* 学习目标

* 能够掌握Oracle的存储过程

    * procedure

* 能够掌握使用JDBC调用Oracle的存储过程

    * CallableStatement

        * {call 存储过程的名字(?,?)}

        * ...

* 能够掌握Oracle的存储函数

    * function

    * 返回值，用途

* 能够掌握使用JDBC调用Oracle的存储函数

    * { ? =call name(?,?)}

* 能够掌握Oracle的程序包对象

    * package

    * package body

* 能够掌握使用JDBC调用程序包对象

    * {call 包名.存储过程的名字(?,?)}

* 能够掌握Oracle的触发器

    * create trigger name

        before|after insert update delete on tablename for each row

        begin

        end;

    * 自动增长主键

    * 备份

* 能够掌握Oracle的视图

    * 虚拟表

        * 安全性

* 简单性

* 隔离性

* 角度性

* create view emp5 as select ...

* 能够掌握Oracle的闪回，备份，恢复

* delete commit,flashback,scn

* exp 备份

* imp 恢复

---

* 回顾

* PL/SQL

` * 网络瓶颈

* 变量，常量，运算符，条件语句，循环语句，游标（数组，容器），异常

* 变量：number（3,0），varchar2，date，...

%type,%rowtype

* 常量：constant

* 运算符：:=,=,||,=>,..

* if c then

elsif c then

else

end if;

* loop

exit when c

end loop;

* while c loop

end loop;

* for mcount in[reverse] 1..3 loop

end loop;

* 隐式游标

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

* 显示 游标

定义： Cursor cursor_name is select * from employee;

```
open cursor_name ;
    loop
     exit when sql%notfound;
     fetch cursor_name into ...
    end loop;
close cursor_name;
for empinfo in cursor_name loop
end loop;
```

* 异常

```
EXCEPTION
when ZERO_DIVIDE then
when others then
raise
...
```

* 存储过程

```
create or replace procedure pro_add(
)
...
is
begin
end;
begin
    pro_add;
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
 * 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;

* 删除存储过程
  drop procedure pro_add_procedure;
```

* 能够掌握使用JDBC调用Oracle的存储过程

```
关键语法：
 * CallableStatement call=conn.prepareCall("{call 存储过程名(?,?,?,?,?,?)}");
    @Test
    public void test1() {
        Connection conn = null;
        CallableStatement call = null;
        try {
            conn = ConnectionUtils.getConnection();
            String sql="{call pro_add_procedure(?,?,?,?,?,?)}";
            // pro_add_procedure('lg1231','小白','441521199909092119',30,p_flag
            call=conn.prepareCall(sql);
            call.setString(1, "lg1231");
            call.setString(2, "小白");
            call.setString(3, "441521199909092119");
            call.setInt(4, 30);
            call.registerOutParameter(5,java.sql.Types.NUMERIC);
            call.registerOutParameter(6, java.sql.Types.VARCHAR);
            call.execute();
            int flag = call.getInt(5);
            String message=call.getString(6);
            System.out.println(flag+" "+message);
        } catch (Exception e) {
        } finally {
            ConnectionUtils.close(conn, call, null);
        }
    }
```

* 能够掌握Oracle的存储函数

* Oracle存储函数概述

    * 函数的主要任务是在数据库层编写业务逻辑

* 存储过程和存储函数的区别

    * 返回值的区别,函数一定要有1个返回值或有多个通过输出参数的返回值,而存储过程是通过输出参数返回的,可以有多个或者没有。

    * 调用的区别,函数可以在sql语句中直接调用,而存储过程必须单独调用.

    * 函数一般情况下是用来计算并返回一个计算结果，而存储过程一般是用来完成特定的数据操作（比如修改、插入数据库表或执行某些DDL语句等等）

```
* 语法
 create or replace funciton 函数名 [(函数列表)]
    return 数据类型
  as
    定义变量;
  begin
    执行语句;
    return 返回的值;
  end;

* 测试
/*
 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;


/*
示例1：查询指定工号的员工年龄
定义存存函数 fun_get_age，参数:工号编号 e_no，类型%type
返回年龄result，number类型
*/
create or replace function fun_get_age(
    e_no employee.emp_no%type
)
    return number
as
```

```
    e_age employee.emp_age%type;
begin
    select emp_age into e_age from employee where emp_no=e_no;
    return e_age;
end;

/
--调用
 select fun_get_age('lg001') from dual;

/*
 示例2：查询返回指定工号的员工姓名及年龄，并计算和返回员工的平均年龄
 分析：定义函数fun_get_avg_age，入参：工号为e_no,输出参数信息：姓名：e_name，年龄：e
 返回值：平均年龄：result number(4,1)
*/
create or replace function fun_get_avg_age(
       e_no employee.emp_no%type,
       e_name out employee.emp_name%type,
       e_age out employee.emp_age%type
)
       return number
as
       avg_age number(4,1);
begin
     select emp_name,emp_age into e_name,e_age from employee where emp_no=e_no;
     select avg(emp_age) into avg_age from employee;
     return avg_age;
end;
/

--调用
declare
   e_no employee.emp_no%type:='lg001';
   e_name  employee.emp_name%type;
   e_age  employee.emp_age%type;
   avg_age number;
begin
  avg_age:=fun_get_avg_age(e_no,e_name,e_age);
  dbms_output.put_line('姓名：'||e_name||'，年龄:'||e_age||'，平均年龄:'||avg_age
end;
```

```
结果：
姓名：张大，年龄:19，平均年龄:18.5
```

