批处理

> Using mysql in Batch Mode：https://dev.mysql.com/doc/refman/8.0/en/batch-mode.html
>
> Executing SQL Statements from a Text File：https://dev.mysql.com/doc/refman/8.0/en/mysql-batch-commands.html

mysql 命令的 Batch Mode

之前我们都是在终端交互式地使用 `mysql` 命令，每次输入一条语句，然后马上就能看到输出结果。

当需要重复、批量执行语句时，可以把需要执行的语句写到一个文本文件中，然后用以下命令执行：

```
mysql -h host -u user -p < batch-file
```

如果想要让部分语句失败时仍能继续执行，可以加上 `--force` 选项。

这种模式也方便语句的复制、粘贴、修改，如果是交互式使用可能需要重新输入语句。

对于输出结果比较多的情况，也可以输出到分页程序 `less` 或文件中：

```
mysql -h host -u user -p < batch-file | less
mysql -h host -u user -p < batch-file > mysql.out
```

假设创建一个文本文件 `batch.sql`：

```
use lab04

select CID
from Customer
where City = (
    select City
    from Customer
    where CID = 1
);
```

使用 mysql 命令的 batch mode 执行：

```
▶ mysql -u root -p < batch.sql
Enter password:
CID
1
3
```

batch mode 默认的输出格式与交互式不同，没有表格的边框线，可以通过 `-t` 选项切换到交互式输出格式：

```
▶ mysql -u root -p -t < batch.sql
Enter password:
+-----+
| CID |
+-----+
|   1 |
|   3 |
+-----+
```

可以通过 `-v` 选项输出执行的每条语句：

```
▶ mysql -u root -p -t -v < batch.sql
Enter password:
--------------
select CID
from Customer
where City = (
    select City
    from Customer
    where CID = 1
)
--------------

+-----+
| CID |
+-----+
```

```
|   1 |
|   3 |
+-----+
```

source 和 \. 命令

在交互式模式中，也可以通过 `source` 和 `\.` 命令执行文件中的 SQL 语句：

```
▶ mysql -u root -p
...
mysql> source batch.sql
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
+-----+
| CID |
+-----+
|   1 |
|   3 |
+-----+
2 rows in set (0.00 sec)

mysql> \. batch.sql
Database changed
+-----+
| CID |
+-----+
|   1 |
|   3 |
+-----+
2 rows in set (0.00 sec)
```

\! 命令

> mysql Client Commands：https://dev.mysql.com/doc/refman/8.0/en/mysql-commands.html

为了方便在批处理文件中插入调试信息，可以使用 `\!` 命令来执行系统终端命令 `echo`：

```
\! echo '[INFO] Start executing SQL statements'
```

用户变量

> User-Defined Variables：https://dev.mysql.com/doc/refman/8.0/en/user-variables.html

可以使用 SET 语句来定义用户变量：

```
SET @var_name = expr [, @var_name = expr] ...
```

- 用户变量以 `@` 开头，不区分大小写
- 如果不使用引号包围 `var_name`，允许的字符有：字母、数字、小数点 `.`、下划线 `_`、美元符号 `$`
- 必须初始化
- 支持的数据类型有：整型、decimal、浮点型、字符串

用户变量定义后就可以在其他语句中使用：

```
mysql> set @v1 = 1;
Query OK, 0 rows affected (0.01 sec)

mysql> select @v1, @v1 + 1, City
    -> from Customer
    -> where CID = @v1;
+------+---------+--------+
| @v1  | @v1 + 1 | City   |
+------+---------+--------+
|    1 |       2 | 北京   |
+------+---------+--------+
1 row in set (0.00 sec)
```

用户变量是属于当前会话 (session) 的，其他会话的用户无法看到其他会话的用户变量，结束回话后用户变量会被清除。

随机数函数

> Mathematical Functions：https://dev.mysql.com/doc/refman/8.0/en/mathematical-functions.html

返回 [0, 1) 之间的随机浮点数：

```
RAND([N])
```

- N 是随机数种子，不指定则使用默认的种子

```
mysql> select rand();
+--------------------+
| rand()             |
+--------------------+
| 0.5122167472841375 |
+--------------------+
1 row in set (0.00 sec)
```

返回 [i, j) 之间的随机整数：

```
FLOOR(i + RAND() * (j - i))
```

- [0, 100) 之间的随机整数

```
mysql> select floor(0 + rand() * (100 - 0));
+-------------------------------+
| floor(0 + rand() * (100 - 0)) |
+-------------------------------+
|                             8 |
+-------------------------------+
1 row in set (0.00 sec)
```

存储对象 Stored Objects

> Stored Objects：https://dev.mysql.com/doc/refman/8.0/en/stored-objects.html

存储对象包含以下类型：

- 存储过程 Stored Procedure
- 存储函数 Stored Function
- 触发器 Trigger
- 事件 Event
- 视图 View

术语之间的层次关系：

```
存储对象 Stored Objects
|- 存储程序 Stored Programs
    |- 存储例程 Stored Routines
        |- 存储过程 Stored Procedures
        |- 存储函数 Stored Functions
    |- 触发器 Triggers
    |- 事件 Events
|- 视图 Views
```

下一节要介绍的流程控制语句只能在存储程序 (Stored Programs) 中使用，所以使用比较存储函数 (Stored Functions) 和存储过程 (Stored Procedure) 进行演示和讲解。

