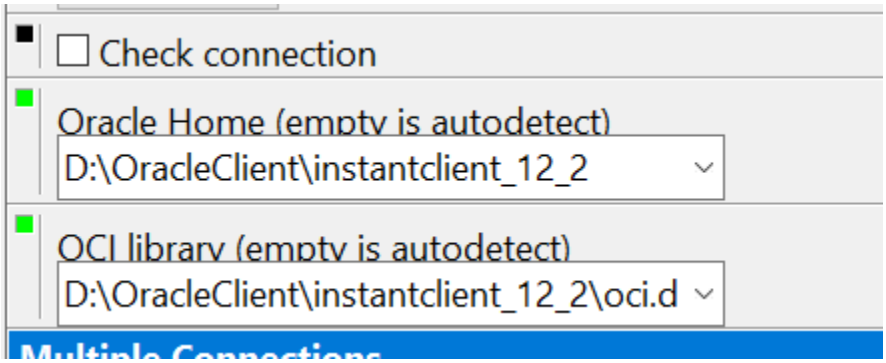
ORACLE

1. PLSQL安装

1.1.



2. 数据表基本操作

2.1. 创建数据表

2.1.1. 主键

主键就是区别这个表的唯一关键字

CREATE TABLE tb_emp1 (id NUMBER(11) , name varchar2(20));

.... id NUMBER(11) PRIMARY KEY.....

..... , PRIMARY KEY(name,did));

ALTER TABLE tb_emp1 ADD CONSTRAINTS pk_id PRIMARY KEY (id)

主键自增(ORACLE 12c 新加了自增特性/之前需要序列和触发器实现)

一个表只能有一个字段使用自增约束,且该字段必须是主键的一部分,默认情况下,初始值为1 ,每增加一条记录,字段值自动加1 **create table t1 (userid number GENERATED ALWAYS AS IDENTITY PRIMARY KEY ,uname varchar2(200));**

序列和触发器

```
create sequence seq_sal_his
 start with 1
 increment by 1;

create or replace trigger trig_sal_his_seq
 before insert on sal_his
 for each row
begin
 select seq_sal_his.nextval into :new.id from dual;
end trig_sal_his_seq;
```

序列

```
create sequence mysq
increment by 1
start with 1;

INSERT INTO tb_1 values( mysq.nextval,'xxxx' )
```

2.1.2. 外键

外键就是跟其他表联系的字段,on delete no action/cascade/set null ,默认no action
,cascade级联操作,比如有个考试表中有个SID指向学生表中的SID外键,删除掉某个学生SID,考试表对应的元组会被同时删除

```
..... , CONSTRAINT fk_emp_dept FOREIGN KEY(did) REFERENCES  tb_dept(did)
);

ALTER TABLE tb_emp5 ADD CONSTRAINTS  fk_name FOREIGN KEY (did)
REFERENCES tb_dept (did) on DELETE CASCADE
```

2.1.3. 其他约束

NOT NULL / UNIQUE /DEFAULT ' '

CHECK

(gender VARCHAR2(20) , CONSTRAINT check_name CHECK(GENDER='男' or GENDER ='女'));

ALTER TABLE tb_emp ADD CONSTRAINT check_name CHECK (age>=15 and age<=20)

ALTER TABLE tb_emp DROP CONSTRAINTS check_name

2.2. 修改数据表

2.2.1. 修改表名 : ALTER TABLE tb_1 RENAME TO tb_2

2.2.2. 修改字段数据类型 : ALTER TABLE tb_1 MODIFY name VARCHAR2(20) ; //原本有个name varchar2(22) 的字段