* 能够掌握使用JDBC调用Oracle的存储函数

```
关键语法：
 * CallableStatement call=conn.prepareCall("{?=call 存储函数(?)}");
@Test
    public void test2() {
        Connection conn = null;
        CallableStatement call = null;
        try {
            conn = ConnectionUtils.getConnection();
            String sql="{?=call fun_get_age(?)}";
            call=conn.prepareCall(sql);
            call.registerOutParameter(1, oracle.jdbc.OracleTypes.NUMBER);
            call.setString(2, "lg001");
            call.execute();
            int empAge=call.getInt(1);
            System.out.println(empAge);
        } catch (Exception e) {
        } finally {
            ConnectionUtils.close(conn, call, null);
        }
    }
* 结果：19
@Test
    public void test3() {
        Connection conn = null;
        CallableStatement call = null;
        try {
            conn = ConnectionUtils.getConnection();
            String sql="{?=call fun_get_avg_age(?,?,?)}";
            call=conn.prepareCall(sql);
            call.registerOutParameter(1, oracle.jdbc.OracleTypes.NUMBER);
```

```
            call.setString(2, "lg001");
            call.registerOutParameter(3, oracle.jdbc.OracleTypes.VARCHAR);
            call.registerOutParameter(4, oracle.jdbc.OracleTypes.NUMBER);
            call.execute();
            double avgAge=call.getDouble(1);
            String name=call.getString(3);
            int age=call.getInt(4);
            System.out.println("工号:"+"lg001"+",姓名:"+name+",年龄:"+age+",平均
        } catch (Exception e) {
        } finally {
            ConnectionUtils.close(conn, call, null);
        }
    }
结果:
 工号:lg001,姓名:张大,年龄:19,平均年龄:18.5
```

