数据库第六次上机

2024/5/9

本次上机概览

本次的上机内容主要有：

- 存储过程
- 游标使用
- 自定义函数*
- 触发器

（MySQL讲解及代码示例见PPT末）请注意区分MySQL、OpenGauss、SQL Server不同示例

关于作业提交

TASK1： 1-1~1-4
TASK2： 2-1~2-6

请在**PDF/WORD等任何方便助教阅读查看的文档**中按照各个作业要求提交相关内容，**记得标清题号**。

若为PDF/WORD单文档文件直接提交即可，其他提交压缩包，命名为“学号_姓名_第*次实验”。

提交网址：软件学院云平台**第六次上机**北航软件学院-云平台 (buaa.edu.cn)
（按要求提交）

作业截止时间为**本周末24:00之前，提交方式为提交到云平台。**

本次上机任务

— 简易图书馆书籍借阅系统

Task1：

1. 新建数据库 Library

2. 新建如下表：（键与约束自行合理设定即可，字段可自由增加，比如 ID）

账户(用户名, 密码, …)

书库(ISBN, 书名, 数量, …)

借阅记录(用户名, ISBN, 借书时间, 到期时间, 还书时间, …)

Ps：到期时间约定为借书时间+30天。

关于时间和日期类的文档：
https://docs.microsoft.com/zh-cn/sql/t-sql/data-types/date-and-time-types?view=sql-server-2017
https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html

本次上机任务

提交要求：请在**PDF/WORD等任何方便助教阅读查看的文档中**完成以下提交内容：需要提交每个查询的**SQL语句和查询结果截图，请标好题号**。

Task1：

通过调用存储过程或者自定义函数实现如下用户操作：

1-1 用户密码检查/修改： 接收四个参数（用户名，密码，新密码，动作），若动作为1，则检查用户名和密码是否和密码表中存的相符，相符则返回 true，不相符返回false； 若动作为2， 则首先检查用户名、密码是否相符，若不相符返回false，相符则将密码表中的密码改成新密码，返回true。密码要求只包含数字和字母，长度大于等于4、小于等于10；

1-2 借书：接收两个参数（用户名，ISBN），没有足够的书、用户不存在或一个人借阅两本同样的书时返回false，合法执行后，借阅记录表会新增一条记录，书库对应书的数量也需要减1，并返回true；

1-3 还书：接收两个参数（用户名，ISBN），未查询到借阅记录时返回false，合法执行后，借阅记录表对应记录会修改还书时间，书库对应书的数量需要加1，并返回true；

1-4 查看当前借阅记录：接受一个参数（用户名），返回该用户名的当前借阅中的记录(用户名, ISBN, 到期时间)

◆ 说明:

1. 每道题使用存储过程或者自定义函数任一实现即可。
2. 存储过程/函数的参数等细节自行设计，但执行操作之后表要有对应的修改。
3. 每项功能需要自定义至少3条测试用例进行测试。

注意至少要有一题使用到游标，不必考虑效率，只要使用到游标即可（比如用户密码检查：建立账户表的游标，然后遍历其来判断是否存在该用户，根据结果再继续执行判断密码是否相符）

本次上机任务

提交要求：请在**PDF/WORD等任何方便助教阅读查看的文档中**完成以下提交内容：需要提交每个查询的**SQL语句和查询结果截图，请标好题号**。

Task2：完成下列触发器相关实验

2-1：建表：fruits（fid，fname，price），sells（fid，cid，sellTime，quantity），customer（cid，cname，level)，在fruits表和customer 表插入至少一条数据。

2-2：写一个sells表触发器**check_fid_exist**，当插入新的用户购买记录之前，检查插入的新的购买记录中的fid值在fruits表中是否存在。若不存在，则引发错误，提示信息为"该水果数据不存在"

2-3：写一个sells表触发器**check_cid_exist**，当插入新的用户购买记录之前，检查新的购买记录中的用户cid在customer表中是否存在。若不存在，则将该用户ID插入到customer表中（cname为空，level设为normal）

2-4：写一个sells表触发器**triADD**：当插入新的用户购买记录之后，检查该用户购买的总价值（每种水果价格 * 销售量的和 ）超过1万元就设置customer表的level为VIP，超过2万元设置为SVIP，低于1万元则置为normal。

