MySQL必知必会+SQL初级语法整理

Author : Yunqiu Xu

`MySQL` `SQL` `Database`

Chapter 1 有关MySQL安装及数据库的创建

1.1 MySQL安装

- Win OS可选择Installer 进行安装；
- 默认connector为python3.4 , 可自定义为2.7

1.2 数据库创建及基本操作

```
CREATE DATABASE xyq; --创建新数据库
SHOW DATABASES; --查看已有数据库
CREATE DATABASE IF EXISTS xyq;--加入警告：如果xyq已经存在，则输出一个警告而非错误信息
SHOW WARNINGS / SHOW ERRORS; --查看警告
SHOW CREATE DATABASE xyq; --查看编码方式
CREATE DATABASE xyq CHARACTER SET utf8; --创建数据库时改变默认编码
ALTER xyq CHARACTER SET utf8;--修改已经存在的数据库的编码
DROP DATABASE xyq; --删除数据库
HELP ITEMS; --Seek help
```

- 完整格式：

```
CREATE {DATABASE|SCHEMA} --{}中的元素二选一
    [IF NOT EXISTS] --[]为可选
    db_name[DEFAULT]CHARACTER SET[=]charset_name;
```

1.3 SELECT 子句优先度

```
SELECT
FROM
WHERE
```

```
GROUP BY
HAVING
ORDER BY
LIMIT
```

Chapter 2 使用MySQL

```
USE xyq;--必须先用USE选择某个数据库才能用里面的数据
SHOW TABLES [FROM xyq]; --查看数据库中的表
SHOW COLUMNS [FROM table_name][FROM xyq];
```

Chapter 3 检索数据

3.1 SELECT

```
SELECT col_name FROM table_name;
SELECT * FROM table_name; --all columns
SELECT col1,col2,col3 FROM table1;
--SQL语句倾向不使用空格而是将语句分为多行
--SQL不区分大小写，但是习惯将SQL大写表示而instance小写表示
--语句用分号结尾
```

3.2 ORDER

```
SELECT col1
FROM table
ORDER BY col2 DESC, col3;--先按col2降序，再col3默认升序

--DESC只应用于其前面的col，多列降序为col1 DESC,col2 DESC...
--内建函数对区分大小写不敏感，需要DBA改变需求
```

3.3 LIMIT

```
--LIMIT位于ORDER子句后
LIMIT n; --no more than n rows
```

```
LIMIT n1,n2; --start from n1, no more than n2
```

Chapter 4 过滤数据

4.1 WHERE

```
SELECT name,price
FROM Products
WHERE price=3.5
ORDER BY xxx
LIMIT n;
```

4.2 AND&OR

```
WHERE id='433' AND price=5
WHERE (ITEM1 AND ITEM2) OR ITEM3 --优先级: ()>AND>OR
```

4.3 IN&NOT

```
WHERE place IN ("TIANJIN","HANGZHOU")
--IN功能与OR相同，更清楚直观，也可以包含其他SELECT语句

WHERE NOT id='250'
--NOT等价于<>
```

Chapter 5 通配符与正则表达式

- Wildcard：用来匹配值的一部分的特殊字符
- search pattern：字面值OR通配符组成的搜索标准

5.1 LIKE

```
SELECT id,name
```

```
FROM Products
WHERE name LIKE "Fish%"; -- %任意字符任意次数

-- _匹配单个字符
--[JM]% J or M开头的所有内容
-- [^JM]% 反义
```

5.2 REGEXP

- 注意下LIKE和正则的区别，之前的笔记里未提及，之后可以扩充下

Chapter 6 计算字段

- 意义：直接从DB中检索出已经格式化的数据

```
CONCAT()
SELECT CONCAT(name,'(',country,')') AS newcol
SELECT colA*colB AS colC --执行算术计算
```

Chapter 7 函数

7.1 文本处理

```
Left()/Right()--返回串左边/右边的字符
Length()--返回串长度
Locate()--找出串的一个子串
Lower()/Upper()--大小写
LTrim()/RTrim()--去除空格

Soundex() --读音近似匹配
    WHERE Soundex(a)=Soundex('sb') --返回a列中读音类似sb的
```

