

# 1、简介

---

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

## 1.1、优点

---

1. 可用性强
2. 可扩展性强
3. 数据安全性强
4. 稳定性强

## 1.2、文件结构

---

数据库的物理存储结构是由一些多种物理文件组成，主要有**数据文件、控制文件、重做日志文件、归档日志文件、参数文件、口令文件、警告文件**等

- 控制文件：存储实例、数据文件及日志文件等信息的二进制文件。alter system set control\_files='路径'。V\$CONTROLFILE。
- 数据文件：存储数据，以.dbf做后缀。一句话：一个表空间对多个数据文件，一个数据文件只对一个表空间。dba\_data\_files/v\$datafile。
- 日志文件：即Redo Log Files和Archivelog Files。记录数据库修改信息。ALTER SYSTEM SWITCH LOGFILE;。V\$LOG。
- 参数文件：记录基本参数。spfile和pfile。
- 警告文件：show parameter background\_dump\_dest---使用共享服务器连接
- 跟踪文件：show parameter user\_dump\_dest---使用专用服务器连接

# 2、Oracle 11gXE

---

## 2.1、介绍

---

Oracle是这样介绍XE的：11g XE（Express Edition）简化版是在Oracle11gR2基础之上一个入门级的小体量数据库，免费用于开发/部署与发布，下载很快，使用简单。

### 限制

Oracle是这样解释XE的，一个强大的，性能已经得到证实的业界领先的软件，而且升级简单无需其他成本和复杂的移植。

为什么需要升级，因为这是一个精简版，精简的根本在于如下的限制，正是因为有了这个限制，XE才是真正的精简版而不至于才华横溢。

### 资源项 限制

CPU 一台机器上不超过一CPU

内存 内存不会超过1G

数据 数据库存储的数据量不会超过11G

## 2.2、下载

---

Oracle 11gXE: <https://www.oracle.com/database/technologies/xe-prior-releases.html>

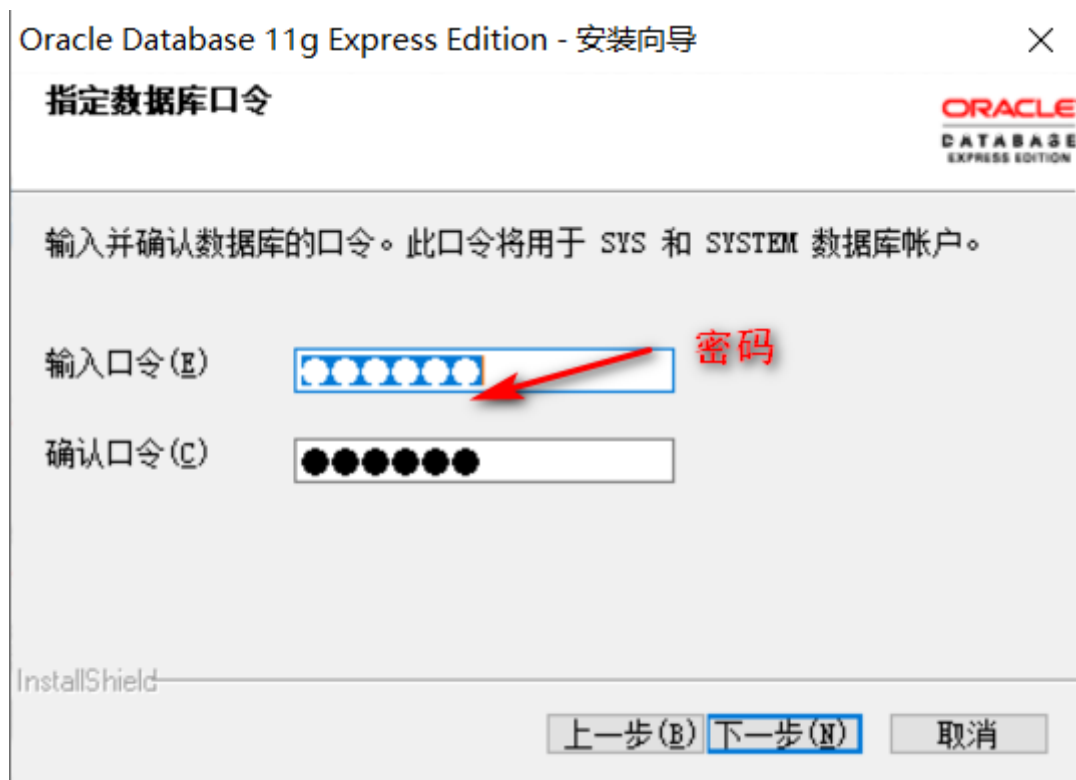
instantclient: <https://www.oracle.com/database/technologies/instant-client/downloads.html>

### 3.3、安装

参考文档: <https://my.oschina.net/hongjunzhan/blog/1553158>

#### Oracle 11gXE安装





测试

```
C:\Windows\system32\cmd.exe - sqlplus
Microsoft Windows [版本 10.0.18362.657]
(c) 2019 Microsoft Corporation. 保留所有权利。

C:\Users\Beloved>sqlplus

SQL*Plus: Release 11.2.0.2.0 Production on 星期日 2月 16 10:19:29 2020

Copyright (c) 1982, 2014, Oracle. All rights reserved.

请输入用户名: system
输入口令:

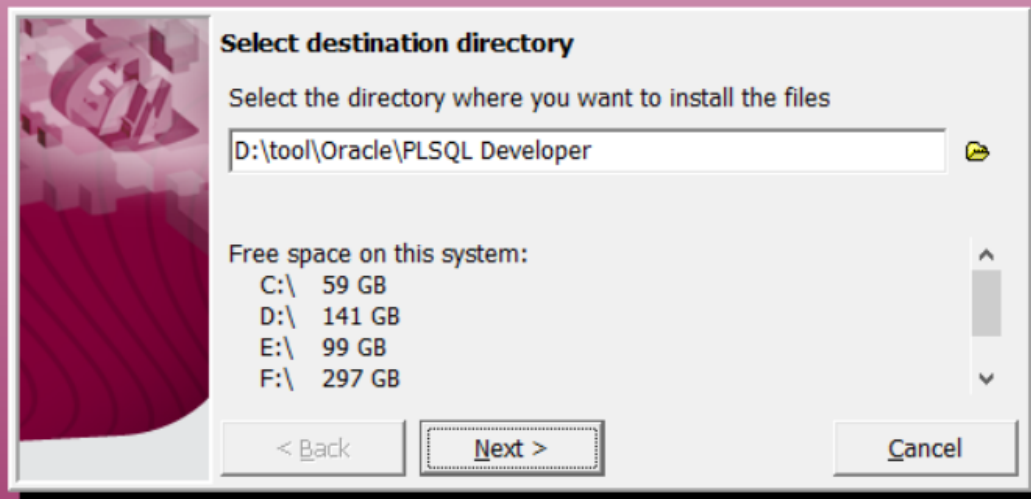
连接到:
Oracle Database 11g Express Edition Release 11.2.0.2.0 - 64bit Production

SQL> _
```

## PLSQL Developer安装

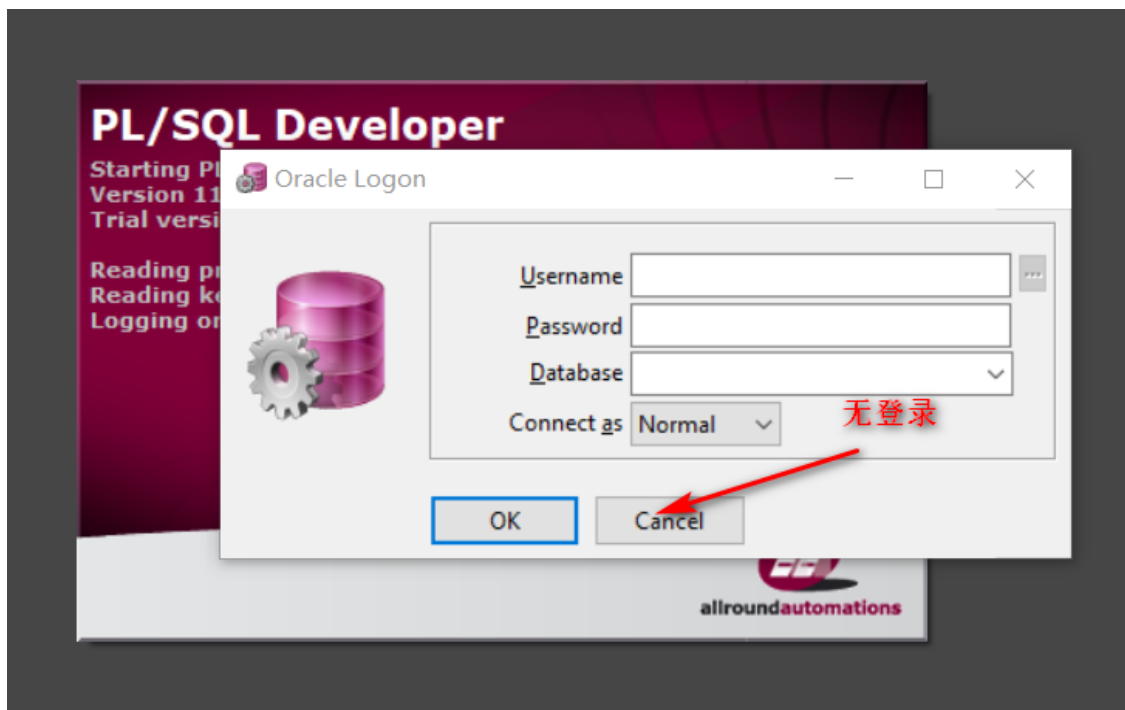
需要先下载instantclient文件: <https://www.oracle.com/database/technologies/instant-client/downloads.html>

修改盘符, 全部默认安装

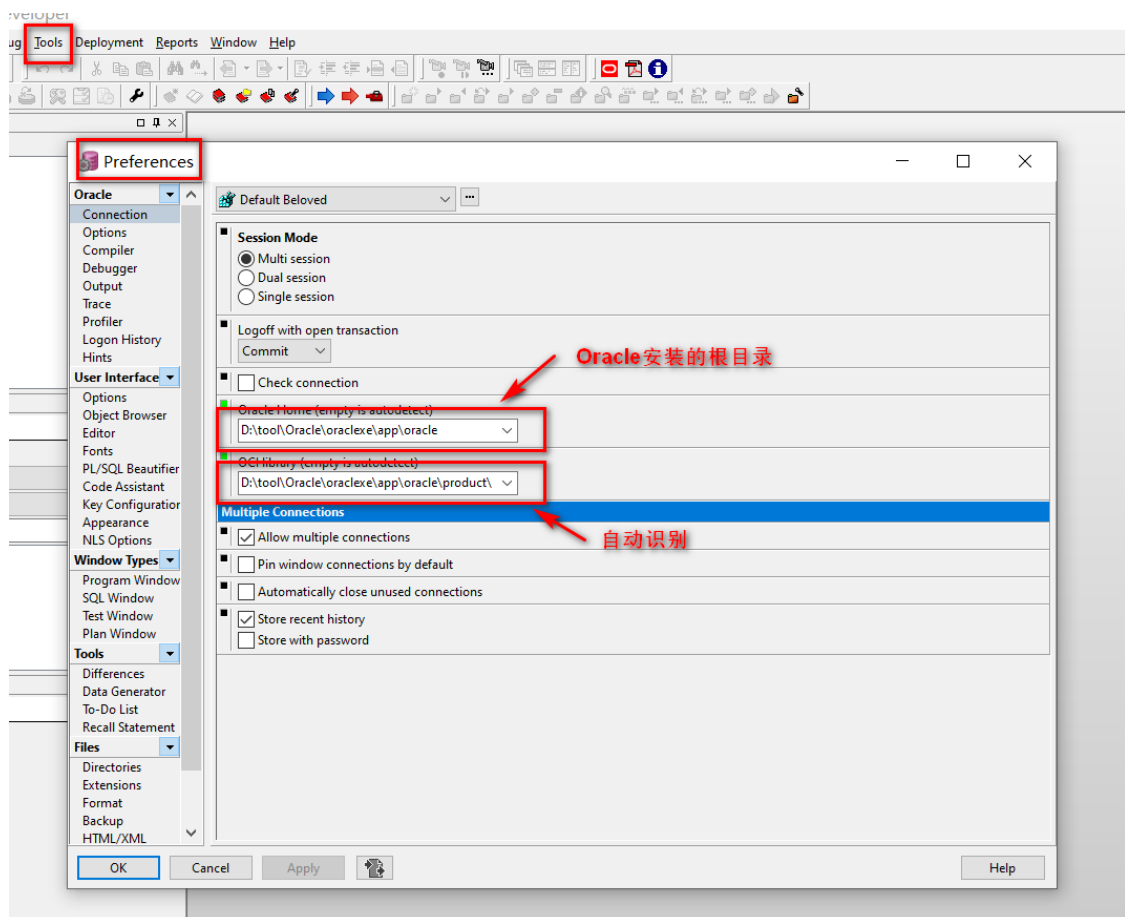


### 3、OracleXE和PLSQL Developer的配置

1. 首次使用PL/SQL，以无用户登录状态打开PL/SQL



点击“Cancel”按钮即可无用户登录，打开后选择“tool”工具的Preference菜单项，修改Oracle Home和OCI Library路径。



2. 将下载的instantclient\_11\_1解压到Oracle的根目录下，在解压的根目录下创建tnsname.ora文件。

名称	修改日期	类型	大小
adrci.sym	2008/10/29 20:09	SYM 文件	5 KB
BASIC_README	2008/10/29 20:09	文件	1 KB
genezi.exe	2008/10/29 20:09	应用程序	46 KB
genezi.sym	2008/10/29 20:09	SYM 文件	26 KB
mfc71.dll	2008/10/15 20:11	应用程序扩展	1,036 KB
msvcr71.dll	2008/10/15 20:11	应用程序扩展	340 KB
oci.dll	2008/10/29 20:06	应用程序扩展	645 KB
oci.sym	2008/10/29 20:06	SYM 文件	417 KB
ocjdbc11.dll	2008/10/8 12:58	应用程序扩展	130 KB
ocjdbc11.sym	2008/10/8 12:58	SYM 文件	23 KB
ociw32.dll	2008/10/27 2:10	应用程序扩展	458 KB
ociw32.sym	2008/10/27 2:10	SYM 文件	72 KB
ojdbc.jar	2008/10/8 12:00	WinRAR 压缩文件	1,846 KB
ojdbc6.jar	2008/10/8 12:00	WinRAR 压缩文件	1,942 KB
orannzsb11.dll	2008/10/15 12:47	应用程序扩展	1,492 KB
orannzsb11.sym	2008/10/15 12:47	SYM 文件	374 KB
oraocci11.dll	2008/10/28 2:41	应用程序扩展	1,299 KB
oraocci11.sym	2008/10/29 20:09	SYM 文件	412 KB
oraociei11.dll	2008/10/29 20:08	应用程序扩展	123,452 KB
oraociei11.sym	2008/10/29 20:08	SYM 文件	7,032 KB
orasql11.dll	2008/10/29 20:09	应用程序扩展	417 KB
orasql11.sym	2008/10/29 20:09	SYM 文件	35 KB
tnsname.ora	2020/2/16 11:52	ORA 文件	1 KB

3. 修改tnsname.ora

```
tnsnames.ora
1 XE
2 (DESCRIPTION =
3   (ADDRESS = (PROTOCOL = TCP)(HOST = 127.0.0.1)(PORT = 1521))
4   (CONNECT_DATA =
5     (SERVER = DEDICATED)
6     (SERVICE_NAME = XE)
7   )
8 )
9
10 HLB_prod =
11 (DESCRIPTION=
12   (ADDRESS=(PROTOCOL=tcp)(HOST=192.168.2.136)(PORT=1521))
13   (CONNECT_DATA=
14     (SID=PROD)
15   )
16 )
```

