1 Oracle 概述

1.1 常见数据库

数据库（Database）是按照数据结构来组织、存储和管理数据的仓库。数据库通常分为层次式数据库、网络式数据库和关系式数据库三种；不同的数据库是按不同的数据结构来联系和组织的。将反映和实现数据联系的方法称为数据模型。层次结构模型实质上是一种有根结点的定向有序树，按照层次模型建立的数据库系统称为层次模型数据库系统；按照网状数据结构建立的数据库系统称为网状数据库系统；关系式数据结构把一些复杂的数据结构归结为简单的二元关系(即二维表格形式)，由关系数据结构组成的数据库系统被称为关系数据库系统。

数据库管理系统(Database Management System)是一种操纵和管理数据库的大型软件，用于建立、使用和维护数据库，简称 DBMS。它对数据库进行统一的管理和控制，以保证数据库的安全性和完整性。用户通过 DBMS 访问数据库中的数据。数据库管理系统是数据库系统的核心，是管理数据库的软件。

常见的关系型数据库有：DB2,Sybase,Oracle,MySQL,Access,MS SQL Server...

1.2 Oracle 简介

Oracle 甲骨文公司是第一个跨整个产品线（数据库、业务应用软件和应用软件开发与决策支持工具）开发和部署 100%基于互联网的企业软件的公司。Oracle 是世界领先的信息管理软件供应商和世界第二大独立软件公司。其主要的有：

数据库服务器：oracle(9i,10g/11g,12c),MySQL

应用服务器：WegLogic，GlassFish

开发语言：Java

开发集成环境：NetBean

oracle 数据库是当前最主流的数据库之一。

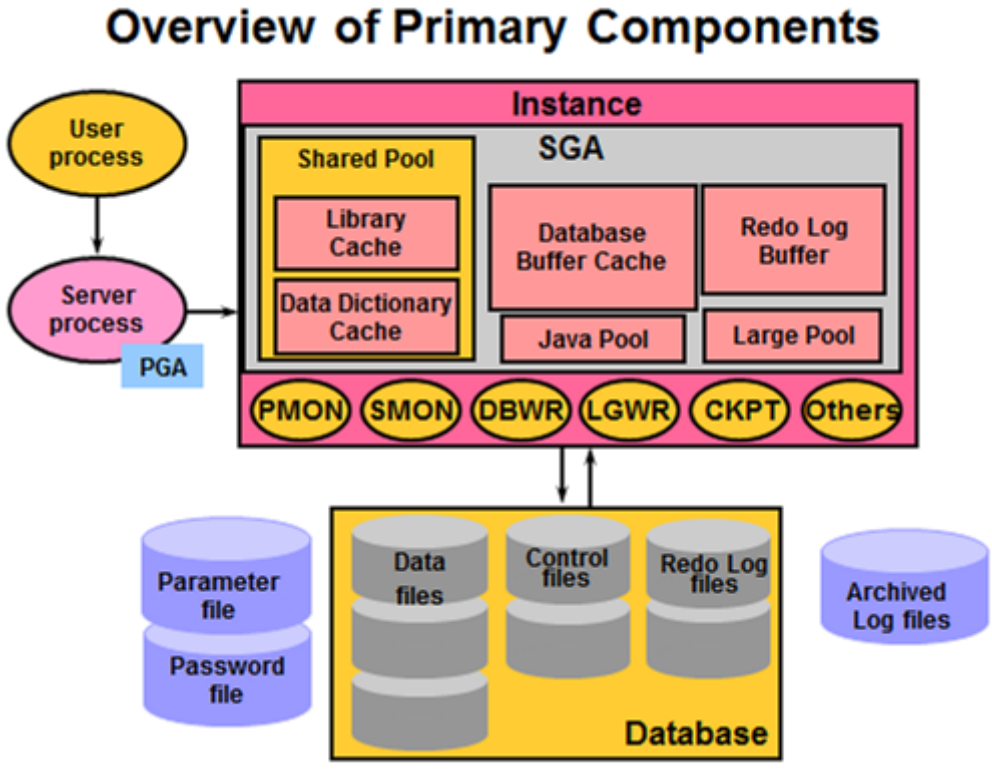
2 Oracle 安装与组成

2.1 安装 oracle 11g 数据库

详见《oracle 11g 32 位安装.docx》

2.2 Oracle 11g 数据库的组成

Oracle 的整体架构：



上图示；一般 Oracle 数据库管理系统由：实例和数据库两部分组成。

1、**数据库**是一系列物理文件的集合（数据文件，控制文件，联机日志，参数文件等）；Oracle 数据库由操作系统文件组成，这些文件也称为数据库文件，为数据库信息提供实际物理存储区。Oracle 数据库包括逻辑结构和物理结构。数据库的物理结构包含数据库中的一组操作系统文件。数据库的逻辑结构是指数据库创建之后形成的逻辑概念之间的关系，如表、视图、索引等对象。

2、**实例**则是一组 Oracle 后台进程/线程以及在服务器分配的共享内存区。

Oracle 可以创建多个 oracle 数据库，一个 oracle 数据库将又由实例和数据库构成。如默认安装时创建的 orcl 数据库外还可再创建其它数据库。创建的数据库将在$oracleHome/oradata/数据库名 目录下以一个个的*.DBF 文件体现出来。

2.3 Oracle 11g 数据库服务

Oracle * VSS Writer Service -- Oracle 卷映射拷贝写入服务，VSS（Volume Shadow Copy Service）能够让存储基础设备（比如磁盘，阵列等）创建高保真的时间点映像，即映射拷贝（shadow copy）。它可以在多卷或者单个卷上创建映射拷贝，同时不会影响到系统的系统能。(非必须启动)

OracleDBConsole* -- Oracle 数据库控制台服务;在运行 Enterprise Manager(企业管理器 EM)的时候，需要启动这个服务；此服务被默认设置为自动开机启动的(非必须启动)

OracleJobScheduler* -- Oracle 作业调度服务。此服务被默认设置为禁用状态(非必须启动)

OracleMTSRecoveryService -- 服务端控制。该服务允许数据库充当一个微软事务服务器 MTS、COM/COM+对象和分布式环境下的事务的资源管理器。恢复、闪回需要开启该服务(非必须启动)

OracleOraDb11g_home1ClrAgent -- Oracle 数据库.NET 扩展服务的一部分。(非必须启动)

OracleOraDb11g_home1TNSListener -- 监听器服务，服务只有在数据库需要远

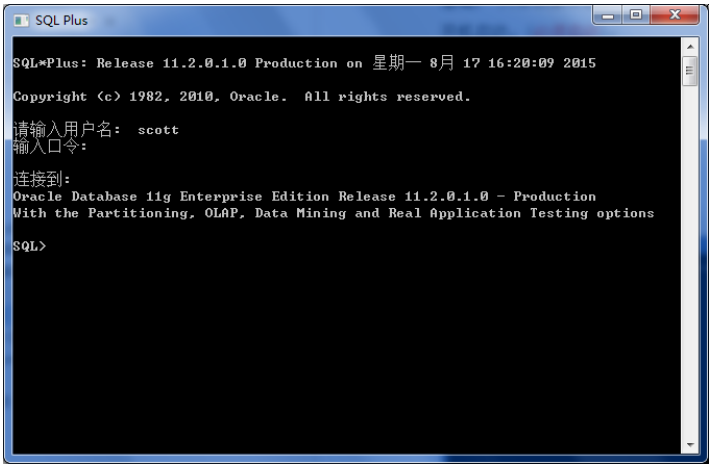
程访问或使用 SQL Developer 等工具的时候才需要，此服务被默认的设置为开机启动(非必须启动)

OracleService* -- 数据库服务，是 Oracle 核心服务该服务，是数据库启动的基础， 只有该服务启动，Oracle 数据库才能正常操作。此服务被默认的设置为开机启动。(必须启动)

3 连接 Oracle

3.1 SQL Plus 连接

打开 SQL Plus：



在上述界面中可以输入用户名，如在安装时解锁了的用户 scott,口令为：tiger

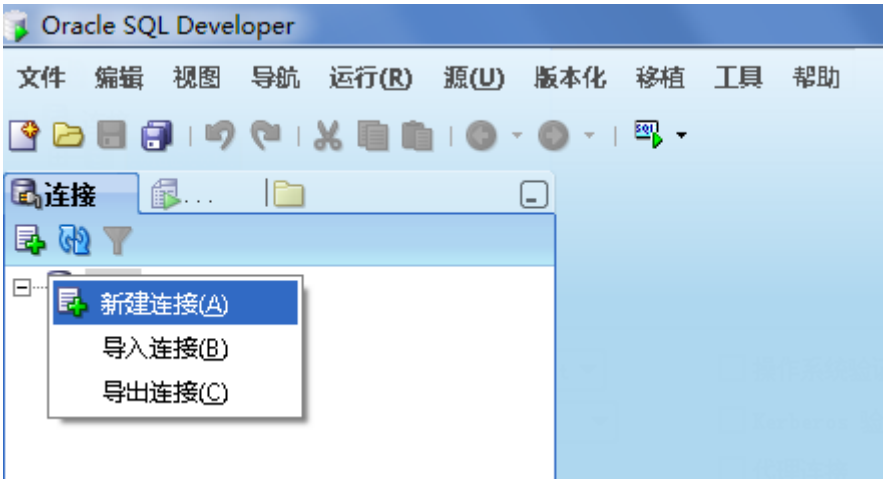
输入语句查询该用户下的对象：

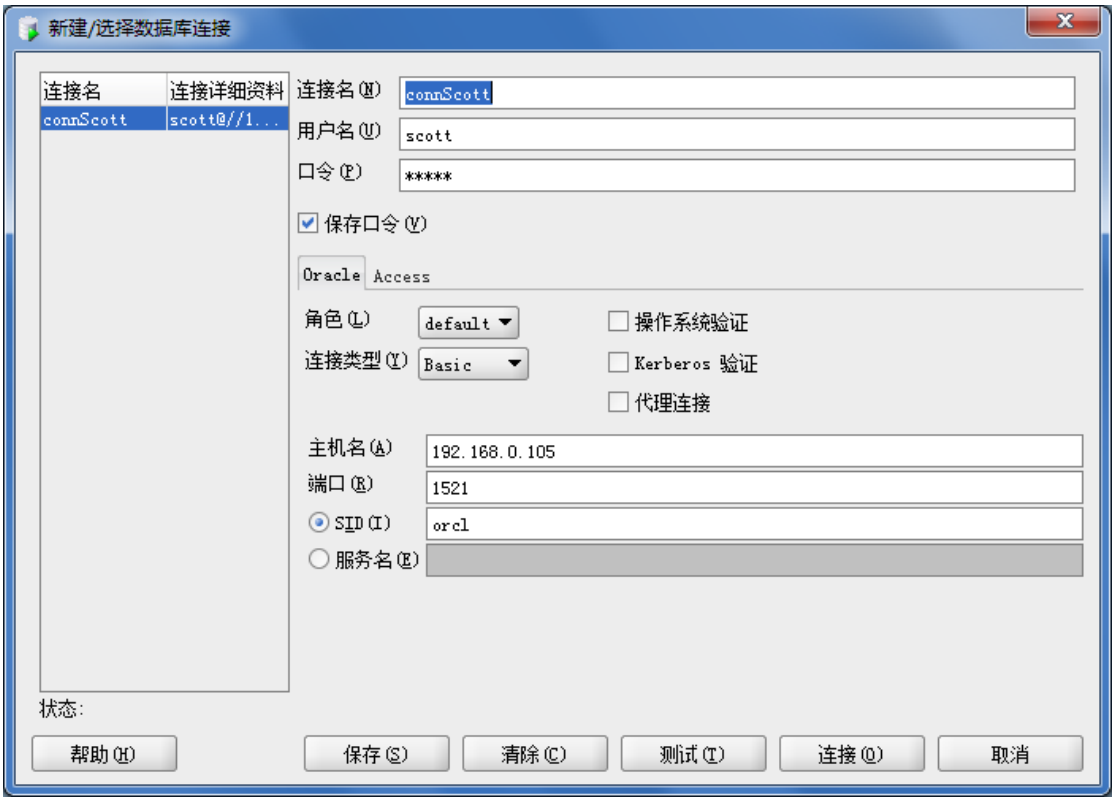
```
SQL> select table_name from tabs;

TABLE_NAME
------------------------------
SALGRADE
BONUS
EMP
DEPT
```

另外；也可以直接在命令行中输入 sqlplus scott/tiger 进入并登录

3.2 SQL Developer 连接

打开 SQL Developer ；在出现界面的左边右击鼠标，新建连接：

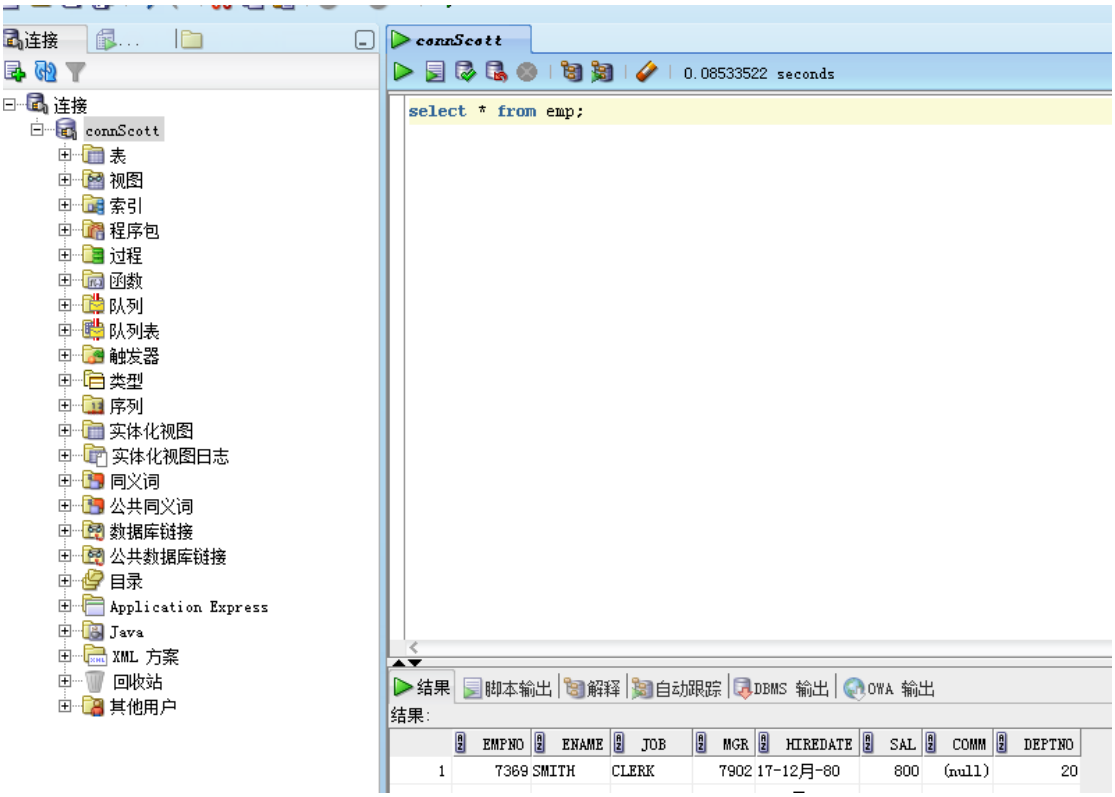




注意在上图中；

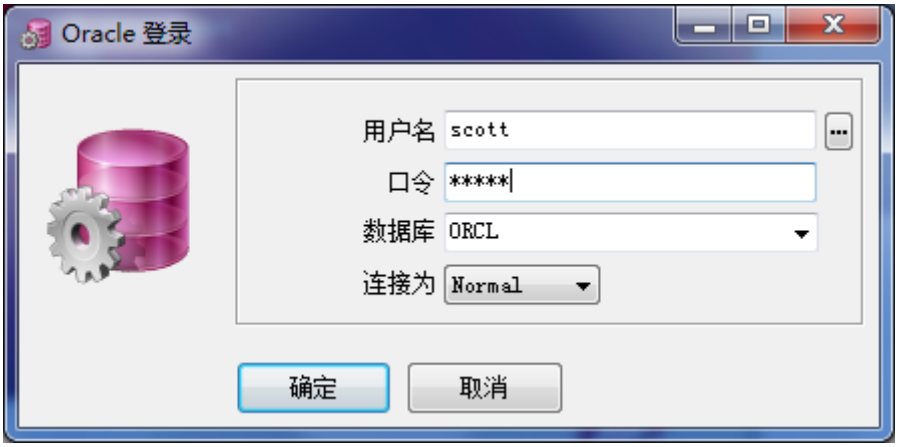
主机名：如果是本机的按照配置在网络管理中的服务的配置设置，可以为 localhost；如果是连接其它机器的数据库则指定其 ip；

