第一章　Oracle数据库基础

Neusoft 东软

本章要点

- 数据库管理系统概述
- 关系数据库完整性原则
- Oracle数据库对象、安全及网络访问
- 安装Oracle软件和创建数据库
- 常用工具SQL*Plus及企业管理器
- DBCA数据库配置助手
- NETCA/NETMGR网络配置工具
- 数据库实例的启动和停止
- Oracle的网络连接基本管理

数据库管理系统概述

- 数据库（DB）
- 数据库管理系统（ Database Management System，DBMS）管理数据库的软件。具有对数据存储、安全、一致性、并发操作、恢复和访问等功能。
- 数据词典（系统目录），也是一种数据，只不过这些数据记录的是数据库中存放的各种对象的定义信息和其他一些辅助管理信息，包括名字、结构、位置、类型等。这些数据被称为元数据（metadata）。
- 数据库系统管理员（DBA），数据库管理人员。

数据库的发展

- 数据管理主要经历过程:
  - 手工管理阶段： 应用程序管理数据、数据不保存、不共享、不具有独立性。
  - 文件管理阶段： 文件系统管理数据、数据可长期保存、但共享性差、冗余度大、独立性差。
  - 数据管理阶段: 数据库系统管理数据、数据结构复杂、冗余小、易扩充、较高的独立性、统一数据控制。

数据库的特征

- 数据结构化
- 实现数据共享
- 减少数据冗余
- 数据独立性

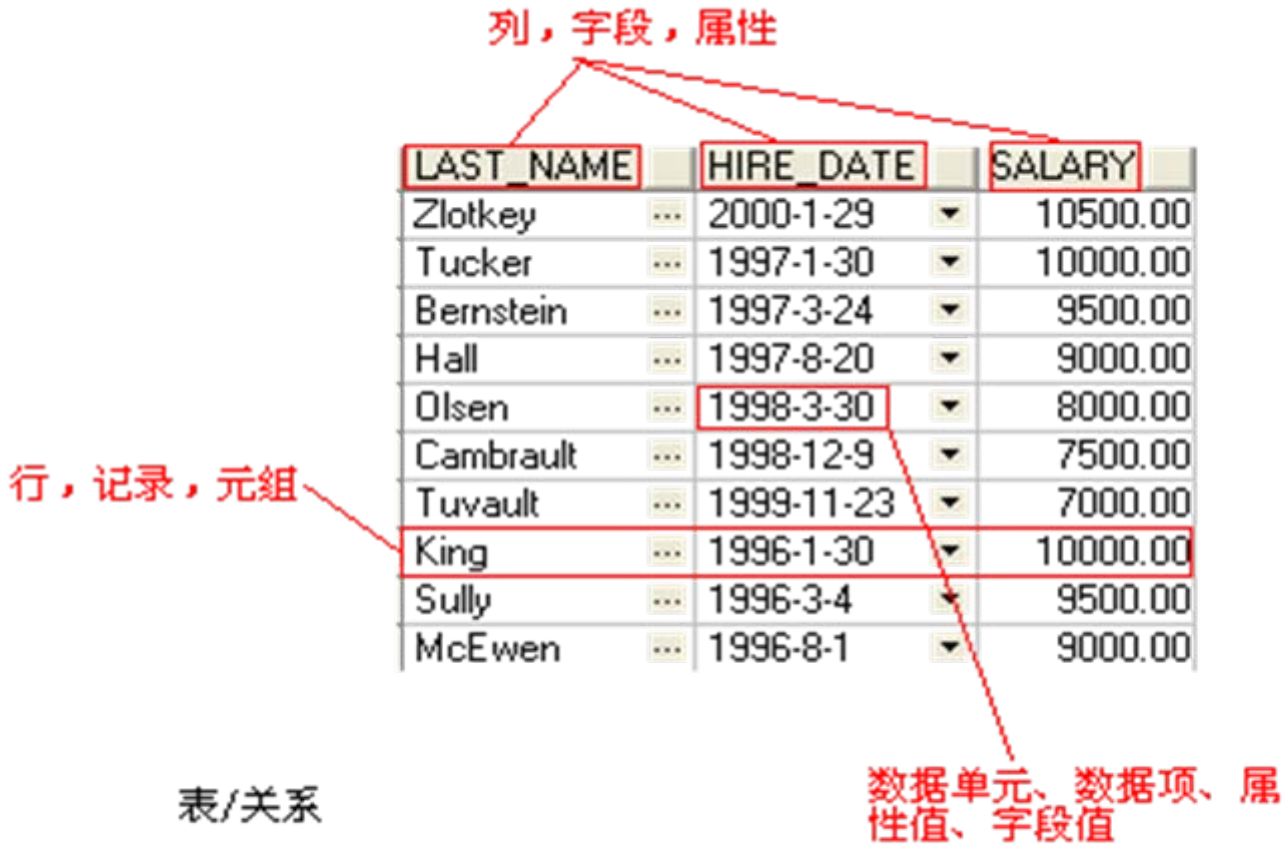
数据库的类型

- 按数据模型特点分:
  - 网状型数据库
  - 层次型数据库
  - 关系型数据库

关系数据库的数据结构

- 关系数据库是指一些相关的表和其他数据库对象的集合。对于关系数据库来说，关系就是表的同义词。
- 表是由行和列组成（类似二维数组的结构）。
  - 列包含一组命名的属性（也称字段）。
  - 行包含一组记录，每行包含一条记录。
  - 行和列的交集称为数据项，指出了某列对应的属性在某行上的值，也称为字段值。
  - 列需定义数据类型，比如整数或者字符型的数据。

关系数据库的数据结构图示



关系数据库的数据操作语句

- 常用数据库操作语句有SELECT，INSERT，UPDATE和DELETE
- 常用数据库定义语句有CREATE，ALTER和DROP
- 其他语句

关系数据库的完整性原则

- 关系模型的完整性原则
  - 实体完整性原则
  - 引用完整性原则
- 主关键字（primary key）指表中的某一列或多列组合，该列或组合的值唯一标识一行。即主关键字的值必须唯一或不允许为NULL。
- 外关键字（foreign key）指一个表中的一列或一组列，他们在其他表中作为主键或唯一键而存在。一个表的外键被认为是对另一个表中主键的引用。

常见关系数据库

- Oracle
- DB2
- Sybase
- Microsoft SQL Server
- ACCESS

Oracle数据库简介

- Oracle数据库软件是Oracle公司开发的关系型数据库产品，支持各种操作系统平台，包括Windows、Linux和Unix等，目前Oracle在关系型数据库产品领域内处于领先地位。
- Oracle的最新的数据库软件版本是10g，8i和9i的版本也仍然在广泛使用中。

Oracle数据库的典型特征

- 支持海量存储、多用户并发高性能事务处理。
- 多种备份和恢复策略。包括高级复制，物理和逻辑的24*7备份和恢复工具，异地容载实现等。
- 开放式联结。给各种其他应用提供了统一的接口，并可以接入很多其他传统应用程序。
- 遵循SQL语言规范，支持各种操作系统、用户接口和网络通信协议的工业标准。
- 第一个实现网格计算的数据库（10g版本）

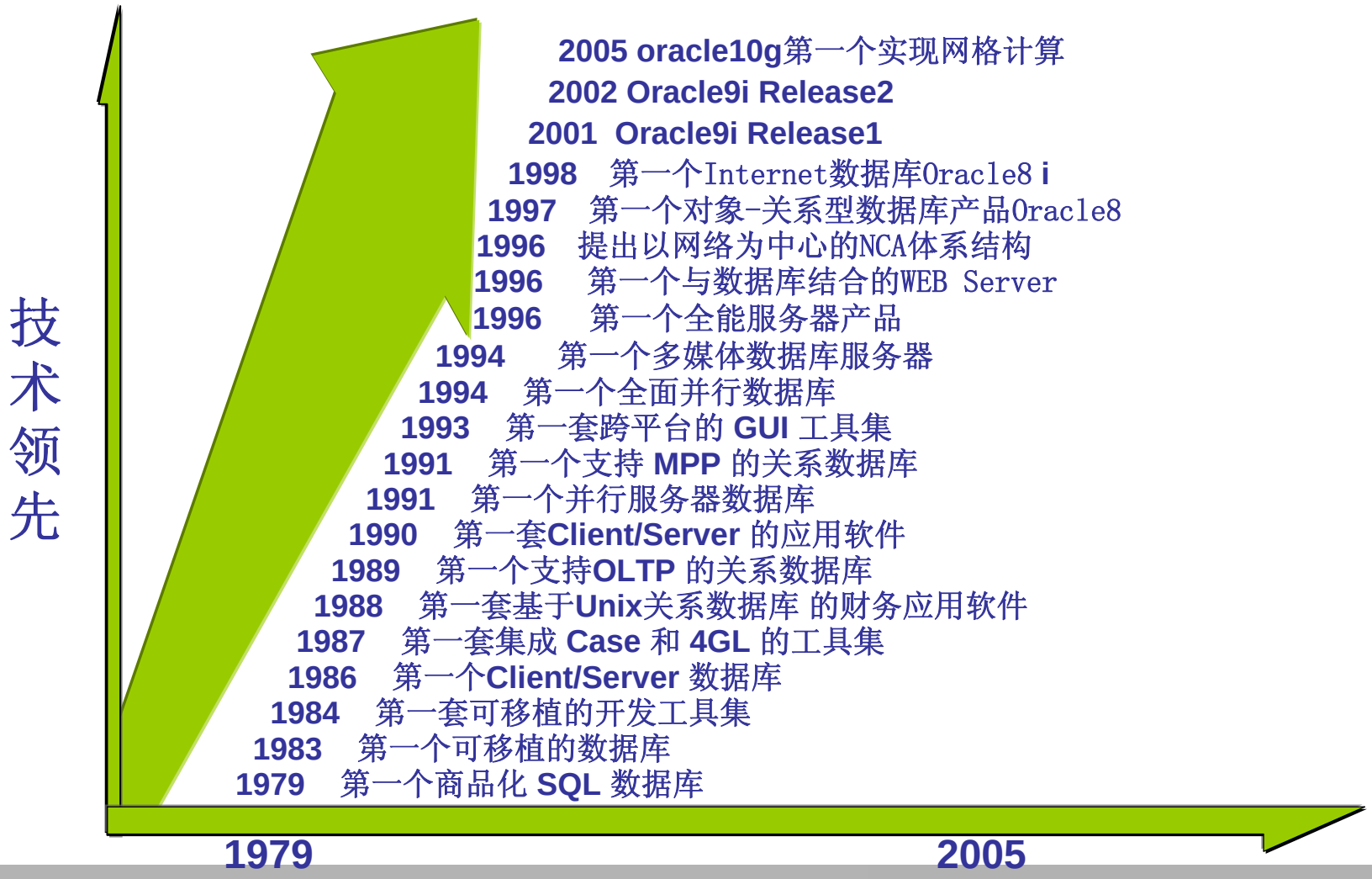
Oracle数据库的典型特征（续）

- 应用集群实现可用性和可伸缩性
- 业界领先的安全性
- 借助自我管理使数据库降低管理成本

Oracle公司发展史

- 1977年，Larry Ellison、Bob Miner和Ed Oates建立SDL。引入了E.F.Codd教授的新型数据库，引入了SQL语言。
- 1978年，Relational Software公司诞生。
- 1979年，第一个商业数据库产品出现。

Oracle的第一



Oracle 在中国

- 中国数据库市场最大厂商
- 在应用软件领域迅速增长
- 在各个行业中应用广泛

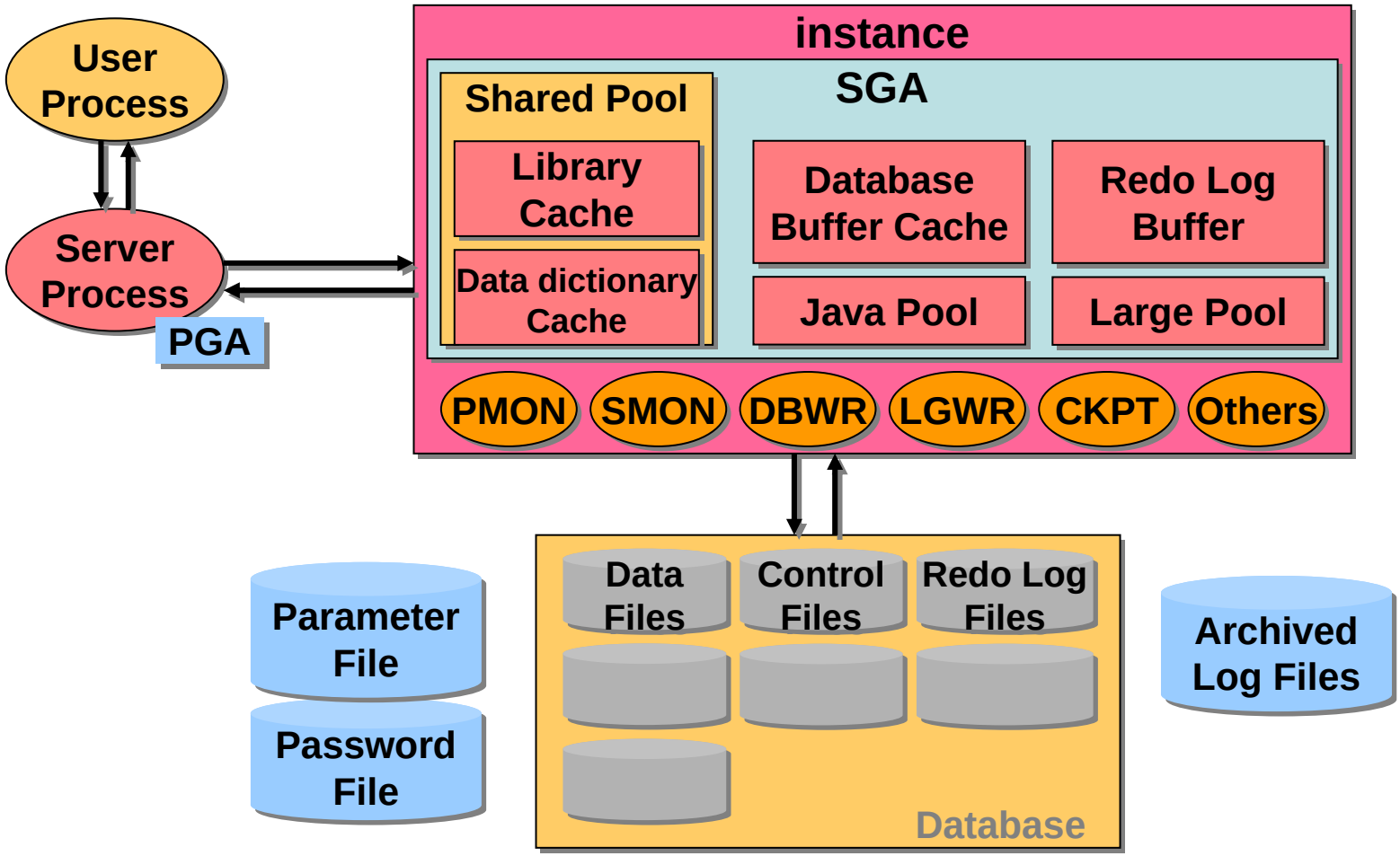
Oracle主要产品

- 数据库
  oracle7.3、oracle8、 oracle8i、 oracle9i、oracle10g
- 应用服务器：IAS 9i。
- ERP产品：企业管理组件，包括财务管理、人力资源管理、生产管理等模块。
- 开发工具包：包括了Form、Report、Designer、Jdeveloper等可视化的开发工具包，可方便快捷的开发基于Oracle数据库的应用程序。
- 数据仓库产品：包括Discover、OWR、Express等数据仓库构建、数据挖掘与分析等软件包。

Oracle数据库特点

- 支持大数据库、多用户的高性能的事务处理
- Oracle遵守数据存取语言、操作系统、用户接口和网络通信协议的工业标准
- 实施安全性控制和完整性控制
- 支持分布式数据库和分布处理
- 具有可移植性、可兼容性和可连接性
- 全球化、跨平台的数据库

Oracle体系结构



数据库文件与存储

- 数据库核心文件：数据文件（Data files），重做日志文件（Redo Log Files）和控制文件（Control files）。
- 数据文件：存放的是用户的数据和系统的数据。一个Oracle数据库一般会包含多个数据文件。
- 重做日志文件：记录了系统改变的日志。主要用于数据库的恢复。因此，又称为“重做日志文件”。
- 控制文件：记录数据库的一些核心配置数据。

数据库文件与存储（续）

- 参数文件（Parameter file），参数文件是在实例启动的时候，配置实例运行相关的一些参数（比如内存分配的大小，实例运行出错的日志存放位置等）。
- 表空间是Oracle内部自己的一套数据存储组织形式。从物理上获得空间后，Oracle自己来管理这些空间的使用。这些可用的空间组织成逻辑存储的主要单位称之为表空间。
- 表空间由一个或多个数据文件组成，逻辑结构（表空间）和物理结构（文件）关联后，其后对于空间的分配则完全是由Oracle内部在表空间中自动进行管理的。

数据库对象

表	表是用来存放用户数据的对象，由行和列组成，列就是字段，行就是表中的记录
约束	保证数据完整性的规则，设置在单个字段或者多个字段组合上，写入这些字段的数据必须符合约束的限制
视图	虚表，是一个命名的查询，用于改变基表数据的显示，简化查询。访问方式与表相同，同样可使用查询语句
索引	构建于表的单字段或者字段组合上，用于加速对表中数据的查询
序列	产生顺序的不重复数字串，被作为主键约束值的参照
同义词	数据库对象的别名
存储过程	用于完成某种特定的功能的PL/SQL程序，存储在数据库中
函数	用于进行复杂计算的PL/SQL函数，返回一个计算结果，存储在数据库中
触发器	由事件触发而执行的PL/SQL程序，用于在特定的时机执行特定的任务，存储在数据库中
包	一组相关的函数和存储过程的命名集合，存储在数据库中

数据库安全

用户	用于组织和管理数据库对象，通常一个应用软件的数据库对象被存放在一个数据库用户下。使用数据库用户连接数据库后，可以对这些数据库对象执行操作。
方案	数据库对象的命名集合，一个方案唯一对应一个数据库用户，方案的命名与用户命名完全相同，访问数据库对象的时，采取“方案名.对象名”的方式来访问
权限	权限决定了数据库用户在数据库中能够做什么，如果用户没有权限，那么对数据库不能执行任何操作。权限由高权限用户授予。
角色	一组命名的权限，用户简化对权限的管理操作，可以一次将多个权限（一个角色的权限）授予一个或多个用户。
配额	当用户创建存储对象（如表、索引）时，需要一定的数据库的存储空间来存放这些对象，配额就是分配给用户的可使用存储空间的限制。

数据库网络访问

数据库名	数据库的名称，在控制文件中有记录，在参数文件中通过db_name指定
实例名	数据库的内存区域和后台进程的集合的总称，在参数文件中通过instance_name指定，通常与db_name相同
服务名	数据库在操作系统上被作为一个服务对待，所以对外数据库以服务的形式出现，通常访问数据库被叫做访问数据库服务，服务名通过参数service_name指定
连接字符串	通过网络访问远端服务器上的数据库时，用于描述数据库访问地址的字符串，通常的结构是“主机名（或ip）:端口号:服务名”，例如：172.17.0.100:1521：oracl
服务命名	连接字符串的别名，连接字符串书写过于复杂，所以使用服务命名代替，服务命名被用于通过网络连接数据库，通常的使用格式是：用户名/口令@服务命名
监听器	在服务端运行的一个进程，用于监听客户端到数据库的连接请求，在通过网络访问数据库时必须启动

安装需求

- 为了创建安装Oracle数据库,必须满足下面的条件:
  - 操作系统权限
  - 数据库权限
  - 足够的磁盘空间
  - 足够的内存空间

安装前的准备工作

- 创建有权限的操作系统用户和组
- 设置环境变量
- 创建安装需要的目录
- 执行安装文件开始安装

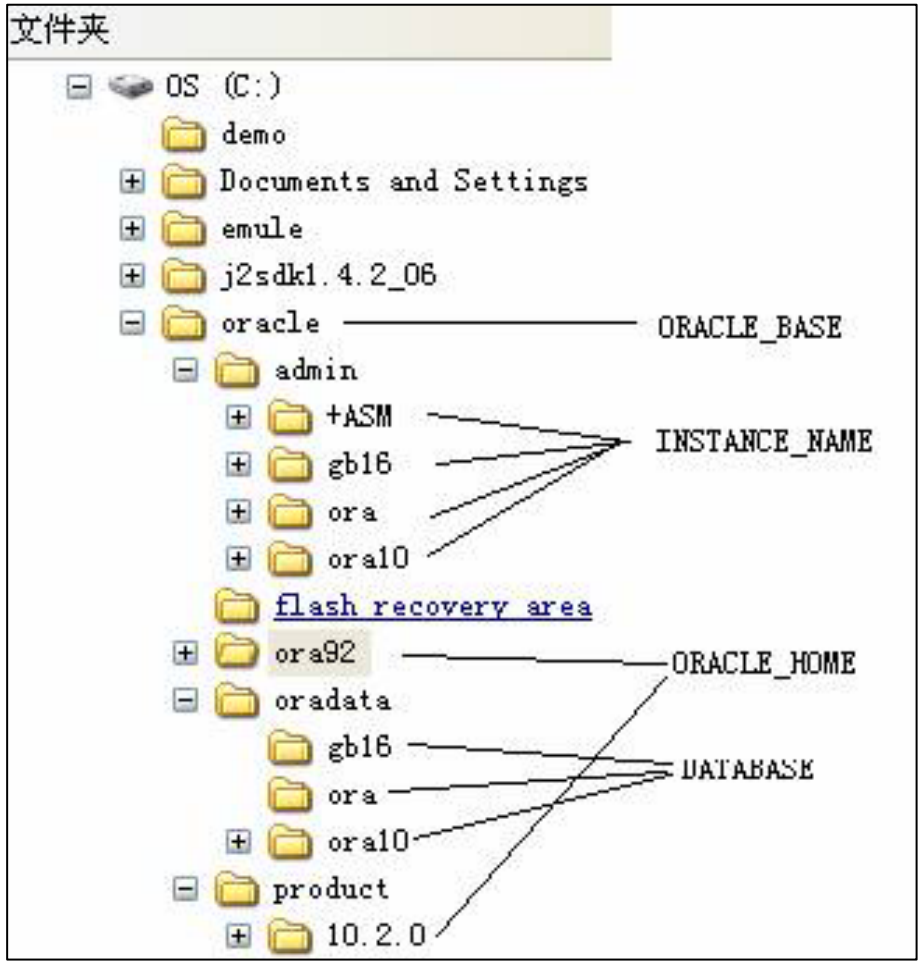
需要的操作系统用户和组

- UNIX环境
  - 需要创建一个Oracle用户和一个DBA组,并且Oracle用户属于这个组
- Windows环境
  - 不需要创建用户和组,使用在administrator组的用户安装数据库

设置环境变量

- ORACLE_BASE
- ORACLE_HOME
- ORACLE_SID
- PATH

Oracle目录结构



创建安装需要的目录

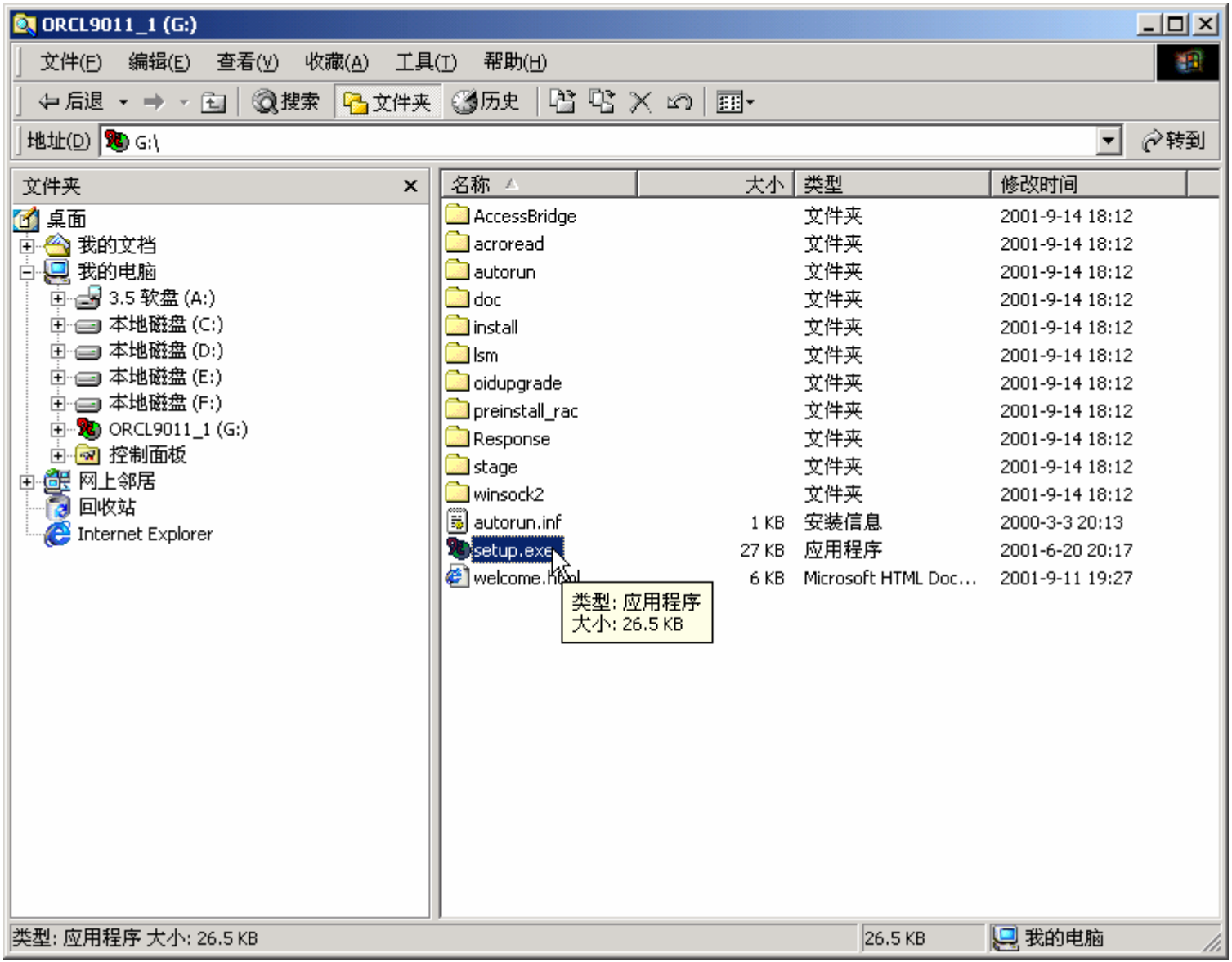
- Windows:
  - 不需要创建，windows平台上，在安装的时候会自动创建所需要的目录
- UNIX:
  - 创建目录结构,例如:
  - /oracle/product/版本号
  - 将对这个目录的访问权限完全授予Oracle用户

