# Oracle概述

## 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...

## 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。

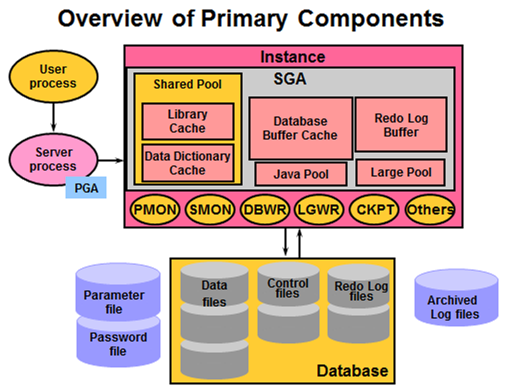
# Oracle安装与组成

## 安装oracle 11g数据库

详见《oracle 11g 32位安装.docx》

## 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。

## 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)

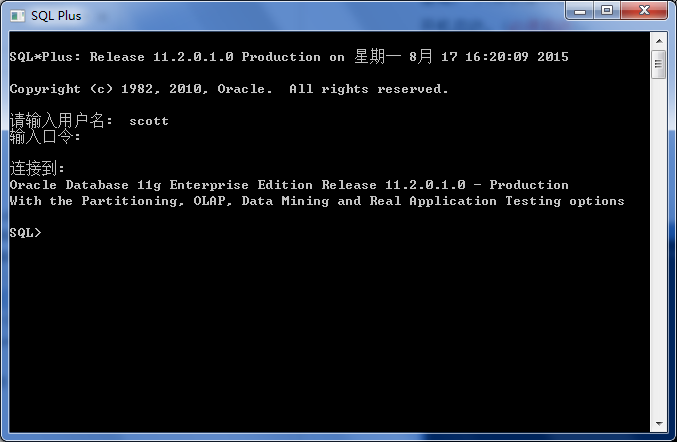
**OracleOraDb11g\_home1TNSListener** -- 监听器服务，服务只有在数据库需要远程访问或使用SQL Developer等工具的时候才需要，此服务被默认的设置为开机启动(非必须启动)

