MySql

# 数据库概念

### 什么是数据库

数据库（database）是按照数据结构组织、存储和管理数据的仓库，简而言之就是存储数据的仓库

### 数据库分类

发展历程：层次式数据库，网络式数据库，关系型数据库

关系型数据库：利用表来存储数据，利用表和表之间的关系保存数据之间的关系

### 常见的关系型数据库

SQLserver

mysql

oracle

…

任何关系型数据库的语法都是sql

### mysql介绍

mysql是一个关系型数据库，由瑞典的mysql AB 公司开发，目前属于oracle旗下

特点是体积小，速度快，成本低，开源

### mysql安装配置

详见安装文档

### mysql常见概念

数据库：一个mysql服务器中可以有多个库，一个库就是表的集合

表：类似于一个excel表格，用于存储数据

表记录：表中的一行数据

# SQL分类



### DQL

### DML

### DDL

### DCL

### TCL

# Mysql数据库基本命令

### 登录

#### mysql –u 用户名 –P密码 –h ip

#### 例：mysql –uroot –p1234 –h 127.0.0.1

#### 注意：

##### 红色部分不可省略，灰色部分”密码 –h ip” 可省略，如果是本机，ip可以省略

##### –u空格用户名 –h空格ip中间的空格可以省略也可不省略

##### -p后面可以不写密码，换行继续写，\*安全

### 退出

##### quit;

##### exit;

##### 快捷键Ctrl+C

### 启动/关闭服务

##### net stop mysql

##### net start mysql

##### mysql为服务名

# SQL语句

|  |  |
| --- | --- |
| **lamp** | **注释：**除了 SQL 标准之外，大部分 SQL 数据库程序都拥有它们自己的专有扩展！ |

### 注意

#### 结束有英文分号

#### Query OK, 1 row affected (0.00 sec)：操作成功，一行被改变

#### 不区分大小写(命令不区分，数据库名区分,字段等)

### 库的CRUD

##### Create ：create database db\_name

##### Read :show databases;

##### Update :alter database db\_name character set gbk;

##### Delete:***drop database*** db\_name;

### 表的CRUD

### 表记录的CRUD

### 数据类型

#### 字符串

##### char(n) 定长字符串

n表示字符串的最大长度不能超过这个值，n最大为255

存储时分配固定长度的空间，一般用于存储固定长度的字符串，如手机号、身份证号等

效率高，相对空间浪费

Sqlser中多余空间空格补充，而mysql中不会

##### varchar(n) 不定长字符串

n最大为65535，n只是规定最长字符串

存储时，先计算字符串的大小，然后再分配对应长度的空间，一般存储不固定长度的字符，如姓名、昵称等，节省空间

#### 数值类型

|  |  |
| --- | --- |
| **数据库** | **Java(类似)** |
| tinyint | byte |
| smallint | short |
| int | int |
| bigint | long |
| float | float |
| double | double |

#### 大数据类型

##### blob：大二进制类型，可以存储二进制数据，如图片、音频、视频，最大4gb

##### text：大文本类型，可以存储大量字符数据，最大5gb

text为mysql方言，其他数据库中为clob

#### 日期类型

##### date

日期：年-月-日

日期 2008-08-08

##### time

时间：时：分：秒

时间 20:08:00

##### datetime

年-月-日 时：分：秒

2008-08-08 20:08:00

##### timestamp

存储的时间戳，就是java中 Date类的getTime()的值

1997-1-1到当前的毫秒值

#### 逻辑型

##### bit

只能取0和1

#### 注意：操作时字符型和日期需要单引号，数值型不需要

# 约束条件

都可建表时放到 数据类型后边

#### 主键约束：

##### 所约束的值必须唯一且非NULL，不能重复

##### 建表时： 字段名1 数据类型 primary key

##### 表已存在添加主键:

alter table table\_name add primary key(field);

##### 自动增长字段: 字段名1 数据类型 primary key anto\_increment

只能主键或者有唯一索引的键才能设置 auto\_increment，***一个表只能由一个自增字段***

#### 唯一约束(唯一索引？？)

##### 字段名1 数据类型 unique

只能主键或者有唯一索引的键才能设置 ，可以为NULL， auto\_increment***一个表只能由一个自增字段***

#### 非空约束

##### 字段名1 数据类型 ***not null***

#### 外键约束

foreign KEY(ID) reference 表名(ID)

