第 17 章 程序设计

SQL 语言是操作关系型数据库的通用语言，是面向集合的非过程性语言，但很多情况下，数据库应用还需要面向过程功能进行复杂的处理，为了满足这种需要，不同的数据库产品一般都提供了面向过程的编程语言。Oracle 的编程语言为 PL/SQL，SQL Server 为 T-SQL。

本章主要内容包括：

- PL/SQL 和 T-SQL 简介
- 注释方式
- 程序基本结构
- 信息输出：Hello, world!
- 变量声明与赋值
- 条件处理
- 循环
- 异常处理

17.1 PL/SQL 和 T-SQL 简介

PL/SQL 表示“Procedural Language extensions to the Structured Query Language.”，即对 SQL 语言在面向过程功能方面的扩展(指对变量、条件转向、循环等一般编程语言所提供功能的支持)，是 Oracle 提供的、用于操作 Oracle 数据库的编程语言。1988 年的 Oracle 6 开始引入 PL/SQL 1.0，Oracle 7 升级到 2.0，从 Oracle 8 开始，PL/SQL 与 Oracle 数据库版本同步。PL/SQL 沿用了 Ada 编程语言的语法结构，而 Ada 又借用了 Pascal 的一些主要语法形式，如赋值、比较运算符以及单引号分隔的字符串。Ada 语言是为美国国防部开发的，主要用于实时、高安全的嵌入式系统，如飞机和导弹等。

对应于 Oracle 的 PL/SQL 语言，SQL Server 用于操作数据库的编程语言为 Transact-SQL，简称 T-SQL。SQL Server 软件本身源自 Sybase，T-SQL 的名称及其语法结构也源自 Sybase 的 T-SQL 语言。

17.1 注释方式

为了提高程序代码的可读性，必要的注释是不可少的。PL/SQL 与 T-SQL 的注释方法相同，分为多行注释和单行注释两种形式。

- /*…*/：用于多行注释，这与 C 语言的注释方式相同。
- --：用于单行注释。

17.2 程序基本结构

PL/SQL 程序的结构为：

```
[declare]
    declaration_statements
begin
```

execution_statements

[exception]

exception_handling_statements

end;

/

各个部分的功能如下：

- declare 部分：用于变量定义。
- begin…end 部分：用于程序处理以及异常处理。
- exception 部分：用于捕获和处理异常，类似 Java 中的 catch…finally 程序段。
- 分号：表示程序段的结束。

PL/SQL 程序中的语句以及程序段的最后都以分号作为结尾。

程序处理部分必须至少包含一个语句，如果不执行任何功能，则可以使用 null，如：

```
begin
    null;
end;
/
```

T-SQL 与 PL/SQL 不同，并没有固定的程序结构。

17.3 信息输出：Hello, world!

信息输出是编程语言的基本功能，本节以输出字符串“Hello, world”为例，说明 PL/SQL 和 T-SQL 如何在屏幕上输出信息。

17.3.1 PL/SQL 使用 dbms_output.put_line()

PL/SQL 使用 dbms_output.put_line()或 dbms_output.put()执行信息输出任务，输出内容包括字符串或变量值。dbms_output 是 Oracle 的一个包(package)，包可以看作执行相似任务的存储过程或函数集合。

- dbms_outpub.put_line()：输出字符串或变量值，并换行。
- dbms_output.put()：输出字符串或变量值，不换行。要继续执行 dbms_output.new_line 向输出缓冲区加入换行符，否则不会输出结果至屏幕。

为了使用 dbms_output.put_line()或 dbms_output.put()函数，首先要设置 SQL*Plus 的 serveroutput 环境变量为 on 以开辟输出缓冲区。

下面示例输出“Hello, world!”：

```
SQL> set serveroutput on
SQL> begin
  2     dbms_output.put_line('hello, world!');
  3  end;
  4  /
hello, world!

PL/SQL 过程已成功完成。
```

使用 dbms_output.put()函数完成同样效果：

```
SQL> begin
  2     dbms_output.put('Hello, ');
  3     dbms_output.new_line;
```

```
  4     dbms_output.put('world!');
  5     dbms_output.new_line;
  6  end;
  7  /
Hello,
world!

PL/SQL 过程已成功完成。
```

