# Oracle教程

目录

[Oracle教程 1](#_Toc299920496)

[一、 为什么要学习Oracle 8](#_Toc299920497)

[二、 Oracle公司 8](#_Toc299920498)

[三、 Oracle安装、启动及缷载 9](#_Toc299920499)

[3.1. 安装 9](#_Toc299920500)

[3.2. Oracle的卸载、删除 9](#_Toc299920501)

[四、 Oracle体系结构 10](#_Toc299920502)

[4.1. 体系结构 10](#_Toc299920503)

[4.2. 会话 11](#_Toc299920504)

[4.3. 内存结构 11](#_Toc299920505)

[4.4. Oracle 实例进程结构 12](#_Toc299920506)

[4.5. Oracle 物理组件 13](#_Toc299920507)

[4.6. Oracle 逻辑组件 13](#_Toc299920508)

[4.7. 模式 15](#_Toc299920509)

[4.8. 数据文件（\*.dbf） 15](#_Toc299920510)

[4.9. 控制文件(\*.tcl)和日志文件(\*.log) 16](#_Toc299920511)

[4.10. 逻辑存储结构 16](#_Toc299920512)

[4.11. 数据字典 17](#_Toc299920513)

[五、 Oracle术语解释详解(摘自网络) 20](#_Toc299920514)

[4.1. 数据库名 21](#_Toc299920515)

[4.2. 数据库实例名 23](#_Toc299920516)

[4.3. 数据库域名 24](#_Toc299920517)

[4.4. 数据库服务名 24](#_Toc299920518)

[六、 SYS和SYSTEM用户 25](#_Toc299920519)

[七、 启动Oracle 25](#_Toc299920520)

[八、 Oracle管理工具介绍 26](#_Toc299920521)

[九、 查看Oracle 27](#_Toc299920522)

[1. 查看当前的Oracle实例名称和主机名称： 27](#_Toc299920523)

[2. 查看用户帐户及状态 27](#_Toc299920524)

[十、 Oracle用户解锁 27](#_Toc299920525)

[1. 对scott用户解锁 27](#_Toc299920526)

[2. 更改scott用户的密码： 27](#_Toc299920527)

[3. 锁定scott用户： 27](#_Toc299920528)

[十一、 使用OEM 27](#_Toc299920529)

[十二、 SQL/PLUS连接数据库 28](#_Toc299920530)

[12.1. SQL\*Plus工具 28](#_Toc299920531)

[12.2. 连接方式 28](#_Toc299920532)

[12.3. 连接远程数据库 29](#_Toc299920533)

[十三、 SQL\*PLUS常用命令 30](#_Toc299920534)

[十四、 表空间 49](#_Toc299920535)

[14.1. 创建表空间 49](#_Toc299920536)

[14.2. 表空间状态属性 53](#_Toc299920537)

[14.3. 删除表空间 56](#_Toc299920538)

[14.4. 增加表空间尺寸 56](#_Toc299920539)

[14.5. 查看剩余空间 56](#_Toc299920540)

[14.6. 合并空闲块 57](#_Toc299920541)

[14.7. 设置默认表空间 57](#_Toc299920542)

[14.8. 撤销表空间 58](#_Toc299920543)

[14.9. 查看当前用户的缺省表空间 61](#_Toc299920544)

[十五、 用户管理 61](#_Toc299920545)

[15.1. 创建用户 61](#_Toc299920546)

[15.2. 删除用户 63](#_Toc299920547)

[15.3. 修改用户 63](#_Toc299920548)

[15.4. 修改口令 63](#_Toc299920549)

[15.5. 使密码失效 64](#_Toc299920550)

[15.6. 修改默认表空间 64](#_Toc299920551)

[15.7. 锁定用户 64](#_Toc299920552)

[15.8. 权限的概述 65](#_Toc299920553)

[15.9. 系统权限 66](#_Toc299920554)

[15.10. 对象权限 70](#_Toc299920555)

[15.11. 查看和撤消权限 72](#_Toc299920556)

[15.12. 角色 75](#_Toc299920557)

[15.13. 创建角色 77](#_Toc299920558)

[15.14. 为角色授予权限 78](#_Toc299920559)

[15.15. 为用户授予角色 78](#_Toc299920560)

[15.16. 修改和删除角色 79](#_Toc299920561)

[十六、 数据类型 81](#_Toc299920562)

[16.1. 关于number、int、integer的说明 83](#_Toc299920563)

[16.2. binary\_float和binary\_double 83](#_Toc299920564)

[16.3. 伪列 84](#_Toc299920565)

[十七、 表和约束 84](#_Toc299920566)

[17.1. 创建表 84](#_Toc299920567)

[17.2. 管理表中的列 86](#_Toc299920568)

[17.3. 重命名表 87](#_Toc299920569)

[17.4. 移动表 87](#_Toc299920570)

[17.5. 删除表 88](#_Toc299920571)

[17.6. 约束分类 88](#_Toc299920572)

[17.7. 查看约束 93](#_Toc299920573)

[17.8. 指定级联操作类型 94](#_Toc299920574)

[十八、 SQL基础 95](#_Toc299920575)

[18.1. SQL简介 95](#_Toc299920576)

[18.2. 数据定义语言 97](#_Toc299920577)

[18.3. 数据操纵语言 97](#_Toc299920578)

[18.4. 事务控制语言 98](#_Toc299920579)

[18.5. 数据控制语言 99](#_Toc299920580)

[18.6. SQL 操作符 99](#_Toc299920581)

[18.7. MEGER语句 103](#_Toc299920582)

[十九、 Oracle函数 106](#_Toc299920583)

[19.1. 字符函数 106](#_Toc299920584)

[19.2. 数字函数 108](#_Toc299920585)

[19.3. 聚合函数 109](#_Toc299920586)

[19.4. 日期时间函数 110](#_Toc299920587)

[19.5. 转换函数 111](#_Toc299920588)

[19.6. 日期时间格式参数 113](#_Toc299920589)

[19.7. 分析函数 114](#_Toc299920590)

[19.8. 通用函数 115](#_Toc299920591)

[19.9. GROUP BY子句 116](#_Toc299920592)

[19.10. HAVING子句 117](#_Toc299920593)

[19.11. 实用方法 118](#_Toc299920594)

[19.12. 练习 118](#_Toc299920595)

[二十、 子查询 119](#_Toc299920596)

[20.1. 使用子查询 120](#_Toc299920597)

[20.2. IN\NOT IN操作符 122](#_Toc299920598)

[20.3. ANY和ALL操作符 123](#_Toc299920599)

[20.4. 常见查询错误 124](#_Toc299920600)

[20.5. 多列子查询 125](#_Toc299920601)

[20.6. 关联子查询 126](#_Toc299920602)

[20.7. 嵌套子查询 127](#_Toc299920603)

[二十一、 高级查询 128](#_Toc299920604)

[22.1. UNION操作符 128](#_Toc299920605)

[22.2. 层次化查询 129](#_Toc299920606)

[22.3. 格式化查询 131](#_Toc299920607)

[22.4. 遍历查询 133](#_Toc299920608)

[22.5. 扩展Group By子句 134](#_Toc299920609)

[二十二、 模式对象 137](#_Toc299920610)

[23.1. 同义词 137](#_Toc299920611)

[23.2. 序列 138](#_Toc299920612)

[23.3. 索引 140](#_Toc299920613)

[23.4. 视图 155](#_Toc299920614)

[二十三、 分区表 159](#_Toc299920615)

[24.1. 分区 159](#_Toc299920616)

[24.2. 创建分区表 161](#_Toc299920617)

[24.3. 范围分区（Range） 161](#_Toc299920618)

[24.4. 列表分区(List) 164](#_Toc299920619)

[24.5. 散列分区(Hash) 165](#_Toc299920620)

[24.6. 组合范围散列分区 167](#_Toc299920621)

[24.7. 复合范围散列分区 168](#_Toc299920622)

[24.8. 维护表分区 169](#_Toc299920623)

[二十四、 PL/SQL基础 173](#_Toc299920624)

[24.1. PL/SQL概述 173](#_Toc299920625)

[24.2. PL/SQL代码编写规则 173](#_Toc299920626)

[24.3. PL/SQL程序块 175](#_Toc299920627)

[24.4. 常量和数据类型 177](#_Toc299920628)

[24.5. 变量 179](#_Toc299920629)

[24.6. 运算符和注释 181](#_Toc299920630)

[24.7. IF条件语句 182](#_Toc299920631)

[24.8. CASE语句 184](#_Toc299920632)

[24.9. 循环语句 186](#_Toc299920633)

[24.10. 复合变量 190](#_Toc299920634)

[24.11. 记录类型 192](#_Toc299920635)

[24.12. 表类型 194](#_Toc299920636)

[24.13. 集合类型 195](#_Toc299920637)

[24.14. 集合方法 199](#_Toc299920638)

[24.15. 游标 205](#_Toc299920639)

[24.16. 异常处理 222](#_Toc299920640)

[二十五、 PL/SQL高级 228](#_Toc299920641)

[25.1. 子程序 228](#_Toc299920642)

[25.2. 存储过程 229](#_Toc299920643)

[25.3. 函数 236](#_Toc299920644)

[25.4. 程序包 240](#_Toc299920645)

[25.5. 触发器 243](#_Toc299920646)

## 为什么要学习Oracle

1. 目前主流数据库：

微软： SQL Server、access

瑞典MySql： AB公司MySql

IBM公司： DB2

美国SyBase公司： SyBase

美国Oracle公司： Oracle

该使用什么样的数据库？

一般考虑这些方面：

* 负载量多大
* 成本
* 安全性

1. 商场需求量大

## Oracle公司

* Oracle:神喻 甲骨文
* 地位：全球第一大数据库厂商、全球第二大独立软件供应商
* 历史：

1970年 数据库起家

1980年 oracle6版本

1990年 oracle7挑战杯，向多元化产品进军。以数据库为优先发展方向。

1995年 oracle8版本

1999年 oracle8i版本

2002年 oracle9i版本

2003年 oracle10g版本

* 产品：
* Jdeveloper------JBuilder
* Developer2000----VB
* Designer2000----数据库建模
* 数据库

oracle10g

* 应用软件

ERP（应业资源计划）

Oracle Application 11i

## Oracle安装、启动及缷载

### 安装

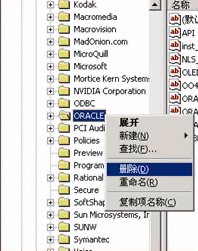
* 操作系统最好为windows 2000
* 内存256M以上
* 硬盘2G以上
* 建议：

1. 不要重复安装
2. 如果是vista或windows7操作系统，以系统管理员身份登录
3. 安装目录中不要包含中文、空格

### Oracle的卸载、删除

相对来说，Oracle的卸载比较麻烦。

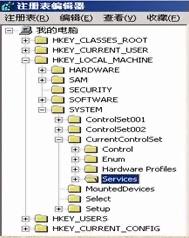
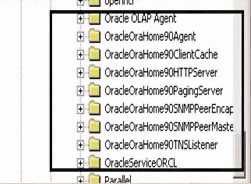
1. 运行Oracle的universal installer
2. 运行regedit，进入注册表，删除Oracle相关信息。
3. 进入HKEY\_LOCAL\_MACHINE\SOFTWARE\ORACLE,右键，删除



1. HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services下以Oracle开头的键值。



1. 重启计算机，然后删除硬盘上的Oracle安装目录。如果该目录不让删除，那么把该目录改成别的名字，再重启计算机，再删除它。



## Oracle体系结构

1. Oracle是对象关系型数据库系统
2. Oracle数据库基于客户端/服务器技术

### 体系结构

1. Oracle服务器由Oracle数据库和Oracle实例组成

**Oracle数据库：**是数据的集合，一个逻辑单元，一系列的数据文件、日志文件、控制文件等。物理文件，存在硬盘上。

**Oracle实例：**用于维护数据库。管理数据库的后台进程和内存结构的集合称为，一系列的进程，在内存中。

1. Oracle數據庫包括

实例

**内存结构**

**后台进程**

**后台进程**

**PMON SMON DBWR LGWR CKPT** 其他

**数据库**

**数据库**

数据文件

数据文件

数据文件

控制文件

控制文件

日志文件

日志文件

参数文件

口令文件

**SGA**

用户进程

服务器进程

**PGA**

共享池

数据缓冲区

日志缓冲区

物理結構：物理结构包含数据库中的一组操作系统文件。

邏輯結構：逻辑结构指数据库创建之后形成的逻辑概念之间的关系

### 会话

会话是用户与 Oracle 服务器的单个连接

当用户与服务器建立连接时创建会话

当用户与服务器断开连接时关闭会话

### 内存结构

**系统全局区（SGA）：**当Oracle实例启动时分配系统全局区

**程序全局区（PGA）：**当服务器进程启动时分配程序全局区

1. 系统全局区（SGA）

内存结构：

1. 共享池：

共享池是对SQL、PL/SQL程序进行语法分析、编译、执行的内存区域

共享池由库缓存和数据字典缓存组成。

共享池的大小直接影响数据库的性能

1. 數據緩衝區

用于存储从磁盘数据文件中读入的数据，所有用户共享。

服务器进程将读入的数据保存在数据缓冲区中，当后续的请求需要这些数据时可以在内存中找到，不需要再从磁盘读取，提高了读取速度。

数据缓冲区的大小对数据库的读取速度有直接的影响。

1. 日志緩衝區

日志记录数据库的所有修改信息，日志信息首先产生于日志缓冲区。

当日志缓冲区的日志数据达到一定数量时，由后台进程将日志数据写入日志文件中。

相对来说，日志缓冲区对数据库的性能影响较小。

1. 程序全局區
2. 包含单个服务器进程所需的数据和控制信息
3. PGA是在用户进程连接到数据库并创建一个会话时自动分配的，保存每个与Oracle 数据库连接的用户进程所需的信息
4. PGA为非共享区，只能单个进程使用，当一个用户会话结束，PGA释放

### Oracle 实例进程结构

實例進程分為：

1. 用戶進程

用户进程是一个需要与Oracle服务器进行交互的程序。当用户运行一个应用程序准备向数据库服务器发送请求时，即创建了用户进程

1. 服務器進程

服务器进程用于处理连接到该实例的用户进程的请求。当用户连接至 Oracle 数据库实例创建会话时，即产生服务器进程

1. 后臺進程

后台进程是Oracle数据库为了保持最佳系统性能和协调多个用户请求而设置的。 Oracle 实例启动时即创建一系列后台进程

后台进程

PMON

SMON

DBWR

LGWR

CKPT

其他

1. PMON 进程监控进程

清理出现故障的进程。

释放所有当前挂起的锁定。

释放故障进程使用的资源

1. SMON 系统监控进程

在实例失败之后，重新打开数据库时自动恢复实例。

整理数据文件的自由空间，将相邻区域结合起来。

释放不再使用的临时段

1. DBWR 数据写入进程

管理数据缓冲区，将最近使用过的块保留在内存中。

将修改后的缓冲区数据写入数据文件中。

1. LGWR 日志写入进程

负责将日志缓冲区中的日志数据写入日志文件。

系统有多个日志文件，该进程以循环的方式将数据写入文件

### Oracle 物理组件

物理组件就是Oracle数据库所使用的操作系统物理文件。物理文件可分为三类：

物理组件

数据文件

控制文件

日志文件

* 数据文件用于存储数据库数据，如表、索引数据等。
* 控制文件是记录数据库物理结构的二进制文件
* 日志文件记录对数据库的所有修改信息，用于故障恢复

### Oracle 逻辑组件

数据库的逻辑结构是从逻辑的角度分析数据库的组成。Oracle 的逻辑组件包括:

数据库

表空间

段

区

数据块

模式

* 1. 表空间：

A、表空间是数据库中最大的逻辑单位，一个 Oracle 数据库至少包含一个表空间，就是名为SYSTEM的系统表空间。

B、每个表空间是由一个或多个数据文件组成的，一个数据文件只能与一个表空间相关联。

C、表空间的大小等于表空间中所有数据文件大小之和

创建表空间的语法是：

|  |
| --- |
| CREATE TABLESPACE tablespacename  DATAFILE ‘filename’ [SIZE integer [K|M]]  [AUTOEXTEND [OFF|ON]]; |

* 1. 段

A、段是构成表空间的逻辑存储结构，段由一组区组成。

B、按照段所存储数据的特征，将段分为四种类型，即数据段、索引段、回退段和临时段。

* 1. 区

A、区为段分配空间，它由连续的数据块组成。

B、当段中的所有空间已完全使用，自动为该段分配新区。

C、区不能跨数据文件存在，只能存在于一个数据文件中

* 1. 数据块

A、数据块是Oracle服务器所能分配、读取或写入的最小存储单元。

B、Oracle服务器以块为单位管理数据文件的存储空间。

* 理解：

数据库：相当于小区

表空间：相于小区中的房子，小区至少应该有一栋房子。

段：小区中的一个单元。

区：单元中的一栋房子

数据块：房子中的房间

### 模式

模式是对用户所创建的数据库对象的总称。

模式对象包括表、视图、索引、同义词、序列、过程和程序包等

### 数据文件（\*.dbf）

数据文件(Data File)是指存储数据库数据的文件。数据库中的所有数据最终都保存在数据文件中，例如，表中的记录和索引等。如果数据文件中的某些数据被频繁访问，则这些数据会被存储在内存缓冲区中。读取数据时，Oracle系统会首先从内存的数据缓冲区中查找相关数据信息，如果找不到，则从数据库文件中的把数据读取出来，存放在内存的数据缓冲区中，供查询合用;存储数据时，修改后的数据信息，也是先存放在内存的数据缓冲区中，在满足写入条件(例如执行提交操作)时，由Oracle的后台进行DBWR将数据写入数据文件。

数据文件一般有以下几个特点：

* 一个表空间由一个或多个数据文件组成。
* 一个数据文件只对应一个数据库。而一个数据库通常包含多个数据文件。
* 数据文件可以通过设置其参数，实现其自动扩展的功能。

如果想要了解数据文件的信息，可以查询数据字典dba\_data\_files和

v$datafile。

可以通过查询视图来查看数据库包含哪些数据文件：

|  |
| --- |
| select file\_name from dba\_data\_files; |

### 控制文件(\*.tcl)和日志文件(\*.log)

* 控制文件

控制文件(Control File)是一个很小的二进制文件，用于描述和维护数据库的物理结构。在Oracle数据库中，控制文件相当重要，它存放有数据库中数据文件和日志文件的信息。Oracle数据库在启动时需要访问控制文件，在数据库的使用过程中，数据库需要不断更新控制文件，由此可见，一旦控制文件受损，那么数据库将无法工常工作。

查看控制文件的语句：

|  |
| --- |
| select name from v$controlfile; |

* 日志文件(重做日志文件)

重做日志文件(Redo Log File)是记录数据库中所有修改信息的文件，简称日志文件。其中，修改信息包括 数据库中数据的修改信息和数据库结构的修改信息等，例如删除表中的一行数据或删除表中的一个列，如果只是进行了查询操作，则该操作不会被记录到日志文件中。日志文件是数据库系统的最重要文件之一，它可以保证数据库安全，是进行数据库备份与恢复的重要手段。如果日志文件受损，数据库同样可能无法正常运行。

### 逻辑存储结构

数据库--表空间--段--区--数据块

* 表空间

1. 表空间是Oracle中最大逻辑存储结构，它与物理上的一个或多个数据文件相对应，每个Oracle数据库都至少有一个表空间，表空间的大小等构成该表空间的所有数据文件大小的总和。表空间用于存储用户在数据库中创建的所有内容，例如用户在创建表时，可以指定一个表空间存储该表，如果用户没有指定表空间，则Oracle系统会将用户创建的内容存储到默认的表空间中。
2. 系统默认安装的表空间：

|  |  |
| --- | --- |
| 表空间 | 说明 |
| system | 系统表空间，用于存储系统的数据字典，系统和管理信息和用户数据表等。 |
| sysaux | 辅助系统表空间。用于减少系统表空间的负荷，提高系统作业效率。该表空间由Oracle系统自动维护，一般不用于存储用户数据。 |
| temp | 临时表空间。用于存储临时数据，例如存储排序时产生默认数据。一般情况下，数据库中所有用户都使用temp做为默认的临时表空间。 |
| undotbsl | 撤销表空间。用于在自动撤销管理方式下存储撤销信息。在撤销表空间中，除了回退段以外，不能建立任何其他类型的段，所以，用户不可以在撤销表空间中创建任何数据库对象。 |
| users | 用户表空间。用于存储永久性用户对象和私有信息。通常可以将自定义的一些表信息创建在users表空间中。 |

1. 查看Oracle数据库中存在表空间信息：

|  |
| --- |
| select tablespace\_name from dba\_tablespaces; |

### 数据字典

Oracle数据字典(Data Dictionary)是存储在数据库中的所有对象信息的知识库，Oracle数据库管理系统使用数据字典获取对象信息和安全信息，而用户和数据库系统管理员则用数据字典来查询数据库信息。

Oracle数据字典保存有数据库中对象和段的信息，例如表、视较长、索引、包、存储过程以及与用户、权限、角色、审计和约束等相关的信息。如查看Oracle用户是dba\_users视图。

#### 数据字典视图类型

|  |  |
| --- | --- |
| 视图类型 | 说明 |
| USER视图 | USER视图的名称以user\_为前缀，用来记录用户对象的信息。例如user\_tables视图。(只能获得用户定义的表信息) |
| ALL视图 | ALL视图的名称以all\_为前缀，用来记录用户对象的信息以及被授权访问的对象信息。例如all\_synonyms视图，用来查看所有同义词。 |
| DBA视图 | DBA视图的名称以dba\_为前缀，用来记录数据库实例的所有对象的信息。例如dba\_tables视图，可以获取数据库实例的所有表信息。 |
| V$视图 | V$视图的名称以v$为前缀，用来记录与数据库活动相关的性能统计动态信息。例如v$datafile视图，可以获得一些数据文件的统计信息。 |
| GV$视图 | GV$视图的名称以gv$为前缀，用来记录分布式环境下所有实例的动态信息。例如gv$lock视图。可以获得数据库实例的信息。 |

#### 基本数据字典

|  |  |
| --- | --- |
| **字典名称** | **说明** |
| dba\_tables | 所有用户的所有表的信息 |
| dba\_tab\_columns | 所有用户的表的字段信息 |
| dba\_views | 所有用户的所有视图信息 |
| dba\_synonyms | 所有用户的同义词信息 |
| dba\_sequences | 所有用户的序列信息 |
| dba\_constraints | 所有用户的表的约束信息 |
| dba\_indexes | 所有用户的表的索引简要信息 |
| dba\_ind\_columns | 所有用户的索引的字段信息 |

#### 与数据库组件相关的数据字典

|  |  |  |
| --- | --- | --- |
| **组件** | **数据字典中的表或视图** | **说明** |
| 数据库 | v$datafile | 记录系统的运行情况 |
| 表空间 | dba\_tablespaces | 系统表空间的基本信息 |
| dba\_free\_space | 系统表空间的空闲空间的信息 |
| 控制文件 | v$controlfile | 系统控制文件的基本信息 |
| v$controlfile\_record\_section | 系统控制文件中记录文档段的信息 |
| v$parameter | 系统各参数的基本信息 |
| 数据文件 | dba\_data\_files | 系统数据文件以及表空间的基本信息 |
| v$filestat | 来自控制文件的数据文件信息 |
| v$datafile\_header | 记录数据文件头部分的基本信息 |

#### 常用动态性能视图

|  |  |
| --- | --- |
| **视图名称** | **说明** |
| v$fixed\_table | 显示当前发行的固定对象的说明 |
| v$instance | 显示当前实例的信息 |
| v$latch | 显示锁存器的统计数据 |
| v$librarycache | 显示有关库缓存性能的统计数据 |
| v$rollstat | 显示联机的回滚段的名字 |
| v$rowcache | 显示活动数据字典的统计 |
| v$sga | 显示有关系统全局区的总结信息 |
| v$sgastat | 显示有关系统全局区的详细信息 |
| v$sort\_usage | 显示临时段的大小及会话 |
| v$sqlarea | 显示SQL区的SQL信息 |

#### 实例

1. 查看dba\_tables中scott的对象

|  |
| --- |
| select table\_name,tablespace\_name,owner from dba\_tables where owner=’scott’; |

1. 获得当前会话信息

|  |
| --- |
| select username,terminal from v$session where username is not null; |

1. 获得当前数据实例的使用情况

|  |
| --- |
| select instance\_name,host\_name,status from v$instance; |

1. 相看表或视图结构

|  |
| --- |
| describe v$instance;或desc v$instance; |

## Oracle术语解释详解(摘自网络)

数据库名、实例名、数据库域名、全局数据库名、服务名，这是几个令很多初学者容易混淆的概念。相信很多初学者都与我一样被标题上这些个概念搞得一头雾水。我们现在就来把它们弄个明白。

### 数据库名

#### 什么是数据库名？

数据库名就是一个数据库的标识，就像人的身份证号一样。他用参数DB\_NAME表示，如果一台机器上装了多全数据库，那么每一个数据库都有一个数据库名。在数据库安装或创建完成之后，参数DB\_NAME被写入参数文件pfile中，格式如下：

[code:1:1aaf156970]

.........

db\_name="orcl" #(不允许修改)

db\_domain=dbcenter.toys.com

instance\_name=orcl

service\_names=orcl.dbcenter.toys.com

control\_file=(...............

.........

[/code:1:1aaf156970]

在每一个运行的ORACLE8i数据库中都有一个数据库名(db\_name),如果一个服务器程序中创建了两个数据库，则有两个数据库名。其控制参数据分属在不同的pfile中控制着相关的数据库。

在创建数据库时就应考虑好数据库名，并且在创建完数据库之后，数据库名不能修改，即使要修改也会很麻烦。因为，数据库名还被写入控制文件中，控制文件是以二进制型式存储的，用户无法修改控制文件的内容。假设用户修改了参数文件中的数据库名，即修改DB\_NAME的值。但是在Oracle启动时，由于参数文件中的DB\_NAME与控制文件中的数据库名不一致，导致数据库启动失败，将返回ORA-01103错误。

#### 数据库名的作用

数据库名是在安装数据库、创建新的数据库、创建数据库控制文件、修改数据结构、备份与恢复数据库时都需要使用到的。

有很多Oracle安装文件目录是与数据库名相关的，如：

winnt: d:\oracle\product\10.1.0\oradata\DB\_NAME\...

Unix: /home/app/oracle/product/10.1.0/oradata/DB\_NAME/...

pfile:

winnt: d:\oracle\product\10.1.0\admin\DB\_NAME\pfile\ini.ora

Unix: /home/app/oracle/product/10.1.0/admin/DB\_NAME/pfile/init$ORACLE\_SID.ora

跟踪文件目录：

winnt: /home/app/oracle/product/10.1.0/admin/DB\_NAME/bdump/...

另外，在创建数据时，careate database命令中的数据库名也要与参数文件中DB\_NAME参数的值一致，否则将产生错误。

同样，修改数据库结构的语句alter database，当然也要指出要修改的数据库的名称。

如果控制文件损坏或丢失，数据库将不能加载，这时要重新创建控制文件，方法是以nomount方式启动实例，然后以create controlfile命令创建控制文件，当然这个命令中也是指指DB\_NAME。

还有在备份或恢复数据库时，都需要用到数据库名。总之，数据库名很重要，要准确理解它的作用。

#### 查询当前数据名

方法一:select name from v$database;

方法二：show parameter db

方法三：查看参数文件

#### 修改数据库名

前面建议：应在创建数据库时就确定好数据库名，数据库名不应作修改，因为修改数据库名是一件比较复杂的事情。那么现在就来说明一下，如何在已创建数据之后，修改数据库名。步骤如下：

1. 关闭数据库。
2. 修改数据库参数文件中的DB\_NAME参数的值为新的数据库名。
3. 以NOMOUNT方式启动实例，修建控制文件(有关创建控制文件的命令语法，请参考oracle文档)

### 数据库实例名

#### 什么是数据库实例名？

数据库实例名是用于和操作系统进行联系的标识，就是说数据库和操作系统之间的交互用的是数据库实例名。实例名也被写入参数文件中，该参数为instance\_name，在winnt平台中，实例名同时也被写入注册表。

数据库名和实例名可以相同也可以不同。在一般情况下，数据库名和实例名是一对一的关系，但如果在oracle并行服务器架构(即oracle实时应用集群)中，数据库名和实例名是一对多的关系。这一点在第一篇中已有图例说明。

#### 查询当前数据库实例名

方法一：select instance\_name from v$instance;

方法二：show parameter instance

方法三：在参数文件中查询。

#### 数据库实例名与ORACLE\_SID

虽然两者都表是oracle实例，但两者是有区别的。instance\_name是oracle数据库参数。而ORACLE\_SID是操作系统的环境变量。ORACLD\_SID用于与操作系统交互，也就是说，从操作系统的角度访问实例名，必须通过ORACLE\_SID。在winnt平台，ORACLE\_SID还需存在于注册表中。

且ORACLE\_SID必须与instance\_name的值一致，否则，你将会收到一个错误，在unix平台，是“ORACLE not available”,在winnt平台，是“TNS:协议适配器错误”。

#### 数据库实例名与网络连接

数据库实例名除了与操作系统交互外，还用于网络连接的oracle服务器标识。当你配置oracle主机连接串的时候，就需要指定实例名。当然8i以后版本的网络组件要求使用的是服务名SERVICE\_NAME。这个概念接下来说明。

### 数据库域名

#### 什么是数据库域名？

在分布式数据库系统中，不同版本的数据库服务器之间，不论运行的操作系统是unix或是windows，各服务器之间都可以通过数据库链路进行远程复制，数据库域名主要用于oracle分布式环境中的复制。举例说明如：

全国交通运政系统的分布式数据库，其中：

福建节点： fj.jtyz

福建厦门节点： xm.fj.jtyz

江西： jx.jtyz

江西上饶：sr.jx.jtyz

这就是数据库域名。

数据库域名在存在于参数文件中，他的参数是db\_domain.