2-5：写两个sells表触发器**triDEL**和**triUPT**，若删除或修改sells表记录，也重新计算并重置客户的level值。

2-6：通过DML语句进行触发器效果验证，如插入/更新/删除交易记录，查看用户等级变化。

TIPS: 使用存储过程将重复流程简化

MYSQL中触发器中不能对本表进行 insert ,update ,delete操作，以免递归循环触发

基础知识

— 存储过程基本概念

- 存储过程定义：

存储过程是一组为了完成特定功能的SQL 语句，其经编译后存储在数据库中，用户通过指定存储过程的名字并给出参数来执行它。

在SQL Server 的系列版本中，存储过程分为两类：系统提供的存储过程和用户自定义存储过程。

基础知识

— 存储过程基本概念

- 系统存储过程：

主要存储master 数据库中，并以sp_为前缀。
系统存储过程主要是从系统表中获取信息，从而为系统管理员服务。

- 常用的系统存储过程：

```
exec sp_databases;                          --查看数据库
exec sp_tables;                             --查看表
exec sp_columns student;                    --查看列
exec sp_helpIndex student;                  --查看索引
exec sp_helpConstraint student;             --约束
exec sp_helptext 'sp_stored_procedures';    --查看存储过程创建、定义语句
（经常用到这句话来查看存储过程，like sp_helptext sp_getLoginInfo. ）
exec sp_rename student, stuInfo;            --修改表、索引、列的名称
exec sp_renamedb myTempDB, myDB;            --更改数据库名称
exec sp_defaultdb 'master', 'myDB';         --更改登录名的默认数据库
exec sp_helpdb;                             --数据库帮助，查询数据库信息
```

基础知识

— 存储过程基本概念

- 用户自定义存储过程:

用户自定义存储过程是由用户创建，并能完成某一特定功能的存储过程，如：查询用户所需数据信息。

- 存储过程的好处:
  - 重复使用。存储过程可以重复使用，从而可以减少数据库开发人员的工作量。
  - 提高性能。存储过程在创建的时候就进行了编译，将来使用的时候不用再重新编译。一般的SQL语句每执行一次就需要编译一次，所以使用存储过程提高了效率。
  - 减少网络流量。存储过程位于服务器上，调用的时候只需要传递存储过程的名称以及参数就可以了，因此降低了网络传输的数据量。
  - 安全性。参数化的存储过程可以防止SQL注入式的攻击，而且可以将Grant、Deny以及Revoke权限应用于存储过程。

- 简单使用时只需要关注语句声明，参数定义和语句主体;

基础知识

— 创建存储过程

MySQL:

- 大致语法:

```
CREATE PROCEDURE procedure_name (
        [ IN | OUT | INOUT ] param_name type [ ,... ]
)
[ BEGIN ] sql_statement [ END ]
```

- 说明:

1. 在定义存储过程和函数前后使用DELIMITER修改sql分隔符会体验更好
（例如在定义前改为$$，定义后改回;）
2. 使用CALL执行存储过程
（CALL sp_name([parameter[,...]])）
3. mysql的存储过程不能使用return语句，只有存储函数才有此功能。

(30条消息) mysql 存储过程中不能使用 return 的解决办法_gd2008的博客-CSDN博客

相关文档:

https://dev.mysql.com/doc/refman/8.0/en/create-procedure.html
https://www.mysqlzh.com/doc/223.html
https://www.runoob.com/w3cnote/mysql-stored-procedure.html

基础知识

— 创建存储过程

openGauss：

- 大致语法：

CREATE [OR REPLACE] PROCEDURE procedure_name
[({[argmode] [argname] argtype [{ DEFAULT | := | = } expression]}[,...])]
[{ IMMUTABLE | STABLE | VOLATILE } | { SHIPPABLE | NOT SHIPPABLE } | {PACKAGE} | [NOT] LEAKPROOF | { CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT } | {[EXTERNAL] SECURITY INVOKER | [EXTERNAL] SECURITY DEFINER | AUTHID DEFINER | AUTHID CURRENT_USER} | COST execution_cost | ROWS result_rows | SET configuration_parameter { [TO | =] value | FROM CURRENT }][...]
{ IS | AS } plsql_body

