1. **Oracle入门**
2. **数据库概述**

数据库（Database）是按照数据结构来组织、存储和管理数据的仓库，它产生于距今五十年前。简单来说是本身可视为电子化的文件柜——存储电子文件的处所，用户可以对文件中的数据运行新增、截取、更新、删除等操作。

**常见的数据模型**

* 1. 层次结构模型: 层次结构模型实质上是一种有根结点的定向有序树，IMS(Information Manage-mentSystem)是其典型代表。
  2. 网状结构模型：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，其典型代表是DBTG(Data Base Task Group)。
  3. 关系结构模型：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)。常见的有Oracle、mssql、mysql等

1. **主流数据库**

|  |  |  |  |
| --- | --- | --- | --- |
| **数据库名** | **公司** | **特点** | **工作环境** |
| mssql | **微软** | **只能能运行在windows平台，体积比较庞大，占用许多系统资源， 但使用很方便，支持命令和图形化管理，收费。** | **中型企业** |
| **Mysql** | **甲骨文** | **是个开源的数据库server，可运行在多种平台,** **特点是响应速度特别快，主要面向中小企业** | **中小型企业** |
| PostgreSQL |  | **号称“世界上最先进的开源数据库“，可以运行在多种平台下，是tb级数据库，而且性能也很好** | **中大型企业** |
| oracle | **甲骨文** | **获得最高认证级别的ISO标准安全认证，性能最高， 保持开放平台下的TPC-D和TPC-C的世界记录。但价格不菲** | **大型企业** |
| db2 | **IBM** | **DB2在企业级的应用最为广泛，  在全球的500家最大的企业中,几乎85%以上用DB2数据库服务器。收费** | **大型企业** |
| **Access** | **微软** | **Access是一种桌面数据库，只适合数据量少的应用，在处理少量 数据和单机访问的数据库时是很好的，效率也很高** | **小型企业** |

1. **Oracle数据库概述**

 ORACLE数据库系统是美国ORACLE公司（甲骨文）提供的以分布式数据库为核心的一组软件产品，是目前最流行的客户/服务器(CLIENT/SERVER)或B/S体系结构的数据库之一。

* + **拉里·埃里森**

****

* + **就业前景**

从就业与择业的角度来讲，计算机相关专业的大学生从事oracle方面的技术是职业发展中的最佳选择。

其一、就业面广：全球前100强企业99家都在使用ORACLE相关技术，中国政府机构，大中型企事业单位都能有ORACLE技术的工程师岗位。

其二、技术层次深：如果期望进入IT服务或者产品公司（类似毕博、DELL、IBM等），Oracle技术能够帮助提高就业的深度。

其三、职业方向多：Oracle数据库管理方向、Oracle开发及系统架构方向、Oracle数据建模数据仓库等方向。

1. **如何学习**

认真听课、多思考问题、多动手操作、有问题一定要问、多参与讨论、多帮组同学

**高级部分**

**PL/SQL编程**

**存储过程**

**和触发器**

**ORACLE**

**锁管理**

**索引、约束**

**视图和事务**

**ORACLE**

**表的查询**

**ORACLE**

**的函数**

**ORACLE**

**角色和权限**

**备份与恢复**

**ORACLE**

**表管理**

**ORACLE**

**体系结构**

**ORACLE**

**基本使用**

**ORACLE**

**用户管理**

**基础部分**

1. **体系结构**

oracle的体系很庞大，要学习它，首先要了解oracle的框架。oracle的框架主要由物理结构、逻辑结构、内存分配、后台进程、oracle例程、系统改变号 (System Change Number)组成

* + **物理结构**

物理结构包含三种数据文件:

1. 控制文件
2. 数据文件
3. 在线重做日志文件
   * **逻辑结构**

功能：数据库如何使用物理空间

组成：表空间、段、区、块的组成层次

1. **oracle安装、卸载和启动**
   * **硬件要求**

**物理内存：1GB**

**可用物理内存：50M**

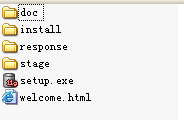
**交换空间大小：3.25GB**

**硬盘空间：10GB**

* + **安装**
    1. 安装程序成功下载，将会得到如下2个文件：

****

解压文件将得到database文件夹，文件组织如下：

****

点击setup.exe执行安装程序，开始安装。

* + 1. 点击安装程序将会出现如下安装界面，步骤 1/9：配置安全更新

****

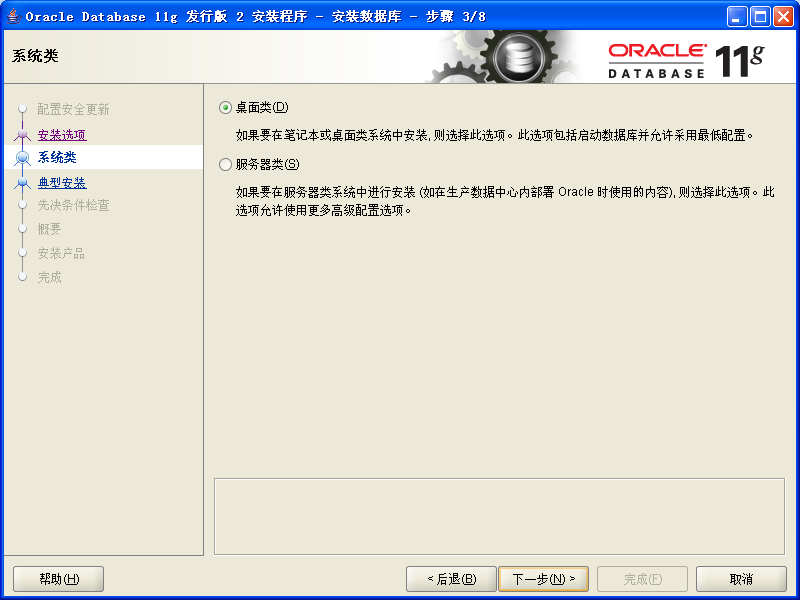
填写电子邮件地址(可以不填)，去掉复选框，点击下一步

* + 1. 步骤2/9：选择安装选项



勾选第一个，安装和配置数据库，点击下一步

* + 1. 步骤3/8：选择系统类



勾选第一个：桌面类，点击下一步

* + 1. 步骤4/8：配置数据库安装



选择安装路径，选择数据库版本(企业版)，选择字符集(默认值)

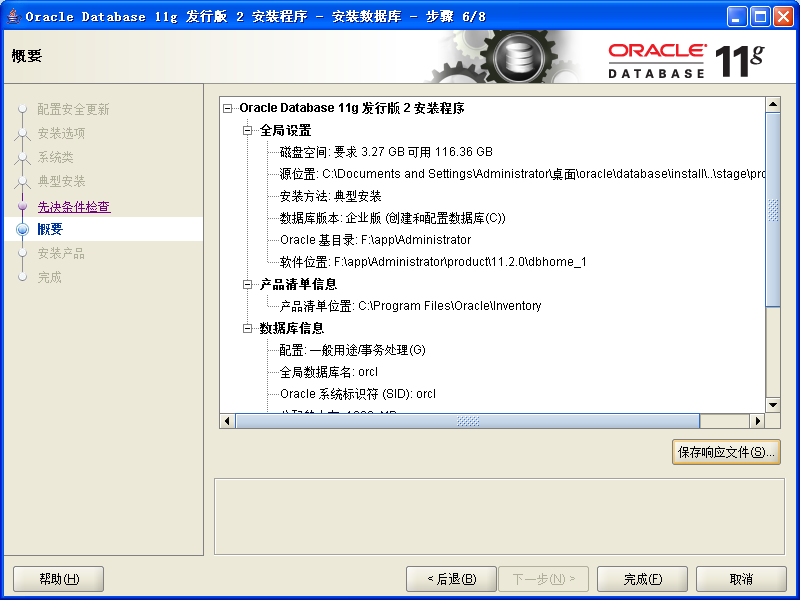
填写全局数据库名，管理口令

* + 1. 步骤5/8：先决条件检查



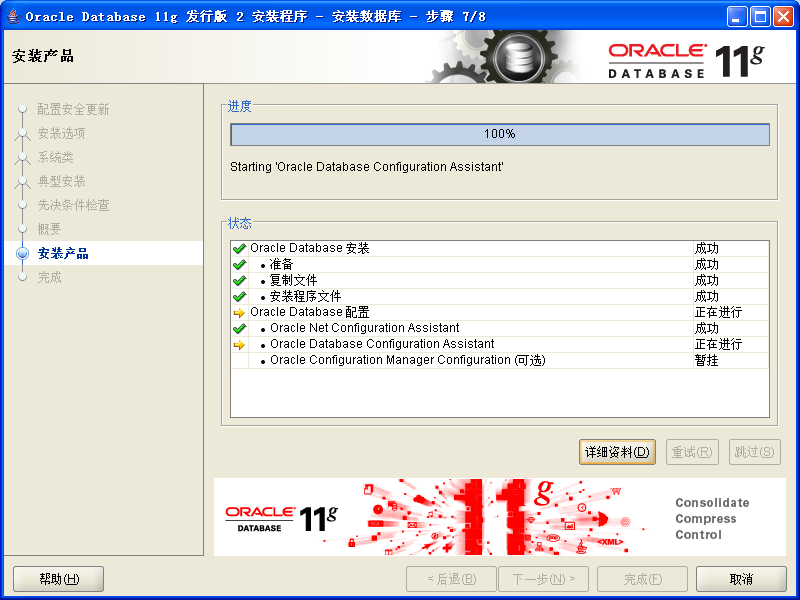
如果你的电脑满足要求但仍然显示检查失败，这时候直接忽略，勾选全部忽略

* + 1. 步骤6/8：概要信息



核对将要安装数据的详细信息，并保存响应文件，以备以后查看。然后点击完成数据库安装

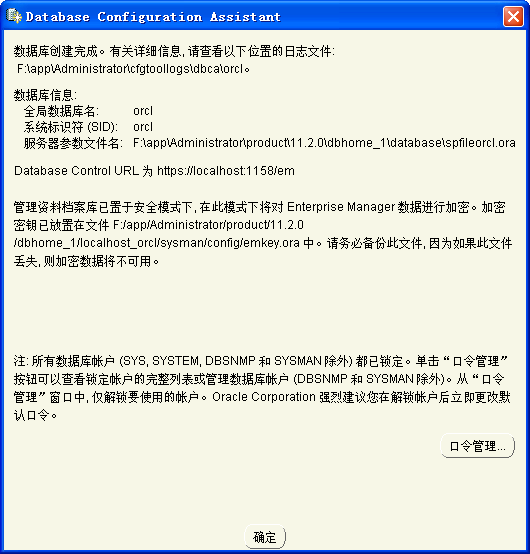
* + 1. 步骤7/8：安装产品





产品安装过程中将会出现以上2个界面

* + 1. 步骤8/8：完成安装



* + **卸载Oracle**
    1. 在运行services.msc打开服务，停止Oracle的所有服务。
    2. oracle11G自带一个卸载批处理\app\Administrator\product\11.2.0\dbhome\_1\deinstall\deinstall.bat
    3. 运行该批处理程序将自动完成oracle卸载工作，最后手动删除\app文件夹（可能需要重启才能删除）
    4. 运行regedit命令，打开注册表窗口。删除注册表中与Oracle相关的内容，具体如下：
       - 删除HKEY\_LOCAL\_MACHINE/SOFTWARE/ORACLE目录。
       - 删除HKEY\_LOCAL\_MACHINE/SYSTEM/CurrentControlSet/Services中所有以oracle或OraWeb为开头的键。
       - 删除HKEY\_LOCAL\_MACHINE/SYSETM/CurrentControlSet/Services/Eventlog/application中所有以oracle开头的键。
       - 删除HKEY\_CLASSES\_ROOT目录下所有以Ora、Oracle、Orcl或EnumOra为前缀的键。
       - 删除HKEY\_CURRENT\_USER/SOFTWARE/Microsoft/windows/CurrentVersion/Explorer/MenuOrder/Start Menu/Programs中所有以oracle 开头的键。
       - 删除HKDY\_LOCAL\_MACHINE/SOFTWARE/ODBC/ODBCINST.INI中除Microsoft ODBC for Oracle注册表键以外的所有含有Oracle的键。
       - 删除环境变量中的PATHT CLASSPATH中包含Oracle的值。
       - 删除“开始”/“程序”中所有Oracle的组和图标。
       - 删除所有与Oracle相关的目录，包括：

C:\Program file\Oracle目录。

ORACLE\_BASE目录。

C:\Documents and Settings\系统用户名、LocalSettings\Temp目录下的临时文件。

1. **oracle中的数据库**

存储的是数据对象

触发器

视图

表

同义词

Oracle

存储的是多个数据库

用户数据库

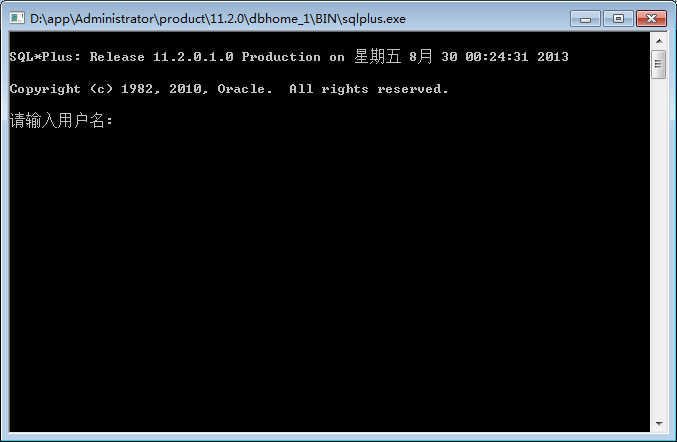
Pubs

Model

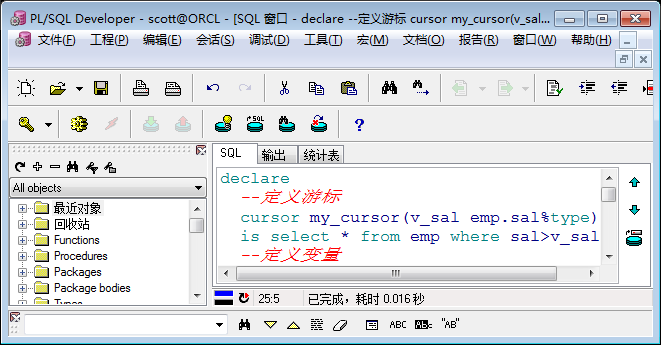
Master

SQLSERVER

1. **常用的工具**
   * **Sql Plus**

****

* + **Sql Developer**

****

* + **Oracle Enterprise Manager**

****

1. **用户和权限**
2. **用户介绍**

ORACLE用户是学习ORACLE数据库中的基础知识，下面就介绍下类系统常用的默认ORACLE用户:

* 1. **sys用户：**超级用户，完全是个SYSDBA(管理数据库的人)。拥有dba，sysdba，sysoper等角色或权限。是oracle权限最高的用户，登录时不能用normal。
  2. **system用户：**超级用户，默认是SYSOPT(操作数据库的人)，不过它也能以SYSDBA的权限登陆。拥有普通dba角色权限。
  3. **scott用户：**是个演示用户，是让你学习Oracle用的。

