* 学习目标

* 能够理解PL/SQL的概述

  * CUD-->解决网络瓶颈

* 能够掌握PL/SQL的语法

  * declare begin end;

  * 变量：%type，%rowtype

  * 常量：constant

  * 运算符：:=,=,||, ..,=>

  * 条件语句

    if c then

    elsif then

    else

    end if;

  * 循环语句

    * loop

      ...

      exit when ...

      ..

     end loop;

    * while c loop

      end loop;

    * for m_count in[reverse] 1..5 loop

     end loop;

  * 异常

    EXCEPTION

when ZERO_DEVIDE then

when ohters then

sqlcode,sqlerrm

* 自定义异常

age _exfeption EXCEPTION;

raise

* 能够掌握Oracle的游标

* Cursor

* 隐式　(CUD)

sql，%found,%notfound,%rowcount,%isopen

* 显示(R)

Cursor name is select ....

open

fetch(循环--->%notfound)

close

* 能够掌握Oracle的存储过程

* create or replace procedure pro_add(in,out)

is

...

begin

....

end;

* begin

pro_add;

pro_add(a,b,c,d,e);

end;

---

* 回顾
  * 表与表之间链接
    * 交叉链接（笛卡尔积）

      select * from employee,teacher;

    * 内链接

      select * from employee e inner join teacher t on e.tea_no=t.tea_no ;

      select * from employee e join teacher t on e.tea_no=t.tea_no;

      select * from employee e , teacher t where e.tea_no=t.tea_no;

    * 外链接
      * 左

        select * from employee e left join teacher t on e.tea_no=t.tea_no;

        select * from employee e , teacher t where e.tea_no=t.tea_no(+);

      * 右

        select * from employee e right join teacher t on e.tea_no=t.tea_no;

        select * from employee e , teacher t where e.tea_no(+)=t.tea_no;

      * 全

        select * from employee e full join teacher t on e.tea_no=t.tea_no;

  * 数据结构：B树，B-树，B+树
    * B树：Oracle默认索引存储机制
    * B+树; mysql默认索引存储机制、
  * 索引
    * 提供查询性能，避免全部扫描
    * create unique|bitmap index index_name on employee(emp_age);
    * create unique|bitmap index index_name on employee(emp_name,emp_age);
    * drop index index_name ;
  * SQL优化（少让数据库翻译，不要让索引失效）
    * 尽量不要写‘*’
    * SQL语句用大写

* 尽量使用内链接，少用外链接，少用子查询
* 在链接表过程，where写在on右边
* 索引相关
    * 尽量不要使用不等于 !=,<>
    * 尽量不要 like %dd%
    * order by 后面字段，尽量加索引
    * 在where 后面条件，判断时候，类型要一致
    * 尽量少用 not
    * 尽量少用 or，1 or 2 in（1,2）
    * 索引列要设置成not null
    * 尽量少用 is null,is not null
    * 子查询当中，尽量用exists代替in

* 序列
  * create sequence seq_no
    start with 1
    increment by 1
    maxvalue 5000
    cycle|nocycle
    cache 30
  * seq_no.nextval,seq_no.currval

* 分区表

```
create table employee(
...
)
partition by range(col_name)(
partiton P1 values less than ()
partiton P1 values less than ()
partiton P1 values less than (maxvalue)
```

)

partition byl ist(col_name)(

partiton P1 values ()

partiton P1 values ()

partiton P1 values (default)

)

