**第一章 入门**

**一，安装与使用**

1，安装时“最后一步未响应”：

若以前安装过MySQL，重新安装时要完全删除以前安装的痕迹，且注意可能需要在安装配置中选择新的MySQL Server名称。

2，进入MySQL数据库：

命令行进入：mysql -u 用户名 -p 密码;

**二，基本命令**

-- 查看数据库：

show databases; （注意以;结束）

2, 创建数据库

create database DBName[字符集声明，整理声明];

3，删除数据库

drop database DBName;

4, 修改数据库

无法修改数据库名称，只能修改字符集和整理集

5，选择数据库：对某张表进行操作时，首先要进入该表所在的数据库

use DBName;

6, 查看数据库中表

show tables;

7，建表：

create table TbName(

    列1名称 列1类型,

    列2名称 列2类型,

    ...

);

类型：int-整形 varchar-字符串

【案例】

create table TbName(

    id int,

    title varchar(20),

    name varchar(10),

    content varchar(50)

);

8， 修改表名

rename table oldName to newName;

9, 删除表

drop table TbName;

10, 查看表结构

desc TbName;

11, 默认建表使用utf-8，而我们一般使用gbk，修改字符集

set names gbk;    (固定格式)

**三，表的增删改**

1，向表中增加一行

Insert into TbName (列1，列2，...)      //若缺省，默认插入所有列的值

values

(值1， 值2，...),

(值1， 值2，...)

...;

注意，插入时，除整形和浮点型不加单引号，其他所有类型均加单引号

【案例】

insert into msg

(id, title, name, content)

values

(1, '1标题', '张三', '三哥'),

(3,'3标题', '刘备', '双剑'),

(4,'4标题', '关羽', '青龙偃月刀'),

(5,'5标题', '张飞', '丈八蛇矛');

2, 查找

查找整张表；select \* from TbName;

查找部分列：select 列名1,列名2,... from TbName;

查找部分行的部分列：select 列名1,列名2,... from TbName where 条件;

3，修改某一行的部分列

update TbName set 列名1 = newValue1, 列名2 = newValue2 ... where 条件;

【案例】

update msg set id = 2, title = '蜀国' where name = '刘备';

4，删除某一行

delete from TbName where 条件;

【案例】

delete from msg where id = 1;

5，命令行中某一句打错，使用'\c'    直接返回

**第二章 基本类型**

**一，基本数据类型**

数值型：

    整形：

    tinyint/smallint/mediumint/int/bigint

    浮点型：

    float(d,m) / decimal(d, m)

字符型：

    char(m) / varchar(m) / text

日期时间型：

    date / time / datetime / year

**二，整形**

1，整形所占字节与存储范围的关系：



2，样式： XXXint(M) unsigned zerofill

     unsigned：无符号类型，默认有符号

     zerofill：0填充，与M一起使用，M表示宽度，当数值不足M位时，用0填充，补足M位（用zerofill修饰默认为unsigned无符号）

3，默认非空：在建表或新增列中跟 “not null default 默认值”

**三， 小数型**

float(M, D)：浮点小数，M表示总宽度，D表示小数个数

decimal(M, D)：定点小数，M,D类似

区别：decimal精度更高

**四，字符型**

1，char(M) : 定长型， M代表宽度（0~255）

 varchar(M)：变长型，M代表宽度（0~65535）（可容纳的字符数）

2，当实际字符串长度为n（n < M）, char类型实占m位字符，不足用空格补齐。在取出char型数据时，会将后面的空格去掉。

因此，若char型数据原本带有空格，会被清除。

  varchar则只占n+(1~2)位字符，取出时只取存入的字符数。

3，char取出速度较快

4，text：文本类型，可存放较大字符串，速度较慢，不用加默认值。

**五，时间日期型**

Year类型：1个字节 表示 1901-2155，[0000，表示错误时选择]

如果输入2位，‘00-69’表示2000-2069年

       ‘70-99’表示1970-1999年

如果记的麻烦，输入的时候，输入4位

Date类型：典型格式 1992-08-12

日期类型：‘1000-01-01’->‘9999-12-31’

Time类型：典型格式 hh:mm:ss

时间类型：‘-838:59:59’->‘838:59:59’

Datetime类型   典型格式：‘1989-05-06 14:32:08’