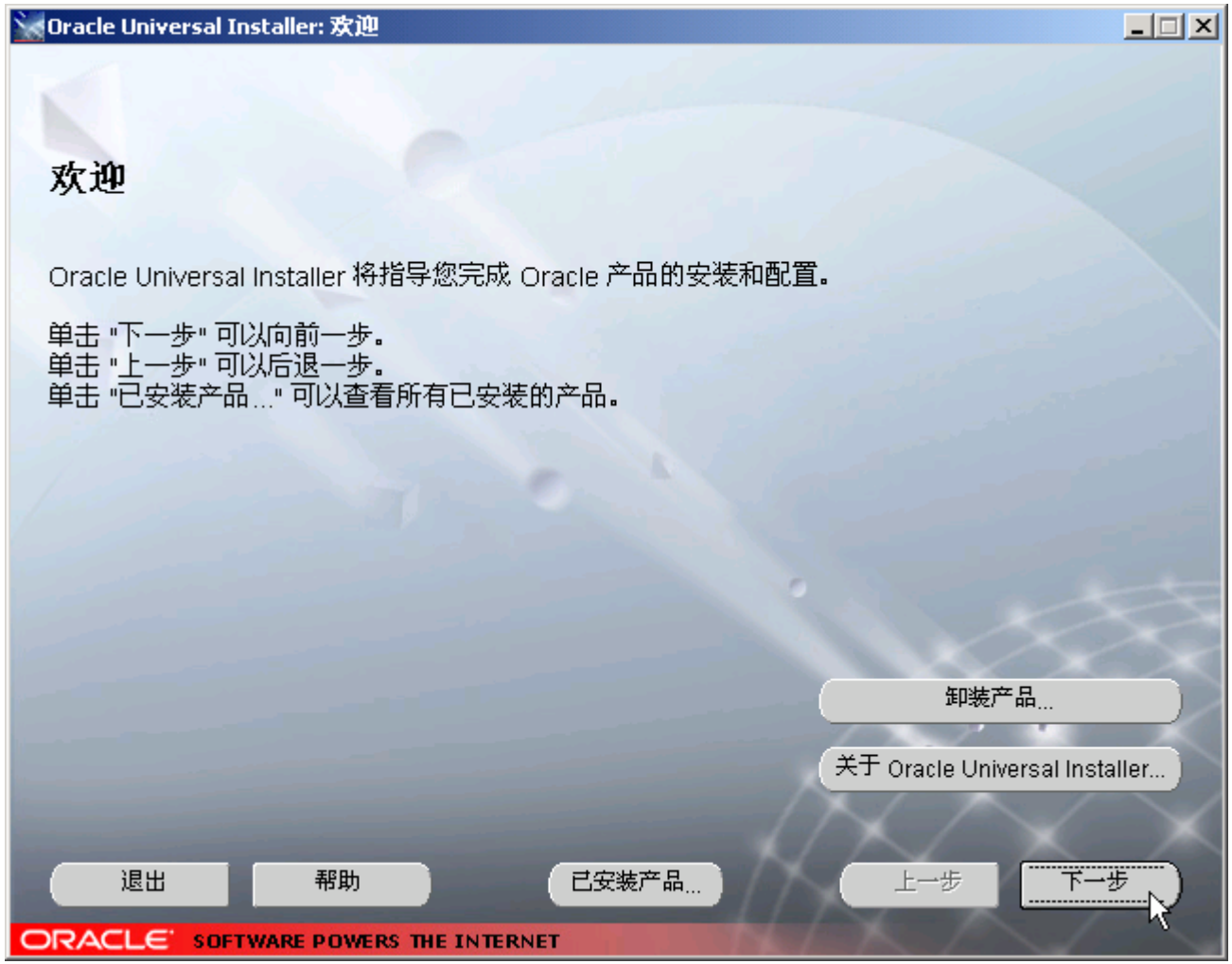
安装命令

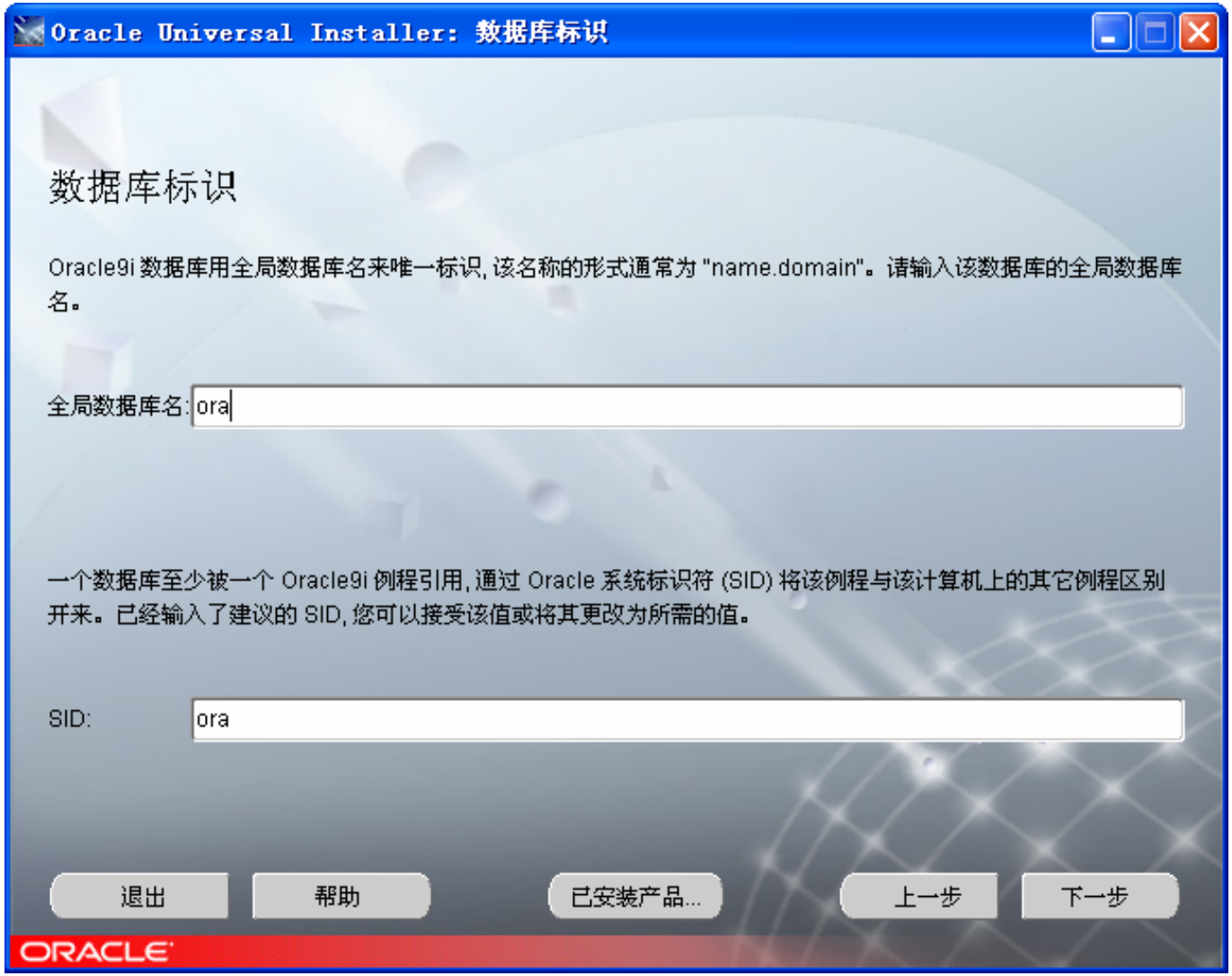
- UNIX
  - 在命令行下运行./runInstaller
- Windows
  - 运行安装目录中的setup.exe文件

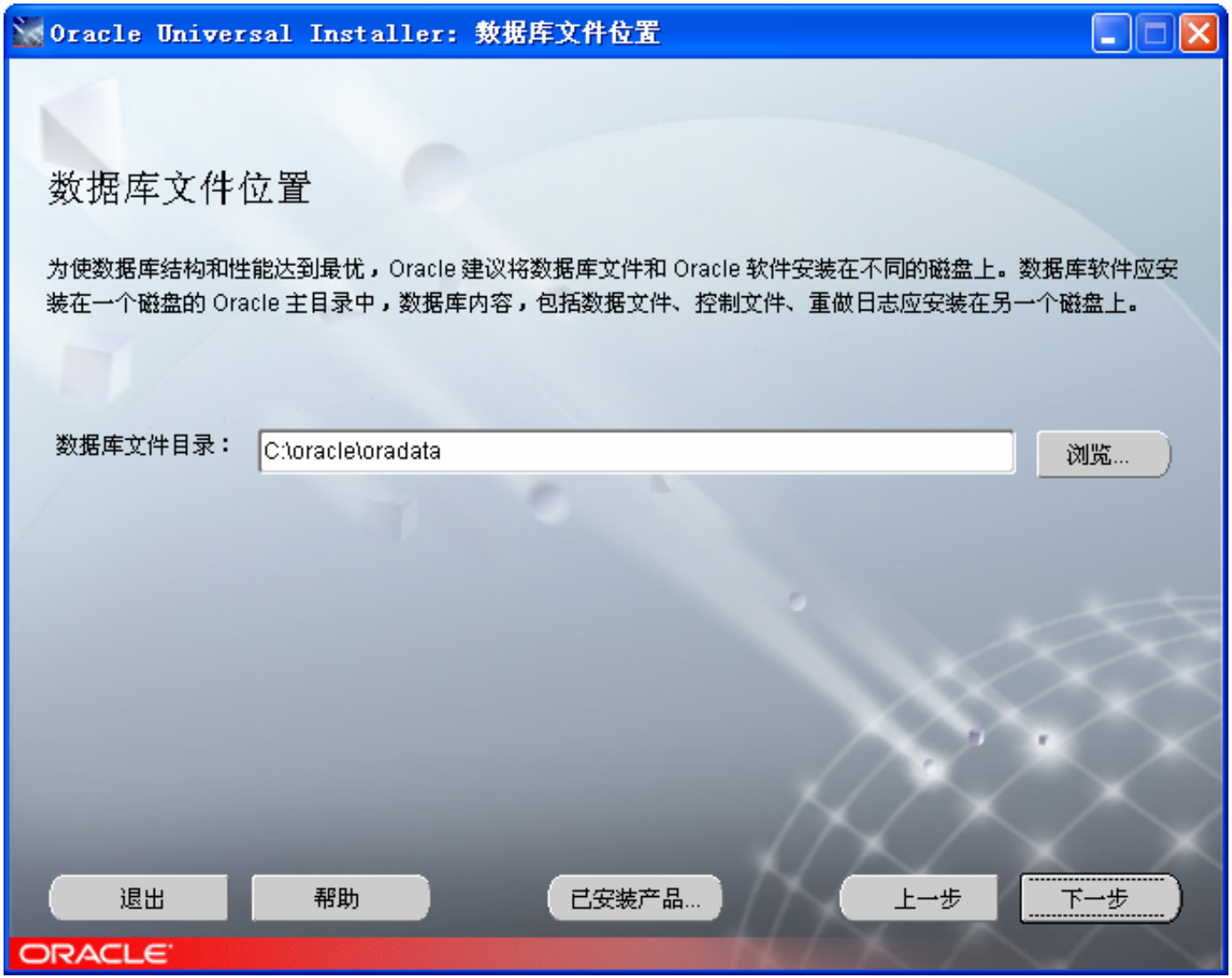
主要安装步骤

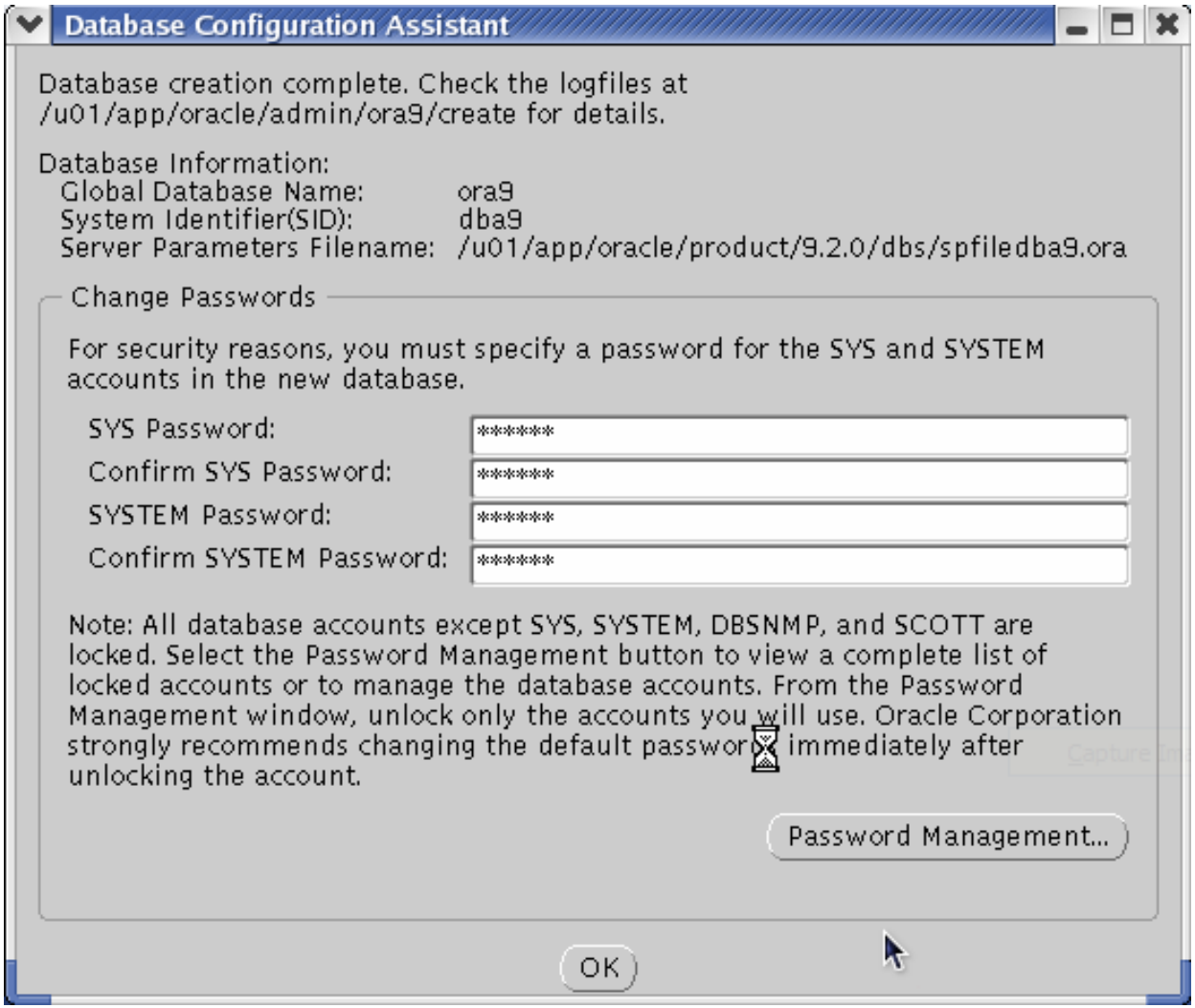
- 选择安装的产品
- 选择安装的类型
- 选择数据库配置类型
- 输入全局数据库名称和SID
- 设置数据文件存放目录
- 选择数据库字符集
- 设置网络配置
- 创建数据库











校验是否安装成功

- 在命令行下输入

```
Sqlplus system /密码
```

- 如果出现如下提示，则数据库可正常使用。

```
连接到:
Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.1.0 – Production
SQL>
```

Oracle数据库常用工具

- 朴素而强大的工具
  - SQL*Plus
- 图形化的管理工具
  - Oracle的企业管理（Oracle Enterprise Manager）
- DBCA数据库配置助手
- NETCA/NETMGR网络配置工具

SQL*PLUS工具

- SQL*Plus是强大的ORACLE内嵌工具。
- 提供Sql语句的执行环境
- 也可以用于管理数据库
- 有自己的一套SQL*Plus命令
- 可跨平台运行
- SQL*Plus命令与sql语句不相同
- 提供了标准SQL扩充命令的支持

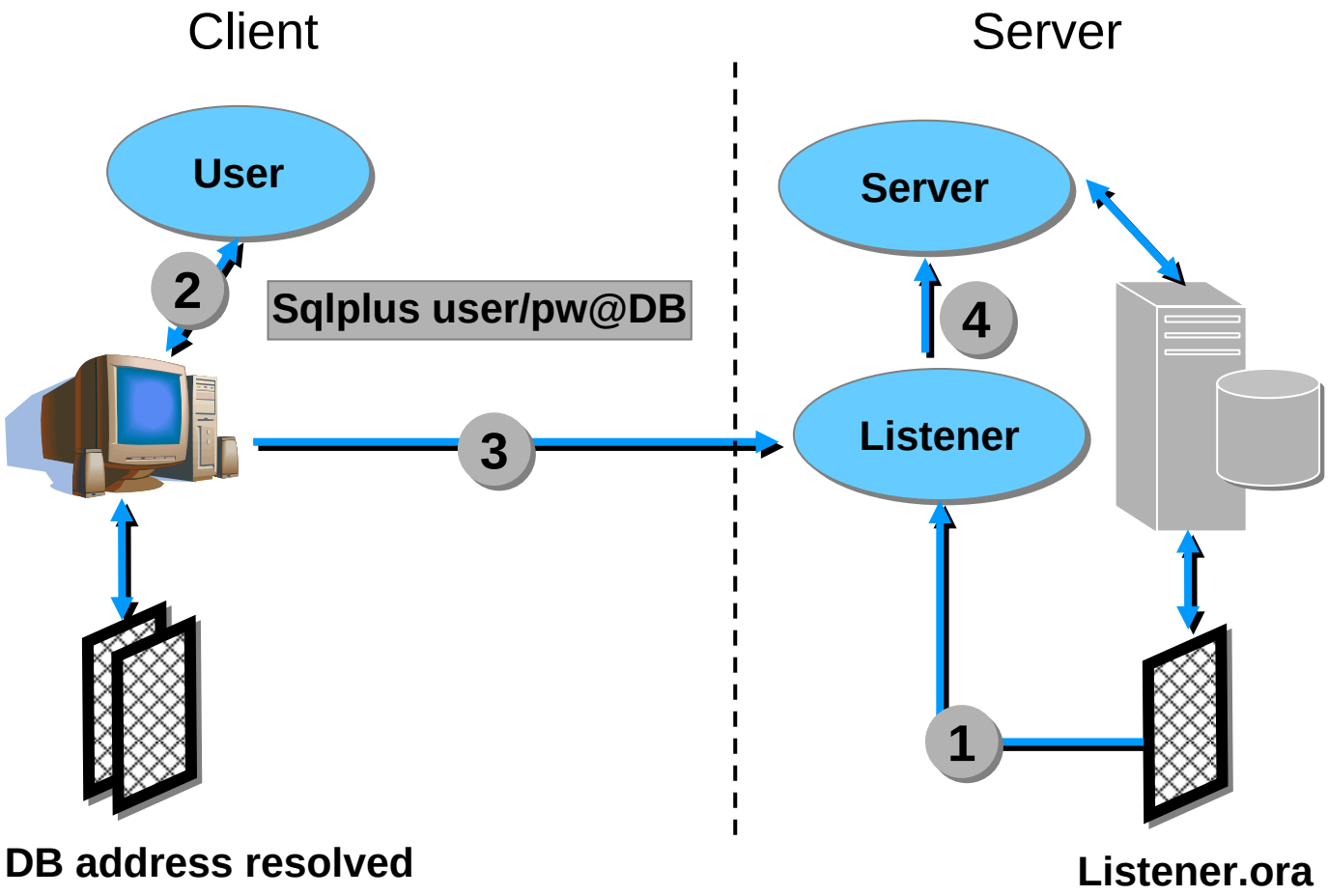
企业管理器OEM

- 图形化管理控制台，可用来管理、诊断以及调优数据库
- 有两种运行方式：
  - 独立运行（Standalone）
  - 基于OMS （Oracle Managerment Server）的OEM

实例的启动和停止

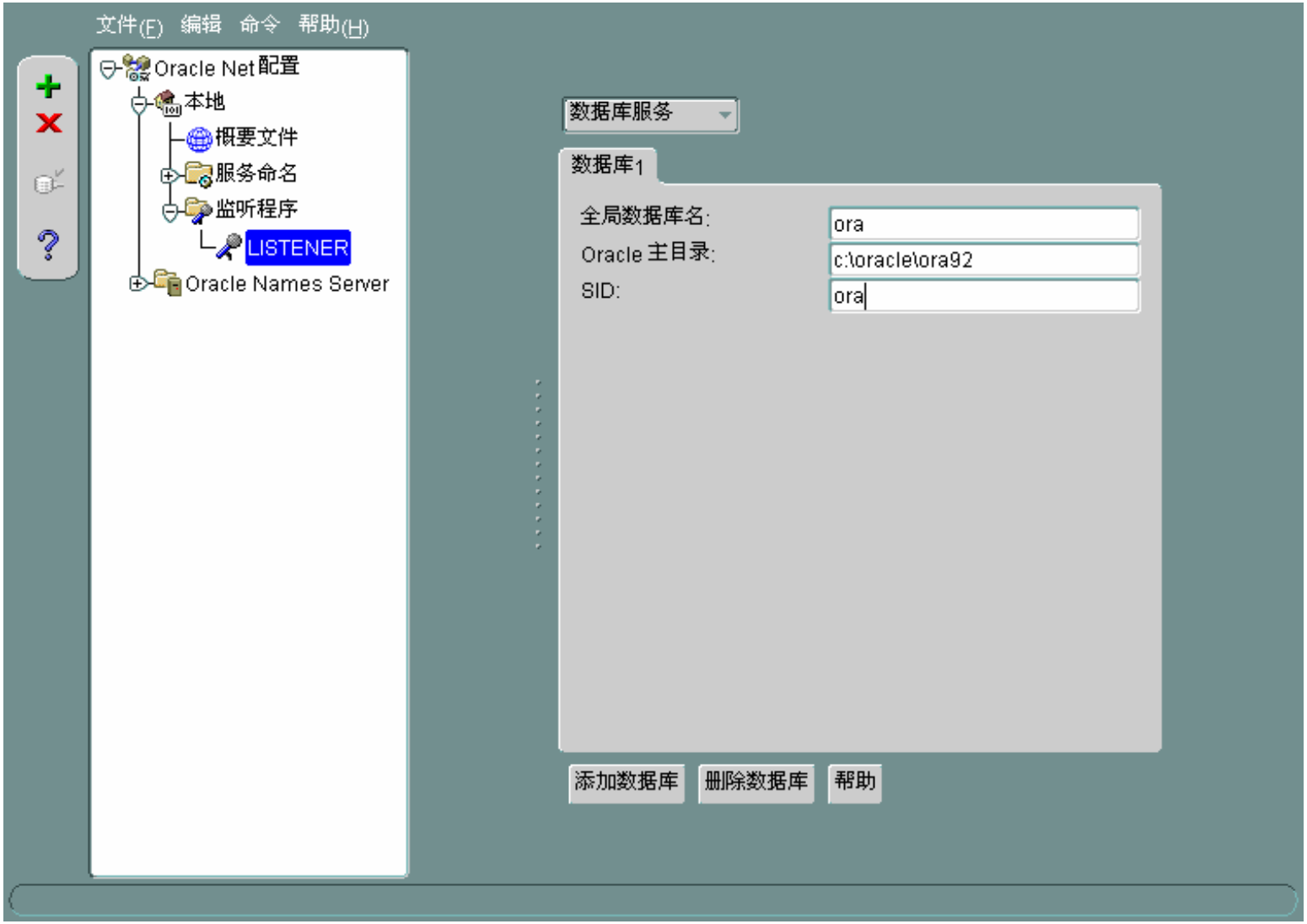
- 通常使用SYS用户以SYSDBA的身份来操作实例。
- 启动数据库实例的命令为startup，启动过程分三步
  - 启动实例，分配内存启动后台进程等。
  - 打开数据库的控制文件，并把实例和数据库关联起。
  - 打开数据库文件，可以访问数据了，也就是启动过程中提示的“数据库已经打开”。
- 关闭数据库的命令是shutdown ， 常用shutdown immediate
- 必须先启动Windows服务，然后才能对实例进行启停管理。

Oracle的网络连接结构



Oracle网络服务器配置

- Oracle服务器端要想提供网络服务，必须配置和启动监听。
- Oracle提供了两个图形化的网络配置工具
  - Net Configuration Assistant（简称NetCA）
  - Net Manager（简称NetMgr）



监听的启动和停止

- LSNRCTL是监听器的管理工具。
- 最常用的三个命令
  - lsnrctl start（启动监听）
  - lsnrctl stop（停止监听）
  - lsnrctl status（查看当前监听的运行情况）

Oracle网络客户端配置

- 客户端连接Oracle服务器，先配置Oracle客户端网络，再使用客户端配置的网络名连接Oracle服务器。
- Oracle客户端的配置文件包括sqlnet.ora和tnsnames.ora，在$ORACLE_HOME\network\admin目录中。
  - Sqlnet.ora文件内包含客户端连接服务器所采用的途径和方法配置信息。
  - tnsnames.ora文件内则包含采用最常见的连接方法（本地命名策略）时的客户端的网络配置详细信息。

使用NetCA工具配置

本章小结

- 本章主要介绍了数据库管理系统的基本概念和Oracle数据库的特点，同时对Oracle公司的主流数据库产品进行了介绍。
- 讲述了Oracle9i数据库在windows平台上的安装需求和安装步骤。
- 讲解了Oracle主要的管理和开发工具—企业管理器和Sqlplus的主要功能，对Oracle数据库的一些基本工具的使用进行了介绍。
- 讲解了oracle实例的管理及网络连接的基本操作。

练习

- 1.熟练Oracle体系结构，查看数据库文件及存储，数据库基本对象，数据库网络访问相关参数。
- 2.练习使用SQLPLUS工具。
- 3.练习使用企业管理器。
- 4.练习使用DBCA配置助手。
- 5.练习使用NETCA/NETMGR网络配置工具。
- 6.练习启动和关闭数据库实例。
- 7.练习进行Oracle网络服务器及网络客户端配置。

第二章　编写简单的SELECT语句

Neusoft 东软

本章要点

- SQL语言简介
- 基本查询语句
- SQL语句的书写规则
- 算术表达式的使用
- 空值（NULL）的应用
- 列别名的使用
- 连接运算符的使用
- DISTINCT关键字的用法
- SQL*PLUS/iSQL*PLUS命令的介绍

SQL语言简介

- SQL称结构化查询语言 (Structured Query Language)
- SQL 是操作和检索关系型数据库的标准语言。已在Oracle、DB2等数据库管理系统上得到了广泛应用。
- 使用SQL语句，程序员和数据库管理员可以完成如下的任务:
  - 改变数据库的结构
  - 更改系统的安全设置
  - 增加用户对数据库或表的许可权限
  - 在数据库中检索需要的信息
  - 对数据库的信息进行更新

SQL语句分类

- Select查询语句
- DML语句（数据操作语言）
  Insert / Update / Delete / Merge
- DDL语句（数据定义语言）
  Create / Alter / Drop / Truncate
- DCL语句（数据控制语言）
  Grant / Revoke
- 事务控制语句
  Commit / Rollback / Savepoint

课程案例环境简介

- employees（员工信息表）
  - 主要有employee_id（员工编号)、last_name（姓）、job_id(职位)、salary（工资）等。
- jobs（职位信息表）
  - 主要有job_id(职位)、job_title（职位全称）等。
- salgrades（工资级别表）
  - 主要有grade_level（工资级别）、lowest_salary（最低工资）、highest_salary（最高工资）等。

课程案例环境简介（续）

- departments(部门信息表)
  - 主要包括department_id（部门编号）、department_name(部门名称)、location_id（位置编号）等。
- locations（位置信息表）
  - 主要包括location_id(位置编号)、street_adress（地址）、city（城市）等。

