**MySQL学习笔记**

# 1 概念

库与表的概念

表：多行多列的数据。

库：多个表形成一个库。

多个库存放在数据库服务器上。

SQL概念

SQL是结构化查询语言。

DML：数据操作语言，数据的增删改查，使用者角度。

DDL：数据定义语言，建表、建库、建视图等，建设者的角度。

DCL：数据控制语言，数据管理库权限，管理者角度。

## 1.1 数据库的连接

mysql -h -u -p

-u 用户名

-p 密码

-h host主机

例如：mysql -h192.168.8.102 -uroot -p123456

## 1.2 库级知识

显示数据库: show databases;

选择数据库: use dbname;

创建数据库: create database dbname charset utf8;

删除数据库: drop database dbname;

# 2 表的增删改查操作

## 2.1 insert 插入数据操作

规则： 1）往哪张表添加行？ **INSERT INTO 表名**

2）给那几个列添加值？ **（列名称）可省略**

3）分别是什么值？ **VALUES （列对应的值）**

插入操作时注意事项：列和值必须一一对应，并且符合类型要求。

|  |
| --- |
| **INSERT INTO user** (**uid**, **name**, **age**) **VALUES** (1, **'张三'**, 25); **INSERT INTO user** (**name**, **age**) **VALUES** (**'李四'**, 26); **INSERT INTO user** (**name**) **VALUES** (**'王五'**); **INSERT INTO user VALUES** (5, **'刘备'**, 52); *# 忽略列时需要写完所有对应列的值 #INSERT INTO user VALUES ('关羽',45);# 列没有严格对应，执行错误*  **INSERT INTO user** (**name**, **age**) **VALUES** (**'关羽'**, 45),(**'张飞'**, 46);*# 一次插入多行数据* |

## 2.2 update 修改数据操作

规则： 1）改哪一张表？ **UPDATE 表名**

2）改哪几列的值？ **SET 列名=值1，列名=值2......**

3）在哪些行生效？ **WHERE 表达式**

注意事项：一定要有where约束条件，即在哪些生效。

|  |
| --- |
| **UPDATE user SET age**=27 **WHERE name**=**'王五'**; **UPDATE user SET name**=**'赵六'**,**age**=28 **WHERE uid**=4; *# 修改多列用逗号隔开* |

## 2.3 delete 删除数据操作

表数据最小单位是行，所以是按行删除，不必指定列。

规则： 1）删除一张表的数据？ **DELETE FROM 表名**

2）删掉表中的哪些行？ **WHERE 表达式**

注意事项：删除必须写where约束条件，也不能写常量，如where 1，否则会删除整张表数据。

|  |
| --- |
| **DELETE FROM user WHERE age**=26;  **DELETE FROM user WHERE uid**>2; |

## 2.4 select 查询数据操作

规则： 1）查询哪些列数据？ **SELECT列名1 列名2......**

2）从哪张表查询？ **FROM 表名**

3）选择哪些行生效？ **WHERE** **表达式**

|  |
| --- |
| **SELECT** *\** **FROM user**; *# 实际开发中很少使用* **SELECT** *\** **FROM user WHERE name**=**'关羽'**; **SELECT name**,**age FROM user WHERE age**<30; *# 查询符合条件的指定列* |

# 3 综合查询操作

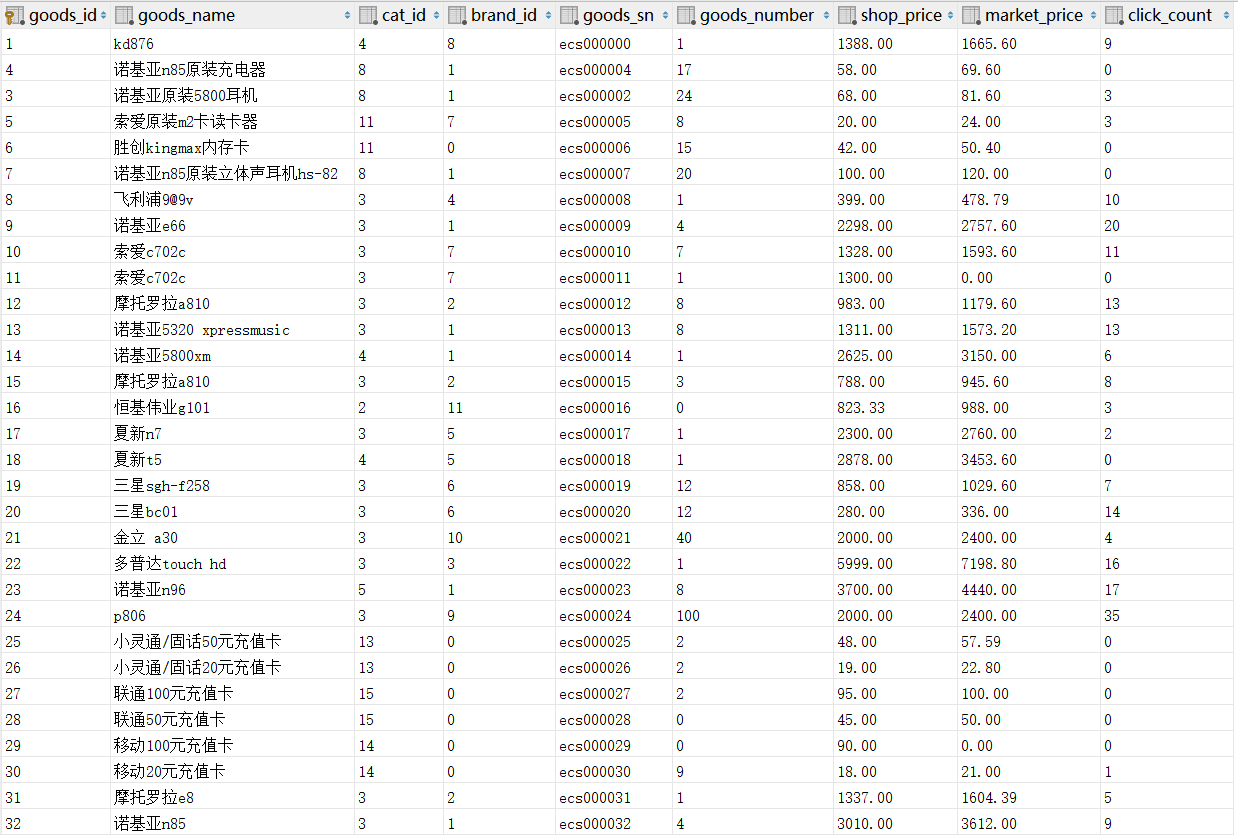
MySQL的比较运算和逻辑运算符如下图所示，注意and比or的优先级高，当表达式比较复杂时，习惯使用括号，避免出现歧义。

in用法：in的后面跟着枚举，例如in (3,11)