2.2.3. 修改字段名: ALTER TABLE tb_1 RENAME COLUMN name TO tname

2.2.4. 添加字段: ALTER TABLE tb_1 ADD column1 VARCHAR(2) NOT NULL

2.2.5. 删除字段: ALTER TABLE tb_1 DROP COLUNM column1

2.3. 删除数据表

2.3.1. DROP TABLE tb_1

存在外键约束的时候,主表不能被直接删除,先删除外键: **ALTER TABLE tb_1 DROP CONSTRAINTS fk_name**

3. 数据类型和运算符

3.1. 数值数据类型

3.1.1. 整数类型和小数类型NUMBER（precision, scale）

NUMBER(5,2) 可变长 ：有效数字 五位,小数点后两位

NUMBER整数部分允许的长度为（precision-scale），无论scale是正数还是负数。

如果precision小于scale，表示存储的是没有整数的小数

NUMBER(5,-2) :小数点像左两位四舍五入 1234567 -> 1234600

NUMBER(2,3): 小数点向右三位四舍五入,而且有效数字为2位

NUMBER(11):有效长度11位,无小数,输入带小数的值小数部分会被截取值不能大于限制,但是当小数位多出之后,后被四舍五入截取

3.2. 日期和时间类型

3.2.1. DTAE / TIMESTAMP

DATE 只显示日期: INSERT INTO tb_1 VALUES(to_date('2005-01-01 13:13:20 ' ,' yyyy-mm-dd HH24:mi:ss'));只显示日期,时间被省略掉了

TIMESTAMP: INSERT INTO tb_1 VALUES(to_timestamp('2013-9-12 12:03:22.999 ' ,' yyyy-mm-dd hh24 :mi:ss:ff'));

3.3. 字符串类型

3.3.1. CHAR / VARCHAR2 / NVARCHAR2 / NCHAR / LONG

CHAR 用于描述定长的字符型数据

NCHAR 用于存储Unicode字符集的定长字符类型数据

VARCHAR2 用来存储可变长的字符型数据

NVARCHAR2 用来存储Unicode字符集的可变长字符型数据

LONG 用来存储变长的字符串

CHAR和VARCHAR2 的区别

1.char是固定长度字符,varchar2是可变长度字符;char会自动补齐插入数据的尾部空格,后者不会

2.char的处理速度更快,缺点是浪费空间.所以对存储不大但在速度山有要求的可以使用char 类型,反之可以使用carchar

3.如果一个varchar2
列经常被修改.而且每次被修改的数据的长度不同,这会引起"行迁移"现象,这会造成多余的IO,是数据库设计和调整中尽力要避免的,在这种情况下使用char更好

3.4. 二进制类型

3.4.1. RAW 可变长

3.5. 运算符

3.5.1. 算术

3.5.2. 比较:> / < / >= / <= / != / <> /IS NULL / Like / Between / In / Exsist

3.5.3. 连接: || 用来连接两个字符串 ' a' || 'aa'

3.5.4. 逻辑:not 的优先级 > and 的优先级 > or 的优先级

4. 插入更新删除数据

4.1. INSERT INTO tb_1(id,name) values(2,'name')

4.2. UPDATE tb_1 SET name ='name' WHERE id = 2

4.3. DELETE FROM tb_1 WHERE id =2

5. SELECT

5.1. 单表查询

5.1.1. IN /BETWEEN AND/LIKE/IS NULL

SELECT * FROM tb_1 WHERE age IN(20,22) AND grade BETWEEN 2 AND 20 AND class IS NULL AND name LIKE 'a%'

LIKE: _ 匹配单个字符, % 匹配多个字符

5.1.2. DISTINCT

SELECT DISTINCT id FROM tb_1

5.1.3. ORDER BY

SELECT * FROM tb_1 ORDER BY id,name DESC(先按照id排序 再按照name,DESC降序排序,默认为升序排序(A-Z))

5.1.4. GROUP BY HAVING

GROUP BY 通常和聚合函数一起使用

SELECT t_ad，COUNT(*) FROM tb_1 GROUP BY t_id

HAVING 和 GROUP BY

一起限定显示记录所需满足的条件,只有满足条件的分组才会被显示

```
SELECT onum ,SUM(quantity*price) AS orderTotal
FROM order GROUP BY onum
 HAVING SUM(quantity*price)>=100    ORDER BY orderTotal
```

