# MySQL数据库

## 数据库初识

### 概念篇

什么是数据库？

数据库就是一个文件系统，通过标准SQL语言（W3C组织规定）操作文件系统中数据，用来存放数据的软件系统

表面上来看我们学习的是MySQL，实际上学习的是SQL语言，MySQL数据库是在SQL语言基础上进行优化得来的数据库

不仅仅是MySQL，市面上所有的数据库软件都是通用的

为什么使用数据库？

因为使用文件存储数据有安全性的问题。对数据的增删改查也不灵活，存放大量数据时也不方便，所以才有了数据库

数据库的特征：

* 数据结构化
* 实现数据共享
* 减少数据冗余
* 数据独立性

数据库类型分为：

* 网状数据库
  + 采用记录类型为节点的网状数据模型
* 层次数据库
  + 采用层次模型模拟现实世界中按层次组织起来的事物
* 关系数据库
  + 采用二维表结构组织和管理数据，并规定了表内和表见数据的依赖关系

### SQL分类

**DDL （数据定义语言）**

数据定义语言 - Data Definition Language

用来定义数据库的对象，如数据表、视图、索引等

**DML （数据操纵语言）**

数据处理语言 - Data Manipulation Language

在数据库表中更新，增加和删除记录

如 update， insert， delete

**DCL （数据控制语言）**

数据控制语言 – Data Control Language

指用于设置用户权限和控制事务语句

如grant，revoke，if…else，while，begin transaction

**DQL （数据查询语言）**

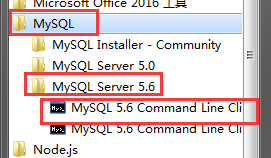
数据查询语言 – Data Query Language

select

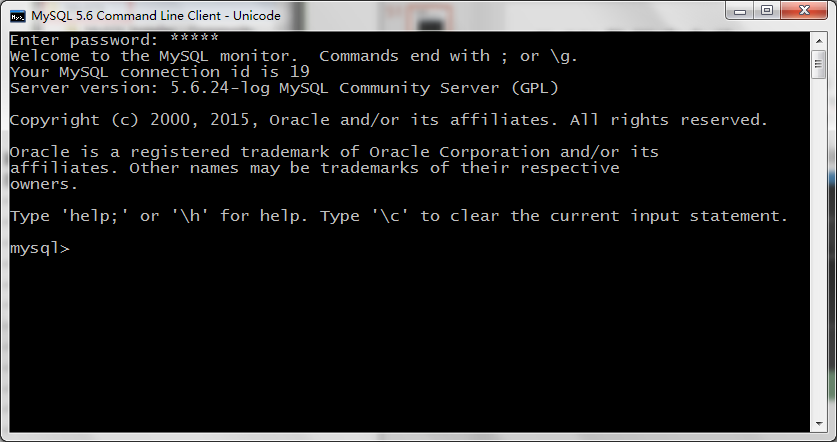
**学习SQL语言，按照学习路线：**

1. **数据库的操作**
2. **数据表的操作**
3. **数据表中数据的操作**

安装MySQL后，在开始菜单中依次选择以下目录，打开MySQL命令行工具



首先要求你输入管理员用户名root的密码，输入完成后，按下回车键



mysql> 是MySQL有、命令行工具的提示符，可以在他的后面跟上MySQL命令：

启动数据库服务：

net start mysql;

开启数据库：

mysql –n root –p

开启指定数据库：

mysql （-h主机ip -P3306）-u 用户名 –p

退出命令行工具的代码为：

quit / exit

选择将要操作的数据库：

use 数据库名;

查看目标数据库相关信息：（数据库版本号，字符集等）

show create database 数据库名；

查看具体的表结构：

describe 表名；

查看数据库服务器和字符集：

show variables like ‘%character%’；

查看MySQL数据列（column）的字符集

show full columns from user;

引用扩展名为sql的文件：

source 文件名.sql；

查看MySQL服务实例支持的存储引擎： show engines；

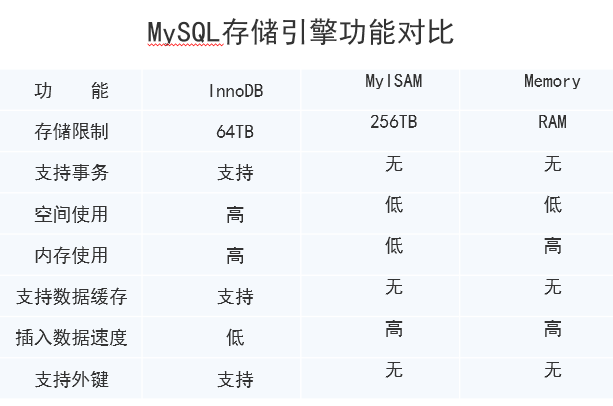
一般MySQL5.0默认的存储引擎为**InnoDB**，使用MySQL命令【set default storage engine = 存储引擎类型】，可以临时的将他设置为默认引擎