相关文档：

https://opengauss.org/zh/docs/1.0.0/docs/Developerguide/CREATE-PROCEDURE.html
https://www.bookstack.cn/read/opengauss-1.0-zh/6d7ccdd0a6ceac88.md
https://www.modb.pro/db/31262

基础知识

— 创建存储过程

SQL Server：

- 大致语法：

```
CREATE [ OR ALTER ] { PROC | PROCEDURE } procedure_name
    [ { @parameter data_type } [ = default ] [ OUT | OUTPUT | [READONLY] ] [ ,...n ]
    // @参数1 数据类型 [=默认值] [OUT | OUTPUT | [READONLY]]
    // @参数2 ...
AS { [ BEGIN ] sql_statement [;] [ ...n ] [ END ] }
[;]
```

- 说明：

1. OUT | OUTPUT 用于指示参数是输出参数。 使用 OUTPUT 参数可将值返回给过程的调用方。
2. 使用EXECUTE（或EXEC）执行存储过程
   （语法大致为EXEC[UTE] 存储过程名 [@参数n = value][,...n]，具体可参见文档）
3. 要在同一个sql文件里创建存储过程和执行其它查询的话，**记得在创建语句后加GO**。

相关文档：

https://docs.microsoft.com/zh-cn/sql/t-sql/statements/create-procedure-transact-sql?view=sql-server-2017

https://docs.microsoft.com/zh-cn/sql/t-sql/language-elements/execute-transact-sql?view=sql-server-2017

基础知识

— 游标基本概念

- 为什么要使用游标?

我们知道，关系数据库所有的关系运算其实是集合与集合的运算，它的输入是集合输出同样是集合，有时需要对结果集**逐行**进行处理，这时就需要用到游标，往往与存储过程搭配使用。

我们对游标的使用一般遵循“五步法”：声明游标—>打开游标—>读取数据—>关闭游标—>删除游标。以下就从这五步对游标的使用进行说明。

- 游标的特点:
  1. 面向行思维
  2. 游标绑定了一个DQL语句，提供了一种能从包括多条数据记录的结果集中每次提取一条记录的机制
  3. 临时性：关闭数据库管理系统（DBMS）后游标消失(不存储！)

基础知识

— 创建游标

SQL Server：

- 大致语法：

DECLARE cursor_name CURSOR [LOCAL | GLOBAL] （说明游标的作用域）
[FORWARD_ONLY | SCROLL] （说明游标的方向）
[STATIC | KEYSET | DYNAMIC | FAST_FORWARD] （说明游标的类型）
FOR select_statement
[FOR UPDATE [OF column_name [,…n]]] （说明游标可更新的列）
[;]

说明：

1. 打开游标使用OPEN cursor_name。
2. 相关参数意义见后面PPT。
3. 还可以指定游标的读取时是否对行上锁，完整语法见详细文档。

相关文档：

https://docs.microsoft.com/zh-cn/sql/t-sql/language-elements/declare-cursor-transact-sql?view=sql-server-2017

基础知识

— 游标的作用域

- **LOCAL**

说明所声明的游标为局部的，其作用域为创建它的批处理、存储过程或触发器，即在批处理、调用它的存储过程或触发器执行完成后，该游标被系统隐式释放。但若游标作为存储过程OUTPUT 的输出参数，在存储过程终止后给游标变量分配参数可以继续引用游标，如果 OUTPUT 参数将游标传递回来，则游标在最后引用它的变量释放或离开作用域时释放。

- **GLOBAL**

指定该游标的作用域对来说连接是全局的。在由连接执行的任何存储过程或批处理中，都可以引用该游标名称。该游标仅在断开连接时隐式释放。

注意：如果 GLOBAL 和 LOCAL 参数都未指定，则默认值由 **default to local cursor** 数据库选项的设置控制。在早期版本中，所有游标都是全局的。