# 数据库操作

### 创建数据库

#### 语法：create database [if not exists] 数据库名 [character set charset\_name]

##### If not exists可以防止报错，数据库不存在时才创建

##### create database db\_book 创建db数据库

##### character set charset\_name设置编码

### 查看数据库

#### show databases

##### 查看所有数据库，结尾s

mysql> show databases;

+--------------------+

| Database |

+--------------------+

| information\_schema |

| accountbook |

| mydb1 |

| mysql |

| performance\_schema |

| test |

+--------------------+

6 rows in set (0.00 sec)

#### show create database db\_name

##### 查看数据库的创建语句

mysql> show create database mydb1;

+----------+----------------------------------------------------------------+

| Database | Create Database |

+----------+----------------------------------------------------------------+

| mydb1 | CREATE DATABASE `mydb1` /\*!40100 DEFAULT CHARACTER SET utf8 \*/ |

+----------+----------------------------------------------------------------+

1 row in set (0.00 sec)

#### 查看当前使用的数据库

##### select database(); 注意括号

### 修改数据库

#### 数据库的名称无法修改，可修改数据库编码

#### 格式： Alter database 数据库名 character set 编码名称;

#### 例： Alter database mydb1 character set utf8;

### 删除数据库

#### 格式：drop database [if exists] db\_name;

### 选择数据库

#### 格式：use db\_name; 这里不加分号也行

# 表操作

### 创建表

#### 语法：

create table table\_name(

字段名1 数据类型,

字段名1 数据类型,

…

字段名n 数据类型//最后一个字段末尾没有逗号

);//注意结尾有分号

#### 例：

create table employee(

id int,

name varchar(10),

gender varchar(1),

birthday date,

entry\_date date,

job varchar(30),

salary double

resume text

)

#### 注意

最后一个字段末尾没有逗号

### 查看表

#### 查看所有表

show tabes;

#### 查看表结构

desc table\_name;

#### 查看建表语句

show create table table\_name

### 修改表

#### 增加列

##### alter table 表名 add 列名 数据类型;

#### 修改列(modify修改数据类型)

##### alter table 表名 modify 列名 数据类型

alter table employee modify id int primary key auto\_increment;

设置id字段为主键和字段增长

#### 删除列

##### alter table 表名 drop 列名

#### 修改列名

##### alter table 表名 change 旧列名 新列名 数据类型

#### 修改表名

##### alter table 表名 rename to 新表名

##### reanme table 旧表名 to 新表名

#### 修改列顺序

##### alter table 表名 modify 列名1 数据类型 after 列名2

将列名1放到列名2后 ,before不行

#### 修改表字符集

##### alter table 表名 character set 编码



### 删除表

##### drop table 表名

# 记录操作

### 插入记录(insert)

#### 语法：insert into 表名[(列名1，列名2，列名3…)] values (value1，value2…);

#### 练习：

向employee2表中添加3条记录

insert into employee2 (id,name,gender,birthday,entry\_date,job,salary,resume) values(null,'tom','m','1999-9-1','2020-10-10','doctor',5000,'good');

insert into employee2 values(null,'rose','w','2008-1-10','2019-8-10','teacher',3000,'good');

insert into employee2(id,name,gender) values(null,'jerry','m');

insert into employee2 values(null,'张飞','男','2008-1-10','2019-8-10','将军',3000,'猛男一个');

#### 注意：

##### 若值是字符串或日期，需要用单引号包裹

##### 如果插入的字段为全部字段，列名可省略

##### 若字段是自增，插入null或0值也没问题，也能自增

#### 乱码问题

##### 原因：如果安装时mysql选择的编码为utf-8，而cmd窗口默认用的GBK，即编码不一致导致。

##### 解决1(临时方法，当前窗口有效)：用sql语句，set names gbk;解决，只是告诉mysql当前数据为gbk，不会改变mysql底层的编码存储方式。

##### 解决2(永久)：在mysql安装目录下有my.ini文件，修改default-character-set=gbk，不会改变mysql底层的编码存储方式。

### 更新记录(update)

#### 语法：update 表名 set 列名=值表达式[,列名=值表达式,…] [where 条件表达式]

#### 练习：

将所有员工的薪水改为2000

update employee2 set salary=2000;