InnoDB存储引擎的特点有：

* 支持外键
* 支持事物
* 最新版的MySQL还支持全文检索

除开InnoDB存储引擎，还有MyINAM引擎，MyINAM引擎的特点是：

* + - * 具有检查和修复表的大多数工具
      * 可以被压缩
      * 最早支持全局索引
      * 不支持事务
      * 不支持外键



### DOS窗口下中文乱码

你可能会发现， 在Navicat软件下的增删改查过程中没有任何问题，然而移步到DOS窗口下时会发现有中文乱码的情况，这是因为在安装数据库的时候选择的数据库默认编码表为utf8，然而DOS窗口下解析的码表为GBK字符集，且DOS窗口下的编码表是不可修改的，那么如何才能在DOS窗口下显示中文？

1. 临时修改码表为utf8，适合长期使用图形界面用户

set names = utf8； // 临时修改一旦退出DOS窗口，码表就会恢复原状

1. 直接修改MySQL配置文件，一次修改，一劳永逸

找到MySQL的安装目录（使用 show variables like ‘%character%’; 代码可以查到，最后一行）

在安装目录下找到my.ini 文件，修改其中default-character-set=gbk即可

### 超级管理员密码重置

停止mysql服务器：运行输入services.msc，停止mysql服务

在cmd下输入mysqld --skip-grant-tables 启动服务器 光标不动 （不要关闭该窗口）

新打开cmd 输入mysql -u root -p 不需要密码

use mysql;

update user set password=password('abc') WHERE User='root';

关闭两个cmd窗口 在任务管理器结束mysqld 进程

在服务管理页面 重启mysql 服务。

## 数据库的操作

### 切换库和查看正在使用的库

切换当前使用的数据库：

use 数据库名;

查看当前正在使用的数据库

select database();

### 数据库的增删改查

创建数据库：

create database 数据库名;

create database if not exists 数据库名;

create database 数据库名 character set 字符集 collate 比较规则;

// 字符集默认utf8，可以不写，比较规则可以不写

使用if not exists关键字，如果当前数据库名已存在的情况下，则不创建数据库，如果不使用该关键字，返回该数据库已创建----推荐使用第一个

删除数据库：

drop database 数据库名;

简单粗暴，直接删除就好

修改数据库：

**alter database 数据库名称** character set 字符集 collate 比较规则

// 字符集默认utf8，可以不写，比较规则可以不写

查询数据库：

select version()； // 查看当前数据库版本信息

show databases； // 查看已创建的数据库

show create database 数据库名； // 查看当前数据库相关信息，例如字符集，数据库版本号等

show databases; // 显示所有数据库

### 数据库的备份和恢复

在MySQL的安装目录下的bin目录中，有个mysqldump命令，可以对数据库进行备份操作

**备份操作：**在CMD命令提示符下（不需要进入数据库）：

mysqldump -u 用户名 -p 数据库名 > 存储文件的路径.mysql

完成操作后输入密码，他就会自动在指定目录下生成一个sql文件

右键以文本方式打开sql文件，发现数据库的备份实际上就是将数据库以SQL命令的方式将文件导出，里面是反复的建表、插入数据的操作

**恢复操作：**恢复数据库的方法有很多种，标准的方法是在CMD命令提示符下操作（不需要进入数据库）

mysql –u 用户名 –p 数据库名 < sql文件目录

可以在数据库内执行命令：

source 备份文件路径名.sql

或者直接右键以记事本打开sql文件，复制其中所有的代码，在数据库内粘贴直接运行，也是可行的

还有一种方法是图形化界面，直接鼠标操作将文件引入

有一点需要注意！

SQL文件其实就是建表插入数据的过程，但是其中并不包括创建数据库的过程，所以在恢复数据库之前，都要手动创建一个库来装他

## 数据表的增删改查

**什么是数据表：**

表是数据库中最基本的逻辑单元，由行和列组成，用户保存在数据库中的基本数据库都是由表的形式存储

E-R（Entity-Relationship）模型即实体-关系模型主要用于定义数据的存储需求。E-R模型由实体、属性和关系三个基本要素构成。

**数据表的组成：**

1. 表名：每张表都有自己的名称，被称为表名
2. 列名：表头的每一个单元格中的信息，被称为当前列的列名
3. 记录：表中的每一行数据，被称为记录，每行中的每个单元格自身的数据是有类型的

所以在创建表的时候需要制定 表名，列名，每列（单元格）的数据类型

### 数据表的增删改查

添加表：

create table 表名( 列名 数据类型(长度) ) ;

删除表

drop table 表名;

修改表

rename table 原名 to 改名; // 修改表名

alter table 表名 character set 编码表名

查询表：

desc 表名; // 查询表的相关信息描述

show tables； // 显示当前库中的所有表

show create table 表名; // 查看建表语句和字符集

复制表：

create table 新表表名 like 旧表表名; // 可以将源表表格式完全复制（包括主键），生成一个新的空表

creatr table 新表表名 select \* from 源表 // 可以将源表表格式完全复制（不包括主键）+数据复制到新表中

### 表内列的增删改查

添加列：

alter table 表名 add 列名 类型（长度）;