* 能够掌握Oracle的程序包对象

  * Oracle程序包概述

    * 程序包其实本质就是一个封装的过程，主要用于封装存储过程。

```
* 语法
创建包:
  CREATE OR REPLACE
   PACKAGE 包名 AS
    变量声明或声明包中存储过程(procedure 存储过程名[(参数列表)];
   END 包名;
创建包体:
CREATE OR REPLACE
 PACKAGE BODY 包名 AS
    procedure 存储过程名[(参数列表)]  AS
    BEGIN
      -- 开始业务，存储过程业务处理
      sql等;
    END 存储过程名;
 END 包名;

新建包及包体
新建程序包，实现添加员工逻辑，并进行测试
```

```
/*
准备工作，新建表，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
insert into employee(emp_no,emp_name,emp_id,emp_age) values('lg005','张5','4415
commit;
select * from employee;

/*
 建立程序包，并定义添加员工信息的存储过程（只声明，不用实现）：pro_add_employee，输入
 输出参数为 e_flag out number,表示操作编号；e_message out varchar2表示详细的操作结
*/
create or replace package pac1
as
procedure pro_add_employee(
    e_no employee.emp_no%type,
    e_name employee.emp_name%type,
    e_id employee.emp_id%type,
    e_age employee.emp_age%type,
    e_flag out number,
    e_message out varchar2
);
end pac1;

/*
新建包体，并编写存储过程的业务，往数据表中添加一条员工记录，并把操作结果信息存放于输出
*/
create or replace package body pac1
as
```

```
 procedure pro_add_employee(
    e_no employee.emp_no%type,
    e_name employee.emp_name%type,
    e_id employee.emp_id%type,
    e_age employee.emp_age%type,
    e_flag out number,
    e_message out varchar2
 )
is
begin
      insert into employee(emp_no,emp_name,emp_id,emp_age) values(e_no,e_name,e
      commit;
      e_flag:=1;
      e_message:='保存成功！';
      EXCEPTION
        when DUP_VAL_ON_INDEX then
           e_flag:=-20001;
           e_message:='工号已经存在，请核查！';
        when others then
           e_flag:=sqlcode;
           e_message:=sqlerrm;
end pro_add_employee;
end pac1;

--调用存储过程
declare
   e_flag number(3,0);
   e_message varchar2(50);
begin
  pac1.pro_add_employee('a018','张17','111521199909092116',18,e_flag,e_message)
  dbms_output.put_line(e_flag||':'||e_message);
end;

* 包中定义多个存储过程
 新建包及包体
--定义包并声明添加员工和根据员工编号查询员工年龄的两个存储过程
create or replace package pac3
as
 procedure pro_add_employee(
     e_no employee.emp_no%type,
```

```
99      e_name employee.emp_name%type,
100     e_id employee.emp_id%type,
101     e_age employee.emp_age%type,
102     e_flag out number,
103     e_message out varchar2
104  );
105  procedure get_emp_age(
106     e_no employee.emp_no%type,
107     e_age out employee.emp_age%type
108    );
109 end pac3;
110 --新建包体，并实现两个存储过程的业务
111 create or replace package body pac3
112 as
113  procedure pro_add_employee(
114     e_no employee.emp_no%type,
115     e_name employee.emp_name%type,
116     e_id employee.emp_id%type,
117     e_age employee.emp_age%type,
118     e_flag out number,
119     e_message out varchar2
120  )
121  is
122  begin
123     insert into employee(emp_no,emp_name,emp_id,emp_age) values(e_no,e_name,e_i
124     commit;
125     e_flag:=1;
126     e_message:='添加成功！';
127     EXCEPTION
128       when others then
129         e_flag:=-20004;
130         e_message:='添加失败！';
131  end pro_add_employee;
132
133  procedure get_emp_age(
134     e_no employee.emp_no%type,
135     e_age out employee.emp_age%type
136    )
137  is
138
```

```
 begin
    select emp_age into e_age from employee
    where emp_no=e_no;
 end get_emp_age;
end pac3;

--调用包中的存储过程，添加员工信息
declare
s_flag1 number(10,2);
s_message1 varchar2(30);
begin
  pac3.pro_add_employee('aa05','aaaa333','5332432432343',21,s_flag1,s_message1)
  dbms_output.put_line(s_flag1||'||'||s_message1);
end;

--调用包中的存储过程，查询员工年龄
declare
s_age number(3);
begin
  pac3.get_emp_age('lg004',s_age);
  dbms_output.put_line(s_age);
end;
```

* 能够掌握使用JDBC调用程序包对象

```
 * 关键语法
    * CallableStatement call=conn.prepareCall("{call 包名.存储过程名(?,?,?,?,?,?
@Test
    public void test4() {
        Connection conn = null;
        CallableStatement call = null;
        try {
            conn = ConnectionUtils.getConnection();
            String sql="{call pac3.get_emp_age(?,?)}";
            call=conn.prepareCall(sql);
            call.setString(1, "lg001");
            call.registerOutParameter(2, oracle.jdbc.OracleTypes.NUMBER);
```

```
            call.execute();
            int empAge=call.getInt(2);
            System.out.println("lg001:"+empAge);
        } catch (Exception e) {
        } finally {
            ConnectionUtils.close(conn, call, null);
        }
    }
 * 结果：
 lg001:19
```