将张飞的工资改为1000

update employee2 set salary=1000 where name='张飞';

将张飞的工资在原来的基础上增加500

update employee2 set salary=salary+500 where name='张飞';

### 删除记录(delete)

#### 语法：delete from 表名 [where 条件表达式]

#### 删除所有数据

##### delete from 表名; 一行行删除数据

##### turncate table 表名; 摧毁整个表重建建 数据太多的情况效率更高

### 查询记录(查询)

#### 语法

SELECT <目标表的列名或表达式序列>

FROM <基本表名或视图序列>

[WHERE <行条件表达式>]

[GROUP BY <列名序列>]

[HAVING <组条件表达式>]]

[ORDER BY <列名 [ASC|DESC]> ,…]

[ ]表示成分可有可无

#### 函数

##### Ifnull(x,0):如果x为null，返回数值0，null和任何数据相加为null

#### 聚合函数

##### 注意

只对列聚合，不对行!

SELECT NAME,MAX(chinese) FROM exam;，这里写了name，返回的名字不是最高分的人的名字，NAME默认会返回所有行，但max只有一行。此时只能用嵌套查询

##### count()函数

求记录数，count不会忽略null，如果有id列有一个值为null，那么count(id)的记录时不会加上有null值的行.

##### avg/max/sum…等

计算null值时会忽略

##### 练习：

统计一个班共多少学生

select count(\*) from exam;

统计数学大于75分的学生有多少个

select count(\*) from exam where math>75;

统计一个班的数学总成绩

select sum(math) from exam;

统计一个班的英语平均分

select avg(english) from exam;

select avg(ifnull(english,0)) from exam;

求班级总分的最高分和最低分

select max(chinese+math+ifnull(english,0)) from exam;

select min(chinese+math+ifnull(english,0)) from exam;

查询总分最多的学生

select \* from exam where (chinese+math+ifnull(english,0))=(

select max(chinese+math+ifnull(english,0)) from exam);

#### 别名

##### Select name [as] n from exam; mysql中as可省略

#### 去重distinct

##### SELECT DISTINCT column\_name,column\_name FROM table\_name;

#### Where子句

##### 下面的运算符可以在 WHERE 子句中使用：

|  |  |
| --- | --- |
| 运算符 | 描述 |
| = | 等于 |
| <> | 不等于。注释：在 SQL 的一些版本中，  该操作符可被写成 !=，mysql中二者都可使用 |
| > | 大于 |
| < | 小于 |
| >= | 大于等于 |
| <= | 小于等于 |
| BETWEEN值1 AND 值2 | 在某个范围内，mysql包括二端的值 |
| LIKE | 搜索某种模式 like ‘66’ 不使用通配符等同于 =’66’,数组可以 like 66  如果有通配符，必须 加单引号  在 SQL 中，可使用以下通配符：   |  |  | | --- | --- | | % | 替代 0 个或多个字符 | | \_ | 替代一个字符 | |
| IN | 指定针对某个列的多个可能值 |

##### REGEXP/RLIKE

效果相同，前面可加not

|  |  |
| --- | --- |
| [*charlist*] | 字符列中的任何单一字符 |
| [^*charlist*] 或 [!*charlist*] | 不在字符列中的任何单一字符 |

##### （1/true,0/false）不带比较运算符的 WHERE 子句：

WHERE 子句并不一定带比较运算符，当不带运算符时，会执行一个隐式转换。当 0 时转化为 false，1 转化为 true。例如：

SELECT studentNO FROM student WHERE 0

则会返回一个空集，因为每一行记录 WHERE 都返回 false。

SELECT studentNO FROM student WHERE 1

返回 student 表所有行中 studentNO 列的值。因为每一行记录 WHERE 都返回 true。

##### not

表达式结果取反，例如：not in… not between…

not math>85

##### exists

EXISTS 运算符用于判断查询子句是否有记录，如果有一条或多条记录存在返回 True，否则返回 False。

SELECT Websites.name, Websites.url

FROM Websites

WHERE NOT EXISTS

(SELECT count FROM access\_log WHERE Websites.id = access\_log.site\_id AND count > 200);