基础知识

— 游标的方向

- **FORWARD_ONLY**

1. 指定游标只能从**第一行滚动到最后一行**。**FETCH NEXT 是唯一支持的提取选项**。如果在指定 FORWARD_ONLY 时不指定 STATIC、KEYSET 和 DYNAMIC 关键字，则游标作为 **DYNAMIC** 游标进行操作。
2. 如果 FORWARD_ONLY 和 SCROLL 均未指定，则除非指定 STATIC、KEYSET 或 DYNAMIC 关键字，否则默认为 **FORWARD_ONLY**。
3. STATIC、KEYSET 和 DYNAMIC 游标默认为 SCROLL。与 ODBC 和 ADO 这类数据库 API 不同， Transact-SQL 中，STATIC、KEYSET 和 DYNAMIC 游标支持 FORWARD_ONLY。

- **SCROLL**

指定所有的提取选项（FIRST、LAST、PRIOR、NEXT、RELATIVE、ABSOLUTE）均可用。如果未在 ISO DECLARE CURSOR 中指定 SCROLL，则 NEXT 是唯一支持的提取选项。如果也指定了 FAST_FORWARD，则不能指定 SCROLL。

基础知识

— 游标的类型

- **STATIC**（源数据库操作不影响数据集中数据）

  定义一个游标，以创建将由该游标使用的数据的临时复本。对游标的所有请求都从 **tempdb** 中的这一临时表中得到应答；因此，在对该游标进行提取操作时返回的数据中**不反映对基表所做的修改**，并且该游标**不允许修改，**是**只读**的。

- **KEYSET**（只有被标识行的数据的改动会被记录到结果集）

  指定当游标打开时，游标中行的成员身份和顺序已经固定。对行进行唯一标识的键集内置在 **tempdb** 内一个称为 **keyset** 的表中。

- **DYNAMIC**（数据集中数据根据源数据动态改变）

  定义一个游标，以**反映在滚动游标时对结果集内的各行所做的所有数据更改**。行的数据值、顺序和成员身份在每次提取时都会更改。

- **FAST_FORWARD**

  指定启用了性能优化的 FORWARD_ONLY、READ_ONLY 游标。如果指定了 SCROLL 或 FOR_UPDATE，则不能也指定 FAST_FORWARD。

基础知识

— 游标读取数据

语法：

```
FETCH [ [ NEXT | PRIOR | FIRST | LAST | ABSOLUTE { n | @nvar } | RELATIVE { n | @nvar } ]
      FROM
]
{ { [ GLOBAL ] cursor_name } | @cursor_variable_name }
[ INTO @variable_name [ ,...n ] ] --into说明将读取的游标数据存放到指定的变量中
```

说明：

1. **NEXT**为默认的游标提取选项
2. 只进游标只支持NEXT
3. 动态游标不支持ABSOLUTE（返回从游标起始处开始向后的第 n 行）
4. N列的表需要N个变量去INTO（从左至右填充，变量的数目必须与游标选择列表中的列数一致）

相关文档：https://docs.microsoft.com/zh-cn/sql/t-sql/language-elements/fetch-transact-sql?view=sql-server-2017

基础知识

— 游标提取数据示例

- 说明:

指定游标当前行:
WHERE CURRENT OF 游标名

游标状态全局变量:
@@FETCH_STATUS记录上一个FETCH语句的执行状态

0	FETCH 语句成功。
-1	FETCH 语句失败或行不在结果集中。
-2	提取的行不存在。
-9	游标未执行提取操作。

```
-- ------[游标状态变量]--------
DECLARE curAccount CURSOR
    SCROLL DYNAMIC
FOR
    SELECT * FROM Account
        WHERE LEN(Password) <= 8
            OR Password IS NULL
FOR UPDATE OF Password;
OPEN curAccount;

WHILE @@FETCH_STATUS = 0    -- 逐行处理
BEGIN
    FETCH NEXT FROM curAccount;
    UPDATE Account          -- DELETE操作同理
        SET Password = 'MustBeChangedSoon'
        WHERE CURRENT OF curAccount;
END

CLOSE curAccount;
DEALLOCATE curAccount;
```