17.3.2 T-SQL 使用 print 或 select

SQL Server 使用 print 输出信息，不用额外配置：

```
1> print 'Hello, world!'
2> go
Hello, world!
```

17.4 变量声明与赋值

编程语言中，提供变量是其基本功能。变量的基本操作包括声明和赋值，PL/SQL 和 T-SQL 的变量声明和赋值语法有很大不同。

17.4.1 PL/SQL 的变量声明及赋值

PL/SQL 在程序段的 declare 部分进行变量声明，其语法形式为：

variable_name datatype;

另一种方式是使用%type，使变量直接继承相关列的数据类型：

variable_name table_name.column_name %type;

对变量赋值使用：

variable_name := value;

如果要把查询结果赋值给变量，则可以在 select 子句中使用下面形式赋值：

select *column_name* into *variable_name*

下面示例定义 myvar 变量为字符串类型，赋值后，输出其结果：

```
SQL> declare
  2     myvar varchar2(10);
  3  begin
  4     myvar:='world!';
  5     dbms_output.put_line('Hello, '|| myvar);
  6  end;
  7  /
Hello, world!

PL/SQL 过程已成功完成。
```

也可以在变量定义的同时，对其赋值：

```
SQL> declare
  2     myvar varchar2(10) :='world!';
  3  begin
  4     dbms_output.put_line('Hello, '|| myvar);
  5  end;
  6  /
Hello, world!
```

```
PL/SQL 过程已成功完成。
```

下面示例把 emp 表的 sal 列之和赋予变量，并输出其结果：

```
SQL> declare
  2     sumsal number(10,2);
  3  begin
  4     select sum(sal) into sumsal from emp;
  5     dbms_output.put_line(sumsal);
  6  end;
  7  /
24925

PL/SQL 过程已成功完成。
```

也可以在一个查询中把多个列值赋给多个变量：

```
SQL> declare
  2     dno emp.deptno%type;
  3     sumsal number(10,2);
  4  begin
  5     select  deptno, sum(sal) into dno, sumsal
  6     from emp
  7     where deptno=10
  8     group by deptno;
  9     dbms_output.put_line(dno||': '||sumsal);
 10  end;
 11  /
10: 8750

PL/SQL 过程已成功完成。
```

PL/SQL 的上述赋值方式要求查询结果不能为空也不能为多行，否则会出现异常。

17.4.2 T-SQL 的变量声明及赋值

T-SQL 的变量名称以@开始，声明变量使用 declare 关键字，可以在一个语句中声明多个变量，每个变量声明语句都要使用一个 declare 关键字。

变量定义的语法如下：

declare *variable_name* as *datatype*, *variable_name* as *datatype*, …

其中的 as 关键字可以省略。

如果赋值操作不涉及数据库中表的数据，则语法形式比较简单，赋值使用 set 或 select，使用 select 赋值时，不会输出任何信息：

set *variable_name=value*

select *variable_name=value*

也可以在变量定义时对其赋值：

declare *variable_name* as *datatype =value*

把查询结果赋值给变量，T-SQL 的语法形式与普通查询中使用列别名的用法类似，即在查询语句的 select 子句中使用下面语法形式赋值：

select *variable_name=column_name*

要注意，如果查询的结果为多行，则只会把最后一行的相应列值赋给变量，这与 PL/SQL 的处理方式不同，在 PL/SQL 中，不允许把多行查询结果赋给变量。

下面示例定义@i，@j 以及@k 三个整型变量，对@i 及@j 赋值，求和后赋值给@k，然后输出结果：

```
1> declare @i as int
2> declare @j as int
3> declare @k as int
4> set @i=3
5> set @j=2
6> set @k=@i+@j
7> print @k
8> go
5
```

也可以修改为：

```
1> declare @i as int=3, @j as int=2, @k as int
2> set @k=@i+@j
3> print @k
4> go
5
```