#### 查询数据库域名

方法一：select value from v$parameter where name = 'db\_domain';

方法二：show parameter domain

方法三：在参数文件中查询。

#### 全局数据库名

全局数据库名=数据库名+数据库域名，如前述福建节点的全局数据库名是：oradb.fj.jtyz

不能超过8个字符，不能是中文

### 数据库服务名

#### 什么是数据库服务名？

从oracle9i版本开始，引入了一个新的参数，即数据库服务名。参数名是SERVICE\_NAME。

如果数据库有域名，则数据库服务名就是全局数据库名；否则，数据库服务名与数据库名相同。

#### 查询数据库服务名

方法一：select value from v$parameter where name = 'service\_name';

方法二：show parameter service\_name

方法三：在参数文件中查询。

#### 数据库服务名与网络连接

从oracle8i开始的oracle网络组件，数据库与客户端的连接主机串使用数据库服务名。之前用的是ORACLE\_SID,即数据库实例名。

## SYS和SYSTEM用户

1. SYS是超级用户，具有最高权限，具有sysdba角色，具有create database权限。9i和10g中默认的密码是manager
2. System是管理操作员，权限也很大，具有sysoper角色。没有create database权限。默认的密码是chage\_on\_install
3. 一般对数据库维护使用system用户登录就可以了。

## 启动Oracle

成功安装Oracle后在Windows服务中会生成很多Oracle开头的服务项。其中有二个核心的服务需要启动。

一个是OracleServiceXXX，

另一个是OracleOraHomeXXTNSListener

1. 启动

1、启动监听(命令：lsnrctl start [监听名])

2、启动数据库(命令：oradim -startup -sid [实例名])

1. 默认的三个用户

1、sys(9里默认密码为chage\_on\_install)

2、system(9里默认密码为manager)

3、scott(9里默认密码为tiger，10里需要解锁)

1. 用户登录：
2. 使用SQL\*PLUS。在程序中找到SQL\*PLUS

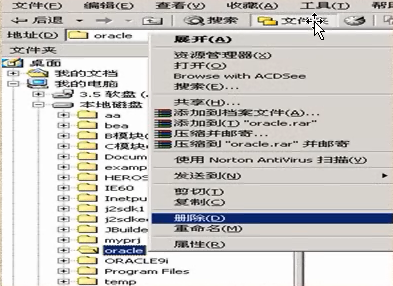
用户名：scott

密码：自己指定

主机字符串：实例名称,可以为空

1. cmd中执行 命令：sqlplus 用户名/密码,如果是SYS，必须以DBA身份登录,写法是：sqlplus sys/test as sysdba
2. 在运行中执行sqlplusw

## Oracle管理工具介绍



* SQL\*PLUS

SQL\*PLUS是Oracle自带的工具软件，主要执行SQL语句，pl/sql块。

步骤：

1. 开始--程序--Oracle目录--application development--SQL\*PLUS
2. 在运行栏输入：sqlplusw
3. 开始--程序--Oracle目录--application development--SQL\*PLUS worksheet

输入用户名密码、主机字符串(即实例名称)

* SQL\*PLUS DOS工具

SQL\*PLUS DOS下操作Oracle的工具，其功能和SQL\*PLUS相似。

1. 在运行栏中输入sqlplus
2. 在Oracle主目录\oraXX\bin\执行sqlplus.exe

* Oracle企业管理器(oem oracle enterprise manager)

开始--程序--Oracle目录--oracle oraclehomeXX--enterprise manager console即可启动一个图形界面环境。

* pl/sql developer

属于第三方软件，主要用于开发、测试，优化oracle pl/sql的存储过程，如触发器。需单独安装。

## 查看Oracle

### 查看当前的Oracle实例名称和主机名称：

select instance\_name,host\_name from v$instance;

### 查看用户帐户及状态

select username,account\_status from dba\_users;

## Oracle用户解锁

Oracle数据库自带了许多用户，例如system、sys和scott等，但是默认情况下，Oracle只对其中5个用户进行了解锁，这5个用户分别是：sys、system、DBSUNMP、sysman和mgmt\_view，而其它用户则处于锁定状态。对用户帐户解锁的语法为：

### 对scott用户解锁

alter user scott account unlock;

### 更改scott用户的密码：

alter user scott identified by tiger;

### 锁定scott用户：

alter user scott account lock;

## 使用OEM

Oracle Enterprise Manager(OEM)提供了基本Web界面的、可用于管理单个Oracle数据库的工具。由于OEM采用基于Web的应用，它对数据库的访问也采用HTTP/HTTPS协议，即使用3层结构访问Oracle数据库系统。

在成功安装Oracle后，OEM也被安装完毕。

地址：<https://localhost:1158/em>

第一次访问时，可能会出现证书错误的信息，选择继续访问。进入后可以安装证书。

## SQL/PLUS连接数据库

### SQL\*Plus工具

SQL\*Plus工具主要用于数据查询和数据处理。利用SQL\*Plus可以将

SQL和Oracle专有的PL/SQL结合起来进行数据查询和处理。

SQL\*Plus工具可以实现以下功能：

* 连接数据库，完成数据库管理。
* 对数据表可以执行插入、修改、删除、查询操作、以及执行SQL、PL/SQL块。
* 查询结果的格式化、运算处理、保存、打印以及输出Web格式。
* 显示任何一个表的字段定义，并与终端用户交互。
* 运行存储在数据库中的子程序或包。

启动/停止数据库实例。

### 连接方式

1. 开始--程序--Oracle目录--application development--SQL\*PLUS worksheet

输入用户名密码、主机字符串(即实例名称)

1. 启动SQL\*Plus，连接到默认数据库

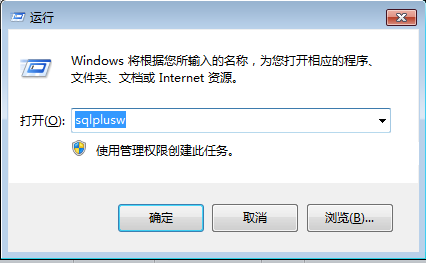
开始--程序--Oracle目录--application development--SQL\*PLUS

输入用户名、密码。将连接到默认的数据库。

可以使用select name from v$databases;来查看使用的实例。

1. 从命令行连接数据库

在运行栏输入：sqlplusw



或



@orcl表示连接到orcl实例，如果不写将连接琶默认的实例。

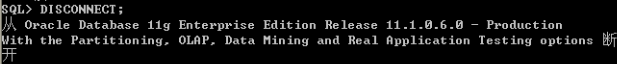
1. 使用SQL\*Plus命令连接与断开数据库
   * + 连接：

connect也可以简写成为conn



* + - 断开连接

### 连接远程数据库



* 方法一

首先配置TNS，方法如下：

TNS连接远程计算机的设置。SYNO为远程计算机的SID

PMSYNO.BJ.COM =

(DESCRIPTION =

(ADDRESS\_LIST =

(ADDRESS =

(PROTOCOL = TCP)

(HOST = 192.168.0.203)

(PORT = 1521)

)

)

(CONNECT\_DATA =

(SERVER = DEDICATED)

(SID = SYNO)

)

)

然后建立连接数据库语句：

CREATE DATABASE LINK link\_name(任意启名)   
CONNECT TO username   
IDENTIFIED BY password   
USING sql\_string;

例如：CREATE DATABASE LINK pm\_syno CONNECT TO pmexp IDENTIFED BY pmexp USING syno(前面TNS配好的那个)；

* 方法二(推荐)

本地：sqlplus username/password@servicename

客户端：

使用Net Configuration Assitant的工具。Net Configuration Assitant共提供了4个配置功能：

监听程序配置，

命名方法配置，

本地网络服务名配置

目录使用配置。

配置网络服务名,再使用sqlplus username/password@servicename连接

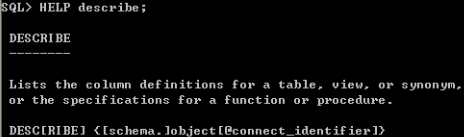
## SQL\*PLUS常用命令

注意：命令使用分号(;)表示结束并执行

#### 常用命令

|  |  |
| --- | --- |
| **命令** | **说明** |
| HELP [topic] | 查看命令的使用方法，topic表示需要查看的命令名称。  例如：HELP DESC |
| HOST | 使用该命令可以从SQL\*Plus环境切换到操作系统环境，  以便执行操作系统命令 |
| HOST 操作系统命令 | 执行操作系统命令，例如：HOST notepad.exe |
| CLEAR SCR[EEN] | 清除屏幕内容 |
| SHOW [ALL | USER | SGA  | ERRORS | REL[EASE]  | PARAMETERS ] | 查看SQL\*Plus的所有系统变量值信息、当前是哪个用户  在使用SQL\*Plus、显示SGA大小、查看错误信息、数据  库版本信息、系统初始化参数信息 |
| DESC[RIBE] | 查看对象的结构，这里的对象可以是表、视图、存储过  程、函数和包等。例如：DESC dual |

##### HELP命令



##### HOST命令

* 不带参数的HOST命令，进入Oracle目录下。



* 带有参数的HOST命令



##### CLEAR SCREEN命令



##### SHOW命令



##### DESCRIBE命令

语法：

**DESC[RIBE] { [ schema. ] object [ @connect\_identifier ] }**

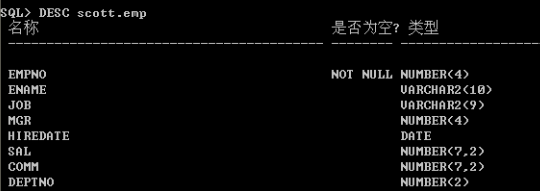
语法说明如下：

DESC[RIBE]：DESCRIBE可以简写为DESC。

schema：指定对象所属的用户名，或者所属的用户模式名称。

object：表示对象的名称，如表名或视图名等。

@connect\_identifier ： 表示数据库连接字符串。



#### 连接命令

* **conn[ect]**

**用法:**

conn 用户名/密码@网络服务名 [as sysdba/sysoper]

**说明：**

当特权用户身份连接时，必须带上as sysdba或是as sysoper

* **disc[onnect]**

**用法:**

disc

**说明：**

用于断开与当前数据库的连接

* **passw[ord]**

**用法:**

passw，然后输入旧密码、新密码

**说明：**

用于修改用户的密码。如果想要修改其它用户的密码，需要用sys/system登陆。

* **show user**

显示当前用户名

* **exit**

提交未保存数据，并断开连接，退出。

#### 编辑SQL语句

在SQL\*Plus中输入SQL语句时，一旦执行该语句，则SQL\*Plus会将该语句保存到缓冲区中，该语句不再获得焦点，也就不能再接受编辑操作。如果还需要对缓冲区中的语句进行编辑操作，则可以使用SQL\*Plus语句实现。

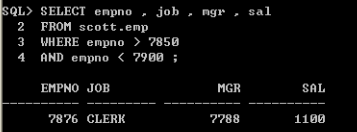
Oracle数据库提供了一些SQL\*Plus语句，这些语句由相应的命令实现，例如APPEND（可以简写为A）命令。如下表所示:

##### 命令

|  |  |
| --- | --- |
| **命令** | **说明** |
| A[PPEND] text | 将text附加到当前行之后 |
| C[HANGE] /old/new | 将当前行中的old替换为new.如果只传递一个old参数，将表示替换为空 |
| CL[EAR] BUFF[ER] | 清除缓存区中的所有行 |
| I[NPUT] text | 插入指定的文本text |
| DEL m n | 删除从第m行到第n行之间的命令行，如果只带有m参数，表示从m行开始删除到命令行的最后。不带参数表示只删除当前行。 |
| L[IST] n | 列出第n行，如果不带参数表示显出缓冲区中的所有内容。 |
| R[UN]或 / | 显示缓冲区中保存的语句，并运行这些语句 |
| n | 将第n行作为当前行 |
| n text | 使用text文本替代第n行信息 |
| 0 text | 在第一行之前插入text文本 |

##### 实例：

1. 输入测试SQL语句



1. 添加一列，即在第1行中操作

输入1

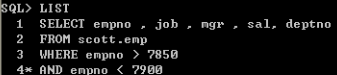


添加一列，使用append命令



dempno列名写错了，使用change命令修改

查看缓冲区中的内容

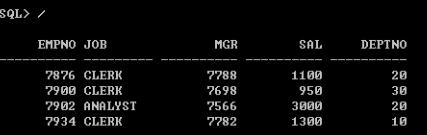


删除第4行



再次查看

运行



#### save命令

* 概述

使用SAVE命令可以将当前缓冲区的内容保存到文件中，这样，即使缓冲区中的内容被覆盖，也保留有前面的执行语句。SAVE命令的语法如下：

|  |
| --- |
| SAV[E] [ FILE ] file\_name [ CRE[ATE] | REP[LACE] | APP[END] ] |

语法说明如下：

file\_name ：表示将SQL\*Plus缓冲区的内容保存到由file\_name指定的文件中。可以使用完整路径，也可以只使用一个文件名称。

CREATE ：表示创建一个file\_name文件，并将缓冲区中的内容保存到该文件。该选项为默认值。

APPEND：如果file\_name文件已经存在，则将缓冲区中的内容，追加到file\_name文件的内容之后；如果该文件不存在，则创建该文件。

REPLACE：如果file\_name文件已经存在，则覆盖file\_name文件的内容；如果该文件不存在，则创建该文件。

* 【示例】：

默认路径为Oracle安装目录BIN目录下



使用路径

再次执行，默认参数(create)



使用replace参数



使用append参数



#### GET命令

* 概述

使用GET命令可以读取文件内容到缓冲区。

使用GET命令的语法如下：

|  |
| --- |
| GET [ FILE ] file\_name [ LIST | NOLIST ] |

语法说明如下：

file\_name ：表示一个指定文件，将该文件的内容读入SQL\*Plus缓冲区中。

LIST：列出缓冲区中的语句。默认

NOLIST：不列出缓冲区中的语句。

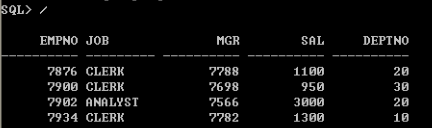
* 【示例】

默认列出了内容。使用LIST参数



使用NOLIST参数

读入缓冲区后，就可以使用run或/来执行缓冲区中的内容。



#### start或@命令

* 概述

get命令需要分二步才能运行，一是get到缓冲区中，二才运行。

START命令可以读取文件中的内容到缓冲区中，然后在SQL\*Plus中运行这些内容。START命令的语法如下：

|  |
| --- |
| STA[RT] { url | file\_name } |

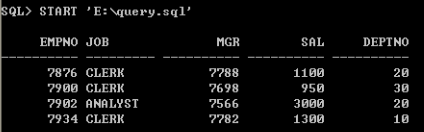
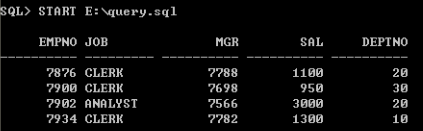
语法说明如下：

url：用来指定一个URL地址，例如http://host.domain/script.sql。

file\_name ：指定一个文件。该命令将file\_name文件的内容读入 SQL\*Plus缓冲区中，然后运行缓冲区中的内容。

* 【示例】

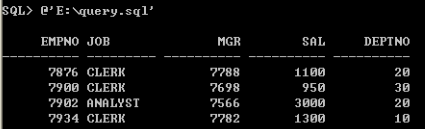
注意有个’符号



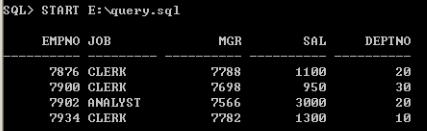
不使用’符号也可以使用

使用@来代替start

注意有个’符号



不使用也可以



如果某些SQL语句使用比较频繁、内容比较复杂，就可以保存在某个文件中，然后使用这个命令来加载运行。

#### EDIT命令

* 概述

使用EDIT命令，可以将SQL\*Plus缓冲区的内容复制到一个名为afiedt.buf的文件中，然后启动操作系统中默认的编辑器打开这个文件，并且文件内容能够进行编辑。在Windows操作系统中，默认的编辑器是Notepad（记事本）。

语法:

|  |
| --- |
| ED[IT] [ file\_name ] |

其中，file\_name默认为afiedt.buf，也可以指定一个其他的文件。

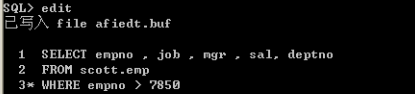
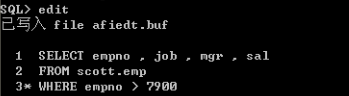
* 【示例】

使用默认

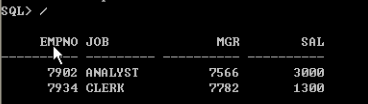


直接关闭记事本

可以对打开的记事本中的内容进行编辑。修改保存



执行



也可以使用sql>edit d:\test.sql来编辑

#### SPOOL命令

* 概述

使用SPOOL命令实现将SQL\*Plus中的输出结果复制到一个指定的文件中，或者把查询结果发送到打印机中，直到使用SPOOL OFF命令为止。

SPOOL命令的语法如下：

|  |
| --- |
| SPO[OL] [ file\_name [ CRE[ATE] | REP[LACE] | APP[END]] | OFF | OUT ] |

file\_name 指定一个操作系统文件。

CREATE 创建一个指定的file\_name文件。

REPLACE 如果指定的文件已经存在，则替换该文件。

APPEND 将内容附加到一个已经存在的文件中。

OFF 停止将SQL\*Plus中的输出结果复制到file\_name文件中，并关闭该文件。

OUT 启动该功能，将SQL\*Plus中的输出结果复制到file\_name文件。

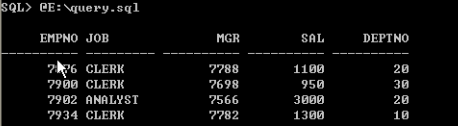
* 【示例】

将屏幕的输出结果追加到E:\query.sql文件中



执行E:\query.sql文件

执行关闭输出



输出结果将会被保存到E:\query.sql文件中。可以打开E:\query.sql查看。



一般用于将结果进行统计，对结果进行分析。

#### 临时变量

* 概述

在Oracle数据库中，可以使用变量来编写通用的SQL语句，在运行SQL语句时，为变量输入值，就会在SQL语句中将变量替换成这些值。

临时变量只在使用它的SQL语句中有效，变量值不能保留。临时变量也被称为替换变量。在SQL语句中，如果在某个变量前面使用了&符号，那么就表示该变量是一个临时变量。执行SQL语句时，系统会提示用户为该变量提供一个具体的数据。

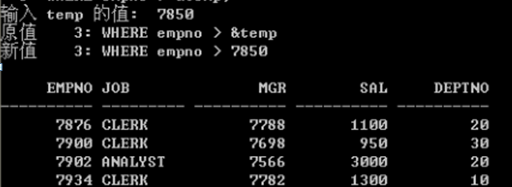
在SQL语句中，如果希望重新使用某个变量并且不希望重新提示输入值，那么可以使用&&符号来定义临时变量。

* 【示例】

使用&来表示临时变量，变量名为temp



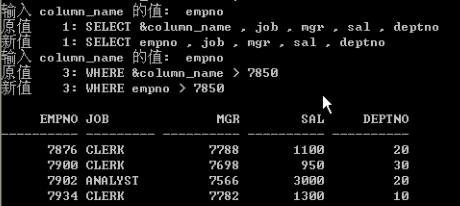
执行



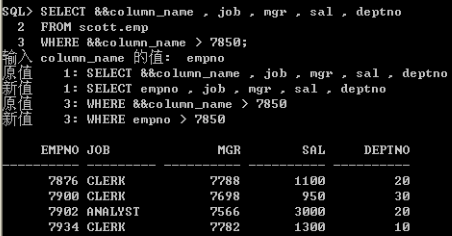
使用列名临时变量



执行



使用&&表示使用前一个临时变量



可以发现，只需要输入一次。

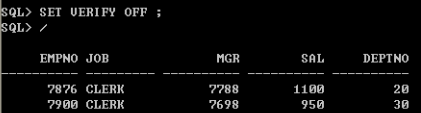
如果不想输出原值和新值信息，在使用临时变量时，还可以使用SET VERIFY命令和SET DEFINE命令，

其中：

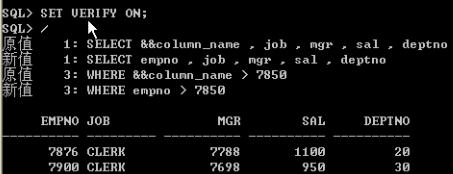
SET VERIFY [ ON | OFF ] 用来指定是否输出原值和新值信息。

SET DEF[INE] 用于指定一个除字符&之外的字符，作为定义变量的字符。、

* 关闭显示原值和新值
* 显示原值和新值



#### 已定义变量



* 概述

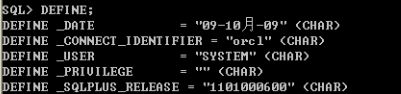
已定义变量是指具有明确定义的变量，该变量的值会一直保留到被显式地删除、重定义或退出SQL\*Plus为止。

在SQL语句中，可以在使用变量之前对变量进行定义，然后在同一个SQL语句中可以多次使用这个变量。定义变量，可以使用DEFINE或ACCEPT命令；删除变量，可以使用UNDEFINE命令。

* 使用DEFINE命令

|  |  |
| --- | --- |
| **命令** | **说明** |
| DEF[INE] | 显示所有的已定义变量 |
| DEF[INE] variable | 显示指定变量的名称、值和其数据类型 |
| DEF[INE] variable = value | 创建一个CHAR类型的用户变量，并且为该变量赋初始值 |

Oracle中，已经定义了一些已定义变量，可以使用DEFINE来查看



* 使用ACCEPT命令

使用ACCEPT命令也可以定义变量，并且定制一个用户提示，用于提示用户输入指定变量的数据。在定义变量时，可以明确地指定该变量是NUMBER还是DATE等数据类型。为了安全性的原因，还可以将用户输入的信息隐藏起来。

ACCEPT命令的语法如下：

|  |
| --- |
| ACC[EPT] variable [ data\_type ] [ FOR[MAT] format ] [ DEF[AULT]  default ] [ PROMPT text | NOPR[OMPT] ] [ HIDE ] |

语法说明如下：

variable 用于一个指定接收值的变量。如果该名称的变量不存在，那么SQL\*Plus自动创建该变量。

data\_type 指定变量的数据类型，可以使用的类型有CHAR、NUM[BER]、DATE、BINARY\_FLOAT和BINARY\_DOUBLE。默认的数据类型为CHAR。而DATE类型的变量实际上也是以CHAR变量存储的。

FORMAT 指定变量的格式，包括A15（15个字符）、9999（一个4位数字）和DD-MON-YYYY（日期）。A用于表示字符，9用于表示数字

DEFAULT 用来为变量指定一个默认值。

PROMPT 用于表示在用户输入数据之前显示的文本消息。

HIDE 表示隐藏用户为变量输入的值。

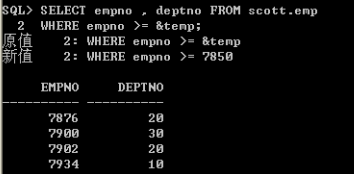
* 【示例】

定义一个变量

查看变量



执行语句



使用ACCEPT定义一个变量为test，变量类型为NUMBER，格式为4位数字，提示信息为“你好，输入一个EMPNO值”，并隐藏输入的值



执行并输入值，输入时不会显示

再输入语句



#### COLUMN命令



* 概述

COLUMN命令可以对列的显示效果，包括列标题和列数据，进行格式化。该命令的语法如下：

|  |
| --- |
| COL[UMN] [ { column\_name | ALI[AS] alias } [ options ] ] |

语法说明如下：

column\_name 指定列名。

ALIAS alias 指定要格式化的列的别名。

options 指定用于格式化列或别名的一个或者多个选项。

* OPTION选项

|  |  |
| --- | --- |
| **选项** | **说明** |
| FOR[MAT] format | 将列或列名的显示格式设置为由format字符串指定的格式，format可以使用的格式如下表 |
| HEA[DING] text | 设置由text字符串指定的列标题 |
| JUS[TIFY] [ { LEFT |  CENTER | RIGHT } ] | 将列的输出信息设置为左对齐、居中对齐或右对齐 |
| WRA[PPED] | 在输出结果中将一个字符串的末尾换行显示。该选项可  能导致单个单词跨越多行 |
| WOR[D\_WRAPPED] | 与WRAPPED选项类似，但是单个单词不会跨越多行 |
| CLE[AR] | 清除列的格式化 |
| TRUNCATED | 删除第一行的字符串，具体使用时主要用来截断字符串 |
| NULL text | 指定列为空值时显示的内容 |
| PRINT | 显示列标题 |
| NOPRINT | 隐藏列标题 |

* format格式元素

|  |  |  |
| --- | --- | --- |
| **元素** | **说明** | **举例** |
| An | 为[VAR]CHAR类型的列内容设置宽度。如果内容  超过指定的宽度，则内容自动换行 | A5 |
| 9 | 设置NUMBER列的显示格式 | 999 999 |
| $ | 浮动的货币符号 | $9999 |
| L | 本地货币符号 | L9999 |
| . | 小数点位置 | 9999.99 |
| , | 千位分隔符 | 9,999 |

* 【示例】

输入SQL语句，查看效果



效果



使用COLUMN命令格式化

设置empno标题为员工编号，四位数字等格式



设置完成后

再次查询



#### 设置pagesize和linesize

* PAGESIZE

使用PAGESIZE命令，可以设置每一页的大小，从而控制每一页显示的数据量。PAGESIZE命令的语法如下：

|  |
| --- |
| SET PAGESIZE n |

其中，参数n表示每一页大小的正整数，最大值可以为50,000，默认值为14。

页并不是仅仅由输出的数据行构成的，而是由SQL\*Plus显示到屏幕上的所有输出结果构成，包括标题和空行等。

* LINESIZE

使用LINESIZE命令可以设置一行数据可以容纳的字符数量，默认数量为80。如果LINESIZE的值比较小，那么表中一行数据，有可能在屏幕上需要分多行显示；如果LINESIZE的值设置大一些，一行数据就可以在屏幕的一行中进行显示了。

LINESIZE命令的语法如下：

|  |
| --- |
| SET LINESIZE n |

其中，n表示屏幕上一行数据可以容纳的字符数量，有效范围是1 – 32767。

#### 创建简单报表

* 报表

所谓报表就是用表格、图表等格式来动态显示数据。计算机上的报表的主要特点是数据动态化，格式多样化，并且实现报表数据和报表格式的完全分离，用户可以只修改数据，或者只修改格式。

* 为报表设置页眉和页脚

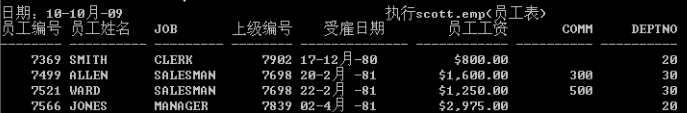
|  |  |
| --- | --- |
| **命令** | **说明** |
| TTI[TLE] [ printspec [ text | variable ] ...] |  [ OFF | ON ] | 指定出现在报表中每一个页面顶  端的页眉。其中printspec的可选  值有LEFT、CENTER、RIGHT、  BOLD、FORMAT text、COL n、  S[KIP] [n]和TAB n。ON选项表示  启用设置；OFF选项表示取消设  置。还可以带有用户变量（内容  由系统来维护），如SQL.PNO页  号、SQL.LNO行号等 |
| BTI[TLE] [ printspec [ text | variable ] ...] |  [ OFF | ON ] | 指定出现在报表中每一个页面底  端的页脚 |

* 【示例】

使用ttile设置页眉



\_DATE为系统已定义变量。重新执行查询scott.emp表



使用btitle设置页脚

|  |
| --- |
| BTITLE center ‘谢谢使用报表’ |

#### 计算小计

* BREAK命令和COMPUTE命令

在执行查询操作或创建报表时，可以使用BREAK和COMPUTE命令，用来对输出的结果进行统计计算。其中使用BREAK命令，可以让SQL\*Plus根据列值的范围分隔输出结果，可以使得重复的列值不进行显示;COMPUTE子句可以让SQL\*Plus计算一列的值，并在最后输出显示。用法如下：

|  |
| --- |
| BRE[AK] [ ON column\_name ] SKIP n  COMP[UTE] function LABEL label OF column\_name ON break\_column\_name |

语法说明如下：

column\_name 表示对哪一列执行操作。

SKIP n 表示在指定列的值变化之前插入n个空行。

function 表示执行的操作，例如SUM（求和）、MAXIMUM（最大值）、MINIMUM（最小值）、AVG（平均值）、COUNT（非空值的列数）、NUMBER（行数）、VARIANCE（方差）以及STD（均方差）等。

break\_column\_name要与BRE[AK]中的column\_name一致

LABEL 指定显示结果时的文本信息。

【示例】：对emp的工资进行统计



|  |
| --- |
| break on deptno--表示根据deptno对下面的数据进行分组  compute sum of sal on deptno--根据deptno进行分组，计算分组后的和。  select empno,ename,mar,hiredate,sal,deptno from  scott.emp order by deptno; |

## 表空间

表空间是Oracle数据库中最大的逻辑存储结构，它与操作系统中的数据文件相对应，用于存储数据库中用户创建的所有内容。

在Oracle中，表空间可以分为基本表空间、临时表空间、大文件表空间、非标准数据块表空间和撤销表空间等。

其中，基本表空间一般指用户使用的永久性表空间，用于存储用户的永久性数据；临时表空间用于存储排序或汇总过程中产生的临时数据；大文件表空间用于存储大型数据（例如LOB）；非标准数据块表空间用于在一个数据库实例中创建数据块大小不同的表空间；撤销表空间用于存储事务的撤销数据，在数据恢复时使用。

### 创建表空间

* 语法

|  |
| --- |
| CREATE [ TEMPORARY | UNDO ] TABLESPACE tablespace\_name  [ DATAFILE | TEMPFILE 'file\_name' SIZE size K | M [ REUSE ]  [ AUTOEXTEND OFF | ON  [ NEXT number K | M MAXSIZE UNLIMITED | number K | M ]  ][ , …]]  [ MININUM EXTENT number K | M ]  [ BLOCKSIZE number K]  [ ONLINE | OFFLINE ]  [ LOGGING | NOLOGGING ]  [ FORCE LOGGING ]  [ DEFAULT STORAGE storage ]  [ COMPRESS | NOCOMPRESS ]  [ PERMANENT | TEMPORARY ]  [ EXTENT MANAGEMENT DICTIONARY | LOCAL  [ AUTOALLOCATE | UNIFORM SIZE number K | M ] ]  [ SEGMENT SPACE MANAGEMENT AUTO | MANUAL ]; |

* 说明

TEMPORARY | UNDO 指定表空间的类型。TEMPORARY表示创建临时表空间；UNDO表示创建撤销表空间；不指定类型，则表示创建的表空间为永久性表空间。

tablespace\_name 指定新表空间的名称。

DATAFILE | TEMPFILE 'file\_name‘ 指定与表空间相关联的数据文件。一般使用DATAFILE，如果是创建临时表空间，则需要使用TEMPFILE；file\_name指定文件名与路径。可以为一个表空间指定多个数据文件。

SIZE size 指定数据文件的大小。

REUSE 如果指定的数据文件已经存在，则使用REUSE关键字可以清除并重新创建该数据文件。如果文件已存在，但是又没有指定REUSE关键字，则创建表空间时会报错。

AUTOEXTEND OFF | ON 指定数据文件是否自动扩展。OFF表示不自动扩展；ON表示自动扩展。默认情况下为OFF。

NEXT number 如果指定数据文件为自动扩展，则NEXT子句用于指定数据文件每次扩展的大小。

MAXSIZE UNLIMITED | number 如果指定数据文件为自动扩展，则MAXSIZE子句用于指定数据文件的最大大小。如果指定UNLIMITED，则表示大小无限制，默认为此选项。

MININUM EXTENT number 表空间中的盘区可以分配到的最小的尺寸。

BLOCKSIZE number 如果创建的表空间需要另外设置其数据块大小，而不是采用初始化参数db\_block\_size指定的数据块大小，则可以使用此子句进行设置。此子句仅适用于永久性表空间。

ONLINE | OFFLINE 指定表空间的状态为在线（ONLINE）或离线（OFFLINE）。如果为ONLINE，则表空间可以使用；如果为OFFLINE，则表空间不可使用。默认为ONLINE。

LOGGING | NOLOGGING 指定存储在表空间中的数据库对象的任何操作是否产生日志。LOGGING表示产生；NOLOGGING表示不产生。默认为LOGGING。

FORCE LOGGING 此选项用于强制表空间中的数据库对象的任何操作都产生日志，将忽略LOGGING或NOLOGGING子句。

DEFAULT STORAGE storage 指定保存在表空间中的数据库对象的默认存储参数。当然，数据库对象也可以指定自己的存储参数。

COMPRESS | NOCOMPRESS 指定是否压缩数据段中的数据。COMPRESS表示压缩；NOCOMPRESS表示不压缩。数据压缩发生在数据块层次中，以便压缩数据块内的行，消除列中的重复值。默认为COMPRESS。

PERMANENT | TEMPORARY 指定表空间中数据对象的保存形式。PERMANENT表示持久保存；TEMPORARY表示临时保存。

EXTENT MANAGEMENT DICTIONARY | LOCAL 指定表空间的管理方式。

DICTIONARY表示采用数据字典的形式管理；LOCAL表示采用本地化管理形式管理。默认为LOCAL。

AUTOALLOCATE | UNIFORM SIZE number 指定表空间中的盘区大小。AUTOALLOCATE表示盘区大小由Oracle自动分配，此时不能指定大小；UNIFORM SIZE number表示表空间中的所有盘区大小相同，都为指定值。默认为AUTOALLOCATE。

SEGMENT SPACE MANAGEMENT AUTO | MANUAL 指定表空间中段的管理方式。AUTO表示自动管理方式；MANUAL表示手动管理方式。默认为AUTO。

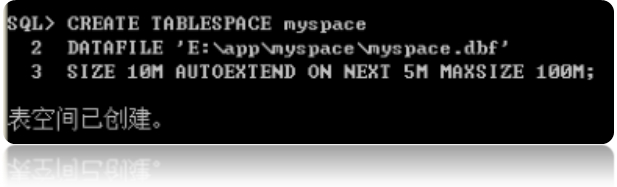
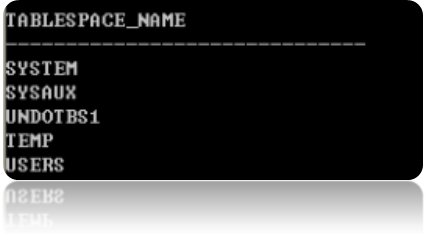
* 【示例】

查看Oracle数据库中已经存在的表空间

结果：



创建表空间，名称为myspace,指定了一个数据文件，存放在E盘下，文件大小为10M，自动增长，一次增长为5M,最大增长为100M



再次查询



或者

|  |
| --- |
| SQL>;create tablespace ts\_data default storage（initial 10M next 10M maxextents 5000 pctincrease 0） datafile 'path/data\_01.dbf' size 500M;  SQL>;create tablespace ts\_index default storage（initial 5M next 5M maxextents 5000 pctincrease 0） datafile 'path/index\_01.dbf' size 500M; |

### 表空间状态属性

表空间的状态属性主要有在线（ONLINE）、离线（OFFLINE）、只读（READ ONLY）和读写（READ WRITE）这4种，其中只读与读写状态属于在线状态的特殊情况。通过设置表空间的状态属性，可以对表空间的使用进行管理。

* 在线

当表空间的状态为ONLINE时，才允许访问该表空间中的数据。

如果表空间不是ONLINE状态的，可以使用ALTER TABLESPACE语句将其状态修改为ONLINE，语句形式如下：

ALTER TABLESPACE tablespace\_name ONLINE;

* 离线

当表空间的状态为OFFLINE时，不允许访问该表空间中的数据。例如向表空间中创建表或者读取表空间中的表的数据等操作都将无法进行。这时可以对表空间进行脱机备份；也可以对应用程序进行升级和维护等。

如果表空间不是OFFLINE状态的，可以使用ALTER TABLESPACE语句将其状态修改为OFFLINE，语句形式如下：

ALTER TABLESPACE tablespace\_name OFFLINE parameter;

其中，parameter表示将表空间切换为OFFLINE状态时可以使用的参数。主要可以应用如下几个参数：

* NORMAL

以正常方式切换到离线状态。执行检查点，将内容写入文件，关闭数据文件。默认方式

* TEMPORARY

以临时方式切换到离线状态，执行检查点时Oracle不检查数据文件是不是可以使用

* IMMEDIATE

立即切换到离线状态，不执行检查点

* FOR RECOVER

以恢复方式切换到离线状态。

* 只读

当表空间的状态为READ ONLY时，虽然可以访问表空间中的数据，但访问仅仅限于阅读，而不能进行任何更新或删除操作，目的是为了保证表空间的数据安全。

如果表空间不是READ ONLY状态的，可以使用ALTER TABLESPACE语句将其状态修改为READ ONLY，语句形式如下：

ALTER TABLESPACE tablespace\_name READ ONLY;

不过，将表空间的状态修改为READ ONLY之前，需要注意如下事项：

* 表空间必须处于ONLINE状态
* 表空间不能包含任何事务的回退段
* 表空间不能正处于在线数据库备份期间
* 读写

当表空间的状态为READ WRITE时，可以对表空间进行正常访问，包括对表空间中的数据进行查询、更新和删除等操作。

如果表空间不是READ WRITE状态的，可以使用ALTER TABLESPACE语句将其状态修改为READ WRITE，语句形式如下：

ALTER TABLESPACE tablespace\_name READ WRITE;