删除列：

alter table 表名 drop 列名;

修改列：

-- 修改列名( 改名时类型和长度必需书写，可以照着源类型照敲 )

alter table 表名 change 原列名 新列名 类型（长度） 约束

-- 修改类型

alter table 表名 modify 列名 类型（长度） 约束;

查询列：

select 列名 from 表名 // 具体查询之后会讲

**所有库和表结构操作语句都是DDL（定义语言）**

### 数据库中的数据类型

1. 字符型：
   1. char（长度）； 定长，指定一个长度，存储的数据长度小于这个长度，后面空着的长度也会占据
   2. varchar（长度）； 变长，指定一个长度，存储的数据长度小于这个长度，那么长度会自动缩小，推荐使用
2. 数值型：
   1. tinyint： 等价于java中的byte
   2. smallint： 等价于java中的short
   3. int ： 等价于java中的int
   4. bigint： 等价于java中的long
   5. float： 等价于java中的float
   6. double： 等价于java中的double
3. 日期型
   1. date： 只有日期（2000-1-1）
   2. time： 只有时间（20:30:55）
   3. datetime： 日期时间都有，必须按照格式书写，不写则报错
   4. timestamp： 日期时间都有，可以留空不写，数据库自动将当前时间放在单元格中

4． 选择型：

a) enum（’’,’’）：枚举类型，在里面嵌入一些值可以进行单项选择

b) set（’’,’’）： 集合型，在里面嵌入一些值，可以进行多项选择

**default：设置属性的默认值，如果没有插入该数据则为默认值**

### constraint约束

#### 约束的分类

1. **primary key：（主键约束）**

记录每一行的当前列都是区分其他行的信息记录，这个列就是主键，要求每个表必须有一个主键信息（不写也不报错），因为要区分其他行，所以主键约束后会附带有非空约束和唯一约束

1. **foreign key（外键约束）**

在列及引用列上建议的一种强制性依赖关系

foreign key(当前表的列) references 目标表(目标表中的列)

1. **not null：（非空约束）**

约束着每一行的当前列中的数据不能为null/空

1. **unique：（唯一性约束）**

每一行的该列内读数据都是唯一的，不允许重复

1. **auto\_increment（自增约束）**

自增长约束，一般配合主键使用，每插入一条数据，该列上的数值就+1

自增长要求本身就是索引且只能为整数

1. **check（检查性约束）**

列上指定一个必须满足的条件

#### 约束的操作

1. 创建表时添加约束

1.1 添加列级约束：只需要在该列后面的，之前直接将约束写在上面即可

列级约束可以添加：【主键】【唯一】【非空】【默认】，不支持外键和检查

-- 创建用户表

CREATE TABLE t\_user(

id INT(2) PRIMARY KEY AUTO\_INCREMENT, -- 主键约束

name VARCHAR(5) NOT NULL UNIQUE, -- 非空约束，唯一约束

sex CHAR(1) DEFAULT ‘男’ -- 默认值

)

1.2 **添加表级约束：**表级约束是在字段全部写完之后在尾部添加的，格式是：

**[ constraint 约束名 ] 约束(被约束的列) [ references 被引用的表(列) ]**

CREATE TABLE t\_tel(

id INT(2),

number VARCHAR(5),

CONSTRAINT pk PRIMARY KEY(id),

CONSTRAINT fk FOREIGN KEY(id) REFERENCES t\_user(id)

)

2. 某一列添加约束

2.1 列级别添加约束

**alter table 表名 add 约束名(列名)**

ALTER TABLE t\_tel ADD UNIQUE(number);

2.2 表级别添加约束

**alter table 表名 modify column 字段名 字段类型（长度）约束类型**

ALTER TABLE t\_tel MODIFY COLUMN id INT(2) PRIMARY KEY;

## 数据增删改查&运算符介绍

### 表内数据的增删改查

**insert，update，delete这种对表内的数据进行操作的都属于EML操作语言**

添加数据：

insert into 表名(列名，列名…)values(值，值…) //按照列名添加数据，适合插入当前行单列数据

insert into 表名 value(值，值…) // 按照该行默认列的顺序添加输入，适合插入当前行全部列数据

insert into 表名() values（值，值…），（值，值…）… // 多行添加数据

1. 添加数据必须具备的几个条件：
   1. 添加的数据必须和当前行的列类型相同
   2. 添加数据的长度不能大于定义长度
   3. 添加空值可以填 null
   4. 添加数据时遇到自增的值也可以填入null，他会自动填入自增的值
   5. 添加的数据必须和列顺序对应（没有指定顺序则按默认顺序）
   6. 日期和字符串类型数据需要单引号引起来