上述两个示例中的赋值语句可以改为使用 select 关键字：

```
1> declare @i as int=3, @j as int=2, @k as int
2> select @k=@i+@j
3> print @k
4> go
5
```

下面示例沿用上节 PL/SQL 中的内容，把查询结果赋值给变量。

把 emp 表的 sal 总和赋值给@sumsal，并打印其结果：

```
1> declare @sumsal as numeric(10,2)
2> select @sumsal=sum(sal) from emp
3> print @sumsal
4> go
24925.00
```

也可以在一个查询中，给多个变量同时赋值：

```
1> declare @sumsal as numeric(10,2), @dno as tinyint
2> select @dno=deptno, @sumsal=sum(sal)
3> from emp
4> where deptno=10
5> group by deptno
6> print cast(@dno as varchar)+': '+cast(@sumsal as varchar)
7> go
10: 8750.00
```

如果删除上述代码中第 4 行的 where 条件，则查询结果会出现多行，只把最后一行的列值赋给变量，不会出现异常，如果查询结果为空，则变量会被赋值为 null，也不会出现异常。

17.5 条件处理

与普通编程语言相似，PL/SQL 和 T-SQL 的条件处理语句使用 if...else 实现。

17.5.1 PL/SQL 中的条件处理

简单的条件处理可以采用下面的语法形式：

if *condition1* then

statements1;

else

statements2;

end if;

下面示例对当前日期加一天，然后检查两个日期的年份是否相同来判断当前日期是否为当年的最后一天：

```
SQL> begin
  2     if(extract(year from current_timestamp)
  3         <> extract(year from current_timestamp+1))
  4     then
  5        dbms_output.put_line('Today is the last day of the year');
  6     else
  7        dbms_output.put_line('Today is not the last day of the year');
  8     end if;
  9  end;
 10  /
Today is not the last day of the year

PL/SQL 过程已成功完成。
```

也可以在 if 语句中嵌入 elsif 语句进行多重条件判断：

if *condition1* then

statements1;

elsif *condition2* then

statements2;

elsif *condition3* then

statements3;

…

else

statementsn;

end if ;

下面示例使用 if … else 嵌套判断以下几个方面是否为真：

今天是当年的最后一天。

今天是当月的最后一天，但不是当年的最后一天。

今天不是当月的最后一天。

```
SQL> begin
  2     if extract(year from current_timestamp)
  3         <> extract(year from current_timestamp+1)
  4     then
  5        dbms_output.put_line( 'Today is the last day of the year.');
  6        elsif extract(month from current_timestamp)
  7                 <> extract(month from current_timestamp+1)
  8        then
  9            dbms_output.put_line( 'Today is last day of the month' ||
 10                                    'but not the last day of the year.');
 11     else
 12        dbms_output.put_line( 'Today is not the last day of the month.');
 13  end if;
 14  end;
 15  /
Today is not the last day of the month.
```

```
PL/SQL 过程已成功完成。
```

17.5.2 T-SQL 中的条件处理

如果执行单个条件判断，可以使用 if … else 的简单形式：

if *condition*
    statement1
else
    statement2

若条件为真或假时需要执行多条语句，可以使用 begin … end 将其括住，从而把要执行的语句作为一个执行单元，这在 PL/SQL 中不需要。

与上节 Oracle 的示例要求相同，下面示例也对当前日期加一天，然后检查两个日期的年份是否相同来判断当前日期是否为当年的最后一天：

```
1> if year(current_timestamp) <> year(current_timestamp+1)
2>    print 'Today is the last day of the year.'
3> else
4>    print 'Today is not the last day of the year.'
5> go
Today is not the last day of the year.
```

进行多条件判断时，if … else 可以多重嵌套：

if *condition1*
    statements1
else
  if *condition2*
     statements2
  else
     statements3

下面示例使用 if … else 嵌套，完成与上节 PL/SQL 程序的多重条件判断相同的功能：

今天是当年的最后一天。

今天是当月的最后一天，但不是当年的最后一天。

今天不是当月的最后一天。

```
1> if year(current_timestamp) <> year(current_timestamp+1))
2>    print 'Today is the last day of the year.'
3> else
4> if month(current_timestamp) <> month(current_timestamp+1)
5>    print 'Today is last day of the month but not the last day of the year.'
6> else
7>    print 'Today is not the last day of the month.'
8> go
Today is not the last day of the month.
```

