

**期末项目设计报告**

|  |  |  |  |
| --- | --- | --- | --- |
| 题 目 | 基于Oracle的影院售票系统的数据库设计 | | |
| 课程 | Oracle数据库应用 | | |
| 学 院 | 计算机学院 | | |
| 专 业 | 软件工程 | 年级 | 2018级 |
| 学生姓名 | 高李美 | 学号 | 201810424301 |
| 指导教师 | 赵卫东 | 职称 | 副教授 |

|  |  |  |  |
| --- | --- | --- | --- |
| **评分项** | **评分标准** | **满分** | **得分** |
| 文档整体 | 文档内容详实、规范，美观大方 | 10 |  |
| 表设计 | 表，表空间设计合理，数据合理 | 20 |  |
| 用户管理 | 权限及用户分配方案设计正确 | 10 |  |
| PL/SQL设计 | 存储过程和函数设计正确 | 25 |  |
| 备份方案 | 备份方案设计正确 | 25 |  |
| 容灾方案 | DataGuard设计正确 | 10 |  |
| **得分合计** | | |  |

2021 年 6 月 15 日

# 一、概要设计

## 1.1 绪论

随着现代社会的发展，人们的生活水平逐渐提高，人们更加注重精神粮食的富足，面对时代潮流，电影院行业在逐渐壮大，面对海量的数据，电影院要花费大量的人力和财力来进行存储和维护，业务具有数据海量化的特点。由于业务数据不断增长带来的压力，决定采用oracle数据库系统来完成电影院售票系统的数据库系统设计。

## **1.2 数据库部署模式**

越来越多的人选择去电影院度过愉快时光、庞大观影数据为电影院带来巨大利润的同时,也带来了信息系统风险的相对集中，这使得电影院信息系统连续运行的要求也越来越高。加强信息系统灾备体系建设，保障业务连续运行，已经成为影响影院竞争能力的一个重要因素。对RTO=0/RPO=0的系统决定数据库采用RAC+DataDataGuard模式。根据RAC+DataDataGuard模式的特点，有如下要求: .1 主机与备机在物理.上要分开。为了实现容灾的特性，需要在物理。上分割主机和备机。. .2 进行合理 的设计，充分实现DATAGUARD的功能。 注: RTO ( RecoveryTime object): 恢复时间目标，灾难发生后信息系统从停顿到必须恢复的时间要求。 RPO (Recovery Point Object): 恢复点目标，指一个过去的时间点，当灾难或紧急事件发生时，数据可以恢复到的时间点。

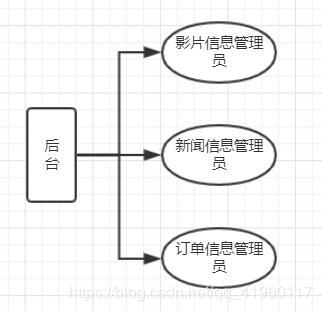
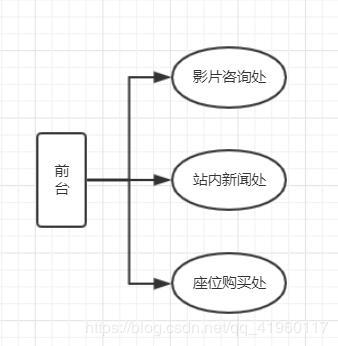
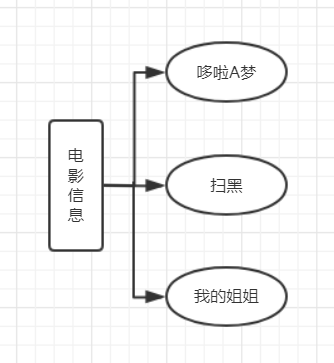
## **1.3 Oracle数据库**

Oracle Database，又名Oracle RDBMS，或简称Oracle。是甲骨文公司的一款关系数据库管理系统。它是在数据库领域一直处于领先地位的产品。可以说Oracle数据库系统是目前世界上流行的关系数据库管理系统，系统可移植性好、使用方便、功能强，适用于各类大、中、小、微机环境。它是一种高效率、可靠性好的、适应高吞吐量的数据库方案。

## **1.4 项目概述**

本项目是基于Oracle的电影院系统的数据库设计，电影院包括前台与后台两个主要部分，前台是影片咨询处、站内新闻处、座位购买处，后台是影片信息管理员、新闻信息管理员、订单信息管理员，这些部门信息部分及订单信息（本数据库主要插入了关于“扫黑”、“哆啦A梦”、“我的姐姐”三部电影的影片信息）采用了相应的oracle数据库表设计。

## **1.5 ER模型图**



# 二、详细设计

本数据库设计创建了两个角色：glm1和glm2

本数据库设计创建了两个用户：lm1和lm2

## 2.1 创建两类用户、两个角色，设计权限及用户分配方案。

1）以system登录到pdborcl，创建角色glm1、glm2和用户lm1、lm2，并授权和分配空间：

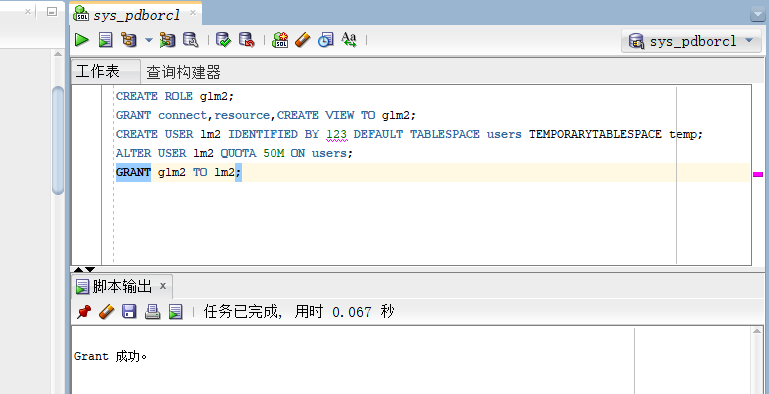
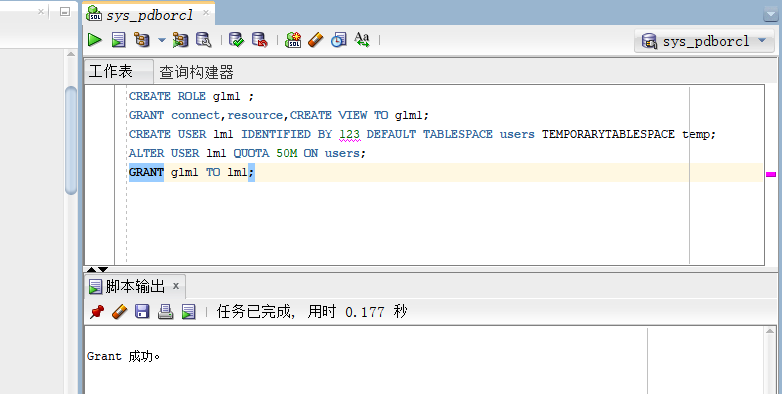
> CREATE ROLE glm1 ;

> GRANT connect,resource,CREATE VIEW TO glm1;

> CREATE USER lm1 IDENTIFIED BY 123 DEFAULT TABLESPACE users TEMPORARYTABLESPACE temp;

> ALTER USER lm1 QUOTA 50M ON users;

> GRANT glm1 TO lm1;



2）新用户lm1连接到pdborcl，创建表mytable和视图myview，插入数据，最后将myview的SELECT对象权限授予hr用户和lm1用户。

> CREATE TABLE mytable (id number,name varchar(50));

> INSERT INTO mytable(id,name)VALUES(1,'lili');

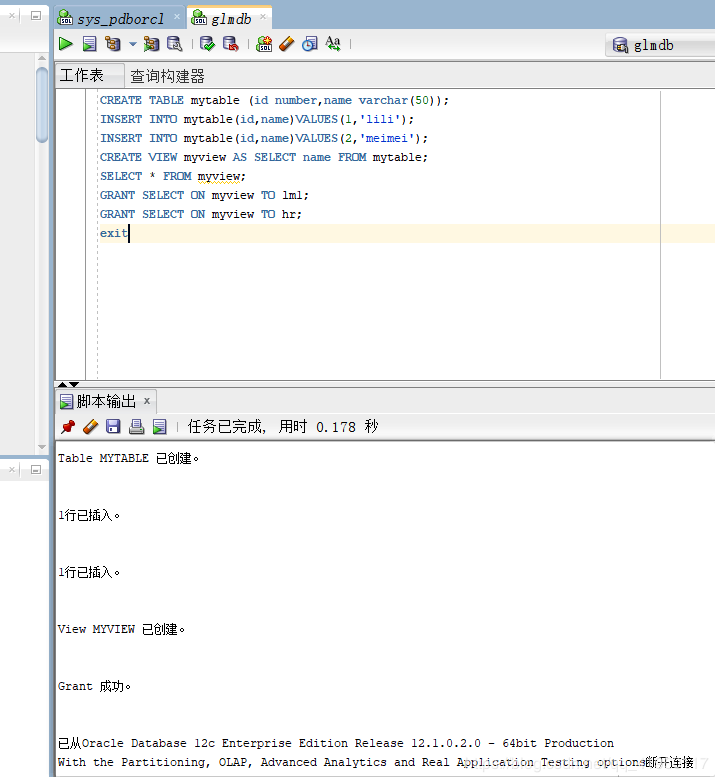
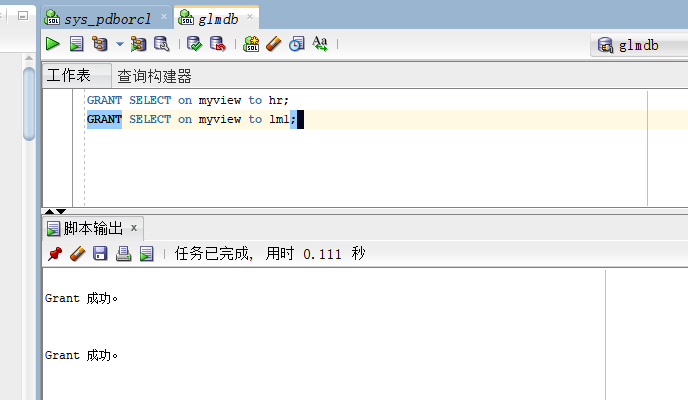
> INSERT INTO mytable(id,name)VALUES(2,'meimei');

> CREATE VIEW myview AS SELECT name FROM mytable;

> SELECT \* FROM myview;

> GRANT SELECT ON myview TO lm1;

> GRANT SELECT ON myview TO hr;

3）数据库和表空间占用分析:当我们的实验做完后，数据库pdborcl中包含了不同的角色和用户。 所有用户都使用表空间users存储表的数据。表空间中存储了很多相同名称的表mytable和视图myview，但分别属性于不同的用户，不会引起混淆。随着用户往表中插入数据，表空间的磁盘使用量会增加。

### **2.1.1 以 system身份登录进行查看。**

$ sqlplus system/123@pdborcl

SQL>SELECT tablespace\_name,FILE\_NAME,BYTES/1024/1024 MB,MAXBYTES/1024/1024 MAX\_MB,autoextensible FROM dba\_data\_files WHERE tablespace\_name='USERS';

SQL>SELECT a.tablespace\_name "表空间名",Total/1024/1024 "大小MB",

free/1024/1024 "剩余MB",( total - free )/1024/1024 "使用MB",

Round(( total - free )/ total,4)\* 100 "使用率%"

from (SELECT tablespace\_name,Sum(bytes)free

FROM dba\_free\_space group BY tablespace\_name)a,

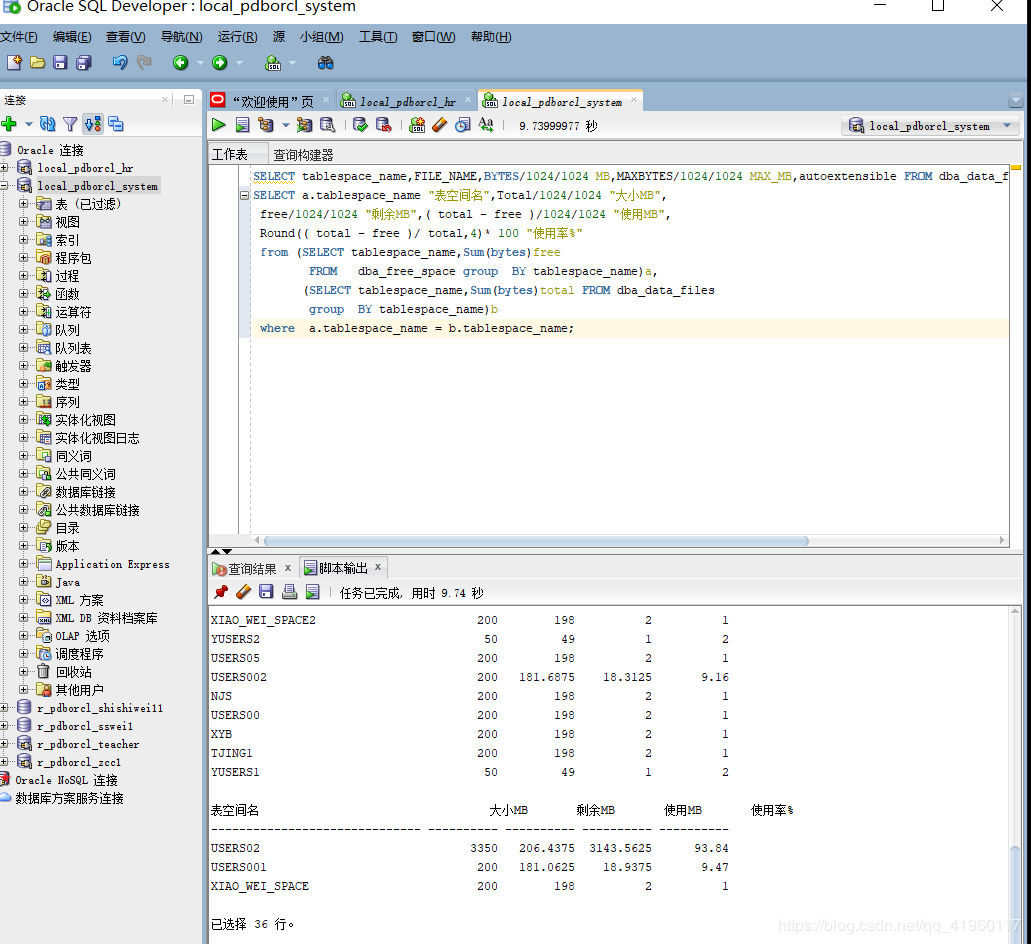
(SELECT tablespace\_name,Sum(bytes)total FROM dba\_data\_files

