# PL/SQL

## 什么是 PL/SQL

PL/SQL（Procedure Language/SQL）是 Oracle 对 sql 语言的过程化扩展，指 在 SQL 命令语言中增加了过程处理语句（如分支、循环等），使 SQL 语言具有 过程处理能力。把 SQL 语言的数据操纵能力与过程语言的数据处理能力结合起来，使得 PLSQL 面向过程但比过程语言简单、高效、灵活和实用。

基本语法结构

|  |
| --- |
| [declare  --声明变量  ]  begin  --代码逻辑  [exception  --异常处理  ]  end; |

## 变量

声明变量的语法：

|  |
| --- |
| 变量名 类型（长度）； |

变量赋值的语法：

|  |
| --- |
| 变量名:=变量值 |

### 变量的声明案例

|  |
| --- |
| -- todo 目标: 掌握变量的用法  -- todo 1 声明变量水费单价 v\_price、水费字数 v\_usenum、吨数 v\_usenum2、金额 v\_money。  -- todo 2 对水费单价=2.24、字数=8012、进行赋值  -- todo 3 吨数根据水费字数换算，规则为水费字数除以1000，并且四舍五入，保留两位小数。  -- todo 4 计算金额，金额=单价\*吨数。  -- todo 5 输出单价 、数量和金额。 |

实现

|  |
| --- |
| declare  -- todo 1 声明变量水费单价 v\_price、水费字数 v\_usenum、吨数 v\_usenum2、金额 v\_money。  v\_price number(10, 2);  v\_usenum number;  v\_usenum2 number(10, 2);  v\_money number(10, 2);  begin  -- todo 2 对水费单价=2.24、字数=8012、进行赋值  v\_price := 2.24;  v\_usenum := 8012;  -- todo 3 吨数根据水费字数换算，规则为水费字数除以1000，并且四舍五入，保留两位小数。  v\_usenum2 := round(v\_usenum / 1000, 2);  -- todo 4 计算金额，金额=单价\*吨数。  v\_money := v\_usenum2 \* v\_price;  -- todo 5 输出单价 、数量和金额。  DBMS\_OUTPUT.PUT\_LINE('单价: ' || v\_price || ', 数量: ' || v\_usenum2 || ', 金额: ' || v\_money);  end; |

### select into 方式 赋值

语法结构：

|  |
| --- |
| select 列名 into 变量名 from 表名 where 条件 |

注意：结果必须是一条记录 ，有多条记录和没有记录都会报错

|  |
| --- |
| -- todo 目标: select into 方式 赋值  -- todo 需求: 计算 业主编号为 1, 2012年12月 应付水费.  -- todo 1 声明变量: v\_price 单价, v\_usenum 水费字数, v\_num0 上月字数, v\_num1 本月字数, v\_usenum2 使用吨数, v\_money 水费金额  -- todo 2 单价赋值=3.45  -- todo 3 使用 select into 给 水费字数, 上月字数, 本月字数 赋值  -- todo 4 求吨数  -- todo 5 计算应付金额 = 吨数 \* 单价  -- todo 6 输出 单价 吨数 应付金额 上月字数 本月字数 |

实现

|  |
| --- |
| declare  -- todo 目标: select into 方式 赋值  -- todo 1 声明变量: v\_price 单价, v\_usenum 水费字数, v\_num0 上月字数, v\_num1 本月字数, v\_usenum2 使用吨数, v\_money 水费金额  v\_price number(10, 2);  v\_usenum number;  v\_num0 number;  v\_num1 number;  v\_usenum2 number(10, 2);  v\_money number(10, 2);  begin  -- todo 2 单价赋值=3.45  v\_price := 3.45;  -- todo 3 使用 select into 给 水费字数, 上月字数, 本月字数 赋值  select USENUM, NUM0, NUM1 into v\_usenum,v\_num0,v\_num1  from C##WATERUSER.T\_ACCOUNT  where YEAR='2012' and month='01' and OWNERUUID=1;  -- todo 4 求吨数  v\_usenum2 := round(v\_usenum / 1000, 2);  -- todo 5 计算应付金额 = 吨数 \* 单价  v\_money := v\_price \* v\_usenum2;  -- todo 6 输出 单价 吨数 应付金额 上月字数 本月字数  DBMS\_OUTPUT.PUT\_LINE('单价: ' || v\_price || ', 吨数: ' || v\_usenum2 || ', 应付金额: ' || v\_money);  end; |

## 引用变量

语法

|  |
| --- |
| 变量名 表名.列名%TYPE; |

作用：引用某表某列的字段类型, 跟指定列的类型保持一致.

|  |
| --- |
| **-- todo 目标: 引用变量 改造上面的代码**  -- todo 1 声明变量: v\_price 单价, v\_usenum 水费字数, v\_num0 上月字数, v\_num1 本月字数, v\_usenum2 使用吨数, v\_money 水费金额  -- todo 2 单价赋值=3.45  -- todo 3 使用 select into 给 水费字数, 上月字数, 本月字数 赋值  -- todo 4 求吨数  -- todo 5 计算应付金额 = 吨数 \* 单价  -- todo 6 输出 单价 吨数 应付金额 上月字数 本月字数 |

实现

|  |
| --- |
| declare  -- todo 目标: select into 方式 赋值  -- todo 1 声明变量: v\_price 单价, v\_usenum 水费字数, v\_num0 上月字数, v\_num1 本月字数, v\_usenum2 使用吨数, v\_money 水费金额  v\_price number(10, 2);  v\_usenum T\_ACCOUNT.USENUM%type;  v\_num0 T\_ACCOUNT.NUM0%type;  v\_num1 T\_ACCOUNT.NUM1%type;  v\_usenum2 number(10, 2);  v\_money number(10, 2);  begin  -- todo 2 单价赋值=3.45  v\_price := 3.45;  -- todo 3 使用 select into 给 水费字数, 上月字数, 本月字数 赋值  select USENUM, NUM0, NUM1 into v\_usenum,v\_num0,v\_num1  from C##WATERUSER.T\_ACCOUNT  where YEAR='2012' and month='01' and OWNERUUID=1;  -- todo 4 求吨数  v\_usenum2 := round(v\_usenum / 1000, 2);  -- todo 5 计算应付金额 = 吨数 \* 单价  v\_money := v\_price \* v\_usenum2;  -- todo 6 输出 单价 吨数 应付金额 上月字数 本月字数  DBMS\_OUTPUT.PUT\_LINE('单价: ' || v\_price || ', 吨数: ' || v\_usenum2 || ', 应付金额: ' || v\_money);  end; |

## 记录型变量(行变量)

定义语法

|  |
| --- |
| 行变量名 表名%rowtype; |

作用: 标识某个表的行记录类型

使用记录的列值的语法

|  |
| --- |
| 行变量名.列名 |

|  |
| --- |
| -- todo **目标: 记录型变量**  -- todo 1 声明变量: v\_price 单价, **v\_account 行变量**, v\_usenum2 使用吨数, v\_money 水费金额  -- todo 2 单价赋值=3.45  -- todo 3 使用 select into 给 **行变量** 赋值  -- todo 4 求吨数  -- todo 5 计算应付金额 = 吨数 \* 单价  -- todo 6 输出 单价 吨数 应付金额 上月字数 本月字数 |

实现

|  |
| --- |
| -- todo 目标: 记录型变量  declare  -- todo 1 声明变量: v\_price 单价, v\_account 行变量, v\_usenum2 使用吨数, v\_money 水费金额  v\_price number(10, 2);  v\_account **t\_account%rowtype;**  v\_usenum2 number(10, 2);  v\_money number(10, 2);  begin  -- todo 2 单价赋值=3.45  v\_price := 3.45;  -- todo 3 使用 select into 给 **行变量** 赋值  select \* **into v\_account**  from C##WATERUSER.T\_ACCOUNT  where YEAR='2012' and month='01' and OWNERUUID=1;  -- todo 4 求吨数  v\_usenum2 := round(**v\_account.usenum** / 1000, 2);  -- todo 5 计算应付金额 = 吨数 \* 单价  v\_money := v\_price \* v\_usenum2;  -- todo 6 输出 单价 吨数 应付金额 上月字数 本月字数  DBMS\_OUTPUT.PUT\_LINE('单价: ' || v\_price || ', 吨数: ' || v\_usenum2 || ', 应付金额: ' || v\_money);  end; |

## 异常

默认 开启事务

执行 sql1(增删改)

执行 sql2(增删改)

.... ...