默认数据库名

传输协议

本地ip地址

端口号

远程连接配置

```
1 XE =
2   (DESCRIPTION =
3     (ADDRESS = (PROTOCOL = TCP)(HOST = 127.0.0.1)(PORT = 1521))
4     (CONNECT_DATA =
5       (SERVER = DEDICATED)
6       (SERVICE_NAME = XE)
7     )
8   )
9
10 HLB_prod =
11   (DESCRIPTION=
12     (ADDRESS=(PROTOCOL=tcp)(HOST=192.168.2.136)(PORT=1521))
13     (CONNECT_DATA=
14       (SID=PROD)
15     )
16   )
```

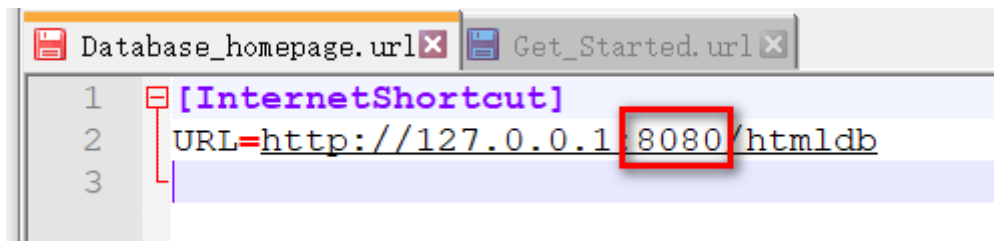
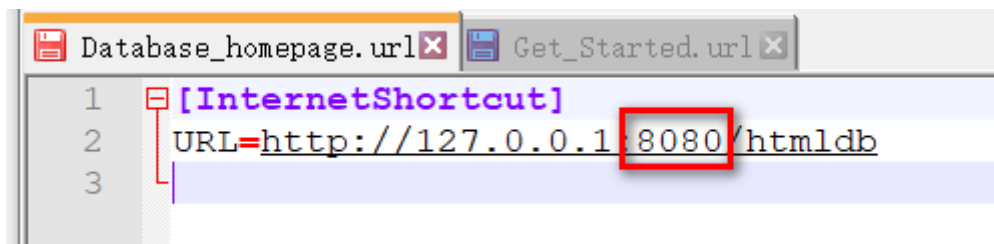
## 4、解决Oracle占用8080端口问题

### 4.1、修改

oraclexe\app\oracle\product\11.2.0\server目录下Database\_homepage和Get\_Started文件

文件 (D:) > tool > Oracle > oraclexe > app > oracle > product > 11.2.0 > server

名称	修改日期	类型	大小
nls	2020/2/16 15:26	文件夹	
oci	2020/2/16 15:26	文件夹	
odbc	2020/2/16 15:26	文件夹	
ode.net	2020/2/16 15:26	文件夹	
odp.net	2020/2/16 15:26	文件夹	
oledb	2020/2/16 15:26	文件夹	
opmn	2020/2/16 15:26	文件夹	
oracore	2020/2/16 15:26	文件夹	
oramts	2020/2/16 15:26	文件夹	
plsql	2020/2/16 15:26	文件夹	
precomp	2020/2/16 15:26	文件夹	
rdbms	2020/2/16 15:26	文件夹	
relnotes	2020/2/16 15:26	文件夹	
slax	2020/2/16 15:26	文件夹	
sqlplus	2020/2/16 15:26	文件夹	
svrm	2020/2/16 15:26	文件夹	
xdk	2020/2/16 15:26	文件夹	
Database_homepage	2020/2/17 11:19	Internet 快捷方式	1 KB
Get_Started	2020/2/17 11:19	Internet 快捷方式	1 KB
Online_forum	2014/5/29 12:06	Internet 快捷方式	1 KB
Online_help	2020/2/16 15:26	Internet 快捷方式	1 KB
Read_Documentation	2014/5/29 12:06	Internet 快捷方式	1 KB
Register	2014/5/29 12:06	Internet 快捷方式	1 KB



## 4.2、使用DBA身份登录Oracle



## 4.3、sql窗口执行以下语句

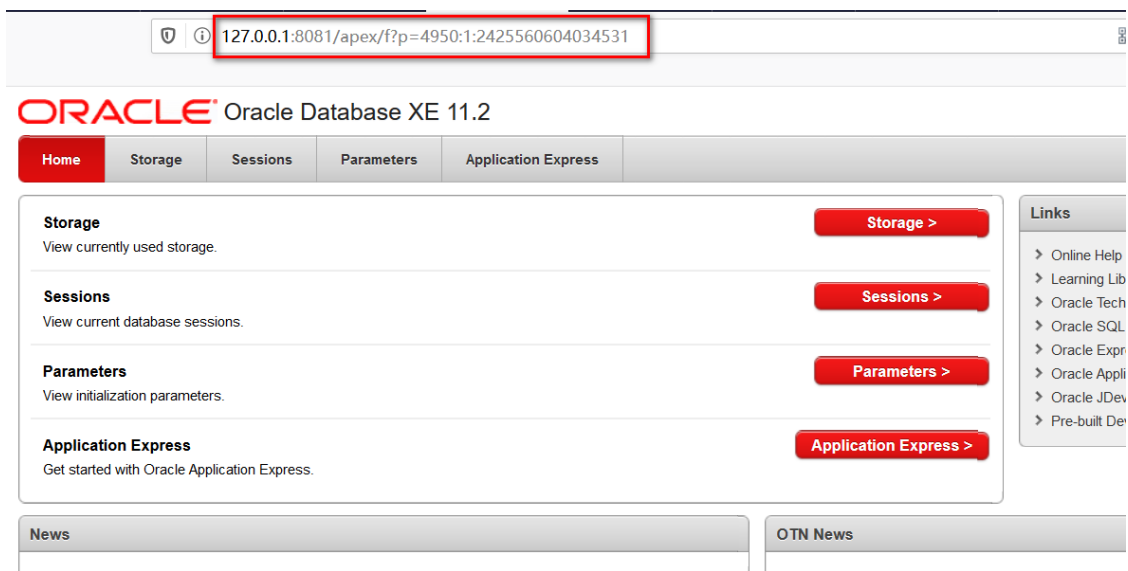
```
1 begin
2 dbms_xdb.sethttpport('8081');
3 dbms_xdb.setftpport('0');
4 end;
```

## 4.4、注意

- 修改完后最好重启对应的服务

OracleJobSchedulerXE	禁用	本地系统
OracleMTSRecoveryService	禁用	本地系统
OracleServiceXE	正在...	自动
OracleXEClrAgent	禁用	本地系统
OracleXETNSListener	正在...	自动

- 重启电脑
- 测试是否修改成功



# 5、Oracle权限

## 5.1、权限分类

- 系统权限：系统规定用户使用数据库的权限。（系统权限是对用户而言）。
- 实体权限：某种权限用户对其它用户的表或视图的存取权限。（是针对表或视图而言的）。

## 5.2、系统权限

DBA: 拥有全部特权，是系统最高权限，只有DBA才可以创建数据库结构。

RESOURCE:拥有Resource权限的用户只可以创建实体，不可以创建数据库结构。

CONNECT:拥有Connect权限的用户只可以登录Oracle，不可以创建实体，不可以创建数据库结构。

**对于普通用户：授予connect, resource权限。**

**对于DBA管理用户：授予connect, resource, dba权限。**



## 常用命令

1.查看用户被赋予了哪些角色。

```
select * from user_role_privs;
```

2.查看当前用户所拥有的全部权限

```
Select * from session_privs;
```

3.查看用户所授予的系统权限

```
Select * from user_sys_privs;
```

4.查看用户所授予的对象权限

```
select * from user_tab_privs;
```

5.系统权限回收：系统权限只能由DBA用户回收

```
Revoke connect, resource from 用户名;
```

## 5.3、实体权限

---

select, update, insert, alter, index, delete, all //all包括所有权限

execute //执行存储过程权限

user01:

```
SQL> grant select, update, insert on product to user02;
```

```
SQL> grant all on product to user02;
```

user02:

```
SQL> select * from user01.product;
```

// 此时user02查user\_tables，不包括user01.product这个表，但如果查all\_tables则可以查到，因为他可以访问。

将表的操作权限授予全体用户：

```
SQL> grant all on product to public; // public表示是所有的用户，这里的all权限不包括drop。
```

实体权限数据字典

```
SQL> select owner, table_name from all_tables; // 用户可以查询的表
```

```
SQL> select table_name from user_tables; // 用户创建的表
```

```
SQL> select grantor, table_schema, table_name, privilege from all_tab_privs; // 获权可以存取的表  
(被授权的)
```

```
SQL> select grantee, owner, table_name, privilege from user_tab_privs; // 授出权限的表(授出的权限)
```

DBA用户可以操作全体用户的任意基表(无需授权，包括删除)

下面就是oracle的权限一览表：

权限	所能实现的操作
分析	
ANALYZE ANY	分析数据库中的任何表、簇或索引
审计	
AUDIT ANY	审计数据库中的任何模式对象
AUDIT SYSTEM	启用与停用语句和特权的审计选项
簇	
CREATE CLUSTER	在自有的模式中创建一个簇
CREATE ANY CLUSTER	在任何一个模式中创建一个簇；操作类似于CREATE ANY TABLE
ALTER ANY CLUSTER	改变数据库中的任何一个簇
DROP ANY CLUSTER	删除数据库中的任何一个簇
数据库	
ALTER DATA BASE	改变数据库；不管操作系统的特权，经由Oracle把文件添加到操作系统中
数据库链接	
CREATE DATABASE LINK	在自有模式中创建专用数据库链接
索引	
CREATE ANY INDEX	在任何表的任何模式中创建一条索引
ALTER ANY INDEX	改变数据库中的任何索引
DROP ANY INDEX	删除数据库中的任何索引
库	
CREATE LIBRARY	在自有模式中创建调出库
CREATE ANY LIBRARY	在任何模式中创建调出库
DROP LIBRARY	删除自有模式中的调出库

权限	所能实现的操作
DROP ANY LIBRARY	删除任何模式中的调出库
特权	
GRANT ANY PRIVILEGE	授予任何系统特权（不包括对象特权）
过程	
CREATE PROCEDURE	在自有模式中创建存储的过程、函数和包
CREATE ANY PROCEDURE	在任何模式中创建存储的过程、函数和包（这要求用户还要有ALTER ANY TABLE 、 BACKUP ANY TA B LE 、 DROP ANY TABLE 、 SELECT ANY TABLE 、 INSERT ANY TABLE 、 UPDATE ANY TABLE 、 DELETE ANY TABLE 或GRANT ANY TABLE 特权
ALTER ANY PROCEDURE	编译任何模式中的任何存储的过程、函数或包
DROP ANY PROCEDURE	删除任何模式中的任何存储的过程、函数或包
EXECUTE ANY PROCEDURE	执行任何过程或函数（独立的或成组的）， 或在任何模式中引用任何包变量
环境资源文件	
CREATE PROFILE	创建环境资源文件
ALTER PROFILE	改变数据库中的任何环境资源文件
DROP PROFILE	删除数据库中的任何环境资源文件
ALTER RESOURCE COST	设置所有的用户会话中使用的资源开销
特权	所能实现的操作
公共数据库链接	
CREATE PUBLIC DATABASE LINK	创建公共数据库链接
DROP PUBLIC DATABASE LINK	删除公共数据库链接

权限	所能实现的操作
公共同义词	
CREATE PUBLIC SYNONYM	创建公共同义词
DROP PUBLIC SYNONYM	删除公共同义词
角色	
CREATE ROLE	创建角色
ALTER ANY ROLE	改变数据库中的任何一个角色
DROP ANY ROLE	删除数据库中的任何一个角色
GRANT ANY ROLE	授权数据库中的任何一个角色
回滚段	
CREATE ROLLBACK SEGMENT	创建回滚段
ALTER ROLLBACK SEGMENT	改变回滚段
DROP ROLLBACK SEGMENT	删除回滚段
会话	
CREATE SESSION	连接到数据库
ALTER SESSION	发出ALTER SESSION 语句
RESTRICTED SESSION	当数据库利用STARTUP RESTRICT 启动时进行连接（OSOPER与 OSDBA角色包含此特权）
序列	
CREATE SEQUENCE	在自有模式中创建序列
CREATE ANY SEQUENCE	在任何模式中创建任何序列

权限	所能实现的操作
ALTER ANY SEQUENCE	在任何模式中改变任何序列
DROP ANY SEQUENCE	在任何模式中删除任何序列
SELECT ANY SEQUENCE	在任何模式中引用任何序列
快照	
CREATE SNAPSHOT	在自有模式中创建快照（用户还必须具有CREATE TABLE 特权）
CREATE ANY SNAPSHOT	在任何模式中创建快照（用户还必须具有CREATE ANY TABLE特权）
ALTER SNAPSHOT	改变任何模式中的任何快照
DROP ANY SNAPSHOT	删除任何模式中的任何快照
同义词	
CREATE SYNONYM	在自有模式中创建同义词
CREATE ANY SYNONYM	在任何模式中创建任何同义词
DROP ANY SYNONYM	在任何模式中删除任何同义词
系统	
ALTER SYSTEM	发出ALTER SYSTEM 语句
表	
CREATE TABLE	在自有模式中创建表。还使被授权者能在自有模式下的表中创建索引，包括那些用于完整性约束的索引（被授权者必须有表空间的定额或UNLIMITED TABLESPACE 特权）
CREATE ANY TABLE	在任何模式中创建表（假如被授权者有CREATE ANY TABLE 特权并在另一个用户模式中创建了一张表，那么拥有者必须在那个表空间上有空间定额。表的拥有者不必具有CREATE [ANY] TABLE 特权）
ALTER ANY TABLE	改变任何模式中的任何表并编译任何模式中的任何视图
BACKUP ANY TABLE	在任何模式中使用表的导出工具执行一个增量导出操作

权限	所能实现的操作
DROP ANY TABLE	删除或截断任何模式中的任何表
LOCK ANY TABLE	锁定任何模式中的任何表或视图
特权	所能实现的操作
COMMENT ANY TABLE	对任何模式中的任何表、视图或列进行注释
SELECT ANY TABLE	对任何模式中的任何表、视图或快照进行查询
INSERT ANY TABLE	把行插入到任何模式中的任何表或视图中
UPDATE ANY TABLE	修改任何模式中的任何表或视图中的行
DELETE ANY TABLE	删除任何模式中的任何表或视图中的行
表空间	
CREATE TABLESPACE	创建表空间；不管用户有何操作系统特权，经由Oracle把文件添加到操作系统中
ALTER TABLESPACE	改变表空间；不管用户有何操作系统特权，经由Oracle把文件添加到操作系统中
MANAGE TABLESPACE	使任何表空间脱机，使任何表空间联机，开始和结束对任何表空间的备份
DROP TABLESPACE	删除表空间
UNLIMITED TABLESPACE	使用任何没有数量限制的表空间。此特权忽略了所分配的任何具体定额。假如被取消的话，被授权者的模式对象仍然保留，但是进一步的表空间分配被拒绝，除非这一分配是具体的表空间定额允许的。此系统特权仅可以授予用户，而不授予角色。一般而言，应分配具体的表空间定额，而不授予此系统特权
事务	
FORCE TRANSACTION	强迫提交或回滚本地数据库中悬而未决的自有的分布式事务
FORCE ANY TRANSACTION	强迫提交或回滚本地数据库中悬而未决的任何分布式事务
触发器	
CREATE TRIGGER	在自有模式中创建触发器