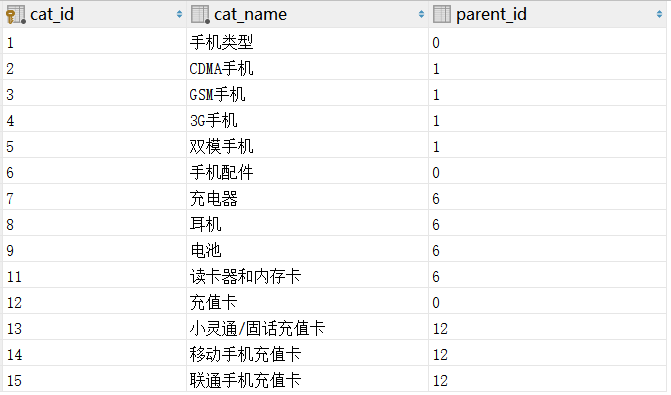
between通常和 and连用，例如 between 100 and 500



练习使用的商品表如下图所示：



练习使用的栏目表如下图所示：



## 3.1 where 条件约束

where是针对磁盘数据文件，后面一般都是列变量的表达式。

模糊查询时，百分号%表示匹配所有字符，下划线\_表示匹配一个字符。

|  |
| --- |
| **⑴** 主键为32的商品  **SELECT goods\_id**,**goods\_name**,**shop\_price FROM** goods **WHERE goods\_id**=32;  **⑵** 不属第3栏目的所有商品  **SELECT cat\_id**,**goods\_name**,**shop\_price FROM** goods **WHERE cat\_id**!=3;  **⑶** 本店价格高于3000元的商品  **SELECT goods\_id**,**goods\_name**,**shop\_price FROM** goods **WHERE shop\_price**>3000;  **⑷** 本店价格低于或等于100元的商品  **SELECT goods\_id**,**goods\_name**,**shop\_price FROM** goods **WHERE shop\_price**<=100;  **⑸** 取出第4栏目或第11栏目的商品(不用or)  **SELECT cat\_id**,**goods\_name**,**shop\_price FROM** goods **WHERE cat\_id IN** (4,11);  **⑹** 取出100<=价格<=500的商品(不许用and)  **SELECT goods\_id**,**goods\_name**,**shop\_price FROM** goods **WHERE shop\_price BETWEEN** 100 **AND** 500;  **⑺** 不属于第3栏目且不属于第11栏目的商品(and,或not in分别实现)  ① **SELECT cat\_id**,**goods\_name**,**shop\_price FROM** goods **WHERE** (**cat\_id**!=3) && (**cat\_id**!=11);  ② **SELECT cat\_id**,**goods\_name**,**shop\_price FROM** goods **WHERE cat\_id NOT IN** (3,11);  **⑻** 取出价格大于100且小于300,或者大于3000小于4000商品  **SELECT goods\_id**,**goods\_name**,**shop\_price FROM** goods **WHERE** (**shop\_price**>100 && **shop\_price**<300) || (**shop\_price**>3000 && **shop\_price**<4000);  **⑼** 取出第3个栏目下面价格<1000或>3000,并且点击量>5的系列商品  **SELECT cat\_id**,**goods\_name**,**shop\_price**,**click\_count FROM** goods **WHERE** (**cat\_id**=3) && (**shop\_price**<1000 || **shop\_price**>3000) && (**click\_count**>=5);  **⑽** 取出第1个栏目下面的商品(注意:第1栏目下面没商品,但其子栏目下有)  **SELECT cat\_id**,**goods\_name**,**shop\_price FROM** goods **WHERE cat\_id IN** (2,3,4,5);*# 手动查看第1栏目的子栏目有2 3 4 5*  **⑾** 取出名字以"诺基亚"开头的商品  **SELECT cat\_id**,**goods\_name**,**shop\_price FROM** goods **WHERE goods\_name LIKE '诺基亚%'**;  **⑿** 取出诺基亚N系列的手机  **SELECT goods\_id**,**goods\_name**,**shop\_price FROM** goods **WHERE goods\_name LIKE '诺基亚N\_\_'**;  **⒀** 取出名字不以"诺基亚"开头的商品  **SELECT goods\_id**,**goods\_name**,**shop\_price FROM** goods **WHERE goods\_name NOT LIKE '诺基亚%'**;  **⒁** 取出第3个栏目下面价格在1000到3000之间,并且点击量>5 的"诺基亚"开头的商品  **SELECT cat\_id**,**goods\_name**,**shop\_price**,**click\_count FROM** goods **WHERE** (**cat\_id**=3) && (**shop\_price**>1000 && **shop\_price**<3000) && (**click\_count**>5) && (**goods\_name LIKE '诺基亚%'**);  **⒂** 把num值处于[20,29]之间,改为20，num值处于[30,39]之间的,改为30  **UPDATE** mian **SET num**=*floor*(**num**/10)\*10 **WHERE num BETWEEN** 20 **AND** 39;  **⒃** 把good表中商品名为'诺基亚xxxx'的商品,改为'HTCxxxx'  **UPDATE** goods **SET goods\_name**=*INSERT*(**goods\_name**,1,3,**'HTC'**) **WHERE goods\_name LIKE '诺基亚%'**; *# 注：字符串位置从1开始* |

## 3.2 group 数据分组

使用group时首先会对该列重新排序，再做统计，所以比较耗资源，尽量避免使用。

|  |
| --- |
| **⑴** 查出最贵和最便宜的商品的价格  **SELECT** *max*(**shop\_price**),*min*(**shop\_price**) **FROM** goods;  **⑵** 查出最新和最旧的商品编号  **SELECT** *max*(**goods\_id**),*min*(**goods\_id**) **FROM** goods;  **⑶** 查询该店所有商品的库存总量  **SELECT** *sum*(**goods\_number**) **FROM** goods;  **⑷** 查询所有商品的平均价格  **SELECT** *avg*(**shop\_price**) **FROM** goods;  **⑸** 查询该店一共有多少种商品  **SELECT** *count*(*\**) **FROM** goods;  **⑹** 查询每个栏目下面最贵商品价格、最便宜商品价格、商品平均价格、商品库存量、商品库存总价格、商品种类  **SELECT** *max*(**shop\_price**),*min*(**shop\_price**),*avg*(**shop\_price**),*sum*(**goods\_number**),*count*(*\**) **FROM** goods **GROUP BY cat\_id**; |

## 3.3 having 筛选数据

having是针对where条件结果进行筛选，是对内存数据操作，所以having必须用在where之后。

|  |
| --- |
| **⑴** 查询该店的商品比市场价所节省的价格  **SELECT goods\_name**,**market\_price**-**shop\_price FROM** goods;  **⑵** 查询每个商品所积压的货款  **SELECT goods\_name**,**shop\_price**\***goods\_number FROM** goods;  **⑶** 查询该店积压的总货款  **SELECT** *sum*(**shop\_price**\*goods.**goods\_number**) **FROM** goods;  **⑷** 查询该店每个栏目下面积压的货款  **SELECT cat\_id**,*sum*(**shop\_price**\***goods\_number**) **FROM** goods **GROUP BY cat\_id**;  **⑸** 查询比市场价省钱200元以上的商品及该商品所省的钱  **SELECT goods\_id**,**goods\_name**,(**market\_price**-goods.**shop\_price**) **AS** save **FROM** goods **HAVING** save>200;  **⑹** 查询积压货款超过2W元的栏目,以及该栏目积压的货款  **SELECT cat\_id**,*sum*(**shop\_price**\*goods.**goods\_number**) **AS** catPrice **FROM** goods **GROUP BY cat\_id HAVING** catPrice>20000;  **⑺** 查询出2门及2门以上不及格者的平均成绩    **SELECT name**,*avg*(**score**),*sum*(**score**<60) **AS** fail **FROM** result **GROUP BY name HAVING** fail>=2; |

## 3.4 order by 排序和limit限制取出条目

升序：asc（默认）

降序：desc

当按某列排序无法满足要求时，可以在列的内部再继续排序。

实际使用中oder by经常和limit配合一起使用。

limit有两个参数，分别是limit 偏移量，取出条目。可以使用在分页上。

|  |
| --- |
| (1)按价格由高到低排序  **SELECT goods\_id**,**goods\_name**,**shop\_price FROM** goods **ORDER BY shop\_price ASC**;  (2)按发布时间由早到晚排序  **SELECT goods\_id**,**goods\_name**,**shop\_price FROM** goods **ORDER BY goods\_id DESC**;  (3)接栏目由低到高排序,栏目内部按价格由高到低排序  **SELECT cat\_id**,**goods\_name**,**shop\_price FROM** goods **ORDER BY cat\_id ASC**, **shop\_price DESC**;  (4)取出价格最高的前三名商品  **SELECT goods\_id**,**goods\_name**,**shop\_price FROM** goods **ORDER BY shop\_price DESC LIMIT** 0,3;  (5)取出点击量前三名到前5名的商品  **SELECT goods\_name**,**shop\_price**,**click\_count FROM** goods **ORDER BY click\_count DESC LIMIT** 0,5; |

## 3.5 子查询

### 3.5.1 where子查询

内层select查询的结果充当外层select的where条件的查询。

|  |
| --- |
| 取出每个栏目下最新的商品  **SELECT cat\_id**,**goods\_id**,**goods\_name**,**shop\_price FROM** goods **WHERE goods\_id IN** (**SELECT** *max*(**goods\_id**) **FROM** goods **GROUP BY cat\_id**); |

### 3.5.2 from子查询

内层select作为一张表（AS table），外层select从内层表取出数据。

|  |
| --- |
| 取出每个栏目下最新的商品  **SELECT cat\_id**,**goods\_id**,**goods\_name**,**shop\_price FROM** (**SELECT** *\** **FROM** goods **ORDER BY cat\_id ASC**,**goods\_id DESC**) **AS** tmp **GROUP BY cat\_id**; |

### 3.5.3 exists子查询

exists指定一个子查询，检测行的存在。该子查询实际上并不返回任何数据，而是返回值True或False。

exists子查询能完成的where子查询 in也能完成。

|  |
| --- |
| 取出没有商品的栏目  **SELECT** *\** **FROM** category **WHERE NOT** *exists*(**SELECT** *\** **FROM** goods **WHERE** category.**cat\_id**=goods.**cat\_id**); |