如果没有问题, 提交生效(commit);

如果出现问题, 可以使用官方提示异常, 也可以自定义异常.

在运行程序时出现的错误叫做异常

发生异常后，语句将停止执行，控制权转移到 PL/SQL 块的异常处理部分

异常有两种类型：

预定义异常 - 当 PL/SQL 程序违反 Oracle 规则或超越系统限制时隐式引发

用户定义异常 - 用户可以在 PL/SQL 块的声明部分定义异常，自定义的异常通过 RAISE 语句显式引发

预定义异常

Oracle 预定义异常 21 个

|  |  |
| --- | --- |
| 命名的系统异常 | 产生原因 |
| ACCESS\_INTO\_NULL | 未定义对象 |
| CASE\_NOT\_FOUND | CASE 中若未包含相应的 WHEN ，并且没有设置 ELSE 时 |
| COLLECTION\_IS\_NULL | 集合元素未初始化 |
| CURSER\_ALREADY\_OPEN | 游标已经打开 |
| DUP\_VAL\_ON\_INDEX | 唯一索引对应的列上有重复的值 |
| INVALID\_CURSOR | 在不合法的游标上进行操作 |
| INVALID\_NUMBER | 内嵌的 SQL 语句不能将字符转换为数字 |
| NO\_DATA\_FOUND | 使用 select into 未返回行 |
| TOO\_MANY\_ROWS | 执行 select into 时，结果集超过一行 |
| ZERO\_DIVIDE | 除数为 0 |
| SUBSCRIPT\_BEYOND\_COUNT | 元素下标超过嵌套表或 VARRAY 的最大值 |
| SUBSCRIPT\_OUTSIDE\_LIMIT | 使用嵌套表或 VARRAY 时，将下标指定为负数 |
| VALUE\_ERROR | 赋值时，变量长度不足以容纳实际数据 |
| LOGIN\_DENIED | PL/SQL 应用程序连接到 oracle 数据库时，提供了不正确的用户名或密码 |
| NOT\_LOGGED\_ON | PL/SQL 应用程序在没有连接 oralce 数据库的情况下访问数据 |
| PROGRAM\_ERROR | PL/SQL 内部问题，可能需要重装数据字典＆ pl./SQL 系统包 |
| ROWTYPE\_MISMATCH | 宿主游标变量与 PL/SQL 游标变量的返回类型不兼容 |
| SELF\_IS\_NULL | 使用对象类型时，在 null 对象上调用对象方法 |
| STORAGE\_ERROR | 运行 PL/SQL 时，超出内存空间 |
| SYS\_INVALID\_ID | 无效的 ROWID 字符串 |
| TIMEOUT\_ON\_RESOURCE | Oracle 在等待资源时超时 |

语法结构：

|  |
| --- |
| exception  when 异常类型 then  异常处理逻辑 |

根据上例中的代码，添加异常处理部分

|  |
| --- |
| -- todo 目标: 异常处理  -- todo 1 声明变量: v\_price 单价, v\_account 行变量, v\_usenum2 使用吨数, v\_money 水费金额  -- todo 2 单价赋值=3.45  -- todo 3 使用 select into 给 水费字数, 上月字数, 本月字数 赋值  -- todo 4 求吨数  -- todo 5 计算应付金额 = 吨数 \* 单价  -- todo 6 输出 单价 吨数 应付金额 上月字数 本月字数  -- todo 7 处理异常  -- todo 7.1 处理 未找到数据异常 no\_data\_found  -- todo 7.2 处理 查询条件有误, 返回多条数据异常 too\_many\_rows |

实现

|  |
| --- |
| -- todo 目标: 异常处理  declare  -- todo 1 声明变量: v\_price 单价, v\_account 行变量, v\_usenum2 使用吨数, v\_money 水费金额  v\_price number(10, 2);  v\_account t\_account%rowtype;  v\_usenum2 number(10, 2);  v\_money number(10, 2);  begin  -- todo 2 单价赋值=3.45  v\_price := 3.45;  -- todo 3 使用 select into 给 水费字数, 上月字数, 本月字数 赋值  select \* into v\_account  from C##WATERUSER.T\_ACCOUNT  where YEAR='2012' and month='01' and OWNERUUID=1; -- 正常  -- where YEAR='2012' and OWNERUUID=1; -- 模拟异常1: 数据多了  -- where YEAR='2030' and month='01' and OWNERUUID=1; --模拟异常2: 数据没找到  -- todo 4 求吨数  v\_usenum2 := round(v\_account.usenum / 1000, 2);  -- todo 5 计算应付金额 = 吨数 \* 单价  v\_money := v\_price \* v\_usenum2;  -- todo 6 输出 单价 吨数 应付金额 上月字数 本月字数  DBMS\_OUTPUT.PUT\_LINE('单价: ' || v\_price || ', 吨数: ' || v\_usenum2 || ', 应付金额: ' || v\_money);  -- todo 7 处理异常  exception  -- todo 7.1 处理 未找到数据异常 no\_data\_found  when no\_data\_found then  DBMS\_OUTPUT.PUT\_LINE('没有找到对应数据, 请核实!');  -- todo 7.2 处理 查询条件有误, 返回多条数据异常 too\_many\_rows  when too\_many\_rows then  DBMS\_OUTPUT.PUT\_LINE('查询条件有误, 返回多条数据异常, 请核实!');  end; |

## 条件判断

基本语法 1

|  |
| --- |
| if 条件 then  业务逻辑  end if; |

基本语法 2

|  |
| --- |
| if 条件 then  业务逻辑  else  业务逻辑  end if; |

基本语法 3

|  |
| --- |
| if 条件1 then  业务逻辑  elsif 条件2 then  业务逻辑  else  业务逻辑  end if; |

业务：设置三个等级的水费 5 吨以下 2.45 元/吨、 5 吨到 10 吨部分 3.45 元/吨，

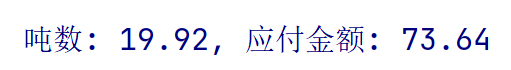
超过 10 吨部分 4.45，根据使用水费的量来计算阶梯水费。

|  |
| --- |
| -- todo 目标: 使用判断if计算阶梯水费  -- todo 业务: 设置三个等级的水费 5 吨以下 2.45 元/吨、 5 吨到 10 吨部分 3.45 元/吨，超过 10 吨部分 4.45，根据使用水费的量来计算阶梯水费。  -- todo 1 声明变量: v\_price1 <=5吨单价, v\_price2 (5,10]吨单价, v\_price3 >10吨单价, v\_account 行变量, v\_usenum2 使用吨数, v\_money 水费金额  -- todo 2 单价赋值: <5吨单价=2.45, [5,10)吨单价=3.45, >=10吨单价=4.45  -- todo 3 使用 select into 给 行变量 赋值  -- todo 4 求吨数  -- todo 5 计算阶梯水费  -- todo 6 输出 吨数 应付金额  -- todo 7 处理异常  -- todo 7.1 处理 未找到数据异常 no\_data\_found  -- todo 7.2 处理 查询条件有误, 返回多条数据异常 too\_many\_rows |