权限	所能实现的操作
CREATE ANY TRIGGER	在任何模式中创建与任何模式的任何表相关的任何触发器
ALTER ANY TRIGGER	启用、停用或编译任何模式中的任何触发器
DROP ANY TRIGGER	删除任何模式中的任何触发器
用户	
CREATE ANY USER	创建用户；分配任意表空间上的定额，设置缺省和临时表空间，指定一个环境资源文件（在CREATE USER 语句中）
BECOME ANY USER	成为另一个用户（这是任何一个执行完全数据库导入的用户所需要的）
ALTER USER	改变其他用户：修改任意用户的口令或验证方法，分配表空间定额，设置缺省或临时表空间，在ALTER USER 语句中指定环境资源文件与缺省角色（不必改变自有口令）
DROP USER	删除另一个用户
视图	
CREATE VIEW	在自有模式中创建视图
CREATE ANY VIEW	在任意模式中创建视图。要在另一个用户模式中创建视图，你必须具有CREATE ANY VIEW 特权，拥用者必须在该视图引用的对象上具有所需的特权

## 6、表空间及用户

### 6.1、问题

- 创建一个表空间：ts100，初始大小10m
- 创建一个账户：以自己姓名首字母缩写，密码123，属于这个表空间
- 给“该账户”授予三中权限
- 用“该账户”创建一张表“students”，包含sid、sname、age、birthday、tel列，只要求有主键约束

### 6.2、答案

```

1  -- 创建表空间
2  -- 语法: create tablespace 表间名 datafile '数据文件的存放位置\数据文件名
   (xxx.dbf)' size 表空间大小
3  create tablespace ts100 datafile 'e:/ts100.dbf' size 10m;
4
5  -- 创建用户并指定表空间
6  -- 语法: create user 用户名 identified by 密码 default tablespace 表空间表;
7  create user zh identified by 123 default tablespace ts100;
8
9  -- 给用户授予权限    connect 登录 resources 资源使用 dba 管理员权限

```

```

10 grant connect,resource,dba to zh;
11
12 -- 查看当前用户
13 show user;
14
15 -- 切换用户
16 -- 语法: conn 用户名/密码
17 conn zh/123;
18
19 -- 查看当前用户
20 show user;
21
22 -- 创建students表
23 create table students(
24     sid number(11,0) primary key,
25     sname varchar2(20),
26     age number(3,0),
27     birthday date,
28     tel varchar(11)
29 );
30
31 -- 插入数据
32 insert into students(sid,sname,age,birthday,tel) values(2018104279,'张恒',
33 20, to_date('2000-03-17','yyyy-MM-dd'),'17609208151');
34
35 -- 查询
36 select * from students;

```

## 6.3、扩展

```

1  -- 1 创建表空间,在D盘下创建一个student的表空间,要求5mb,可以自动增长的
2
3  create tablespace student          --创建表空间
4
5  datafile 'e:\student.dbf'          --指定表空间物理文件的存储位置
6
7  size 5M
8
9  autoextend on;  --自动扩展
10
11 -- 2 创建用户,创建一个stu的用户, 密码是stu,默认的表空间是student,建表t1 (age int)
12
13 create user stu  --用户名
14
15 identified by stu  --密码
16
17 default tablespace student;  --默认表空间
18
19 --创建表t1
20 --为了执行命令,在sqlplus只有以分号结尾才可直接执行
21 create table t1(
22     age int
23 );
24
25 -- 3 授权

```



```

26  -- 授予stu 登陆的权限,授予stu查询scott用户emp表的权限
27  grant connect to stu;  --授权
28
29  alter user scott account unlock;  --解锁
30
31  conn scott/tiger;  --切换用户
32
33  grant all on emp to stu;  --授权,这里的all 代表增删改查都已获取权限
34
35  conn stu/stu;  --切换用户
36
37  select * from scott.emp;  --查询成功
38
39  select * from dba_users; --查看数据库里面所有用户,前提是你是有dba权限的帐号,如
    sys,system
40
41  select * from all_users; --查看你能管理的所有用户!
42
43  select * from user_users; --查看当前用户信息 !
44
45  /*
46  连接: conn system/口令;  conn /as sysdba;  conn scott/tiger;(密码默认为
    tiger,默认未解锁)  conn 用户名/口令;
47
48  查看当前用户: show user;
49
50  删除用户/表/表空间: drop user 用户名/table 表名/tablespace 表空间;
51
52  查看表结构: desc 表名;
53
54  修改密码: create user username identified by newpassword;
55
56  授权: grant connect ,resource,dba to username;
57
58  收权 : revoke connect ,resource,dba from username;
59
60  授予用户创建session的权限,即登陆权限(与connect不同的是session只是一个会话的权限,关闭
    服务就没用了):
61
62  grant create session to 用户名;
63
64  授予用户使用表空间的权限: alter user 用户名 quota unlimited on 表空间;
65  或 alter user 用户名 quota *M on 表空间
66
67  授予创建表的权限: grant create table to username;
68
69  授予用户插入、删除,更新表的权限: grant insert/drop/update table to username;
70
71  注意: 即使以上是以管理员登陆并授权但还会提示权限不够,需要指定 "any"
72
73  grant insert/drop/update any table to username;
74
75  授予用户查看指定表的权限:grant select on tablename to username;
76
77  授予用户查看本用户下所有表的权限:  grant select any table to username;
78
79  授予删除/插入/修改表的权限: grant drop/insert /update on tablename to username;
80

```

```
81 授予对指定表特定字段的插入和修改权限，注意，只能是insert和update：
82
83  grant insert(id)/update(id) on tablename to username;
84
85 授予用户alert任意表的权限：    grant alert all table to username;
86
87 操作用户的表：select * from username.tablename;
88 */
```

## 7、基本数据类型

### 字符类型

数据类型	取值范围	说明
varchar2	0~4000	可变长度的字符串
nvarchar2	0~1000	用来存储unicode字符集的变长字符型数据
char	0~2000	用于描述定长的字符型数据
nchar	0~1000	用来存储unicode字符集的定长字符型数据
long	0~2GB	用来存储变长的字符串

### 数值类型

数据类型	说明
number(p,s)	p最大精度是38位（十进制）p代表的是精度，s代表的是保留小数位数；可以用来存储定长的整数和小数
float	用来存储126位数据（二进制）存储的精度是按二进制计算的，精度范围为二进制的1~126，在转化为二进制时需要乘以0.30103
int	当定义整数类型时，可以直接使用NUMBER的子类型INT，顾名思义：INT用于整型数据。

### 日期类型

数据类型	说明
date	用来存储日期和时间，精确到秒。
timestamp	显示的日期比date更精确，精确到小数秒，还能够显示上午还是下午
Date默认的格式	DD-MM月-YY(18-7月-07)
自定义格式	to_date('2018-03-05','yyyy-mm-dd')

### RAW类型

类型	说明
raw(n)	n为字节数，必须指定n，用于存储二进制数据，q最多能存储 2000 字节
long raw	用于存储可变长度的二进制数据，q最多能存储 2 GB

## LOB大对象类型

**CLOB**：最大为(4GB-1)\*数据库块大小，存储单字节或者多字节字符数据，支持事务处理。主要用于存储大型英文字符

**NCLOB**：最大为(4GB-1)\*数据库块大小，存储Unicode数据。支持事务处理。主要用于存储大型非英文字符

**BLOB**：最大为(4GB-1)\*数据库块大小，存储非结构化二进制数据，支持事务处理。可以被认为是没有字符集语义的比特流，主要存储图像、声音、视频等文件。

**BFILE**：LOB地址指向文件系统上的一个二进制文件，维护目录和文件名。不参与事务处理。只支持只读操作。

## 8、SQL操作符

### 8.1、算数操作符

- 算术操作符用于执行数值计算
- 可以在SQL语句中使用算术表达式，算术表达式由数值数据类型的列名、数值常量和连接它们的算术操作符组成
- 算术操作符包括加(+)、减(-)、乘(\*)、除(/)
- 取余(mod)

```

1  -- 5%2
2  SQL> select mod(5,2) from test01;
3
4  MOD(5,2)
5  -----
6              1
7              1

```

### 8.2、比较操作符

- 比较操作符用于比较两个表达式的值
- 比较操作符包括 =、!=、<、>、<=、>=、BETWEEN...AND、IN、LIKE 和 IS NULL等

### 8.3、逻辑操作符

- 逻辑操作符用于组合多个计较运算的结果以生成一个或真或假的结果。
- 逻辑操作符包括与(AND)、或(OR)和非(NOT)。

### 8.4、集合操作符

- 集合操作符将两个查询的结果组合成一个结果

- 常见集合操作符：MINUS(减去),INTERSECT(交集)和UNION ALL(并集);
- union 操作符返回两个查询的不重复的所有行。

```

1  -- 查询你年龄在50以上的，单个语句查询会有重复的数据
2  SQL> select * from test01 where age > 50
3      union
4      select * from test01 where age > 60;
5
6      ID NAME                AGE
7  -----
8      6 张恒3                60
9      7 张恒3                70

```

- intersect 操作符只返回两个查询的公共行。

```

1  SQL> select * from test01 where age > 50
2      intersect
3      select * from test01 where age > 60;
4
5      ID NAME                AGE
6  -----
7      7 张恒3                70

```

- minus 操作符返回从第一个查询结果中排除第二个查询中出现的行。

```

1  SQL> select * from test01 where age > 50
2      minus
3      select * from test01 where age > 60;
4
5      ID NAME                AGE
6  -----
7      6 张恒3                60

```

## 8.5、连接操作符

- 连接操作符用于将多个字符串或数据值合并成一个字符串
- 通过使用连接操作符可以将表中的多个列合并成逻辑上的一行列

```

1  SQL> select (name||age) 我是一行 from test01;
2
3  我是一行
4  -----
5  张恒20
6  admin20
7  张恒330
8  张恒340
9  张恒350
10 张恒360
11 张恒370

```

## 8.6、伪表，伪列

```
1  -- 伪表    dual    虚拟的表，没有实际意义
2  select sysdate 时间 from dual
3  -- 伪列    rownum 标识查询结果集的序号
4  select rownum,t.* from test01 t
5  -- \\连接符号
6  select 'hello' || 'word' from dual
7  -- mod取余
8  select mod(5,2) from dual
```

## 9、事务

### 9.1、什么是事务

- 事务是用户自定义的一个操作序列，包含一组数据库命令，构成单一逻辑工作单元的操作集合
- 这些操作是不可分割的工作逻辑单元，**是执行并发操作的最小控制单元**
- 这些操作**要么全部成功要么全部失败，是一个不可分割的工作单元**
  - 如果某一事物成功，则在该事务中进行的所有数据更改均会提交，成为数据库中的永久组成部分
  - 如果事务遇到错误且必须取消或者回滚，则所有更改均被清除

### 9.2、事务的特性(ACID)

- 原子性 (atomicity) 。一个事务是一个不可分割的工作单位，事务中包括的操作要么都做，要么都不做。
- 一致性 (consistency) 。事务必须是使数据库从一个一致性状态变到另一个一致性状态。一致性与原子性是密切相关的。
- 隔离性 (isolation) 。一个事务的执行不能被其他事务干扰。即一个事务内部的操作及使用的数据对并发的其他事务是隔离的，并发执行的各个事务之间不能互相干扰。
- 持久性 (durability) 。持久性也称永久性 (permanence) ，指一个事务一旦提交，它对数据库中数据的改变就应该是永久性的。接下来的其他操作或故障不应该对其有任何影响。

### 9.3、Oracle事务

#### 9.3.1、事务控制命令

在Oracle中没有开启事务的处理语句，所有的事务都是隐式开启，用户不得显示使用命令开启事务，Oracle中第一条修改数据的语句或一些要求事务处理的场合都是事务隐式自动开启，**当用户想终止一个事务时，必须显示使用commit和rollback语句结束**

- commit 提交事务
- savepoint 事务保存点(savepoint a)
- rollback(to) 回滚【回滚到定义的保存点】rollback to a
- set transaction 设置事务的属性

#### 9.3.2、事务的结束

下列情况Oracle会认为一个事务的结束

- commit
- rollback：如果事务中使用了存储点，则只取消存储点后的事务处理，而且事务并不会终止
- ddl：执行时(意味着前面的DML操作已经commit)

- 用户断开连接时(disconn)
- 用户进程意外，这时用户当前的事务被回滚

### 9.3.3、事务保存点

- savepoint: 保存点是事务中的一点。用于取消部分事务，当结束事务时，会自动删除该事务所定义的所有保存点。当执行rollback时，通过指定保存点可以回退到指定的点
- 事务的重要操作
  1. 设置保存点      savepoint a
  2. 取消部分事务      rollback to a
  3. 取消全部事务      rollback

## 9.4、练习

```

1  -- 查询表中数据
2  SQL> select * from test01;
3
4          ID NAME                AGE
5  -----
6          1 张恒                  20
7          2 admin                  20
8
9  -- 添加记录，此时自动隐式开启事务
10 SQL> insert into test01(id,name,age) values(3,'张恒3',30);
11
12 1 row inserted
13
14 -- 添加成功，此时数据存储在磁盘里，没有提交事务，并没有存储到数据库中
15 SQL> select * from test01;
16
17          ID NAME                AGE
18  -----
19          1 张恒                  20
20          2 admin                  20
21          3 张恒3                  30
22
23 -- 回滚 取消全部事务，这时事务自动关闭
24 SQL> rollback;
25
26 Rollback complete
27
28 -- 再次查询，刚插入的数据没有，已经回滚了
29 SQL> select * from test01;
30
31          ID NAME                AGE
32  -----
33          1 张恒                  20
34          2 admin                  20
35
36 -- 插入数据 事务隐式自动开启
37 SQL> insert into test01(id,name,age) values(3,'张恒3',30);
38
39 1 row inserted
40
41 -- 定义保存点a
42 SQL> savepoint a;

```

```

42
43 Savepoint created
44
45 -- 插入数据
46 SQL> insert into test01(id,name,age) values(4,'张恒4',40);
47
48 1 row inserted
49
50 -- 定义保存点b
51 SQL> savepoint b;
52
53 Savepoint created
54
55 -- 插入数据
56 SQL> insert into test01(id,name,age) values(5,'张恒5',50);
57
58 1 row inserted
59
60 -- 定义保存点c
61 SQL> savepoint c;
62
63 Savepoint created
64
65 -- 回滚到保存点b，事务自动关闭，保存点b后面的数据被回滚
66 SQL> rollback to b;
67
68 Rollback complete
69
70 -- 查询数据
71 SQL> select * from test01;
72
73          ID NAME                AGE
74  -----
75          1 张恒                  20
76          2 admin                 20
77          3 张恒3                 30
78          4 张恒4                 40
79
80 -- 插入数据，事务隐式自动开启
81 SQL> insert into test01(id,name,age) values(5,'张恒5',50);
82
83 1 row inserted
84
85 -- 定义保存点a
86 SQL> savepoint a;
87
88 Savepoint created
89
90 -- 插入数据
91 SQL> insert into test01(id,name,age) values(6,'张恒6',60);
92
93 1 row inserted
94
95 -- 定义保存点b
96 SQL> savepoint b;
97
98 Savepoint created
99

```

```
100  -- 回滚全部事务
101  SQL> rollback;
102
103  Rollback complete
104
105  -- 查询数据
106  SQL> select * from test01;
107
108           ID NAME                AGE
109  -----
110           1 张恒                  20
111           2 admin                 20
112           3 张恒3                 30
113           4 张恒4                 40
114
115  SQL>
```

# 10、SCOTT系统表练习

## 10.1、介绍

Oracle中SCOTT用户密码tiger下 emp、dept、bonus、salgrade表

源码：D:\tool\Oracle\oraclexe\app\oracle\product\11.2.0\server\rdbms\admin\utlsampl.sql

**emp 雇员表**

属性英文名	属性中文名
empno	雇员编号
ename	雇员姓名
job	雇员职位
mgr	雇员对应的领导的编号
hiredate	雇员的雇佣日期
sal	雇员的基本工资
comm	奖金或补助
deptno	所在部门