7.2 日期和时间处理

```
AddDate()--增加一个日期
AssTime()--增加一个时间
```

```
CurDate()--返回当前日期，时间则为Time
Date()--返回日期时间中的日期部分，同理Time，还有Day,Hour，Month等
DateDiff()--返回两个日期的天数差
DayOfWeek()--对于一个日期返回对应的星期几
Now()--返回当前的日期和时间

--日期格式统一为'yyyy-mm-dd'
--如果需要的是日期，尽量使用Date，尽可能少包含无关数据
```

- 举个栗子：

```
WHERE Date(column) BETWEEN 'date_a' AND 'date_b'
```

7.3 数值处理函数

```
ABS()/COS()/EXP()/SIN()/TAN()/SQRT()
PI() --圆周率
RAND() --random
```

Chapter 8 汇总与分组

8.1 aggregate function: 运行在行组上，计算并返回单个值的函数

```
AVG()/COUNT()/MAX()/MIN()/SUM()
```

- 几个栗子：

```
SELECT AVG(col) AS newcol FROM table;--查询某列的均值
SELECT COUNT(*) AS newcol FROM table;--查询行数

DISTINCT parameter--Aggregate different values
AVG(DISTINCT col) --忽略col中列值相同的部分
```

8.2 分组数据

- GROUP BY--分组

- GROUP BY 可以包含任意数目列或进行嵌套(在最后的分组中汇总)
- 除了聚集计算外，SELECT中每个列都必须在GB子句中给出
- NULL作为一个单独的分组返回
- GB在WHERE后ORDER前，使用HAVING过滤而不是WHERE
  HAVING--对组进行过滤
- WHERE没有分组这一概念，指定的是列而不是分组
- 目前遇到的所有需要使用WHERE的场合其实都可以用HAVING来替代

- 举个栗子：

```
SELECT a FROM b GROUP BY c HAVING d;
```

- 分组与排序的差别：
  - OB的输出为sorted，GB不一定sorted;
  - 任意列都可以使用OB，但GB只能使用选择列或表达式列;
  - OB为可选项，但如果与聚集函数一起使用列，必须使用GB;

Chapter 9 Subquery嵌套子查询

- 举个栗子:
  - 子查询1: 从OrderItems中选择产品ID为1234的order_num
  - 子查询2: 从Orders表中选择order_num为子查询1结果的用户ID
  - 母查询: 从Customers表中选择用户ID为子查询2结果的用户名及联系方式

```
SELECT cust_name,cust_contact
FROM Customers
WHERE cust_id IN(SELECT cust_id
    FROM Orders
    WHERE order_num IN(SELECT order_num
        FROM OrderItems
        WHERE prod_id='1234'));
```

- 注意子查询中的SELECT只能查询单列，检索多列会显示错误
- 再举一个栗子：将计算字段引入子查询

```
SELECT name,
(SELECT COUNT(*)
```

```
    FROM Orders
    WHERE Orders.ID=Customers.ID) AS
    new_col
    FROM Customers
    ORDER BY name;
```

Chapter 10 JOIN 联结

10.1 基本概念

- relational table
- primary key
- scale 可伸缩性
- JOIN: 在SELECT语句中关联表的一种机制
- JOIN的创建: 规定要联结的表+规定联结方式

10.2 JOIN的实例

```
SELECT cust_name,prod_name,prod_price
FROM Customers,Products
WHERE Customers.cust_id=Products.cust_id
```

- 要匹配的两列列名需要加前缀，防止混淆
- 此处WHERE用于过滤
  - 如果没有则是第一个表的每一行和第二个表的每一行直接配对, 不考虑逻辑上是否可以匹配, 输出形式为笛卡尔积
  - Cartesian Product: 无联结条件的表返回的结果,行数目为两个表行数之积