删除数据：

delete form 表名 where 删除条件; // 如果不写删除条件，则默认把所有行都删掉，容易挨揍

truncate table 表名 ; // 删除当前表内所有数据

如果删除某个表的全部数据，建议使用truncate，二者的区别为：

1. delete是逐行删除的，而truncate是将当前表整张删除，然后在创建一个新的表，在性能上truncate好于delete
2. delete是删除数据，属于DML（操作语言），而truncate是删除表后创建表，属于DDL（定义语言），DML操作语言可以回滚到执行前状态

修改数据：

update 表名 set 属性=值 , 属性=值… where 条件…; // 如果不添加条件，那么他默认把该列的所有行数据都修改掉

查询数据：

select 列名 from 表名; // 查询当前表指定列的信息

select \* from 表名 // 查询当前表所有列的信息

select distinct

列名 AS 别名，列名 AS 别名

from 表名

order by 列名 desc

where 逻辑表达式; // 下面解释

注意事项：

**1. distinct**是去重关键字，可有可无，如果添加distinct关键字的话，查询到多行重复数据的话只会显示一行，并且如果查询多行去重的话，必须要每行完全重复才会去掉

**2. AS**表示在查询指定的列后面跟上别名，查询的时候表头可以显示别名

**3. order by**表示排序，在order by后面跟上指定的列，可以按照指定的列进行升序或者降序排序，**asc为升序，desc为降序**

**4. where**表示条件，后面跟上逻辑表达式，返回的是true就是满足条件，false就是不满足条件

### SQL中的运算符

1. 基本运算符：

【大于 >】【小于 <】【等于 =】【大于等于 >=】【小于等于 <=】【不等于 <>】【不等于 !=】

建议使用 <> 而不是使用 != ，因为 != 并不是SQL语言中的写法，部分数据库不支持

1. 逻辑运算符：

【与 and】【与 &&】【或 or】【或 | |】【非 not】【非 ! 】

建议使用英文形式的与或非，因为符号类型不属于SQL语言，部分数据库不支持

1. 特殊运算符：
   1. 是否为空：

【is null】【is not null】

* 1. 包含以下任意一个值：

in(值，值，值)

在in的括号中不仅可以书写特定的值，也可以书写一整列，作为参数传进去，被称为子表查询

* 1. 满足在指定的区间内：

between 最小值 and 最大值;

* 1. 模糊查询（类似正则表达式）：

like ‘\_张%’ **// 表示第二个字符为‘张’的所有字符串**

% 表示任意个数的字符

\_ 占位符，表示任意单个字符

* 1. 分页查询

limit 起始位置，查询行数

## 常用函数

### 数学函数

abs（x）; 返回x的绝对值

PI（）; 返回圆周率

DOM（x，y） 返回x被y除的余数，可以用 % 代替

round（x，y）; x为小数，返回一个四舍五入的近似值，保留小数点后y为

truncate（x，y）; x为小数，保留小数点后y为，直接删除不会进行取舍

sign（x）; 返回参数的符号

rand; 随机数

### 字符串函数

char\_length(str) 返回str所包含的字符个数

length(str) 返回字符串str的长度（字节）

concat（s1,s2…） 连接多个字符串

concat\_ws(x,s1,s2…) 连接多个字符串，每个字符串之间用x作为分隔符

substring(s,n,len) 截取字符串s从n开始的len个字符

lower(str)或lcase(str) str内所有字母转换为小写

upper(str)或ucase(set) str内所有字母转换为大写

left(str,n) 返回str最左面n长度的字符

right(str,n) 返回str最右面n长度的字符

reverse(str) 将str字符串翻转

locate(s1,s2) 返回s2在s1中首次出现的下标

### 日期函数

【 年份：%Y 】—【 月份: %m 】—【 日期：%d 】—【 小时：%H 】—【 分钟：%i 】—【 秒钟：%s 】

【 年：year 】— 【 月：month】—【 日：day 】

curdate(); 获取当前日期 YYYY-mm-dd

curtime(); 获取当前时间 HH:ii:ss

now(); 获取当前日期+时间 %Y-%m-%d %H:%i:%s

date\_format(time,format); 将日期时间以默认显示格式转换为format格式显示（默认%Y-%m-%d %H:%i:%s）

time\_format(time,format) 将时间以默认显示转换为format格式显示（默认%H:%i:%s）

str\_to\_date(str,format); 将字符串格式的日期通过相同format转换为date类型默认格式

timestampdiff(month,s1,s2); 计算两个字符串类型日期的时间差，自定义返回类型

如果系统获取日期出现乱码请修改码表： set lc\_time\_names = ‘zh\_CN’;

set names = gbk;

### 流程控制函数

select case 条件1 when 值1 then ‘一’

when 值2 then ‘二’else ‘三’end;

case函数开始，如果条件1满足值1，返回‘一’，或者满足值2，返回‘二’，都不满足则返回‘三’end结束

select case when 1>0 then ‘结果1’ else ‘结果2’end;