**dept 部门表**

属性英文名	属性中文名
deptno	部门编号
dname	部门名称
loc	部门所在位置

**salgrade 工资等级表**



属性英文名	属性中文名
grade	工资等级
losal	此等级的最低工资
hisal	此等级的最高工资

### bonus 工资表

属性英文名	属性中文名
ename	雇员名字
job	雇员职位
sal	雇员工资
comm	雇员资金

## 10.2、练习

```

1  -- Oracle练习的三个表emp,dept,salgrade,bonus在scott用户下
2  -- emp员工表(empno员工号/ename员工姓名/job工作/mgr上级编号/
3  --      hiredate受雇日期/sal薪金/comm佣金/deptno部门编号)
4  -- dept部门表(deptno部门编号/dname部门名称/loc部门所在位置)
5  -- salgrade工资等级表(grade工资等级/losal此等级的最低工资/hisal此等级的最高工资)
6  -- bonus 工资表(ename雇员名字/job雇员职位/sal雇员工资/comm雇员资金)
7  -- 1.显示部门号为10 的部门名、员工名和工资。
8  select d.dname 部门名,e.ename 员工名,e.sal 工资
9  from scott.emp e,scott.dept d
10 where e.deptno = d.deptno and e.deptno = 10;
11 -- 2.显示各个员工的姓名,工资,及其工资的级别
12 select e.ename 姓名,e.sal 工资,s.grade 工资级别
13 from scott.emp e,scott.salgrade s
14 where e.sal between s.losal and s.hisal
15 -- 3.显示员工BLAKE 的上级领导的姓名。
16 -- BLAKE 的上级领导的编号
17 select e.mgr from scott.emp e where e.ename = 'BLAKE';
18
19 select e.ename
20 from scott.emp e
21 where e.empno = (select e.mgr from scott.emp e where e.ename = 'BLAKE');
22 -- 4.显示与scott同一部门的所有员工。
23 select deptno from scott.emp where ename = 'SCOTT'
24
25 select *
26 from scott.emp e
27 where e.deptno = (select deptno from scott.emp where ename = 'SCOTT')
28
29
30 -- 5. 显示工资比部门 30 的所有员工的工资高的员工的姓名、工资和部门号。
31 -- 查询部门30最高的工资
32 select max(sal) from scott.emp e where e.deptno = 30;
33
34 select e.ename 姓名,e.sal 工资,e.deptno 部门号

```

```

35 from scott.emp e
36 where e.sal > (select max(sal) from scott.emp e where e.deptno = 30)
37 -- 6. 查询与smith 部门和岗位完全相同的所有雇员。
38 -- 查询SMITH的部门和岗位
39 select d.deptno deptno,d.loc loc
40 from scott.emp e,scott.dept d
41 where e.ename = 'SMITH' and e.deptno = d.deptno
42
43 select e.*
44 from scott.emp e,scott.dept d,(select d.deptno deptno,d.loc loc from
scott.emp e,scott.dept d where e.ename = 'SMITH' and e.deptno = d.deptno)
d2
45 where e.deptno = d.deptno and d2.deptno = d.deptno and d2.loc = d.loc
46 -- 7. 显示高于部门平均工资的员工的信息。
47 -- 查询每个部门的平均工资
48 select deptno deptno,avg(sal) avgсал from scott.emp group by deptno
49
50 select e.*,avgs.avgсал
51 from scott.emp e,(select deptno deptno,avg(sal) avgсал from scott.emp group
by deptno) avgs
52 where e.deptno = avgs.deptno and e.sal > avgs.avgсал
53 -- 8. 查询部门工资总和高于员工工资总和1/3的部门名及工资总和
54 -- 查询每个部门工资总和
55 select deptno, sum(sal) dsum from scott.emp group by deptno
56 -- 查询部门名
57 select d2.deptno,d.dname,d2.dsum from scott.dept d,(select deptno, sum(sal)
dsum from scott.emp group by deptno) d2 where d.deptno = d2.deptno
58
59 select d3.dname,d3.dsum
60 from (select d2.deptno,d.dname,d2.dsum from scott.dept d,(select deptno,
sum(sal) dsum from scott.emp group by deptno) d2 where d.deptno =
d2.deptno) d3
61 where d3.dsum > (select sum(sal)/3 from scott.emp)

```

## 11、常用函数

### 11.1、日期函数

日期函数对日期值进行运算，并生成日期数据类型或数值类型的结果

- sysdate: 查询当前时间

```

1 SQL> select sysdate from dual;
2
3 SYSDATE
4 -----
5 2020/2/20 10:44:55

```

- add\_months(d,): 日期d, 前后第几个月

```

1 SQL> select add_months(sysdate,-1) from dual;
2
3 ADD_MONTHS(SYSDATE,-1)
4 -----
5 2020/1/20 10:50:09
6
7 SQL> select add_months(sysdate,1) from dual;
8
9 ADD_MONTHS(SYSDATE,1)
10 -----
11 2020/3/20 10:50:24

```

- months\_between(sysdate,datetime): 两个时间相差几个月

```

1 SQL> select months_between(sysdate,to_date('2020-01-15','yyyy-mm-dd'))
2 from dual;
3 MONTHS_BETWEEN(SYSDATE,TO_DATE('2020-01-15','YYYY-MM-DD'))
4 -----
5 1.17593525985663

```

- last\_day(d): 当前月份的最后一天

```

1 SQL> select last_day(sysdate) from dual;
2
3 LAST_DAY(SYSDATE)
4 -----
5 2020/2/29 10:57:0
6
7 SQL> select last_day(to_date('2020-01-15','yyyy-mm-dd')) from dual;
8
9 LAST_DAY(TO_DATE('2020-01-15','YYYY-MM-DD'))
10 -----
11 2020/1/31

```

- next\_day(d,day): 当前时间的下一个星期几的时间。day: 参数可以是1-7, 星期天-星期六

```

1 SQL> select next_day(sysdate,1) from dual;
2
3 NEXT_DAY(SYSDATE,1)
4 -----
5 2020/2/23 11:01:44

```

## 11.2、Round, trunc函数

- Round(number,[decimals]): 截取数字
  - number: 待做截取处理的数值
  - decimals: 指明要保留小数点后面的位数。忽略所截去的小数点部分, 并四舍五入。如果是负数则表示从小数点开启左边的位数, 相应整数数字用0填充, 小数被去掉

```

1 SQL> select round(12345.6789,2) from dual;
2
3 ROUND(12345.6789,2)

```

```

4  -----
5      12345.68
6
7  SQL> select round(12345.6789,0) from dual;
8
9  ROUND(12345.6789,0)
10 -----
11      12346
12
13 SQL> select round(12345.6789,-2) from dual;
14
15 ROUND(12345.6789,-2)
16 -----
17      12300
18 SQL> select round(12345.6789) from dual;
19
20 ROUND(12345.6789)
21 -----
22      12346

```

- `trunc(number,[decimals])`
  - 和round用法一样, **没有四舍五入**

```

1  SQL> select trunc(12345.6789,2) from dual;
2
3  TRUNC(12345.6789,2)
4  -----
5      12345.67
6
7  SQL> select trunc(12345.6789,0) from dual;
8
9  TRUNC(12345.6789,0)
10 -----
11      12345
12
13 SQL> select trunc(12345.6789,-2) from dual;
14
15 TRUNC(12345.6789,-2)
16 -----
17      12300
18
19 SQL> select trunc(12345.6789) from dual;
20
21 TRUNC(12345.6789)
22 -----
23      12345

```

## 11.3、对日期进行Round

可以使用ROUND 和TRUNC 函数对日期四舍五入。

1	CC	四舍五入年的后两位
2	SCC	四舍五入年的后两位
3	SYYYY	四舍五入年
4	YYYY	四舍五入年

5	YEAR	四舍五入年
6	SYEAR	四舍五入年
7	YYY	四舍五入年
8	YY	四舍五入年
9	Y	四舍五入年
10	IYYY	四舍五入ISO年
11	IY	四舍五入ISO年
12	IY	四舍五入ISO年
13	I	四舍五入ISO年
14	Q	四舍五入季度
15	MONTH	四舍五入月
16	MON	四舍五入月
17	MM	四舍五入月
18	RM	四舍五入月
19	WW	四舍五入每年的第一个星期
20	IW	四舍五入每月的第一个ISO星期
21	W	四舍五入每月的第一个星期
22	DDD	四舍五入日
23	DD	四舍五入日
24	J	四舍五入日
25	DAY	四舍五入日每星期的第一个天
26	DY	四舍五入日每星期的第一个天
27	D	四舍五入日每星期的第一个天
28	HH	四舍五入小时
29	HH12	四舍五入小时
30	HH24	四舍五入小时
31	MI	四舍五入分钟

```

1  -- 对年份四舍五入
2  SQL> select round(to_date('1984-07-01', 'yyyy-mm-dd'), 'YYYY') from dual;
3
4  ROUND(TO_DATE('1984-07-01', 'YYYY-MM-DD'), 'YYYY')
5  -----
6  1985/1/1

```

## 11.4、字符函数

- `initcap(char)` : 首字母大写, 其余全部小写

```

1  SQL> select initcap('saAXzax') from dual;
2
3  INITCAP('SAAXZAX')
4  -----
5  Saaxzax

```

- `lower(char)` : 全部小写

```

1  SQL> select lower('saAXzax') from dual;
2
3  LOWER('SAAXZAX')
4  -----
5  saaxzax

```

- `upper(char)` : 全部大写

```

1 SQL> select upper('saAXzax') from dual;
2
3 UPPER('SAAXZAX')
4 -----
5 SAAXZAX

```

- `ltrim(c1,c2)`：函数是按照c2中的字符一个一个的截断c1的字符，而且还是从左开始执行的，一旦遇到c2中的字符，c1中的字符都会相对应的截断，一直到c1的字符没有c2的字符为止才会结束。

```

1 SQL> select ltrim('ZXCvbnmASD','ZXC') from dual;
2
3 LTRIM('ZXCvbnmASD','ZXC')
4 -----
5 vbnmASD
6
7 SQL> select ltrim('ZXCvbnmASD','XC') from dual;
8
9 LTRIM('ZXCvbnmASD','XC')
10 -----
11 ZXCvbnmASD
12 SQL> select ltrim('abbcbddcc','abc') from dual;
13
14 LTRIM('ABBCBDDCC','ABC')
15 -----
16 ddcc

```

- `rtrim(c1,c2)`：删除右边出现的字符串，和ltrim用呀一样

```

1 SQL> select rtrim('abbcbddcc','abc') from dual;
2
3 RTRIM('ABBCBDDCC','ABC')
4 -----
5 abbcbdd

```

- `lpad(string, padded_length, [pad_string])`：在列的左边粘贴字符
  - `string`：准备被填充的字符串
  - `padded_length`：填充之后的字符串长度，也就是该函数返回的字符串长度，如果这个数量比原字符串的长度要短，lpad函数将会把字符串截取成从左到右的n个字符
  - `pad_string`：填充字符串，是个可选参数，这个字符串是要粘贴到string的左边，如果这个参数未写，lpad函数将会在string的左边粘贴空格

```

1 SQL> select lpad('abc',10,'x') from dual;
2
3 LPAD('ABC',10,'X')
4 -----
5 XXXXXXXXabc
6
7 SQL> select lpad('abc',2,'x') from dual;
8
9 LPAD('ABC',2,'X')
10 -----
11 ab
12
13 SQL> select lpad('abc',10,'xyx') from dual;

```

```

14
15 LPAD('ABC',10,'XYX')
16 -----
17 XyXyYxXabc

```

- `rpads ( string, padded_length, [ pad_string ] )` : 在列的右边粘贴字符。与pad相反
- `replace(char, searchstring,[rep string])` : 替换字符

```

1 SQL> select replace('abc','c','123') from dual;
2
3 REPLACE('ABC','C','123')
4 -----
5 ab123
6

```

- `substr (char, m, n)` : 从第m个字符开始, 保留n个, 如果m是负数, 就从右边开始

```

1 SQL> select substr('abcder',3,5) from dual;
2
3 SUBSTR('ABCDEF',3,5)
4 -----
5 cder
6
7 SQL> select substr('abcder',3,1) from dual;
8
9 SUBSTR('ABCDEF',3,1)
10 -----
11 c
12
13 SQL> select substr('abcder',-1,1) from dual;
14
15 SUBSTR('ABCDEF',-1,1)
16 -----
17 r

```

- `concat (expr1, expr2)` : 合并字符串

```

1 SQL> select concat('aaa','bbb') from dual;
2
3 CONCAT('AAA','BBB')
4 -----
5 aaabbb

```

- `length` : 字符串长度

```

1 SQL> select length('aaa') from dual;
2
3 LENGTH('AAA')
4 -----
5 3

```

## 11.5、数字函数

- `abs(n)`: 绝对值

```

1 SQL> select abs(-10) from dual;
2
3     ABS(-10)
4 -----
5           10

```

- `ceil(n)`: 向上取整

```

1 SQL> select ceil(-10.12) from dual;
2
3     CEIL(-10.12)
4 -----
5           -10
6
7 SQL> select ceil(10.12) from dual;
8
9     CEIL(10.12)
10 -----
11           11

```

- `floor(n)`: 向下取整

```

1 SQL> select floor(10.56) from dual;
2
3     FLOOR(10.56)
4 -----
5           10
6
7 SQL> select floor(-10.56) from dual;
8
9     FLOOR(-10.56)
10 -----
11          -11

```

- `power(m,n)`: 计算m的n次方

```

1
2 SQL> select power(4,0) from dual;
3
4     POWER(4,0)
5 -----
6           1
7
8 SQL> select power(2,3) from dual;
9
10    POWER(2,3)
11 -----
12           8

```

- `mod(m,n)`: 取余



```

1 SQL> select mod(4,3) from dual;
2
3 MOD(4,3)
4 -----
5          1

```

- round(m,n): 截取数字, 并四舍五入

```

1 SQL> select round(4.32141,3) from dual;
2
3 ROUND(4.32141,3)
4 -----
5          4.321

```

- sqrt(n): 开平方根

```

1 SQL> select sqrt(16) from dual;
2
3 SQRT(16)
4 -----
5          4

```

## 11.6、转换函数

- to\_char: 将日期转换字符串

```

1 SQL> select to_char(sysdate, 'yyyy"年"mm"月"dd"日"') from dual;
2
3 TO_CHAR(SYSDATE, 'YYYY"年"MM"月"DD"日"')
4 -----
5 2020年02月20日

```

- to\_date: 将字符串转换为日期

```

1 SQL> select to_date('2020-02-20', 'yyyy-mm-dd') from dual;
2
3 TO_DATE('2020-02-20', 'YYYY-MM-DD')
4 -----
5 2020/2/20

```

## 11.7、分组函数

- avg: 平均值
- min: 最小
- max: 最大
- sum: 和
- count: 统计
- group by: 用于将信息划分为更小的组, 每一组行返回针对该组的单个结果
- having: 用于指定 GROUP BY 子句检索行的条件

## 11.8、分析函数

- over(partition by ... order by ...)
- over()在什么条件之上
- partition by 按哪个字段划分组;
- order by 按哪个字段排序;

## 排序

- asc: 正序。由小到大
- desc: 倒序。由大到小
- row\_number: 返回连续的排位, 不论值是否相等

```

1 SQL> select row_number() over(order by e.sal) PX,e.* from scott.emp e;
2
3          PX EMPNO ENAME          JOB          MGR HIREDATE          SAL
4  COMM DEPTNO
5  -----
6      1  7369 SMITH          CLERK          7902 1980/12/17      800.00
7      20
8      2  7900 JAMES          CLERK          7698 1981/12/3       950.00
9      30
10     3  7876 ADAMS          CLERK          7788 1987/5/23      1100.00
11     20
12     4  7521 WARD           SALESMAN       7698 1981/2/22      1250.00
13  500.00 30
14     5  7654 MARTIN         SALESMAN       7698 1981/9/28      1250.00
15 1400.00 30
16     6  7934 MILLER         CLERK          7782 1982/1/23      1300.00
17     10
18     7  7844 TURNER         SALESMAN       7698 1981/9/8       1500.00
19  0.00 30
20     8  7499 ALLEN           SALESMAN       7698 1981/2/20      1600.00
21 300.00 30
22     9  7782 CLARK           MANAGER        7839 1981/6/9       2450.00
23     10
24    10  7698 BLAKE           MANAGER        7839 1981/5/1       2850.00
25     30
26    11  7566 JONES           MANAGER        7839 1981/4/2       2975.00
27     20
28    12  7902 FORD            ANALYST        7566 1981/12/3      3000.00
29     20
30    13  7788 SCOTT           ANALYST        7566 1987/4/19      3000.00
31     20
32    14  7839 KING            PRESIDENT      1981/11/17      5000.00
33     10
34
35 14 rows selected

