链滴

5-MYSQL 数据库其他功能

作者：Carey

原文链接：https://ld246.com/article/1614771999463

来源网站：链滴

3.8 VIEW视图

视图：虚拟表，保存有实表的查询结果，相当于别名

利用视图,可以隐藏表的真实结构,在程序中利用视图进行查询,可以避免表结构的变化,而修改程序,降低序和数据库之间的耦合度

创建方法：

```
CREATE   VIEW view_name [(column_list)]
AS select_statement
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

查看视图定义：

```
SHOW CREATE VIEW view_name #只能看视图定义
SHOW CREATE TABLE view_name # 可以查看表和视图
```

删除视图：

```
DROP VIEW [IF EXISTS]
view_name [, view_name] ...
[RESTRICT | CASCADE]
```

注意：视图中的数据事实上存储于"基表"中，因此，其修改操作也会针对基表实现；其修改操作受基限制

范例：

```
create view v_st_co_sc as select st.name,co.course,sc.score from students st inner join scores s
 on st.`StuID`=sc.stuid inner join courses co on sc.courseid=co.courseid;
show table status like 'v_st_co_sc'\G
```

```
select * from v_st_co_sc;
```

3.9 FUNCTION 函数

函数：分为系统内置函数和自定义函数

- 系统内置函数参考：

https://dev.mysql.com/doc/refman/8.0/en/sql-function-reference.html
https://dev.mysql.com/doc/refman/5.7/en/sql-function-reference.html

- 自定义函数：user-defined function UDF，保存在mysql.proc (MySQL8.0 中已经取消此表)表中

创建UDF语法

```
CREATE [AGGREGATE] FUNCTION function_name(parameter_name type,[parameter_name
type,...])
RETURNS {STRING|INTEGER|REAL}
runtime_body
```

说明：

- 参数可以有多个,也可以没有参数
- 无论有无参数，小括号（）是必须的
- 必须有且只有一个返回值

查看函数列表：

```
SHOW FUNCTION STATUS;
```

查看函数定义

```
SHOW CREATE FUNCTION function_name
```

删除UDF

```
DROP FUNCTION function_name
```

调用自定义函数语法

```
SELECT function_name(parameter_value,...)
```

范例：MySQL8.0 默认开启二进制不允许创建函数

```
#默认MySQL8.0开启二进制日志,而不允许创建函数
CREATE FUNCTION simpleFun() RETURNS VARCHAR(20) RETURN "Hello World";
(1418, 'This function has none of DETERMINISTIC, NO SQL, or READS SQL DATA in its declarat
on and binary logging is enabled (you *might* want to use the less safe log_bin_trust_functio
_creators variable)')
mysql root@(none):hellodb> select @@log_bin;
+-----------+
| @@log_bin |
+-----------+
```

```
| 1         |
+-----------+
1 row in set
Time: 0.006s
mysql root@(none):hellodb> show variables like 'log_bin_trust_function_creators';
+--------------------------------+-------+
| Variable_name                  | Value |
+--------------------------------+-------+
| log_bin_trust_function_creators | OFF   |
+--------------------------------+-------+
1 row in set
Time: 0.014s
#打开此变量允许二进制日志信息函数创建
set global log_bin_trust_function_creators=ON;
CREATE FUNCTION simpleFun() RETURNS VARCHAR(20) RETURN "Hello World";
show function status like 'simple%'\G
#Mariadb10.3 默认没有开启二进制日志,所以可以创建函数
```

MySQL中的变量

两种变量：系统内置变量和用户自定义变量

- 系统变量：MySQL数据库中内置的变量，可用@@var_name引用

```
#系统变量mysql8.0版本帮助查询地址
https://dev.mysql.com/doc/refman/8.0/en/server-option-variable-reference.html
```

- 用户自定义变量分为以下两种
  - 普通变量：在当前会话中有效，可用@var_name引用
  - 局部变量：在函数或存储过程内才有效，需要用DECLARE 声明，之后直接用 var_name引用

自定义函数中定义局部变量语法

```
DECLARE 变量1[,变量2,... ]变量类型 [DEFAULT 默认值]
```

说明：局部变量的作用范围是在BEGIN...END程序中,而且定义局部变量语句必须BEGIN...END的第一定义

为变量赋值语法

```
SET parameter_name = value[,parameter_name = value...]
SELECT INTO parameter_name
```

范例：自定义的普通变量