5.1.5. 子查询 ANY(SOME) /ALL /EXISTS

Exists：子查询至少返回一行时条件为true。

Not Exists：子查询不返回任何一行时条件为**true**。

In：与子查询返回结果集中某个值相等。

Not In：与子查询返回结果集中任何一个值不相等。

>ANY：比子查询返回结果中的某个值大。

=ANY：与子查询返回结果中的某个值相等。

<ANY：比子查询返回结果中的某个值小。

>ALL：比子查询返回结果中的所有值都大。

<ALL ：比子查询返回结果中的所有值都小。

5.2. 多表查询

5.2.1. JOIN

INNER JOIN:相当于多表查询 **WHERE** 子句 **(**内连接**)**,相当于**JOIN**

```
SELECT
  a.sno
FROM
  sc a
  JOIN sc b ON a.sno = b.SNO
  where a.CNO = 'c001' and b.CNO = 'c002'
```

可以实现**sc**表中的**CNO** 既满足 **= c001** 也满足 **c002**的条件

FROM table_b LEFT OUTER JION table_a ON table_b.id = table_a.id :
返回所有**b**表中的所有**select**的记录和右表中连接字段相等的**select**记录

demo

SID	SNAME	SAGE	MATH	ENG	CN
1	Paul	12	120	120	120
2	Eric	14	110	110	110
3	Bob	11	140	120	111
4	Hong	(Null)	(Null)	(Null)	(Null)

```
SELECT
STU.SID, STU.SNAME,STU.SAGE,GRADE.MATH,GRADE
.ENG,GRADE.CN
from STU  left outer join  GRADE
on GRADE.SID=STU.SID
```

FROM table_b RIGHT OUTER JION table_a ON table_b.id = table_a.id :

返回所有a表中的所有select的记录和b表中连接字段相等的select记录

SID	SNAME	SAGE	MATH	ENG	CN
1	Paul	12	120	120	120
2	Eric	14	110	110	110
3	Bob	11	140	120	111

```
SELECT
STU.SID, STU.SNAME,STU.SAGE,GRADE.MATH,GRADE
.ENG,GRADE.CN
from STU  RIGHT outer join  GRADE
on GRADE.SID=STU.SID
```

6. Oracle函数

6.1. 聚合函数

6.1.1. AVG 平均值 AVG（表达式）

6.1.2. SUM 求和 SUM(表达式)

6.1.3. MIN、MAX 最小值、最大值 MIN(表达式)、MAX(表达式)

6.1.4. COUNT 数据统计 COUNT（表达式）

COUNT(*) 统计表的总行数

6.2. 常见函数

6.2.1. DECODE()

```
DECODE(value_expression,search_value,  result_value,default_value)

SELECT JOB,DECODE(JOB,
'CLERK','general work','MANAGER','administrative work',
'ANALYST', 'technician work','SALESMAN','sale work',
NULL, 'not known') from EMP
```

6.2.2. CASE()

```
-- select sal ,(case  when sal<1000 then 'lower'
--    when sal<=2000 and sal>1000 then 'modest'
--    when sal<=4000 and sal>2000 then 'high'
-- else 'too high'  end) aa from EMP
```

6.2.3. TRANSLATE()

TRANSLATE(expr,from,to)

expr: 代表一串字符，from 与 to

是从左到右一一对应的关系，如果不能对应，则视为空值。

select translate('abcbbaadef','ba','#@') from dual

（b将被＃替代，a将被@替代）

select translate('abcbbaadef','bad','#@') from dual

（b将被＃替代，a将被@替代，d对应的值是空值，将被移走）

因此：结果依次为：@#c##@@def 和@#c##@@ef

6.2.4. TO_CHAR()

完成从日期数据到字符串数据的转换过程。

TO_CHAR(sysdate, 'yyyy, month dd, HH24:MI:SS')

6.2.5. TO_DATE()

指定的字符串转换成一个日期和时间。

TO_DATE('2008-09-30', 'yyyy-mm-dd')