```

- rank: 具有相等值的行排位相同, 序数随后跳跃

```

1 SQL> select rank() over(order by e.sal) PX,e.* from scott.emp e;
2
3          PX EMPNO ENAME          JOB          MGR HIREDATE          SAL
4  COMM DEPTNO

```

4	-----							
5		1	7369	SMITH	CLERK	7902	1980/12/17	800.00
6		20						
6		2	7900	JAMES	CLERK	7698	1981/12/3	950.00
7		30						
7		3	7876	ADAMS	CLERK	7788	1987/5/23	1100.00
8		20						
8		4	7521	WARD	SALESMAN	7698	1981/2/22	1250.00
9	500.00	30						
9		4	7654	MARTIN	SALESMAN	7698	1981/9/28	1250.00
10	1400.00	30						
10		6	7934	MILLER	CLERK	7782	1982/1/23	1300.00
11		10						
11		7	7844	TURNER	SALESMAN	7698	1981/9/8	1500.00
12	0.00	30						
12		8	7499	ALLEN	SALESMAN	7698	1981/2/20	1600.00
13	300.00	30						
13		9	7782	CLARK	MANAGER	7839	1981/6/9	2450.00
14		10						
14		10	7698	BLAKE	MANAGER	7839	1981/5/1	2850.00
15		30						
15		11	7566	JONES	MANAGER	7839	1981/4/2	2975.00
16		20						
16		12	7902	FORD	ANALYST	7566	1981/12/3	3000.00
17		20						
17		12	7788	SCOTT	ANALYST	7566	1987/4/19	3000.00
18		20						
18		14	7839	KING	PRESIDENT		1981/11/17	5000.00
19		10						
20	14 rows selected							

- dense\_rank: 具有相等值的行排位相同, 序号是连续的

1	SQL> select dense_rank() over(order by e.sal) PX,e.* from scott.emp e;							
2								
3		PX	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL
	COMM	DEPTNO						
4	-----							
	-----							
5		1	7369	SMITH	CLERK	7902	1980/12/17	800.00
		20						
6		2	7900	JAMES	CLERK	7698	1981/12/3	950.00
		30						
7		3	7876	ADAMS	CLERK	7788	1987/5/23	1100.00
		20						
8		4	7521	WARD	SALESMAN	7698	1981/2/22	1250.00
	500.00	30						
9		4	7654	MARTIN	SALESMAN	7698	1981/9/28	1250.00
	1400.00	30						
10		5	7934	MILLER	CLERK	7782	1982/1/23	1300.00
		10						
11		6	7844	TURNER	SALESMAN	7698	1981/9/8	1500.00
	0.00	30						
12		7	7499	ALLEN	SALESMAN	7698	1981/2/20	1600.00
	300.00	30						

13	8	7782	CLARK	MANAGER	7839	1981/6/9	2450.00
	10						
14	9	7698	BLAKE	MANAGER	7839	1981/5/1	2850.00
	30						
15	10	7566	JONES	MANAGER	7839	1981/4/2	2975.00
	20						
16	11	7902	FORD	ANALYST	7566	1981/12/3	3000.00
	20						
17	11	7788	SCOTT	ANALYST	7566	1987/4/19	3000.00
	20						
18	12	7839	KING	PRESIDENT		1981/11/17	5000.00
	10						
19							
20	14 rows selected						

## 12、同义词

**Oracle synonym** 同义词是数据库当前用户通过给另外一个用户的对象创建一个别名，然后可以通过对别名进行查询和操作，等价于直接操作该数据库对象。Oracle同义词常常是给表、视图、函数、过程、包等制定别名，可以通过CREATE 命令进行创建、ALTER 命令进行修改、DROP 命令执行删除操作。

### 12.1、同义词的优点

- 简化SQL语句
- 隐藏对象的名称和所有者
- 提供对对象的公共访问

### 12.2、类型

- 公有同义词：只能具有DBA用户才能进行创建，所有用户都可以访问的。
- 私有同义词：只能当前用户可以访问，前提：当前用户具有create synonym 权限。

### 12.3、创建

**普通用户不能创建同义词，同义词的创建只能由管理员创建，或者拥有dba权限的用户创建**

**普通用户创建同义词，需要dba用户授权，同义词创建权限**

#### 13.3.1、dba用户创建

**语法结构：**

```
CREATE [OR REPLACE] [PUBLIC] SYNONYM [当前用户.]synonym_name FOR [其他用户.]object_name;
```

**解析：**

1、create [or replace] 命令create建表命令一样，当当前用户下同义词对象名已经存在的时候，就会删除原来的同义词，用新的同义词替代上。

2、[public]：创建的是公有同义词，在实际开发过程中比较少用，因为创建就代表着任何用户都可以通过自己用户访问操作该对象，一般我们访问其他用户对象时，需要该用户进行授权给我们。

3、用户名.object\_name：oracle用户对象的权限都是自己用户进行管理的，需要其他用户的某个对象的操作权限，只能通过对象拥有者（用户）进行授权给当前用户。或者当前用户具有系统管理员权限（DBA），即可通过用户名.object\_name操作该对象。

```

1  -- dba用户创建私有同义词
2  create synonym zh.emp for Beloved.emp;
3
4  -- dba用户创建共有同义词
5  create public synonym emp for Beloved.emp;

```

### 13.3.2、普通用户创建

dba用户授权

语法: grant create [public] synonym to 用户名

普通用户创建同义词

语法: create [public] synonym 同义词名 for 用户.表名;

```

1  -- dba用户授权私有同义词权限
2  grant create synonym to zh;
3
4  -- dba用户授权共有同义词权限
5  grant create public synonym to zh;
6
7  -- 普通用户创建私有同义词
8  create synonym emp for Beloved.emp;
9
10 -- 普通用户创建共有同义词
11 create public synonym emp for Beloved.emp;

```

## 12.4、删除

同义词删除只能通过同义词拥有者的用户或者具有DBA权限的用户才能删除。

语法结构:

DROP [PUBLIC] SYNONYM [用户.]synonym\_name;

### 12.4.1、普通用户删除

普通用户只能删除自己的私有同义词

```

1  -- 普通用户删除私有同义词
2  drop synonym emp;

```

### 12.4.2、dba用户删除

dba用户可以删除所有用户的私有同义词，也可以删除共有同义词

```

1  -- dba用户删除私有同义词
2  drop synonym zh.emp;
3
4  -- dba用户删除共有同义词
5  drop public synonym emp;

```

### 12.4.3、dba用户删除普通用户创建同义词权限

```
1  -- 删除创建私有同义词权限
2  revoke create synonym from zh;
3
4  -- 删除创建共有同义词权限
5  revoke create public synonym from zh;
```

## 13、序列

### 13.1、序列定义

**序列(SEQUENCE)是序列号生成器，可以为表中的行自动生成序列号，产生一组等间隔的数值(类型为数字)。不占用磁盘空间，占用内存。其主要用途是生成表的主键值，可以在插入语句中引用，也可以通过查询检查当前值，或使序列增至下一个值。**

### 13.2、创建序列

**创建序列需要CREATE SEQUENCE系统权限。**

序列的创建语法如下：

```
CREATE SEQUENCE 序列名

[INCREMENT BY n]

[START WITH n]

[{MAXVALUE/ MINVALUE n | NOMAXVALUE}]

[{CYCLE | NOCYCLE}]

[{CACHE n | NOCACHE}];
```

其中：

- 1) **INCREMENT BY**用于定义序列的步长，如果省略，则默认为1，如果出现负值，则代表Oracle序列的值是按照此步长递减的。
- 2) **START WITH** 定义序列的初始值(即产生的第一个值)，默认为1。
- 3) **MAXVALUE** 定义序列生成器能产生的最大值。选项NOMAXVALUE是默认选项，代表没有最大值定义，这时对于递增Oracle序列，系统能够产生的最大值是10的27次方;对于递减序列，最大值是-1。
- 4) **MINVALUE**定义序列生成器能产生的最小值。选项NOMAXVALUE是默认选项，代表没有最小值定义，这时对于递减序列，系统能够产生的最小值是-10的26次方;对于递增序列，最小值是1。
- 5) **CYCLE**和**NOCYCLE** 表示当序列生成器的值达到限制值后是否循环。**CYCLE**代表循环，**NOCYCLE**代表不循环。如果循环，则当递增序列达到最大值时，循环到最小值;对于递减序列达到最小值时，循环到最大值。如果不循环，达到限制值后，继续产生新值就会发生错误。
- 6) **CACHE(缓冲)**定义存放序列的内存块的大小，默认为20。NOCACHE表示不对序列进行内存缓冲。对序列进行内存缓冲，可以改善序列的性能。

大量语句发生请求，申请序列时，为了避免序列在运用层实现序列而引起的性能瓶颈。Oracle序列允许将序列提前生成cache x个先存入内存，在发生大量申请序列语句时，可直接到运行最快的内存中去得到序列。但cache个数也不能设置太大，因为在数据库重启时，会清空内存信息，预存在内存中的序列会丢失，当数据库再次启动后，序列从上次内存中最大的序列号+1 开始存入cache x个。这种情况也会在数据库关闭时也会导致序号不连续。

7) **NEXTVAL** 返回序列中下一个有效的值, 任何用户都可以引用。

8) **CURRVAL** 中存放序列的当前值NEXTVAL 应在CURRVAL 之前指定, 二者应同时有效。

```
1  -- 查看用户是否有create sequence权限
2  select * from session_privs;
3
4  -- 创建升序序列
5  create sequence seq1
6  start with 1
7  increment by 1;
8
9  -- 创建降序序列
10 create sequence seq2
11 start with -1
12 increment by -1;
13
14 SQL> select seq1.currval,seq1.nextval from dual;
15
16      CURRVAL      NEXTVAL
17  -----
18           1           1
19
20 SQL> select seq1.currval,seq1.nextval from dual;
21
22      CURRVAL      NEXTVAL
23  -----
24           2           2
25
26 SQL> select seq2.currval,seq2.nextval from dual;
27
28      CURRVAL      NEXTVAL
29  -----
30          -1          -1
31
32 SQL> select seq2.currval,seq2.nextval from dual;
33
34      CURRVAL      NEXTVAL
35  -----
36          -2          -2
```

## 13.3、序列的使用

调用NEXTVAL将生成序列中的下一个序列号, 调用时要指出序列名, 即用以下方式调用: **序列名.NEXTVAL**

CURRVAL用于产生序列的当前值, 无论调用多少次都不会产生序列的下一个值。如果序列还没有通过调用NEXTVAL产生过序列的下一个值, 先引用CURRVAL没有意义。调用CURRVAL的方法同上, 要指出序列名, 即用以下方式调用: **序列名.CURRVAL**

```
1  SQL> insert into students values(seq1.nextval,'张三',22,to_date('1999-02-
2  03','yyyy-mm-dd'),'123456789');
3  1 row inserted
4
5  SQL> select * from students;
```

```

6
7          SID SNAME                AGE BIRTHDAY      TEL
8  -----
9          4 张三                    22 1999/2/3      123456789
10         1 张恒                    20 2000/3/17     17609208151
11
12 SQL> select seq1.nextval from dual;
13
14         NEXTVAL
15  -----
16         5
17
18 SQL> insert into students values(seq1.nextval, '李四',22,to_date('2005-02-
19 03', 'yyyy-mm-dd'), '123456789');
20
21 1 row inserted
22
23 SQL> select * from students;
24
25          SID SNAME                AGE BIRTHDAY      TEL
26  -----
27          4 张三                    22 1999/2/3      123456789
28          6 李四                    22 2005/2/3      123456789
29          1 张恒                    20 2000/3/17     17609208151

```

## 13.4、序列的修改

- 必须是序列的拥有者或对序列有 ALTER any sequence 权限
- 只有将来的序列值会被改变
- 改变序列的初始值只能通过删除序列之后重建序列的方法实现

```

1 SQL> alter sequence seq1 start with 10;
2 alter sequence seq1 start with 10
3 -- 初始值无法修改
4 ORA-02283: 无法变更启动序列号
5
6 SQL>
7 SQL> -- 修改序列步长为3最大值10不循环
8 SQL> alter sequence seq1 increment by 3 maxvalue 10 nocycle;
9
10 Sequence altered
11
12
13 SQL> select seq1.currval,seq1.nextval from dual;
14
15         CURRVAL      NEXTVAL
16  -----
17         9            9
18
19 SQL> select seq1.currval,seq1.nextval from dual;
20 select seq1.currval,seq1.nextval from dual
21
22 -- 当序列到达最大值，不循环的情况下报错
23 ORA-08004: 序列 SEQ1.NEXTVAL exceeds MAXVALUE 无法实例化

```



## 13.5、查询序列

- 通过数据字典USER\_OBJECTS可以查看用户拥有的序列。

```
1 SQL> select object_name,object_type from user_objects;
2
3 OBJECT_NAME          OBJECT_TYPE
4 -----
5 SYS_C006997          INDEX
6 STUDENTS             TABLE
7 SEQ2                 SEQUENCE
8 SEQ1                 SEQUENCE
9
```

- 通过数据字典USER\_SEQUENCES可以查看序列的设置。

```
1 SQL> select * from user_sequences;
2
3 SEQUENCE_NAME MIN_VALUE MAX_VALUE INCREMENT_BY CYCLE_FLAG ORDER_FLAG
4 CACHE_SIZE LAST_NUMBER
5 -----
6 SEQ1          1         10         3 N          N          20
7          12
8 SEQ2          -1E27        -1        -1 N          N          20
9          -21
```

## 13.6、删除序列

```
1 SQL> drop sequence seq1;
2
3 Sequence dropped
4
5 SQL> drop sequence seq2;
6
7 Sequence dropped
```

# 14、视图

- 视图以经过定制的方式显示来自一个或多个表的数据
- 视图可以视为'虚拟表'或'存储的查询'
- 创建视图所依据的表称为'基表'

## 14.1、优点

- 提供了另外一种级别的表安全性
- 隐藏的数据的复杂性
- 简化的用户的SQL命令
- 隔离基表结构的改变
- 通过重命名列，从另一个角度提供数据

## 14.2、视图与表的区别

- 表占磁盘空间, 视图不占
- 视图不能加索引
- 视图可以简化操作
- 提高安全性 (可以给不同用户分配不同的视图)

## 14.3、视图的创建

要在当前方案中创建视图, 用户必须具有create view系统权限; 要在其他方案中创建视图, 用户必须具有create any view系统权限. 视图的功能取决于视图拥有者的权限

语法:

```
create [ or replace ] [ force ] view [schema.]view_name
    [ (column1,column2,...) ]
as
select ...
[ with check option ]      [ constraint constraint_name ]
[ with read only ];
```

解析

**or replace:** 如果存在同名的视图, 则使用新视图"替代"已有的视图

**force:** "强制"创建视图, 不考虑基表是否存在, 也不考虑是否具有使用基表的权限

**column1,column2,...:** 视图的列名, 列名的个数必须与select查询中列的个数相同; 如果select查询包含函数或表达式, 则必须为其定义列名.此时, 既可以用column1, column2指定列名, 也可以在select查询中指定列名.

**with check option:** \*指定对视图执行的dml操作必须满足"视图子查询"的条件即,对通过视图进行的增删改操作进行'检查',要求增删改操作的数据, 必须是select查询所能查询到的数据, 否则不允许操作并返回错误提示. 默认情况下, 在增删改之前"并不会检查"这些行是否能被select查询检索到.