基础知识

— CURSOR简单使用示例

Users(user_name, user_pass)
示例：查询users表的前10条记录，并以逗号拼接的形式返回字符串

```
DELIMITER //
CREATE PROCEDURE test_cursor (out result VARCHAR(255))
BEGIN
  declare name VARCHAR(20);
  declare pass VARCHAR(20);
  declare cnt int default 0;
  declare cur_test CURSOR FOR SELECT user_name,user_pass FROM users;
  OPEN cur_test;
  repeat
    FETCH cur_test into name,pass;
    SELECT concat_ws(',',result,name,pass) INTO result;
    set cnt = cnt + 1;
  until cnt = 10
  END repeat;
  CLOSE cur_test;
end; //
```

基础知识

— 游标关闭和释放

- **关闭游标：**

语法：CLOSE { { [GLOBAL] cursor_name } | cursor_variable_name }
说明：需要该游标事先声明并已打开

- **释放游标：**

语法：DEALLOCATE { { [GLOBAL] cursor_name } | @cursor_variable_name }
说明：
1. 对游标进行操作的语句**使用游标名称或游标变量引用游标。DEALLOCATE 删除游标与游标名称或游标变量之间的关联。如果一个名称或变量是最后引用游标的名称或变量，则将释放游标，游标使用的任何资源也随之释放**。
2. 用于保护提取隔离的滚动锁在 DEALLOCATE 上释放。用于保护更新（包括通过游标进行的定位更新）的事务锁一直到事务结束才释放。

基础知识

— FOR MySQL 玩家

- 语法差异:

游标从属于一个存储过程，只能在存储过程里declare。

INTO子句是必须的

不需要释放游标（没有deallocate）

- 功能差异:

无修饰词
只读只进 (READ_ONLY && FORWARD_ONLY)

详细可参阅:
https://dev.mysql.com/doc/refman/8.0/en/cursors.html

openGauss：https://www.modb.pro/db/30545

```
-- ------[简单游标过程示例]--------
DROP PROCEDURE cur_account_demo;
DELIMITER GO
CREATE PROCEDURE cur_account_demo()
BEGIN
  -- 数据接收变量
  DECLARE u VARCHAR(50);
  DECLARE p VARCHAR(50);
  -- fetch循环结束标志
  DECLARE done INT DEFAULT FALSE;
  -- 游标
  DECLARE cur_account CURSOR FOR SELECT * FROM account;
  -- 结束标志绑定到游标
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
  OPEN cur_account;

  -- 开始fetch循环
  fetch_loop: LOOP
    FETCH cur_account INTO u, p;
    IF done THEN    -- 读取结束时退出循环
      LEAVE fetch_loop;
    END IF;
    SELECT u, p;    -- 这里写操作主体
  END LOOP;

  CLOSE cur_account;
END; GO
DELIMITER ;
CALL cur_account_demo();
```

基础知识

— 游标的局限性和使用场景

- 游标的缺点：

1. 游标的缺点是针对优点而言的，也就是只能一行一行操作，在数据量大的情况下，是不适用的，**速度过慢**。
2. 数据库大部分是面对集合的，业务会比较复杂，而**使用游标容易产生死锁**，影响其他的业务操作，不可取。
3. 当数据量大时，使用游标会造成**内存不足**现象，因为游标其实是相当于把磁盘数据整体放入了内存中。

- 游标的使用场景：

1. **数据量较小**
2. 用在循环处理、存储过程、函数中使用，用来查询结果集，就比如：我们需要从表中循环判断并得到想要的结果集，这时候使用游标操作很方便速度也很快。

基础知识

—自定义函数基本概念

建议大家查看官方文档，示例和展示效果都会比ppt好

- 自定义函数定义：

与编程语言中的函数类似，SQL Server 用户定义函数是接受参数、执行操作（例如复杂计算）并将操作结果以值的形式返回的例程。 返回值可以是单个标量值或结果集。

- 使用自定义函数的优点类似存储过程：

1. 允许模块化程序设计：只需创建一次函数并将其存储在数据库中，以后便可以在程序中调用任意次。 用户定义函数可以独立于程序源代码进行修改。

2. 执行速度更快：与存储过程相似，Transact-SQL 用户定义函数通过缓存计划并在重复执行时重用它来降低 Transact-SQL 代码的编译开销。 这意味着每次使用用户定义函数时均无需重新解析和重新优化，从而缩短了执行时间。