```
select count(*) from `成绩表` into @num;
select count(*) into @num from `成绩表`;
select @num;
+------+
| @num |
+------+
| 16   |
+------+
```

3.10 PROCEDURE 存储过程

存储过程：多表SQL的语句的集合，可以独立执行，存储过程保存在mysql.proc表中

存储过程优势

存储过程把经常使用的SQL语句或业务逻辑封装起来,预编译保存在数据库中,当需要时从数据库中直调用,省去了编译的过程，提高了运行速度，同时降低网络数据传输量

存储过程与自定义函数的区别

存储过程实现的过程要复杂一些,而函数的针对性较强

存储过程可以有多个返回值,而自定义函数只有一个返回值

存储过程一般可独立执行,而函数往往是作为其他SQL语句的一部分来使用

无参数的存储过程执行过程中可以不加(),函数必须加 ( )

创建存储过程

```
CREATE PROCEDURE sp_name ([ proc_parameter [,proc_parameter ...]])
routime_body
proc_parameter : [IN|OUT|INOUT] parameter_name type
```

说明：其中IN表示输入参数，OUT表示输出参数，INOUT表示既可以输入也可以输出；param_nam表示参数名称；type表示参数的类型

查看存储过程列表

```
SHOW PROCEDURE  STATUS;
```

查看存储过程定义

```
SHOW CREATE PROCEDURE sp_name
```

调用存储过程

```
CALL sp_name ([ proc_parameter [,proc_parameter ...]])
```

说明:当无参时,可以省略"()",当有参数时,不可省略"()

存储过程修改

ALTER语句修改存储过程只能修改存储过程的注释等无关紧要的东西,不能修改存储过程体,所以要修存储过程,方法就是删除重建

删除存储过程

```
DROP PROCEDURE [IF EXISTS] sp_name
```

范例：创建无参存储过程

```
mysql> delimiter //
mysql> create procedure showTime()
```

```
-> BEGIN
-> select now();
-> END//
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;
mysql> call showTime;
+---------------------+
| now()               |
+---------------------+
| 2021-03-01 21:02:06 |
+---------------------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
#delimiter作用修改SQL语句默认结束符
```

范例：创建含参存储过程：只有一个IN参数

```
mysql> delimiter //
mysql> create procedure selectById(IN id SMALLINT unsigned)
-> BEGIN
-> select * from students where stuid = id;
-> END//
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;
mysql> call selectById(2);
+-------+------------+-----+--------+---------+-----------+
| StuID | Name       | Age | Gender | ClassID | TeacherID |
+-------+------------+-----+--------+---------+-----------+
|     2 | Shi Potian |  22 | M      |       1 |         7 |
+-------+------------+-----+--------+---------+-----------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
```

流程控制

存储过程和函数中可以使用流程控制来控制语句的执行

- IF：用来进行条件判断。根据是否满足条件，执行不同语句
- CASE：用来进行条件判断，可实现比IF语句更复杂的条件判断
- LOOP：重复执行特定的语句，实现一个简单的循环
- LEAVE：用于跳出循环控制，相当于SHELL中break
- ITERATE：跳出本次循环，然后直接进入下一次循环，相当于SHELL中continue
- REPEAT：有条件控制的循环语句。当满足特定条件时，就会跳出循环语句
- WHILE：有条件控制的循环语句

3.11 TRIGGER 触发器

触发器的执行不是由程序调用，也不是由手工启动，而是由事件来触发、激活从而实现执行

创建触发器

```
CREATE [DEFINER = { user | CURRENT_USER }]
TRIGGER trigger_name
trigger_time trigger_event
ON tbl_name FOR EACH ROW
trigger_body
```

说明：

trigger_name：触发器的名称

trigger_time：{ BEFORE | AFTER }，表示在事件之前或之后触发

trigger_event:：{ INSERT |UPDATE | DELETE }，触发的具体事件

tbl_name：该触发器作用在表名

范例：

```
#创建触发器，在向学生表INSERT数据时，学生数增加，DELETE学生时，学生数减少
create table student_info( stu_id int(11) not null auto_increment, stu_name varchar(255) defau
t null, primary key (stu_id) );
create table student_count( student_count int(11) default 0 );
insert into student_count values(0);
#增加一个学生信息，学生人数表+1
create trigger trigger_student_count_insert after insert on student_info for each row update s
udent_count set student_count=student_count+1;
#减少一个学生信息，学生人数表-1
create trigger trigger_student_count_delete after delete on student_info for each row update
tudent_count set student_count=student_count-1;
```

查看触发器

```
#在当前数据库对应的目录下，可以查看到新生成的相关文件：trigger_name.TRN,table_name.TRG
SHOW TRIGGERS;
#查询系统表information_schema.triggers的方式指定查询条件，查看指定的触发器信息。
USE information_schema;
SELECT * FROM triggers WHERE trigger_name='trigger_student_count_insert';
```

删除触发器

