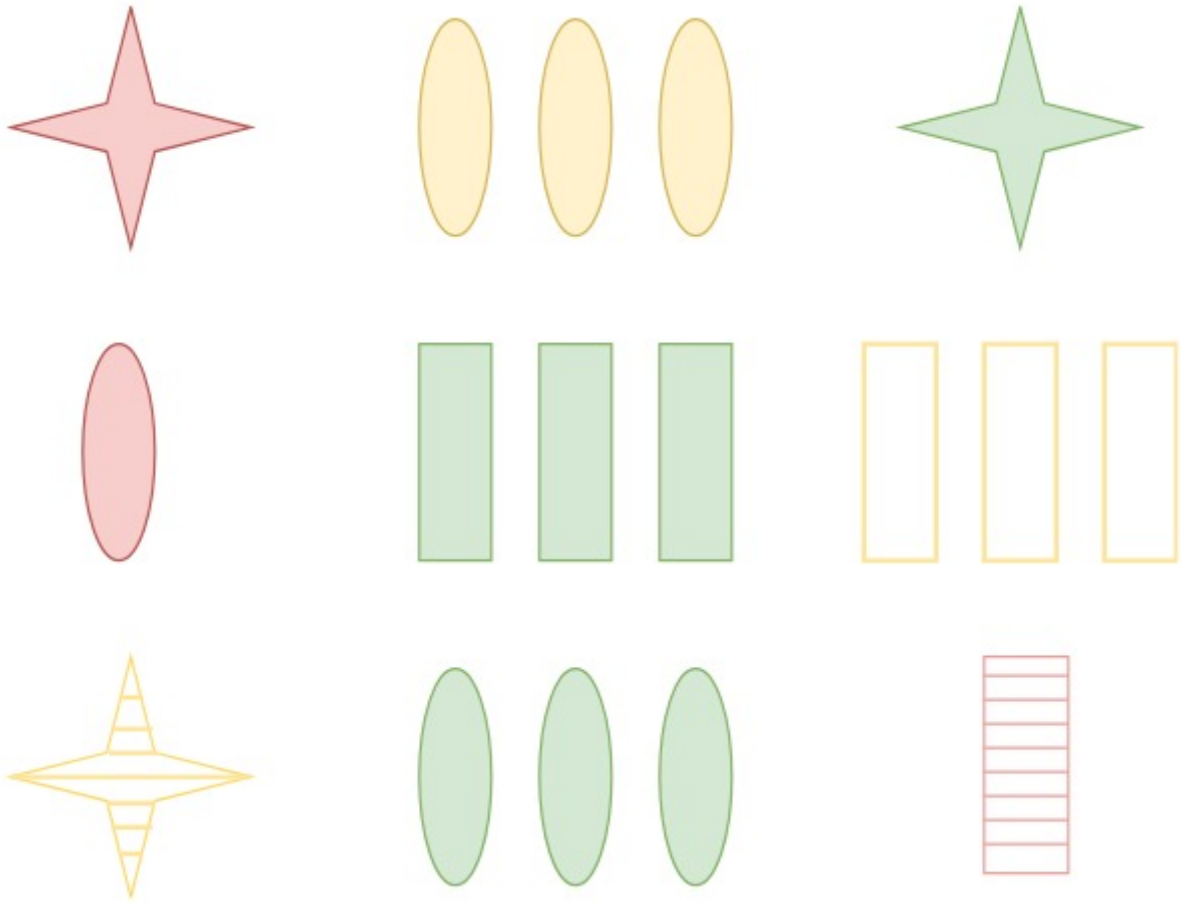
16丨游标：当我们需要逐条处理数据时，该怎么做？

我们在编写 SQL 语句的时候通常是面向集合进行思考，这种思考方式更让我们关注结果集的特征，而不是具体的实现过程。面向集合的思考方式与面向过程的思考方式各有特点，我们该如何理解它们呢？