### 3.5.4 子查询练习

|  |
| --- |
| (1)查询出最新一行商品  **SELECT goods\_id**,**goods\_name**,**shop\_price FROM** goods **WHERE goods\_id**=(**SELECT** *max*(**goods\_id**) **FROM** goods);  (2)查询出编号为19的商品的栏目名称(where和连接查询)  where子查询：  **SELECT cat\_id**,**cat\_name FROM** category **WHERE cat\_id**=(**SELECT cat\_id FROM** goods **WHERE goods\_id**=19);  连接查询：  **SELECT** category.**cat\_id**,category.**cat\_name FROM** category **INNER JOIN** goods **ON** category.**cat\_id**=goods.**cat\_id  WHERE** goods.**goods\_id**=19;  (3)用where和from型子查询方式把每个栏目下面最新的商品取出来  where子查询：  **SELECT goods\_id**,**cat\_id**,**goods\_name FROM** goods **WHERE goods\_id IN** (**SELECT** *max*(**goods\_id**) **FROM** goods **GROUP BY cat\_id**);  from子查询：  **SELECT goods\_id**,**cat\_id**,**goods\_name FROM** (**SELECT** *\** **FROM** goods **ORDER BY cat\_id ASC**,**goods\_id DESC**) **AS** tmp **GROUP BY** tmp.**cat\_id**;  (4)用exists型子查询,查出所有有商品的栏目  **SELECT** *\** **FROM** category **WHERE** *exists*(**SELECT** *\** **FROM** goods **WHERE** category.**cat\_id**=goods.**cat\_id**); |

## 3.6 连接查询

### 3.6.1 INNER JOIN内连接查询

把两张独立代表拼接成一张大表，拼接时指定匹配条件。

格式：表1 INNER JOIN 表2 ON 匹配条件

|  |
| --- |
| 把boy表和girl表同一组的人取出来  **SELECT** *\** **FROM** boy **INNER JOIN** girl **ON** boy.**hid**=girl.**hid**; |

### 3.6.2 LEFT JOIN左连接查询

以左边的表为标准，有匹配条件取出值来，没有则填充默认值。

|  |
| --- |
| 以boy表为标准，把boy表和girl表进行同组匹配  **SELECT** *\** **FROM** boy **LEFT JOIN** girl **ON** boy.**hid**=girl.**hid**; |

### 3.6.3 RIGHT JOIN右连接查询

以右边的表为标准，有匹配条件取出值来，没有则填充默认值。

|  |
| --- |
| 以girl表为标准，把boy表和girl表进行同组匹配  **SELECT** *\** **FROM** boy **RIGHT JOIN** girl **ON** boy.**hid**=girl.**hid**; |

|  |
| --- |
| (1)取出所有商品的商品名,栏目名,价格  **SELECT** goods.**goods\_id**,goods.**goods\_name**,category.**cat\_name**,goods.**shop\_price FROM** goods **LEFT JOIN** category **ON** goods.**cat\_id**=category.**cat\_id**;  (2)取出第4个栏目下的商品的商品名,栏目名,价格  **SELECT** goods.**goods\_id**,goods.**goods\_name**,category.**cat\_name**,goods.**shop\_price FROM** goods **LEFT JOIN** category **ON** goods.**cat\_id**=category.**cat\_id WHERE** goods.**cat\_id**=4;  (3) 查出 2006-6-1 到2006-7-1之间举行的所有比赛，并且用以下形式列出：主队名 比分 客队 比赛时间  表名：m  +-----+------+------+------+------------+  | mid | hid | gid | mres | matime |  +-----+------+------+------+------------+  | 1 | 1 | 2 | 2:0 | 2006-05-21 |  | 2 | 2 | 3 | 1:2 | 2006-06-21 |  | 3 | 3 | 1 | 2:5 | 2006-06-25 |  | 4 | 2 | 1 | 3:2 | 2006-07-21 |  +-----+------+------+------+------------+  队名：t  +------+----------+  | tid | tname |  +------+----------+  | 1 | 国安 |  | 2 | 申花 |  | 3 | 公益联队 |  +------+----------+  写法一：  **SELECT** ttb.**tname**,ttb.**mres**,t.**tname**,ttb.**matime FROM**  (**SELECT** *\** **FROM** m **LEFT JOIN** t **ON** m.**hid**=t.**tid**)**AS** ttb  **LEFT JOIN** t **ON** ttb.**gid**=t.**tid**  **WHERE** ttb.**matime BETWEEN '2006-06-01' AND '2006-07-01'**;  写法二：  **SELECT** t1.**tname**,**mres**,t2.**tname**,**matime FROM** m **LEFT JOIN** t **AS** t1 **ON** m.**hid** = t1.**tid  LEFT JOIN** t **AS** t2 **ON** m.**gid** = t2.**tid  WHERE matime BETWEEN '2006-06-01' AND '2006-07-01'**; |

## 3.7 union查询

把两条或多条sql查询的结果合并成一个结果集。

主要用途：两张不同的表中有相同的列时，把结果集合并。或者简化两个复杂的where条件。

使用条件：两个表的列必须相同，但是例外的是列名称可以不相同。

注意：

①完全相等的行将会合并，合并是耗时的操作，一般不让union合并，使用union all则不合并。

②union子句中一般不用order by，在union合并后可以再order by。

|  |
| --- |
| (1)合并表a和表b，不合并相同的行  **SELECT** *\** **FROM** a **UNION ALL SELECT** *\** **FROM** b;  (2)对两个表相同id的行进行num列求和。  **SELECT** tmp.**id**,*sum*(tmp.**num**) **FROM**  (**SELECT** *\** **FROM** a **UNION ALL SELECT** *\** **FROM** b) **AS** tmp  **GROUP BY** tmp.**id**; |

# 4 表的列操作

## 查看表字段信息

DESC表名

例如：DESC student;

## 查看表的所有信息

SHOW CREATE TABLE 表名;

|  |
| --- |
| **SHOW CREATE TABLE** student; |

## 查看当前库下所有表信息

**SHOW TABLE status**;

或列形式输出：**SHOW TABLE status \G**;

或过滤表输出：**SHOW TABLE status WHERE name**=**'goods' \G**;

## 新建表

CREATE TABLE 表名(列名 列类型 列属性，……);

|  |
| --- |
| **CREATE TABLE** student (  **sid TINYINT UNSIGNED**,  **name VARCHAR**(20),   **age INT** ); |

## 添加列

ALTER TABLE 表名 ADD COLUMN 列名 列类型 列属性，……

|  |
| --- |
| 在最后添加列：  **ALTER TABLE** student **ADD COLUMN sn TINYINT**(6) **ZEROFILL**;  在指定位置添加列：  **ALTER TABLE user ADD height TINYINT** **AFTER weight**; |

## 删除表

DROP TABLE 表名;

|  |
| --- |
| **DROP TABLE** t1; |

## 清空表内容

TRUNCATE 表名;

TRUNCATE相当与drop和creat表两个操作。

## 删除列

ALTER TABLE 表名 DROP 列名;

|  |
| --- |
| **ALTER TABLE** student **DROP COLUMN** sn; |

## 修改表名

ALTER TABLE 旧表名 RENAME TO 新表名;

|  |
| --- |
| **ALTER TABLE** student **RENAME TO** stu; |

## 修改列属性

ALTER TABLE表名MODIFY sid 列名 列类型 列属性;

|  |
| --- |
| **ALTER TABLE** student **MODIFY sid TINYINT UNSIGNED**; |

## 修改列名称和属性

ALTER TABLE 表名 CHANGE 旧列名 新列名 列类型 列属性;

|  |
| --- |
| **ALTER TABLE** student **CHANGE sid** **id INT UNSIGNED**; |

## 添加主键约束

ALTER TABLE 表名 ADD CONSTRAINT PRIMARY KEY 表名(列名);

|  |
| --- |
| **ALTER TABLE** student **ADD CONSTRAINT PRIMARY KEY** student(**sid**); |

## 删除主键约束

ALTER TABLE 表名 DROP PRIMARY KEY;

|  |
| --- |
| **ALTER TABLE** student **DROP PRIMARY KEY**; |

添加外键约束

alter table 从表 add constraint 外键（形如：FK\_从表\_主表） foreign key 从表(外键字段) references 主表(主键字段);

删除外键约束

alter table 表名 drop foreign key 外键（区分大小写）;

# 5 类型

建表的过程就是声明列的过程。

## 整型

|  |
| --- |
| **整数类型         字节数       有符号范围（默认）       无符号范围**  TINYINT         1字节         (-128，127)           (0，255)  SMALLINT       2字节     (-32768，32767)       (0，65 535)  MEDIUMINT     3字节     (-，-1) (0，-1)  INT   4字节   (-，-1) (0，-1)  BIGINT         8字节  (-，-1) (0，-1) |

可选参数：整型[(M)] [UNSIGNED] [ZEROFILL]

UNSIGNED：无符号数，例如：age TINYINT UNSIGNED;

ZEROFILL：其中参数M表示字节宽度，自动会转为无符号整型，一般适用于学号、编码等宽度无符号数字，可以用0填充，例如：sn TINYINT(6) ZEROFILL;

## 浮点型

注意：浮点数有精度损失的。

|  |  |  |  |
| --- | --- | --- | --- |
| **浮点类型** | **字节数** | **有符号范围（默认）** | **无符号范围** |
| FLOAT | 4 | -3.402823466E+38到-1.175494351E-38 | 0和1.175494351E-38到3.402823466E+38 |
| DOUBLE | 8 | -1.7976931348623157E+308到-2.2250738585072014E-308 | 0和2.2250738585072014E-308到 1.7976931348623157E+308 |