SELECT基本查询语句

- 基本查询语句语法：

```
SELECT    *|{[DISTINCT] 列名|表达式 [别名][,...]}
FROM      表名;
```

  - “*”号的使用
  - 在查询语句中查找特定的列

第一条查询语句

- 例**2-1** 查询公司所有部门的信息。

```
SELECT *
FROM   departments;
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
100	Finance	108	1700
110	Accounting	205	1700
200	Operations		1700

已选择9行。

第一条查询语句（续）

- **例2-2 查询公司所有部门的信息。**

```
SELECT department_id,
    department_name,manager_id,location_id
FROM   departments;
```

试比较哪条语句执行效率更高?

在查询语句中查找特定列

- //例2-3

```
SELECT department_name, location_id
FROM   departments;
```

DEPARTMENT_NAME	LOCATION_ID
Administration	1700
Marketing	1800
Shipping	1500
IT	1400
Sales	2500
Executive	1700
Finance	1700
Accounting	1700
Operations	1700

已选择9行。

SQL语句的书写规则

- SQL语句相关概念:
  - 关键字（Keyword）：SQL语言保留的字符串，在自己的语法使用。例如，SELECT 和FROM 是关键字。
  - 语句（statement）：一条完整的SQL命令。例如，SELECT * FROM departments;是一条语句。
  - 子句（clause）：部分的SQL语句，通常是由关键字加上其他语法元素构成。例如，SELECT *是子句，FROM departments也是子句。

SQL语句书写规则

- 不区分大小写。也就是说SELECT，select，Select，执行时效果是一样的。
- 可以单行来书写，也可以书写多行。
- 关键字不可以缩写、分开以及跨行书写。如SELECT不可以写成SEL或SELE CT等形式。
- 每条语句需要以分号（；）结尾。

本书中SQL语句规范

- 关键字大写，其他语法元素（如列名、表名等）小写。
- 语句分多行书写，增强代码可读性。通常以子句分行。
- 代码适当缩进。

算术表达式的使用

- 算术运算符：+，-，*，/
- 算术表达式中优先级规则：
  - 先算乘除，后算加减。
  - 同级操作符由左到右依次计算。
  - 括号中的运算优先于其他运算符。
- 对NUMBER型数据可以使用算数操作符创建表达式（+ - * /）
- 对DATE型数据可以使用部分算数操作符创建表达式（+ -）

优先级示例

```
SELECT employee_id, last_name, salary, salary+400
FROM  employees;
```

```
SELECT employee_id, last_name, salary, 400+salary*12
FROM  employees;
```

```
SELECT employee_id, last_name, salary, (400+salary)*12
FROM employees;
```

空值（NULL）

- NULL：表示未定义的，未知的。空值不等于零或空格。任意类型都可以支持空值。
- 空值（NULL）在算术表达式中的使用
  - 包括空值的任何算术表达式都等于空
  - 包括空值的连接表达式等于与空字符串连接，也就是原来的字符串

```
SELECT  last_name, salary,
    (400+salary)*12+(400+salary) *12*commission_pct
FROM    employees;
```

列别名的使用

- iSQL*PLUS及SQL*PLUS列标题的显示规则：
- iSQL*PLUS：
  - 不管SELECT子句的列或表达式是什么形式，列标题都显示为居中对齐并且大写。
- SQL*PLUS：
  - 根据列数据类型不同而不同。
  - 字符与时间类型：列标题显示为左对齐并且大写。
  - 字符类型：列标题显示为右对齐并且大写。

使用列别名的方法

- 列别名基本书写方法有两种方式:
  - 第一种方式：列名 列别名
  - 第二种方式：列名 AS 列别名
- 以下三种情况，列别名两侧需要添加双引号（""）：
  - 列别名中包含有空格
  - 列别名中要求区分大小写
  - 列别名中包含有特殊字符

使用列别名的方法（续）

- 例**2-11**

```
SELECT employee_id id, last_name as employee_name,
    salary "Salary", (400+salary)*12 "Annual Salary"
FROM  employees;
```

ID	EMPLOYEE_NAME	Salary	Annual Salary
100	King	24000	292800
101	Kochhar	17000	208800
102	De Haan	17000	208800
103	Hunold	9000	112800

...

已选择26行。

连接运算符的使用

- 采用双竖线（||）来做连接运算符，来更清楚地表达实际意思。

```
SELECT  first_name||' '||last_name||"'s phone number is
    '||phone_number "employee's Phone number"
FROM   employees;
```

DISTINCT关键字的用法

- DISTINCT取消重复行

```
SELECT  DISTINCT department_id
FROM  employees;
```

```
SELECT   DISTINCT department_id, job_id
FROM   employees;
```

Oracle9i的三种SQL*PLUS

- Sqlplus
- Sqlplusw
- iSqlplus

SQL语句与SQL*PLUS/iSQL*PLUS命令的区别

- SQL语句与SQL*PLUS/iSQL*PLUS命令有以下主要区别:
  - SQL语句是开发语言，而SQL*PLUS/iSQL*PLUS是Oracle使用的工具。
  - SQL语句直接访问Oracle数据库，并返回结果；而SQL*PLUS/iSQL*PLUS命令是在返回结果上进行处理，如显示格式等。
  - SQL*PLUS/iSQL*PLUS命令只是使每个客户端环境有所不同，不会直接访问数据库。
  - SQL语句不可以缩写，而SQL*PLUS/iSQL*PLUS命令可以缩写。
  - SQL*PLUS/iSQL*PLUS命令结尾可以不加分号（;）。

常用命令

- DESC[RIBE]命令：显示表结构
  - DESC employees
- SET命令：设置环境变量
  - 语法：SET *系统变量 值*
  - *SET ECHO {ON|OFF}*：控制是否把刚执行的语句在屏幕上显示出来。SET ECHO ON其中OFF是默认值。
  - SET HEADING {ON|OFF}：控制是否显示列标题。默认是ON

SET命令示例

```
SET ECHO ON
```

```
SET HEADING OFF
```

```
SELECT *
FROM   departments;
```

```
SET ECHO OFF
```

```
SET HEADING ON
```

列标题的默认显示

- Sql*plus的默认显示
  - Date和character 型数据左对齐
  - Numeric 型数据右对齐
  - 列标题默认显示为大写
- iSql*plus的默认显示
  - 列标题缺省居中对齐
  - 列标题默认显示为大写

本章小结

- 基本的SQL查询语句的构成。
- 课程案例环境的设计。
- 第一条查询语句的书写。
- 如何在查询语句中查找特定的列
- SQL语句的书写规则
- 算术表达式的使用
- 空值（NULL）的应用
- 列别名的使用
- 连接运算符的使用
- DISTINCT关键字的用法
- SQL*PLUS/iSQL*PLUS命令的介绍

练习

- 1.查询员工表中所有员工的信息。
- 2.查询员工表中员工的员工号、姓名、每个员工涨工资100元以后的年工资（按12个月计算）。
- 3.查询员工first_name和last_name，要求结果显示为“姓last_name名first_name”格式。
- 4.查询所有员工所从事的工作有哪些类型（要求去掉重复值）。

第三章　限制数据和对数据排序

Neusoft 东软

带有限制条件的查询

EMP

EMPNO	ENAME	SAL	...	DEPTNO
7839	KING	5100		10
7369	SMITH	800		20
7499	ALLEN	1600		30
7782	CLARK	2450		10
...				

查询**20**部门的所有员工

```
SELECT *
FROM emp
WHERE deptno= 20;
```

EMP

EMPNO	ENAME	SAL	...	DEPTNO
7369	SMITH	800		20
7566	JONES	2975		20
7902	FORD	3000		20

选择表中的部分行

- 使用WHERE子句限定返回的记录
- WHERE子句在 FROM 子句后
- 语法：

```
SELECT    *|{[DISTINCT] 列名|表达式 [别名][,...]}
FROM    表名
[WHERE 条件];
```

- **例3-1 查询公司月薪高于12000的员工信息。**

```
SELECT  employee_id, last_name, salary
FROM     employees
WHERE  salary >= 12000;
```

比较运算符

运算符	含义
=	等于
>	大于
>=	大于等于
<	小于
<=	小于等于
<>	不等于

使用比较运算符

- 使用比较运算符需要遵循以下原则：
  - 字符及日期类型需要在两端用单引号；
  - 字符类型大小写敏感；
  - 日期类型格式敏感，默认格式’DD-MON-RR’;

- **例3-4 查询在1999年1月1日以后进入公司的雇员信息。**

```
SELECT last_name, hire_date
FROM employees
WHERE hire_date >= '01-1月-1999';
```

转换英文环境

- 例3-5 查询在1999年1月1日以后进入公司的雇员信息（英文环境）

```
ALTER SESSION SET NLS_LANGUAGE='AMERICAN';
```

```
SELECT last_name, hire_date
FROM employees
WHERE hire_date >= '01-JAN-1999';
```

特殊比较运算符

运算符	含义
BETWEEN...AND...	确定范围，在两个值之间 (包含比较值)
IN（列表）	确定集合
LIKE	字符串匹配查询
IS NULL	判断空值

BETWEEN…AND…

- 例3-6 查询月薪在4200元到6000公司的雇员。

```
SELECT      employee_id, last_name, salary
FROM        employees
WHERE       salary BETWEEN 4200 AND 6000;
```

EMPLOYEE_ID	LAST_NAME	SALARY
104	Ernst	6000
107	Lorentz	4200
124	Mourgos	5800
200	Whalen	4400
202	Fay	6000

使用IN运算符

- IN运算符主要对指定的值进行比较查看的时候使用。
- **例3-7 查询部门编号为10、90或110的雇员信息。**

```
SELECT employee_id, last_name, salary, department_id
FROM   employees
WHERE  department_id IN (10, 90, 110);
```

EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
205	Higgins	12000	110
206	Gietz	8300	110
100	King	24000	90
101	Kochhar	17000	90
102	De Haan	17000	90
200	Whalen	4400	10

已选择6行。

使用LIKE运算符

- 使用LIKE运算符完成模糊查询功能 。
- 使用通配符来代替未知的信息。常用通配符有 %和_ 。
  - %可以代替任意长度字符（包括长度为0）。
  - _可以代替一个字符。
- **例3-8 查询last_name首字母是S的雇员信息。**

```
SELECT employee_id, last_name, salary
FROM employees
WHERE last_name LIKE 'S%';
```

EMPLOYEE_ID	LAST_NAME	SALARY
111	Sciarra	7700

使用LIKE运算符（续）

- %与_组合使用

- **例3-9 查询last_name第二个字母是b的雇员信息。**

```
SELECT employee_id, last_name, salary
      FROM employees
      WHERE last_name LIKE '_b%';
```

EMPLOYEE_ID	LAST_NAME	SALARY
174	Abel	11000

使用LIKE运算符（续）

- 使用ESCAPE 标识符来查找带特殊符号的字符号。
- **例3-11 查询JOB_ID以“FI_”开头的雇员信息。**

```
SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE job_id LIKE 'FI\_%' ESCAPE '\';
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
109	Faviet	FI_ACCOUNT	9000
110	Chen	FI_ACCOUNT	8200
111	Sciarra	FI_ACCOUNT	7700
112	Urman	FI_ACCOUNT	7800
113	Popp	FI_ACCOUNT	6900
108	Greenberg	FI_MGR	12000

已选择6行。

IS NULL运算符

- 查询包含空值的记录

- **例3-12 未分配部门的雇员信息。**

```
SELECT employee_id, last_name, salary, department_id
FROM employees
WHERE department_id IS NULL;
```

EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
178	Grant	7000	

逻辑运算符

运算符	含义
AND	如果组合的条件都是TRUE,返回TRUE。NULL和FALSE组合，返回FALSE。
OR	如果组合的条件之一是TRUE,返回TRUE。NULL和TRUE组合，返回TRUE。
NOT	如果下面的条件是FALSE,返回TRUE。

AND运算符

- 例**3-13** 查询月薪在**4200**元到**6000**元公司的雇员。

```
SELECT  employee_id, last_name, salary
FROM    employees
WHERE   salary>=4200
AND     salary<=6000;
```

EMPLOYEE_ID	LAST_NAME	SALARY
104	Ernst	6000
107	Lorentz	4200
124	Mourgos	5800
200	Whalen	4400
202	Fay	6000

AND运算符（续）

- 例**3-14** 月薪大于**10000**元，并且在**60**和**90**号部门工作的员工。

```
SELECT   last_name, salary, department_id
FROM     employees
WHERE  salary>10000
AND   department_id in (60,90);
```

LAST_NAME	SALARY	DEPARTMENT_ID
King	24000	90
Kochhar	17000	90
De Haan	17000	90

OR运算符

- 例**3-15** 月薪大于**10000**元，或者在**60**和**90**号部门工作的员工。

```
SELECT  last_name, salary, department_id
FROM          employees
WHERE   salary>10000
OR    department_id in (60,90);
```

LAST_NAME	SALARY	DEPARTMENT_ID
King	24000	90
Kochhar	17000	90
De Haan	17000	90
Hunold	9000	60
Ernst	6000	60
Lorentz	4200	60
Greenberg	12000	100
Zlotkey	10500	80
Abel	11000	80
Hartstein	13000	20
Higgins	12000	110

已选择11行。

使用NOT运算符

- **例3-16 查找职位不是IT_PROG，ST_CLERK,FI_ACCOUNT的员工信息。**

```
SELECT  last_name, job_id, salary
FROM    employees
WHERE   job_id  NOT IN ('IT_PROG', 'ST_CLERK',
    FI_ACCOUNT');
```

使用NOT运算符（续）

- NOT运算符还可以和BETWEEN…AND，LIKE，IS NULL一起使用。
  - ...WHERE department_id NOT IN (60, 90);
  - ... WHERE salary NOT BETWEEN 10000 AND 25000;
  - ... WHERE last_name NOT LIKE 'D%'
  - ... WHERE manager_id IS NOT NULL

运算符优先级

- 括号'()'优先于其他操作符。

优先级	运算分类	运算符举例
1	数学运算符	*, \, +, -
2	连接运算符	||
3	通用比较运算符	=, <>, <, >, <=, >=
4	其他比较运算符	IS [NOT] NULL, LIKE, [NOT] BETWEEN, [NOT] IN
5	逻辑非	NOT
6	逻辑与	AND
7	逻辑或	OR

运算符的优先级（续）

- **例3-17 查找职位是FI_ACCOUNT或工资超过16000的职位是AD_VP的员工。**

```
SELECT    last_name, job_id, salary, department_id
FROM      employees
WHERE  job_id = 'FI_ACCOUNT‘
OR          job_id = 'AD_VP‘  AND    salary > 16000;
```

LAST_NAME	JOB_ID	SALARY	DEPARTMENT_ID
Kochhar	AD_VP	17000	90
De Haan	AD_VP	17000	90
Faviet	FI_ACCOUNT	9000	100
Chen	FI_ACCOUNT	8200	100
Sciarra	FI_ACCOUNT	7700	100
Urman	FI_ACCOUNT	7800	100
Popp	FI_ACCOUNT	6900	100

已选择7行。

运算符的优先级（续）

- **例3-18查找工作超过16000并且职位是FI_ACCOUNT或是AD_VP的员工。**

```
SELECT      last_name, job_id, salary, department_id
FROM        employees
WHERE       (job_id = 'FI_ACCOUNT'
OR          job_id = 'AD_VP')
AND    salary > 16000;
```

LAST_NAME	JOB_ID	SALARY	DEPARTMENT_ID
Kochhar	AD_VP	17000	90
De Haan	AD_VP	17000	90

ORDER BY的使用

- ORDER BY子句后的语法结构如下：

```
SELECT  *|{[DISTINCT] 列名|表达式 [别名][,...]}
FROM    表名
[WHERE  条件]
[ORDER BY  {列名|表达式|别名} [ASC|DESC],...];
```

- **例3-19 查看公司员工信息，按照员工部门降序排列。**

```
SELECT  last_name, job_id, salary, department_id
FROM    employees
ORDER BY  department_id DESC;
```

不同数据类型排序规则(以升序为例)

- 数字升序排列小值在前，大值在后。即按照数字大小顺序由小到大排列。
- 日期升序排列相对较早的日期在前，较晚的日期在后。例如：’01-SEP-06’在’01-SEP-07’前。
- 字符升序排列按照字母由小到大的顺序排列。即由A-Z排列；中文升序按照字典顺序排列。
- 空值在升序排列中排在最后，在降序排列中排在最开始。

ORDER BY排序

- 使用列别名排序，多列排序。

例3-20 查看员工信息，结果按照年薪升序排列。

```
SELECT last_name, job_id, salary*12 annual, department_id
FROM    employees
ORDER BY annual;
```

例3-21 查看员工信息，结果按照按照job_id升序排列，月薪按照降序排列。

```
SELECT last_name, job_id, salary, department_id
FROM employees
ORDER BY job_id, salary desc;
```

ORDER BY特殊使用

- ORDER BY子句可以出现在SELECT子句中没有出现过的列。
- ORDER BY子句后的列名，可以用数字来代替。这个数字是SELECT语句后列的顺序号。

- **例3-22 查看公司员工信息，按照月薪由高到低排列，而具体的工资数不显示。**

```
SELECT last_name, job_id, hire_date
FROM employees
ORDER BY salary;
```

ORDER BY特殊使用（续）

- **例3-23 查看员工信息，结果按照按照job_id升序排列，月薪按照降序排列。**

```
SELECT last_name, job_id, salary, department_id
FROM employees
ORDER BY 2, 3 desc;
```

本章小结

- 选择表中的部分行
- 比较运算符的使用
- 特殊比较运算符的使用
- 逻辑运算符的使用
- ORDER BY的使用

练习

- 1.查询last_name是Chen的员工的信息。
- 2.查询参加工作时间在1997-7-9之后，并且不从事IT_PROG工作的员工的信息。
- 3.查询员工last_name的第三个字母是a的员工的信息。
- 4.查询除了10、20、110号部门以外的员工的信息。
- 5.查询部门号为50号员工的信息，先按工资降序排序，再按姓名升序排序。
- 6.查询没有上级管理的员工(经理号为空)的信息。
- 7.查询员工表中工资大于等于4500并且部门为50或者60的员工的姓名(last_name), 工资，部门号。

第四章　单行函数

Neusoft 东软

本章要点

- 字符函数
- 数字函数
- 日期函数
- 转换函数
- 通用函数

单行函数介绍

- 语法:

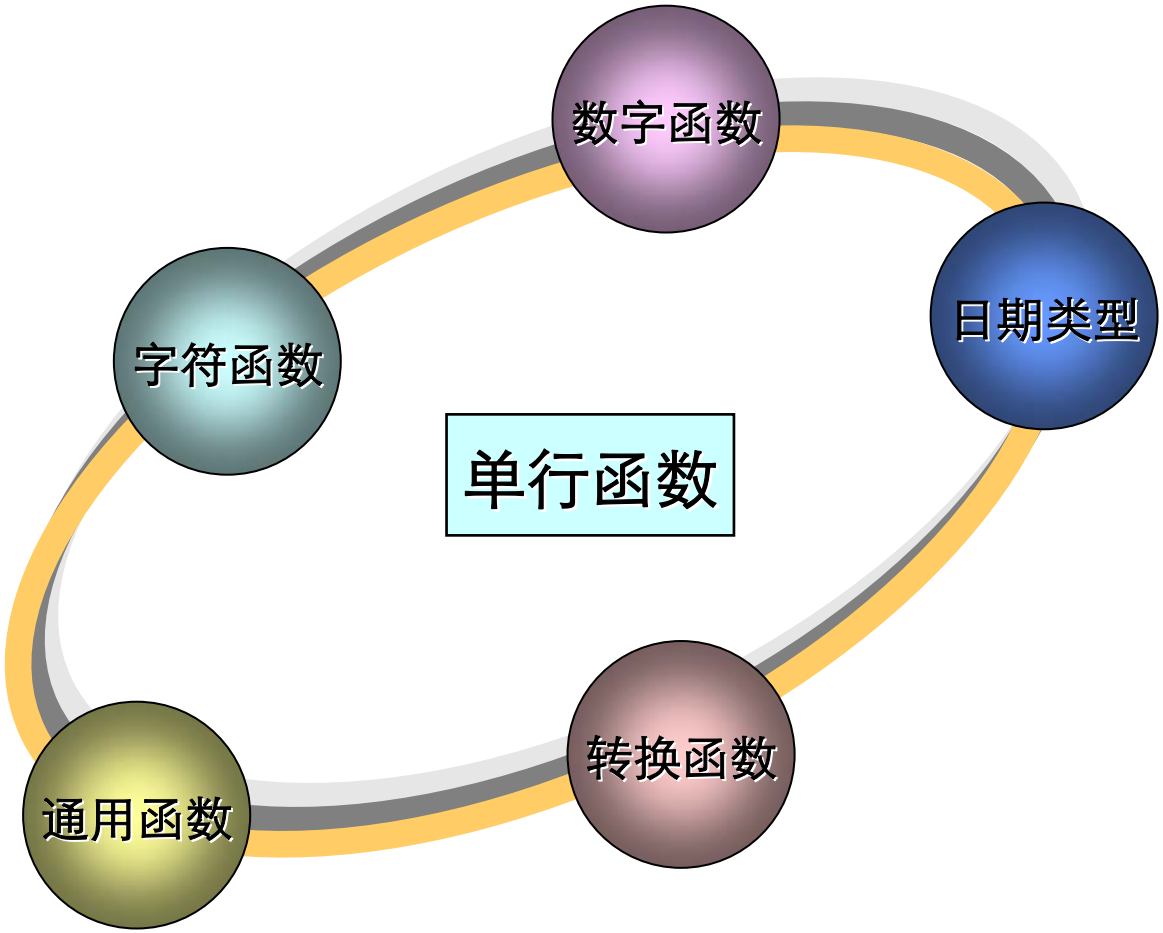
  函数名[(参数1，参数2,…)]

- 其中的参数可以是以下之一:
  - 用户定义的变量
  - 变量
  - 列名
  - 表达式

单行函数介绍（续）

- 单行函数还有以下的一些特征：
  - 单行函数对单行操作
  - 每行返回一个结果
  - 有可能返回值与原参数数据类型不一致（转换函数）
  - 单行函数可以写在SELECT、WHERE、ORDER BY子句中
  - 有些函数没有参数，有些函数包括一个或多个参数
  - 函数可以嵌套

单行函数的分类



字符函数

- 字符函数：主要指参数类型是字符型，不同函数返回值可能是字符型或数字类型。
  - LOWER
  - UPPER
  - INITCAP
  - CANCAT
  - SUBSTR
  - LENGTH
  - INSTR
  - TRIM
  - REPLACE

大小写转换函数

- LOWER(*列名|表达式*)：全小写
- UPPER(*列名|表达式*)：全大写
- INITCAP(*列名|表达式*)：首字母大写

其他字符函数

- CONCAT(*列1|表达式1,列2|表达式2*)：字符串连接
- SUBSTR(*列名|表达式,m[,n]*)：字符串截取
- LENGTH(*列名|表达式*)：返回字符串长度
- INSTR(*列名|表达式,'string', [,m], [n]*)：返回一个字符串在另一个字符串中的位置。
- LPAD(*列名|表达式,n,'string'*)：左填充
- RPAD(*列名|表达式, n,'string'*)：右填充
- TRIM(*[leading|trailing|both,]trim_character FROM trim_source*)：去掉左右两边指定字符。
- REPLACE (*文本, 查找字符串, 替换字符串*)：替换字符串

字符函数

- 例**4-15** 查找公司员工编号，用户名（**first_name**与**last_name**连接成一个字符串），职位编号及**last_name**的长度，要求职位从第四位起匹配**'ACCOUNT'**，同时**last_name**中至少包含一个**’e’**字母。

```
SELECT employee_id, CONCAT(first_name, last_name) NAME,
    job_id, LENGTH (last_name) length
FROM    employees
WHERE  SUBSTR(job_id, 4) = 'ACCOUNT'
AND    INSTR(last_name, 'e')>0;
```

数字函数

- ROUND(*列名|表达式, n*)：四舍五入函数。
- TRUNC(*列名|表达式,n*)：截断函数。
- MOD(*m,n*)：取余函数。

数字函数示例

```
SELECT ROUND(65.654,2),ROUND(65.654,0),
ROUND(65.654,-1)
FROM  DUAL;
```

```
SELECT TRUNC(65.654,2),TRUNC(65.654,0),
TRUNC(65.654,-1)
FROM  DUAL;
```

```
SELECT employee_id, last_name, salary,   MOD(salary,900)
FROM employees
WHERE department_id=90;
```

日期函数

- 常用的日期运算如下:
  - 日期类型列或表达式可以加减数字，功能是在该日期上加减对应的天数。如：’10-MON-06’+15结果是’25-MON-06’。
  - 日期类型列或表达式之间可以进行减操作，功能是计算两个日期之间间隔了多少天。如：’10-MON-06’-‘4-MON-06’结果四舍五入后是6天。
  - 如果需要加减相应小时或分钟，可以使用n/24来实现。

常用日期函数

- SYSDATE：返回系统日期
- MONTHS_BETWEEN：返回两个日期间隔的月数
- ADD_MONTHS：在指定日期基础上加上相应的月数
- NEXT_DAY：返回某一日期的下一个指定日期
- LAST_DAY：返回指定日期当月最后一天的日期
- ROUND(date[,'fmt'])对日期进行指定格式的四舍五入操作。按照YEAR、MONTH、DAY等进行四舍五入。
- TRUNC(*date[,'fmt']*)对日期进行指定格式的截断操作。按照YEAR、MONTH、DAY等进行截断。
- EXTRACT：返回从日期类型中取出指定年、月、日

日期函数

- 例4-20 MONTHS_BETWEEN函数演示——公司员工服务的月数。

```
SELECT  last_name, salary,
    MONTHS_BETWEEN(SYSDATE,hire_date) months
FROM   employees
ORDER BY months;
```

- 例4-21 ADD_MONTHS函数演示——99年公司员工转正日期。

```
SELECT  last_name, salary, hire_date,
    ADD_MONTHS(hire_date,3) new_date
FROM    employees
WHERE  hire_date>'01-1月-1999';
```

日期函数（续）

- 例4-22 NEXT_DAY函数演示——下周一的日期。

```
SELECT  NEXT_DAY('02-2月-06','星期一') NEXT_DAY
FROM   DUAL;
```

- 例4-23 LAST_DAY函数演示——06年2月2日所在月份最后一天。

```
ELECT LAST_DAY('02-2月-2006') "LAST DAY"
FROM   DUAL;
```

日期函数（续）

- 例**4-24 ROUND**函数演示——**98**年入职员工入职日期按月四舍五入。

```
SELECT      employee_id, hire_date, ROUND(hire_date,
    'MONTH')
FROM        employees
WHERE       SUBSTR(hire_date,-2,2)='98';
```

LAST_NAME	HIRE_DATE	ROUND(HIRE
Urman	07-3月 -98	01-3月 -98
Matos	15-3月 -98	01-3月 -98
Vargas	09-7月 -98	01-7月 -98
Taylor	24-3月 -98	01-4月 -98

日期函数（续）

- 例4-25 TRUNC函数演示——98年入职员工入职日期按月截断。

```
SELECT      employee_id, hire_date, TRUNC(hire_date,
    'MONTH')
FROM        employees
WHERE       SUBSTR(hire_date,-2,2)='98';
```

EMPLOYEE_ID	HIRE_DATE	TRUNC(HIRE
112	07-3月 -98	01-3月 -98
143	15-3月 -98	01-3月 -98
144	09-7月 -98	01-7月 -98
176	24-3月 -98	01-3月 -98