我们用下面这张图开启今天的学习。这张图中一共有 9 个图形，每个图形有不同的特征，包括形状、纹理、颜色和个数等。



当我们看到这张图时，有时候会不由自主地按照某个属性进行分类，比如说按照红色分类，那么 1、4、9 就是一类。这实际上就是属于同一个条件下的查询结果集。或者我们也可以按照物体的个数来划分，比如都有 3 个物体的，那么对应的就是 2、5、6、8，这就是对应着“都包括 3 个物体”的查询结果集。

你能看出来集合思维更像是从整体的角度来考虑，然后把整个数据集按照不同的属性进行划分，形成不同的子集合。面向集合的思考方式，让我们关注“获取什么”，而不是“如何获取”，这也可以说是 SQL 与传统编程最大的区别之一，因为 SQL 本身是以关系模型和集合论为基础的。

然而也有一些情况，我们不需要对查询结果集中的所有数据行都采用相同的处理方式，需要每次处理一行或者一部分行，这时就需要面向过程的编程方法了。游标就是这种编程方式的体现。如果你之前已经有了一些面向过程的编程经验，那么对于游标的理解也会比较容易。

关于游标，你需要掌握以下几个方面的内容：

1. 什么是游标？我们为什么要使用游标？
2. 如何使用游标？使用游标的常用步骤都包括哪些？
3. 如何使用游标来解决一些常见的问题？

什么是游标？

在数据库中，游标是个重要的概念，它提供了一种灵活的操作方式，可以让我们从数据结果集中每次提取一条数据记录进行操作。游标让 SQL 这种面向集合的语言有了面向过程开发的能力。可以说，游标是面向过程的编程方式，这与面向集合的编程方式有所不同。

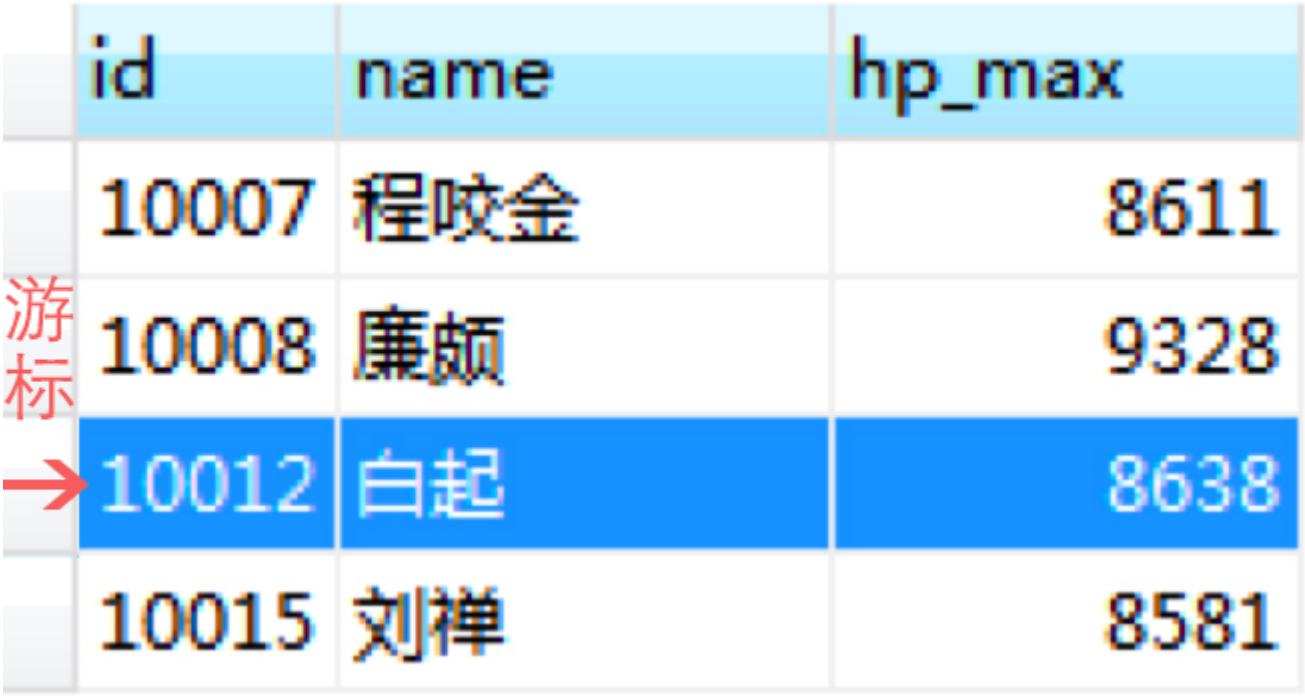
在 SQL 中，游标是一种临时的数据库对象，可以指向存储在数据库表中的数据行指针。这里游标充当了指针的作用，我们可以通过操作游标来对数据行进行操作。