可选参数：浮点型[(M,D)] [UNSIGNED] [ZEROFILL]

其中M表示显示的值（包括小数位）最大位数，D表示小数位数。

例如：sara FLOAT(7,3) 规定显示的值不会超过7位数字，小数点后面带有 3位数字。

对于小数点后面的位数超过允许范围的值，MySQL 会自动将它四舍五入为最接近它的值。

## 定点型

把整数和小数部分分开存储，比较精确。

可选参数：DECIMAL[(M[,D])] [UNSIGNED] [ZEROFILL]

例如：sara DECIMAL(6,2);

## 字符串

|  |
| --- |
| **字符串类型     字节数        描述及存储需求**      CHAR         0～255           定长字符串      VARCHAR      0～255          变长字符串      TINYBLOB     0～255         不超过 255 个字符的二进制字符串      TINYTEXT     0～255         短文本字符串      BLOB         0～65535       二进制形式的长文本数据      TEXT         0～65535       长文本数据      MEDIUMBLOB   0～-1 二进制形式的中等长度文本数据      MEDIUMTEXT   0～-1 中等长度文本数据      LOGNGBLOB     0～-1 二进制形式的极大文本数据      LONGTEXT     0～-1 极大文本数据 |

注意：

char型，如果不够M个字符，内部用空格补齐，取出时会把右侧空格删除，意味着本身有的空格会被丢弃。varchar类型则不会。

速度上定长比不定长的更快。

blob是二进制类型，用来储存图像、音频等类型。可以防止字符集问题导致信息丢失。

## 枚举和集合

枚举：ENUM('value1','value2',...)，插入值时，只能在枚举范围中的一个。

集合：SET('value1','value2',...)，插入值时，值是在集合的一个或多个。

## 日期时间类型

|  |  |  |  |
| --- | --- | --- | --- |
| **类型** | **字节数** | **范围** | **格式** |
| DATE | 4 | 1000-01-01～9999-12-31 | YYYY-MM-DD |
| TIME | 3 | '00:00:00'～'23:59:59' | HH:MM:SS |
| YEAR | 1 | 1901～2155 | YYYY |
| DATETIME | 8 | 1000-01-01 00:00:00～9999-12-31 23:59:59 | YYYY-MM-DD HH:MM:SS |
| TIMESTAMP | 4 | 1970-01-01 00:00:00～2037年某时 | YYYY-MM-DD HH:MM:SS |

注意：

TIMESTAMP是系统自动填充的时间戳，效率不高，不建议使用，通常用int unsigned来储存时间戳。

|  |
| --- |
| 获取时间戳  **SELECT** *UNIX\_TIMESTAMP*() ; *#当前时间戳1451588688* **SELECT** *UNIX\_TIMESTAMP*(**'2016-01-01 00:00:00'**);  把日期时间转为时间戳  **SELECT** *FROM\_UNIXTIME*(1451588888, **'%Y-%m-%d %h:%i:%s'**); *# 2016-01-01 03:08:08* |

# 6 列的属性

## 6.1 列的默认值

实际使用中避免列的默认值为null，创建表时给列一个初始值。

在列的类型后面添加NOT NULL DEFAULT 值。

例如：**id INT UNSIGNED NOT NULL DEFAULT** 0

## 6.2 主键与自增

主键：primary key，能够区分每一行，一般和auto\_increment一起使用。

有两种方式声明该列为主键：

方式一：在列的类型之后声明primary key

例如：**id INT UNSIGNED PRIMARY KEY**

方式二：声明完列之后，在最后声明那一列为主键primary key(列名)

例如：**PRIMARY KEY**(**id**)

主键与自增搭配使用：

例如：**id INT PRIMARY KEY AUTO\_INCREMENT**

提高效率建表原则：定长与变长分离，常用和不常用分离。

# 7 视图

视图又称为虚拟表，是通过sql查询的结果集。

## 视图用途

(1)权限控制：有些不想给用户看的列，比如进货价格，通过视图可以控制该列不被用户看到。

(2)简化复杂的查询，通过视图把查询结果保存下来，视图作为查询条件。

创建视图：CREATE VIEW 视图名称 AS 查询语句

例如：**CREATE VIEW** vuser **AS SELECT username**,**gender FROM user**;

注意：当视图和物理表的列关系是一一对应（类似可逆函数），即视图相当于物理表的子表时，无论修改视图还是修改物理表，都会自动对应修改。如果不是一一对应的不可以修改。

## 视图的algorithm

视图有两种算法：merge(合并)和temptable(临时表)

merge算法：把建视图语句和查询视图语句合并后查询物理表的语句。

temptable算法：先从物理表查询得到临时表，然后再查询临时表得到结果。

创建视图时可以指定算法，不指定算法则让系统自适应选择算法。

CREATE [ALGORITHM = {MERGE | TEMPTABLE}] VIEW AS 查询语句

例如：**CREATE ALGORITHM** = **MERGE VIEW** vuser **AS SELECT username**,**gender FROM user**;

## 视图操作

和表的操作类似，把table改为view则可。

查看视图：**DESC** 视图名

查看视图所有信息：**SHOW CREATE VIEW**视图名;

删除视图：**DROP VIEW** 视图名

查看哪些表为视图：**SHOW TABLE status WHERE comment**=**'view'**;

# 8 表引擎

|  |
| --- |
| **CREATE TABLE** account(  **id INT UNSIGNED PRIMARY KEY**,  **name CHAR**(10) **NOT NULL DEFAULT ''**,  **balance INT NOT NULL DEFAULT** 0 )**ENGINE** innodb **CHARSET** *utf8*; |

引擎不同，组织数据方式也不同，常用的Myisam和InnoDB两种引擎，如下图。



Myisam和InnoDB的6个区别：

|  |  |  |
| --- | --- | --- |
|  | **MyISAM** | **InnoDB** |
| **构成上的区别** | 每个MyISAM在磁盘上存储成三个文件。   .frm文件存储表定义。  .MYD 数据文件。  .MYI索引文件。 | 基于磁盘的资源是InnoDB表空间数据文件和它的日志文件，InnoDB 表的大小只受限于操作系统文件的大小，一般为 2GB |
| **事务处理上方面** | MyISAM类型的表强调的是性能，其执行数度比InnoDB类型更快，但是不提供事务支持 | InnoDB提供事务支持事务，外部键等高级数据库功能 |
| **SELECT**  **UPDATE,**  **INSERT，Delete操作** | 如果执行大量的SELECT，MyISAM是更好的选择 | **①** 如果你的数据执行大量的**INSERT或UPDATE**，出于性能方面的考虑，应该使用InnoDB表    **② DELETE   FROM table**时，InnoDB不会重新建立表，而是一行一行的删除。    **③LOAD  TABLE FROM MASTER**操作对InnoDB是不起作用的，解决方法是首先把InnoDB表改成MyISAM表，导入数据后再改成InnoDB表，但是对于使用的额外的InnoDB特性（例如外键）的表不适用。 |
| **对AUTO\_INCREMENT的操作** | 每表一个AUTO\_INCREMEN列的内部处理。    **MyISAM为INSERT和UPDATE操作自动更新这一列**。这使得AUTO\_INCREMENT列更快（至少10%）。在序列顶的值被删除之后就不能再利用。(当AUTO\_INCREMENT列被定义为多列索引的最后一列，可以出现重使用从序列顶部删除的值的情况）。    AUTO\_INCREMENT值可用ALTER TABLE或myisamch来重置    对于AUTO\_INCREMENT类型的字段，InnoDB中必须包含只有该字段的索引，但是在MyISAM表中，可以和其他字段一起建立联合索引    更好和更快的auto\_increment处理 | 如果你为一个表指定AUTO\_INCREMENT列，在数据词典里的InnoDB表句柄包含一个名为自动增长计数器的计数器，它被用在为该列赋新值。    自动增长计数器仅被存储在主内存中，而不是存在磁盘上    关于该计算器的算法实现，请参考    **AUTO\_INCREMENT列在InnoDB里如何工作** |
| **表的具体行数** | select count(\*) from table,MyISAM只要简单的读出保存好的行数，注意的是，当count(\*)语句包含   where条件时，两种表的操作是一样的 | InnoDB 中不保存表的具体行数，也就是说，执行select count(\*) from table时，InnoDB要扫描一遍整个表来计算有多少行 |
| **锁** | 表锁 | 提供行锁(locking on row level)，提供与 Oracle 类型一致的不加锁读取(non-locking read in    SELECTs)，另外，InnoDB表的行锁也不是绝对的，如果在执行一个SQL语句时MySQL不能确定要扫描的范围，InnoDB表同样会锁全表，例如update table set num=1 where name like “%aaa%” |