实现

|  |
| --- |
| declare  -- todo 目标: 使用判断if计算阶梯水费  -- todo 业务: 设置三个等级的水费 5 吨以下 2.45 元/吨、 5 吨到 10 吨部分 3.45 元/吨，超过 10 吨部分 4.45，根据使用水费的量来计算阶梯水费。  -- todo 1 声明变量: v\_price1 <=5吨单价, v\_price2 (5,10]吨单价, v\_price3 >10吨单价, v\_account 行变量, v\_usenum2 使用吨数, v\_money 水费金额  v\_price1 number(10, 2);  v\_price2 number(10, 2);  v\_price3 number(10, 2);  v\_account T\_ACCOUNT%rowtype;  v\_usenum2 number(10, 2);  v\_money number(10, 2);  begin  -- todo 2 单价赋值: <5吨单价=2.45, [5,10)吨单价=3.45, >=10吨单价=4.45  v\_price1 := 2.45;  v\_price2 := 3.45;  v\_price3 := 4.45;  -- todo 3 使用 select into 给 行变量 赋值  select \* into v\_account from T\_ACCOUNT  where YEAR='2012' and MONTH='01' and OWNERUUID=1;  -- todo 4 求吨数  v\_usenum2 := round(v\_account.USENUM / 1000, 2);  -- todo 5 计算阶梯水费  if v\_usenum2<=5 then  v\_money := v\_usenum2 \* v\_price1;  elsif v\_usenum2<=10 then  v\_money := 5 \* v\_price1 + (10 - v\_usenum2) \* v\_price2;  else  v\_money := 5 \* v\_price1 + (10 - 5) \* v\_price2 + (v\_usenum2 - 10) \* v\_price3;  end if;  -- todo 6 输出 吨数 应付金额  DBMS\_OUTPUT.PUT\_LINE('吨数: ' || v\_usenum2 || ', 应付金额: ' || v\_money);  -- todo 7 处理异常  -- todo 7.1 处理 未找到数据异常 no\_data\_found  -- todo 7.2 处理 查询条件有误, 返回多条数据异常 too\_many\_rows  exception  when no\_data\_found then  DBMS\_OUTPUT.PUT\_LINE('没有找到数据, 请核实!');  when too\_many\_rows then  DBMS\_OUTPUT.PUT\_LINE('返回多条数据, 请核实!');  end; |

结果:



## 循环

### 无条件循环

语法结构

|  |
| --- |
| loop  --循环语句  end loop; |

|  |
| --- |
| -- todo 目标: 使用 loop 输出 1 ~ 100  -- todo 1 定义变量 v\_num  -- todo 2 初始化变量 v\_num等于1  -- todo 3 编写 loop 循环  -- todo 3.1 进入循环, 输出 v\_num变量的值  -- todo 3.2 变量 v\_num 加 1  -- todo 3.3 当 v\_num>100时退出 |

实现

|  |
| --- |
| declare  -- todo 目标: 使用 loop 输出 1 ~ 100  -- todo 1 定义变量 v\_num  v\_num number;  begin  -- todo 2 初始化变量 v\_num等于1  v\_num := 1;  -- todo 3 编写 loop 循环  loop  -- todo 3.1 进入循环, 输出 v\_num变量的值  DBMS\_OUTPUT.PUT\_LINE(v\_num);  -- todo 3.2 变量 v\_num 加 1  v\_num := v\_num + 1;  -- todo 3.3 当 v\_num>100时退出  exit when v\_num>100;  end loop;  end; |

### 条件循环

语法结构

|  |
| --- |
| while 条件  loop    end loop; |

|  |
| --- |
| -- todo 目标: 使用 while 输出 1 ~ 100  -- todo 1 初始化变量 v\_num等于1  -- todo 2 编写 while 循环, 指定继续执行条件 v\_num<=100  -- todo 2.1 进入循环, 输出 v\_num变量的值  -- todo 2.2 变量 v\_num 加 1 |

实现

|  |
| --- |
| declare  -- todo 目标: 使用 while 输出 1 ~ 100  -- todo 1 初始化变量 v\_num等于1  v\_num number := 1;  begin  -- todo 2 编写 while 循环, 指定继续执行条件 v\_num<=100  while v\_num<=100 loop  -- todo 2.1 进入循环, 输出 v\_num变量的值  DBMS\_OUTPUT.PUT\_LINE(v\_num);  -- todo 2.2 变量 v\_num 加 1  v\_num := v\_num + 1;  end loop;  end; |

### for循环

基本语法

|  |
| --- |
| for 变量 in 起始值..终止值  loop  end loop; |

目标: 使用 for 输出 1 ~ 100

|  |
| --- |
| begin  for v\_num in 1 .. 100 loop  dbms\_output.put\_line(v\_num);  end loop;  end; |

## 游标

### 什么是游标

游标是系统为用户开设的一个数据缓冲区,存放 SQL 语句的执行结果。

我们可以把游标理解为 PL/SQL 中的结果集。



### 不带参数的游标

语法结构及示例

在声明区声明游标，语法如下：

|  |
| --- |
| cursor 游标名称 is SQL语句; |

使用游标语法

|  |
| --- |
| open 游标名称  loop  fetch 游标名称 into 变量  exit when 游标名称%notfound  end loop;  close 游标名称 |

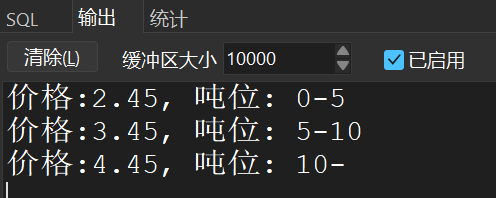
案例

|  |
| --- |
| -- todo 目标: 不带参数的游标  -- todo 需求：打印业主类型为 1 的价格表  -- todo 1 声明行变量 v\_pricetable  -- todo 2 声明游标 cur\_pricetable 保存业主类型为 1 的价格表  -- todo 3 打开游标  -- todo 5 loop 循环  -- todo 6 fetch 提取游标到变量  -- todo 7 当游标到最后一行下面退出循环  -- todo 8 输出 价格: XX, 吨位: YY-ZZ  -- todo 4 关闭游标 |

实现

|  |
| --- |
| declare  -- todo 目标: 不带参数的游标  -- todo 需求：打印业主类型为 1 的价格表  -- todo 1 声明行变量 v\_pricetable  v\_pricetable T\_PRICETABLE%rowtype;  -- todo 2 声明游标 cur\_pricetable 保存业主类型为 1 的价格表  cursor cur\_pricetable is  select \* from T\_PRICETABLE where OWNERTYPEID=1;  begin  -- todo 3 打开游标  open cur\_pricetable;  -- todo 5 loop 循环  loop  -- todo 6 fetch 提取游标到变量  fetch cur\_pricetable into v\_pricetable;  -- todo 7 当游标到最后一行下面退出循环  exit when cur\_pricetable%notfound;  -- todo 8 输出 价格: XX, 吨位: YY-ZZ  DBMS\_OUTPUT.PUT\_LINE('价格: ' || v\_pricetable.PRICE || ', 吨位: ' || v\_pricetable.MINNUM ||'-' || v\_pricetable.MAXNUM);  end loop;  -- todo 4 关闭游标  close cur\_pricetable;  end; |

运行结果如下：



### 带参数的游标

语法1: 创建带参数的游标

|  |
| --- |
| cursor cur\_pricetable(参数名 参数类型) is SQL; |

语法2: 使用带参数的游标

|  |
| --- |
| open cur\_pricetable(2); |

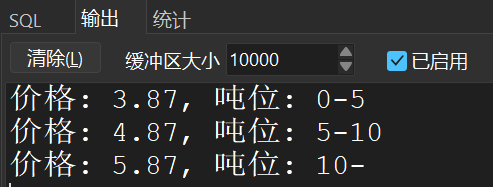
我们的查询语句的条件值有可能是在运行时才能决定的，比如性业主类型， 可能是运行时才可以决定，那如何实现呢？我们接下来学习带参数的游标，修改上述案例

案例:

|  |
| --- |
| -- todo 目标: 带参数的游标  -- todo 需求：打印根据参数值 业主类型 显示 价格表  -- todo 1 声明行变量 v\_pricetable  -- todo 2 声明带参数游标 cur\_pricetable 根据参数值获取指定类型的价格表  -- todo 3 打开游标  -- todo 5 loop 循环  -- todo 6 fetch 提取游标到变量  -- todo 7 当游标到最后一行下面退出循环 notfound  -- todo 8 输出 价格: XX, 吨位: YY-ZZ  -- todo 4 关闭游标 |