日期时间类型   范围：1000-01-01 00:00:00->9999-12-31 23:59-59

注意：

实际开发中很少用时间日期类型表示一个精确到秒的列，往往用时间戳表示；

时间戳是1970-01-01 00：00：00到当前的秒数，用int型表示，再格式化显示格式的时间

1. **查询语句**

**一，select五种子句**

* where：条件查询
* group by：分组统计
* having：筛选，对where查询后的结果进行进一步的筛选
* order by：排序
* limit：限制条目

注意:

（1）“ select 列1， 列2， ...  from tbname where 条件;”查询的结果本身就是一个子表，可以在上面进行相同的操作。

（2）对列的操作即相当于对变量进行操作，可直接在select后进行判断、运算等。

**二，where**

1,常用运算符：

<小于      >大于

!=或<>不等于   =等于

<=大于等于     >=小于等于

in在某集合内, in (值1，值2...值n),等于值1—n任意一个都行（in在某集合内）

between在某范围内, between 值1 and 值2，表示在值1和值2之间(between在某范围内)

2，逻辑运算符

与：and  &&        或：or ||          非：not ！

3，模糊查询

若想查找以“洛基亚”开头的所有商品：like

‘%’----通配任意字符

‘\_’----通配单个字符

select goods\_id from goods where good\_name likes '洛基亚%';

select goods\_id from goods where good\_name likes '洛基亚\_\_';

**三，gruop by：分组统计**

Max：求最大

Min：求最小

Sum：求总和

Avg：求平均

Count：求总个数

1，用法select c, avg(列1) from tbname group by  列名c；

以某个列名c进行分组，选择每一组中的统计情况；

2，注意若select选择了其他列（非统计值、非分组列），则往往采用该组中第一个出现行的列值。

**四，having,**

1，having与where异同点:

having与where类似,可筛选数据  where后的表达式怎么写,having就怎么写

where针对表中的列发挥作用,查询原来表中数据（选出的数据可以看成一个表）

having针对查询结果中的列发挥作用,筛选数据

#where 对表起作用  having是对结果进行筛选

#可将列名当成变量看，也可对列或统计结果取别名，用as

select cat\_id, sum(shop\_price) as sp from goods group by cat\_id having sp > 200;

【案例】

#stu(name, sub,score)

-- 查询所有同学的平均分

select avg(score) from stu group by name;

-- 查询所有同学的不及格科目数量

select sum(score < 60) from stu group by name;

-- 查询两门及两门以上不及格同学的平均分

select name, sum(score < 60) as gk, avg(score) from stu group by name having gk >= 2;

**五，order by：各种排序场合**

order by 列1 [desc/asc], 列2 [desc/asc]  ... ;

字段名“desc”声明降序，默认升序（asc）

**六，limit：语句最后输出，限制输出条目**

用法：limit [offset,] n;

offset：偏移量，可缺省

n：取出条目

若offset不写，相当于limit 0, n;

取出价格第4到第6高的商品

Select goods\_id,goods\_name,shop\_price from goods order by shop\_price desc limit 3,3;

五个子句的顺序：

where，group by，having，order by，limit;

**七，子查询**

1，where型：将内层的查询结果作为外层的查询条件

eg：查询最新商品（以goods\_id最大为最新）

select \* from goods where goods\_id = 最大的goods\_id；

最大的goods\_id：select max(goods\_id) from from goods;

select \* from goods where goods\_id = (select max(goods\_id) from goods);

2，from型：将内层的查询结果供外层再次查询，注意，将内层的查询结果看成临时表，“as 临时表名”；

eg：查询每个栏目下最新商品、

#先按栏目id升序排序，再按商品id降序排序

select goods\_id, cat\_id, goods\_name from goods order by cat\_id asc, goods\_id desc;

#从查询结果中选取最大商品id，group by选择第一个

select \* from (

select goods\_id, cat\_id, goods\_name from goods order by  cat\_id asc, goods\_id desc) as temp group by cat\_id;

3，exists型：把外层的查询结果代入内层，看内层是否成立，选择出成立的行。

eg：查询其下存在商品的栏目

select \* from category where exists (select  \* from goods where goods.cat\_id = category.cat\_id);

【案例】

商品表

goods (goods\_id,  cat\_id, goods\_sn, goods\_name,  click\_count, brand\_id, goods\_number,

market\_price, shop\_price, all\_time );

栏目表

category (cat\_id, cat\_name, parent\_id);

商标表

brand( brand\_id, brand\_name);