1. **常用命令**

学习oracle，首先我们必须要掌握常用的基本命令，oracle中的命令比较多，常用的命令如下：

* 1. **登录命令(sqlplus)**

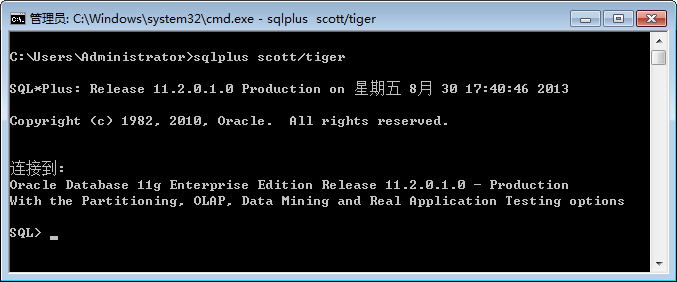
**说明：**用于登录到oracle数据库

**用法：**sqlplus 用户名/密码 [as sysdba/sysoper]

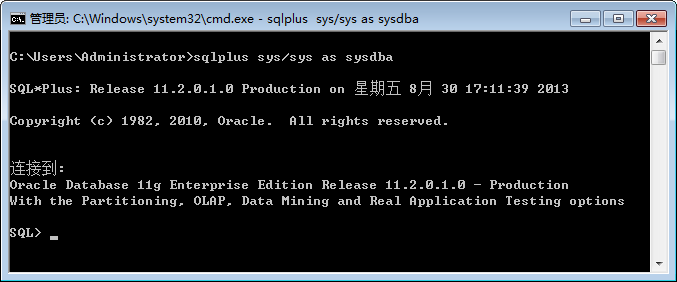
**注意：**当用特权用户登录时，必须带上sysdba或sysoper

**例子：**

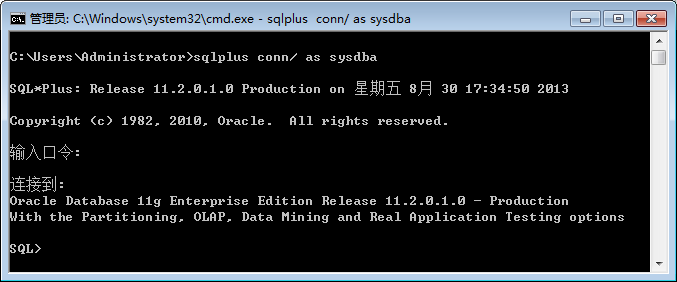
**普通用户登录**

****

**sys用户登录**

****

**操作系统的身份登录**

****

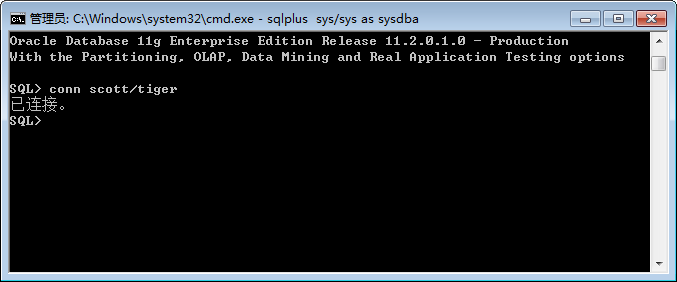
* 1. **连接命令(conn)**

**说明：**用于连接到oracle数据库，也可实现用户的切换

**用法：**conn 用户名/密码 [as sysdba/sysoper]

**注意：**当用特权用户连接时，必须带上sysdba或sysoper

**例子：**

****

* 1. **断开连接(disc)**

**说明：**断开与当前数据库的连接

**用法：**disc

* 1. **显示用户名(show user)**

**说明：**显示当前用户名

**用法：**show user

* 1. **退出(exit)**

**说明：**断开与当前数据库的连接并会退出

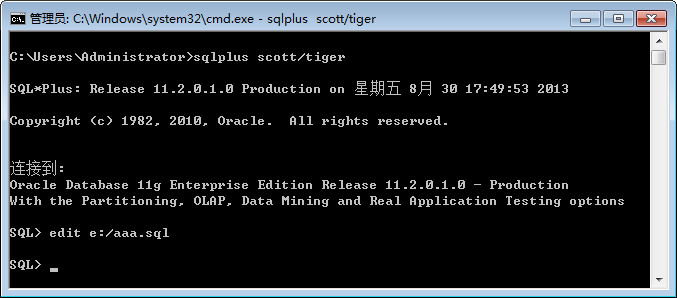
**用法：exit**

* 1. **编辑脚本(edit/ed)**

**说明：**编辑指定或缓冲区的sql脚本

**用法：**edit [文件名]

**列子：**

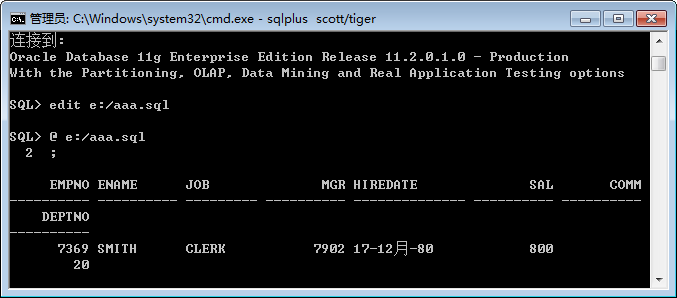
****

* 1. **运行脚本 (start/@)**

**说明：**运行指定的sql脚本

**用法：**start/@ 文件名

**列子：**

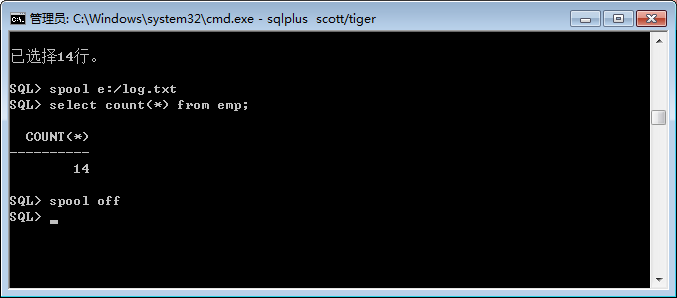
****

* 1. **印刷屏幕 (spool)**

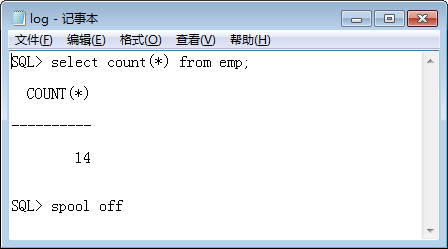
**说明：**将sql\*plus屏幕中的内容输出到指定的文件

**用法：**开始印刷-**>**spool 文件名 结束印刷->spool off

**列子：**

****

**文件内容**

****

* 1. **显示宽度 (linesize)**

**说明：**设置显示行的宽度，默认是80个字符

**用法：**set linesize 120

* 1. **显示页数 (pagesize)**

**说明：**设置每页显示的行数，默认是14页

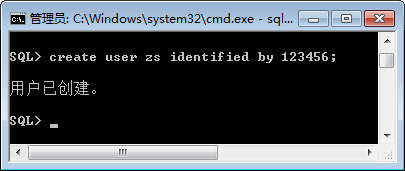
**用法：**set pagesize 20

1. **用户管理**
   1. **创建用户**

**说明：**Oracle中需要创建用户一定是要具有dba(数据库管理员)权限的用户才能创建，而且创建的新用户不具备任何权限，连登录都不可以。

**用法：**create user 新用户名 identified by 密码

例子：



* 1. **修改密码**

**说明：**修改用户密码一般有两种方式，一种是通过命令password修改，另一种是通过语句alter user实现，如果要修改他人的密码，必须要具有相关的权限才可以

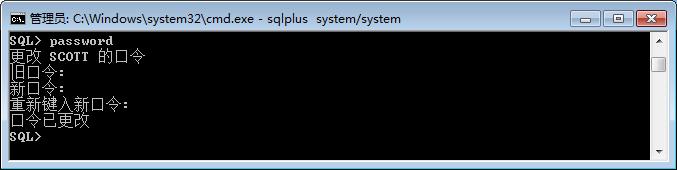
**用法：**

方式一 password [用户名]

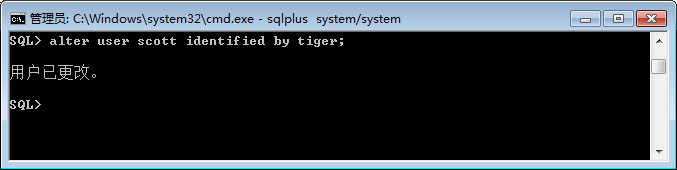
方式二 alert user 用户名 identified by 新密码

**例子：**

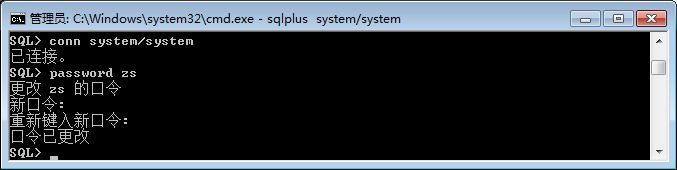
修改当前用户(方式一)



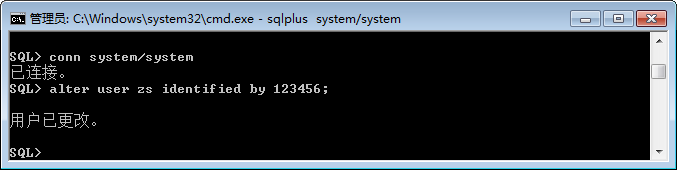
修改当前用户(方式二)



修改其他用户(方式一)



修改其他用户(方式二)



* 1. **用户禁用与启用**

**说明：**Oracle中想要禁用或启用一个账户也同样是使用alter user 命令来完成，只是语法和修改密码有所不同。

**用法：**

禁用 alert user 用户名 account lock

启用 alert user 用户名 account unlock

* 1. **删除用户**

**说明：**Oracle中要删除一个用户，必须要具有dba的权限。而且不能删除当前用户，如果删除的用户有数据对象，那么必须加上关键字cascade。

**用法：**drop user 用户名 [cascade]

1. **用户权限与角色**
   1. **权限**

Oracle中权限主要分为两种，系统权限和实体权限。

* **系统权限：**系统规定用户使用数据库的权限。（系统权限是对用户而言)。
* DBA: 拥有全部特权，是系统最高权限，只有DBA才可以创建数据库结构。
* RESOURCE:拥有Resource权限的用户只可以创建实体，不可以创建数据库结构。
* CONNECT:拥有Connect权限的用户只可以登录Oracle，不可以创建实体，不可以创建数据库结构。

**注意：**

对于普通用户：授予connect, resource权限。

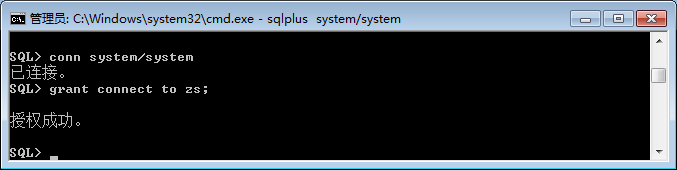
对于DBA管理用户：授予connect，resource, dba权限。

* **授予系统权限**

**说明：**要实现授予系统权限只能由DBA用户授出。

**用法：**grant 系统权限1[,系统权限2]… to 用户名1[,用户名2]….

**例子：**

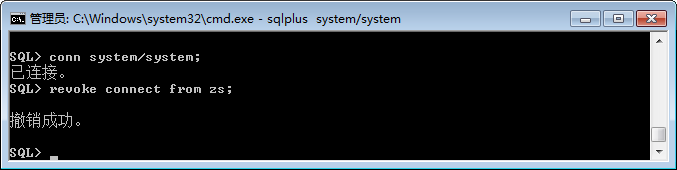


* **系统权限回收：**

**说明：**系统权限只能由DBA用户回收

**用法：**revoke 系统权限 from 用户名

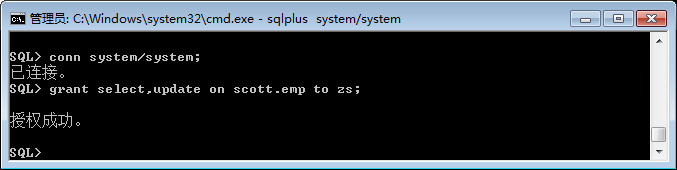
**例子：**

****

* **实体权限：**某种权限用户对其它用户的表或视图的存取权限。（是针对表或视图而言的）。主要包括select, update, insert, alter, index, delete, all其中all包括所有权限。
* **授予实体权限**

**用法：**grant 实体权限1[,实体权限2]… on 表名 to用户名1[,用户名2]….

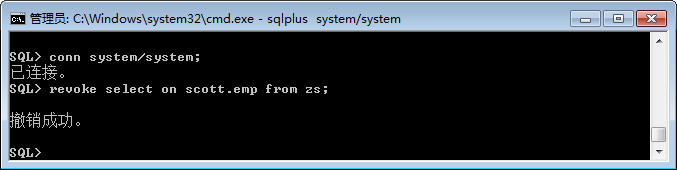
**例子：**



* **实体权限回收**

**用法：**revoke 实体权限 on 表名from 用户名

**例子：**



* **查询用户拥有哪里权限：**

SQL> select \* from role\_tab\_privs;//查询授予角色的对象权限

SQL> select \* from role\_role\_privs;//查询授予另一角色的角色

SQL> select \* from DBA\_tab\_privs;//查询直接授予用户的对象权限

SQL> select \* from dba\_role\_privs;//查询授予用户的角色

SQL> select \* from dba\_sys\_privs;//查询授予用户的系统权限

SQL> select \* from role\_sys\_privs;//查询授予角色的系统权限

SQL> Select \* from session\_privs;// 查询当前用户所拥有的权限

* 1. **角色**

角色。角色是一组权限的集合，将角色赋给一个用户，这个用户就拥有了这个角色中的所有权限。

* **系统预定义角色**

预定义角色是在数据库安装后，系统自动创建的一些常用的角色。下面我们就简单介绍些系统角色：

* + - * CONNECT, RESOURCE, DBA这些预定义角色主要是为了向后兼容。其主要是用于数据库管理。oracle建议用户自己设计数据库管理和安全的权限规划，而不要简单的使用这些预定角色。将来的版本中这些角色可能不会作为预定义角色。
      * DELETE\_CATALOG\_ROLE， EXECUTE\_CATALOG\_ROLE，SELECT\_CATALOG\_ROLE这些角色主要用于访问数据字典视图和包。
      * EXP\_FULL\_DATABASE， IMP\_FULL\_DATABASE这两个角色用于数据导入导出工具的使用。
* **自定义角色**

Oracle建议我们自定义自己的角色，使我们更加灵活方便去管理用户

* **创建角色**

SQL> create role admin;

* **授权给角色**

SQL> grant connect,resource to admin;

* 撤销角色的权限

SQL> revoke connect from admin;

* 删除角色

SQL> drop role admin;

1. **Sql查询与函数**
2. **SQL概述**

SQL（Structured Query Language)结构化查询语言，是一种数据库查询和程序设计语言，用于存取数据以及查询、更新和管理关系数据库系统。同时也是数据库脚本文件的扩展名。

* SQL语言主要包含5个部分
  + 数据定义语言Data Definition Language(DDL），用来建立数据库、数据对象和定义其列。例如：CREATE、DROP、ALTER等语句。
  + 数据操作语言Data Manipulation Language(DML），用来插入、修改、删除、查询，可以修改数据库中的数据。例如：INSERT（插入）、UPDATE（修改）、DELETE（删除）语句
  + 数据查询语言 (Data Query Language, DQL) 是SQL语言中，负责进行数据查询而不会对数据本身进行修改的语句，这是最基本的SQL语句。例如：SELECT（查询）
  + 数据控制语言Data Controlling Language（DCL），用来控制数据库组件的存取允许、存取权限等。例如：GRANT、REVOKE、COMMIT、ROLLBACK等语句。
  + 事务控制语言（Transactional Control Language，TCL），用于维护数据的一致性，包括COMMIT（提交事务）、ROLLBACK（回滚事务）和SAVEPOINT（设置保存点）3条语句

1. **Oracle的数据类型**

|  |  |  |  |
| --- | --- | --- | --- |
| **类型** | | **参数** | **描述** |
| 字符类型 | char | 1~2000字节 | 固定长度字符串，长度不够的用空格补充 |
| varchar2 | 1~4000字节 | 可变长度字符串，与CHAR类型相比，使用VARCHAR2可以节省磁盘空间，但查询效率没有char类型高 |
| 数值类型 | Number(m,n) | m(1~38)  n(-84~127) | 可以存储正数、负数、零、定点数和精度为38位的浮点数，其中，M表示精度，代表数字的总位数；N表示小数点右边数字的位数 |
| 日期类型 | date | 7字节 | 用于存储表中的日期和时间数据，取值范围是公元前4712年1月1日至公元9999年12月31日，7个字节分别表示世纪、年、月、日、时、分和秒 |
| 二进制数据类型 | row | 1~2000字节 | 可变长二进制数据，在具体定义字段的时候必须指明最大长度n |
| long raw | 1~2GB | 可变长二进制数据 |
| LOB数据类型 | clob | 1~4GB | 只能存储字符数据 |
| nclob | 1~4GB | 保存本地语言字符集数据 |
| blob | 1~4GB | 以二进制信息保存数据 |