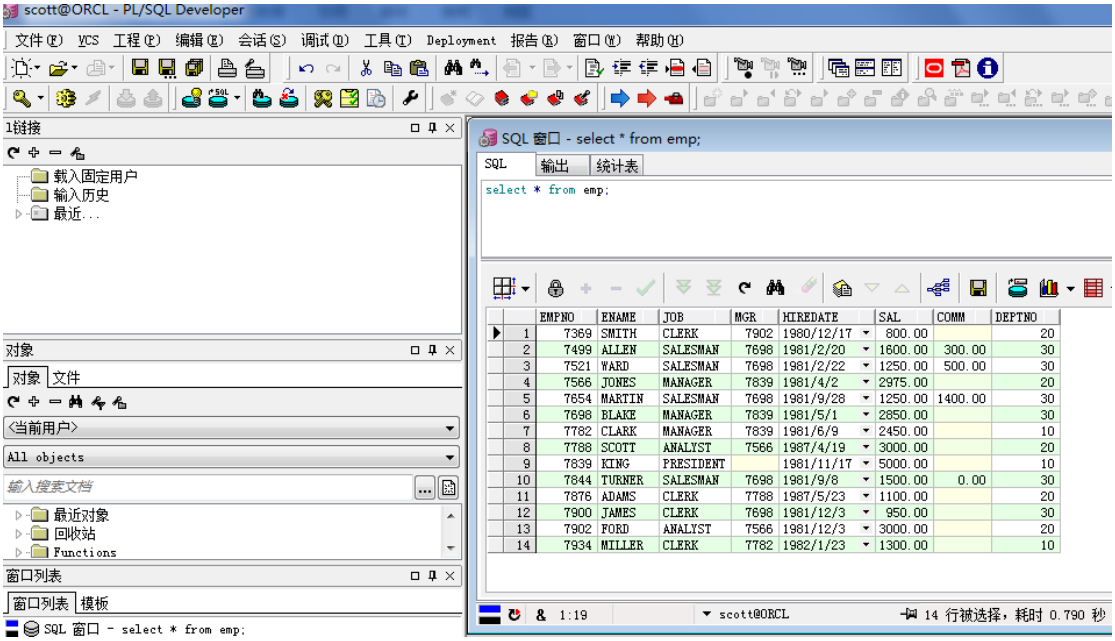
SID：是指定数据库服务器上的全局数据库名称，默认安装的话一般是 orcl



3.3 PLSQL Developer 连接

安装 PLSQL Develper；见《PLSQL Developer 安装及注册.docx》





3.4 Jdbc 连接

1、在安装目录下找到 oracle 的驱动包；

如下路径可以找到 oracle 的驱动包：

C:\oracle11g\product\11.2.0\dbhome_1\jdbc\lib 复制 ojdbc6.jar 到项目中进行连接测试；

2、新建 java 项目测试连接；

```java
public class Connect2Oracle {

    public static void main(String[] args) {
        Connection conn = null;
        Statement stat = null;
        try {
            Class.forName("oracle.jdbc.OracleDriver");
            String url = "jdbc:oracle:thin:@localhost:1521:orcl";
            String user = "scott";
            String password = "tiger";
            conn = DriverManager.getConnection(url, user, password);
            stat = conn.createStatement();
            ResultSet resultSet = stat.executeQuery("select * from emp");
            while(resultSet.next()){
                System.out.println("用户名为: " + resultSet.getString("ename")
                        + " 的用户的工资为: " + resultSet.getDouble("sal"));
            }
            resultSet.close();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                if(conn != null){
                    conn.close();
                }
                if(stat != null){
                    stat.close();
                }
```

4 SQL Plus 设置与常用命令

4.1 显示设置

```
-- 设置每行显示的最长字符数
set linesize 120

-- 设置一页显示的行数
set pagesize 20

-- 设置是否显示一页的记录数
set feedback on/off

-- 打开或取消oracle自带的输出方法dbms_output，并输出内容
set serveroutput on/off

-- 格式化列的内容：将列名对应的列的值格式化为四位数值长度
```

```
col 表中对应的列名 for 9999
column 表中对应的列名 format 9999


【示例】
-- 表明将 empno 列名对应的列值格式为 4 位长度的数值型
col empno for 9999


-- 格式化列的内容：将列名对应的列的值格式化为10位字母长度
col 表中对应的列名 for a10

【示例】
-- 表明将ename列名对应的列值格式为10位长度的字符型
col ename for a10
```

4.2 常用命令

命令	说明
show all	查看系统所有变量值
show user	显示当前连接用户
show error	显示错误
desc 表名	显示表的结构；如：desc emp
/* */ --	多行注释 单行注释
/	执行缓冲区中的语句
ed	打开默认编辑器，Windows 系统中默认是 notepad.exe，把缓冲区中最后一条 SQL 语句调入 afiedt.buf 文件中进行编辑（如果提示没有 afiedt.buf 请使用管理员身份打开 SLQ Plus）；常用于语句比较长需要修改时。
spool 文件地址 spool 文件地址 append	假脱机命令；将命令行的内容（从设置后开始的命令行内容）记录到文本。添加

spool off	append 的意思是在原有的文本内容上追加后续的命令行的内容；需要注意的是所有的这些内容都将在 spool off 之后才记录。 如： spool d:\test\test.txt spool d:\test\test.sql append spool off
clear screen 或者 host cls	清屏
exit	退出 SQL Plus

5 表空间

表空间是数据库中最大的逻辑单位，Oracle 数据库采用表空间将相关的逻辑组件组合在一起，一个 Oracle 数据库至少包含一个表空间。每个表空间由一个或多个数据文件组成，一个数据文件只能与一个表空间相联系。

在每一个数据库中都有一个名为 SYSTEM 的表空间，即系统表空间，该表空间是在创建数据库或数据库安装时自动创建的，用于存储系统的数据字典表、程序单元、过程、函数、包和触发器等。

5.1 表空间类型

永久性表空间：一般保存表、视图、过程和索引等的数据

临时性表空间：只用于保存系统中短期活动的数据

撤销表空间：用来帮助回退未提交的事务数据

5.2 操作与运用

创建表空间

【语法】

```
CREATE TABLESPACE 表空间名
   DATAFILE '数据文件路径' SIZE 大小
   [AUTOEXTEND ON] [NEXT 大小]
   [MAXSIZE 大小];
```

【说明】[]里面内容可选项；数据文件路径中若包含目录需要先创建

SIZE 为初始表空间大小，单位为 K 或者 M

AUTOEXTEND ON 是否自动扩展

NEXT 为文件满了后扩展大小

MAXSIZE 为文件最大大小，值为数值或 UNLIMITED（表示不限大小）

【示例】

```
CREATE TABLESPACE test_ts
   DATAFILE 'd:\oracle_data\test01.dbf' SIZE 10M
   AUTOEXTEND ON;
```

查询表空间

```
--管理员角色查看表空间
SELECT file_name,tablespace_name,bytes,autoextensible
FROM dba_data_files
WHERE tablespace_name='test_TS';
```

修改表空间

【语法】

```
ALTER TABLESPACE 表空间名
   ADD DATAFILE '文件路径' SIZE 大小
   [AUTOEXTEND ON] [NEXT 大小]
   [MAXSIZE 大小];
```

【示例】

```
ALTER TABLESPACE test_ts
   ADD DATAFILE 'd:\oracle_data\test02.DBF' SIZE 5M
```

```
    AUTOEXTEND ON;
```

删除表空间

【语法】

```
DROP TABLESPACE 表空间名;
DROP TABLESPACE 表空间名 INCLUDING CONTENTS AND DATAFILES;
```

【说明】

第一个删除语句只删除表空间；第二个删除语句则删除表空间及数据文件

【示例】

```
DROP TABLESPACE test_ts;
DROP TABLESPACE test_ts INCLUDING CONTENTS AND DATAFILES;
```

6 数据库用户

6.1 系统常见用户

用户	说明
sys	超级用户，主要用来维护系统信息和管理实例，以 SYSDBA 或 SYSOPER 角色登录。密码为在安装时设置的管理口令，如一般设置为：orcl
system	默认的系统管理员，拥有 DBA 权限，通常用来管理 Oracle 数据库的用户、权限和存储，以 Normal 方式登录。密码为在安装时设置的管理口令，如一般设置为：orcl
scott	示范用户，使用 users 表空间。一般该用

	户默认密码为 tiger

6.2 用户管理

Oracle 中有个模式（schema）的概念，它是用户的所有数据库对象的集合；一般在创建用户的同时会自动创建一个这样的模式，名称和用户名称一样。

6.2.1 查询系统用户

```
select * from all_users;
或
select * from dba_users; --更详细的用户信息
```

6.2.2 解锁用户

【语法】

```
ALTER USER 用户名 ACCOUNT UNLOCK;
```

【示例】解锁 hr 用户

```
alter user hr account unlock;
```

6.2.3 创建用户

【语法】

```
CREATE USER 用户名 IDENTIFIED BY 密码
   DEFAULT TABLESPACE 表空间;
```

【示例】

```
CREATE USER test IDENTIFIED BY test
   DEFAULT TABLESPACE test_ts
```

```
    TEMPORARY TABLESPACE temp;
```

6.2.4 修改用户密码

【语法】

```
ALTER USER 用户名 identified by 密码
```

【示例】

```
ALTER USER test identified by it;
```

6.2.5 删除用户

【语法】

```
DROP USER 用户名 CASCADE;
```

【示例】

```
DROP USER test CASCADE;
```

7 DCL 数据控制语言

7.1 授予

【语法 1】

```
GRANT 角色权限（角色）[,角色权限] TO 用户;
```

【示例 1】

```
--授予CONNECT和RESOURCE两个角色
GRANT connect, resource TO test;
```

【备注】使用如下语句可以查看 resource 角色下的权限

```
SELECT * FROM DBA_SYS_PRIVS WHERE GRANTEE='RESOURCE'
```

【语法 2】

```
GRANT 操作 ON 模式.对象  TO 用户;
```

【示例 2】

```
--允许用户查看、更新 EMP 表中的记录
GRANT select,update ON SCOTT.emp TO  test;


--查看当前用户的系统权限
select * from user_sys_privs;

--查看当前用户的对象权限
select * from user_tab_privs;

--查看当前用户的所有角色
select * from user_role_privs;
```

7.2 撤销

【语法 1】

```
REVOKE 角色权限（角色）[,角色权限] FROM 用户;
```

【示例 1】

```
--撤销CONNECT和RESOURCE两个角色
REVOKE connect, resource FROM test;
```

【语法 2】

```
REVOKE 操作 ON 模式.对象  FROM 用户;
```

【示例 2】

```
--撤销用户查看、更新 EMP 表中的记录的操作
REVOKE select,update ON SCOTT.emp FROM  test;
```

8 DDL 数据定义语言

8.1 创建表

【语法】

```
CREATE TABLE <table_name>(
column1 DATATYPE [NOT NULL] [PRIMARY KEY],
column2 DATATYPE [NOT NULL],
...
[constraint <约束名> 约束类型 (要约束的字段)
... ] );
```

【说明】

```
DATATYPE --是 Oracle 的数据类型
NUT NULL --可不可以允许资料有空的（尚未有资料填入）
PRIMARY KEY --是本表的主键
constraint --是对表里的字段添加约束.(约束类型有
            Check,Unique,Primary key,not null,Foreign key);
```

【示例】

```
create table t_student(
```

```
s_id number(8) PRIMARY KEY,
s_name varchar2(20) not null,
s_sex varchar2(8),
clsid number(8),
constraint u_1 unique(s_name),
constraint c_1 check (s_sex in ('MALE','FEMALE'))
);
```

--从现有的表创建表及复制其数据

【语法】

```
CREATE TABLE <table_name> as <SELECT 语句>
```

【示例】

```
create table emp as select * from scott.emp;

create table emp as select empno,ename from scott.emp --表结构只有
empno 和 ename 两个字段及该两字段对应的数据

--如果只复制表的结构不复制表的数据则:
create table emp as select * from scott.emp where 1=2;
```

8.2 修改表

【语法 1】向表中添加新字段

```
ALTER TABLE <table_name> ADD (字段 1 类型 [NOT NULL],
字段 2 类型 [NOT NULL] ... );
```

【示例 1】

```
alter table t_student add (s_age number(3),s_address varchar2(20));
```

【语法 2】修改表中字段

```
ALTER TABLE <table_name> MODIFY(字段 1 类型,字段 2 类型 ... );
```

【示例 2】

```
alter table t_student modify(s_name varchar2(50),s_address
varchar2(100));
```

【语法 3】删除表中字段

```
ALTER TABLE <table_name> DROP(字段 1,字段 2... );
```

【示例 3】

```
alter table t_student drop(s_age,s_address);
```

【语法 4】修改表字段名称

```
ALTER TABLE <table_name> RENAME COLUMN 原字段名称 TO 新字段名称;
```

【示例 4】

```
alter table t_student rename column s_id to s_no;
```

8.3 删除表

【语法 1】

```
--删除表结构及数据（删除后可在回收站查看并恢复）
DROP TABLE <table_name>;
--删除表结构及数据（删除后不可在回收站查看并恢复）
DROP TABLE <table_name> PURGE;
```

【示例 1】

```
drop table t_student;
```

8.4 回收站

8.4.1 查看回收站

```
--查看回收站
show recyclebin; 或 select * from recyclebin;
```

8.4.2 清空回收站

```
--清空回收站
purge recyclebin;
```

8.5 oracle 数据类型

数据类型	描述
VARCHAR2(size)	可变长度的字符串,其最大长度为 size 个字节;size 的最大值是 4000,而最小值是 1;你必须指定一个 VARCHAR2 的 size;
NVARCHAR2(size)	可变长度的字符串,依据所选的国家字符集,其最大长度为 size 个字符或字节;size 的最大值取决于储存每个字符所需的字节数,其上限为 4000;你必须指定一个 NVARCHAR2 的 size;

NUMBER(p,s)	精度为 p 并且数值范围为 s 的数值;精度 p 的范围从 1 到 38;数值范围 s 的范围是从-84 到 127; 例如:NUMBER(5,2) 表示整数部分最大 3 位,小数部分为 2 位; NUMBER(5,-2) 表示数的整数部分最大为 7 其中对整数的倒数 2 位为 0,前面的取整。NUMBER 表示使用默认值,即等同于 NUMBER(5);
LONG	可变长度的字符数据,其长度可达 2G 个字节;
DATE	有效日期范围从公元前 4712 年 1 月 1 日到公元后 9999 年 12 月 31 日
RAW(size)	长度为 size 字节的原始二进制数据,size 的最大值为 2000 字节; 你必须为 RAW 指定一个 size;
LONG RAW	可变长度的原始二进制数据, 其最长可达 2G 字节;
CHAR(size)	固定长度的字符数据,其长度为 size 个字节;size 的最大值是 2000 字节,而最小值和默认值是 1;
NCHAR(size)	也是固定长度。根据 Unicode 标准定义
CLOB	一个字符大型对象,可容纳单字节的字符;不支持宽度不等的字符集;最大为 4G 字节
NCLOB	一个字符大型对象,可容纳单字节的字符;不支持宽度不等的字符集;最大为 4G 字节;储存国家字符集
BLOB	一个二进制大型对象;最大 4G 字节
BFILE	包含一个大型二进制文件的定位器,其储存在数据库的外面; 使得可以以字符流 I/O 访问存在数据库服务器上的外部 LOB;最大大小为 4G 字节.