* PL/SQL概述

* PL/SQL是 Procedure Language & Structured Query Language 的缩写

* PL /SQL是一种高级数据库程序设计语言，该语言专门用于在各种环境下对Oracle数据库进行访问。

* PL /SQL语言集成于数据库服务器中，所以PL/SQL代码可以对数据进行快速高效的处理。

* PL/SQL好处

* 对于客户/服务器环境来说，真正的瓶颈是网络上。

无论网络多快，只要客户端与服务器进行大量的数据交换。

应用运行的效率自然就回受到影响。如果使用PL/SQL进行编程，

将这种具有大量数据处理的应用放在服务器端来执行。

自然就省去了数据在网上的传输时间

* 能够掌握PL/SQL的语法

```
* pl/sql的构成
[DECLARE]
    --声明变量等;
BEGIN
    --程序主要部分，一般用来执行过程语句或SQL语句;
[EXCEPTION]
    --异常处理;
```

```
END;

* 运算符
=     等于
  比较运算符
<>,!=,~=,^=     不等于
<     小于
>     大于
<=     小于或等于
>=     大于或等于
 +     加号        算术运算符
 -     减号
 *     乘号
 /     除号
:=     赋值号      赋值运算符
 =>     关系号      关系号
..     范围运算符      范围运算符
||     字符连接符      连接运算符
is null     是空值        逻辑运算符
between and     介于两者之间
in      在一系列值中间
and     逻辑与
or     逻辑或
not     取反

*变量与常量
 * 变量：variable_name data_type[(size)][:=init_value];
 * 常量：variable_name CONSTANT  data_type[(size)] :=init_value;

* 数据类型
 *  常用标准类型：CHAR(CHARATER,NCHAR),VARCHAR2,NUMBER(P,S),DATE,BOOLEAN等
 * 属性类型：%TYPE与%ROWTYPE
    * %TYPE:可以用来定义数据变量的类型与已定义的数据变量（表中的列）一致。
    * %ROWTYPE：与某一数据库表的结构一致(修改数据库表结构，可以实时保持一致）
      访问方式声明为rowtype的变量名.字段名.

* 常量,变量,%TYPE,%ROWTYPE基本使用
/*
 准备工作：创建员工表，插入测试数据
*/
```

```
--drop table employee;
CREATE TABLE employee
(
     emp_no VARCHAR2(8) PRIMARY KEY NOT NULL,   --工号，主键，非空
     emp_name VARCHAR2(30) NOT NULL,--姓名,非空
     emp_id VARCHAR2(18), --身份证号，代表18位整数
     emp_age NUMBER(3,0)  --年龄
);
insert into employee(emp_no,emp_name,emp_id,emp_age) values('lg001','张大','441
insert into employee(emp_no,emp_name,emp_id,emp_age) values('lg002','张二','441
insert into employee(emp_no,emp_name,emp_id,emp_age) values('lg003','张三','441
insert into employee(emp_no,emp_name,emp_id,emp_age) values('lg004','张四','441
