**Morning：**

1. 视图介绍

**-What’s视图**

一张虚拟表，内容与真实表相似，有字段有记录，但不存储数据；

更新视图数据即是更新基表，反之亦然！

**-优点与限制**

优点：按需定义视图结构，用户只能看到视图中的数据，屏蔽其他数据。

限制：不能创建索引；不能使用子查询；以下情形的视图不可更新：

-包含聚合函数、group by、having、union、limit等

-from一个不可更新的视图

-使用了临时表

1. 视图基本使用

|  |  |
| --- | --- |
| 创建视图 | create view 视图名称 as SQL查询;  create view 视图名称(字段名列表) as SQL查询; |
| 查看视图 | #查看当前库下所有表的状态信息  show table status;  show table status where comment=”view” \G;  #查看创建视图具体命令  show create view 视图名; |
| 删除视图 | drop view 视图名; |
| 增删改查 | 参考：阶段3 --> 1.DBA --> DAY03 --> page5基表操作 |

Tips： 创建视图后，在数据库文件夹中会创建.frm表结构文件

**更新视图数据即是更新基表，反之亦然！**

常用案例：

mysql> create view v1 as select name,uid,shell from user

-> where shell='/bin/bash';

mysql> grant select on db9.v1 to yaya100@’192.168.4.50’

-> identified by “123456”;

#用户授权对视图具有查询权限，屏蔽其他数据

1. 视图进阶使用

**-创建视图完整格式**

**Create** [or replace]

[algorithm = {undefined | merge | temptable}]

[definer = {user | CURRENT\_USER}]

[SQL SECURITY {DEFINER | INVOKER}]

**VIEW** 视图名[(字段名)]

**AS** SQL查询

[with [cascaded | local] CHECK OPTION]

**-设置字段别名**

多表查询时，查询结果存在重复的字段名，此时视图字段需要定义别名

方法1：重新定义视图字段名

mysql> create view v3(a,b,c,d,e,f,g) as

-> select \* from t1,t2 where t1.uid=t2.uid;

方法2：定义多表别名、表字段别名

mysql> create view v4 as

-> select a.uid u1, a.shell s1, b.uid u2, b.shell s2

-> from t1 a,t2 b where a.uid=b.uid;

**-重要选项说明**

1. 覆盖创建

[or replace] #视图名已存在时覆盖创建

2）视图处理算法

[algorithm = {undefined | merge | temptable}]

#算法，访问视图时的处理方式。undefined时默认使用merge

#merge(替换方式)：将定义时的SQL语句与访问时的语句合并；

#temptable(具体化方式)：将定义视图时的SQL执行结果置于临时表，再执行访问临时表操作。

mysql> CREATE ALGORITHM=temptable view v9 as

-> select uid,name,shell from user2

-> where uid between 0 to 50;

3）检查范围

[WITH [cascaded | local] CHECK OPTION]

#决定用户对视图表作修改时的限制范围。

#不写时无限制，只写with check option时默认为cascaded

#local：只检查当前视图的限制(满足创建视图时的where条件)

#cascaded：同时要满足基表的限制(当基表也为视图，也要满足基表限制)

**Afternoon：**

1. 存储过程

**-What’s存储过程**

相当于MySQL语句组成的脚本，可保存一系列SQL命令，可使用变量，判断，流程控制等。

编译执行，可以提高性能，减轻网络负担；避免重复编写SQL操作。

**-基本使用**

1）创建存储过程  
 mysql> delimiter // #修改分隔符为//

mysql> create procedure 名称()

-> begin

-> select \* from db9.user limit 3;

-> create table if not exists db9.a(id int);

-> End

-> //

mysql> delimiter ; #分隔符恢复为;

注意： 创建存储过程必须在库里

if not exists，如果表存在也不报错

2）查看存储过程

mysql> show procedure status;

mysql> select db,name,type,body from **mysql.proc**

-> where name =”存储过程名” \G;

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

| db | name | type | body |

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

| sys | 存储过程名 | 类型 | 存储过程的代码 |

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

3）调用/删除存储过程

mysql> call 存储过程名(); #调用存储过程

mysql> drop procedure 存储过程名; #删除存储过程，不需加()

2、变量与参数

**-变量类型**

|  |  |  |
| --- | --- | --- |
| 名称 | 描述 | 定义与输出 |
| 会话变量 | 只影响登陆用户，结束连接即失效 | 输出：select @@变量名; |
| 全局变量 | 影响整个数据库服务器 |
| 用户变量 | 连接服务后的整个连接过程有效，  断开后所有用户变量失效。 | 定义：set @变量名=值;  输出：select @变量名; |
| 局部变量 | 存储过程的bggin/end中定义，只在存储过程有效 | 定义：declare 变量名 类型  赋值：set 变量名=值;  调用：不需加@ |