* 能够掌握Oracle的触发器

  * 触发器概述

  * 触发器是在事件发生时隐式地自动运行的PL/SQL程序块，不能接收参数，不能被调用。，一般监听的事件是数据的增删改事件。

```
语法：
CREATE [OR REPLACE] TRIGGER  trigger_name
    {BEFORE|AFTER}        ——触发时机
    {INSERT|DELETE|UPDATE
    ON{table_name }
    [FOR EACH ROW]
    trigger_body;
备注：trigger _body就是pl/sql块

* 测试
 * 备份删除的数据
 * 删除员工信息时，同时把要删除员工信息备份到另一张表中，以防止误操作时数据丢失或便于以
--  1.准备工作：创建员工表，插入测试数据
--drop table employee;
--drop table emp_del_rec;
CREATE TABLE employee
(
     emp_no number(4,0) PRIMARY KEY,   --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
);
CREATE TABLE emp_del_rec
(
    emp_no number(4,0)  NOT NULL,   --工号，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
    emp_operator_date date--操作日期
);
insert into employee(emp_no,emp_name,emp_id,emp_age) values(1001,'张大','441521
insert into employee(emp_no,emp_name,emp_id,emp_age) values(1002,'张二','441521
insert into employee(emp_no,emp_name,emp_id,emp_age) values(1003,'张三','441521
insert into employee(emp_no,emp_name,emp_id,emp_age) values(1004,'张四','441521
commit;

--创建触发器，删除数据时可以把删除的数据备份到其它表中
create or replace trigger tri_emp_del_rec
   before delete on employee for each row
 declare
 begin
       --:old.列名，可以引用在增删改前原表字段数据，没有数据为空;
       --:new.列名，可以引用在增删改后字段新值，处理后对应记录不存在，则数据为空
       insert into emp_del_rec(emp_no,emp_name,emp_id,emp_age,emp_operator_date
 end;
/

--测试触发器,删除数据及观察结果
delete from employee where emp_no=1003;
select * from emp_del_rec;
select * from employee;

* 表级触发器，删除时整表备份
 * 表级触发器里不能再用:old或:new。
/*
 创建员工表，插入测试数据
*/
--drop table employee;
--drop table emp_del_rec;
```

```
CREATE TABLE employee
(
     emp_no number(4,0) 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
);
CREATE TABLE emp_del_rec
(
     emp_no number(4,0)  NOT NULL,   --工号，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
     emp_operator_date date default sysdate --操作日期
);
insert into employee(emp_no,emp_name,emp_id,emp_age) values(1001,'张大','441521
insert into employee(emp_no,emp_name,emp_id,emp_age) values(1002,'张二','441521
insert into employee(emp_no,emp_name,emp_id,emp_age) values(1003,'张三','441521
insert into employee(emp_no,emp_name,emp_id,emp_age) values(1004,'张四','441521
commit;

--创建触发器，删除数据时可以把删除的数据备份到其它表中,要使用before,不能用after
create or replace trigger tri_emp_del_rec
   before delete on employee
 declare
 begin
       Insert into emp_del_rec(emp_no,emp_name,emp_id,emp_age) select emp_no,em
end;
/

--测试触发器,删除数据及观察结果
delete from employee;
select * from emp_del_rec;
select * from employee;

 * 实现主键自动增长
--通过触发器+序列完自动增长
--准备工作，继续使用上面的员工表
--1.创建序列
create  sequence seq_emp_no;
```

```
--2.创建触发器
create or replace trigger tri_gen_pk
 before insert on employee for each row
begin
select seq_emp_no.nextval into :new.emp_no from dual;
end;
/
--测试触发器
insert into employee(emp_name,emp_id,emp_age) values('张三','441521199909092113
insert into employee(emp_name,emp_id,emp_age) values('张四','441521199909092114
select * from employee;