10.3 等值(内部)JOIN：基于两个表之间的相等测试

10.4 联结多个表

```
SELECT col1,col2,col3
FROM table1,table2,table3
```

```
WHERE table1.colx1=table2.colx2
AND table2.coly1=table3.coly2;
```

10.5 高级联结：给表起别名以简化语句

```
SELECT col AS newcol FROM table;
SELECT col FROM table FROM newtable;
```

- 自联结：给同一个表两个别名T1，T2，便于在SELECT语句中多次使用相同的表
  - 也可以通过子查询实现自联结，但比较麻烦
- 举个栗子: 找到contact为xyq的公司，并给出客户资料

```
SELECT t1.id,t1.name,t1.contact
FROM tables AS t1,tables AS t2
WHERE t1.name=t2.name
AND t2.contact="xyq";
```

10.6 高级联结：自然联结

- 使用SELECT *通配符
- 标准联结返回所有数据，相同的列多次出现;
- 自然联结排除多次出现，每个列只返回一次;

10.7 高级联结：外部联结，包含没有关联行的行

- 内部：`FROM 表1 INNER JOIN 表2 ON 联结条件`

```
SELECT T1.c1,T2.c2
FROM T1 INNER JOIN T2
ON T1.cx=T2.cy;
```

- 外部：`FROM 表1 LEFT OUTER JOIN 表2 ON 联结条件`

```
SELECT T1.c1,T2.c2
FROM T1 LEFT OUTER JOIN T2
ON T1.cx=T2.cy;
--LEFT指从T1中选择所有行，若选择T2则是RIGHT
```

```
--这里联结条件不用WHERE子句,而是ON子句
```

Chapter 11 组合查询：用UNION组合多条SELECT语句

`SELECT query1 UNION SELECT query2;`

- UNION中的每个查询需包含相同的列，表达式或聚集函数，但次序可以颠倒
- UNION默认取消重复的行，若不取消使用UNION ALL
- 只需要一条ORDER BY，在结尾

Chapter 12 MySQL全文本搜索功能FULLTEXT

12.1 启用全文本搜索支持

`CREATE TABLE new_table(列及所需数据格式) ENGINE=MyISAM`

- 注意这里不同的引擎后面会讲

12.2 进行全文本搜索`Match()Against()`

- 举个栗子: 指定表达式b来检索cola

```
WHERE MATCH(cola) AGAINST('b')
--搜索不区分大小写
--全文本搜索类似LIKE，但是可以对结果排序
```

12.3 布尔文本搜索

- 没有FULLTEXT索引也可以使用，但性能降低！

```
WHERE MATCH(a) Against('heavy' IN BOOLEAN MODE)
```

Chapter 13 数据管理

13.1 插入数据

```
INSERT INTO tablename(col1, col2,...,coln)
VALUES(value1, NULL,...,value n);
--给出列名并赋值比较繁琐，但更安全
--不需要给所有列提供值，且可以交换顺序
--插入部分行:空值填入NULL
```

- 再举一个栗子：插入检索出的数据

```
INSERT INTO tablename(col1,col2,col3)
SELECT col1,col2,col3 FROM oldtable;
--新表和SELECT中的列名不一定一致，只是查找出来并填充而已
```

13.2 更新和删除数据

13.2.1 更新数据`UPDATE-SET`

```
UPDATE 要更新的表
SET 要更新的列及新值
WHERE 过滤条件;
```

- 注意不要省略WHERE，否则会变成更新所有行
- 举个栗子：将cust_id为12345的客户名设置为xyq, 联系方式设置为12345678

```
UPDATE Customers
SET cust_name='xyq',cust_contact='12345678'
WHERE cust_id='12345';
```

13.2.2 删除数据：删除表的内容而非表本身

```
DELETE FROM 表
WHERE 过滤条件;
```

- DELETE 删除的是某行而非某列，删除指定列需要使用UPDATE
- 不加WHERE为删除所有行，但不是删除表本身(DROP)