3. 减少网络流量：基于某种无法用单一标量的表达式表示的复杂约束来过滤数据的操作，可以表示为函数。 然后，此函数便可以在 WHERE 子句中调用，以减少发送至客户端的数字或行数。

基础知识

—自定义函数基本概念

- 自定义函数的限制：

1. 不能用于执行修改数据库状态的操作。
2. 不能包含将表作为其目标的 OUTPUT INTO 子句。
3. 不能返回多个结果集。 如果需要返回多个结果集，请使用存储过程。
4. 错误处理受限制。用户定义函数不支持 TRY...CATCH、@ERROR 或 RAISERROR。
5. 不能调用存储过程（但是可调用扩展存储过程（部分的系统存储过程，以xp开头））
6. 不允许 SET 语句。
7. 不支持动态SQL。

.....

参考链接：
https://docs.microsoft.com/zh-cn/sql/relational-databases/user-defined-functions/create-user-defined-functions-database-engine?view=sql-server-2017

基础知识

— 创建自定义函数

MySQL:

- 大致语法:

```
CREATE FUNCTION function_name (
      param_name type [ ,... ]
)
RETURNS type
[ BEGIN ] sql_statement [ END ]
```

- 说明:

1. 直接使用 函数名(参数) 调用函数。
2. MySQL不能返回表。
3. MySQL如果创建函数的时候报Error Code 1418， 需要set global log_bin_trust_function_creators = 1;

- 相关文档:

https://dev.mysql.com/doc/refman/8.0/en/create-procedure.html

基础知识

— 创建自定义函数

openGauss：

- 详细语法与示例请参考：
- https://opengauss.org/zh/docs/1.0.0/docs/Developerguide/CREATE-FUNCTION.html

- 部分截图：

```
--定义函数为SQL查询。
postgres=# CREATE FUNCTION func_add_sql(integer, integer) RETURNS integer
    AS 'select $1 + $2;'
    LANGUAGE SQL
    IMMUTABLE
    RETURNS NULL ON NULL INPUT;

--利用参数名用 PL/pgSQL 自增一个整数。
postgres=# CREATE OR REPLACE FUNCTION func_increment_plsql(i integer) RETURNS integer AS $$
        BEGIN
                RETURN i + 1;
        END;
$$ LANGUAGE plpgsql;
```

基础知识

— 创建自定义函数

SQL Server:

- 大致语法:

```
CREATE [ OR ALTER ] FUNCTION function_name （
      [ { @parameter [ AS ] data_type [ = default ] [ READONLY ] } [ ,...n ] ]
)
RETURNS data_type
[ AS ]
   BEGIN
      function_body
      RETURN expression
   END
[ ; ]
```

- 说明:

1. 返回值可以是表，具体参见文档或示例。
2. 直接使用 dbo.函数名(参数) 调用函数
3. 要在同一个sql文件里创建和执行函数的话，记得在创建语句后加GO。

- 相关文档:

https://docs.microsoft.com/zh-cn/sql/t-sql/statements/create-function-transact-sql?view=sql-server-2017

基础知识

—存储过程和自定义函数

- 存储过程和自定义函数的区别包括但不限于：

1. 一般来说，存储过程实现的功能要复杂一点，而函数的实现的功能针对性比较强。存储过程，功能强大，可以执行包括修改表等一系列数据库操作；用户定义函数不能用于执行一组修改全局数据库状态的操作。

2. 对于存储过程来说可以返回参数，如记录集，而函数只能返回值或者表对象。函数只能返回一个变量；而存储过程可以返回多个。存储过程的参数可以有IN,OUT,INOUT三种类型，而函数只能有IN，存储过程声明时不需要返回类型，而函数声明时需要描述返回类型，且函数体中必须包含一个有效的RETURN语句。

3. 存储过程，可以使用非确定函数，不允许在用户定义函数主体中内置非确定函数。

4. 存储过程一般是作为一个独立的部分来执行（ EXECUTE 语句执行），而函数可以作为查询语句的一个部分来调用（SELECT语句中调用），由于函数可以返回一个表对象，因此它可以在查询语句中位于FROM关键字的后面。 SQL语句中不可用存储过程，而可以使用函数。