```
DROP TRIGGER trigger_name;
```

3.12 Event 事件

3.12.1 Event 事件介绍

事件（event）是MySQL在相应的时刻调用的过程式数据库对象。一个事件可调用一次，也可周期性启动，它由一个特定的线程来管理的，也就是所谓的"事件调度器"。

事件和触发器类似，都是在某些事情发生的时候启动。当数据库上启动一条语句的时候，触发器就启

了，而事件是根据调度事件来启动的。由于它们彼此相似，所以事件也称为临时性触发器。

事件取代了原先只能由操作系统的计划任务来执行的工作，而且MySQL的事件调度器可以精确到每钟执行一个任务，而操作系统的计划任务（如：Linux下的CRON或Windows下的任务计划）只能精到每分钟执行一次。

事件的优缺点

优点：一些对数据定时性操作不再依赖外部程序，而直接使用数据库本身提供的功能，可以实现每秒执行一个任务，这在一些对实时性要求较高的环境下就非常实用

缺点：定时触发，不可以直接调用

3.12.2 Event 管理

3.12.2.1 相关变量和服务器选项

MySQL事件调度器event_scheduler负责调用事件，它默认是关闭的。这个调度器不断地监视一个事件

是否要调用， 要创建事件，必须打开调度器

服务器系统变量和服务器选项：

event_scheduler：默认值为OFF，设置为ON才支持Event，并且系统自动打开专用的线程

范例：开启和关闭event_scheduler

```
#默认事件调度功能是关闭的
mysql> select @@event_scheduler;
+-------------------+
| @@event_scheduler |
+-------------------+
| OFF               |
+-------------------+
1 row in set (0.00 sec)
#临时开启事件调度功能
set global event_schediler=1;
#开启事件调度功能后,自启动一个event_scheduler线程
mysql> mysql> show processlist;
+----+-----------------+-----------+------+---------+------+-----------------------+-------------------+
| Id | User            | Host      | db   | Command | Time | State                 | Info              |
+----+-----------------+-----------+------+---------+------+-----------------------+-------------------+
|  5 | event_scheduler | localhost | NULL | Daemon  | 2717 | Waiting on empty queue | NULL              |
| 13 | root            | localhost | db1  | Query   |    0 | starting              | show processlist |
+----+-----------------+-----------+------+---------+------+-----------------------+-------------------+
2 rows in set (0.00 sec)
#临时关闭事件调度功能
set global event_scheduler=0;
#持久开启事件调度
```

```
[root@centos8 ~]#vim /etc/my.cnf.d/mariadb-server.cnf
[mysqld]
event_scheduler=ON
[root@centos8 ~]#systemctl restart mysqld
```

3.12.2.2 管理事件

create event 语句创建一个事件。每个事件由两个主要部分组成，第一部分是事件调度（eventsched le），表示事件何时启动以及按什么频率启动，第二部分是事件动作（event action ），这是事件启时执行的代码，事件的动作包含一条SQL语句，它可能是一个简单地insert或者update语句，也可以一个存储过程或者 benin...end语句块，这两种情况允许我们执行多条SQL

一个事件可以是活动（打开）的或停止（关闭）的，活动意味着事件调度器检查事件动作是否必须调，停止意味着事件的声明存储在目录中，但调度器不会检查它是否应该调用。在一个事件创建之后，立即变为活动的，一个活动的事件可以执行一次或者多次

创建Event

```
CREATE
[DEFINER = { user | CURRENT_USER }]
EVENT
[IF NOT EXISTS]
event_name
ON SCHEDULE schedule
[ON COMPLETION [NOT] PRESERVE]
[ENABLE | DISABLE | DISABLE ON SLAVE]
[COMMENT 'comment']
DO event_body;
schedule:
AT timestamp [+ INTERVAL interval] ...
| EVERY interval
[STARTS timestamp [+ INTERVAL interval] ...]
[ENDS timestamp [+ INTERVAL interval] ...]
interval:
quantity {YEAR | QUARTER | MONTH | DAY | HOUR | MINUTE |
WEEK | SECOND | YEAR_MONTH | DAY_HOUR | DAY_MINUTE |
DAY_SECOND | HOUR_MINUTE | HOUR_SECOND | MINUTE_SECOND}
```

说明：

event_name ： 创建的event名字，必须是唯一确定的

ON SCHEDULE：计划任务

schedule: 决定event的执行时间和频率（注意时间一定要是将来的时间，过去的时间会出错），有种形式 AT和EVERY

[ON COMPLETION [NOT] PRESERVE]： 可选项，默认是ON COMPLETION NOT PRESERVE 即计任务执行完毕后自动drop该事件；ON COMPLETION PRESERVE则不会drop掉

[COMMENT 'comment'] ： 可选项，comment 用来描述event；相当注释，最大长度64个字节

[ENABLE | DISABLE] ： 设定event的状态，默认ENABLE：表示系统尝试执行这个事件， DISABLE：闭该事情，可以用alter修改

DO event_body: 需要执行的sql语句，可以是复合语句

提示：event事件是存放在mysql.event表中

查看Event