**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)

# 连接Oracle

## 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

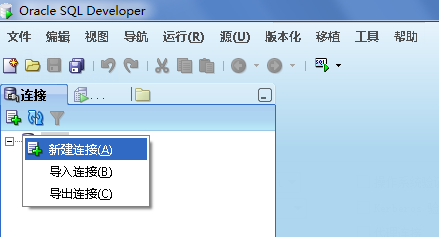
输入语句查询该用户下的对象：

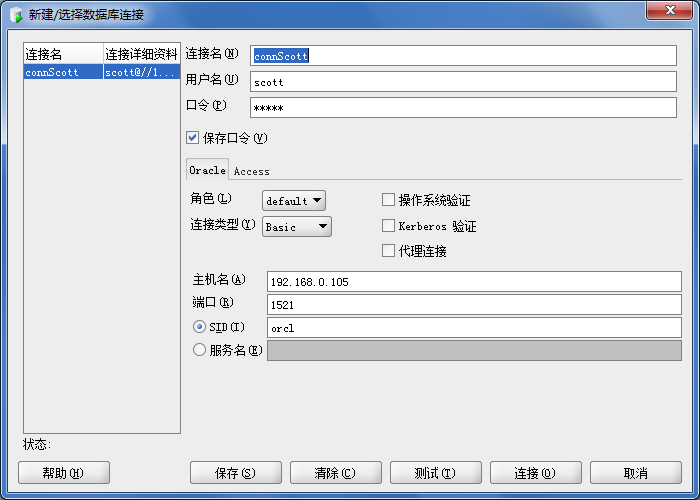


另外；也可以直接在命令行中输入sqlplus scott/tiger 进入并登录

## SQLDeveloper 连接

打开SQL Developer；在出现界面的左边右击鼠标，新建连接：

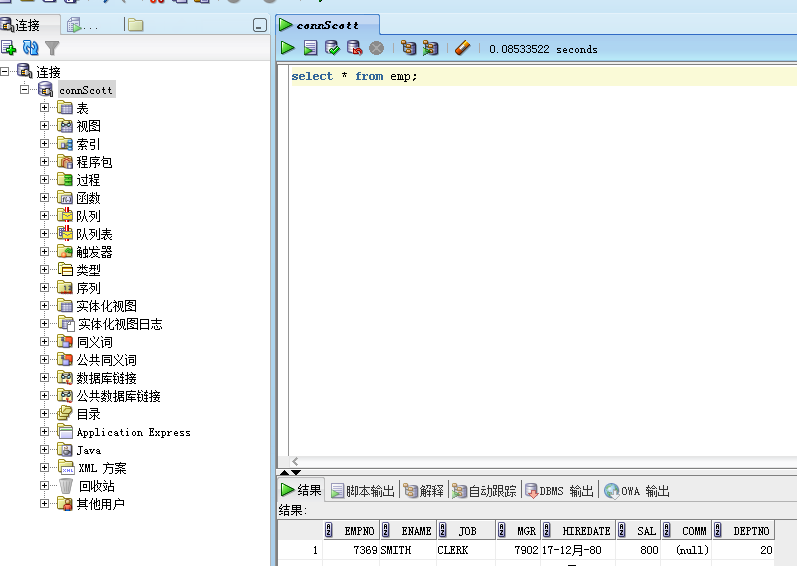




注意在上图中；

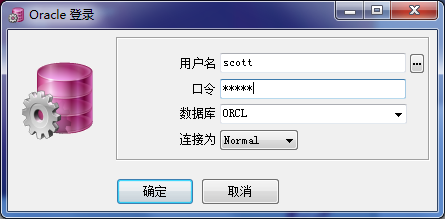
主机名：如果是本机的按照配置在网络管理中的服务的配置设置，可以为localhost；如果是连接其它机器的数据库则指定其ip；

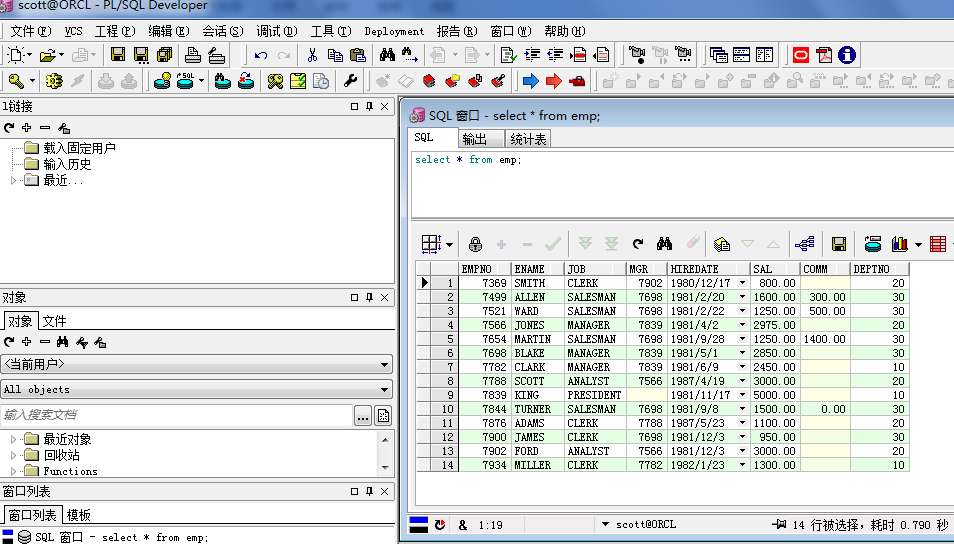
SID：是指定数据库服务器上的全局数据库名称，默认安装的话一般是orcl



## PLSQLDeveloper 连接

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





## 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；

1. 新建java项目测试连接；



# SQLPlus 设置与常用命令

## 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for9999  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 for9999  *-- 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 |

## 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  spool off | 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:\itcast\itcast.txt  spool d:\itcast\test.sql append  spool off |
| clear screen或者 host cls | 清屏 |
| exit | 退出SQL Plus |

# 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。

## 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

## 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】  CREATETA BLESPACE itcast\_ts  DATAFILE'd:\oracle\_data\itcast01.dbf'SIZE10M  AUTO EXTENDON; |

**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='ITCAST\_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】  ALTERTABLESPACE itcast\_ts  ADDDATAFILE'd:\oracle\_data\itcast02.DBF'SIZE5M  AUTOEXTENDON; |

**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】  DROPTABLESPACE itcast\_ts;  DROPTABLESPACE itcast\_ts INCLUDINGCONTENTSANDDATAFILES; |