实现

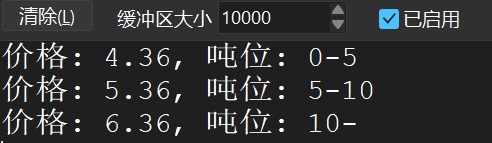
|  |
| --- |
| declare  -- todo 目标: 带参数的游标  -- todo 需求：打印根据参数值 业主类型 显示 价格表  -- todo 1 声明行变量 v\_pricetable  v\_pricetable T\_PRICETABLE%rowtype;  -- todo 2 声明带参数游标 cur\_pricetable 根据参数值获取指定类型的价格表  cursor cur\_pricetable(v\_ownertypeid number) is  select \* from T\_PRICETABLE where ownertypeid=v\_ownertypeid;  begin  -- todo 3 打开游标  open cur\_pricetable(2);  -- todo 5 loop 循环  loop  -- todo 6 fetch 提取游标到变量  fetch cur\_pricetable into v\_pricetable;  -- todo 7 当游标到最后一行下面退出循环 notfound  exit when cur\_pricetable%notfound;  -- todo 8 输出 价格: XX, 吨位: YY-ZZ  DBMS\_OUTPUT.PUT\_LINE('价格: ' || v\_pricetable.PRICE || ', 吨位: ' || v\_pricetable.MINNUM || '-' || v\_pricetable.MAXNUM);  end loop;  -- todo 4 关闭游标  close cur\_pricetable;  end; |



### for 循环提取游标值

我们每次提取游标，需要打开游标 关闭游标 循环游标 提取游标 控制循环的 退出等等，好麻烦！有没有更简单的写法呢？有！用 for 循环一切都那么简单 ，

上例的代码可以改造为下列形式



案例

|  |
| --- |
| -- todo 目标: 带参数的游标  -- todo 需求：使用for 打印根据参数值显示指定业主类型 的 价格表  -- todo 1 声明带参数游标 cur\_pricetable 根据参数值获取指定类型的价格表  -- todo 2 使用 for 循环遍历  -- todo 2.1 打印 价格: XX, 吨位: YY-ZZ |

实现

|  |
| --- |
| declare  -- todo 目标: 带参数的游标  -- todo 需求：使用for 打印根据参数值显示指定业主类型 的 价格表  -- todo 1 声明带参数游标 cur\_pricetable 根据参数值获取指定类型的价格表  cursor cur\_pricetable(v\_ownertypeid number) is  select \* from t\_pricetable where ownertypeid=v\_ownertypeid;  begin  -- todo 2 使用 for 循环遍历  for v\_pricetable in cur\_pricetable(1) loop  -- todo 2.1 打印 价格: XX, 吨位: YY-ZZ  DBMS\_OUTPUT.PUT\_LINE('价格: ' || v\_pricetable.PRICE ||', 吨位: ' || v\_pricetable.MINNUM || '-' || v\_pricetable.MAXNUM);  end loop;  end; |

结果:



# 存储函数

## 什么是存储函数

存储函数又称为自定义函数。可以接收一个或多个参数，返回一个结果。

在函数中我们可以使用 PL/SQL 进行逻辑的处理。

select max(xx), 函数(xxx), substr(列1, 2, 3) from 表

## 存储函数语法结构

创建或修改存储过程的语法如下:

|  |
| --- |
| CREATE [ OR REPLACE ] FUNCTION 函数名称  （参数名称 参数类型, 参数名称 参数类型, ... ）  RETURN 结果变量数据类型  IS  变量声明部分;  BEGIN  逻辑部分;  RETURN 结果变量;  [EXCEPTION  异常处理部分]  END; |

## 案例

需求：创建存储函数，根据地址 ID 查询地址名称。

语句:

|  |
| --- |
| create function fn\_getaddress(v\_id number)  return varchar2  is  v\_name varchar2(30);  begin  select name into v\_name from t\_address where id=v\_id;  return v\_name;  end; |

测试此函数：

|  |
| --- |
| select name from t\_address where id = 3; |

输出内容



需求：查询业主 ID ，业主名称，业主地址，业主地址使用刚才我们创建的函数

来实现。

|  |
| --- |
| select id 编号,name 业主名称,fn\_getaddress(addressid) 地址  from t\_owners |

查询结果如下：



案例

|  |
| --- |
| -- todo 目标: 创建存储函数 和 使用存储函数  -- todo 需求1: 创建存储函数 fn\_getAddressByID，根据 地址ID 查询 地址名称。  -- todo 1.1 搭存储函数架子 指定 参数和返回类型  -- todo 1.2 声明变量 地址名称 v\_name  -- todo 1.3 通过 select into 赋值  -- todo 1.4 返回 地址名称 v\_name  -- todo 需求2：查询业主 ID ，业主名称，业主地址(业主地址使用刚才我们创建的函数)  -- todo 验证 |

实现

|  |
| --- |
| -- todo 目标: 创建存储函数 和 使用存储函数  -- todo 需求1: 创建存储函数 fn\_getAddressByID，根据 地址ID 查询 地址名称。  -- todo 1.1 搭存储函数架子 指定 参数和返回类型  create or replace function fn\_getAddressByID(v\_id number)  return varchar2  is  -- todo 1.2 声明变量 地址名称 v\_name  v\_name varchar2(32);  begin  -- todo 1.3 通过 select into 赋值  select name into v\_name from T\_ADDRESS where ID=v\_id;  -- todo 1.4 返回 地址名称 v\_name  return v\_name;  end;  -- todo 需求2：查询业主 ID ，业主名称，业主地址(业主地址使用刚才我们创建的函数)  select id, name, **fn\_getAddressByID(ADDRESSID)** as addr\_name from T\_OWNERS;  -- todo 验证  select  t1.id, t1.name, t2.name  from T\_OWNERS t1, T\_ADDRESS t2  where t1.ADDRESSID=t2.ID(+)  order by t1.id  ; |

# 存储过程

## 什么是存储过程

存储过程是被命名的 PL/SQL 块，存储于数据库中，是数据库对象的一种。

应用程序可以调用存储过程，执行相应的逻辑。

存储过程与存储函数都可以封装一定的业务逻辑并返回结果，存在区别如下：

1、存储函数中有返回值，且必须返回；而存储过程没有返回值，可以通过传出参数返回多个值。

2、存储函数可以在[select 语句中](https://www.baidu.com/s?wd=select%E8%AF%AD%E5%8F%A5&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLPAPbnvRLP161rHTvP1ub0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPj6kPH6vPWnk)直接使用，而存储过程不能。过程多数是被应用程序所调用。

3、存储函数一般都是封装一个查询结果，而存储过程一般都封装一段事务代码。

## 存储过程语法结构

创建或修改存储过程的语法如下:

|  |
| --- |
| CREATE [ OR REPLACE ] PROCEDURE 存储过程名称  （参数名 in 类型, 参数名 类型, 参数名 out 类型）  IS|AS  变量声明部分;  BEGIN  逻辑部分  [EXCEPTION  异常处理部分]  END; |

参数只指定类型，不指定长度

过程参数的三种模式：

IN 传入参数（默认）

OUT 传出参数，主要用于返回程序运行结果

IN OUT 传入传出参数

### 创建不带传出参数的存储过程：添加业主信息

|  |
| --- |
| --增加业主信息序列  create sequence seq\_owners start with 11;  --增加业主信息存储过程  create or replace procedure pro\_owners\_add  (  v\_name varchar2,  v\_addressid number,  v\_housenumber varchar2,  v\_watermeter varchar2,  v\_type number  )  is  begin  insert into t\_owners  values( seq\_owners.nextval, v\_name, v\_addressid, v\_housenumber, v\_watermeter, sysdate, v\_type );  commit;  end; |