7. PL/SQL

7.1. 基础

7.1.1. DEMO

```
DECLARE v_str varchar2 (20);
BEGIN v_str :='你好！ ';
dbms_output.put_line (v_str); END;
```

7.1.2. 循环

else is 79

```
DECLARE
v_abs number(8);
BEGIN
  v_abs := ABS(86+20*3-15**2);
  if(v_abs >80) then
    DBMS_OUTPUT.put_line('is'||v_abs);
    ELSE
      DBMS_OUTPUT.put_line('else is'||v_abs);
    END if;
END;
```

OK

```
DECLARE
v_abs number(8);
BEGIN
  v_abs:= 88;
  IF v_abs<60 then dbms_output.put_line ('bujige');
  elsif v_abs>=60 and v_abs<70 then dbms_output.put_line ('jige');
  elsif v_abs>70 and v_abs<100 then dbms_output.put_line ('ok');
  end if;
END;
```

ERRO

```
DECLARE
fno VARCHAR2(20);
BEGIN
  SELECT SNO  into fno FROM STUDENT WHERE SAGE >23;
  case fno
  when 's001' THEN dbms_output.put_line ('s001');
  when  ' s002' THEN  dbms_output.put_line ('s002');
  else dbms_output.put_line ('erro');
  end case;
END;
```

```
DECLARE
fno VARCHAR2(20);
BEGIN
  SELECT SNO  into fno FROM STUDENT WHERE SAGE >23;
  case
  when fno ='s001' THEN dbms_output.put_line ('s001');
  when  fno =' s002' THEN  dbms_output.put_line ('s002');
  else dbms_output.put_line ('erro');
  end case;
END;
```

```
DECLARE vsum number(4):= 10;
begin
  <<bb_loop>>
  LOOP
  dbms_output.put_line ('now vsum = '|| vsum);
  vsum := vsum -2;
  if( vsum <1 ) then
    dbms_output.put_line ('exits loop ,now vsum='|| vsum);
    exit bb_loop;
  end if;
  end loop;
END;
```

```
now vsum = 10
now vsum = 8
now vsum = 6
now vsum = 4
now vsum = 2
exits loop ,now vsum=0
```

```
DECLARE a number :=0;
begin
  <<repeat_loop>>

    a :=a+1;
    if mod(a,3)!=0
    then dbms_output.put_line(a);
    end if ;

  if a < 13 then
    goto repeat_loop;
  end if;

end;
/
```

7.1.3. 异常

异常语法结构

DELCARE e_DuplicateActors EXCEPTION;

EXCEPTION

WHEN exception_name THEN sequence_of_statements1;

WHEN exception_name THEN sequence_of_statements2;

[WHEN OTHERS THEN sequence_of_statements3;]

预定义异常

Oracle错误编号	等价异常的名称	说明
ORA-0001	DUP_VAL_ON_INDEX	违反唯一性约束
ORA-1403	NO_DATA_FOUND	没有找到数据
ORA-1422	TOO_MANY_ROWS	SELECT INTO语句查询到的记录超过了一行
ORA-1476	ZERO_DIVIDE	被0整除
ORA-6502	VALUE_ERROR	切割运算、算术运算或转换运算中出现的错误

7.2. 高级

7.2.1. 入参的三种模式

in模式参数:输入模式的参数，用于接收参数，在子程序内部，不能进行修改。默认的参数模式：in。不能对其设置数据类型的范围,它由传入的数据决定

```
pname in varchar
```