修改表空间的状态为READ WRITE，也需要保证表空间处于ONLINE状态。

* 【示例】：、

查看表空间及状态



修改为离线状态



修改为在线状态



修改为只读状态



修改为读写状态



### 删除表空间

删除表空间及其包含的所有数据对象

|  |
| --- |
| SQL>drop tablespace data including contents; |

相关系统表：

user(dba)\_tablespaces

### 增加表空间尺寸

假定表空间ts\_data由path/data\_01.dbf和path/data\_02.dbf（500M）组成

增加一个数据文件：

|  |
| --- |
| SQL>alter tablespace ts\_data add datafile 'path/data\_03.dbf' size 500M; |

扩大原有文件大小：

|  |
| --- |
| SQL>alter database datafile 'path/data\_01.dbf' resize 1000M; |

### 查看剩余空间

|  |
| --- |
| SQL>select tablespace\_name,sum(bytes),max(bytes) from dba\_free\_space group by tablespace\_name; |

注意:空闲数据块总和sum(bytes)够用并不意味每个空闲块都满足分配需要，所以当表空间不够分配扩展块的时候，还要查看最大空闲数据块max(bytes)的大小。

### 合并空闲块

如果表空间上的数据对象经常发生类似drop-create的变动，加之未采用统一的扩展块尺寸，使那些采用较大扩展块的数据对象不能利用较小的空间碎片，造成空间浪费。可通过将较小的空闲块合并成较大的空闲块的方法，减少空间浪费。

|  |
| --- |
| SQL>alter tablespace tbsdata coalesce; |

### 设置默认表空间

在Oracle中，用户的默认永久性表空间为system，默认临时表空间为temp。如果所有用户都使用默认的表空间，无疑会增加system与temp表空间的竞争性。

Oracle允许使用自定义的表空间作为默认永久性表空间，使用自定义临时表空间作为默认临时表空间。设置默认表空间的语法如下：

|  |
| --- |
| ALTER DATABASE DEFAULT [ TEMPORARY ] TABLESPACE tablespace\_name; |

如果使用TEMPORARY关键字，则表示设置默认临时表空间；如果不使用该关键字，则表示设置默认永久性表空间。

* 【示例】

查询当前默认表空间

|  |
| --- |
| SQL> SELECT default\_tablespace FROM user\_users; |

查看基本空间和临时表空间

|  |
| --- |
| SELECT property\_name , property\_value  FROM database\_properties WHERE property\_name  IN ('DEFAULT\_PERMANENT\_TABLESPACE' , 'DEFAULT\_TEMP\_TABLESPACE'); |

新用户的默认表空间为USERS

创建新表空间



修改默认基本表空间



修改临时表空间



再次查看

|  |
| --- |
| SELECT property\_name , property\_value  FROM database\_properties WHERE property\_name  IN ('DEFAULT\_PERMANENT\_TABLESPACE' , 'DEFAULT\_TEMP\_TABLESPACE');  SQL> SELECT default\_tablespace FROM user\_users; |

### 撤销表空间

为了实现对数据回退、恢复、事务回滚以及撤销等操作，Oracle数据库提供了一部分存储空间，专门保存撤销记录，将修改前的数据保存到该空间中，所以这部分空间被称为撤销表空间。多个撤销表空间可以存在于一个数据库中，但是在任何给定的时间内只有一个撤销表空间是可以获得的。

* 创建撤销表空间

创建撤销表空间需要使用CREATE UNDO TABLESPACE语句 。

例如，使用CREATE UNDO TABLESPACE语句，创建一个名称为

UNDOTBS的撤销表空间，如下：

|  |
| --- |
| SQL> create undo tablespace undotbs  2 datafile ‘e:\app\myspace\undo01.dbf' size 20m  3 autoextend on;  表空间已创建。 |

* 修改撤销表空间的数据文件

由于撤销表空间主要由Oracle系统自动管理，所以对撤销表空间的数据文件的修改也主要限于以下几种形式：

1. 为撤销表空间添加新的数据文件

|  |
| --- |
| alter tablespace undotbs  add datafile ' e:\app\myspace\undo02.dbf' size 10m; |

1. 修改撤销表空间的数据文件大小

|  |
| --- |
| alter database datafile ' e:\app\myspace\undo02.dbf' resize 15m; |

1. 设置撤销表空间的数据文件的状态为ONLINE或OFFLINE

|  |
| --- |
| alter tablespace undotbs offline; |

* 切换撤销表空间

一个数据库中可以有多个撤销表空间，但数据库一次只能使用一个撤销表空间。默认情况下，数据库使用的是系统自动创建的undotbs1撤销表空间。如果要将数据库使用的撤销表空间切换成其他表空间，使用ALTER SYSTEM语句修改参数undo\_tablespace的值即可。切换撤销表空间后，数据库中新事务的撤销数据将保存在新的撤销表空间中。

如：

|  |
| --- |
| alter system set undo\_tablespace = undotbs02; |

* 修改撤销记录保留的时间

在自动撤销记录管理方式中，可以指定撤销信息在提交之后需要保留的时间，以防止在长时间的查询过程中出现snapshot too old错误。

在自动撤销管理方式下，DBA使用UNDO\_RETENTION参数，指定撤销记录的保留时间。由于UNDO\_RETENTION参数是一个动态参数，在Oracle实例的运行中，可以通过ALTER SYSTEM SET UNDO\_RETENTION语句，来修改撤销记录保留的时间。

撤销记录保留时间的单位是秒，默认值为900，即15分钟。

例如，将撤销记录的保留时间修改为10分钟，如下：

|  |
| --- |
| SQL> alter system set undo\_retention = 600;  SQL> show parameter undo;  NAME TYPE VALUE  ------------------------- --------- ----------  undo\_management string AUTO  undo\_retention integer 600  undo\_tablespace string UNDOTBS |

* 删除撤销表空间

删除撤销表空间同样需要使用DROP TABLESPACE语句，但删除的前提是该撤销表空间此时没有被数据库使用。如果需要删除正在被使用的撤销表空间，则应该先进行撤销表空间的切换操作。

如：

|  |
| --- |
| drop tablespace undotbs02 including contents and datafiles; |

### 查看当前用户的缺省表空间

select username,default\_tablespace from user\_users;

## 用户管理

在用户连接Oracle数据库时，Oracle为了防止非授权用户的访问，需要用户提供账号和口令。在创建用户时，对用户可以使用的安全参数进行限制，从而可以对用户的操作进行一定的规范。这些都是通过对用户进行管理，从而确保数据库的安全。

在创建一个用户时，可以为用户指定表空间、临时表空间和资源文件等，另外，还可以使用PASSWORD EXPIRE子句和ACCOUNT子句，对该用户进行相应的管理。

### 创建用户

创建一个新的数据库用户时，需要使用CREATE USER语句，该语句的

* 语法如下：

|  |
| --- |
| CREATE USER user\_name  IDENTIFIED BY password  [ DEFAULT TABLESPACE default\_tablespace |  TEMPORARY TABLESPACE temp\_tablespace |  PROFILE profile  QUOTA [ integer K | M ] | UNLIMITED ON tablespace  | PASSWORD EXPIRE  | ACCOUNT LOCK | UNLOCK ]; |

* 语法说明
* user\_name 指定要创建的数据库用户的名称。
* password 表示用户的口令。
* DEFAULT TABLESPACE 为用户指定默认表空间。如果用户没有指定该表空间，则系统将使用用户默认的表空间来存储。
* TEMPORARY TABLESPACE 为用户指定临时表空间。如果没有指定用户临时表空间，则系统将使用默认的临时表空间来存储。
* PROFILE 表示用户的资源文件，该资源文件必须在之前已经被创建。
* QUOTA [ integer K | M ] | UNLIMITED ON tablespace 表示用户在表空间中可以使用的空间总大小。
* PASSWORD EXPIRE 强制用户在每一次登录数据库后必须修改口令。
* ACCOUNT LOCK | UNLOCK 表示锁定或者解锁某个用户账号。
* 【示例】

创建表空间



创建用户并指定默认的表空间



### 删除用户

|  |
| --- |
| Drop user test; |

如果该用户下面已经存在表等一些数据库对象。则必须用级联删除

|  |
| --- |
| Drop user test cascade; |

### 修改用户

修改用户时，需要使用ALTER USER语句，该语句的语法如下：

|  |
| --- |
| ALTER USER user\_name  IDENTIFIED BY password  [ DEFAULT TABLESPACE tablespace |  TEMPORARY TABLESPACE tablespace |  PROFILE profile  QUOTA [ integer K | M ]  | UNLIMITED ON tablespace  | PASSWORD EXPIRE  | ACCOUNT LOCK | UNLOCK ]; |

使用方式与创建用户时一致

### 修改口令

修改口令，需要使用ALTER USER … IDENTIFIED BY语句，如下：

|  |
| --- |
| SQL> alter user user1 identified by user1;  用户已更改。 |

其中，BY关键字后面是新口令，该口令可以是任何字符串。

也可以使用GRANT命令修改用户的口令，如下：

|  |
| --- |
| SQL> grant connect to user1 identified by aa;  授权成功。 |

### 使密码失效

提示用户第一次连接的时候需要修改密码，让用户的密码到期

|  |
| --- |
| SQL>ALTER USER test PASSWORD expire ; |

### 修改默认表空间

对用户默认表空间进行修改，需要使用ALTER USER … DEFAULT

TABLESPACE语句。例如：

|  |
| --- |
| SQL> alter user user1  2 default tablespace temp0901;  用户已更改。 |

对用户默认表空间修改后，先前已经创建的表仍然存储在原表空间中。如果创建新表，则将存储在新表空间中。

### 锁定用户

|  |
| --- |
| SQL> alter user user1 account lock;  用户已更改。 |

用户被锁定后，如果再使用该用户连接数据库，将返回错误信息。被锁定的用户必须由数据库管理员解锁后才能够使用。

解锁时使用UNLOCK选项。例如，将用户USER1解锁，如下：

|  |
| --- |
| SQL> alter user user1 account unlock;  用户已更改。 |

### 权限的概述

权限是数据库中执行某种操作的权力，是预先定义好的、执行某种SQL语句或访问其他用户模式对象的能力。创建一个用户，表示该用户具有了连接和操作数据库的资格，但是要对数据库进行实际操作，则该用户还需要具有相应的操作权限。

在Oracle数据库中，根据系统管理方式的不同，可以将权限分为两类：

* 系统权限

指在系统级控制数据库的存取和使用机制。系统级控制决定是否可以连接到数据库，在数据库中可以进行哪些操作等。

* 对象权限

指在模式对象上控制存取和使用的机制。

例如，SELECT ANY TABLE是系统权限，表示可以查看任何表；而SELECT ON table1是对象权限，表示对表table1的查看权限。

1. 系统权限

Oracle提供了多种系统权限，例如创建会话、创建表、创建视图和创建用户等。DBA在创建一个用户时，可以将其中的一些权限授予用户。

系统权限是针对用户来设置的，用户必须被授予相应的系统权限，才可以连接到数据库中进行相应的操作。

在Oracle数据库中，用户SYSTEM和SYS都是数据库管理员，具有DBA所有系统权限，包括SELECT ANY DICTIONARY权限。所以，SYSTEM和SYS可以查询数据字典中以“DBA\_”开头的数据字典视图。



1. 对象权限

在Oracle中，可以授权的数据库对象包括表、视图、序列、存储过程和函数等，对象权限一般是针对用户模式对象的。例如，用户访问模式对象中的表。

对象权限是用户与用户之间对表、视图等模式对象的相互存取权限。例如，以SYSTEM用户登录到数据库，可以查询SCOTT用户模式中的表DEPT。



### 系统权限

|  |  |
| --- | --- |
| **系统权限** | **说明** |
| CREATE SESSION | 连接数据库 |
| CREATE TABLESPACE | 创建表空间 |
| ALTER TABLESPACE | 修改表空间 |
| DROP TABLESPACE | 删除表空间 |
| CREATE USER | 创建用户 |
| ALTER USER | 修改用户 |
| DROP USER | 删除用户 |
| CREATE TABLE | 创建表 |
| CREATE ANY TABLE | 在任何用户模式中创建表 |

|  |  |
| --- | --- |
| DROP ANY TABLE | 删除任何用户模式中的表 |
| **系统权限** | **说明** |
| ALTER ANY TABLE | 修改任何用户模式中的表 |
| SELECT ANY TABLE | 查询任何用户模式中基本表的记录 |
| INSERT ANY TABLE | 向任何用户模式中的表插入记录 |
| UPDATE ANY TABLE | 修改任何用户模式中的表的记录 |
| DELETE ANY TABLE | 删除任何用户模式中的表的记录 |
| CREATE VIEW | 创建视图 |
| CREATE ANY VIEW | 在任何用户模式中创建视图 |
| DROP ANY VIEW | 删除任何用户模式中的视图 |
| CREATE ROLE | 创建角色 |
| ALTER ANY ROLE | 修改任何角色 |

|  |  |
| --- | --- |
| **系统权限** | **说明** |
| GRANT ANY ROLE | 将任何角色授予其他用户 |
| ALTER DATABASE | 修改数据库结构 |
| CREATE PROCEDURE | 创建存储过程 |
| CREATE ANY PROCEDURE | 在任何用户模式中创建存储过程 |
| ALTER ANY PROCEDURE | 修改任何用户模式中的存储过程 |
| DROP ANY RPOCEDURE | 删除任何用户模式中的存储过程 |
| CREATE PROFILE | 创建配置文件 |
| ALTER PROFILE | 修改配置文件 |
| DROP PROFILE | 删除配置文件 |

#### 授予系统权限

向用户授予系统权限需要使用GRANT语句，其语法如下：

|  |
| --- |
| GRANT system\_privilege [ , … ] TO  { user\_name [ , … ] | role\_name [ , … ] | PUBLIC }  [ WITH ADMIN OPTION ] ; |

* 语法说明如下：
* system\_privilege 表示系统权限，例如CREATE TABLE。多个权限之间使用逗号隔开。
* user\_name 被授予权限的用户。可以是多个用户。
* role\_name 也可以将权限授予某些角色。
* PUBLIC 表示Oracle系统的所有用户。
* WITH ADMIN OPTION 如果指定此选项，则被授予权限的用户可以将 该权限再授予其他用户。
* 【示例】

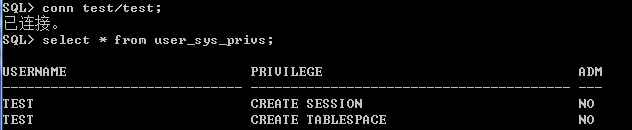


#### 查看用户的系统权限

当前用户具有哪些系统权限，以及当前用户是否有将权限授予其他用户的权力，可以通过数据字典user\_sys\_privs来了解。

数据字典user\_sys\_privs有如下3个字段：

* username 当前用户的用户名。
* privilege 当前用户拥有的系统权限。
* admin\_option 当前用户是否有权力将该权限授予其他用户。其值为YES表示有，为NO表示没有。
* 【示例】



### 对象权限

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **FUNCTION** | **PROCEDURE** | **PACKAGE** | **SEQUENCE** | **TABLE** | **VIEW** |
| ALTER |  |  |  | √ | √ |  |
| DELETE |  |  |  |  | √ | √ |
| EXECUTE | √ | √ | √ |  |  |  |
| INDEX |  |  |  |  | √ |  |
| INSERT |  |  |  |  | √ | √ |
| READ |  |  |  |  |  |  |
| REFERENCE |  |  |  |  | √ |  |
| SELECT |  |  |  | √ | √ | √ |
| UPDATE |  |  |  |  | √ | √ |

#### 授予对象权限

授予对象权限同样需要使用GRANT语句，其语法如下：

|  |
| --- |
| GRANT object\_privilege [ , … ] | ALL [ PRIVILEGES ] ON <schema.>object\_name  TO { user\_name [ , … ] | role\_name [ , … ] | PUBLIC }  [ WITH GRANT OPTION ] ; |

* 语法说明如下
* object\_privilege 表示对象权限。
* ALL [ PRIVILEGES ] 使用ALL关键字，可以授予对象上的所有权限。也可以在ALL关键字后面添加PRIVILEGES。
* schema 用户模式。
* object\_name 对象名称。
* user\_name 被授予权限的用户。可以是多个用户。
* role\_name 也可以将权限授予某些角色。
* PUBLIC 表示Oracle系统的所有用户。
* WITH GRANT OPTION 允许用户将该对象权限授予其他用户。与授予 系统权限的WITH ADMIN OPTION子句相类似。
* 【示例】

授予test用户对scott用户的emp表的查看权限

|  |
| --- |
| GRANT select ON scott.emp TO test ; |

授予test用户对scott用户的emp表的所有权限

|  |
| --- |
| Grant all on scott.emp to test; |

还可以控制到列

|  |
| --- |
| Grant update(ename) on emp to test; 可以控制到列(还有insert) |

### 查看和撤消权限

ORACLE中数据字典视图分为3大类,用前缀区别，分别为：USER，ALL 和 DBA，许多数据字典视图包含相似的信息。

* USER\_\*： 有关用户所拥有的对象信息，即用户自己创建的对象信息
* ALL\_\*： 有关用户可以访问的对象的信息，即用户自己创建的对象的信息加上其他用户创建的对象但该用户有权访问的信息
* DBA\_\*： 有关整个数据库中对象的信息
* 存储权限信息的数据字典视图

|  |  |
| --- | --- |
| **数据字典视图** | **描述** |
| DBA\_COL\_PRIVS | 包含数据库中所有授予表列上的对象权限信息 |
| ALL\_COL\_PRIVS\_MADE | 包含当前用户作为对象权限的授予者，在所有列  上的对象权限信息 |
| ALL\_COL\_PRIVS\_RECD | 包含当前用户作为对象权限的接收者，在所有列  上的对象权限信息 |
| DBA\_TAB\_PRIVS | 包含数据库所有的对象权限信息 |
| DBA\_SYS\_PRIVS | 包含数据库中所有的系统权限信息 |
| SESSION\_PRIVS | 包含当前数据库用户可以使用的权限信息 |

#### 查看用户权限

##### 查看所有用户

select \* from dba\_user;

select \* from all\_users;

select \* from user\_users;

##### 查看用户系统权限

select \* from dba\_sys\_privs;

select \* from all\_sys\_privs;

select \* from user\_sys\_privs;

##### 查看用户对象权限

select \* from dba\_tab\_privs;  
select \* from all\_tab\_privs;  
select \* from user\_tab\_privs;

#### 撤销系统权限

撤销用户的权限需要使用REVOKE语句，其语法如下：

|  |
| --- |
| REVOKE system\_privilege [ , … ] FROM  { user\_name [ , … ] | role\_name [ , … ] | PUBLIC } ; |

撤销某用户的系统权限时，如果该用户将权限授予了其他用户，则其他用户的权限不受影响。所以WITH ADMIN OPTION选项应该慎重使用。



* 【示例】

|  |
| --- |
| SQL>REVOKE CREATE TABLE FROM test; |

#### 撤销对象权限

撤销对象权限也需要使用REVOKE语句，其语法如下：

|  |
| --- |
| REVOKE object\_privilege [ , … ] | ALL [ PRIVILEGES ]  ON <schema.>object\_name  FROM { username [ , … ] | role\_name [ , … ] | PUBLIC } ; |

与撤销系统权限不同的是，在撤销某用户的对象权限时，如果该用户将

权限授予了其他用户，则其他用户的相应权限也将被撤销。



* 【示例】

|  |
| --- |
| SQL>REVOKE select on SCOTT.EMP FROM test; |

### 角色

#### 概述

角色是具有名称的一组相关权限的组合，即将不同权限组合在一起就形成了角色。可以使用角色为用户授权，同样也可以从用户中回收角色。

由于角色集合了多种权限，所以为用户授予某个角色时，相当于为用户授予多种权限。这样就避免了向用户逐一授权，从而简化了用户权限的管理。例如，为两个用户授予4个不同的权限。在未使用角色时，需要8次操作才能完成。如果使用角色，将这4个权限组成一个角色，然后将这个角色授予这两个用户，只需要两次操作就能完成。



#### 角色的优点

角色的优点和特性可以概括为如下几个方面：

* 并不是一次一个地将权限直接授予一个用户，而是先创建角色，向该角 色授予一些权限，然后再将该角色授予多个用户。
* 在增加或者删除一个角色的权限时，被授予该角色的所有用户都会自动 获得或者失去相应权限。
* 可以将多个角色授予一个用户。
* 可以为角色设置口令。

#### 系统预定义角色

|  |  |
| --- | --- |
| **角色名称** | **说明** |
| CONNECT | 具有最终用户的典型权限和最基本的权限，例如：  ALTER SESSION、CREATE CLUSTER、CREATE  DATABASE LINK、CREATE SEQUENCE、以及  CREATE SESSION、CREATE SYNONYM和CREATE  VIEW等 |
| RESOURCE | 主要是授予开发人员的，包括以下系统权限：  CREATE CLUSTER、CREATE PROCEDUER、  CREATE SEQUENCE、CREATE TABLE、CREATE  TRIGGER和CREATE TYPE等 |
| DBA | 拥有系统的所有系统权限 |
| EXP\_FULL\_DATABASE | 具有数据库逻辑备份时的数据导出权限，包括：  EXECUTE ANY PROCEDURE和SELECT ANY  TABLE等 |

|  |  |
| --- | --- |
| **角色名称** | **说明** |
| IMP\_FULL\_DATABASE | 具有数据库逻辑备份时的数据导入权限，包括：  BACKUP ANY TABLE和SELECT ANY TABLE等 |
| DELETE\_CATALOG\_ROLE | 具有删除和重建数据字典所需要的权限 |
| EXECUTE\_CATALOG\_ROLE | 具有查询数据字典的权限 |
| SELECT\_CATALOG\_ROLE | 具有从数据字典中执行部分存储过程和函数的  权限 |

### 创建角色

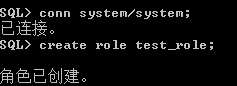
使用CREATE ROLE语句创建角色的语法如下：

|  |
| --- |
| CREATE ROLE role\_name  [ NOT IDENTIFIED | IDENTIFIED BY password ]; |

* 语法说明如下：
* role\_name 创建的角色名。
* NOT IDENTIFIED | IDENTIFIED BY password 可以为角色设置口令。其中，NOT IDENTIFIED表示该角色不需要口令就可以被启用或修改；
* IDENTIFIED BY password表示必须通指定口令才能启用或修改该角色，默认情况下没有口令。

要创建角色，必须具有CREATE ROLE系统权限。

* 【示例】



### 为角色授予权限

新创建的角色还不具有任何权限，可以使用GRANT语句向该角色授予权限，其语法形式与向用户授予权限基本相同。

* 【示例】

向角色emp\_manager授予scott.emp表上的SELECT、UPDATE、

INSERT和DELETE对象权限，如下：

|  |
| --- |
| SQL> CONNECT scott/tiger  已连接。  SQL> GRANT SELECT , UPDATE , INSERT , DELETE  2 ON scott.emp TO emp\_manager ;  授权成功。 |

与GRANT语句相对应，使用REVOKE语句可以撤销角色的权限。

### 为用户授予角色

为用户授予角色，同样是使用GRANT语句。

* 【示例】

创建用户emp\_user，并为该用户授予emp\_manager角色，以及连接数据库所必需的CREATE SESSION权限，如下：

|  |
| --- |
| SQL> CONNECT system/admin  已连接。  SQL> CREATE USER emp\_user IDENTIFIED BY user\_password ;  用户已创建。  SQL> GRANT emp\_manager , CREATE SESSION TO emp\_user ;  授权成功。 |

### 修改和删除角色

查看角色信息相关的数据字典

|  |  |
| --- | --- |
| **数据字典视图** | **描述** |
| DBA\_ROLES | 记录数据库中所有的角色 |
| DBA\_ROLE\_PRIVS | 记录所有已经被授予用户和角色的角色 |
| USER\_ROLES | 包含已经授予当前用户的角色信息 |
| ROLE\_ROLE\_PRIVS | 包含角色授予的角色信息 |
| ROLE\_SYS\_PRIVS | 包含为角色授予的系统权限信息 |
| ROLE\_TAB\_PRIVS | 包含为角色授予的对象权限信息 |
| SESSION\_ROLES | 包含当前会话所包含的角色信息 |

#### 修改角色

使用ALTER ROLE语句可以对角色的口令进行修改，该语句的语法如下：

|  |
| --- |
| ALTER ROLE role\_name  [ NOT IDENTIFIED |  IDENTIFIED BY password  ]; |

* NOT IDENTIFIED表示不需要口令就可以启用或修改该角色；
* IDENTIFIED BY password表示必须通过指定口令才能启用或修改该角色。
* 【示例】

取消角色ROLE1的口令

|  |
| --- |
| SQL> alter role role1 not identified;  角色已丢弃。 |

为角色ROLE1增加口令

|  |
| --- |
| SQL> alter role role1 identified by password;  角色已丢弃。 |

#### 删除角色

当不再需要某个角色时，可以使用DROP ROLE语句将角色删除，该语句的语法如下：

|  |
| --- |
| DROP ROLE role\_name; |

* 【示例】

使用DROP ROLE语句删除角色ROLE1

|  |
| --- |
| SQL> drop role role1;  角色已删除。 |

角色被删除后，对于使用该角色的用户来说，相应的权限同时被回收。

#### 查看所有角色

select \* from dba\_roles;

#### 查看用户所拥有的角色：

select \* from dba\_role\_privs;

select \* from user\_role\_privs;

#### 查看某个角色的具体权限

如grant connect,resource,create session,create view to TEST;

查看RESOURCE具有那些权限，用SELECT \* FROM DBA\_SYS\_PRIVS WHERE GRANTEE='RESOURCE';

## 数据类型

Oracle提供了许多数据类型，下表中列出部分、常用的数据类型。

|  |  |
| --- | --- |
| **数据类型** | **说明** |
| CHAR [ (  length [ BYTE | CHAR ]  ) ] | 固定长度的字符数据。表示长度为length个字节或字符。BYTE表示按字节个数定义长度；CHAR表示按字符个数定义长度 |
| VARCHAR2 ( length [ BYTE |  CHAR ] ) | 可变长度的字符数据。表示长度最多可为length个字  节或字符。最大长度可为4000字节 |
| NCHAR [ ( length ) ] | 固定长度的Unicode字符数据。最大长度为4000字节 |
| NVARCHAR2 ( length ) | 可变长度的字符数据。表示长度为length个字符 |
| NUMBER [ ( precision [ , scale  ] ) ]和  NUMERIC [ ( precision [ , scale  ] ) ] | 可变长度的数字。precision是数字可用的最大位数（如果有小数点的话，是小数点前后位数之和），最大可为38；scale则表示小数点右边的最大位数。如果不指定precision和scale，则表示为小数点前后共38位的数字 |
| INT、INTEGER和SMALLINT | NUMBER的子类型。38位精度的整数 |
| BINARY\_FLOAT | 32位浮点数 |
| BINARY\_DOUBLE | 64位浮点数 |
| DATE | 日期和时间。包括世纪、4位年份、月、日、时（24小时格式）、分和秒。可以存储公元前4712年1月1日和公元后4712年12月31日之间的日期和时间 |
| TIMESTAMP [ ( seconds\_precision ) ] | 日期和时间。包括世纪、4位年份、月、日、时（24小时格式）、分和秒 |
| CLOB | 可变长度的单字节字符数据。最多存储128TB |
| NCLOB | 可变长度的Unicode字符数据。最多存储128TB |
| BLOB | 可变长度的二进制数据。最多存储128TB |
| BFILE | 指向外部文件的指针。外部文件本身不存储在数据库中 |

### 关于number、int、integer的说明

oracle本来就没有int类型，为了与别的数据库兼容，新增了int类型作为number类型的子集。

* int类型只能存储整数;
* number可以存储浮点数，也可以存储整数；
* number(8,1)存储小数位为1位，总长度为8的浮点数，如果小数位数不足，则用0补全；
* number(8)存储总长度为8的整数；
* int和integer相当于number(38)

### binary\_float和binary\_double

Oracle中引入了两种新的数据类型：binary\_float和binary\_double。这两种数据类型是对number数据类型的补充。

* Binary\_float：存储一个单精度的32位浮点数。
* Binary\_double：存储一个双精度的64位浮点数。
* 他们的优点：

1. 需要的存储空间较小：Binary\_float需要5个字节，Binary\_double需要9个字节，number需要32个字节。

2. 可以表示的数字范围更大

3. 执行运算的速度更快：binary\_float和binary\_double类型通常是在硬件中执行运算，而number类型在运算之前首先使用软件进行转换。

4. 运算封装：对binary\_float和binary\_double类型进行的数学运算是封装的，也就是说结果要么是数字，要么是一个特殊值。

5. 取整透明：binary\_float和binary\_double类型使用2进制来表示数字，而number则使用十进制来表示。

下面是关于 binary\_float和binary\_double的4个特殊值

Binary\_float\_nan 非binary\_float 类型的数字 (nan)

Binary\_float\_infinity Binary\_float类型的无穷大 (inf)

Binary\_double\_nan ary\_double 类型的数字 (nan)

Binary\_double\_infinity Binary\_double类型的无穷大 (inf)

### 伪列

Oracle中伪列就像一个表列，但是它并没有存储在表中。伪列可以从表中查询，但不能插入、更新和删除它们的值。常用的伪列有ROWID和ROWNUM

* ROWID 是表中行的存储地址，该地址可以唯一地标识数据库中的一行，可以使用 ROWID 伪列快速地定位表中的一行
* ROWNUM 是查询返回的结果集中行的序号，可以使用它来限制查询返回的行数

## 表和约束

### 创建表

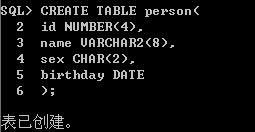
创建表需要使用CREATE TABLE语句，其语法如下：

|  |
| --- |
| CREATE TABLE [ schema. ] table\_name(  column\_name data\_type [ DEFAULT expression ]  [ [ CONSTRAINT constraint\_name ] constraint\_def ]  [ , … ]  )[ TABLESPACE tablespace\_name ]; |

语法说明

* schema 指定表所属的用户名，或者所属的用户模式名称。
* table\_name 创建的表的名称。
* column\_name 表中的列的名称，可以有多个列，多个列之间使用逗号（,）隔开。同一个表中的列的名称必须惟一。
* data\_type 列的数据类型。
* DEFAULT expression 列的默认值。如果在向表中添加数据时，没有指定该列的数据，则该列将使用默认值。
* CONSTRAINT constraint\_name 为约束命名。如果不使用此子句，Oracle将自动为约束建立默认的名称。如果是创建表级约束，则必须使用此子句为约束命名。
* constraint\_def 为列指定约束。例如非空约束、惟一约束等。
* TABLESPACE tablespace\_name 可以为表指定存储表空间。如果不使用此子句，则使用默认表空间存储新表。

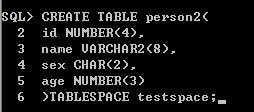
示例：



查询表存储在哪个表空间中



指定表空间



再次查询

### 管理表中的列

* 增加列

为表增加列的语法形式如下：

|  |
| --- |
| ALTER TABLE table\_name ADD column\_name data\_type; |

【例】为表person增加email列，如下：

|  |
| --- |
| SQL> ALTER TABLE person ADD email VARCHAR2(20);  表已更改。 |

* 修改列的名称

修改表中的列的名称的语法如下：

|  |
| --- |
| ALTER TABLE table\_name RENAME COLUMN column\_name TO new\_column\_name; |

【例】将person表中的birthday列的名称修改为age，如下：

|  |
| --- |
| SQL> ALTER TABLE person RENAME COLUMN birthday TO age;  表已更改。 |

* 修改列的数据类型

修改表中的列的数据类型的语法如下：

|  |
| --- |
| ALTER TABLE table\_name MODIFY column\_name new\_data\_type; |

【例】将person表中的age列的数据类型修改为NUMBER(4)，如下：

|  |
| --- |
| SQL> ALTER TABLE person MODIFY age NUMBER(4);  表已更改。 |

* 删除列

删除表中的列时可以分为一次删除一列和一次删除多列。一次删除一列的语法形式如下：

|  |
| --- |
| ALTER TABLE table\_name DROP COLUMN column\_name; |

一次删除多列的语法形式如下：

|  |
| --- |
| ALTER TABLE table\_name DROP (column\_name , …); |

对比两种语法可以看出，删除一列时需要使用COLUMN关键字，而删除多列时则不需要。

### 重命名表

对于已存在的表，还可以修改其名称。重命名表有两种语法形式，一种是使用ALTER TABLE语句，语法如下：

|  |
| --- |
| ALTER TABLE table\_name RENAME TO new\_table\_name； |

另一种是直接使用RENAME语句，语法如下：

|  |
| --- |
| RENAME table\_name TO new\_table\_name; |

### 移动表

在创建表时可以为表指定存储空间，如果不指定，Oracle会将该表存储到默认表空间中。根据需要可以将表从一个表空间中移动到另一个表空间中。语法如下：

|  |
| --- |
| ALTER TABLE table\_name MOVE TABLESPACE tablespace\_name; |

### 删除表

使用DROP TABLE语句可以删除表。删除表后，该表中的所有数据也将被删除。一般情况下，用户只能删除自己模式中的表；如果需要删除其他模式中的表，则该用户必须具有DROP ANY TABLE的系统权限。

删除表的语法如下：

|  |
| --- |
| DROP TABLE table\_name [ CASCADE CONSTRAINTS ] [ PURGE ]; |

语法说明如下：

* + CASCADE CONSTRAINTS 指定删除表的同时，删除所有引用这个表的视图、约束、索引和触发器等。
  + PURGE 表示删除该表后，立即释放该表所占用的资源空间。