9 DML 数据操作语言

9.1 新增

【语法 1】

```
INSERT INTO table_name (column1,column2,...)
VALUES ( value1,value2, ...);
```

【示例 1】

```
insert into emp (empno,ename) values(1111,'test');
```

【语法 2】

```
INSERT INTO <table_name> <SELECT 语句>;
```

【示例 2】

```
create table t1 as select * from emp where 1=2;
insert into t1 select * from emp where sal>2000;
```

9.2 修改

【语法 1】

```
UPDATE table_name SET column1=new value,column2=new value,...
WHERE <条件>;
```

【示例 1】

```
update emp set sal=3000 where ename='test';
```

9.3 查询

9.3.1 伪表 dual

DUAL 是一个虚拟表，用来构成 select 的语法规则，oracle 保证 dual 里面永远只有一条记录。以用它来做很多事情，如：

1. 查看当前用户

```
select user from dual;
```

2. 用来调用系统函数

```
--查询系统的当前时间并格式化
select to_char(sysdate,'yyyy-mm-dd hh24:mi:ss') from dual;
```

3. 得到序列的下一个值或当前值

```
--获得序列seq的下一个值
select seq.nextval from dual;

--获得序列seq的当前值
select seq.currval from dual;
```

4. 可以用做计算器

```
select 2*8 from dual;
```

9.3.2 伪列 rowid

rowid 是物理结构上的，在每条记录 insert 到数据库中时，都会有一个唯一的物理记录，同一条记录在不同查询中对应的 rowid 相同。

【用法】

```
SELECT ROWID,字段名... FROM 表名;
```

【示例】

```
select rowid, emp.* from emp;
```

9.3.3 伪列 rownum

rownum 是根据 sql 查询出的结果给每行分配一个逻辑编号；每次的查询都会有不同的编号。编号从 1 开始。

【用法】

```
SELECT ROWNUM,字段名... FROM 表名;
```

【注意】

ROWNUM 不能使用大于号“>”

即 select rownum, emp.* from emp where rownum > 2 是不对的，没有任何结果

【示例】

```
select rownum, emp.* from emp;

/* 关于分页：由于不能使用>，所以为了达到分页目的得如下执行；如获取第 2 页数据（每页 3 条）*/
select * from (select rownum r,emp.* from emp where rownum < 7) where r > 3;


/* 关于排序：由于 rownum 是查询结果的行编号，排序后这个编号便有可能被打乱，如果需要该编号和排序的结果列表序号保持一致可以如下执行*/
select rownum,t.* from (select empno,ename from emp order by empno desc) t;
```

9.3.4 连接查询

准备查询数据，将 scott 用户下的 dept 表复制到 test 用户下。

使用 sys 用户登录系统；替 test 用户创建 dept 表，表结构和数据来自 scott.dept。

```
--执行语句如下
create table test.dept as select * from scott.dept;
```

1、 等值查询

```
--查询emp表中各用户对应的部门名称
select empno,ename,dname from emp,dept where emp.deptno=dept.deptno;

--练习：按部门统计员工的人数，要求显示部门号、部门名称、和部门人数
select d.deptno,d.dname,count(e.empno) from dept d,emp e
where d.deptno=e.deptno
group by d.deptno,d.dname;
```

2、**左外/右外连接查询**：左外连接是在等号左边的集合，无论条件是否成立均在结果集合，写法就是在等号右边使用(+)，这个写法是 oracle 专用的，如果需要全数据库类型通用应该使用 left join）

```
--按部门统计员工的人数，要求显示部门号、部门名称、和部门人数，部门下没有人的也将显示
select d.deptno,d.dname,count(e.empno) from dept d,emp e
where d.deptno=e.deptno(+) group by d.deptno,d.dname;

--上述语句的通用数据库写法(left join方式)
select d.deptno,d.dname,count(e.empno) from dept d left join emp e
on d.deptno=e.deptno group by d.deptno,d.dname;
```

3、**自连接查询**：查询的 2 张表是同一张表，一般是该表的字段之间存在上下级关系

```
--查询员工和老板的上下级关系
select e.ename || ' 的老板是: '|| b.ename from emp e,emp b
where e.mgr=b.empno;
```

【注意】上述查询语句中的||表示为字符的连接

9.3.5 组合查询

1、 计算部门工资总和，最高工资，最低工资

```
select deptno,sum(sal),max(sal),min(sal) from emp group by deptno;
```

2、**部门平均工资**

```
--查询部门的平均工资
select deptno,avg(sal) from emp group by deptno;

--查询平均工资大于2000的部门，并按照平均工资降序排序
select deptno,avg(sal) 平均工资 from emp
group by deptno
having avg(sal)>2000
order by 平均工资 desc ;

--查询除了20部门以外，平均工资大于2000的部门
select deptno,avg(sal) from emp
where deptno <> 20
group by deptno
having avg(sal)>2000;
```

【注意】SQL 语句中的各子句执行顺序：

from->where->group by->having->select->order by

3、 子查询：将子查询放入括号中；group by 后不能使用子查询；select、from、where 后面都可以使用子查询；可以将子查询看作一张新表

```

```

```
--select后面的子查询
select (select dname from dept where deptno=10),ename from emp where
deptno=10;


--from后面的子查询
select * from (select ename,sal from emp);


--将子查询视为一个表
select e.ename,e.sal from (select ename,sal from emp) e;


--where后面的子查询：查询工资比10号部门员工中任意一个员工的工资低的
员工信息
select * from emp where sal < (select min(sal) from emp where deptno=10);
```

4、**其它查询**

```
--查询姓名是5个字符的员工，且第二个字符是C，使用_只匹配一个字符并且
不能标识0或多个字符
select * from emp where ename like '_C___';



--查询员工姓名中含有'_'的员工，使用\转义字符
select * from emp where ename like '%\_%' escape '\';
```

9.4 删除

```
--根据条件删除表数据
delete from emp where empno=0000


--清空表数据（表还在），不写日志，省资源，效率高，属于数据定义语言
```

```
--先创建要清空数据的表
create table myemp as select * from emp;

--清空表数据
truncate table myemp;
```

10 TCL 事务控制语言

10.1 提交

事务的提交比较简单；直接在执行 DML 语句后进行提交即可，如果不提交事务则刚刚通过 DML 语句进行修改的内容还未保存到数据库中，只在当前用户的连接会话中有效。要永久变更数据需要显示地执行提交、回滚或者退出当前回话（如退出 sqlplus）。

提交的命令为：commit;

10.2 保存点与回滚

保存点 savepoint 一般与回滚 rollback 配合使用。在设置了 savepoint 后事务的粒度可以控制的更加细化，可以回滚到特定的保存点。

【语法】保存点 savepoint

```
SAVEPOINT <savepoint_name>;
```

【示例】

```
--创建一个保存点，名称为 a
savepoint a;
```

【注意】当创建保存点之后执行的 DML 操作，可以进行回滚，而保存点之前未提交的 DML 操作不受影响。

【语法】回滚

```
ROLLBACK [TO savepoint];
```

【示例】

--回滚到保存点 a,即在保存点 a 之后的所有未提交的 DML 都无效。

```
rollback to a;
```

/*保存点与回滚完整示例*/

```
--1、创建保存点a
savepoint a;
--2、插入emp数据 it1
insert into emp(empno,ename) values(1234,'it1');
--3、创建保存点b
savepoint b;
--4、插入emp数据 it2
insert into emp(empno,ename) values(1235,'it2');
--5、查看emp表数据，存在it1、it2两条数据
select ename from emp;
--6、回滚到保存点b，即it2数据将消失
rollback to b;
--7、查看emp表数据，存在it1的数据，it2已不在
select ename from emp;
--8、提交数据
commit;
--9、查看emp表数据，存在it1的数据
select ename from emp;
--10、回滚到保存点a，将报错保存点不存在的错误信息
rollback to a;
```

11 运算符

11.1 算术运算符

+、-、*、/

11.2 比较（关系）运算符

=、!=、<>、< 、 > 、 <= 、 >= 、 between...and... 、in 、like 、is null

11.3 逻辑运算符

AND(逻辑与)，表示两个条件必须同时满足

OR(逻辑或)，表示两个条件中有一个条件满足即可

NOT(逻辑非)，返回与某条件相反的结果

11.4 连接运算符

||

【示例】

```
select '工号为：' || empno || '的员工的姓名为：'|| ename from emp;
```

11.5 集合运算符

union（并集无重复）

union all（并集有重复）

intersect（交集，共有部分）

minus（减集，第一个查询具有，第二个查询不具有的数据）

【注意】：列数相关，对应列的数据类型兼容，不能含有 Long 类型的列，第一个

select 语句的列或别名作为结果标题

```
--union(并集将去重复)
select * from emp where deptno=10
union
select * from emp where deptno=20;

--intersect(交集) 查询工资即属于1000~2000区间和1500~2500区间的工资
select ename,sal from emp where sal between 1000 and 2000
intersect
select ename,sal from emp where sal between 1500 and 2500;

--minus(减集)
select ename,sal from emp where sal between 1000 and 2000
minus
select ename,sal from emp where sal between 1500 and 2500;
```

11.6 运算符优先级

优先级	运算符
1	算术运算符
2	连接符
3	比较符
4	IS[NOT]NULL, LIKE, [NOT]IN
5	[NOT] BETWEEN
6	NOT
7	AND
8	OR

可以使用括号改变优先级顺序；OR 的优先级最低，算术运算符的优先级最高。

12 常用函数

12.1 数值型函数

round(x[,y])

【功能】返回四舍五入后的值

【参数】x,y，数字型表达式,如果 y 不为整数则截取 y 整数部分，如果 y>0 则四舍五入为 y 位小数，如果 y 小于 0 则四舍五入到小数点向左第 y 位。

【返回】数字

【示例】

```
select   round(5555.6666,2.1),round(5555.6666,-2.6),round(5555.6666)
from dual;
```

返回：　5555.67　　，　5600　　，　5556

trunc(x[,y])

【功能】返回 x 按精度 y 截取后的值

【参数】x,y，数字型表达式,如果 y 不为整数则截取 y 整数部分，如果 y>0 则截取到 y 位小数，如果 y 小于 0 则截取到小数点向左第 y 位，小数前其它数据用 0 表示。

【返回】数字

【示例】

```
select trunc(5555.66666,2.1),
trunc(5555.66666,-2.6),trunc(5555.033333)  from dual;
```

返回：5555.66　　5500　　5555

12.2 字符型函数

LENGTH(c1)

【功能】返回字符串的长度;

【说明】多字节符(汉字、全角符等)，按 1 个字符计算

【参数】C1 字符串

【返回】数值型

【示例】

```
select length('甲骨文'),length('test 甲骨文') from dual;


LENGTH('甲骨文') LENGTH('test 甲骨文')
---------------- ----------------------
               4                     10
```

LPAD(c1,n[,c2])、RPAD(c1,n[,c2])

【功能】在字符串 c1 的左（右）边用字符串 c2 填充，直到长度为 n 时为止

【说明】如果 c1 长度大于 n，则返回 c1 左边 n 个字符

【参数】C1 字符串

n 追加后字符总长度

c2 追加字符串,默认为空格

【返回】字符型

【示例】

```
select lpad('test',10,'*'),rpad('test',10,'*') from dual;
```

REPLACE(c1,c2[,c3])

【功能】将字符表达式值中，部分相同字符串，替换成新的字符串

【参数】

c1 希望被替换的字符或变量

c2 被替换的字符串

c3 要替换的字符串，默认为空(即删除之意，不是空格)

【返回】字符型

【示例】

```
select replace('he love you','he','i') from dual;
```

SUBSTR(c1,n1[,n2])

【功能】取子字符串

【说明】多字节符(汉字、全角符等)，按 1 个字符计算

【参数】在字符表达式 c1 里，从 n1 开始取 n2 个字符;若不指定 n2,则从第 n1 个字符直到结束的字串.