17.5.3 case 语句

case 语句可以在 Oracle 或 SQL Server 中用于 select 查询，根据列的不同取值显示不同内容，两者用法相同，都符合 SQL 标准。

另外，Oracle 的 case 语句还可用于 PL/SQL 程序语句中的多条件判断，与一般编程语言中的 case 关键字用法类似，这种用法 T-SQL 不支持。

case 语句用于 select 语句有两种方式：

case *column_name* when *m1* then *n1*
when *m2* then *n2*
……
end
及
case when *column_condition1* then *n1*
when *column_condition2* then *n2*
……
end
显然第二种方式的功能更强。
下面示例是在 SQL Server 上使用 case 的第一种方式把 deptno 转换为相应的字符串：

```
1> select ename,  deptno,
2>        case deptno
3>            when 10 then 'ACCOUNTING'
4>            when 20 then 'DALLAS'
5>            when 30 then 'CHICAGO'
6>            else 'UNKOWN'
7>        end as loc
8> from emp
9> where sal>1700
10> go
ename      deptno loc
---------- ------ ----------
JONES          20 DALLAS
BLAKE          30 CHICAGO
CLARK          10 ACCOUNTING
KING           10 ACCOUNTING
FORD           20 DALLAS

(5 行受影响)
```

使用 case 的第二种方式：

```
1> select ename, sal,
2>        case when sal>1700 then 'high salary'
3>             when sal<=1700 then 'low salary'
4>        end as sal_grade
5> from emp
6> where deptno=30
7> go
ename      sal       sal_grade
---------- --------- -----------
ALLEN        1600.00 low salary
WARD         1250.00 low salary
MARTIN       1250.00 low salary
BLAKE        2850.00 high salary
TURNER       1500.00 low salary
JAMES         950.00 low salary

(6 行受影响)
```

上面的命令可以不加修改地在 Oracle 正确执行，这里不再举例。
在程序语句中使用 case 的方式与 select 语句相似，只是这里的条件一般不包含列名称，如下面示例所示：

```
SQL> set serveroutput on
SQL> declare a number:=1;
  2  begin
  3     case a when 1 then dbms_output.put_line('a=1');
  4            when 2 then dbms_output.put_line('a=2');
  5            else dbms_output.put_line('unkown');
  6     end case;
  7  end;
  8  /
a=1

PL/SQL 过程已成功完成。
```

17.6 循环

循环结构用于在指定条件下多次执行相同的代码，使得我们可以完成复杂的数据处理逻辑，是面向过程程序设计语言的重要特征。

17.6.1 PL/SQL 中的循环

PL/SQL 循环可以使用 loop、for、while 三种语法结构，每种语法结构又有不同的变形。

loop 循环代码块以 loop 开始，以 end loop 结束。用 exit、exit when 或 return 结束循环。使用 exit 语句退出循环，要与 if 语句结合使用，而 exit when 语句则可以直接把退出循环的条件放在其后。下面以得出前 100 个正整数之和为例，说明 exit when 形式的用法。

```
SQL> declare
  2     mysum int:=0;
  3     i int:=1;
  4  begin
  5     loop
  6        exit when i>100;
  7        mysum:=mysum+i;
  8        i:=i+1;
  9     end loop;
 10     dbms_output.put_line('sum='||mysum);
 11  end;
 12  /
sum=5050

PL/SQL 过程已成功完成。
```

for 循环结构有两种用法，一种是给出循环变量的起始范围，另一种是遍历查询结果集。这两种用法都不需要给出退出循环的条件，也不用声明循环变量。

用 for 循环的第一种用法，再次执行上面的求和功能：

```
SQL> declare
  2     mysum int:=0;
  3  begin
  4     for i in 1..100
  5     loop
  6        mysum:=mysum+i;
  7     end loop;
  8     dbms_output.put_line('sum='||mysum);
  9  end;
 10  /
sum=5050
```

```
PL/SQL 过程已成功完成。
```