- 建议：更新或删除前使用SELECT测试，以免WHERE过滤错误

13.3 创建和操作表

13.3.1 创建表：`表名+列名&定义+ENGINE类型`

```
CREATE TABLE new_table
(

col1 CHAR(10)      NULL,     --可以添加空值NULL
col2 CHAR(20)      NOT NULL, --不可以添加空值
col3 INTEGER       NOT NULL,
col4 DECIMAL(8,2) ,
col5 INTEGER       NOT NULL    DEFAULT 1, --指定默认值为1
PRIMARY KEY(col2, col3), --指定主键，不允许使用空值
)ENGINE=InnoDB; --指定ENGINE
```

- 注意，默认值只支持常量，不允许使用函数

13.3.2 几种ENGINE

- InnoDB事务处理引擎，不支持全文本搜索;
- MEMORY在功能上等同MyISAM,但数据存储在内存中，速度快，适合临时表;
- MyISAM(默认)支持全文本搜索，但不能进行事务处理;

13.3.3 更新表`ALTER TABLE`

```
ALTER TABLE Vendors ADD abc CHAR(20); --增加一个新列
ALTER TABLE Vendors DROP COLUMN abc ; --删除某列
```

- ALTER TABLE的另一种用法：定义外键

```
ALTER TABLE Vendors
ADD CONSTRAINT colx
FOREIGN KEY (col_id) REFERENCES Orders(order_id)
```

13.3.4 删除表

```
DROP TABLE 表名; --没有撤销，永久删除
```

13.3.5 重命名表

```
RENAME TABLE old_table TO new_name;
```

Chapter 14 使用视图

- VIEW是虚拟的TABLE，不包含数据，只包含使用时动态检索数据的查询

14.1 视图的规则

- 唯一命名;
- 视图数目没有限制;
- 需要权限;
- 可以嵌套(从其他试图中检索数据的查询构建新视图)；
- 可以用ORDER BY 但如果检索数据的SELECT中也有ORDER BY，会被覆盖;
- 不能索引，但可以与表一起使用(编写一条联结表和视图的SELECT)

14.2 使用视图：

```
CREATE/DROP VIEW viewname AS SELECT...;
SHOW CREATE VIEW viewname; --查看创建视图的语句
```

- 使用时则直接`SELECT ...FROM viewname;`即可

14.3 视图的更新

```
DROP VIEW viewname; --删除视图
```

- 视图可以用UPDATE,INSERT,DELETE进行更新;

```
CREATE OR REPLACE VIEW --有则更新，无则创造，等价于先DROP再CREATE
```

- 存在分组联结子查询、并、聚集函数、DISTINCT时不能更新

14.4 视图的常见应用

- 重用SQL语句;
- 简化复杂的SQL操作(类似python里定义函数或class);
- 使用部分而非整个表，并可提供权限以保护数据;
- 更改数据格式和表示，视图可返回与原表格式/表示不同的数据
- 举个栗子：用视图重新格式化检索出的数据

```
CREATE VIEW viewname AS
SELECT CONCAT(RTRIM(name),'(',RTRIM(country),')')
AS new_col
FROM table1
ORDER BY name;

--调用：
SELECT * FROM viewname;
```

Chapter 15 使用存储过程

- 存储过程：为方便以后使用而保存的一条或多条SQL语句的集合

15.1 存储过程的优势与不足

- PROs:
  - 将处理封装在单元中，简化操作，提升性能;
  - 不需要反复建立一系列处理步骤，保证了数据一致性(步骤越多越易出错);
  - 简化对变动的管理
- CONs:
  - 不同DBMS语法有不同，且存储过程的编写比较复杂

15.2 执行存储过程`CALL`

```
CALL procedure_name(@para1,@para2,@para3);
```

- 执行这个存储过程即计算并返回参数1参数2参数3

15.3 创建存储过程

```
CREATE PROCEDURE 存储过程名(如果有para的话在这里加)
BEGIN
SELECT Avg(price) AS priceaverage
FROM products;  --定义函数priceaverage
END;
```

- 注意！！如果在命令行直接进行，为避免两个";"造成的换行错误，需要用DELIMITER 对换行进行转义：

```
DELIMITER //
--将'//'作为换行符
```