```

* 能够掌握Oracle的视图

  * 视图的概述

    * 视图是从一个或几个基本表（或视图）中导出的虚拟的表。

    * 视图的定义存在数据库中，与此定义相关的数据并没有再存一份于数据库中。

    * 通过视图看到的数据存放在基表中。

    * 视图看上去非常像数据库的物理表，对它的操作同任何其它的表一样。

    * 当通过视图修改数据时，实际上是在改变基表中的数据；相反地，基表数据的改变也会自动反映在由基表产生的视图中。

    * 单表视图一般用于查询和修改，会改变基本表的数据；多表视图一般用于查询，不会改变基本表的数据。

  * 视图的特性

    * 安全性

      * 用户只能查询和修改能看到的数据

      * 基表中重要的字段信息，可以不通过视图给用户

      * 用户对视图不可以随意的更改和删除，可以保证数据的安全性

    * 简单性

      * 隐藏数据复杂性

      * 使用多个表中相关列或行的集合定义单个视图（表与表的链接的结果集定义成视

图）

* 隔离性

* 将应用程序与基表（表结构定义或者对表结构修改）进行隔离

* 如果视图的定义查询引用为四列表的三列，并且向表中添加了第五列，则视图的定义不受影响，并且使用该视图的所有应用程序都不受影响。

* 角度性

* 与基表不同的角度显示数据

* 可以重命名视图的列，而不影响视图所基于的表。

```
* 语法
 create view 视图名 as 查询语句;
 drop view 视图名;
* 测试
--drop table employee;
--drop table my_class;
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
     mc_no CHAR(4)  --班级编号
);
CREATE TABLE my_class
(
     mc_no CHAR(5) PRIMARY KEY NOT NULL,   --编号，主键，非空
     mc_name VARCHAR2(30) NOT NULL--名称,非空
);
insert into my_class(mc_no,mc_name) values('c001','软件一班');
insert into my_class(mc_no,mc_name) values('c002','软件二班');
insert into employee(emp_no,emp_name,mc_no) values('lg001','张一','c001');
insert into employee(emp_no,emp_name,mc_no) values('lg002','张二','c001');
insert into employee(emp_no,emp_name,mc_no) values('lg003','张三','c002');
commit;

* 安全性
  * 用户只能查询和修改能看到的数据
  * 基表中重要的字段信息，可以不通过视图给用户
```

```
  * 用户对视图不可以随意的更改和删除，可以保证数据的安全性
--直接创建视图没有权限，会报错：ORA-01031: insufficient privileges
--登陆system用户，给用户scott授创建视图的权限
grant create view to scott;
--登陆scott用户，创建视图
create or replace view view_employee
as
select emp_no,emp_name from employee ;
--通过视图查询数据
select * from view_employee;

* 简单性
  * 隐藏数据复杂性
  * 使用多个表中相关列或行的集合定义单个视图（表与表的链接的结果集定义成视图）
--基于多个基表的视图，不建议使用视图进行增删改操作
create or replace view view_my_class_employee
as
select mc.mc_name,s.emp_no,s.emp_name from employee s inner join my_class mc or

--查询多个基表的视图
select * from view_my_class_employee;

* 隔离性
  * 将应用程序与基表（表结构定义或者对表结构修改）进行隔离
  * 如果视图的定义查询引用为四列表的三列，并且向表中添加了第五列，
    则视图的定义不受影响，并且使用该视图的所有应用程序都不受影响。
-- 创建视图
create or replace view view_employee
as
select emp_no,emp_name from employee;
-- 查询视图
select * from view_employee;
-- java程序查询视图
    @Test
    public void test6() {
        Connection conn = null;
        PreparedStatement st =null;
        ResultSet rs=null;
        try {
            conn=ConnectionUtils.getConnection();
```

```
            String sql="select * from view_employee";
            st = conn.prepareStatement(sql);
            rs=st.executeQuery();
            while(rs.next()) {
                String empNo=rs.getString(1);// 数组下标是从开始，JDBC不从零开始,
                String eName=rs.getString(2);
                System.out.println(empNo+":"+eName);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }finally {
            ConnectionUtils.close(conn, st, rs);
        }
    }
结果：
lg001   :张一
lg002   :张二
lg003   :张三
-- 更改表，然后继续查询
 alter table employee add (emp_address VARCHAR2(20));
-- 查询没有影响
 select * from view_employee;

* 角度性
  * 与基表不同的角度显示数据
  * 可以重命名视图的列，而不影响视图所基于的表。
-- 创建视图，查询的时候列取别名
create or replace view view_emp
as
select emp_no e_no,emp_name e_name from employee;
-- 查询视图
select emp_no,emp_name from employee;
select e_no,e_name from view_emp;

* 创建基于视图的视图
create or replace view view_emp1
as
select e_no,e_name from view_emp;

--查询基于视图的视图
```

```
select * from view_emp1;

* dml视图
--通过视图添加数据,需要保证基表的其它数据项可以为空
insert into view_employee(emp_no,emp_name) values('lg007','张七');
--通过视图修改数据
update view_employee set emp_name='张2' where emp_no='lg007'
--通过视图删除数据
delete from view_employee  where emp_no='lg007'

* 删除视图
drop view view_emp;
drop view view_my_class_employee;
drop view view_employee;

```

* 能够掌握Oracle的闪回，备份，恢复

* Oracle闪回概述

* Oracle可以在删除之后进行数据和表对象的闪回，任何闪回技术和恢复技术都是基于系统的某一个时间点的，因 Oracle的恢复和闪回技术也是一样，

* SCN： System Change Number 和 系统的时间值一一对应,SCN可以作为恢复的时间点。

```
* 假如已经delete 数据并且已经commit，如何闪回数据??
-- 查看当前时刻的SCN号?
 select to_char(sysdate,'yyyy-mm-dd hh24:mi:ss'), timestamp_to_scn(sysdate) fro
-- 使用管理员system,用dba角色查看undo表空间的参数(在sqlplus运行语句)
* 运行结果:
SQL> show parameter undo
NAME                                 TYPE        VALUE
------------------------------------ -----------
undo_management                      string      AUTO
undo_retention                       integer     900
undo_tablespace                      string      UNDOTBS1
默认的情况下有效的闪回时间为900(15分钟)（单位是秒） ,但是管理员可以通过设置语句来修改
 -- 修改当前系统的undo表空间的参数（可以不修改）
```