【返回】字符型

【示例】

```
select substr('123456789',4,4),substr('123456789',3) from dual;
```

12.3 日期函数

sysdate

【功能】: 返回当前日期。

【参数】: 没有参数，没有括号

【返回】: 日期

【示例】select sysdate from dual;

add_months(d1,n1)

【功能】: 返回在日期 d1 基础上再加 n1 个月后新的日期。

【参数】: d1，日期型，n1 数字型

【返回】: 日期

【示例】select sysdate,add_months(sysdate,3) from dual;

months_between(d1,d2)

【功能】：返回日期 d1 到日期 d2 之间的月数。

【参数】：d1，d2 日期型

【返回】：数字

如果 d1>d2，则返回正数

如果 d1<d2，则返回负数

【示例】

```
select sysdate,
months_between(sysdate,to_date('2015-01-01','YYYY-MM-DD')) 距2015元旦,
months_between(sysdate,to_date('2016-01-01','YYYY-MM-DD')) 距2016元旦 from dual;
```

extract(c1 from d1)

【功能】：日期/时间 d1 中，参数(c1)的值

【参数】：d1 日期型(date)/日期时间型(timestamp),c1 为字符型(参数)

【参数表】：c1 对应的参数表详见示例

【返回】：字符

【示例】

```
select
extract(YEAR from timestamp '2015-5-1 12:26:18 ' ) 年,
extract(MONTH from timestamp '2015-5-1 12:26:18 ' ) 月,
extract(DAY from timestamp '2015-1-5 12:26:18 ' ) 日,
extract(hour from timestamp '2015-5-1 12:26:18 ' ) 小时,
extract(minute from timestamp '2015-5-1 12:26:18' ) 分钟,
extract(second from timestamp '2015-5-1 12:26:18 ' ) 秒
from dual;

select extract (YEAR from date '2015-5-1' ) from dual;

select sysdate 当前日期,
extract(YEAR from sysdate ) 年,
```

```
extract(MONTH from sysdate ) 月,
extract(DAY from sysdate ) 日
from dual;

--如下语句也可获取年份、月份等
select to_number(to_char(sysdate,'yyyy')) from dual;
```

12.4 转换函数

TO_CHAR(x[[,c2],C3])

【功能】将日期或数据转换为 char 数据类型

【参数】

x 是一个 date 或 number 数据类型。

c2 为格式参数

c3 为 NLS 设置参数

【返回】varchar2 字符型

【示例】

```
select to_char（sysdate,'YYYY-MM-DD HH24:MI:SS'） FROM dual;
select to_char（1210.7, '$9,999.00'） FROM dual;
```

TO_DATE(X[,c2[,c3]])

【功能】将字符串 X 转化为日期型

【参数】c2,c3,字符型，参照 to_char()

【返回】字符串

如果 x 格式为日期型(date)格式时，则相同表达：date x

如果 x 格式为日期时间型(timestamp)格式时，则相同表达：timestamp x

【示例】

```
select to_date('201212','yyyymm'),
to_date('2012.12.20','yyyy.mm.dd'),
```

```
(date '2012-12-20') XXdate,
to_date('2012-12-20 12:31:30','yyyy-mm-dd hh24:mi:ss'),
to_timestamp('2012-12-20 12:31:30','yyyy-mm-dd hh24:mi:ss'),
(timestamp '2012-12-20 12:31:30') XXtimestamp
from dual;
```

TO_NUMBER(X[[,c2],c3])

【功能】将字符串 X 转化为数字型

【参数】c2,c3,字符型

【返回】数字串

【示例】

```
select TO_NUMBER('201212') + 3,TO_NUMBER('450.05') + 1 from dual;
--等同上述结果
select '201212' + 3 from dual;
```

12.5 聚合函数

sum:求和

avg:求平均数

count：计数

max：求最大值

min：求最小值

12.6 分析函数

分析函数中了解 rank()/dense_rank()/row_number()的使用：

```
--查询部门的员工工种情况，并在部门内重新进行排序；PARTITION BY类似
group by,根据ORDER BY排序字段的值重新由1开始排序。

--RANK 使用相同排序排名一样，后继数据空出排名；即有2个排序为1的，那
```

```
么接下来的排序号则为3
select deptno,ename,job,rank() over(partition by deptno order by job)
as myRank from emp e;

--DENSE_RANK使用，使用相同排序排名一样，后继数据不空出排名；即有2个
排序为1的，那么接下来的排序号则为2
select deptno,ename,job,dense_rank() over(partition by deptno order by
job) as myDenseRank from emp e;

--ROW_NUMBEK使用，不管排名是否一样，都按顺序排名；即有2个排序为1的，
那么排序号不会重现重复
select deptno,ename,job,row_number() over(partition by deptno order by
job) as myRowNumber from emp e;
```

12.7 其它函数

NVL()/NVL2()

【语法】NVL (expr1, expr2)

【功能】若 expr1 为 NULL，返回 expr2；expr1 不为 NULL，返回 expr1。注意两者的类型要一致

【示例】将员工的奖金如果是空的话则设置为 0

```
select ename,sal,comm,nvl(comm,0) from emp;
```

【语法】NVL2 (expr1, expr2, expr3)

【功能】expr1 不为 NULL，返回 expr2；expr2 为 NULL，返回 expr3。

expr2 和 expr3 类型不同的话，expr3 会转换为 expr2 的类型

【示例】

```
select ename,job,nvl2(job,'job 有值','job 无值') from emp;
```

decode(条件,值 1,翻译值 1,值 2,翻译值 2,...值 n,翻译值 n,缺省值)

【功能】根据条件返回相应值

【参数】c1, c2, ...,cn,字符型/数值型/日期型，必须类型相同或 null

注：值 1……n 不能为条件表达式,这种情况只能用 case when then end 解决

含义解释：

decode(条件,值 1,翻译值 1,值 2,翻译值 2,...值 n,翻译值 n,缺省值)

该函数的含义如下：

IF 条件=值 1 THEN

RETURN(翻译值 1)

ELSIF 条件=值 2 THEN

RETURN(翻译值 2)

.....

ELSIF 条件=值 n THEN

RETURN(翻译值 n)

ELSE

RETURN(缺省值)

END IF

【示例】根据员工的部门号，条件判断找到对应的部门名称

```
select
ename,deptno,decode(deptno,10,'ACCOUNTING',20,'RESEARCH',30,'SALES'
,'无部门') from emp;
```

13 视图

13.1 视图简介

视图是由一个或者多个表组成的虚拟表；那些用于产生视图的表叫做该视图的基表。视图不占用物理空间，这个也是相对概念，因为视图本身的定义语句还

是要存储在数据字典里的。视图只有逻辑定义。每次使用的时候只是重新执行SQL。一个视图也可以从另一个视图中产生。视图没有存储真正的数据，真正的数据还是存储在基表中。一般出于对基本的安全性和常用的查询语句会建立视图；并一般情况下不对视图进行新增、更新操作。

【语法】

```
--创建视图
CREATE [OR REPLACE] VIEW <view_name>
AS
<SELECT 语句>;



--删除视图
DROP VIEW <view_name> ;
```

13.2 视图操作

```
-- 授予test用户 创建视图 的权限
grant create view to test;

-- 登录test，创建视图
create or replace view v_emp
as
select empno,ename from emp;

--通过视图查询数据
select * from v_emp;

--通过视图添加数据,需要保证基表的其它数据项可以为空
insert into v_emp(empno,ename) values(3333,'test3');

--通过视图修改数据
```

```
update v_emp set ename='甲骨文3' where empno=3333;

--通过视图删除数据
delete from v_emp  where empno=3333;

--基于多个基表的视图，不建议使用视图进行增删改操作
create or replace view v_dept_emp
as
select dept.deptno,dept.dname,ename from emp inner join dept on
emp.deptno=dept.deptno;

--查询多个基表的视图
select * from v_dept_emp;

--创建基于视图的视图
create or replace view vv_emp
as
select ename from v_emp;

--查询基于视图的视图
select * from vv_emp;

--删除视图
drop view v_emp;
drop view v_dept_emp;
drop view vv_emp;
```

14 同义词

同义词是数据库模式对象的一个别名，经常用于简化对象访问和提高对象访问的安全性。在使用同义词时，Oracle 数据库将它翻译成对应模式对象的名字。与视图类似，同义词并不占用实际存储空间，只有在数据字典中保存了同义词的定义。在 Oracle 数据库中的大部分数据库对象，如表、视图、同义词、序列、存储过程等，数据库管理员都可以根据实际情况为他们定义同义词。隐藏对象名称和所有者。

14.1 私有同义词

私有 Oracle 同义词由创建它的用户所有；创建的用户需要具有 CREATE SYNONYM 权限。

【语法】

```
CREATE SYNONYM <synonym_name> for <tablename/viewname...>
```

【示例】

```
--管理员 授权用户test创建同义词的权限
grant create synonym to test;

--创建私有同义词
create synonym syn_emp for emp;
create synonym syn_v_emp for v_emp;--为视图v_emp创建私有同义词（别名）

--使用私有同义词
select empno,ename from syn_emp;
update syn_emp set ename='test5' where empno='1234';

--删除同义词
drop synonym syn_emp;
```

14.2 公有同义词

公有 Oracle 同义词由一个特殊的用户组 Public 所拥有。顾名思义，数据库中所有的用户都可以使用公有同义词。公有同义词往往用来标示一些比较普通的数据库对象，这些对象常需要引用。公有同义词一般由管理员用户创建及删除，普通用户需要创建及删除需要 create public synonym 和 drop public synonym 权限。

【语法】

```
CREATE PUBLIC SYNONYM <synonym_name> for <tablename/viewname...>

--登陆sys管理员用户，授权用户test创建、删除(公有的删除权限需要特别给
```

```
定)公有同义词权限
grant create public synonym,drop public synonym to test;
--revoke create public synonym,drop public synonym from test;

--登陆test用户创建公有同义词 conn test/test;
create public synonym syn_public_emp for emp;

--使用公有同义词
select * from syn_public_emp;

-- 登录system管理员 conn system/orcl; 创建test2并授权
--create user test2 identifiea by test2 default tablespace test_ts;
--grant connect,resource to test2;

--为其它用户test2授权使用公有同义词（需要给予使用表的权限）
grant select,update on test.emp to test2;
--revoke select,update on test.emp from test2;

--登陆test2用户下使用公有同义词syn_public_emp
select * from syn_public_emp;
update syn_public_emp set  ename='甲骨文5' where empno=5555;

--删除同义词
--登陆test，删除公有同义词
drop public synonym syn_public_emp;
```

15 索引

索引是建立在数据库表中的某些列的上面，是与表关联的，可提供快速访问数据方式，但会影响增删改的效率；常用类型（按逻辑分类）：单列索引和组合索引、唯一索引和非唯一索引。

什么时候要创建索引

（1）在经常需要搜索、主键、连接的列上

（2）表很大，记录内容分布范围很广

（3）在经常需要根据范围进行搜索的列上创建索引，因为索引已经排序，其指

定的范围是连续的

（4）在经常使用在 WHERE 子句中的列上面创建索引

什么时候不要创建索引

（1）表经常进行 INSERT/UPDATE/DELETE 操作

（2）表很小(记录超少)

（3）列名不经常作为连接条件或出现在 WHERE 子句中

（4）对于那些定义为 text, image 和 bit 数据类型的列不应该增加索引

15.1 创建索引

【语法】

CREATE [UNIQUE] INDEX <index_name> ON <table_name>(字段 [ASC|DESC]);

【说明】

UNIQUE --确保所有的索引列中的值都是可以区分的。

[ASC|DESC] --在列上按指定排序创建索引。

(创建索引的准则：

1.如果表里有几百行记录则可以对其创建索引(表里的记录行数越多索引的效果就越明显)。

2.不要试图对表创建两个或三个以上的索引。

3.为频繁使用的行创建索引。)

【示例】

```
--创建单列唯一索引，表中的列值将不允许重复
create unique index index_emp_empno on emp(empno);

--创建单列非唯一索引
create index index_emp_ename on emp(ename);
```

```
--创建组合列、唯一索引
create unique index index_emp_ename_job on emp(ename,job);

--创建组合列、非唯一索引
create index index_emp_job_sal on emp(job,sal);
```

15.2 删除索引

【语法】

```
DROP INDEX <index_name>;
```

【示例】

```
--删除索引
drop index index_emp_empno;
drop index index_emp_ename;
drop index index_emp_ename_job;
drop index index_emp_job_sal;
```

16 序列

序列是 oracle 提供的一个产生唯一数值型值的机制。通常用于表的主健值，序列只能保证唯一，不能保证连续。

16.1 创建序列

【语法】

```
CREATE SEQUENCE <sequencen_name>
[INCREMENT BY n]
```

[START WITH n]

[MAXVALUE n][MINVALUE n]

[CYCLE|NOCYCLE]

[CACHE n|NOCACHE];

INCREMENT BY n --表示序列每次增长的幅度;默认值为 1.

START WITH n --表示序列开始时的序列号。默认值为 1.

MAXVALUE n --表示序列可以生成的最大值(升序).

MINVALUE n --表示序列可以生成的最小值(降序).

CYCLE --表示序列到达最大值后，在重新开始生成序列.默认值为 NOCYCLE。

CACHE n--允许更快的生成序列.预先生成 n 个序列值到内存(如果没有使用完，那下次序列的值从内存最大值之后开始；所以 n 不应该设置太大)

【示例】

```
--创建递增序列
create sequence seq_test
increment by 1
start with 1
maxvalue 1000
nocycle;

--创建递减序列
create sequence seq_test2
increment by -1
start with 5
maxvalue 5
minvalue 1
nocycle;
```

16.2 序列使用

1、NEXTVAL 返回序列下一个值；第一次访问时，返回序列的初始值，后继每次调用时，按步长增加的值返回

【语法】

```
select <sequence_name>.nextval from dual;
```

【示例】

```
select seq_test.nextval from dual;
```

2、CURRVAL 返回序列的当前值.注意在刚建立序列后,序列的 CURRVAL 值为 NULL，所以不能直接使用。使用过 NEXTVAL 访问序列后才能使用

【语法】查看序列的当前值

```
select <sequence_name>.currval from dual;
```

【示例】

```
select seq_test.nextval from dual;
select seq_test.currval from dual;
```

运用序列

```
-- 创建序列
create sequence seq_emp_empno
start with 1000
increment by 1
maxvalue 9000
minvalue 1000
nocycle;

-- 使用序列作为主键插入emp表的empno列
insert into emp(empno,ename)
values(seq_emp_empno.nextval,'test1');
insert into emp(empno,ename)
```

```
 values(seq_emp_empno.nextval,'test2');

-- 查看emp表数据
select empno,ename from emp;

-- 查看当前序列的值
select seq_emp_empno.currval from dual;


--修改序列
alter sequence seq_emp_empno
maxvalue 9999
cycle;
```

16.3 删除序列

【语法】

```
DROP SEQUENCE <sequence_name>
```

【示例】

```
drop sequence seq_test;
```

16.4 序列与 sys_guid

sys_guid 和序列都可以作为主键值。

```
--使用SYS_GUIL函数，32位，由时间戳和机器标识符生成，保证唯一
select sys_guid() from dual;
```

17 分区表

17.1 分区表用途

分区表通过对分区列的判断，把分区列不同的记录，放到不同的分区中。分区完全对应用透明。Oracle 的分区表可以包括多个分区，每个分区都是一个独立的段（SEGMENT），可以存放到不同的表空间中。查询时可以通过查询表来访问各个分区中的数据，也可以通过在查询时直接指定分区的方法来进行查询。

分区表的优点：

（1）由于将数据分散到各个分区中，减少了数据损坏的可能性；

（2）可以对单独的分区进行备份和恢复；

（3）可以将分区映射到不同的物理磁盘上，来分散 IO；

（4）提高可管理性、可用性和性能。

数据量大的表，一般大于 2GB；数据有明显的界限划分；对于 Long 和 Long Raw 类型列不能使用分区。

17.2 分区表类型

一般包括范围分区，散列分区，列表分区、复合分区（范围-散列分区，范围-列表分区）、间隔分区和系统分区等。

17.2.1 范围分区

范围分区根据数据库表中某一字段的值的范围来划分分区。

【语法】

在 Create Table 语句后增加

```
PARTITION BY RANGE(column_name)
(
PARTITION part1 VALUES LESS THAN (range1) [TABLESPACE tbs1],
PARTITION part2 VALUES LESS THAN (range2) [TABLESPACE tbs2],
```

```
            ....
PARTITION partN VALUES LESS THAN (MAXVALUE) [TABLESPACE tbsN]
);
```