1. **DDL语言**
   1. **Create table命令**

用于创建表。在创建表时，经常会创建该表的主键、外键、唯一约束、Check约束等

* **语法结构**

|  |
| --- |
| create table 表名(  [字段名] [类型] [约束]  ………..  CONSTRAINT fk\_column  FOREIGN KEY(column1,column2,…..column\_n)  REFERENCES tablename(column1,column2,…..column\_n)  ) |

* **例子：**

|  |
| --- |
| create table student(  stuNo char(32) primary key,*--主键约束*  stuName varchar2(20) not null,*--非空约束*  cardId char(20) unique,*--唯一约束*  sex char(2) check(sex='男' or sex='女'),*--检查约束*  address varchar2(100) default '地址不详'*--默认约束*  )  create table mark(  mid int primary key,*--主键约束*  stuNo char(32) not null,  courseName varchar2(20) not null,*--非空约束*  score number(3) not null check(score>=0 and score<=100),  *--非空约束,检查约束*  constraint fk\_stuno foreign key(stuno) references student(stuNo)  *--表级外键约束*  ) |

* 1. **Alter table命令**

对已经存在的表进行修改，可以新增或删除字段，修改字段名或其类型和类型长度。

* 修改表名

|  |
| --- |
| alter table old\_table rename to new\_table |

* 修改字段名

|  |
| --- |
| alter table table\_name rename column old\_column to new\_column |

* 添加字段

|  |
| --- |
| alter table table\_name add new\_column varhcar2(200) default ‘null’ |

* 修改数据类型

|  |
| --- |
| alter table table\_name modifiy filedname carchar2(100) |

* 删除字段

|  |
| --- |
| alter table table\_name drop column column\_name |

* 添加主键

|  |
| --- |
| alter table table\_name add constraint pk\_nameprimary key(column\_id) |

* 删除主键

|  |
| --- |
| alter table table\_name drop constraint pk\_name |

* 添加外键

|  |
| --- |
| alter table archives  add constraint FK\_BUYING\_REFERENCE\_USER\_INF foreign key (personnel\_id)  references personnel (personnel\_id); |

* 删除外键

|  |
| --- |
| alter table table\_name drop constraint fk\_name |

* 1. **Drop table命令**

用于从数据库中删除表及全部数据

|  |
| --- |
| drop table table\_name |

* 1. **Truncate table命令**

可以快速删除表的记录并释放空间，不使用事务处理，速度快且效率高，但无法回滚事务。

|  |
| --- |
| truncatetable table\_name |

* 1. **其他create命令**
* CREATE INDEX：创建数据表索引
* CREATE PROCEDURE：创建存储过程
* CREATE FUNCTION：创建用户函数
* CREATE VIEW：创建视图
* CREATE TRIGGER：创建触发程序
* CREATE SEQUENCE
  1. **SEQUENCE**

在oracle中sequence就是所谓的序列号，每次取的时候它会自动增加，一般用在需要按序列号排序的地方。

* **创建语法**

|  |
| --- |
| CREATE SEQUENCE **sequence\_name**  INCREMENTBY **1** --每次加几个  STARTWITH **1** --从1开始计数  NOMAXVALUE--不设置最大值  NOCYCLE--一直累加，不循环  CACHE **10** --使序列号预分配 |

* **列子**

|  |
| --- |
| SELECT sequence\_marks.currval from marks  insert into table\_name values(sequence\_marks.nextval) |

* **删除**

|  |
| --- |
| drop sequence **sequence\_name** |

1. **DML语言**
   1. **INSERT语句**

|  |
| --- |
| Insert into table\_name(column1,column2…….column\_n) values(val1….) |

* 1. **UPDATE语句**

|  |
| --- |
| update table\_name set column1=value,…. where [condition…] |

* 1. **DELETE语句**

|  |
| --- |
| Delete from table\_name where [condition…] |

1. **DQL语言**
   1. **概述**

* **基本语法**

|  |
| --- |
| **SELECT column\_list[\*查询所有数据]**  **[ INTO new\_table ]**  **FROM table\_source**  **[ WHERE search\_condition ]**  **[ GROUP BY group\_by\_expression ]**  **[ HAVING search\_condition ]**  **[ ORDER BY order\_expression [ ASC | DESC ] ]** |

* **例子**

|  |
| --- |
| SQL>select \* from em--查询所有数据  SQL>select ename,job from em--查询指定的字段数据  SQL> select \* from emp where sal>1000--加条件 |

* 1. **聚合函数**

聚合函数对一组值执行计算并返回单一的值。聚合函数忽略空值。聚合函数经常与 SELECT 语句的 GROUP BY 子句一同使用。不能在 WHERE 子句中使用组函数。

* AVG(expression): 返回集合中各值的平均值

|  |
| --- |
| --查询所有人都的平均工资  select avg(sal) from emp |

* COUNT(expression): 以 Int32 形式返回集合中的项数

|  |
| --- |
| --查询工资低于2000的人数  select count(\*) from emp where sal<2000 |

* MAX(expression): 返回集合中的最大值

|  |
| --- |
| --查询最高工资  select max(sal) from emp |

* MIN(expression): 返回集合中的最小值

|  |
| --- |
| --查询最低工资  select max(sal) from emp |

* SUM(expression): 返回集合中所有值的总和

|  |
| --- |
| --查询部门编号为20的工资总和  select sum(sal) from emp where deptno=20 |

* 1. **排序函数**

用于根据指定的列对结果集进行排序

* ORDER BY 语句

|  |
| --- |
| --查询所有信息并按工资排序  select \* from emp order by sal |

* 升序(asc)

|  |
| --- |
| --查询所有信息并按工资升序排序  select \* from emp order by sal asc |

* 降序(desc)

|  |
| --- |
| --查询所有信息并按工资降序排序  select \* from emp order by sal desc |

* 1. **分组函数**

分组函数作用于一组数据，并对一组数据返回一个值。

* GROUP BY 子句

|  |
| --- |
| --查询每个部门的平均工资  select avg(sal) from emp group by deptno |

* HAVING 子句

|  |
| --- |
| --查询部门的平均工资大于2000  select avg(sal) from emp group by deptno having avg(sal)>2000 |

* 1. **连接查询**

连接查询是关系数据库中最主要的查询，主要包括内连接、外连接和交叉连接等。通过连接运算符可以实现多个表查询。

* **内连接**

内连接也叫连接，是最早的一种连接。还可以被称为普通连接或者自然连接，内连接是从结果表中删除与其他被连接表中没有匹配行的所有行，所以内连接可能会丢失信息。

* 等值连接:

|  |
| --- |
| select \* from emp inner join dept on emp.deptno=dept.deptno  select \* from emp,dept where emp.deptno=dept.deptno |

* 不等值连接:

|  |
| --- |
| select \* from emp inner join dept on emp.deptno!=dept.deptno |

* **外连接**

外连接分为三种：左外连接，右外连接，全外连接。对应SQL：LEFT/RIGHT/FULL OUTER JOIN。通常我们省略outer 这个关键字。写成：LEFT/RIGHT/FULL JOIN。

* + 左外连接(left join): 是以左表的记录为基础的

|  |
| --- |
| select \* from emp left join dept on emp.deptno=dept.deptno |

* + 右外连接(right join): 和left join的结果刚好相反,是以右表(BL)为基础的

|  |
| --- |
| select \* from emp right join dept on emp.deptno=dept.deptno |

* + 全外连接(full join): 左表和右表都不做限制，所有的记录都显示，两表不足的地方用null 填充

|  |
| --- |
| select \* from emp full join dept on emp.deptno=dept.deptno |

* **交叉连接**

交叉连接即笛卡儿乘积，是指两个关系中所有元组的任意组合。一般情况下，交叉查询是没有实际意义的。

|  |
| --- |
| select \* from cross full join dept |

* 1. **常用查询**
     + **like模糊查询**

|  |
| --- |
| --查询姓名首字母为S开始的员工信息  select \* from emp where ename like 'S%'  --查询姓名第三个字母为A的员工信息  select \* from emp where ename like '\_\_A%' |

* + - **is null/is not null 查询**

|  |
| --- |
| --查询没有奖金的雇员信息  select \* from emp where comm is null  --查询有奖金的雇员信息  select \* from emp where comm is not null |

* + - **in查询**

|  |
| --- |
| --查询雇员编号为7566、7499、7844的雇员信息  select \* from emp where empno in(7566,7499,7844) |

* + - **exists/not exists查询(效率高于in)**

|  |
| --- |
| --查询有上级领导的雇员信息  select \* from emp e where exists  (select \* from emp where empno=e.mgr)  --查询没有上级领导的雇员信息  select \* from emp e where not exists  (select \* from emp where empno=e.mgr) |

* + - **all查询**

|  |
| --- |
| --查询比部门编号为20的所有雇员工资都高的雇员信息  select \* from emp where sal > all(select sal from emp where deptno=20) |

* + - **union合并不重复**

|  |
| --- |
| select \* from emp where comm is not null  union  select \* from emp where sal>3000 |

* + - **union all合并重复**

|  |
| --- |
| select \* from emp where comm is not null  union all  select \* from emp where sal>3000 |

* 1. **子查询**

当一个查询是另一个查询的条件时，称之为子查询。子查询是一个 SELECT 语句，它嵌套在一个 SELECT、SELECT...INTO 语句、INSERT...INTO 语句、DELETE 语句、或 UPDATE 语句或嵌套在另一子查询中。

* + - **在CREATE TABLE语句中使用子查询**

|  |
| --- |
| --创建表并拷贝数据  create table temp(id,name,sal) as select empno,ename,sal from emp |

* + - **在INSERT语句中使用子查询**

|  |
| --- |
| --当前表拷贝  insert into temp(id,name,sal) select \* from temp  --从其他表指定字段拷贝  insert into temp(id,name,sal) select empno,ename,sal from emp |

* + - **在DELETE语句中使用子查询**

|  |
| --- |
| --删除SALES部门中的所有雇员  delete from emp where deptno in  (select deptno from dept where dname='SALES') |

* + - **在UPDATE语句中使用子查询**

|  |
| --- |
| --修改scott用户的工资和smith的工资一致  update emp set sal=(select sal from emp where ename='SMITH') where ename='SCOTT'  --修改black用户的工作,工资,奖金和scott一致  update emp set(job,sal,comm)=(select job,sal,comm from emp where ename='SCOTT') where ename='BLAKE' |

* + - **在SELECT语句中使用子查询**

|  |
| --- |
| --查询和ALLEN同一部门的员工信息  select \* from emp where deptno in  (select deptno from emp where ename='ALLEN')  --查询工资大于部门平均工资的雇员信息  select \* from emp e  (select avg(sal) asal,deptno from emp group by deptno) t  where e.deptno=t.deptno and e.sal>t.asal |

1. **TCL语言**
   1. **COMMIT**

|  |
| --- |
| commit --提交事务 |

* 1. **ROLLBACK**

|  |
| --- |
| rollback to p1 --回滚到指定的保存点  rollback --回滚所有的保存点 |

* 1. **SAVEPOINT**

|  |
| --- |
| savepoint p1 --设置保存点 |

* 1. **只读事务**

只读事务是指只允许执行查询的操作，而不允许执行任何其它dml操作的事务，它的作用是确保用户只能取得某时间点的数据。

|  |
| --- |
| set transaction read only |

1. **oracle函数**
   1. **字符串函数**

字符串函数是oracle中比较常用的，下面我们就介绍些常用的字符串函数:

* + - concat：字符串连接函数，也可以使用’||’

|  |
| --- |
| --将职位和雇员名称显示在一列中  select concat(ename,concat('(',concat(job,')'))) from emp  select ename || '(' || job || ')' from emp |

* + - length：返回字符串的长度

|  |
| --- |
| --查询雇员名字长度为5个字符的信息  select \* from emp where length(ename)=5 |

* + - lower：将字符串转换成小写

|  |
| --- |
| --以小写方式显示雇员名  select lower(ename) from emp |

* + - upper：将字符串转换成大写

|  |
| --- |
| --以大写方式显示雇员名  select upper (ename) from emp |

* + - substr：截取字符串

|  |
| --- |
| --只显示雇员名的前3个字母  select substr(ename,0,3) from emp |

* + - replace：替换字符串

|  |
| --- |
| --将雇员的金额显示为\*号  select ename,replace(sal,sal,’\*’) from emp |

* + - instr：查找字符串

|  |
| --- |
| --查找雇员名含有’LA’字符的信息  select \* from emp where instr(ename,’LA’)>0 |

* 1. **日期函数**
     + sysdate：返回当前session所在时区的默认时间

|  |
| --- |
| --获取当前系统时间  select sysdate from dual |

* + - add\_months：返回指定日期月份+n之后的值，n可以为任何整数

|  |
| --- |
| --查询当前系统月份+2的时间  select add\_months(sysdate,2) from dual  --查询当前系统月份-2的时间  select add\_months(sysdate,-2) from dual |

* + - last\_day：返回指定时间所在月的最后一天

|  |
| --- |
| --获取当前系统月份的最后一天  select last\_day(sysdate) from dual |

* + - months\_between：返回月份差，结果可正可负，当然也有可能为0

|  |
| --- |
| --获取入职日期距离当前时间多少天  select months\_between(sysdate, hiredate) from emp |

* + - trunc：为指定元素而截去的日期值

|  |
| --- |
| --获取当前系统年，其他默认  select trunc(sysdate,'yy') from dual  --查询81年2月份入职的雇员  select \* from emp  where trunc(hiredate,'mm')=trunc(to\_date('1981-02','yyyy-mm'),'mm') |

* 1. **转换函数**
     + to\_char：将任意类型转换成字符串

|  |
| --- |
| --日期转换  select to\_char(sysdate, 'yyyy-mm-dd hh24:mi:ss') from dual  --数字转换  select to\_char(-100.789999999999,'L99G999D999') from dual |

* + - 数字格式控制符

|  |  |
| --- | --- |
| **符号** | **描述** |
| 9 | 代表一位数字，如果当前位有数字，显示数字，否则不显示(小数部分仍然会强制显示) |
| 0 | 强制显示该位，如果当前位有数字，显示数字，否则显示0 |
| $ | 增加美元符号显示 |
| L | 增加本地货币符号显示 |
| . | 小数点符号显示 |
| , | 千分位符号显示 |

* + - to\_date：将字符串转换成日期对象

|  |
| --- |
| --字符转换成日期  select to\_date('2011-11-11 11:11:11', 'yyyy-mm-dd hh24:mi:ss') from dual |

* + - to\_number：将字符转换成数字对象

|  |
| --- |
| --字符转换成数字对象  select to\_number('209.976')\*5 from dual  select to\_number('209.976', '9G999D999')\*5 from dual |

* 1. **数学函数**
     + abs：返回数字的绝对值

|  |
| --- |
| select abs(-1999) from dual |

* + - ceil：返回大于或等于n的最小的整数值

|  |
| --- |
| select ceil(2.48) from dual |

* + - floor：返回小于等于n的最大整数值

|  |
| --- |
| select floor(2.48) from dual |

* + - round：四舍五入

|  |
| --- |
| select round(2.48) from dual  select round(2.485,2) from dual |

* + - bin\_to\_num：二进制转换成十进制

|  |
| --- |
| select bin\_to\_num(1,0,0,1,0) from dual |

1. **锁**
2. **概述**

锁是实现数据库并发控制的一个非常重要的技术。当事务在对某个数据对象进行操作前，先向系统发出请求，对其加锁。加锁后事务就对该数据对象有了一定的控制，在该事务释放锁之前，其他的事务不能对此数据对象进行更新操作。