总结：

where 表达式 ：选出表中使表达式成立的一行

group by 字段（列）：按字段分组，通常与统计函数结合，统计以该字段为一组的某种情况，每组选出其中一种情况

Having 表达式：数据存储于表中，表在硬盘或内存中以文件形式存在， where查询出的结果可以看成一张表，临时存储于缓冲区中，having即是针对查询的结果进一步筛选。

Order by 字段：按字段进行排序，desc降序（默认），asc升序

可多字段进行排序；order by 字段1，字段2，...

limit [offset,]n ：限制条目

**八，Union**

1，union将两次或多次查询的结果连接起来，

结构：select \* from ta union select \* from ta

要求：每次查询的列数要求一致，推荐列名一致，若列名不一致，则以第一次查询的列名为准

2，若两次查询结果中存在完全一样的行（列名，列值相同），则相同行会进行合并（去重复）.

如果要求不去重复，使用union all

3，当查询子君中有order by,limit，需用（）单独包括每次子查询；

如果只有order by，则最后order by 放在union查询的结尾，否则每次单独order by排序最终进行联合，顺序也被打乱

若子句中order by 配合 limit使用，无需放在结尾，因为limit需要选择每次排序后的结果

#取第3个栏目价格前3高的商品和第4个栏目价格前2高的商品,union来实现

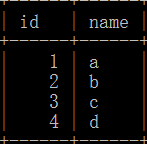
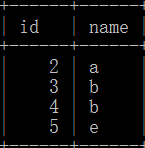
（Select good\_id, cat\_id, good\_name, price from goods where cat\_id = 3 order by price desc limit 3）

Union

（Select good\_id, cat\_id, good\_name, price from goods where cat\_id = 4 order by price desc limit 2）

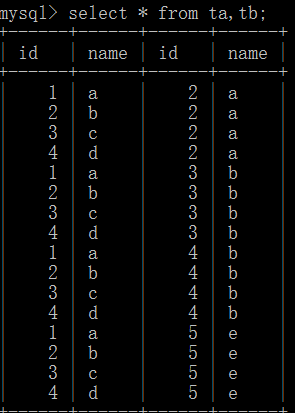
**九，左连接，右连接，内连接**

表A: 表B:

**表A×表B：笛卡尔积**

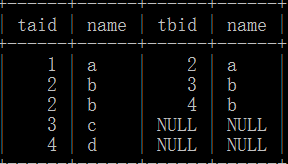
select \* from ta,tb;



**左连接**：ta left join tb on 匹配条件

获取左边表所有记录，以左边表为基准，去匹右边表记录，若在右边未找到匹配，则用null代替

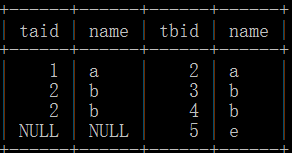
select ta.id as taid, ta.name, tb.id as tbid, tb.name from ta left join tb on ta.name = tb.name;

右连接：ta right join tb on 匹配条件

获取右边表所有记录，以右边表为基准，去匹配左边表，若未匹配，用null表示

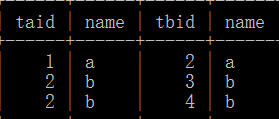
select ta.id as taid, ta.name, tb.id as tbid, tb.name from ta right join tb on ta.name = tb.name;

Tips：左右连接可互换，一般推荐使用左连接

内连接：ta inner join tb on 匹配条件，获取左右表中相互匹配的记录

select ta.id as taid, ta.name, tb.id as tbid, tb.name from ta inner join tb on ta.name = tb.name;

可用where子句替换

select ta.id as taid, ta.name, tb.id as tbid, tb.name from ta, tb where ta.name = tb.name;



#进行复杂sql查询时，没有想法时，首先可以进行切分

#1，选择出在规定时间进行比赛的队伍编号、比赛结果和时间

Select hostTeamID, macthResult, guestID, matchTime from Macth ;

#2，找出主队名字

Select hostTeamID, teamName, macthResult, guestID, matchTime from Macth left join Team on Macth.hostTeamID = Team.teamID;