存储过程和存储函数

> CREATE PROCEDURE and CREATE FUNCTION Statements：https://dev.mysql.com/doc/refman/8.0/en/create-procedure.html

CREATE PROCEDURE 语句可以用来定义存储过程，CREATE FUNCTION 语句可以用来定义存储函数：

```
CREATE
    [DEFINER = user]
    PROCEDURE sp_name ([proc_parameter[,...]])
```

```
    [characteristic ...] routine_body

CREATE
    [DEFINER = user]
    FUNCTION sp_name ([func_parameter[,...]])
    RETURNS type
    [characteristic ...] routine_body

proc_parameter:
    [ IN | OUT | INOUT ] param_name type

func_parameter:
    param_name type

type:
    Any valid MySQL data type

characteristic: {
    COMMENT 'string'
  | LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
}

routine_body:
    Valid SQL routine statement
```

比如创建一个表 `Customer`，创建一个存储过程 `insertCustomer` 用于插入一些数据：

```
create table Customer (
    CID int primary key,
    City char(20)
);

delimiter //

create procedure insertCustomer (city char(20))
  begin
    declare cid int default 0;
    while cid < 100 do
      insert into Customer values (cid, city);
      set cid = cid + 1;
    end while;
  end;//

delimiter ;
```

使用 `CALL` 语句执行存储过程：

```
call insertCustomer('Beijing');
```

比如创建一个 `hello` 函数，参数传入姓名，返回打招呼的字符串：

```
create function hello (name char(20))
    returns char(50) deterministic
    return concat('Hello, ', name, '!');
```

创建好之后就可以在表达式 (expression) 中使用存储函数：

```
mysql> select hello('world');
+----------------+
| hello('world') |
+----------------+
| Hello, world!  |
+----------------+
1 row in set (0.00 sec)
```

流程控制

> Flow Control Statements：https://dev.mysql.com/doc/refman/8.0/en/flow-control-statements.html

MySQL 支持在存储程序 (Stored Programs) 中使用以下语句：

- CASE
- IF
- ITERATE
- LEAVE
- LOOP
- REPEAT
- WHILE

支持在存储函数 (Stored Functions) 中使用以下语句：

- RETURN

CASE 语句

> CASE Statement：https://dev.mysql.com/doc/refman/8.0/en/case.html

```
CASE case_value
    WHEN when_value THEN statement_list
    [WHEN when_value THEN statement_list] ...
    [ELSE statement_list]
END CASE

CASE
    WHEN search_condition THEN statement_list
    [WHEN search_condition THEN statement_list] ...
    [ELSE statement_list]
END CASE
```

比如创建一个根据等级返回分数的函数 `get_score`，“优” 是 90 分，“良” 是 80 分，“中” 是 70 分，“及格” 是 60 分，其他返回 0 分：

```
delimiter //

create function get_score(grade char(10))
  returns int deterministic
  case grade
    when '优' then return 90;
    when '良' then return 80;
    when '中' then return 70;
    when '及格' then return 60;
    else return 0;
  end case;//

delimiter ;
```

`delimiter` 用于修改 MySQL 语句的定界符，使我们能够在函数中使用以分号结尾的语句。

`deterministic` 表示对于相同的输入，该函数总是返回相同的输出，目前必须指定（其他包含 SQL 语句的情况后续再说明）。

```
mysql> select get_score('良');
+------------------+
| get_score('良')  |
+------------------+
|               80 |
+------------------+
1 row in set (0.00 sec)
```

IF 语句

> IF Statement：https://dev.mysql.com/doc/refman/8.0/en/if.html

```
IF search_condition THEN statement_list
    [ELSEIF search_condition THEN statement_list] ...
    [ELSE statement_list]
END IF
```

创建一个比较两个数大小的函数，返回比较结果的字符串：

```
delimiter //

create function simple_cmp(n int, m int)
  returns varchar(40) deterministic
  begin
    declare s varchar(40);

    if n > m then set s = '>';
    elseif n = m then set s = '=';
    else set s = '<';
    end if;

    set s = concat(n, ' ', s, ' ', m);
    return s;
  end;//

delimiter ;
```

`begin` 和 `end` 是复合语句，用于包含多条语句。

```
mysql> select simple_cmp(1, 2);
+------------------+
| simple_cmp(1, 2) |
+------------------+
| 1 < 2            |
+------------------+
1 row in set (0.00 sec)
```

WHILE 语句

> WHILE Statement：https://dev.mysql.com/doc/refman/8.0/en/while.html

```
[begin_label:] WHILE search_condition DO
    statement_list
END WHILE [end_label]
```

创建一个计算 [0, 100] 的偶数个数的函数：

```
delimiter //

create function cnt_even()
  returns int deterministic
  begin
    declare curr, cnt int default 0;
    while curr <= 100 do
      set cnt = cnt + 1;
      set curr = curr + 2;
    end while;
    return cnt;
  end;//

delimiter ;
```

存储程序中局部变量定义的语法是：

```
DECLARE var_name [, var_name] ... type [DEFAULT value]
```

```
mysql> select cnt_even();
+------------+
| cnt_even() |
+------------+
|         51 |
+------------+
1 row in set (0.00 sec)
```