commit;
/*
定义常量或变量、赋值使用示例：定义常量c_name直接存放员公司名字，
定义变量e_name,e_age分别用来存放工号为lh001的员工（通过查询表）的姓名及年龄，
定义变量e_mark存放员工成绩，设定员工成绩为95.5,最后输出该员工的公司、姓名、年龄及成绩
*/
declare
    c_name constant varchar(20):='亮哥教育';
    e_name varchar(30);
    e_age number(3,0);
    e_mark number(3,1);
begin
  --赋值方式一：使用SELECT INTO给变量赋值
  select emp_name,emp_age into e_name,e_age from employee
  where emp_no='lg001';
   --赋值方式二：使用赋值操作符“:=”给变量赋值
  e_mark:=95.5;
   --输出相关信息，DBMS_OUTPUT.PUT_LINE为具有输出功能的函数
  dbms_output.put_line('公司:'||c_name);
  dbms_output.put_line('姓名:'||e_name);
  dbms_output.put_line('年龄:'||e_age);
  dbms_output.put_line('分数:'||e_mark);
end;
结果:
公司:亮哥教育
姓名:张大
年龄:19
分数:95.5
```

```
88
89
90  --%TYPE的使用示例：在上一示例基础上
91  declare
92    c_name constant varchar2(20):='亮哥教育';
93    e_name employee.emp_name%type;
94    e_age employee.emp_age%type;
95    e_mark number(3,1);
96  begin
97    select emp_name,emp_age into e_name,e_age from employee
98    where emp_no='lg001';
99    e_mark:=95.5;
100   dbms_output.put_line('公司:'||c_name);
101   dbms_output.put_line('姓名:'||e_name);
102   dbms_output.put_line('年龄:'||e_age);
103   dbms_output.put_line('分数:'||e_mark);
104 end;
105 结果：
106 公司:亮哥教育
107 姓名:张大
108 年龄:19
109 分数:95.5
110
111 --%ROWTYPE的使用
112 declare
113   c_name constant varchar2(20):='亮哥教育';
114   empinfo employee%rowtype;
115   e_mark number(3,1);
116 begin
117   select * into empinfo from employee
118   where emp_no='lg001';
119   e_mark:=95.5;
120   dbms_output.put_line('公司:'||c_name);
121   dbms_output.put_line('姓名:'||empinfo.emp_name);
122   dbms_output.put_line('年龄:'||empinfo.emp_age);
123   dbms_output.put_line('分数:'||e_mark);
124 end;
125 结果：
126 公司:亮哥教育
127 姓名:张大
```