在数据库中有两种基本的锁类型：排它锁（Exclusive Locks，即X锁）和共享锁（Share Locks，即S锁）。当数据对象被加上排它锁时，其他的事务不能对它读取和修改。加了共享锁的数据对象可以被其他事务读取，但不能修改。

根据保护的对象不同，Oracle数据库锁可以分为以下几大类：

* DML锁（data locks，数据锁），用于保护数据的完整性
* DDL锁（dictionary locks，字典锁），用于保护数据库对象的结构，如表、索引等的结构定义
* 内部锁和闩（internal locks and latches），保护数据库的内部结构

1. **DML锁**

DML锁的目的在于保证并发情况下的数据完整性，在Oracle数据库中，DML锁主要包括TM锁和TX锁，其中TM锁称为表级锁，TX锁称为事务锁或行级锁。

* 1. **行级锁**

当事务执行数据库插入、更新、删除操作时，该事务自动获得操作表中操作行的排它锁

|  |
| --- |
| --不允许其他用户对雇员表的部门编号为20的数据进行修改  select \* from emp where deptno=20 for update  --不允许其他用户对雇员表的所有数据进行修改  select \* from emp for update  --如果已经被锁定，就不用等待  select \* from emp for update nowait  --如果已经被锁定，更新的时候等待5秒  select \* from emp for update wait 5 |

* 1. **锁模式**
     + 0(none)
     + 1(null)
     + 2(rs)：行共享
     + 3(rx)：行排他
     + 4(s)：共享
     + 5(srx)：共享行排他
     + 6(x)：排他

数字越大，锁级别越高

* 1. **表级锁**

当事务获得行锁后，此事务也将自动获得该行的表锁(行排他),以防止其它事务进行DDL语句影响记录行的更新

* + - 行共享锁(RS锁)：允许用户进行任何操作，禁止排他锁

|  |
| --- |
| lock table emp in row share mode |

* + - 行排他锁(RX锁)：允许用户进行任何操作，禁止共享锁

|  |
| --- |
| lock table emp in row exclusive mode |

* + - 共享锁(R锁)：其他用户只能看，不能修改

|  |
| --- |
| lock table emp in share mode |

* + - 排他锁(X锁)：其他用户只能看，不能修改，不能加其他锁

|  |
| --- |
| lock table emp in exclusive mode |

* + - 共享行排他(SRX锁)：比行排他和共享锁级别高，不能添加共享锁

|  |
| --- |
| lock table emp in share row exclusive mode |

* 1. **锁兼容性**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **S** | **X** | **RS** | **RX** | **SRX** | **N/A** |
| **S** | Y | N | Y | N | N | Y |
| **X** | N | N | N | N | N | Y |
| **RS** | Y | N | Y | Y | Y | Y |
| **RX** | N | N | Y | Y | N | Y |
| **SRX** | N | N | Y | N | N | Y |
| **N/Y** | Y | Y | Y | Y | Y | Y |

* 1. **死锁**

当两个事务需要一组有冲突的锁，而不能将事务继续下去的话，就出现死锁。

1. 用户A修改A表，事务不提交
2. 用户B修改B表，事务不提交
3. 用户A修改B表，阻塞
4. 用户B修改A表，阻塞

Oracle系统能自动发现死锁，并会自动选择工作量最少的事务进行撤销和释放所有锁

* 1. **悲观锁和乐观锁**

数据的锁定分为两种方法，第一种叫做悲观锁，第二种叫做乐观锁

* + - 悲观锁：就是对数据的冲突采取一种悲观的态度，也就是说假设数据肯定会冲突，所以在数据开始读取的时候就把数据锁定住。
    - 乐观锁：就是认为数据一般情况下不会造成冲突，所以在数据进行提交更新的时候，才会正式对数据的冲突与否进行检测，如果发现冲突了，则让用户返回错误的信息，让用户决定如何去做。

1. **DDL锁**
   1. **排它DDL锁**

创建、修改、删除一个数据库对象的DDL语句获得操作对象的排它锁。

* 1. **共享DDL锁**

需在数据库对象之间建立相互依赖关系的DDL语句通常需共享获得DDL锁

* 1. **分析锁**

分析锁是一种独特的DDL锁类型，ORACLE使用它追踪共享池对象及它所引用数据库对象之间的依赖关系

1. **内部锁和闩**

这是ORACLE中的一种特殊锁，用于顺序访问内部系统结构。当事务需向缓冲区写入信息时，为了使用此块内存区域，ORACLE首先必须取得这块内存区域的闩锁，才能向此块内存写入信息。

1. **数据库对象**
2. **概述**

ORACLE数据库主要有如下数据库对象：

* tablespace and datafile(表空间和数据文件)
* table(表)
* constraints(约束)
* index(索引)
* view(试图)
* sequence(序列)
* synonyms(同义词)
* DB-link(数据库链路)

1. **表空间和数据文件**

表空间是数据库的逻辑组成部分，从物理上讲，数据库数据是存放在数据文件中，从逻辑上讲数据库则是存放在表空间中，表空间是由一个或多个数据文件组成。

**表空间**

**数据**

**文件**

* **表空间**
  + 某一时刻只能属于一个数据库
  + 由一个或多个数据文件组成
  + 可进一步划分为逻辑存储
  + 表空间主要分为两种
    - System表空间
      * 随数据库创建
      * 包含数据字典
      * 包含system还原段
    - 非system表空间
      * 用于分开存储段
      * 易于空间管理
      * 控制分配给用户的空间量
* **数据文件**
  + 只能属于一个表空间和一个数据库
  + 是方案对象数据的资料档案库
* **创建表空间**
  + 语法

CREATE TABLESPACE tablespacename

[DATAFILE clause]

[MINIMUM EXTENT integer[k|m]]

[BLOCKSIZE integer[k]]

[LOGGING|NOLOGGING]

[DEFAULT storage\_clause]

[ONLINE|OFFLINE]

[PERMANENT|TEMPORARY]

[extent\_management\_clause]

[segment\_management\_clause]

* + 例子

|  |
| --- |
| *--创建本地管理表空间*  create tablespace firstSpance  datafile 'e:/firstspance.dbf'size 100M  extent management local uniform size 256k  *--修改文件大小*  alter database datafile 'e:/firstspance.dbf' resize 110m  *--删除表空间*  drop tablespace firstSpance INCLUDING CONTENTS and datafiles  *--使用数据库表空间*  *--创建用户指定表空间*  create user guest identified by 123456  default tablespace firstSpance  *--表中指定表空间*  create table account(  accountid number(4),  accountName varchar2(20)  )tablespace firstSpance  *--表空间脱机*  alter tablespace firstSpance offline  *--表空间联机*  alter tablespace firstSpance online  *--表空间只读，不能进行dml操作*  alter tablespace firstSpance read only |

1. **同义词**

Oracle数据库中提供了同义词管理的功能。同义词是数据库方案对象的一个别名，经常用于简化对象访问和提高对象访问的安全性。Oracle同义词有两种类型，分别是公用Oracle同义词与私有Oracle同义词。

* **公有同义词**
  + **语法**

CREATE [OR REPLACE] PUBLIC SYNONYM sys\_name FOR [SCHEMA.] object\_name

* + **创建（需拥有**CREATE PUBLIC SYNONYM权限才可以创建**）**

|  |
| --- |
| *--创建同义词*  create public synonym syn\_emp for scott.emp  *--访问同义词*  select \* from syn\_emp |

* + **删除**

|  |
| --- |
| drop public synonym syn\_emp |

* **私有同义词**
  + **语法**

CREATE [OR REPLACE] SYNONYM sys\_name FOR [SCHEMA.] object\_name

* + **创建**

|  |
| --- |
| *--创建同义词*  create synonym syn\_pri\_emp for emp  *--访问同义词*  select \* from syn\_ pri \_emp |

* + **删除**

|  |
| --- |
| drop public synonym syn\_emp |

1. **表分区**

当表中的数据量不断增大，查询数据的速度就会变慢，应用程序的性能就会下降，这时就应该考虑对表进行分区。表进行分区后，逻辑上表仍然是一张完整的表，只是将表中的数据在物理上存放到多个表空间(物理文件上)，这样查询数据时，不至于每次都扫描整张表。

* **优点：**
  + 改善查询性能：对分区对象的查询可以仅搜索自己关心的分区，提高检索速度。
  + 增强可用性：如果表的某个分区出现故障，表在其他分区的数据仍然可用；
  + 维护方便：如果表的某个分区出现故障，需要修复数据，只修复该分区即可；
  + 均衡I/O：可以把不同的分区映射到磁盘以平衡I/O，改善整个系统性能。
* **使用场合**
  + 表的大小超过2GB
  + 表中包含历史数据，新的数据被增加都新的分区中
* **常见分区方法:**
* 范围 --- 8
* Hash --- 8i
* 列表 --- 9i
* 组合 --- 8i
  1. **范围分区**

范围分区将数据基于范围映射到每一个分区，这个范围是你在创建分区时指定的分区键决定的。这种分区方式是最为常用的，并且分区键经常采用日期。

* **特点:**
  + 最早、最经典的分区算法
  + Range分区通过对分区字段值的范围进行分区
  + Range分区特别适合于按时间周期进行数据的存储。日、周、月、年等。
  + 数据管理能力强(数据迁移、数据备份、数据交换)
  + 范围分区的数据可能不均匀
  + 范围分区与记录值相关，实施难度和可维护性相对较差
* **例子**
  + **按值划分**

|  |
| --- |
| *--创建*  CREATE TABLE book (  bookid NUMBER(5),  bookname VARCHAR2(30),  price NUMBER(8)  )PARTITION BY RANGE (price)*--分区字段*  (  PARTITION P1 VALUES LESS THAN (4) TABLESPACE system,  PARTITION P2 VALUES LESS THAN (8) TABLESPACE system,  PARTITION P3 VALUES LESS THAN (maxvalue) TABLESPACE system,  )  *--MAXVALUE代表了一个不确定的值,这个值高于其它分区中的任何分区键的值* |

* + **按日期划分**

|  |
| --- |
| CREATE TABLE student (  stuno NUMBER(5),  stuname VARCHAR2(30),  birthday date  )PARTITION BY RANGE (birthday)*--分区字段*  (  PARTITION P1990 VALUES LESS THAN (to\_date('1990-01-01','yyyy-mm-dd')) TABLESPACE system,  PARTITION P1991 VALUES LESS THAN (to\_date('1991-01-01','yyyy-mm-dd')) TABLESPACE system  ); |

* 1. **Hash分区(散列分区)**

这类分区是在列值上使用散列算法，以确定将行放入哪个分区中。当列的值没有合适的条件时，建议使用散列分区。散列分区为通过指定分区编号来均匀分布数据的一种分区类型。如果你要使用hash分区，只需指定分区的数量即可。建议分区的数量采用2的n次方，这样可以使得各个分区间数据分布更加均匀。

* + - **特点**
  + 基于分区字段的HASH值，自动将记录插入到指定分区。
  + 分区数一般是2的幂
  + 易于实施
  + 总体性能最佳
  + 适合于静态数据
  + HASH分区适合于数据的均匀存储
  + 数据管理能力弱
  + HASH分区对数据值无法控制
    - **例子**

|  |
| --- |
| CREATE TABLE classes (  clsno NUMBER(5),  clsname VARCHAR2(30)  )PARTITION BY HASH(clsno)*--分区字段*  (  PARTITION ph1 tablespace system,  PARTITION ph2 tablespace system  ) |

* 1. **List分区(列表分区)**

该分区的特点是某列的值只有几个，基于这样的特点我们可以采用列表分区。

* + - **特点**
  + List分区通过对分区字段的离散值进行分区
  + List分区是不排序的，而且分区之间也没有关联
  + List分区适合于对数据离散值进行控制
  + List分区只支持单个字段
  + List分区具有与range分区相似的优缺点
  + 数据管理能力强
  + 各分区的数据可能不均匀
    - **例子**

|  |
| --- |
| CREATE TABLE users (  userid NUMBER(5),  username VARCHAR2(30),  province char(5)  )PARTITION BY list(province)*--分区字段*  (  PARTITION pl1 values('广东') tablespace system,  PARTITION pl2 values('江西') tablespace system,  PARTITION pl3 values('广西') tablespace system,  PARTITION pl4 values('湖南') tablespace system  ); |

* 1. **组合分区**

常见的组合分区主要有范围散列分区和范围列表分区

* + - **特点**
  + 既适合于历史数据，又适合于数据均匀分布
  + 与范围分区一样提供高可用性和管理性
  + 实现粒度更细的操作
    - **组合范围列表分区**

这种分区是基于范围分区和列表分区，表首先按某列进行范围分区，然后再按某列进行列表分区，分区之中的分区被称为子分区。

* + - * **例子**

|  |
| --- |
| CREATE TABLE student (  stuno NUMBER(5),  stuname VARCHAR2(30),  birthday date,  province char(5)  )PARTITION BY RANGE (birthday) *--主分区字段*  subpartition BY LIST(province)*--子分区字符*  (  PARTITION P1990 VALUES LESS THAN(to\_date('1990-01-01','yyyy-mm-dd')) TABLESPACE system  (  SUBPARTITION pl1 values('广东') tablespace system,  SUBPARTITION pl2 values('江西') tablespace system,  SUBPARTITION pl3 values('广西') tablespace system,  SUBPARTITION pl4 values('湖南') tablespace system  ),  PARTITION P1991 VALUES LESS THAN(to\_date('1991-01-01','yyyy-mm-dd')) TABLESPACE system  (  SUBPARTITION p21 values('广东') tablespace system,  SUBPARTITION p22 values('江西') tablespace system,  SUBPARTITION p23 values('广西') tablespace system,  SUBPARTITION p24 values('湖南') tablespace system  )  ); |

* + - **组合范围散列分区**

这种分区是基于范围分区和散列分区，表首先按某列进行范围分区，然后再按某列进行散列分区。

* + - * **例子**

|  |
| --- |
| CREATE TABLE student (  stuno NUMBER(5),  stuname VARCHAR2(30),  birthday date  )PARTITION BY RANGE(birthday) *--主分区字段*  SUBPARTITION BY HASH(stuno)*--子分区字符*  (  PARTITION P1990 VALUES LESS THAN(to\_date('1990-01-01','yyyy-mm-dd')) TABLESPACE system  (  SUBPARTITION ph12 tablespace system,  SUBPARTITION ph13 tablespace system  ),  PARTITION P1991 VALUES LESS THAN(to\_date('1991-01-01','yyyy-mm-dd')) TABLESPACE system  (  SUBPARTITION ph21 tablespace system,  SUBPARTITION ph22 tablespace system  )  ); |

* 1. **表分区常用操作**
     + **添加分区**

|  |
| --- |
| *--添加主分区*  alter table book add partition p4 values less than(maxvalue) tablespace system  *--添加子分区*  ALTER TABLE student MODIFY PARTITION P1990  ADD SUBPARTITION pl5 values('福建') |

* + - **删除分区**

|  |
| --- |
| *--删除主分区*  ALTER TABLE student DROP PARTITION P1990  *--删除子分区*  ALTER TABLE student DROP SUBPARTITION p15 |

* + - **重命名表分区**

|  |
| --- |
| ALTER TABLE student RENAME PARTITION P21 TO P2 |

* + - **显示数据库所有分区表的信息**

|  |
| --- |
| select \* from DBA\_PART\_TABLES |

* + - **显示当前用户所有分区表的信息**

|  |
| --- |
| select \* from USER\_PART\_TABLES |

* + - **查询指定表分区数据**

|  |
| --- |
| select \* from users partition(pl2)*--主分区*  select \* from users subpartition(phl2)*--子分区* |