#### ORDER BY 排序

##### 注意：

Asc/Desc只对当前字段有效

mysql默认升序 Asc

##### order by math;

math升序

##### order by math desc;

math降序

##### order by math , engish desc;

先math升序，再engish降序

##### order by math desc , engish desc;

先math降序再engish降序

#### GROUP BY 列名 HAVING

##### HAVING是在分组后筛选，注意与WHERE区别，WHERE 是在分组前

# 数据库的备份与恢复

#### 备份数据库

##### 在cmd命令行，无需登录mysql 管理员模式运行

##### Mysqldump –u<用户名> –p[密码] 数据库名称 >备份文件位置

账户root 密码admin

##### mysqldump -uroot -p mydb1>C:/1.sql 密码后输安全

##### Enter password: \*\*\*\*\*

##### mysqldump -uroot -padmin mydb1>C:/2.sql

#### 还原数据库

##### 在cmd命令行，无需登录mysql 管理员模式运行

和备份差不多，但箭头是反方向

##### Mysqldump –u<用户名> –p[密码] 数据库名称 <备份文件位置

# 多表设计

### 一对一(1:1)

#### 在任何一方保存加入另一方的主键作为外键

### 一对多(1:N)

#### 在N那一方保存加入另一方的主键作为外键

### 多对多(M:N)

#### 将他们联系的属性单独建表，然后把M和N表的主键作为新表的外键

# 多表查询

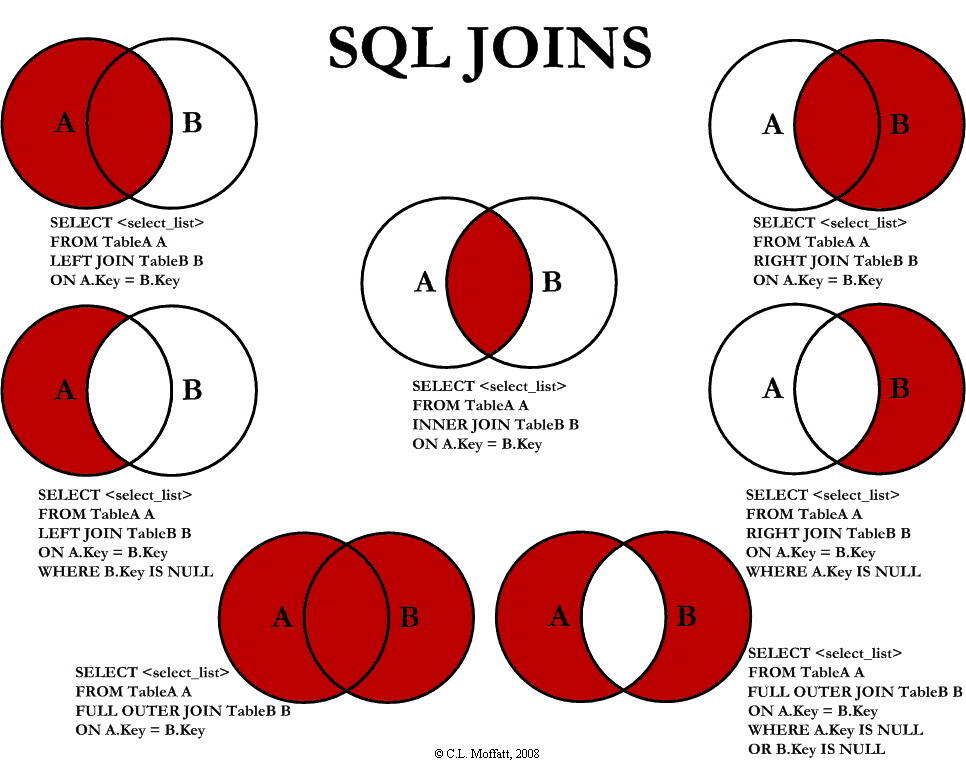
### 笛卡尔积(R x S)