# 9 字符集和校对集

乱码原因：文字本来的字符集和展示的字符集不一致造成的。

解决乱码方法：

查看数据库的字符集是否一致：**SHOW VARIABLES LIKE '%charact%'**

把character\_set\_client，character\_set\_connection，character\_set\_results三个设置为一致的字符集即可，

简写为set names utf8;

校对集：就是排序规则。

# 10 索引

索引是数据的目录，能快速定位需要取数据的位置。

索引提高列查询的速度，降低了增、删、改的速度。

一般在查询频繁的列或值重复度低的列添加索引。

## 单列索引

key：普通索引

例如：key cat\_id(cat\_id)

unique key：唯一索引

例如：unique key email(email)

primary key：主键索引，一张表只能存在一个

例如：primary key (id)

fulltext：全文索引，在中文环境下几乎无效。一般用第三方解决方案sphinx

索引长度：建索引时，可以取列的部分字符作为索引，节省空间。

例如：unique key email(email(10)); 取email列的前10个字符作为索引。

## 多列索引

把两列或多列的值作为整体后再建索引，左前缀规则。

例如：key xm(xing, ming)

|  |
| --- |
| **CREATE TABLE name** (  **xing CHAR**(2),  **ming CHAR**(10),  **KEY** xm(**xing**,**ming**) );  在select前面添加关explain键字，得到结果的possible\_key可以查看是否使用到索引查询。  **EXPLAIN SELECT** *\** **FROM name WHERE xing**=**'刘' AND ming**=**'备'**;*# 使用索引xm查询* **EXPLAIN SELECT** *\** **FROM name WHERE xing**=**'刘'**;*# 使用索引xm查询* **EXPLAIN SELECT** *\** **FROM name WHERE ming**=**'备'**;*# 没有使用索引xm查询* |

## 冗余索引

在某个列上可能存在多个索引，

比如某个表：key xm(xing, ming)，key ming(ming)，对于列ming来说，索引xm 和 ming两个索引覆盖，叫做冗余索引。

## 索引操作

查看索引：**SHOW INDEX FROM** 表名;

删除索引：**ALTER TABLE** 表名 **DROP INDEX** 索引名;

添加索引：

普通索引：**ALTER TABLE** 表名 **ADD INDEX** 索引名(列名);

唯一索引：**ALTER TABLE** 表名 **ADD UNIQUE** 索引名(列名);

主键索引：**ALTER TABLE** 表名 **ADD PRIMARY KEY**(列名);

# 11 事务

事务特性：隔离性、原子性、一致性、持久性。

启动事务：**START TRANSACTION**;

提交事务：**COMMIT**;

取消事务：**ROLLBACK**;

注意commit一旦发生之后，rollback无法回滚。

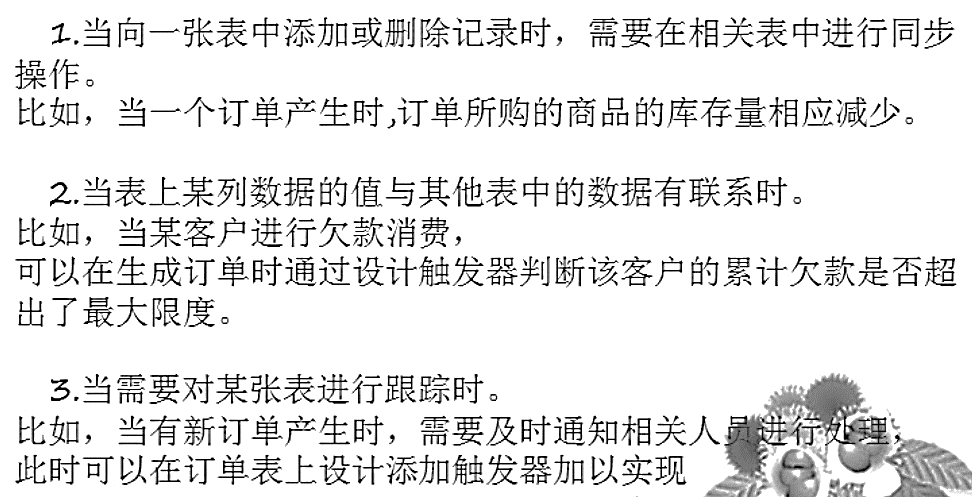
|  |
| --- |
| **START TRANSACTION**;*# 启动事务* **UPDATE** account **SET balance**=**balance**+1000 **WHERE name**=**'曹操'**; **UPDATE** account **SET balance**=**balance**-1000 **WHERE name**=**'刘备'**; **COMMIT**;*# 提交事务* |

# 12 触发器

触发器是一类特殊的事务，可以监视某种数据操作(insert|update|delete)，并触发相关操作(insert|update|delete)。

## 使用场合

有时碰到表中某些数据改变，希望同时引起改变其他数据改变的需求，利用触发器可以满足这样的需求。例如商城中的有客户下订单后，库存量、购买人数等这些数据需要跟着改变。



作用：在表中某些特定数据变化时自动完成某些查询，运用触发器不仅可以简化程序，并且可以增加程序的灵活性。

## 触发器操作

查看触发器：**SHOW TRIGGERS**;

删除触发器：**DROP TRIGGER** 触发器名;

## 创建触发器语法的四要素

监视地点(table表)

监视事件(insert | update | delete)

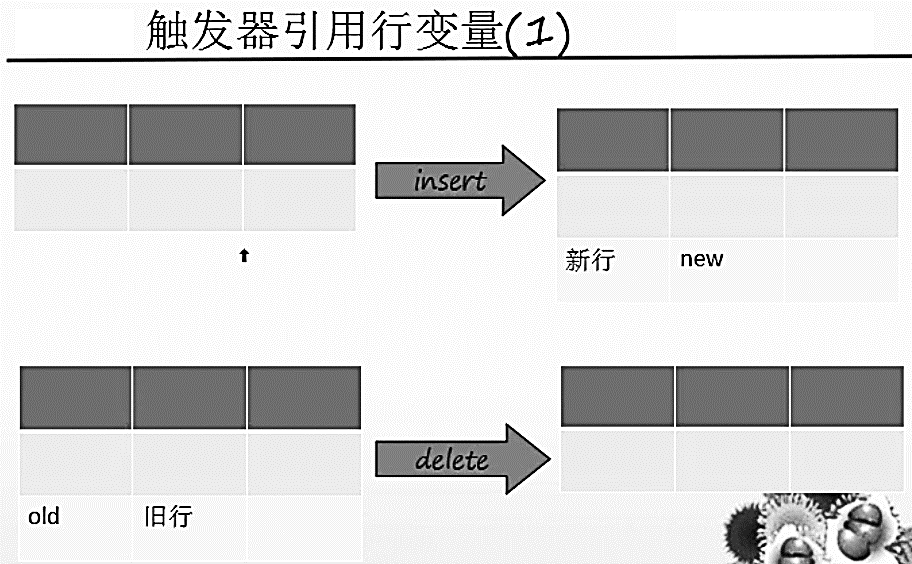
触发时间(before | after)

触发事件(insert | update | delete)

## 创建触发器语法

|  |
| --- |
| **CREATE TRIGGER** 触发器名  **BEFORE 或 AFTER** *# 触发时间*  **INSERT 或UPDATE 或 DELETE** *# 监视事件*  **ON 表名** *# 监视地点*  **FOR EACH ROW** *#在mysql中必须写，行级触发器，在oracle可以不写，表示语句级触发器*  **BEGIN** *# 开始触发* sql语句1 sql语句2 ……  **END** *# 结束触发* |

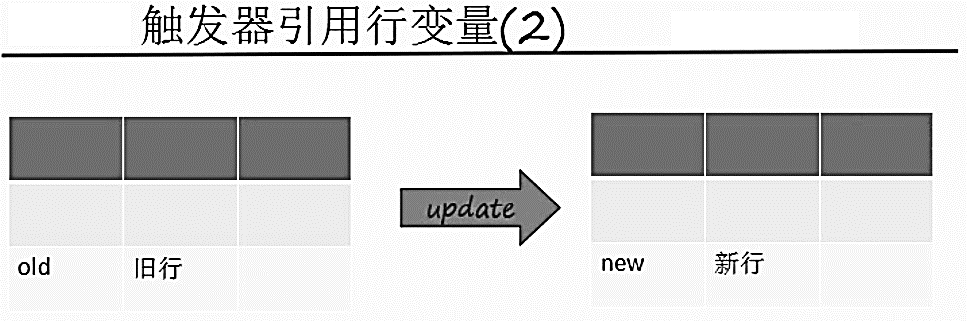
## 触发器变量



**监控insert行为时，引用变量用new**

**监控delete行为时，引用变量用old**

|  |
| --- |
| (1)订单表插入数据时，触发商品表对应的数据修改的触发器。  **CREATE TRIGGER** tr1 **AFTER INSERT ON** orders **FOR EACH ROW BEGIN  UPDATE** goods **SET num**=**num**-*new*.**much WHERE gid**=*new*.**gid**; **END**;  改进版，当购买数量大于库存数量是，默认为库存数量,防止爆仓。  **CREATE TRIGGER** tr4 **BEFORE INSERT ON** orders **FOR EACH ROW BEGIN  DECLARE** rnum **SMALLINT UNSIGNED DEFAULT** 0;  **SELECT num INTO** rnum **FROM** goods **WHERE gid**=*new*.**gid**;  **IF** *new*.**much**>rnum **THEN  SET** *new*.**much** = rnum;  **END IF**;   **UPDATE** goods **SET num**=**num**-*new*.**much WHERE gid**=*new*.**gid**; **END**;  (2)订单表删除数据时，触发商品表对应的数据修改的触发器（实际中订单只能失效，不能删除）。  **CREATE TRIGGER** tr2 **AFTER DELETE ON** orders **FOR EACH ROW BEGIN  UPDATE** goods **SET num**=**num**+*old*.**much WHERE gid**=*old*.**gid**; **END**; |