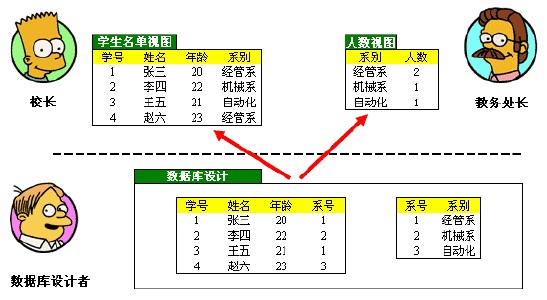
* + - **删除分区表一个分区的数据**

|  |
| --- |
| alter table book truncate partition p11 |

**15257956864 vvee4051**

1. **视图**
2. **概述**

视图是基于一个表或多个表或视图的逻辑表，本身不包含数据，通过它可以对表里面的数据进行查询和修改。视图基于的表称为基表。视图是存储在数据字典里的一条select语句。 通过创建视图可以提取数据的逻辑上的集合或组合。



* 为什么使用视图
  + 控制数据访问
  + 简化查询
  + 数据独立性
  + 避免重复访问相同的数据
  + 使用修改基表的最大好处是安全性，即保证那些能被任意人修改的列的安全性
* Oracle中视图分类
  + 关系视图
  + 内嵌视图
  + 对象视图
  + 物化视图

1. **关系视图**

关系视图是作为数据库对象存在的，创建之后也可以通过工具或数据字典来查看视图的相关信息。关系视图是4种视图中最简单，同时也最常用的视图。

* 语法

|  |
| --- |
| CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view\_name [(alias[, alias]...)]  AS subquery  [WITH CHECK OPTION [CONSTRAINT constraint]]  [WITH READ ONLY] |

* 1. OR REPLACE:若所创建的试图已经存在，ORACLE自动重建该视图
  2. FORCE:不管基表是否存在ORACLE都会自动创建该视图
  3. NOFORCE:只有基表都存在ORACLE才会创建该视图
  4. Alias:为视图产生的列定义的别名
  5. subquery:一条完整的SELECT语句，可以在该语句中定义别名
  6. WITH CHECK OPTION:插入或修改的数据行必须满足视图定义的约束
  7. WITH READ ONLY:该视图上不能进行任何DML操作
* 例子

|  |
| --- |
| create or replace view view\_Account\_dept  as  select \* from emp where deptno=10  *--只读视图*  create or replace view view\_Account\_dept  as  select \* from emp where deptno=10 order by sal  with read only  *--约束视图*  create or replace view view\_Account\_dept  as  select \* from emp where deptno=10  with check option |

* 查询视图

|  |
| --- |
| select \* from emp where view\_Account\_dept |

* 修改视图

通过OR REPLACE 重新创建同名视图即可

* 删除视图

DROP VIEW VIEW\_NAME语句删除视图

* 视图上的DML 操作原则
  1. 简单视图可以执行DML操作；
  2. 在视图包含GROUP函数，GROUP BY子句，DISTINCT关键字时不能执行delete语句
  3. 在视图包含GROUP函数，GROUP BY子句，DISTINCT关键字，ROWNUM为例，列定义为表达式时不能执行update语句
  4. 在视图包含GROUP函数，GROUP BY子句，DISTINCT关键字，ROWNUM为例，列定义为表达式，表中非空的列子视图定义中未包括时不能执行insert语句
  5. 可以使用WITH READ ONLY来屏蔽DML操作

1. **内嵌视图**

内嵌视图是在from语句中的可以把表改成一个子查询。内嵌视图不属于任何用户，也不是对象，内嵌视图是子查询的一种。

* 例子

|  |
| --- |
| Select \* from  (select \* from emp where deptno=10)  where sal>2000 |

1. **对象视图**

对象类型在数据库编程中有许多好处，但有时，应用程序已经开发完成。为了迎合对象类型而重建数据表是不现实的。对象视图正是解决这一问题的优秀策略。

1. **物化视图**

常用于数据库的容灾，不是传统意义上虚拟视图，是实体化视图，和表一样可以存储数据、查询数据。主备数据库数据同步通过物化视图实现，主备数据库通过data link连接，在主备数据库物化视图进行数据复制。当主数据库垮掉时，备数据库接管，实现容灾。

* 语法

|  |
| --- |
| create materialized view materialized\_view\_name  build [immediate|deferred] *--1.创建方式*  refresh [complete|fast|force|never] *--2.物化视图刷新方式*  on [commit|demand] *--3.刷新触发方式*  start with (start\_date) *--4.开始时间*  next (interval\_date) *--5.间隔时间*  with [primary key|rowid] *--默认 primary key*  ENABLE QUERY REWRITE *--7.是否启用查询重写*  as *--8.关键字*  select statement; *--9.基表选取数据的select语句* |

1. 创建方式
   * + - immediate(默认):立即
       - deferred:延迟，至第一次refresh时，才生效
2. 物化视图刷新方式
   * + - force(默认):如果可以快速刷新，就执行快速刷新,否则，执行完全刷新
       - complete:完全刷新，即刷新时更新全部数据，包括视图中已经生成的原有数据
       - fast:快速刷新，只刷新增量部分。前提是，需要在基表上创建物化视图日志。该日志记录基表数据变化情况，所以才能实现增量刷新
       - never:从不刷新
3. 刷新触发方式
   * + - on commit:基表有commit动作时，刷新视图，不能跨库执行(因为不知道别的库的提交动作)
       - on demand，在需要时刷新，根据后面设定的起始时间和时间间隔进行刷新，或者手动调用dbms\_mview包中的过程刷新时再执行刷新。
4. 开始时间和间隔时间
   * + - 4和5即开始刷新时间和下次刷新的时间间隔。如：start with sysdate next sysdate+1/1440表示马上开始，刷新间隔为1分钟。（与 on commit选项冲突）
5. 创建模式
   * + - primary key(默认):基于基表的主键创建
       - rowed:不能对基表执行分组函数、多表连结等需要把多个rowid合成一行的操作
6. 是否启用查询重写
   * + - 如果设置了初始化参数query\_rewrite\_enabled=true则默认就会启用查询重写。但是，数据库默认该参数为false。并且，不是什么时候都应该启用查询重写。所以，该参数应该设置为false，而在创建特定物化视图时，根据需要开启该功能。
7. 注意
   * + - 如果选择使用了上面第4,5选项，则不支持查询重写功能（原因很简单，所谓重写，就是将对基表的查询定位到了物化视图上，而4、5选项会造成物化视图上部分数据延迟，所以，不能重写）。

* 例子

|  |
| --- |
| *--创建增量刷新的物化视图时应先创建存储的日志空间*  *--在scott.emp表中创建物化视图日志*  create materialized view log on emp  tablespace users with rowid;  *--开始创建物化视图*  *--方式一*  create materialized view mv\_emp  tablespace users *--指定表空间*  build immediate *--创建视图时即生成数据*  refresh fast *--基于增量刷新*  on commit *--数据DML操作提交就刷新*  with rowid *--基于ROWID刷新*  as select \* from emp  *--方式二*  create materialized view mv\_emp2  tablespace users *--指定表空间*  refresh fast *--基于增量刷新*  start with sysdate *--创建视图时即生成数据*  next sysdate+1/1440 */\*每隔一分钟刷新一次\*/*  with rowid *--基于ROWID刷新*  as select \* from emp  *--删除物化视图日志*  drop materialized view mv\_emp |

1. **索引**
2. **概述**

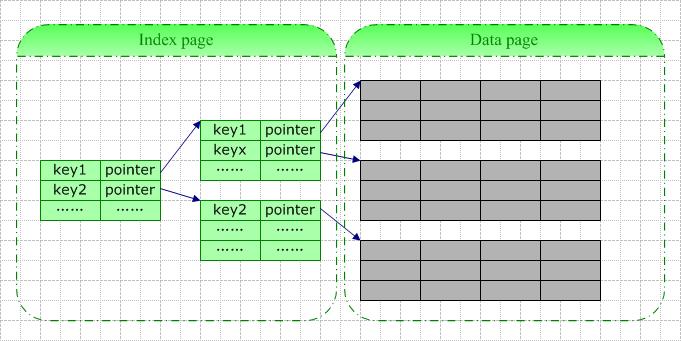
索引是建立在表上的可选对象，设计索引的目的是为了提高查询的速度。但同时索引也会增加系统的负担，进行影响系统的性能。

索引一旦建立后，当在表上进行DML操作时，Oracle会自动维护索引，并决定何时使用索引。

索引的使用对用户是透明的，用户不需要在执行SQL语句时指定使用哪个索引及如何使用索引，也就是说，无论表上是否创建有索引，SQL语句的用法不变。用户在进行操作时，不需要考虑索引的存在，索引只与系统性能相关。

* 索引的原理

当在一个没有创建索引的表中查询符合某个条件的记录时，DBMS会顺序地逐条读取每个记录与查询条件进行匹配，这种方式称为全表扫描。全表扫描方式需要遍历整个表，效率很低。



* 索引的类型

Oracle支持多种类型的索引，可以按列的多少、索引值是否唯一和索引数据的组织形式对索引进行分类，以满足各种表和查询条件的要求。

* + 单列索引和复合索引
  + B树索引
  + 位图索引
  + 函数索引
* 创建索引

|  |
| --- |
| CREATE [UNIQUE] | [BITMAP] INDEX index\_name  ON table\_name([column1 [ASC|DESC],column2  [ASC|DESC],…] | [express])  [TABLESPACE tablespace\_name]  [PCTFREE n1]  [STORAGE (INITIAL n2)]  [NOLOGGING]  [NOLINE]  [NOSORT] |

* UNIQUE：表示唯一索引，默认情况下，不使用该选项。
* BITMAP：表示创建位图索引，默认情况下，不使用该选项。
* PCTFREE：指定索引在数据块中的空闲空间。对于经常插入数据的表，应该为表中索引指定一个较大的空闲空间。
* NOLOGGING：表示在创建索引的过程中不产生任何重做日志信息。默认情况下，不使用该选项。
* ONLINE：表示在创建或重建索引时，允许对表进行DML操作。默认情况下，不使用该选项。
* NOSORT：默认情况下，不使用该选项。则Oracle在创建索引时对表中记录进行排序。如果表中数据已经是按该索引顺序排列的，则可以使用该选项。

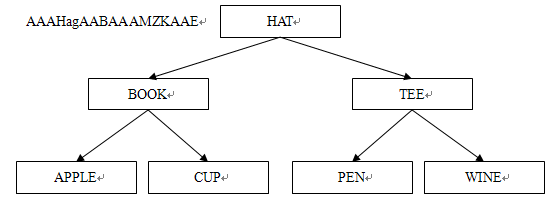
1. **单列索引和复合索引**

一个索引可以由一个或多个列组成。基于单个列所创建的索引称为单列索引，基于两列或多列所创建的索引称为多列索引。

1. **B树索引**

B树索引是Oracle数据库中最常用的一种索引。当使用CREATE INDEX语句创建索引时，默认创建的索引就是B树索引。B树索引就是一棵二叉树，它由根、分支节点和叶子节点三部分构成。叶子节点包含索引列和指向表中每个匹配行的ROWID值。叶子节点是一个双向链表，因此可以对其进行任何方面的范围扫描。

B树索引中所有叶子节点都具有相同的深度，所以不管查询条件如何，查询速度基本相同。另外，B树索引能够适应各种查询条件，包括精确查询、模糊查询和比较查询。



* 例子

|  |
| --- |
| *--创建B树索引，属于单列索引*  create index idx\_emp\_job on emp(job)  *--创建B树索引，属于复合索引*  create index idx\_emp\_nameorsal on emp(ename,sal)  *--创建唯一的B树索引，属于单列索引*  create unique index idx\_emp\_ename on emp(ename)  *--删除索引*  drop index idx\_emp\_job5  drop index idx\_emp\_nameorsal  drop index idx\_emp\_ename  *--如果表已存在大量的数据，需要规划索引段*  create index idx\_emp\_nameorsal on emp(ename,sal)  pctfree 30 tablespace system |

1. **位图索引**

在B树索引中，保存的是经排序过的索引列及其对应的ROWID值。但是对于一些基数很小的列来说，这样做并不能显著提高查询的速度。所谓基数，是指某个列可能拥有的不重复值的个数。比如性别列的基数为2（只有男和女）。

因此，对于象性别、婚姻状况、政治面貌等只具有几个固定值的字段而言，如果要建立索引，应该建立位图索引，而不是默认的B树索引。

* 例子

|  |
| --- |
| *--创建位图索引,单列索引*  create bitmap index idx\_bm\_job on emp(job)  *--创建位图索引,复合索引*  create bitmap index idx\_bm\_jobordeptno on emp(job,deptno)  *--删除位图索引*  drop index idx\_bm\_job  drop index idx\_bm\_jobordeptno |

1. **函数索引**

函数索引既可以使用B树索引，也可以使用位图索引，可以根据函数或表达式的结果的基数大小来进行选择，当函数或表达式的结果不确定时采用B树索引，当函数或表达式的结果是固定的几个值时采用位图索引。

* 例子

|  |
| --- |
| *--合并索引*  alter index idx\_emp\_ename COALESCE |

1. **并和重建索引**

表在使用一段时间后，由于用户不断对其进行更新操作，而每次对表的更新必然伴随着索引的改变，因此，在索引中会产生大量的碎片，从而降低索引的使用效率。有两种方法可以清理碎片：合并索引和重建索引。

* 合并索引就是将B树叶子节点中的存储碎片合并在一起，从而提高存取效率，但这种合并并不会改变索引的物理组织结构。

|  |
| --- |
| *--创建B树类型的函数索引*  create index idx\_fun\_emp\_hiredate on emp(to\_char(hiredate,'yyyy-mm-dd'))  *--创建位图类型的函数索引*  create index idx\_fun\_emp\_job on emp(upper(job)) |

* 重建索引相当于删除原来的索引，然后再创建一个新的索引，因此，CREAT INDEX语句中的选项同样适用于重建索引。如果在索引列上频繁进行UPDATE和DELETE操作，为了提高空间的利用率，应该定期重建索引。

1. **管理索引的原则**

使用索引的目的是为了提高系统的效率，但同时它也会增加系统的负担，进行影响系统的性能，因为系统必须在进行DML操作后维护索引数据。

在新的SQL标准中并不推荐使用索引，而是建议在创建表的时候用主键替代。因此，为了防止使用索引后反而降低系统的性能，应该遵循一些基本的原则:

* 1. 小表不需要建立索引。
  2. 对于大表而言，如果经常查询的记录数目少于表中总记录数目的15%时，可以创建索引。这个比例并不绝对，它与全表扫描速度成反比。
  3. 对于大部分列值不重复的列可建立索引。
  4. 对于基数大的列，适合建立B树索引，而对于基数小的列适合建立位图索引。
  5. 对于列中有许多空值，但经常查询所有的非空值记录的列，应该建立索引。
  6. LONG和LONG RAW列不能创建索引。
  7. 经常进行连接查询的列上应该创建索引。
  8. 在使用CREATE INDEX语句创建查询时，将最常查询的列放在其他列前面。
  9. 维护索引需要开销，特别时对表进行插入和删除操作时，因此要限制表中索引的数量。对于主要用于读的表，则索引多就有好处，但是，一个表如果经常被更改，则索引应少点。
  10. 在表中插入数据后创建索引。如果在装载数据之前创建了索引，那么当插入每行时，Oracle都必须更改每个索引。

1. **ROWID和ROWNUM**
   1. ROWID

rowid是一个伪列，是用来确保表中行的唯一性，它并不能指示出行的物理位置，但可以用来定位行。rowid是存储在索引中的一组既定的值（当行确定后）。我们可以像表中普通的列一样将它选出来, 利用rowid是访问表中一行的最快方式。rowid的是基于64位编码的18个字符显示（数据对象编号(6)+文件编号(3) +块编号(6)+行编号(3)=18位）

|  |
| --- |
| select rowid from emp |

* ROWID的使用

|  |
| --- |
| *--快速删除重复的记录*  delete from temp t where rowid not in(  select max(rowid) from temp  where t.id=id and t.name=name and t.sal = sal  ) |