【说明】

MAXVALUE：当分区列值都不在设置的范围内时，新增数据将到这个分区中

【示例】

```
-- 创建表，并设置分区
create table myemp
( empno number(4) primary key,
  ename varchar2(10),
  hiredate date,
  sal    number(7,2),
  deptno number(2)
)
partition by range(sal)
(
  partition p1 values less than(1000),
  partition p2 values less than(2000),
  partition p3 values less than(maxvalue)
);

-- 插入数据
insert into myemp(empno,ename,hiredate,sal,deptno)
select empno,ename,hiredate,sal,deptno from emp;

-- 查看工资1000-2000的数据
select * from myemp partition(p2);

-- 删除工资小于1000的数据
delete from myemp partition(p1);

-- 查看数据
select * from myemp;
```

17.2.2列表分区

列表分区明确指定了根据某字段的某个具体值进行分区，而不是像范围分区那样根据字段的值范围来划分的。

【语法】

在 Create Table 语句后增加

```
PARTITION BY LIST(column_name)
(
PARTITION part1 VALUES (values_list1),
PARTITION part2 VALUES (values_list2),
           ....
PARTITION partN VALUES (DEFAULT)
);
```

其中：column_name 是以其为基础创建列表分区的列。

part1...partN 是分区的名称。

values_list 是对应分区的分区键值的列表。

DEFAULT 关键字允许存储前面的分区不能存储的记录。

【示例】

```
-- 创建表，并设置分区
create table myemp2
( empno number(4) primary key,
  ename varchar2(10),
  hiredate date,
  sal    number(7,2),
  deptno number(2)
)
partition by list(deptno)
(
  partition dept10 values(10),
  partition dept20 values(20),
  partition dept30 values(30),
  partition deptx  values(default)
```

```
);

-- 插入数据
insert into myemp2(empno,ename,hiredate,sal,deptno)
select empno,ename,hiredate,sal,deptno from emp;

-- 查看部门20的数据
select * from myemp2 partition(dept20);

-- 删除部门30的数据
delete from myemp2 partition(dept30);

-- 查看数据
select * from myemp2;
```

18 PL/SQL

pl/sql:块结构语言，是 sql（Structured Query Language）语言的一种扩展，结合了 oracle 过程语言（procedural language）进行使用。

pl/sql 块由三部分构成：声明部分、执行部分、异常部分。

PL/SQL 结构

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

18.1 运算符

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

18.2 变量与常量

数据类型：

常用标准类型： CHAR(CHARATER,NCHAR),VARCHAR2,NUMBER(P,S),DATE,BOOLEAN等。

属性类型： %TYPE 与 %ROWTYPE

%TYPE:可以用来定义数据变量的类型与已定义的数据变量（表中的列）一致。

%ROWTYPE:与某一数据库表的结构一致(修改数据库表结构,可以实时保持一致)；访问方式声明为 rowtype 的 变量名.字段名。

18.2.1基本类型

声明

【变量声明】

<变量名> 类型[:=初始值];

【示例】

```
name varchar2(20) := 'test';
```

【常量声明】

<变量名> CONSTANT 类型:=初始值;

【示例】

```
pi constant number(5,3):=3.14;
```

运用

```
/*定义常量或变量、赋值使用示例*/
DECLARE
      p_empno constant number(4):=7369;
      p_ename varchar2(10);
      p_sal number(7,2);
      p_comm number(7,2);
BEGIN
      --赋值方式一：使用select into给变量赋值
      select ename,sal into p_ename,p_sal from emp where empno =p_empno;

      --赋值方式二：使用赋值操作符“:=”给变量赋值
      p_comm:=500;

      --输出相关信息，DBMS_OUTPUT.PUT_LINE为具有输出功能的函数
      dbms_output.put_line('员工号:'|| p_empno||',姓名:'||
p_ename||',工资:'|| p_sal||',奖金:'|| p_comm);
END;
```

【注意】

dbms_output 是 oracle 提供的输出对象

put_line 是其一个方法，用于输出一个字符串

new_line 是其一个方法，用于输出新的一行（换行）

18.2.2%type 类型

声明

【声明】

变量名称 表名.字段%type;

【示例:】

--表示变量 name 的类型和 emp.ename 的类型相同

name emp.ename%type;

运用

```
/*定义常量或变量、赋值使用示例*/
DECLARE
      p_empno constant number(4):=7369;
      p_ename emp.ename%type;
      p_sal emp.sal%type;
      p_comm emp.comm%type;
BEGIN
      --赋值方式一：使用select into给变量赋值
      select ename,sal into p_ename,p_sal from emp where empno =
p_empno;

      --赋值方式二：使用赋值操作符“:=”给变量赋值
      p_comm:=500;

      --输出相关信息，DBMS_OUTPUT.PUT_LINE为具有输出功能的函数
      dbms_output.put_line('员工号:'|| p_empno||',姓名:'||
p_ename||',工资:'|| p_sal||',奖金:'|| p_comm);
END;
```

18.2.3%rowtype 类型

声明

【声明】

变量名称 表%rowtype;

【示例:】

--表示变量 test 的类型为 emp 表的行类型;也有 .empno; .ename; .sal ;等属性

test emp%rowtype;

运用

```
/*定义常量或变量、赋值使用示例*/
DECLARE
      p_empno constant number(4):=7369;
      emp_info emp%rowtype;
      p_comm emp.comm%type;
BEGIN
      --赋值方式一：使用select into给变量赋值
      select * into emp_info from emp where empno = p_empno;

      --赋值方式二：使用赋值操作符“:=”给变量赋值
      p_comm:=500;

      --输出相关信息，DBMS_OUTPUT.PUT_LINE为具有输出功能的函数
      dbms_output.put_line('员工号:'|| p_empno||',姓名:'||
emp_info.ename ||',工资:'|| emp_info.sal ||',奖金:'|| p_comm);
END;
```

18.3 控制语句

18.3.1条件语句

【语法】

```
IF <条件 1> THEN
    语句
[ELSIF <条件 2> THEN
   语句]
          .
          .
          .
[ELSIF <条件 n> THEN
   语句]

[ELSE
    语句]

END IF;
```

【示例】

```
/*
根据员工的工资判断其工资等级（工资大于等于 5000 为 A 级，工资大于等于 4000 为 B 级，工资大于等于 3000 为 C 级，工资大于等于 2000 为 D 级，其它为 E 级）
*/
DECLARE
     p_empno number(4):=7566;
     p_sal emp.sal%type;
BEGIN
     --用变量代替条件语句中的真值
     select sal into p_sal from emp where empno = p_empno;

     IF p_sal >= 5000 THEN
        dbms_output.put_line('员工号为：' || p_empno || '的员工的工资级别为：A级');
     ELSIF p_sal >= 4000 THEN
```

```
        dbms_output.put_line('员工号为：' || p_empno || '的员工的工
资级别为：B级');
    ELSIF p_sal >= 3000 THEN
        dbms_output.put_line('员工号为：' || p_empno || '的员工的工
资级别为：C级');
    ELSIF p_sal >= 2000 THEN
        dbms_output.put_line('员工号为：' || p_empno || '的员工的工
资级别为：D级');
    ELSE
        dbms_output.put_line('员工号为：' || p_empno || '的员工的工
资级别为：E级');
    END IF;
END;
```

18.3.2循环语句

1、LOOP

```
LOOP
   语句;
   EXIT WHEN <条件>
END LOOP;
```

【示例】

```
/*
计算 1-10 的总和
*/
DECLARE
    p_sum number(4):=0;
    p_num number(2):=1;
BEGIN
    LOOP
    p_sum := p_sum + p_num;
    p_num := p_num + 1;
    EXIT WHEN p_num > 10;
    END LOOP;
```

```
      dbms_output.put_line('1-10的总和为：' || p_sum);
END;
```

2、WHILE LOOP

```
WHILE <条件>
LOOP
   语句;
END LOOP;
```

【示例】

```
/*
计算 1-10 的总和
*/
DECLARE
      p_sum number(4):=0;
      p_num number(2):=1;
BEGIN
      WHILE p_num <= 10
      LOOP
      p_sum := p_sum + p_num;
      p_num := p_num + 1;
      END LOOP;
      dbms_output.put_line('1-10的总和为：' || p_sum);
END;
```

3、FOR

```
FOR <循环变量> IN [REVERSE] 下限..上限
LOOP
   语句;
END LOOP;
```

【说明】..两点表示范围，1..4 表示时将从 1 到 4 进行循环，起始（例如 1）写前边，REVERSE 表示反转，循环时变成从 4 到 1 进行。

【示例】

```
/*
计算 1-10 的总和
*/
DECLARE
      p_sum number(4):=0;
      p_num number(2):=1;
BEGIN
      FOR p_num IN 1..10
      LOOP
      p_sum := p_sum + p_num;
      END LOOP;
      dbms_output.put_line('1-10的总和为：' || p_sum);
END;
```

18.3.3顺序语句

指定顺序执行的语句；主要包括 null 语句。null 语句：是一个可执行语句，相当于一个占位符或不执行操作的空语句。主要用来提高程序语句的完整性和程序的可读性。

```
/*
输出 1-10 的数字但跳过数字 4
*/
DECLARE
      flag number(2):=0;
BEGIN
      WHILE flag < 10
      LOOP
      flag := flag + 1;
      if flag = 4 then
         null;-- 占位，不能去掉
```

```
      else
         dbms_output.put_line(flag);
      end if;
      END LOOP;
END;
```

18.4 异常处理

18.4.1异常语法

```
EXCEPTION
   WHEN <异常类型> THEN
            语句;
   WHEN OTHERS THEN
            语句;
```

常配套使用的函数：

SQLCODE 函数：返回错误代码，

SQLERRM 函数：返回错误信息

例如输出异常信息：DBMS_OUTPUT.PUT_LINE('其它异常,代码号:'||SQLCODE||',异常描述:'||SQLERRM);

18.4.2预定义异常

预定义异常指 PL/SQL 程序违反 Oracle 规则或超越系统限制时隐式引发(由 oracle 自动引发)。

常见的预定义异常：

CURSOR_ALREADY_OPEN 试图"OPEN"一个已经打开的游标

DUP_VAL_ON_INDEX 试图向有"UNIQUE"中插入重复的值

INVALID_CURSOR 试图对以关闭的游标进行操作

INVALID_NUMBER 在 SQL 语句中将字符转换成数字失败

LOGIN_DENIED 使用无效用户登陆

NO_DATA_FOUND 没有找到数据时

NOT_LOGIN_ON 没有登陆 Oracle 就发出命令时

PROGRAM_ERROR PL/SQL 存在诸如某个函数没有"RETURN"语句等内部问题

STORAGE_ERROR PL/SQL 耗尽内存或内存严重不足

TIMEOUT_ON_RESOURCE Oracle 等待资源期间发生超时

TOO_MANY_ROWS "SELECT INTO"返回多行时

VALUE_ERROR 当出现赋值错误

ZERO_DIVIDE 除数为零

【示例】

```
/*
预定义异常捕获并处理
*/
DECLARE
     p_result number(2);
BEGIN
     p_result := 1/0;
     dbms_output.put_line('没有异常！');
     EXCEPTION
     WHEN ZERO_DIVIDE THEN
          dbms_output.put_line('除数不能为0！代码为：'||sqlcode||',
异常信息为：' || sqlerrm);
     WHEN OTHERS THEN
          dbms_output.put_line('其它异常！代码为：'|| sqlcode || ',
异常信息为：' || sqlerrm);
END;
```

18.4.3 自定义异常

自定义异常：程序在运行过程中，根据业务等情况，认为非正常情况，可以自定义异常。对于这种异常，主要分三步来处理：

1、**定义相关异常**；在声明部分定义相关异常，

格式：<自定义异常名称> EXCEPTION;

2、**抛出异常**；在出现异常部分抛出异常，

格式：RAISE <异常名称>;

3、**处理异常**；在异常处理部分对异常进行处理，

格式：when <自定义异常名称> then ...,

处理异常也可以使用 RAISE_APPLICATION_ERROR(ERROR_NUMBER,ERROR_MESSAGE)存储过程进行处理，

其中参数 ERROR_NUMBER 取值为-20999~-20000 的负整数，参数 ERROR_MESSAGE 为异常文本消息。

【示例】

```
/*
判断 emp 中相应 empno 对应用户的奖金是否低于 500，如果低于则抛出并处理自定义异常
*/
DECLARE
      p_comm emp.comm%type;
      --自定义异常，名称为comm_exception
      comm_exception EXCEPTION;
BEGIN
      select nvl(comm,0) into p_comm from emp where empno=7499;
      if p_comm >= 500 then
       dbms_output.put_line('奖金大于等于500。');
      else
       RAISE comm_exception;
      end if;
      EXCEPTION
      WHEN comm_exception THEN
```

```
        RAISE_APPLICATION_ERROR(-20001,'奖金低于500，太少了！');
        --dbms_output.put_line('奖金低于500！');
    WHEN OTHERS THEN
        dbms_output.put_line('其它异常！代码为：'|| sqlcode || '，
异常信息为：' || sqlerrm);
END;
```

19 游标

19.1 显式游标

游标是映射在结果集中一行数据上的位置实体，使用游标，便可以访问结果集中的任意一行数据了，将游标放置到某行后，即可对该行数据进行操作；从上向下依次迭代结果集。

19.1.1游标语法

【定义语法】

CURSOR <游标名> IS <SELECT 语句> ;

【操作】

OPEN <游标名> --打开游标

FETCH <游标名> INTO 变量 1,变量 2,变量 3,....变量 n,;

或者

FETCH <游标名> INTO 行对象; --取出游标当前位置的值

CLOSE <游标名> --关闭游标

【属性】

%NOTFOUND --如果 FETCH 语句失败，则该属性为"TRUE"，否则为"FALSE";

%FOUND --如果 FETCH 语句成果，则该属性为"TRUE"，否则为"FALSE";

%ROWCOUNT --返回游标当前行的行数;

%ISOPEN --如果游标是开的则返回"TRUE"，否则为"FALSE";

19.1.2游标使用

1、使用游标显示员工表中所有的员工姓名、工作和工资

```
declare
    cursor cur_emp is select ename,job,sal from emp;
    p_ename emp.ename%type;
    p_job emp.job%type;
    p_sal emp.sal%type;
begin
    --打开游标
    open cur_emp;
    loop
        --取游标数据，从上往下移动一行
        fetch cur_emp into p_ename, p_job, p_sal;
        --如果下移后没有数据，则退出
        exit when cur_emp%notfound;
        --如果存在数据，则处理
        dbms_output.put_line('姓名为：' || p_ename ||'，工作为：' ||
p_job ||'，工资为：' || p_sal);
    end loop;
    --关闭游标
    close cur_emp;
end;
```