**监控update行为时，引用变量update前用old，update后用new。**

|  |
| --- |
| (3)修改订单表数据时，触发商品表对应的数据修改的触发器。  **CREATE TRIGGER** tr3 **BEFORE UPDATE ON** orders **FOR EACH ROW BEGIN  UPDATE** goods **SET num**=**num**+*old*.**much**-*new*.**much WHERE gid**=*new*.**gid**; **END**; |

# 13 存储过程

把若干条sql语句封装起来并起个名字，在过程中把数据存储到数据库中。

## 创建存储过程语法

定义语法：

|  |
| --- |
| **CREATE PROCEDURE** *名称*()  **BEGIN** *# sql语句* **END**; |

调用存储过程：**CALL** *存储过程名字*();

查看存储过程：**SHOW PROCEDURE STATUS**;

删除存储过程：**DROP PROCEDURE***存储过程名字*;

存储过程是可以编程的，意味着可以使用变量、表达式、控制结构来完成复杂的功能。

## 声明变量

**DECLARE** *变量名 变量类型 [default 默认值]*

注意：声明变量必须在begin和end之间声明。

变量可以参与sql语句的运算。

**SET** 变量名 := 表达式

|  |
| --- |
| **CREATE PROCEDURE** *test1*()  **BEGIN  DECLARE** leng **INT DEFAULT** 0;  **DECLARE** widch **INT DEFAULT** 0;  **SET** leng := 5;  **SET** widch := 6;  **SELECT** leng\*widch;  **END**; |

## 参数

参数分为in、 out、 inout类型。in表示输入类型，out表示输出类型，inout表示输入输出类型。

|  |
| --- |
| (1)in和out类型  **CREATE PROCEDURE** *cuArea*(**in** r **INT**, **OUT** area **INT**)  **BEGIN  SET** area:=0;*# 如果输出area参与运算时必须设置area的初始值，因为null参与运算的值都为null* **SET** area := 3.14\*r\*r;  **END**;  调用： **CALL** *cuArea*(10,*@area*); **SELECT** @area;*# 结果：314*  (2)inout类型  **CREATE PROCEDURE** *add\_1*(**INOUT** v **INT**)  **BEGIN  SET** v := v + 1;  **END**;  *# 定义变量和调用* **SET** *@v* := 1; **CALL** *add\_1*(@v); **SELECT** @v; *# 结果：2* |

## if条件控制结构

**IF** 条件1 **THEN**  
**ELSEIF**条件2 **THEN**

**......**  
**ELSE**  
**END IF**;

其中ELSEIF和ELSE可以没有。

|  |
| --- |
| **CREATE PROCEDURE** *compare*(v1 **INT**,v2 **INT**)  **BEGIN  IF** v1>v2 **THEN  SELECT** *concat*(v1,**'大于'**,v2);  **ELSEIF** v1<v2 **THEN  SELECT** *concat*(v1,**'小于'**,v2);  **ELSE  SELECT** *concat*(v1,**'等于'**,v2);  **END IF**;  **END**; |

## case选择控制结构

**CASE** 变量  
 **WHEN** 值 **THEN 表达式**;  
 ......  
 **ELSE 不满足条件最后的默认结果**;  
 **END CASE**;

注意：else 可以省略。

|  |
| --- |
| **CREATE PROCEDURE** *cs*()  **BEGIN  DECLARE** v **INT**;  **SET** v := *floor*(*rand*()\*10);  **CASE** v  **WHEN** 0 **THEN SELECT '星期日'**;  **WHEN** 1 **THEN SELECT '星期一'**;  **WHEN** 2 **THEN SELECT '星期二'**;  **WHEN** 3 **THEN SELECT '星期三'**;  **WHEN** 4 **THEN SELECT '星期四'**;  **WHEN** 5 **THEN SELECT '星期五'**;  **WHEN** 6 **THEN SELECT '星期六'**;  **ELSE SELECT 'unknown day'**;  **END CASE**;  **END**; |

## while循环结构

**WHILE** 条件 **DO**  执行语句  
**END WHILE**;  
注意：避免死循环

|  |
| --- |
| **CREATE PROCEDURE** *cusum* (v **INT**)  **BEGIN  DECLARE** s **INT DEFAULT** 0;  **WHILE** v>0 **DO  SET** s := s + v;  **SET** v := v-1;  **END WHILE**;  **SELECT** s;  **END**;  调用： **CALL** *cusum*(100); *# 结果：5050* |

## repeat循环结构

**REPEAT**  执行语句......  
 **UNTIL** 条件 **END REPEAT**;

|  |
| --- |
| **CREATE PROCEDURE** *cuSum2*(v **INT**)  **BEGIN  DECLARE** sum **INT DEFAULT** 0;  **REPEAT  SET** sum := sum+v;  **SET** v:=v-1;  **UNTIL** v<=0 **END REPEAT**;  **SELECT** sum;  **END**;  *# 调用* **CALL** *cuSum2*(100); *# 结果：5050* |

# 14 游标

一条sql的select语句取出对应的n条资源，取出资源的接口(句柄)就是游标，沿着游标，每次只取出一行，取出的行可以任意的逻辑控制了，而select没有这种功能。

游标语法:

|  |
| --- |
| **DECLARE** 游标名 **CURSOR FOR select语句**; *# (1)声明游标*  **DECLARE EXIT HANDLER FOR NOT FOUND 表达式** ;*# 设置触发边界标志*  **OPEN**游标名; *# (2)打开游标* **FETCH**游标名**INTO** 变量1, 变量2......; *# (3)取值* **CLOSE**游标名; *# (4)关闭游标* |

用循环读取游标数据，结束条件是判断是否去到最后一条数据（事先计算出来的总数）。

|  |
| --- |
| **CREATE PROCEDURE** *cursor1*()  **BEGIN  DECLARE** tmp\_name **VARCHAR**(20);  **DECLARE** tme\_num **INT**;  **DECLARE** cnt **INT**;  **DECLARE** i **INT DEFAULT** 0;   **DECLARE** get\_goods **CURSOR FOR SELECT name**,**num FROM** goods;  **OPEN** get\_goods;  **SELECT** *count*(*\**) **INTO** cnt **FROM** goods;  **WHILE** i<cnt **DO  SET** i:=i+1;  **FETCH** get\_goods **INTO** tmp\_name,tme\_num;*# 取一条数据* **SELECT** tmp\_name,tme\_num;  **END WHILE**;   **CLOSE** get\_goods;  **END**;  *# 调用* **CALL** *cursor1*(); |

在mysql的cursor中可以用declare exit或continue handler for not fond来操作越界标志。类似于js中的事件，当读取游标完毕则触发该事件。其中exit和continue的区别是是否执行后面的sql语句。

|  |
| --- |
| (1)触发越界后执行exit，不执行后面的sql语句  **CREATE PROCEDURE** *cursor2*()  **BEGIN  DECLARE** tmp\_name **VARCHAR**(20);  **DECLARE** tme\_num **INT**;  **DECLARE** isEnd **BOOL DEFAULT FALSE**;   **DECLARE** get\_goods **CURSOR FOR SELECT name**,**num FROM** goods;  **DECLARE EXIT HANDLER FOR NOT FOUND SET** isEnd:=**TRUE** ;*# 设置触发边界标志* **OPEN** get\_goods;   **WHILE** !isEnd **DO  FETCH** get\_goods **INTO** tmp\_name,tme\_num;*# 取一条数据* **SELECT** tmp\_name,tme\_num;  **END WHILE**;   **CLOSE** get\_goods;  **END**;  *# 调用* **CALL** *cursor2*();  (2)触发越界后执行continue，继续后面的sql语句。  **CREATE PROCEDURE** *cursor3*()  **BEGIN  DECLARE** tmp\_name **VARCHAR**(20);  **DECLARE** tme\_num **INT**;  **DECLARE** isEnd **BOOL DEFAULT FALSE**;   **DECLARE** get\_goods **CURSOR FOR SELECT name**,**num FROM** goods;  **DECLARE CONTINUE HANDLER FOR NOT FOUND SET** isEnd:=**TRUE** ;*# 设置触发边界标志* **OPEN** get\_goods;   **FETCH** get\_goods **INTO** tmp\_name,tme\_num;*# 进入循环前先取一条数据* **WHILE** !isEnd **DO  SELECT** tmp\_name,tme\_num;  **FETCH** get\_goods **INTO** tmp\_name,tme\_num;*# 取一条数据* **END WHILE**;   **CLOSE** get\_goods;  **END**;  *# 调用* **CALL** *cursor3*(); |