EXTRACT函数

- EXTRACT函数语法

```
EXTRACT ([YEAR] [MONTH][DAY]
FROM     [日期类型表达式])
```

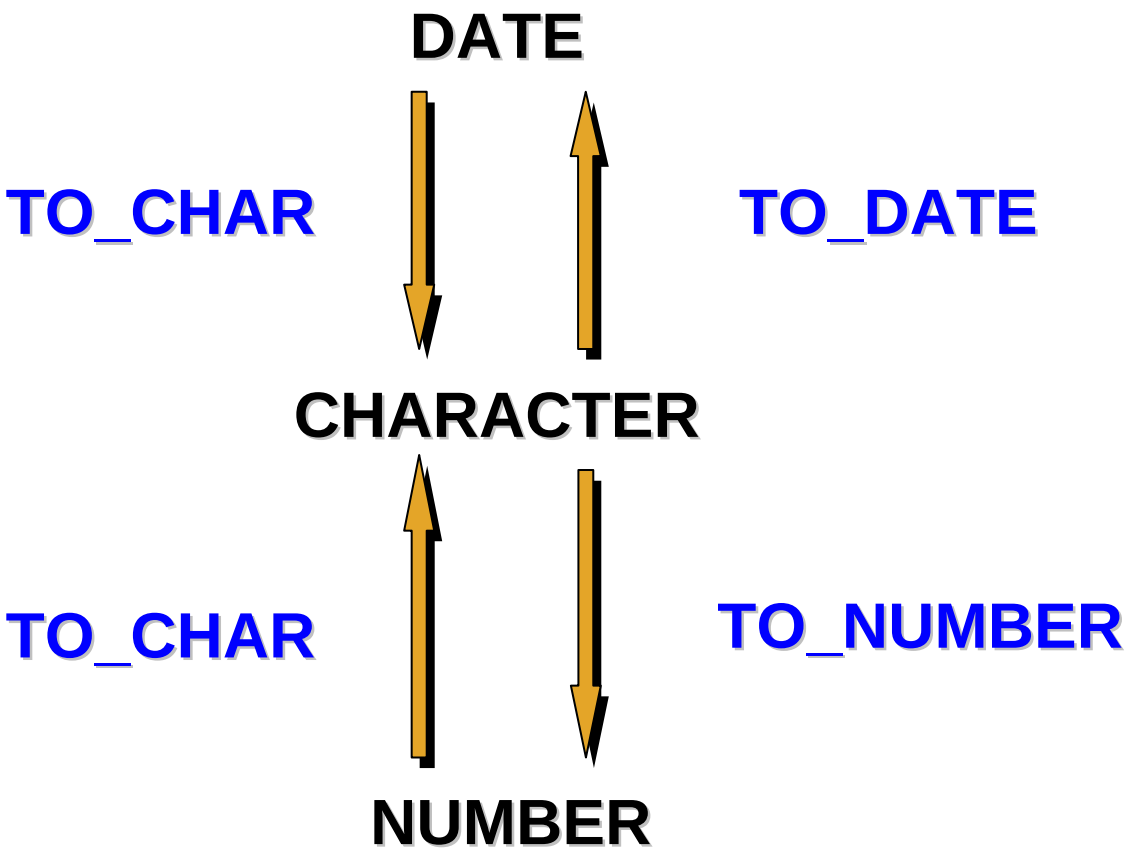
- **例4-26** 部门编号是90的部门中所有员工入职月份。

```
SELECT  last_name, hire_date, EXTRACT (MONTH FROM
    HIRE_DATE) MONTH
FROM   employees
WHERE  department_id=90;
```

数据类型显性转换

- 通常是在字符类型，日期类型，数字类型之间进行显性转换。主要有3个显性函数:
  - TO_CHAR
  - TO_NUMBER
  - TO_DATE

数据类型显性转换



TO_CHAR 函数

- TO_CHAR(date|number [,‘fmt’])把日期类型/数字类型的表达式或列转换为变长类型字符类型。
  - ‘fmt’指的是需要显示的格式:
  - 需要写在单引号中，并且是大小写敏感
  - 可包含任何有效的日期格式
  - fm表示去除生成结果的前导零或空格

常用日期格式

- YYYY：4位数字表示年份；
- YY：2位数字表示年份，但是无世纪转换(与RR区别在后面章节介绍)；
- RR：2位数字表示年份，有世纪转换（与YY区别在后面章节介绍）；
- YEAR：年份的英文拼写；
- MM：两位数字表示月份；
- MONTH：月份英文拼写；

常用日期格式（续）

- DY：星期的英文前三位字母；
- DAY：星期的英文拼写；
- D：数字表示一星期的第几天，星期天是一周的第一天。
- DD：数字表示一个月中的第几天；
- DDD：数字表示一年中的第几天。

常用时间格式

- AM 或PM：上下午表示；
- HH 或HH12或HH24：数字表示小时。HH12代表12小时计时，HH24代表24小时计时；
- MI：数字表示分钟；
- SS：数字表示秒；

一些特殊格式

- TH：显示数字表示的英文序数词，如：DDTH显示天数的序数词。
- SP：显示数字表示的拼写。
- SPTH：显示数字表示的序数词的拼写。

TO_CHAR(SYSDATE,'DDSPTH') → THIRTIETH

- “字符串”:如在格式中显示字符串，需要两端加双引号。

TO_CHAR(SYSDATE,'DD “of” MONTH ') → 18 of 10月

日期到字符型转换

- 例**4-29 TO_CHAR**函数进行日期到字符型复杂格式转换演示。

```
ALTER SESSION SET NLS_LANGUAGE = AMERICAN;
```

```
SELECT employee_id, last_name,
TO_CHAR(hire_date,'Day ",the" Ddspth "of" YYYY HH24:MI:SS')
    hire_date
FROM  employees
WHERE  department_id=90;
```

数字到字符型转换

- 具体格式如下：
  - 9：一位数字；
  - 0：一位数字或前导零；
  - $：显示为美元符号；
  - L：显示按照区域设置的本地货币符号；
  - .：小数点；
  - ,：千位分割符；

数字到字符型转换（续）

- 例4-30 TO_CHAR函数进行数字到字符型格式转换

```
SELECT  last_name, TO_CHAR(salary, '$99,999.00') salary
FROM   employees
WHERE  last_name = 'King';
```

- 注意：进行数字类型到字符型转换，格式中的宽度一定要超过实际列宽度，否则会显示为###。

```
SELECT          last_name, TO_CHAR(salary, '$9,999.00') salary
FROM   employees
WHERE           last_name = 'King';
```

TO_NUMBER和TO_DATE函数

- TO_NUMBER(*char[,'fmt']*) 把字符类型列或表达式转换为数字类型。
  - 使用格式和TO_CHAR中转换成字符类型中的格式相同
  - 在转换时让Oracle知道字符串中每部分的功能
- TO_DATE(*char[,'fmt']*) 把字符类型列或表达式转换为日期类型。
  - 格式和TO_CHAR中转换成字符类型中的格式相同。

RR的功能

- 当前年份0-49，指定日期是0-49时，返回当前世纪。
- 当前年份0-49，指定日期是50-99时，返回上个世纪。
- 当前年份50-99，指定日期是0-49时，返回下个世纪。
- 当前年份50-99，指定日期是50-99时，返回当前世纪。

TO_DATE字符到日期型格式转换

- 假设当前日期是**2006**年。

```
SELECT
TO_CHAR(TO_DATE('12-2月-96','DD-MON-YY'),'DD-MON-YYYY')  YY96,
TO_CHAR(TO_DATE('12-2月-96','DD-MON-RR'),'DD-MON-YYYY') RR96,
TO_CHAR(TO_DATE('12-2月-08','DD-MON-YY'),'DD-MON-YYYY') YY08,
TO_CHAR(TO_DATE('12-2月-08','DD-MON-RR'),'DD-MON-YYYY') RR08
FROM   DUAL;
```

YY96	RR96	YY08	RR08
12-2月 -2096	12-2月 -1996	12-2月 -2008	12-2月 -2008

其他函数

- 与空值（NULL）相关的一些函数，完成对空值（NULL）的一些操作。主要包括以下函数:
  - NVL
  - NVL2
  - NULLIF
  - COALESCE
- 条件处理函数:
  - CASE表达式
  - DECODE

NVL 函数

- NVL (表达式1, 表达式2)函数功能是空值转换，把空值转换为其他值，解决空值问题。表达式1是需要转换的列或表达式，表达式2是如果第一个参数为空时，需要转换的值。
  - NVL(comm,0)
  - NVL(hire_date,'01-JAN-06')
  - NVL(job_id,'No Job Yet')
- 注意：数据格式可以是日期,字符,数字。但数据类型必须匹配

NVL2函数

- NVL2(*表达式1, 表达式2, 表达式3*)函数是对第一个参数进行检查。如果第一个参数不为空，则输出第二个参数；如果第一个参数为空，则输出第三个参数。表达式1可以为任何数据类型

```
SELECT last_name, salary,
 NVL2(commission_pct, salary +commission_pct,salary) income
FROM   employees
WHERE last_name LIKE '_a%';
```

NULLIF

- NULLIF (*表达式1*, *表达式2*)函数主要是完成两个参数的比较。当两个参数不相等时，返回值是第一个参数值；当两个参数相等时，返回值是空值。

```
SELECT last_name, LENGTH(last_name) LEN_last_NAME,
    email, LENGTH(email)  LEN_EMAIL,
   NULLIF(LENGTH(last_name), LENGTH(email)) result
FROM   employees
WHERE last_name LIKE 'D%';
```

COALESCE

- COALESCE（*表达式1，表达式2，... 表达式n*)函数是对NVL函数的扩展。COALESCE函数的功能是返回第一个不为空的参数，参数个数不受限制。

```
SELECT  last_name,
    COALESCE(commission_pct, salary*1.1, 100)
     comm,department_id
FROM   employees
WHERE  department_id in (50,80)
ORDER BY COMM;
```

CASE表达式

- 语法

```
CASE expr
        WHEN comparison_expr1 THEN return_expr1
        [WHEN comparison_expr2 THEN return_expr2
         WHEN comparison_exprn THEN return_exprn
         ELSE else_expr]
END
```

CASE表达式

- 例子

```
SELECT last_name, commission_pct,
       (CASE commission_pct
          WHEN 0.1  THEN '低’
          WHEN 0.2  THEN '中'
          WHEN 0.3  THEN '高'
          ELSE ‘无'
        END) Commission
     FROM employees
WHERE commission_pct IS NOT NULL
ORDER BY last_name;
```

DECODE

- DECODE(字段|*表达式, 条件1,结果1[,条件2,结果2…,][,缺省值]*)

```
SELECT last_name, commission_pct,
  decode( commission_pct,
            0.1,'低',
            0.2,'中',
            0.3 , '高',
            '无') Commission
FROM employees
WHERE commission_pct IS NOT NULL
ORDER BY last_name;
```

本章小结

- 单行函数介绍
- 字符函数的使用
- 数字函数的使用
- 日期函数的使用
- 转换函数的使用
- 其他函数的使用

练习

- 1.计算2000年1月1日到现在有多少月，多少周（四舍五入）。
- 2.查询员工last_name的第三个字母是a的员工的信息(使用2个函数)。
- 3.使用trim函数将字符串‘hello’、‘ Hello ’、‘bllb’、‘ hello ’分别处理得到下列字符串ello、Hello、ll、hello。
- 4.将员工工资按如下格式显示：123,234.00 RMB
- 5.查询员工的last_name及其经理（manager_id），要求对于没有经理的显示“No Manager”字符串。

练习

- 6.将员工的参加工作日期按如下格式显示：月份/年份。
- 7.在employees表中查询出员工的工资，并计算应交税款：如果工资小于1000,税率为0，如果工资大于等于1000并小于2000，税率为10%，如果工资大于等于2000并小于3000，税率为15%，如果工资大于等于3000，税率为20%。

第五章　多表查询

Neusoft 东软

本章要点

- 多表连接的定义
- 笛卡尔积
- 等价连接
- 不等价连接
- 外连接
- ANSI SQL：1999标准的连接语句

笛卡尔积

- 笛卡尔积是把表中所有的记录作乘积操作，生成大量的结果，而通常结果中可用的值有限。笛卡尔积出现的原因多种多样，通常是由于连接条件缺失造成的。

```
SELECT last_name, job_id, department_name
FROM  employees, departments;
```

LAST_NAME	JOB_ID	DEPARTMENT_NAME
King	AD_PRES	Administration
Kochhar	AD_VP	Administration
De Haan	AD_VP	Administration
Hunold	IT_PROG	Administration
...		
Hartstein	MK_MAN	Operations
Fay	MK_REP	Operations
Higgins	AC_MGR	Operations
Gietz	AC_ACCOUNT	Operations

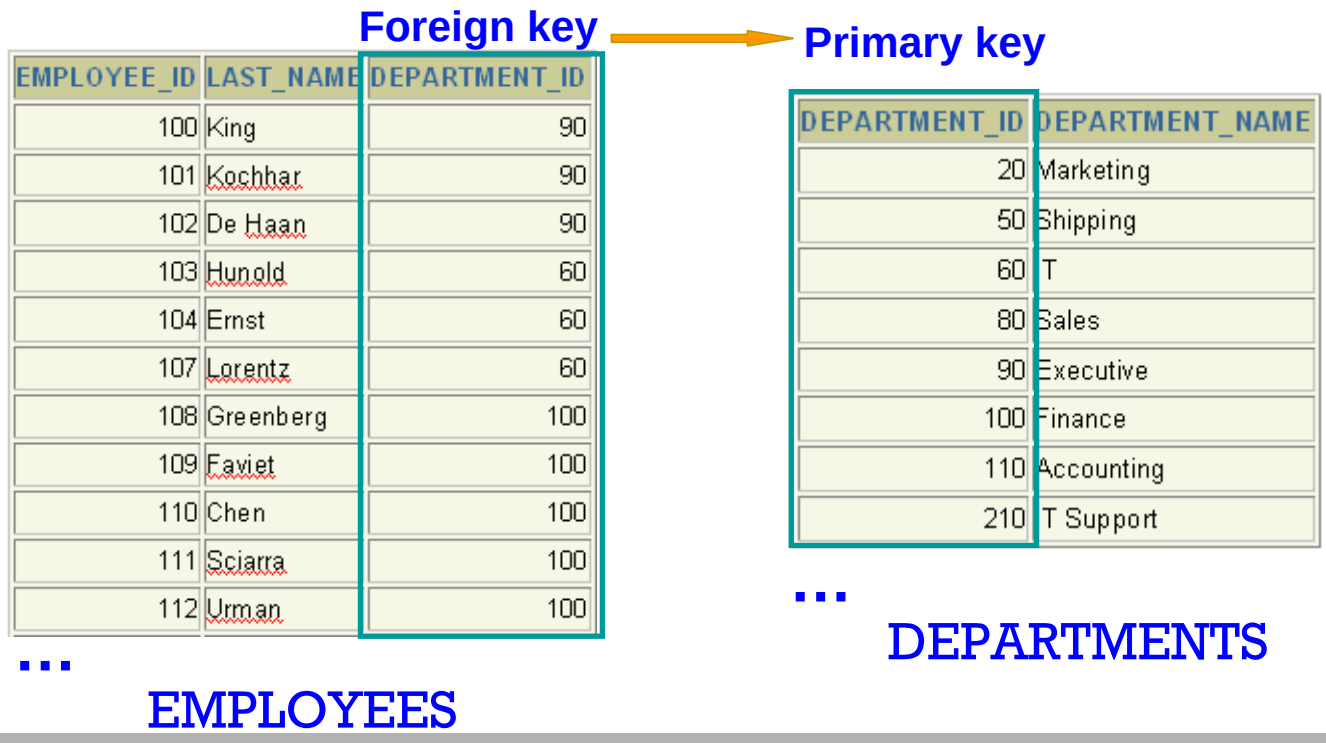
已选择234行。

多表连接主要分为

- 等价连接
- 不等价连接
- 外连接
- 自连接

等价连接

- 等价连接又称简单连接或内连接。就是当两个表的公共字段相等的时候把两个表连接在一起。公共字段是两个表中有相同含义的列。



等价连接

- 等价连接的语法结构

```
SELECT      table1.column, table2.column
FROM  table1, table2
WHERE       table1.column1 = table2.column2;
```

  - 在 WHERE 子句中写连接条件
  - 当多个表中有重名列时，必须在列的名字前加上表名作为前缀

等价连接的示例

```
SELECT  employees.last_name, employees.job_id,
employees.department_id, departments.department_name
FROM   employees, departments
WHERE  employees.department_id = departments.department_id;
```

LAST_NAME	JOB_ID	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	AD_ASST	10	Administration
Hartstein	MK_MAN	20	Marketing
Fay	MK_REP	20	Marketing
Mourgos	ST_MAN	50	Shipping
...			
Urman	FI_ACCOUNT	100	Finance
Popp	FI_ACCOUNT	100	Finance
Higgins	AC_MGR	110	Accounting
Gietz	AC_ACCOUNT	110	Accounting

已选择25行。

等价连接中的记录筛选

- 多表连接中，记录筛选语句同样写在WHERE语句中，用逻辑AND和连接判断语句写在一起。

```
SELECT  employees.last_name, employees.job_id,
employees.department_id,departments.department_name
FROM   employees, departments
WHERE  employees.department_id=departments.department_id
AND     job_id LIKE '%MAN%';
```

LAST_NAME	JOB_ID	DEPARTMENT_ID	DEPARTMENT_NAME
Hartstein	MK_MAN	20	Marketing
Mourgos	ST_MAN	50	Shipping
Zlotkey	SA_MAN	80	Sales

表别名的书写

- 等价连接表别名示例

```
SELECT  e.last_name, e.job_id, e.department_id,
    d.department_name
FROM   employees e, departments d
WHERE  e.department_id = d.department_id
AND  job_id LIKE '%MAN%';
```

LAST_NAME	JOB_ID	DEPARTMENT_ID	DEPARTMENT_NAME
Hartstein	MK_MAN	20	Marketing
Mourgos	ST_MAN	50	Shipping
Zlotkey	SA_MAN	80	Sales

表别名的书写（续）

- 关于表别名需要注意以下几点：
  - 表别名长度不超过30个字符;
  - 表别名定义在FROM子句中;
  - 如果已经定义了表别名，那么只能使用表别名而不能使用原表名;
  - 表别名的有效范围只是当前语句。
- SQL语句的书写顺序是：
  - SELECT FROM WHERE ORDER BY
- 而实际的执行顺序是：
  - FROM WHERE SELECT ORDER BY

两个以上表连接

- 查找特定城市员工信息。

```
SELECT  e.last_name, e.job_id, e.department_id,
    d.department_name,l.city
FROM   employees e, departments d, locations l
WHERE   e.department_id = d.department_id
AND   d.location_id = l.location_id
AND   l.city IN ('Southlake','Oxford');
```

LAST_NAME	JOB_ID	DEPARTMENT_ID	DEPARTMENT_NAME	CITY
Zlotkey	SA_MAN	80	Sales	Oxford
Abel	SA_REP	80	Sales	Oxford
Taylor	SA_REP	80	Sales	Oxford
Hunold	IT_PROG	60	IT	Southlake
Ernst	IT_PROG	60	IT	Southlake
Lorentz	IT_PROG	60	IT	Southlake

已选择6行。

不等价连接

- 除了等号之外，在表连接语句中还可以使用其它的运算符。这种使用除等号之外运算符的连接语句被称为不等价连接。
- 使用不等价连接查询可以查询两个表中具有非等值关系的数据。操作符可以是比较运算符，也可以是between…and 或者是in、like。

不等价连接演示

```
DESC salgrades;
SELECT *
FROM salgrades;
```

名称	是否为空?	类型
GRADE_LEVEL		VARCHAR2(4)
LOWEST_SALARY		NUMBER
HIGHEST_SALARY		NUMBER

GRAD	LOWEST_SALARY	HIGHEST_SALARY
L1	1000	2999
L2	3000	5999
L3	6000	8999
L4	9000	14999
L5	15000	22999
L6	23000	30000

已选择6行。

不等价连接示例

```
SELECT    e.last_name, e.job_id, e.salary, s.grade_level
FROM      employees e, salgrades s
WHERE     e.salary BETWEEN s.lowest_salary AND
      s.highest_salary  AND e.job_id in('IT_PROG','SA_REP')
ORDER BY s.grade_level;
```

LAST_NAME	JOB_ID	SALARY	GRAD
Lorentz	IT_PROG	4200	L2
Taylor	SA_REP	8600	L3
Grant	SA_REP	7000	L3
Ernst	IT_PROG	6000	L3
Abel	SA_REP	11000	L4
Hunold	IT_PROG	9000	L4

已选择6行。

外连接

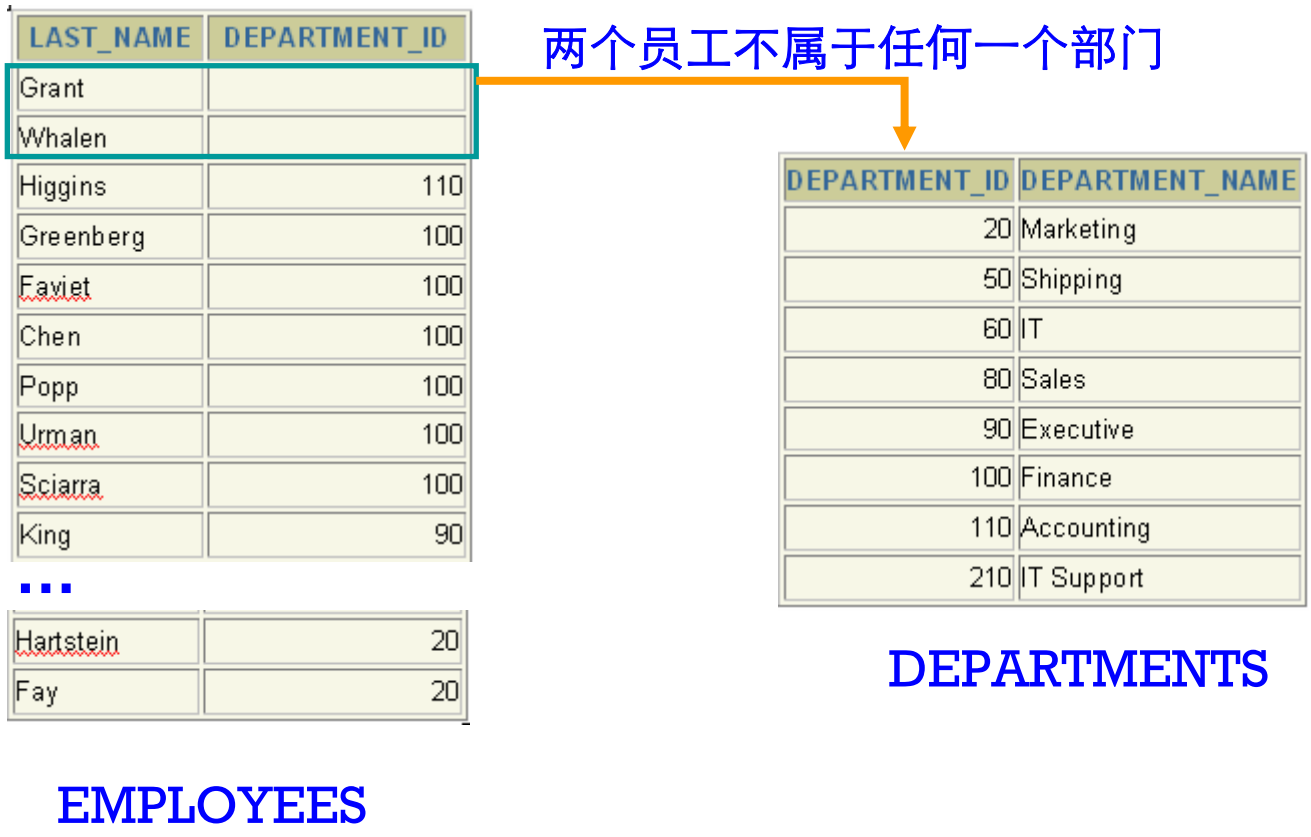
- 为了查找到所有记录，包括没有匹配的记录，需要用外连接语句来实现。
- 右外连接：

```
SELECT      table1.column, table2.column
FROM  table1, table2
WHERE       table1.column(+) = table2.column;
```

- 左外连接，如果右边的列所在表是缺乏表：

```
SELECT      table1.column, table2.column
FROM  table1, table2
WHERE       table1.column = table2.column(+);
```

外连接（续）



外连接示例

- 例**5-8** 所有部门信息，不管部门是否有员工。

```
SELECT  e.last_name, e.job_id, e.department_id,
    d.department_name
FROM   employees e, departments d
WHERE  e.department_id(+) = d.department_id;
```

LAST_NAME	JOB_ID	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	AD_ASST	10	Administration
Hartstein	MK_MAN	20	Marketing
Fay	MK_REP	20	Marketing
Mourgos	ST_MAN	50	Shipping
...			
Urman	FI_ACCOUNT	100	Finance
Popp	FI_ACCOUNT	100	Finance
Higgins	AC_MGR	110	Accounting
Gietz	AC_ACCOUNT	110	Accounting
			Operations

已选择26行。

外连接示例（续）

- 例**5-9** 所有员工信息，不管员工是否有部门。

```
SELECT     e.last_name, e.job_id, e.department_id,
   d.department_name
FROM       employees e, departments d
WHERE  e.department_id= d.department_id(+);
```

LAST_NAME	JOB_ID	DEPARTMENT_ID	DEPARTMENT_NAME
King	AD_PRES	90	Executive
Kochhar	AD_VP	90	Executive
De Haan	AD_VP	90	Executive
Hunold	IT_PROG	60	IT
...			
Grant	SA_REP		
Whalen	AD_ASST	10	Administration
Hartstein	MK_MAN	20	Marketing
Fay	MK_REP	20	Marketing
Higgins	AC_MGR	110	Accounting
Gietz	AC_ACCOUNT	110	Accounting

已选择26行。

ANSI SQL：1999标准的连接语法

- Oracle9i除了Oracle自己的连接语法外，同时支持美国国家标准协会（ANSI）的SQL：1999标准的连接语法。

```
SELECT          table1.column, table2.column
FROM  table1
[CROSS JOIN table2] |
[NATURAL JOIN table2] |
[JOIN table2 USING (column_name)] |
[JOIN table2
  ON(table1.column_name = table2.column_name)] |
[LEFT|RIGHT|FULL OUTER JOIN table2
  ON (table1.column_name = table2.column_name)];
```

ANSI SQL：1999的连接语法

- CROSS JOIN：交叉连接，生成笛卡尔积；
- NATURAL JOIN：自然连接；
- USING (*column_name*)：USING子句，通过名字来具体指定连接
- JOIN *table2*

  ON (*table1.column_name = table2.column_name*)：等价连接语句；
- [LEFT|RIGHT|FULL OUTER JOIN：左外连接|右外连接|全外连接。

交叉连接

- 交叉连接子句（CROSS JOIN）是在SQL99标准中，为了生成笛卡尔积而设计的。

```
SELECT     last_name, job_id, department_name
FROM       employees
CROSS JOIN departments;
```

LAST_NAME	JOB_ID	DEPARTMENT_NAME
King	AD_PRES	Administration
Kochhar	AD_VP	Administration
De Haan	AD_VP	Administration
Hunold	IT_PROG	Administration
...		
Hartstein	MK_MAN	Operations
Fay	MK_REP	Operations
Higgins	AC_MGR	Operations
Gietz	AC_ACCOUNT	Operations

已选择234行。

自然连接

- NATURAL JOIN是SQL99中新增语句，连接条件是两个表中所有的值和数据类型都相同的同名列。如果仅列名相同而数据类型不同，则报错。

```
SELECT        department_id, department_name, city
FROM  departments
NATURAL JOIN locations;
```

DEPARTMENT_ID	DEPARTMENT_NAME	CITY
60	IT	Southlake
50	Shipping	South San Francisco
10	Administration	Seattle
90	Executive	Seattle
100	Finance	Seattle
110	Accounting	Seattle
200	Operations	Seattle
20	Marketing	Toronto
80	Sales	Oxford

已选择9行。

USING子句

- USING (*column_name*)子句也是SQL99新增子句，可以较灵活的完成在多表连接，多列列名相同时，使用其中的一列同名列连接，而不需写连接条件的功能。
- USING子句和NATURAL JOIN不能在一条语句中同时书写。

USING子句示例

```
SELECT      last_name, job_id, department_name
FROM   employees
JOIN     departments
USING(department_id);
```

LAST_NAME	JOB_ID	DEPARTMENT_NAME
Whalen	AD_ASST	Administration
Hartstein	MK_MAN	Marketing
Fay	MK_REP	Marketing
Mourgos	ST_MAN	Shipping
...		
Sciarra	FI_ACCOUNT	Finance
Urman	FI_ACCOUNT	Finance
Popp	FI_ACCOUNT	Finance
Higgins	AC_MGR	Accounting
Gietz	AC_ACCOUNT	Accounting

已选择25行。

ON子句

- 例

```
SELECT e.last_name, e.job_id,
    e.department_id,d.department_name
FROM  employees e
JOIN  departments d
ON （e.department_id = d.department_id）;
```

LAST_NAME	JOB_ID	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	AD_ASST	10	Administration
Hartstein	MK_MAN	20	Marketing
Fay	MK_REP	20	Marketing
Mourgos	ST_MAN	50	Shipping

...