```
SHOW EVENTS [{FROM | IN} schema_name]
[LIKE 'pattern' | WHERE expr]
```

注意：事件执行完即释放，如立即执行事件，执行完后，事件便自动删除，多次调用事件或等待执行件,才可以用上述命令查看到。

修改Event

```
ALTER
[DEFINER = { user | CURRENT_USER }]
EVENT event_name
[ON SCHEDULE schedule]
[ON COMPLETION [NOT] PRESERVE]
[RENAME TO new_event_name]
[ENABLE | DISABLE | DISABLE ON SLAVE]
[COMMENT 'comment']
[DO event_body]
```

注意：alter event语句可以修改事件的定义和属性。可以让一个事件成为停止的或者再次让它活动，可以修改一个事件的名字或者整个调度。然而当一个使用 ON COMPLETION NOT PRESERVE 属性义的事件最后一次执行后，事件直接就不存在了，不能修改

删除Event

```
DROP EVENT [IF EXISTS] event_name
```

3.12.2.3 范例

范例：创建立即启动事件

```
create database testdb;
use testdb;
#创建一个表记录每次事件调度的名字和事件戳
create table events_list(event_name varchar(20) not null,event_started timestamp not null);
#临时关闭事件调度功能
set global event_scheduler=0;
show variables like 'event_scheduler';
+----------------+-------+
| Variable_name   | Value |
+----------------+-------+
| event_scheduler | OFF   |
+----------------+-------+
#创建一次性事件
create event event_now on schedule at now() do insert into events_list values('event_now',no
());
#因为事件调度功能禁用，所有表中无记录
select * from events_list;
#查看事件
```

```
show events\G
#开启事件调度功能
set global event_scheduler=1;
#事件立即执行，每秒插入一条记录
select * from events_list;
#事件执行完成后自动删除
show events;
```

3.13 MySQL 用户管理

相关数据库和表

元数据数据库：mysql
系统授权表：db, host, user,columns_priv, tables_priv, procs_priv, proxies_priv

用户帐号:

'USERNAME'@'HOST'
@'HOST': 主机名： user1@'web1.magedu.org'
IP地址或Network
通配符： %
示例：wang@172.16.%.%
user2@'192.168.1.%'
mage@'10.0.0.0/255.255.0.0'

创建用户：CREATE USER

```
CREATE USER 'USERNAME'@'HOST' [IDENTIFIED BY 'password'];
#示例:
create user 'cy'@'192.168.10.0/255.255.255.0' identified by '123456';
create user 'zhangzhuo'@'192.168.10.%' identified by '123456';
```

新建用户的默认权限：USAGE

用户重命名：RENAME USER

```
RENAME USER old_user_name TO new_user_name;
#示例
rename user 'cy'@'192.168.10.0/255.255.255.0' to 'chengyang'@'10.0.0.%';
```

删除用户：

```
DROP USER 'USERNAME'@'HOST'
#示例
drop user 'chengyang'@'10.0.0.%';
```

修改密码：

注意：

- 新版mysql中用户密码可以保存在mysql.user表的authentication_string字段中
- 如果mysql.user表的authentication_string和password字段都保存密码，authentication_string先生效

```
#方法1,用户可以也可通过此方式修改自已的密码
SET PASSWORD FOR 'user'@'host' = PASSWORD('password'); #MySQL8.0 版本不支持此方法,为password函数被取消
set password for root@'localhost'='123456' ; #MySQL8.0版本支持此方法,此方式直接将密码123456加密后存放在mysql.user表的authentication_string字段
#方法2
ALTER  USER test@'%' IDENTIFIED BY 'centos'; #通用改密码方法, 用户可以也可通过此方式修改已的密码,MySQL8 版本修改密码

#方法3 此方式MySQL8.0不支持,因为password函数被取消
UPDATE mysql.user SET password=PASSWORD('password') WHERE clause;
#mariadb 10.3
update mysql.user set authentication_string=password('ubuntu') where user='mage';
#此方法需要执行下面指令才能生效:
FLUSH PRIVILEGES;
```

忘记管理员密码的解决办法:

1. 启动mysqld进程时，为其使用如下选项:

```
--skip-grant-tables
--skip-networking
```

2. 使用UPDATE命令修改管理员密码
3. 关闭mysqld进程，移除上述两个选项，重启mysqld

范例:

```
[11:47:57 root@centos8 ~]#vim /etc/my.cnf.d/mysql-server.cnf
skip-grant-tables
skip-networking  #mysql8.0不需要
[11:48:44 root@centos8 ~]#systemctl restart mysqld.service
#方法1
flush privileges;
set password for root@'localhost'='Zhangzhuo@1234';

[root@centos8 ~]#vim /etc/my.cnf
[mysqld]
#skip-grant-tables
#skip-networking
[11:56:32 root@centos8 ~]#systemctl restart mysqld.service
```

3.14 权限管理和DCL语句

3.14.1 权限类别

权限类别:

- 管理类
- 程序类
- 数据库级别

- 表级别
- 字段级别

管理类：

- CREATE USER
- FILE
- SUPER
- SHOW DATABASES
- RELOAD
- SHUTDOWN
- REPLICATION SLAVE
- REPLICATION CLIENT
- LOCK TABLES
- PROCESS
- CREATE TEMPORARY TABLES

程序类：针对 FUNCTION、PROCEDURE、TRIGGER

- CREATE
- ALTER
- DROP
- EXCUTE

库和表级别：针对 DATABASE、TABLE

- ALTER
- CREATE
- CREATE VIEW
- DROP INDEX
- SHOW VIEW
- WITH GRANT OPTION：能将自己获得的权限转赠给其他用户

数据操作

- SELECT
- INSERT
- DELETE
- UPDATE

字段级别

- SELECT(col1,col2,...)

- UPDATE(col1,col2,...)
- INSERT(col1,col2,...)

所有权限

- ALL PRIVILEGES 或 ALL

3.14.2 授权

授权：GRANT，授权之后如要立即生效需要使用flush privileges命令生效

```
GRANT priv_type [(column_list)],... ON [object_type] priv_level TO 'user'@'host'
[IDENTIFIED BY 'password'] [WITH GRANT OPTION];
priv_type: ALL [PRIVILEGES]
object_type:TABLE | FUNCTION | PROCEDURE
priv_level: *(所有库) |*.*  | db_name.* | db_name.tbl_name | tbl_name(当前库
的表) | db_name.routine_name(指定库的函数,存储过程,触发器)
with_option: GRANT OPTION
| MAX_QUERIES_PER_HOUR count
| MAX_UPDATES_PER_HOUR count
| MAX_CONNECTIONS_PER_HOUR count
| MAX_USER_CONNECTIONS count
```

参考：https://dev.mysql.com/doc/refman/5.7/en/grant.html

范例：

```
GRANT SELECT (col1), INSERT (col1,col2) ON mydb.mytbl TO 'someuser'@'somehost';
grant all on *.* to 'zhangzhuo'@'%';
GRANT ALL PRIVILEGES ON *.* TO 'root'@'10.0.0.%'  WITH GRANT OPTION;

#创建用户和授权同时执行的方式在MySQL8.0取消了
GRANT ALL ON wordpress.* TO wordpress@'192.168.8.%' IDENTIFIED BY 'magedu';
GRANT ALL PRIVILEGES ON *.* TO 'root'@'192.168.8.%' IDENTIFIED BY 'magedu' WITH GRAN
 OPTION;
```

3.14.3 取消权限

取消授权：REVOKE

```
REVOKE priv_type [(column_list)] [, priv_type [(column_list)]] ... ON
[object_type] priv_level FROM user [, user] ...
```

参考：https://dev.mysql.com/doc/refman/5.7/en/revoke.html

范例：

```
revoke all on *.* from 'zhangzhuo'@'%';
```

3.14.4 查看指定用户获得的授权

```
Help SHOW GRANTS
```

```
SHOW GRANTS FOR 'user'@'host';
SHOW GRANTS FOR CURRENT_USER[()];
```

注意：

MariaDB服务进程启动时会读取mysql库中所有授权表至内存

(1) GRANT或REVOKE等执行权限操作会保存于系统表中，MariaDB的服务进程通常会自动重读授权表，使之生效

(2) 对于不能够或不能及时重读授权表的命令，可手动让MariaDB的服务进程重读授权表：

mysql> FLUSH PRIVILEGES;

3.15 MySQL的图形化的远程管理工具

在MySQL数据库中创建用户并授权后，可以使用相关图形化工具进行远程的管理。

常见的图形化管理工具：

- Navicat
- SQLyog