2、使用游标显示指定部门下的所有的员工姓名、工作和工资

代参数的游标

【定义】

CURSOR <游标名>(参数列表) IS <SELECT 语句>;

【示例】

```
declare
```

```
    cursor cur_emp(dno emp.deptno%type) is select ename,job,sal from
emp where deptno=dno;
    r_cur_emp cur_emp%rowtype;
begin
    --打开游标
    open cur_emp(20);
    loop
        --取游标数据，从上往下移动一行
        fetch cur_emp into r_cur_emp;
        --如果下移后没有数据，则退出
        exit when cur_emp%notfound;
        --如果存在数据，则处理
        dbms_output.put_line('姓名为：' || r_cur_emp.ename || '，工
作为：' || r_cur_emp.job || '，工资为：' || r_cur_emp.sal);
    end loop;
    --关闭游标
    close cur_emp;
end;
```

--**参考**：使用 while 循环实现

```
declare
  cursor cur_dept_emps(dno emp.deptno%type) is select ename,job,sal
from emp where deptno=dno;
  emp_info cur_dept_emps%rowtype;
begin
  open cur_dept_emps(20);
  fetch cur_dept_emps into emp_info;
  while cur_dept_emps%found
  loop
    dbms_output.put_line('员工姓名为：'||emp_info.ename||'，工作为：
'||emp_info.job||'，工资为：'||emp_info.sal);
    fetch cur_dept_emps into emp_info;
  end loop;
  close cur_dept_emps;
end;
```

--**参考**：使用 for 循环实现

```
declare
  cursor cur_dept_emps(dno emp.deptno%type) is select ename,job,sal
from emp where deptno=dno;
  emp_info cur_dept_emps%rowtype;
```

```
begin
  for emp_info in cur_dept_emps(20)
  loop
    if cur_dept_emps%found then
    dbms_output.put_line('员工姓名为：'||emp_info.ename||', 工作为：
'||emp_info.job||', 工资为：'||emp_info.sal);
    end if;
  end loop;
end;
```

3、使用游标按员工的工种涨工资,总裁 800，经理 600，其他人员 300

```
declare
    cursor cur_emp is select empno,job from emp;
    p_empno emp.empno%type;
    p_job emp.job%type;
begin
    --打开游标
    open cur_emp;
    loop
        --取游标数据，从上往下移动一行
        fetch cur_emp into p_empno, p_job;
        --如果下移后没有数据，则退出
        exit when cur_emp%notfound;
        --如果存在数据，则处理
        if 'PRESIDENT'= p_job then
          update emp set sal = sal + 800 where empno = p_empno;
        elsif 'MANAGER' = p_job then
          update emp set sal = sal + 600 where empno = p_empno;
        else
          update emp set sal = sal + 300 where empno = p_empno;
        end if;
    end loop;
    --关闭游标
    close cur_emp;
    --提交修改
    commit;
end;
```

19.2 隐式游标

当执行一个 SQL 语句时，Oracle 会自动创建一个隐式游标，隐式游标主要处理 DML 语句，该游标的名称是 sql。隐试游标不能进行"OPEN","CLOSE","FETCH"这些操作。

属性:

%NOTFOUND --如果 DML 语句没有影响到任何一行时，则该属性为"TRUE"，否则为"FALSE";

%FOUND --如果 DML 语句影响到一行或一行以上时，则该属性为"TRUE"，否则为"FALSE";

%ROWCOUNT --返回游标当最后一行的行数;

【示例】

```
/*
通过更新语句判断隐式游标的存在
*/


begin
    update emp set comm=comm + 300 where empno = 7369;
    if sql%notfound then
       dbms_output.put_line('empno对应的员工不存在');
    else
       dbms_output.put_line('empno对应的员工数为:' || sql%rowcount);
    end if;
end;
```

20 存储过程与存储函数

20.1 存储过程

存储过程是命名的 pl/sql 程序块，封装数据业务操作，具有模块化、可重用、可维护、更安全特点；并且可以被程序调用。一般有 4 类型的存储过程，分别为不带参数、带输入参数、带输出参数、带输入输出参数。

20.1.1语法

【语法】

```
CREATE [OR REPLACE] PROCEDURE <过程名>[(参数列表)] IS|AS
[局部变量声明]
BEGIN
   可执行语句
 [EXCEPTION
   异常处理语句]
END [<过程名>];
```

OR REPLACE：如果系统已存在该存储过程，将被替换

参数列表：**参数不需要声明长度**，可选

参数变量的类型:in 为默认类型,表示输入; out 表示只输出;in out 表示即输入又输出;

【调用方式】

在 PL/SQL 块中直接使用过程名;

在 PL/SQL 程序外使用 exec[ute] <过程名>[(参数列表)];

20.1.2无参存储过程

```
-- 授予test创建存储过程的权限
grant create procedure to test;

/*
使用无参存储过程，注意无参存储过程创建时不能使用()
*/

create or replace procedure pro_helloWorld
 as
begin
 dbms_output.put_line('Hello World.');
end;

-- 方式一：调用存储过程，可加可不加()
begin
 pro_helloWorld;
end;

-- 方式二：调用存储过程，可加可不加()
exec pro_helloWorld;
```

20.1.3有输入参数存储过程

```
/*
使用有输入参存储过程
*/
create or replace procedure pro_add_emp(
        p_empno in emp.empno%type,
        p_ename in varchar2,
        p_sal number
)
 as
```

```
begin
 --将输入参数对应的数据插入emp表
 insert into emp(empno, ename,sal) values(p_empno, p_ename, p_sal);
end;
/

-- 调用存储过程，向emp表插入新数据
begin
 pro_add_emp(2001,'test2001',3000);
 pro_add_emp(2002,'test2002',2000);
 pro_add_emp(2003,'test2003',4000);
end;
```

20.1.4有输出参数存储过程

```
/*
使用有输出参存储过程，计算 1 到 10 的总和并通过参数返回
*/
create or replace procedure pro_1to10_sum(
       p_sum out number
)
 as
 tem_sum number(4):=0;
begin
 for i in 1..10
 loop
  tem_sum := tem_sum + i;
 end loop;
 p_sum := tem_sum;
end;
/

-- 调用存储过程
declare
 p_sum number(4);
begin
 pro_1to10_sum(p_sum);
```

```
 dbms_output.put_line('1至10的和为：' || p_sum);
end;
```

20.1.5有输入输出参数存储过程

```
/*
使用有输入、输出参存储过程；根据 empno 查询该员工号对应的员工的姓名和
工资
*/
create or replace procedure pro_query_enameAndSal_by_empno(
        s_empno emp.empno%type,
        s_ename out emp.ename%type,
        s_sal out emp.sal%type
)
 as
begin
 select ename,sal into s_ename, s_sal from emp where empno= s_empno;
end;
/

-- 调用存储过程
declare
 p_ename emp.ename%type;
 p_sal emp.sal%type;
begin
 --pro_query_enameAndSal_by_empno(7369, p_ename, p_sal);
 pro_query_enameAndSal_by_empno(7369, s_sal => p_sal, s_ename =>
p_ename);
 dbms_output.put_line('员工号为7369的员工名称为：' || p_ename||', 其
工资为：' || p_sal);
end;
```

20.1.6程序中调用存储过程

```
package cn.test;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

import oracle.jdbc.OracleTypes;

public class TestProcedure {

   public static void main(String[] args) {
      Connection conn = null;
      CallableStatement call = null;
      try {
         Class.forName("oracle.jdbc.OracleDriver");
         String url = "jdbc:oracle:thin:@localhost:1521:orcl";
         conn = DriverManager.getConnection(url, "test",
"test");
         call = conn.prepareCall("{call
pro_query_enameAndSal_by_empno(?,?,?)}");
         //设置输入型参数
         call.setInt(1, 7369);
         //注册输出型参数
         call.registerOutParameter(2, OracleTypes.VARCHAR);
         call.registerOutParameter(3, OracleTypes.NUMBER);
         //调用存储过程
         call.execute();
         //获取返回值
         String ename = call.getString(2);//员工名称
         double sal = call.getDouble(3);//员工工资
         System.out.println("员工号为7369的员工名称为: " + ename
+ ", 工资为: " + sal);
      } catch (Exception e) {
         e.printStackTrace();
      } finally {
         try {
            if(call != null){
```

```
                call.close();
            }
            if(conn != null){
                conn.close();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

}
```

20.1.7 删除存储过程

【语法】

```
DROP PROCEDURE <过程名>;
```

【示例】

```
drop procedure pro_1to10_sum;
```

20.2 存储函数

存储函数与过程不同的是，存储函数有 return 语句；一般情况下如果在需要一个返回值时可使用存储函数。

20.2.1 语法

```
CREATE [OR REPLACE] FUNCTION <函数名>[(参数列表)] RETURN 数据类型
IS|AS
```

```
[局部变量声明]
BEGIN
    可执行语句
 [EXCEPTION
    异常处理语句]
RETURN 返回值;
END [<函数名>];
```

变量的类型:in 为默认类型,表示输入; out 表示只输出;in out 表示即输入又输出;

【使用方式】

直接在 select 中使用和其它系统函数使用方式一样;

在 PL/SQL 块中调用使用;

20.2.2无参存储函数

```
/*
使用无参存储函数；注意创建时函数名称不能使用()
但是在调用时候可加可不加()
*/
create or replace function fun_helloWorld
return varchar2
 as
begin
 return 'Hello World';
end;
/

-- 方式1：调用存储函数
select fun_helloWorld() from dual;
```

```
-- 方式2：调用存储函数
declare
str varchar2(20);
begin
 str :=fun_helloWorld;
 dbms_output.put_line(str);
end;
```

20.2.3有输入参数存储函数

```
/*
使用存储函数：根据员工号，查询并返回该员工的年薪
*/
create or replace function fun_get_annualSal_by_empno(p_empno
emp.empno%type)
return number
as
p_sal emp.sal%type;
p_comm emp.comm%type;
begin
  select sal,comm into p_sal, p_comm from emp where empno=p_empno;
  return 12*p_sal + nvl(p_comm,0);
end;
/

-- 调用存储函数
select fun_get_annualSal_by_empno(7369) from dual;
```

20.2.4有输入输出参数存储函数

```
/*
使用具有输入输出参数的存储函数：根据员工号，查询并返回该员工的年薪，
```

```
姓名，奖金
*/
create or replace function fun_get_annualSal_by_empno2(
p_empno emp.empno%type,
p_ename out emp.ename%type,
p_comm out emp.comm%type
)
return number
as
p_sal emp.sal%type;
begin
  select ename,sal,nvl(comm,0) into p_ename,p_sal, p_comm from emp
where empno=p_empno;
  return 12*p_sal + p_comm;
end;
/

-- 调用存储函数
declare
p_annualSal number(10,2);
p_ename emp.ename%type;
p_comm emp.comm%type;
begin
  p_annualSal := fun_get_annualSal_by_empno2(7499,p_ename,p_comm);
  dbms_output.put_line('员工姓名为：'||p_ename||',奖金为：
'||p_comm||', 年薪为：'||p_annualSal);
end;
```

20.2.5程序中调用存储函数

```
package cn.test;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

import oracle.jdbc.OracleTypes;
```

```
public class TestFunction {

    public static void main(String[] args) {
        Connection conn = null;
        CallableStatement call = null;
        try {
            Class.forName("oracle.jdbc.OracleDriver");
            String url = "jdbc:oracle:thin:@localhost:1521:orcl";
            conn = DriverManager.getConnection(url, "test",
"test");
            call = conn.prepareCall("{? = call
fun_get_annualSal_by_empno2(?,?,?)}");
            //注册存储函数返回值
            call.registerOutParameter(1, OracleTypes.DOUBLE);
            //设置输入参数，员工号
            call.setInt(2, 7499);
            //注册输出参数，员工姓名
            call.registerOutParameter(3, OracleTypes.VARCHAR);
            //注册输出参数，奖金
            call.registerOutParameter(4, OracleTypes.DOUBLE);
            call.execute();
            System.out.println("员工姓名为：" + call.getString(3)
+ "，奖金为：" + call.getDouble(4)
                    + "，年薪为：" + call.getDouble(1));
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                if(call != null){
                    call.close();
                }
                if(conn != null){
                    conn.close();
                }
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

}
```

20.2.6删除存储函数

【语法】

```
DROP FUNCTION <函数名>;
```

【示例】

```
drop function fun_helloWorld;
drop function fun_get_annualSal_by_empno;
drop function fun_get_annualSal_by_empno2;
```

20.3 存储过程与存储函数的区别

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

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

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

21 程序包

21.1 简介

包(Package)是一组相关过程、函数、变量、常量、类型和游标等 PL/SQL 程序设计元素的组合。包具有面向对象设计的特点，是对这些 PL/SQL 程序设计元素的封装。包的包括两部分：定义一个包（包头）、实现一个包（包体）；只有当定义包后才能实现包体.其中包体中的函数名与过程名须和包头中定义的函数、

过程一样。

1、包和包体必须有相同的名字；

2、包的开始没有 begin 语句，与存储过程和函数不同；

3、在包头部分定义函数和过程的名称和参数，具体实现在包体中定义；

4、在包体内声明常量、变量、类型定义、异常、及游标时不使用 declare；

5、包体内的过程和函数的定义不要 create or replace 语句；

6、包定义和包体两者分离。

21.2 创建包

21.2.1定义包（包头）

```
CREATE [OR REPLACE] PACKAGE <包名> AS|IS
                --公共类型和对象声明
                --子程序说明
END;
```

【注意】包的定义中不需要 begin 关键字；函数和过程的定义也不需要 create or replace。另外，包中定义的变量、常量在包体中可直接使用。

【示例语法】

```
create or replace package <Package_name> as

  -- 定义自定义类型
  type <TypeName> is <Datatype>;

  -- 公共常量定义
  <ConstantName> constant <Datatype> := <Value>;--声明常量

  -- 公共变量定义
  <VariableName> <Datatype>;   --数据类型

  -- 公共函数或存储过程定义
  function <FunctionName>(<Parameter> <Datatype>) return <Datatype>;
--函数
```

```
  procedure <ProcedurenName>(<Parameter> <Datatype>); --存储过程

end [Package_name];
```

21.2.2实现包（包体）

```
CREATE [OR REPLACE] PACKAGE BODY <包名> AS
                --公共类型和对象声明
                --子程序主体
      [BEGIN]
            -初始化语句
END;
```

【示例语法】

```
create or replace package body <Package_name> as

  -- 私有自定义类型(包内可用)
  type <TypeName> is <Datatype>;

  -- 私有常量(包内可用)
  <ConstantName> constant <Datatype> := <Value>

  -- 私有变量(包内可用)
  <VariableName> <Datatype>;

  -- 函数或存储过程的实现
  function <FunctionName>(<Parameter> <Datatype>) return <Datatype>
as --函数实现
    <LocalVariable> <Datatype>;
  begin
   <Statement>;
    return(<Result>);
  end;
```

```
  procedure <ProcedureName>(<Parameter> <Datatype>) as --存储过程实现
    <LocalVariable> <Datatype>;
  begin
    <Statement>;
  end;

  begin
  --初始化包体
  <Statement>;

end [Package_name];
```