使用这种形式时，循环变量 i 不用预先定义，其增长步长不能改变，只能为 1，用户只能设置其初值和终值。在循环体外部不能引用循环变量的值，如果要在循环体外部引用其值，则要在循环体内先把其值传递给另外一个变量。如下面示例所示：

```
SQL> declare
  2     j int;
  3     mysum int:=0;
  4  begin
  5     for i in 1..100 loop
  6        mysum:=mysum+i;
  7        j:=i;
  8     end loop;
  9     dbms_output.put_line('sum='||mysum||', i='||j);
 10  end;
 11  /
sum=5050, i=100

PL/SQL 过程已成功完成。
```

如果要从大到小遍历，可以附加 reverse 关键字，即：for i in reverse 1..100 loop

用 for 循环的第二种用法遍历查询结果集：

```
SQL> begin
  2     for dept_rec in (select * from dept)
  3     loop
  4        dbms_output.put_line(dept_rec.dname);
  5     end loop;
  6  end;
  7  /
ACCOUNTING
RESEARCH
SALES
OPERATIONS

PL/SQL 过程已成功完成。
```

while 循环形式与 loop 相似。下面示例使用 while 循环形式求出前 100 个自然数的和：

```
SQL> declare
  2     i int:=1;
  3     mysum int:=0;
  4  begin
  5     while(i<=100) loop
  6        mysum:=mysum+i;
  7        i:=i+1;
  8     end loop;
  9     i:=i-1;
 10     dbms_output.put_line('sum='||mysum||', '||'i='||i);
 11  end;
 12  /
sum=5050, i=100

PL/SQL 过程已成功完成。
```

17.6.2 T-SQL 中的循环

T-SQL 中的循环语句只有 while 一种语法形式，其结构如下所示：

while *condition*
begin
    statements
end

与上节示例功能类似，下面使用 T-SQL 循环得到前 100 个自然数的和：

```
1> declare @i as int, @sum as int;
2> set @i=1;
3> set @sum=0;
4> while @i<=100
5> begin
6>    set @sum=@sum+@i;
7>    set @i=@i+1;
8> end
9> print 'The sum is: '+cast(@sum as varchar);
10> go
The sum is: 5050
```

17.6.3 break 与 continue

一般程序设计语言对于循环处理都提供了 break 及 continue 语句，break 用于满足指定条件时退出循环，continue 语句用于满足指定条件时，跳过循环中的部分语句。

PL/SQL 不支持 break 关键字，可以使用 exit 代替：

```
SQL> declare
  2     i int:=0;
  3     mysum int:=0;
  4  begin
  5     while(i<=99) loop
  6        i:=i+1;
  7        mysum:=mysum+i;
  8        if(mysum>3000) then
  9           exit;
 10        end if;
 11     end loop;
 12     dbms_output.put_line('sum='||mysum||', '||'i='||i);
 13   end;
 14   /
sum=3003, i=77

PL/SQL 过程已成功完成。
```

在 while 循环中使用 continue，求出前 100 个自然数中不是 9 的倍数的和：

```
SQL> declare
  2     i int:=0;
  3     mysum int:=0;
  4  begin
  5     while(i<=99) loop
  6        i:=i+1;
  7        if(mod(i,9)=0) then
  8           continue;
  9        end if;
 10        mysum:=mysum+i;
 11     end loop;
```

```
 12     dbms_output.put_line('sum='||mysum||', '||'i='||i);
 13  end;
 14  /
sum=4456, i=100

PL/SQL 过程已成功完成。
```

T-SQL 对于 break 及 continue 都是支持的。

如果要在满足指定条件时退出循环，可以使用 break 关键字。

求前 100 个自然数的和，如果和大于 3000，则退出循环并输出此时的和以及最后一个作为加数的自然数：

```
1> declare @i as int, @sum as int;
2> set @i=1;
3> set @sum=0;
4> while @i<=100
5> begin
6>    set @sum=@sum+@i;
7>    set @i=@i+1;
8>    if(@sum>3000) break;
9> end
10> print 'sum='+cast(@sum as varchar)+',  i='+cast(@i-1 as varchar);
11> go
sum=3003,  i=77
```

