert into teacher VALUES ("901","老张","男","1957-01-09","教授","电信");
 insert into teacher VALUES ("902","老李","女","1947-11-29","教授","材料");
 insert into teacher VALUES ("903","老王","男","1977-01-24","副教授","通信");

 create table course(
    course_id varchar(20) primary key,
    course_name varchar(20) not null,
    teacher_id varchar(20) not null,
    FOREIGN KEY(teacher_id) REFERENCES teacher(teacher_id)
 )CHARSET=utf8;
  insert into course VALUES ("1001","控制理论","901");
  insert into course VALUES ("1002","微机原理","902");
  insert into course VALUES ("1003","控制理论","903");

 create table score(
    course_id varchar(20) not null,
    student_id varchar(20) not null,
    degree decimal(18,1),
    FOREIGN KEY(course_id) references course(course_id),
    FOREIGN KEY(student_id) references student(student_id),
    primary key(course_id,student_id)
 )CHARSET=utf8;
   insert into score VALUES ("1001","101",89.4);
   insert into score VALUES ("1001","102",59.3);
   insert into score VALUES ("1001","103",49.5);
   insert into score VALUES ("1001","104",39.4);
   insert into score VALUES ("1002","101",99.1);
   insert into score VALUES ("1002","102",29.4);
   insert into score VALUES ("1002","104",78.4);
   insert into score VALUES ("1003","101",69.9);
   insert into score VALUES ("1003","103",56.4);
   insert into score VALUES ("1003","104",98.2);
   insert into score VALUES ("1003","105",45.4);
```

简单练习

```
-- 查询所有学生记录
```

```
select * from student
-- 查询学生表中的部分字段
select student_id, student_name from student;
-- 查询教师表中所有部分的不重复序列--
select DISTINCT dept from teacher;
-- 查询成绩中介于60到80的
select * from score where degree between 60.0 and 80.0;
-- 查询score=29.4或者78.4的数据
select * from score where degree in (29.4,78.4,49.8);
-- 查询学生表中9003的男同学
select * from student where class="9003" and student_gender="男"
-- 成绩升序或者降序，默认升序asc，降序desc
select * from score order by degree;
-- 成绩升序，学号降序，默认升序asc，降序desc
select * from score order by degree asc,student_id desc;
-- 查询选择1001课程的人数
select count(*) from score where course_id = "1001";
-- 查找score最高分的学生学号和课程号
select student_id,course_id from score where degree=(select max(degree) from
score );
-- 查询选择1001的课的学生平均成绩
select avg(degree) from score where course_id="1001";
-- 查询每一门课的学生平均成绩
select course_id, avg(degree) from score group by course_id;
-- 查询score表中以3结尾,且至少有两个学生选择的课程的平均成绩,以have带条件，% 模糊查询
select course_id  from score group by course_id
having count(course_id)>2 and course_id like "%3";
```

多表联查

```
-- 查询所有学生的姓名，课程号和成绩
-- 共同字段包含学生的学号

select student_name,course_id,degree from student,score
where student.student_id = score.student_id;

-- 查询所有学生的学号，课程名和成绩
-- 共同字段包含课程号
select student_id,course_name,degree from course,score
where course.course_id = score.course_id;

-- 查询所有学生的姓名，课程名和成绩
-- 共同字段包含学生的学号和课程号
select student_name,course_name,degree from student,course,score
where student.student_id=score.student_id and course.course_id =
score.course_id;
```

子表查询

```
-- 查询9003班的学生每一门课的平均分
select student_id from student where class="9003";
select course_id,avg(degree) from score
where student_id in (select student_id from student where class="9003")
```

```
group by course_id;

-- 查询选择课“1001”并且成绩高于"102"号同学的所有同学的成绩
select degree from score where course_id="1001" and student_id="102";
select * from score where degree>(select degree from score where
course_id="1001" and student_id="102") and course_id="1001";


-- 查询和学号为“104”同学同年出生的所有学生的学号，姓名和生日
select year(student_birth) from student where student_id="104";
select student_id,student_name,student_birth from student WHERE
 year(student_birth)=(select year(student_birth) from student where
student_id="104");
```

多层嵌套查询

```
-- 查询老王带的课的学生成绩
select teacher_id from teacher where teacher_name="老王";
select course_id from course where teacher_id=(select teacher_id from teacher
where teacher_name="老王");
select  * from score where course_id = (select course_id from course where
teacher_id=(select teacher_id from teacher where teacher_name="老王"));
```

条件查询

```
-- 查询9003和9013班的学生  in
select * from student where class in ("9003","9013");

-- 查询材料和电信专业不同职称教师的姓名和职称  union 取并集
select * from teacher where dept="材料" and teacher_title not in (select
teacher_title from teacher where teacher_title="电信")
union
select * from teacher where dept="电信" and teacher_title not in (select
teacher_title from teacher where teacher_title="材料");