# 数据库用户

## 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 |

## 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。

### 查询系统用户

|  |
| --- |
| **select \* from all\_users;**  **或**  **select \* from dba\_users;**--更详细的用户信息 |

### 解锁用户

|  |
| --- |
| 【语法】  ALTER USER用户名ACCOUNT UNLOCK;  【示例】解锁hr用户  alteruser hr accountunlock; |

### 创建用户

|  |
| --- |
| 【语法】  CREATE USER 用户名 IDENTIFIED BY 密码  DEFAULT TABLESPACE 表空间;  【示例】  CREATE USER itcast IDENTIFIED BY itcast  DEFAULT TABLE SPACE itcast\_ts  TEMPORARY TABLESPACE temp; |

### 修改用户密码

|  |
| --- |
| 【语法】  ALTER USER 用户名 identified by 密码  【示例】  ALTER USER itcast identifiedby it; |

### 删除用户

|  |
| --- |
| 【语法】  DROP USER 用户名 CASCADE;  【示例】  DROPUSER itcast CASCADE; |

# DCL数据控制语言

## 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*  GRANTconnect,resourceTO itcast;  【备注】使用如下语句可以查看resource角色下的权限  SELECT\*FROMDBA\_SYS\_PRIVS WHEREGRANTEE='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 itcast;  *--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; |

## 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 itcast;  【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 itcast; |

# DDL数据定义语言

## 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】  createtable t\_student(  s\_id number(8)PRIMARYKEY,  s\_name varchar2(20)notnull,  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】  createtable emp asselect\*fromscott.emp;  createtable emp asselect empno,ename fromscott.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:*  createtable emp asselect\*fromscott.emp where1=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】  altertable 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】  altertable 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】  altertable 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】  altertable t\_student renamecolumn s\_id to s\_no; |

## 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】  droptable t\_student; |

## 回收站

### 查看回收站

|  |
| --- |
| *--查看回收站*  showrecyclebin;或select\*fromrecyclebin; |

### 清空回收站

|  |
| --- |
| *--清空回收站*  purgerecyclebin; |

## 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. |

# DML数据操作语言

## 新增

|  |
| --- |
| 【语法1】  INSERT INTO table\_name (column1,column2,...)  VALUES ( value1,value2, ...);  【示例1】  Insert into emp (empno,ename)values(1111,'itcast');  【语法2】  INSERT INTO <table\_name><SELECT 语句>;  【示例2】  createtable t1 asselect\*from emp where1=2;  insertinto t1 select\*from emp where sal>2000; |

## 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=3000where ename='itcast'; |

## 查询

### 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; |

1. 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; |

1. 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; |

1. 可以用做计算器

|  |
| --- |
| select2\*8from dual; |

### 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】  selectrowid, emp.\*from emp; |

### 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】  selectrownum, 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(selectrownum r,emp.\*from emp whererownum<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\*/  selectrownum,t.\*from(select empno,ename from emp orderby empno desc) t; |

### 连接查询

准备查询数据，将scott用户下的dept表复制到itcast用户下。

|  |
| --- |
| 使用sys用户登录系统；替itcast用户创建dept表，表结构和数据来自scott.dept。  *--执行语句如下*  createtable itcast.dept asselect\*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; |

1. **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(+)groupby 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 groupby d.deptno,d.dname; |

1. **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 |

### 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 groupby deptno; |

1. **部门平均工资**

|  |
| --- |
| *--查询部门的平均工资*  select deptno,avg(sal)from emp groupby 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** |

1. **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 empwhere 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 <(selectmin(sal)from emp where deptno=10); |

1. **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 '\'; |

## 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; |

# TCL事务控制语言

## 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;

## 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*  insertinto 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*  rollbackto 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*  rollbackto a; |

# 运算符

## 算术运算符

+、-、\*、/

## 比较（关系）运算符

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

、is null

## 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

## 连接运算符

||

|  |
| --- |
| 【示例】  select'工号为：'|| empno ||' 的员工的姓名为：'|| ename from emp; |

## 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 between1000and2000  intersect  select ename,sal from emp where sal between1500and2500;  *--minus(减集)*  select ename,sal from emp where sal between1000and2000  minus  select ename,sal from emp where sal between1500and2500; |

## 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。

# 常用函数

## 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】  Selectround(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】  selecttrunc(5555.66666,2.1),  trunc(5555.66666,-2.6),trunc(5555.033333)from dual;  返回：5555.66 5500 5555 |