Urman	FI_ACCOUNT	100	Finance
Popp	FI_ACCOUNT	100	Finance
Higgins	AC_MGR	110	Accounting
Gietz	AC_ACCOUNT	110	Accounting

已选择25行。

SQL99实现两表以上连接

- 例**5-14** 查找特定城市员工信息。

```
SELECT e.last_name, e.job_id, e.department_id,
    d.department_name,l.city
FROM   employees e
JOIN     departments d
ON        e.department_id = d.department_id
JOIN     locations l
ON        d.location_id = l.location_id
WHERE  l.city IN ('Southlake','Oxford');
```

SQL99:左外连接

- 在LEFT OUTER JOIN中，会返回所有左边表中的行，即使在右边的表中没有可对应的列值。

- **例5-15 所有员工信息，不管员工是否有部门。**

```
SELECT  e.last_name, e.job_id, e.department_id,
    d.department_name
FROM employees e
LEFT OUTER JOIN  departments d
ON         e.department_id= d.department_id;
```

SQL99:右外连接

- RIGHT OUTER JOIN

- 例**5-16** 所有部门信息，不管部门是否有员工。

```
SELECT  e.last_name, e.job_id, e.department_id,
    d.department_name
FROM employees e
RIGHT OUTER JOIN  departments d
ON        e.department_id= d.department_id;
```

SQL99:全外连接

- 全外连接（FULL OUTER JOIN）主要功能是返回两表连接中等价连接结果，及两个表中所有等价连接失败的记录。

```
SELECT  e.last_name, e.job_id, e.department_id,
    d.department_name
FROM   employees e
FULL OUTER JOIN  departments d
ON        e.department_id= d.department_id;
```

本章小结

- 本章讲述了什么是多表连接，以及多表连接的几种类型，通过例子说明了以前的多表连接语法和新的Sql99标准中的连接语法的差异。
- 笛卡尔积
- 等价连接
- 不等价连接
- 外连接
- ANSI SQL：1999标准的连接语句

练习

- 1.查询员工的编号，姓名，以及部门名称(分别使用Oracle语法，自然连接，using子句，on子句)。
- 2.查询部门名称为Shipping的员工的编号、姓名及所从事的工作。
- 3.查询员工的编号，姓名，以及部门名称，包括没有员工的部门。
- 4.查询员工的编号，姓名，以及部门名称，包括不属于任何部门的员工。

第六章　分组函数

Neusoft 东软

本章要点

- 分组函数基本概念
- SUM、AVG、MIN、MAX、COUNT函数使用
- GROUP BY子句
- HAVING子句

分组函数概念

- 分组函数是对表中一组记录进行操作，每组只返回一个结果。即首先要对表记录进行分组，然后再进行操作汇总，每组返回一个结果。分组时可能是整个表分为一组，也可能根据条件分成多组。
- 分组函数常用到以下的五个函数:
  - MIN
  - MAX
  - SUM
  - AVG
  - COUNT

MIN函数和MAX函数

- MIN和MAX函数主要是返回每组的最小值和最大值。
  - MIN([DISTINCT|ALL]表达式)
  - MAX([DISTINCT|ALL]表达式)

- **例6-1 员工最低工资及最高工资的示例。**

```
SELECT  MIN(salary), MAX(salary)
FROM   employees;
```

MIN(SALARY)	MAX(SALARY)
2500	24000

MIN函数和MAX函数示例

- 例6-2 员工姓最开始及最后的示例。

```
SELECT        MIN(last_name), MAX(last_name)
FROM   employees;
```

- 例6-3 员工最低工资及最高工资的示例 。

```
SELECT        MIN(hire_date), MAX(hire_date)
FROM   employees;
```

SUM函数和AVG函数

- SUM和AVG函数分别返回总和及平均值。
  - SUM([DISTINCT|ALL]n)
  - AVG([DISTINCT|ALL]n)
- SUM和AVG函数都是只能够对数字类型的列或表达式操作。
- **例6-4 公司员工总工资及平均工资的示例。**

```
SELECT      SUM(salary), AVG(salary)
FROM        employees;
```

COUNT函数

- COUNT函数的主要功能是返回每组记录的条数。
  - COUNT({*|[DISTINCT|ALL]表达式})

- **例6-5 公司IT_PROG职位的员工人数的示例。**

```
SELECT        COUNT(*)
FROM  employees
WHERE         job_id='IT_PROG';
```

- **例6-6 公司有部门员工人数的示例。**

```
SELECT COUNT(department_id)
FROM employees;
```

组函数中DISTINCT

- DISTINCT会消除重复记录后再使用组函数

- **例6-7 公司有员工部门数的示例。**

```
SELECT        COUNT(DISTINCT department_id)
FROM   employees;
```

组函数中空值处理

- 所有组函数对空值都是忽略的。

- **例6-8 员工平均奖金的示例——忽略空值。**

```
SELECT      AVG(commission_pct)
FROM        employees;
```

- **例6-9 员工平均奖金的示例——空值转化。**

```
SELECT      AVG(NVL(commission_pct,0))
FROM        employees;
```

GROUP BY子句

```
SELECT      列名, 组函数(列名)
FROM        表名
[WHERE      条件]
[GROUP BY   分组列]
[ORDER BY   列名];
```

- 组函数忽略空值,可以使用NVL,NVL2,COALESCE 函数处理空值
- 结果集隐式按升序排列,如果需要改变排序方式可以使用Order by 子句

GROUP BY子句示例

- 例6-10 每个部门的总工资。

```
SELECT   department_id, SUM(salary)
FROM     employees
GROUP BY department_id ;
```

- 例6-11 相同职位且经理相同的员工平均工资。

```
SELECT          job_id,manager_id, AVG(salary)
FROM            employees
GROUP BY job_id,manager_id
ORDER BY job_id;
```

GROUP BY子句注意问题

- 在GROUP BY子句使用中，有两点需要注意：
  - GROUP BY子句后的列可以不在SELECT语句中出现。
  - SELECT子句中出现的非分组函数列必须在GROUP BY子句中出现。
- **例6-12 公司每个职位的平价工资，职位列不显示，同时结果按照平均工资排序。**

```
SELECT      AVG(salary)
FROM        employees
GROUP BY job_id
ORDER BY AVG(salary);
```

GROUP BY子句注意问题示例

- 例**6-13** 分组汇总错误示例

```
SELECT        department_id, job_id, AVG(salary)
FROM  employees
GROUP BY department_id;
```

- 例**6-14** 分组汇总正确示例

```
SELECT        department_id, job_id, AVG(salary)
FROM  employees
GROUP BY department_id,job_id;
```

HAVING子句

- **例6-15 组函数筛选示例。**

```
SELECT   job_id, MAX(salary)
FROM   employees
WHERE  MAX(salary)>=9000
GROUP BY job_id;
```

- 原因是Oracle查询语句的执行顺序是:
  - FROM WHERE GROUP BY SELECT ORDER BY

HAVING子句

- 语法结构如下:

```
SELECT      列名, 组函数
FROM        表名
[WHERE      条件]
[GROUP BY   分组列]
[HAVING     组函数表达式]
[ORDER BY   列名];
```

HAVING子句

- 例**6-16** 组函数筛选示例。

```
SELECT      job_id, MAX(salary)
FROM        employees
GROUP BY    job_id
HAVING      MAX(salary)>=9000;
```

HAVING子句

- 总结SELECT语句执行过程：
  - 通过FROM子句中找到需要查询的表；
  - 通过WHERE子句进行非分组函数筛选判断；
  - 通过GROUP BY子句完成分组操作；
  - 通过HAVING子句完成组函数筛选判断；
  - 通过SELECT子句选择显示的列或表达式及组函数；
  - 通过ORDER BY子句进行排序操作。

HAVING子句

- **例6-17 组函数演示。**

```
SELECT        department_id, MAX(salary)
FROM  employees
WHERE         department_id BETWEEN 30 AND 90
GROUP BY  department_id
HAVING        MAX(salary)>=9000
ORDER BY  MAX(salary);
```

组函数的嵌套

- 组函数可以实现嵌套操作，嵌套级数是2级。

- **例6-18 组函数嵌套演示。**

```
SELECT          MAX(COUNT(employee_id))
FROM  employees
GROUP BY department_id;
```

本章小结

- MIN函数和MAX函数
- SUM函数和AVG函数
- COUNT函数
- 组函数中DISTINCT消除重复行
- 组函数中空值处理
- 通过GROUP BY子句进行分组汇总
- GROUP BY子句使用需要注意的两个问题
- HAVING子句的使用
- 组函数的嵌套

练习

- 1.查询部门平均工资在8000元以上的部门名称及平均工资。
- 2.查询工作编号中不是以“SA_”开头并且平均工资在8000元以上的工作编号及平均工资，并按平均工资降序排序。
- 3.查询部门人数在4人以上的部门的部门名称及最低工资和最高工资。
- 4.查询工作不为SA_REP，工资的和大于等于25000的工作编号和每种工作工资的和。
- 5.显示经理号码，这个经理所管理员工的最低工资，不包括经理号为空的，不包括最低工资小于3000的，按最低工资由高到低排序。

第七章　子查询

Neusoft 东软

本章要点

- 子查询语法介绍
- 单行子查询
- 多行子查询
- 相关子查询

子查询介绍

- 子查询的语法结构:

```
SELECT  查询列
FROM  表名
WHERE  列名 操作符
    (SELECT  查询列
            FROM  表名);
```

- 括号内的查询叫做子查询（Subquery）或者内部查询（Inner Query），外面的查询叫做主查询（Main query）或外部查询（Outer query）。

子查询示例

- 例7-1 和Chen员工在相同部门的员工信息。

```
SELECT        last_name, job_id, salary, department_id
FROM  employees
WHERE         department_id = (SELECT department_id
FROM employees
WHERE last_name = 'Chen');
```

- 关于子查询，需要注意以下几点：
  - 子查询需要写在括号中；
  - 子查询需要写在运算符的右端；
  - 子查询可以写在WHERE，HAVING，FROM子句中；
  - 子查询中通常不写ORDER BY子句。

子查询的种类

单行子查询

多行子查询

多列子查询

单行子查询

- 单行子查询，子查询返回的记录有且只有一条。单行子查询要求使用单行操作符，即:
  - >大于
  - >= 大于等于
  - < 小于
  - <= 小于等于
  - =等于
  - <> 不等于

单行子查询示例

- 例**7-2 WHERE**子句中单行子查询示例。

```
SELECT      last_name, job_id, salary, department_id
FROM        employees
WHERE       department_id =
                              (SELECT department_id
                              FROM   employees
                              WHERE  last_name = 'Chen')
AND    salary >
            (SELECT salary
             FROM   employees
             WHERE  last_name = 'Chen');
```

HAVING子句中单行子查询语句

- 除了在WHERE子句中外，也可以在HAVING子句中书写子查询。
- **例7-4** 员工人数高于各部门平均人数。

```
SELECT      department_id, COUNT(employee_id)
FROM        employees
GROUP BY    department_id
HAVING      COUNT(employee_id) >
           (SELECT AVG(COUNT(employee_id))
            FROM   employees
            GROUP BY department_id);
```

多行子查询

- 多行子查询，子查询返回记录的条数可以是一条或多条。多行子查询需要使用多行操作符。常用的多行操作符包括:
  - IN
  - ANY
  - ALL
- IN操作符和以前介绍的功能一致，判断是否与子查询的任意一个返回值相同。返回的结果可以是一条或多条。

多行子查询（续）

```
SELECT employee_id, last_name
FROM   employees
WHERE  salary =
              (SELECT   MIN(salary)
               FROM     employees
               GROUP BY department_id);
```

- 返回结果

  ERROR at line 4:

  ORA-01427: single-row subquery returns more than one row

多行子查询（续）

- 例7-6 多行子查询示例——IN操作符。

```
SELECT          a.last_name, a.salary
FROM   employees a
WHERE           a.employee_id IN
                (SELECT b.manager_id
                 FROM   employees b);
```

多行子查询（续）

- ANY：表示任意的。
  - < ANY 比子查询返回的任意一个结果小即可，即小于返回结果的最大值。
  - = ANY 和子查询中任意一个结果相等即可，相当于IN。
  - > ANY比子查询返回的任意一个结果大即可，即大于返回结果的最小值。
- ALL：表示所有的。
  - < ALL 比子查询返回的所有的结果都小，即小于返回结果的最小值。
  - > ALL比子查询返回的所有的结果都大，即大于返回结果的最大值。
  - = ALL 无意义，逻辑上也不成立。

多行子查询示例

- 例**7-7** ANY操作符。

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE          salary > ANY
               (SELECT salary
                FROM   employees
                WHERE  department_id = 80)
AND    department_id <> 80;
```

多行子查询示例（续）

- 例**7-8** ALL操作符。

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE          salary > ALL
               (SELECT salary
                FROM   employees
                WHERE  department_id = 80)
AND    department_id <> 80;
```

FROM语句中子查询

- 例**7-9 FROM**子句查询示例——分组判断。找出比员工本职位（**job_id**）平均工资高的员工信息。

```
SELECT          e.last_name, e.salary, e.job_id, j.avgsal
FROM   employees e, (SELECT   job_id,
                AVG(salary) avgsal
                FROM     employees
                GROUP BY job_id) j
WHERE   e.job_id = j.job_id
AND     e.salary > j.avgsal;
```

子查询中空值问题

- **例7-14 子查询空值示例。**

```
SELECT a.last_name, a.salary   FROM   employees a
   WHERE  a.employee_id NOT IN
                  (SELECT b.manager_id  FROM   employees b);
返回结果：未选定行
```

- 出现这种情况的原因有两个：
  - 子查询中返回值中包含有空值；
  - NOT IN操作符对空值不忽略。
- NOT IN操作符相当于<> ALL，即除了列表值的所有值，就包括了空值NULL，结果即为空。

相关子查询

- 相关子查询中，内部查询需引用外部查询的列，进行交互判断。相关子查询的执行方式是一行行操作。外部查询每执行一行操作，内部查询都要执行一次。

- **例7-16 公司经理希望找到比本职位工资高的员工的信息，可通过分组判断实现。**

```
SELECT  last_name, salary, job_id
FROM   employees e
WHERE  salary >(SELECT  AVG(salary)
               FROM   employees
               WHERE   job_id = e.job_id);
```

EXISTS和NOT EXISTS操作符

- 相关子查询还可使用EXISTS和NOT EXISTS操作符。
- EXISTS判断存在与否。具体操作如下：
  - 子查询中如果有记录找到，子查询语句不会继续执行，返回值为TRUE；
  - 子查询中如果到表的末尾也没有记录找到，返回值为FALSE。
- EXISTS子查询并没有确切记录返回，只判断是否有记录。而且只要找到相关记录，子查询就不需要再执行，然后再进行下面的操作。这样大大提高了语句的执行效率。
- NOT EXISTS正好相反，判断子查询是否没有返回值。如果没有返回值，表达式为真，如果找到一条返回值，则为假。

EXISTS操作符示例

- **例7-17 查找公司中的经理。**

```
SELECT    last_name, job_id, salary, department_id
FROM      employees e
WHERE     EXISTS (SELECT    '1'
          FROM   employees
          WHERE     manager_id = e.employee_id);
```

- 因为EXISTS子句中，并没有确切记录返回，只返回真或假。所以’1’只是占位用，无实际意义。

NOT EXISTS操作符示例

- **例7-18 查找公司中不是经理的员工。**

```
SELECT  last_name, job_id, salary, department_id
FROM   employees e
WHERE  NOT EXISTS (SELECT  '1'
                   FROM   employees
                   WHERE manager_id =  e.employee_id);
```

- NOT EXISTS操作符因为运算方法与NOT IN不同，只会返回TRUE或FALSE，不会返回空值，所以不需要考虑子查询去除空值的问题。

本章小结

- 子查询的基本介绍
- 单行子查询
- 多行子查询
- FROM语句中子查询
- 子查询中空值问题
- 相关子查询
- EXISTS和NOT EXISTS操作符

练习

- 1.查询工资高于编号为113的员工工资，并且和102号员工从事相同工作的员工的编号、姓名及工资。
- 2.工资最高的员工姓名和工资。
- 3.查询部门最低工资高于100号部门最低工资的部门的编号、名称及部门最低工资。
- 4.查询员工工资为其部门最低工资的员工的编号和姓名及工资。
- 5.显示经理是KING的员工姓名，工资。
- 6.显示比员工‘Abel’参加工作时间晚的员工姓名，工资，参加工作时间。

第八章　数据操作与事务控制

Neusoft 东软

本章要点

- INSERT语句
- UPDATE语句
- DELETE语句
- 9i新增MERGE语句
- COMMIT命令
- ROLLBACK命令
- 管理锁

数据操作语言

- 数据操作语言（DML： Data Manipulation Language）
- 主要包括以下语句:
  - INSERT
  - UPDATE
  - DELETE
  - MERGE
- 事务是一组相关的DML语句的逻辑组合。事务控制主要包括下列命令:
  - COMMIT
  - ROLLBACK
  - SAVEPOINT

插入数据

- 语法如下：

```
INSERT INTO 表名[(列名1[,列名2，…，列名n])]
VALUES (值1[,值2，…，值n]);
```

  - 一次只插入一行
  - NULL的使用，连续的单引号（‘’）也可以表示空值。
  - 插入日期型数据
  - 插入特殊字符
  - 插入多行数据

插入单行数据

- 例**8-1** 将一个新成立部门的信息写入**departments**表

```
INSERT INTO departments
VALUES(300,'Operations',110,1500);
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
300	Operations	110	1500
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
100	Finance	108	1700
110	Accounting	205	1700
210	IT Support		1700

10 rows selected.

插入空值

- 例8-2 将一个新成立部门的信息写入departments中，其中管理者未知。

```
INSERT INTO departments
VALUES(310,'Operations', NULL,1500);
```

或

```
INSERT INTO departments
VALUES(310,'Operations', ’’,1500);
```

或

```
INSERT INTO departments
      （department_id,department_name,location_id）
VALUES(310,'Operations',1500);
```

插入日期型数据

- 例**8-5** 将一新入职员工信息写入**employees**表

```
INSERT INTO
    employees(employee_id,last_name,email,hire_date,job_id)
VALUES(210,'Wang','SWANG',' 10-9月-06','IT_PROG');
```

或

```
INSERT INTO
    employees(employee_id,last_name,email,hire_date,job_id)
VALUES(210,'Wang','SWANG',TO_DATE('2006-9-10','YYYY-
    MM-DD'),'IT_PROG');
```

插入特殊字符

```
INSERT INTO test VALUES（‘&TEST&’）；
```

- 查看ESCAPE转义符用哪个符号表示。

```
SHOW ESCAPE；  ——查看ESCAPE状态
escape OFF       ——返回ESCAPE状态为OFF
SET ESCAPE ON；  ——设定ESCAPE状态为ON
SHOW ESCAPE；  ——查看ESCAPE状态
escape "\" (hex 5c)  ——返回ESCAPE符号为“\”
```

- INSERT语句中使用“\”符对特殊符号转义。

```
INSERT INTO test
VALUES（‘\&TEST\&’）；
```

插入多行数据

- 语法

INSERT INTO*表名*[(*列名1*[,*列名2*，…，*列名n*])] *子查询*；

- **例8-7 将受雇日期在“1995-1-1”之前的员工信息复制到hemployees表中。**

```
INSERT INTO hemployees
SELECT * FROM employees
WHERE hire_date<TO_DATE('1995-1-1','YYYY-MM-DD');
```

修改数据

- UPDATE语法结构

```
UPDATE 表名 SET 列名=表达式[，列名=表达式，…]
[WHERE 条件表达式];
```

- UPDATE简单修改

```
UPDATE employees SET salary=salary*(1+0.2);
```

```
UPDATE employees
SET salary=salary+200,manager_id=103
WHERE department_id=60;
```

修改数据（续）

- 嵌入子查询修改

```
UPDATE employees
SET department_id=10,
    salary=500+(SELECT AVG(salary) FROM employees)
WHERE job_id=(SELECT job_id FROM employees WHERE
     employee_id=110)
AND employee_id<>110;
```

- 除基于表自身实现嵌入子查询的方式实现修改操作外，也可以在子查询中基于其他表实现修改操作。

删除数据

- DELETE语法结构

```
DELETE [FROM] 表名
[WHERE 条件表达式];
```

- DELETE删除数据

```
DELETE FROM departments
WHERE department_id=210;
```

删除数据（续）

- **例8-12** 删除管理者编号（manager_id）为205的部门，相应部门的员工予以解聘，不包括205号员工。

```
DELETE FROM employees
WHERE department_id
IN (SELECT department_id FROM departments WHERE
    manager_id =205)
AND employee_id<>205;
```

- DELETE语句不能删除被其他表引用了的记录值。

合并数据

- MERGE语句减少了多条INSERT 和UPDATE语句的反复应用。降低了在代码编写时因语法错误而带来的风险。减少对表读取操作的次数，从而提高执行效率。
- MERGE语法结构

```
MERGE INTO [schema .] table [t_alias]
USING [schema .] { table | view | subquery } [t_alias]
ON ( join condition )
WHEN MATCHED THEN
UPDATE SET col1=col1_val[,col2=col2_val…]
WHEN NOT MATCHED THEN
INSERT（column_list）
VALUES（column_values）;
```

合并数据示例

```
MERGE INTO emp a
USING employees b  ON(a.employee_id=b.employee_id)
WHEN MATCHED THEN
    UPDATE SET
    a.email=b.email,a.phone_number=b.phone_number,
    a.salary=b.salary, a.manager_id=b.manager_id,
    a.department_id=b.department_id
WHEN NOT MATCHED THEN
    INSERT
     (employee_id,email,phone_number,salary,manager_id,departm
     ent_id)
    VALUES(b.employee_id, b.email, b.phone_number, b.salary,
     b.manager_id, b.department_id);
```

事务处理

- 事务（Transaction）也称工作单元，是一个或多个SQL语句所组成的序列，这些SQL操作作为一个完整的工作单元，要么全部执行，要么全部不执行。通过事务的使用，能够使一系列相关操作关联起来，防止出现数据不一致现象。
- 在ORACLE数据库中，事务由以下语句组成：
  - 一组相关的DML语句，修改的数据在该组语句中保持一致
  - 一个 DDL语句
  - 一个 DCL语句

事务的特征

- 可用四个字母的缩写表示：即ACID
- 原子性（Atomicity）
  - 事务就像一个独立的工作单元。原子性保证要么所有的操作都成功，要不全都失败。如果所有的动作都成功了，我们就说这个事务成功了，不然就是失败的，然后回滚。
- 一致性（Consistency）
  - 一旦事务完成了（不管是成功的，还是失败的），整个系统处于操作规则的统一状态，也就是说，数据不会损坏。

事务的特征（续）

- 隔离性（Isolation）
  - 事务的隔离性是指数据库中一个事务的执行不能被其它事务干扰。所以，事务应该隔离起来，目的为了防止同时的读和写操作。这就需要事务与锁同时使用。
- 持久性（Durability）
  - 事务的持久性也称为永久性（Permanence），指事务一旦提交，则其对数据库中数据的改变就是永久的。

事务控制

- 事务控制的命令主要有以下三个：
  - 事务提交：COMMIT
  - 事务回滚：ROLLBACK
  - 设立保存点：SAVEPOINT（作为辅助命令使用）
- 事务开始于上一个事务结束后执行的第一个SQL语句，事务结束于下面的任一种情况的发生：
  - 执行了COMMIT 或者ROLLBACK命令
  - 隐式提交（单个的DDL或DCL语句）或自动提交
  - 用户退出
  - 系统崩溃

事务控制（续）

- 设置保存点语法：
  - SAVEPOINT *保存点名称*; ——定义保存点
  - ROLLBACK TO*保存点名称*; ——回滚到已定义保存点

事务控制（续）

- 事务的控制分：显式控制及隐式控制。使用COMMIT和ROLLBACK命令是显式控制。有些情况下，事务被隐式控制，事务隐式控制可分为隐式提交和隐式回滚。
- 当下列情况时，事务被隐式提交。
  - 执行一个DDL 语句
  - 执行一个DCL 语句
  - 从 SQL*Plus正常退出（即使用EXIT或QUIT命令退出）
- 当下列情况时，事务被隐式回滚。
  - 从SQL*Plus中强行退出
  - 客户端连接到服务器端异常中断
  - 系统崩溃

事务自动提交

- 设置格式：

```
SET AUTOCOMMIT [ON|OFF];
```

- **例8-14 SQL*Plus自动提交的应用示例**

```
SHOW AUTOCOMMIT;     ——查看AUTOCOMMIT变量状态
autocommit OFF
SET AUTOCOMMIT ON;    ——把变量状态设置为ON
INSERT INTO test VALUES ('TEST');
已创建 1 行
提交完成                    ——已经自动提交
```

事务控制示例