-- 查询选择1001课程，且成绩至少高于选择1002的成绩的同学  any
select * from score where course_id="1001" and degree> any(select degree from
score where  course_id="1002");

-- 查询选择1002课程，且成绩高于所有选择1003的成绩的同学  all
select * from score where course_id="1002" and degree> all(select degree from
score where  course_id="1003");
```

SQL四种连接查询

内连接

inner join 或者join

准备数据

```
use test_join;
```

```
create table person(
    id int,
    name varchar(20),
    cardId int
)ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;

create table card(
    id INT,
    name varchar(20)
)ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;

insert into card  values (1,"工行卡");
insert into card  values (2,"农行卡");
insert into card  values (3,"建设行卡");
insert into card  values (4,"中国行卡");
insert into card  values (5,"浦发行卡");
insert into person  values (1,"张三",1);
insert into person  values (2,"李四",3);
insert into person  values (3,"王五",5);
```

测试

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

id	name	cardId	id	name
1	张三	1	1	工行卡
2	李四	3	3	建设行卡

两张表中的数据，通过某个字段相等查询出相关记录数据。

外连接

左连接

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

id	name	cardId	id	name
1	张三	1	1	工行卡
2	李四	3	3	建设行卡
3	王五	6	0	

左连接会将左边的表中的所有数据都取出来，而右边表中的数据，如果相等，则显示；如果不存在没补null

右连接

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

id	name	cardId	id	name
1	张三	1	1	工行卡
2	李四	3	3	建设行卡
0		0	2	农行卡
0		0	4	中国行卡
0		0	5	浦发行卡

右连接会将右边的表中的所有数据都取出来，而左边表中的数据，如果相等，则显示；如果不存在没补null

完全外连接

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

id	name	cardId	id	name
1	张三	1	1	工行卡
2	李四	3	3	建设行卡
3	王五	6	0	
0		0	2	农行卡
0		0	4	中国行卡
0		0	5	浦发行卡

SQL 事务

最小的不可分割的工作单元，事物能够保证一个业务的完整性。

持久性

例如 在银行转账系统中，a取出100，b存入100。

```
update user set money=money-100 where name="a";
update user set money=money+100 where name="b";
```

假设一条数据成功，一条数据失败，那么就出现数据前后不一致的问题。

多条sql语句就要求同时成功或者同时失败的要求。

MySQL默认事务开启：执行sql时，效果立马体现，且不回滚。

```
use transaction;
set autocommit = 0;  手动关闭事务的自动提交
-- drop table if exists user;
-- create table user(
--  id int,
--     name varchar(20),
--     money int
-- );

-- insert into user values(1,"a",1000);
-- select * from user;
rollback;
select * from user;
-- insert into user values(2,"b",1000);
commit;  手动提交
```

隔离性

read uncommitted—>read committed—>repeatable read—>serializable

隔离界别越高，性能越差，默认为repeatable read

read uncommitted

读未提交的

两个事务a和b，a事务对数据进行操作，但没有提交，b可以看见a操作的结果。

```
set global transcation isolation level read uncommitted;
insert into user values(1,"顾客",1000);
insert into user values(2,"淘宝",1000);
start transcation;
事务a：update user set money=money-700 where name="顾客";
事务b: update user set money=money+700 where name="淘宝";
事务a: rollback;
```

脏读:一个事务读到另一个数据没有提交的数据，实际开发中不允许出现这种情况。

read committed

读已经提交的

```
set global transcation isolation level read committed;
insert into user values(1,"顾客",1000);
insert into user values(2,"淘宝",1000);
start transcation;
事务a：insert into user values(3,"骗子",400);
事务b: select avg(money) from user;
```

不可重复读:读取同一个表的数据，但是得到前后不一致的结果。

repeatable read

重读读

```
set global transcation isolation level repeatable read;
事务a：
start transaction;

事务b:
start transaction;
insert into user values(4,"hehe",1000);
commit;

事务a：
无法读到事务b修稿的数据
```

幻读:事务a和事务b同时操作一张表，但事务a提交的表也不能被事务b读到，就会造成幻读。。

serializable

串行化

```
set global transcation isolation level serializable;
事务a：
start transaction;

事务b:
start transaction;
insert into user values(5,"bobo",1000);
commit;

事务a：
可以查询到事务a的操作；
insert into user values(5,"BOBO",1000);
会卡死，串行化。
```

当一个表被其他事务操作时，其他事务里面的写操作不可以进行，进入排队状态，直到其他事务结束后，该事物才会进行写入操作（并且没有等待超时）。

开启事务

手动开启事务，begin或者start transcation开启事务。

```
start;
update user set money=money-100 where name="a";
update user set money=money+100 where name="b";
rollback;
commit;
```

事务特征

A：原子性：事务的最小单位，不可以分割。

C：一致性：事务要求，同一个事务中的sql语句，必须保证同时成功或者失败。

I：隔离性：事务之间具有隔离性。

D：持久性：事务一旦结束，不可以返回。