* 1. ROWNUM

ROWNUM是一个序列，是oracle数据库从数据文件或缓冲区中读取数据的顺序。它取得第一条记录则rownum值为1，第二条为2，依次类推。

|  |
| --- |
| select rownum,emp.\* from emp |

* ROWID的使用

|  |
| --- |
| *--取前3条记录*  select \* from emp where rownum<=3*--方式一*  select \* from emp where rownum!=4*--方式二*  *--分页*  select \* from emp where empno not in(  select empno from emp where rownum<5*--方式一*  ) and rownum <4 |

1. **PL/SQL编程**
2. **介绍**

PL/SQL是oracle在标准sql语言上的扩展，PL/SQL不仅允许嵌入sql语言，还可以定义变量和常量，允许使用例外处理各种错误，这样使它的功能变得更加强大。

PL/SQL也是一种语言，叫做过程化sql语言(procedural language/sql),通过此语言可以实现复杂功能或者复杂的计算。

* 优点
  1. 提高应用程序的运行性能
  2. 模块化的设计思想
  3. 减少网络传输量
  4. 提高安全性
* 缺点
  1. 可移植性差
  2. 违反MVC设计模式
  3. 无法进行面向对象编程
  4. 无法做成通用的业务逻辑框架
  5. 代码可读性差，相当难维护
* 分类

编写PL/SQL程序实际上就是编写PL/SQL块，要完成相对简单的应用功能，可能需要编写一个PL/SQL块，如果想要实现复杂的能够，可能需要在一个PL/SQL中嵌入其他的PL/SQL块。

1. **PL/SQL基础**
   1. 编写规范
      1. 注释

--单行注释

/\*块注释\*/

* + 1. 标识符的命名规范
       - 定义变量：建议用v\_作为前缀v\_price
       - 定义常量：建议用c\_作为前缀c\_pi
       - 定义游标：建议用\_cursor作为后缀emp\_cursor
       - 定义例外：建议用e\_作为前缀e\_error
  1. **块结构**

PL/SQL块由三个部分组成：定义部分、执行部分、例外处理部分

|  |
| --- |
| Declare  */\**  *定义部分(可选)：定义常量、变量、游标、例外，复杂数据类型*  *\*/*  begin  */\**  *执行部分(必须):要执行的PL/SQL语句和SQL语句*  *\*/*  exception  */\*例外部分(可选):处理运行各种错误\*/*  end |

1. ：只定义执行部分

|  |
| --- |
| begin  */\**  *dbms\_output是oracle提供的包(类似java开发包)*  *该包包含一些过程，put\_line就是其一个过程*  *\*/*  dbms\_output.put\_line('HELLO WORLD'); *--控制台输出*  end; |

1. ：定义声明部分和执行部分

|  |
| --- |
| declare  *--声明变量*  v\_name varchar2(20);  v\_sal number(7,2);  begin  *--执行查询*  select ename,sal into v\_name,v\_sal  from emp where rownum=1;  *--控制台输出*  dbms\_output.put\_line('用户名：' || v\_name);  dbms\_output.put\_line('工资：' || v\_sal);  end; |

1. ：定义声明部分、执行部分和例外部分

|  |
| --- |
| declare  *--声明变量*  v\_name varchar2(20);  v\_sal number(7,2);  begin  *--执行查询,条件中的&表示从控制接受数据*  select ename,sal into v\_name,v\_sal  from emp where empno=&no;  *--控制台输出*  dbms\_output.put\_line('用户名：' || v\_name);  dbms\_output.put\_line('工资：' || v\_sal);  exception  *--例外处理(no\_data\_found)*  when no\_data\_found then  dbms\_output.put\_line('执行查询没有结果');  end; |

* 1. 预定义例外
     1. case\_not\_found预定义例外

在开发pl/sql块中编写case语句时，如果在when子句中没有包含必须的条件分支，就会触发case\_not\_found例外。

* + 1. cursor\_already\_open预定义例外

当重新打开已经打开的游标时，会隐含的触发cursor\_already\_open例外。

* + 1. dup\_val\_on\_index预定义例外

在唯一索引所对应的列上插入重复的值时，会隐含的触发例外

* + 1. invalid\_cursorn预定义例外

当试图在不合法的游标上执行操作时，会触发该例外

* + 1. invalid\_number预定义例外

当输入的数据有误时，会触发该例外

* + 1. no\_data\_found预定义例外

当执行select into没有返回行，就会触发该例外

* + 1. too\_many\_rows预定义例外

当执行select into语句时，如果返回超过了一行，则会触发该例外

* + 1. zero\_divide预定义例外

当执行2/0语句时，则会触发该例外

* + 1. value\_error预定义例外

当在执行赋值操作时，如果变量的长度不足以容纳实际数据，则会触发该例外value\_error

* + 1. others
  1. 变量类型分类

在编写PL/SQL时，可以定义变量和常量，常用的类型主要有：

* 标量类型(scalar)
* 复合类型(composite)
* 参照类型(reference)
* lob(large object)
  1. 标量类型:常用类型

|  |
| --- |
| declare  *--定义一个变长字符串*  v\_name varchar2(20);  *--定义小数,并赋值*  v\_sal number(7,2) :=9.8;  *--定义整数*  v\_num number(4);  *--定义日期*  v\_birthday date;  *--定义布尔类型，不能为空，初始值为false*  v\_flg boolean not null default false;  *--使用%type类型*  v\_job emp.job%type;  begin  v\_flg := true;  v\_birthday :=sysdate;  dbms\_output.put\_line('当前时间:' || v\_birthday);  end; |

* 1. 复合类型：可以存放多个值。主要包括PL/SQL记录、PL/SQL表、嵌入表和varray这四种类型

**记录类型：**类似于c中的结构体

|  |
| --- |
| declare  *--定义记录类型*  type emp\_record\_type is record(  empno emp.empno%type,  ename emp.ename%type,  sal emp.sal%type  );  *--定义变量引用记录类型*  v\_record emp\_record\_type;  begin  *--使用记录类型*  select empno,ename,sal into v\_record from emp where rownum=1;  *--控制台输出*  dbms\_output.put\_line('雇员编号：' || v\_record.empno);  dbms\_output.put\_line('雇员姓名：' || v\_record.ename);  dbms\_output.put\_line('雇员工资：' || v\_record.sal);  end; |

**表类型:**类似于java语言中的数组

|  |
| --- |
| declare  *--声明表类型*  type emp\_table\_type is table of varchar2(20)  index by PLS\_INTEGER;*--表示表按整数来排序*  v\_enames emp\_table\_type;*--定义变量引用表类型*  begin  select ename into v\_enames(0) from emp where rownum=1;  select ename into v\_enames(1) from emp where empno=7499;  select ename into v\_enames(2) from emp where empno=7698;  *--输出*  dbms\_output.put\_line('下标0：' || v\_enames(0));  dbms\_output.put\_line('下标1：' || v\_enames(1));  dbms\_output.put\_line('下标2：' || v\_enames(2));  end; |

**varray类型:**可变长数组

|  |
| --- |
| declare  *--定义varray类型*  type varray\_list is varray(20) of number(4);  *--定义变量引用varray类型*  v\_list varray\_list:=varray\_list(7369,7499,7566);  begin  *--for i in v\_list.first..v\_list.last*  for i in 1..v\_list.count  loop  dbms\_output.put\_line(v\_list(i));  end loop;  end; |

**PL/SQL集合方法**

* + 1. exists():用于确定特定集合元素是否存在
    2. count：用于返回集合变量的元素总个数
    3. limit：用于返回varray变量所允许的最大元素个数
    4. first：用于返回集合变量中的一个元素的下标
    5. last：用于返回集合变量中最后一个元素的下标
    6. prior()：返回当前元素前一个元素的下标
    7. next()：返回当前元素后一个元素的下标
    8. extend：为集合变量添加元素，此方法适合用于嵌套表和varray
    9. trim：从集合变量尾部删除元素，此方法适用于嵌套表和varray
    10. delete：从集合变量中删除特定的元素，此方法适用于嵌套表和index-by表
  1. 参照类型:类似c语言中的指针，oracle的游标

1. **PL/SQL控制语句**
2. **条件分支语句**
   * 1. **if—then**

|  |
| --- |
| declare  *--声明变量*  v\_empno emp.empno%type;  v\_sal emp.sal%type;  begin  *--根据雇员编号查询工资*  select empno,sal into v\_empno,v\_sal from emp where empno=&no;  *--如果工资小于2000就加100*  if v\_sal<2000  then  *--工资加100*  update emp set sal = sal+100 where empno=v\_empno;  *--提交*  commit;  end if;  end; |

* + 1. **if—then—else**

|  |
| --- |
| declare  *--声明变量*  v\_loginname varchar2(10);  v\_password varchar2(10);  begin  *--从控制台接收数据*  v\_loginname := '&ln';  v\_password := '&pw';  if v\_loginname = 'admin' and v\_password = '123456'  then  dbms\_output.put\_line('用户登录成功！');  else  dbms\_output.put\_line('用户登录失败！');  end if;  end; |

* + 1. **if—then—elsif—else**

|  |
| --- |
| declare  *--声明变量*  v\_empno emp.empno%type;  v\_job emp.job%type;  begin  *--根据雇员编号查询职位*  select empno,job into v\_empno,v\_job from emp where empno=&no;  */\*如果雇员所属职位是manager工资加1000*  *职位是salesman工资加500*  *其他职位加200*  *\*/*  if v\_job = 'MANAGER' then  *--MANAGER职位工资加1000*  update emp set sal = sal+1000 where empno=v\_empno;  elsif v\_job = 'SALESMAN' then  *--SALESMAN职位工资加500*  update emp set sal = sal+500 where empno=v\_empno;  else  *--其他职位工资加200*  update emp set sal = sal+200 where empno=v\_empno;  end if;  *--提交*  commit;  end; |

* + 1. **case**

|  |
| --- |
| declare  *--声明变量*  v\_mark number(4);  v\_outstr varchar2(40);  begin  *--从控制台接收成绩*  v\_mark := &m;  case  when v\_mark<=100 and v\_mark>=90 then  v\_outstr := '优秀';  when v\_mark<90 and v\_mark>=80 then  v\_outstr := '良好';  when v\_mark<80 and v\_mark>=70 then  v\_outstr := '中等';  when v\_mark<70 and v\_mark>=60 then  v\_outstr := '及格';  when v\_mark<60 and v\_mark>=0 then  v\_outstr := '不及格';  else  v\_outstr := '成绩输入有误';  end case;  *--控制台输出*  dbms\_output.put\_line(v\_outstr);  end; |

1. **循环语句**
2. **loop**

LOOP 要执行的语句;

EXIT WHEN <条件语句> /\*条件满足，退出循环语句\*/

END LOOP;

其中：EXIT WHEN 子句是必须的，否则循环将无法停止。

|  |
| --- |
| declare  v\_num number(4):=1;  begin  *--从控制台接收数据并插入到account表中*  loop  insert into account values(v\_num,'&name');  exit when v\_num =10;  v\_num :=v\_num+1;  end loop;  end; |

1. **while**

WHILE <布尔表达式> LOOP要执行的语句;END LOOP;

其中：

* 循环语句执行的顺序是先判断<布尔表达式>的真假，如果为真则循环执行，否则退出循环
* 在WHILE循环语句中仍然可以使用EXIT或EXIT WHEN子句

|  |
| --- |
| declare  v\_num number(4):=1;  begin  *--从控制台接收数据并插入到account表中*  while v\_num<11  loop  insert into account values(v\_num,'&name');  v\_num :=v\_num+1;  end loop;  end; |

1. **for**

FOR 循环计数器IN [ REVERSE ] 下限.. 上限LOOP要执行的语句;END LOOP;

其中：

* 每循环一次，循环变量自动加1；使用关键字REVERSE，循环变量自动减1
* 跟在IN REVERSE 后面的数字必须是从小到大的顺序，但不一定是整数，可以是能够转换成整数的变量或表达式
* 可以使用EXIT WHEN子句退出循环

|  |
| --- |
| *--输出1到10的数*  begin  for i in 1..10  loop  *--控制台输出*  dbms\_output.put\_line(i);  end loop;  end;  *--输出一组数据*  declare  *--声明表类型*  type emp\_table\_type is table of varchar2(20)  index by PLS\_INTEGER;*--表示表按整数来排序*  v\_enames emp\_table\_type;*--定义变量引用表类型*  begin  *--所有值赋予v\_enames*  select ename bulk collect into v\_enames from emp;  *--输出*  for i in 1..v\_enames.count  loop  dbms\_output.put\_line(v\_enames(i));  end loop;  end; |

1. **动态SQL**

* **概述**

当编写PL/SQL块时，静态SQL语句只能完成一些固定任务，为了使得PL/SQL块可以灵活处理SQL语句需要使用动态SQL。比如要在PL/SQL块中嵌入DDL语句和DCL语句时，必须使用动态SQL，另外，如果在PL/SQL块中需要执行更加灵活的select语句和DML语句，也需要使用动态SQL。

动态SQL包括本地动态SQL和DBMS\_SQL动态SQL两种实现方法。其中本地动态SQL只适用于oracle服务器端，而DBMS\_SQL动态SQL不仅适用于oracle服务器端，而且可以在客户端应用程序中使用。

* **动态sql语法**

|  |
| --- |
| EXECUTE IMMEDIATE dyamic\_sql\_string  [INTO define\_variable\_list]  [USING bind\_argument\_list]; |

dyamic\_sql\_string ：是动态SQL语句字符串

define\_variable\_list：INTO子句用于接受SELECT语句选择的记录者

bind\_argument\_list：USING子句用于绑定输入参数变量

* **案例一**:执行DML

|  |
| --- |
| declare  *--定义变量*  v\_sql varchar2(100);  begin  *--赋值*  v\_sql :='create table account(aid number(4),aname varchar2(20))';  *--执行动态SQL*  execute immediate v\_sql;  end; |

* **案例二：**获取单条数据

|  |
| --- |
| declare  *--定义sql*  v\_sql varchar2(100);  v\_emp emp%rowtype;  v\_id emp.empno%type:=&no;  begin  *--执行动态SQL*  execute immediate 'select \* from emp where empno=:id'  into v\_emp  using v\_id;  *--输出*  dbms\_output.put\_line(v\_emp.ename || v\_emp.job);  end; |

* **案例三：**获取集合数据

|  |
| --- |
| declare  type v\_table\_list is table of emp.ename%type  index by binary\_integer;  v\_lsit v\_table\_list;  begin  *--执行动态SQL*  execute immediate 'select ename from emp where deptno=:no'  bulk collect into v\_lsit  using &dno;  *--输出*  for i in 1..v\_lsit.count  loop  dbms\_output.put\_line('雇员名:' || v\_lsit(i));  end loop;  end; |

* **在DML中使用RETURNING**:获取执行DML后的值

|  |
| --- |
| declare  v\_sal emp.sal%type;  begin  *--执行静态SQL*  update emp set sal =sal+200 where empno=&no  returning sal into v\_sal;  *--输出执行sql后的工资*  Dbms\_Output.put\_line('修改后的工资：'||v\_sal);  end; |

* **FORALL:**批量绑定

|  |
| --- |
| declare  type v\_table\_empno is table of emp.empno%type;  v\_sql varchar2(100);  v\_list v\_table\_empno :=v\_table\_empno(7499,7369,7566);  begin  v\_sql := 'update emp set sal=sal+100 where empno=:no';  forall i in 1..v\_list.count  execute immediate v\_sql using v\_list(i);  end; |

1. **游标**
2. **概述**

游标（cursor）是系统为用户开设的一个数据缓冲区，存放SQL语句的执行结果。每个游标区都有一个名字。用户可以用SQL语句逐一从游标中获取记录，并赋给主变量，交由主语言进一步处理。

* **游标作用**
  + 指定结果集中特定行的位置。
  + 基于当前的结果集位置检索一行或连续的几行。
  + 在结果集的当前位置修改行中的数据。
  + 对其他用户所做的数据更改定义不同的敏感性级别。
  + 可以已编程的方式访问数据库。
* **避免使用游标**
  + 在创建游标时，最需要考虑的事情是，“是否有办法避免使用游标？”，因为游标的效率较差，如果游标操作的数据超过1万行，那么就应该改写
  + 如果使用了游标，就要尽量避免在游标循环中再进行表连接的操作。
* **游标类型**
  + 静态游标：结果集已经确实的游标，分为隐式和显示游标。
  + 隐式游标
  + 显式游标
  + REF游标