PL/SQL 中调用存储过程

|  |
| --- |
| call pro\_owners\_add('赵伟',1,'999-3','132-7',1); |

案例

|  |
| --- |
| -- todo 目标: 创建存储过程 和 使用存储过程  -- todo 准备工作 创建业主信息序列  -- todo 需求1: 创建不带传出参数的存储过程 pro\_owners\_add：添加业主信息  -- todo 1 声明参数: v\_name, v\_addressid, v\_housenumber, v\_watermeter, v\_type  -- todo 2 插入 insert into  -- todo 3 提交事务  -- todo 需求2: 调用存储过程 添加业主信息 call 存储过程(... ...); |

实现

|  |
| --- |
| -- todo 需求1: 创建不带传出参数的存储过程 pro\_owners\_add：添加业主信息  -- todo 1 声明参数: v\_name, v\_addressid, v\_housenumber, v\_watermeter, v\_type  create or replace procedure pro\_owners\_add(  v\_name varchar2,  v\_addressid number,  v\_housenumber varchar2,  v\_watermeter varchar2,  v\_type number  )  is  begin  -- todo 2 插入 insert into  insert into T\_OWNERS (ID, NAME, ADDRESSID, HOUSENUMBER, WATERMETER, ADDDATE, OWNERTYPEID)  values (seq\_owners.nextval, v\_name, v\_addressid, v\_housenumber, v\_watermeter, sysdate, v\_type);  -- todo 3 提交事务  commit;  end;  -- todo 需求2: 调用存储过程 添加业主信息 call 存储过程(... ...);  call pro\_owners\_add('蜘蛛精', 1, '盘丝洞5号', '199', 2); |

### 创建带传出参数的存储过程

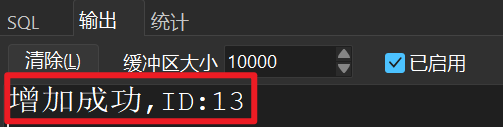
需求：添加业主信息，传出参数为新增业主的 ID

|  |
| --- |
| --增加业主信息存储过程  create or replace procedure pro\_owners\_add  (  v\_name varchar2,  v\_addressid number,  v\_housenumber varchar2,  v\_watermeter varchar2,  v\_type number,  v\_id **out** number  )  is  begin  select seq\_owners.nextval into v\_id from dual;    insert into t\_owners  values( v\_id, v\_name, v\_addressid, v\_housenumber, v\_watermeter, sysdate, v\_type );    commit;  end; |

PL/SQL 调用该存储过程

|  |
| --- |
| declare  v\_id number;--定义传出参数的变量  begin  pro\_owners\_add('王旺旺',1,'922-3','133-7',1,v\_id);  DBMS\_OUTPUT.put\_line('增加成功,ID:'||v\_id);  end; |

执行成功后输出结果：



案例

|  |
| --- |
| -- todo 需求3: 创建带传出参数的存储过程：添加业主信息  -- todo 1 调用序列下一个值 赋值给 v\_id  -- todo 2 插入 insert into  -- todo 3 提交事务  -- todo 需求4: 调用存储过程 添加业主信息 且 打印返回的结果  -- todo 1 声明变量 v\_id  -- todo 2 调用存储过程 注意: 不需要 call  -- todo 3 打印返回 v\_id |

实现

|  |
| --- |
| -- todo 需求3: 创建带传出参数的存储过程：添加业主信息  create or replace procedure pro\_owners\_add\_2(  v\_name varchar2,  v\_addressid number,  v\_housenumber varchar2,  v\_watermeter varchar2,  v\_type number,  v\_id out number  )  is  begin  -- todo 1 调用序列下一个值 赋值给 v\_id  select seq\_owners.nextval into v\_id from dual;  -- todo 2 插入 insert into  insert into T\_OWNERS (ID, NAME, ADDRESSID, HOUSENUMBER, WATERMETER, ADDDATE, OWNERTYPEID)  values (v\_id, v\_name, v\_addressid, v\_housenumber, v\_watermeter, sysdate, v\_type);  -- todo 3 提交事务  commit;  end;  declare  -- todo 需求4: 调用存储过程 添加业主信息 且 打印返回的结果  -- todo 1 声明变量 v\_id  v\_id number;  begin  -- todo 2 调用存储过程 注意: 不需要 call  pro\_owners\_add\_2('牛魔王', 1, '火焰山', '1008', 10, v\_id);  -- todo 3 打印返回 v\_id  DBMS\_OUTPUT.PUT\_LINE('添加成功! v\_id: ' || v\_id);  end; |

# 触发器

## 什么是触发器

数据库触发器是一个与表相关联的、存储的 PL/SQL 程序。每当一个特定的 数据操作语句(Insert,update,delete)在指定的表上发出时，Oracle 自动地执行触发器中定义的语句序列。

触发器可用于

. 数据确认

. 实施复杂的安全性检查

. 做审计，跟踪表上所做的数据操作等

. 数据的备份和同步

触发器分类

. 前置触发器（BEFORE）

. 后置触发器（AFTER）

## 创建触发器的语法

语法：

|  |
| --- |
| CREATE [or REPLACE] TRIGGER 触发器名  BEFORE | AFTER  [DELETE ][[or] INSERT] [[or]UPDATE [OF 列名]] ON 表名  [FOR EACH ROW ][WHEN(条件) ]  declare  ……  begin  PLSQL 块  End ； |

FOR EACH ROW 作用是标注此触发器是行级触发器 语句级触发器

在触发器中触发语句与伪记录变量的值

|  |  |  |
| --- | --- | --- |
| 触发语句 | :old | :new |
| Insert | 所有字段都是空(null) | 将要插入的数据 |
| Update | 更新以前该行的值 | 更新后的值 |
| delete | 删除以前该行的值 | 所有字段都是空(null) |

## 案例

### 后置触发器

需求：当用户修改了业主信息表的数据时记录修改前与修改后的值

|  |
| --- |
| --创建业主名称修改日志表:用于记录业主更改前后的名称  create table t\_owners\_log  (  updatetime date,  ownerid number,  oldname varchar2(30),  newname varchar2(30)  );  --创建后置触发器，自动记录业主更改前后日志  create or replace trigger tri\_owners\_log  after  update of name  on t\_owners  for each row  declare  begin  insert into t\_owners\_log  values(sysdate, :old.id, :old.name, :new.name);  end; |

测试：

|  |
| --- |
| --更新数据  update t\_owners set name='杨小花' where id=3;  commit;  --查询日志表  select \* from t\_owners\_log; |

案例

|  |
| --- |
| -- todo 目标: 创建触发器  -- todo 1 创建业主名称修改日志表:用于记录业主更改前后的名称  -- todo t\_owners\_log(updatetime, ownerid, oldname, newname)  -- todo 2 创建后置触发器，自动记录业主更改前后日志 tri\_owners\_log  -- todo 2.1 修改 t\_owners表的name列之后操作  -- todo 2.2 数据插入到  -- todo 3 验证  -- todo 3.1 更新数据  -- todo 3.2 查询日志表 |

实现

|  |
| --- |
| -- todo 目标: 创建触发器  -- todo 1 创建业主名称修改日志表:用于记录业主更改前后的名称  -- todo t\_owners\_log(updatetime, ownerid, oldname, newname)  create table t\_owners\_log(  updatetime date,  ownerid number,  oldname varchar2(32),  newname varchar2(32)  );  -- todo 2 创建后置触发器，自动记录业主更改前后日志 tri\_owners\_log  -- todo 2.1 修改 t\_owners表的name列之后操作  -- todo 2.2 数据插入到  create or replace trigger tri\_owners\_log  after  update of name on T\_OWNERS  for each row  declare  begin  insert into t\_owners\_log  values(sysdate, :old.id, :old.name, :new.name);  end;  -- todo 3 验证  -- todo 3.1 更新数据  update T\_OWNERS set name='小牛头' where id=120;  commit;  -- todo 3.2 查询日志表  select \* from T\_OWNERS\_LOG; |