## 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】  selectlength('励锋科技'),length('itcast励锋科技')from dual;  LENGTH('励锋科技') LENGTH('ITCAST励锋科技')  ------------------ ------------------------  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】  selectlpad('itcast',10,'\*'),rpad('itcast',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】  selectreplace('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; |

## 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】  selectsysdate,  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; |

## 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')+1from dual;  *--等同上述结果*  Select '201212'+3 from dual; |

## 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

## 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(partitionby deptno orderby job)as myDenseRank from emp e;  *--ROW\_NUMBER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(partitionby deptno orderby job)as myRowNumber from emp e; |

## 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; |

# 视图

## 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> ; |

## 视图操作

|  |
| --- |
| *-- 授予itcast用户创建视图的权限*  Grant create view to itcast;  *-- 登录itcast，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,'itcast3');  *--通过视图修改数据*  update v\_emp set ename='励锋科技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; |

# 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。

## 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itcast创建同义词的权限*  Grant create synonym to itcast;  *--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='itcast5'where empno='1234';  *--删除同义词*  dropsynonym syn\_emp; |

## 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itcast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 itcast;  *--revoke create public synonym,drop public synonym from itcast;*  *--登陆itcast用户创建公有同义词 conn itcast/itcast;*  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itcast2并授权*  *--create user itcast2 identified by itcast2 default tablespace itcast\_ts;*  *--grant connect,resource to itcast2;*  *--为其它用户itcast2授权使用公有同义词（需要给予使用表的权限）*  Grant select,update on itcast.emp to itcast2;  *--revoke select,update on itcast.emp from itcast2;*  *--登陆itcast2用户下使用公有同义词syn\_public\_emp*  select\*from syn\_public\_emp;  update syn\_public\_emp set ename='励锋科技5'where empno=5555;  *--删除同义词*  *--登陆itcast，删除公有同义词*  Drop public synonym syn\_public\_emp; |

# 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

## 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); |

## 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; |

# 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。

## 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*  createsequence seq\_test2  incrementby-1  startwith5  maxvalue5  minvalue1  nocycle; |

## 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 with1000  Increment by1  maxvalue9000  minvalue1000  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,'itcast1');  insert into emp(empno,ename)  values(seq\_emp\_empno.nextval,'itcast2');  *-- 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; |

## 删除序列

|  |
| --- |
| 【语法】  DROP SEQUENCE <sequence\_name>  【示例】  Drop sequence seq\_test; |

## 序列与sys\_guid

sys\_guid和序列都可以作为主键值。

|  |
| --- |
| *--使用SYS\_GUID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; |

# 分区表

## 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。

## 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。

### 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)primarykey,  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*  insertinto 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*  deletefrom myemp partition(p1);  *-- 查看数据*  select\*from myemp; |

### 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*  createtable myemp2  ( empno number(4)primarykey,  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*  insertinto 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; |

# 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; |

## 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 |

## 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。

### 基本类型

**声明**

|  |
| --- |
| 【变量声明】  <变量名> 类型[:=初始值];  【示例】  name varchar2(20) := 'itcast';  【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） |

### %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 constantnumber(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; |

### %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 constantnumber(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; |

## 控制语句

### 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 >=5000THEN  dbms\_output.put\_line('员工号为：'|| p\_empno ||'的员工的工资级别为：A级');  ELSIF p\_sal >=4000THEN  dbms\_output.put\_line('员工号为：'|| p\_empno ||'的员工的工资级别为：B级');  ELSIF p\_sal >=3000THEN  dbms\_output.put\_line('员工号为：'|| p\_empno ||'的员工的工资级别为：C级');  ELSIF p\_sal >=2000THEN  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; |

### 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;  EXITWHEN 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;  ENDLOOP;  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;  ENDLOOP;  dbms\_output.put\_line('1-10的总和为：'|| p\_sum);  END; |

### 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 =4then  null;*-- 占位，不能去掉*  else  dbms\_output.put\_line(flag);  endif;  ENDLOOP;  END; |

## 异常处理

### 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);

### 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; |

### 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;

1. **抛出异常**；在出现异常部分抛出异常，

格式：RAISE　<异常名称>；

1. **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;  --nvl(comm,0)如果comm为null就填充0  if p\_comm >=500then  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; |