### 连接(都是先做笛卡尔积在条件筛选？)

在使用 **join** 时，**on** 和 **where** 条件的区别如下：

* 1、 **on** 条件是在生成临时表时使用的条件，它不管 **on** 中的条件是否为真，都会返回左边表中的记录。
* 2、**where** 条件是在临时表生成好后，再对临时表进行过滤的条件。这时已经没有 **left join** 的含义（必须返回左边表的记录）了，条件不为真的就全部过滤掉。

MySQL中不支持 FULL OUTER JOIN 提供了union(需要相同数量的列名和相同的列名，是合并行)



##### 内连接查询

Inner join …on …查询左边有，右边也有的数据，即交集

内连接 inner可省略

##### 左外连接查询

left join …on …在内连接基础上，加上左边表的数据

因为右表中没匹配，会以null显示

##### 右外连接查询

right join …on …在内连接基础上，加上左边表的数据

因为左表中没匹配，会以null显示

##### 全外连接

Full join …on…在内连接基础上，加上左边表和右边表数据，若不匹配以null显示

注意：mysql不支持full join on 但可以用union

SQL UNION 语法(要求相同的列名，相同的列数)

SELECT column\_name(s) FROM table1  
UNION  
SELECT column\_name(s) FROM table2;

**注释：**默认地，UNION 操作符选取不同的值。如果允许重复的值，请使用 UNION ALL。

SQL UNION ALL 语法

SELECT column\_name(s) FROM table1  
UNION ALL  
SELECT column\_name(s) FROM table2;

**注释：**UNION 结果集中的列名总是等于 UNION 中第一个 SELECT 语句中的列名。

# 数据库事务

### 事务的概念

事务指逻辑上的一组操作，要么全部成功，要么全部不成功

假如二条sql语句，要么一起执行成功，要么一起失败，

### 管理事务

数据库默认支持事务，数据库的事务是一条SQL语句被认为是一个事务

#### start transaction

开始一个事务

#### commit

以上事务一起提交执行

例如在delete删除某条记录时，commit之前，只是增加删除标记，并未真正删除。

#### rollback

以上事务一起回退不执行

### JDBC中的事务控制

Connection对象的三个方法

mysql数据库默认执行executeUpdate ()后，自动提交事务

#### conn.setAutoCommit(false)

自动提交为false，相当于开启事务，等同于sql中的start transaction

#### conn.commit()

提交事务

#### conn.rollback

回退事务

#### 实例：

*Connection* conn=null;  
*PreparedStatement* pre=null;  
*ResultSet* res=null;  
try {  
  
 conn= JDBCUtil.*getConnection*();  
 //不要自动提交  
 conn.setAutoCommit(false);  
  
 pre = conn.prepareStatement("update users set account=? where id=8");  
 pre.setString(1,"DEF");  
 pre.executeUpdate();

int i = 1 / 0;

pre = conn.prepareStatement("update users set account=? where id=7");  
 pre.setString(1,"abc");  
 pre.executeUpdate();  
 //如果设置了自动提交，而没有commit，数据实际没变  
 conn.commit();  
} catch (SQLException e) {  
 e.printStackTrace();  
 if (conn!=null){  
 try {  
 //虽然 出现了异常commit没有执行，但是还是要rollback释放一下资源

//同时结束当前事务  
 conn.rollback();  
 } catch (SQLException ex) {  
 ex.printStackTrace();  
 }  
 }  
}

### 事务的四大特性(ACID)

#### 原子性(Atomicity)

事务是一个不可分隔的工作单位，事务中的操作要么一起发送，要么一起不发生

#### 一致性(Consistency)

事务前后数据的完整性必须保持一致

#### 隔离性(lsolation)

多个事务同时访问数据库，一个事务不能被其它事务所干扰

#### 持久性(Durability)

事务一旦提交，对数据库的改变是永久的、不可逆的。

### 隔离性(lsolation)

#### 隔离性分析

多个用户同时查询同一条数据：无需隔离

多个用户同时修改同一条数据：需要隔离

一个用户查询，另一个用户修改数据:根据情况，讨论隔离级别

#### 隔离性可能造成的问题

##### 脏读

一个事务读到另一个事务未提交的数据，然后事务未提交rollback后，另一个事务读取的数据又是最开始的数据

设置隔离级别最低：set session transaction isolation level read uncommitted;