**with read only:** 创建的视图只能用于查询数据, 而不能用于更改数据.

```
1  -- 查询用户是否具有create view权限
2  select * from session_privs;
3
4  -- 没有权限切换dba用户给用户授权
5  conn sys/123456 @XE as sysdba;
6  grant create view to zh;
7  conn zh/123456 @XE;
8
9
10 -- 创建简单的视图
11 create or replace view v1(id,name,age) as
12 select sid,sname,age from students;
13
14 -- 查询视图结构
15 desc v1;
16
17 -- 对视图进行DML操作
18 select * from v1;
19 insert into v1 values(5, 'aaa', 23);
20 update v1 set age = 25 where id = 5;
21 delete v1 where id = 5;
22
```

```

23
24 SQL> insert into v1 values(5, 'aaa', 23);
25
26 1 row inserted
27
28 SQL> select * from v1;
29
30      ID NAME      AGE
31 -----
32      4 张三      22
33      6 李四      22
34      5 aaa       23
35      1 张恒      20
36
37 SQL> select * from students;
38
39      SID SNAME      AGE BIRTHDAY      TEL
40 -----
41      4 张三      22 1999/2/3      123456789
42      6 李四      22 2005/2/3      123456789
43      5 aaa       23
44      1 张恒      20 2000/3/17     17609208151
45
46 SQL> update v1 set age = 25 where id = 5;
47
48 1 row updated
49
50 SQL> select * from students;
51
52      SID SNAME      AGE BIRTHDAY      TEL
53 -----
54      4 张三      22 1999/2/3      123456789
55      6 李四      22 2005/2/3      123456789
56      5 aaa       25
57      1 张恒      20 2000/3/17     17609208151
58
59 SQL> delete v1 where id = 5;
60
61 1 row deleted
62
63 SQL> select * from students;
64
65      SID SNAME      AGE BIRTHDAY      TEL
66 -----
67      4 张三      22 1999/2/3      123456789
68      6 李四      22 2005/2/3      123456789
69      1 张恒      20 2000/3/17     17609208151

```

对视图进行DML操作，基表也会发生改变

## 14.4、创建只读视图

```

1  -- 创建只读视图 使用or replace替换之前的视图
2  create or replace view v1(id,name,age) as
3  select sid,sname,age from students
4  with read only;

```

```

5
6 SQL> create or replace view v1(id,name,age) as
7   2 select sid,sname,age from students
8   3 with read only;
9
10 View created
11
12 SQL> select * from v1;
13
14          ID NAME                AGE
15  -----
16          4 张三                22
17          6 李四                22
18          1 张恒                20
19
20 SQL> insert into v1 values(5, 'aaa',23);
21 insert into v1 values(5, 'aaa',23)
22
23 ORA-42399: 无法对只读视图执行 DML 操作

```

无法对只读视图执行DML 操作

## 14.5、创建检查约束视图with check option

```

1 create or replace view v1(id,name,age) as
2 select sid,sname,age from students where age = 20
3 with check option;
4
5 SQL> create or replace view v1(id,name,age) as
6   2 select sid,sname,age from students where age = 20
7   3 with check option;
8
9 View created
10
11 SQL> select * from v1;
12
13          ID NAME                AGE
14  -----
15          1 张恒                20
16
17 SQL> insert into v1 values(5, 'aaa',23);
18 insert into v1 values(5, 'aaa',23)
19
20 ORA-01402: 视图 WITH CHECK OPTION where 子句违规
21
22 SQL> insert into v1 values(5, 'aaa',20);
23
24 1 row inserted

```

**创建检查视图：**对通过视图进行的增删改操作进行检查，要求增删改操作的数据必须是select查询所能查询到的数据

age=23不在查询范围内，违反检查约束，所以无法插入；

## 14.6、连接视图

连接视图是指基于多个表所创建的视图，即，定义视图的查询是一个连接查询。主要目的是为了简化连接查询；

### 14.6.1、创建

```
1  -- 创建连接视图
2  create or replace view v1(sid,sname,age,tname) as
3  select s.sid,s.sname,s.age,t.tname from students s,teachers t where s.tid =
   t.tid
4
5  SQL> create or replace view v1(sid,sname,age,tname) as
6      2 select s.sid,s.sname,s.age,t.tname from students s,teachers t where
   s.tid = t.tid
7      3 ;
8
9  View created
10
11 SQL> select * from v1;
12
13          SID SNAME                AGE TNAME
14  -----
15          1  张恒                20  苏老师
16          2  张三                25  苏老师
17          3  李四                35  苏老师
```

### 14.6.2、连接视图的DML

```
1  SQL> insert into v1 values(4,'aaa',20,'张老师');
2  insert into v1 values(4,'aaa',20,'张老师')
3
4  ORA-01776: 无法通过联接视图修改多个基表
```

在视图上进行的所有DML操作，最终都会在基表上完成；

select 视图没有什么限制，但insert/delete/update有一些限制；

### 14.6.3、键值保存表

如果连接视图中的一个“基表的键”(主键、唯一键)在它的视图中仍然存在，并且“基表的键”仍然是“连接视图中的键”(主键、唯一键)；即，某列在基表中是主键\唯一键，在视图中仍然是主键\唯一键，则称这个基表为“键值保存表”。

一般地，由主外键关系的2个表组成的连接视图，外键表就是键值保存表，而主键表不是。

### 14.6.4、连接视图的更新准则

#### 1. 一般准则

- 任何DML操作，只能对视图中的键值保存表进行更新，即，“不能通过连接视图修改多个基表”；
- 在DML操作中，“只能使用连接视图定义过的列”；
- “自连接视图”的所有列都是可更新(增删改)的

#### 2. insert准则

- 在insert语句中不能使用“非键值保存表”中的列(包括“连接列”)；
- 执行insert操作的视图，至少应该“包含”键值保存表中所有设置了约束的列；

- 如果在定义连接视图时使用了WITH CHECK OPTION 选项, 则“不能”针对连接视图执行insert操作

### 3. update准则

- 键值保存表中的列是可以更新的;
- 如果在定义连接视图时使用了WITH CHECK OPTION 选项, 则连接视图中的连接列(一般就是“共有列”)和基表中的“其他共有列”是“不可”更新的, 连接列和共有列之外的 其他列是“可以”更新的、

### 4. delete准则

- 如果在定义连接视图时使用了WITH CHECK OPTION 选项, 依然“可以”针对连接视图执行delete操作

## 14.6.5、可更新连接视图

如果创建连接视图的select查询‘不包含’如下结构, 并且遵守连接视图的‘更新准则’, 则这样的连接视图是“可更新”的:

- 集合运算符(union,intersect,minus)
- DISTINCT关键字
- GROUP BY, ORDER BY, CONNECT BY或START WITH子句
- 子查询
- 分组函数
- 需要更新的列不是由‘列表表达式’定义的
- 基表中所有NOT NULL列均属于该视图

## 14.7、创建复杂视图

复杂视图是指包含函数、表达式、或分组数据的视图。主要目的是为了简化查询。主要用于执行查询操作, 并不用于执行DML操作。

注意: 当视图的select查询中包含函数或表达式时, 必须为其定义列别名。

```

1  -- 创建复杂视图
2  create or replace view v1(ages) as
3  select sum(age) from students;
4
5  SQL> create or replace view v1(ages) as
6      2  select sum(age) from students;
7
8  View created
9
10 SQL> select * from v1;
11
12      AGES
13  -----
14      80

```

## 14.8、强制创建视图

正常情况下, 如果基表不存在, 创建视图就会失败。但是可以使用force选项强制创建视图(前提: 创建视图的语句没有语法错误!), 此时该视图处于失效状态

```

1  SQL> create or replace view v1 as
2      2  select * from test;
3  create or replace view v1 as

```

```

4  select * from test
5
6  -- 此时没有基表，语句报错
7  ORA-00942: 表或视图不存在
8
9  -- 使用force强制创建视图
10 SQL> create force view v1 as
11     2 select * from test;
12 -- 警告：创建的视图带有编译错误。
13 Warning: View created with compilation errors
14
15 -- 查询视图，没有test基表报错
16 SQL> select * from v1;
17 select * from v1
18
19 ORA-04063: view "ZH.V1" 有错误
20
21 -- 查询视图状态
22 select object_name,status from user_objects where object_name='V1';
23 SQL> select object_name,status from user_objects where object_name='V1';
24 -- INVALID: 视图状态为不可用
25
26 OBJECT_NAME                                STATUS
27 -----
28 V1                                           INVALID
29
30 -- 创建基表 插入测试数据
31 SQL> create table test(id number);
32 Table created
33 SQL> insert into test values(1);
34 1 row inserted
35
36 -- 查询视图
37 SQL> select * from v1;
38
39 ID
40 ---
41 1
42
43 -- 查询视图状态
44 SQL> select object_name,status from user_objects where object_name='V1';
45 -- VALID: 视图状态为可用
46
47 OBJECT_NAME                                STATUS
48 -----
49 V1                                           VALID

```

强制创建视图之后创建基表，查询视图之后，视图状态才会发生改变。INVALID：视图状态为不可用。  
VALID：视图状态为可用

## 14.9、更改视图

在对视图进行更改(或重定义)之前，需要考虑如下几个问题：

1. 由于视图只是一个虚表，其中没有数据，所以更改视图只是改变数据字典中对该视图的定义信息，视图的所有基础对象都不会受到任何影响
2. 更改视图之后，依赖于该视图的所有视图和PL/SQL程序都将变为INVALID(失效)状态

3. 如果以前的视图中具有with check option选项, 但是重定义时没有使用该选项, 则以前的此选项将自动删除

### 14.9.1、更改视图的定义

执行create or replace view语句。这种方法代替了先删除(“权限也将随之删除”)后创建的方法, 会保留视图上的权限, 但与该视图相关的存储过程和视图会失效。

```
1  -- 创建只读视图 使用or replace替换之前的视图
2  create or replace view v1(id,name,age) as
3  select sid,sname,age from students
4  with read only;
```

### 14.9.2、视图的重新编译

**语法:** alter view 视图名 compile;

**作用:** 当视图依赖的基表改变后, 视图会“失效”。为了确保这种改变“不影响”视图和依赖于该视图的其他对象, 应该使用alter view 语句“明确的重新编译”该视图, 从而在运行视图前发现重新编译的错误。视图被重新编译后, 若发现错误, 则依赖该视图的对象也会失效; 若没有错误, 视图会变为“有效”。

**权限:** 为了重新编译其他模式中的视图, 必须拥有alter any table系统权限。

**注意:** 当访问基表改变后的视图时, oracle会“自动重新编译”这些视图。

```
1  -- 查询视图状态
2  SQL> select object_name,status from user_objects where object_name='V1';
3  -- VALID: 视图状态为可用
4  OBJECT_NAME                                STATUS
5  -----
6  V1                                           VALID
7
8  -- 修改基表
9  SQL> alter table students modify(sid number(20));
10
11  Table altered
12
13  -- 查询视图状态
14  select object_name,status from user_objects where object_name='V1';
15  SQL> select object_name,status from user_objects where object_name='V1';
16  -- INVALID: 视图状态为不可用
17  OBJECT_NAME                                STATUS
18  -----
19  V1                                           INVALID
20
21  -- 重新编译
22  SQL> alter view v1 compile;
23
24  View altered
25
26  -- 查询视图状态
27  SQL> select object_name,status from user_objects where object_name='V1';
28  -- VALID: 视图状态为可用
29  OBJECT_NAME                                STATUS
30  -----
31  V1                                           VALID
```



## 14.10、查看视图

使用数据字典视图

- dba\_views——DBA视图描述数据库中的所有视图
- all\_views——ALL视图描述用户“可访问的”视图
- user\_views——USER视图描述“用户拥有的”视图
- dba\_tab\_columns——DBA视图描述数据库中的所有视图的列(或表的列)
- all\_tab\_columns——ALL视图描述用户“可访问的”视图的列(或表的列)
- user\_tab\_columns——USER视图描述“用户拥有的”视图的列(或表的列)

```
1  -- 查看当前方案所有视图信息
2  select view_name,text from user_views;
3  -- 查询当前方案中指定视图(或表)的列名信息
4  select * from user_tab_columns where table_name='v1';
5
6  SQL> select view_name,text from user_views;
7
8  VIEW_NAME                                TEXT
9  -----
10 v1                                         select sid,sname,age from students
11                                         with read only
```

## 14.11、删除视图

- 可以删除当前模式中的任何视图;
- 如果要删除其他模式中的视图, 必须拥有DROP ANY VIEW系统权限;
- 视图被删除后, 该视图的定义会从词典中被删除, 并且在该视图上授予的“权限”也将被删除。
- 视图被删除后, 其他引用该视图的视图及存储过程等都会失效。

```
1  -- 删除视图
2  SQL> drop view v1;
3
4  View dropped
```

参考文档: <https://www.cnblogs.com/cxdanger/p/9914307.html>

## 15、索引

- 索引是与表相关的一个可选结构
- 用以提高SQL 语句执行的性能
- 减少磁盘I/O
- 使用CREATE INDEX 语句创建索引
- 在逻辑上和物理上都独立于表的数据
- Oracle 自动维护索引

### 15.1、索引的分类

- BTree索引
  - B树索引的存储结构类似书的索引结构, 有分支和叶两种类型的存储数据块, 分支块相当于书的大目录, 叶块相当于索引到的具体的书页。Oracle用B树机制存储索引条目, 以保证用最短

路径访问键值。默认情况下大多使用B\*树索引，该索引就是通常所见的唯一索引、逆序索引等。

- 位图索引
  - 位图索引存储主要用于节省空间，减少oracle对数据块的访问。它采用位图偏移方式来与表的行ID号对应，采用位图索引一般是重复值太多的表字段。位图索引之所以在实际密集型OLTP（联机事物处理）中用的比较少，是因为OLTP会对表进行大量的删除、修改、新建操作。Oracle每次进行操作都会对要操作的数据块加锁。以防止多人操作容易产生的数据库锁等待甚至死锁现象。在OLAP（联机分析处理）中应用位图有优势，因为OLAP中大部分是对数据库的查询操作，而且一般采用数据仓库技术，所以大量数据采用位图索引节省空间比较明显。当创建表的命令中包含有唯一性关键字时，不能创建位图索引，创建全局分区索引时也不能用位图索引。

## 15.2、索引原理

- 若没有索引，搜索某个记录时（例如查找name='wish'）需要搜索所有的记录，因为不能保证只有一个wish，必须全部搜索一遍
- 若在name上建立索引，oracle会对全表进行一次搜索，将每条记录的name值哪找升序排列，然后构建索引条目（name和rowid），存储到索引段中，查询name为wish时即可直接查找对应地方
- 创建了索引并不一定会使用，oracle自动统计表的信息后，决定是否使用索引，表中数据很少时使用全表扫描速度已经很快，没有必要使用索引

## 15.3、标准索引

### 创建

语法: create index 索引名 on 表名(字段)

```
1 SQL> create index i1 on students(sname);
2
3 Index created
```

### 删除

语法: drop index 索引名

```
1 SQL> drop index i1;
2
3 Index dropped
```

## 15.4、唯一索引

- 唯一索引确保在定义索引的列中没有重复值
- Oracle自动在表的主键列上创建唯一索引
- 使用CREATE UNIQUE INDEX语句创建唯一索引

语法: create unique index 索引名 on 表名(字段)

```
1 SQL> create unique index i1 on students(sname);
2
3 Index created
```

## 15.5、组合索引

- 组合索引是在表的多个列上创建的索引
- 索引中列的顺序是任意的
- 如果 SQL 语句的 WHERE 子句中引用了组合索引的所有列或大多数列，则可以提高检索速度

语法: create index 索引名 on 表名(字段列表)

```
1 SQL> create index i1 on students(sid,sname);
2
3 Index created
```