#3，找出客队名称

Select hostTeamID, teamName, macthResult, guestID, teamName, matchTime

From

Macth left join Team

on Macth.hostTeamID = Team.teamID

Left join Team --继续左连接

on Macth.guestTeamID = Team.teamID

------由于两次左连接是同一张表，会导致无法分别teamName是哪张表，因此需要给表取别名。

Select hostTeamID, t1.teamName, macthResult, guestID, t2.teamName, matchTime

From

Macth left join Team as it

on Macth.hostTeamID = t1.teamID

Left join Team as t2 --继续左连接

on Macth.guestTeamID = t2.teamID

#4，选出正确时间

Select hostTeamID, t1.teamName, macthResult, guestID, t2.teamName, matchTime

From

Macth left join Team as it

on Macth.hostTeamID = t1.teamID

Left join Team as t2 --继续左连接

on Macth.guestTeamID = t2.teamID

where matchTime between ‘2006-6-1’ and ‘2006-7-1’;（matchTime >= ‘2006-6-1’ and matchTime <= ‘2006-7-1’）

**十，DDL：数据库定义语言**

1，建表

Create table 表名(

列名1 列类型 [列属性] ---->**列声明**

列名2 列类型 [列属性]

....

)charset = utf8/gbk...

列属性：not null、primary key、auto\_increment...

修改表结构

2，增加列

1. Alter table 表名 add **列声明**;

默认新生成的列在末尾

1. 可以用after来声明新增的列在哪个列后面

Alter table 表名 add **列声明** after 列名;

1. 同时可以用first声明新增的列在第一个

Alter table 表名 add **列声明** first;

3，修改列：可修改列名称、属性（不一定改名字）

Alter table 表名 change 旧列的名称 新的**列声明**

4，删除列： Alter table 表名 drop 列名;

**第四章**

**一，视图view**

1，视图是select查询结果形成的虚拟表

2，当某个查询结果出现的十分频繁时，可以将该查询结果建立视图保存起来，以便后面查询使用。

3，建立视图

Create view 视图名 as select语句;

建立视图后，对视图的操作就像对表一样

4，删除视图

Drop view 视图名;

5，视图的作用

1. 简化查询
2. 权限控制：可以封闭表的权限，仅开放视图权限
3. 大数据分表操作：可以将大数据量的视图进行分表，如将一个表分成四张视图

Create view t1 as select \* from t where (id % 4 + 1) = 1; ...

同时也可以将多张表进行合并；

Create view t as select \* from t1 union select \* from t2 ...

6，视图的修改

Alter view 视图名 as select语句;

7，视图和表的关系:

1. 视图是表的查询结果，表改变了，视图自然改变。
2. 当视图改变时，表不一定改变；

若视图中的列与表中的列是一一对应关系时，（视图中都是表中自然列）视图改变，表改变；若视图中的列是统计数据时，视图改变，表不改变。

1. 视图insert必须注意：视图必须包含表中没有默认值的列。 ？

8，视图algorithm属性

Create algorithm = merge/temptable/undefined view 视图名 as select语句;

Merge：默认，使用该方法时，当查询使用该视图时，将查询视图的语句和定义视图的语句进行合并再查。

Temptable：当查询该视图时，将视图当成一个临时表进行查询（类似from子查询）

Undefined：未定义，系统自动选择。

Merge：语法规则，将查询视图的where语句与定义视图的where语句进行合并，再进行查询。

#查询每个栏目下的最贵商品

#定义视图

Create view lux as select good\_id, cat\_id, good\_name, price from goods order by cat\_id asc, price desc;

#视图下查询

Select \* from lux gruop by cat\_id;

#最终执行：合并两句，结果会先执行gruop by导致错误

select good\_id, cat\_id, good\_name, price from goods gruop by cat\_id order by cat\_id asc, price desc;

2，采用temptable；

#定义视图

Create algorithm = temptable view lux as select good\_id, cat\_id, good\_name, price from goods order by cat\_id asc, price desc;

#视图下查询

Select \* from lux gruop by cat\_id;

#最终执行：作为临时表

Select \* from ( select good\_id, cat\_id, good\_name, price from goods order by cat\_id asc, price desc ) as lux gruop by cat\_id;

**二，字符集和校队集**

1，MySQL中字符集设置多样，其中有：

字段（列）字符集

表字符集

数据库字符集

服务器字符集

eg: create table t(

Id int charset gbk //字段字符集

)charset utf8; //表字符集

2，系统变量

-character\_set\_server：服务器字符集

-character\_set\_client：客户端数据使用的字符集

-character\_set\_connection：连接层字符集

-character\_set\_results：返回的查询结果字符集

-character\_set\_database：数据库使用的字符集

-character\_set\_system：系统元数据（字段等）使用字符集

3，数据存储流程

连接层utf8

Gbk->utf8->utf8

Gbk<-utf8<-utf8

服务器

中国(Utf8)