21.2.3应用

```
/*
创建一个包含有变量、存储过程和函数的包；其中
存储过程可根据员工号查询并输出员工的姓名和工资
函数中利用定义的变量，然后则根据员工号查询出该员工奖金并返回
*/
--定义包
create or replace package pack_1 as

--定义存储过程
procedure pro_1(p_empno emp.empno%type);

--定义函数
function fun_1(p_empno emp.empno%type) return number;

end;

-- 【切记】需要先定义并编译（执行）包，之后再实现并编译（执行）包体
--实现包（包体）
create or replace package body pack_1 as
--定义包变量
p_comm number(7,2);

procedure pro_1(p_empno emp.empno%type) as
  p_ename emp.ename%type;
  p_sal emp.sal%type;
```

```
  begin
    select ename,sal into p_ename,p_sal from emp where empno=p_empno;
    dbms_output.put_line('员工号为：'||p_empno||'，对应的员工姓名为：
'|| p_ename||',工资为：'||p_sal);
  end;

function fun_1(p_empno emp.empno%type) return number as
  f_comm emp.comm%type;
  begin
    select nvl(comm,0) into p_comm from emp where empno=p_empno;
    return p_comm;
  end;

end;

--打开输出
set serveroutput on;

begin
--调用包中的存储过程
pack_1.pro_1(7499);
--调用包中的函数
dbms_output.put_line('7499对应的奖金为：'||pack_1.fun_1(7499));
end;
```

21.2.4程序中调用包

1、创建包

```
/*
利用包编写一个带有游标类型输出参数的存储过程
根据部门编号查询该部门下的员工
*/
create or replace package pack_2 as
--自定义一个游标类型
type empCursor is ref cursor;

procedure pro_1(p_deptno in emp.deptno%type, p_empCursor out
empCursor);
end;
```

```
create or replace package body pack_2 as

procedure pro_1(p_deptno in emp.deptno%type, p_empCursor out
empCursor) as
  begin
    open p_empCursor for select * from emp where deptno=p_deptno;
  end;

end;
```

2、程序中调用上述包

```
package cn.test;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;

import oracle.jdbc.OracleCallableStatement;
import oracle.jdbc.OracleTypes;

public class TestPackage {

   public static void main(String[] args) {
      Connection conn = null;
      CallableStatement call = null;
      try {
         Class.forName("oracle.jdbc.OracleDriver");
         String url = "jdbc:oracle:thin:@localhost:1521:orcl";
         conn = DriverManager.getConnection(url, "test",
"test");
         call = conn.prepareCall("{call pack_2.pro_1(?,?)}");
         //设置输入型参数
         call.setInt(1, 10);
         //注册输出型参数(类型为游标)
```

```
            call.registerOutParameter(2, OracleTypes.CURSOR);
            call.execute();
            ResultSet rs =
((OracleCallableStatement)call).getCursor(2);
            if(rs != null){
                while(rs.next()){
                    System.out.println("员工号为: " + rs.getInt(1) +
", 姓名为: " + rs.getString(2));
                }
                rs.close();
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                if(call != null){
                    call.close();
                }
                if(conn != null){
                    conn.close();
                }
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

}
```

21.3 删除包

【语法】

```
DROP PACKAGE <包名>;
```

【示例】

```
drop package pack_1;
```

22 触发器

22.1 语法

【语法】

```
CREATE [OR REPLACE] TRIGGER <触发器名>
BEFORE|AFTER
INSERT|DELETE|UPDATE [OF <列名>] ON <表名>
[FOR EACH ROW]
 <pl/sql 块>
```

【说明】

关键字"BEFORE"在操作完成前触发;"AFTER"则是在操作完成后触发;

关键字"FOR EACH ROW"指定触发器每行触发一次，若不指定则为表级触发器.

关键字"OF <列名>" 不写表示对整个表的所有列.

pl/sql 块中不能使用 commit;

【特殊变量】

:new --为一个引用最新的行值;

:old --为一个引用以前的行值;

这些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 ,delect 只有:old;

22.2 行级触发器

【示例 1】涨工资

```
/*
```

```
/*
触发器使用：给员工涨工资（涨后工资应该大于涨前）后，在后台输出更新前和更新后的工资
*/
create or replace trigger tri_emp_upd_sal
  after
  update of sal on emp
  for each row
begin
  if :old.sal < :new.sal then
    dbms_output.put_line('更新前工资为:'||:old.sal||',更新后工资为:'||:new.sal);
  else
    raise_application_error(-20002,'工资不能越涨越低！');
  end if;
end;
/

-- 更新工资值，并触发行级触发器
update emp set sal = 8888 where empno = 1002;
```

【示例 2】触发器+序列实现主键自增长

```
/*
触发器使用：给emp表的empno添加触发器，在插入记录时自动填入值
*/
-- 1、创建序列
create sequence seq_emp_empno;

-- 2、创建触发器
create or replace trigger tri_emp_ins_empno
  before
  insert on emp
  for each row
begin
  -- 给将要插入表的记录:new 中的empno设置sequence中的值
  select seq_emp_empno.nextval into :new.empno from dual;
end;
/

-- 新增员工数据，测试触发器+序列的组合使用
insert into emp(ename,sal) values('test002',2000);
commit;
```

22.3 表级触发器

```
/*
触发器使用：删除表的同时备份表数据到另一张备份表
*/
-- 1、从emp表结果中创建一张表并复制数据
create table emp2  as select * from emp;

-- 2、创建备份表emp_bak
create table emp_bak as select * from emp2 where 1=2;

-- 3、创建表触发器，当对表操作时触发
create or replace trigger tri_emp2_del
  before
  delete on emp2
begin
  -- 将emp2表中的数据备份到emp_bak
  insert into emp_bak select * from emp2;
end;
/

-- 4、测试删除emp2表的数据
delete from emp2;
select * from emp2;
select * from emp_bak;
```

22.4 开启禁用触发器

【禁用某个触发器】

ALTER TRIGGER <触发器名> DISABLE

【示例】

```
alter trigger tri_emp_upd_sal disable;
update emp set sal = 8888 where empno = 1002;
```

【重新启用触发器】

ALTER TRIGGER <触发器名> ENABLE

【示例】

```
alter trigger tri_emp_upd_sal enable;
update emp set sal = 8888 where empno = 1002;
```

【禁用表的所有触发器】

ALTER TABLE <表名> DISABLE ALL TRIGGERS;

【示例】

```
alter table emp disable all triggers;
```

【启用表的所有触发器】

ALTER TABLE <表名> ENABLE ALL TRIGGERS;

【示例】

```
alter table emp enable all triggers;
```

【删除触发器】

DROP TRIGGER <触发器名>;

【示例】

```
drop trigger tri_emp_upd_sal;
```

23 数据字典

Oracle 数据字典中，对象名称多数以"USER."，"ALL."，"DBA."。前缀"USER."视图中记录通常记录执行查询的帐户所拥有的对象的信息，"ALL."视图中记录包

括"USER"记录和授权至 PUBLIC 或用户的对象的信息，"DBA."视图包含所有数据库对象，而不管其所有者。

视图名	描述
ALL_CATALOG	All tables, views, synonyms, sequences accessible to the user
ALL_COL_COMMENTS	Comments on columns of accessible tables and views
ALL_COL_GRANTS_MADE	Grants on columns for which the user is owner or grantor
ALL_COL_GRANTS_RECD	Grants on columns for which the user or PUBLIC is the grantee
ALL_COL_PRIVS	Grants on columns for which the user is the grantor, grantee, owner, or an enabled role or PUBLIC is the grantee
ALL_COL_PRIVS_MADE	Grants on columns for which the user is owner or grantor
ALL_COL_PRIVS_RECD	Grants on columns for which the user, PUBLIC or enabled role is the grantee
ALL_CONSTRAINTS	Constraint definitions on accessible tables
ALL_CONS_COLUMNS	Information about accessible columns in constraint definitions
ALL_DB_LINKS	Database links accessible to the user
ALL_DEF_AUDIT_OPTS	Auditing options for newly created objects
ALL_DEPENDENCIES	Dependencies to and from objects accessible to the user
ALL_ERRORS	Current errors on stored objects that user is allowed to create
ALL_INDEXES	Descriptions of indexes on tables accessible to the user
ALL_IND_COLUMNS	COLUMNs comprising INDEXes on accessible TABLES
ALL_OBJECTS	Objects accessible to the user
ALL_REFRESH	All the refresh groups that the user can touch
ALL_REFRESH_CHILDREN	All the objects in refresh groups, where the user can touch the group
ALL_SEQUENCES	Description of SEQUENCEs accessible to the user
ALL_SNAPSHOTS	Snapshots the user can look at
ALL_SOURCE	Current source on stored objects that user is allowed to create
ALL_SYNONYMS	All synonyms accessible to the user
ALL_TABLES	Description of tables accessible to the user
ALL_TAB_COLUMNS	Columns of all tables, views and clusters
ALL_TAB_COMMENTS	Comments on tables and views accessible to the user
ALL_TAB_GRANTS_MADE	User's grants and grants on user's objects
ALL_TAB_GRANTS_RECD	Grants on objects for which the user or PUBLIC is the grantee
ALL_TAB_PRIVS	Grants on objects for which the user is the grantor, grantee, owner, or an enabled role or PUBLIC is the grantee
ALL_TAB_PRIVS_MADE	User's grants and grants on user's objects
ALL_TAB_PRIVS_RECD	Grants on objects for which the user, PUBLIC or enabled role is the grantee
ALL_TRIGGERS	Triggers accessible to the current user
ALL_TRIGGER_COLS	Column usage in user's triggers or in triggers on user's tables

ALL_USERS	Information about all users of the database
ALL_VIEWS	Text of views accessible to the user
USER_AUDIT_CONNECT	Audit trail entries for user logons/logoffs
USER_AUDIT_OBJECT	Audit trail records for statements concerning objects, specifically: table, cluster, view, index, sequence, [public] database link, [public] synonym, procedure, trigger, rollback segment, tablespace, role, user
USER_AUDIT_SESSION	
USER_AUDIT_STATEMENT	Audit trail records concerning grant, revoke, audit, noaudit and alter system
USER_AUDIT_TRAIL	Audit trail entries relevant to the user
USER_CATALOG	Tables, Views, Synonyms and Sequences owned by the user
USER_CLUSTERS	Descriptions of user's own clusters
USER_CLU_COLUMNS	Mapping of table columns to cluster columns
USER_COL_COMMENTS	Comments on columns of user's tables and views
USER_COL_GRANTS	Grants on columns for which the user is the owner, grantor or grantee
USER_COL_GRANTS_MADE	All grants on columns of objects owned by the user
USER_COL_GRANTS_RECD	Grants on columns for which the user is the grantee
USER_COL_PRIVS	Grants on columns for which the user is the owner, grantor or grantee
USER_COL_PRIVS_MADE	All grants on columns of objects owned by the user
USER_COL_PRIVS_RECD	Grants on columns for which the user is the grantee
USER_CONSTRAINTS	Constraint definitions on user's own tables
USER_CONS_COLUMNS	Information about accessible columns in constraint definitions
USER_CROSS_REFS	Cross references for user's views and synonyms
USER_DB_LINKS	Database links owned by the user
USER_DEPENDENCIES	Dependencies to and from a users objects
USER_ERRORS	Current errors on stored objects owned by the user
USER_EXTENTS	Extents comprising segments owned by the user
USER_FREE_SPACE	Free extents in tablespaces accessible to the user
USER_INDEXES	Description of the user's own indexes
USER_IND_COLUMNS	COLUMNs comprising user's INDEXes or on user's TABLES
USER_JOBS	All jobs owned by this user
USER_OBJECTS	Objects owned by the user
USER_OBJECT_SIZE	Sizes, in bytes, of various pl/sql objects
USER_OBJ_AUDIT_OPTS	Auditing options for user's own tables and views
USER_REFRESH	All the refresh groups
USER_REFRESH_CHILDREN	All the objects in refresh groups, where the user owns the refresh group
USER_RESOURCE_LIMITS	Display resource limit of the user
USER_ROLE_PRIVS	Roles granted to current user
USER_SEGMENTS	Storage allocated for all database segments
USER_SEQUENCES	Description of the user's own SEQUENCEs

USER_SNAPSHOTS	Snapshots the user can look at
USER_SNAPSHOT_LOGS	All snapshot logs owned by the user
USER_SOURCE	Source of stored objects accessible to the user
USER_SYNONYMS	The user's private synonyms
USER_SYS_PRIVS	System privileges granted to current user
USER_TABLES	Description of the user's own tables
USER_TABLESPACES	Description of accessible tablespaces
USER_TAB_AUDIT_OPTS	Auditing options for user's own tables and views
USER_TAB_COLUMNS	Columns of user's tables, views and clusters
USER_TAB_COMMENTS	Comments on the tables and views owned by the user
USER_TAB_PRIVS	Grants on objects for which the user is the owner, grantor or grantee
USER_TAB_PRIVS_MADE	All grants on objects owned by the user
USER_TAB_PRIVS_RECD	Grants on objects for which the user is the grantee
USER_TRIGGERS	Triggers owned by the user
USER_TRIGGER_COLS	Column usage in user's triggers
USER_TS_QUOTAS	Tablespace quotas for the user
USER_USERS	Information about the current user
USER_VIEWS	Text of views owned by the user
AUDIT_ACTIONS	Description table for audit trail action type codes. Maps action type numbers to action type names
COLUMN_PRIVILEGES	Grants on columns for which the user is the grantor, grantee, owner, or an enabled role or PUBLIC is the grantee
DICTIONARY	Description of data dictionary tables and views
DICT_COLUMNS	Description of columns in data dictionary tables and views
DUAL	
GLOBAL_NAME	global database name
INDEX_HISTOGRAM	statistics on keys with repeat count
INDEX_STATS	statistics on the b-tree
RESOURCE_COST	Cost for each resource
ROLE_ROLE_PRIVS	Roles which are granted to roles
ROLE_SYS_PRIVS	System privileges granted to roles
ROLE_TAB_PRIVS	Table privileges granted to roles
SESSION_PRIVS	Privileges which the user currently has set
SESSION_ROLES	Roles which the user currently has enabled.
TABLE_PRIVILEGES	Grants on objects for which the user is the grantor, grantee, owner, or an enabled role or PUBLIC is the grantee
ACCESSIBLE_COLUMNS	Synonym for ALL_TAB_COLUMNS
ALL_COL_GRANTS	Synonym for COLUMN_PRIVILEGES
ALL_JOBS	Synonym for USER_JOBS
ALL_TAB_GRANTS	Synonym for TABLE_PRIVILEGES
CAT	Synonym for USER_CATALOG
CLU	Synonym for USER_CLUSTERS