group BY tablespace\_name)b

where a.tablespace\_name = b.tablespace\_name;

autoextensible是显示表空间中的数据文件是否自动增加。

MAX\_MB是指数据文件的最大容量。

* 

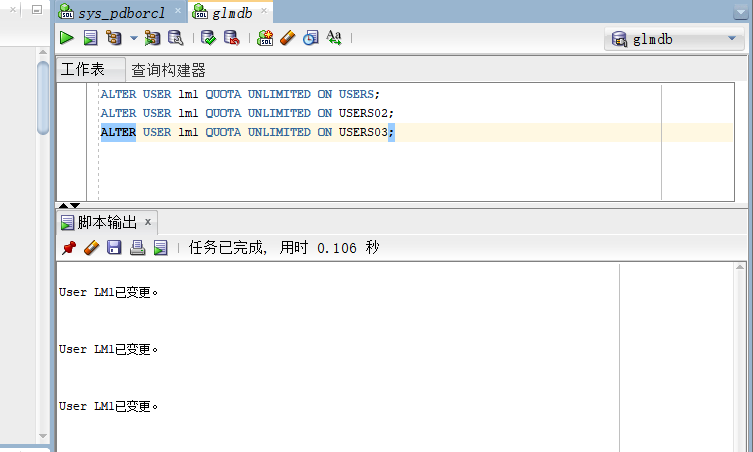
## **2.2 创建表以及插入数据、分配表空间**

1）以system登录到pdborcl，让用户glm1使用3个表空间：USERS,USERS02,USERS03。在表空间中创建两张表：订单表(orders)与订单详情表(order\_details)。

> ALTER USER lm1 QUOTA UNLIMITED ON USERS;

> ALTER USER lm1 QUOTA UNLIMITED ON USERS02;

> ALTER USER lm1 QUOTA UNLIMITED ON USERS03;



使用perfectism账号创建本实验的表，表创建在上述3个分区，自定义分区策略。

2）在lm1用户下先测试即将要创建的表，若有就删除该表。

declare

num number;

begin

select count(1) into num from user\_tables where TABLE\_NAME = 'ORDER\_DETAILS';

if num=1 then

execute immediate 'drop table ORDER\_DETAILS cascade constraints PURGE';

end if;

select count(1) into num from user\_tables where TABLE\_NAME = 'ORDERS';

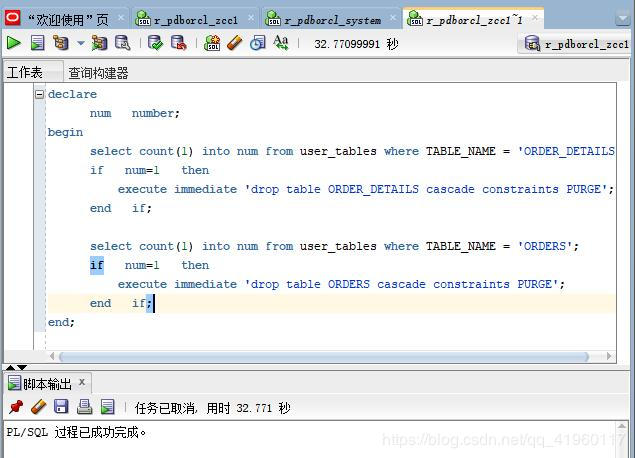
if num=1 then

execute immediate 'drop table ORDERS cascade constraints PURGE';

end if;

end;

如上脚本在用户lm1权限下对即将创建的表进行查询验证，如果在user\_tables里面找到了即将插入'ORDER\_DETAILS'表和'ORDERS'表，那就对其彻底删除，PURGE是清空回收站。



3）在用户lm2创建'ORDERS'表，并创建了7个以时间为条件的分区。

CREATE TABLE ORDERS

(

ORDER\_ID NUMBER(10, 0) NOT NULL

, CUSTOMER\_NAME VARCHAR2(40 BYTE) NOT NULL

, CUSTOMER\_TEL VARCHAR2(40 BYTE) NOT NULL

, ORDER\_DATE DATE NOT NULL

, EMPLOYEE\_ID NUMBER(6, 0) NOT NULL

, DISCOUNT NUMBER(8, 2) DEFAULT 0

, TRADE\_RECEIVABLE NUMBER(8, 2) DEFAULT 0

, CONSTRAINT ORDERS\_PK PRIMARY KEY

(

ORDER\_ID

)

USING INDEX

(

CREATE UNIQUE INDEX ORDERS\_PK ON ORDERS (ORDER\_ID ASC)

LOGGING

TABLESPACE USERS

PCTFREE 10

INITRANS 2

STORAGE

(

BUFFER\_POOL DEFAULT

)

NOPARALLEL

)

ENABLE

)

TABLESPACE USERS

PCTFREE 10

INITRANS 1

STORAGE

(

BUFFER\_POOL DEFAULT

)

NOCOMPRESS

NOPARALLEL

PARTITION BY RANGE (ORDER\_DATE)

(

PARTITION PARTITION\_2015 VALUES LESS THAN (TO\_DATE(' 2016-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS\_CALENDAR=GREGORIAN'))

NOLOGGING

TABLESPACE USERS

PCTFREE 10

INITRANS 1

STORAGE

(

INITIAL 8388608

NEXT 1048576

MINEXTENTS 1

MAXEXTENTS UNLIMITED

BUFFER\_POOL DEFAULT

)

NOCOMPRESS NO INMEMORY

, PARTITION PARTITION\_2016 VALUES LESS THAN (TO\_DATE(' 2017-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS\_CALENDAR=GREGORIAN'))

NOLOGGING

TABLESPACE USERS

PCTFREE 10

INITRANS 1

STORAGE

(

BUFFER\_POOL DEFAULT

)

NOCOMPRESS NO INMEMORY

, PARTITION PARTITION\_2017 VALUES LESS THAN (TO\_DATE(' 2018-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS\_CALENDAR=GREGORIAN'))

NOLOGGING

TABLESPACE USERS

PCTFREE 10

INITRANS 1

STORAGE

(

BUFFER\_POOL DEFAULT

)

NOCOMPRESS NO INMEMORY

, PARTITION PARTITION\_2018 VALUES LESS THAN (TO\_DATE(' 2019-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS\_CALENDAR=GREGORIAN'))

NOLOGGING

TABLESPACE USERS02

PCTFREE 10

INITRANS 1

STORAGE

(

BUFFER\_POOL DEFAULT

)

NOCOMPRESS NO INMEMORY

, PARTITION PARTITION\_2019 VALUES LESS THAN (TO\_DATE(' 2020-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS\_CALENDAR=GREGORIAN'))

NOLOGGING

TABLESPACE USERS02

PCTFREE 10

INITRANS 1

STORAGE

(

BUFFER\_POOL DEFAULT

)

NOCOMPRESS NO INMEMORY

, PARTITION PARTITION\_2020 VALUES LESS THAN (TO\_DATE(' 2021-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS\_CALENDAR=GREGORIAN'))

NOLOGGING

TABLESPACE USERS02

PCTFREE 10

INITRANS 1

STORAGE

(

BUFFER\_POOL DEFAULT

)

NOCOMPRESS NO INMEMORY

, PARTITION PARTITION\_2021 VALUES LESS THAN (TO\_DATE(' 2022-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS\_CALENDAR=GREGORIAN'))

NOLOGGING

TABLESPACE USERS03

PCTFREE 10

INITRANS 1

STORAGE

(

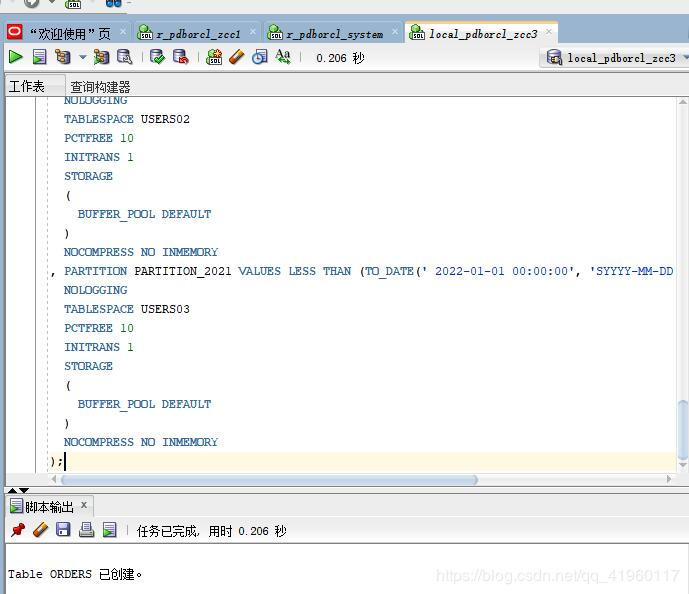
BUFFER\_POOL DEFAULT

)

NOCOMPRESS NO INMEMORY

);

分析：以上脚本在lm2用户权限下创建了ORDERS表，将ORDER\_ID作为主键，起名为ORDERS\_PK，使用索引（USING INDEX），表空间为USERS，就是将表段放到USERS中，块保留10%的空间留给更新该块数据使用（PCTFREE 10），初始化事务槽的个数（INITRANS 1），还设置了存储参数（STORAGE）BUFFER\_POOL默认缓冲池，作用是将buffer\_pool设置表的数据块读到内存中，对应放在哪个池中，不压缩表数据（NOCOMPRESS ），不指定对表进行DML操作时的并行度（NOPARALLEL）。接着以订单时间限制2016-01-01 00:00:00、2017-01-01 00:00:00、2018-01-01 00:00:00、2019-01-01 00:00:00、2020-01-01 00:00:00、2021-01-01 00:00:00、2022-01-01 00:00:00分别设置了7个分区：PARTITION\_2015、PARTITION\_2016、PARTITION\_2017、PARTITION\_2018、PARTITION\_2019、PARTITION\_2020、PARTITION\_2021，其中PARTITION\_2015、PARTITION\_2016、PARTITION\_2017使用的表空间为USERS；PARTITION\_2018、PARTITION\_2019、PARTITION\_2020使用的表空间为USERS02；PARTITION\_2021使用的表空间为USERS03。



4）创建order\_details表，表段放在表空间USERS中，分区依赖外键order\_id。

CREATE TABLE order\_details

(

id NUMBER(10, 0) NOT NULL

, order\_id NUMBER(10, 0) NOT NULL

, product\_name VARCHAR2(40 BYTE) NOT NULL

, product\_num NUMBER(8, 2) NOT NULL

, product\_price NUMBER(8, 2) NOT NULL

, CONSTRAINT order\_details\_fk1 FOREIGN KEY (order\_id)

REFERENCES orders ( order\_id )

ENABLE

)

TABLESPACE USERS

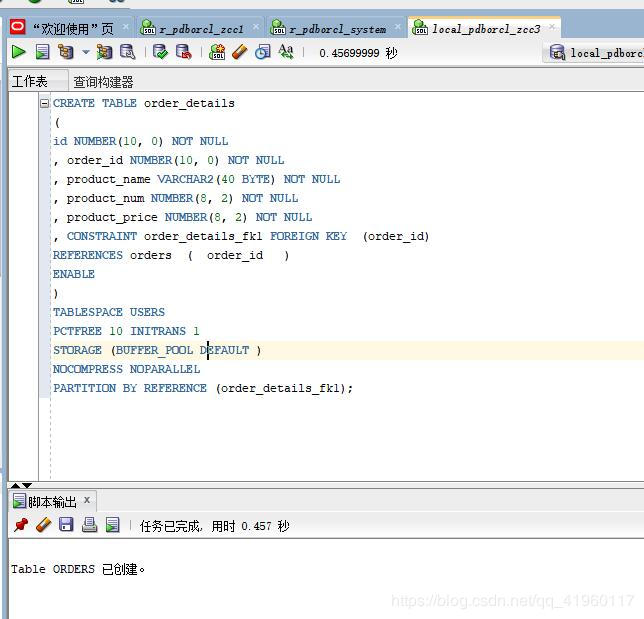
PCTFREE 10 INITRANS 1

STORAGE (BUFFER\_POOL DEFAULT )

NOCOMPRESS NOPARALLEL

PARTITION BY REFERENCE (order\_details\_fk1);

该语句创建了order\_details表，以order\_id为外键（名为order\_details\_fk1），依赖于orders表的order\_id,分区依赖于外键。



5）使用system用户给lm2的账号分配上述分区的使用权限。使用system用户给你的用户分配可以查询执行计划的权限。

ALTER USER lm2 QUOTA UNLIMITED ON USERS;

ALTER USER lm2 QUOTA UNLIMITED ON USERS02;

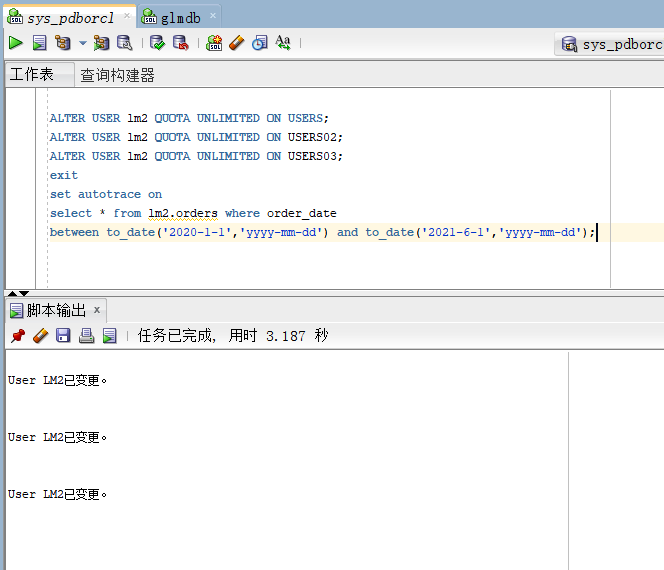
ALTER USER lm2 QUOTA UNLIMITED ON USERS03;

exit

set autotrace on

select \* from lm2.orders where order\_date

between to\_date('2020-1-1','yyyy-mm-dd') and to\_date('2021-6-1','yyyy-mm-dd');