1>0 结果为true，则返回‘结果1’，否则返回‘结果2’

if(expr1，expr2，expr3)

expr1 返回true，执行expr2，否则执行expr3

## 分组查询

|  |  |  |  |
| --- | --- | --- | --- |
| **Table student** | | | |
| **id** | name | sex | fen |
| **01** | 曾庆龙 | 男 | 87 |
| **02** | 崔英楠 | 男 | 89 |
| **03** | 崔玉莹 | 女 | 69 |
| **04** | 房鹤年 | 男 | 72 |
| **05** | 高吉娟 | 女 | 91 |

**分组函数** 又被成为聚合函数/聚集函数：

【max（列名）；最大值】，【min（列名）；最小值】，【sum（列名）；取和值】，【avg（列名）；平均值】【count（列名）；计数器】

// 如果某行参与运算的之有null值，那么该行并不参与运算

// 这种求最大值最小值等等的函数都是针对一整列进行运算的，因此他不能作为筛选条件放在where关键字后面

**group by关键字** 分组查询

// 按照性别进行分组，查询男生女生分别多少人

select sex , count(\*) from student group by sex;

**having关键字** 过滤条件

// 查询班级内男生分数大于男生平均分的人的所有信息

select \* from student having fen > avg(fen)

**having关键字和where关键字的相同点和不同点：**

1. 二者的作用相同，都是起到筛选过滤的作用
2. 相对于group by来说，二者的执行顺序不同，where是先进行过滤，在执行分组，而having是先执行分组，在执行过滤
3. having后面可以跟分组函数，而where后面不可以跟分组函数
4. 使用where的地方都可以用having代替，而使用having的地方未必可以用where替换

**查询语句的书写顺序：**

select → from → where → group by → having → order by

查询 → 选择表 → 条件 → 分组 → 分组条件 → 排序

**查询语句的解析顺序：**

from → where → group by → having → select → order by

选择表 → 过滤 → 分组 → 分组过滤 → 查询 → 排序

## 多表连接查询

用户信息-手机卡号信息，app应用信息三张表



当我们要查询的数据不在同一张表的时候，就需要用到多表查询，也叫连接查询

多表连接查询的方法有很多种，他们分为：

* 内连接
  + 等值连接
  + 非等值连接
  + 自连接
* 外连接
  + 左外连接
  + 右外连接
  + 全外连接
* 交叉连接

按照年代来分类，有两种分类：

1. SQL 92标准，仅支持内连接（也支持一点外连接）

2. SQL 99标准，之前内外链接及全连接（其中全外连接除外）

### 内连接

#### 1. 等值连接

将两张表连接在一起，通过添加等值条件对查询到的数据进行过滤

1. 如果两张表有相同的字段名需要用【表明.字段名】进行区分

2. 如果表明太长，可以在from后面为表明起别名，表明与别名之前用空格隔开

**直接连接查询测试：**

例如：查询每个用户的基本信息及注册的手机号

书写SQL：

SELECT u.u\_id , u.u\_name , u.u\_sex , t.t\_number

FROM t\_user u , t\_tel t

根据上面查询的结果来看，查询的结果是每个用户都注册了每一个手机号，产生了**笛卡尔积**

笛卡尔积：全程叫做笛卡尔乘积现象，在MySQL中可以理解为第一张表中的每一行都同其他表的每一行查询产生结果

针对笛卡尔积解决的办法：在查询的时候添加一个等值条件做为限制：

SELECT u.u\_id , u.u\_name , u.u\_sex , t.t\_number

FROM t\_user u , t\_tel t

WHERE u.u\_id = t.u\_id

只有在查询的用户的id与手机号对应的用户id相同时，证明该用户持有该手机号，才会产生查询结果

#### 2. 非等职连接

非等职连接

#### 3. 自连接

**内连接查询：**等价于关联查询

select \* from 表1 inner join 表2 on 条件

// on代表where，一般选择用关联查询替代

**左外连接查询：**无论右侧的表如何，左侧的都必须查询

select \* from 表1 left outer join 表2 on 条件

**右外连接查询：**无论左侧的表如何，右侧的都必须查询

select \* from 表1 right outer join 表2 on 条件

**全连接查询：**将左外连接和右外连接的查询结果合并到一起

select \* from 表1 full outer join 表2 on 条件

// 上面就是全连接的代码演示，但是mysql并不支持全连接

// 不支持并不代表就不能用，可以使用下面的代码进行操作：

select \* from 表1 left outer join 表2 on 条件

union all

select \* from 表1 right outer join 表2 on 条件

// 使用union all关键字将左外连接和右外连接的结果放到了一起，实现了全连接的意义，但是这种做法是将两种连接的结果拼到了一起，会产生重复数据，推荐使用下面的写法：

select \* from 表1 left outer join 表2 on 条件

union

select \* from 表1 right outer join 表2 on 条件

**子查询：**将一段select查询出来的结果当成表进行二次使用

**in关键字** // student表中id和 teacher中任意一个匹配上，就可以使用

select \* from student where id in (select id from teacher where name <> ‘孟范立’)