客户端

中国（gbk）

MySQL服务器在收到请求后，

1，先将character\_set\_client转换成character\_set\_connection

2，在进行数据库内部操作之前，再将character\_set\_connection转换为内部操作字符集：

- 使用每个数据字段的CHARACTER SET设定的字符集；  
  - 若不存在，则使用对应数据表的DEFAULT CHARACTER SET设定值(MySQL扩展，非SQL标准)；  
  - 若不存在，则使用对应数据库的DEFAULT CHARACTER SET设定值；  
  - 若不存在，则使用character\_set\_server设定值。

3，数据库操作结束后，将结果从内部操作字符集转换成character\_set\_results发送给连接层

4，连接层再将character\_set\_results转换成character\_set\_connection，再从character\_set\_connection转换为character\_set\_client。

4，设置

设置客户端字符集：set character\_set\_connection = ‘gbk’;

设置连接层字符集：set character\_set\_connection = ‘utf-8;

设置服务端字符集：set character\_set\_connection = ‘utf-8’;

若三者字符集都是N，可用set names N;;

5，内部字符集：字段，表，数据库，服务器字符集

6，什么时候会乱码？

1，client声明和事实不符，设置的客户端字符集set character\_set\_connection = ‘latin’;输入的却是中文

2，results声明和客户端不符，客户端为gbk，返回为latin

7，什么时候会丢失数据？

当connection或内部数据的字符集占用位数比client小时，会导致转换时丢失。

注意：建表时声明的字符集是属于内部字符集，不是客户端字符集，客户端字符集时客户端输入时（如命令行insert插入）所用字符集

8，校对集：字符集的排序方式

（1）当需要用order by对数据排序时，可显示指定排序方式

（2）用法：

Create table tname(...) charset utf8 collate utf8\_general\_ci;

注意：

使用的字符集必须是相应字符集合法的校对集

（3）校对集以字符序对应的字符集名称开头；以\_ci(表示大小写不敏感)、\_cs(表示大小写敏感)或\_bin(表示按编码值比较)结尾。

（4）查看字符集 show character set;

查看校对集 show collation;

**三，触发器：**

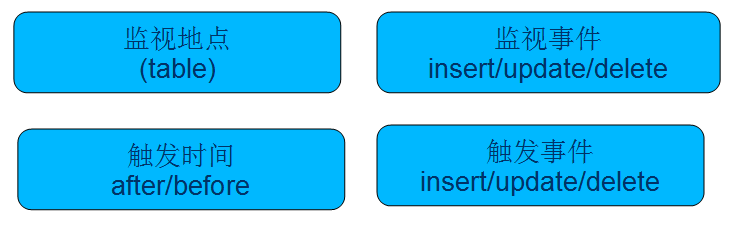
1，监视某张表的变化，当监视到发生对该表的操作时，会相应地触发某个操作。

数据库中某两个操作之间往往存在一定的逻辑联系，如订单的入库会导致库存表的改变。

2，监视的操作：insert、delete、update

触发的操作：insert、delete、update

3，触发器四要素：



监视地点：触发器监听的表

监视事件：触发器监听的动作

触发时间：after：监视事件发生后，再进行触发事件

before：在监视事件发生前，进行触发事件

触发事件：触发的操作

4，语法

1. 创建触发器

Create trigger 触发器名称

After/before insert/update/delete on 表名

For each row #MySQL中固定用法

Begin

sql语句; #insert/update/delete，可有多条sql触发语句， ... #每句以;结尾

...

End$

#由于触发语句以;结尾，MySQL中认为;即为语句的结束，因此可以使用

delimiter $ 设置$为结尾（没有分号）

1. 删除触发器

Drop trigger 触发器名称;