6）表创建成功后，插入数据，数据能并平均分布到各个分区。每个表的数据都应该大于1万行，对表进行联合查询。

> declare dt date; m number(8,2); V\_EMPLOYEE\_ID NUMBER(6);

> v\_order\_id number(10); v\_name varchar2(100); v\_tel varchar2(100);

> v number(10,2); v\_order\_detail\_id number; begin

> /\* system login:

> ALTER USER "TEACHER" QUOTA UNLIMITED ON USERS; ALTER USER "TEACHER"

> QUOTA UNLIMITED ON USERS02; ALTER USER "TEACHER" QUOTA UNLIMITED ON

> USERS03;

> \*/

> v\_order\_detail\_id:=1; delete from order\_details; delete from orders; for i in 1..18000 loop

> if i mod 6 =0 then

> dt:=to\_date('2015-3-2','yyyy-mm-dd')+(i mod 60); --PARTITION\_2015

> elsif i mod 6 =1 then

> dt:=to\_date('2016-3-2','yyyy-mm-dd')+(i mod 60); --PARTITION\_2016

> elsif i mod 6 =2 then

> dt:=to\_date('2017-3-2','yyyy-mm-dd')+(i mod 60); --PARTITION\_2017

> elsif i mod 6 =3 then

> dt:=to\_date('2018-3-2','yyyy-mm-dd')+(i mod 60); --PARTITION\_2018

> elsif i mod 6 =4 then

> dt:=to\_date('2019-3-2','yyyy-mm-dd')+(i mod 60); --PARTITION\_2019

> else

> dt:=to\_date('2020-3-2','yyyy-mm-dd')+(i mod 60); --PARTITION\_2020

> end if;

> V\_EMPLOYEE\_ID:=CASE I MOD 6 WHEN 0 THEN 11 WHEN 1 THEN 111 WHEN 2 THEN 112

> WHEN 3 THEN 12 WHEN 4 THEN 121 ELSE 122 END;

> --插入订单

> v\_order\_id:=i;

> v\_name := 'meimei'|| 'meimei';

> v\_name := 'lili' || i;

> v\_tel := '123456' || i;

> insert /\*+append\*/ into ORDERS (ORDER\_ID,CUSTOMER\_NAME,CUSTOMER\_TEL,ORDER\_DATE,EMPLOYEE\_ID,DISCOUNT)

> values (v\_order\_id,v\_name,v\_tel,dt,V\_EMPLOYEE\_ID,dbms\_random.value(100,0));

> --插入订单y一个订单包括3个产品

> v:=dbms\_random.value(10000,4000);

> v\_name:='哆啦A梦'|| (i mod 3 + 1);

> insert /\*+append\*/ into ORDER\_DETAILS(ID,ORDER\_ID,PRODUCT\_NAME,PRODUCT\_NUM,PRODUCT\_PRICE)

> values (v\_order\_detail\_id,v\_order\_id,v\_name,2,v);

> v:=dbms\_random.value(1000,50);

> v\_name:='扫黑'|| (i mod 3 + 1);

> v\_order\_detail\_id:=v\_order\_detail\_id+1;

> insert /\*+append\*/ into ORDER\_DETAILS(ID,ORDER\_ID,PRODUCT\_NAME,PRODUCT\_NUM,PRODUCT\_PRICE)

> values (v\_order\_detail\_id,v\_order\_id,v\_name,3,v);

> v:=dbms\_random.value(9000,2000);

> v\_name:='我的姐姐'|| (i mod 3 + 1);>

> v\_order\_detail\_id:=v\_order\_detail\_id+1;

> insert /\*+append\*/ into ORDER\_DETAILS(ID,ORDER\_ID,PRODUCT\_NAME,PRODUCT\_NUM,PRODUCT\_PRICE)

> values (v\_order\_detail\_id,v\_order\_id,v\_name,1,v);

> --在触发器关闭的情况下，需要手工计算每个订单的应收金额：

> select sum(PRODUCT\_NUM\*PRODUCT\_PRICE) into m from ORDER\_DETAILS where ORDER\_ID=v\_order\_id;

> if m is null then

> m:=0;

> end if;

> UPDATE ORDERS SET TRADE\_RECEIVABLE = m - discount WHERE ORDER\_ID=v\_order\_id;

> IF I MOD 1000 =0 THEN

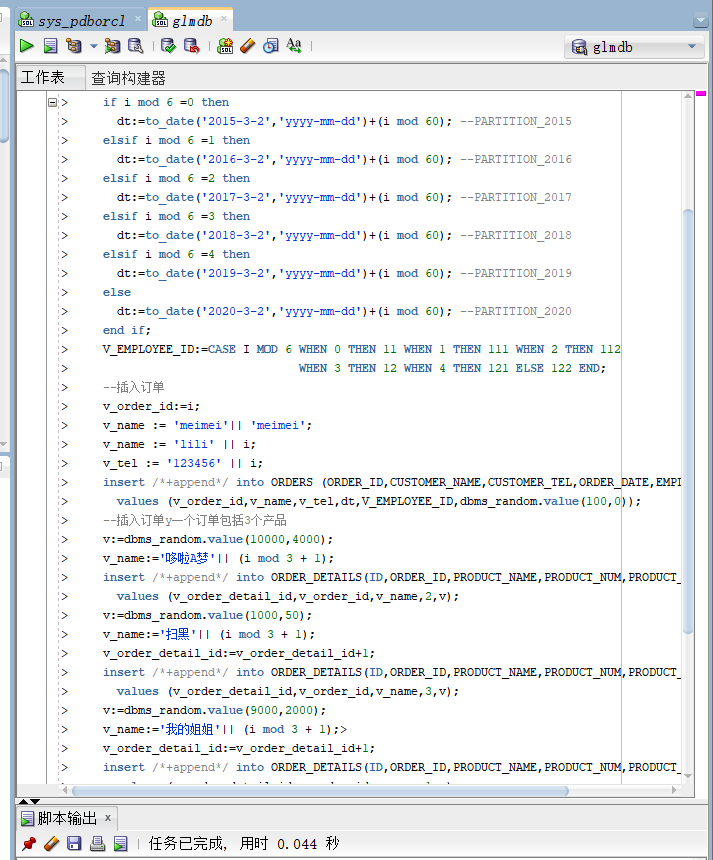
> commit; --每次提交会加快插入数据的速度

> END IF;

> end loop;

> end;

7）运行脚本进行批量插入数据，数据平均分布到各个分区。orders（一万八千行数据），order\_details（五万二千行数据）



8）查询数据、以及其数量

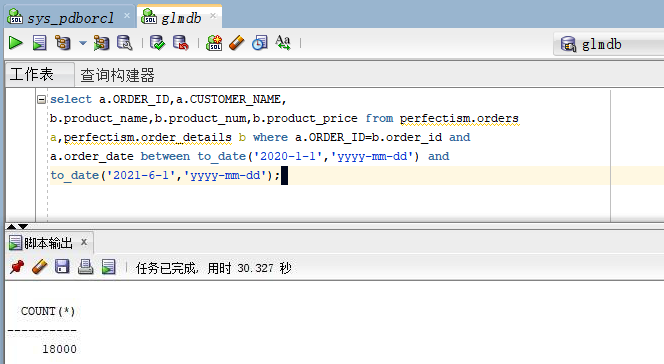
> select a.ORDER\_ID,a.CUSTOMER\_NAME,

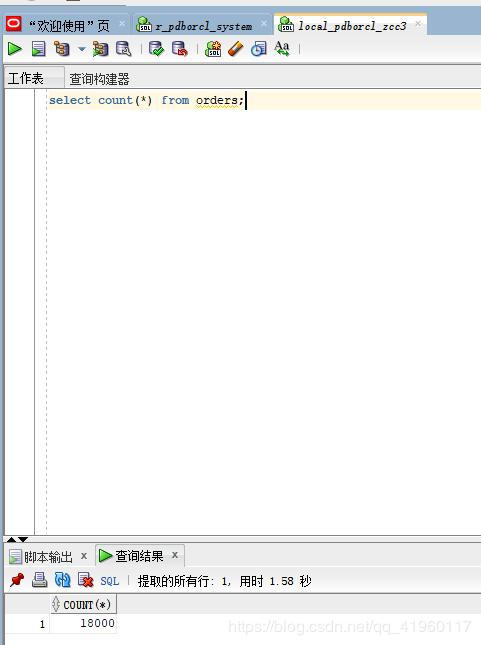
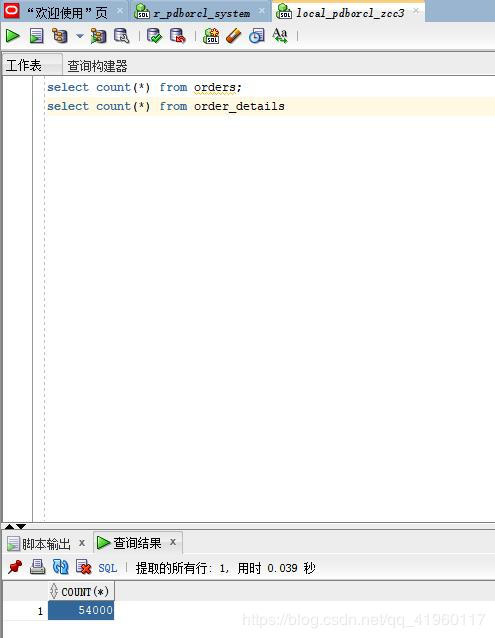
> b.product\_name,b.product\_num,b.product\_price from perfectism.orders

> a,perfectism.order\_details b where a.ORDER\_ID=b.order\_id and

> a.order\_date between to\_date('2020-1-1','yyyy-mm-dd') and

> to\_date('2021-6-1','yyyy-mm-dd');





查询语句是对orders与order\_details两张表进行了联合查询，查询了订单编号、顾客名字、电影的名字、电影票数量、电影票价格，查询的条件是订单日期在'2020-1-1'与'2021-6-1'之间的。

### **2.2.1 插入数据**

1）在lm2用户下删除可能存在的表（DEPARTMENTS、EMPLOYEES、ORDER\_ID\_TEMP、ORDER\_DETAILS、ORDERS、PRODUCTS）、序列（SEQ\_ORDER\_DETAILS\_ID、SEQ\_ORDER\_ID）、视图（VIEW\_ORDER\_DETAILS）、包（MYPACK）

--删除表和触发器等等

declare

num number;

begin

select count(1) into num from user\_tables where TABLE\_NAME = 'DEPARTMENTS';

if num=1 then

execute immediate 'drop table DEPARTMENTS cascade constraints PURGE';

end if;

select count(1) into num from user\_tables where TABLE\_NAME = 'EMPLOYEES';

if num=1 then

execute immediate 'drop table EMPLOYEES cascade constraints PURGE';

end if;

select count(1) into num from user\_tables where TABLE\_NAME = 'ORDER\_ID\_TEMP';

if num=1 then

execute immediate 'drop table ORDER\_ID\_TEMP cascade constraints PURGE';

end if;

select count(1) into num from user\_tables where TABLE\_NAME = 'ORDER\_DETAILS';

if num=1 then

execute immediate 'drop table ORDER\_DETAILS cascade constraints PURGE';

end if;

select count(1) into num from user\_tables where TABLE\_NAME = 'ORDERS';

if num=1 then

execute immediate 'drop table ORDERS cascade constraints PURGE';

end if;

select count(1) into num from user\_tables where TABLE\_NAME = 'PRODUCTS';

if num=1 then

execute immediate 'drop table PRODUCTS cascade constraints PURGE';

end if;

select count(1) into num from user\_sequences where SEQUENCE\_NAME = 'SEQ\_ORDER\_DETAILS\_ID';

if num=1 then

execute immediate 'drop SEQUENCE SEQ\_ORDER\_DETAILS\_ID';

end if;

select count(1) into num from user\_sequences where SEQUENCE\_NAME = 'SEQ\_ORDER\_ID';

if num=1 then

execute immediate 'drop SEQUENCE SEQ\_ORDER\_ID';

end if;

select count(1) into num from user\_views where VIEW\_NAME = 'VIEW\_ORDER\_DETAILS';

if num=1 then

execute immediate 'drop VIEW VIEW\_ORDER\_DETAILS';

end if;

SELECT count(object\_name) into num FROM user\_objects\_ae WHERE object\_type = 'PACKAGE' and OBJECT\_NAME='MYPACK';

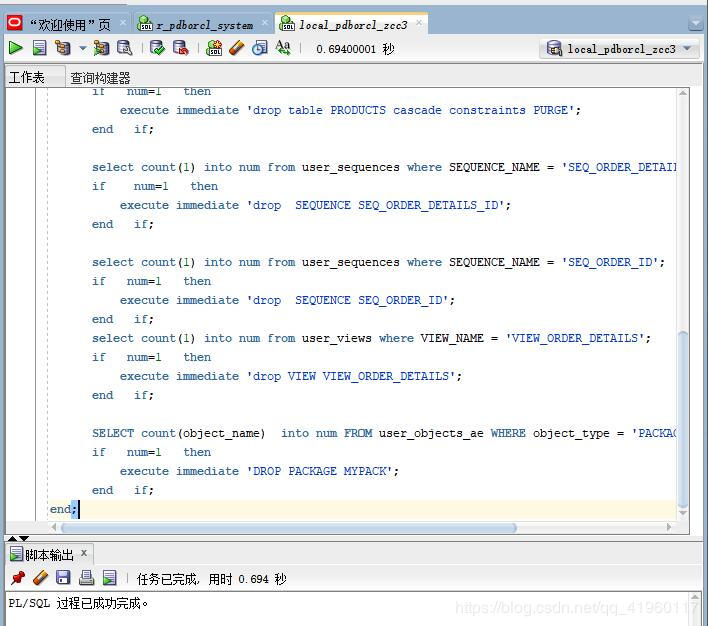
if num=1 then

execute immediate 'DROP PACKAGE MYPACK';

end if;

end;