**exists关键字** // 直接这样写，两张表之间没有建立连接，只有where左面的表有数据，返回的就是true，打印的都是主表

select \* from student where id in (select id from teacher where name <> ‘孟范立’)

// 想要有效查询就要使两张表之间建立一个条件

select \* from student where id in (select id from teacher where name <> ‘孟范立’**where teacher.id = student.id**)

**多行子查询：**表示和子表中的所有行进行比较

**any关键字** // 找到比person表中最小的age还要小的age

select \* from student where age < any(select \* from person)

**all关键字** // 找到比person表中最大的age还要大的age

select \* from student where age > all(select age from person)

## 数据库事务

数据库的事务

1. 主要是用来解决针对并发操作数据库时会出现的一些问题。
2. 数据库的事务是由一条或多条SQL语句组成的代码块，又被称为工作单元，他们要么全部执行成功，要么全部执行失败，不允许发生成功一部分，失败一部分的情况发生

### 事务的特性

支持数据库事务的操作，要具备四个特性：

1. **原子性（Atomicity）**
   1. 事务就像“原子”一样，不可被分割，组成事务的DML操作语句要么全成功，要么全失败，不可能出现部分成功部分失败的情况。
2. **一致性（Consistency）**
   1. 一旦事务完成，不管是成功的，还是失败的，整个系统处于数据一致的状态。
   2. 某张表的某行被操作后，和他有关系的所有其他的表也会被操作
3. **隔离性（Isolation）**
   1. 一个事务的执行不会被另一个事务所干扰，同时有两个事务，第二个需要等待第一个执行完成后在执行自己，或者在第一个执行之前执行
4. **持久性（Durability）**
   1. 也称为永久性，指事务一旦提交，对数据的改变就是永久的，不可以再被回滚。

### 事务隔离级别

事务的隔离级别：

* 【脏读】：用户可以读取到还没有确定是否执行成功的数据信息
* 【不可重复读】：指用户可以读取到已经处理并确认提交的数据信息
* 【虚读/幻读】：a修改当前行信息，而后b插入了一条和a修改前一模一样的信息，a以为修改失败（已经成功）

他们分别对应着：

* Serializable（串行化）
  + 最严格的隔离级别，可避免脏读，不可重复读，幻读
* 虚读/幻读 → Repeatable read（可重复读）
  + 可避免不可重复读，脏读
* 不可重复读 → Read committed（读已提交）
  + 可避免脏读
* 脏读 → Read uncommitted（读未提交）
  + 最低级别隔离，任何情况都无法避免

MySQL中查看当前事务隔离级别：

select @@tx\_isolation

自动提交设置：

书写SQL代码的时候，向数据库表中插入一条记录insert，随后去查看数据表，发现数据已经添加成功，说明数据是提交，如果执行SQL语句后发现数据没有插入进去，需要手动提交，那么就是自动提交关闭状态，一般来说数据库都有设置自动提交

查看当前是否为自动提交：

show variables like 'autocommit';

修改自动提交属性：

set autocommit=off/on;

no为自动提交，off为关闭自动提交

修改自动提交属性还有：

set autocommit = 0/1 ;

1是自动提交，0是手动提交

**事务控制关键字：**

begin 开始

savepoint name 定义一个保存点

rollback to name 回滚到指定保存点

rollback 回滚到开始（begin处）

commit 提交事务

通常使用commit的时候就代表事务结束了

在执行begin关键字的时候系统自动将提交方式修改为手动提交，当执行到commit关键字的时候意味着该事务已经执行结束，系统在将自动提交修改至之前的设置

代码演示：

begin; -- 表示开始进行事务

insert into t\_user values(null,'克洛克达尔',SYSDATE(),'男');

SAVEPOINT p1; -- 设置一个保存点，p1为保存点的名称

insert into t\_user values(null,'月光莫里亚',SYSDATE(),'男');

ROLLBACK to p1; -- 回滚至指定的保存点，如果不写to p1，则直接回滚至begin

commit; -- 提交，表示当前事务的结束

细节：回滚SQL语句时如果有insert语句中包含auto\_increment自增的时候，自增的值是不会回滚的

## 视图&索引技术

### MySQL视图

什么是视图：

View视图就是一张虚拟的表，他是MySQL 5.1版本后出现的新特性，通过真实存在的表动态生成的数据。

如果我们进行复杂查询的时候，每次都需要查询相同的几张表，这个时候就可以把他们封装为视图，进而对视图进行操作，省时省力

视图的特点：

1. 可选择型的显示数据的一部分，起到保护数据的作用

2. 对SQL语句的重用，平时需要查询多张表的sql现在只需要一个视图就可以解决

3. 同时也简化了SQL的复杂操作

视图的创建基于普通表，创建视图的语法：

create **view** 视图名

-- 创建一个视图，视图的数据参考查询的结果

AS select 列名 from 表名

修改当前视图数据

-- 1. 修改当前视图内的数据为查询的结果

ALTER VIEW myv1

AS

SELECT \* FROM t\_user

-- 2. 如果当前视图已存在就修改（覆盖），不存在就创建

CREATE OR REPLACE VIEW myv1

AS

SELECT \* FROM t\_app;