年龄:19
分数:95.5

* 控制语句
语法1：
```
if <boolean表达式> then
    执行语句;
end if;
```
语法2：
```
if <boolean表达式 >then
    执行语句;
else
    执行语句;
end if;
```
语法3：
```
if <boolean表达式> then
    执行语句;
elsif <boolean表达式> then
    执行语句;
elsif <boolean表达式> then
    执行语句;
else
    执行语句;
end if;
```
使用示例
1。判断工号为lg002的员工的小孩是否可以上小学（国家规定需要6周岁才可以上小学），如果可
2。判断工号为lg001的员工的小孩是否可以上小学（国家规定需要6周岁才可以上小学），不论可
3。判断工号为lg001员工的小孩是否可以上小学（国家规定需要6周岁才可以上小学），超过6岁瘪

```
/*
 准备工作：创建员工表，插入测试数据
*/
--drop table employee;
CREATE TABLE employee
(
     emp_no CHAR(8) PRIMARY KEY NOT NULL,   --工号，主键，非空
     emp_name VARCHAR2(30) NOT NULL,--姓名,非空
     emp_id VARCHAR2(18), --身份证号，代表18位整数
     child_age NUMBER(3,0)  --年龄
```

```
);
insert into employee(emp_no,emp_name,emp_id,child_age) values('lg001','张大','4
insert into employee(emp_no,emp_name,emp_id,child_age) values('lg002','张二','4
insert into employee(emp_no,emp_name,emp_id,child_age) values('lg003','张三','4
insert into employee(emp_no,emp_name,emp_id,child_age) values('lg004','张四','4
commit;

/*
示例练习:
1。判断工号为lg002的员工的小孩是否可以上小学（国家规定需要6周岁才可以上小学），如果可
2。判断工号为lg001的员工的小孩是否可以上小学（国家规定需要6周岁才可以上小学），不论可
3。判断工号为lg001员工的小孩是否可以上小学（国家规定需要6周岁才可以上小学），超过6岁瘪
根据员工情况输入相关结果。
*/
--练习1
declare
   e_age employee.emp_age%type;
begin
  select emp_age into e_age from employee
  where emp_no='lg002';
  if e_age>=6 then
    dbms_output.put_line('此员工年龄为:'||e_age||'岁,可以上小学');
  end if;
end;
结果:
此员工孩子年龄为:6岁,可以上小学

--练习2
declare
  c_age employee.child_age%type;
begin
  select child_age into c_age from employee
  where emp_no='lg001';
  if c_age>=6 then
    dbms_output.put_line('此员工孩子年龄为:'||c_age||'岁,可以上小学');
  else
    dbms_output.put_line('此员工孩子年龄为:'||c_age||'岁,不可以上小学');
  end if;
end;
结果:
```

```
此员工孩子年龄为:5岁,不可以上小学

--练习3
declare
  c_age employee.child_age%type;
begin
  select child_age into c_age from employee
  where emp_no='lg003';
  if c_age>6 then
      dbms_output.put_line('此员工孩子年龄为:'||c_age||'岁,超龄');
  elsif c_age=6 then
      dbms_output.put_line('此员工孩子年龄为:'||c_age||'岁,可以上小学');
  else
      dbms_output.put_line('此员工孩子年龄为:'||c_age||'岁,不可以上小学');
  end if;
end;
结果:
 此员工孩子年龄为:7岁,超龄


* 循环控制
loop循环
语法:
loop
    执行语句;
    exit when <条件语句
end loop;
示例:
由于某次考试难度太大，需要给整个班级的员工加分，每次加5分，但最高分不能超过90分

/*
 准备工作：创建员工表，插入测试数据
*/
--drop table employee;
CREATE TABLE employee
(
     emp_no CHAR(8) PRIMARY KEY NOT NULL,   --工号，主键，非空
     emp_name VARCHAR2(30) NOT NULL,--姓名,非空