Exit



2）创建表、索引、视图和序列。

----创建DEPARTMENTS表

CREATE TABLE DEPARTMENTS

(

DEPARTMENT\_ID NUMBER(6, 0) NOT NULL

, DEPARTMENT\_NAME VARCHAR2(40 BYTE) NOT NULL

, CONSTRAINT DEPARTMENTS\_PK PRIMARY KEY

(

DEPARTMENT\_ID

)

USING INDEX

(

CREATE UNIQUE INDEX DEPARTMENTS\_PK ON DEPARTMENTS (DEPARTMENT\_ID ASC)

NOLOGGING

TABLESPACE USERS

PCTFREE 10

INITRANS 2

STORAGE

(

INITIAL 65536

NEXT 1048576

MINEXTENTS 1

MAXEXTENTS UNLIMITED

BUFFER\_POOL DEFAULT

)

NOPARALLEL

)

ENABLE

)

NOLOGGING

TABLESPACE USERS

PCTFREE 10

INITRANS 1

STORAGE

(

INITIAL 65536

NEXT 1048576

MINEXTENTS 1

MAXEXTENTS UNLIMITED

BUFFER\_POOL DEFAULT

)

NOCOMPRESS NO INMEMORY NOPARALLEL;

--创建EMPLOYEES表

CREATE TABLE EMPLOYEES

(

EMPLOYEE\_ID NUMBER(6, 0) NOT NULL

, NAME VARCHAR2(40 BYTE) NOT NULL

, EMAIL VARCHAR2(40 BYTE)

, PHONE\_NUMBER VARCHAR2(40 BYTE)

, HIRE\_DATE DATE NOT NULL

, SALARY NUMBER(8, 2)

, MANAGER\_ID NUMBER(6, 0)

, DEPARTMENT\_ID NUMBER(6, 0)

, PHOTO BLOB

, CONSTRAINT EMPLOYEES\_PK PRIMARY KEY

(

EMPLOYEE\_ID

)

USING INDEX

(

CREATE UNIQUE INDEX EMPLOYEES\_PK ON EMPLOYEES (EMPLOYEE\_ID ASC)

NOLOGGING

TABLESPACE USERS

PCTFREE 10

INITRANS 2

STORAGE

(

INITIAL 65536

NEXT 1048576

MINEXTENTS 1

MAXEXTENTS UNLIMITED

BUFFER\_POOL DEFAULT

)

NOPARALLEL

)

ENABLE

)

NOLOGGING

TABLESPACE USERS

PCTFREE 10

INITRANS 1

STORAGE

(

INITIAL 65536

NEXT 1048576

MINEXTENTS 1

MAXEXTENTS UNLIMITED

BUFFER\_POOL DEFAULT

)

NOCOMPRESS

NO INMEMORY

NOPARALLEL

LOB (PHOTO) STORE AS SYS\_LOB0000092017C00009$$

(

ENABLE STORAGE IN ROW

CHUNK 8192

NOCACHE

NOLOGGING

TABLESPACE USERS

STORAGE

(

INITIAL 106496

NEXT 1048576

MINEXTENTS 1

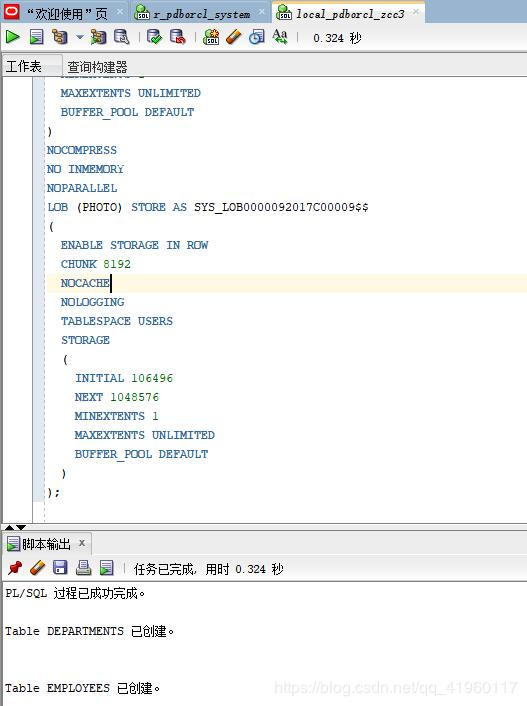
MAXEXTENTS UNLIMITED

BUFFER\_POOL DEFAULT

)

);

如上建表语句创建了DEPARTMENTS、EMPLOYEES表，并且使用索引，表空间都是USERS



3）为EMPLOYEES创建索引。

--创建索引

CREATE INDEX EMPLOYEES\_INDEX1\_NAME ON EMPLOYEES (NAME ASC)

NOLOGGING

TABLESPACE USERS

PCTFREE 10

INITRANS 2

STORAGE

(

INITIAL 65536

NEXT 1048576

MINEXTENTS 1

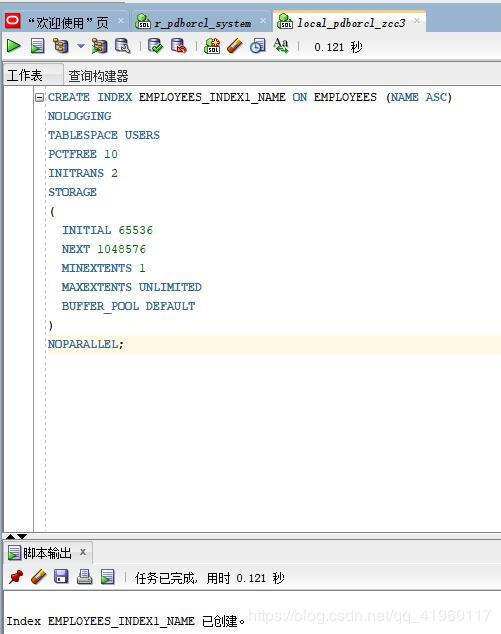
MAXEXTENTS UNLIMITED

BUFFER\_POOL DEFAULT

)

NOPARALLEL;

该语句为EMPLOYEES创建了索引EMPLOYEES\_INDEX1\_NAME。



4）为EMPLOYEES表创建约束。

--创建约束

--给EMPLOYEES表加外键约束，外键为DEPARTMENTS的DEPARTMENT\_ID，别名为EMPLOYEES\_FK1

ALTER TABLE EMPLOYEES

ADD CONSTRAINT EMPLOYEES\_FK1 FOREIGN KEY

(

DEPARTMENT\_ID

)

REFERENCES DEPARTMENTS

(

DEPARTMENT\_ID

)

ENABLE;

--给EMPLOYEES表加外键约束，外建为本表的MANAGER\_ID，别名为EMPLOYEES\_FK2

ALTER TABLE EMPLOYEES

ADD CONSTRAINT EMPLOYEES\_FK2 FOREIGN KEY

(

MANAGER\_ID

)

REFERENCES EMPLOYEES

(

EMPLOYEE\_ID

)

ON DELETE SET NULL ENABLE;

--给表加加check约束

--给表EMPLOYEES加check约束，条件为 SALARY>0

ALTER TABLE EMPLOYEES

ADD CONSTRAINT EMPLOYEES\_SALARY CHECK

(SALARY>0)

ENABLE;

--给EMPLOYEES表加check约束，条件为EMPLOYEE\_ID<>MANAGER\_ID

ALTER TABLE EMPLOYEES

ADD CONSTRAINT EMPLOYEES\_CHK2 CHECK

(EMPLOYEE\_ID<>MANAGER\_ID)

ENABLE;

--给EMPLOYEES表加check约束，条件为MANAGER\_ID<>EMPLOYEE\_ID

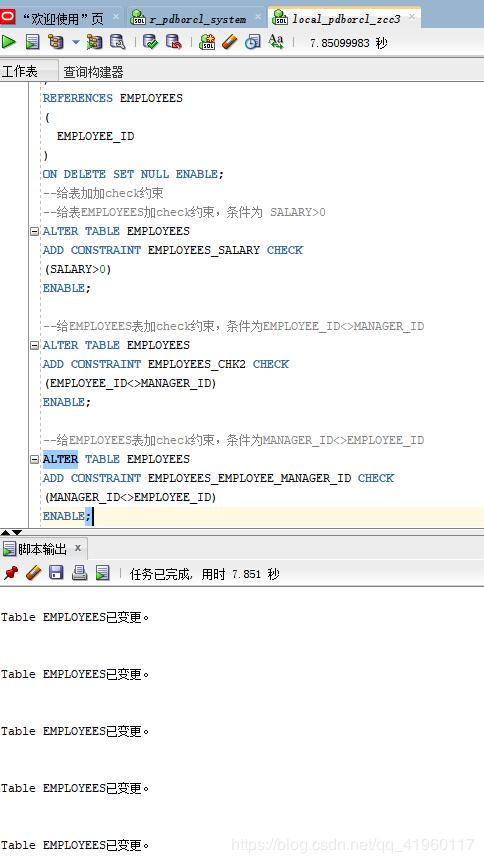
ALTER TABLE EMPLOYEES

ADD CONSTRAINT EMPLOYEES\_EMPLOYEE\_MANAGER\_ID CHECK

(MANAGER\_ID<>EMPLOYEE\_ID)

ENABLE;

该语句为EMPLOYEES表创建如下约束：给EMPLOYEES表添加加外键约束，外键为DEPARTMENTS的DEPARTMENT\_ID，别名为EMPLOYEES\_FK1；给EMPLOYEES表添加加外键约束，外建为本表的MANAGER\_ID，别名为EMPLOYEES\_FK2；给表EMPLOYEES添加check约束，条件为 SALARY>0；给EMPLOYEES表添加check约束，条件为EMPLOYEE\_ID<>MANAGER\_ID；给EMPLOYEES表添加check约束，条件为MANAGER\_ID<>EMPLOYEE\_ID。



5）创建PRODUCTS表并且为其添加约束

--创建PRODUCTS表

CREATE TABLE PRODUCTS

(

PRODUCT\_NAME VARCHAR2(40 BYTE) NOT NULL

, PRODUCT\_TYPE VARCHAR2(40 BYTE) NOT NULL

, CONSTRAINT PRODUCTS\_PK PRIMARY KEY

(

PRODUCT\_NAME

)

ENABLE

)

LOGGING

TABLESPACE "USERS"

PCTFREE 10

INITRANS 1

STORAGE

(

INITIAL 65536

NEXT 1048576

MINEXTENTS 1

MAXEXTENTS 2147483645

BUFFER\_POOL DEFAULT

);

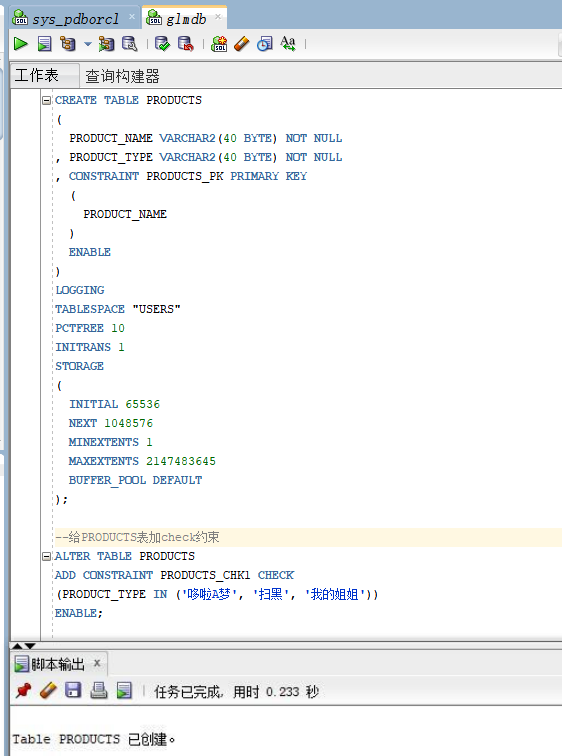
--给PRODUCTS表加check约束

ALTER TABLE PRODUCTS

ADD CONSTRAINT PRODUCTS\_CHK1 CHECK

(PRODUCT\_TYPE IN ('哆啦A梦', '扫黑', '我的姐姐'))

ENABLE;



6）创建ORDER\_ID\_TEMP表、ORDERS表并且为ORDERS增加了外键和主键约束。

--创建临时表ORDER\_ID\_TEMP用于触发器存储临时ORDER\_ID

CREATE GLOBAL TEMPORARY TABLE "ORDER\_ID\_TEMP"

( "ORDER\_ID" NUMBER(10,0) NOT NULL ENABLE,

CONSTRAINT "ORDER\_ID\_TEMP\_PK" PRIMARY KEY ("ORDER\_ID") ENABLE

) ON COMMIT DELETE ROWS ;

COMMENT ON TABLE "ORDER\_ID\_TEMP" IS '用于触发器存储临时ORDER\_ID';

--创建ORDERS表，并且分区：PARTITION\_BEFORE\_2016、PARTITION\_BEFORE\_2017

CREATE TABLE ORDERS

(

ORDER\_ID NUMBER(10, 0) NOT NULL

, CUSTOMER\_NAME VARCHAR2(40 BYTE) NOT NULL

, CUSTOMER\_TEL VARCHAR2(40 BYTE) NOT NULL

, ORDER\_DATE DATE NOT NULL

, EMPLOYEE\_ID NUMBER(6, 0) NOT NULL

, DISCOUNT NUMBER(8, 2) DEFAULT 0

, TRADE\_RECEIVABLE NUMBER(8, 2) DEFAULT 0

)

TABLESPACE USERS

PCTFREE 10

INITRANS 1

STORAGE

(

BUFFER\_POOL DEFAULT

)

NOCOMPRESS

NOPARALLEL

PARTITION BY RANGE (ORDER\_DATE)