例 最开始account=XX

事务一：

开启事务

Start transaction

修改 account =pp

update users set account='PP' where id=8;

事务二：

读取 数据account 等于PP

事务一：rollback

事务二：select 读取 account等于 XX

##### 不可重复读

一个事务读到另一个事务已提交的数据

和脏读顺序相反

##### 虚读(幻读)

发生概率非常低，在读取整表时，多次读取的结果不一样

### 隔离级别

为防止以上问题，数据库提供了不同的隔离级别，越安全效率越低

#### read uncommitted;

不做任何隔离

#### read committed

可以防止脏读，但不能防止 不可重复读和虚读幻读

#### repeatable read

可以防止脏读和不可重复读，但是不能防止虚度（幻读）

#### serializable

可以防止所有隔离性问题,但是数据库被设置为了串行的数据库，性能低

一般用B和C，mysql默认 repeatable read

oracle 默认是 read committed 没有repeatable read

### 操作数据库的隔离级别

#### 查询

SELECT @@tx\_isolation;

#### 设置隔离级别

set <[session][global]> transaction isolation level 隔离级别;

默认是session，表示当前连接有效,global表示全局修改

### 数据库的锁(整个数据库)

#### 共享锁

共享锁可以和共享锁共存，共享锁和排他锁不能共存，在serializable级别下查询时会添加共享锁，其他级别不会

#### 排他锁

排它锁和任何锁不能共存，在任何隔离级别下做增删改操作都会添加排它锁

### 数据库死锁

彼此握着对方的资源互不释放，导致彼此都无法继续执行

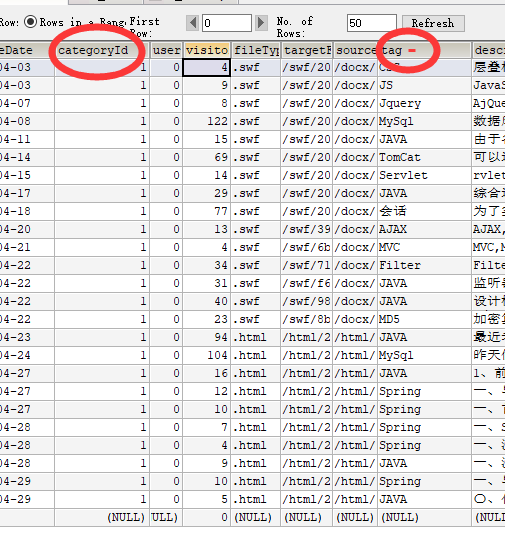
### 数据库活锁

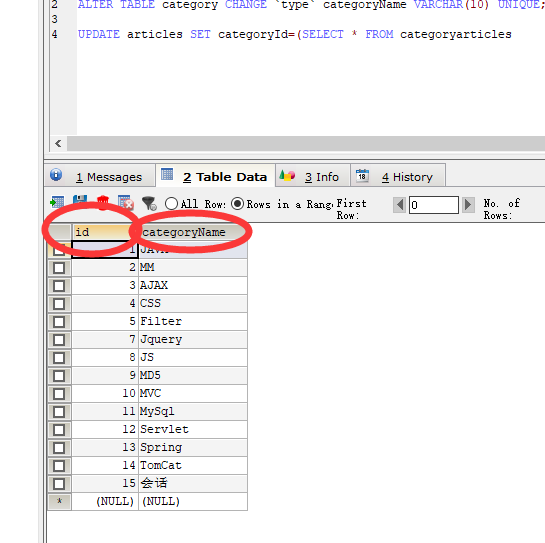
彼此互相谦让，导致彼此都无法继续执行

# 项目中发现的问题点

### 需求：

我想把第一个图片里面的categoryId的值改为第二图中的id值，根据tag和categroyName的对应关系





### 实现

UPDATE articles a INNER JOIN category c ON a.tag=c.categoryName SET categoryId=c.id;