比如我们查询了 heros 数据表中最大生命值大于 8500 的英雄都有哪些：

复制代码

```
SELECT id, name, hp_max FROM heros WHERE hp_max > 8500
```

查询结果（4 条数据）：



id	name	hp_max
10007	程咬金	8611
10008	廉颇	9328
10012	白起	8638
10015	刘禅	8581

这里我们就可以通过游标来操作数据行，如图所示此时游标所在的行是“白起”的记录，我们也可以在结果集上滚动游标，指向结果集中的任意一行。

如何使用游标?

游标实际上是一种控制数据集的更加灵活的处理方式。

如果我们想要使用游标，一般需要经历五个步骤。不同 DBMS 中，使用游标的语法可能略有不同。

第一步，定义游标。

复制代码

```
DECLARE cursor_name CURSOR FOR select_statement
```

这个语法适用于 MySQL，SQL Server，DB2 和 MariaDB。如果是用 Oracle 或者 PostgreSQL，需要写成：

复制代码

```
DECLARE cursor_name CURSOR IS select_statement
```

要使用 SELECT 语句来获取数据结果集，而此时还没有开始遍历数据，这里 select_statement 代表的是 SELECT 语句。

下面我用 MySQL 举例讲解游标的使用，如果你使用的是其他的 RDBMS，具体的游标语法可能略有差异。我们定义一个能够存储 heros 数据表中的最大生命值的游标，可以写为：

复制代码

```
DECLARE cur_hero CURSOR FOR
        SELECT hp_max FROM heros;
```

第二步，打开游标。

复制代码

```
OPEN cursor_name
```

当我们定义好游标之后，如果想要使用游标，必须先打开游标。打开游标的时候 SELECT 语句的查询结果集就会送到游标工作区。

第三步，从游标中取得数据。

复制代码

```
FETCH cursor_name INTO var_name ...
```

这句的作用是使用 cursor_name 这个游标来读取当前行，并且将数据保存到 var_name 这个变量中，游标指针指到下一行。如果游标读取的数据行有多个列名，则在 INTO 关键字后面赋值给多个变量名即可。

第四步，关闭游标。

复制代码

```
CLOSE cursor_name
```

有 OPEN 就会有 CLOSE，也就是打开和关闭游标。当我们使用完游标后需要关闭掉该游标。关闭游标之后，我们就不能再检索查询结果中的数据行，如果需要检索只能再次打开游标。

最后一步，释放游标。

复制代码

```
DEALLOCATE cursor_namec
```

有 DECLARE 就需要有 DEALLOCATE，DEALLOCATE 的作用是释放游标。我们一定要养成释放游标的习惯，否则游标会一直存在于内存中，直到进程结束后才会自动释放。当你不需要使用游标的时候，释放游标可以减少资源浪费。

上面就是 5 个常用的游标步骤。我来举一个简单的例子，假设我想用游标来扫描 heros 数据表中的数据行，然后累计最大生命值，那么该怎么做呢？

我先创建一个存储过程 calc_hp_max，然后在存储过程中定义游标 cur_hero，使用 FETCH 获取每一行的具体数值，然后赋值给变量 hp，再用变量 hp_sum 做累加求和，最后再输出 hp_sum，代码如下：