1. **隐式游标**

PL/SQL为所有SQL数据操作语句（DML）隐式声明游标，用户不能直接命名和控制此类游标。当用户在PL/SQL中使用数据库操作语言（DML）时，Oracle预定义一个名为SQL的隐式游标，通过检查隐式游标的属性可以获取与最近执行的SQL语句相关的信息。常常用来判断插入、更新、删除是否成功，但要放在COMMIT语句之前

* 隐式游标的属性：
* %Found ：Fetch语句（获取记录）执行情况 True or False。
* %NotFound ：最后一条记录是否提取出 True or False。
* %ISOpen ： 游标是否打开True or False。
* %RowCount ：游标当前提取的行数 。
* 例子

|  |
| --- |
| begin  update emp set sal =sal+200 where empno=7499;  if SQL%found then  Dbms\_Output.put\_line('(found)存在记录');  else  Dbms\_Output.put\_line('((found))没有记录');  end if;  if SQL%notfound then  Dbms\_Output.put\_line('(notfound)没有记录');  else  Dbms\_Output.put\_line('((notfound))存在记录');  end if;  if SQL%rowcount >0 then  Dbms\_Output.put\_line('(rowcount)存在记录');  else  Dbms\_Output.put\_line('(rowcount)没有记录');  end if;  if SQL%isopen then  Dbms\_Output.put\_line('隐式游标已打开');  else  Dbms\_Output.put\_line('隐式游标已关闭');  end if;  end; |

1. **显式游标**

用户显示声明的游标，即指定结果集。当查询返回结果超过一行时，就需要一个显式游标。

* 使用显式游标，需要进行四个步骤：
  + 1. 声明游标：划分存储区域，注意此时并没有执行Select 语句。

|  |
| --- |
| CURSOR 游标名( 参数 列表) [返回值类型] IS Select 语句 |

* + 1. 打开游标：执行Select 语句，获得结果集存储到游标中，此时游标指向结果集头，而不是第一条记录。

|  |
| --- |
| Open 游标名( 参数 列表) |

* + 1. 获取记录：移动游标取一条记录

|  |
| --- |
| Fetch 游标名 InTo 临时记录或属性类型变量 |

* + 1. 关闭游标：将游标放入缓冲池中，没有完全释放资源。可重新打开。

|  |
| --- |
| Close 游标名 |

* for循环游标
  + 语法

|  |
| --- |
| For 变量名 In 游标名  Loop  数据处理语句;  End Loop; |

* + 例子

|  |
| --- |
| declare  *--定义游标*  cursor my\_cursor  is  select \* from emp where deptno=10;  *--定义对象变量*  v\_emp emp%rowtype;  begin  *--读取游标数据*  for v\_emp in my\_cursor  loop  Dbms\_Output.put\_line('雇员名：' || v\_emp.ename);  Dbms\_Output.put\_line('职位：' || v\_emp.job);  Dbms\_Output.put\_line('工资：' || v\_emp.sal);  Dbms\_Output.put\_line('奖金：' || v\_emp.comm);  Dbms\_Output.put\_line('===============================');  end loop;  end; |

* loop循环游标
  + 语法

|  |
| --- |
| Loop  Fatch 游标名 InTo 临时记录或属性类型变量;  Exit When 游标名%NotFound;  End Loop； |

* + 例子

|  |
| --- |
| declare  *--定义游标*  cursor my\_cursor  is  select \* from emp where deptno=10;  *--定义游标变量*  v\_emp my\_cursor%rowtype;  begin  *--打开游标*  open my\_cursor;  *--循环*  loop  *--读取一行数据*  fetch my\_cursor into v\_emp;  exit when my\_cursor%notfound;  *--显示数据*  Dbms\_Output.put\_line('雇员名：' || v\_emp.ename);  Dbms\_Output.put\_line('职位：' || v\_emp.job);  Dbms\_Output.put\_line('工资：' || v\_emp.sal);  Dbms\_Output.put\_line('奖金：' || v\_emp.comm);  Dbms\_Output.put\_line('===============================');  end loop;  *--关闭游标*  close my\_cursor;  end; |

* 更新
  + 语法

|  |
| --- |
| *--声明*  Cursor 游标名 IS SELECT 语句 For Update [Of 更新列列名];  *--更新*  Update 表名 SET 更新语句 Where Current Of 游标名; |

* + 例子

|  |
| --- |
| declare  *--定义游标*  cursor my\_cursor  is select deptno from emp for update;  *--定义变量*  v\_depono emp.deptno%type;  v\_sal emp.sal%type;  begin  *--打开游标*  open my\_cursor;  *--循环*  loop  *--读取一行数据*  fetch my\_cursor into v\_depono;  exit when my\_cursor%notfound;  *--加工资业务*  if v\_depono = 10 then  v\_sal := 200;  elsif v\_depono = 20 then  v\_sal := 400;  else  v\_sal := 600;  end if;  *--执行更新*  update emp set sal = sal +v\_sal where current of my\_cursor;  end loop;  *--关闭游标*  close my\_cursor;  end; |

* 删除
  + 语法

|  |
| --- |
| *--声明*  Cursor 游标名 IS SELECT 语句 For Delete [Of 更新列列名];  *--更新*  Delete From 表名 Where Current Of 游标名; |

* + 例子

|  |
| --- |
| declare  *--定义游标*  cursor my\_cursor  is select sal from emp for update;  *--定义变量*  v\_sal emp.sal%type;  begin  *--打开游标*  open my\_cursor;  *--循环*  loop  *--读取一行数据*  fetch my\_cursor into v\_sal;  exit when my\_cursor%notfound;  *--删除业务*  if v\_sal < 2000 then  *--执行删除*  delete from emp where current of my\_cursor;  end if;  end loop;  *--关闭游标*  close my\_cursor;  end; |

* 带参数游标
  + 语法

|  |
| --- |
| CURSOR 游标名 [(parameter[,parameter],...)] IS Select语句;  open 游标名(值) |

* + 例子

|  |
| --- |
| declare  *--定义游标*  cursor my\_cursor(v\_sal emp.sal%type)  is select \* from emp where sal>v\_sal;  *--定义变量*  v\_emp emp%rowtype;  begin  *--打开游标*  open my\_cursor(&sal);  *--循环*  loop  *--读取一行数据*  fetch my\_cursor into v\_emp;  exit when my\_cursor%notfound;  *--显示数据*  Dbms\_Output.put\_line('雇员名：' || v\_emp.ename);  Dbms\_Output.put\_line('职位：' || v\_emp.job);  Dbms\_Output.put\_line('工资：' || v\_emp.sal);  Dbms\_Output.put\_line('奖金：' || v\_emp.comm);  Dbms\_Output.put\_line('===============================');  end loop;  *--关闭游标*  close my\_cursor;  end; |

1. **REF游标**
   * **概述**

动态关联结果集的临时对象。即在运行的时候动态决定执行查询。REF游标又分为强类型REF游标和弱类型REF游标

* + **作用**

实现在程序间传递结果集的功能，利用REF CURSOR也可以实现BULK SQL，从而提高SQL性能

* + **静态游标和REF 游标的区别**
    1. 静态游标是静态定义，REF 游标是动态关联
    2. 使用REF 游标需REF 游标变量
    3. REF 游标能做为参数进行传递，而静态游标是不可能的
  + **使用REF游标步骤**
    1. 声明REF 游标类型，确定REF 游标类型
       - 强类型REF游标：指定retrun type，REF 游标变量的类型必须和return type一致。

|  |
| --- |
| Type REF游标名 IS Ref Cursor Return 结果集返回记录类型 |

* + - * 弱类型REF游标：不指定return type，能和任何类型的CURSOR变量匹配，用于获取任何结果集。

|  |
| --- |
| Type REF游标名 IS Ref Cursor |

* + 1. 声明Ref 游标类型变量

|  |
| --- |
| 变量名 已声明Ref游标类型 |

* + 1. 打开REF游标，关联结果集

|  |
| --- |
| Open Ref 游标类型变量 For 查询语句返回结果集 |

* + 1. 获取记录，操作记录

|  |
| --- |
| Fatch REF游标名 InTo 临时记录类型变量或属性类型变量列表 |

* + 1. 关闭游标，完全释放资源

|  |
| --- |
| Close 游标名 |

* + **强类型REF游标例子**

|  |
| --- |
| declare  *--定义强REF类型*  type refemp\_cursor is ref cursor return emp%rowtype;  *--定义游标变量*  v\_refemp refemp\_cursor;  *--定义变量*  v\_emp emp%rowtype;  begin  *--打开游标*  open v\_refemp for select \* from emp;  *--读取emp数据*  loop  fetch v\_refemp into v\_emp;  exit when v\_refemp%notfound;  *--显示数据*  dbms\_output.put\_line(v\_emp.ename);  end loop;  *--关闭游标*  close v\_refemp;  end; |

* + **弱类型REF游标例子**

|  |
| --- |
| declare  *--定义弱REF类型*  type refemp\_cursor is ref cursor;  *--定义游标变量*  v\_refemp refemp\_cursor;  *--定义变量*  v\_emp emp%rowtype;  v\_dept dept%rowtype;  begin  *--打开游标*  open v\_refemp for select \* from emp;  *--读取emp数据*  loop  fetch v\_refemp into v\_emp;  exit when v\_refemp%notfound;  *--显示数据*  dbms\_output.put\_line(v\_emp.ename);  end loop;  *--关闭游标*  close v\_refemp;    *--打开游标*  open v\_refemp for select \* from dept;  *--读取dept数据*  loop  fetch v\_refemp into v\_dept;  exit when v\_refemp%notfound;  *--显示数据*  dbms\_output.put\_line(v\_dept.dname);  end loop;  *--关闭游标*  close v\_refemp;  end; |

1. **子程序和程序包**
2. **子程序**
   * **概述：**命名的 PL/SQL 块，编译并存储在数据库中
   * **组成部分：**声明部分、可执行部分、异常处理部分(可选)
   * **分类：**过程、函数
   * **子程序的优点：**
     + **模块化：**将程序分解为逻辑模块
     + **可重用性：**可以被任意数目的程序调用
     + **可维护性：**简化维护操作
     + **安全性：**通过设置权限，使数据更安全
3. **存储过程**
   * **概述**
     + 过程是用于完成特定任务的子程序
     + 存储于数据库中并可由任意匿名块调用
     + 能够接受参数
     + 在参数声明中的数据类型区分符应为无限制的
   * **优点**
     + 增强了数据安全性
     + 提高了数据库性能
     + 节省内存
     + 提高了开发的工作效率
     + 完整性
   * **语法**

|  |
| --- |
| CREATE [OR REPLACE] PROCEDURE <过程名> [(参数1,  … 参数N) ] IS|AS  [局部声明]  BEGIN  可执行语句；  EXCEPTION  [例外处理程序]；  END [<过程名>]; |

* + **例子**

|  |
| --- |
| *--创建*  create procedure proc\_first  as  begin  dbms\_output.put\_line('HELLO WORLD');  end;  *--调用*  begin  proc\_first;  end; |

* + **参数模式**
    - **In参数：**输入，接受值，默认模式

参数可以是一个常数、数据量、初始化变量或表达式

|  |
| --- |
| *--接受参数雇员编号，显示姓名和工资信息*  create or replace procedure proc\_sal(in\_empno in emp.empno%type)  is  v\_ename emp.ename%type;  v\_sal emp.sal%type;  begin  select ename,sal into v\_ename,v\_sal from emp where empno=in\_empno;  dbms\_output.put\_line('姓名：' || v\_ename);  dbms\_output.put\_line('工资：' || v\_sal);  exception  when no\_data\_found then  dbms\_output.put\_line('输入雇员编号有误！');  end; |

* + - **Out参数：**输出，将值返回给子程序的调用程序

参数应为一个变量,不能为一个常量或一个表达式

|  |
| --- |
| *--统计雇员人数*  create or replace procedure proc\_count(out\_count out number)  is  begin  select count(\*) into out\_count from emp;  end;  *--调用*  declare  v\_count number(2);  begin  proc\_count(v\_count);  dbms\_output.put\_line('雇员人数：' || v\_count);  end; |

* + - **In out参数：**输入输出，接受值并返回已更新的值

参数应为一个变量,不能为一个常量或一个表

被赋值，它的值也可以对另一个变量赋值

|  |
| --- |
| *--输入参数为雇员编号，返回该雇员的工资*  create or replace procedure proc\_nowsal(in\_empno emp.empno%type  ,out\_sal out emp.sal%type)  is  begin  select sal into out\_sal from emp where empno=in\_empno;  end;  *--调用*  declare  v\_sal emp.sal%type;  begin  proc\_nowsal(7369,v\_sal);  dbms\_output.put\_line('雇员工资：' || v\_sal);  end; |

* + **删除过程**

|  |
| --- |
| drop procedure proc\_nowsal |

1. **函数**
   * **概述**
     + 函数是可以返回值的已命名的 PL/SQL 子程序
     + 参数与返回值类型不用指定长度
   * **限制**
     + 函数只能接受 IN 参数，而不能接受 IN OUT 或 OUT 参数
     + 函数的返回类型也必须是数据库类型
     + 可以在PL/SQL块或SQL语句中访问
   * **语法**

|  |
| --- |
| CREATE [OR REPLACE] FUNCTION <函数名> [(参数1, …  参数N) ] RETURN datatype IS  [局部声明]  BEGIN  可执行语句;  [EXCEPTION]  [例外处理程序]  END [<函数名>]; |

* + **例子**

|  |
| --- |
| *--格式化日期*  create or replace function fun\_datetochar(fn\_date date)  return varchar2  is  rstr varchar2(30);  begin  rstr := to\_char(fn\_date,'yyyy') || '年'  || to\_char(fn\_date,'mm') || '月'  || to\_char(fn\_date,'dd') || '日'  || to\_char(fn\_date,'hh24') || '时'  || to\_char(fn\_date,'mi') || '分'  || to\_char(fn\_date,'ss') || '秒';  return rstr;  end;  *--在sql中调用*  select fun\_datetochar(hiredate) from emp;  *--在PL/SQL块中调用*  begin  dbms\_output.put\_line('当前时间：'||fun\_datetochar(sysdate));  end; |

* + **删除函数**

|  |
| --- |
| drop function fun\_datetochar |

1. **函数和过程的区别**

|  |  |
| --- | --- |
| **过程** | **函数** |
| 作为 PL/SQL 语句执行 | 作为表达式的一部分调用 |
| 在规格说明中不包含RETURN 子句 | 必须在规格说明中包含 RETURN 子句 |
| 不返回任何值 | 必须返回单个值 |
| 可以包含 RETURN 语句，但是与函数不同，它不能用于返回值 | 必须包含至少一条 RETURN 语句 |

1. **程序包**
   * **概述**
     + 程序包是对相关过程、函数、变量、游标和异常等对象的封装
     + 程序包由规范和主体两部分组成
   * **优点**
     + 模块化
     + 更轻松的应用程序设计
     + 信息隐藏
     + 新增功能
     + 性能更佳
   * **包的开发步骤**

1.将每个存储过程调式正确

2.用文本编辑软件将各个存储过程和函数集成在一起

3.按照包的定义要求将集成的文本的前面加上包定义

4.按照包的定义要求将集成的文本的前面加上包主体

5.使用SQLPLUS或开发工具进行调式

* + **语法**
    - **包规范语法**

|  |
| --- |
| create [or replace] package 包名  as  *--游标类型或过程或函数声明*  procedure 过程名(参数列表);  end; |

* + - **包主体语法**

|  |
| --- |
| create [or replace] package body 包名  as  *--过程或函数实现*  procedure 过程名(参数列表)  as  begin…end;  end; |

* + **例子：**实现添加部门，在包中定义一个函数判断添加的部门是否存在， 如果不存在就在过程中实现新增
    - **创建规范**

|  |
| --- |
| create or replace package pack\_first  as  *--函数*  function isdept(in\_dname varchar2) return boolean;  *--过程*  procedure adddept(in\_deptno number,in\_dname varchar2,in\_loc varchar2);  end; |