注意: 触发器中不能使用 提交 commit

# 定时器 job

## why

定时器通过自动执行任务，例如在夜间备份数据库，每周生成销售报表，或定期清理无效数据，从而提高数据库管理和应用程序开发的效率，并确保数据的完整性和可靠性。

## 通用语法

### 提交job语法

|  |
| --- |
| DBMS\_JOB.SUBMIT (  job OUT BINARY\_INTEGER,  what IN VARCHAR2,  next\_date IN DATE DEFAULT SYSDATE,  interval IN VARCHAR2 DEFAULT NULL  ); |

参数含义如下:

* jobid：输出参数，JOB 的 ID，当提交任务后，系统会返回一个 JOB 的 ID，会写到此参数变量中。用来唯一地标示 一个任务。该参数既可由用户指定也可由**系统自动赋予**。
  + 作业号 通常由系统自建的 jobseq 序列来取值
  + select jobseq.nextval from dual; ---DBA
* what：输入参数，是一个将被执行的 PL/SQL 代码块，或者是存储过程名，如果是存储过程名，则**要在后面添加“;”号** 
  + SQL 语句要用单引号括起来
  + 语句中的单引号要用**两个单引号**书写 单引号结合
    - 例如 ename=‘王翦’ 写在作业里
    - ename=''王翦''
* next\_date：输入参数，指定何时将运行这个工作。一个 时间类型变量而不是一个字符串，所以需要注意不要把它 当成字符串，不需要对该参数加引号。用来调度任务队列 中该任务下一次运行的时间。
  + 开始时间
  + 指定日期时间 启动作业
* interval：输入参数， interval 是一个字符串类型，注意添加引号。**next\_date 是建立这个 job时 希望第一次运行的时间**，interval 是 next\_date 之后**下一次** 运行的时间间隔
  + 周期性间隔时间

### job设定interval 的例子

|  |  |
| --- | --- |
| **运行频率** | **时间表达式** |
| 每天运行一次 | SYSDATE + 1 |
| 每小时运行一次 | SYSDATE + 1/24 |
| 每10分钟运行一次 | SYSDATE + 10/(60\*24) |
| 每30秒运行一次 | SYSDATE + 30/(60\*24\*60) |
| 每隔一星期运行一次 | SYSDATE + 7 |
| 每个月最后一天运行一次 | TRUNC(LAST\_DAY(ADD\_MONTHS(SYSDATE,1))) + 23/24 |
| 每年1月1号零时 | TRUNC(LAST\_DAY(TO\_DATE(EXTRACT(YEAR FROM SYSDATE) |
| 每天午夜12点 | TRUNC(SYSDATE + 1) |
| 每天早上8点30分 | TRUNC(SYSDATE + 1) + (8\*60+30)/(24\*60) |
| 每星期二中午12点 | NEXT\_DAY(TRUNC(SYSDATE ), 'TUESDAY' ) + 12/24 |
| 每个月第一天的午夜12点 | TRUNC(LAST\_DAY(SYSDATE ) + 1) |
| 每个月最后一天的23点 | TRUNC (LAST\_DAY (SYSDATE)) + 23 / 24 |
| 每个季度最后一天的晚上11点 | TRUNC(ADD\_MONTHS(SYSDATE + 2/24, 3 ), 'Q' ) -1/24 |
| 每星期六和日早上6点10分 | TRUNC(LEAST(NEXT\_DAY(SYSDATE, "SATURDAY"), NEXT\_DAY(SYSDATE, "SUNDAY"))) + (6\*60+10)/(24\*60) |

### 查看正在运行的 job

|  |
| --- |
| select  job,log\_user,last\_date,next\_date,interval,what  from user\_jobs; |

### 修改作业

#### 修改作业的sql语句

* 不想修改的信息直接写 NULL
* 填写的信息会将原值覆盖

|  |
| --- |
| begin  dbms\_job.change(21,'update tb\_emp set salary=salary-10 where name=''李斯'';', NULL, NULL);  commit;  end; |

#### 修改下次运行时间

|  |
| --- |
| begin  dbms\_job.next\_date(21, sysdate+5/(24\*60));  commit;  end; |

#### 修改时间间隔 interval

* 将间隔时间改成 NULL 既是下次运行后就不在运行

|  |
| --- |
| begin  dbms\_job.INTERVAL(21,null);  commit;  end; |

### 删除 job

|  |
| --- |
| declare  job\_id number := 18;  begin  dbms\_job.remove(job\_id);  commit;  end; |

## 案例1: 通过 job 定时执行存储过程

### 准备工作: 建表 和 创建存储过程

|  |
| --- |
| -- 1.1 建表  create table tb\_date(dt date);  -- 1.2 创建存储过程  create or replace procedure pro\_date  as  begin  insert into tb\_date values(sysdate);  commit;  end;  -- 测试  call pro\_date();  select \* from tb\_date;  truncate table tb\_date; |

### 提交 job

|  |
| --- |
| -- 2 提交 job  DECLARE  job\_id NUMBER;  BEGIN  -- 提交 job 并执行存储过程 pro\_date  dbms\_job.submit(job\_id,  'pro\_date;',  SYSDATE,  'SYSDATE+(10)/(24\*60\*60)');  -- 运行 job  dbms\_job.run(job\_id);  dbms\_output.put\_line(job\_id);  end; |

### 查看 job

|  |
| --- |
| -- 3 查看 job  select  job,log\_user,last\_date,next\_date,interval,what  from user\_jobs; |

### 检查

|  |
| --- |
| -- 4 检查  select \* from tb\_date order by dt desc; |

### 停止 job

|  |
| --- |
| -- 5 修改job  ---停止定时器  declare  job\_id number := 18;  begin  dbms\_job.remove(job\_id);  commit;  end; |

### 检查

|  |
| --- |
| -- 6 检查  select  job,log\_user,last\_date,next\_date,interval,what  from user\_jobs; |

### 完整代码

|  |
| --- |
| -- 案例1: 通过 job 定时执行存储过程  -- 1 准备工作: 建表 和 创建存储过程  -- 1.1 建表  create table tb\_date(dt date);  -- 1.2 创建存储过程  create or replace procedure pro\_date  as  begin  insert into tb\_date values(sysdate);  commit;  end;  -- 测试  call pro\_date();  select \* from tb\_date;  -- 2 提交 job  DECLARE  job\_id NUMBER;  BEGIN  -- 提交 job 并执行存储过程 pro\_date  dbms\_job.submit(job\_id,  'pro\_date;',  SYSDATE,  'SYSDATE+(10)/(24\*60\*60)');  -- 运行 job  dbms\_job.run(job\_id);  dbms\_output.put\_line(job\_id);  end;  -- 3 查看 job  select  job,log\_user,last\_date,next\_date,interval,what from  user\_jobs;  -- 4 检查  select \* from tb\_date;  -- 5 停止定时器  declare  job\_id number := 6;  begin  dbms\_job.remove(job\_id);  commit;  end; |

## 目标2: 通过 job 定时执行 SQL语句

### 准备工作

|  |
| --- |
| -- 1 准备工作: 建表 和 插入数据  -- 创建雇员表 tb\_emp  CREATE TABLE tb\_emp (  id NUMBER PRIMARY KEY,  name VARCHAR2(100),  salary NUMBER  );  -- 插入数据  INSERT INTO tb\_emp (id, name, salary) VALUES (1, '秦始皇', 100000);  INSERT INTO tb\_emp (id, name, salary) VALUES (2, '李斯', 80000);  INSERT INTO tb\_emp (id, name, salary) VALUES (3, '王翦', 70000);  -- 提交事务  COMMIT;  select \* from tb\_emp;  update tb\_emp set salary=salary+1000 where name='王翦'; |