# 15 mysql用户与权限管理

## 15.1用户连接mysql服务器

【用户】<----------->【mysql服务器】认证用户依据有3个参数：

你从哪里来？host

你是谁？user

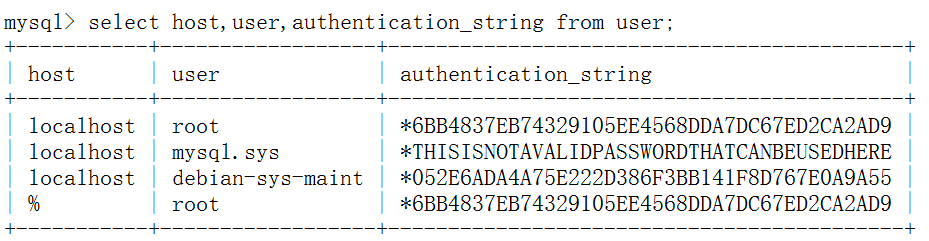
你的密码是多少？password

连接数据库：mysql -h192.168.8.102 -uroot -p123456;

查看当前登录用户：**SELECT** *user*();

可以通过库名mysql里的user表来查看有哪些用户可以登录。

|  |
| --- |
| **USE** mysql; **DESC user**;*# 查看表结构*  **SELECT host**,**user**,**password FROM user**; |



### 修改host域，使用IP能连接起来

|  |
| --- |
| **UPDATE user SET host**=**'192.168.8.101' WHERE host**=**'::1'**; **FLUSH PRIVILEGES**; *# 冲刷权限* |

### 修改用户密码

|  |
| --- |
| **UPDATE user SET password**=*password*(**'123456'**) **WHERE host**=**'192.168.8.101'**; **FLUSH PRIVILEGES**; *# 冲刷权限* |

## 15.2授权给用户

用户有没有权限执行操作(例如：insert、update、delete等)

### 新建一个用户并授权

**GRANT [***权限1, 权限2......***] ON** *数据库名.该库下的表名* **TO** *用户名@主机名* **IDENTIFIED BY** *密码*;

例如：授权给主机为192.168.8.n局域网内的用户root所有数据库权限

**GRANT ALL ON** *\**.*\** **TO** *root*@*'192.168.8.%'* **IDENTIFIED BY '123456'**;

常用权限

all，creat，drop，insert，update，delete，select......

针对某个库做授权

**GRANT ALL ON** *test*.*\** **TO** *root*@*'192.168.8.%'*;

**FLUSH PRIVILEGES**; *# 冲刷权限*

针对某个表授权

**GRANT select,update,insert ON** *test*.*goods* **TO** *root*@*'192.168.8.%'*;

**FLUSH PRIVILEGES**; *# 冲刷权限*

### 收回用户权限

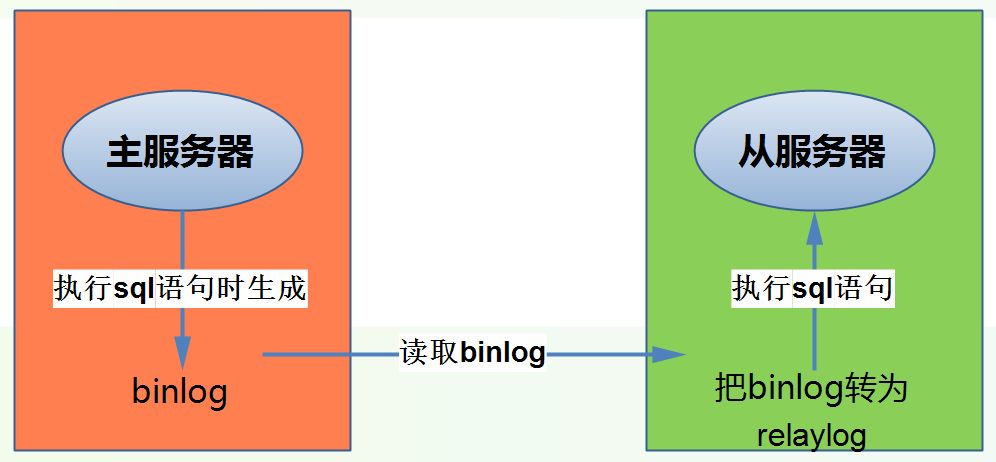
**REVOKE [***权限1, 权限2......***] ON** *数据库名.该库下的表名* **FROM** *用户名@主机名*;

例如：收回某用户的所有权限

**REVOKE ALL ON** *\**.*\** **FROM** *root*@*'192.168.8.%'*;

# 16 mysql主从复制

原理



配置：

◎主服务器配置binlog

◎从服务器配置relaylog

◎主服务器授权给从服务器一个帐号

◎从服务器连接主服务器

## 16.1 配置主服务器binlog

(1)编辑配置文件：sudo vim /etc/mysql/my.cnf

在[mysqld]字段下

①给主服务器起一个名字，注意是数字，一般为ip最后一个数字

server-id = 101

②开启logbin二进制日志，并给logbin起个名字

log-bin = mysql\_bin

③指定日志格式，分别有mixed（由系统决定row还是statement）、row（记录磁盘变化）、statement（记录执行语句），各有各的应用场景。

binlog-format = mixed

④重启mysql服务器

(2)给从服务器添加一个授权帐号

**GRANT REPLICATION CLIENT**,**REPLICATION SLAVE ON** *\**.*\** **TO** *slsrv*@*'192.168.8.%'* **IDENTIFIED BY '123456'**;

**FLUSH PRIVILEGES**; *# 冲刷权限*

(3)查看mysql是否具备充当主服务器条件

**SHOW MASTER STATUS**;

## 16.2 配置从服务器relaylog

(1)从服务器也开启binlog，防止数据丢了可以恢复。

编辑配置文件：sudo vim /etc/mysql/my.cnf

在[mysqld]字段下

①给主服务器起一个名字，防止多个主从混乱

server-id = 103

②开启logbin二进制日志，并给logbin起个名字

log-bin = mysql\_bin

③指定日志格式，分别有mixed（由系统决定row还是statement）、row（记录磁盘变化）、statement（记录执行语句），各有各的应用场景。

binlog-format = mixed

④开启relaylog日志，并给relaylog起个名字

relay-log=mysql\_relay

⑤重启mysql服务器sudo service mysql restart

(2)在从服务器通过语句指定要复制的主服务器(可以一主多从，不能多主)

|  |
| --- |
| **CHANGE MASTER TO  MASTER\_HOST**=**'192.168.8.101'**,  **MASTER\_USER** = **'repl'**,  **MASTER\_PASSWORD** = **'123456'**,  **MASTER\_LOG\_FILE** = **'mysql\_bin.000001'**, *# 用语句SHOW MASTER STATUS;查看*  **MASTER\_LOG\_POS** = 0; |