## 15.6、反向键索引

- 反向键索引反转索引列键值的每个字节
- 通常建立在值是连续增长的列上，使数据均匀地分布在整个索引上
- 创建索引时使用 REVERSE 关键字

索引是存在硬盘上，如果数据的修改导致数据的顺序要修改（如删除一条数据），系统会锁定一大块索引数据。

语法: create index 索引名 on 表名(字段) reverse

```
1 SQL> create index i1 on students(sname) reverse;
2
3 Index created
```

## 15.7、位图索引

- 位图索引适合创建在低基数列上
- 位图索引不直接存储 ROWID，而是存储字节位到 ROWID 的映射
- 减少响应时间
- 节省空间占用

Oracle 企业版才有添加位图索引的功能

语法: create bitmap index 索引名 on 表名(字段)

```
1 SQL> create bitmap index i1 on students(sid);
2 create bitmap index i1 on students(sid)
3
4 ORA-00439: 未启用功能: Bit-mapped indexes
```

## 15.8、基于函数的索引

- 基于一个或多个列上的函数或表达式创建的索引
- 表达式中不能出现聚合函数
- 不能在 LOB 类型的列上创建
- 创建时必须具有 QUERY REWRITE 权限

语法: create index i 索引名 on 表名(函数 字段)

```

1 SQL> create index i1 on students(sum(age));
2 create index i1 on students(lower(age))
3
4 ORA-00934: 此处不允许使用分组函数
5
6 SQL> create index i1 on students(lower(age));
7
8 Index created

```

## 15.9、索引组织表

- 索引组织表的数据是根据主键排**不仅可以存储数据，还可以存储为表建立的索引**序后的顺序进行排列的，这样就提高了访问的速度。
- 但是这是由牺牲插入和更新性能为代价的(每次写入和更新后都要重新进行重新排序)。
- 适合数据不会改变的情况使用(如软件说明书)
- 针对查询比较频繁，数据存储不多的表可以采用，比如数据字典、参数表等**

```

1 SQL> CREATE TABLE ind_org_tab (
2     vencode NUMBER(4) PRIMARY KEY,
3     venname VARCHAR2(20)
4 )
5 ORGANIZATION INDEX;

```

## 15.10、分区表索引

- 表的数据存储在不同的分区上，可以创建分区表的索引，可以分为本地的和全局的
- 分区表本地索引语法
  - Create index 索引名 on 表名(列名) local
- 分区表全局索引
  - Create index 索引名 on 表名(列名) global

## 15.11、索引中的分区

- 索引对象也会占用存储空间，可以将索引存储在不同的分区中
- 与分区有关的索引有三种类型：
  - 局部分区索引 - 在分区表上创建的索引，在每个表分区上创建独立的索引，索引的分区范围与表一致
  - 全局分区索引 - 在分区表或非分区表上创建的索引，索引单独指定分区的范围，与表的分区范围或是否分区无关
  - 全局非分区索引 - 在分区表上创建的全局普通索引，索引没有被分区

```

1 Create index 索引名 on 表名(列名) global
2 Partition by range(列名)(
3     Partition a values less then(1500),
4     Partition a values less then(maxvalue)
5 );

```

## 15.12、总结

- 使用索引原则：

1. 在大表上建立索引才有意义
  2. 在where 字句或是连接条件上经常引用的列上建立索引
  3. 索引的层次不要超过4层。
- 索引的不足：
    1. 占磁盘空间
    2. 更新时消耗时间

## 16、PL/SQL

### 16.1、简介

SQL语言只是访问、操作数据库的语言，并不是一种具有**流程控制**的程序设计语言，而只有程序设计语言才能用于应用软件的开发。**PL/SQL 是对SQL 的扩展**。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**。由于该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。除此之外，可以在Oracle数据库的某些客户端工具中使用PL/SQL语言也是该语言的一个特点

在PL/SQL中可以使用的SQL语句有：insert, update, delete, select .... into, commit, rollback, savepoint

**在PL/SQL中只能使用SQL中的DML操作，不能使用DDL操作。如 (create table) ，只能使用动态方式**

### 16.2、优点

- 支持SQL，在PL/SQL 中可以使用：
  - 数据操纵命令
  - 事务控制命令
  - 游标控制
  - SQL 函数和SQL 运算符
  - 完全支持SQL数据类型
- 支持面向对象编程(OOP)
- 可移植性，可运行在任何操作系统和平台上的Oracle 数据库
- 更佳的性能

### 16.3、PL/SQL块

- PL/SQL 块是构成PL/SQL 程序的基本单元
- PL/SQL块 分为三个部分，声明部分、可执行部分和异常处理部分

```
1 DECLARE
2     --声明部分： 在此声明PL/SQL用到的变量,类型及游标，以及局部的存储过程和函数
3 BEGIN
4     -- 执行部分： 过程及SQL 语句 ，即程序的主要部分
5 EXCEPTION
6     -- 执行异常部分： 错误处理
7 END;
```

```

1  -- 控制台输出命令
2  SQL> set serveroutput on
3  SQL>
4  SQL> begin
5      2      dbms_output.put_line('hello PL/SQL');
6      3  end;
7      4  /  -- /表示代码块结束
8  hello PL/SQL
9
10 PL/SQL procedure successfully completed

```

### 注意

- 执行部分不能省略。
- 在控制台输出，需要开启set serveroutput on
- 代码块结束使用/表示结束

## 16.4、数据类型

### 16.4.1、基本数据类型

类型标识符	说明
number	数字型
int	整数型
pls_integer	整数型，产生溢出时出现错误
binary_integer	整数型，表示带符号的整数
char	定长字符型，最大255个字符
varchar2	变长字符型，最大2000个字符
long	变长字符型，最长2GB
date	日期型
boolean	布尔型（TRUE、FALSE、NULL三者之一）

### 16.4.2、属性类型

- 用于引用数据库列的数据类型，以及表示表中一行的记录类型
- 属性类型有两种：
  - %TYPE - 引用变量和数据库列的数据类型
  - %ROWTYPE - 提供表示表中一行的记录类型
- 使用属性类型的优点：
  - 不需要知道被引用的表列的具体类型
  - 如果被引用对象的数据类型发生改变，PL/SQL 变量的数据类型也随之改变

```

1  v_sal emp.sal%TYPE;
2  v_row emp%ROWTYPE;

```

### 16.4.3、变量和常量

- PL/SQL 块中可以使用变量和常量

- 在声明部分声明, 使用前必须先声明
- 声明时必须指定数据类型, 每行声明一个标识符
- 在可执行部分的SQL 语句和过程语句中使用
- 声明变量和常量的语法:

```
1 variable_name [CONSTANT] datatype [NOT NULL] [:= | DEFAULT
  initial_value]
```

- `variable_name` - 分配给变量的名称。
- `CONSTANT` - 可选的。如果指定, 变量的值是恒定的, 不能被更改。
- `datatype` - 要分配给变量的数据类型。
- 给变量赋值有两种方法:
  - 使用赋值语句:=
  - 使用SELECT INTO 语句

## 16.4.4、练习

### 16.4.4.1、:=赋值

```
1  -- 声明变量
2  SQL> declare
3      2  pi number(5,4) := 3.14;
4      3  begin
5      4      pi := 3.1415; -- 可以改变
6      5      dbms_output.put_line(pi);
7      6  end;
8      7  /
9
10 3.1415
11
12 PL/SQL procedure successfully completed
13
14
15 -- 声明常量 constant
16 SQL> declare
17      2  pi constant number(5,4) := 3.14;
18      3  begin
19      4      pi := 3.1415; -- 不可以改变, 报错
20      5      dbms_output.put_line(pi);
21      6  end;
22      7  /
23 declare
24 pi constant number(5,4) := 3.14;
25 begin
26 pi := 3.1415;
27 dbms_output.put_line(pi);
28 end;
29
30 ORA-06550: 第 4 行, 第 3 列:
31 PLS-00363: 表达式 'PI' 不能用作赋值目标
32 ORA-06550: 第 4 行, 第 3 列:
33 PL/SQL: Statement ignored
```

### 16.4.4.2、select into赋值

```

1  -- select into 赋值
2  SQL> declare
3      2  age number(3);
4      3  begin
5      4      select s.age into age from students s where s.sid = 1;
6      5      dbms_output.put_line(age);
7      6  end;
8      7  /
9  20
10 PL/SQL procedure successfully completed

```

### 16.4.4.3、%type

根据表中的一列的类型定义一个变量

```

1  SQL> -- %type
2  SQL> declare
3      2  sname students.sname%type;
4      3  begin
5      4      select s.sname into sname from students s where s.sid = 1;
6      5      dbms_output.put_line(sname);
7      6  end;
8      7  /
9  张三
10
11 PL/SQL procedure successfully completed
12

```

### 16.4.4.4、%rowtype

根据表中的一行的类型定义一个变量

```

1  SQL> -- %rowtype 根据表中的一行的类型定义一个变量
2  SQL> declare
3      2  s_row students%rowtype;
4      3  begin
5      4      select * into s_row from students s where s.sid = 1;
6      -- 这个一行。输出必须指定这一行中的字段
7      5      dbms_output.put_line(s_row);
8      6  end;
9      7  /
10
11 ORA-06550: 第 5 行, 第 3 列:
12 PLS-00306: 调用 'PUT_LINE' 时参数个数或类型错误
13 ORA-06550: 第 5 行, 第 3 列:
14 PL/SQL: Statement ignored
15
16 SQL>
17 SQL> -- %rowtype 根据表中的一行的类型定义一个变量
18 SQL> declare
19      2  s_row students%rowtype;
20      3  begin
21      4      select * into s_row from students s where s.sid = 1;
22      5      dbms_output.put_line(s_row.sname);
23      6  end;
24      7  /

```



```
25 | 张三
26 |
27 | PL/SQL procedure successfully completed
```

## 16.5、控制结构

PL/SQL 支持的流程控制结构:

- 条件控制
  - IF 语句
  - CASE 语句
- 循环控制
  - LOOP 循环
  - WHILE 循环
  - FOR 循环
- 顺序控制
  - GOTO 语句
  - NULL 语句

### 16.5.1、条件控制

#### 16.5.1.1、if语句

IF 语句根据条件执行一系列语句，有三种形式: if-THEN、IF-THEN-ELSE 和 IF-THEN-ELSIF

```
1  语法: IF 条件表达式 THEN
2          执行语句
3      END IF;
4  或者
5      IF 条件表达式 THEN
6          执行语句
7      ELSIF 条件表达式 THEN
8          执行语句
9      ELSE
10         执行语句
11     END IF;
```

```
1  SQL> -- if
2  SQL> declare
3      2 age int := 5;
4      3 begin
5      4     if age = 0 then
6      5         dbms_output.put_line('男');
7      6     elsif age = 1 then
8      7         dbms_output.put_line('女');
9      8     else
10     9         dbms_output.put_line('不能识别');
11    10     end if;
12    11 end;
13    12 /
14    不能识别
15
16    PL/SQL procedure successfully completed
```

### 16.5.1.2、case语句

- CASE 语句用于根据单个变量或表达式与多个值进行比较
- 执行CASE 语句前，先计算选择器的值

```
1  语法: case 字符串变量
2          when 结果值1 THEN 执行语句;
3          when 结果值2 THEN 执行语句;
4          . . .
5          else 执行语句;
6      end case;
7  或者
8      case
9          when 条件表达式1 THEN 执行语句;
10         when 条件表达式2 THEN 执行语句;
11         . . .
12         else 执行语句;
13     end case;
```

```
1  SQL> -- case01
2  SQL> declare
3      2 age int := 0;
4      3 begin
5      4 case age
6      5     when 0 then
7      6         dbms_output.put_line('男');
8      7     when 2 then
9      8         dbms_output.put_line('女');
10     9     else
11     10         dbms_output.put_line('不能识别');
12     11 end case;
13     12 end;
14     13 /
15 男
16
17 PL/SQL procedure successfully completed
18
19 -----
20 -----
21 SQL>
22 SQL> -- case02
23 SQL> declare
24     2 score int := 80;
25     3 begin
26     4 case
27     5     when score >= 90 then
28     6         dbms_output.put_line('优');
29     7     when score >= 60 then
30     8         dbms_output.put_line('良');
31     9     when score > 0 then
32     10         dbms_output.put_line('差');
33     11 else
34     12         dbms_output.put_line('输入错误');
35     13 end case;
36     14 end;
```

```
37 15 /
38 良
39
40 PL/SQL procedure successfully completed
```

## 16.5.2、循环控制

- 循环控制用于重复执行一系列语句
- 循环控制的三种类型：
  - LOOP - 无条件循环
  - WHILE - 根据条件循环
  - FOR - 循环固定的次数

无论是使用for还是while循环，都是以loop循环作为基础

### 16.5.2.1、loop

表达式：循环结束条件

```
1 LOOP
2     EXIT WHEN 条件表达式;
3     执行语句
4 END LOOP;
```

```
1 SQL> -- loop
2 SQL> declare
3     2 i int := 0;
4     3 begin
5     4     loop
6     5         exit when i > 10;
7     6         dbms_output.put_line(i);
8     7         i := i+1;
9     8     end loop;
10    9 end;
11    10 /
12    0
13    1
14    2
15    3
16    4
17    5
18    6
19    7
20    8
21    9
22    10
23
24 PL/SQL procedure successfully completed
```

### 16.5.2.2、while

表达式：循环条件

```
1 while 表达式
2 LOOP
3     执行语句
4 END LOOP;
```

```
1 SQL> -- while
2 SQL> declare
3     2 i int := 0;
4     3 begin
5     4     while i < 10
6     5         loop
7     6             dbms_output.put_line(i);
8     7             i := i+1;
9     8         end loop;
10    9 end;
11   10 /
12 0
13 1
14 2
15 3
16 4
17 5
18 6
19 7
20 8
21 9
22
23 PL/SQL procedure successfully completed
```

### 16.5.2.3、for

```
1 For 变量 in 起始值..结束值
2 LOOP
3     执行语句
4 END LOOP;
```

```
1 SQL> -- for
2 SQL> begin
3     2 for i in 1..10
4     3     loop
5     4         dbms_output.put_line(i);
6     5     end loop;
7     6 end;
8     7 /
9 1
10 2
11 3
12 4
13 5
14 6
15 7
16 8
17 9
18 10
19
```

### 16.5.3、跳转控制

- 有的业务逻辑只用顺序、分支和循环控制可能效率很低，如果引入跳转控制可以极大的提高程序效率。
  - GOTO 语句- 无条件地转到标签指定的语句
  - NULL 语句- 什么也不做的空语句

节点之后要加return或null

```

1  SQL> -- 跳转结构
2  SQL> declare
3      2  i int := 2;
4      3  begin
5          4  if i = 1 then
6              5  goto a;
7          6  elsif i = 2 then
8              7  goto b;
9          8  else
10             9  goto c;
11          10 end if;
12          11 <<a>>
13             12 dbms_output.put_line('跳转到a');
14             13 return;
15          14 <<b>>
16             15 dbms_output.put_line('跳转到b');
17             16 return;
18          17 <<c>>
19             18 null;
20          19 end;
21          20 /
22  跳转到b
23
24  PL/SQL procedure successfully completed
25
26
27  SQL>
28  SQL> -- 跳转结构
29  SQL> declare
30      2  i int := 5;
31      3  begin
32          4  if i = 1 then
33              5  goto a;
34          6  elsif i = 2 then
35              7  goto b;
36          8  else
37              9  goto c;
38          10 end if;
39          11 <<a>>
40             12 dbms_output.put_line('跳转到a');
41             13 return;
42          14 <<b>>
43             15 dbms_output.put_line('跳转到b');
44             16 return;
45          17 <<c>>

```

```

46 18      null;
47 19 end;
48 20 /
49
50 PL/SQL procedure successfully completed

```

## 16.6、异常

### EXCEPTION

- 在运行程序时出现的错误叫做异常
- 发生异常后，语句将停止执行，控制权转移到 PL/SQL 块的异常处理部分
- 异常有两种类型：
  - 预定义异常 - 当 PL/SQL 程序违反 Oracle 规则或超越系统限制时隐式引发
  - 用户定义异常 - 用户可以在 PL/SQL 块的声明部分定义异常，自定义的异常通过 RAISE 语句显式引发