### 约束分类

数据库完整性（Database Integrity）是指数据库中数据的正确性和相容性，用来防止用户向数据库中添加不合语义的数据。数据库完整性是由各种各样的完整性约束来保证的，可以说，数据库完整性设计就是数据库完整性约束的设计。

按照不同的角度可以将表的完整性约束分成不同的类别。主要可以选取两个角度：约束的作用域和约束的用途。

按照约束的作用域可以将表的完整性约束分为如下两大类：

* 表级约束 应用于表，对表中的多个列起作用。
* 列级约束 应用于表中的一列，只对该列起作用。

按照约束的用途可以将表的完整性约束分为5类

|  |  |  |
| --- | --- | --- |
| **约束** | **简写** | **说明** |
| NOT NULL | C | 非空约束。指定一列不允许存储空值。这实际就  一种强制的CHECK约束 |
| PRIMARY KEY | P | 主键约束。指定表的主键。主键由一列或多列组成  ，惟一标识表中的一行 |
| UNIQUE | U | 惟一约束。指定一列或一组列只能存储惟一的值 |
| CHECK | C | 检查约束。指定一列或一组列的值必须满足的条件 |
| FOREIGN KEY | R | 外键约束。指定表的外键。外键引用另外一个表中  的一列，在自引用的情况中，则引用本表中的一列 |

约束就是指对插入数据的各种限制，例如：人员的姓名不能为空，人的年龄只能在0~150岁之间。约束可以对数据库中的数据进行保护。

约束可以在建表的时候直接声明，也可以为已建好的表添加约束。

* NOT NULL：非空约束

例如：姓名不能为空

|  |
| --- |
| CREATE TABLE person  (  pid NUMBER ,  name VARCHAR(30) NOT NULL  ) ;  -- 插入数据  INSERT INTO person(pid,name) VALUES (11,'张三');  -- 错误的数据，会受到约束限制，无法插入  Oracle 教程  -50-  INSERT INTO person(pid) VALUES (12); |

* PRIMARY KEY：主键约束

不能重复，不能为空

例如：身份证号不能为空。

现在假设pid字段不能为空，且不能重复。

|  |
| --- |
| DROP TABLE person ;  CREATE TABLE person  (  pid NUMBER PRIMARY KEY ,  name VARCHAR(30) NOT NULL  ) ;  -- 插入数据  INSERT INTO person(pid,name) VALUES (11,'张三');  -- 主键重复了  INSERT INTO person(pid,name) VALUES (11,'李四'); |

* UNIQUE：唯一约束，值不能重复（空值除外）

人员中有电话号码，电话号码不能重复。

|  |
| --- |
| DROP TABLE person ;  CREATE TABLE person  (  pid NUMBER PRIMARY KEY NOT NULL ,  name VARCHAR(30) NOT NULL ,  tel VARCHAR(50) UNIQUE  ) ;  -- 插入数据  INSERT INTO person(pid,name,tel) VALUES (11,'张三','1234567');  -- 电话重复了  INSERT INTO person(pid,name,tel) VALUES (12,'李四','1234567'); |

* CHECK：条件约束，插入的数据必须满足某些条件

例如：人员有年龄，年龄的取值只能是0~150岁之间

|  |
| --- |
| DROP TABLE person ;  CREATE TABLE person  (  pid NUMBER PRIMARY KEY NOT NULL ,  Oracle 教程  -51-  name VARCHAR(30) NOT NULL ,  tel VARCHAR(50) NOT NULL UNIQUE ,  age NUMBER CHECK(age BETWEEN 0 AND 150)  ) ;  -- 插入数据  INSERT INTO person(pid,name,tel,age) VALUES (11,'张三','1234567',30);  -- 年龄的输入错误  INSERT INTO person(pid,name,tel,age) VALUES (12,'李四','2345678',-100); |

* Foreign Key：外键

例如：有以下一种情况：

一个人有很多本书：

|- Person表

|- Book表：而且book中的每一条记录表示一本书的信息，一本书的信息属于一个人

|  |
| --- |
| CREATE TABLE book  (  bid NUMBER PRIMARY KEY NOT NULL ,  name VARCHAR(50) ,  -- 书应该属于一个人  pid NUMBER  ) ; |

如果使用了以上的表直接创建，则插入下面的记录有效：

|  |
| --- |
| INSERT INTO book(bid,name,pid) VALUES(1001,'JAVA',12) ; |

以上的代码没有任何错误，但是没有任何意义，因为一本书应该属于一个人，所以在此处的pid的取值应该与person表中的pid一致。

此时就需要外键的支持。修改book的表结构

|  |
| --- |
| DROP TABLE book ;  CREATE TABLE book  (  bid NUMBER PRIMARY KEY NOT NULL ,  name VARCHAR(50) ,  -- 书应该属于一个人  pid NUMBER REFERENCES person(pid) ON DELETE CASCADE  -- 建立约束：book\_pid\_fk，与person中的pid为主-外键关系  --CONSTRAINT book\_pid\_fk FOREIGN KEY(pid) REFERENCES person(pid)  ) ;  INSERT INTO book(bid,name,pid) VALUES(1001,'JAVA',12) ; |

### 查看约束

* 视图USER\_CONSTRAINTS

通过查询数据字典视图USER\_CONSTRAINTS，可以了解当前用户模式中所有约束的基本信息。

|  |  |  |
| --- | --- | --- |
| **列** | **类型** | **说明** |
| owner | VARACHAR2(30) | 约束的所有者 |
| constraint\_name | VARACHAR2(30) | 约束名 |
| constraint\_type | VARACHAR2(1) | 约束类型，值为P、R、C、U、V或O |
| table\_name | VARACHAR2(30) | 约束定义所针对的表名 |
| status | VARACHAR2(8) | 约束的状态。值为ENABLED或DISABLED |
| deferrable | VARACHAR2(14) | 该约束是否为可延迟的。值为  DEFERRABLE或NOTDEFERRABLE |
| deferred | VARACHAR2(9) | 约束是立即执行还是延迟执行。  值为IMMEDIATE或DEFERRED |

* 视图USER\_CONS\_COLUMNS

通过查询数据字典视图USER\_CONS\_COLUMNS，可以了解定义约束的列。

|  |  |  |
| --- | --- | --- |
| **列** | **类型** | **说明** |
| owner | VARCHAR2(30) | 约束的所有者 |
| constraint\_name | VARCHAR2(30) | 约束名 |
| table\_name | VARCHAR2(30) | 约束定义所针对的表名 |
| column\_name | VARCHAR2(4000) | 约束定义所针对的列名 |

### 指定级联操作类型

在添加FOREIGN KEY约束时，还可以指定级联操作的类型，主要用于确定当删除（ON DELETE）父表中的一条记录时，如何处理子表中的外键字段。有如下3种引用类型：

* + CASCADE 此关键字用于表示当删除主表中被引用列的数据时，级联删除子表中相应的数据行。
  + SET NULL 此关键字用于表示当删除主表中被引用列的数据时，将子表中相应引用列的值设置为NULL值。这种情况要求子表中的引用列支持NULL值。
  + NO ACTION 此关键字用于表示当删除主表中被引用列的数据时，如果子表的引用列中包含该值，则禁止该操作执行。默认为此选项。

|  |
| --- |
| CREATE TABLE book  (  bid NUMBER PRIMARY KEY NOT NULL ,  name VARCHAR(50) ,  -- 书应该属于一个人  pid NUMBER ,  -- 建立约束：book\_pid\_fk，与person中的pid为主-外键关系  CONSTRAINT book\_pid\_fk FOREIGN KEY(pid) REFERENCES person(pid) ON DELETE CASCADE  ) ; |

## SQL基础

### SQL简介

* + SQL 是 Structured Query Language（结构化查询语言）的首字母缩写词
  + SQL 是数据库语言，Oracle 使用该语言存储和检索信息
  + 表是主要的数据库对象，用于存储数据
  + 通过 SQL可以实现与 Oracle 服务器的通信

发送SQL语句查询

select \* from emp



* + SQL 支持下列类别的命令：
  + 数据定义语言（DDL）
  + 数据操纵语言（DML）
  + 事务控制语言（TCL）
  + 数据控制语言（DCL）

数据定义语言

CREATE

ALTER

DROP

数据操纵语言

SELECT

DELETE

INSERT

UPDATE

事务控制语言

COMMIT

SAVEPOINT

ROLLBACK

事务控制语言

GRANT

REVOKE

### 数据定义语言

数据定义语言用于改变数据库结构，包括创建、更改和删除数据库对象

用于操纵表结构的数据定义语言命令有：

* + CREATE TABLE
  + ALTER TABLE
  + TRUNCATE TABLE
  + DROP TABLE

### 数据操纵语言

数据操纵语言用于检索、插入和修改数据

数据操纵语言是最常见的SQL命令

数据操纵语言命令包括：

* + SELECT
  + INSERT
  + UPDATE
  + DELETE
* 利用现有的表创建表

语法

|  |
| --- |
| CREATE TABLE <new\_table\_name> AS SELECT column\_names FROM <old\_table\_name>; |

示例

|  |
| --- |
| SQL> CREATE TABLE newitemfile AS SELECT \* FROM itemfile; |
| SQL> CREATE TABLE newitemfile1 AS SELECT itemcode, itemdesc, qty\_hand  FROM itemfile; |
| SQL> CREATE TABLE newitemfile2 AS SELECT \* FROM itemfile WHERE 1 = 2; |

* 选择无重复的行

在SELECT子句，使用DISTINCT关键字

|  |
| --- |
| SQL> SELECT DISTINCT vencode FROM vendor\_master; |

* 使用列别名

为列表达式提供不同的名称

该别名指定了列标题

|  |
| --- |
| SQL> SELECT itemcode,itemdesc, max\_level, max\_level\* 2“New Maximum Level”FROM itemfile; |
| SQL> SELECT itemcode, itemdesc, max\_level, max\_level\*2 AS NEW\_MAXLEVEL  FROM itemfile; |

其它操作请参考SQL-99

### 事务控制语言

事务是最小的工作单元，作为一个整体工作。

保证事务的整体成功或失败，称为事务控制。

用于事务控制的语句有：

* COMMIT - 提交并结束事务处理
* ROLLBACK - 撤销事务中已完成的工作
* SAVEPOINT – 标记事务中可以回滚的点

|  |
| --- |
| SQL> UPDATE order\_master  SET del\_date = ‘30-8月-05’  WHERE orderno <= ’o002’;  SQL> SAVEPOINT mark1;  SQL> DELETE FROM order\_master WHERE orderno = ‘o002’;  SQL> SAVEPOINT mark2;  SQL> ROLLBACK TO SAVEPOINT mark1;  SQL> COMMIT; |

### 数据控制语言

数据控制语言为用户提供权限控制命令

用于权限控制的命令有：

* GRANT 授予权限
* REVOKE 撤销已授予的权限

关于授权和撤销已在用户管理中讲过，此处不再细说。

### SQL 操作符

Oracle 支持的 SQL 操作符分类如下

* 算术操作符
* 逻辑操作符
* 比较操作符
* 集合操作符
* 连接操作符

#### 算术操作符

算术操作符用于执行数值计算。

可以在SQL语句中使用算术表达式，算术表达式由数值数据类型的列名、数值常量和连接它们的算术操作符组成。

算术操作符包括加(+)、减(-)、乘(\*)、除(/)。

|  |
| --- |
| SQL> SELECT itemdesc, max\_level - qty\_hand avble\_limit  FROM itemfile WHERE p\_category='spares'; |
| SQL > SELECT itemdesc, itemrate\*(max\_level - qty\_hand)  FROM itemfile  WHERE p\_category='spares'; |

#### 比较操作符

比较操作符用于比较两个表达式的值。

比较操作符包括 =、!=、<、>、<=、>=、BETWEEN…AND、IN、LIKE 、IS NULL。

|  |
| --- |
| SQL> SELECT itemdesc, re\_level  FROM itemfile  WHERE qty\_hand < max\_level/2; |
| SQL> SELECT orderno FROM order\_master  WHERE del\_date IN (‘06-1月-05’,‘05-2月-05'); |
| SQL> SELECT vencode,venname,tel\_no  FROM vendor\_master  WHERE venname LIKE 'j\_\_\_s'; |

#### 逻辑操作符

逻辑操作符用于组合多个计较运算的结果以生成一个或真或假的结果。

逻辑操作符包括AND、OR、NOT。

显示 2005-5月-10 至 2005-5月-26的订单信息

|  |
| --- |
| SQL> SELECT \* FROM order\_master  WHERE odate > ‘10-5月-05'  AND del\_date < ‘26-5月-05’; |

#### 集合操作符

集合操作符可以将两个或多个查询返回的结果组合起来，常用的集合操作符包括：UNION、UNION ALL、INTERSECT和MINUS。这些集合操作符具有相同的优先级，当同时使用多个操作符时，会按照从左到右的方式引用这些集合操作符。当使用集合操作符时，必须确保不同查询的列个数和数据类型都要匹配。

集合操作符包括：

* UNION、UNION ALL

使用UNION获取两个结果集的并集。使用集合操作符可以将两个或者多个查询返回的行组合起来。使用UNION操作符获取两个结果集的并集，语法如下：

select\_statement UNION [ ALL ] select\_statement

[UNION [ ALL ] select\_statement ] [ … ]

将多个查询结果集合并，形成一个新的结果集。如果指定ALL，则包括

重复的行；如果不指定ALL，则对重复的行只保留一行。相当于结果集的

OR运算。



|  |
| --- |
| SQL> SELECT empno , ename , sal , deptno FROM scott.emp  2 WHERE empno > 7800  3 UNION ALL  4 SELECT empno , ename , sal , deptno FROM scott.emp  5 WHERE deptno = 10 ORDER BY deptno ASC ; |

* INTERSECT
* MINUS

INTERSECT 操作符只返回两个查询的公共行。

|  |
| --- |
| SQL> SELECT orderno FROM order\_master  INTERSECT  SELECT orderno FROM order\_detail; |

MINUS 操作符返回从第一个查询结果中排除第二个查

询中出现的行。

|  |
| --- |
| SQL> SELECT orderno FROM order\_master  MINUS  SELECT orderno FROM order\_detail; |

#### 连接操作符

连接操作符用于将多个字符串或数据值合并成一个字符串

示例：通过使用连接操作符可以将表中的多个列合并成逻辑上的一行列

|  |
| --- |
| SQL> SELECT (venname|| ' 的地址是 '  ||venadd1||' '||venadd2 ||' '||venadd3) address  FROM vendor\_master WHERE vencode='V001'; |

#### 操作符的优先级

* SQL 操作符的优先级从高到低的顺序是：
  1. 算术操作符 --------最高优先级
  2. 连接操作符
  3. 比较操作符
  4. NOT 逻辑操作符
  5. AND 逻辑操作符
  6. OR 逻辑操作符 --------最低优先级

### MEGER语句

使用MERGE语句，可以对指定的两个表执行合并操作

* 语法

|  |
| --- |
| MEGER INTO table1\_name  USING table2\_name ON join\_condition  WHEN MATCHED THEN UPDATE SET …  WHEN NOT MATCHED THEN INSERT … VALUES … |

语法说明如下：

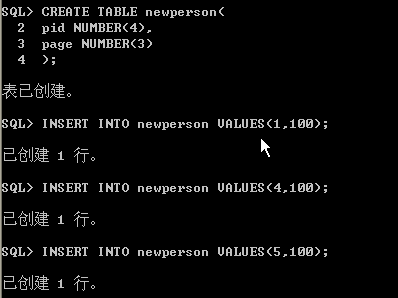
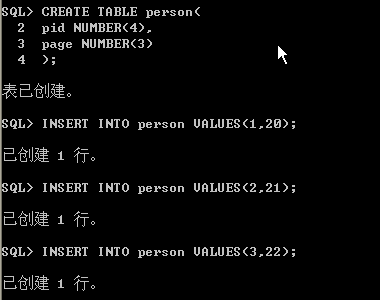
* table1\_name 表示需要合并的目标表。
* table2\_name 表示需要合并的源表。
* join\_condition 表示合并条件。
* WHEN MATCHED THEN UPDATE 表示如果符合合并条件，则执行更新操作。
* WHEN NOT MATCHED THEN INSERT 表示如果不符合合并条件，则执行插入操作。
* UPDATE和INSERT

如果只是希望将源表中符合条件的数据合并到目标表中，可以只使用UPDATE子句；如果只是希望将源表中不符合合并条件的数据合并到目标表中，可以只使用INSERT子句。

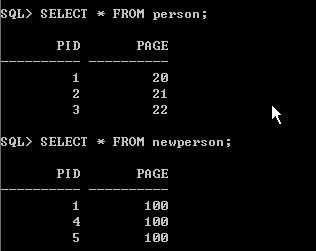
在UPDATE子句和INSERT子句中，都可以使用WHERE子句指定更新或插入的条件。这时，对于合并操作来说，提供了两层过滤条件，第一层是合并条件，由MERGE语句中的ON子句指定；第二层是UPDATE或INSERT子句中指定的WHERE条件。从而使得合并操作更加灵活和精细。

注意：为了实现合并操作，二个表的结构、数据类型应该完全相同。

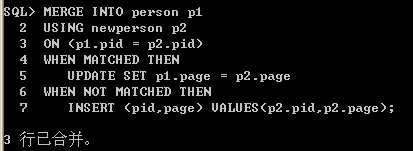
* 示例



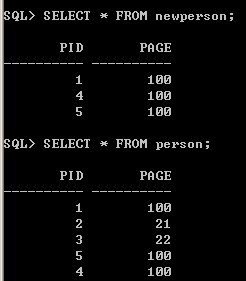
数据



执行合并



重新执行查询



## Oracle函数

### 字符函数

字符函数接受字符输入并返回字符或数值

字符串函数是Oracle系统中比较常用的一种函数。在使用字符串函数时，可以

接受字符参数，这些字符可以是一个任意有效的表达式，也可以来自于表中的一列。

然后字符函数会按照某种方式处理输入参数，并返回一个结果。

以下是一些其它的字符函数：

|  |  |
| --- | --- |
| **函数** | **说明** |
| ASCII(string) | 返回给定ASCII字符string的十进制值 |
| CHR(integer) | 返回给定整数integer所对应的ASCII字符 |
| CONCAT(string1 , string2) | 连接字符串string1和字符串string2 |
| INITCAP(string) | 字符串string第一个字母变为大写，其余字母不变 |
| INSTR(string1 , string2[  , start][ , occurrence]) | 在string1中查找字符串string2，返回string2所在  的位置。从start位置开始查找；返回string2第  occurrence次出现的位置 |
| LOWER(string) | 将字符串string的全部字母转换为小写 |
| UPPER(string) | 将字符串string的全部字母转换为大写 |
| LPAD(string , count [ , char ]) | 使用指定的字符char在字符串string的左边填充。其  中string为被操作的字符串，count为填充的字符总  数，char为可选项，表示要填充的字符，默认为空格 |
| RPAD(string , start [ , char ]) | 使用指定的字符在字符串string的右边填充。各参数  的意义同LPAD |
| LTRIM(string [ , char ]) | 删除字符串string中左边出现的字符char，char的默  认值为空格 |
| RTRIM(string [ , char ]) | 删除字符串string中右边出现的字符char，char的默  认值为空格 |
| REPLACE(string , char1[ , char2]) | 替换字符串。其中string表示被操作的字符串，char1  表示要查找的字符，char2表示要替换的字符。如果没  有指定char2，则要替换的字符默认为空字符串，即每  查找到指定的字符串时，删除该字符串 |
| SUBSTR(string , start [ , count ]) | 获取源字符串string的子串，其中string为源字符串  ；start表示输出的子字符串的第一个字符在源字符串  中的位置；count表示输出的子字符串的字符数目 |
| LENGTH(string) | 返回字符串参数string的长度 |

### 数字函数

当检索的数据为数字数据类型时，可以使用数字函数进行数学计算。

|  |  |
| --- | --- |
| **函数** | **说明** |
| ABS(value) | 获取value数值的绝对值 |
| CEIL(value) | 返回大于或等于value的最小整数值 |
| FLOOR(value) | 返回小于或等于value的最大整数值 |
| SIN(value) | 获取value的正弦值 |
| COS(value) | 获取value的余弦值 |
| EXP(value) | 返回以e为底的value的指数值，其中e约等于  2.71828183 |
| LN(value) | 返回value的自然对数 |
| LOG(value) | 返回value的以10为底的对数 |
| POWER(value , exponent) | 返回value的exponent的指数值 |
| ROUND(value , precision) | 对value按precision精度四舍五入 |
| MOD(value , divisor) | 取余 |
| SQRT(value) | 返回value的平方根。若value为负数，则该函数无  意义 |
| TRUNC(value1[ , value2 ]) | 返回对value1截断的结果，value2为可选参数，表  示对第几位小数截断。如果不指定value2，则从  value1的0位小数处截断；如果value2为负数，则  对value1在小数点  左边的第|value|位处截断。例如TRUNC(5.77) = 5 ；  TRUNC(5.77 , 1) = 5.7 ；TRUNC(5.77 , -1) = 0 |

### 聚合函数

检索数据不仅仅是把现有的数据简单地从表中取出来，很多情况下，还需要对数据执行各种统计计算。

|  |  |
| --- | --- |
| **函数** | **说明** |
| AVG(x) | 返回对一个数字列或计算列求取的平均值 |
| SUM(x) | 返回一个对数字列或计算列的汇总和 |
| MAX(x) | 返回一个数字列或计算列中的最大值 |
| MIN(x) | 返回一个数字列或计算列中的最小值 |
| COUNT(x) | 返回记录的统计数量 |

### 日期时间函数

SQL语句主要通过日期时间函数操纵日期和时间数据。在Oracle系统中，默认的日期格式为DD-MON-YY。

日期时间函数对日期值进行运算，并生成日期数据类型或数值类型的结果

日期时间函数包括：

|  |  |  |
| --- | --- | --- |
| **函数** | **说明** | **示例** |
| SYSDATE() | 获取系统当前的日期值 | select sysdate from dual; |
| CURRENT\_TIMESTAMP() | 获取当前的日期和时间值 | select  CURRENT\_TIMESTAMP  from dual; |
| CURRENT\_DATE() |  | select current\_date from dual; |
| ADD\_MONTHS(date , count) | 在指定的日期date上增加  count个月 | Select  add\_months(sysdate,2)  from dual; |
| LAST\_DAY(date) | 返回日期date所在月的最  后一天 | Select  last\_day(sysdate)  from dual; |
| MONTHS\_BETWEEN(  date1 , date2) | 返回date1和date2间隔  多少个月 | select  months\_between(sysdate,  to\_date('2005-11-12','yyyy-mm-dd'))  from dual; |
| NEW\_TIME(date , 'this' , 'other') | 将时间从this时区转变为  other时区 |  |
| NEXT\_DAY(date , 'day') | 返回指定日期之后下一个  星期几的日期。这里的day  表示星期几 | SELECT  next\_day(to\_date('20050620','YYYYMMDD'),1)  FROM dual; |
| GREATEST(date1 , date2 , …) | 从日期列表中选出最早的  日期 |  |

经典用法：

### 转换函数

在执行运算时，经常需要将一种数据类型转换为另一种数据类型，这种转换可以是隐式转换，也可以是显式转换。隐式转换是在运算的过程中，由系统自动完成的，不需要用户考虑，例如字符串1（即‘1’）可以被隐式转换为数字1。而显式转换则需要调用相应的转换函数来实现，例如，使用TO\_DATE函数，将字符串类型转换成日期类型。

|  |  |
| --- | --- |
| **函数** | **说明** |
| CAST(value AS type) | 将value转换为type所指定的兼容数据类型 |
| CONVERT(value , source\_char\_set , dest\_char\_set) | 将value从源字符集source\_char\_set转换为结果  字符集dest\_char\_set |
| DECODE(value , search , result , default) | 将value与search相比较，如果相等，该函数返  回result值，否则返回default值 |
| BIN\_TO\_NUM(value) | 将二进制数字value转换为number类型 |
| TO\_TIMESTAMP(value) | 将字符串value转换为一个TIMESTAMP类型 |
| TO\_CHAR(value[ , format]) | 将value转换为一个VARCHAR2字符串。可以  指定一个可选参数format来说明value的格式 |
| TO\_NUMBER(value[ , format]) | 将数字字符串value转化成数字数据 |
| TO\_DATE(string , 'format') | 按照指定的格式format，把字符串转换成日期  数据。如果省略了format格式，则默认的日期  格式为DD-MON-YY |

### 日期时间格式参数

|  |  |  |
| --- | --- | --- |
| **类别** | **参数** | **说明** |
| 世纪 | CC | 两位的世纪表示。如21 |
| SCC | 两位的世纪表示，如果是公元前则有一个负号（-）。如-10 |
| 季度 | Q | 一位的季度。如1 |
| 年份 | YYYY | 完整的4位年份。如2009 |
| YY | 年份的最后两位数字。如09 |
| YEAR | 年份全拼，全部大写。如TWO THOUSAND NINE |
| 月 | MM | 两位的月份。如01 |
| MONTH | 月份的全拼。右端补齐空格，总长9个字符。如JANUARY |
| MON | 月份的前3个字母。如JAN |
| 周 | WW | 本年份中的第几周。例如32 |
| 日 | DDD | 第几天，3位数字。如100 |
| DD | 第几天，两位数字。如50 |
| D | 第几天，一位数字。如9 |
| DAY | 周几的全拼或文本形式。如MONDAY、星期一 |
| DY | 周几的前3个字母。如MON |
| 时 | HH24 | 24小时格式的小时数，两位数字。如23 |
| HH | 12小时格式的小时数，两位数字。如11 |
| 分 | MI | 两位的分钟数。如55 |
| 秒 | SS | 两位的秒数。如30 |
| MS | 毫秒数。如100 |
| 后缀 | AM或PM | 设置后缀为AM（上午）或PM（下午） |
| A.M.或P.M. | 设置后缀为A.M.或P.M. |

### 分析函数

分析函数根据一组行来计算聚合值。用于计算完成聚集的累计排名、移动平均数。分析函数为每组记录返回多个行。

以下三个分析函数用于计算一个行在一组有序行中的排位，序号从1开始

* **ROW\_NUMBER**

返回连续的排位，不论值是否相等

* **RANK**

具有相等值的行排位相同，序数随后跳跃

* **DENSE\_RANK**

具有相等值的行排位相同，序号是连续的

|  |
| --- |
| SELECT d.dname, e.ename, e.sal, DENSE\_RANK()  OVER (PARTITION BY e.deptno ORDER BY e.sal DESC)  AS DENRANK  FROM emp e, dept d WHERE e.deptno = d.deptno; |

### 通用函数

* NVL()函数

NVL (expr1, expr2)->expr1为NULL，返回expr2；不为NULL，返回expr1。注意两者的类型要一致

|  |
| --- |
| select nvl(comm,0) from emp; |

* NULLIF()函数

如果表达式exp1与exp2的值相等则返回null，否则返回exp1的值

|  |
| --- |
| NULLIF (expr1, expr2) |

* NVL2()函数

NVL2 (expr1, expr2, expr3) ->expr1不为NULL，返回expr2；为NULL，返回expr3。expr2和expr3类型不同的话，expr3会转换为expr2的类型

|  |
| --- |
| select empno, ename, sal, comm, nvl2(comm, sal+comm, sal) total from emp; |

* COALESCE()函数

依次考察各参数表达式，遇到非null值即停止并返回该值。

|  |
| --- |
| select empno, ename, sal, comm, coalesce(sal+comm, sal, 0)总收入 from emp; |

* CASE表达式

|  |
| --- |
| select empno, ename, sal,  case deptno  when 10 then '财务部'  when 20 then '研发部'  when 30 then '销售部'  else '未知部门'  end 部门  from emp; |

* DECODE()函数

和case表达式类似，decode()函数也用于实现多路分支结构

|  |
| --- |
| select empno, ename, sal,  decode(deptno,  10, '财务部',  20, '研发部',  30, '销售部',  '未知部门')  部门  from emp; |

* 单行函数嵌套

|  |
| --- |
| select empno, lpad(initcap(trim(ename)),10,' ') name, job, sal from emp; |

### GROUP BY子句

出现在SELECT列表中的字段或者出现在order by 后面的字段，如果不是包含在分组函数中，那么该字段必须同时在GROUP BY子句中出现。包含在 GROUP BY子句中的字段则不必须出现在SELECT列表中。

可使用where字句限定查询条件

可使用Order by子句指定排序方式

如果没有GROUP BY子句，SELECT列表中不允许出现字段（单行函数）与分组函数混用的情况。

|  |
| --- |
| select empno, sal from emp; //合法  select avg(sal) from emp; //合法  select empno, initcap(ename), avg(sal) from emp; //非法 |

不允许在 WHERE 子句中使用分组函数。

|  |
| --- |
| select deptno, avg(sal)  from emp  where avg(sal) > 2000;  group by deptno; |

### HAVING子句

|  |
| --- |
| select deptno, job, avg(sal)  from emp  where hiredate >= to\_date('1981-05-01','yyyy-mm-dd')  group by deptno,job  having avg(sal) > 1200  order by deptno,job; |

### 实用方法

1. 找出每个月倒数第三天受雇的员工（如：2009-5-29）

|  |
| --- |
| select \* from emp where last\_day(hiredate)-2=hiredate; |

1. 找出25年前雇的员工

|  |
| --- |
| select \* from emp where hiredate<=add\_months(sysdate,-25\*12); |

1. 所有员工名字前加上Dear ,并且名字首字母大写

|  |
| --- |
| select 'Dear ' || initcap(ename) from emp; |

1. 找出姓名为5个字母的员工

|  |
| --- |
| select \* from emp where length(ename)=5; |

1. 找出姓名中不带R这个字母的员工

|  |
| --- |
| select \* from emp where ename not like '%R%'; |

1. 显示所有员工的姓名的第一个字

|  |
| --- |
| select substr(ename,0,1) from emp; |

1. 找到2月份受雇的员工

|  |
| --- |
| select \* from emp where to\_char(hiredate,'fmmm')='2'; |

### 练习

1. 显示所有员工，按名字降序排列，若相同，则按工资升序排序
2. 假设一个月为30天，找出所有员工的日薪，不计小数
3. 列出员工加入公司的天数(四舍五入）
4. 分别用case和decode函数列出员工所在的部门，deptno=10显示'部门10',

deptno=20显示'部门20'

deptno=30显示'部门30'

deptno=40显示'部门40'

否则为'其他部门'

1. 分组统计各部门下工资>500的员工的平均工资
2. 统计各部门下平均工资大于500的部门
3. 算出部门30中得到最多奖金的员工奖金
4. 算出部门30中得到最多奖金的员工姓名
5. 算出每个职位的员工数和最低工资
6. 算出每个部门,，每个职位的平均工资和平均奖金(平均值包括没有奖金)，如果平均奖金大于300，显示“奖金不错”，如果平均奖金100到300，显示“奖金一般”，如果平均奖金小于100，显示“基本没有奖金”，按部门编号降序，平均工资降序排列
7. 列出员工表中每个部门的员工数，和部门no
8. 得到工资大于自己部门平均工资的员工信息
9. 分组统计每个部门下，每种职位的平均奖金（也要算没奖金的人）和总工资(包括奖金)

## 子查询

在外部的SELECT、UPDATE或DELETE语句内部使用SELECT语句，这个内部SELECT语句称为子查询（Subquery）。使用子查询，主要是将子查询的结果作为外部主查询的查找条件。

子查询在其他查询的基础上，提供一种进一步有效的方式来表示条件。子查询是一个SELECT语句，可以在外部SELECT语句的FROM、WHERE、HAVING等子句中使用，另外在UPDATE和DELETE语句中也可以使用子查询。

在子查询中可以使用两种比较操作符：单行操作符和多行操作符。

* + 单行操作符 例如=、>、>=、<、<=、<>、!=
  + 多行操作符 例如ALL、ANY、IN、EXISTS

可以把子查询分为两种类型：单行子查询和多行子查询。

* + 单行子查询 向外部的SQL语句只返回一行数据，或者不返回任何内容。

单行子查询可以放到SELECT语句的WHERE子句和HAVING子句中。

* + 多行子查询 向外部的SQL语句返回多行。要处理返回多行记录的子查

询，外部查询需要使用多行操作符。

另外，子查询还有下面3种子类型，这3种子类型可以返回一行或多行查询结果:

* + 多列子查询 向外部的SQL语句返回多列。
  + 关联子查询 引用外部的SQL语句中的一列或多列。在关联子查询中，可以使用EXISTS和NOT EXISTS操作符。
  + 嵌套子查询 在子查询中包含有子查询。

指定子查询时，需要注意以下几点：

* 子查询需要使用括号（()）括起来。
* 子查询要放在比较操作符的右边。
* 当子查询的返回值是一个集合而不是一个值时，不能使用单行操作符， 而必须根据需要使用ANY、IN、ALL或EXISTS等操作符。

### 使用子查询

#### 在FROM子句中使用子查询

当在FROM子句中使用子查询时，该子句会被作为视图对待，因此也被称为内嵌视图。使用子查询的一般用法形式如下：

|  |
| --- |
| SELECT column\_list FROM (  SELECT column\_name FROM table\_name WHERE condition  GROUP BY exp HAVING having); |

例如：

|  |
| --- |
| SELECT deptno FROM (  SELECT deptno FROM scott.emp GROUP BY deptno ORDER BY COUNT(deptno) ASC  ); |

#### 在WHERE子句中使用子查询

在SELECT语句的WHERE子句中可以使用子查询，表示将子查询返回的结果作为外部的WHERE条件。

|  |
| --- |
| SQL> SELECT empno , ename , sal , deptno FROM scott.emp  2 WHERE deptno = (  3 SELECT deptno FROM scott.emp WHERE empno = 7782 ); |