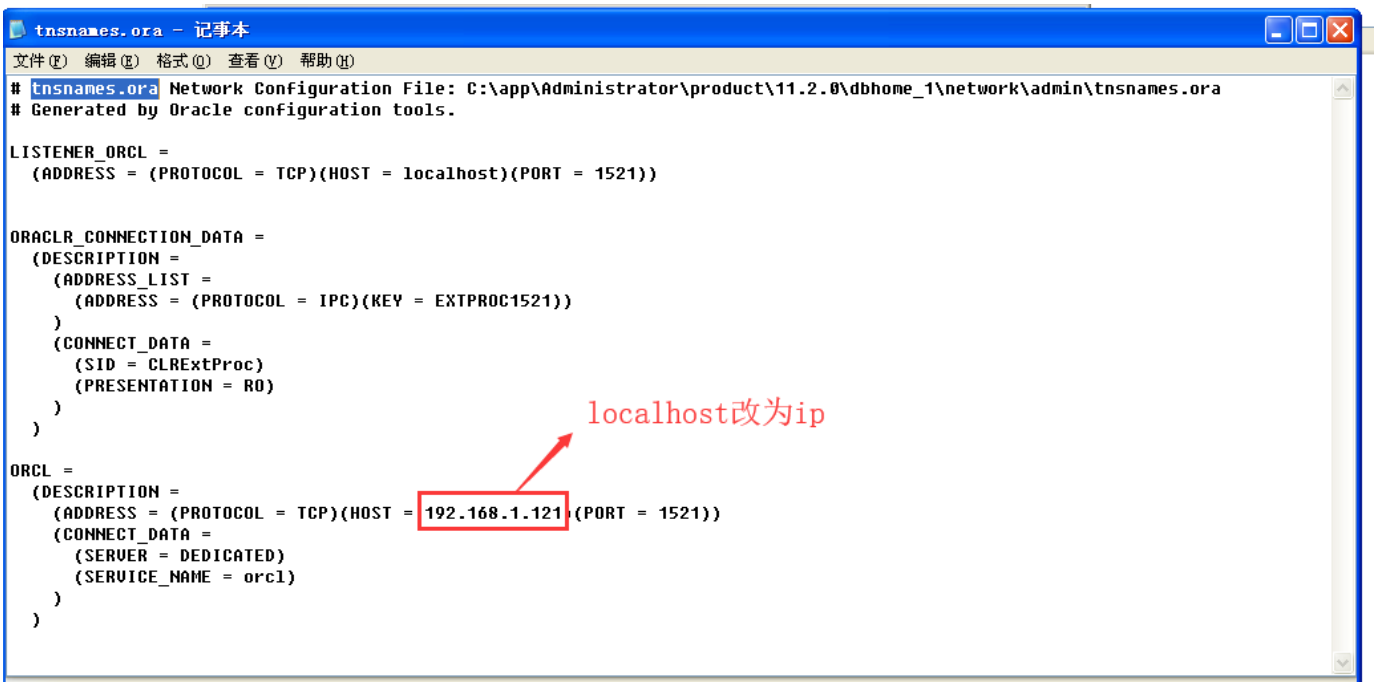
```
 alter system set undo_retention=1800 scope=both;

* 闪回删除的表数据
/*
scott用户
创建员工表，及测试数据
*/
--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;
--查看并记录当前的ＳＣＮ号
select to_char(sysdate,'yyyy-mm-dd hh24:mi:ss'), timestamp_to_scn(sysdate) from
--错误删除数据
delete employee where emp_no='lg003';
commit;
--查看表记录，emp_no为lg003数据已被删除
select * from employee;

--闪回刚才删除数据，需要先修改表的列可以移动：  登陆system用户：alter table scott.emp
flashback table employee to scn 2765382;
--查看数据，发现已被恢复
 select * from employee;
```