查看视图的结构

DESC myv1;

SHOW CREATE VIEW myv1;

SHOW CREATE VIEW myv1\G; -- 命令行下执行

删除视图：

drop view 视图名;

-- 支持批量删除

drop view 视图1，视图2，视图3

# 在视图（简单视图）中进行DML操作可以更改源表，只要视图格式和源表一致（没有去重分组之类的）

### MySQL索引

#### 索引概述

索引：可以快速的查询数据表中的特定记录，是提高数据库性能的重要方式，MySQL中所有数据类型都可以被索引

索引的优点是提高检索数据的速度

索引的缺点是创建和维护过于耗费时间

**索引分类**

1. 普通索引 index 索引名（列名）
2. **惟一性索引 unique index**
3. **全文索引 fulltext index**
   1. **只能创建在char，varchar或text类型字段上**
   2. **只有MyISAM存储引擎中支持（MySQL例外）**
4. 单列索引 index 索引名（列名（长度））
5. 多列索引 index 索引名（列名，列名）
   1. 只有使用了索引中的第一个字段时才会触发索引
   2. 如果没有使用第一段索引，那么多列就不会起作用
6. 空间索引 spatial index 索引名（列名）
   1. 只能在myisam存储引擎中使用
   2. 索引字段必须有非空约束
   3. 空间数据类型包括：
      * 1. geometry
        2. point
        3. linestring
        4. polygon（平时很少用到）

#### 创建索引的三种方式：

在创建表的同时创建索引：

create table 表名（

列名 数据类型 ，

列名 数据类型（长度），

unique index 索引名（列名 asc） - - 唯一索引

ASC为排序，可升序可降序，选择性书写，不写也不报错

）engine=myisam; //修改搜索引擎，只有全文索引需要修改

在已存在的表中创建索引：

create index 索引名on 表名 （列名 属性名（长度））

使用alter table语句来创建索引：

alter table 表名 add index 索引名 （列名 属性名（长度））

删除索引

drop index 索引名 on 表名

索引的设计规则：

1. 索引的设计规则
2. 选择唯一性的索引
3. 尽量选择数据量少的作为索引
4. 为常作为查询条件的字段建立索引
5. 为经常需要排序、分组和联合操作的字段建立索引
6. 限制索引的数目
7. 删除不再使用或者很少使用的索引
8. 尽量使用前缀来索引

## 用户和权限管理

创建新用户：

create user ‘用户名’@‘localhost’ identified by‘密码’

grant 权限1，权限2…数据库名 表名 to 用户名@用户地址

# MySQL高级

## MySQL的变量

什么是变量：MySQL中有变量的存在，在MySQL中有两种类型的变量

1. 会话变量（用户变量，用户自定义的变量）

2. 系统变量（全局变量）

**1、定义会话变量**

-- 定义

set @name = ‘hello world’;

-- 因为MySQL中没有比较符号，就用赋值号代替，但是有时候会用混，所以MySQL有个特定的赋值号

set @name **:**= ‘hello world’;

-- 可以将从表中查询出的值付给变量，但是必须是某一行中的一个字段，因为MySQL中不支持数组

1. SET @message = (SELECT msg FROM abc WHERE id = 2)

2. SELECT @message := msg , @id := id FROM abc WHERE id = 1;

**2、修改会话变量**

针对变量的修改有两种类型：

1. 修改会话变量（只在当前会话有效，重新登陆的时候会失效）

set 变量名 = 值;

2. 修改全局变量

1. set global 变量名 = 值

2. set @@global.变量名 = 值

修改变量需要注意：

1. 修改并不是实时生效的，例如xx为0，你修改为1，但是查询是却还是0，这个时候只要重新登陆再次查询就可以更新成功了

2. 如果修改结束后要立即使用的话需要使用局部修改，否则更新不及时

**3、查询变量**

-- 查询系统变量（其中like为可选的查询参数）

show variables [like '']

-- 查询允许用户使用的变量（全局变量）的值，可以一次查询多个

select @@变量名

4、定义局部变量

declare 变量名 类型（长度）[ default 值 ]

## 流程控制语句

if判断语句：

if（表达式1，表达式2，表达式3）

当表达式1结果为true时执行表达式2，否则执行表达式3

-- 判断id是否小于10，并输出结果

SELECT \* , if(userid < 10 , 'id小于10' , 'id大于10') AS '判断' FROM t\_user

while do循环：

while 表达式 do

循环体

end while

## 函数

什么是函数：用于实现某种功能的语句块

什么是自定义函数：由用户创建，用来实现某种功能的语句块

需要注意：

1. 函数是一个整体，在定义函数的时候不能中断

2. 函数体内每条指令都是一个独立的个体，需要使用 ；作为语句结束的分隔符

3. 一旦遇到；后语句就会自动执行，所以在创建自定义函数之前需要修改语句结束符号delimiter ;