     emp_id VARCHAR2(18), --身份证号，代表18位字符
     emp_mark NUMBER(3,1)  --分数
```

```
);
insert into employee(emp_no,emp_name,emp_id,emp_mark) values('lg001','张大','44
insert into employee(emp_no,emp_name,emp_id,emp_mark) values('lg002','张二','44
insert into employee(emp_no,emp_name,emp_id,emp_mark) values('lg003','张三','44
insert into employee(emp_no,emp_name,emp_id,emp_mark) values('lg004','张四','44
commit;
select * from employee;

/*
由于某次考试难度太大，需要给整个班级的员工加分，每次加5分，但最高分不能超过90分
*/
declare
 e_mark employee.emp_mark%type;
begin
  loop
      select max(emp_mark) into e_mark from employee;
    exit when (e_mark+5)>90;
      update employee set emp_mark=emp_mark+5;
  end loop;
  commit;
end;


while循环
语法：
while <布尔表达式> loop
    执行语句；
end loop;

使用示例
继续使用上面的表，及完成：“由于某次考试难度太大，需要给整个班级的员工加分，每次加5分，
declare
 e_mark employee.emp_mark%type:=0;
begin
  select max(emp_mark) into e_mark from employee;
  while (e_mark+5)<=90  loop
    update employee set emp_mark=emp_mark+5;
    select max(emp_mark) into e_mark from employee;
  end loop;
  commit;
```

```
end;

for循环
语法：
for 循环计数器 in [REVERSE] 下限..上限 loop
    执行语句;
end loop;
说明：..两点表示范围，1..4表示时将从1到4进行循环，小（例如 1）写前边，REVERSE表示反转
使用示例
declare
  m_count number(3,0);
begin
   for m_count in 1..4 loop
      dbms_output.put_line(m_count);
   end loop;
end;
结果
1
2
3
4
declare
  m_count number(3,0);
begin
  for m_count in reverse 1..4 loop
     dbms_output.put_line(m_count);
  end loop;
end;
结果：
4
3
2
1

继续使用上面的表，及完成：“由于某次考试难度太大，需要给整个班级的员工加分，每次加5分，

declare
  m_timer number(3,0);
  m_count number(3,0);
  e_mark employee.emp_mark%type;
```

```
begin
  select max(emp_mark) into e_mark from employee;
  --dbms_output.put_line(e_mark);
  -- e_mark:=51;
  --m_timer:=(90-e_mark)/5;--假如e_mark:=51,m_timer=8,加了8次5之后等于91，超过90，
  -- floor 向下取整数的函数（不会四舍五入）;
  m_timer:=floor((90-e_mark)/5);
  for m_count in 1..m_timer loop
    update employee set emp_mark=emp_mark+5;
  end loop;
  dbms_output.put_line(m_count);
end;

*顺序控制
用于按指定顺序执行的语句。主要是null语句
null语句：是一个可执行语句，相当于一个占位符或不执行操作的空语句。主要用来提高程序语句
--广东或香港等有些地方不喜欢4的数字，在使用数字时跳过4
declare
 m_count number(3,0):=0;
begin
  while m_count<10 loop
    m_count:=m_count+1;
    if m_count=4 then
      null;--去掉null会的报错
    else
      dbms_output.put_line('m_count='||m_count);
    end if;
  end loop;
end;
结果：
m_count=1
m_count=2
m_count=3
m_count=5
m_count=6
m_count=7
m_count=8
m_count=9
m_count=10