复制代码

```
CREATE PROCEDURE `calc_hp_max`()
BEGIN
        -- 创建接收游标的变量
        DECLARE hp INT;
        -- 创建总数变量
        DECLARE hp_sum INT DEFAULT 0;
        -- 创建结束标志变量
        DECLARE done INT DEFAULT false;
        -- 定义游标
        DECLARE cur_hero CURSOR FOR SELECT hp_max FROM heros;

        OPEN cur_hero;
        read_loop:LOOP
        FETCH cur_hero INTO hp;
        SET hp_sum = hp_sum + hp;
        END LOOP;
        CLOSE cur_hero;
        SELECT hp_sum;
END
```

你会发现执行`call calc_hp_max()`这一句的时候系统会提示 1329 错误，也就是在 LOOP 中当游标没有取到数据时会报的错误。

当游标溢出时（也就是当游标指向到最后一行数据后继续执行会报的错误），我们可以定义一个 continue 的事件，指定这个事件发生时修改变量 done 的值，以此来判断游标是否已经溢出，即：

复制代码

```
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = true;
```

同时在循环中我们需要加上对 done 的判断，如果游标的循环已经结束，就需要跳出 read_loop 循环，完善的代码如下：

复制代码

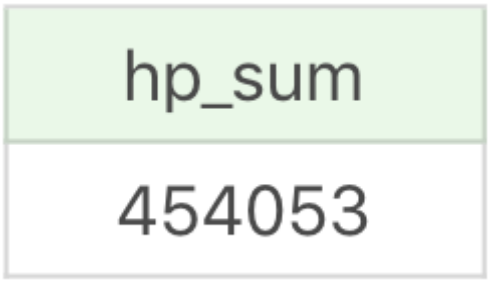
```
CREATE PROCEDURE `calc_hp_max`()
BEGIN
        -- 创建接收游标的变量
        DECLARE hp INT;

        -- 创建总数变量
        DECLARE hp_sum INT DEFAULT 0;
        -- 创建结束标志变量
      DECLARE done INT DEFAULT false;
        -- 定义游标
        DECLARE cur_hero CURSOR FOR SELECT hp_max FROM heros;
        -- 指定游标循环结束时的返回值
      DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = true;

        OPEN cur_hero;
        read_loop:LOOP
        FETCH cur_hero INTO hp;
        -- 判断游标的循环是否结束
        IF done THEN
                        LEAVE read_loop;
        END IF;

        SET hp_sum = hp_sum + hp;
        END LOOP;
        CLOSE cur_hero;
        SELECT hp_sum;
END
```

运行结果（1 行数据）：



在游标中的循环中，除了使用 LOOP 循环以外，你还可以使用 REPEAT... UNTIL...以及 WHILE 循环。它们同样需要设置 CONTINUE 事件来处理游标溢出的情况。

所以你能看出，使用游标可以让我们对 SELECT 结果集中的每一行数据进行相同或者不同的操作，从而很精细化地管理结果集中的每一条数据。

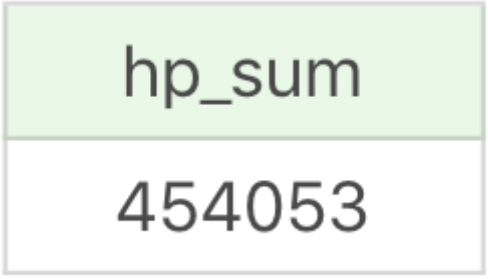
使用游标来解决一些常见的问题

我刚才讲了一个简单的使用案例，实际上如果想要统计 hp_sum，完全可以通过 SQL 语句来完成，比如：

复制代码

```
1 SELECT SUM(hp_max) FROM heros
```

运行结果（1 行数据）：



那么游标都有什么用呢？

当你需要处理一些复杂的数据行计算的时候，游标就会起到作用了。我举个例子，还是针对 heros 数据表，假设我们想要对英雄的物攻成长（对应 attack_growth）进行升级，在新版本中大范围提升英雄的物攻成长数值，但是针对不同的英雄情况，提升的幅度也不同，具体提升的方式如下。