(

PARTITION PARTITION\_BEFORE\_2016 VALUES LESS THAN (TO\_DATE(' 2016-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS\_CALENDAR=GREGORIAN'))

NOLOGGING

TABLESPACE USERS

PCTFREE 10

INITRANS 1

STORAGE

(

INITIAL 8388608

NEXT 1048576

MINEXTENTS 1

MAXEXTENTS UNLIMITED

BUFFER\_POOL DEFAULT

)

NOCOMPRESS NO INMEMORY

, PARTITION PARTITION\_BEFORE\_2017 VALUES LESS THAN (TO\_DATE(' 2017-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS\_CALENDAR=GREGORIAN'))

NOLOGGING

TABLESPACE USERS02

PCTFREE 10

INITRANS 1

STORAGE

(

INITIAL 8388608

NEXT 1048576

MINEXTENTS 1

MAXEXTENTS UNLIMITED

BUFFER\_POOL DEFAULT

)

NOCOMPRESS NO INMEMORY

);

--为ORDERS表增加外键和主键约束

ALTER TABLE ORDERS

ADD CONSTRAINT ORDERS\_PK PRIMARY KEY

(

ORDER\_ID

)

USING INDEX ORDERS\_PK

ENABLE;

ALTER TABLE ORDERS

ADD CONSTRAINT ORDERS\_FK1 FOREIGN KEY

(

EMPLOYEE\_ID

)

REFERENCES EMPLOYEES

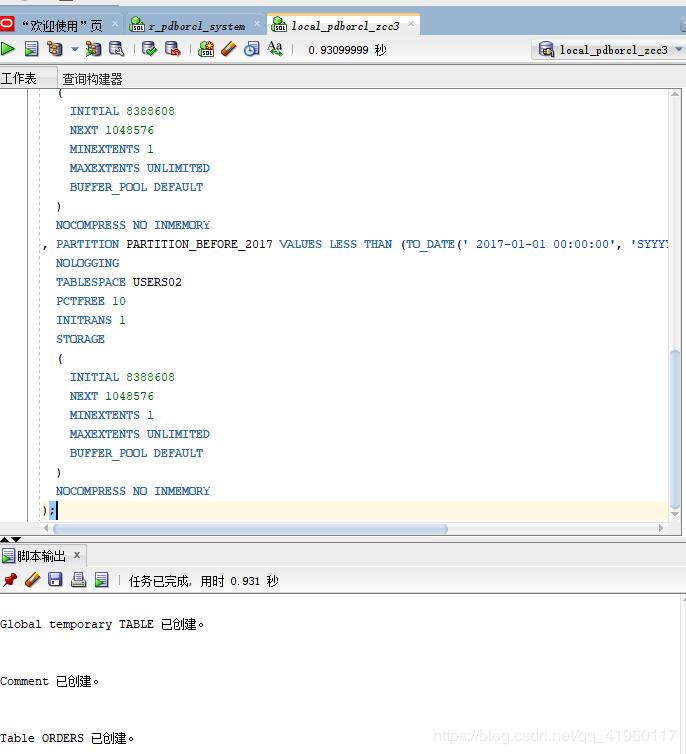
(

EMPLOYEE\_ID

)

ENABLE;

该语句创建了创建了ORDER\_ID\_TEMP表、ORDERS表。临时表ORDER\_ID\_TEMP用于触发器存储临时ORDER\_ID；创建ORDERS表时，对其分区：PARTITION\_BEFORE\_2016、PARTITION\_BEFORE\_2017，分区条件：ORDER\_DATE，随后对ORDERS增加了外键和主键约束，主键名：ORDERS\_PK，外键名：ORDERS\_FK1。



7）插入数据、开启触发器、动态增加分区：PARTITION\_BEFORE\_2018。

--插入DEPARTMENTS，EMPLOYEES数据

INSERT INTO DEPARTMENTS(DEPARTMENT\_ID,DEPARTMENT\_NAME) values (1,'董事会');

INSERT INTO EMPLOYEES(EMPLOYEE\_ID,NAME,EMAIL,PHONE\_NUMBER,HIRE\_DATE,SALARY,MANAGER\_ID,DEPARTMENT\_ID)

VALUES (1,'周董事长',NULL,NULL,to\_date('2010-1-1','yyyy-mm-dd'),50000,NULL,1);

INSERT INTO DEPARTMENTS(DEPARTMENT\_ID,DEPARTMENT\_NAME) values (11,'影院售票系统前台');

INSERT INTO EMPLOYEES(EMPLOYEE\_ID,NAME,EMAIL,PHONE\_NUMBER,HIRE\_DATE,SALARY,MANAGER\_ID,DEPARTMENT\_ID)

VALUES (11,'影片咨询处',NULL,NULL,to\_date('2010-1-1','yyyy-mm-dd'),50000,1,1);

INSERT INTO EMPLOYEES(EMPLOYEE\_ID,NAME,EMAIL,PHONE\_NUMBER,HIRE\_DATE,SALARY,MANAGER\_ID,DEPARTMENT\_ID)

VALUES (111,'站内新闻处',NULL,NULL,to\_date('2010-1-1','yyyy-mm-dd'),50000,11,11);

INSERT INTO EMPLOYEES(EMPLOYEE\_ID,NAME,EMAIL,PHONE\_NUMBER,HIRE\_DATE,SALARY,MANAGER\_ID,DEPARTMENT\_ID)

VALUES (112,'座位购买处',NULL,NULL,to\_date('2010-1-1','yyyy-mm-dd'),50000,11,11);

INSERT INTO DEPARTMENTS(DEPARTMENT\_ID,DEPARTMENT\_NAME) values (12,'影院售票系统后台');

INSERT INTO EMPLOYEES(EMPLOYEE\_ID,NAME,EMAIL,PHONE\_NUMBER,HIRE\_DATE,SALARY,MANAGER\_ID,DEPARTMENT\_ID)

VALUES (12,'影片信息管理员',NULL,NULL,to\_date('2010-1-1','yyyy-mm-dd'),50000,1,1);

INSERT INTO EMPLOYEES(EMPLOYEE\_ID,NAME,EMAIL,PHONE\_NUMBER,HIRE\_DATE,SALARY,MANAGER\_ID,DEPARTMENT\_ID)

VALUES (121,'新闻信息管理员',NULL,NULL,to\_date('2010-1-1','yyyy-mm-dd'),50000,12,12);

INSERT INTO EMPLOYEES(EMPLOYEE\_ID,NAME,EMAIL,PHONE\_NUMBER,HIRE\_DATE,SALARY,MANAGER\_ID,DEPARTMENT\_ID)

VALUES (122,'订单信息管理员',NULL,NULL,to\_date('2010-1-1','yyyy-mm-dd'),50000,12,12);

insert into products (product\_name,product\_type) values ('duolaameng','哆啦A梦');

insert into products (product\_name,product\_type) values ('duolaameng','哆啦A梦');

insert into products (product\_name,product\_type) values ('duolaameng','哆啦A梦');

insert into products (product\_name,product\_type) values ('saohei','扫黑');

insert into products (product\_name,product\_type) values ('saohei','扫黑');

insert into products (product\_name,product\_type) values ('saohei','扫黑');

insert into products (product\_name,product\_type) values ('wodejiejie','我的姐姐');

insert into products (product\_name,product\_type) values ('wodejiejie','我的姐姐');

insert into products (product\_name,product\_type) values ('wodejiejie','我的姐姐');

--批量插入订单数据，注意ORDERS.TRADE\_RECEIVABLE（订单应收款）的自动计算,注意插入数据的速度

declare

dt date;

m number(8,2);

V\_EMPLOYEE\_ID NUMBER(6);

v\_order\_id number(10);

v\_name varchar2(100);

v\_tel varchar2(100);

v number(10,2);

begin

for i in 1..18000

loop

if i mod 2 =0 then

dt:=to\_date('2015-3-2','yyyy-mm-dd')+(i mod 60);

else

dt:=to\_date('2016-3-2','yyyy-mm-dd')+(i mod 60);

end if;

V\_EMPLOYEE\_ID:=CASE I MOD 6 WHEN 0 THEN 11 WHEN 1 THEN 111 WHEN 2 THEN 112

WHEN 3 THEN 12 WHEN 4 THEN 121 ELSE 122 END;

--插入订单

v\_order\_id:=SEQ\_ORDER\_ID.nextval; --应该将SEQ\_ORDER\_ID.nextval保存到变量中。

v\_name := 'meimei'|| 'meimei';

v\_name := 'lili' || i;

v\_tel := '1123456' || i;

insert /\*+append\*/ into ORDERS (ORDER\_ID,CUSTOMER\_NAME,CUSTOMER\_TEL,ORDER\_DATE,EMPLOYEE\_ID,DISCOUNT)

values (v\_order\_id,v\_name,v\_tel,dt,V\_EMPLOYEE\_ID,dbms\_random.value(100,0));

--插入订单y一个订单包括3个产品

v:=dbms\_random.value(10000,4000);

v\_name:='哆啦A梦'|| (i mod 3 + 1);

insert /\*+append\*/ into ORDER\_DETAILS(ID,ORDER\_ID,PRODUCT\_NAME,PRODUCT\_NUM,PRODUCT\_PRICE)

values (SEQ\_ORDER\_DETAILS\_ID.NEXTVAL,v\_order\_id,v\_name,2,v);

v:=dbms\_random.value(1000,50);

v\_name:='扫黑'|| (i mod 3 + 1);

insert /\*+append\*/ into ORDER\_DETAILS(ID,ORDER\_ID,PRODUCT\_NAME,PRODUCT\_NUM,PRODUCT\_PRICE)

values (SEQ\_ORDER\_DETAILS\_ID.NEXTVAL,v\_order\_id,v\_name,3,v);

v:=dbms\_random.value(9000,2000);

v\_name:='我的姐姐'|| (i mod 3 + 1);

insert /\*+append\*/ into ORDER\_DETAILS(ID,ORDER\_ID,PRODUCT\_NAME,PRODUCT\_NUM,PRODUCT\_PRICE)

values (SEQ\_ORDER\_DETAILS\_ID.NEXTVAL,v\_order\_id,v\_name,1,v);

--在触发器关闭的情况下，需要手工计算每个订单的应收金额：

select sum(PRODUCT\_NUM\*PRODUCT\_PRICE) into m from ORDER\_DETAILS where ORDER\_ID=v\_order\_id;

if m is null then

m:=0;

end if;

UPDATE ORDERS SET TRADE\_RECEIVABLE = m - discount WHERE ORDER\_ID=v\_order\_id;

IF I MOD 1000 =0 THEN

commit; --每次提交会加快插入数据的速度

END IF;

end loop;

--统计用户的所有表，所需时间很长：2千万行数据，需要1600秒，该语句可选

--dbms\_stats.gather\_schema\_stats(User,estimate\_percent=>100,cascade=> TRUE); --estimate\_percent采样行的百分比

end;

ALTER TRIGGER "ORDERS\_TRIG\_ROW\_LEVEL" ENABLE;

ALTER TRIGGER "ORDER\_DETAILS\_SNTNS\_TRIG" ENABLE;

ALTER TRIGGER "ORDER\_DETAILS\_ROW\_TRIG" ENABLE;

--最后动态增加一个PARTITION\_BEFORE\_2018分区：

ALTER TABLE ORDERS

ADD PARTITION PARTITION\_BEFORE\_2018 VALUES LESS THAN (TO\_DATE(' 2018-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS\_CALENDAR=GREGORIAN'));

ALTER INDEX ORDERS\_INDEX\_DATE

MODIFY PARTITION PARTITION\_BEFORE\_2018

NOCOMPRESS;

该语句对DEPARTMENTS，EMPLOYEES表和ORDERS、ORDER\_DETAILS表插入数据。开启了触发器，最后还动态增加了一个分区：PARTITION\_BEFORE\_2018分区。



8）递归查询周董事长及其所有下属，子下属员工。

--递归查询某个员工(周董事长)及其所有下属，子下属员工。

WITH A (EMPLOYEE\_ID,NAME,EMAIL,PHONE\_NUMBER,HIRE\_DATE,SALARY,MANAGER\_ID,DEPARTMENT\_ID) AS

(SELECT EMPLOYEE\_ID,NAME,EMAIL,PHONE\_NUMBER,HIRE\_DATE,SALARY,MANAGER\_ID,DEPARTMENT\_ID

FROM employees WHERE employee\_ID = 1

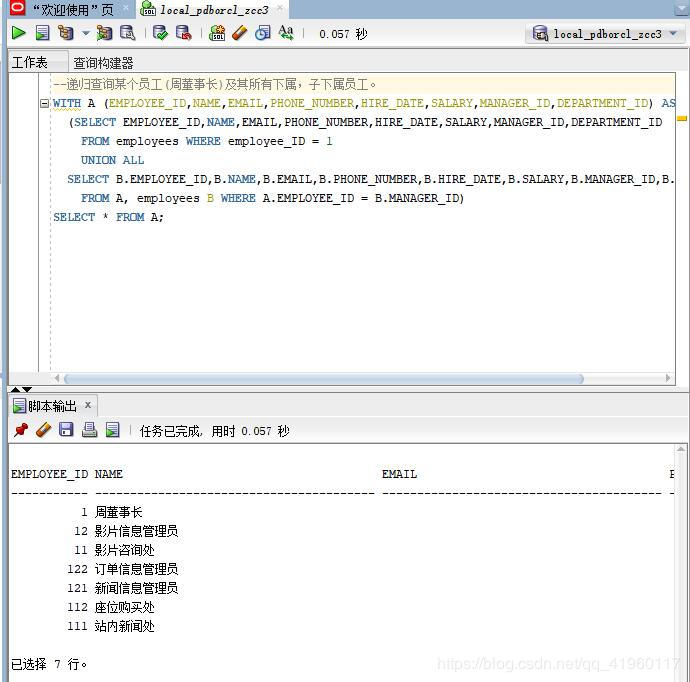
UNION ALL

SELECT B.EMPLOYEE\_ID,B.NAME,B.EMAIL,B.PHONE\_NUMBER,B.HIRE\_DATE,B.SALARY,B.MANAGER\_ID,B.DEPARTMENT\_ID

FROM A, employees B WHERE A.EMPLOYEE\_ID = B.MANAGER\_ID)