如果在循环执行过程中，当某个条件满足时，需要跳过某些步骤，则可以在这些步骤之前使用 continue 关键字。

下面示例求前 100 个自然数的和，但不包括 9 的倍数：

```
1> declare @i as int, @sum as int;
2> set @i=0;
3> set @sum=0;
4> while @i<=99
5> begin
6>    set @i=@i+1;
7>    if(@i%9=0) continue;
8>    set @sum=@sum+@i;
9> end
10> print 'sum= '+cast(@sum as varchar)+',  i='+cast(@i as varchar);
11> go
sum= 4456,  i=100
```

在 PL/SQL 中求余数使用 mod 函数，在 T-SQL 中使用%运算符。

17.7 异常处理

异常通常指不该发生的错误。一个应用上线运行时，不能指望终端用户的输入总是符合开发者的期望，也不能要求服务器总是正常可用的，这些异常情况都需要程序员提前考虑到，并给出相应的处理措施。程序设计语言使用异常处理跟踪和回应异常，若程序员编写了异常处理代码，当程序执行过程中出现异常时，会由异常处理代码捕捉到，程序会跳转至异常处理模块，执行其中的内容。

异常通常分为系统预定义异常及用户自定义异常两类。

17.7.1 PL/SQL 的 exception … when

PL/SQL 程序段在 begin … end 中位于最后的 exception 部分解决异常问题。

先给出一个简单的示例，说明如何在 exception 部分捕获被 0 除时的异常，下面示例 0 未作除数，不会出现异常：

```
SQL> set serveroutput on
SQL> begin
  2     dbms_output.put_line(10/2);
  3     dbms_output.put_line('No error.');
  4  exception
  5     when zero_divide then
  6        dbms_output.put_line('Error! Divided by zero.');
  7  end;
  8  /
5
No error.

PL/SQL 过程已成功完成。
```

把上面的除数改为 0，再重新执行：

```
SQL> begin
  2     dbms_output.put_line(10/0);
  3     dbms_output.put_line('No error.');
  4  exception
  5     when zero_divide then
  6        dbms_output.put_line('Error! Divided by zero.');
  7  end;
  8  /
Error! Divided by zero.

PL/SQL 过程已成功完成。
```

关键字 when 用于给出异常类型，then 后面的语句用于对异常的处理。如果存在多种异常情况，则可以在以上 exception 部分使用多个 when 语句分别处理。

如果程序中不包含上述捕获异常的代码，则执行时会出现下列错误而终止：

```
SQL> begin
  2     dbms_output.put_line(10/0);
  3  end;
  4  /
begin
*
第 1 行出现错误:
ORA-01476: 除数为 0
ORA-06512: 在 line 2
```

when 后面的异常类型可以是 Oracle 预定义(在 sys.standard 中定义)也可以是用户自定义，上面示例所使用的 zero_divide 即为 Oracle 预定义异常类型。

如果要捕获其他非预定义也非用户自定义的异常，则可以使用 others 关键字。

下面示例使用了 Oracle 预定义异常类型 value_error 来捕获字符串赋值时的越界异常：

```
SQL> declare
  2     a varchar2(1);
  3     b varchar2(2):='AB';
  4  begin
  5     a:=b;
```

```
  6  exception
  7     when value_error then
  8    dbms_output.put_line('Can''t put ['||b||'] in a one-character string.');
  9  end;
 10  /
Can't put [AB] in a one-character string.

PL/SQL 过程已成功完成。
```

下面程序段会出现两个异常，而异常处理部分只能捕获后一个：

```
SQL> declare
  2      a varchar2(1);
  3  begin
  4      declare
  5          b varchar2(2);
  6      begin
  7          select 'AB' into b from dual where 1=2;
  8          a:=b;
  9       exception
 10           when value_error then
 11           dbms_output.put_line('Can''t put ['||b||'] in a one-character string.');
 12       end;
 13  end;
 14  /
declare
*
第 1 行出现错误:
ORA-01403: 未找到任何数据
ORA-06512: 在 line 7
```