* Oracle数据库备份

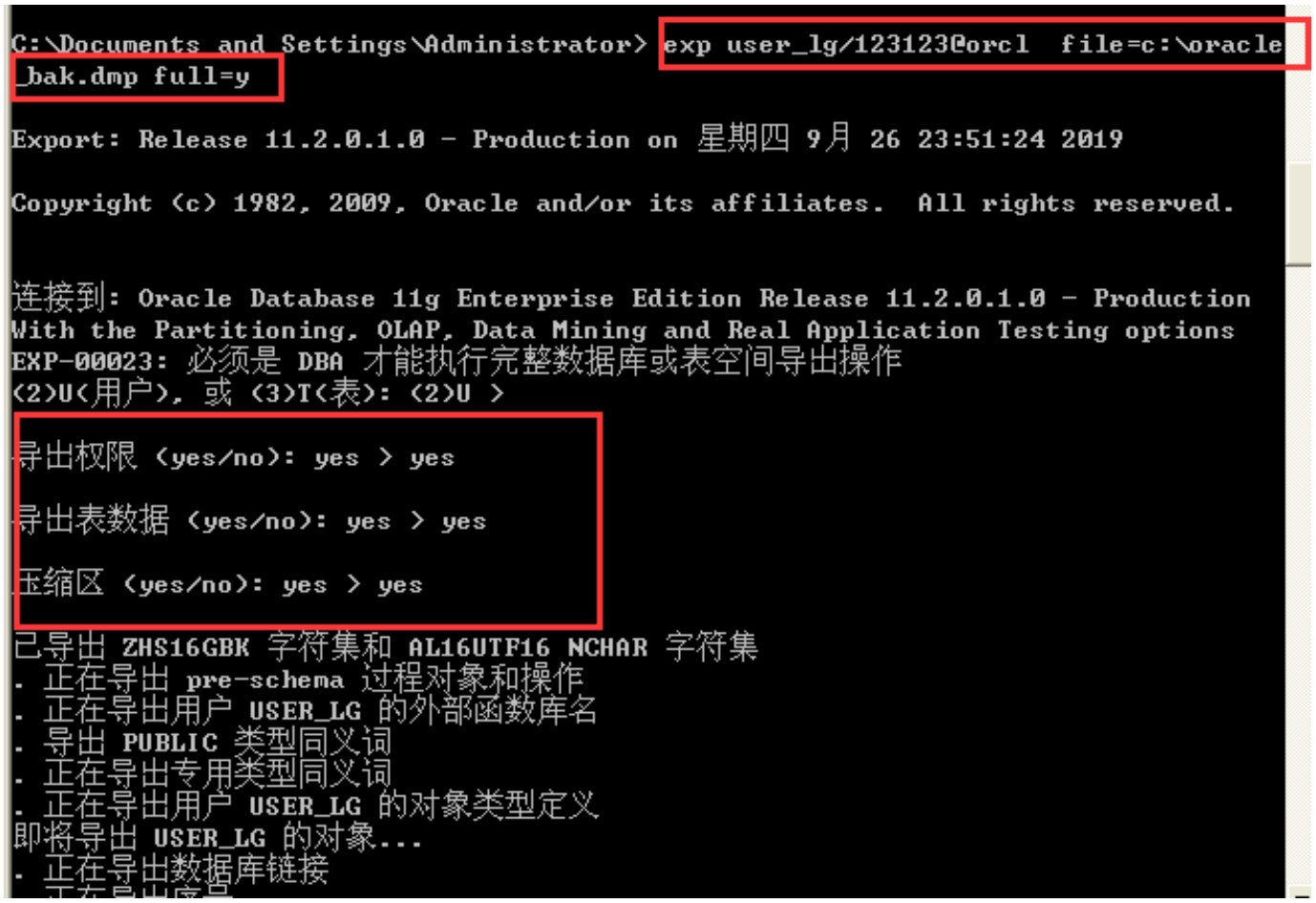
```
* 数据备份
 * 创建表空间
 CREATE TABLESPACE lgdata168
```

```
 DATAFILE 'c:\oracle_data\lgdata168.DBF'
 SIZE 10M AUTOEXTEND ON;
 * 创建用户
 CREATE USER  user_lg
 IDENTIFIED BY 123123
 DEFAULT TABLESPACE lgdata168
 TEMPORARY TABLESPACE TEMP;
 * 授权
 grant connect, resource TO user_lg;
 grant create view to user_lg;
 * 登录user_lg用户操作
 --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('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;
select * from employee;
* 备份
  * 在服务端进行操作
  * 修改tnsnames.ora文件：localhost改为192.168.1.121
  * 在cmd执行命令
    完整备份
    exp user_lg/123123@orcl  file=c:\oracle_bak.dmp full=y
    指定表备份
    exp user_lg/123123@orcl  file=c:\oracle_employee_bak.dmp tables=(employee)
```

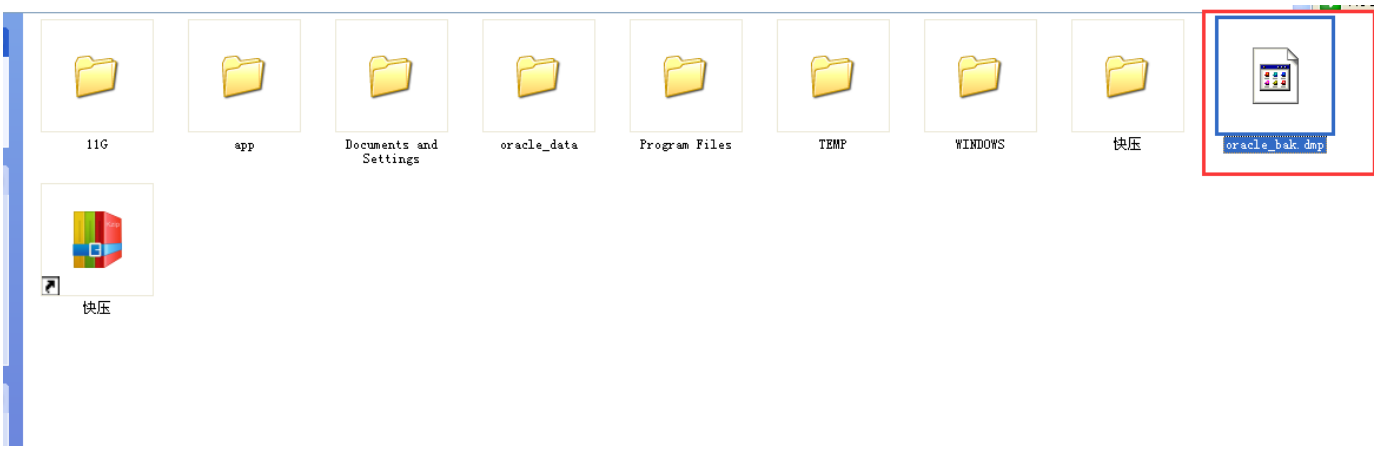
* 修改tnsnames.ora文件：localhost改为192.168.1.12