1. 查看触发器

Show triggers \G; #横置

5，如何在触发事件中引用监视事件中行的值

（1）对于insert监视事件，新增的行用new来表示，行中每一列的值用new.列名表示。

（2）对于delete事件，删除的行用old来表示，行中每一列的值用old.列名表示。

（3）对于update事件，修改前的行用old表示，old.列名表示修改前的行值

修改过的行用new表示，new.列名表示修改前的行值。

6，after与before区别

After：先完成数据的增删改，再触发事件；

触发语句晚于前面的增删改，无法影响前面动作

Before：先完成触发事件，再进行增删改

触发语句先于前面增删改，有机会判断、审查前面动作。

Before应用案例：

1，如前面订单事件，当订单数量大于库存数量时，会产生错误。

因此，我们可以先完成触发事件，判断订单操作是否合法？

若不合法，则在触发事件上进行修改，再完成订单操作

【案例】库存表stage(id, name, c)与订单表offer(oid, sid, m)之间的联系

-- 触发器

-- 库存表

create table stage(

id int,

name varchar(10),

c int

)charset utf8;

-- 订单表

create table offer(

oid int,

sid int,

m int

)charset utf8;

-- 插入数据

insert into stage

values

(1,'猪',22),

(2,'羊',19),

(3,'狗',12),

(4,'猫',8);

delimiter $ #设置$为结束符

-- 创建插入触发器，当插入订单时，库存表相应商品数量改变

create trigger ins

after insert on offer

for each row

begin

update stage set c = c - new.m where id = new.sid;

end$

insert into offer

values

(1, 2, 5)$

-- 创建删除触发器

create trigger del

after delete on offer

for each row

begin

update stage set c = c + old.m where id = old.sid;

end$

delete from offer where oid = 1$

-- 创建修改触发器：修改订单sid

create trigger upda

after update on offer

for each row

begin

update stage set c = c + old.m where id = old.sid; #增加旧的

update stage set c = c - new.m where id = new.sid; #删除新的

end$

update offer set sid = 3 where oid = 1$

-- 删除触发器

drop trigger ins$

-- 修改ins触发器，改为before

create trigger ins

before insert on offer

for each row

begin

if new.m > 5 then

set new.m = 5;

end if;

update stage set c = c - new.m where id = new.sid;

end$

insert into offer

values

(2, 4, 7)$

**四，存储引擎和事务**

1，存储引擎：数据库对于存储的数据存在不同的存储和管理方式，称为存储引擎。

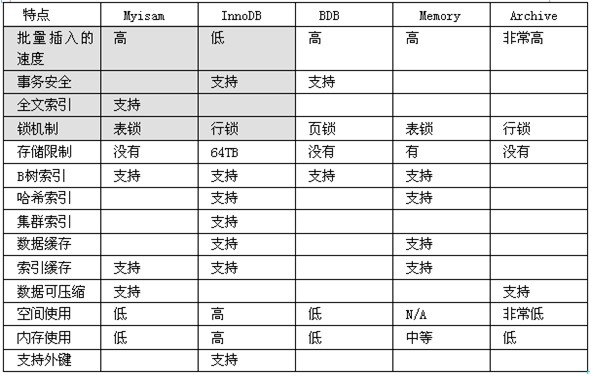
对于用户来说，不同存储引擎存储的数据是一致的；而对于服务器而言，不同的存储引擎有着不同的管理方式。

2，常用的存储引擎：

Myisam：默认，批量插入速度快，但不支持事务安全，采用表锁，安全性要求不高的选择

InnovationDB；批量插入速度慢，支持事务安全，采用行锁，安全性要求高的选择。

Memory：对于临时表采用，速度最快。



支持

3，设置引擎

Create table tname(....) charset = utf8 engine = myisam;

4，事务（transaction）：基于安全性，一组要求必须连续执行的sql语句。如银行事务

四大特性（ACID）:

一组操作，要么全部执行，要么都不执行。 ---->原子性 Atomicity

事务发生前后，数据总额依然匹配。 ----->一致性Consistence

在所有操作执行完成之前，其他会话不能看到指向过程中数据的变化。---->隔离性Isolation

事务产生的影响不能撤销。 ----->持久性 Durability

如果事务产生错误影响，也不能撤销，只能通过“补偿性事务”恢复。

5，语法

启动事务：start transaction;

执行事务的sql语句....