```
DELETE FROM test ;
ROLLBACK;                    ——撤消DELETE操作
INSERT INTO test VALUES('A');
SAVEPOINT insert_a;          ——定义insert_a保存点
INSERT INTO test VALUES('B');
SAVEPOINT insert_b;          ——定义insert_b保存点
INSERT INTO test VALUES('C');
ROLLBACK TO insert_b;    ——撤消操作到insert_b保存点
DELETE FROM test WHERE test_str = 'A';
COMMIT;                          ——将所有修改写入数据库
ROLLBACK;                    ——所有操作已经COMMIT提交，不能
    回滚
```

读一致性

- 读一致性保证了不同会话在同一时间查看数据时，数据一致。
- ORACLE在两个不同级别上提供读一致性：语句级读一致性和事务级一致性。
- 事务级一致性，当一个会话正在修改数据时，其它的会话将看不到该会话未提交的修改。
- 语句级读一致性，保证单个查询所返回的数据与该查询开始时刻相一致。所以一个查询从不会看到在查询执行过程中提交的其它事务所作的任何修改。

锁的概念

- 锁用来在多用户并发访问和操作数据库时保证数据的一致性。锁由Oracle自动管理。如一个DML操作，ORACLE默认的机制是在DML操作涉及到的行上加锁（行级别），但不会在更高的级别（表级别）上加更严格的锁，比如只改某行的数据不会锁住整个表。这提供了很好的并发性，因为整个表没有锁定，只是某些行被锁定了，其他用户可以修改其他行数据。
- 查询不需要任何锁。

锁的概念（续）

- 锁的生命周期，锁在被相关的操作申请并持有后，会一直保持到事务的结束。事务结束后，锁才会被释放。
- 锁的内部维护机制是采用排队机制（enqueue），一个对象的排他锁被持有后，该对象相同级别的锁被其他事务申请时候，所有等待该锁的事务都在一个等待队列中排队，其他事务处于等待状态。直到该锁被释放，等待的事务才重新竞争使用该资源。
- **锁的模式**
  - 排他锁模式（Exclusive）排他锁在被释放之前，会阻止其锁住的资源被其他任何事务共享。
  - 共享模式（Share）

锁的分类

- 两种类型的锁，DML锁和DDL锁。
- DML锁，也称数据锁，用于在数据被多个不同的用户改变时，保证数据的完整性。
- DDL锁，也称为数据字典锁，执行DDL语句时，DDL语句涉及到的对象获得DDL锁。由于被持有的时间很短，因此很少看到冲突的DDL锁，并且以nowait方式被请求。

锁的分类（续）

- 3种DDL锁类型：
  - 排他的DDL锁，很多对象的创建、修改和删除定义时候都需要获得该锁。比如执行Create Table、Drop Table等时会获得表上的排他DDL锁。
  - 共享的DDL锁。在执行Grant、Create Procedure等命令时，会获得命令相关操作对象的共享DDL锁。
  - Breakable Parse Lock，用来在共享SQL区校验语句。

锁的常见问题

- 如使用锁的方式不当，可能会产生锁冲突，不适当的使用方式包括:
  - 不必要的高级别的锁
  - 长时间运行的事务
  - 没有提交的事务
  - 其他产品产生了高级别的锁
- 死锁：当两个或者两个以上的用户彼此等待被对方锁定的资源时，就有可能产生死锁。

本章小结

- INSERT语句
- UPDATE语句
- DELETE语句
- MERGE语句
- 事务的概念及事务控制
- 锁的概念及加锁语句的应用

练习

- 1.向departments表中的部门编号、部门名称、区域编号三列插入两条纪录，分别为：300，‘QQQ’，1500和310，‘TTT’，1700。观察执行结果。
- 2.使用两种方法完成向列操作，试在新部门的管理者和工作地区编号还没有确定的情况下，向部门表中插入新部门信息如下：部门编号 320及330，部门名称F1及F2。
- 3.按顺序执行下列操作：
  1、插入一个新的部门信息，开始事务。部门编号350，名称 人力资源 管理者 100 区域编号 1700。
  2、建立保存点a。

练习（续）

3、查询插入的数据是否存在。

4、删除所有部门编号大于200的部门。

5、建立保存点b。

6、查询还有哪些部门信息存在。

7、更新部门编号为10的部门的管理者的编号为110。

8、查询当前部门信息。

9、执行回滚操作，但不回滚到事务的最开始，而是回滚到保存点b。

10、提交事务，提交后事务已结束。

11、查看最终数据修改结果。

第九章　表和约束

Neusoft 东软

本章要点

- 创建表
- 修改表
- 删除表
- 重命名
- 截断表
- 数据完整性约束

创建表

- Oracle中数据库对象命名原则
  - 必须由字母开始，长度在 1–30个字符之间。
  - 名字中只能包含 A–Z, a–z, 0–9, _ (下划线)，$ 和 #。
  - 同一个Oracle服务器用户所拥有的对象名字不能重复。
  - 名字不能为Oracle的保留字。
  - 名字是大小写不敏感

创建表（续）

- 建表语法

```
CREATE TABLE [schema.]table
(column datatype [DEFAULT expr][, ...]);
```

创建表示例

```
CREATE TABLE DOSSIER (
    ID NUMBER(4),
    CNAME VARCHAR2(20 ),
    BIRTHDAY DATE,
    STATURE    NUMBER(3),
    WEIGHT NUMBER(5, 2),
    COUNTRY_CODE CHAR(2 ) DEFAULT ‘01’);
```

默认值应用

- 插入默认值

```
insert into dossier
(ID,CNAME,BIRTHDAY, STATURE,WEIGHT  )
values
(2,'姚明' , to_date('1980.9.12','yyyy.mm.dd'),226, 134 );
已创建 1 行。
```

- 使用默认值修改

```
update dossier set country_code =default where id=2;
```

使用子查询创建表

- 使用子查询创建表的语法

```
CREATE TABLE table[(column, column...)]
AS subquery;
```

- 新表的字段列表必须与子查询中的字段列表匹配
- 字段列表可以省略

使用子查询创建表示例

```
CREATE TABLE dept10
AS
SELECT employee_id, last_name, salary + 1000 newSalary
FROM employees
WHERE department_id = 10;
```

- Select列表中的表达式列需要给定别名，如果没有别名会产生错误

引用另一个用户的表

- 如果一个表不属于当前用户，如果引用它，必须把方案名放在表名的前面。例如，scott.emp

```
SELECT *
FROM scott.emp;
```

Oracle中表的分类

- Oracle 数据库中的表分为下面两类:
  - 用户表：由用户创建和维护的表的集合，它包含用户所使用的数据。
  - 数据字典：由Oracle 服务器创建和维护的表的集合，它包含数据库信息，比如是表的定义，数据库结构的信息等，可以把它理解为表的表，由Oracle服务器创建和维护。如user_tables。

使用数据字典

- 查询数据字典

```
SELECT table_name
FROM  user_tables;
```

- 察看数据字典结构

```
DESC user_tables
```

数据类型与列定义

数据类型	说明
VARCHAR2 (size)	可变长度字符数据，最小字符数是 1；最大字符数是 4000
CHAR(size)	固定长度字符数据，长度的大小以字节为单位，默认和最小字符数为 1；最大字符数为 2000
NUMBER(p,s)	数字，精度为p，小数为s ，p是数据的总长度，s是小数点右边的数字长度；p的范围从1到38，s的范围从-84到127
DATE	日期和时间类型
LONG	最大2G的可变长度字符数据
LONG RAW	可变长度原始二进制数据，最大2G
CLOB	最大可存储4G的字符数据
BLOB	最大可存储4G二进制的数据
BFILE	最大可存储4G数据,保存在数据库外部的文件里
ROWID	十六进制串，表示行在表中唯一的行地址

其他类型

- ROWID：伪列，是表中虚拟的列，是系统自动产生的，每一行记录中都包含ROWID，表示这一行的唯一地址，ROWID标识了Oracle如何定位行，通过 ROWID 能快速定位这行记录。

```
SELECT rowid,cname
FROM dossier;
```

ROWID	CNAME
AAAHjXAAOAAAADaAAA	姚明
AAAHjXAAOAAAADaAAC	成龙

ALTER语句

- 添加列语法:

```
ALTER TABLE table
ADD (column datatype[DEFAULT expr]
        [, column datatype]...);
```

- 修改列语法:

```
ALTER TABLE table
MODIFY(column datatype[DEFAULT expr]
            [, column datatype]...);
```

- 删除列语法:

```
ALTER TABLE table
DROP (column);
```

添加新列

- 增加列原则:
  - 可以添加或修改列
  - 不能指定新添加列的位置，新列会成为最后一列。

- 如在dossier表上增加性别字段:

```
ALTER TABLE dossier ADD  (sex CHAR(1));
```

修改已存在的列

- 把dossier表性别(sex) 列，修改为长度为2

```
ALTER TABLE dossier MODIFY  (sex CHAR(2));
```

- 添加默认值

```
ALTER TABLE dossier
MODIFY  (sex  DEFAULT ‘男’);
```

  - 默认值不会影响已经存在的值。
  - 默认值只影响新增加的字段值。

删除列

- 可以用DROP子句从表中删除列，包括列的定义和数据。
- 删除列原则:
  - 列可以有也可以没有数据。
  - 表中至少保留一列。
  - 列被删除后，不能再恢复。
  - 被外键引用的列，不能被删除。

删除列（续）

- 删除列语法一

```
ALTER TABLE emp DROP COLUMN sex;
```

- 删除列语法二

```
ALTER TABLE table DROP (column[,column]);
```

- 删除dept10表的两个字段“last_name”和“newsalary”。

```
ALTER TABLE dept10 DROP (last_name,newsalary);
```

删除表

- 删除表语法:

```
DROP TABLE table;
```

  - 只有表的创建者
  - 或具有DROP ANY TABLE权限的用户才能删除表
- 删除表原则:
  - 表中所有的数据和结构都被删除。
  - 任何视图和同义词被保留但无效。
  - 所有与其相关的约束和索引被删除。
  - 任何未完成的事务被提交。

```
DROP TABLE emp;
```

重命名表

- 重命名语句语法:

```
RENAME old_name TO new_name;
```

  - 必须是对象的所有者

- 把emp表重新命名为empl

```
RENAME emp TO empl;
```

截断表

- 截断表语法:

```
TRUNCATE TABLE table;
```

  - 执行TRUNCATE语句的前提，必须是表的所有者
  - 或者有DELETE ANY TABLE系统权限来截断表。

```
TRUNCATE TABLE emp;
```

约束的描述

- Oracle服务器用约束 *(constraints)* 来防止无效数据输入到表中。约束做下面的事:
  - 多个表之间的具体关系，比如两个表之间的主外键关系。
  - 表在插入、更新行或者删除行的时候强制表中的数据遵循约束规则。
  - 对于成功的操作，约束条件是必须被满足的。
  - 如果表之间有依赖关系，使用约束可以防止表或表中相关数据的删除。

Oracle中约束类型

约束	说明
NOT NULL	指定列不能包含空值
UNIQUE	指定列的值或者列的组合的值对于表中所有的行必须是唯一的
PRIMARY KEY	表的每行的唯一性标识
FOREIGN KEY	在列和引用表的一个列建立并且强制的列之间关系
CHECK	指定一个必须为真的条件

约束命名

- 约束命名原则：所有的约束定义存储在数据字典中。
- 如果给约束一个有意义的名字，约束易于维护，约束命名必须遵守标准的对象命名规则。
- 如果没有给约束命名，Oracle服务器将用默认格式SYS_C*n*产生一个名字，这里 *n* 是一个唯一的整数，来保证名称的唯一性。
- 建议至少应该给表的主、外键按照命名原则来命名，如可以采用这样的原则来命名，表名_字段名_约束类型。

生成与维护约束

- 约束的语法如下：

```
CREATE TABLE [schema.] table
 (column datatype [ DEFAULTexpr][column_constraint],
   ...[table_constraint][,...]);
```

  - 约束可以在两个级别上定义，表级约束与列级约束。
  - 列级约束能够定义完整性约束的任何类型。
  - 表级约束除了NOT NULL之外，能够定义完整性约束的任何类型。

NOT NULL约束

- NOT NULL约束在列级被指定

```
CREATE TABLE COUNTRY (
    COUNTRY_CODE CHAR(2 )  PRIMARY KEY,
COUNTRY_NAME VARCHAR2(50)  NOT NULL);
```

- 而不可以指定为表级约束

```
CREATE TABLE COUNTRY (
    COUNTRY_CODE CHAR(2 )  PRIMARY KEY,
COUNTRY_NAME VARCHAR2(50) ,
 NOT NULL(COUNTRY_NAME));
```

UNIQUE约束

- 唯一性约束条件确保所在的字段或者字段组合不出现重复值
- 唯一性约束条件的字段允许出现空值
- Oracle将为唯一性约束条件创建对应的唯一性索引

UNIQUE约束（续）

```
CREATE TABLE employees(
    employee_id      NUMBER(6),
    last_name        VARCHAR2(25) NOT NULL,
    email            VARCHAR2(25),
    salary           NUMBER(8,2),
    commission_pct   NUMBER(2,2),
    hire_date        DATE NOT NULL,
...
    CONSTRAINT emp_email_uk UNIQUE(email));
```

PRIMARY KEY约束

- 主键从功能上看相当于非空且唯一
- 一个表中只允许一个主键
- 主键是表中能够唯一确定一个行数据的字段
- 主键字段可以是单字段或者是多字段的组合
- Oracle为主键创建对应的唯一性索引

PRIMARY KEY约束示例

```
CREATE TABLE   departments(
    department_id        NUMBER(4),
    department_name      VARCHAR2(30)
      CONSTRAINT dept_name_nn NOT NULL,
    manager_id           NUMBER(6),
    location_id          NUMBER(4),
      CONSTRAINT dept_id_pk PRIMARY KEY(department_id));
```

FOREIGN KEY约束

- 外键确保了相关的两个字段的两个关系：
  - 子表外键列的值必须在主表参照列值的范围内，或者为空
  - 外键参照的是主表的主键或者唯一键。
- 主表外键值被子表参照时，主表记录不允许被删除。
  - ON DELETE CASCADE 指出当父表中的行被删除时，子表中相依赖的行也将被级联删除。
  - ON DELETE SET NULL 当父表的值被删除时，把涉及到子表的外键值设置为空。

FOREIGN KEY约束示例

```
CREATE TABLE employees(
    employee_id      NUMBER(6),
    last_name        VARCHAR2(25) NOT NULL,
    email            VARCHAR2(25),
    salary           NUMBER(8,2),
    commission_pct   NUMBER(2,2),
    hire_date        DATE NOT NULL,
...
    department_id    NUMBER(4),
    CONSTRAINT emp_dept_fk FOREIGN KEY (department_id)
      REFERENCES departments(department_id),
    CONSTRAINT emp_email_uk UNIQUE(email));
```

FOREIGN KEY约束

- 删除外键

```
Alter table employees drop constraint EMP_DEPT_FK;
```

- 添加使用ON DELETE CASCADE子句外键

```
Alter table employees add constraint EMP_DEPT_FK foreign key
    (department_id)
References departments (department_id)
ON DELETE CASCADE;
```

CHECK 约束

- 定义在字段上的每一记录都要满足的条件
- 在check中定义检查的条件表达式，数据需要符合设置的条件
- 条件表达式不允许使用:
  - SYSDATE, UID, USER, USERENV 等函数
  - 参照其他记录的值

```
..., salary          NUMBER(2)
    CONSTRAINT emp_salary_min
        CHECK (salary > 0),...
```

增加约束

- 增加约束语法如下:

```
ALTER TABLE table ADD [CONSTRAINT constraint] type
    (column);
```

删除约束

- 删除约束

```
Alter table dossier drop constraint dossier_countrycode_fk;
```

```
Alter table country drop primary key CASCADE;
```

约束的启用和禁用

- 禁用约束
  - 如果有大批量数据导入时，我们可以采用禁用约束的方法，主要的好处，首先效率高，另外有主外键约束的表之间导入时，不用考虑导入的先后顺序。

- 禁用约束语法：

```
ALTER TABLE table  DISABLE CONSTRAINT constraint
    [CASCADE];
```

启用约束

- 启用约束语法:

```
ALTER TABLE table  ENABLE CONSTRAINT constraint;
```

相关数据字典

- 和约束相关的数据字典有USER_CONSTRAINTS和USER_CONS_COLUMNS。
- USER_CONSTRAINTS表：查看表上所有的约束。USER_CONS_COLUMNS表查看与约束相关的列名，该视图对于那些由系统指定名字的约束特别有用。
- 在约束类型中，C代表CHECK，P代表PRIMARY KEY，R代表FOREIGN KEY， U代表UNIQUE，NOT NULL约束实际上是一个CHECK约束。

```
select constraint_name,constraint_type
from user_constraints;
```

本章小结

- 创建表： 建表语法，用子查询基于另一个表创建表。
- 修改表： 修改表结构；修改列宽，改变列数据类型和添加列。
- 删除表： 删除行和表结构。
- 重命名： 重命名一个表、视图、序列或同义词。
- 截断： 从表中删除所有行，并且释放该表已使用的存储空间。
- 约束类型
- 约束相关数据字典
  - USER_CONSTRAINTS
  - USER_CONS_COLUMNS

练习

- 1.创建表date_test,包含列d，类型为date型。试向date_test表中插入两条记录，一条当前系统日期记录，一条记录为“1998-08-18”。
- 2.创建与departments表相同表结构的表dtest，将departments表中部门编号在200之前的信息插入该表。
- 3.创建与employees表结果相同的表empl，并将其部门编号为前50号的部门的信息复制到empl表。
- 4.试创建student表，要包含以下信息:

  学生编号（sno）：字符型（定长）4位 主键

  学生姓名（sname）：字符型（变长）8位 唯一

  学生年龄（sage）：数值型 非空

练习(续)

- 5.试创建sc表（成绩表），要包含以下信息:
  学生编号（sno）： 字符型（定长）4位 主键 外键
  课程编号（cno）： 字符型（变长）8位 主键
  选课成绩（grade）： 数值型
- 6.试为student增加一列学生性别 默认值 “女”。
- 7.试修改学生姓名列数据类型为定长字符型10位。
- 8.试修改学生年龄列允许为空。
- 9.试为选课成绩列添加校验（check）约束为1-100;
- 10.试删除sc表中的外键约束。

第十章　其他数据库对象

Neusoft 东软

本章要点

- 视图
- 序列
- 索引
- 同义词

视图

- 视图是虚表。是一个命名的查询，用于改变基表数据的显示，简化查询。视图的访问方式与表的访问方式相同。

- 视图的好处：
  - 可以限制对基表数据的访问，只允许用户通过视图看到表中的一部分数据
  - 可以使复杂的查询变的简单
  - 提供了数据的独立性，用户并不知道数据来自于何处
  - 提供了对相同数据的不同显示

简单视图和复杂视图

- 简单视图：只涉及到一个表，而且SELECT子句中不包含函数表达式列（包括单行函数和分组函数）。
- 复杂视图：涉及到一个或多个表，SELECT子句中包含函数表达式列（单行函数或负责函数）。

创建视图

```
CREATE [OR REPLACE] VIEW view
  [(alias[, alias]...)]
 AS subquery
[WITH CHECK OPTION [CONSTRAINT constraint]]
[WITH READ ONLY [CONSTRAINT constraint]];
```

- 在子查询中可以加入复杂的SELECT.

创建视图示例

- 例**10-3** 查询**50**部门的员工的年薪的视图

```
CREATE OR REPLACE VIEW salvu50
AS
SELECT  employee_id ID_NUMBER, last_name NAME,
          salary*12 ANN_SALARY
   FROM    employees
   WHERE   department_id = 50;
```

```
DESC salvu50;
```

```
SELECT * FROM salvu50;
```

通过视图执行DML操作

- 创建一个测试用表EMP_DML

```
CREATE TABLE emp_dml AS
SELECT employee_id,last_name,salary
FROM employees
WHERE department_id=50;
```

- 创建视图v_emp1，是个简单视图。

```
CREATE OR REPLACE VIEW v_emp1 AS
SELECT employee_id,salary FROM emp_dml;
```

- 通过视图进行DML操作。

```
UPDATE v_emp1 SET salary=salary+100;
```

不能通过视图删除记录的条件

- 视图中包含分组函数
- 视图中含有GROUP BY子句
- 视图中含有DISTINCT关键字
- 视图中包含伪列ROWNUM

不能通过视图修改记录的条件

- 视图中包含分组函数
- 视图中含有GROUP BY子句
- 视图中含有DISTINCT关键字
- 视图中包含伪列ROWNUM
- 视图中要修改的列包含表达式

不能通过视图添加记录的条件

- 视图中包含分组函数
- 视图中含有GROUP BY子句
- 视图中含有DISTINCT关键字
- 视图中包含伪列ROWNUM
- 视图中要修改的列包含表达式
- 视图中没有表的NOT NULL列。

WITH CHECK OPTION

- WITH CHECK OPTION实质是给视图加一个“CHECK”约束，该CHECK约束的条件就是视图中的子查询的WHERE条件，以后如果想通过该视图执行DML操作，不允许违反该CHECK约束。

WITH CHECK OPTION示例

- **例10-6 WITH CHECK OPTION例子**

```
CREATE OR REPLACE VIEW v_emp3
AS
SELECT employee_id,salary FROM emp_dml
WHERE employee_id=141
WITH CHECK OPTION CONSTRAINT v_emp3_ck;
```

- 该约束的条件为视图中WHERE条件，即“employee_id=141”，如视图v_emp3想执行DML操作，不能把记录的employee_id字段值改成其他编号（只能是141），如果违反了，执行出错，会出现错误提示。

WITH READ ONLY

- WITH READ ONLY的视图是只读的，不允许通过该视图执行DML语句。
- **例10-7 WITH READ ONLY例子,创建一个视图v_emp4**

```
CREATE OR REPLACE VIEW v_emp4
AS
SELECT employee_id,salary FROM emp_dml
WITH READ ONLY;
```

- 通过该视图进行更新操作

```
UPDATE v_emp4 SET salary=salary+100;
```

删除视图

- 删除视图的语法

```
DROP VIEW view;
```

内联视图

- 内联视图（Inline View）,是一个在SQL语句内可以使用的子查询的别名。是一个命名的SQL语句，但不是真正的数据库的视图对象。最常见的内联视图的例子就是主查询中的FROM子句中，包含的是一个命名的子查询。
- **例10-8 内联视图的例子**

```
SELECT last_name,department_name
FROM DEPARTMENTS a,
(SELECT last_name,department_id FROM EMPLOYEES) b
WHERE a.department_id=b.department_id;
```

Top-N分析

- Top-N查询主要是实现查找表中最大或最小的N条记录功能。
- Top-N分析语法：

```
SELECT [列名], ROWNUM
FROM   (SELECT [列名]
        FROM 表名
        ORDER  BY Top-N操作的列)
WHERE  ROWNUM <=  N;
```

Top-N分析（续）

- ROWNUM是一个伪列。功能是在每次查询时，返回结果集的顺序数。第一行显示为1，第二行为2，以此类推。
- 伪列是使用上类似于表中的列，而实际并没有存储在表中的特殊对象。
- 对ROWNUM只能执行<、<=运算，不能执行>、>=或一个区间运算Between..And等 。

序列

- 序列是一种用于产生唯一数字列值的数据库对象。一般使用序列自动地生成主码值或唯一键值。序列可以是升序或降序。
- 序列特点:
  - 可以为表中的记录自动产生唯一序列值。
  - 由用户创建并且可被多个用户共享。
  - 典型应用是生成主键值，用于标识记录的唯一性。
  - 允许同时生成多个序列号，而每一个序列号是唯一的。
  - 可替代应用程序中使用到的序列号。
  - 使用缓存加速序列的访问速度。

创建序列

- 创建序列的语法：

```
CREATE SEQUENCE [schema.]序列名
[INCREMENT BY n]
[START WITH n]
[MAXVALUE n | NOMAXVALUE]
[MINVALUE n | NOMINVALUE]
[CYCLE | NOCYCLE]
[CACHE n | NOCACHE];
```

- 创建序列，必须有CREATE SEQUENCE或CREATE ANY SEQUENCE权限。
- 序列被创建后，可以通过查询数据字典视图USER_SEQUENCES查看序列信息。

序列语法说明

- 当全部缺省时，则该序列为上升序列，由1开始，增量为1，没有上限，缓存中序列值个数为20。
- INCREMENT BY n： n可为正的或负的整数，但不可为0。默认值为1。
- START WITH n：指定生成的第一个序列号，默认为1。对于升序，默认为序列的最小值。对于降序，默认为序列的最大值。
- MAXVALUE n：指定n为序列可生成的最大值。
- NOMAXVALUE：为默认情况。指定升序最大值为10^{27}，指定降序指定最大值为-1。

序列语法说明（续）

- MINVALUE n：指定n为序列的最小值。
- NOMINVALUE：为默认情况。指定升序默认最小值为1。指定降序默认最小值为-10^{26}。
- CYCLE：指定序列使用循环。即序列达到了最大值，则返回最小值重新开始。默认为NOCYCLE。
- CACHE n：定义n个序列值保存在缓存中，默认值为20个。

创建序列

- **例10-12　创建序列test_seq。**

```
CREATE SEQUENCE test_seq
START WITH 10          ——序列从10开始
INCREMENT BY 2         ——序列每次增加2
MAXVALUE 100           ——序列最大值100
MINVALUE 9             ——序列最小值9
CYCLE                  ——序列循环。每次增加2，一直到100后回
                         到9从新开始
CACHE 10;              ——缓存中序列值个数为10
```

伪列

- 可用语句sequence_name.CURRVAL和sequence_name.NEXTVAL来访问序列。
  - CURRVAL当前序列正被分配的序列值。
  - NEXTVAL在序列中增加新值并返回此值。
  - CURRVAL和NEXTVAL都返回NUMBER值。
- 下列语句可使用NEXTVAL和CURRVAL伪列：
  - SELECT语句的非子查询的目标列名列表中。
  - INSERT语句中的子查询的SELECT目标列名列表中。
  - INSERT语句的VALUES子句中。
  - UPDATE语句的SET子句中。

伪列（续）

- 下列语句不允许使用NEXTVAL和CURRVAL伪列：
  - 在对视图查询的SELECT目标列名列表中。
  - 使用了DISTINCT命令的SELECT语句中。
  - SELECT语句中使用了GROUP BY、HAVING或ORDER BY子句时。
  - 在SELECT、DELETE或UPDATE语句的子查询中。
  - 在CREATE TABLE或ALTER TABLE语句中的默认值表达式中。

序列与伪列的应用示例

- 创建序列student_seq：

```
CREATE SEQUENCE student_seq
START WITH 10000
INCREMENT BY 1;
```

- 使用序列student_seq生成student表中sid列插入值：

```
INSERT INTO student
VALUES (student_seq.NEXTVAL, 'Scott', 'Computer Science', 11);
```

- 查看student_seq序列当前值：

```
SELECT student_seq.CURRVAL FROM dual;
```

修改序列

- 修改序列的语法如下:

```
ALTER  SEQUENCE  [schema.]序列名
      [INCREMENT BY n]
      [MAXVALUE n | NOMAXVALUE]
      [MINVALUE n | NOMINVALUE]
      [CYCLE | NOCYCLE]
      [CACHE n | NOCACHE];
```

- 必须是序列的所有者，或者有ALTER ANY SEQUENCE 权限才能修改序列。
- 修改序列的语法除没有START WITH子句外。

修改序列示例

- 正确修改

```
ALTER SEQUENCE test_seq
INCREMENT BY 4    ——序列每次增加4
MAXVALUE 1000     ——序列最大值1000
NOCACHE;          ——不设定缓存
```

- 错误修改

```
ALTER SEQUENCE test_seq
INCREMENT BY 4
MAXVALUE 100      ——最大值100小于已经分配的序列值200
NOCACHE;
```

删除序列

- 对序列的删除必须是序列的所有者或者具有DROP ANY SEQUENCE权限的用户才可以完成。
- 删除序列的语法如下:

```
DROP SEQUENCE [schema.] 序列名;
```

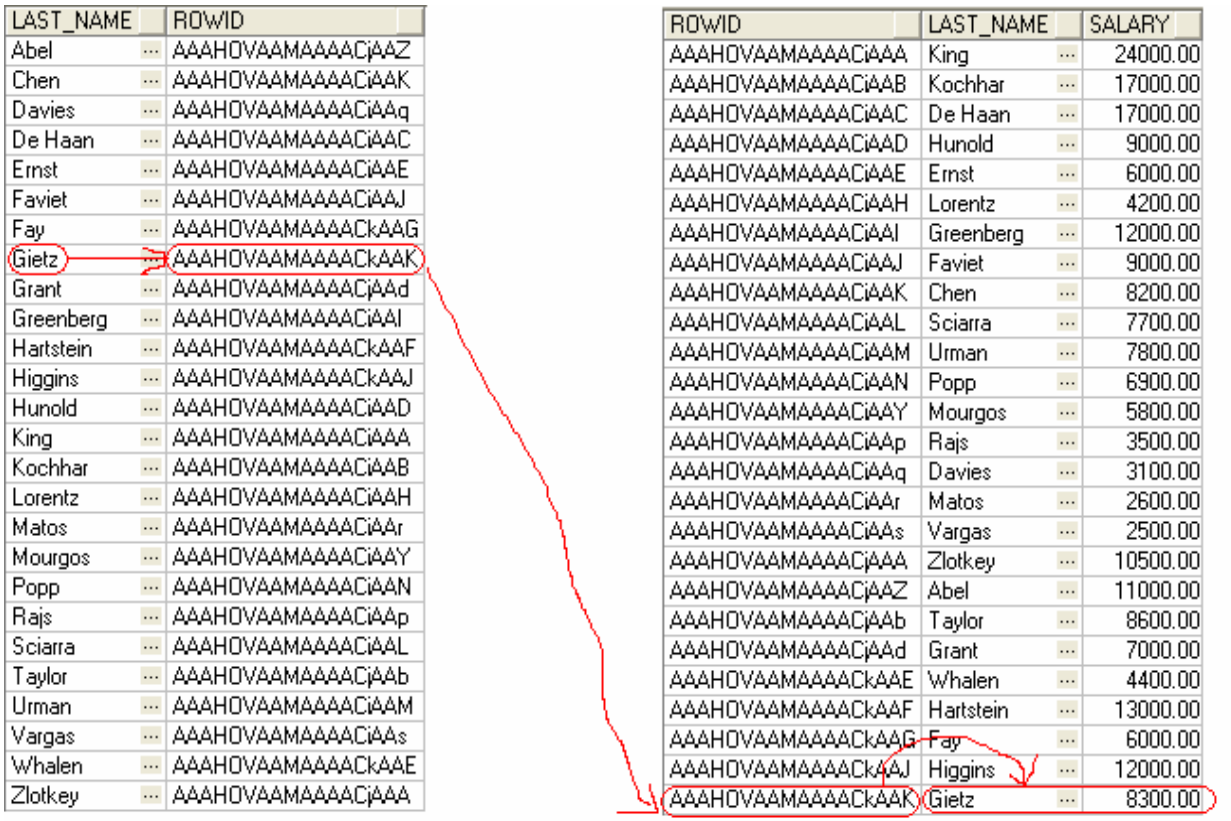
- **例10-15　删除序列student_seq删除序列student_seq**

```
DROP SEQUENCE student_seq;
```

索引

- 索引是:
  - 方案（schema）中的一个数据库对象
  - 在 Oracle数据库中用来加速对表的查询速度
  - 通过使用快速路径访问方法快速定位数据,减少了磁盘的I/O
  - 与表独立存放，但需要依附于表，是在表的基础上创建的
  - 由 Oracle数据库自动维护

索引



LAST_NAME	ROWID
Abel	AAAHOVAAMAAAACjAAZ
Chen	AAAHOVAAMAAAACiAAK
Davies	AAAHOVAAMAAAACiAAq
De Haan	AAAHOVAAMAAAACiAAC
Ernst	AAAHOVAAMAAAACiAAE
Faviet	AAAHOVAAMAAAACiAAJ
Fay	AAAHOVAAMAAAACkAAG
Gietz	AAAHOVAAMAAAACkAAK
Grant	AAAHOVAAMAAAACjAAd
Greenberg	AAAHOVAAMAAAACiAAI
Hartstein	AAAHOVAAMAAAACkAAF
Higgins	AAAHOVAAMAAAACkAAJ
Hunold	AAAHOVAAMAAAACiAAD
King	AAAHOVAAMAAAACiAAA
Kochhar	AAAHOVAAMAAAACiAAB
Lorentz	AAAHOVAAMAAAACiAAH
Matos	AAAHOVAAMAAAACiAAr
Mourgos	AAAHOVAAMAAAACiAAY
Popp	AAAHOVAAMAAAACiAAN
Rajs	AAAHOVAAMAAAACiAAp
Sciarra	AAAHOVAAMAAAACiAAL
Taylor	AAAHOVAAMAAAACjAAb
Urman	AAAHOVAAMAAAACiAAM
Vargas	AAAHOVAAMAAAACiAAs
Whalen	AAAHOVAAMAAAACkAAE
Zlotkey	AAAHOVAAMAAAACjAAA

索引

ROWID	LAST_NAME	SALARY
AAAHOVAAMAAAACiAAA	King	24000.00
AAAHOVAAMAAAACiAAB	Kochhar	17000.00
AAAHOVAAMAAAACiAAC	De Haan	17000.00
AAAHOVAAMAAAACiAAD	Hunold	9000.00
AAAHOVAAMAAAACiAAE	Ernst	6000.00
AAAHOVAAMAAAACiAAH	Lorentz	4200.00
AAAHOVAAMAAAACiAAI	Greenberg	12000.00
AAAHOVAAMAAAACiAAJ	Faviet	9000.00
AAAHOVAAMAAAACiAAK	Chen	8200.00
AAAHOVAAMAAAACiAAL	Sciarra	7700.00
AAAHOVAAMAAAACiAAM	Urman	7800.00
AAAHOVAAMAAAACiAAN	Popp	6900.00
AAAHOVAAMAAAACiAAY	Mourgos	5800.00
AAAHOVAAMAAAACiAAp	Rajs	3500.00
AAAHOVAAMAAAACiAAq	Davies	3100.00
AAAHOVAAMAAAACiAAr	Matos	2600.00
AAAHOVAAMAAAACiAAs	Vargas	2500.00
AAAHOVAAMAAAACjAAA	Zlotkey	10500.00
AAAHOVAAMAAAACjAAZ	Abel	11000.00
AAAHOVAAMAAAACjAAb	Taylor	8600.00
AAAHOVAAMAAAACjAAd	Grant	7000.00
AAAHOVAAMAAAACkAAE	Whalen	4400.00
AAAHOVAAMAAAACkAAF	Hartstein	13000.00
AAAHOVAAMAAAACkAAG	Fay	6000.00
AAAHOVAAMAAAACkAAJ	Higgins	12000.00
AAAHOVAAMAAAACkAAK	Gietz	8300.00

表

创建索引

- 创建索引有两种方式：自动或者手动
- 自动: 当在表上定义一个PRIMARY KEY 或者UNIQUE约束条件时,Oracle数据库自动创建一个对应的唯一索引.
- 手动: 用户可以创建索引以加速查询，在需要创建索引的字段上创建需要的索引。

创建一个索引

- 在一列或者多列上创建索引.

```
CREATE INDEX indexname
ON table (column[, column]...);
```

- 下面的索引将会提高对EMPLOYEES表基于LAST_NAME 字段的查询速度.

```
CREATE INDEX          emp_last_name_idx
ON  employees(last_name);
Index created.
```

测试索引

- 创建测试环境表。

```
CREATE TABLE e1 AS SELECT * FROM employees;
INSERT INTO e1 SELECT * FROM e1;--多次运行
UPDATE e1 SET employee_id=ROWNUM;  --更新所有记录的
    employee_id，以使其数值唯一
commit;--提交
```

- 测试无索引检索时间。

```
set timing on;   --设置环境变量timing
SELECT last_name,salary FROM e1
WHERE employee_id=210000; --没有索引的情况下做测试
已用时间:  00: 00: 06.05 –没有索引的时候用时6秒
```

测试索引（续）

- 测试建索引后检索时间。

```
CREATE INDEX e1_id ON e1(employee_id); --创建索引
SELECT last_name,salary FROM e1
WHERE employee_id=210000; --再次在有索引的情况下做测试
已用时间:  00: 00: 00.00 –测试结果，几乎没有消耗什么时间
set timing off; --取消时间统计
```

适合创建索引情况

- 查询列的数据范围很广泛
- 查询列中包含大量的NULL值
- WHERE条件中的列或者多表连接的列适合创建索引
- 欲查询的表数据量很大，而且大多数的查询得到结果集的数量占总记录量的2%～4%

不适合创建索引的情况

- 很小数据量的表
- 在查询中不常用来作为查询条件的列
- 查询最终得到的结果集很大
- 频繁更新的表（索引对于DML操作是有部分负面影响的）
- 索引列作为表达式的一部分被使用时（比如常查询的条件是SALARY*12，此时在SALARY列上创建索引是没有效果的）

删除索引

- 删除索引的语法

```
DROP INDEX index;
```

- 删除索引后，索引中的数据及定义被删除，索引所占的数据空间被释放，但表中的数据仍然存在。
- 常用与索引相关的数据字典视图有：
  - USER_INDEXES：用户的索引对象的定义，包含索引的名字和索引的一些相关属性（比如唯一性等）
  - USER_IND_COLUMNS：用户索引对象的列的定义信息，会列出索引名字，表名字，和列的名字等。

同义词

- 同义词（Synonyms）是指向数据库对象（如：表、视图、序列、存储过程等）的数据库指针。
- 使用同义词好处:
  - 可以简化对数据库对象的访问。
  - 方便对其他用户表的访问。
  - 简化过长的对象名称。
  - 节省大量的数据库空间，对不同用户的操作同一张表没有多少差别。
  - 扩展的数据库的使用范围，能够在不同的数据库用户之间实现无缝交互
  - 同义词可以创建在不同一个数据库服务器上，通过网络实现连接。

创建同义词

- 创建同义词的语法如下：

```
CREATE [PUBLIC] SYNONYM 同义词
FOR  [schema.]对象名;
```

- 同义词两种类型：
  - 私有（PRIVATE）。是在指定的方案中创建的，并且只允许拥有它的方案访问
  - 公有（PUBLIC）。由PUBLIC方案所拥有，所有的数据库方案都可以引用他们。

创建和删除同义词

- 创建employees表的别名。

```
CREATE SYNONYM  s_emp
FOR  hr.employees;
```

- 删除同义词。

```
DROP SYNONYM s_emp;
```

- 只有数据库管理员才拥有公有同义词的创建和删除权限。

本章小结

- 视图的概念及应用
- 序列的概念及应用
- 索引的概念及应用
- 同义词的概念及应用。

练习

- 1.试创建视图v_emp_80，包含80号部门的员工编号，姓名，年薪列。
- 2.从视图v_emp_80中查询年薪在12万元以上的员工的信息。
- 3.创建试图v_dml，包含部门编号大于100号的部门的信息。
- 4.从视图v_dml插入如下记录:部门编号360,部门名称AAA,管理者编号101,区域编号1700.

练习（续）

- 5.从视图v_dml中删除300号以上的部门信息。
- 6.给表employees创建同义词em。
- 7.使用同义词em统计各部门员工的人数。

第十一章　PL/SQL概述

Neusoft 东软

本章要点

- PL/SQL基本概念
- PL/SQL的变量
- PL/SQL控制结构
- 与Oracle交互

PL/SQL概述

- 什么是PL/SQL
  - PL/SQL也是一种程序语言。PL 是Procedural Language的缩写
  - PL/SQL是Oracle数据库对SQL语句的扩展，增加了编程语言的特点
  - 数据操作和查询语句被包含在PL/SQL代码的过程性单元中，经过逻辑判断、循环等操作完成复杂的功能或者计算

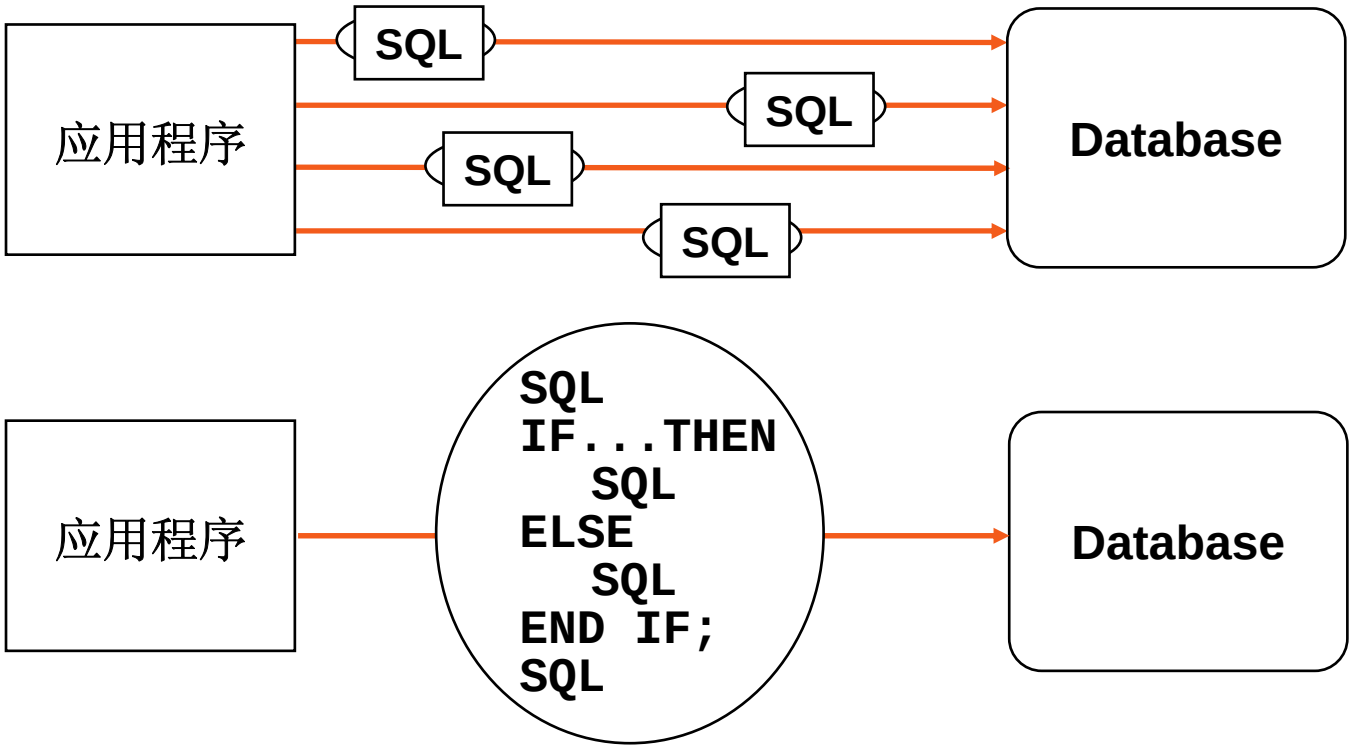
PL/SQL示例

```
CREATE OR REPLACE PROCEDURE get_bonus
( p_empno in number) IS
v_bonus number(9,2);    --奖金数目
BEGIN
/*  根据员工的工资计算奖金并把结果插入到测试表中*/
SELECT sal into v_sal FROM emp WHERE empno=p_empno ;
v_bonus:=case
            when v_sal<1000 then 0.2* v_sal
            when v_sal between 1000 and 2000 then 0.4* v_sal
            else 0.5* v_sal
      end ;
   INSERT INTO testpl1(empno,sal,bonus)
     values(p_empno,v_sal,v_bonus);
     COMMIT;
END get_bonus;
```

PL/SQL的优点

- 改善了性能。PL/SQL以整个语句块发给服务器，这个过程在单次调用中完成，降低了网络拥挤。而如果不使用PL/SQL，每条SQL语句都有单独的传输交互，在网络环境下占用大量的服务器时间，同时导致网络拥挤。
- 可重用性。PL/SQL能运行在任何ORACLE环境中（不论它的操作系统和平台），在其他ORACLE能够运行的操作系统上无需修改代码。
- 模块化。每个PL/SQL单元可以包含一个或多个程序块，程序中的每一块都实现一个逻辑操作，从而把不同的任务进行分割，由不同的块来实现，块之间可以是独立的或是嵌套的。

PL/SQL提高了性能



PL/SQL块结构

- DECLARE -- 可选部分
  - 变量、常量、游标、用户定义异常声明
- BEGIN -- 必要部分
  - SQL语句
  - PL/SQL语句
- EXCEPTION --可选部分
  - 程序出现异常时，捕捉异常并处理异常
- END; -- 必要部分

PL/SQL块例子

```
DECLARE
   v_dept_id      employees.department_id%TYPE;
BEGIN
  SELECT department_id INTO v_dept_id FROM employees
    WHERE employee_id = 100;
  DELETE departments WHERE department_id = v_dept_id;
  COMMIT;
EXCEPTION
  WHEN OTHERS THEN
    ROLLBACK;
    INSERT INTO exception_table (message)
          VALUES ('Some error occurred in the database.');
    COMMIT;
END;
```

PL/SQL块类型

- PL/SQL块按存储方式及是否带名称等分为以下几种类型:
  - 匿名块：一般在要运行的应用中说明，运行时传递给PL/SQL引擎处理，只能执行一次，不能被存储在数据库中。
  - 过程，函数和包（Procedure，Function & Package）：是命名为PL/SQL块，被存储在数据库中，能够被多次执行，可以用外部程序来显示执行。
  - 触发器（Trigger）：是命名为PL/SQL块，被存储在数据库中，能够被多次执行，当相应的触发事件发生时自动被执行。

变量声明

- PL/SQL中可使用标识符来声明变量，常量，游标，用户定义的异常等，并在SQL语句或过程化的语句中使用。
- 标识符的命名和Oracle对数据库对象的命名原则相同，如不能超过30个字符长，第一个字符必须为字母等。
- 对标识符的命名最好遵循项目组相关命名规范，如
  v_LastName
  c_BirthDay
- 也可以把数据类型缩写加入标识符名称中。如
  v_vcLastName
  c_dtBirthDay

变量声明

- 在声明部分声明和初始化变量
- 在执行部分为变量赋新值，在表达式中使用变量
- 通过参数把值传递到PL/SQL 块中
- 通过输出变量或者参数将值传出PL/SQL块

声明变量和常量：语法

```
identifier [CONSTANT] datatype [NOT NULL] [:= |
    DEFAULT expr];
```

- 定义的标示符名称应遵循命名规则
- 在声明常量和变量的时候可以为其设置初始化值
- 可以使用赋值运算符(:=)或者DEFAULT保留字来初始化标识符
- 在声明时，每行只能声明一个标识符。
- 变量名称不要和数据库中表名或字段名相同，
- 声明常量时必须加关键字CONSTANT

PL/SQL中的变量类型

- 简单变量
- 复合（组合）变量
- 外部变量

简单变量

- 简单变量不包括任何组件，只能保存一个值
- 基本类型：
  - BINARY_INTEGER
  - NUMBER [(precision, scale)]
  - CHAR [(maximum_length)]
  - VARCHAR2(maximum_length)
  - DATE
  - LONG
  - LONG RAW
  - CLOB / BLOB / BFILE
  - BOOLEAN

简单变量的声明

例:

```
v_nation        CHAR(1);
v_count         BINARY_INTEGER := 0;
v_sal           NUMBER(9,2) := 0;
v_hire_date     DATE := SYSDATE ;
v_flag          BOOLEAN NOT NULL := TRUE;
c_PI            CONSTANT   NUMBER(6,5) := 3.14159;
```

%TYPE 属性

- 通过%TYPE属性声明一个变量，实际上就是参照变量或者表中字段的类型作为变量的类型，并且保持同步。变量将遵循下面的类型声明:
  - 已经声明过的变量的类型
  - 数据库中的表的字段的类型
- 可以作为%TYPE的前缀的可以是
  - 数据库表和列
  - 前面声明的变量名称
- PL/SQL在运行程序时确定变量的数据类型和大小

%TYPE属性

```
v_ename          emp.ename%TYPE;
v_hiredate       emp.hiredate%TYPE;
v_student        NUMBER(7,2);
v_teacher        v_student%TYPE := 10;
```

- 使用%TYPE 属性的好处：
  - 在编程时，可以不去查询数据库中字段的数据类型
  - 数据库中字段的数据类型可能被改变
  - 为了和前面的变量的类型始终保持一致

复合数据类型

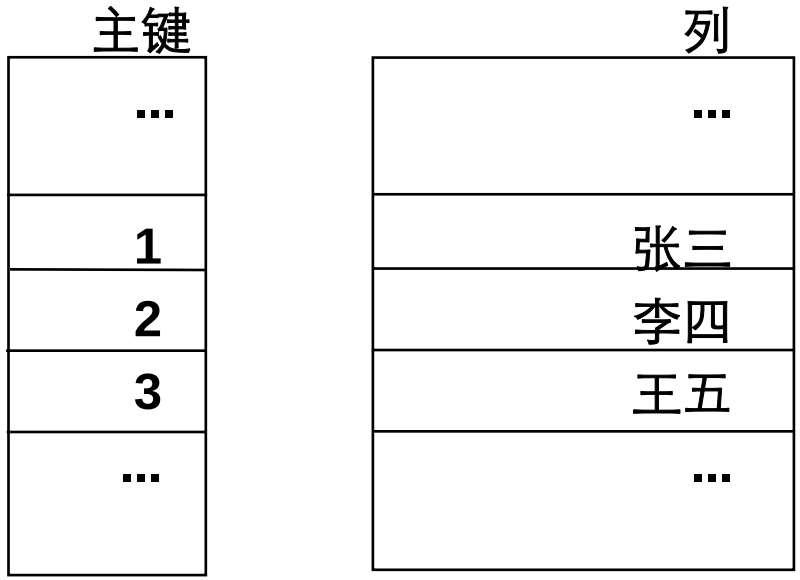
- 复合组合变量也叫做组合变量，在复合变量中包括多个内部组件，每个组件都可以单独存放值，因此一个复合变量可以存放多个值。
- 复合变量类型不是数据库中已经存在的数据类型，因此复合变量在声明类型之前，首先要先创建复合类型，复合类型创建后可以多次使用，以便定义多个复合变量。

- 复合数据类型:
  - PL/SQL TABLES 表类型
  - PL/SQL RECORDS 记录类型

复合数据类型- TABLE

- TABLE类型结构类似于其他语言中的数组类型，由两个组件组成：
  - 数据类型为BINARY_INTEGER的主键
  - 数据类型为一个简单类型的列
- Table类型结构很像数组，但是不像数组一样实现定义数组长度， Table类型没有长度限制，可以动态增长。

PL/SQL TABLE 结构



声明一个PL/SQL TABLE

- 语法:

```
TYPE type_name IS TABLE OF
    {column_type | variable%TYPE
    | table.column%TYPE} [NOT NULL]
    | table.%ROWTYPE
    [INDEX BY BINARY_INTEGER];
identifier          type_name;
```

声明一个PL/SQL TABLE

- 例子:

```
TYPE name_table_type IS TABLE OF VARCHAR2(16)
   INDEX BY BINARY_INTEGER;
v1_name    name_table_type;
v2_name    name_table_type;
```

PL/SQL RECORDS

- 包含一个或几个组件，每个组件称为一个域，域的数据类型可以是标量、RECORD或PL/SQL TABLE
- 类似于3GL中的记录结构
- 在使用RECORD变量时把多个域的集合作为一个逻辑单元使用，对记录类型变量赋值或则引用，都需要使用“记录变量名.域名”的方式来实现。
- 主要用于从表中取出查询到的行数据

PL/SQL RECORD结构

Field1 (数据类型)	Field2 (数据类型)	Field3 (数据类型)

声明PL/SQL RECORD

- 语法:

```
TYPE type_name IS RECORD
    (field_declaration[, field_declaration]…);
identifier          type_name;
```

- 例子:

```
    TYPE emp_record_type IS RECORD
       (last_name        VARCHAR2(25),
        first_name       VARCHAR2(25),
        sal        NUMBER(8));
     emp_record          emp_record_type;
```

%ROWTYPE属性

- 与%TYPE作用类似，用于定义不确定的变量类型
- 变量类型将定义为由数据库的表的字段集合构成的RECORD类型
- %ROWTYPE的前缀是数据库的表名，或者另一个已经定义好的RECORD变量
- RECORD中的域，与表的字段的名称和数据类型完全相同

%ROWTYPE 属性：优点

- 数据库中表字段的数据类型和数目可能不知道
- 数据库中表字段的个数和数据类型会在运行中改变

```
dept_records_dept%ROWTYPE;
emp_records_emp%ROWTYPE;
```

PL/SQL操作符

- 算术操作符：+、-、*、/、**
- 比较操作符：=、>、<、<= 、>=、<>、!=
- 关系操作符：IS NULL、IN、LIKE等
- 逻辑操作符：AND、OR、NOT
- 其他操作符：||、:=

PL/SQL中函数的应用

- 过程性语句中可以使用下面函数：
  - 单行数值函数：MOD，ROUND，TRUNC，CEIL，FLOOR等函数
  - 单行字符函数：CHR，CONCAT，INITCAP，LENGTH，LOWER，LPAD，LTRIM，REPLACE，RPAD，RTRIM，SUBSTR，TRIM，UPPER等函数
  - 日期函数：ADD_MONTHS，LAST_DAY，MONTHS_BETWEEN，NEXT_DAY，ROUND，SYSDATE，TRUNC等函数
  - TIMESTAMP函数
  - GREATEST，LEAST函数
  - 转换函数：TO_CHAR，TO_DATE，TO_NUMBER等函数

PL/SQL中函数的应用

- 过程性语句不可直接使用下面函数:
  - DECODE
  - 组函数: AVG， MIN， MAX， COUNT， SUM，STDDEV，VARIANCE等函数

PL/SQL 块语法规则

- 语句可以写在多行
- 词汇通过空格分隔
- 每条语句必须通过分号结束
- 标识符的规定:
  - 最多可以包含30个字符
  - 不能包含保留字，若有使用双引号括起来
  - 必须以字母字符开始
  - 不能与数据库的表或者列名称相同

PL/SQL 注释

- 添加注释可以提高程序的可读性，使程序更加易于理解，也更易于将来的维护。
- 注释可以是
  - /* 和*/之间的多行注释
  - 单行注释，以 — 开始
- 建议以下地方应使用注释：
  - 程序头部：说明程序的主要功能，程序的作者，创建日期，修改日期及本次修改内容，各主要输入参数，输出参数的说明。
  - 声明部分：说明主要变量，常量，游标等。
  - 程序体中重要的算法：说明主要的算法，思路。

PL/SQL条件语句（IF语句）

- IF语句语法：

```
IF condition THEN
    statements;
[ELSIF condition THEN
    statements; ]
[ELSE
    statements; ]
END IF;
```

- 最多允许一个ELSE子句
- 判断条件可以是一个或者多个条件的组合，通过连接操作符（AND/OR/NOT）连接在一起

IF-THEN-ELSE 语句

- 例，在3个数中找出最大值

```
. . .
      IF (a > b)  and  ( a > c) THEN
         x := a;
      ELSE
          x := b;
          IF c > x THEN
                x := c;
          END IF;
      END IF;;
. . .
```

IF-THEN-ELSIF 语句

- 例，根据工资计算税并返回：

```
. . .
   IF v_sal > =3000 THEN
      RETURN (.2 * v_sal );
   ELSIF v_sal >= 2000 THEN
      RETURN (.1 * v_sal);
   ELSE
      RETURN 0;
   END IF;
. . .
```

构造逻辑条件

- 使用IS NULL操作符，处理空值
- 任何包含空值的数学表达式，其值为NULL
- 在字符表达式中出现的空值，会被作为空串处理

逻辑关系

- 使用比较操作符构造布尔条件

AND	*TRUE*	*FALSE*	*NULL*
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

OR	*TRUE*	*FALSE*	*NULL*
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

NOT	
TRUE	FALSE
FALSE	TRUE
NULL	NULL

PL/SQL循环语句

- 循环用于多次执行某些语句。
- 循环控制主要包括下列语句:
  - 简单循环
  - WHILE循环
  - FOR循环

简单循环

- 简单循环的特点，循环体至少执行一次，LOOP语句的语法如下：

```
LOOP
    语句体;
    [EXIT | EXIT WHEN 条件;]
END LOOP
```

- 在使用LOOP语句时必须使用EXIT语句，强制循环结束，否则将死循环。

简单循环

- 根据x值计算y值