Tips：会话变量+全局变量=系统变量

**-变量的定义与使用**

1）系统变量

show global variables like “变量名” #查看全局变量

set global 变量名=值; #设置全局变量

show session variables like “变量名” #查看会话变量

set session 变量名=值; #设置会话变量

select @@变量名 #输出系统变量

1. 用户变量

set @变量名=值; #定义用户变量并赋值

select max(uid) into @y from user; #使用sql命令赋值

select @y; #输出用户变量值

1. 局部变量

mysql> delimiter //

mysql> create procedure say48()

-> begin

-> declare x int default 9; #声明局部变量x

-> set x =1000; #局部变量x赋值为1000

-> select count(id) into x from db9.user; #使用sql命令赋值

-> select x; #输出局部变量值

-> end //

mysql> delimiter ;

**-参数类型**

1）存储过程参数的定义

mysql> create procedure 名称**（**

-> **关键字 参数名 数据类型,**

-> **关键字 参数名 数据类型 ）**

-> begin

-> ...

|  |  |  |
| --- | --- | --- |
| 关键字 | 名称 | 描述 |
| in | 输入参数 | 给存储过程传值，必须在调用时赋值；  在存储过程内部该参数值不允许修改 |
| out | 输出参数 | 该参数可在存储过程内部被改变，并可返回 |
| inout | 输入/输出参数 | 调用时赋值，并可被改变和返回 |

Tips：不指定关键字，默认为in，存储过程调用时不需要加@

1. in类型案例

mysql> delimiter //

mysql> create procedure say(in user char(10))

-> begin

-> select user;

-> select \* from user where name=user;

-> end //

mysql> delimiter ;

mysql> call say("root"); #对in参数给值，不允许为空

3）out类型案例

mysql> delimiter //

mysql> create procedure p1(in user char(10),out number int)

-> begin

-> select uid into number from user where name=user;

-> end //

mysql> delimiter ;

mysql> call p1("lisi",@num); #使用用户变量对out参数占位

mysql> select @num;

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

| @num |

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

| 1000 |

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

1. inout类型案例

mysql> delimiter //

mysql> create procedure p2(inout x char(30))

-> begin

-> select name into x from user where id=1;

-> end //

mysql> delimiter ;

mysql> set @y="abc"；

mysql> call p2(@y); #必须使用用户变量给值

mysql> select @y;

+-------+

| @y |

+-------+

| root |

+-------+

**-算术运算**



Tips：等号右边可以使用变量

3、流程控制

**-条件测试**

1）数值的比较

|  |  |
| --- | --- |
| 类型 | 用途 |
| = | 等于 |
| >、>= | 大于、大于等于 |
| <、<= | 小于、小于等于 |
| != | 不等于 |
| Between .. And .. | 在..与..之间 |

2）其他条件

|  |  |
| --- | --- |
| 类型 | 用途 |
| OR、AND、! | 逻辑或、与、非 |
| IN .. 、Not IN .. | 在..范围内、不在..范围内 |
| IS NULL | 字段值为空 |
| IS NOT NULL | 字段值不为空 |
| LIKE | 模糊匹配 |
| REGEXP | 正则匹配 |

**-选择结构**

if 条件测试1 then

代码1 ...

elseif 条件测试2 then

代码2 ...

else

代码3 ...

end if;

**-循环结构**

1）while条件式循环（条件成立则执行命令）

while 条件判断 do

循环体

end while;

1. loop死循环

loop

循环体

end loop;

1. repeat条件式循环（条件成立则结束循环）

repeat

循环体 ...

until 条件判断

end repeat;

**-控制循环的执行**

1）跳出循环

leave **标签名**

2）放弃本次循环，执行下一次

iterate **标签名**

1. 循环控制案例

mysql> delimiter //

mysql> create procedure gjq()

-> begin

-> declare i int default 0;

-> lab1:while i<=10 do

-> set i=i+1;

-> if i=3 then

-> iterate lab1;

-> elseif i=5 then

-> leave lab1;

-> end if;

-> select i;

-> end while;

-> end //

mysql> delimiter ;

4、【扩展】SQL的定义、执行、取消

delimiter //

create procedure deltab(in tabname char(10))

begin

declare t\_name char(30);

set t\_name=tabname;

set @y=CONCAT("drop table db5.",t\_name);

PREPARE stmt FROM @y; #定义SQL命令

EXECUTE stmt; #执行SQL命令

DEALLOCATE PREPARE s #取消SQL命令

end //

delimiter ;