* + - **创建主体**

|  |
| --- |
| create or replace package body pack\_first  as  *--函数*  function isdept(in\_dname varchar2)  return boolean  is  num number(1);  begin  select count(\*) into num from dept where dname = in\_dname;  if num>0 then  return true;  else  return false;  end if;  end;  *--过程*  procedure adddept(in\_deptno number,in\_dname varchar2,in\_loc varchar2)  is  is\_dept boolean;  begin  is\_dept:=isdept(in\_dname);  if is\_dept then  dbms\_output.put\_line('添加的部门已经存在');  else  insert into dept values(in\_deptno,in\_dname,in\_loc);  end if;  commit;  exception  when others then  dbms\_output.put\_line('添加部门失败!');  end;  end; |

* + - **调用**

|  |
| --- |
| begin  pack\_first.adddept(70,'训导部','珠海');  end; |

* + - **删除**

|  |
| --- |
| *--删除规范*  drop package pack\_first  *--删除主体*  drop package body pack\_first |

* + **在程序包中使用游标：**输入部门编号返回员工信息
    - **创建规范**

|  |
| --- |
| create or replace package pack\_cursor  as  *--ref游标*  type my\_cursor is ref cursor return emp%rowtype;  *--过程*  procedure findEmp(in\_deptno number,inCursor out my\_cursor);  end; |

* + - **创建主体**

|  |
| --- |
| create or replace package body pack\_cursor  as  *--过程*  procedure findEmp(in\_deptno number,inCursor out my\_cursor)  is  begin  open inCursor for select \* from emp where deptno=in\_deptno;  end;  end; |

* + - **调用**

|  |
| --- |
| declare  v\_myCursor pack\_cursor.my\_cursor;  v\_emp emp%rowtype;  begin  pack\_cursor.findEmp(10,v\_myCursor);  loop  fetch v\_myCursor into v\_emp;  exit when v\_myCursor%notfound;  dbms\_output.put\_line('姓名：' || v\_emp.ename);  end loop;  close v\_myCursor;  end; |

1. **触发器和内置程序包**
2. **触发器**
   * **概述**

触发程序 (trigger) 是一种在数据库中，在对数据库有异动的动作时，先行拦截并处理的一种数据库对象，它大部份会设在数据表中，作为强制运行特定动作的程序。

* + **作用**
    - 自动生成数据
    - 自定义复杂的安全权限
    - 提供审计和日志记录
    - 启用复杂的业务逻辑
  + **触发器类型**
    - DML触发器
    - 替代触发器
    - 系统触发器
  + **DML触发器分类**
    - 按触发事件划分
      * Insert—新增数据触发
      * Update—修改数据触发
      * delete—删除数据触发
    - 按触发时间划分
      * before—先触发后执行
      * after—先执行后触发
    - 按触发级别划分
      * 语句级—每个dml语句不论影响了多少行都只触发一次
      * 记录级—针对dml语句影响的每个行都会触发一次
  + **触发器语法**

|  |
| --- |
| CREATE [OR REPLACE] TRIGGER trigger\_name  AFTER | BEFORE | INSTEAD OF  [INSERT] [[OR] UPDATE [OF column\_list]]  [[OR] DELETE]  ON table\_or\_view\_name  [REFERENCING {OLD [AS] old / NEW [AS] new}]  [FOR EACH ROW]  [WHEN (condition)]  trigger\_body; |

* + **组成部分**
    - **触发器语句（事件）**

定义激活触发器的 DML 事件和 DDL 事件

* + - **触发器限制**

执行触发器的条件，该条件必须为真才能激活触发器

* + - **触发器操作（主体）**

包含一些 SQL 语句和代码，它们在发出了触发器语句且触发限制的值为真时运行

* + **行级触发器：**实现对emp表工资高于2000不能进行DML操作

|  |
| --- |
| *--创建触发器*  create or replace trigger trigger\_emp\_row  after *--触发时间*  insert or update or delete*--触发事件*  on emp *--目标表*  for each row *--触发级别*  *--主体*  begin  if :old.sal>2000 then  *--直接产生异常函数(-20000~-20999)*  raise\_application\_error(-20001,'不允许修改工资大于2000的雇员信息');  end if;  end; |

* + **行级触发器相关标识**

|  |  |  |
| --- | --- | --- |
| **触发语句** | **:old** | **:new** |
| insert | 无定义，所有列为null | 将插入值 |
| update | 更新前，行的原始值 | 可以修改值 |
| delete | 行删除前的原始值 | 无定义，所有列为null |

* + **语句级触发器：**实现周六日不能对emp表进行DML操作

|  |
| --- |
| *--创建触发器*  create or replace trigger trigger\_emp\_table  before *--触发时间*  insert or update or delete*--触发事件*  on emp *--目标表*  *--主体*  declare  v\_week varchar2(6);  begin  *--获取星期*  select to\_char(sysdate,'day') into v\_week from dual;  if v\_week='星期六' or v\_week='星期日' then  *--直接产生异常函数(-20000~-20999)*  raise\_application\_error(-20002,'休息日不允许修改雇员信息');  end if;  end; |

* + **INSTEAD OF(替代)触发器**
    - 定义
      * 替代触发器可以定义在视图上，而DML触发器只能定义在表上
      * 替代触发器是行级的
    - **用途**
      * 允许对无法变更的视图进行修改
      * 修改视图嵌套表的列

|  |
| --- |
| *--创建替代触发器*  create or replace trigger trigger\_view  instead of*--替代触发器*  update  on view\_temp  for each row  *--主体*  begin  if :old.dname='SALES' then  raise\_application\_error(-20003,'不能修改SALES部门信息');  end if;  end; |

* + **触发器谓语**

|  |  |
| --- | --- |
| **表达式** | **谓语状态** |
| inserting | 触发语句为insert，则为真(true)，否则为假(false) |
| updating | 触发语句为update，则为真(true)，否则为假(false) |
| deleting | 触发语句为delete，则为真(true)，否则为假(false) |

* + **禁用触发器**

|  |
| --- |
| ALTER TRIGGER trigger\_emp\_table DISABLE |

* + **启用触发器**

|  |
| --- |
| ALTER TRIGGER trigger\_emp\_table ENABLE |

* + **删除触发器**

|  |
| --- |
| DROP TRIGGER trigger\_emp\_table |

1. **内置程序包**
   * **作用**
     + 扩展数据库的功能
     + 为 PL/SQL 提供对 SQL 功能的访问
     + 用户 SYS 拥有所有程序包
     + 是公有同义词
     + 可以由任何用户访问
   * **常用内置程序包**

|  |  |
| --- | --- |
| **程序报名** | **说明** |
| STANDARD和DBMS\_STANDARD | 定义和扩展PL/SQL语言环境 |
| DBMS\_LOB | 提供对 LOB数据类型进行操作的功能 |
| DBMS\_OUTPUT | 处理PL/SQL块和子程序输出调试信息 |
| DBMS\_RANDOM | 提供随机数生成器 |
| DBMS\_SQL | 允许用户使用动态 SQL |
| DBMS\_XMLDOM | 用DOM模型读写XML类型的数据 |
| DBMS\_XMLPARSER | XML解析，处理XML文档内容和结构 |
| DBMS\_XMLQUERY | 提供将数据转换为 XML 类型的功能 |
| DBMS\_XSLPROCESSOR | 提供XSLT功能，转换XML文档 |
| UTL\_FILE | 用 PL/SQL 程序来读写操作系统文本文件 |

* + **DBMS\_OUTPUT:** 显示 PL/SQL 块和子程序的调试信息

|  |
| --- |
| *--打印\*三角*  begin  for i in 1..10 loop  for j in 1..i loop  dbms\_output.put('\*');  end loop;  dbms\_output.new\_line;  end loop;  end; |

* + **DBMS\_RANDOM：**可用来生成随机整数

|  |
| --- |
| declare  v\_num number;  begin  for i in 1..10 loop  v\_num :=abs(dbms\_random.random mod 36);  dbms\_output.put\_line('产生随机数'||i||':'||v\_num);  end loop;  end; |

* + **DBMS\_XMLQUERY:**用于将查询结果转换为 XML 格式

|  |
| --- |
| declare  v\_xml CLOB;  begin  v\_xml := Dbms\_Xmlquery.getXML('select ename,job,sal from emp');  dbms\_output.put\_line(v\_xml);  end; |

1. **备份与恢复**
2. **基本概念**
   * **数据丢失的分类**
     + 逻辑丢失：用户错误操作和应用程序错误造成
     + 物理丢失：数据库物理结构破坏
   * **备份和恢复的分类**
     + 逻辑备份和逻辑恢复
     + 物理备份和物理恢复
   * 物理备份
     + 物理数据备份可分为冷备份(cold)和热备份(hot)
     + 即脱机备份(offline backup)和联机备份(online backup)

离线

在线

1.恢复到故障时

2.基于时间点恢复

底版本7.0没有

8I 开始有恢复管理器

1. **逻辑备份**
   * 定义
     + 逻辑备份使用EXP程序完成。
     + 逻辑备份可分为：指定表的逻辑备份、指定用户的逻辑备份和整库逻辑备份。
   * 语法

|  |
| --- |
| EXP  USERID 用户名/口令 FULL 导出整个文件 (N)  FILE 输出文件 (EXPDAT.DMP) OWNER 所有者用户名列表  TABLES 表名列表 |

* + 例子

|  |
| --- |
| *--导出整个库*  exp system/system full=y file=e:/20131005.dmp  *--导出指定的用户*  exp system/system owner=scott file=e:/20131005.dmp  *--导出指定的表*  exp scott/tiger tables=emp,dept file=e:/20131005.dmp |

1. **逻辑恢复**
   * 定义
     + 逻辑恢复使用IMP程序完成
   * 语法

|  |
| --- |
| IMP  USERID 用户名/口令 FULL 导入整个文件 (N)  FROMUSER 所有者用户名列表 TABLES 表名列表  FILE 输入文件 (EXPDAT.DMP) TOUSER 用户名列表 |

* + 例子

|  |
| --- |
| *--导入指定的表*  imp scott/tiger tables=temp fromuser=scott file=e:/temp.dmp;  *--导入指定用户，如果用户不存在需先创建*  imp admin/admin file=e:/20131005.dmp fromuser=scott  *--导入表丢失的数据*  imp admin/admin file=e:/scott.dmp tables=emp fromuser=scott ignore=y  *--导入整库数据*  imp system/system full=y file=e:/full.dmp |

1. **物理备份与恢复**
   * **脱机备份与脱机恢复**
     + 定义
       - 脱机备份是在关闭数据库后进行的完全镜像备份，其中包括参数文件、网络连接文件、控制文件、数据文件和联机重做日志文件。
       - 脱机恢复是用备份文件将数据库恢复到备份时的状态。
       - 进行冷备份时可能出现备份不全或备份了垃圾数据的状况。

* 脱机备份步骤

|  |
| --- |
| *1.关闭数据库*  SQL>CONNECT sys/sys as sysdba；  SQL>shutdown immediate；  *2.创建备份文件的目录：*  如 e:\oraclebak  *3.备份*  查询数据文件位置：  SQL> select file\_name from dba\_data\_files;  查询控制文件位置：  SQL> select name from v$controlfile;  查询联机重做日志文件位置：  SQL> select member from v$logfile;  参数文件位置：D:\app\Administrator\product\11.2.0\dbhome\_1\dbs  网络连接文件位置：D:\app\Administrator\product\11.2.0\dbhome\_1\NETWORK\ADMIN  将以上位置的文件复制到建立的备份目录下  *4.启动数据库*  SQL>startup |

* 脱机恢复步骤

|  |
| --- |
| *1.关闭数据库*  SQL>CONNECT sys/sys as sysdba  SQL>shutdown immediate  *2.把所有备份文件手动拷贝到原来所在的位置*  *3.启动数据库*  SQL>startup |

* + **联机备份与恢复**
    - 使用恢复管理器（Recovery Manager, RMAN）
      * RMAN能够备份整个数据库或数据库部件，如表空间、数据文件、控制文件、归档文件以及spfile参数文件。
      * RMAN也允许您进行增量数据块级别的备份，增量RMAN备份是时间和空间有效的，因为他们只备份自上次备份以来有变化的那些数据块。
      * 通过RMAN，也提供了其它更多功能，如数据库的克隆、采用RMAN建立备用数据库、利用RMAN备份与移动裸设备（RAW）上的文件等工作将变得更方便简单。
    - 使用RMAN进行热备份
    1. 启用archivelog模式

|  |
| --- |
| SQL>connect sys/sys as sysdba;  SQL>shutdown immediate;  SQL>startup mount;  SQL>alter database archivelog; |

* + 1. 连接到本地数据库

|  |
| --- |
| C:\>rman target / |

* + 1. 启动数据库

|  |
| --- |
| RMAN>startup |

* + 1. 备份
       - 数据库全备份

|  |
| --- |
| *--备份默认路径*  RMAN> backup database;  *--指定备份集的路径*  RMAN> backup database format 'E:/db\_%U'; |

* + - * 备份一个表空间

|  |
| --- |
| RMAN>backup tablespace users format 'E:/ts\_%U'; |

* + - * 备份一个数据文件

|  |
| --- |
| *--查看数据文件命令*  SQL>select file#,a.name,b.name from v$datafile a, v$tablespace b where a.ts#=b.ts#;  *--执行*  RMAN>backup datafile 'D:\APP\ADMINISTRATOR\ORADATA\ORCL\EXAMPLE01.DBF' format 'E:/df\_%U'; |

* + - * 备份控制文件

|  |
| --- |
| RMAN>Backup current controlfile format 'E:/ctl\_%U'; |

* + - * 备份归档日志文件

|  |
| --- |
| RMAN> backup filesperset 20 format 'E:/al\_%U' archivelog all |

* + - * 查看数据库的备份

|  |
| --- |
| *--查看备份数据文件*  RMAN>list backup of database;  *--查看备份控制文件*  RMAN> list backup of controlfile;  *--查看备份归档日志文件*  RMAN> list backup of archivelog all; |

* + - * 查看废弃的数据库备份

|  |
| --- |
| RMAN> report obsolete; |

* + - * 删除废弃的数据库备份

|  |
| --- |
| RMAN> delete obsolete; |

* + - 使用RMAN进行热恢复
      * 恢复所有文件

|  |
| --- |
| RMAN> restore database; |

* + - * 恢复表空间

|  |
| --- |
| RMAN> restore tablespace users; |

* + - * 恢复一个数据文件

|  |
| --- |
| RMAN> restore datafile 'D:\APP\ADMINISTRATOR\ORADATA\ORCL\EXAMPLE01.DBF'; |

* + **三种备份与恢复方法的比较**
    - **逻辑备份与恢复** 
      * 优点：
        1. 方便，灵活，不受操作系统，数据库版本等限制。
        2. 适合处理小数据量，特别适合开发人员迁移数据库。
      * 缺点：可能有数据库丢失。不能实时,速度慢。
    - **冷备份** 
      * 优点：方便简单。
      * 缺点：
        1. 在进行数据库冷备份的过程中数据库必须处于关闭状态。
        2. 数据库只能完成基于某一时间点上的恢复。
        3. 若磁盘空间有限，冷备份只能将备份数据拷贝到磁带等其他外部存储上，速度会更慢。
        4. 不能按表或按用户恢复。不同操作系统不行，不同版本不行
    - **恢复管理器（Recovery Manager,　RMAN）方法**
      * **优点：**
        1. 可在表空间或数据文件级备份，备份时间短。
        2. 可达到秒级恢复(恢复到某一时间点上)。
        3. 可对几乎所有数据库实体作恢复。
        4. 恢复是快速的，在大多数情况下在数据库仍工作时恢复。
        5. 备份时数据库仍可用。
      * **缺点：**
        1. 因难以维护，所以要特别仔细小心，不允许“以失败而告终”。
        2. 若热备份不成功，所得结果不可用于时间点的恢复。
        3. 不能出错，否则后果严重。