COLS	Synonym for USER_TAB_COLUMNS
DBA_AUDIT_CONNECT	Synonym for USER_AUDIT_CONNECT
DBA_AUDIT_RESOURCE	Synonym for USER_AUDIT_RESOURCE
DBA_REFRESH_CHILDREN	Synonym for USER_REFRESH_CHILDREN
DICT	Synonym for DICTIONARY
IND	Synonym for USER_INDEXES
OBJ	Synonym for USER_OBJECTS
SEQ	Synonym for USER_SEQUENCES
SM$VERSION	Synonym for SM_$VERSION
SYN	Synonym for USER_SYNONYMS
TABS	Synonym for USER_TABLES
V$ACCESS	Synonym for V_$ACCESS
V$ARCHIVE	Synonym for V_$ARCHIVE
V$BACKUP	Synonym for V_$BACKUP
V$BGPROCESS	Synonym for V_$BGPROCESS
V$CIRCUIT	Synonym for V_$CIRCUIT
V$COMPATIBILITY	Synonym for V_$COMPATIBILITY
V$COMPATSEG	Synonym for V_$COMPATSEG
V$CONTROLFILE	Synonym for V_$CONTROLFILE
V$DATABASE	Synonym for V_$DATABASE
V$DATAFILE	Synonym for V_$DATAFILE
V$DBFILE	Synonym for V_$DBFILE
V$DBLINK	Synonym for V_$DBLINK
V$DB_OBJECT_CACHE	Synonym for V_$DB_OBJECT_CACHE
V$DISPATCHER	Synonym for V_$DISPATCHER
V$ENABLEDPRIVS	Synonym for V_$ENABLEDPRIVS
V$FILESTAT	Synonym for V_$FILESTAT
V$FIXED_TABLE	Synonym for V_$FIXED_TABLE
V$LATCH	Synonym for V_$LATCH
V$LATCHHOLDER	Synonym for V_$LATCHHOLDER
V$LATCHNAME	Synonym for V_$LATCHNAME
V$LIBRARYCACHE	Synonym for V_$LIBRARYCACHE
V$LICENSE	Synonym for V_$LICENSE
V$LOADCSTAT	Synonym for V_$LOADCSTAT
V$LOADTSTAT	Synonym for V_$LOADTSTAT
V$LOCK	Synonym for V_$LOCK
V$LOG	Synonym for V_$LOG
V$LOGFILE	Synonym for V_$LOGFILE
V$LOGHIST	Synonym for V_$LOGHIST
V$LOG_HISTORY	Synonym for V_$LOG_HISTORY
V$MLS_PARAMETERS	Synonym for V_$MLS_PARAMETERS
V$MTS	Synonym for V_$MTS
V$NLS_PARAMETERS	Synonym for V_$NLS_PARAMETERS

V$NLS_VALID_VALUES	Synonym for V_$NLS_VALID_VALUES
V$OPEN_CURSOR	Synonym for V_$OPEN_CURSOR
V$OPTION	Synonym for V_$OPTION
V$PARAMETER	Synonym for V_$PARAMETER
V$PQ_SESSTAT	Synonym for V_$PQ_SESSTAT
V$PQ_SLAVE	Synonym for V_$PQ_SLAVE
V$PQ_SYSSTAT	Synonym for V_$PQ_SYSSTAT
V$PROCESS	Synonym for V_$PROCESS
V$QUEUE	Synonym for V_$QUEUE
V$RECOVERY_LOG	Synonym for V_$RECOVERY_LOG
V$RECOVER_FILE	Synonym for V_$RECOVER_FILE
V$REQDIST	Synonym for V_$REQDIST
V$RESOURCE	Synonym for V_$RESOURCE
V$ROLLNAME	Synonym for V_$ROLLNAME
V$ROLLSTAT	Synonym for V_$ROLLSTAT
V$ROWCACHE	Synonym for V_$ROWCACHE
V$SESSION	Synonym for V_$SESSION
V$SESSION_CURSOR_CACHE	Synonym for V_$SESSION_CURSOR_CACHE
V$SESSION_EVENT	Synonym for V_$SESSION_EVENT
V$SESSION_WAIT	Synonym for V_$SESSION_WAIT
V$SESSTAT	Synonym for V_$SESSTAT
V$SESS_IO	Synonym for V_$SESS_IO
V$SGA	Synonym for V_$SGA
V$SGASTAT	Synonym for V_$SGASTAT
V$SHARED_SERVER	Synonym for V_$SHARED_SERVER
V$SQLAREA	Synonym for V_$SQLAREA
V$STATNAME	Synonym for V_$STATNAME
V$SYSSTAT	Synonym for V_$SYSSTAT
V$SYSTEM_CURSOR_CACHE	Synonym for V_$SYSTEM_CURSOR_CACHE
V$SYSTEM_EVENT	Synonym for V_$SYSTEM_EVENT
V$THREAD	Synonym for V_$THREAD
V$TIMER	Synonym for V_$TIMER
V$TRANSACTION	Synonym for V_$TRANSACTION
V$TYPE_SIZE	Synonym for V_$TYPE_SIZE
V$VERSION	Synonym for V_$VERSION
V$WAITSTAT	Synonym for V_$WAITSTAT
V$_LOCK	Synonym for V_$_LOCK

24 角色

Oracle 提供了三种标准的角色（role）：CONNECT、RESOURCE 和 DBA。

1. CONNECT Role(连接角色)

临时用户，特别是那些不需要建表的用户，通常只赋予他们 CONNECT role。CONNECT 是使用 Oracle 的简单权限，这种权限只有在对其他用户的表有访问权时，包括 select、insert、update 和 delete 等，才会变得有意义。

2. RESOURCE Role(资源角色)

更可靠和正式的数据库用户可以授予 RESOURCE role。RESOURCE 提供给用户另外的权限以创建他们自己的表、序列、过程、触发器、索引和簇。

3. DBA Role(数据库管理员角色)

DBA role 拥有所有的系统权限--包括无限制的空间限额和给其他用户授予各种权限的能力。

除此以上角色外；还可以自行创建角色。用户创建的 role 可以由表或系统权限或两者的组合构成。为了创建 role，用户必须具有 CREATE ROLE 系统权限。

24.1 创建角色

创建角色后，可以对角色授予权限；授权的语法和前面授权给用户的语法相同。

【语法】

```
CREATE ROLE <role_name>;
```

【示例】

```
-- system 用户登录，授予test 创建角色的权限
grant create role to test;

-- 创建角色
create role role_test;

-- 授予emp的select 操作权限给role_test角色
grant select on emp to role_test;

-- 给scott用户授予role_test的角色
```

```
grant role_test to scott;
```

24.2 删除角色

【语法】

```
DROP ROLE <role_name>;
```

【示例】

```
drop role role_test;
```

25 闪回

25.1 闪回简介

在 Oracle 的操作工程中，会不可避免地出现操作失误或者用户失误，例如不小心删除了一个表等，这些失误和错误可能会造成重要数据的丢失，最终导致 Oracle 数据库停止。

在传统操作上，当发生数据丢失、数据错误问题时，解决的主要办法是数据的导入导出、备份恢复技术，这些方法都需要在发生错误前，有一个正确的备份才能进行恢复。为了减少这方面的损失，Oracle 提供了闪回技术。有了闪回技术，就可以实现数据的快速恢复，而且不需要数据备份。

闪回特点：

传统的恢复技术缓慢:它是整个数据库或者一个文件恢复，不只恢复损坏的数据在数据库日志中每个修改都必须被检查；

闪回速度快:通过行和事务把改变编入索引，仅仅改变了的数据会被恢复；

闪回命令容易，没有复杂步骤。

25.2 闪回类型

主要有三种闪回：闪回表(flashback table)、闪回删除(flashback drop)、闪回数据库（flashback database)；一般情况下对数据库的闪回需要配置闪回数据库，然后自动产生闪回日志；再根据闪回日志恢复数据库。

25.3 闪回查询

根据闪回日志可以快速查询在某个时间点的数据。

```
--查看 10 秒之前的 emp 表
select * from emp as of timestamp sysdate - interval '10' second;
select * from emp as of scn timestamp_to_scn(sysdate - interval '10'
second);
【说明】
as of timestamp 是固定写法,查询某个时间点对应的数据
as of scn 查询某 scn 对应的数据
sysdate - interval '10' second 是时间值的计算

--通过查询某个时间的数据来更新现有数据
--将 7499 员工的姓名更新为 5 分钟之前的姓名
update emp e set ename =
(select ename from emp
as of timestamp systimestamp - interval '5' minute where empno=e.empno)
where empno=7499;
```

25.4 闪回表

闪回表(flashback table)实际上是将表中的数据快速恢复到过去的一个焦点或者系统改变号 SCN 上；对进行表闪回的表必须 row movement 为 enable。

SCN: System Change Number.

实现表的闪回，需要使用到与撤销表空间相关的 undo 信息，通过 show parameter undo 命令可以了解这些信息。

```
conn sys/orcl as sysdba
show parameters undo;     // undo 表空间
alter system set undo_retention=1200 scope=both;
```

undo_retention：数据保留时间长度（默认是 900 秒）

scope 参数的值：

momory-当前 session 中有效

spfile: 修改配置文件，但当前会话中无效

both:当前会话有效，同时修改配置文件

undo 表空间:保存了所有的操作记录(2G 的空间) 因为有了该表空间才可以进行闪回

【语法】

```
flashback table [schema.]table_name[,...n] to {[scn] | [timestamp]
[[enable | disable] triggers]};
```

【说明】

scn：表示通过系统改变号进行闪回；scn 系统改变号一般和系统时间相对应；查看当前系统时间和所对应系统 scn：

```
select to_char(sysdate,'yyyy-mm-dd hh24:mi:ss'),
timestamp_to_scn(sysdate) from dual;
```

timestamp：表示通过时间戳的形式来进行闪回；

enable|disable triggers：表示触发器恢复之后的状态，默认为 disable。

rowid 这个伪列是 Oracle 默认提供给每一个表的，主要用于记录每一行数据存储的磁盘物理地址。当删除一行记录后，后面的记录依次跟进上来，当需要恢复某一个中间的行时，就需要行具备行移动功能（alter table <表名>

```
enable row movement;)
```

【示例】

```
-- 授权用户闪回表的权限
grant flashback any table to test;

-- 查看当前时间点或scn号
select to_char(sysdate,'yyyy-mm-dd hh24:mi:ss'),
timestamp_to_scn(sysdate) from dual;

-- 删除数据
delete from emp where empno = 7449;
commit;

--允许行移动
alter table emp enable row movement;

-- 方式一；使用时间点闪回表
flashback table emp to timestamp to_timestamp('时间格式字符串
','yyyy-mm-dd HH24:mi:ss');

-- 方式二；使用SCN闪回表
flashback table emp to scn SCN号;
```

25.5 闪回删除

闪回删除(flashback drop)。当整个表被删除并在回收站查询到的话；可以对表进行闪回。show recyclebin：可以显示当前用户 recyclebin 中的表。

系统参数 recyclebin 控制表删除后是否到回收站，show parameter recyclebin 可以查看该参数的状态。

对于系统参数的修改有两种，全局的修改和会话的修改：

（1）alter system set param_name=param_value;

（2）alter session set param_name=param_value;

show recyclebin; --查看回收站

purge recyclebin; --清空回收站

【语法】

```
flashback table table_name to before drop [rename to new_name];
```

【说明】

rename to new_name：如果在删除原表之后又重新创建了一个一样名称的表，那么恢复回收站的表时可以对表名进行重命名

【示例】

```
-- 删除表
drop table emp;

-- 恢复表
flashback table emp to before drop;
```

26 数据备份与恢复

26.1 数据备份

```
--全表备份
exp test/test@orcl file=d:\database\oracle_data\test.dmp full=y;

--指定表备份
exp test/test@orcl file=d:\database\oracle_data\test_emp_dept.dmp
tables=(emp,dept);
```

【说明】full：完整导出数据库，一般使用 system 具有管理员权限的用户在**命令行**下进行操作。

26.2 数据恢复

```
--全表恢复
imp test/test@orcl ignore=y file=d:\database\oracle_data\test.dmp
full=y;

--指定表恢复
imp test/test@orcl ignore=y
file=d:\database\oracle_data\test_emp_dept.dmp tables=(emp,dept);
```

【说明】ignore：忽略创建错误

27 性能优化

1、 查两张以上表时，把记录少的放在右边

2、 WHERE 子句中的连接顺序

ORACLE 采用自上而下的顺序解析 WHERE 子句,根据这个原则,那些可以过滤掉最大数量记录的条件应写在 WHERE 子句最后。

例如：查询员工的编号，姓名，工资，部门名

如果 emp.sal>1500 能过滤掉半数记录的话，

```
select emp.empno,emp.ename,emp.sal,dept.dname
from emp,dept
where (emp.deptno = dept.deptno) and (emp.sal > 1500)
      .......
```

3、 SELECT 子句中避免使用*号

ORACLE 在解析的过程中,会将*依次转换成所有的列名，这个工作是通过查询数据字典完成的，这意味着将耗费更多的时间

4、 避免对大表进行无条件或无索引的的扫描

5、 清空表时用 TRUNCATE 替代 DELETE

6、 尽量多使用 COMMIT；因为 COMMIT 会释放回滚点

7、 用索引提高查询效率，善用索引

避免在索引列上使用 NOT；因为 Oracle 服务器遇到 NOT 后，他就会停止目前的工作，转而执行全表扫描。

避免在索引列上使用计算；WHERE 子句中，如果索引列是函数的一部分，优化器将不使用索引而使用全表扫描，这样会变得慢

例如，SAL 列上有索引，

低效：

```
SELECT EMPNO,ENAME
FROM EMP
WHERE SAL*12 > 24000;
```

高效：

```
SELECT EMPNO,ENAME
FROM EMP
WHERE SAL > 24000/12;
```

8、**字符串型，能用=号，不用 like**；=号表示精确比较，like 表示模糊比较

9、 用 >= 替代 >

低效：

```
SELECT * FROM EMP WHERE DEPTNO > 3
```

首先定位到 DEPTNO=3 的记录并且扫描到第一个 DEPT 大于 3 的记录

高效：

```
SELECT * FROM EMP WHERE DEPTNO >= 4
```

直接跳到第一个 DEPT 等于 4 的记录

10、 用 IN 替代 OR

```
select * from emp where sal = 1500 or sal = 3000 or sal = 800;
select * from emp where sal in (1500,3000,800);
```

11、 用 exists 代替 in；not exists 代替 not in

not in 字句将执行一个内部的排序和合并，任何情况下，not in 是最低效的，子查询中全表扫描；表连接比 exists 更高效

12、 用 UNION ALL 替换 UNION

当 SQL 语句需要 UNION 两个查询结果集合时,这两个结果集合会以 UNION-ALL 的方式被合并, 然后在输出最终结果前进行排序. 如果用 UNION ALL 替代 UNION, 这样排序就不是必要了. 效率会因此得到提高。

13、 避免使用耗费资源的操作

带有 DISTINCT,UNION,MINUS,INTERSECT 的 SQL 语句会启动 SQL 引擎 执行耗费资源的排序(SORT)功能. DISTINCT 需要一次排序操作, 而其他的至少需要执行两次排序. 通常, 带有 UNION, MINUS , INTERSECT 的 SQL 语句都可以用其他方式重写。

最后；同样的操作有些时候可以在程序上处理的就程序上处理，毕竟在内存中的执行速度比在硬盘上执行要高非常多。