在异常捕获部分添加 others 捕获其他异常则可以解决这个问题：

```
SQL> declare
  2      a varchar2(1);
  3  begin
  4      declare
  5          b varchar2(2);
  6      begin
  7          select 'AB' into b from dual where 1=2;
  8          a:=b;
  9       exception
 10         when value_error then
 11         dbms_output.put_line('Can''t put ['||b||'] in a one-character string.');
 12       end;
 13       exception
 14           when others then
 15           dbms_output.put_line('Errors:  ['||SQLERRM||'].');
 16  end;
 17  /
Errors:  [ORA-01403: 未找到任何数据].

PL/SQL 过程已成功完成。
```

捕获预定义异常比较简单，只要用户在异常处理部分指定这种异常即可。如果发生了相关联的错误，那么就将捕获这个异常，从而不会传播到外部。不过，有时候可能会发生非预定义的异常,这种情况下,用户必须明确声明这个异常,然后可以通过异常处理部分捕获它，这种异常一般称为用户自定义异常。

要捕获一个没有预定义的异常时，可以执行下面三个步骤：

- 在 PL/SQL 块的声明部分给出异常的名称，并且为异常指定数据类型为 exception。
- 使用 pragma exception_init 语句将所声明的异常与 Oracle 服务器错误代码相关联。
- 在 PL/SQL 的异常处理部分包含所声明的异常。

scott 用户的 dept 表的主键为 deptno，如果插入记录的 deptno 的值与表中已有的值重复，则违反了主键约束，这时 Oracle 会返回服务器错误代码 ORA_00001，如下所示：

```
SQL> conn scott/tiger
已连接。
SQL> insert into dept values(10,'abc','def');
insert into dept values(10,'abc','def')
*
第 1 行出现错误:
ORA-00001: 违反唯一约束条件 (SCOTT.PK_DEPT)
```

如果我们用一个 PL/SQL 块向 dept 表添加记录，要在 PL/SQL 块中捕获这种错误，可以按照上面所说的步骤，用下面代码实现：

```
SQL> conn scott/tiger
已连接。
SQL> set serveroutput on
SQL> declare
  2   deptno_already_in_use exception;
  3   pragma exception_init(deptno_already_in_use,-00001);
  4  begin
  5    insert into dept values(10,'abc','def');
  6  exception
  7    when deptno_already_in_use then
  8       dbms_output.put_line('Choose another deptno!');
  9  end;
 10  /
Choose another deptno!

PL/SQL 过程已成功完成。
```

17.7.2 T-SQL 的 try … catch

在 T-SQL 程序中，把可能出现异常的程序语句放置于 try 部分(begin try … end try)，错误处理语句放置于 catch 部分(begin catch … end catch)，如果在 try 部分未发生错误，则 catch 部分的语句会被忽略，若 try 部分的语句发生了异常，则程序控制转移至相应的 catch 部分。

下面语句在 try 部分不会发生错误，catch 部分的语句也不会被执行：

```
1> begin try
2>    print 10/2;
3>    print 'No error.';
4> end try
5> begin catch
6>    print 'Error! Divided by zero.';
7> end catch
8> go
5
No error.
```

把上面的除数 2 改为 0，显然会有错误，catch 部分的代码被执行：

```
1> begin try
2>    print 10/0;
3>    print 'No error.';
```

```
4> end try
5> begin catch
6>    print 'Error! Divided by zero.';
7> end catch
8> go
Error! Divided by zero.
```

在一般 T-SQL 程序设计中，编程者应该在 catch 部分分析错误原因，然后给出相应提示信息或其他处理语句。

catch 部分捕获到的错误以错误号来标识，编程者通过调用 error-number()函数得到捕获到的错误号，然后根据错误号给出相关错误提示信息，或者调用 error_message()函数显示系统预先定义的错误信息，SQL Server 的所有预定义错误号及对应错误信息可以通过查询 sys.messages 得到。另外，SQL Server 不支持用户自定义异常。