SELECT \* FROM A;

其中用了表连接UNION ALL（列数相同，类型也要相同）虽然都是查的是employees表，但是这张表却可以同时作为员工表和经理表，查询条件就是A表的员工编号等于B表的经理编号。



9）查询部门表，同时显示部门名字。

--查询部门表，同时显示部门的负责人姓名。嵌套子查询，建立了两个字表temp，temp1

select d.\* ,temp1.name from

(select e.name,e.DEPARTMENT\_ID from employees e,

(select DEPARTMENT\_ID,min(nvl(manager\_id,0)) minnum from employees group by employees.DEPARTMENT\_ID) temp

where e.DEPARTMENT\_ID in temp.DEPARTMENT\_ID and nvl(e.manager\_id,0) in temp.minnum group by e.DEPARTMENT\_ID,e.name ) temp1,

departments d

where d.department\_id=temp1.department\_id order by temp1.department\_id;

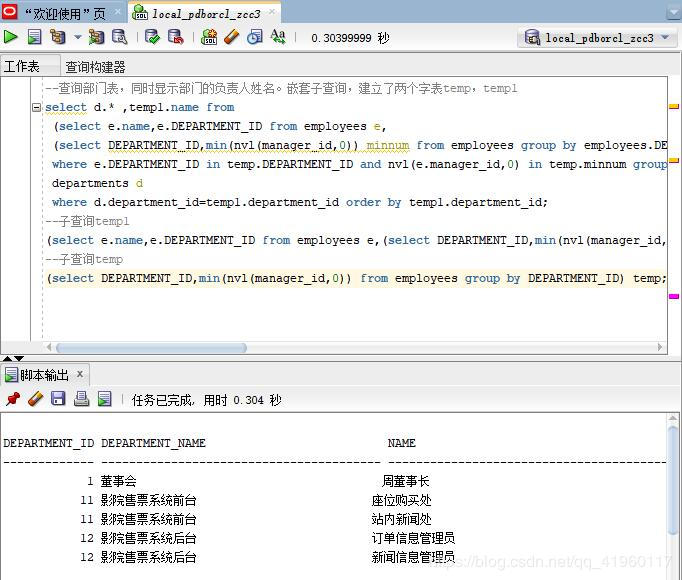
--子查询temp1

(select e.name,e.DEPARTMENT\_ID from employees e,(select DEPARTMENT\_ID,min(nvl(manager\_id,0)) minnum from employees group by employees.DEPARTMENT\_ID) temp where e.DEPARTMENT\_ID in temp.DEPARTMENT\_ID and nvl(e.manager\_id,0) in temp.minnum group by e.DEPARTMENT\_ID,e.name ) temp1;

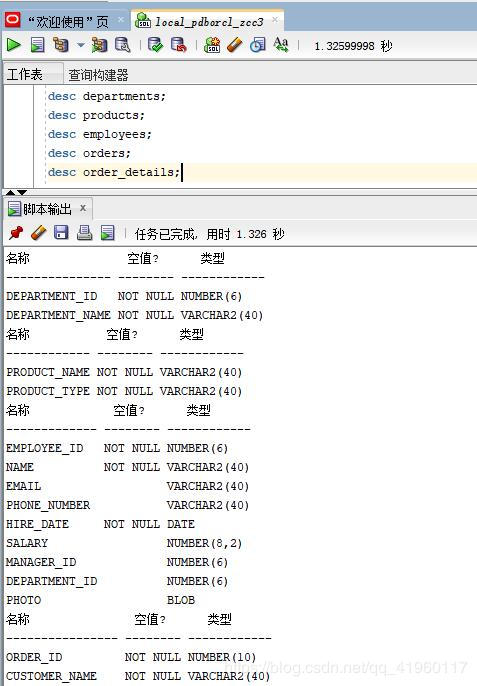
--子查询temp

(select DEPARTMENT\_ID,min(nvl(manager\_id,0)) from employees group by DEPARTMENT\_ID) temp;

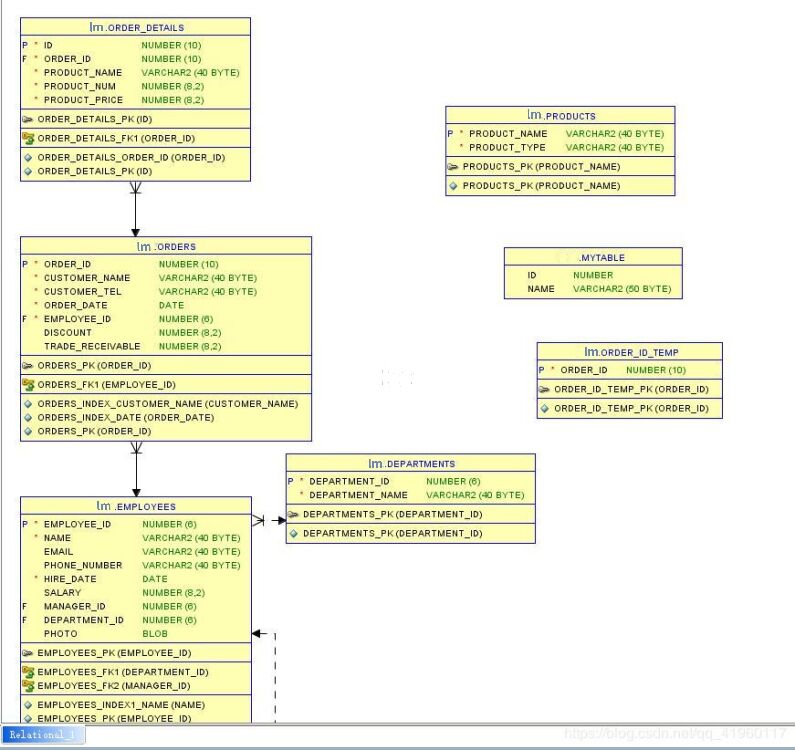
分析：该语句查询嵌套了两个子查询：temp和temp1，子查询temp的作用是把查询的结果通过部门编号分组，每组通过聚合函数min()来找到每个组的经理编号的最小值，该最小值就是部门负责人的经理编号，董事长为null,用nvl()函数将其变为0；将其数据通过temp的临时表给temp1,查询出最小值经理编号的员工和部门编号；最后通过临时表temp1给父查询，查询出各个部门的所有信息。



10）查看departments、PRODUCTS、 EMPLOYEES、ORDERS、ORDER\_DETAILS表的结构。



11）数据关系图



# 创建一个包(Package)

包名是MyPack，在包中用PL/SQL语言设计一些存储过程和函数，实现比较复杂的业务逻辑。

1） 创建一个包myPack

create or replace PACKAGE MyPack IS

/\*

本实验以实验4为基础。

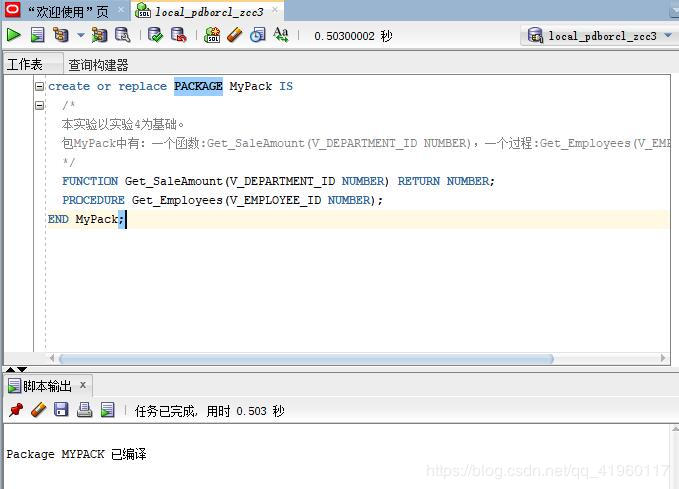
包MyPack中有：一个函数:Get\_SaleAmount(V\_DEPARTMENT\_ID NUMBER)，一个过程:Get\_Employees(V\_EMPLOYEE\_ID NUMBER)

\*/

FUNCTION Get\_SaleAmount(V\_DEPARTMENT\_ID NUMBER) RETURN NUMBER;

PROCEDURE Get\_Employees(V\_EMPLOYEE\_ID NUMBER);

END MyPack;



2）在MyPack中创建一个函数SaleAmount ，查询部门表，统计每个部门的销售总金额，每个部门的销售额是由该部门的员工(ORDERS.EMPLOYEE\_ID)完成的销售额之和。函数SaleAmount要求输入的参数是部门号，输出部门的销售金额

create or replace PACKAGE BODY MyPack IS

FUNCTION Get\_SaleAmount(V\_DEPARTMENT\_ID NUMBER) RETURN NUMBER

AS

N NUMBER(20,2); --注意，订单ORDERS.TRADE\_RECEIVABLE的类型是NUMBER(8,2),汇总之后，数据要大得多。

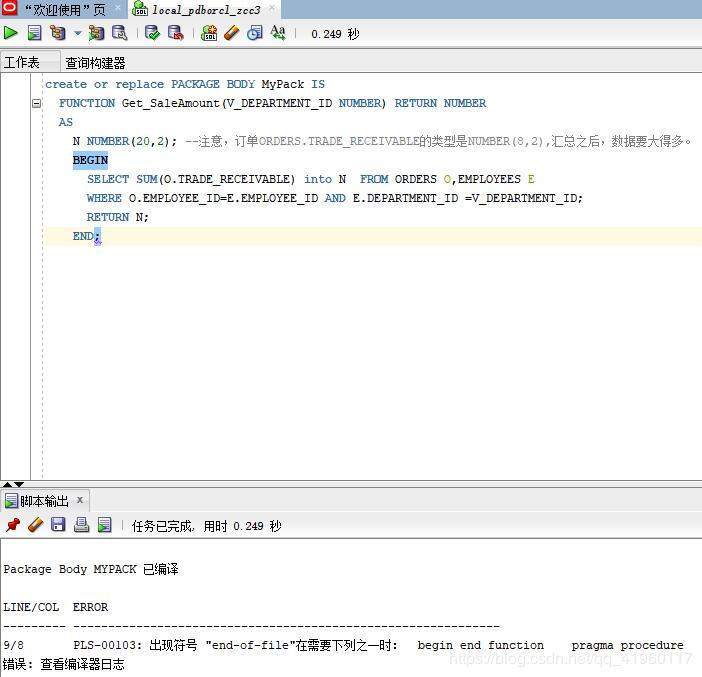
BEGIN

SELECT SUM(O.TRADE\_RECEIVABLE) into N FROM ORDERS O,EMPLOYEES E

WHERE O.EMPLOYEE\_ID=E.EMPLOYEE\_ID AND E.DEPARTMENT\_ID =V\_DEPARTMENT\_ID;

RETURN N;

END;



3）在MyPack中创建一个过程，在过程中使用游标，递归查询某个员工及其所有下属，子下属员工。过程的输入参数是员工号，输出员工的ID,姓名，销售总金额。信息用dbms\_output包中的put或者put\_line函数。输出的员工信息用左添加空格的多少表示员工的层次（LEVEL）。

PROCEDURE GET\_EMPLOYEES(V\_EMPLOYEE\_ID NUMBER)

AS

LEFTSPACE VARCHAR(2000);

begin

--通过LEVEL判断递归的级别

LEFTSPACE:=' ';

--使用游标

for v in

(SELECT LEVEL,EMPLOYEE\_ID,NAME,MANAGER\_ID FROM employees

START WITH EMPLOYEE\_ID = V\_EMPLOYEE\_ID

CONNECT BY PRIOR EMPLOYEE\_ID = MANAGER\_ID)

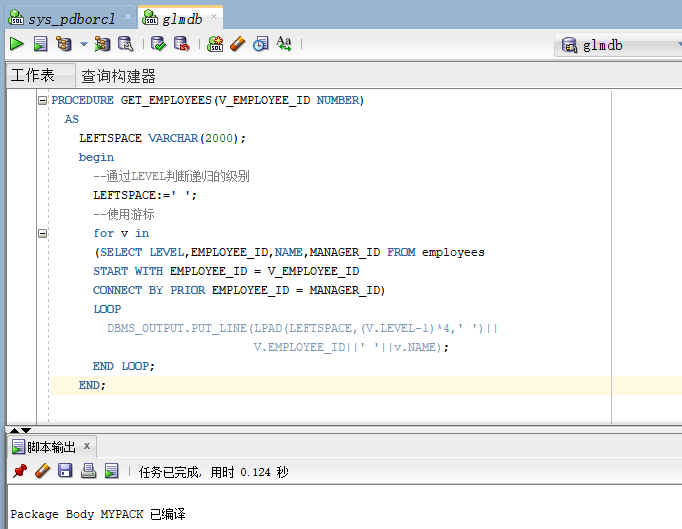
LOOP

DBMS\_OUTPUT.PUT\_LINE(LPAD(LEFTSPACE,(V.LEVEL-1)\*4,' ')||

V.EMPLOYEE\_ID||' '||v.NAME);

END LOOP;

END;



上面的代码通过游标来获取循环得到的数据，参数是一个number类型的员工编号，里面用到了start with…connect by， connect by是结构化查询中用到的，其基本语法是：

select … from tablename

start with 条件1

connect by 条件2

where 条件3;

简单说来是将一个树状结构存储在一张表里，比如一个表中存在两个字段:org\_id，parent\_id，那么通过表示每一条记录的parent是谁，就可以形成一个树状结构，用上述语法的查询可以取得这棵树的所有记录，就可以通过这个来查询出一个员工的上司和下属。其中：条件1 是根结点的限定语句，当然可以放宽限定条件，以取得多个根结点，实际就是多棵树。条件2 是连接条件，其中用PRIOR表示上一条记录，比如 CONNECT BY PRIOR org\_id = parent\_id；就是说上一条记录的org\_id 是本条记录的parent\_id，即本记录的父亲是上一条记录。条件3 是过滤条件，用于对返回的所有记录进行过滤。