#### 在HAVING子句中使用子查询

在SELECT语句中使用HAVING子句，可以实现对数据进行分组过滤。在HAVING子句中，如果使用子查询，那么就可以实现对子查询返回的结果根据分组进行过滤。

|  |
| --- |
| SQL> SELECT deptno , AVG(sal) FROM scott.emp GROUP BY deptno  2 HAVING AVG(sal) < (  3 SELECT AVG(sal) FROM scott.emp ); |

#### 在UPDATE语句中使用子查询

在UPDATE语句中使用子查询，可以将子查询返回的结果赋值给需要更新的列。

【例】将员工编号为7839的员工的工资设置为平均工资，如下：

|  |
| --- |
| SQL> UPDATE scott.emp SET sal = (  2 SELECT AVG(sal) FROM scott.emp )  3 WHERE empno = 7839 ; |

#### 在DELETE语句中使用子查询

在DELETE语句中使用子查询，可以根据子查询返回的结果删除指定的行。

【例】删除工作地点在NEW YORK的所有员工信息，如下：

|  |
| --- |
| SQL> DELETE FROM scott.emp WHERE deptno IN(  2 SELECT deptno FROM scott.dept WHERE loc = 'NEW YORK') ; |

### IN\NOT IN操作符

多行子查询可以向外部的SQL语句返回一行或多行记录。要处理返回多行记录的子查询，外部查询需要使用多行操作符。

使用IN操作符，用来检查在一个值列表中是否包含指定的值。这个值列表可以是子查询的返回结果。

【例】

|  |
| --- |
| SQL> SELECT empno , ename , sal , deptno FROM scott.emp  2 WHERE deptno IN(  3 SELECT deptno FROM scott.dept  4 WHERE dname IN ('ACCOUNTING' , 'SALES')) ; |

NOT IN操作符用来检查在一个值列表中是否不包含指定的值，NOT IN执行的操作正好与IN在逻辑上相反。

【例】

|  |
| --- |
| SQL> SELECT empno , ename , sal , deptno FROM scott.emp  2 WHERE deptno NOT IN (  3 SELECT deptno FROM scott.dept  4 WHERE dname IN ('ACCOUNTING' , 'SALES')); |

### ANY和ALL操作符

在进行多行子查询时，使用ANY操作符，用来将一个值与一个列表中的所有值进行比较，这个值只需要匹配列表中的一个值即可，然后将满足条件的数据返回。其中，值列表可以是子查询的返回结果。

使用ANY操作符之前，必须使用一个单行操作符，例如=、>、<、<=等。

【例】

|  |
| --- |
| SQL> SELECT empno , ename , sal , deptno FROM scott.emp  2 WHERE sal > ANY (  3 SELECT AVG(sal) FROM scott.emp GROUP BY deptno) ; |

在进行子查询时，使用ALL操作符，用来将一个值与一个列表中的所有值进行比较，这个值需要匹配列表中的所有值，然后将满足条件的数据返回。其中，值列表可以是子查询的返回结果。

使用ALL操作符之前，必须使用一个单行操作符，例如=、>、<、<=等。

【例】

|  |
| --- |
| SQL> SELECT empno , ename , sal , deptno FROM scott.emp  2 WHERE sal > ALL (  3 SELECT AVG(sal) FROM scott.emp GROUP BY deptno); |

### 常见查询错误

#### ORDER BY子句

在子查询的SELECT语句中，可以使用FROM子句、WHERE子句、GROUP BY子句和HAVING子句等，但是有些情况下不能使用ORDER BY子句，例如在WHERE子句中使用子查询时，子查询语句中就不能使用ORDER BY子句。

【例】

|  |
| --- |
| SQL> SELECT empno , ename , sal , deptno FROM scott.emp  2 WHERE deptno IN (  3 SELECT deptno FROM scott.emp  4 WHERE empno > 7782 ORDER BY deptno DESC) ; |

#### 操作符错误

多行子查询可以返回多行记录，如果接收子查询结果的操作符是单行操作符，那么在执行语句时，可能会出现错误提示。

【例】

|  |
| --- |
| SQL> SELECT empno , ename , sal , deptno FROM scott.emp  2 WHERE deptno = (  3 SELECT deptno FROM scott.dept  4 WHERE dname IN ('ACCOUNTING' , 'SALES')) ; |

### 多列子查询

* 概述

单行子查询是指子查询只返回单行单列数据；多行子查询是指子查询返回多行单列数据，二者都是针对单列而言。所以前面的示例中，子查询都是只返回单个的列。

多列子查询则是指返回多列数据的子查询语句。当多列子查询返回单行数据时，在WHERE子句中可以使用单行操作符；返回多行数据时，在WHERE子句中必须使用多行操作符。

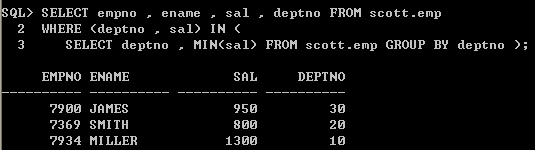
* 使用方式

使用子查询比较多个列的数据时，可以使用下面两种方式：

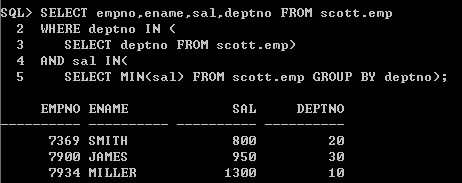
* 成对比较 要求多个列的数据必须同时匹配。
* 非成对比较 通过指定连接关键字，例如AND或OR等，指定多个列的数据是否必须同时匹配。如果使用AND关键字，表示同时匹配，这样就可以实现与成对比较同样的结果；如果使用OR关键字，表示不必同时匹 配。

【例】

成对比较



非成对比较



### 关联子查询

关联子查询会引用外部查询中的一列或多列，这种子查询之所以被称为关联子查询，是因为它的确与外部语句相关。具体实现时，外部查询中的每一行都传递给子查询，子查询依次读取传递过来的每一行的值，并将其应用到子查询上，直到外部查询中的所有行都处理完为止，然后返回子查询的结果。

【例】查询各个部门中，哪些员工的工资低于其所在部门的平均工资。

|  |
| --- |
| SQL> SELECT empno , ename , sal , deptno FROM scott.emp outer  2 WHERE sal < (  3 SELECT AVG(sal) FROM scott.emp inner  4 WHERE inner.deptno = outer.deptno ) ; |

#### EXISTS操作符

在关联子查询中可以使用EXISTS或NOT EXISTS操作符。其中，EXISTS操作符用于检查子查询所返回的行是否存在，它可以在非关联子查询中使用，但是更常用于关联子查询。

使用EXISTS操作符，只是检查子查询返回的数据是否存在，因此，在子查询语句中可以不返回一列，而返回一个常量值，这样可以提高查询的性能。

在执行的操作逻辑上，NOT EXISTS操作符的作用与EXISTS操作符相反。在需要检查数据行中是否不存在子查询返回的结果时，就可以使用NOT EXISTS。

【例】

|  |
| --- |
| SQL> SELECT empno , ename , sal , hiredate  2 FROM scott.emp outer WHERE NOT EXISTS (  3 SELECT 1 FROM scott.emp inner  4 WHERE outer.empno = inner.empno  5 AND SUBSTR(hiredate,8,2) = '80'); |

### 嵌套子查询

所谓嵌套子查询，是指在子查询内部使用其他子查询。嵌套子查询的嵌套层次最多为255层。大多数情况下，嵌套子查询都在外层子查询的WHERE子句中。子查询的执行顺序是从内层到外层。

【例】已知工作地点在NEW YORK和CHICAGO的两个部门，要求根据这两个部门的平均工资中的最大值，获取工资大于这个最大值的的员工信息 ：

|  |
| --- |
| SQL> SELECT empno , ename , sal , deptno FROM scott.emp  2 WHERE sal > (  3 SELECT MAX(AVG(sal)) FROM scott.emp WHERE deptno IN (  4 SELECT deptno FROM scott.dept  5 WHERE loc IN ('NEW YORK' , 'CHICAGO') )  6 GROUP BY deptno ) ; |

## 高级查询

### UNION操作符

集合操作符可以将两个或多个查询返回的结果组合起来，常用的集合操作符包括：UNION、UNION ALL、INTERSECT和MINUS。这些集合操作符具有相同的优先级，当同时使用多个操作符时，会按照从左到右的方式引用这些集合操作符。当使用集合操作符时，必须确保不同查询的列个数和数据类型都要匹配。

使用集合操作符可以将两个或者多个查询返回的行组合起来。 使用UNION操作符获取两个结果集的并集，语法如下：

|  |
| --- |
| select\_statement UNION [ ALL ] select\_statement  [UNION [ ALL ] select\_statement ] [ … ] |

将多个查询结果集合并，形成一个新的结果集。如果指定ALL，则包括重复的行；如果不指定ALL，则对重复的行只保留一行。相当于结果集的OR运算。



|  |
| --- |
| SQL> SELECT empno , ename , sal , deptno FROM scott.emp  2 WHERE empno > 7800  3 UNION ALL  4 SELECT empno , ename , sal , deptno FROM scott.emp  5 WHERE deptno = 10 ORDER BY deptno ASC ; |

INTERSECT和MINUS和操作符见十八单SQL操作符的集合操作符内容。

### 层次化查询

层次化查询有时也称为CONNECT BY查询，是Oracle特有的特性。这类查询可以选取数据，并且使用层次化的次序返回结果。

【例】

EMP表中包含了一列MGR，指出谁是雇员的经理，由于经理也是雇员，所以经理的信息也存储在EMP表中。这样EMP表就是一个自引用表，表中的MGR列是一个自引用列，它指回EMP表中的EMPNO列，MGR表示一个员工的管理者（如果有的话）。

介绍了EMP表中雇员和管理者之间的关系，如果想清楚的显示雇员和管理者的关系，可以使用CONNECT BY和START WITH子句。

|  |
| --- |
| SELECT [LEVEL],column,expression,…  FROM table  [WHERE where\_clause]  [[START WITH start\_condition] [CONNECT BY PRIOR prior\_condition]]; |

其中：

* LEVEL 表示一个“伪列”，代表树的第几层。
* start\_conditon 定义了层次化查询的起点。当编写层次化查询时必须指定START WITH子句。
* prior\_condition 定义了父行和子行之间的关系，当编写层次化查询时 必须定义CONNECT BY PRIOR子句。
* 树

根节点 根节点是位于树顶端的节点。图中根节点是KING。

父节点 父节点的下面有一个或多个节点。例如，KING是下列节点的父节点：JONES、BLAKE、CLARK。

子节点 子节点之上有一个父节点。例如，JONES的父节点是KING。

叶节点 叶节点是没有子节点的节点。例如，ADAMS和MILLER。



### 格式化查询

#### 使用伪列LEVEL

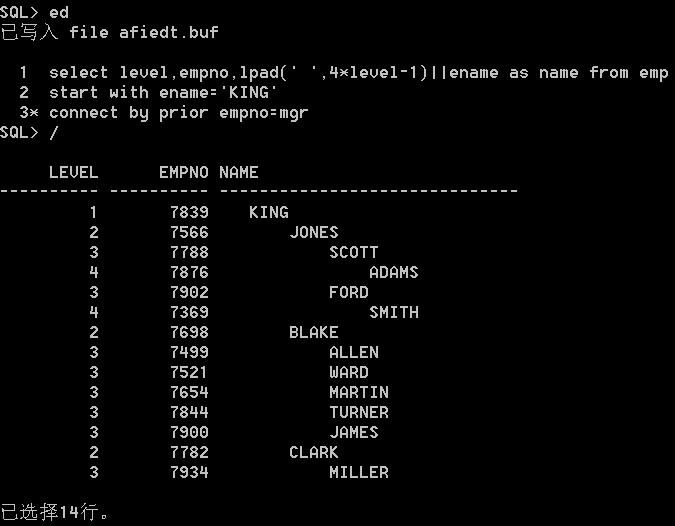
LEVEL是存在于Oracle所执行的所有查询中的伪列，它是一个数值，可以指出节点在树中所处的层次。在层次化查询中，LEVEL值会将起始的根节点作为层次1。下面这个查询使用伪列LEVEL显示节点在树中的层次：

|  |
| --- |
| SQL> select level,empno,mgr,ename from emp  2 start with ename='KING'  3 connect by prior empno=mgr  4 order by level; |

#### 格式化层次化查询的结果

可以用LEVEL和LPAD()函数对层次化查询结果进行格式化处理，方法是在数据的左边填充空格。下面这个查询根据不同LEVEL填充不同个数的空格，从而缩进显示员工的名字。示例如下：

|  |
| --- |
| SQL> select level,lpad(' ',4\*level-1)||ename  2 from emp  3 start with ename='KING'  4 connect by prior empno=mgr; |



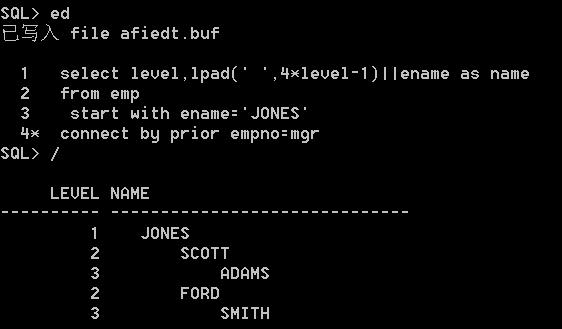
### 遍历查询

#### 从非根节点开始遍历

上面的查询都是从根节点KING开始查询，可以使用START WITH子句从任何节点开始查询。下面这个查询从JONES开始查询，

示例如下：

|  |
| --- |
| SQL> select level,lpad(' ',4\*level-1)||ename  2 from emp  3 start with ename='JONES'  4 connect by prior empno=mgr; |



#### 从下向上遍历树

不一定非要按照从父节点到子节点的顺序从上至下遍历树，也可以从某个子节点开始，自下而上遍历。实现的方法是交换父节点与子节点在CONNECT BY PRIOR子句中的顺序。例如，CONENCT BY PRIOR MGR=EMPNO可以将子节点的MGR连接到父节点的EMPNO上。

下面这个查询从SMITH开始，向上遍历，直到根节点KING：

|  |
| --- |
| SQL> select level,lpad(' ',4\*level-1)||ename  2 from emp  3 start with ename='SMITH'  4 connect by prior mgr=empno; |

#### 从层次化查询中删除节点和分支

可以用WHERE子句从查询树中除去某个特定的节点，下面这个查询使用WHERE ENAME!='JONES'子句从结果中除去JONES子节点。示例如下：

|  |
| --- |
| SQL> select level,lpad(' ',4\*level-1)||ename  2 from emp  3 where ename!='JONES'  4 start with ename='KING'  5 connect by prior empno=mgr; |

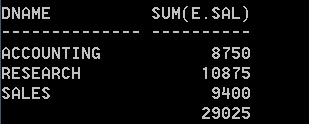
### 扩展Group By子句

#### ROLLUP子句

ROLLUP子句扩展GROUP BY子句，为每一个分组返回一条小计记录，并为全部分组返回总计。

GROUP BY可以根据列值对记录进行分组。例如，下面这个查询计算每个部门的工资总计，首先要对EMP表和DEPT表进行连接查询，然后使用GROUP BY按部门名称进行分组，最后使用SUM()函数得到每个部门的工资总计。ROLLUP获得所有部门的工资总计。示例如下：

|  |
| --- |
| SQL> select d.dname,sum(e.sal)  2 from emp e,dept d  3 where e.deptno=d.deptno  4 group by rollup(d.dname)  5 order by d.dname; |



#### CUBE子句

CUBE子句对GROUP BY进行扩展，返回CUBE中所有列组合的小计信息，同时在最后显示总计信息。下面这个例子将DNAME和JOB传递给CUBE，查询部门名称、岗位和工资总计，示例如下：

|  |
| --- |
| SQL> select e.job,d.dname,sum(e.sal)  2 from emp e,dept d  3 where e.deptno=d.deptno  4 group by cube(e.job,d.dname)  5 order by e.job,d.dname; |

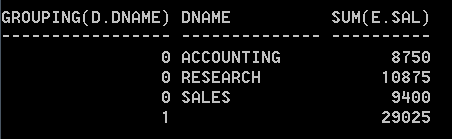


#### GROUPING()函数

GROUPING()函数可以接受一列，返回0或者1。如果列值为空，那么GROUPING()函数返回1；如果列值非空，则返回0。GROUPING()函数只能在使用ROLLUP或CUBE的查询中使用。当需要在返回空值的地方显示某个值时，GROUPING()函数就非常有用。

下面这个查询使用GROUPING()函数来确定列值是否为空，如果具有非空列值的记录，GROUPING()函数返回0；否则，返回1：

|  |
| --- |
| SQL> select grouping(d.dname),d.dname,sum(e.sal)  2 from emp e,dept d  3 where e.deptno=d.deptno  4 group by rollup(d.dname)  5 order by d.dname; |



## 模式对象

### 同义词

#### 概述

同义词，是表、索引和视图等模式对象的一个别名。与视图等一样，同义词并不占用任何实际的存储空间，只在Oracle的数据字典中保存其定义描述。在使用同义词时，Oracle会将其翻译为对应对象的名称。

Oracle中的同义词主要分为如下两类：

* 公有同义词

在数据库中的所有用户都可以使用。

* 私有同义词

创建它的用户私人拥有。不过，用户可以控制其他用户是否有权使用自己的同义词。

#### 创建同义词

创建同义词的语法如下：

|  |
| --- |
| CREATE [PUBLIC] SYNONYM synonym\_name FOR schema\_object |

其中，PUBLIC关键字用来指定创建的同义词是否为公有同义词；

synonym\_name为创建的同义词名称；

schema\_object为同义词所针对的对象。

#### 删除同义词

|  |
| --- |
| DROP [PUBLIC] SYNONYM synonym\_name |

### 序列

#### 概述

是Oracle提供的用于产生一系列惟一数字的数据库对象。使用序列可以实现自动产生主键值。序列也可以在多用户并发环境中使用，为所有用户生成不重复的顺序数字，而且不需要任何额外的I/O开销。

与视图一样，序列并不占用实际的存储空间，只是在数据字典中保存它的定义信息。用户要在自己的模式中创建序列，必须具有CREATE SEQUENCE系统权限；如果要在其他模式中创建序列，则必须具有CREATE ANY SEQUENCE系统权限。

#### 创建序列

使用CREATE SEQUENCE语句创建序列的语法如下：

|  |
| --- |
| CREATE SEQUENCE sequence\_name  [START WITH start]  [INCREMENT BY increment]  [MINVALUE minvalue | NOMINVALUE]  [MAXVALUE maxvalue | NOMAXVALUE]  [CACHE cache | NOCACHE]  [CYCLE | NOCYCLE]  [ORDER | NOORDER] |

在上述语法中，各组成部分的含义如下：

* sequence\_name用来指定待创建的序列名称。
* start用来指定序列的开始位置。在默认情况下，递增序列的起始值为MINVALUE，递减序列的起始值为MAXVALUE。
* increment用来表示序列的增量。该参数值为正数，则生成一个递增序列，为负数则生成一个递减序列。其默认值为1。
* minvalue用来指定序列中的最小值。
* maxvalue用来指定序列中的最大值。
* CACHE | NOCACHE 指定是否产生序列号预分配，并存储在内存中。
* CYCLE | NOCYCLE 用来指定当序列达到MAXVALUE或MINVALUE 时，是否可复位并继续下去。如果使用CYCLE，则如果达到极限，生成的下一个数据将分别是MINVALUE或者MAXVALUE；如果使用NOCYCLE，则如果达到极限并试图获取下一个值时，将返回一个错误。
* ORDER | NOORDER 用来指定是否可以保证生成的序列值是按顺序 产生的。如果使用ORDER，则可以保证；而如果使用NOORDER，则只能保证序列值的惟一性，而不能保证序列值的顺序。

#### 管理序列

使用序列时，需要使用序列的两个伪列NEXTVAL和CURRVAL。其中，NEXTVAL将返回序列生成的下一值；CURRVAL返回序列的当前值。例如，stu\_seq.NEXTVAL，stu\_seq.CURRVAL.

修改序列使用ALTER SEQUENCE语句，除了序列的起始值之外，可以重新定义序列的任何子句和参数进行修改。如果要修改序列的起始值，则必须先删除序列，然后再重新创建该序列。对序列进行修改后，在缓存中的序列值将全部丢失，可以通过数据字典USER\_SEQUENCES获取序列的信息。

### 索引

#### 索引及其分类

在关系数据库中，索引是一种与表有关的数据库结构，它是除表以外的另一个重要模式对象。索引是建立在表的一列或多个列上的辅助对象，目的是提高表中数据的访问速度。

索引是表示数据的另一种方式，它提供的数据顺序不同于数据在磁盘上的物理存储顺序。索引的特殊作用是在表内重新排列记录的物理位置。

Oracle中常用的索引类型有：B树索引、反向键索引、位图索引、基于函数的索引、簇索引、全局索引和局部索引等。

#### 创建索引语法

|  |
| --- |
| CREATE UNIQUE | BTIMAP INDEX <schema>.<index\_name>  ON <schema>.<table\_name>  (<column\_name> | <expression> ASC | DESC,  <column\_name> | <expression> ASC | DESC,…)  TABLESPACE<tablespace\_name>  STORAGE<storage\_settings>  LOGGING | NOLOGGING  COMPUTE STATISTICS  NOCOMPRESS | COMPRESS<nn>  NOSORT | REVERSE  PARTITION | GLOBAL PARTITION<partition\_setting>; |

* 语法说明:

语法中各关键字或子句的含义如下：

* UNIQUE | BITMAP

在创建索引时，如果指定关键字UNIQUE，则要求表中的每一行在索引都包含惟一的值；如果指定BITMAP关键字，将创建一个位图索引；如果都省略，则默认创建B树索引。

* ASC 表示该列为升序排列。ASC为默认排列顺序。
* DESC 表示该列为降序排列。
* TABLESPACE 用来在创建索引时，为索引指定存储空间。
* STORAGE 用户可以使用该子句来进一步设置存储索引的表空间存储参数，以取代表空间的默认存储参数。
* LOGGING | NOLOGGING

LOGGING用来指定在创建索引时创建相应的日志记录；NOLOGGING则用来指定不创建相应的日志记录。默认使用LOGGING。

* COMPUTE STATISTICS

用来指定在创建索引的过程中直接生成关于索引的统计信息。这样可以避免以后再对索引进行分析操作。

* NOCOMPRESS | COMPRESS<nn>

COMPRESS用来指定在创建索引时对重复的索引值进行压缩，以节省索引的存储空间；NOCOMPRESS则用来指定不进行任何压缩。默认使用NOCOMPRESS。

* NOSORT | REVERSE

NOSORT用来指定在创建索引时，Oracle将使用与表中相同的顺序来创 建索引，省略再次对索引进行排序的操作；REVERSE则指定以相反的 顺序存储索引值。

如果表中行的顺序与索引期望的顺序不一致，则使用NOSORT子句将会导致索引创建失败。

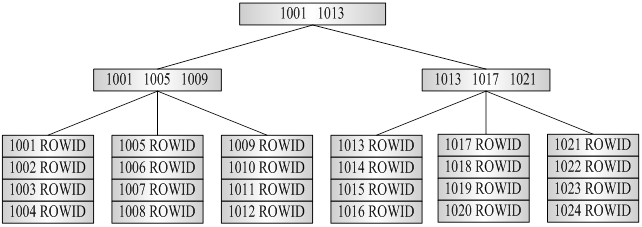
* PARTITION | NOPARTITION

使用该子句，可以在分区表和未分区表上对创建的索引进行分区。

#### B树索引

##### 概述

B树索引是Oracle中默认并且最常用的索引。其逻辑结构图如图所示。



从B树索引的逻辑结构图可以看出，B树索引的组织结构类似于一棵树，其中主要数据都集中在叶子节点上。每个叶子节点中包括：索引列的值和记录行对应的物理地址ROWID。

##### 搜索过程

采用B树索引可以确保无论索引条目位于何处，Oracle都只需要花费相同的I/O就可以获取它。

例如，采用上述B树索引搜索编号为1008的节点，其搜索过程如下：

（1）访问根节点，比较出1008大于1001，但是小于1013。

（2）搜索左边分支，从左边分支中判断出1008大于1005，但是小于1009。

（3）搜索左边分支的第二个叶子节点，找到要查询的索引条目，并根据索引条目中的ROWID进而找到所要查询的记录位置。

（4）从表中获取相关数据。

使用B树索引只需要四步：

1. 访问根节点
2. 访问分支节点
3. 访问叶子节点
4. 获得数据

##### 创建B树索引及分类

创建一个B树索引，需要使用CREATE INDEX语句。如果用户要在自己的模式中创建索引，则必须具有CREATE INDEX系统权限；如果用户想要在其他用户模式中创建索引，则必须具有CREATE ANY INDEX系统权限。

1. 创建普通索引

创建索引时，在ON关键字后面指定索引基于的表名和列名，使用TABLESPACE指定存储索引的表空间。

默认情况下，当用户为表定义一个主键时，系统将自动为该列创建一个B树索引。另外，当一个列已经包含索引时，则无法再在该列上创建索引。

1. 创建惟一索引

索引可以是惟一的，也可以是不惟一的。惟一的B树索引可以保证索引列上不会有重复的值。创建惟一索引需要使用UNIQUE关键字。

1. 创建复合索引

复合索引，是指基于表中多个字段的索引。

* 老夏问答：

Q：什么情况下会使用复合索引？

1. 当SQL语句的where子句中有用到复合索引的领导字段时，Oracle优化器会考虑使用复合索引来访问
2. 当某几个字段在SQL语句的where子句中经常通过and操作符联合在一起使用作为过滤词，并且这几个字段合在一起时选择性比各自单个字段的选择性要更好时，可能考虑使用这个几个字段来建立复合索引。
3. 当有几个查询语句都是查询同样的几个字段值时，则可以考虑在这几个字段上建立复合索引。

Q：复合索引的字段排序原则

1. 确保在where子句中使用的字段是复合索引的领导字段。
2. 如果某个字段在where子句中最被频繁使用，则在建立复合索引时，考虑将这个字段排在第一位(create index时)。
3. 如果所有的字段在where子句中使用频率相同，则将最具选择性的字段排在前面，将最不具选择性的字段排在后面。
4. 如果所有的字段在where子句中使用频率相同，如果数据在物理上是按某个字段排序的，则考虑将这个字段排在第一位。

* 示例

1. 创建普通B树索引



1. 创建唯一索引



1. 创建复合索引

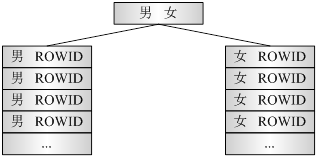


#### 位图索引

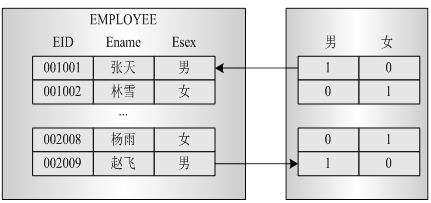
位图索引不同于B树索引，它不存储ROWID值，也不存储键值，主要用于在比较特殊的列上创建索引。

当列的基数很低时，为其建立B树所有显然已经不合适。“基数低”表示在索引列中，所有列值的数量比表中行的数量少。例如“性别”列只有2个取值。

Oracle建议，当一个列的所有取值数量与行的总数比小于1%时，对该列也就不再适合建立B树索引，而适用于位图索引。



例如有一个EMPLOYEE员工表，在该表中有一列是员工性别，该列的取值只有两种：男或女。如果在该列上使用B树索引，创建的B树将只有男分支和女分支。可以选取在该列上创建位图索引。位图索引以及对应的表行概念示意图如图所示。



位图索引适用于在表中基数较小的列上创建。创建位图索引需要使用BITMAP关键字，如下：

|  |
| --- |
| SQL> create bitmap index esex\_bitmap\_index on employee(esex)  2 tablespace testspace;  索引已创建。 |

在表上放置单独的位图索引是没有意义的。只有对多个列建立位图索引，系统才可以有效地利用它们提高查询的速度。

**位图索引不能是惟一索引，也不能对其进行键压缩。**

位图索引的作用来源于与其他位图索引的结合。当在多个列上进行查询时，Oracle对这些列上的位图进行布尔AND和OR运算，最终找到所需要的结果。

* 老夏问答

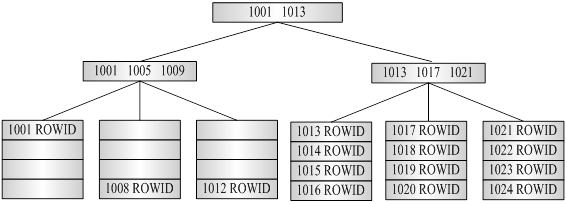
Q：什么情况下位图索引能改善查询性能呢？

1. where子句包含多个词语中低基数的字段。
2. 单个词在这些中低基数的字段上选取大量的行
3. 已经有位图索引创建某些或全部的这些中低基数的字段上
4. 被查询的表包含很多行
5. 可以在单一个表上建立多个位图索引，因此，位图索引能够改善包含冗长where子句的复杂查询的性能，在合计查询和星形模型的联系查询语句中，位图索引可以提供比较优良的性能。

#### 反向键索引

反向键索引是一种特殊的B树索引，适用于在含有序列数的列上创建索引。

在常规的B树索引中，如果主键列是递增的，那么在向表中添加新的数据时，B树索引将直接访问最后一个数据，而不是一个节点一个节点的访问。这种情况造成的现象是，随着数据行的不断增加，以及原有数据行的删除，B树索引将变得越来越不均匀。不均匀的B树索引如下图：



这时，可以选择在该列上创建反向键索引。反向键索引的工作原理是：

如果用户使用序列编号在表中添加新的记录，则反向键索引首先反向转换每个列键值的字节，然后在反向后的新数据上进行索引。

例如，如果用户输入的索引键为2009，则反向键索引将其反向转换为9002。这样可以将索引键变成非递增的，也就意味着，如果将这个索引键添加到叶子节点中，则可能会在任意的叶子节点中进行，从而使得新的数据在值的范围分布上比原来更均匀。

* 创建反射键索引

反向键索引适用于在表中严格排序的列上创建。创建反向键索引需要使用REVERSE关键字，如下：

|  |
| --- |
| SQL> create index eid\_reverse\_index on employee(eid)  2 reverse  3 tablespace huizhi;  索引已创建。 |

上面为EMPLOYEE表的EID列，创建了名为EID\_REVERSE\_INDEX的反向键索引。

在查询时，用户只需要像常规方式一样查询数据，而不需要关心键的反向处理，系统会自动完成该处理。

#### 基于函数的索引

基于函数的索引只是常规的B树索引，只不过它存放的数据是由表中的数据应用函数后得到的，而不是直接存放表中的数据本身。

在Oracle中，经常遇到字符大小写或数据类型转换等问题。这时，就可以引用函数对这些数据进行转换。

例如，有一个表EMPLOYEE，在这个表的ENAME列中有一个值XIAOXUE。如果使用小写的字符串xiaoxue查找该员工记录，则无法找到。

查询如下：

|  |
| --- |
| SQL> select \* from employee where ename = ‘xiaoxue';  未选定行 |

这时，可以引用函数来解决这个问题。例如，引用UPPER()函数，将查询时遇到的每个值都转换成大写。查询如下：

|  |
| --- |
| SQL> select \* from employee where upper(ename) = upper(‘xiaoxue');  EID ENAME  ---------- --------------  1 XIAOXUE |

使用这种查询方式，用户不是基于表中存储的记录进行搜索的。虽然只在某一列上建立了索引，但Oracle会被迫执行全表搜索，为所遇到的各个行都计算UPPER()函数。

创建基于函数的索引，可以提高在查询条件中使用函数和表达式时查询的执行速度。如果用户要在自己的模式中创建基于函数的索引，则必须具有QUERY REWRITE系统权限；如果用户想要其他模式中创建基于函数的索引，则必须具有CREATE ANY INDEX和GLOBAL QUERY REWRITE权限。

创建基于函数的索引时，Oracle会首先对包含索引列的函数值或表达式值进行求值，然后对求值后的结果进行排序，最后存储到索引中。

* 示例

例如，为EMPLOYEE表中的EDATE列创建一个基于函数TO\_CHAR()的函数索引，如下：

|  |
| --- |
| SQL>create index edate\_ index on employee(to\_char(edate,'YYYY-MM-DD'))  2 tablespace huizhi;  索引已创建。 |

上面为EMPLOYEE表中的EDATE列，创建了一个名为EDATE\_ NDEX的函数索引。创建该索引后，如果在查询条件中包含有相同的函数，则可以提高查询的执行速度。下面的查询将会使用EDATE\_FUNC\_INDEX索引：

|  |
| --- |
| SQL> select edate from employee  2 where to\_char(edate,'YYYY-MM-DD') = '2009-02-02'; |

#### 管理索引

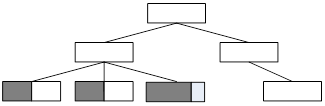
* 合并索引

随着对表的不断更新，在表的索引中将会产生越来越多的存储碎片，这些碎片会影响索引的使用效率。合并索引和重建索引是两种清除索引存储碎片的方式。

合并索引是指将B树中叶子节点的存储碎片合并在一起，这种合并不会

改变索引的物理组织结构。

|  |
| --- |
| SQL> ALTER index ename\_index coalesce deallocate unused;  索引已更改。 |