- 举个栗子：

```
CREATE PROCEDURE name()
BEGIN
SELECT Avg(price) AS priceaverage
FROM products;
END//
--这里才算是正确换行了,此时换行符为//
DELIMITER ; --最后将换行符更改为";"

CALL name(priceaverage); --调用存储，并返回参数
```

15.4 删除存储过程

```
DROP PROCEDURE name;
```

15.5 使用参数

- @para放在存储过程名后面的括号里，之后在CALL的时候也要加上参数;
- IN 指传递给存储过程，OUT指从存储过程传出一个值返回给使用者
- 所有的MySQL变量都以@开始;

- 若想要显示出参数，使用SELECT @某个参数名;
- 举个栗子：

```
--先创建存储过程
CREATE PROCEDURE productpricing(
OUT p1 DECIMAL(8,2),
OUT ph DECIMAL(8,2),
OUT pa DECIMAL(8,2))
BEGIN
SELECT Min(prod_price)
INTO p1 --最低价格
FROM products;
SELECT Max(prod_price)
INTO ph --最高价格
FROM products;
SELECT Avg(prod_price)
INTO pa --均价
FROM products;
END;

--再调用存储过程:
CALL productpricing(@pricelow,@pricehigh,@priceaverage);
SELECT @priceaverage; --显示检索出的平均价格
SELECT @pricelow,@pricehigh;
```

- 再举一个栗子：展示`IN/OUT`参数

```
CREATE PROCEDURE ordertotal(
IN onumber INT, --接受订单号
OUT ototal DECIMAL(8,2) --返回订单的合计
)
BEGIN
SELECT SUM(item_price*quantity)
FROM orderitems
WHERE order_num=onumber
INTO ototal;
END;

--调用存储过程：
CALL ordertotal(20005,@total);--传入订单号，返回合计
SELECT @total; --显示合计
```

15.6 检查存储过程

```
SHOW CREATE PROCEDURE procedure_name;
-- --添加注释
```

Chapter 16 使用游标curser

- 定义: 存储在服务器上的查询，不是SELECT语句，而是指该语句检索出来的结果集;
- 存储游标后可以根据需要滚动或是浏览其中的数据;
- 主要用于交互式应用;

16.1 使用游标的步骤：

- 声明游标，定义要使用的SELECT 语句;
- 声明后打开游标以供使用(用刚定义的SELECT将数据世纪检索出来);
- 对于填有数据的游标，根据需要检索各行;
- 结束使用时关闭游标;

16.2 创建游标

```
CREATE PROCEDURE processorders()
BEGIN
DECLARE abc CURSER
FOR
SELECT xyq FROM fuck;
END;
```

- abc为游标名,存储过程处理完成后游标就消失(只局限于存储过程, 感觉有些类似局部变量)

```
OPEN/CLOSE abc; --打开/关闭游标
```

16.3 使用游标中的数据

```
OPEN abc;
FETCH abc INTO o; --使用FETCH访问游标的每一行
CLOSE abc;
```

- 举个栗子：循环检索数据，从第一行到最后一行;

```
CREATE PROCEDURE processorders()
BEGIN

--Declare local variables
DECLARE done BOOLEAN DEFAULT 0;
DECLARE o INT;

--Declare the cursor
DECLARE ordernumbers CURSOR
FOR
SELECT order_num FROM orders;

--Declare continue handler
--02000是一个未找到条件，当REPEAT不再继续时，该条件为真，结束loop
DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET done=1;

--Open the cursor
OPEN ordernumbers;

--Loop through all rows
REPEAT --反复执行直到done为TRUE
--Get order number
FETCH ordernumbers INTO o;
    --在FETCH 和UNTIL之间可以插入任意操作
    --增加一个名为t的变量存储每个订单的合计
    CALL ordertotal(o,1,t);
    --相当于创建了一个新表ordertotals保存存储生成的结果
    INSERT INTO ordertotals(order_num,total)
    VALUES(o,t);
UNTIL done END REPAET; --END OF LOOP

--Close the cursor
CLOSE ordernumbers;
END;

--查看该表
SELECT * FROM ordertotals;
```