事务终止于一下4种事件：（1）commit或rollback语句

1. DDL/DDC隐式提交，如再次输入：start transaction;

（3）用户退出时自动提交（4）系统强行关闭时取消事务

其中常用的是commit和rollback；

Commit：结束当前事务，并将所有数据变更提交数据库生效。

Rollback：结束当前事务，并取消当前所有数据变更

6，事务基本原理：

1. 若未启动事务，对表的操作语句一提交就直接作用到表的数据文件。
2. 若启动事务，对表的操作会首先提交到事务日志文件中，只有事务结束后（commit）后，才会作用到表的数据文件中去。

**五，数据库备份与恢复**

1，数据库备份：增量备份：备份增加的数据

整体备份：备份所有数据

2，系统自带备份工具：mysqldump，可以导出库、表

3，语句：

（1）导出表：

Mysqldump -u 用户名 -p 密码 数据库名 表1 表2 ... > 地址/备份文件名称

导出的是建表create和插入insert语句

（2）导出库

Mysqldump -u 用户名 -p 密码 数据库名 > 地址/备份文件名称

（3）以库为单位导出

Mysqldump -u用户名 -p密码 -B 库1 库2 库3 > 地址/备份文件名称

（4）导出所有库

MySQLdump -u 用户名 -p 密码 -A > 地址/备份文件名称

4，恢复

1. 登录命令行

1，恢复库级数据库；用备份库的文件恢复（3,4）

Source 备份文件地址;

2，恢复表：用表的备份文件（1,2）

Use 库名;

Source 备份文件地址;

1. 不登录命令行

1，恢复数据库

MySQL -u 用户名 -p 密码 < 备份文件地址

2，恢复数据库中的表

MySQL -u 用户名 -p 密码 库名 < 备份文件地址

**六，索引**

1，索引是针对数据库数据建立的目录，用于加快查询速度

MySQL常用两种索引：二叉树索引、哈希函数

2，坏处：增加了增删改的速度（在操作的同时需要对索引进行操作）；

索引文件占据空间

案例：针对某大数据文件的存储时，插入数据之前先清除空表的所有索引，再插入数据，最后统一建立索引。

3，索引的创建原则：

1. 不要过度索引
2. 推荐在where查询最频繁的列上建立索引
3. 尽量索引散列的数据，过于集中的值加索引意义不大

4，索引类型

普通索引：index，仅加快查询速度

唯一索引：unique，要求该索引的列上每一行的值不重复

主键索引：primary key，主键上的索引

主键必唯一，唯一不一定是主键；表上只能由一个主键，可有多个唯一

全文索引：fulltext，根据文本中某个单词进行索引

5，语法

1. 查看索引

Show index from 表名;

Show index from 表名 \G;

1. 建立索引

1，Alter table 表名 add index/unique/fulltext [索引名] (列名);

#索引名可选，默认为列名

2，Alter table 表名 add primary key; #建立主键索引

3，直接在create语句中建立索引

Create table tname (

Id int,

Name char(20),

Email char(20),

....

primary key (id),

index/unique/fulltext (name, email)

);

1. 删除索引

1，删除非主键索引

Alter table 表名 drop index 索引名;

2，删除主键索引

Alter table 表名 drop primary key;

6，全文索引是针对一大段文本中某个单词进行索引

1. 语法：

1，选择单词匹配的项

Select \* from t where match(全文索引名) anginst (‘匹配的单词’);

2，查看匹配度

Select id,name... match(全文索引名) anginst (‘匹配的单词’) From t;

其中匹配度为0表示无关

1. 全文索引针对中文索引意义不大，由于中文分词的难度导致全文索引不能很好地匹配。

**七，存储过程**

1，类似于函数，将一段代码封装起来；当要执行该段代码时，直接调用该存储过程。

且在封装的语句中，可存在if-else、while等控制语句

2，语法

1. 查看已有存储过程

Show procedure status [\G]; #横着显示

1. 创建存储过程

Create procedure 存储过程名称()

Begin

封装的代码; #可以有sql语句、控制过程...，以分号结尾

End$

用delimiter $设置$为结束符。

1. 调用存储过程：

Call 名称();

1. 删除存储过程

Drop procedure 名称;

【案例】

（1）#参数参与控制结构

create procedure p2(num int, c char(1))

begin

if c = 'h' then

select \* from account where mon > num;

else

select \* from account where mon < num;

end if;

end$

1. #while完成逻辑计算 1+2+...+100

create procedure p3(num smallint)

begin

declare i smallint;

declare s int;

set i = 1;

set s = 0;

while i <= num do

set s = s + i;

set i = i + 1;

end while;

select s;

end$

注意，存储过程没有返回值，想要显示只能调用select

Tips:

1，数据库database、表table、视图view、存储过程：

创建：Create database 名称 ...

删除：drop database 名称

2，表中数据删除：delete from 表名 where ...

3, 对列名取别名，显示时显示别名