### 16.6.1、预定义异常

```

1  SQL> -- 预定义异常
2  SQL> declare
3      2      age number(3);
4      3      begin
5      4      select s.age into age from students s;
6      5      exception
7      6      when too_many_rows then
8      7      dbms_output.put_line ('返回多行');
9      8      end;
10     9      /
11     返回多行
12
13     PL/SQL procedure successfully completed

```

### 16.6.2、自定义异常

```

1  SQL>
2  SQL> -- 用户自定义异常
3  SQL> -- 1.exc exception;定义异常
4  SQL> -- 2.raise exc;抛出异常
5  SQL> -- 3.when exc then捕获异常
6  SQL> declare
7      2      exc exception;
8      3      v_a int:=1;
9      4      begin
10     5      if v_a<3 then
11     6      raise exc;
12     7      else
13     8      dbms_output.put_line('v_a大于等于3');
14     9      end if;
15    10      exception
16    11      when exc then
17    12      dbms_output.put_line('出问题那');
18    13      end;
19    14      /

```

```
20 出问题那
21
22  PL/SQL procedure successfully completed
23
```

## 16.7、动态SQL

- 动态SQL：在PL/SQL把直接执行字符串的操作称为动态sql
- 编译程序对动态SQL不做处理，而是在程序运行时动态构造语句、对语句进行语法分析并执行
- DDL 语句命令和会话控制语句不能在PL/SQL中直接使用，但是可以通过动态SQL来执行
- 执行动态SQL的语法：

```
1  EXECUTE IMMEDIATE dynamic_sql_string
2
3  [INTO define_variable_list]
4
5  [USING bind_argument_list];
```

### 执行DDL操作

```
1  SQL> begin
2      2      execute immediate 'create table ccc(aa int)';
3      3  end;
4      4  /
5
6  PL/SQL procedure successfully completed
7
8
9  SQL> select * from ccc;
10
11                                     AA
12  -----
```

## 17、存储过程与存储函数

**区别：**是否可以通过return返回函数值。存储函数可以通过return返回函数值，而存储过程不可以。

**相同点：**保存一段可执行的SQL语句，方便开发调用

### 优点

1. 效率高
  - 存储过程编译一次后，就会存到数据库，每次调用时都直接执行。而普通的sql语句我们要保存到其他地方（例如：记事本上），都要先分析编译才会执行。所以想对而言存储过程效率更高。
2. 降低网络流量
  - 存储过程编译好会放在数据库，我们在远程调用时，不会传输大量的字符串类型的sql语句。
3. 复用性高
  - 存储过程往往是针对一个特定的功能编写的，当再需要完成这个特定的功能时，可以再次调用该存储过程。
4. 可维护性高
  - 当功能要求发生小的变化时，修改之前的存储过程比较容易，花费精力少。

## 5. 安全性高

- 完成某个特定功能的存储过程一般只有特定的用户可以使用，具有使用身份限制，更安全。

# 存储过程

## 17.1、创建

### 语法

```
1 create [or replace] procedure 过程名(参数名 in/out 类型)
2 as
3 变量定义
4 begin
5 end;
```

其中参数IN表示输入参数，是参数的默认模式。

OUT表示返回值参数，类型可以使用任意Oracle中的合法类型。

OUT模式定义的参数只能在过程体内部赋值，表示该参数可以将某个值传递回调用他的过程

IN OUT表示该参数可以向该过程中传递值，也可以将某个值传出去

in可以忽略，out不可以

- 存储过程或者存储函数，只能创建或者替换；
- 参数可以带也可以不带；
- as 相当于PLSQL语句中的declare，用来声明变量，游标等，但是不可以省略。
- 要说明，参数是输入参数(in)还是输出参数(out)；
- 为保证调用多个存储过程中处在同一个事务中，所以一般不在存储过程或者存储函数中commit或rollback；

### 练习

```
1  -- 定义存储过程 根据id修改指定学生年龄+5，并打印前后的年龄
2  create or replace procedure UPDATEAGE(id in number)
3  as
4    age number(3);
5  begin
6    select s.age into age from students s where s.sid = id;
7    dbms_output.put_line(age);
8    update students set age = age + 5 where sid = id;
9    select s.age into age from students s where s.sid = id;
10   dbms_output.put_line(age);
11 end;
12
13 -- 调用存储过程
14 begin
15   UPDATEAGE(1);
16 end;
17
18 -- 存储过程out 查询指定学生的年龄
19 create or replace procedure PfindAgeById(id number,age out number)
20 as
21 begin
22   select s.age into age from students s where s.sid = id;
23 end;
24
```



```

25 declare
26   age number(3);
27   begin
28     PFindAgeById(1,age);
29     dbms_output.put_line(age);
30 end;

```

## 注意

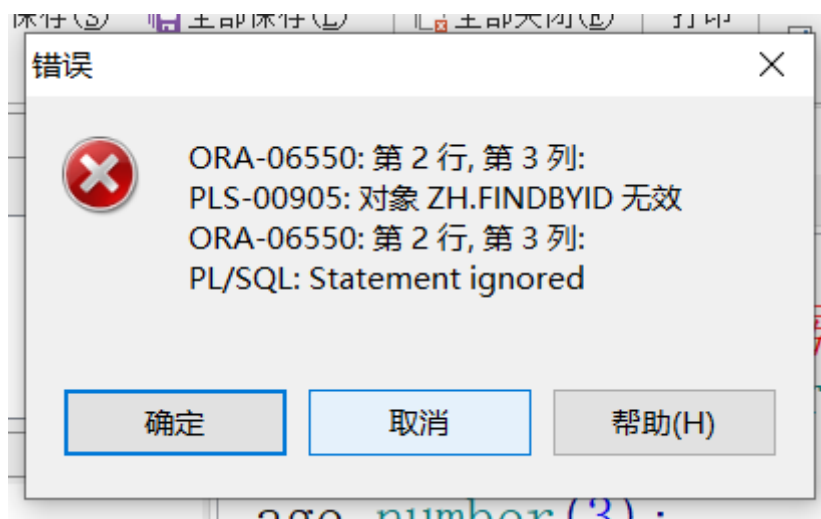
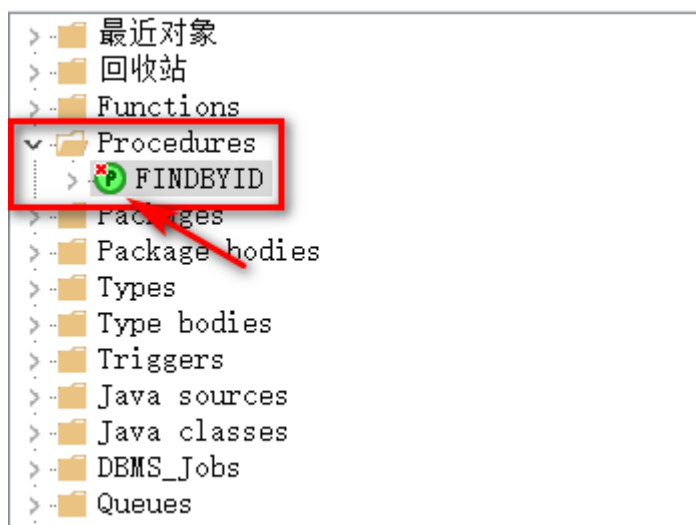
存储过程创建后在Procedures文件夹可以看见

```

1  create or replace procedure UPDATEAGE(id in number)
2  as
3    age number(3);
4  begin
5    -- 语法错误
6    selec s.age into age from students s where s.sid = id;
7    dbms_output.put_line(age);
8    update students set age = age + 5 where sid = id;
9    select s.age into age from students s where s.sid = id;
10   dbms_output.put_line(age);
11 end;

```

存储过程创建过程中，如果出现语法错误，是不会提示。只要在调用的时候才会报错。可以如下图查看

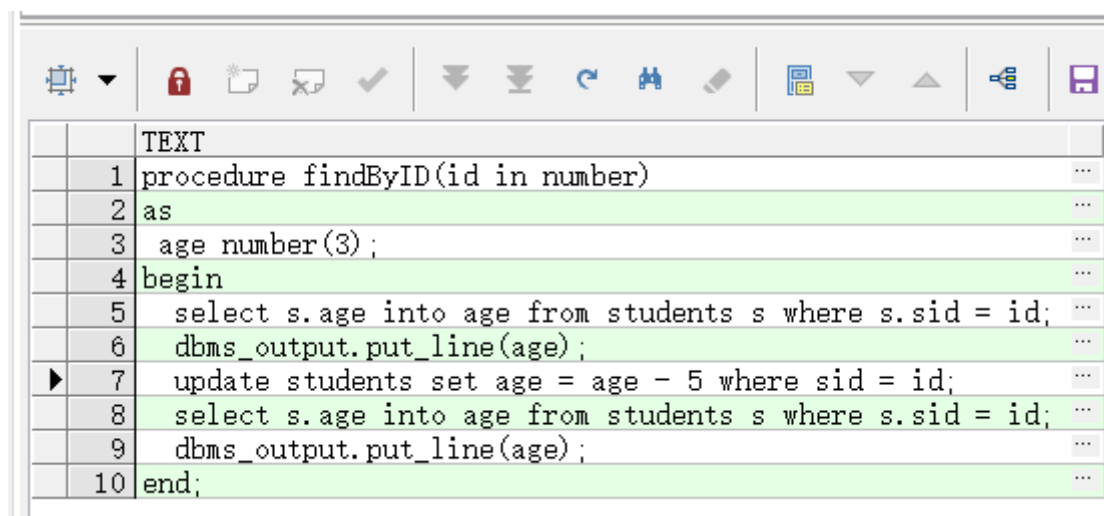


## 17.2、查看

这里的存储名必须为大写

### 17.2.1、查看存储过程的脚本

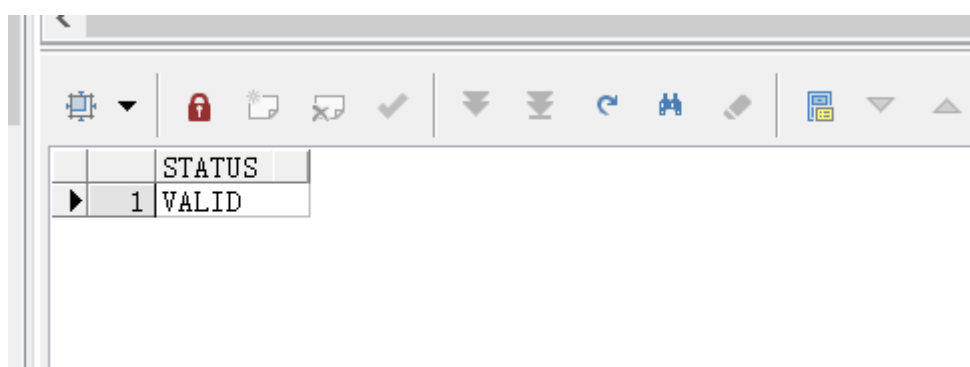
```
1  -- 查看存储过程的脚本
2  select text from user_source where name = 'UPDATEAGE';
```



	TEXT
1	procedure findByID(id in number)
2	as
3	age number(3);
4	begin
5	select s.age into age from students s where s.sid = id;
6	dbms_output.put_line(age);
7	update students set age = age - 5 where sid = id;
8	select s.age into age from students s where s.sid = id;
9	dbms_output.put_line(age);
10	end;

### 17.2.2、查看存储过程的状态

```
1  -- 查看存储过程的状态
2  select status from user_objects where object_name = 'UPDATEAGE';
```



	STATUS
1	VALID

说明: VALID 表示该存储过程有效(即通过编译), INVALID 表示存储过程无效或需要重新编译。当Oracle调用一个无效的存储过程或函数时, 首先试图对其进行编译, 如果编译成功则将状态置成VALID并执行, 否则给出错误信息。

当一个存储过程编译成功, 状态变为VALID, 会不会在某些情况下变成INVALID。结论是完全可能的。比如一个存储过程中包含对表的查询, 如果表被修改或删除, 存储过程就会变成无效INVALID。所以要注意存储过程和函数对其他对象的依赖关系

## 17.3、删除

**DROP PROCEDURE 存储过程名;**

```
1  -- 删除存储过程
2  drop procedure updateAge;
```

## 存储函数

- 函数(Function)为一命名的存储程序，可带参数，并返回一计算值；
- 函数和过程的结构类似，但必须有一个return子句，用于返回函数值。

## 17.1、创建

### 语法

```
1 create [or replace] function 方法名(参数名 in/out 类型) return 参数类型
2 as
3 变量名 类型要和return的返回类型一致
4 begin
5     return 变量名
6 end;
```

### 测试

```
1 -- 存储函数 查询指定学生的年龄
2 create or replace function findAgeById(id number) return number
3 as
4 age number(3);
5 begin
6     select s.age into age from students s where s.sid = id;
7     return age;
8 end;
9
10 -- 调用存储函数
11 declare
12 age number(3);
13 begin
14     age := findAgeById(1);
15     dbms_output.put_line(age);
16 end;
```

## 17.2、查看

查看语法与存储过程一样

## 17.3、删除

**drop function 函数名**

```
1 -- 删除
2 drop function FINDAGEBYID;
```

## 总结

- 一般来讲，存储过程和存储函数的区别在于存储函数可以有一个返回值，而存储过程没有返回值；
- 过程和函数都可以通过out指定一个或多个输出参数。我们可以利用out参数，在过程和函数中实现返回多个值；
  - 存储过程和存储函数都可以有out参数；
  - 存储过程和存储函数都可以有多个out参数；
  - 存储过程可以通过out参数来实现返回值。
- 什么时候用存储过程/存储函数？

- 原则：如果只有一个返回值，用存储函数；否则，就用存储过程。

## 18、触发器(trigger)

触发器的定义就是说某个条件成立的时候，触发器里面所定义的语句就会被自动的执行。因此触发器不需要人为的去调用，也不能调用。

### 18.1、创建

#### 语法

```
1 create [or replace] trigger 触发器名
2 before\after
3 insert\update\delete
4 on 表名
5 for each row -- :new :old
6 begin
7 end;
```

#### 说明

before：执行之前

after：执行之后

insert update delete：触发操作

for each row：行级触发器

old：代表变更前记录

new：代表变更后的记录

:new :old这两个变量只有在使用了关键字"for each row"时才存在.且update语句两个都有,而insert只有:new,delete 只有:old

#### 练习一

```
1 -- 插入一条数据之后，打印‘有一条数据被插入’
2 create or replace trigger t1
3 after
4 insert
5 on students
6 begin
7     dbms_output.put_line('有一条数据被插入');
8 end;
9
10 SQL> insert into students values(4, 'admin', 20);
11
12 有一条数据被插入
13
14 1 row inserted
```

#### 练习二：insert

```
1 -- 插入一条数据之前，打印新插入的年龄
```

```

2  create or replace trigger t2
3  before
4  insert
5  on students
6  for each row
7  begin
8      dbms_output.put_line(:new.age);
9  end;
10
11 SQL> insert into students values(4, 'admin', 20);
12
13 20
14 有一条数据被插入
15
16 1 row inserted

```

### 练习三: update

```

1  -- 修改一条数据之前, 打印新旧数据
2  create or replace trigger t3
3  before
4  update
5  on students
6  for each row
7  begin
8      dbms_output.put_line(:old.age);
9      dbms_output.put_line(:new.age);
10 end;
11
12 SQL> update students set age = 25 where sid = 4;
13
14 20
15 25
16 1 row updated

```

### 练习四: delete

```

1  -- 删除一条数据之后, 打印旧数据
2  create or replace trigger t4
3  after
4  delete
5  on students
6  for each row
7  begin
8      dbms_output.put_line(:old.age);
9  end;
10
11 SQL> delete from students where sid = 4;
12
13 25
14
15 1 row deleted

```

## 18.2、删除

`drop trigger` 触发器名;

```
1 | drop trigger t4;
```