我们用下面示例说明 try … catch 的用法。

创建表 t，用于测试：

```
1> create table t
2> (
3>    a int not null,
4>    b int not null,
5>    c char(4),
6>    constraint pk_t primary key(a),
7>    constraint ck_b check(b>100)
8> )
```

然后执行下面程序段：

```
1> begin try
2>    insert into t values(1, 170, 'a');
3> end try
4> begin catch
5>    if error_number()=2627
6>    begin
7>       print 'Handling PK violation...';
8>    end
9>    else if error_number()=547
10>   begin
11>      print 'Handling CHECK/FK constraint violation...';
12>   end
13>   else if error_number()=515
14>   begin
15>      print 'Handling NULL violation...';
16>    end
17>    else if error_number()=245
18>    begin
19>       print 'Handling conversion error...';
17>    end
21>    else
22>    begin
23>       print 'Handling unknown error...';
24>    end
25>
26>    print 'Error Number  : '+cast(error_number() as varchar(10));
27>    print 'Error Message : '+error_message();
28>    print 'Error Severity: '+cast(error_severity() as varchar(10));
29>    print 'Error State   : '+cast(error_state() as varchar(10));
30>    print 'Error Line    : '+cast(error_line() as varchar(10));
31>    print ' Error Proc  : '+coalesce(error_procedure(), 'Not within proc');
```

```
32> end catch
33> go

(1 行受影响)
```

catch 部分的前一部分是根据捕获的错误号，给出相关提示信息，后一部分通过调用相关系统函数给出捕获到的错误的其他信息。

第一次执行上述程序时，不会出现错误，catch 部分的语句也不会调用。

如果第二次执行上面的语句，则会因为主键存在重复值而给出以下错误信息：

```
Handling PK violation...
Error Number  : 2627
Error Message : 违反了 PRIMARY KEY 约束 'pk_t'。不能在对象'dbo.t'中插入重复键。
 Error Severity: 14
 Error State   : 1
 Error Line    : 2
 Error Proc    : Not within proc
```

如果把添加记录的 insert 语句改为：insert into t values(2, 10, 'a');

重新执行上述程序段，则违反了 b 字段上的 check 约束，会给出以下错误信息：

```
Handling CHECK/FK constraint violation...
Error Number  : 547
Error Message : INSERT 语句与 CHECK 约束"ck_b"冲突。该冲突发生于数据库"law", 表"dbo.t", column 'b'。
Error Severity: 16
Error State   : 0
Error Line    : 2
Error Proc    : Not within proc
```

如果要使得错误处理的代码可以重用，则可以把上述 catch 部分的内容创建为存储过程，以后即可在 T-SQL 代码中直接调用。

创建存储过程如下：

```
1> create procedure dbo.err_message
2> as
3> if error_number()=2627
4> begin
5>      print 'Handling PK violation...';
6> end
7> else if error_number()=547
8> begin
9>    print 'Handling CHECK/FK constraint violation...';
10> end
11> else if error_number()=515
12> begin
13>    print 'Handling NULL violation...';
14> end
15> else if error_number()=245
16> begin
17>    print 'Handling conversion error...';
18> end
19> else
20> begin
21>     print 'Handling unknown error...';
22> end
23>
24> print 'Error Number  : '+cast(error_number() as varchar(10));
25> print 'Error Message : '+error_message();
```

```
26> print 'Error Severity: '+cast(error_severity() as varchar(10));
27> print 'Error State   : '+cast(error_state() as varchar(10));
28> print 'Error Line    : '+cast(error_line() as varchar(10));
29> print 'Error Proc    : '+coalesce(error_procedure(),'Not within proc');
30> go
```

在 T-SQL 程序段中调用上述存储过程：

```
1> begin try
2>    insert into t values(2, 10, 'a');
3> end try
4> begin catch
5>    exec err_message;
6> end catch
7> go

(0 行受影响)
Handling CHECK/FK constraint violation...
Error Number  : 547
Error Message : INSERT 语句与 CHECK 约束"ck_b"冲突。该冲突发生于数据库"law", 表"dbo.t", column 'b'。
Error Severity: 16
Error State   : 0
Error Line    : 2
Error Proc    : Not within proc
```