……

基础知识（MYSQL版本）
— 存储过程示例

```
-- 1. 无参单句
DELIMITER $$
CREATE PROCEDURE Hello()
  SELECT 'Hello Procedure';
$$
DELIMITER ;
CALL Hello();

-- 2. 具名传参
DELIMITER $$
CREATE PROCEDURE Greet(
   IN msg TEXT ,
   IN Sname TEXT
)BEGIN
    IF Sname IS NULL THEN
        SET Sname = 'but who are you';
    END IF;
   SELECT CONCAT(msg,',',Sname,'!');
END;$$
DELIMITER ;
CALL Greet('Salve',null);
```

```
-- 3. 传入参数，语句块
DELIMITER $$
CREATE PROCEDURE Register(
   IN u VARCHAR(50) ,
   IN p VARCHAR(50)
)BEGIN
    IF EXISTS(SELECT * FROM account WHERE Username=u)THEN
        SELECT 'This username has been taken!';
    ELSE
        INSERT INTO account(Username,Password) VALUES(u,p);
    END IF;
END;$$
DELIMITER ;
CALL Register('kahsolt','233');
```

基础知识（MYSQL版本）
— 存储过程示例

```
-- 4.传出参数
DELIMITER $$
CREATE PROCEDURE MaxInRange(
    IN startIndex INT,
    IN endIndex INT,
    OUT ret INT
)BEGIN
    SELECT MAX(Val) INTO ret
        FROM Number
        WHERE ID >= startIndex AND ID <= endIndex;
END;$$
DELIMITER ;
CALL MaxInRange(10, 100, ret);
```

```
-- 5.无参，但能对外界产生影响
DELIMITER $$
CREATE PROCEDURE ShowAccounts()
BEGIN
  SELECT * FROM account;
  SELECT 'Done!';
END;$$
DELIMITER ;
CALL ShowAccounts();
```

基础知识（SQL SERVER）

— 存储过程示例

示例均基于如下假定：

```
-- 结构初始化
CREATE DATABASE Test;
USE Test;
GO
CREATE TABLE Account(
    Username varchar(50) PRIMARY KEY,
    Password varchar(50) NOT NULL
);
GO
CREATE TABLE Number(
    ID FLOAT PRIMARY KEY,
    Val FLOAT NOT NULL
);
GO
INSERT INTO Number(ID, Val)
    VALUES(0, 1.5), (10, 2.3), (50, 7.5), (100, 9.9), (150, 5.3);
GO
```

基础知识

— 存储过程示例

```
-- 1.无参单句
CREATE PROCEDURE Hello
AS
    PRINT 'Hello Procedure'
GO
EXECUTE Hello
GO

-- 2.具名传参，默认值
CREATE OR ALTER PROCEDURE Greet
    @msg TEXT,
    @name TEXT = 'but who are you'       --  设置缺省值
AS
    PRINT CONCAT(@msg, ', ', @name, '!')
GO
EXECUTE Greet 'Salve'                                   -- 使用默认值
EXECUTE Greet @name='Alitheia', @msg='Vale'      -- 自定义参数顺序
GO
```

```
-- 1.陷阱!!
CREATE PROCEDURE Hello
AS
    PRINT 'Hello Procedure'
EXECUTE Hello
GO
```

说明:
这样存储过程里会包含EXECUTE语句。

基础知识

— 存储过程示例

```
-- 3.传入参数，语句块
CREATE OR ALTER PROCEDURE Register(@u VARCHAR(50), @p VARCHAR(50))  -- 函数样式的参数表
AS
    BEGIN
        IF EXISTS(SELECT * FROM Account WHERE Username=@u)
            PRINT 'This username has been taken!'
        ELSE
            INSERT INTO Account(Username, Password) VALUES(@u, @p)
    END
GO
EXECUTE Register 'kahsolt', '233'    -- 调用不加括号
GO
```

说明:
CREATE OR ALTER为SQL SERVER2016 SP1新增语法
低版本会报错，创建单独用CREATE

基础知识

— 存储过程示例

```
-- 4.传出参数
CREATE OR ALTER PROCEDURE MaxInRange
    @startIndex INT,     -- 参数多而长时可不用括号(也是推荐写法！)
    @endIndex INT,       -- 而是展开为多行来书写
    @ret INT OUTPUT      -- 传出参数修饰字OUTPUT
AS
    SELECT @ret = MAX(Val)
        FROM Number
        WHERE ID >= @startIndex AND ID <= @endIndex
GO
DECLARE @ret AS FLOAT
EXECUTE MaxInRange 10, 100, @ret OUTPUT
PRINT CONCAT('MaxInRange is ', @ret)
GO
```