out输出模式的参数，用于输出值，会忽略传入的值。在子程序内部可以对其进行修改。

输出：子程序执行完毕后，out模式参数最终的值会赋值给调用时对应的<实参变量>。

注意：out模式参数的调用，必须通过变量。

in out

模式参数:输入输出模式：能接收传入的实参值；在子程序内部可以修改；可以输出（必须用实参变量调用）

7.2.2. 函数

```
CREATE [ OR REPLACE ] FUNCTION function_name
[ ( parameter_name [ IN | OUT | IN OUT ] type [,...] ) ]
 RETURN type { IS | AS } [ variable type [,...]]
BEGIN
   function_body
END function_name;
```

```
create or REPLACE FUNCTION f_my1(pname in out varchar) return boolean
as
   null_value exception;
begin
   if pname is null
    then  raise null_value;
    return false;
```

```
 end if;
  select ENAME into pname from EMP where ENAME like '%'||pname||'%';
  return true;
  exception
     when null_value then dbms_output.put_line('null value');return false;
     when too_many_rows
    then dbms_output.put_line('结果太多');
    return false;
   when NO_DATA_FOUND
    then dbms_output.put_line('未找到数据');
    return false;
end;
*****************************************************
declare
 v_dname varchar2(20);
begin
 v_dname := 'AA';
 if f_my1(v_dname) then
  dbms_output.put_line(v_dname);
 else
  dbms_output.put_line('出错了');
 end if;
end;
```

函数 必须要有返回值,调用的时候需要在语句里面调用 ,IN OUT

同样可以完成入参和返回参数的作用(create or REPLACE FUNCTION

f_my1(pname in out varchar) return boolean

as

  null_value exception;

```
begin
  if pname is null
   then raise null_value;
   return false;
 end if;
  select ENAME into pname from EMP where ENAME like '%'||pname||'%';
  return true;
  exception
    when null_value then dbms_output.put_line('null value');return false;
    when too_many_rows
   then dbms_output.put_line('结果太多');
   return false;
  when NO_DATA_FOUND
   then dbms_output.put_line('未找到数据');
   return false;
end;
**************************************************
declare
 v_dname varchar2(20);
begin
```

```
 v_dname := 'AA';
 if f_my1(v_dname) then
  dbms_output.put_line(v_dname);
 else
  dbms_output.put_line('出错了');
 end if;
end;)
```

7.2.3. 存储过程

```
CREATE [ OR REPLACE ] PROCEDURE procedure_name
[ ( parameter_name [ IN | OUT | IN OUT ] type [,...] ) ] { IS | AS } [ variable type
[,...]]
 BEGIN
   procedure_body
END procedure_name;

create or REPLACE PROCEDURE p_my
(p_did in number )
as        v_name varchar2(20);
begin
   SELECT ename into v_name FROM EMP
   where EMP.EMPNO = p_did;
   dbms_ouput.put_line(v_name);
end p_my;
begin    p_my(7369);  end;

create  or REPLACE PROCEDURE p_my1
(pno  in number,pname out varchar2)
```

```
as   begin
   SELECT ENAME into pname from EMP where EMPNO =pno;
end;

DECLARE
   v_name varchar2(20);
begin
   p_my1(7369,v_name);
   dbms_output.put_line(v_name);
end;

create or replace  PROCEDURE p_my3( pname in out varchar2 )
as
   null_value exception;
   v_oldname varchar2(30);
begin
   if pname is null then
      dbms_output.put_line('null');
      return ;
   end if;
   v_oldname:=pname;
   SELECT ENAME into pname from EMP where ENAME
   like '%'||pname||'%';
   dbms_output.put_line(pname);
   exception
      when null_value then dbms_output.put_line('null');
      when too_many_rows
    then dbms_output.put_line('结果太多');
   when NO_DATA_FOUND
    then dbms_output.put_line('未找到数据');
```

```
end p_my3;

DECLARE
  v_name varchar2(30);
begin
  v_name :='AA';
  p_my3(v_name);
end;
```

out模式 可以实现返回参数的功能,

调用存储过程后进去的对应参数被

操作之后会被返回出来(

```
create  or REPLACE PROCEDURE p_my1

(pno  in number,pname out varchar2)

as   begin

  SELECT ENAME into pname from EMP where EMPNO =pno;

end;



DECLARE

  v_name varchar2(20);

begin

  p_my1(7369,v_name);

  dbms_output.put_line(v_name);

end;
```

)

IN OUT 模式可以使用一个参数来完成 传入和返回的功能(create or replace PROCEDURE p_my3(pname in out varchar2)

as

  null_value exception;

  v_oldname varchar2(30);

begin

  if pname is null then

    dbms_output.put_line('null');

    return ;

  end if;

  v_oldname:=pname;