### 2 提交 job

|  |
| --- |
| -- 2 提交 job  DECLARE  job\_id NUMBER;  BEGIN  -- 提交 job 并执行存储过程 pro\_date  dbms\_job.submit(job\_id,  'update tb\_emp set salary=salary+1000 where name=''王翦'';',  SYSDATE,  'SYSDATE+(10)/(24\*60\*60)');  -- 运行 job  dbms\_job.run(job\_id);  dbms\_output.put\_line(job\_id);  end; |

* 语句中的单引号要用**两个单引号**书写 单引号结合
  + 例如 ename=‘王翦’ 写在作业里
  + ename=''王翦''

### 查看 job

|  |
| --- |
| -- 3 查看 job  select  job,log\_user,last\_date,next\_date,interval,what  from user\_jobs; |

### 检查

|  |
| --- |
| -- 4 检查  select \* from tb\_emp; |

### **修改作业的sql语句**

|  |
| --- |
| -- 不想修改的信息直接写 NULL  -- 填写的信息会将原值覆盖  begin  dbms\_job.change(21,'update tb\_emp set salary=salary-10 where name=''李斯'';', NULL, NULL);  commit;  end; |

### **修改下次运行时间**

|  |
| --- |
| begin  dbms\_job.next\_date(21, sysdate+5/(24\*60));  commit;  end; |

### 将间隔时间改成 NULL 既是下次运行后就不在运行

|  |
| --- |
| begin  dbms\_job.INTERVAL(21,null);  commit;  end; |

### 删除定时器

|  |
| --- |
| -- 5 停止定时器  declare  job\_id number := 20;  begin  dbms\_job.remove(job\_id);  commit;  end; |

### 完整代码

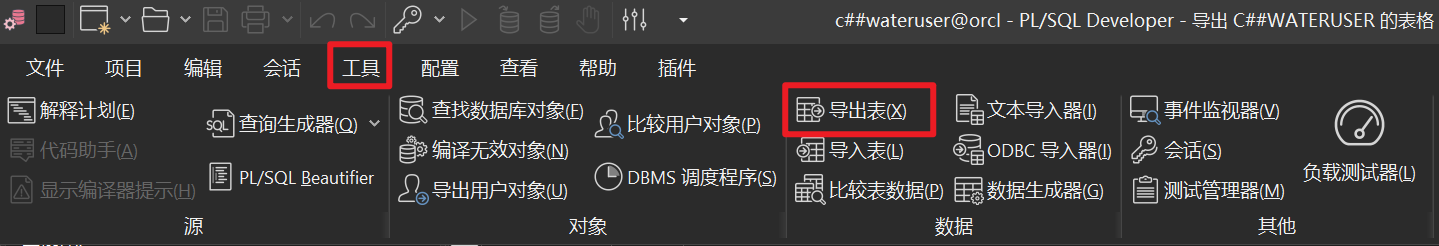
|  |
| --- |
| -- 案例2: 通过 job 定时执行 SQL语句  -- 1 准备工作: 建表 和 插入数据  -- 创建雇员表 tb\_emp  CREATE TABLE tb\_emp (  id NUMBER PRIMARY KEY,  name VARCHAR2(100),  salary NUMBER  );  -- 插入数据  INSERT INTO tb\_emp (id, name, salary) VALUES (1, '秦始皇', 100000);  INSERT INTO tb\_emp (id, name, salary) VALUES (2, '李斯', 80000);  INSERT INTO tb\_emp (id, name, salary) VALUES (3, '王翦', 70000);  -- 提交事务  COMMIT;  select \* from tb\_emp;  update tb\_emp set salary=salary+1000 where name='王翦';  -- 2 提交 job  DECLARE  job\_id NUMBER;  BEGIN  -- 提交 job 并执行存储过程 pro\_date  dbms\_job.submit(job\_id,  'update tb\_emp set salary=salary+1000 where name=''王翦'';',  SYSDATE,  'SYSDATE+(10)/(24\*60\*60)');  -- 运行 job  dbms\_job.run(job\_id);  dbms\_output.put\_line(job\_id);  end;  -- 3 查看 job  select  job,log\_user,last\_date,next\_date,interval,what  from user\_jobs;  -- 4 检查  select \* from tb\_emp;  -- 5 修改作业的sql语句  -- 不想修改的信息直接写 NULL  -- 填写的信息会将原值覆盖  begin  dbms\_job.change(21,'update tb\_emp set salary=salary-10 where name=''李斯'';', NULL, NULL);  commit;  end;  -- 6 修改下次运行时间  begin  dbms\_job.next\_date(21, sysdate+5/(24\*60));  commit;  end;  -- 7 将间隔时间改成 NULL 既是下次运行后就不在运行  begin  dbms\_job.INTERVAL(21,null);  commit;  end;  -- 8 停止定时器  declare  job\_id number := 20;  begin  dbms\_job.remove(job\_id);  commit;  end; |