基础知识

— 存储过程示例

```
-- 5.无参，但能对外界产生影响
CREATE OR ALTER PROCEDURE ShowAccounts
AS
    SELECT * FROM Account    -- 比如产生一个结果集
    PRINT 'Done!'            -- 再打印一些log信息
GO
EXECUTE ShowAccounts
GO
```

基础知识（MYSQL版本）
— 自定义函数示例

```
-- 1.标量值函数
DELIMITER $$
CREATE FUNCTION Login(
    u VARCHAR(50),
    p VARCHAR(50)
)RETURNS VARCHAR(10)
BEGIN
    DECLARE res VARCHAR(10);
    IF EXISTS(SELECT * FROM account WHERE Username=u AND Password=p) THEN
        SET res = 'Success';
    ELSE
        SET res = 'Fail';
    END IF;
    RETURN res;
END;$$
DELIMITER ;
SELECT Login('kahsolt','233');
SELECT Login('misst','466');
```

说明：

MYSQL函数无法返回一个记录集，因此不便于实现内联表值函数/多语句表值函数；

如果需要完成相应功能，可以选用存储过程。

基础知识（SQL SERVER）
—自定义函数示例

```
-- 1.标量值函数
CREATE OR ALTER FUNCTION Login(      -- 函数样式参数表，必须有小括号！
    @u VARCHAR(50),
    @p VARCHAR(50)              -- 只能是传入参数(显然的)
) RETURNS VARCHAR(10) AS
BEGIN
    DECLARE @res AS VARCHAR(10)
    IF EXISTS(SELECT * FROM Account WHERE Username=@u AND Password=@p)
        SET @res = 'Success'
    ELSE
        SET @res = 'Fail'
    RETURN @res                 -- 最后一句必须是RETURN句
END
GO
PRINT dbo.Login('kahsolt', '233')       -- 调用加括号，注明dbo
PRINT dbo.Login('misst', '466')
GO
```

基础知识

—自定义函数示例

```
-- 2.内联表值函数
CREATE FUNCTION Top5Number()
RETURNS TABLE                -- 返回已存在的表或其子表
AS
    -- 函数不能直接对外界产生影响
    -- 因此不能像存储过程一样直接SELECT，只能RETURN数据给外界
    RETURN (SELECT TOP 5 * FROM Number ORDER BY Val)
GO
SELECT * FROM dbo.Top5Number();
GO
```

说明:

写sql存储过程经常需要调用一些函数来使处理过程更加合理，也可以使函数复用性更强，表值函数只能返回一个表，标量值函数可以返回基类型。

基础知识

—自定义函数示例

```
-- 3.多语句表值函数
CREATE FUNCTION SquareNumber()
RETURNS @T TABLE(          -- 返回新的临时表
    ID INT NOT NULL,
    Val FLOAT NOT NULL
) AS
BEGIN
    INSERT INTO @T SELECT ID,Val*Val FROM Number;
    RETURN  -- 不用写@T
END
GO
SELECT * FROM dbo.SquareNumber();
GO
```

说明：多语句表值函数可以看做是标量函数和内联表值函数的结合体。

触发器

- 语法（MySQL）

```
CREATE
    TRIGGER trigger_name
    trigger_time trigger_event
    ON tbl_name FOR EACH ROW
    [trigger_order]
    trigger_body

trigger_time: { BEFORE | AFTER }        //在DML之前/之后执行

trigger_event: { INSERT | UPDATE | DELETE }

trigger_order: { FOLLOWS | PRECEDES } other_trigger_name
//指定同一个表上不同触发器的执行顺序
```

触发器

- MySQL触发器使用详解
- https://www.cnblogs.com/qlqwjy/p/7842647.html

- MySQL中的SIGNAL语句生成自定义错误信息或警告
- https://blog.csdn.net/jkzyx123/article/details/135272894