  SELECT ENAME into pname from EMP where ENAME

  like '%'||pname||'%';

  dbms_output.put_line(pname);

  exception

    when null_value then dbms_output.put_line('null');

    when too_many_rows

   then dbms_output.put_line('结果太多');

  when NO_DATA_FOUND

   then dbms_output.put_line('未找到数据');

```
end p_my3;


DECLARE
  v_name varchar2(30);
begin
  v_name :='AA';
  p_my3(v_name);
end;)
```

7.2.4. 触发器

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE|AFTER} trigger_event
ON table_name
[REFERENCING [OLD AS old_name][NEW AS new_name]]
[WHEN trigger_condition]
[FOR EACH ROW]
[DECLARE]
BEGIN
  trigger_body
END trigger_name;

CREATE  or REPLACE TRIGGER emp_tri1
before INSERT or DELETE or UPDATE on EMP
*****************************************************
begin
  if inserting then
```

```
    INSERT into EMP_BLOG VALUES(user,systimestamp,'insert');
  elsif UPDATING then
    INSERT into EMP_BLOG VALUES(user,systimestamp,'update');
  elsif DELETING then
    INSERT into EMP_BLOG VALUES(user,systimestamp,'delete');
  end if;
end;
*****************************************
update EMP set EMP.JOB ='test' where EMP.EMPNO = 7369;
*****************************************
```

主键自增

```
  create sequence seq_sal_his
   start with 1
   increment by 1;

  create or replace trigger trig_sal_his_seq
   before insert on sal_his
   for each row
  begin
   select seq_sal_his.nextval into :new.id from dual;
  end trig_sal_his_seq;

create or replace trigger trig_sal_his
 after insert or delete or update on emp
 for each row
begin
 if inserting then
  insert into sal_his values (null,:new.empno,:new.ename,0,sysdate);
 elsif updating then
  if :new.sal != :old.sal then
   insert into sal_his values (null,:new.empno,:new.ename,:old.sal,sysdate);
```

```
  end if;
 else
  delete from sal_his where empno = :old.empno;
 end if;
end trig_sal_his;
```

7.2.5. 游标

游标的作用就是用于临时存储从数据库中提取的数据块。

游标有两种类型：显式游标和隐式游标。在前述程序中用到的**SELECT...INTO..**.查询语句，一次只能从数据库中提取一行数据，对于这种形式的查询和**DML**操作，系统都会使用一个隐式游标。但是如果要提取多行数据，就要由程序员定义一个显式游标，并通过与游标有关的语句进行处理。显式游标对应一个返回结果为多行多列的**SELECT**语句。

显示游标

```
declare cursor  curTest1
 is
      select * FROM EMP;
      recEmp emp%ROWTYPE;/* 定义一个游标变量*/
 begin
  open curTest1;
  FETCH curTest1 into recEmp;
  while curTest1%FOUND LOOP
      Dbms_Output.put_line(recEmp.empno||' name is '||recEmp.ename);
      FETCH curTest1 into recEmp;
  end loop;
  close curTest1;
  end;
```

```
declare
 cursor my_cur(did number) is
  select * from emp where deptno = did;
 v_row emp%rowtype;
begin
 open my_cur(20);
 fetch my_cur into v_row;
 loop
  exit when my_cur%notfound;
  dbms_output.put_line(v_row.empno || '===>' || v_row.ename);
  fetch my_cur into v_row;
 end loop;
 dbms_output.put_line('该部门共有'||my_cur%rowcount||'名员工。');
 close my_cur;
end;
```

隐式游标