# 游标

## 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。

### 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"; |

### 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; |

1. 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)isselect 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*  exitwhen 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);  endloop;  *--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)isselect 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;  endloop;  close cur\_dept\_emps;  end;  --**参考**：使用for循环实现  declare  cursor cur\_dept\_emps(dno emp.deptno%type)isselect 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%foundthen  dbms\_output.put\_line('员工姓名为：'||emp\_info.ename||'，工作为：'||emp\_info.job||'，工资为：'||emp\_info.sal);  endif;  endloop;  end; |

1. 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 isselect 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*  exitwhen cur\_emp%notfound;  *--如果存在数据，则处理*  if'PRESIDENT'= p\_job then  update emp set sal = sal +800where empno = p\_empno;  elsif'MANAGER'= p\_job then  update emp set sal = sal +600where empno = p\_empno;  else  update emp set sal = sal +300where empno = p\_empno;  endif;  endloop;  *--关闭游标*  close cur\_emp;  *--提交修改*  commit;  end; |

## 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 +300where empno =7369;  ifsql%notfoundthen  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);  endif;  end; |

# 存储过程与存储函数

## 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。

### 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)]; |

### 无参存储过程

|  |
| --- |
| *-- 授予itcast创建存储过程的权限*  grantcreateprocedureto itcast;  */\**  *使用无参存储过程，注意无参存储过程创建时不能使用()*  *\*/*  createorreplaceprocedurepro\_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()*  execpro\_helloWorld; |

### 有输入参数存储过程

|  |
| --- |
| /\*  使用有输入参存储过程  \*/  createorreplaceprocedure pro\_add\_emp(  p\_empno in emp.empno%type,  p\_ename invarchar2,  p\_sal number  )  as  begin  *--将输入参数对应的数据插入emp表*  insertinto 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,'itcast2001',3000);  pro\_add\_emp(2002,'itcast2002',2000);  pro\_add\_emp(2003,'itcast2003',4000);  end; |

### 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  \*/  createorreplaceprocedure pro\_1to10\_sum(  p\_sum outnumber  )  as  tem\_sum number(4):=0;  begin  for i in1..10  loop  tem\_sum := tem\_sum + i;  endloop;  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; |

### 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  \*/  createorreplaceprocedure 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; |

### 程序中调用存储过程

|  |
| --- |
| **package** cn.itcast;  **import** java.sql.CallableStatement;  **import** java.sql.Connection;  **import** java.sql.DriverManager;  **import** java.sql.SQLException;  **import** oracle.jdbc.OracleTypes;  **publicclass** TestProcedure {  **publicstaticvoid** 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, "itcast", "itcast");  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();  }  }  }  } |

### 删除存储过程

|  |
| --- |
| 【语法】  DROP PROCEDURE <过程名>;  【示例】  dropprocedure pro\_1to10\_sum; |

## 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。

### 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； |

### 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()  \*/  createorreplacefunction fun\_helloWorld  returnvarchar2  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; |

### 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*  *\*/*  createorreplacefunction fun\_get\_annualSal\_by\_empno(p\_empno emp.empno%type)  returnnumber  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;  return12\*p\_sal +nvl(p\_comm,0);  end;  /  *-- 调用存储函数*  select fun\_get\_annualSal\_by\_empno(7369)from dual; |

### 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*  *\*/*  createorreplacefunction fun\_get\_annualSal\_by\_empno2(  p\_empno emp.empno%type,  p\_ename out emp.ename%type,  p\_comm out emp.comm%type  )  returnnumber  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;  return12\*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; |

### 程序中调用存储函数

|  |
| --- |
| **package** cn.itcast;  **import** java.sql.CallableStatement;  **import** java.sql.Connection;  **import** java.sql.DriverManager;  **import** java.sql.SQLException;  **import** oracle.jdbc.OracleTypes;  **publicclass** TestFunction {  **publicstaticvoid** 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, "itcast", "itcast");  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();  }  }  }  } |

### 删除存储函数

|  |
| --- |
| 【语法】  DROP FUNCTION <函数名>;  【示例】  dropfunction fun\_helloWorld;  dropfunction fun\_get\_annualSal\_by\_empno;  dropfunction fun\_get\_annualSal\_by\_empno2; |