(3) 启动从服务器功能

**START SLAVE;**

(4)查看从服务器是否连接到主服务器

**SHOW SLAVE STATUS**;

如果有连接错误，

ping下能否有网络连接，

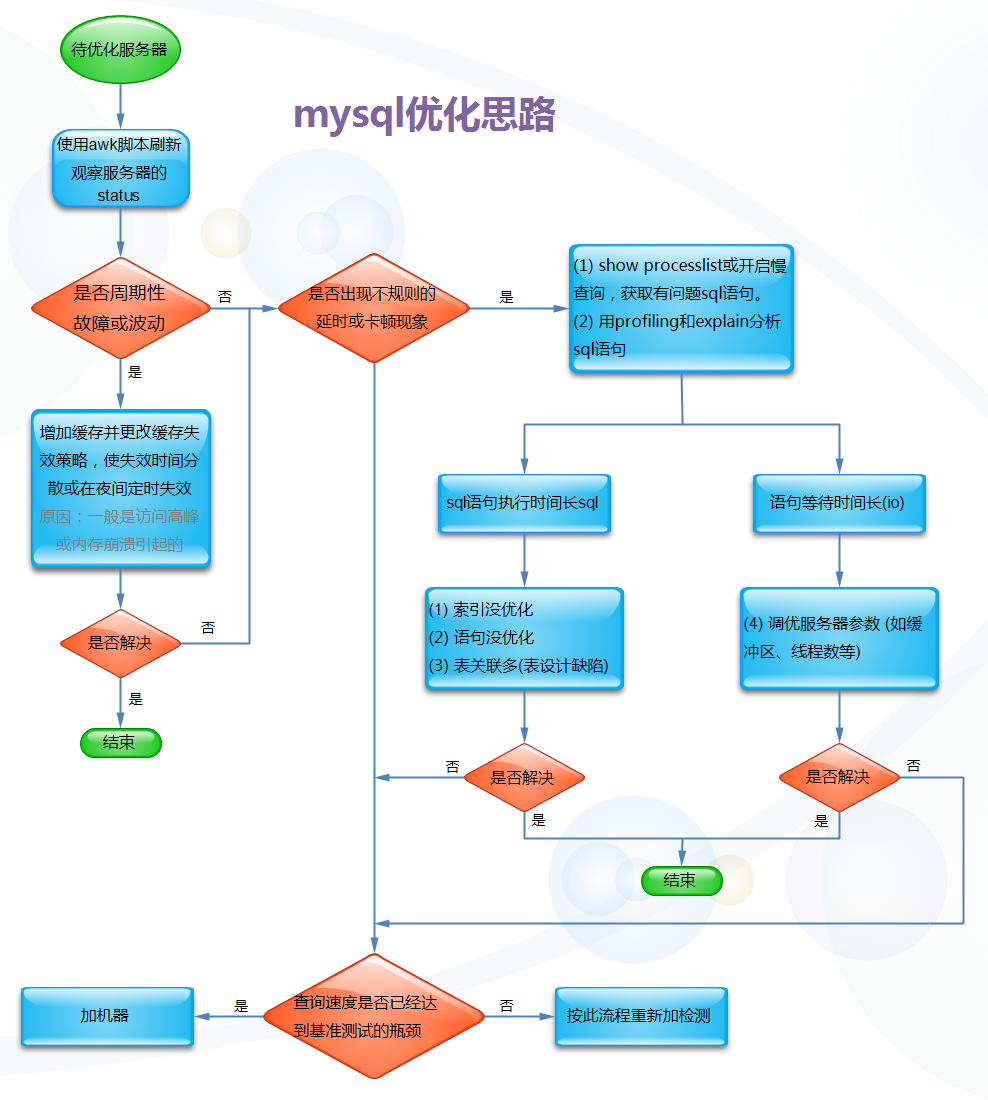
telnet 192.168.8.101:3306判断是否连接上，

如果连接不上，一般是防火墙问题，在终端停止防火墙service iptables stop，

stop slave后再start slave

注意：主从复制的时间间隔一般为毫秒级别，达到秒级别的使用时风险比较大。

# 17 mysql优化



## 17.1 查看服务器状态

在mysql命令行查看服务器状态：**SHOW STATUS** ;

在linux命令行查看服务器状态：**mysqladmin -uroot -p123456 ext**

得到三百多行服务器状态数据，主要看3个参数：

Queries: 查询次数

Threads\_connected: 线程连接数

Threads\_running: 线程运行数

在linux命令行获取上面三个参数值：mysqladmin -uroot -p123456 ext | awk **'/Queries/{q=$4} /Threads\_connected/{c=$4} /Threads\_running/{r=$4} END{printf "%-10d%-10d%-10d\n", q,c,d}'**

一定要结合业务去优化。

## 17.1 btree的左前缀规则

按f1、f2......建立的复合索引，在where条件中，按f1、f2......由左到右的顺序（当and时不用按顺序），索引才会发挥作用。

如果中间某列没有条件或like条件，导致后面的列，索引用不上。

索引也能用于排序和分组，因为分组要先排序后在计算。所以我们的order by或group如果能针对有顺序的表进行，可以避免临时表和文件排序。也就是说我们的order by或group按顺序使用索引的列，则可发挥索引的作用。

注意：复合索引必须针对实际业务场景来建立。

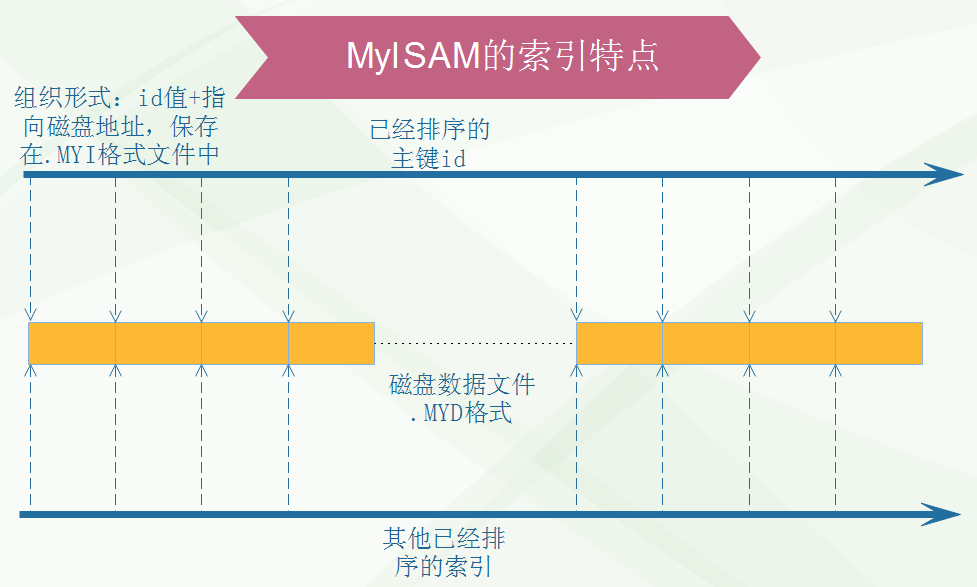
order by 会在内存生成临时表

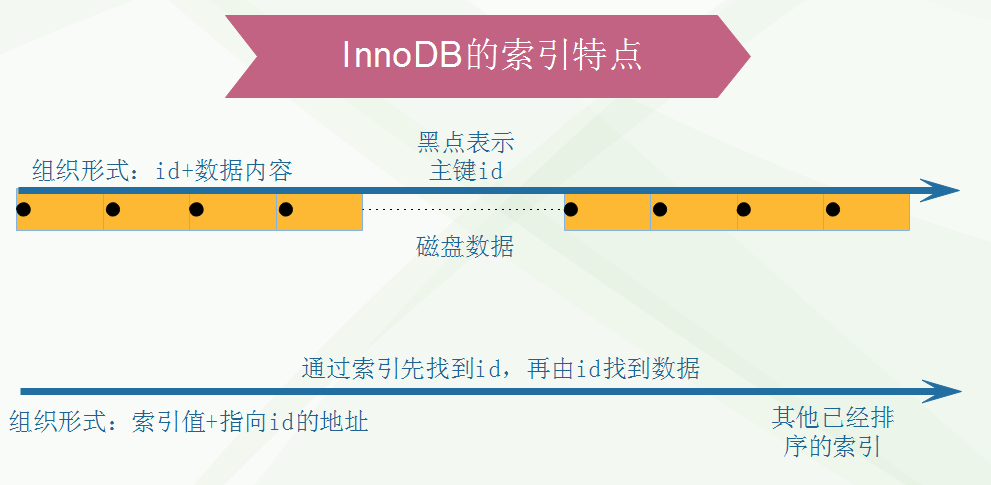
## 17.2 MyISAM和InnoDB的索引特点

打开性能分析选项：set profiling=1;

查看性能分析结果：show profiles;

MyISAM和InnoDB的索引特点，各自指向方式不同。





由图可知，MyISAM的索引指向的是数据在磁盘上的位置，而InnoDB的索引指向主键值的引用，是先指向主键id，然后再从id获取磁盘数据，也就是说InnoDB的索引是间接获取数据，中间多了从id获取数据的过程。

聚簇索引：索引和数据混合一起，如InnoDB的id和数据是混合在一起的。有个缺点，当数据文件比较大时，查询数据时不断翻越磁盘扇区的页，使用时间会比较长。

回行：从索引到获取磁盘数据的过程，查询时回行时比较耗时，如果能直接从索引中获取数据，将会省去回行过程，提高查询速度，因为索引在内存运行的。

索引覆盖：只在索引就能获取想要的数据。如果查询的数据能用到索引覆盖，速度是最快的。用explain 查询语句，看extra项目是否有Using index，如果有则使用到索引覆盖。

## 17.3 分页优化

(1)从业务上优化

实际业务场景中，翻到100页的非常少，可以限制在100内。

(2)从100w+1到100w+10这10行数据

前提是，id是按顺序排列，并且不删除数据。那么就可以用

id>($page-1)\*$n limit $n来分页。

(3)延时索引

把索引覆盖用到第二次查询上。

使用inner join连接查询，注意使用in子查询有陷阱。

每个12个小时mysql压力骤增，导致mysql垮掉

sql文件导入

|  |
| --- |
| use o2o;  set names utf8;  source D:/o2o.sql |

mysql周期性波动或故障解决方案