3. 如果函数体本身只有一段语句，那么就可以省略掉begin和end不写

4. 创建的函数只能在该库中使用

**创建自定义函数：**

-- 定义函数

delimiter $$

create function 方法名（形参） returns 返回值类型

begin -- 函数起始符

函数体;

end$$ -- 函数结束符

-- 定义最简函数

create function 方法名（形参） returns 返回值类型

return 返回值；

-- 创建带有参数的函数

删除自定义函数

drop function 函数名;

查询自定义函数

-- 一般查询

show function status[\G] [ like ‘’ ] -- \G只能在命令行中运行

-- 查询函数创建语句

show create function 函数名[\G] -- \G只能在命令行中运行

## 存储过程

存储过程（Stored Procedure）是一组为了完成特定功能的SQL语句集，是MySQL中一个重要的对象，在首次执行编译后再次执行无需编译

在创建存储过程时需要注意的几个点：

1. 在创建存储过程的时候不能中断（修改结束标识符）

2. 如果存储过程本身只有一句语句，那么可以省略begin end不写

3. 在存储过程中可以定义局部变量，并可以通过查询表的方式给局部变量赋值

4. 当传入的参数于表中字段相同时应用表明 . 来区分变量和字段

5. 调用过程的关键字为【**call 过程名（）；】**

6. 删除某个存储过程的关键字为【**drop procedure 过程名；**】

**1. 创建一般存储过程：**

-- 同创建函数一样，需要修改结束标识符

delimiter $$

create procedure 过程名()

Begin

过程代码

end $$

传入参数时是可以指定类型的，例如in，代表这参数仅支持传入，存储过程的参数共有三种类型：

1. IN：可以做为参数传入

2. OUT：可以做为返回值

3. INOUT：既可以做为参数也可以做为返回值

**2. 有参存储过程 IN**

-- 创建过程，要求实现输入名字，查询到的名字及对应的年龄，且过程要用到局部变量

DELIMITER $$

CREATE PROCEDURE pro5( IN uname VARCHAR(10) )

BEGIN

-- **创建局部变量**用来存储年龄

DECLARE uage INT(2);

-- 一次查询**为变量赋值**

SELECT tab1.`uage` INTO uage

FROM tab1

WHERE tab1.`uname` = uname;

-- 二次查询将结果打印

SELECT uname , uage;

END $$

DELIMITER ;

**3. 有参存储过程 OUT**

DELIMITER $

CREATE PROCEDURE pro2( IN uid INT(2) , OUT uname VARCHAR(10) , OUT uage INT(2) )

BEGIN

-- 这里可以批量的进行变量赋值

SELECT t.uname , t.uage INTO uname , uage

FROM tab1 t

WHERE t.uid = uid;

END $

DELIMITER ;

调用过程时：

-- 定义会话变量，用来接受返回的值

SET @abc := '';

SET @bcd := 0;

CALL pro2(3,@abc,@bcd);

SELECT @abc AS '姓名' , @bcd AS '年龄';

**4. 有参存储过程 INOUT**

-- 传入参数，返回该参数的两倍的值

CREATE PROCEDURE pro3( INOUT a INT(3) )

SET a = a \* 2 ;

-- 创建会话变量，操作存储过程，查询结果

SET @abc := 22;

CALL pro3( @abc );

SELECT @abc;

## event计时器-事件

MySQL中提供了一种可以精确到秒级的定时任务，event，其原理是系统定时驱动执行，一般我们称event为Job定时任务，在MySQL中想要使用Job，需要开启该功能【set global event\_scheduler = 1;】1=on，0=off

event中的时间单位：

【年：year】、【季度：quarter】、【月：month】、【星期：week】、【日：day】、【时：hour】、【分：minute】、【秒：second】

创建一个计时器：

-- 可以在事件名之前加上if not exists

-- 如果该事件已存在就不在创建

CREATE EVENT 事件名

-- 计划任务时间，AT为指定时间后执行，every为每搁固定时间执行一次

ON SCHEDULE AT 时间 单位

-- 事件执行完毕是否保存，不写not为保存

ON COMPLETION NOT PRESERVE

-- do关键字后面跟上需要定时执行的sql代码

-- 可以跟函数，存储过程等等

-- 如果是多条SQL需要用begin-end关键字，并修改结束符

DO SQL代码;

创建并使用event定时任务需要注意系统的event事件开关是否开启，OFF关闭，ON为开启

1. 查看是否开启event计时器【select @@event\_scheduler;】

2. 将计时器修改为开启状态【set global event\_scheduler = 1;】1=on，0=off

删除/停用计时器

drop event 事件名;

查看正在运行的计时器

select \* from mysql.event;

show events\G -- 详细查询，只能在命令行中运行