Chapter 17 使用触发器TRIGGER

17.1 基本概念

- TRIGGER：在MySQL响应一些语句(DELETE,INSERT,UPDATE)时自动执行的语句;
- TRIGGER也可以位于BEGIN 和END之间，例如：
  - 每添加一个客户是都检查电话号码格式是否正确;
  - 每订购一个商品是都从库存数量中减去订购数量

17.2 创建触发器：`唯一的触发器名+关联表+响应的行为+何时执行`

- 每个表的触发器名称唯一，每个数据库中不一定唯一，但建议从始至终唯一

```
CREATE TRIGGER abc AFTER INSERT ON products
FOR EACH ROW SELECT 'Product added';
--创建触发器abc，在INSERT语句成功后执行
--每个新插入的行显示一次字符串'Product added'
```

- 只有表才支持触发器，临时表和视图不支持
- 每个表最多支持六个触发器，在INSERT/UPDATE/DELETE之前和之后
- 单一触发器不能与多个事件/表相关联

17.3 删除触发器`DROP TRIGGER`

17.4 使用触发器

17.4.1 INSERT 触发器

- 可引用一个名为NEW的虚拟表来访问被插入的行
- 若该触发器是BEFORE INSERT的，可以对NEW进行更新(更改被插入的值)
- AFTER INSERT：插入后才生成新订单号，防止插入不成功但是生成了新订单号
- 举个栗子：

```
--创建一个名为neworder的TRIGGER
--每当插入一个新订单到orders，生成一个新订单号保存到order_num中
--触发器从NEW.order_num取得并返回这个值
```

```
CREATE TRIGGER neworder AFTER INSERT ON orders
FOR EACH ROW SELECT NEW.order_num;

--测试
INSERT INTO orders(order_date,cust_id)
VALUES(Now(),10001);
```

17.4.2 DELETE触发器

- 可以引用一个名为OLD的虚拟表访问被删除的行
- OLD与NEW不同，数据全是只读的，不能更新
- 举个栗子：
  - 在任意订单被删除前执行此触发器
  - 使用一条INSERT语句将OLD中的值(即要被删除的订单)存储于名为achieve_orders表中
  - BEFORE DELECT优点：如果订单不能存档的话可以放弃DELETE

```
CREATE TRIGGER delorder BEFORE DELETE ON orders
FOR EACH ROW
BEGIN
INSERT INTO achieve_orders(order_num,cust_id)
VALUES(OLD.order_num,OLD.cust_id);
END;
```

17.4.3 UPDATE触发器

- 可以引用OLD访问以前的值，引用NEW访问更新后的值
- BEFORE UPDATE类型的NEW可以更新，OLD为只读，什么时候都不能更新

Chapter 18 管理事务处理 `COMMIT/ROLLBACK`

18.1 基本概念

- transaction processin: 维护数据库完整性，要么完全执行操作，要么完全不执行
- transaction:指一组SQL语句
- rollback:撤销指定SQL语句的过程
- commit将为存储的SQL语句结果写入数据库表

- savepoint：事务处理中的临时占位符，可以在这里发动回退而非回退整个事务处理

18.2 事务处理的创建

18.2.1 ROLLBACK

```
SELECT * FROM ordertotals; --首先执行一条SELECT表示本表不为空

START TRANSACTION --标识事务的开始
DELETE FROM ordertotals;--事务处理：删除所有行
SELECT * FROM ordertotals;--验证ordertotals为空
ROLLBACK;--回退START TRANSACTION后的所有语句

SELECT * FROM ordertotals; --显示该表不为空
```

- 注意：ROLLBACK 只能在一个事务处理中使用，即在执行一条START TRANSACTION 之后

18.2.2 COMMIT：一般SQL都是隐含提交(自动提交)，但事务处理中需要明确提交

```
START TRANSACTION;
SQL Queries;--若其中一条语句失败，则整个事务不会被提交
COMMIT;
```

18.2.3 保留点：出了问题退回到保留点即可

```
SAVEPOINT delete1;
ROLLBACK TO delete1; --这里出了问题退回到delete1
```