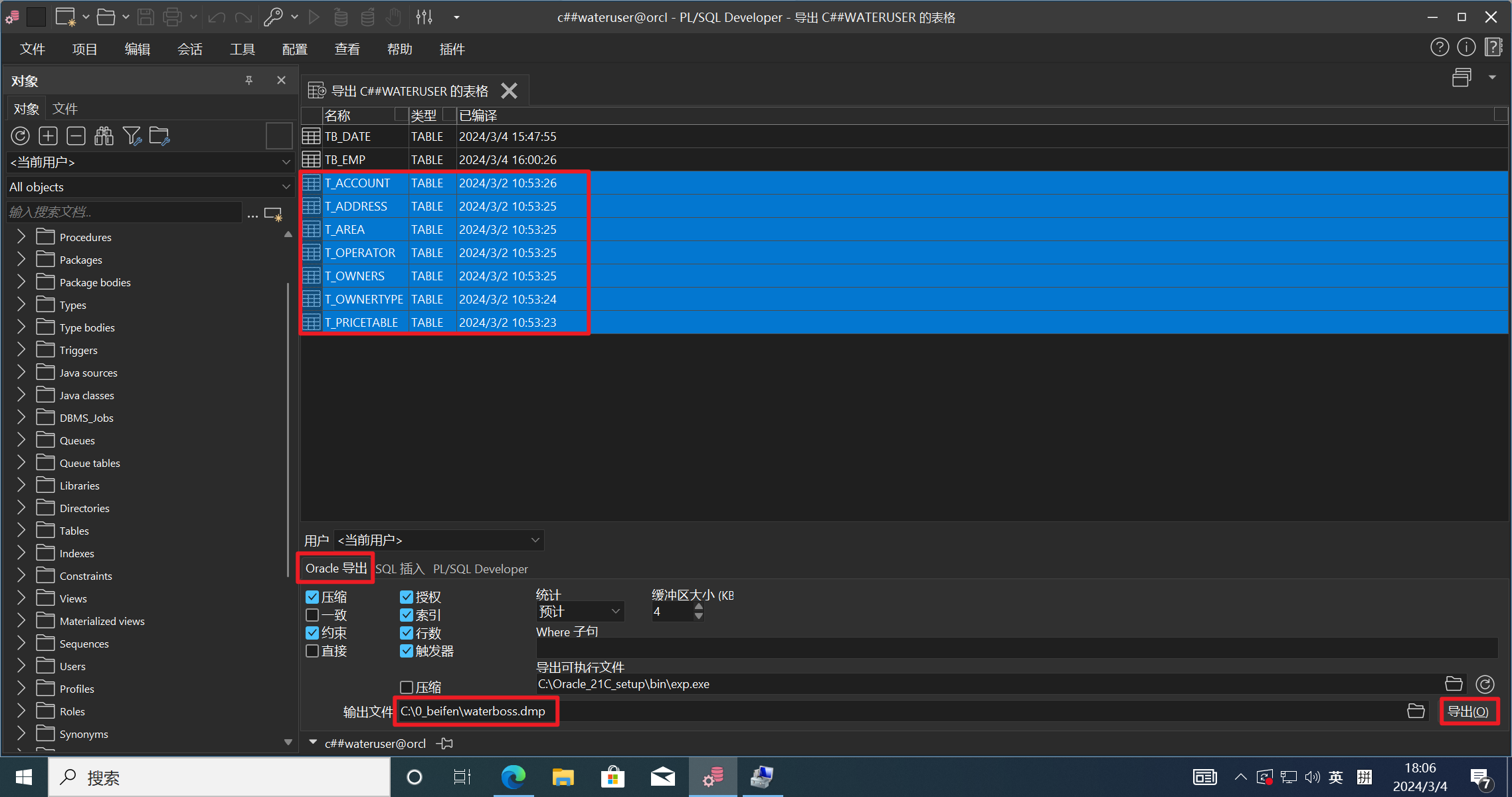
# 备份 导入和导出

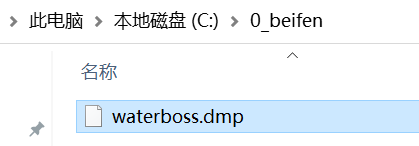
## PL/SQL工具

### dmp 格式

#### 导出





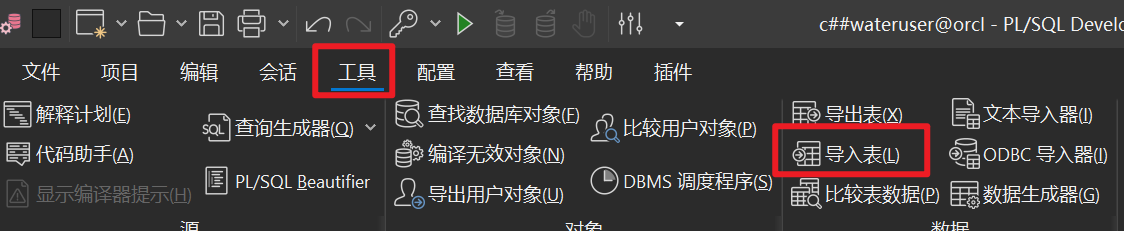


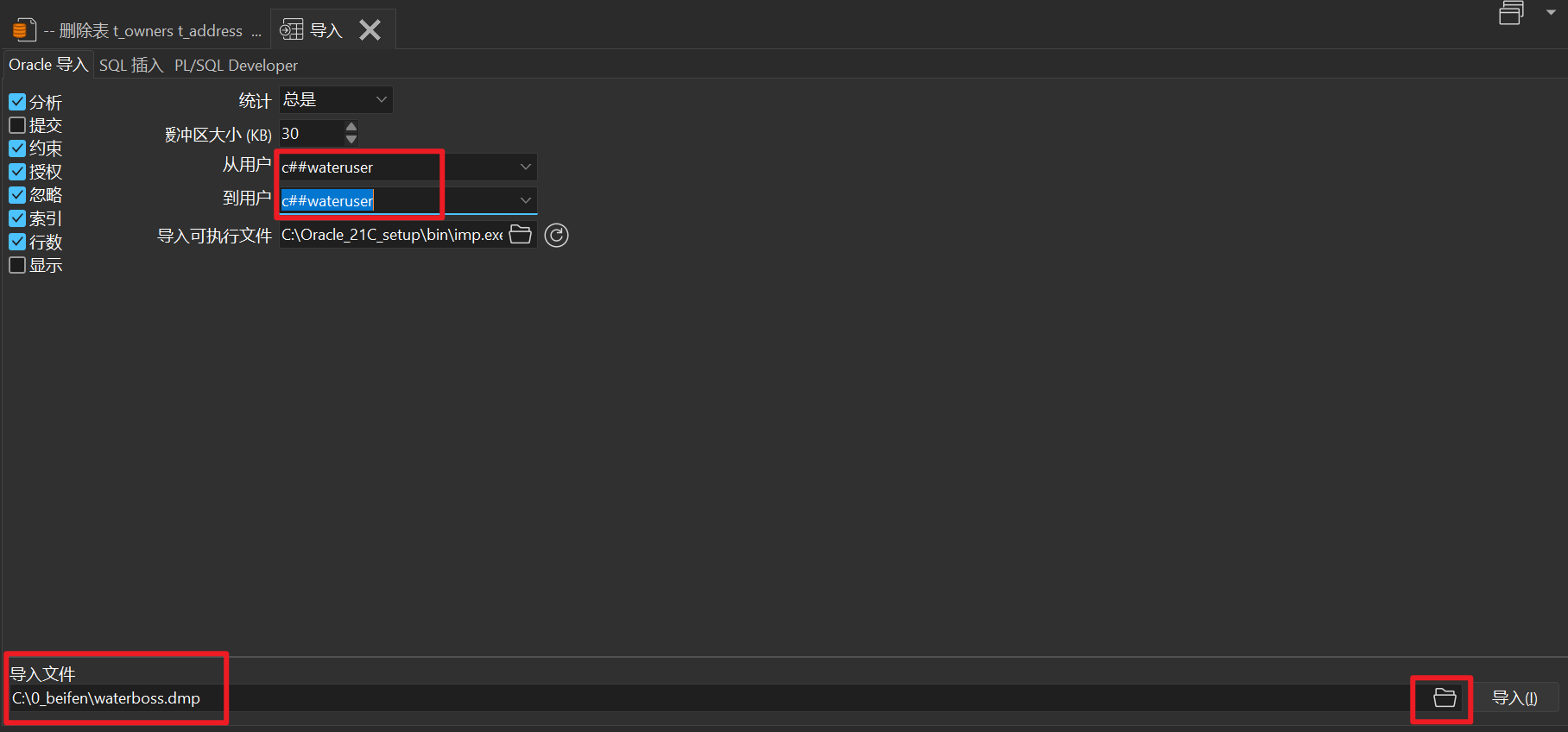
#### 导入

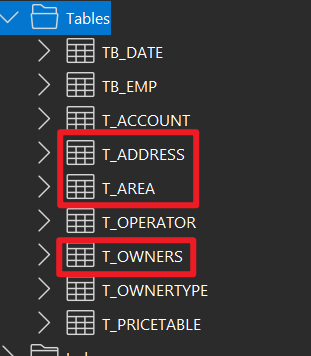
第一步: 删除表

|  |
| --- |
| -- 删除表 t\_owners t\_address t\_area  drop table t\_owners;  drop table t\_address;  drop table t\_area; |

第二步: 导入







## DOS方式

当我们使用一个数据库时，总希望数据库的内容是可靠的、正确的，但由于 计算机系统的故障（硬件故障、软件故障、网络故障、进程故障和系统故障）影 响数据库系统的操作，影响数据库中数据的正确性，甚至破坏数据库，使数据库 中全部或部分数据丢失。因此当发生上述故障后，希望能重构这个完整的数据库

该处理称为数据库恢复 ，而要进行数据库的恢复必须要有数据库的备份工作。

### 整库导出与导入

整库导出命令

|  |
| --- |
| exp system/Aa123456 full=y |

添加参数 full=y 就是整库导出



执行命令后会在当前目录下生成一个叫 EXPDAT.DMP ，此文件为备份文件。

如果想指定备份文件的名称，则添加 file 参数即可，命令如下

|  |
| --- |
| exp system/Aa123456 file=water.dmp full=y |

整库导入命令

|  |
| --- |
| imp system/Aa123456 full=y |

此命令如果不指定 file 参数，则默认用备份文件 EXPDAT.DMP 进行导入

如果指定 file 参数，则按照 file 指定的备份文件进行恢复

|  |
| --- |
| imp system/Aa123456 full=y file=water.dmp |

### 按用户导出与导入

按用户导出

|  |
| --- |
| exp system/Aa123456 owner=c##wateruser file=wateruser.dmp |

按用户导入

|  |
| --- |
| imp system/Aa123456 file=wateruser.dmp fromuser=wateruser |

### 按表导出与导入

按表导出

|  |
| --- |
| exp wateruser/itcast file=a.dmp tables=t\_account,a\_area |

用 tables 参数指定需要导出的表，如果有多个表用逗号分割即可

按表导入

|  |
| --- |
| imp wateruser/itcast file=a.dmp tables=t\_account,a\_area |