```

```
* 异常处理
 * 异常语法
Exception
       when 异常名 then
        执行处理语句;
         --使用others确保不会漏过任何异常
       when others then
        执行处理语句;

经常配套使用的函数:
    SQLCODE函数:返回错误代码,
    SQLERRM函数:返回错误信息

例如输出异常信息:  DBMS_OUTPUT.PUT_LINE('其它异常,代码号:'||SQLCODE||',异常描述:
* 预定义异常
预定义异常指PL/SQL 程序违反 Oracle 规则或超越系统限制时隐式引发(由oracle自动引发)。
常见的预定义异常:
ZERO_DIVIDE:以零作为除数时出现
DUP_VAL_ON_INDEX:试图将重复的值存储在使用唯一索引的数据库列中
INVALID_NUMBER:试图将一个非有效的字符串转换成数字
TOO_MANY_ROWS :在执行SELECT INTO语句后返回多行时出现
VALUE_ERROR:变量的值超出变量大小
CURSOR_ALREADY_OPEN:试图打开已经打开的游标
ACCESS_INTO_NULL:试图给一个没有初始化的对象赋值
LOGIN_DENIED :使用无效的用户名和口令登录Oracle
NO_DATA_FOUND :语句无法返回请求的数据


使用示例
--预定义异常
declare
  s_result number(3,0);
begin
  s_result:=10/0;
  dbms_output.put_line('没有异常');
  Exception
    when zero_divide then
      dbms_output.put_line('除数不能为0,请核查');
    when others then
```

```
        --SQLCODE函数返回错误代码，SQLERRM函数返回错误信息
      dbms_output.put_line('其他异常代码号:'||sqlcode||',异常信息:'||sqlerrm);
end;
结果：
 除数不能为 0 ，请核查

* 自定义异常
自定义异常：程序在运行过程中，编程人员根据业务等情况，认为非正常情况，可以自定义异常。
第一、定义相关异常；在声明部分定义相关异常，格式：<自定义异常名称>  EXCEPTION;
第二、抛出异常；在出现异常部分抛出异常，格式：RAISE  <异常名称>;
第三、处理异常；在异常处理部分对异常进行处理，格式：when <自定义异常名称> then ...,

使用示例，完成以下功能：
根据业务的需要，提取员工的信息时，需要校验年龄不能为负数，
如果为负数，请自定义异常age_exception,并将其抛出，处理。
例如以如下的测试数据，取出身份证号：'441521199909092113'的员工，判断其年龄，并进行异
/*
 创建员工表，插入测试数据
*/
--drop table employee;
CREATE TABLE employee
(
     emp_no CHAR(8) PRIMARY KEY NOT NULL,    --工号，主键，非空
     emp_name VARCHAR2(30) NOT NULL,--姓名,非空
     emp_id VARCHAR2(18), --身份证号，代表18位整数
     emp_age NUMBER(3,0)  --年龄
);
insert into employee(emp_no,emp_name,emp_id,emp_age) values('lg001','张大','441
insert into employee(emp_no,emp_name,emp_id,emp_age) values('lg002','张二','441
insert into employee(emp_no,emp_name,emp_id,emp_age) values('lg003','张三','441
insert into employee(emp_no,emp_name,emp_id,emp_age) values('lg04','张四','4415
commit;

declare
   c_name constant varchar2(20):='亮哥教育';
   e_name employee.emp_name%type;
   e_age employee.emp_age%type;
   e_mark number(3,1);
   age_exception EXCEPTION;
begin
```

```
  select emp_name,emp_age into e_name,e_age from employee
  where emp_id='441521199909092111';
   --年龄小于0岁为异常情况，使用raise抛出异常
  if e_age < 0 then
      raise age_exception;
  end if;
  e_mark:=95.5;
  dbms_output.put_line('公司:'||c_name);
  dbms_output.put_line('姓名:'||e_name);
  dbms_output.put_line('年龄:'||e_age);
  dbms_output.put_line('分数:'||e_mark);
  EXCEPTION
    when age_exception then
      -- dbms_output.put_line('异常描述：年龄不能为负数！');
     RAISE_APPLICATION_ERROR(-20001,'年龄不能为负数！');
    when TOO_MANY_ROWS then
       dbms_output.put_line('异常描述'||Sqlerrm||',原因：身份证号码出现重复');
    when others then
     dbms_output.put_line('其他异常代码号:'||sqlcode||',异常信息:'||sqlerrm);
end;

结果：
  异常描述：年龄不能为负数！
```