## 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）

# 触发器

## 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; |

## 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*  *\*/*  createorreplacetrigger tri\_emp\_upd\_sal  after  updateof sal on emp  foreachrow  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！');  endif;  end;  /  *-- 更新工资值，并触发行级触发器*  update emp set sal =8888where 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*  createsequence seq\_emp\_empno;  *-- 2、创建触发器*  createorreplacetrigger tri\_emp\_ins\_empno  before  inserton emp  foreachrow  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*  insertinto emp(ename,sal)values('itcast002',2000);  commit; |

## 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*  createtable emp2 asselect\*from emp;  *-- 2、创建备份表emp\_bak*  createtable emp\_bak asselect\*from emp2 where1=2;  *-- 3、创建表触发器，当对表操作时触发*  createorreplacetrigger tri\_emp2\_del  before  deleteon emp2  begin  *-- 将emp2表中的数据备份到emp\_bak*  insertinto emp\_bak select\*from emp2;  end;  /  *-- 4、测试删除emp2表的数据*  deletefrom emp2;  select\*from emp2;  select\*from emp\_bak; |

## 开启禁用触发器

|  |
| --- |
| 【**禁用某个触发器**】  ALTER TRIGGER <触发器名> DISABLE  【示例】  altertrigger tri\_emp\_upd\_sal disable;  update emp set sal =8888where empno =1002;  【**重新启用触发器**】  ALTER TRIGGER <触发器名> ENABLE  【示例】  altertrigger tri\_emp\_upd\_sal enable;  update emp set sal =8888where empno =1002;  【**禁用表的所有触发器**】  ALTER TABLE <表名> DISABLE ALL TRIGGERS;  【示例】  altertable emp disablealltriggers;  【**启用表的所有触发器**】  ALTER TABLE <表名> ENABLE ALL TRIGGERS;  【示例】  altertable emp enablealltriggers;  【**删除触发器**】  DROP TRIGGER <触发器名>;  【示例】  droptrigger tri\_emp\_upd\_sal; |

# 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 |

# 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。

## 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itcast 创建角色的权限*  grantcreateroleto itcast;  *-- 创建角色*  createrole role\_itcast;  *-- 授予emp的select 操作权限给role\_itcast角色*  grantselecton emp to role\_itcast;  *-- 给scott用户授予role\_itcast的角色*  grant role\_itcast to scott; |

## 删除角色

|  |
| --- |
| 【语法】  DROP ROLE <role\_name>;  【示例】  droprole role\_itcast; |

# 闪回

## 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。

## 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。

## 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 scntimestamp\_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; |

## 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*  grantflashbackanytableto itcast;  *-- 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*  deletefrom emp where empno =7449;  commit;  *--允许行移动*  altertable emp enablerowmovement;  *-- 方式一；使用时间点闪回表*  flashbacktable emp totimestamp to\_timestamp('时间格式字符串','yyyy-mm-dd HH24:mi:ss');  *-- 方式二；使用SCN闪回表*  flashbacktable emp to SCN号; |

## 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*  droptable emp;  *-- 恢复表*  flashbacktable emp tobeforedrop; |

# 数据备份与恢复

## 数据备份

|  |
| --- |
| *--全表备份*  exp itcast/itcast@orcl file=d:\database\oracle\_data\itcast.dmp full=y;  *--指定表备份*  exp itcast/itcast@orcl file=d:\database\oracle\_data\itcast\_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。 |

## 数据恢复

|  |
| --- |
| *--全表恢复*  imp itcast/itcast@orcl ignore=y file=d:\database\oracle\_data\itcast.dmp full=y;  *--指定表恢复*  imp itcast/itcast@orcl ignore=y file=d:\database\oracle\_data\itcast\_emp\_dept.dmp tables=(emp,dept);  【说明】ignore：忽略创建错误 |

# 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)

.......

1. **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

1. **避免对大表进行无条件或无索引的的扫描**
2. **清空表时用TRUNCATE替代DELETE**
3. **尽量多使用COMMIT；因为COMMIT会释放回滚点**
4. **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;

1. **字符串型，能用=号，不用like**；=号表示精确比较，like表示模糊比较
2. **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

1. **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);

1. **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

1. **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。

1. **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。

# 项目1切换到oracle数据库

1. 添加oracle的驱动类包的到项目下的lib目录
2. 修改applicationContext.xml中的数据库方言
3. 修改db.properties中的数据库驱动类和用户名密码
4. 其它修改（如创建user表时将表名修改为t\_user，user是oracle数据库的关键字）

# oracle常见面试题

见《Oracle 常见面试题.docx》

# 练习题

见《练习题.docx》