* 重建索引

重建索引在消除存储碎片的同时，还可以改变索引的全部存储参数设置，以及改变索引的存储表空间。重建索引需要使用ALTER INDEX … REBUILD语句。

例如，对索引ENAME\_INDEX进行重建，如下：

|  |
| --- |
| SQL> alter index ename\_index rebuild  2 tablespace mytemp;  索引已更改。 |

上面在对索引ENAME\_INDEX进行重建的同时，修改了该索引存储的表空间。

重建索引，实际上是在指定的表空间中重新建立一个新的索引，然后再删除原来的索引。

* 监视索引

监视索引，需要使用ALTER INDEX ... MONITORING USAGE语句，目的是为了确保索引得到有效的利用。

例如，要查看索引ENAME\_INDEX的使用情况，可以打开该索引的监视状态，如下：

|  |
| --- |
| SQL> alter index ename\_index monitoring usage;  索引已更改。 |

如果想关闭索引的监视状态，可以使用ALTER INDEX ... NOMONITORING USAGE语句。例如，关闭索引ENAME\_INDEX的监视状态，如下：

|  |
| --- |
| SQL> alter index ename\_index nomonitoring usage;  索引已更改。 |

* 删除索引

用户可以删除自己模式中的索引。如果要删除其他模式中的索引，则必须具有DROP ANY INDEX系统权限。

一个索引被删除后，它所占用的盘区会全部返回给它所在的表空间，并且可以被表空间中的其他对象使用。通常在如下情况下需要删除某个索引：

* 该索引不需要再使用。
* 该索引很少被使用。索引的使用情况可以通过监视来查看。
* 该索引中包含较多的存储碎片，需要重建该索引。

删除使用CREATE INDEX语句创建的索引。如果索引是使用CREATE INDEX语句显示创建的，则可以使用DROP INDEX语句删除该索引。例如删除索引ENAME\_INDEX，如下：

|  |
| --- |
| SQL> drop index ename\_index;  索引已删除。 |

* 示例：

创建索引



合并索引



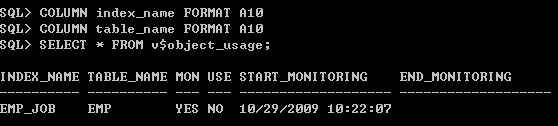
重建索引



监视索引



查看索引状态



关闭监视



删除索引



#### 索引优缺点

* 建立索引的优点

1. 大大加快数据的检索速度;
2. 创建唯一性索引，保证数据库表中每一行数据的唯一性;
3. 加速表和表之间的连接;
4. 在使用分组和排序子句进行数据检索时，可以显著减少查询中分组和排序的时间。

* 索引的缺点

1. 索引需要占物理空间。
2. 当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，降低了数据的维护速度。

#### 创建索引的原则

创建索引一般有以下两个目的：

维护被索引列的唯一性和提供快速访问表中数据的策略。

* 在select操作占大部分的表上创建索引；
* 在where子句中出现最频繁的列上创建索引；
* 在选择性高的列上创建索引（补充索引选择性，最高是1，eg：primary key）
* 复合索引的主列应该是最有选择性的和where限定条件最常用的列，并以此类推第二列……。
* 小于5M的表，最好不要使用索引来查询，表越小，越适合用全表扫描。
* 使用索引的原则
* 查询结果是所有数据行的5%以下时，使用index查询效果最好；
* where条件中经常用到表的多列时，使用复合索引效果会好于几个单列索引。因为当sql 语句所查询的列，全部都出现在复合索引中时，此时由于 Oracle 只需要查询索引块即可获得所有数据，当然比使用多个单列索引要快得多；
* 索引利于select，但对经常insert，delte尤其update的表，会降低效率。
* eg：试比较下面两条SQL语句(emp 表的deptno列上建有ununique index)： 语句A：

|  |
| --- |
| SELECT dname, deptno FROM dept WHERE deptno NOT IN (SELECT deptno FROM emp); |

语句B：

|  |
| --- |
| SELECT dname, deptno FROM dept WHERE NOT EXISTS (SELECT deptno FROM emp WHERE dept.deptno = emp.deptno); |

这两条查询语句实现的结果是相同的，但是执行语句A的时候，ORACLE会对整个emp表进行扫描，没有使用建立在emp表上的deptno索引，执行语句B的时候，由于在子查询中使用了联合查询，ORACLE只是对emp表进行的部分数据扫描，并利用了deptno列的索引，所以语句B的效率要比语句A的效率高。

* where 子句中的这个字段，必须是复合索引的第一个字段；

eg：一个索引是按f1, f2, f3的次序建立的，若where 子句是f2 = : var2, 则因为f2 不是索引的第1个字段，无法使用该索引。

* where 子句中的这个字段，不应该参与任何形式的计算：任何对列的操作都将导致表扫描，它包括数据库函数、计算表达式等等，查询时要尽可能将操作移至等号右边。
* 应尽量熟悉各种操作符对 Oracle 是否使用索引的影响：以下这些操作会显式（explicitly）地阻止 Oracle 使用索引： is null ; is not null ; not in; !=; like ; numeric\_col+0;date\_col+0; char\_col||' '; to\_char; to\_number，to\_date 等。 Eg： Select jobid from mytabs where isReq='0' and to\_date (updatedate) >= to\_Date ( '2001-7-18', 'YYYY-MM-DD')；
* updatedate列的索引也不会生效。

### 视图

#### 概述

视图是一个虚拟表，它同真实表一样包含一系列带有名称的列和行数据。但是视图并不在数据库中存储数据值，它的行和列中的数据来自于定义视图的查询语句中所使用的表，数据库只在数据字典中存储了视图定义本身。

用户可以在视图中进行INSERT、UPDATE和DELETE操作。通过视图修改数据时，实际上是在修改基本表中存储的数据。

视图可以建立在关系表上，也可以建立在其他视图上，或者同时建立在两者之上。创建视图，主是要为了将一个或多个表中经常使用到的数据结合到一起，方便用户的数据操作。 一个比较简单的视图，只能通过它从基本表中检索数据，而不能通过它修改基本表中的数据。创建视图最基本的语法如下：

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

其中：

OR REPLACE ：若所创建的试图已经存在，ORACLE自动重建该视图；

FORCE ：不管基表是否存在ORACLE都会自动创建该视图；

NOFORCE ：只有基表都存在ORACLE才会创建该视图：

alias ：为视图产生的列定义的别名；

subquery ：一条完整的SELECT语句，可以在该语句中定义别名；

WITH CHECK OPTION ：

插入或修改的数据行必须满足视图定义的约束；

WITH READ ONLY ：

该视图上不能进行任何DML操作。

例如：

|  |
| --- |
| CREATE OR REPLACE VIEW dept\_sum\_vw  (name,minsal,maxsal,avgsal)  AS SELECT d.dname,min(e.sal),max(e.sal),avg(e.sal)  FROM emp e,dept d  WHERE e.deptno=d.deptno  GROUP BY d.dname; |

#### 视图的定义原则：

1. 视图的查询可以使用复杂的SELECT语法，包括连接/分组查询和子查询；
2. 在没有WITH CHECK OPTION和 READ ONLY 的情况下，查询中不能使用ORDER BY 子句；
3. 如果没有为CHECK OPTION约束命名，系统会自动为之命名，形式为SYS\_Cn;
4. OR REPLACE选项可以不删除原视图便可更改其定义并重建，或重新授予对象权限。

#### 查询视图

视图创建成功后，可以从视图中检索数据，这点和从表中检索数据一样。还可以查询视图的全部信息和指定的数据行和列。

如：

检索数据：

|  |
| --- |
| SQL>SELECT \* FROM dept\_sum\_vw； |

查询视图定义：

|  |
| --- |
| SQL>SELECT view\_name,text from user\_views; |

其中text显示的内容为视图定义的SELECT语句，可通过DESC USER\_VIEWS得到相关信息。

修改视图：

通过OR REPLACE 重新创建同名视图即可。

#### 视图上的DML 操作：

DML操作应遵循的原则：

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关键字时不能删除数据行；
3. 在视图不出现下列情况时可通过视图修改基表数据或插入数据：
4. 视图中包含GROUP 函数，GROUP BY子句，DISTINCT关键字；
5. 使用表达式定义的列；
6. ROWNUM伪列。
7. 基表中未在视图中选择的其他列定义为非空且无默认值。

视图可用于保持数据库的完整性，但作用有限。

通过视图执行引用完整性约束可在数据库级执行约束。

#### WITH CHECK OPTION 子句限定

通过视图执行的INSERTS和UPDATES操作不能创建该视图检索不到的数据行，因为它会对插入或修改的数据行执行完整性约束和数据有效性检查。

例如：

|  |
| --- |
| SQL>CREATE OR REPLACE VIEW vw\_emp20  AS SELECT \* FROM emp  WHERE deptno=20  WITH CHECK OPTION constraint vw\_emp20\_ck;  视图 已建立。 |

查询结果：

|  |
| --- |
| SQL>SELECT empno,ename,job FROM vw\_emp20;  EMPNO ENAME JOB  --------------------- -------------- -------------  7369 SMITH CLERK  7566 JONES MANAGER  7902 FORD ANALYST |

修改：

|  |
| --- |
| UPDATE vw\_emp20  SET deptno=20  WHERE empno=7902; |

将产生错误：

|  |
| --- |
| UPDATE vw\_emp20  \*  ERROR 位于第一行：  ORA-01402：视图WITH CHECK OPTION 违反WHERE 子句 |

#### 删除视图

DROP VIEW VIEW\_NAME语句删除视图。

删除视图的定义不影响基表中的数据。

只有视图所有者和具备DROP VIEW权限的用户可以删除视图。

视图被删除后，基于被删除视图的其他视图或应用将无效。

## 分区表

### 分区

在大型的数据库应用中，需要处理的数据量可以达到几十到几百GB，甚至达到TB级。为了提高对这些巨型数据库的读写和查询速度，Oracle提供了一种分区技术，用户可以在创建表时应用分区技术，将数据以分区形式保存。

分区是指将巨型的表或索引分隔成相对较小的、可独立管理的部分。分区后的表与未分区的表在执行DML语句时没有任何区别。

对表进行分区时，必须为表中的每一条记录指定所属分区。一条记录属于哪一个分区是由分区表对该记录的匹配字段决定的。分区字段可以是表中一个字段或多个字段的组合，在创建分区表时确定。当用户对分区表进行插入、更新或删除等操作时，Oracle会自动根据分区字段的值来选择存储的分区。

* 表分区的作用

Oracle的表分区功能通过改善可管理性、性能和可用性，从而为各式应用程序带来了极大的好处。通常，分区可以使某些查询以及维护操作的性能大大提高。此外,分区还可以极大简化常见的管理任务，分区是构建千兆字节数据系统或超高可用性系统的关键工具。

分区功能能够将表、索引或索引组织表进一步细分为段，这些数据库对象的段叫做分区。每个分区有自己的名称，还可以选择自己的存储特性。从数据库管理员的角度来看，一个分区后的对象具有多个段，这些段既可进行集体管理，也可单独管理，这就使数据库管理员在管理分区后的对象时有相当大的灵活性。但是，从应用程序的角度来看，分区后的表与非分区表完全相同，使用 SQL DML 命令访问分区后的表时，无需任何修改。

* 什么时候使用分区表：

1. 表的大小超过2GB。
2. 表中包含历史数据，新的数据被增加都新的分区中。

* 表分区的优缺点

表分区有以下优点：

1. 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。
2. 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；
3. 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；
4. 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。

缺点：

分区表相关：已经存在的表没有方法可以直接转化为分区表。不过 Oracle 提供了在线重定义表的功能。

### 创建分区表

Oracle数据库提供了5种对表或索引的分区方法：范围分区、散列分区、列表分区、组合范围散列分区和组合范围列表分区。

范围分区就是对数据表中某个值的范围进行分区，例如根据值的大小进行分区。创建范围分区，需要使用PARTITION BY RANGE子句。

如果不指定分区名，Oracle将自动对分区进行命名。

### 范围分区（Range）

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。举个例子：你可能会将销售数据按照月份进行分区。

当使用范围分区时，请考虑以下几个规则：

1. 每一个分区都必须有一个VALUES LESS THEN子句，它指定了一个不包括在该分区中的上限值。分区键的任何值等于或者大于这个上限值的记录都会被加入到下一个高一些的分区中。
2. 所有分区，除了第一个，都会有一个隐式的下限值，这个值就是此分区的前一个分区的上限值。
3. 在最高的分区中，MAXVALUE被定义。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，也可以理解为高于任何分区中指定的VALUE LESS THEN的值，同时包括空值。

* 示例

例一：

假设有一个CUSTOMER表，表中有数据200000行，我们将此表通过CUSTOMER\_ID进行分区，每个分区存储100000行，我们将每个分区保存到单独的表空间中，这样数据文件就可以跨越多个物理磁盘。下面是创建表和分区的代码，如下：

|  |
| --- |
| CREATE TABLE CUSTOMER  (  CUSTOMER\_ID NUMBER NOT NULL PRIMARY KEY,  FIRST\_NAME VARCHAR2(30) NOT NULL,  LAST\_NAME VARCHAR2(30) NOT NULL,  PHONE VARCHAR2(15) NOT NULL,  EMAIL VARCHAR2(80),  STATUS CHAR(1)  )  PARTITION BY RANGE (CUSTOMER\_ID)  (  PARTITION CUS\_PART1 VALUES LESS THAN (100000) TABLESPACE CUS\_TS01,  PARTITION CUS\_PART2 VALUES LESS THAN (200000) TABLESPACE CUS\_TS02  ) |

例二：按时间划分

|  |
| --- |
| CREATE TABLE ORDER\_ACTIVITIES  (  ORDER\_ID NUMBER(7) NOT NULL,  ORDER\_DATE DATE,  TOTAL\_AMOUNT NUMBER,  CUSTOTMER\_ID NUMBER(7),  PAID CHAR(1)  )  PARTITION BY RANGE (ORDER\_DATE)  (  PARTITION ORD\_ACT\_PART01 VALUES LESS THAN (TO\_DATE('01- MAY -2003','DD-MON-YYYY')) TABLESPACEORD\_TS01,  PARTITION ORD\_ACT\_PART02 VALUES LESS THAN (TO\_DATE('01-JUN-2003','DD-MON-YYYY')) TABLESPACE ORD\_TS02,  PARTITION ORD\_ACT\_PART02 VALUES LESS THAN (TO\_DATE('01-JUL-2003','DD-MON-YYYY')) TABLESPACE ORD\_TS03  ) |

例三：MAXVALUE

|  |
| --- |
| CREATE TABLE RangeTable  (  idd INT PRIMARY KEY ,  iNAME VARCHAR(10),  grade INT  )  PARTITION BY RANGE (grade)  (  PARTITION part1 VALUES LESS THEN (1000) TABLESPACE Part1\_tb,  PARTITION part2 VALUES LESS THEN (MAXVALUE) TABLESPACE Part2\_tb  ); |

### 列表分区(List)

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。

例一

|  |
| --- |
| CREATE TABLE PROBLEM\_TICKETS  (  PROBLEM\_ID NUMBER(7) NOT NULL PRIMARY KEY,  DESCRIPTION VARCHAR2(2000),  CUSTOMER\_ID NUMBER(7) NOT NULL,  DATE\_ENTERED DATE NOT NULL,  STATUS VARCHAR2(20)  )  PARTITION BY LIST (STATUS)  (  PARTITION PROB\_ACTIVE VALUES ('ACTIVE') TABLESPACE PROB\_TS01,  PARTITION PROB\_INACTIVE VALUES ('INACTIVE') TABLESPACE PROB\_TS02  ） |

例二

|  |
| --- |
| CREATE TABLE ListTable  (  id INT PRIMARY KEY ,  name VARCHAR (20),  area VARCHAR (10)  )  PARTITION BY LIST (area)  (  PARTITION part1 VALUES ('guangdong','beijing') TABLESPACE Part1\_tb,  PARTITION part2 VALUES ('shanghai','nanjing') TABLESPACE Part2\_tb，  PARTITION part3 VALUES (default) TABLESPACE Part3\_tb  ); |

### 散列分区(Hash)

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，因为通过在I/O设备上进行散列分区，使得这些分区大小一致。列所在分区是依据分区列的hash值自动分配，因此你并不能控制也不知道哪条记录会被放到哪个分区中，hash分区也可以支持多个依赖列。

例一：

|  |
| --- |
| CREATE TABLE HASH\_TABLE  (  COL NUMBER(8),  INF VARCHAR2(100)  )  PARTITION BY HASH (COL)  (  PARTITION PART01 TABLESPACE HASH\_TS01,  PARTITION PART02 TABLESPACE HASH\_TS02,  PARTITION PART03 TABLESPACE HASH\_TS03  ) |

简写：

|  |
| --- |
| CREATE TABLE emp  (  empno NUMBER (4),  ename VARCHAR2 (30),  sal NUMBER  )  PARTITION BY HASH (empno) PARTITIONS 8  STORE IN (emp1,emp2,emp3,emp4,emp5,emp6,emp7,emp8); |

hash分区最主要的机制是根据hash算法来计算具体某条纪录应该插入到哪个分区中,hash算法中最重要的是hash函数，Oracle中如果你要使用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。

### 组合范围散列分区

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。

|  |
| --- |
| CREATE TABLE SALES  (  PRODUCT\_ID VARCHAR2(5),  SALES\_DATE DATE,  SALES\_COST NUMBER(10),  STATUS VARCHAR2(20)  )  PARTITION BY RANGE(SALES\_DATE) SUBPARTITION BY LIST (STATUS)  (  PARTITION P1 VALUES LESS THAN(TO\_DATE('2003-01-01','YYYY-MM-DD'))TABLESPACE rptfact2009  (  SUBPARTITION P1SUB1 VALUES ('ACTIVE') TABLESPACE rptfact2009,  SUBPARTITION P1SUB2 VALUES ('INACTIVE') TABLESPACE rptfact2009  ),  PARTITION P2 VALUES LESS THAN (TO\_DATE('2003-03-01','YYYY-MM-DD')) TABLESPACE rptfact2009  (  SUBPARTITION P2SUB1 VALUES ('ACTIVE') TABLESPACE rptfact2009,  SUBPARTITION P2SUB2 VALUES ('INACTIVE') TABLESPACE rptfact2009  )  ) |

### 复合范围散列分区

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。

|  |
| --- |
| create table dinya\_test  (  transaction\_id number primary key,  item\_id number(8) not null,  item\_description varchar2(300),  transaction\_date date  )  partition by range(transaction\_date)subpartition by hash(transaction\_id) subpartitions 3 store in (dinya\_space01,dinya\_space02,dinya\_space03)  (  partition part\_01 values less than(to\_date(‘2006-01-01’,’yyyy-mm-dd’)),  partition part\_02 values less than(to\_date(‘2010-01-01’,’yyyy-mm-dd’)),  partition part\_03 values less than(maxvalue)  ); |

### 维护表分区

1. 添加分区

以下代码给SALES表添加了一个P3分区

|  |
| --- |
| ALTER TABLE SALES  ADD PARTITION P3 VALUES  LESS THAN(TO\_DATE('2003-06-01','YYYY-MM-DD')); |

注意：以上添加的分区界限应该高于最后一个分区界限。

以下代码给SALES表的P3分区添加了一个P3SUB1子分区

|  |
| --- |
| ALTER TABLE SALES MODIFY PARTITION P3 ADD SUBPARTITION P3SUB1 VALUES('COMPLETE'); |

1. 删除分区

以下代码删除了P3表分区：

|  |
| --- |
| ALTER TABLE SALES DROP PARTITION P3; |

在以下代码删除了P4SUB1子分区：

|  |
| --- |
| ALTER TABLE SALES DROP SUBPARTITION P4SUB1; |

注意：如果删除的分区是表中唯一的分区，那么此分区将不能被删除，要想删除此分区，必须删除表。

1. 截断分区

截断某个分区是指删除某个分区中的数据，并不会删除分区，也不会删除其它分区中的数据。当表中即使只有一个分区时，也可以截断该分区。通过以下代码截断分区：

|  |
| --- |
| ALTER TABLE SALES TRUNCATE PARTITION P2; |

通过以下代码截断子分区：

|  |
| --- |
| ALTER TABLE SALES TRUNCATE SUBPARTITION P2SUB2; |

1. 合并分区

合并分区是将相邻的分区合并成一个分区，结果分区将采用较高分区的界限，值得注意的是，不能将分区合并到界限较低的分区。以下代码实现了P1 P2分区的合并：

|  |
| --- |
| ALTER TABLE SALES MERGE PARTITIONS P1,P2 INTO PARTITION P2; |

1. 拆分分区

拆分分区将一个分区拆分两个新分区，拆分后原来分区不再存在。注意不能对HASH类型的分区进行拆分。

|  |
| --- |
| ALTER TABLE SALES SBLIT PARTITION P2 AT(TO\_DATE('2003-02-01','YYYY-MM-DD')) INTO (PARTITION P21,PARTITION P22); |

1. 接合分区(coalesca)

结合分区是将散列分区中的数据接合到其它分区中，当散列分区中的数据比较大时，可以增加散列分区，然后进行接合，值得注意的是，接合分区只能用于散列分区中。通过以下代码进行接合分区：

|  |
| --- |
| ALTER TABLE SALES COALESCA PARTITION; |

1. 重命名表分区

以下代码将P21更改为P2

|  |
| --- |
| ALTER TABLE SALES RENAME PARTITION P21 TO P2; |

1. 相关查询

跨分区查询

|  |
| --- |
| select sum( \*) from  (select count(\*) cn from t\_table\_SS PARTITION (P200709\_1)  union all  select count(\*) cn from t\_table\_SS PARTITION (P200709\_2)  ); |

查询表上有多少分区

|  |
| --- |
| SELECT \* FROM useR\_TAB\_PARTITIONS WHERE TABLE\_NAME='tableName' |

查询索引信息

|  |
| --- |
| select object\_name,object\_type,tablespace\_name,sum(value)  from v$segment\_statistics  where statistic\_name IN ('physical reads','physical write','logical reads')and object\_type='INDEX'  group by object\_name,object\_type,tablespace\_name  order by 4 desc |

--显示数据库所有分区表的信息：

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

--显示当前用户可访问的所有分区表信息:

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

--显示当前用户所有分区表的信息：

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

--显示表分区信息 显示数据库所有分区表的详细分区信息：

|  |
| --- |
| select \* from DBA\_TAB\_PARTITIONS |

--显示当前用户可访问的所有分区表的详细分区信息：

|  |
| --- |
| select \* from ALL\_TAB\_PARTITIONS |

--显示当前用户所有分区表的详细分区信息：

|  |
| --- |
| select \* from USER\_TAB\_PARTITIONS |

--显示子分区信息 显示数据库所有组合分区表的子分区信息：

|  |
| --- |
| select \* from DBA\_TAB\_SUBPARTITIONS |

--显示当前用户可访问的所有组合分区表的子分区信息：

|  |
| --- |
| select \* from ALL\_TAB\_SUBPARTITIONS |

--显示当前用户所有组合分区表的子分区信息：

|  |
| --- |
| select \* from USER\_TAB\_SUBPARTITIONS |

--显示分区列 显示数据库所有分区表的分区列信息：

|  |
| --- |
| select \* from DBA\_PART\_KEY\_COLUMNS |

--显示当前用户可访问的所有分区表的分区列信息：

|  |
| --- |
| select \* from ALL\_PART\_KEY\_COLUMNS |

--显示当前用户所有分区表的分区列信息：

|  |
| --- |
| select \* from USER\_PART\_KEY\_COLUMNS |

--显示子分区列 显示数据库所有分区表的子分区列信息：

|  |
| --- |
| select \* from DBA\_SUBPART\_KEY\_COLUMNS |

--显示当前用户可访问的所有分区表的子分区列信息：

|  |
| --- |
| select \* from ALL\_SUBPART\_KEY\_COLUMNS |

--显示当前用户所有分区表的子分区列信息：

|  |
| --- |
| select \* from USER\_SUBPART\_KEY\_COLUMNS |

--怎样查询出oracle数据库中所有的的分区表

|  |
| --- |
| select \* from user\_tables a where a.partitioned='YES' |

--删除一个表的数据

|  |
| --- |
| truncate table table\_name; |

--删除分区表一个分区的数据

|  |
| --- |
| alter table table\_name truncate partition p5; |

## PL/SQL基础

### PL/SQL概述

PL/SQL是Procedure Language/Structuer Query Language的英文缩写，是Oracle对标准SQL规范的扩展，全面支持SQL的数据操作、事务控制等。PL/SQL完全支持SQL数据类型，减少了在应用程序和数据库之间转换数据的操作。 在Oracle数据库以及开发工具中都内置了PL/SQL处理引擎。

### PL/SQL代码编写规则

为了编写正确、高效的PL/SQL块，PL/SQL应用开发人员必须遵从特定的PL/SQL代码编写规则，否则会导致编译错误或运行错误。在编写PL/SQL代码时，应该遵从以下一些规则。

#### 标识符命名规则

当在PL/SQL中使用标识符定义变量、常量时，标识符名称必须以字符开始，并且长度不能超过30个字符。另外，为了提高程序的可读性，Oracle建议用户按照以下规则定义各种标识符：

* 当定义变量时，建议使用v\_作为前缀，例如v\_sal，v\_job等。
* 当定义常量时，建议使用c\_作为前缀，例如c\_rate。
* 当定义游标时，建议使用\_cursor作为后缀，例如emp\_cursor。
* 当定义异常时，建议使用e\_作为前缀，例如e\_integrity\_error。
* 当定义PL/SQL表类型时，建议使用\_table\_type作为后缀，例如sal\_table\_type。
* 当定义PL/SQL表变量时，建议使用\_table作为后缀，例如sal\_table。
* 当定义PL/SQL记录类型时，建议使用\_record\_type作为后缀，例如emp\_record\_type。
* 当定义PL/SQL记录变量时，建议使用\_record作为后缀，例如emp\_record。

#### 大小写规则

当在PL/SQL块中编写SQL语句和PL/SQL语句时，语句既可以使用大写格式，也可以使用小写格式。但是，为了提高程序的可读性和性能，Oracle建议用户按照以下大小写规则编写代码：

* SQL关键字采用大写格式，例如SELECT，UPDATE，SET，WHERE等。
* PL/SQL关键字采用大写格式，例如DECLARE，BEGIN，END等。
* 数据类型采用大写格式，例如INT，VARCHAR2，DATE等。
* 标识符和参数采用小写格式，例如v\_sal，c\_rate等。
* 数据库对象和列采用小写格式，例如emp，sal，ename等。

### PL/SQL程序块

块是PL/SQL的基本程序单元，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，可能只需要编写一个PL/SQL块；而如果要实现复杂的应用功能，那么可能需要在一个PL/SQL块中嵌套其他PL/SQL块。PL/SQL块分为无名块、命名块两种。无名块指未命名的程序块，命名块指过程、函数、包和触发器等。

PL/SQL块由3个部分组成：定义部分、执行部分、异常处理部分。其中，定义部分用于定义常量、变量、游标、异常、复合数据类型等；执行部分用于实现应用模块功能，该部分包含了要执行的PL/SQL语句和SQL语句；异常处理部分用于处理执行部分可能出现的运行错误。

#### PL/SQL块的基本结构

PL/SQL块的基本结构如下所示：

|  |
| --- |
| [DECLARE  … --定义部分]  BEGIN  … --执行部分  [EXCEPTION  … --异常处理部分]  END; |

其中，定义部分以DECLARE开始，该部分是可选的；执行部分以BEGIN开始，该部分是必须的；异常处理部分以EXCEPTION开始，该部分是可选的；而END则是PL/SQL块的结束标记。

* 示例

|  |
| --- |
| SQL> set serveroutput on  SQL> DECLARE  2 a NUMBER; --定义变量  3 BEGIN  4 a:=1+2; --为变量赋值  5 DBMS\_OUTPUT.PUT\_LINE('1+2='||a); --输出变量  6 EXCEPTION --异常处理  7 WHEN OTHERS THEN  8 DBMS\_OUTPUT.PUT\_LINE ('出现异常');  9 END;  10 /  1+2=3  PL/SQL 过程已成功完成。 |

其中，DBMS\_OUTPUT是Oracle所提供的系统包，PUT\_LINE是该包所包含的过程，用于输出字符串信息。当使用DBMS\_OUTPUT包输出数据或消息时，必须要将SQL\*Plus的环境变量SERVEROUTPUT设置为ON。

### 常量和数据类型

#### 数据类型

对于常量与变量的数据类型，除了可以使用与SQL相同的数据类型以外，Oracle还专门为PL/SQL程序块提供了表中所示的特定类型。

|  |  |
| --- | --- |
| **类型** | **说明** |
| BOOLEAN | 布尔型。取值为TRUE、FALSE或NULL |
| BINARY\_INTEGER | 带符号整数，取值范围为-231~231 |
| NATURAL | BINARY\_INTEGER的子类型，表示非负整数 |
| NATURALN | BINARY\_INTEGER的子类型，表示不为NULL的非负整数 |
| POSITIVE | BINARY\_INTEGER的子类型，表示正整数 |
| POSITIVEN | BINARY\_INTEGER的子类型，表示不为NULL的正整数 |
| SIGNTYPE | BINARY\_INTEGER的子类型，取值为-1、0或1 |
| PLS\_INTEGER | 带符号整数，取值范围为-231~231。它与BINARY\_INTEGER类似，都比NUMBER类型表示的范围小，因此占用更少的内存。当使用PLS\_INTEGER值时，如果算法发生溢出，会触发异常 |
| SIMPLE\_INTEGER | Oracle Database 11g的新增类型。它是BINARY\_INTEGER的子类型，其取值范围与BINARY\_INTEGER相同，但不能存储NULL值。当使用SIMPLE\_INTEGER值时，如果算法发生溢出，不会触发异常，只会简单地截断结果 |
| STRING | 与VARCHAR2相同 |
| RECORD | 一组其他类型的组合 |
| REF CURSOR | 指向一个行集的指针 |

#### 常量

声明常量时需要使用CONSTANT关键字，并且必须在声明时就为该常量赋值，而且在程序其他部分不能修改该常量的值。

定义常量的语法形式如下：

|  |
| --- |
| constant\_name CONSTANT data\_type { := | DEFAULT } value ; |

语法说明如下：

* constant\_name 表示常量名。
* data\_type 表示常量的数据类型。
* := | DEFAULT :=为赋值操作符。在初始化常量或变量值时还可以使用DEFAULT关键字代替。
* value 表示为常量赋的值。

PL/SQL程序块中的赋值符号是冒号等号（:=），而不是常见的等号（=），并且在书写时不要将冒号与等号分开，也就是说两者之间不能存在空格。

### 变量

变量是存储值的命名内存区域，以使程序存储和获取操作值。变量的类型决定其中存储的数据类型。如果需要存储一个单独的值，则使用标量变量；如果需要存储多个值（例如表中的一行记录），则需要一个复合类型的变量。

标量变量是包含一个单独的值的变量。标量变量所使用的一般数据类型包括字符、数字、日期和布尔类型。PL/SQL能够使用与SQL相同的数据类型。像其他高级程序语言一样，PL/SQL中的变量命名也要遵循一定的规则。

#### 变量的规则

PL/SQL中的变量命名也要遵循一定的规则，规则如下：

* 变量名以字母开头，不区分大小写。
* 变量名由字母、数字以及$、#或\_和特殊字符组成。
* 变量长度最多包含30个字符。
* 变量名中不能有空格。
* 尽可能避免缩写，用一些具有意义的单词命名。
* 不能用保留字命名。

正确的变量命名，如：Mynumber1，My\_number1，My$money，v\_temp，TEMP，money$$$tree，SN##，try\_again等。

非法的变量命名，如：1Mynumber，my number 1，my&money，vtemp123456-7891011121314151617181920。

#### 声明变量

在PL/SQL中，使用变量前一定要先声明，声明变量的语法格式如下：

|  |
| --- |
| variable\_name date\_type  NOT NULL [DEFAULT | := value] |

其中：

* variable\_name 是定义变量的名称。
* date\_type 表示定义变量的数据类型，常用的数据类型如CHAR， NUMBER，DATE等。
* NOT NULL 表示该变量非空，必须指定默认值。
* DEFAULT 和“：=”作用等价，可互相替换。
* 示例

|  |
| --- |
| DECLARE  c\_emp\_number CONSTANT NUMBER(4) := 7900 ;  v\_emp\_name VARCHAR2(10) ;  v\_emp\_job VARCHAR2(9) ;  v\_emp\_sal NUMBER(7 , 2) DEFAULT 2000 ;  BEGIN  -- PL/SQL程序的执行部分  END ;  / |

### 运算符和注释

#### 运算符

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **关系运算符** | | **一般运算符** | | **逻辑运算符** | |
| = | 等于 | + | 加号 | is null | 是空值 |
| <>、!=、~=、^= | 不等于 | - | 减号 | BETWEEN | 介于两者之间 |
| < | 小于 | \* | 乘号 | IN | 在一列值中间 |
| > | 大于 | / | 除号 | AND | 逻辑与 |
| <= | 小于或等于 | := | 赋值号 | OR | 逻辑或 |
| >= | 大于或等于 | => | 关系号 | NOT | 取反 |
|  |  | .. | 范围运算符 |  |  |
|  |  | || | 字符连接符 |  |  |

#### 注释

注释用于解释单行代码或多行代码的作用，从而提高PL/SQL程序的可读性。当编译并执行PL/SQL代码时，PL/SQL编译器会忽略注释。注释又包括单行注释和多行注释。

* 单行注释

单行注释是指放置在一行上的注释文本，并且单行注释主要用于说明单行代码的作用。在PL/SQL中使用--符号编写单行注释。

* 多行注释

多行注释是指分布到多行上的注释文本，并且其主要作用是说明一段代码的作用。在PL/SQL中使用/\*…\*/来编写多行注释。

* 示例

|  |
| --- |
| SQL> DECLARE  2 v\_sal emp.sal%TYPE; --定义变量v\_sal  3 BEGIN  4 /\*  5 以下代码用于取得雇员工资  6 \*/  7 SELECT sal INTO v\_sal FROM scott.emp  8 WHERE empno=7788;  9 DBMS\_OUTPUT.PUT\_LINE(v\_sal); --输出语句  10 END;  11 / |