- 保留点越多越好

18.2.4 更改默认的提交行为

```
SET autocommit=0; --默认使用autocommit，此处设置为假
```

Chapter 19 全球化和本地化：处理不同字符集和语言

19.1 基本概念

- 字符集：字母和符号的集合
- 编码：某个字符集成员的内部表示
- 校对：规定字符如何比较的指令

19.2 使用字符集和校对顺序

```
SHOW CHARACTER SET; --显示所有可用字符集及每个字符集的描述和默认校对
SHOW COLLATION;--显示所有可用校对
```

- 举个栗子：

```
--创建数据库时指定默认的字符集和校对
SHOW VARIABLES LIKE 'character%';
SHOW VARIABLES LIKE 'collation%';

--给表指定字符集和校对
--创建一个包含两列的表，并指定一个字符集和校对顺序
CREATE TABLE mytable
(
col1 INT,
col2 VARCHAR(10)
)DEFAULT CHARACTER SET hebrew
COLLATE hebrew_general_ci;
```

- 指定和校对按如下顺序：
  - 如果指定CHARACTER SET 和COLLATE两者，则使用这些值;
  - 只指定CHARACTER SET，使用该字符集和默认校对;
  - 都不指定，使用数据库默认

Chapter 20 安全管理

20.1 访问控制

- 实际操作时尽可能少用root，而是创建一系列账号供不同人使用

```
USE mysql; --用户账号与信息储存在mysql数据库内
SELECT user FROM user;--user表的user列包含用户名信息
```

20.2 创建/更新/删除用户

```
CREATE USER username IDENTIFIED BY 'password';
RENAME USER oldname TO newname;
DROP USER username;
```

20.3 设置访问权限

```
SHOW GRANTS FOR username; --查看某个用户名的访问权限
```

- 设置权限需要提供：`要授予的权限+被授予权限的数据库或表+用户名`

```
GRANT SELECT ON world.* TO bitch;
--允许bitch在world数据库的所有表使用查询
REVOKE SELECT ON world.* TO bitch;--撤销权限
--可以授予或撤销的权限可以参考P212
```

- 再举个栗子：设置权限为'允许INSERT'

```
GRANT SELECT,INSERT ON database.* TO username;
```

20.4 更改password

```
SET PASSWORD FOR bitch = Password('new password')
```

Chapter 21 数据库维护及性能优化

21.1 备份数据

```
mysqldump --命令行程序，将数据库内容转存到外部文件
mysqlhotcopy --命令行程序，从一个数据库复制所有数据
BACKUP TABLE/SELECT INTO OUTFILE --转存所有数据到某个外部文件
```

21.2 数据库维护

```
ANALYZE TABLE table_name; --检查表键是否正确
CHECK TABLE tablename; --发现和修复问题
```

21.3 诊断启动问题

```
--help 帮助
--safe-mode 装载减去某些最佳配置的服务器
--verbose显示全文本信息
--version显示版本信息
```

21.4 查看错误文件

```
hostname.err --错误日志，位于data目录
hostname.log --查询日志，位于data目录
```

21.5 改善性能

- 遵循硬件协议;
- 生产DBMS运行在专用服务器中;
- 根据情况调整配置，可以使用`SHOW VARIABLES/STATUS`查看当前配置;
- `SHOW PROCESSLIST`显示所有活动进程，并用`KILL`命令终止其中过慢的
- `EXPLAIN`语句：让MySQL解释如何执行某条SELECT语句
- 存储过程执行效率要高于逐条执行;
- 避免检索多余需求的数据(avoid `SELECT *`);
- 导入数据时关闭自动提交;
- 索引数据库表以改善数据检索的性能;
- 索引会损害插入/删除/更新功能，对于仅仅收集但不常搜索的数据库避免索引;
- 和OR相比，多条`SELECT+UNION`性能更高;
- 使用FULLTEXT 代替LIKE;