* 能够掌握Oracle的游标

  * 游标概述

    * 游标是指oracle在执行增删改查操作时，会把执行结果放在内存分配的缓冲区中，游标就是指向该区的一个指针，可以对结果集的每一行数据分别进行处理。

  * 游标属性：

    * %found　　是否存在结果集或影响的行数，如果存在返回true

    * %notfound　是否存在结果集或影响的行数，如果不存在返回true

    * %rowcount　返回受影响的行数

    * %isopen　　游标是否已经打开。隐式游标中一般是自动打开和关闭的，查询都返回False

  * 游标的分类

* 隐式游标

* 由Oracle在内部声明，数据库自动创建，管理。

* 主要用途是用于增删改数据时，可以返回一个操作成功或失败的相关信息，名称是(sql)

* 显式游标

* 可以定义游标，自己使用

```
* 隐式游标
 示例：访问游标的属性
/*
 创建员工表，插入测试数据
*/
--drop table employee;
CREATE TABLE employee
(
     emp_no CHAR(8) PRIMARY KEY NOT NULL,   --工号，主键，非空
     emp_name VARCHAR2(30) NOT NULL,--姓名,非空
     emp_id VARCHAR2(18), --身份证号，代表18位整数
     emp_age NUMBER(3,0)  --年龄
);
insert into employee(emp_no,emp_name,emp_id,emp_age) values('lg001','张大','441
insert into employee(emp_no,emp_name,emp_id,emp_age) values('lg002','张二','441
insert into employee(emp_no,emp_name,emp_id,emp_age) values('lg003','张三','441
insert into employee(emp_no,emp_name,emp_id,emp_age) values('lg004','张四','441
commit;

--隐匿游标使用
declare
begin
    --insert into employee(emp_no,emp_name,emp_id,emp_age) values('lg005','张6'
    update employee set emp_age=17 where emp_id='441521199909092111' ;
    --delete from  employee  where  emp_id='441521199909092115' ;
    if sql%found then
       dbms_output.put_line('已经影响了数据');
    end if;
    if sql%notfound then
       dbms_output.put_line('没有影响数据');
```

```
    end if;
    if sql%isopen then
       dbms_output.put_line('游标open了');--隐式游标自动打开和关闭的，此时查询已经关
    else
       dbms_output.put_line('游标close了');
    end if;
    dbms_output.put_line('受影响的行:'||sql%rowcount);
end;

* 显示游标
使用步骤：
 * 在declare 声明游标：Cursor 游标名 is 查询语句(可以多个查询字段);
 * 打开游标：open 游标名;
 * 提取游标：fetch 游标名 into 变量1，变量2（变量的个数对应字段数）;
 * 关闭游标：close 游标名;
示例：
/*
 创建员工表，插入测试数据
*/
--drop table employee;
CREATE TABLE employee
(
     emp_no varchar2(8) PRIMARY KEY NOT NULL,   --工号，主键，非空
     emp_name VARCHAR2(30) NOT NULL,--姓名,非空
     emp_id VARCHAR2(18), --身份证号，代表18位整数
     emp_age NUMBER(3,0)  --年龄
);
insert into employee(emp_no,emp_name,emp_id,emp_age) values('gh001','张大','441
insert into employee(emp_no,emp_name,emp_id,emp_age) values('gh002','张二','441
insert into employee(emp_no,emp_name,emp_id,emp_age) values('gh003','张三','441
insert into employee(emp_no,emp_name,emp_id,emp_age) values('gh004','张四','441
commit;

--使用显式游标遍历（loop），查询并输出所有员工的工号和姓名
declare
   e_no employee.emp_no%type;
   e_name employee.emp_name%type;
   Cursor emp_cursor is select emp_no,emp_name from employee;-- 定义游标
begin
   -- 打开游标
```

```
   open emp_cursor;
   loop
     fetch emp_cursor into e_no,e_name;
     exit when emp_cursor%notfound;
     dbms_output.put_line('工号:'||e_no||',姓名:'||e_name);
   end loop;
   -- 关闭游标
   close emp_cursor;
end;
结果:
工号:lg001,姓名:张大
工号:lg002,姓名:张二
工号:lg003,姓名:张三
工号:lg004,姓名:张四

/*
使用显式游标(for)，查询并输出所有员工的工号和姓名,
for循环游标会隐式打开游标、自动创建%ROWTYPE类型变量对应记录行，
处理完所有行后会自动关闭游标.使用起来较方便。
*/
declare
  Cursor emp_cursor is select emp_no,emp_name from employee;
begin
  for empinfo in emp_cursor loop
      dbms_output.put_line('工号:'||empinfo.emp_no||',姓名:'||empinfo.emp_name);
  end loop;
end;
工号:lg001,姓名:张大
工号:lg002,姓名:张二
工号:lg003,姓名:张三
工号:lg004,姓名:张四

--使用显式游标修改数据，把年龄为负的员工年龄改为１８岁,其它都加一岁
declare
 Cursor emp_cursor is select emp_no,emp_age from employee for update;
begin
  for empinfo in emp_cursor loop
    if empinfo.emp_age<0 then
      update employee set emp_age=18 where emp_no=empinfo.emp_no;
     else
```

```
      update employee set emp_age=emp_age+1 where emp_no=empinfo.emp_no;
    end if;
  end loop;
end;
```