* 在cmd命令执行



* 备份好的问题

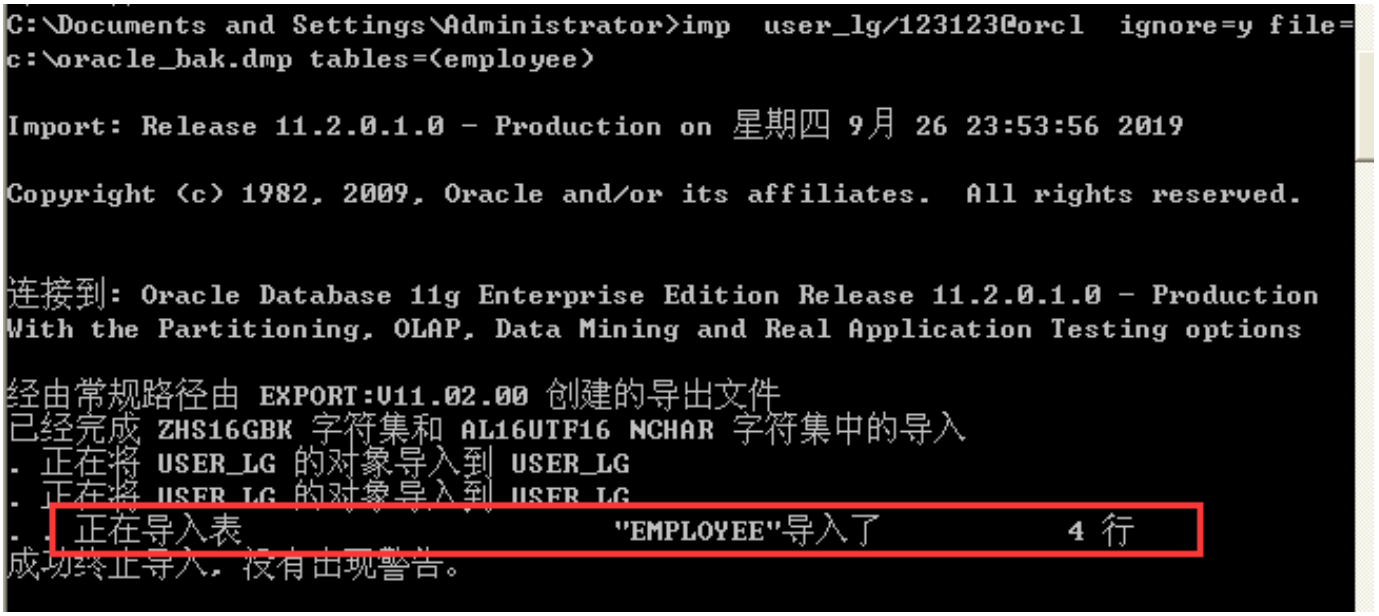


* Oracle数据库恢复

```
* 删除表恢复
 * 导出后可以删除表，再恢复
   * drop table employee
 * 指定表恢复
  * imp  user_lg/123123@orcl  ignore=y file=c:\oracle_bak.dmp tables=(employee)

* 删除表空间和用户后进行恢复
  * 查询用户表空间的信息
    * select username,default_tablespace,temporary_tablespace from dba_users wh
  * 删除表空间和用户
    * drop user user_lg cascade;
    * drop tablespace lgdata168 including contents and datafiles;
  * 创建表空间
    CREATE TABLESPACE lgdata168
    DATAFILE 'c:\oracle_data\lgdata168.DBF'
    SIZE 10M AUTOEXTEND ON;
  * 创建用户
    CREATE USER  user_lg
    IDENTIFIED BY 123123
    DEFAULT TABLESPACE lgdata168
    TEMPORARY TABLESPACE TEMP;
  * 授权
    grant connect, resource TO user_lg;
    grant create view to user_lg;
  * 恢复
```

```
26      * 在服务器cmd上执行
27     imp user_lg/123123@orcl  ignore=y file=c:\oracle_bak.dmp  full=y
```

* 删除表恢复成功截图



* 删除表空间和用户后进行恢复成功截图