### IF条件语句

在PL/SQL块中，IF条件语句可以包含IF、ELSIF（请注意ELSIF关键字的写法）、ELSE、THEN和END IF等关键字。其完整的语法形式如下：

|  |
| --- |
| IF condition1 THEN  statements1  [ ELSIF condition2 THEN  statements2 ] [ , … ]  [ ELSE  statements3 ]  END IF ; |

语法说明如下：

* condition<n> 布尔表达式，其值为TRUE或FALSE。
* statements<n> PL/SQL语句，在对应的条件为TRUE时被执行。
* ELSIF子句可以没有，也可以有一到多条。另外，在IF、ELSIF和ELSE子句中可以嵌入其他IF条件语句。

示例

判断成绩61分所处的等级

|  |
| --- |
| SQL> DECLARE  2 score BINARY\_INTEGER := 61 ;  3 BEGIN  4 IF score >= 90 THEN  5 DBMS\_OUTPUT.PUT\_LINE('优秀') ;  6 ELSIF score >= 80 THEN  7 DBMS\_OUTPUT.PUT\_LINE('良好') ;  8 ELSIF score >= 60 THEN  9 DBMS\_OUTPUT.PUT\_LINE('及格') ;  10 ELSE  11 DBMS\_OUTPUT.PUT\_LINE('不及格') ;  12 END IF ;  13 END ;  14 / |

### CASE语句

从功能上来讲，CASE语句基本上可以实现IF条件语句能实现的所有功能，而从代码结构上来讲，CASE语句具有更好的阅读性，因此，建议读者尽量使用CASE 语句代替IF语句。

Oracle中的CASE 语句分为两种类型：

* 简单CASE表达式 使用表达式确定返回值。
* 搜索CASE表达式 使用条件确定返回值。

#### 简单CASE表达式

简单CASE表达式使用嵌入式的表达式来确定返回值，其语法如下：

|  |
| --- |
| CASE search\_expression  WHEN expression1 THEN result1 ;  …  WHEN expressionN THEN resultN ;  [ ELSE default\_result ; ]  END CASE ; |

语法说明：

* search\_expression 待求值的表达式。
* WHEN expression1 THEN result1

其中，expression1表示要与search\_expression进行比较的表达式。如果二者的值相等，则返回result1；如果不相等，则继续下一次比较。

* ELSE default\_result

如果所有的WHEN子句中的表达式的值都与search\_expression不匹配，则返回default\_result，也就是默认值。如果不设置此选项，而又没有找到匹配的表达式，则Oracle将报错。

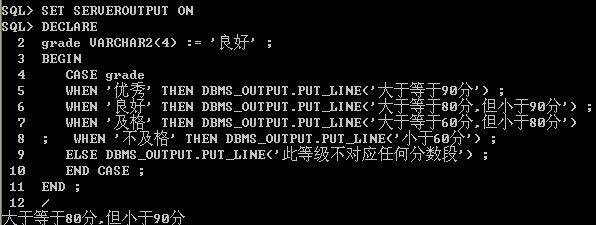
#### 搜索CASE表达式

搜索CASE表达式使用条件来确定返回值，其语法如下：

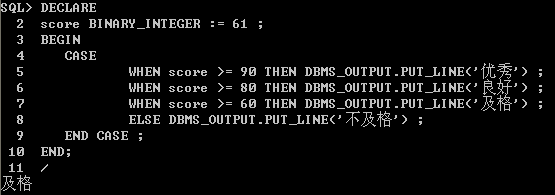
|  |
| --- |
| CASE  WHEN condition1 THEN result1 ;  WHEN condition2 THEN result2 ;  …  WHEN conditionN THEN resultN ;  [ ELSE default\_result ; ]  END CASE ; |

与简单CASE表达式相比较，可以发现CASE关键字后面不再跟随待求表达式，而WHEN子句中的表达式也换成了条件语句（condition），其实搜索CASE表达式就是将待求表达式放在条件语句中进行范围比较，而不再像简单CASE表达式那样只能与单个的值进行比较。

* 示例1



* 示例2



### 循环语句

对于程序中有些具有规律性的重复操作，就需要使用循环语句来完成。循环语句一般由循环体和循环结束条件组成，循环体是指被重复执行的语句集，而循环结束条件则用于终止循环。如果没有循环结束条件，或循环结束条件永远返回FALSE，则循环将陷入死循环。

为了在编写的PL/SQL块中重复执行一条语句或者一组语句，可以使用循环控制结构。编写循环控制结构时，用户可以使用LOOP基本循环、WHILE循环和FOR循环等3种类型的循环语句。

#### LOOP循环

LOOP循环语句是最简单的循环语句，其语法如下：

|  |
| --- |
| LOOP  statements ;  END LOOP ; |

其中，statements是LOOP循环体中的语句块。要想退出LOOP循环，必须在语句块中显式地使用EXIT关键字，否则循环会一直执行，也就是陷入死循环。EXIT关键字的使用形式如下：

|  |
| --- |
| EXIT [ WHEN conditon ] ; |

如上述语法形式，使用WHEN子句可以实现有条件退出；如果不使用WHEN子句，则会无条件退出循环。

#### WHILE循环

WHILE循环是在LOOP循环的基础上添加循环条件，也就是说，只有满足WHILE条件后，才会执行循环体中的内容。WHILE循环语句的语法如下：

|  |
| --- |
| WHILE condition  LOOP  statements ;  END LOOP ; |

如果condition条件语句永远返回TRUE，则WHILE循环将陷入死循环；如果condition条件语句永远返回FALSE，则循环一次也不会执行。

#### FOR循环

当使用基本循环或WHILE循环时，为了防止出现死循环，需要在循环体内不断修改判断条件。而FOR循环则通过指定一个数字范围，以确切地指出循环应该执行多少次。

FOR循环是在LOOP循环的基础上添加循环次数，其语法形式如下：

|  |
| --- |
| FOR loop\_variable IN [ REVERSE ] lower\_bound .. upper\_bound  LOOP  statements ;  END LOOP ; |

语法说明如下：

* loop\_variable

指定循环变量，该变量不需要事先创建。该变量的作用域仅限于循环内部，也就是说只可以在循环内部使用或修改该变量的值。

* IN

为loop\_variable指定取值范围。

* REVERSE

表示“逆向”，实际上就是让loop\_variable从取值范围中逆向取值，指定在每一次循环中loop\_variable的值递减。如果不使用此选项，则每一次循环中loop\_variable的值递增。

* lower\_bound .. upper\_bound

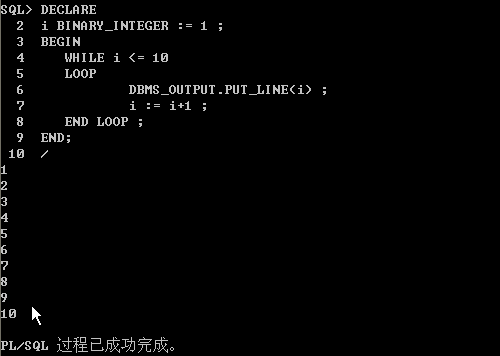
表示取值范围。其中，lower\_bound为循环下限值；upper\_bound为循环上限值；双点号（..）为PL/SQL中的范围符号。如果没有使用REVERSE关键字，则loop\_variable的初始值默认为lower\_bound，每循环一次， loop\_variable的值加1；如果使用了REVERSE关键字，则loop\_variable的初始值默认为upper\_bound，每循环一次，loop\_variable的值减1。

* 示例

loop循环

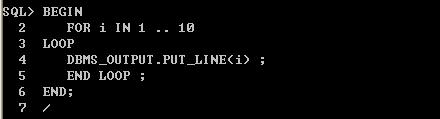


while循环

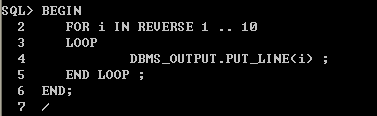


for循环

示例1



示例2



### 复合变量

标量变量只可以保存单个数值，在PL/SQL中与标量变量相对应的就是复合变量。复合变量可以将不同数据类型的多个值存储在一个单元中。由于复合数据类型可以由用户自己根据需要定义其结构，所以复合数据类型也称为自定义数据类型。

常用的复合数据类型有%type和%rowtype。

#### %TYPE类型

在PL/SQL程序块中，有时候会使用表中的数据为变量赋值，例如上一节示例中的v\_emp\_name等变量，就是通过emp表中的数据为其赋值，这种情况下就需要用户事先了解变量所对应的列的数据类型，否则用户无法确定变量的数据类型。

而使用%TYPE类型就可以解决这类问题，%TYPE类型用于隐式地将变量的数据类型指定为对应列的数据类型。使用%TYPE定义变量的形式为：

|  |
| --- |
| variable\_name table\_name.column\_name%TYPE  [ [ NOT NULL ] { := | DEFAULT } value ] ; |

例如：v\_emp\_name emp.ename%TYPE ;

在PL/SQL中，使用%TYPE声明的变量类型与数据表中字段的数据类型相同，当数据表中字段数据类型修改后，PL/SQL程序中相应变量的类型也自动随之改变。

#### %ROWTYPE类型

使用%TYPE可以使一个变量与字段的数据类型保持一致， %TYPE类型只是针对表中的某一列， PL/SQL还提供了%ROWTYPE来存储一行数据,%ROWTYPE类型针对表中的一行，使用%ROWTYPE类型定义的变量,可以存储表中的一行数据。

使用%ROWTYPE定义变量的形式如下：

|  |
| --- |
| variable\_name table\_name%ROWTYPE ; |

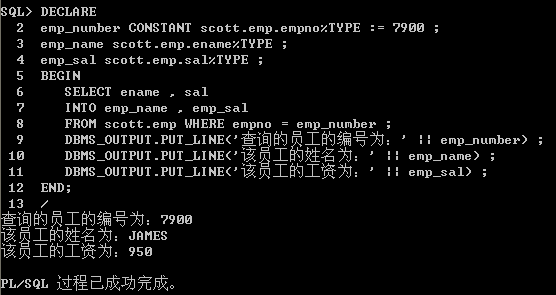
例如，要定义DEPT表中的一行记录，并且数据类型和每列类型一致，可以使用%ROWTYPE，声明如下：

|  |
| --- |
| dept\_record dept%ROWTYPE; |

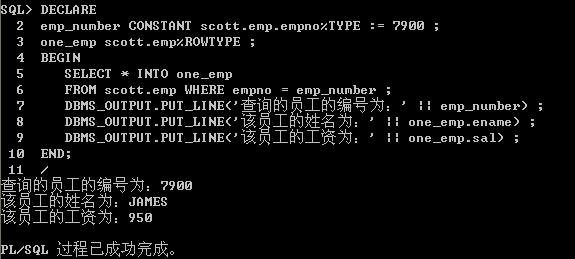
* 示例

|  |
| --- |
| SQL> DECLARE  emp\_number CONSTANT emp.empno%TYPE := 7900 ;  emp\_name emp.ename%TYPE ;  emp\_job emp.job%TYPE ;  emp\_sal emp.sal%TYPE ;  BEGIN  --执行语句  END; |

%TYPE示例



%ROWTYPE



### 记录类型

PL/SQL记录类型和表类型都是用户自定义的复合数据类型，其中记录类型可以存储多个字段值，类似于表中的一行数据；表类型则可以存储多行数据。

记录类型与数据库中的行结构非常相似，使用记录类型定义的变量可以存储由多个列值组成的一行数据。当使用记录类型的变量时，首先需要定义记录的结构，然后定义记录类型的变量。

记录类型与数据库中表的行结构非常相似，使用记录类型定义的变量可以存储由一个或多个字段组成的一行数据。

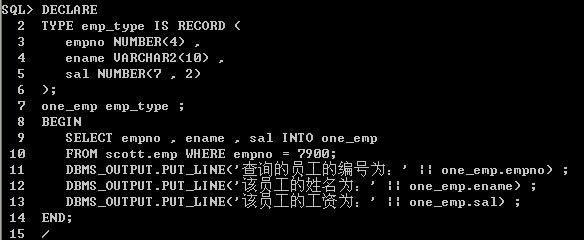
创建记录类型需要使用TYPE语句，其语法如下：

|  |
| --- |
| TYPE record\_name IS RECORD (  field\_name data\_type [ [ NOT NULL ] { := | DEFAULT } value ]  [ , … ]  ) ; |

语法说明如下：

* record\_name 创建的记录类型名称。
* IS RECORD 表示创建的是记录类型（区别于后面的表类型）。
* field\_name 记录类型中的字段名。

示例：



### 表类型

使用记录类型变量只能保存一行数据，这限制了SELECT语句的返回行数，如果SELECT语句返回多行就会出错。而Oracle提供了另外一种自定义类型，也就是表类型，它是对记录类型的扩展，允许处理多行数据，类似于表。

创建表类型的语法如下：

|  |
| --- |
| TYPE table\_name IS TABLE OF data\_type [ NOT NULL ]  INDEX BY BINARY\_INTEGER ; |

语法说明如下：

* table\_name 创建的表类型名称。
* IS TABLE 表示创建的是表类型。
* data\_type 可以是任何合法的PL/SQL数据类型。
* INDEX BY BINARY\_INTEGER 指定系统创建一个主键索引，用于引用表类型变量中的特定行。

|  |
| --- |
| SQL> DECLARE  2 TYPE my\_emp IS TABLE OF emp%ROWTYPE  3 INDEX BY BINARY\_INTEGER ;  4 new\_emp my\_emp ;  5 BEGIN  6 new\_emp (1).empno := 6800 ;  7 new\_emp (1).ename := 'TRACY' ;  8 new\_emp (1).job := 'CLERK' ;  9 new\_emp (1).sal := 2500 ;  10 new\_emp (2).empno := 6900 ;  11 new\_emp (2).ename := 'LUCY' ;  12 new\_emp (2).job := 'MANAGER' ;  13 new\_emp (2).sal := 4000 ;  14 DBMS\_OUTPUT.PUT\_LINE (new\_emp (1).empno || ',' ||  15 new\_emp (1).ename || ',' ||  16 new\_emp (1).job || ',' ||  17 new\_emp (1).sal) ;  18 DBMS\_OUTPUT.PUT\_LINE (new\_emp (2).empno || ',' ||  19 new\_emp (2).ename || ',' ||  20 new\_emp (2).job || ',' ||  21 new\_emp (2).sal) ;  22 END ; |

### 集合类型

为了处理单行单列的数据，开发人员可以使用标量变量；为了处理单行多列的数据，开发人员可以使用PL/SQL记录；而为了处理单列多行的数据，开发人员可以使用PL/SQL集合。例如，为了存放单个雇员的姓名，开发人员可以使用标量变量；而为了存放多个雇员的姓名，开发人员应该使用PL/SQL集合类型。

PL/SQL集合类型是类似于高级语言数组的一种复合数据类型，集合类型包括索引表（PL/SQL表）、嵌套表（Nested Table）和变长数组（VARRAY）等3种类型。当使用这些集合类型时，必须要注意3者之间的区别，以便选择最合适的数据类型。

#### 索引表

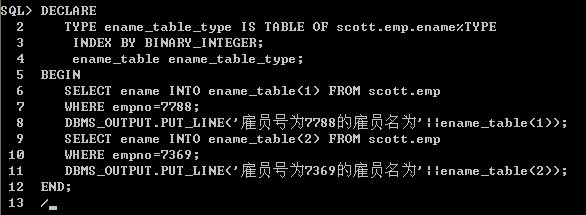
索引表也称为PL/SQL表，用于处理PL/SQL数组的数据类型。但是索引表与高级语言的数组是有区别的：高级语言数组的元素个数是有限制的，并且下标不能为负值；而索引表的元素个数没有限制，并且下标可以为负值。定义索引表的语法如下：

|  |
| --- |
| TYPE type\_name IS TABLE OF element\_type  [NOT NULL]INDEX BY key\_type;  identifier type\_name; |

其中：

* type\_name 用于指定用户自定义数据类型的名称。
* element\_type 用于指定索引表元素的数据类型。
* NOT NULL 表示不允许引用NULL元素。
* key\_type 用于指定索引表元素下标的数据类型（BINARY\_INTEGER、PLS\_INTEGER或VARCHAR2）。
* identifier 用于定义索引表变量。

示例：



#### 嵌套表

嵌套表也是一种用于处理PL/SQL数组的数据类型。同样嵌套表和高级语言的数组也是有区别的：高级语言数组的元素下标从0或1开始，并且元素个数是有限制的；而嵌套表的元素下标从1开始，并且元素个数没有限制；另外，高级语言的数组元素值是有顺序的，而嵌套表元素的数组元素值可以是无序的。索引表类型不能作为表列的数据类型使用，但嵌套表类型可以作为表列的数据类型使用。

当使用嵌套表元素之前，必须首先使用其构造方法初始化嵌套表。

定义嵌套表的语法如下：

|  |
| --- |
| TYPE type\_name IS TABLE OF element\_type;  idetifer type\_name; |

其中：

* type\_name 用于指定嵌套表的类型名。
* element\_type 用于指定嵌套表元素的数据类型。
* identifer 用于定义嵌套表变量。

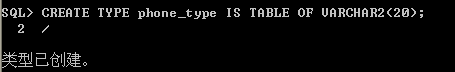
示例：

注意初始化。

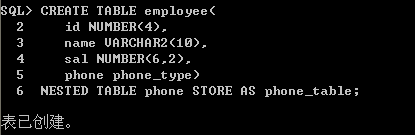


作为列类型：

先创建类型：



创建表:



添加数据：



查询：



#### 变长数组

变长数组（VARRAY）也是一种用于处理PL/SQL数组的数据类型，它也可以作为表列的数据类型使用。该数据类型与高级语言数组非常相似，其元素下标从1开始，并且元素的最大个数是有限制的。

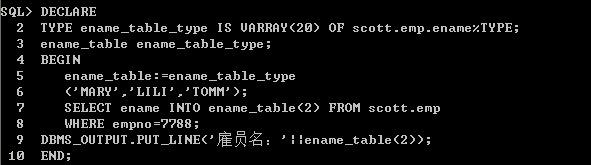
定义VARRAY的语法如下：

|  |
| --- |
| TYPE type\_name IS VARRAY(size\_limit) OF element\_type  [NOT NULL];  identifier type\_name; |

其中：

* type\_name 用于指定VARRAY类型名。
* size\_limit 用于指定VARRAY元素的最大个数。
* element\_type 用于指定元素的数据类型。
* identifier 用于定义VARRAY变量。

示例 :



### 集合方法

集合方法是Oracle所提供的用于操作集合变量的内置函数或过程，其中EXISTS()、COUNT()、LIMIT()、FIRST()、NEXT()、PRIOR()和LAST()是函数，而EXTEND()，TRIM()和DELETE()则是过程。集合方法的调用语法如下：

|  |
| --- |
| collection\_name.method\_name{(parameters)} |

集合方法只能在PL/SQL语句中使用，而不能在SQL语句中调用。另外集合方法EXTEND和TRIM只适用于嵌套表和VARRAY，而不适用于索引表。

#### EXISTS()

该方法用于确定集合元素是否存在，如果集合元素存在，则返回TRUE；如果集合元素不存在，则返回FALSE。示例如下：

|  |
| --- |
| SQL> DECLARE  2 TYPE ename\_table\_type IS TABLE OF emp.ename%TYPE;  3 ename\_table ename\_table\_type;--定义表类型的变量ename\_table  4 BEGIN  5 IF ename\_table.EXISTS(1) THEN  6 ename\_table(1):='SCOTT';  7 ELSE  8 DBMS\_OUTPUT.PUT\_LINE('必须初始化集合元素');  9 END IF;  10 END;  11 / |

#### COUNT()

该集合方法用于返回当前集合变量中的元素总个数。如果集合元素存在数值，则统计结果会包含该元素；如果集合元素为NULL，则统计结果不会包含该元素。示例如下：

|  |
| --- |
| SQL> DECLARE  2 TYPE ename\_table\_type IS TABLE OF emp.ename%TYPE  3 INDEX BY BINARY\_INTEGER;  4 ename\_table ename\_table\_type; --定义索引表类型变量ename\_table  5 BEGIN  6 ename\_table(-5) :='SCOTT'; --向索引表中添加数据  7 ename\_table(1) :='SMITH';  8 ename\_table(5) :='MARY';  9 ename\_table(10) :='BLACK';  10 DBMS\_OUTPUT.PUT\_LINE('该集合元素总个数为：'||ename\_table.count);  11 END;  12 / |

#### LIMIT()

该方法用于返回集合元素的最大个数。因为嵌套表和索引表的元素个数没有限制，所以调用该方法会返回NULL；而对于VARRAY来说，该方法会返回VARRAY所允许的最大元素个数。示例如下：

|  |
| --- |
| SQL> DECLARE  2 TYPE ename\_table\_type IS VARRAY(20) OF emp.ename%TYPE;  3 ename\_table ename\_table\_type :=ename\_table\_type('mary');  4 BEGIN  5 DBMS\_OUTPUT.PUT\_LINE('该集合元素的最大个数：'  6 ||ename\_table.limit); --返回集合元素的最大个数  7 END;  8 / |

#### EXTEND()

该方法用于扩展集合变量的尺寸，并为它们增加元素。注意，该方法只适用于嵌套表和VARRAY。该方法有EXTEND，EXTEND（n），EXTEND(n,i)等3种调用格式，其中EXTEND用于为集合变量添加一个NULL元素，EXTEND(n)用于为集合变量添加n个NULL元素，而EXTEND(n,i)则用于为集合变量添加n个元素（元素值与第i个元素相同）。

示例如下：

|  |
| --- |
| SQL> DECLARE  2 TYPE ename\_table\_type IS VARRAY(20) OF VARCHAR2(10);  3 ename\_table ename\_table\_type;  4 BEGIN  5 ename\_table :=ename\_table\_type('MARY');  6 ename\_table.EXTEND(5,1); --为集合变量添加5个元素  7 DBMS\_OUTPUT.PUT\_LINE('元素总个数：'||ename\_table.COUNT);  8 END;  9 / |

#### TRIM()

该方法用于从集合尾部删除元素，它有TRIM和TRIM(n)两种调用格式。其中TRIM用于从尾部删除一个元素；而TRIM(n)则用于从集合尾部删除n个元素。示例如下：

|  |
| --- |
| SQL> DECLARE  2 TYPE ename\_table\_type IS VARRAY(20) OF VARCHAR2(10);  3 ename\_table ename\_table\_type;  4 BEGIN  5 name\_table :=ename\_table\_type('MARY','SCOTT','SMITH','BLAKE');  6 ename\_table.TRIM(2); --删除两个元素  7 DBMS\_OUTPUT.PUT\_LINE('元素总个数：'||ename\_table.COUNT);  8 END;  9 / |

#### FIRST()、LAST()、PRIOR() 、NEXT()

FIRST()方法用于返回集合变量第一个元素的下标，LAST()方法用于返回集合变量最后一个元素的下标，PRIOR()方法用于返回当前集合元素的前一个元素的下标，而NEXT()方法则用于返回当前集合元素的后一个元素的下标。示例如下：

|  |
| --- |
| SQL> DECLARE  2 TYPE ename\_table\_type IS TABLE OF emp.ename%TYPE  3 INDEX BY BINARY\_INTEGER;  4 ename\_table ename\_table\_type; --定义索引表类型变量ename\_table  5 BEGIN  6 ename\_table(-5) :='SCOTT'; --向索引表中添加数据  7 ename\_table(1) :='SMITH';  8 ename\_table(5) :='MARY';  9 ename\_table(10) :='BLACK';  10 DBMS\_OUTPUT.PUT\_LINE('第一个元素：'||ename\_table.first);  11 DBMS\_OUTPUT.PUT\_LINE('最后一个元素：'||ename\_table.last);  12 DBMS\_OUTPUT.PUT\_LINE('元素5的前一个元素：'||  13 ename\_table.prior(5));  14 DBMS\_OUTPUT.PUT\_LINE('元素5的后一个元素：'||  15 ename\_table.next(5));  16 END; |

#### DELETE()

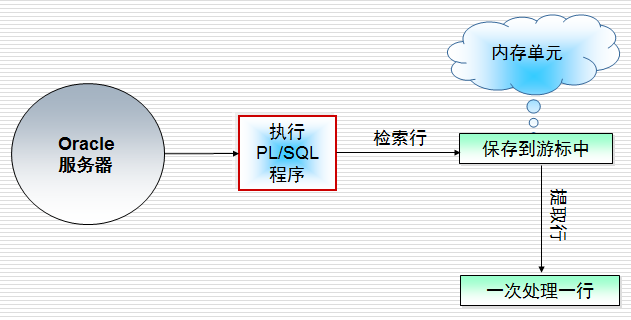
该方法用于删除集合元素，但该方法只适用于嵌套表和索引表，而不适用于VARRAY。该方法有DELETE，DELETE(n)，DELETE(m,n)等3种调用格式。其中DELETE用于删除集合变量的所有元素；DELETE (n)用于删除集合变量的第n个元素；而DELETE(m,n)则用于删除集合变量从m到n之间的所有元素。

|  |
| --- |
| SQL> DECLARE  2 TYPE ename\_table\_type IS TABLE OF emp.ename%TYPE  3 INDEX BY BINARY\_INTEGER; --定义索引表类型  4 ename\_table ename\_table\_type; --定义索引表变量  5 BEGIN  6 ename\_table(-5) :='SCOTT'; --向索引表添加数据  7 ename\_table(1) :='SMITH';  8 ename\_table(5) :='MARY';  9 ename\_table(10) :='BLACK';  10 ename\_table.DELETE(5); --删除索引下标为5的元素  11 DBMS\_OUTPUT.PUT\_LINE('元素总个数：'||ename\_table.COUNT);  12 END;  13 / |

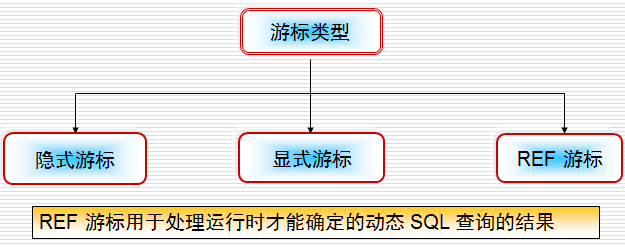
### 游标

PL/SQL的游标指把从数据库中查询出来的数据以临时表的形式存放在内存中。游标可以对存储在内存中的数据进行操作，返回一条或一组记录，或者一条记录也不返回。PL/SQL中的记录和表类型虽然也可以用来存放数据，但对一组存放在内存中的数据进行操作，还是不太方便，游标恰好是实现这一功能的有力工具。

PL/SQL包含隐含游标和显示游标等两种游标类型，其中隐含游标用于处理SELECT INTO和DML语句，而显示游标则专门用于处理SELECT语句返回的多行数据。游标的基本操作有：声明游标、打开游标、提取游标和关闭游标。



#### 游标的类型



#### 隐式游标

* 在PL/SQL中使用DML语句时自动创建隐式游标
* 隐式游标自动声明、打开和关闭，其名为 SQL
* 通过检查隐式游标的属性可以获得最近执行的DML 语句的信息

隐式游标的属性有：

* %FOUND – SQL 语句影响了一行或多行时为 TRUE
* %NOTFOUND – SQL 语句没有影响任何行时为TRUE
* %ROWCOUNT – SQL 语句影响的行数
* %ISOPEN - 游标是否打开，始终为FALSE

示例1：

|  |
| --- |
| SQL> SET SERVEROUTPUT ON  SQL> BEGIN  UPDATE toys SET toyprice=270  WHERE toyid= 'P005';  IF SQL%FOUND THEN--只有在 DML 语句影响一行或多行时，才返回 True  DBMS\_OUTPUT.PUT\_LINE(‘表已更新');  END IF;  END;  / |

示例2：

|  |
| --- |
| SQL> SET SERVEROUTPUT ON  SQL> DECLARE  v\_TOYID TOYS.ID%type := '&TOYID';  v\_TOYNAME TOYS.NAME%Type := '&TOYNAME';  BEGIN  UPDATE TOYS SET NAME = v\_TOYNAME  WHERE toyid=v\_TOYID;  IF SQL%NOTFOUND THEN--如果 DML 语句不影响任何行，则返回 True  DBMS\_OUTPUT.PUT\_LINE('编号未找到。');  ELSE  DBMS\_OUTPUT.PUT\_LINE(‘表已更新');  END IF;  END;  / |

示例3：

|  |
| --- |
| SQL> SET SERVEROUTPUT ON  SQL> BEGIN  UPDATE vendor\_master  SET venname= 'Rob Mathew'  WHERE vencode='V004';  DBMS\_OUTPUT.PUT\_LINE (SQL%ROWCOUNT);-- 返回 DML 语句影响的行数  END;  / |

示例4：

|  |
| --- |
| SQL> SET SERVEROUTPUT ON  SQL> DECLARE  empid VARCHAR2(10);  desig VARCHAR2(10);  BEGIN  empid:= '&Employeeid';  SELECT designation INTO desig  FROM employee WHERE empno=empid;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  DBMS\_OUTPUT.PUT\_LINE('职员未找到');  END;  / |

如果没有与SELECT INTO语句中的条件匹配的行，将引发NO\_DATA\_FOUND异常

示例5：

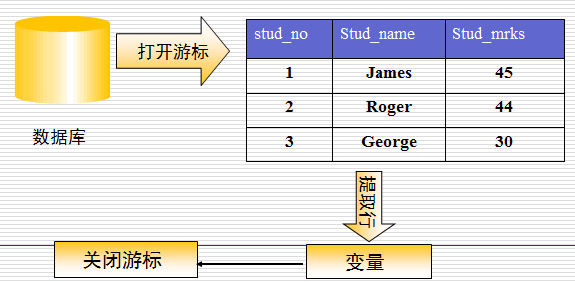
|  |
| --- |
| SQL> SET SERVEROUTPUT ON  SQL> DECLARE  empid VARCHAR2(10);  BEGIN  SELECT empno INTO empid FROM employee;  EXCEPTION  WHEN TOO\_MANY\_ROWS THEN  DBMS\_OUTPUT.PUT\_LINE('该查询提取多行');  END;  / |

如果 SELECT INTO 语句返回多个值，将引发TOO\_MANY\_ROWS异常

#### 显式游标

显式游标在 PL/SQL 块的声明部分定义查询，该查询可以返回多行。

显式游标的操作过程：



##### 声明游标

声明游标，主要是定义一个游标名称来对应一条查询语句，从而可以利用该游标对此查询语句返回的结果集进行单行操作。声明游标的语法如下：

|  |
| --- |
| CURSOR cursor\_name  [ ( parameter\_name [ IN ] data\_type [ { := | DEFAULT } value ] [ , … ] ) ]  IS select\_statement  [ FOR UPDATE [ OF column [ , … ] ] [ NOWAIT ] ] ; |

语法说明如下：

* cursor\_name 新游标名称。
* parameter\_name [ IN ] 为游标定义输入参数，IN关键字可以省略。使 用输入参数可以使游标的应用变得更灵活。用户需要在打开游标时为输入参数赋值，也可使用参数的默认值。输入参数可以有多个，多个参数的设置之间使用逗号（,）隔开。
* data\_type 为输入参数指定数据类型，但不能指定精度或长度。例如字符串类型可以使用VARCHAR2，而不能使用VARCHAR2(10)之类的精确类型。
* select\_statement 查询语句。
* FOR UPDATE 用于在使用游标中的数据时，锁定游标结果集与表中 对应数据行的所有或部分列。
* OF 如果不使用OF子句，则表示锁定游标结果集与表中对应数据行的所有列。如果指定了OF子句，则只锁定指定的列。
* NOWAIT 如果表中的数据行被某用户锁定，那么其他用户的FOR UPDATE操作将会一直等到该用户释放这些数据行的锁定后才会执行。而如果使用了NOWAIT关键字，则其他用户在使用OPEN命令打开游标时会立即返回错误信息。

##### 打开游标

在声明游标时为游标指定了查询语句，但此时该查询语句并不会被Oracle执行。只有打开游标后，Oracle才会执行查询语句。在打开游标时，如果游标有输入参数，用户还需要为这些参数赋值，否则将会报错（除非参数设置了默认值）。

打开游标需要使用OPEN语句，其语法如下：

|  |
| --- |
| OPEN cursor\_name [ ( value [ , … ] ) ] ; |

【例】在PL/SQL中，声明一个游标emp\_cursor对应emp表中的查询操作，此查询操作检索emp表中指定部门的员工的部分信息，语句如下：

|  |
| --- |
| SQL> DECLARE  2 CURSOR emp\_cursor (dept\_num NUMBER := 20)  3 IS  4 SELECT empno , ename , job , sal  5 FROM emp WHERE deptno = dept\_num ;  6 BEGIN  7 … ;  8 END ; |

【例】在前面BEGIN … END块中，使用OPEN语句打开游标emp\_cursor，

并且为输入参数赋值30，指定检索部门编号为30的员工的信息，语句如下：

|  |
| --- |
| OPEN emp\_cursor (30) ; |

##### 检索游标

打开游标后，游标所对应的SELECT语句也就被执行了，如果想要获取结果集中的数据，就需要检索游标。检索游标，实际上就是从结果集中获取单行数据并保存到定义的变量中，这需要使用FETCH语句，其语法如下：

|  |
| --- |
| FETCH cursor\_name INTO variable [ , … ] ; |

其中，variable是用来存储结果集中单行数据的变量，可以选择使用多个普通类型的变量，一对一地接受数据行中的列值；也可以使用一个%ROWTYPE类型的变量，或自定义的记录类型变量，接受数据行中所有的列值。变量需要事先定义。