如果这个英雄原有的物攻成长小于 5，那么将在原有基础上提升 7%-10%。如果物攻成长的提升空间（即最高物攻 attack_max- 初始物攻 attack_start）大于 200，那么在原有的基础上提升 10%；如果物攻成长的提升空间在 150 到 200 之间，则提升 8%；如果物攻成长的提升空间不足 150，则提升 7%。

如果原有英雄的物攻成长在 5—10 之间，那么将在原有基础上提升 5%。

如果原有英雄的物攻成长大于 10，则保持不变。

以上所有的更新后的物攻成长数值，都需要保留小数点后 3 位。

你能看到上面这个计算的情况相对复杂，实际工作中你可能会遇到比这个更加复杂的情况，这时你可以采用面向过程的思考方式来完成这种任务，也就是说先取出每行的数值，然后针对数值的不同情况采取不同的计算方式。

针对上面这个情况，你自己可以用游标来完成转换，具体的代码如下：

复制代码

```
CREATE PROCEDURE `alter_attack_growth`()
BEGIN
        -- 创建接收游标的变量
        DECLARE temp_id INT;
        DECLARE temp_growth, temp_max, temp_start, temp_diff FLOAT;

        -- 创建结束标志变量
        DECLARE done INT DEFAULT false;
        -- 定义游标
        DECLARE cur_hero CURSOR FOR SELECT id, attack_growth, attack_max, attack_start FI
        -- 指定游标循环结束时的返回值
        DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = true;

        OPEN cur_hero;
        FETCH cur_hero INTO temp_id, temp_growth, temp_max, temp_start;
        REPEAT
                        IF NOT done THEN
                                SET temp_diff = temp_max - temp_start;
```

```
                                IF temp_growth < 5 THEN
                                        IF temp_diff > 200 THEN
                                                SET temp_growth = temp_growth * 1.1;
                                        ELSEIF temp_diff >= 150 AND temp_diff <=200 THEN
                                                SET temp_growth = temp_growth * 1.08;
                                        ELSEIF temp_diff < 150 THEN
                                                SET temp_growth = temp_growth * 1.07;
                                        END IF;
                                ELSEIF temp_growth >=5 AND temp_growth <=10 THEN
                                        SET temp_growth = temp_growth * 1.05;
                                END IF;
                                UPDATE heros SET attack_growth = ROUND(temp_growth,3) WHERE
                        END IF;
        FETCH cur_hero INTO temp_id, temp_growth, temp_max, temp_start;
        UNTIL done = true END REPEAT;

        CLOSE cur_hero;
END
```

这里我创建了 alter_attack_growth 这个存储过程，使用了 REPEAT...UNTIL...的循环方式，针对不同的情况计算了新的物攻成长 temp_growth，然后对原有的 attack_growth 进行了更新，最后调用 call alter_attack_growth(); 执行存储过程。

有一点需要注意的是，我们在对数据表进行更新前，需要备份之前的表，我们可以将备份后的表命名为 heros_copy1。更新完 heros 数据表之后，你可以看下两张表在 attack_growth 字段上的对比，我们使用 SQL 进行查询：

复制代码

```
SELECT heros.id, heros.attack_growth, heros_copy1.attack_growth FROM heros JOIN heros_c
```

运行结果（69 条记录）：

id	attack_growth	attack_growth(1)
10000	11.57	11.57
10001	11	11
10002	10.57	10.57
10003	8.775	8.357
……	……	……
10068	15.86	15.86