* 能够掌握Oracle的存储过程

  * 存储过程概述

    * 存储过程属于已命名的pl/sql程序块，封装数据业务操作，具有模块化、可重用、可维护、更安全特点。

  * 存储过程类型

    * 不带参数

    * 带输入参数

    * 带输出参数

    * 带输入输出参数

```
* 创建与调用语法
 * 创建语法
CREATE [OR REPLACE] PROCEDURE procedure_name[(param_list)]
    IS|AS
 ....
BEGIN
   执行语句;
[EXCEPTION]
    异常处理;
END[procedure_name];
说明:
OR REPLACE: 如果系统已存在该存储过程，将被替换
procedure_name: 存储过程名称
param_list: 参数列表，参数不需要声明长度，可选
DECLARE: 局部声明，可选
* 调用语法
 * 命令行调用:
   * exec|execute procedure_name(paramlist);
 * pl/sql块调用:
```

```
  begin
    procedure_name(paramlist);
end;

* 测试
/*
 创建员工表，插入测试数据
*/
--drop table employee;
CREATE TABLE employee
(
     emp_no varchar2(8) PRIMARY KEY NOT NULL,   --工号，主键，非空
     emp_name VARCHAR2(30) NOT NULL,--姓名,非空
     emp_id VARCHAR2(18), --身份证号，代表18位整数
     emp_age NUMBER(3,0)  --年龄
);
insert into employee(emp_no,emp_name,emp_id,emp_age) values('lg001','张大','441
insert into employee(emp_no,emp_name,emp_id,emp_age) values('lg002','张二','441
insert into employee(emp_no,emp_name,emp_id,emp_age) values('lg003','张三','441
insert into employee(emp_no,emp_name,emp_id,emp_age) values('lg004','张四','441
commit;

* 无参存储过程
--新建存储过程(不带参数)，可以新增员工信息
create or replace procedure pro_add_proceure is
begin
  insert into employee(emp_no,emp_name,emp_id,emp_age) values('lg005','张五','4
  commit;
end;

--调用存储过程
begin
  pro_add_proceure;
end;
select * from employee;

* 有输入参数存储过程
--新建存储过程(带输入参数,输入参数类型in可以省略不写），可以新增员工信息
create or replace procedure pro_add_procedure(
    e_no employee.emp_no%type,
```

```
    e_name employee.emp_name%type,
    e_id employee.emp_id%type,
    e_age employee.emp_age%type
) is
begin
  insert into employee(emp_no,emp_name,emp_id,emp_age) values(e_no,e_name,e_id,
  commit;
end;
/
--调用存储过程
begin
  --按位置传递参数
  -- pro_add_procedure('lg008','凌凌漆','441521199909092115',20);
  --按名称传递参数,顺序可变
   pro_add_procedure(e_no=>'lg009',e_id => '441521199909092115',e_name => '凌凌
end;

* 有输出参数存储过程
--新建存储过程(带输出参数），可以新增员工信息
create or replace procedure pro_add_procedure(
       s_flag out number,
       s_message out varchar
)
is
begin
  insert into employee(emp_no,emp_name,emp_id,emp_age) values('a010','张10','44
  commit;
  s_flag:=1;
  s_message:='添加成功';
  EXCEPTION
    when DUP_VAL_ON_INDEX then
      s_flag:=20001;
      s_message:='工号已经被使用，请核查！';
    when others then
      s_flag:=sqlcode;
      s_message:=sqlerrm;
end;

/
--调用存储过程
```

```
declare
   p_flag number(10,0);
   p_message varchar2(50);
begin
   pro_add_procedure(p_flag,p_message);
   dbms_output.put_line(p_flag||':'||p_message);
end;

* 有输入输出参数存储过程
--新建存储过程(带输入输出参数,输入参数类型in可以省略不写），可以新增员工信息
create or replace procedure pro_add_procedure(
    e_no employee.emp_no%type,
    e_name employee.emp_name%type,
    e_id employee.emp_id%type,
    e_age employee.emp_age%type,
    s_flag out number,
    s_message out varchar
)is
begin
  insert into employee(emp_no,emp_name,emp_id,emp_age) values(e_no,e_name,e_id,
  commit;
  s_flag:=1;
  s_message:='添加成功';
  Exception
     when DUP_VAL_ON_INDEX then
       s_flag:=20001;
       s_message:='工号已经被使用，请核查！';
     when others then
       s_flag:=sqlcode;
       s_message:=sqlerrm;
end;
/
--调用存储过程
declare
   p_flag number(10,0);
   p_message varchar2(50);
begin
   pro_add_procedure('lg1231','小白','441521199909092119',30,p_flag,p_message);
   dbms_output.put_line(p_flag||':'||p_message);
end;
```

```
140
141 * 删除存储过程
142   drop procedure pro_add_procedure;
```