【例】创建自定义记录类型emp\_type，然后使用该类型定义变量one\_emp，语句如下：

|  |
| --- |
| TYPE emp\_type IS RECORD (  empno NUMBER(4) , ename VARCHAR2(10) ,  job VARCHAR2(9) , sal NUMBER(7 , 2)) ;  one\_emp emp\_type ; |

在BEGIN … END块中的打开游标的语句后面，使用FETCH语句检索游标emp\_cursor，将游标中的单行数据赋值给one\_emp变量，语句如下：

|  |
| --- |
| FETCH emp\_cursor INTO one\_emp ; |

如果是使用普通类型变量，例如事先创建NUMBER(4)类型的变量emp\_num等，则可以使用如下形式获取游标中的记录值：

|  |
| --- |
| FETCH emp\_cursor INTO emp\_num , emp\_name , emp\_job , emp\_sal ; |

##### 关闭游标

关闭游标需要使用CLOSE语句。游标被关闭后，Oracle将释放游标中SELECT语句的查询结果所占用的系统资源。其语法如下：

|  |
| --- |
| CLOSE cursor\_name ; |

##### 示例

|  |
| --- |
| SQL>SET SERVER OUTPUT ON  SQL>DECLARE  my\_toy\_price toys.toyprice%TYPE;  CURSOR toy\_cur IS--声明游标  SELECT toyprice FROM toys  WHERE toyprice<250;  BEGIN  OPEN toy\_cur; --打开游标  LOOP  FETCH toy\_cur INTO my\_toy\_price;--提取行  EXIT WHEN toy\_cur%NOTFOUND;  DBMS\_OUTPUT.PUT\_LINE  ('TOYPRICE=:玩具单价=：'||my\_toy\_price);  END LOOP;  CLOSE toy\_cur;--关闭游标  END; |

#### 带参数的显式游标

声明显式游标时可以带参数以提高灵活性

声明带参数的显式游标的语法如下：

|  |
| --- |
| CURSOR <cursor\_name>(<param\_name> <param\_type>)  IS select\_statement; |

示例：

|  |
| --- |
| SQL> SET SERVEROUTPUT ON  SQL> DECLARE  desig VARCHAR2(20);  emp\_code VARCHAR2(5);  empnm VARCHAR2(20);  CURSOR emp\_cur(desigparam VARCHAR2) IS  SELECT empno, ename FROM employee  WHERE designation=desig;  BEGIN  desig:= '&desig';  OPEN emp\_cur(desig);  LOOP  FETCH emp\_cur INTO emp\_code,empnm;  EXIT WHEN emp\_cur%NOTFOUND;  DBMS\_OUTPUT.PUT\_LINE(emp\_code||' '||empnm);  END LOOP;  CLOSE emp\_cur;  END; |

#### 使用显式游标更新行

允许使用游标删除或更新活动集中的行

声明游标时必须使用 SELECT … FOR UPDATE语句

语法：

|  |
| --- |
| CURSOR <cursor\_name> IS  SELECT statement FOR UPDATE; |

更新的语法：

|  |
| --- |
| UPDATE <table\_name>  SET <set\_clause>  WHERE CURRENT OF <cursor\_name> |

删除的语法：

|  |
| --- |
| DELETE FROM <table\_name>  WHERE CURRENT OF <cursor\_name> |

示例：

|  |
| --- |
| SQL> SET SERVEROUTPUT ON  SQL> DECLARE  new\_price NUMBER;  CURSOR cur\_toy IS  SELECT toyprice FROM toys WHERE toyprice<100  FOR UPDATE OF toyprice;  BEGIN  OPEN cur\_toy;  LOOP  FETCH cur\_toy INTO new\_price;  EXIT WHEN cur\_toy%NOTFOUND;  UPDATE toys  SET toyprice = 1.1\*new\_price  WHERE CURRENT OF cur\_toy;  END LOOP;  CLOSE cur\_toy;  COMMIT;  END; |

#### 循环游标

* 循环游标用于简化游标处理代码
* 当用户需要从游标中提取所有记录时使用
* 循环游标的语法如下：

|  |
| --- |
| FOR <record\_index> IN <cursor\_name>  LOOP  <executable statements>  END LOOP; |

示例：

|  |
| --- |
| SQL> SET SERVER OUTPUT ON  SQL> DECLARE  CURSOR mytoy\_cur IS  SELECT toyid, toyname, toyprice  FROM toys;  BEGIN  FOR toy\_rec IN mytoy\_cur  LOOP  DBMS\_OUTPUT.PUT\_LINE(  ‘玩具编号：'||' ' ||toy\_rec.toyid||' '  ||‘玩具名称：'||' '||toy\_rec.toyname||' '  ||‘玩具单价：'||' '||toy\_rec.toyprice);  END LOOP;  END; |

#### REF 游标和游标变量

* REF 游标和游标变量用于处理运行时动态执行的 SQL 查询
* 创建游标变量需要两个步骤：

1. 声明 REF 游标类型
2. 声明 REF 游标类型的变量

* 用于声明 REF 游标类型的语法为：

|  |
| --- |
| TYPE <ref\_cursor\_name> IS REF CURSOR  [RETURN <return\_type>]; |

* 打开游标变量的语法如下:

|  |
| --- |
| OPEN cursor\_name FOR select\_statement; |

* 游标变量：

a. 一种引用类型

b. 可以再运行时指向不同的存储位置

c. Close语句关闭游标并释放用于查询的资源

* 游标变量的类型

1. 具有约束的游标变量

- 也就是具有返回类型的游标变量 也称为”强游标"

1. 无约束的游标变量

- 也就是没有返回类型的游标变量 也称为 “弱游标”

示例1:　 弱型游标示例

声明时不需指定查询语句,即不指向任何区域

|  |
| --- |
| declare  　　 type RefEmpCur　 is ref cursor;　 --声明引用游标类型 游标返回的类型没有限制  　　 EmpCur RefEmpCur;　 --游标变量  　　 EmpRow emp%rowtype; --存储游标查询到得结果  　　 flag int:=0;  begin  　　 flag :=&flag;  　　 if flag=0 then  　　　　　 open EmpCur　 for select \* from emp where sal>500 and sal<2000;  　　 elsif　 flag=1 then  　　　　　 open EmpCur for select \* from　 emp where sal>=2000;  　　 elsif　 flag=2 then  　　　　　 open EmpCur for select \* from　 dept;　 --弱类型游标对目标表没有限制，数据可以使来自任何表  　　 else  　　　　　 open EmpCur for select \* from emp;  　　 end if;  　 /\* for循环不能用于Ref游标，因为它是自动打开游标  　　 for EmpRow in EmpCur　 loop  　　　 DBMS\_output.put\_line('empno='||EmpRow.empno);  　　 end loop;  　　 \*/  　　 loop  　　　　　 exit when EmpCur%notfound;　 --如果没有查询到数据就退出  　　　　　 fetch EmpCur into EmpRow;  　　　　　 DBMS\_output.put\_line('empno='||EmpRow.empno);  　　 end loop;  　　 close EmpCur;  end;  / |

示例1:　 强型游标示例

|  |
| --- |
| declare  　　 type RefEmpCur　 is ref cursor return emp%rowtype;　 --游标仅能打开emp表的数据  　　 EmpCur RefEmpCur;　 --游标变量  　　 EmpRow emp%rowtype; --存储游标查询到得结果  　　 flag int:=0;  begin  　　 flag :=&flag;  　　 if flag=0 then  　　　　　 open EmpCur　 for select \* from emp where sal>500 and sal<2000;  　　 elsif　 flag=1 then  　　　　　 open EmpCur for select \* from emp where sal>=2000;  　　 else  　　　　　 open EmpCur for select \* from emp;  　　 end if;  　　 loop  　　　　　 exit when EmpCur%notfound;　 --如果没有查询到数据就退出  　　　　　 fetch EmpCur into EmpRow;  　　　　　 DBMS\_output.put\_line('empno='||EmpRow.empno);  　　 end loop;  　　 close EmpCur;  end;  / |

#### 游标变量的优点和限制

游标变量的功能强大，可以简化数据处理。

* 游标变量的优点有：

1. 可从不同的 SELECT 语句中提取结果集
2. 可以作为过程的参数进行传递
3. 可以引用游标的所有属性
4. 可以进行赋值运算

* 使用游标变量的限制：

1. 不能在程序包中声明游标变量
2. FOR UPDATE子句不能与游标变量一起使用
3. 不能使用比较运算符

#### 使用游标变量执行动态 SQL

可以使用游标变量执行动态构造的 SQL 语句

打开执行动态 SQL 的游标变量的语如下：

|  |
| --- |
| OPEN cursor\_name FOR dynamic\_sqlstring  [USING bind\_argument\_list]; |

示例：

|  |
| --- |
| DECLARE  r\_emp emp%ROWTYPE;  TYPE c\_type IS REF CURSOR;  cur c\_type;  p\_salary NUMBER;  BEGIN  p\_salary := 2500;  OPEN cur FOR 'select \* from emp where sal>:1  order by sal desc'  USING p\_salary;  DBMS\_OUTPUT.PUT\_LINE('薪水大于'|| p\_salary ||'的员工有：');  LOOP  FETCH cur INTO r\_emp;  EXIT WHEN cur%NOTFOUND;  DBMS\_OUTPUT.PUT\_LINE('编号：'|| r\_emp.empno  || ' 姓名：' || r\_emp.ename|| ' 薪水：' || r\_emp.sal );  END LOOP;  CLOSE cur;  END; |

### 异常处理

异常是指PL/SQL程序块在执行时出现的错误。在实际应用中，导致PL/SQL块出现异常的原因有很多，例如程序本身出现逻辑错误，或者程序人员根据业务需要，自定义部分异常错误等等。

Oracle有3种类型的异常：预定义异常、非预定义异常、自定义异常。

当产生异常时，如果程序中没有对该异常进行处理的语句，则整个程序将停止执行，所以，程序员应该对可能出现的异常进行控制，也就是进行异常处理。

#### 异常处理

处理异常需要使用EXCEPTION语句块，其具体语法如下：

|  |
| --- |
| EXCEPTION  WHEN exception1 THEN  statements1 ;  WHEN exception2 THEN  statements2 ;  [ … ]  WHEN OTHERS THEN  statementsN ; |

语法说明如下：

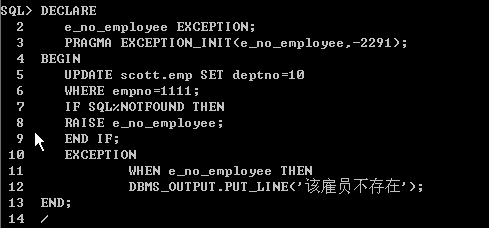
* exception<n> 表示可能出现的异常名称。
* WHEN OTHERS 表示任何其他情况，类似于ELSE，该子句需要放在EXCEPTION语句块的最后。

#### 自定义异常

用户在任何PL/SQL程序块、子程序或包中定义的异常，称为用户自定义异常。声明用户自定义异常需在PL/SQL程序块的声明部分中进行声明，Oracle允许使用PRAGMA关键字，让异常名和异常号联系起来。用户自定义异常声明语法如下：

|  |
| --- |
| exception\_name EXCEPTION;  PRAGMA EXCEPTION\_INTO(exception\_name,exception\_no); |

触发用户自定义异常需用RAISE语句显示触发异常，即RAISE exception\_name ，然后对它进行异常处理。



#### 预定义异常

系统预定义异常就是系统为经常出现的一些异常定义了异常关键字，如被零除或内存溢出等。系统预定义异常无需声明，当系统预定义异常发生时，Oracle系统会自动触发，只需添加相应的异常处理即可。

|  |  |  |  |
| --- | --- | --- | --- |
| **异常名称** | **错误代码** | **错误号** | **含义** |
| ACCESS\_INTO\_NULL | ORA-06530 | -6530 | 试图给未初始化对象的属性赋值 |
| CASE\_NOT\_FOUND | ORA-06592 | -6592 | CASE语句中未找到匹配的WHEN子句，也没有默认的ELSE子句 |
| CURSOR\_ALREADY\_OPEN | ORA-06511 | -6511 | 试图打开一个已经打开的游标 |
| DUP\_VAL\_ON\_INDEX | ORA-00001 | -1 | 试图向具有惟一约束的列中插入重复值 |
| INVALID\_CURSOR | ORA-01001 | -1001 | 试图进行非法游标操作，如关闭一个尚未打开的游标等 |
| LOGIN\_DENIED | ORA-01017 | -1017 | 试图用错误的用户名或密码连接数据库 |
| INVALID\_NUMBER | ORA-01722 | -1722 | 试图将一个无法代表有效数字的字符串转换成数字 |
| NO\_DATA\_FOUND | ORA-01403 | +100 | 数据不存在 |
| NOT\_LOGGED\_ON | ORA-01012 | -1012 | 试图在连接数据库之前访问数据库中的数据 |
| PROGRAM\_ERROR | ORA-06501 | -6501 | PL/SQL内部错误 |
| ROWTYPE\_MISMATCH | ORA-06504 | -6504 | 宿主游标变量与PL/SQL游标变量返回类型不兼容 |
| SELF\_IS\_NULL | ORA-30625 | -30625 | 试图在空对象中调用MEMBER方法 |
| STORAGE\_ERROR | ORA-06500 | -6500 | 内存出现错误，或已用完 |
| SUBSCRIPT\_BEYOND\_COUNT | ORA-06533 | -6533 | 试图通过大于集合元素个数的索引值引用嵌套表或变长数组元素 |
| SUBSCRIPT\_OUTSIDE\_LIMIT | ORA-06532 | -6532 | 试图通过合法范围之外的索引值引用嵌套表或变长数组元素 |
| SYS\_INVALID\_ROWID | ORA-01410 | -1410 | 将字符串转换成通用记录号rowid的操作失败 |
| TIMEOUT\_ON\_RESOURCE | ORA-00051 | -51 | 等待资源时发生超时 |
| TOO\_MANY\_ROWS | ORA-01422 | -1422 | SELECT INTO语句返回多条记录 |
| VALUE\_ERROR | ORA-06502 | -6502 | 发生算术、转换、截断或大小约束错误 |
| ZERO\_DIVIDE | ORA-01476 | -1476 | 试图将0作为除数 |

#### 非预定义异常

除了Oracle预定义好的异常以外，还有一些其他异常也属于程序本身的逻辑错误，例如违反表的外键约束、检查约束等，Oracle只为这些异常提供了错误代码，而这些异常同样需要处理，只不过需要在PL/SQL块中使用PRAGMA EXCEPTION\_INIT语句为该异常设置名称。其语法如下：

|  |
| --- |
| exception\_name EXCEPTION ;  PRAGMA EXCEPTION\_INIT ( exception\_name , oracle\_error\_number ) ; |

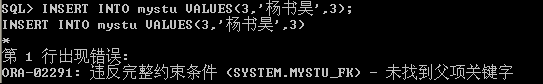
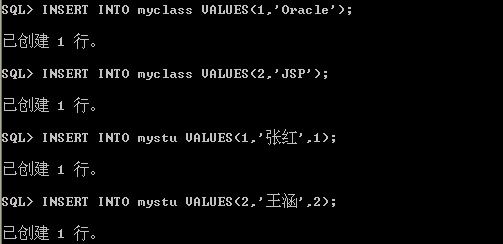
语法说明如下：

* exception\_name

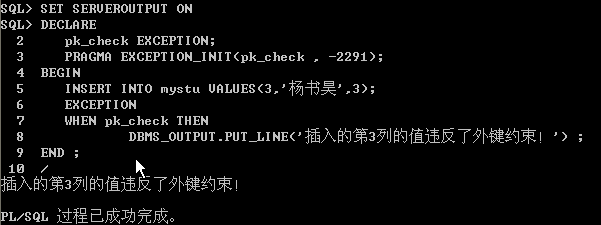
设置异常名称，该名称需要事先使用EXCEPTION类 型进行定义。

* oracle\_error\_number

Oracle错误号，该错误号与错误代码相关联，例如错误代码为ORA-06530，则错误号为-6530。这里说的错误代码是实 际存在的，由Oracle提供。



可以看到上面的错误代码为ORA-02291



## PL/SQL高级

### 子程序

前面介绍了PL/SQL程序块的使用，不过这些PL/SQL都是匿名块，也就是说它们都没有名字，当需要再次使用这些程序块时，只能再次编写程序块的内容，然后由Oracle重新编译并执行。为了提高系统的应用性能，Oracle提供了一系列“命名程序块”，也可以称为“子程序”，包括存储过程、函数、触发器和程序包。这些命名程序块在创建时由Oracle系统编译并保存，需要时可以通过名字调用它们，并且不需要编译。

### 存储过程

存储过程是一组为了完成特定功能的SQL语句集，它大大提高了SQL语句的功能和灵活性。存储过程经编译后存储在数据库中，所以执行存储过程要比执行存储过程中封装的SQL语句更有效率。

存储过程的参数不能指定长度或精度。

创建存储过程需要使用CREATE PROCEDURE语句，其语法如下：

|  |
| --- |
| CREATE [ OR REPLACE ] PROCEDURE procedure\_name  [ ( parameter [ IN | OUT | IN OUT ] data\_type ) [ , … ] ]-- 创建过程，可指定运行过程需传递的参数  { IS | AS }  [ declaration\_section ; ]  BEGIN  procedure\_body ; -- 包括在过程中要执行的语句  [EXCEPTION--处理异常  <exception handlers>]  END [ procedure\_name ] ; |

* 调用存储过程

过程创建好后，其过程体中的内容并没有被执行，仅仅只是被编译，要想执行过程中的内容还需要调用该过程。调用过程有两种形式，一种是使用CALL语句，另一种是使用EXECUTE语句，如下：

|  |
| --- |
| CALL procedure\_name ( [ parameter [ , … ] ] ) ; |

或

|  |
| --- |
| EXEC[UTE] procedure\_name [ ( parameter [ , … ] ) ] ; |

示例：

简单的存储过程：打印一个数字

|  |
| --- |
| CREATE OR REPLACE PROCEDURE myproc  AS i NUMBER ;  BEGIN  i := 100 ;  DBMS\_OUTPUT.put\_line('i = '||i) ;  END ; |

调用存储过程

|  |
| --- |
| exec myproc |

示例2：

|  |
| --- |
| CREATE OR REPLACE PROCEDURE  find\_emp(emp\_no NUMBER)  AS  empname VARCHAR2(20);  BEGIN  SELECT ename INTO empname  FROM EMP WHERE empno = emp\_no;  DBMS\_OUTPUT.PUT\_LINE('雇员姓名是 '|| empname);  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  DBMS\_OUTPUT.PUT\_LINE ('雇员编号未找到');  END find\_emp;  / |

#### 带参数的存储过程

过程参数的三种模式：

* IN

用于接受调用程序的值

默认的参数模式

* OUT

用于向调用程序返回值

* IN OUT

用于接受调用程序的值

并向调用程序返回更新的值

##### IN参数

IN参数是指输入参数，由存储过程的调用者为其赋值（也可以使用默认值）。如果不为参数指定模式，则其模式默认为IN。

在调用带IN参数的存储过程时，就需要为该过程的两个输入参数赋值，赋值的形式主要有如下两种：

* 不指定参数名

不指定参数名，是指调用过程时只提供参数值，而不指定该值赋予哪个参数，Oracle会自动按过程中参数的先后顺序为参数赋值，如果值的个数（或数据类型）与参数的个数（或数据类型）不匹配，则会返回错误。

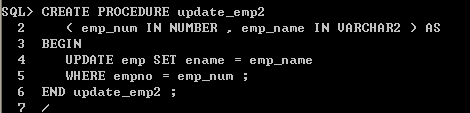
* 指定参数名

指定参数名，是指在调用过程时不仅提供参数值，还指定该值所赋予的参数。在这种情况下，可以不按参数顺序赋值。指定参数名的赋值形式为：

|  |
| --- |
| parameter\_name => value。 |

* 示例

创建存储过程



注意，上面的in参数不能指定精度或长度。

不指定参数名的调用方式：



指定参数名的调用方式



##### OUT参数使用

OUT参数是指输出参数，由存储过程中的语句为其赋值，并返回给用户。使用这种模式的参数，必须在参数后面添加OUT关键字。

调用存储过程时，如果需要显示该过程中OUT参数的返回值，还需要事先使用VARIABLE语句声明对应的变量接受返回值，并在调用过程时绑定该变量，形式如下：

|  |
| --- |
| VARIABLE variable\_name data\_type ;  [ , … ] |

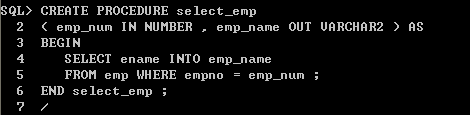
调用

|  |
| --- |
| EXEC[UTE] procedure\_name ( : variable\_name [ , … ] ) ; |

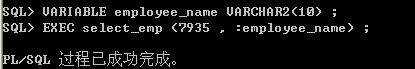
注：在EXECUTE语句中绑定变量时，需要在变量名前添加冒号（:）。

* 示例

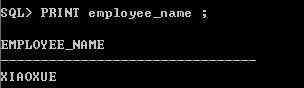
创建过程



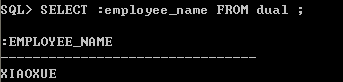
调用



使用PRINT命令查看值



也可以使用SELECT语句查看



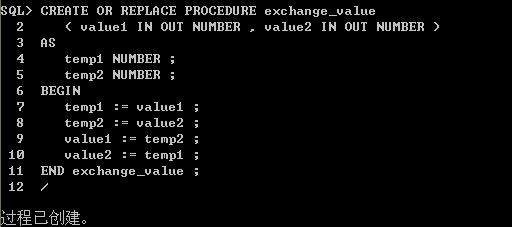
##### IN OUT参数

IN OUT参数同时拥有IN与OUT参数的特性，它既接受用户的传值，又允许在过程体中修改其值，并可以将值返回。使用这种模式的参数需要在参数后面添加IN OUT关键字。不过，IN OUT参数不接受常量值，只能使用变量为其传值。

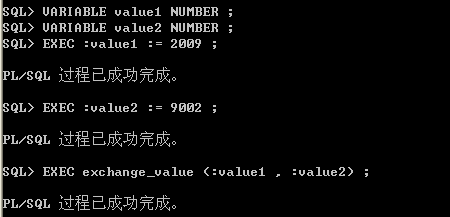
使用EXEC[UTE]命令可以为变量赋值，并且变量名前也需要添加冒号（:）。

* 示例

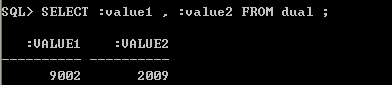
创建过程，交换二个输入参数的值



调用 ，先声明二个变量 ，然后赋值，再调用。



查看值



##### 参数的默认值

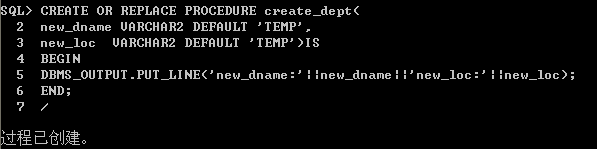
参数的默认值声明语法如下：

|  |
| --- |
| parameter\_name parameter\_type {[DEFAULT | :=]}value |

调用过程时，如果忽略了默认参数，默认值将被用到。

* 示例

创建过程



三种调用方式



#### 操作存储过程

##### 查询存储过程的定义信息

对于创建好的存储过程，如果想要了解其定义信息，可以查询数据字典**user\_source**。

【例】通过数据字典**user\_source**查询存储过程select\_emp的定义信息，如下：

|  |
| --- |
| SQL> COLUMN text FORMAT A30;  SQL> COLUMN name FORMAT A20;  SQL> SELECT \* FROM user\_source WHERE name = 'SELECT\_EMP' ; |

##### 修改与删除存储过程

修改存储过程是在CREATE PROCEDURE语句中添加OR REPLACE关键字，其它内容与创建存储过程一样，其实质是删除原有过程，然后创建一个全新的过程，只不过前后两个过程的名称相同而已。

删除存储过程需要使用DROP PROCEDURE语句，其语法形式如下：

|  |
| --- |
| DROP PROCEDURE procedure\_name ; |

##### 将过程的执行权限授予其他用户

|  |
| --- |
| SQL> GRANT EXECUTE ON find\_emp TO MARTIN;  SQL> GRANT EXECUTE ON swap TO PUBLIC; |

### 函数

函数用于返回特定数据。如果在应用程序中经常需要通过执行SQL语句来返回特定数据，那么可以基于这些操作建立特定的函数。

函数和过程的结构很相似，它们都可以接受输入值并向应用程序返回值。区别在于，过程用来完成一项任务，可能不返回值，也可能返回多个值，过程的调用是一条PL/SQL语句；函数包含RETURN子句，用来进行数据操作，并返回一个单独的函数值，函数的调用只能在一个表达式中。

#### 创建函数

创建函数需要使用CREATE FUNCTION语句，其语法如下：

|  |
| --- |
| **CREATE [OR REPLACE] FUNCTION**  **<function name> [(param1,param2)]**  **RETURN <datatype> IS|AS**  **[local declarations]**  **BEGIN**  **Executable Statements;**  **RETURN result;**  **EXCEPTION**  **Exception handlers;**  **END [ function\_name ];** |

* **data\_type**

返回值的数据类型是必需的，因为调用函数是作为表达式的一部分。

#### 定义函数的限制

* 函数只能接受 IN 参数，而不能接受 IN OUT 或 OUT 参数
* 形参不能是 PL/SQL 类型
* 函数的返回类型也必须是数据库类型

访问函数的两种方式：

* 使用 PL/SQL 块
* 使用 SQL 语句
* 示例

创建函数

|  |
| --- |
| CREATE OR REPLACE FUNCTION fun\_hello  RETURN VARCHAR2  IS  BEGIN  RETURN '朋友，您好';  END;  / |

从 SQL 语句调用函数

|  |
| --- |
| SQL> SELECT fun\_hello FROM DUAL; |

* 示例2

|  |
| --- |
| CREATE OR REPLACE FUNCTION  item\_price\_range (price NUMBER)  RETURN VARCHAR2 AS  min\_price NUMBER;  max\_price NUMBER;  BEGIN  SELECT MAX(ITEMRATE), MIN(ITEMRATE)  INTO max\_price, min\_price  FROM itemfile;  IF price >= min\_price AND price <= max\_price  THEN  RETURN '输入的单价介于最低价与最高价之间';  ELSE  RETURN '超出范围';  END IF;  END;  / |

在PL/SQL中调用

|  |
| --- |
| DECLARE  P NUMBER := 300;  MSG VARCHAR2(200);  BEGIN  MSG := item\_price\_range(300);  DBMS\_OUTPUT.PUT\_LINE(MSG);  END;  / |

#### 过程和函数的比较

|  |  |
| --- | --- |
| **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 |

### 程序包

使用程序包主要是为了实现程序模块化，程序包可以将相关的存储过程、函数、变量、常量和游标等PL/SQL程序组合在一起，通过这种方式可以构建供程序人员重用的代码库。另外，当首次调用程序包中的存储过程或函数等元素时，Oracle会将整个程序包调入内存，在下次调用包中的元素时，Oracle就可以直接从内存中读取，从而提高程序的运行效率。

程序包是对相关过程、函数、变量、游标和异常等对象的封装。

程序包主要分为两个部分：包规范和包主体。其中：

* 包规范用于列出包中可用的存储过程、函数和游标等元素条目（不包含这些元素的实际代码），这些条目属于公有项目，声明程序包中公共对象。包括类型、变量、常量、异常、游标规范和子程序规范等，可以供所有的数据库用户访问;
* 包主体中则包含了元素的实际代码，同时，也可以在包体中创建规范中没有提到的项目，那么这些项目都属于私有项目，只能在包体中使用。（声明程序包私有对象和实现在包规范中声明的子程序和游标）

#### 创建包规范

创建包规范需要使用CREATE PACKAGE语句，其简要语法如下：

|  |
| --- |
| CREATE [ OR REPLACE ] PACKAGE package\_name  { IS | AS }  package\_specification ;  END package\_name ; |

语法说明如下：

* package\_name 创建的包名。
* package\_specification 用于列出用户可以使用的公共存储过程、函数、类型和对象。

程序包优点：

* 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

#### 创建包体

创建包体需要使用CREATE PACKAGE BODY语句，并且在创建时需要指定已创建的包，其简要语法如下：

|  |
| --- |
| CREATE [ OR REPLACE ] PACKAGE BODY package\_name  { IS | AS }  package\_body ;  END package\_name ; |

* 示例

创建程序包

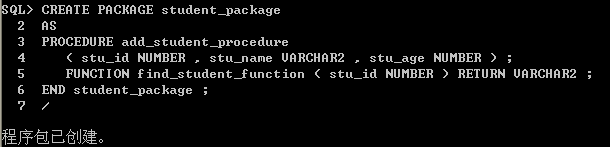
|  |
| --- |
| CREATE OR REPLACE PACKAGE pack\_me  IS  PROCEDURE order\_proc (orno VARCHAR2);  FUNCTION order\_fun(ornos VARCHAR2) RETURN VARCHAR2;  END pack\_me; |

创建包主体

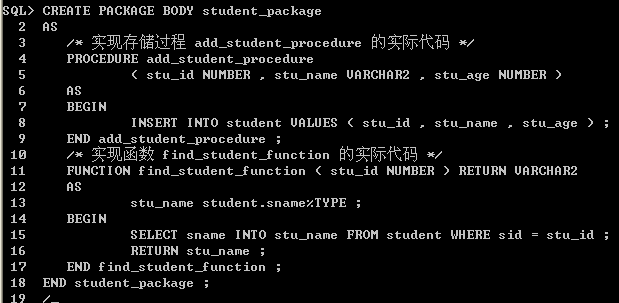
|  |
| --- |
| CREATE OR REPLACE PACKAGE BODY pack\_me AS  PROCEDURE order\_proc (orno VARCHAR2) IS  stat CHAR(1);  BEGIN  SELECT ostatus INTO stat FROM order\_master;  ……  END order\_proc;  FUNCTION order\_fun(ornos VARCHAR2)  RETURN VARCHAR2  IS  icode VARCHAR2(5);  BEGIN  ……  END order\_fun;  END pack\_me;  / |

* 示例2

创建程序包



创建程序包主体



执行存储过程



### 触发器

触发器是一种特殊的存储过程，它在发生某种数据库事件时由Oracle系统自动触发。触发器通常用于加强数据的完整性约束和业务规则等，对于表来说，触发器可以实现比CHECK约束更为复杂的约束。

#### 触发器的类型

Oracle中的触发器的类型主要有DML触发器、替代触发器、系统事件触发器和DDL触发器。

* **DML触发器**

DML触发器由DML语句触发，例如INSERT、UPDATE和DELETE语句。

针对所有的DML事件，按触发的时间可以将DML触发器分为BEFORE触发器与AFTER触发器，分别表示在DML事件发生之前与之后采取行动。

另外，DML触发器也可以分为语句级触发器与行级触发器，其中，语句级触发器针对某一条语句触发一次，而行级触发器则针对语句所影响的每一行都触发一次。例如某条UPDATE语句修改了表中的100行数据，那么针对该UPDATE事件的语句级触发器将被触发一次，而行级触发器将被触发100次。

* **INSTEAD OF触发器**

INSTEAD OF触发器又称替代触发器，用于执行一个替代操作来代替触 发事件的操作。例如针对INSERT事件的INSTEAD OF触发器，它由 INSERT语句触发，当出现INSERT语句时，该语句不会被执行，而是执 行INSTEAD OF触发器中定义的语句。

* **系统事件触发器**

系统事件触发器在发生如数据库启动或关闭等系统事件时触发。

* **DDL触发器**

DDL触发器由DDL语句触发，例如CREATE、ALTER和DROP语句。

DDL触发器同样可以分为BEFORE触发器与AFTER触发器。

#### 创建触发器

创建触发器需要使用CREATE TRIGGER语句，其语法如下：

|  |
| --- |
| CREATE [ OR REPLACE ] TRIGGER trigger\_name  [ BEFORE | AFTER | INSTEAD OF ] trigger\_event  { ON table\_name | view\_name | DATABASE }  [ FOR EACH ROW ]  [ ENABLE | DISABLE ]  [ WHEN trigger\_condition ]  [ DECLARE declaration\_statements ; ]  BEGIN  trigger\_body ;  END [ trigger\_name ] ; |

语法说明：

* trigger\_name 创建的触发器名称。
* BEFORE | AFTER | INSTEAD OF BEFORE表示触发器在触发事件执行之前被激活；AFTER表示触发器在触发事件执行之后被激活。INSTEAD OF表示用触发器中的事件代替触发事件执行。
* trigger\_event 表示激活触发器的事件。
* ON table\_name | view\_name | DATABASE table\_name指定DML触发器所针对的表。如果是INSTEAD OF触发器，则需要指定视图名（view\_name）；如果是DDL触发器或系统事件触发器，则使用ON DATABASE。
* FOR EACH ROW 表示触发器是行级触发器。如果不指定此子句，由默认为语句级触发器。用于DML触发器与INSTEAD OF触发器。
* ENABLE | DISABLE 此选项是Oracle Database 11g新增加的特性，用于指定触发器被创建之后的初始状态为启用状态（ENABLE）还是禁用状态（DISABLE），默认为ENABLE。
* WHEN trigger\_condition 为触发器的运行指定限制条件。例如针对UPDATE事件的触发器，可以定义只有当修改后的数据符合某种条件时才执行触发器中的内容。
* trigger\_body 触发器体。包含触发器的内容。

#### DML触发器

#### INSTEAD OF触发器

#### 行触发器

#### 系统事件触发器