4）测试代码：

-- 函数Get\_SaleAmount()测试方法：select count(\*) from orders;select MyPack.Get\_SaleAmount(1) AS 部门11应收金额,MyPack.Get\_SaleAmount(11) AS 部门12应收金额 from dual;

-- 过程Get\_Employees()测试代码：set serveroutput on

DECLARE

V\_EMPLOYEE\_ID NUMBER; BEGIN

V\_EMPLOYEE\_ID := 1;

MYPACK.Get\_Employees ( V\_EMPLOYEE\_ID => V\_EMPLOYEE\_ID) ;

V\_EMPLOYEE\_ID := 11;

MYPACK.Get\_Employees ( V\_EMPLOYEE\_ID => V\_EMPLOYEE\_ID) ;

END;



上述sql语句的作用是插叙订单数量，并且调用Get\_SaleAmount()函数来查询部门1和部门11的订单销售总额，打开serveroutput，调用包MYPACK里面的Get\_Employees()函数，来查询出该员工的所有上司和下属。该语句只查询了员工编号1和11的上司和下属。=>是Oracle中调用存储过程的时候, 指定参数名进行调用.一般是，某些参数有默认值的时候，你需要跳过某些参数来进行调用。

# 四、备份方案

## **4.1 开始全备份**

在Linux系统终端输入数据库进行level0备份的命令

[oracle@oracle-pc ~]$ cat rman\_level0.sh

[oracle@oracle-pc ~]$ ./rman\_level0.sh



### 4.1.1 查看备份文件

.log是日志文件 dblv0.bak是数据库的备份文件 arclv0\*.bak是归档日期的备份文件 c-1392946895-20191120-01是控制文件和参数的备份

[oracle@oracle-pc ~]$ cd rman\_backup

[oracle@oracle-pc rman\_backup]$ ls

arclv0\_ORCL\_20191120\_dauhb2fm\_1\_1.bak

c-1392946895-20191120-01

dblv0\_ORCL\_20191120\_d7uhb2ap\_1\_1.bak

dblv0\_ORCL\_20191120\_d8uhb2c6\_1\_1.bak

dblv0\_ORCL\_20191120\_d9uhb2ei\_1\_1.bak

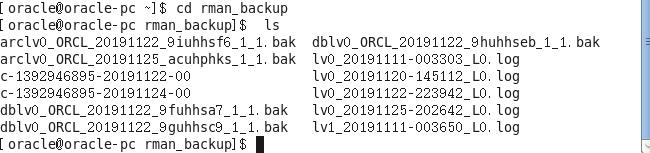
lv0\_20191120-083949\_L0.log

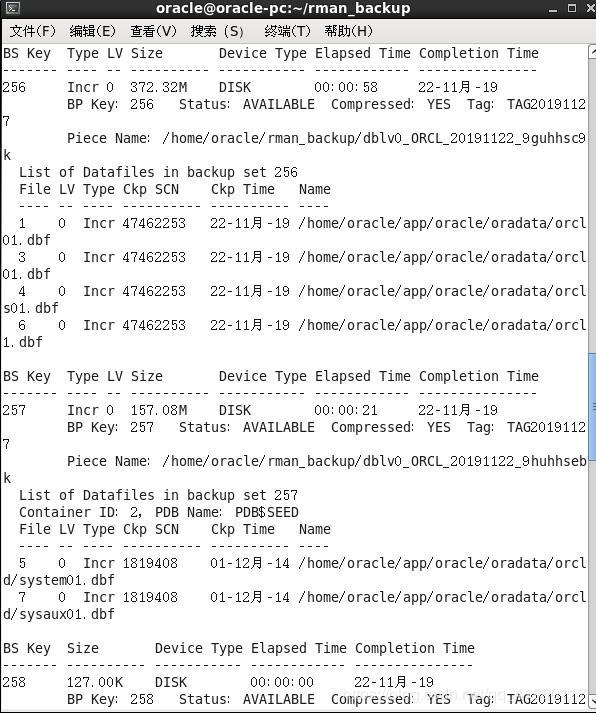
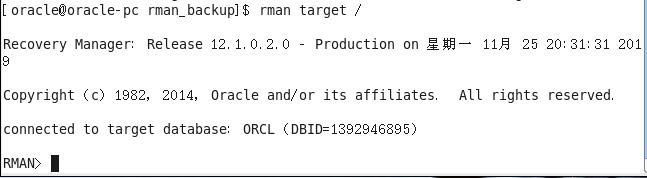


### 4.1.2 查看备份文件的内容

[oracle@oracle-pc ~]$ rman target /

RMAN> list backup;





## 4.2 备份后修改数据

[oracle@oracle-pc ~]$ sqlplus study/123@pdborcl

SQL> create table t2 (id number,name varchar2(50));

Table created.

SQL> insert into t2 values(1,'lili');

1 row created.

SQL> commit;

Commit complete.

SQL> select \* from t1;

ID NAME

---------- --------------------------------------------------

1 lili

SQL> exit

## 4.3 删除数据库文件，模拟数据库文件损坏

### 4.3.1 删除数据库文件

[oracle@oracle-pc ~]$ rm /home/oracle/app/oracle/oradata/orcl/pdborcl/SAMPLE\_SCHEMA\_users01.dbf

IMG_295

### 4.3.2 删除数据库文件后修改数据

[oracle@oracle-pc ~]$ sqlplus study/123@pdborcl

SQL> insert into t1 values(2,'lili');

1 row created.

SQL> commit;

Commit complete.

SQL> select \* from t2;

ID NAME

---------- --------------------------------------------------

2 meimei

1 lili

3 meimei

SQL>

SQL> declare

2 n number;

3 begin

4 for n in 1..10000 loop

5 insert into t2 values(n,'name'||n);

6 end loop;

7 end;

8 /

declare

\*

SQL> select \* from t2;

ID NAME

---------- --------------------------------------------------

2 meimei

1 lili

3 meimei

SQL> exit

删除数据文件后，仍然可以增加一条数据。这是因为增加的数据并没有写入数据文件，而是写到了日志文件中。如果增加的数据较多的时候，就会出问题了。

## 4.4 数据库完全恢复

重启损坏的数据库到mount状态

[oracle@oracle-pc ~]$ sqlplus / as sysdba

SQL> shutdown immediate

ORA-01116: 打开数据库文件 10 时出错

ORA-01110: 数据文件 10: '/home/oracle/app/oracle/oradata/orcl/pdborcl/SAMPLE\_SCHEMA\_users01.dbf'

ORA-27041: 无法打开文件

Linux-x86\_64 Error: 2: No such file or directory

Additional information: 3

SQL> shutdown abort

ORACLE instance shut down.

SQL> startup mount

ORACLE instance started.

Total System Global Area 1577058304 bytes

Fixed Size 2924832 bytes

Variable Size 738201312 bytes

Database Buffers 654311424 bytes

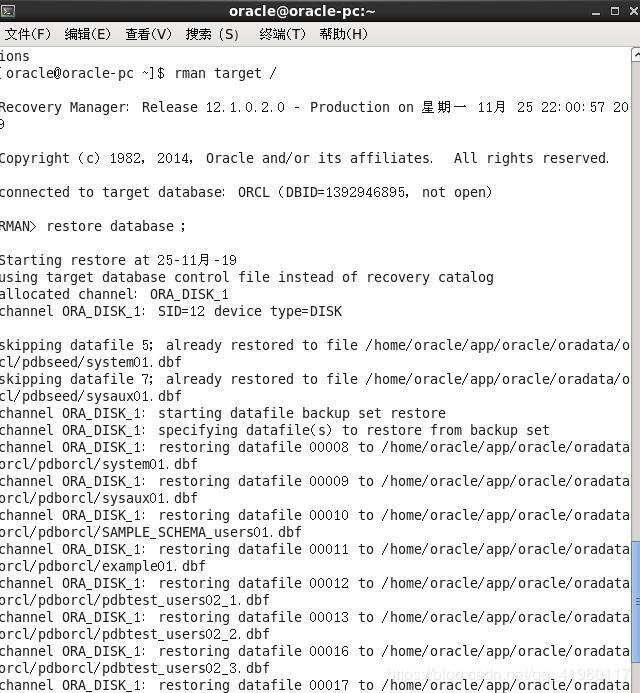
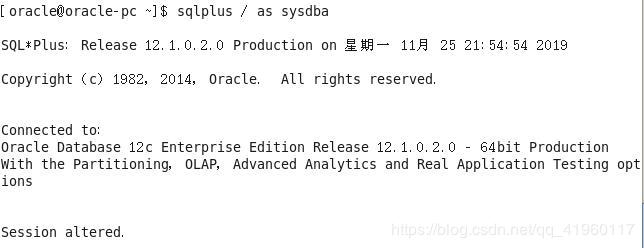
Redo Buffers 13848576 bytes

In-Memory Area 167772160 bytes

Database mounted.

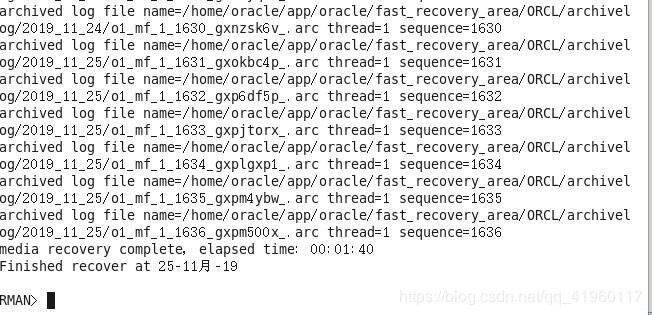
SQL> exit

通过shutdown immediate无法正常关闭数据库，只能通过shutdown abort强制关闭。然后将数据库启动到mount状态。









## 4.5 查询数据是否恢复

查询数据库是否恢复成功

[oracle@oracle-pc ~]$ sqlplus study/123@pdborcl

SQL> select \* from t2;

ID NAME

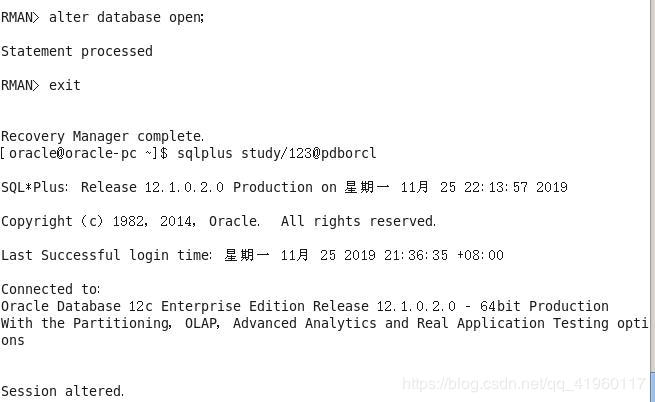
---------- --------------------------------------------------

2 meimei

1 lili

3 meimei

SQL>



由以上查询结果可见，数据100%恢复了

# 五、容灾方案

## 5.1 备库

### **5.1.1 Linux终端下创建会用到的文件夹**

mkdir -p /home/oracle/app/oracle/diag/orcl

mkdir -p /home/oracle/app/oracle/oradata/stdorcl/

mkdir -p /home/oracle/app/oracle/oradata/stdorcl/pdborcl

mkdir -p /home/oracle/arch

mkdir -p /home/oracle/rman

mkdir -p /home/oracle/app/oracle/oradata/stdorcl/pdbseed/

mkdir -p /home/oracle/app/oracle/oradata/stdorcl/pdb/

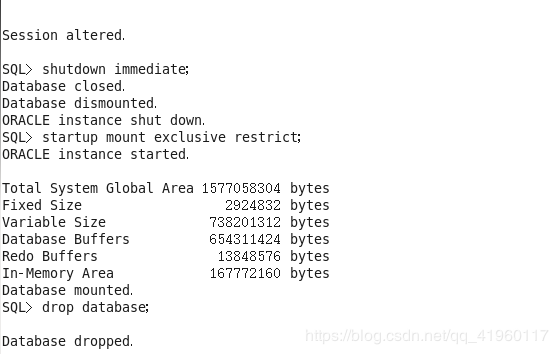
### **5.1.2 删除原有数据库**

$sqlplus / as sysdba

shutdown immediate;

startup mount exclusive restrict;

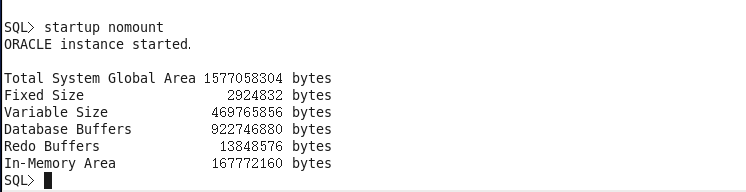
drop database;



### **5.1.3 启动到nomount**

$sqlplus / as sysdba

startup nomount



## 5.2 主库

1）连接到数据库

```sql $sqlplus / sysdba select group#,thread#,members,status from v$log;

alter database add standby logfile group 5 '/home/oracle/app/oracle/oradata/orcl/stdredo1.log' size 50m; alter database add standby logfile group 6 '/home/oracle/app/oracle/oradata/orcl/stdredo2.log' size 50m; alter database add standby logfile group 7 '/home/oracle/app/oracle/oradata/orcl/stdredo3.log' size 50m; alter database add standby logfile group 8 '/home/oracle/app/oracle/oradata/orcl/stdredo4.log' size 50m;

- 主库环境开启强制归档

```sql

ALTER DATABASE FORCE LOGGING;

alter system set LOG\_ARCHIVE\_CONFIG='DG\_CONFIG=(orcl,stdorcl)' scope=both sid='\*';

alter system set log\_archive\_dest\_1='LOCATION=/home/oracle/arch VALID\_FOR=(ALL\_LOGFILES,ALL\_ROLES) DB\_UNIQUE\_NAME=orcl' scope=spfile;

alter system set LOG\_ARCHIVE\_DEST\_2='SERVICE=stdorcl LGWR ASYNC VALID\_FOR=(ONLINE\_LOGFILES,PRIMARY\_ROLE) DB\_UNIQUE\_NAME=stdorcl' scope=both sid='\*';

alter system set fal\_client='orcl' scope=both sid='\*';

alter system set FAL\_SERVER='stdorcl' scope=both sid='\*';

alter system set standby\_file\_management=AUTO scope=both sid='\*';