通过前后两张表的 attack_growth 对比你也能看出来，存储过程通过游标对不同的数据行进行了更新。

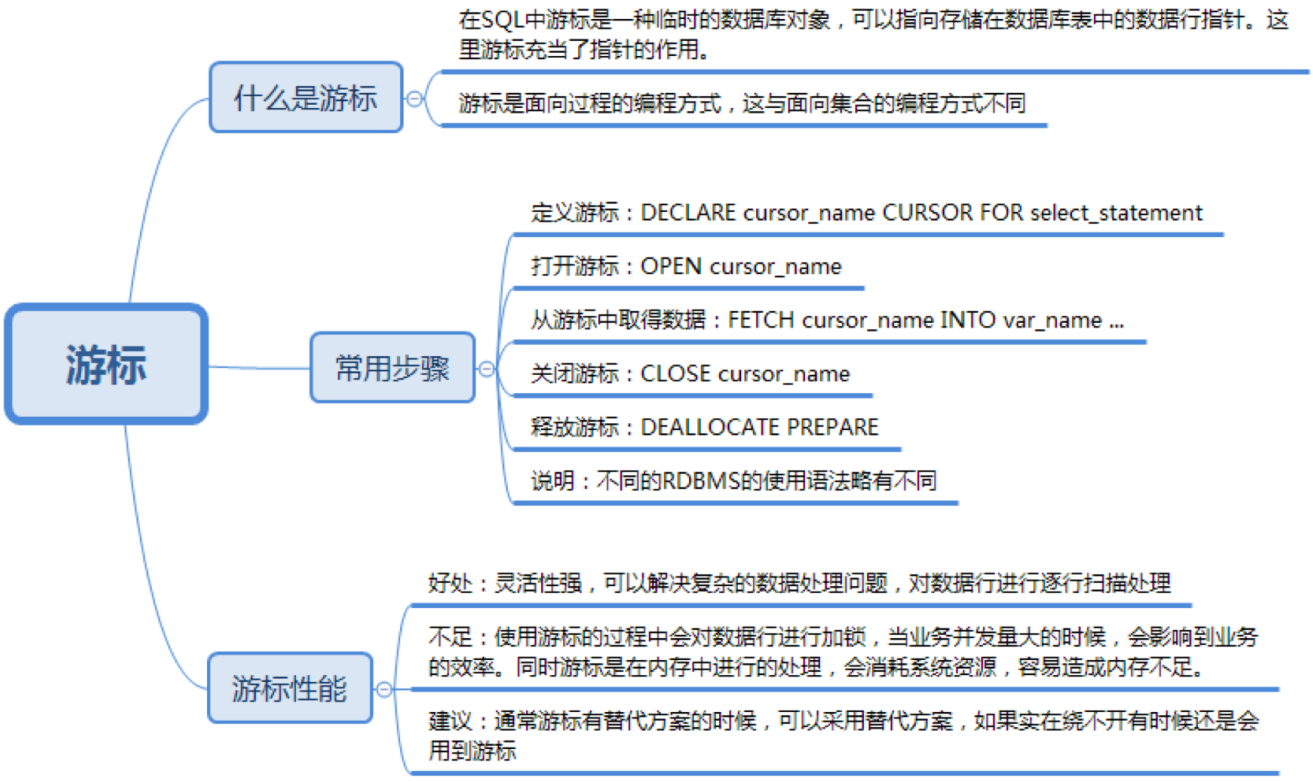
需要说明的是，以上代码适用于 MySQL，如果在 SQL Server 或 Oracle 中，使用方式会有些差别。

总结

今天我们讲解了如何在 SQL 中使用游标，游标实际上是面向过程的思维方式，与面向集合的思维方式不同的地方在于，游标更加关注“如何执行”。我们可以通过游标更加精细、灵活地查询和管理想要的数据行。

有的时候，我们需要找特定数据，用 SQL 查询写起来会比较困难，比如两表或多表之间的嵌套循环查找，如果用 JOIN 会非常消耗资源，效率也可能不高，而用游标则会比较高效。

虽然在处理某些复杂的数据情况下，使用游标可以更灵活，但同时也会带来一些性能问题，比如在使用游标的过程中，会对数据行进行加锁，这样在业务并发量大的时候，不仅会影响业务之间的效率，还会消耗系统资源，造成内存不足，这是因为游标是在内存中进行的处理。如果有游标的替代方案，我们可以采用替代方案。



我们今天讲解了游标，你能用自己的语言介绍下游标的作用吗？另外，我们之前提到过，SQL 本身是一门结构化查询语言，但我们也可以在 SQL 的基础上进行面向过程的开发，完成较为复杂的功能，你能说一下面向过程和面向集合这两种编程方式的区别吗？