```
declare
 cursor my_cur(did number) is
  select * from emp where deptno = did;
  v_count number;
begin
 for v_row in my_cur(20) loop
  dbms_output.put_line(v_row.empno || '===>' || v_row.ename);
  v_count := my_cur%rowcount;
 end loop;
 dbms_output.put_line('该部门共有'||v_count||'名员工。');
end;
```

REF 游标

基础知识

1，什么是 REF游标 ？

动态关联结果集的临时对象。即在运行的时候动态决定执行查询。

2，REF 游标 有什么作用？

实现在程序间传递结果集的功能，利用REF CURSOR也可以实现BULK SQL，从而提高SQL性能。

3，静态游标和REF 游标的区别是什么？

①静态游标是静态定义，REF 游标是动态关联；

②使用REF 游标需REF 游标变量。

③REF 游标能做为参数进行传递，而静态游标是不可能的。

4，什么是REF 游标变量？

REF游标变量是一种 引用 REF游标类型

的变量，指向动态关联的结果集。

5，怎么使用 REF游标 ？

①声明REF 游标类型，确定REF 游标类型；

⑴强类型REF游标：指定retrun type，REF 游标变量的类型必须和return type一致。

语法：Type REF游标名 IS Ref Cursor Return 结果集返回记录类型；

⑵弱类型REF游标：不指定return

type，能和任何类型的CURSOR变量匹配，用于获取任何结果集。

语法：Type REF游标名 IS Ref Cursor；

②声明Ref 游标类型变量；

语法：变量名 已声明Ref 游标类型；

③打开REF游标，关联结果集 ；

语法：Open Ref 游标类型变量 For 查询语句返回结果集；

④获取记录，操作记录；

语法：Fatch REF游标名 InTo

临时记录类型变量或属性类型变量列表；

⑤关闭游标，完全释放资源；

语法：**Close REF**游标名；

功能用法

Oracle提供**REF CURSOR**，通过该功能可以实现在程序间传递结果集的功能，利用**REF CURSOR**也可以实现**BULK SQL**，从而提高**SQL**性能。

```
DECLARE
 TYPE mytable IS TABLE OF emp%ROWTYPE;
 l_data mytable;
 l_refc sys_refcursor;
BEGIN
 OPEN l_refc FOR
 SELECT empno, ename, job, mgr, hiredate, sal, comm, deptno FROM emp;
 FETCH l_refc BULK COLLECT INTO l_data;
 CLOSE l_refc;
 FOR i IN 1 .. l_data.COUNT
 LOOP
 DBMS_OUTPUT.put_line ( l_data (i).ename|| ' was hired since  '|| l_data
(i).hiredate);
 END LOOP;
 END;
```

让**REF**游标作为参数传递

```
create or replace function returnacursor return sys_refcursor
is
  v_csr sys_refcursor;
begin
```

```
    open v_csr for select a1 from test3;
    return v_csr;
end;
```

分页

```
create or replace procedure showPage(
      pageNo in number, --要显示哪一页
      pageSize in number, --每一页显示多少
      rowsCount out number, --总共多少条记录
      pageCount out number, --能分多少页
      cur out sys_refcursor --查询结果
)
as
begin
 select count(empno) into rowsCount from emp;
 if mod (rowsCount,pageSize) != 0 then
    pageCount :=round(rowsCount/pageSize)+1 ;
 else
    pageCount := rowsCount/pageSize;
 end if;
 open cur for select empno,ename from (select rownum rn,a.* from emp a)
     where  rn>= ((pageNo-1)*pageSize)+1 and rn<=pageNo*pageSize;
end;

cs = conn.prepareCall("{call showPage(?,?,?,?,?)}");
cs.setInt(1, 1);
cs.setInt(2, 3);
cs.registerOutParameter(3, oracle.jdbc.OracleTypes.NUMBER);
cs.registerOutParameter(4, oracle.jdbc.OracleTypes.NUMBER);
cs.registerOutParameter(5, oracle.jdbc.OracleTypes.CURSOR);
cs.execute();
```

```
list1.add(cs.getInt(3));
list1.add(cs.getInt(4));
rs = (ResultSet) cs.getObject(5);
while (rs.next()) {
      Emp emp = new Emp();
      emp.setEmpno(rs.getInt(1));
      emp.setEname(rs.getString(2));
      list.add(emp);}
```