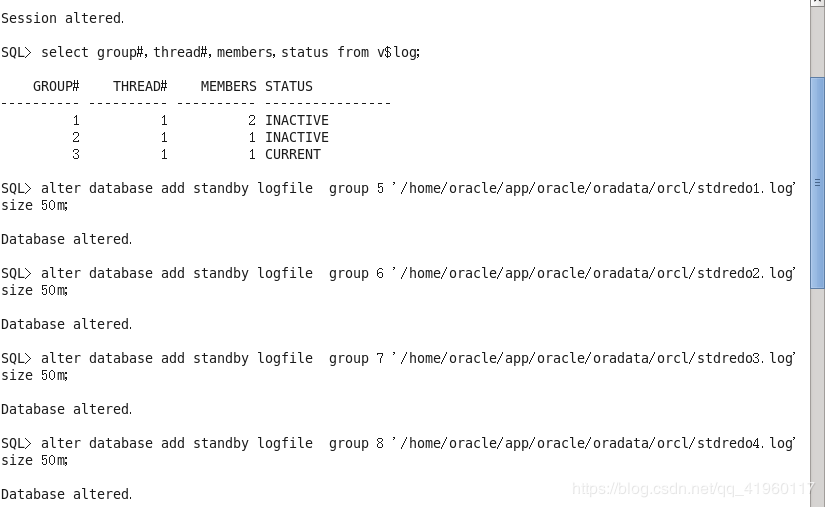
alter system set DB\_FILE\_NAME\_CONVERT='/home/oracle/app/oracle/oradata/stdorcl/','/home/oracle/app/oracle/oradata/orcl/' scope=spfile sid='\*';

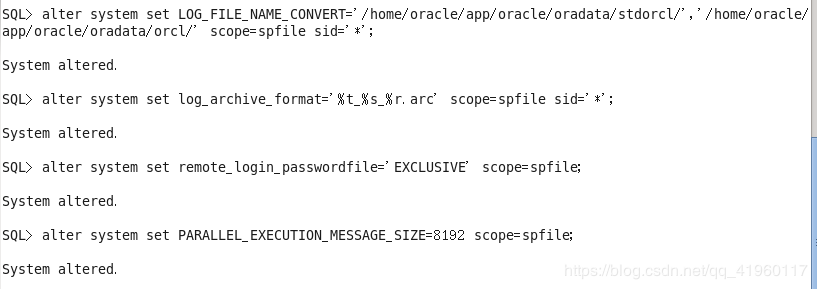
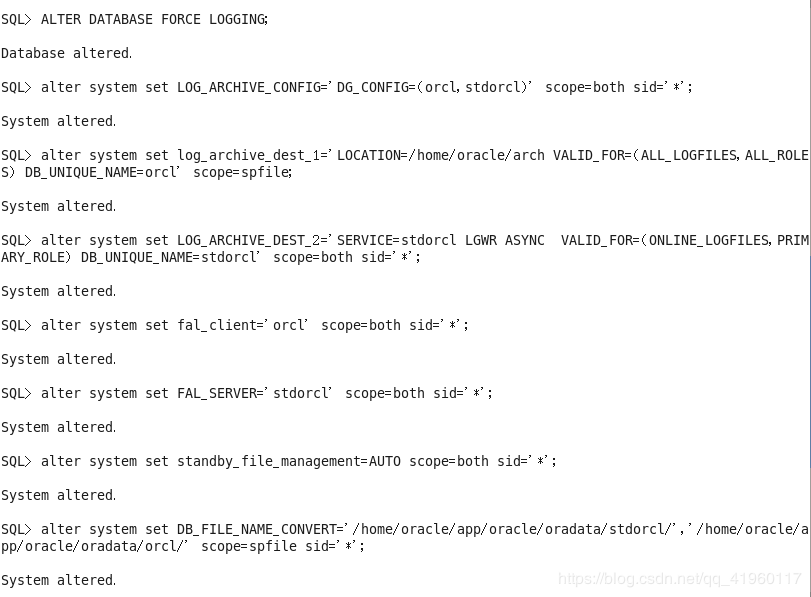
alter system set LOG\_FILE\_NAME\_CONVERT='/home/oracle/app/oracle/oradata/stdorcl/','/home/oracle/app/oracle/oradata/orcl/' scope=spfile sid='\*';

alter system set log\_archive\_format='%t\_%s\_%r.arc' scope=spfile sid='\*';

alter system set remote\_login\_passwordfile='EXCLUSIVE' scope=spfile;

alter system set PARALLEL\_EXECUTION\_MESSAGE\_SIZE=8192 scope=spfile;





1. 编辑主库以及备库的/home/oracle/app/oracle/product/12.1.0/dbhome\_1/network/admin/tnsnames.ora文件 注：注： 此处的ip地址，每个人在进行实验时，或许分配ip地址都不同，在进行文件拷贝之前，最好测试一下主机与从机之间是否能Ping 通。（在完成实验时，还应当具备基础的计算机网络相关知识，网络的基本配置，ip地址，子网掩码，网关号的查看，修改，计算机之间的网络连通性等等）

```sql $gedit /home/oracle/app/oracle/product/12.1.0/dbhome\_1/network/admin/tnsnames.ora

ORCL = (DESCRIPTION = (ADDRESS\_LIST = (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.133.131)(PORT = 1521)) ) (CONNECT\_DATA = (SERVER = DEDICATED) (SERVICE\_NAME = orcl) ) )

stdorcl = (DESCRIPTION = (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.133.133)(PORT = 1521)) (CONNECT\_DATA = (SERVER = DEDICATED) (SID = orcl) ) )

- 在主库上生成备库的参数文件:

```sql

SQL>create pfile from spfile;

File created

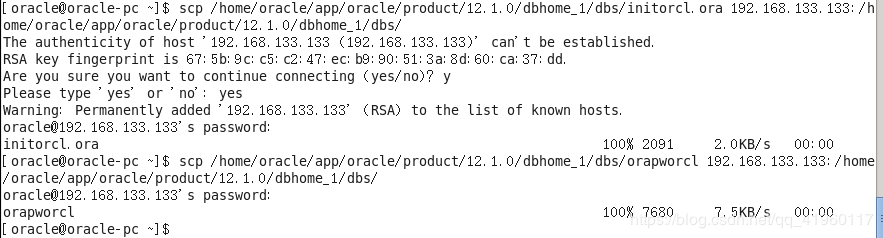
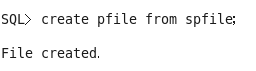
生成/home/oracle/app/oracle/product/12.1.0/dbhome\_1/dbs/initorcl.ora

3）将主库的参数文件，密码文件拷贝到备库:

scp /home/oracle/app/oracle/product/12.1.0/dbhome\_1/dbs/initorcl.ora 192.168.133.133:/home/oracle/app/oracle/product/12.1.0/dbhome\_1/dbs/

scp /home/oracle/app/oracle/product/12.1.0/dbhome\_1/dbs/orapworcl 192.168.133.132:/home/oracle/app/oracle/product/12.1.0/dbhome\_1/dbs/

注： 此处的ip地址，每个人在进行实验时，或许分配ip地址都不同，在进行文件拷贝之前，最好测试一下主机与从机之间是否能Ping 通



4）将主库复制到备库

$rman target sys/123@orcl auxiliary sys/123@stdorcl

5）执行duplicate:

run{

allocate meimeinel c1 type disk;

allocate meimeinel c2 type disk;

allocate meimeinel c3 type disk;

allocate AUXILIARY meimeinel c4 type disk;

allocate AUXILIARY meimeinel c5 type disk;

allocate AUXILIARY meimeinel c6 type disk;

DUPLICATE TARGET DATABASE

FOR STANDBY

FROM ACTIVE DATABASE

DORECOVER

NOFILENAMECHECK;

release meimeinel c1;

release meimeinel c2;

release meimeinel c3;

release meimeinel c4;

release meimeinel c5;

release meimeinel c6;

}

## 5.3 备库

1）在备库上更改参数文件

$gedit /home/oracle/app/oracle/product/12.1.0/dbhome\_1/dbs/initorcl.ora

2）文件内容是：

orcl.\_\_data\_transfer\_cache\_size=0

orcl.\_\_db\_cache\_size=671088640

orcl.\_\_java\_pool\_size=16777216

orcl.\_\_large\_pool\_size=33554432

orcl.\_\_oracle\_base='/home/oracle/app/oracle'#ORACLE\_BASE set from environment

orcl.\_\_pga\_aggregate\_target=536870912

orcl.\_\_sga\_target=1258291200

orcl.\_\_shared\_io\_pool\_size=50331648

orcl.\_\_shared\_pool\_size=301989888

orcl.\_\_streams\_pool\_size=0

\*.\_allow\_resetlogs\_corruption=TRUE

\*.\_catalog\_foreign\_restore=FALSE

\*.audit\_file\_dest='/home/oracle/app/oracle/admin/orcl/adump'

\*.audit\_trail='db'

\*.compatible='12.1.0.2.0'

\*.control\_files='/home/oracle/app/oracle/oradata/orcl/control01.ctl','/home/oracle/app/oracle/fast\_recovery\_area/orcl/control02.ctl','/home/oracle/app/oracle/fast\_recovery\_area/orcl/control03.ctl'

\*.db\_block\_size=8192

\*.db\_domain=''

\*.db\_file\_name\_convert='/home/oracle/app/oracle/oradata/orcl/','/home/oracle/app/oracle/oradata/stdorcl/'

\*.db\_name='orcl'

\*.db\_unique\_name='stdorcl'

\*.db\_recovery\_file\_dest='/home/oracle/app/oracle/fast\_recovery\_area'

\*.db\_recovery\_file\_dest\_size=4823449600

\*.diagnostic\_dest='/home/oracle/app/oracle'

\*.dispatchers='(PROTOCOL=TCP)(dispatchers=1)(pool=on)(ticks=1)(connections=500)(sessions=1000)'

\*.enable\_pluggable\_database=true

\*.fal\_client='stdorcl'

\*.fal\_server='orcl'

\*.inmemory\_max\_populate\_servers=2

\*.inmemory\_size=157286400

\*.local\_listener=''

\*.log\_archive\_config='DG\_CONFIG=(stdorcl,orcl)'

\*.log\_archive\_dest\_1='LOCATION=/home/oracle/arch VALID\_FOR=(ALL\_LOGFILES,ALL\_ROLES) DB\_UNIQUE\_NAME=stdorcl'

\*.log\_archive\_dest\_2='SERVICE=orcl LGWR ASYNC VALID\_FOR=(ONLINE\_LOGFILES,PRIMARY\_ROLE) DB\_UNIQUE\_NAME=orcl'

\*.log\_archive\_format='%t\_%s\_%r.arc'

\*.log\_file\_name\_convert='/home/oracle/app/oracle/oradata/orcl/','/home/oracle/app/oracle/oradata/stdorcl/'

\*.max\_dispatchers=5

\*.max\_shared\_servers=20

\*.open\_cursors=400

\*.parallel\_execution\_message\_size=8192

\*.pga\_aggregate\_target=511m

\*.processes=300

\*.recovery\_parallelism=0

\*.remote\_login\_passwordfile='EXCLUSIVE'

\*.service\_names='ORCL'

\*.sga\_max\_size=1572864000

\*.sga\_target=1258291200

\*.shared\_server\_sessions=200

\*.standby\_file\_management='AUTO'

\*.undo\_tablespace='UNDOTBS1'

此处为完全替换原来文件中的信息

3）在备库增加静态监听

$gedit /home/oracle/app/oracle/product/12.1.0/dbhome\_1/network/admin/listener.ora

4）文件内增加的信息为

SID\_LIST\_LISTENER =

(SID\_LIST =

(SID\_DESC =

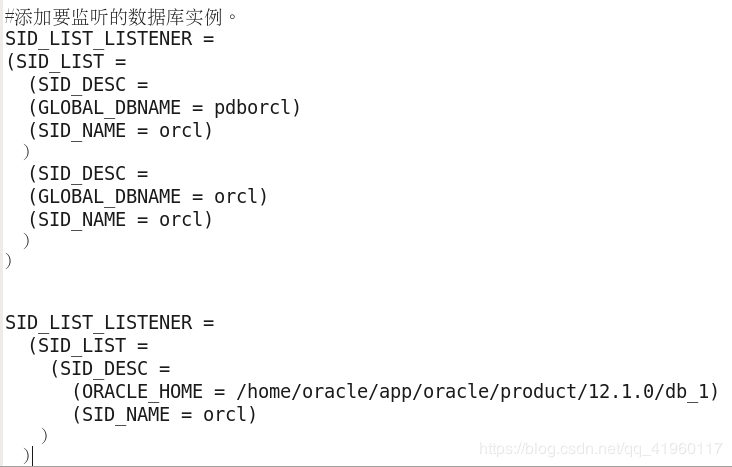
(ORACLE\_HOME = /home/oracle/app/oracle/product/12.1.0/db\_1)

(SID\_NAME = orcl)

)

)

注： 此处应该增添至文件最后（且记得保存）



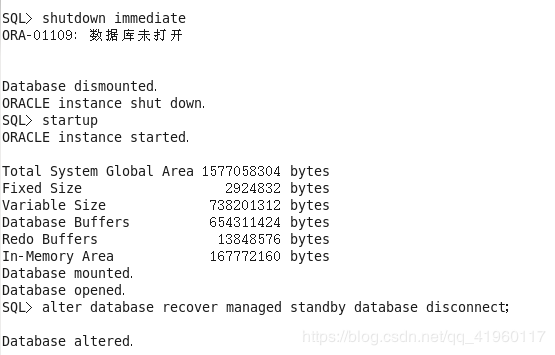
5）重新启动,备库开启实时应用模式

$sqlplus / as sysdba

shutdown immediate

startup

alter database recover managed standby database disconnect;



## 5.4 数据同步测试，主库+备库

### 5.4.1 在主库修改数据（ 创建了一张 t1 的表）

SQL> create table t1 (id number);

Table created.

### 5.4.2 在备库查询修改

SQL> select \* from t1;

no rows selected.