```
X:=1;
 LOOP
     X:=X+1;
     IF X>10 THEN
         EXIT;
     END IF;
  END LOOP;
  Y:=X;
```

WHILE循环

- 使用WHILE 循环，只要条件满足就执行循环：

```
WHILE 条件 LOOP
 语句体;
 END LOOP;
```

- 有可能一次也不执行

WHILE循环

- 根据x值计算y值

```
X:=1;
WHILE X<=10 LOOP
X:=X+1;
END LOOP;
Y := X;
```

FOR循环

- 使用FOR循环，循环执行指定的次数

```
FOR counter IN [REVERSE]
start_range..end_range LOOP
  语句体;
  END LOOP;
```

REVERSE:正常计数器从小到大递增，使用REVERSE将使计数器从大到小递减。

FOR循环

- 根据x值计算y值

```
FOR v_counter in 1..10 loop
    X := x + 1;
end loop
y := x;
```

与Oracle交互

- 在PL/SQL程序块中可以使用各种SQL命令与Oracle进行交互，但是使用的方法根据命令不同也各不相同。
- 与Oracle交互主要涉及以下命令:
  - SELECT语句的使用
  - DML语句的使用
  - 事务处理语句的使用
  - 本地动态SQL： EXECUTE IMMEDIATE命令

PL/SQL中SELECT语句的使用

- 在PL/SQL中通过SELECT从数据库中检索数据:

```
SELECT  [DISTICT|ALL]{*|column[,column,...]}
INTO (variable[,variable,...] |record)
FROM {table|(sub-query)}[alias]
[WHERE 子句]
```

- 必须使用INTO子句
- 查询必须并且只能返回一行
- 可以使用完整的SELECT 语法

PL/SQL中SELECT语句的使用

- 使用Dbms_output包时，要设置这个环境变量serveroutput 。
- 使用set serveroutput on 命令设置环境变量serveroutput为打开状态，从而使得PL/SQL程序能够在SQL*plus中输出结果。
- SELECT INTO语句中可以包含分组函数
- SELECT INTO语句中可以使用参数或变量传递值

查询数据

- 取出员工号为113的员工的姓名和工资

```
DECLARE
      v_LastName employees.last_name%TYPE;
      v_Salary employees.salary%TYPE;
BEGIN
      SELECT last_name, salary
            INTO v_LastName, v_Salary
          FROM employees
           WHERE employee_id = 113;
      DBMS_OUTPUT.PUT_LINE(
      v_LastName || '的工资是' ||  to_char(v_salary ));
END;
```

查询数据的错误

```
DECLARE
      v_LastName employees.last_name%TYPE;
      v_Salary employees.salary%TYPE;
      employee_id employees.employee_id%TYPE;
BEGIN
      employee_id:=113;
      SELECT last_name, salary  INTO v_LastName, v_Salary
          FROM employees
           WHERE employee_id = employee_id;
      DBMS_OUTPUT.PUT_LINE(
      v_LastName ||'的工资是' ||  to_char(v_salary ));
END;
```

查询异常

- PL/SQL 中的SELECT 语句必须返回且只返回一行
- 如果检索到了零行或多于一行，将会引起异常
- SELECT 常见的异常：
  - TOO_MANY_ROWS
  - NO_DATA_FOUND

PL/SQL中DML语句的使用

- 在PL/SQL中，DML语句的使用和在SQL语句中的用法是相同的。
- 通过使用DML 命令，可对数据库中表的数据实现下列操作:
  - INSERT
  - UPDATE
  - DELETE

INSERT语句

- 向部门表中插入编号为300的部门信息

```
BEGIN
  INSERT INTO departments
    (department_id,department_name)
  VALUES (300,'HR');
END;
```

UPDATE语句

- 将300号部门名称改为“Human Resource”。

```
BEGIN
    UPDATE departments
     SET  department_name = ‘Human Resource’
     WHERE department_id = 300;
END;
```

DELETE语句

- 从部门表中删除300号部门记录

```
BEGIN
    DELETE FROM departments
    WHERE department_id = 300;
END;
```

PL/SQL中事务处理语句的使用

- 在PL/SQL中事务控制语句的使用方法和特性与SQL*PLUS中相同:
  - 事务开始于COMMIT或ROLLBACK后的第一个DML语句
  - 使用COMMIT 和ROLLBACK SQL 命令显式的结束事务
  - 可以使用其它的命令开始事务，锁定数据，包括SELECT...FOR UPDATE 和LOCK TABLE

事务控制

- 在下面的PL/SQL 块中，执行事务处理命令

```
BEGIN
      INSERT INTO temp(x, y)
        VALUES (1,  ‘aaa');
      SAVEPOINT a;
      INSERT INTO temp(x,y)
        VALUES (2, ‘bbb');
      SAVEPOINT b;
      ROLLBACK TO SAVEPOINT a;
      COMMIT;
END;
```

本地动态SQL

- 在PL/SQL中对于SQL语言中的DDL和DCL语句不能够进行直接使用，而是需要使用动态SQL来实现相关语句的执行。
- 实现动态SQL有两种方式：调用DBMS_SQL包和本地动态SQL。本地动态SQL是通过执行EXECUTE IMMEIDATE命令来实现的。
- 通过EXECUTE IMMEDIATE可以在PL/SQL中直接执行DDL和DCL语句等。

本地动态SQL

- 本地动态SQL语法：

```
EXECUTE IMMEDIATE ‘动态SQL’
[INTO 变量列表]
[USING 绑定参数列表]
[{RETURNING | RETURN} INTO 输出参数列表];
```

- 语法说明：
  - 动态SQL是指DDL和带参数的SELECT及DML。
  - 绑定参数列表为传入参数值列表，即其类型为IN类型，在执行时与动态SQL中的参数（即占位符，可以理解为形参）进行绑定。
  - 输出参数列表为动态SQL执行后返回的参数列表。

本地动态SQL示例

- 使用动态SQL创建部门表dept。

```
BEGIN
    /*动态SQL为DDL语句*/
    EXECUTE IMMEDIATE 'create table dept(
    deptno number,dname varchar2(10), loc number )';
  END;
```

带参数的动态SQL

```
CREATE OR REPLACE PROCEDURE test_sql
 (p_no number)
IS
   v_name varchar2(10);
   v_loc number;
BEGIN
 /*动态SQL为带参数查询语句*/
    EXECUTE IMMEDIATE ' select dname,loc from dept   where
    deptno=:1 '
    INTO v_name,v_loc
    USING p_no;
```

带参数的动态SQL(续)

```
        DBMS_OUTPUT.PUT_LINE(v_name ||'所在地为
    '||to_char(v_loc));
EXCEPTION
    WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('没有符合条件记录!');
END test_sql;
```

- 占位符":1"相当于函数的形式参数，使用"： "作为前缀。使用USING语句将p_no在执行时将占位符":1"给替换掉。

带参数的动态SQL

- 存储过程调用结果如下：

```
SQL> set serveroutput on
SQL> exec test_sql(10);
SALES所在地为1500。
```

本章小结

- PL/SQL概念
- PL/SQL块结构
- 变量声明
- 控制结构
- 与Oracle交互

练习

- 1.从部门表中找到最大的部门号，将其输出到屏幕
- 2.在部门表中插入一个新部门
- 3.将练习2中的部门从部门表中删除
- 4.定义变量代表员工表中的员工号，根据员工号获得员工工资，如果工资小于4000，输出到屏幕上的内容为员工姓名和增涨10%以后的工资，否则输出到屏幕上的内容为员工姓名和增涨5%以后的工资

第十二章　游标

Neusoft 东软

本章要点

- 游标的定义
- 隐式游标
- 显式游标
- 带参数的游标

游标的概念

- 游标是映射在结果集中一行数据上的位置实体，有了游标，用户就可以使用游标来访问结果集中的任意一行数据，提取当前行的数据后，即可对该行数据进行操作。
- 游标分为显式游标和隐式游标：
  - 当可执行部分发生一个SQL语句时，PL/SQL建立一个隐式游标，它定义SQL标识符，PL/SQL自动管理这一游标。
  - 显式游标由程序员显式说明及控制，用于从表中取出多行数据，并将多行数据一行一行单独处理

隐式游标：SQL 游标

- 隐式游标被用于描述执行的SQL命令
- 游标具有属性，可以使用游标属性测试SQL 命令的结果。
- 游标属性有4种，对显式游标和隐式游标都有效使用时，需要以游标名称作为前缀，使用隐式游标方法及含义如下：
  - SQL%ROWCOUNT 受SQL影响的行数
  - SQL%FOUND Boolean 值，是否还有数据
  - SQL%NOTFOUND Boolean 值，是否已无数据
  - SQL%ISOPEN 总是为FALSE
- 隐式游标使用时，只能取出最后一条执行的SQL语句的隐式游标属性。

隐式SQL 游标属性

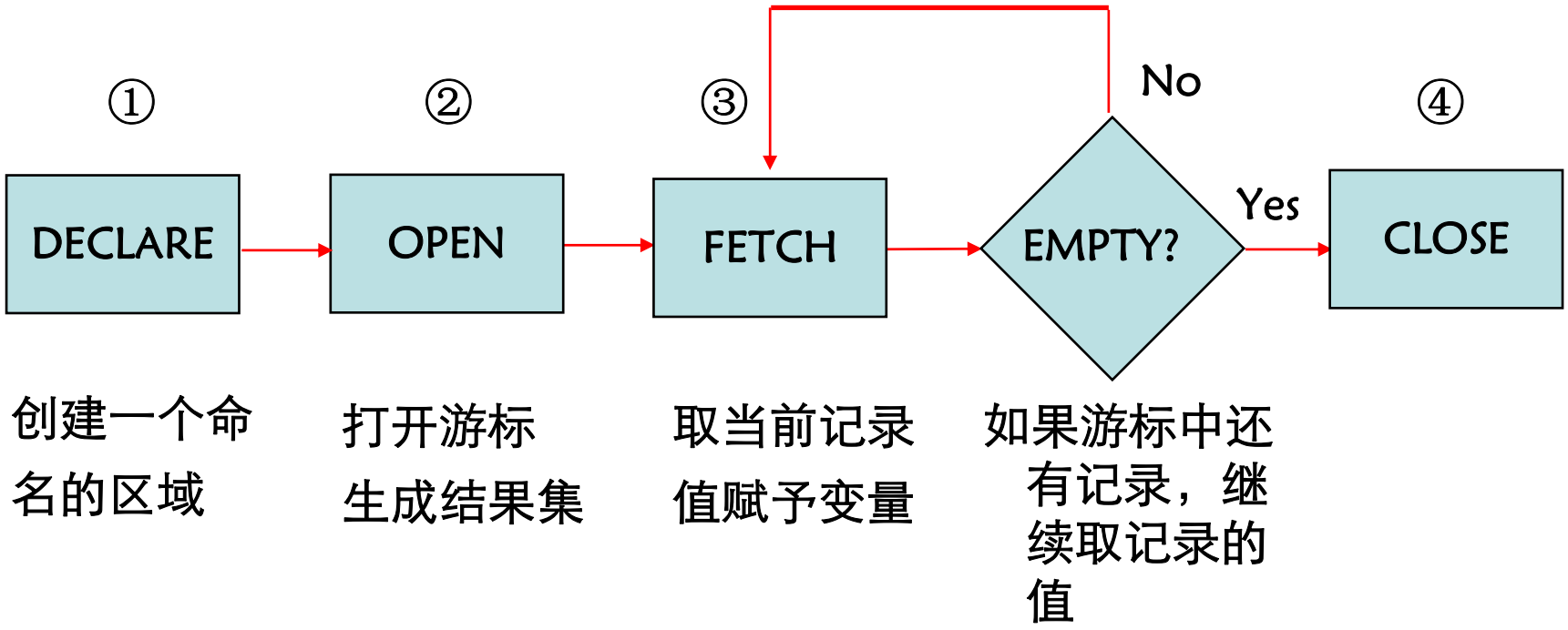
- 从employees表中删除特定部门的员工。显示删除的员工人数

```
DECLARE
  v_rows_deleted  NUMBER;
BEGIN
   DELETE employees WHERE  department_id = 80;
   v_rows_deleted := SQL%ROWCOUNT;
   DBMS_OUTPUT.PUT_LINE(
                  v_rows_deleted||' 条员工信息被删除.');
END
```

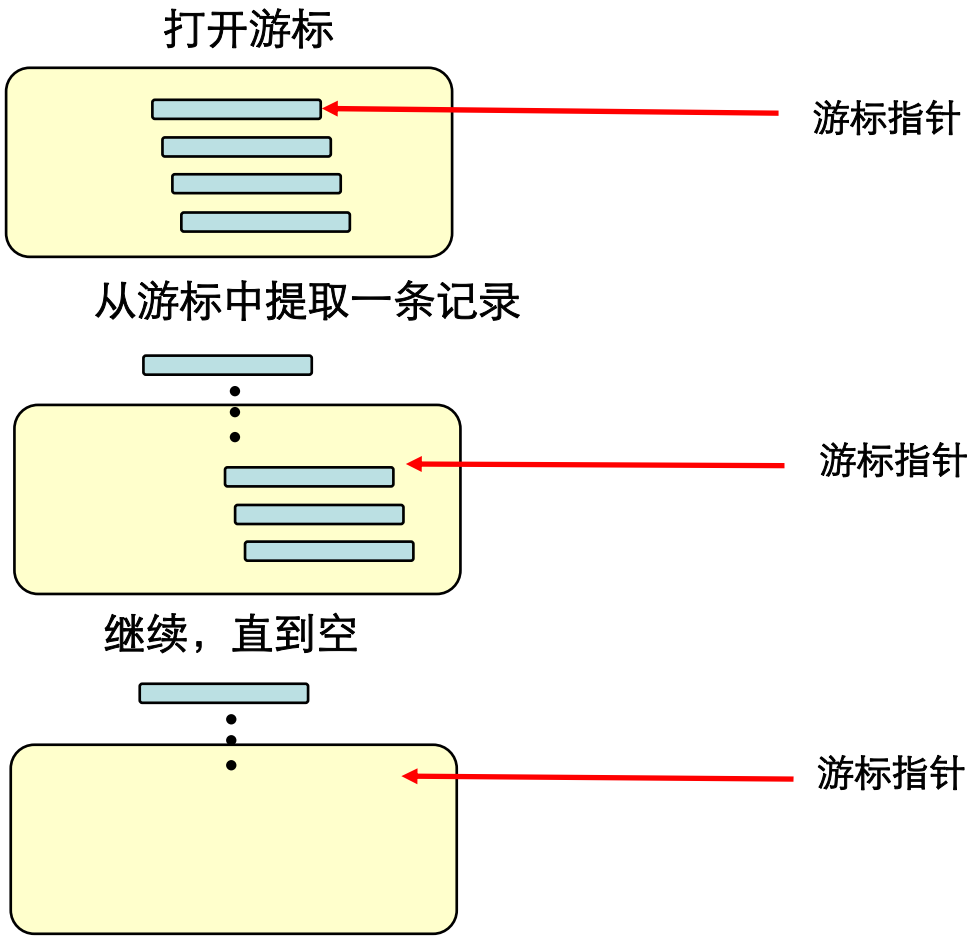
显式游标

- 显式游标可以用于暂存查询取出的多行数据，然后一行一行的进行处理。
- 显式游标按行处理查询返回的多行数据，而SELECT…INTO只能取出一行数据。
- 在PL/SQL块中通过循环手动控制游标的多行取操作，直到取出游标中所有的数据为止。

显式游标操作步骤



控制显式游标



游标的声明

```
DECLARE
CURSOR cursor_name IS
        select_statement;
```

- 在游标声明中，SELECT子查询中不要包含INTO 子句。
- SELECT语句可以从表或视图中选取数据，也可以从多个表或视图中选取数据。

声明游标

- 取得满足某一工资标准的员工信息:

```
DECLARE
  . . .
  CURSOR curEmp IS SELECT ename,salary
   FROM emp
   WHERE salary>2000;
   . . .

BEGIN
. . .
```

打开游标

- 使用游标之前应首先打开游标，打开游标就是执行游标定义中的子查询语句。

```
OPEN cursor_name;
```

- 不论查询有没有返回记录，都不会引起异常

从游标读取数据

- 使用FETCH命令从游标中取得数据。每次提取数据后，游标会自动指向结果集的下一行

```
FETCH cursor_name INTO variable[,variable,...];
```

- 变量应该与游标字段个数相同
- 变量与字段顺序也应该一一对应
- 可使用游标的属性测试判断游标是否还包含更多的数据行

从游标读取数据

- 逐行取得某个部门的员工信息:

```
FETCH curEmp
  INTO v_name, v_sal;
```

游标和记录变量

- 把提取出来的数据存入PL/SQL RECORD变量，便于对结果集的处理

```
CURSOR curEmp IS
   SELECT employee_id, salary, hire_date
   FROM     employees
   WHERE  department_id = 100;
   emp_record          curEmp %ROWTYPE;
BEGIN
   OPEN curEmp ;
   FETCH curEmp INTO emp_record;
. . .
```

关闭游标

- 关闭游标是游标操作的最后一步，可以释放游标所占用的资源，关闭游标之后，关于游标的存取等操作都不能再进行。

```
CLOSE cursor_name;
```

- 如果需要，必须重新打开游标
- 一旦游标已经关闭，不能再提取数据

提取多行数据

- 可使用循环从一个显式游标中取出多行数据
- 每次重复，fetch语句取出一行数据
- 通过使用%NOTFOUND属性，判断上一次的取操作是否成功取到数据
- 也可以通过%FOUND属性来控制循环取操作

显式游标属性

- 通过使用游标属性，获取游标的状态信息

属性	类型	描述
%ISOPEN	Boolean	如果游标打开，则为TRUE
%FOUND	Boolean	一直为TRUE，直到最近提取没有取回行记录
%NOTFOUND	Boolean	如果最近的提取没有返回一条记录，则为TRUE
%ROWCOUNT	Number	返回到当前位置为止游标读取的记录行数。

%ISOPEN 属性

- 只有在游标打开时，才能从游标中取数据
- 在执行提取操作前，可以使用%ISOPEN 游标属性测试游标是否已打开

```
IF curEmp %ISOPEN THEN
   FETCH curEmp INTO v_name, v_sal;
ELSE
   OPEN curEmp;
END IF;
. . .
```

%NOTFOUND 和%ROWCOUNT

- 使用%ROWCOUNT 游标属性，准确获取取出的行数
- 使用%NOTFOUND 游标属性，确定是否已取出所有数据

```
LOOP
  FETCH curEmp
INTO v_name, v_sal;
      EXIT WHEN curEmp %ROWCOUNT > 5
   OR curEmp %NOTFOUND;
  v_sal := v_sal + 500;
. . .
END LOOP;
```

典型游标FOR 循环

- 游标for循环是一种快捷处理游标的方式，它使用for循环依次读取结果集中的行数据，当for循环开始时，游标自动打开，每循环一次系统自动读取游标当前行的数据，当退出for循环时，游标被自动关闭。
- 使用游标for循环的时候不需要也不能使用open语句，fetch语句和close语句，否则会产生错误。

典型游标FOR循环

- 游标FOR循环语法

```
FOR record_name IN cursor_name LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

- 不用声明记录类型变量，Oracle会隐式声明
- FOR循环中的游标不需要执行OPEN、FETCH和CLOSE操作，否则会出错。

游标FOR循环示例

```
DECLARE
    v_LastName employees.last_name%TYPE;
    v_Salary employees.SALARY%TYPE;
    CURSOR curEmp  IS SELECT last_name,salary
    FROM employees WHERE salary>6000;
BEGIN
   FOR curEmployee IN curEmp LOOP
      v_LastName := curEmployee.last_name;
      v_Salary := curEmployee.salary;
      DBMS_OUTPUT.PUT_LINE(v_LastName ||'的工资
      是'||to_char(v_Salary));
  END LOOP;
END;
```

带参数的游标

- 当游标打开时，通过传入不同的参数，生成最终的查询结果
- 带参数游标的语法：

```
CURSOR cursor_name[(parameter[,parameter],...)] IS
    select_statement;
```

定义参数语法：

```
Parameter_name [IN] data_type[{:=|DEFAULT} value]
```

- 在调用时，通过给定不同的参数得到不同的结果集

```
OPEN cursor_name[value[,value]....];
```

带参数游标示例

- 传递部门编号和工作职务到游标中

```
CURSOR emp_cur
   (v_dept NUMBER, v_job VARCHAR2) IS
  SELECT   last_name, salary, hire_date
  FROM     employees
  WHERE    department_id = v_dept
  AND      job_id = v_job;
```

带参数游标示例

```
DECLARE
   CURSOR curDept IS SELECT *  FROM  departments
        ORDER BY  department_id;
   CURSOR curEmp (p_dept VARCHAR2) IS
        SELECT   last_name,salary FROM employees
        WHERE  department_id=p_dept  ORDER BY last_name;
        r_dept   departments %ROWTYPE;
        v_ename  employees.last_name%TYPE;
        v_salary  employees.salary%TYPE;
        v_tot_salary  employees.salary%TYPE;
BEGIN
   OPEN curDept;
   LOOP
         FETCH curDept INTO r_dept;
         EXIT WHEN curDept%NOTFOUND;
```

带参数游标示例

```
DBMS_OUTPUT.PUT_LINE('部门:'||  r_dept.DEPARTMENT_ID||'-
    '||r_dept.DEPARTMENT_NAME);
            v_tot_salary:=0;
            OPEN curEmp(r_dept.DEPARTMENT_ID);
            LOOP
                FETCH curEmp INTO v_ename,v_salary;
                EXIT WHEN curEmp%NOTFOUND;
                v_tot_salary:=v_tot_salary+v_salary;
            END LOOP;
           CLOSE curEmp;
           DBMS_OUTPUT.PUT_LINE('部门总工资:'|| v_tot_salary);
      END LOOP;
      CLOSE curDept;
    END;
```

本章小结

- 游标的概念
- 隐式游标
- 显式游标
- 游标的属性
- 游标for循环
- 带参数的游标

练习

- 1.在屏幕上输出工资最高的前5名员工姓名，参加工作时间，工资
- 2.把参加工作时间在1995年之后的员工姓名（first_name,last_name）,参加工作时间显示在屏幕上
- 3.创建一个新表dept_test,包含字段部门号，部门名称，利用游标遍历部门表，把部门表中的部门号，部门名取出插入到表dept_test中
- 4.工资大于5000的员工姓名(last_name)在"A"到"K"之间的合成一个字符串,在"L"到"M"之间合成一个字符串,在"N"到"Z"之间合成一个字符串，分别在屏幕上输出

第十三章　异常处理

Neusoft 东软

本章要点

- 异常的基本概念
- 异常的处理
- 异常的传播

什么是异常

- PL/SQL用异常和异常处理器来实现错误处理
- Oracle中出现错误的情形通常分为编译时错误（compile-time error）和运行时错误(run-time error)。
- 异常在PL/SQL执行过程中很可能出现
- 对异常如果不进行处理，异常可能会中断程序的运行

异常的种类

- 预定义异常
- 用户自定义异常

预定义异常

- 当PL/SQL违背了Oracle原则或超越了系统依赖的原则就会隐式的产生内部异常。在PL/SQL中异常通过名字处理预定义异常
- 预定义的一些异常名称：
  - NO_DATA_FOUND --没有找到数据
  - TOO_MANY_ROWS --找到多行数据
  - INVALID_CURSOR --失效的游标
  - ZERO_DIVIDE --除数为零
  - DUP_VAL_ON_INDEX –唯一索引中插入了重复值

预定义异常

```
DECLARE
    v_id employees.employee_id%TYPE;
BEGIN
    SELECT      employee_id
            INTO    v_id
            FROM  employees
            WHERE           department_id = 50;
...
```

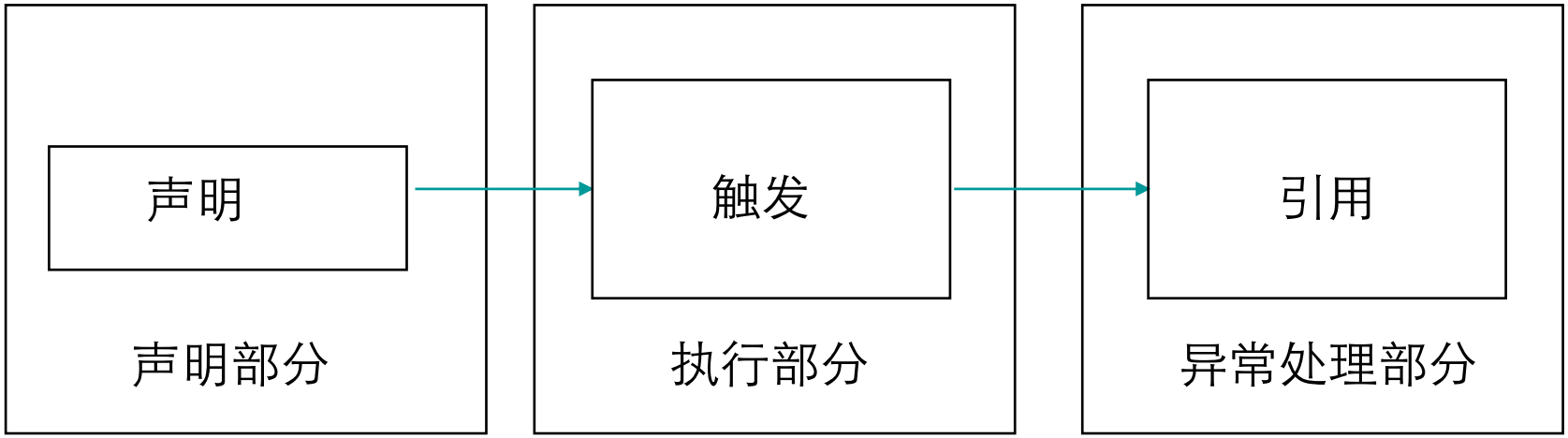
预定义异常(续)

```
...
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('没有50号部门记录 ');
    WHEN TOO_MANY_ROWS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('返回多条记录.');
    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE (' 出现其他错误.');
END;
```

用户自定义异常

- 用户定义的异常不一定是oracle返回的系统错误，系统不会自动触发，需要在声明部分定义。
- 用户定义的异常处理部分基本上和预定义异常相同。

捕获用户自定义异常



命名异常

RAISE语句来显式触发异常

处理触发的异常

用户自定义异常

```
DECLARE
    SALARY_LEVEL                VARCHAR2(1);
    INVALID_SALARY_LEVEL    EXCEPTION;   --声明异常
BEGIN
    SALARY_LEVEL:='D';
    IF SALARY_LEVEL NOT IN('A', 'B', 'C') THEN
      RAISE INVALID_SALARY_LEVEL;  --触发用户自定义异常
    END IF;
EXCEPTION
    WHEN INVALID_SALARY_LEVEL THEN
       DBMS_OUTPUT.PUT_LINE('INVALID SALARY LEVEL');
END;
/
```

异常的处理

- 当一个运行时错误发生时，称为一个异常被抛出
- PL/SQL程序块的异常部分包含了程序处理错误的代码，当异常被抛出时，程序控制离开执行部分转入异常处理部分

异常处理部分语法

```
EXCEPTION
  WHEN exception1 [OR exception2 . . .] THEN
    语句体1;
    . . .
  [WHEN exception3 [OR exception4 . . .] THEN
    语句体2;
    . . .]
[WHEN exceptionN] THEN
    语句体n
    . . .]
[WHEN OTHERS THEN
    语句体n+1
     . . .]
```

捕获异常的规则

- 在异常部分WHEN 子句没有数量限制
- 当异常抛出后，控制无条件转到异常处理部分
- EXCEPTION 关键词开始异常处理部分
- WHEN OTHERS 为最后的一条子句
- 在异常块中，只有一个句柄会处理异常

异常捕获相关的函数

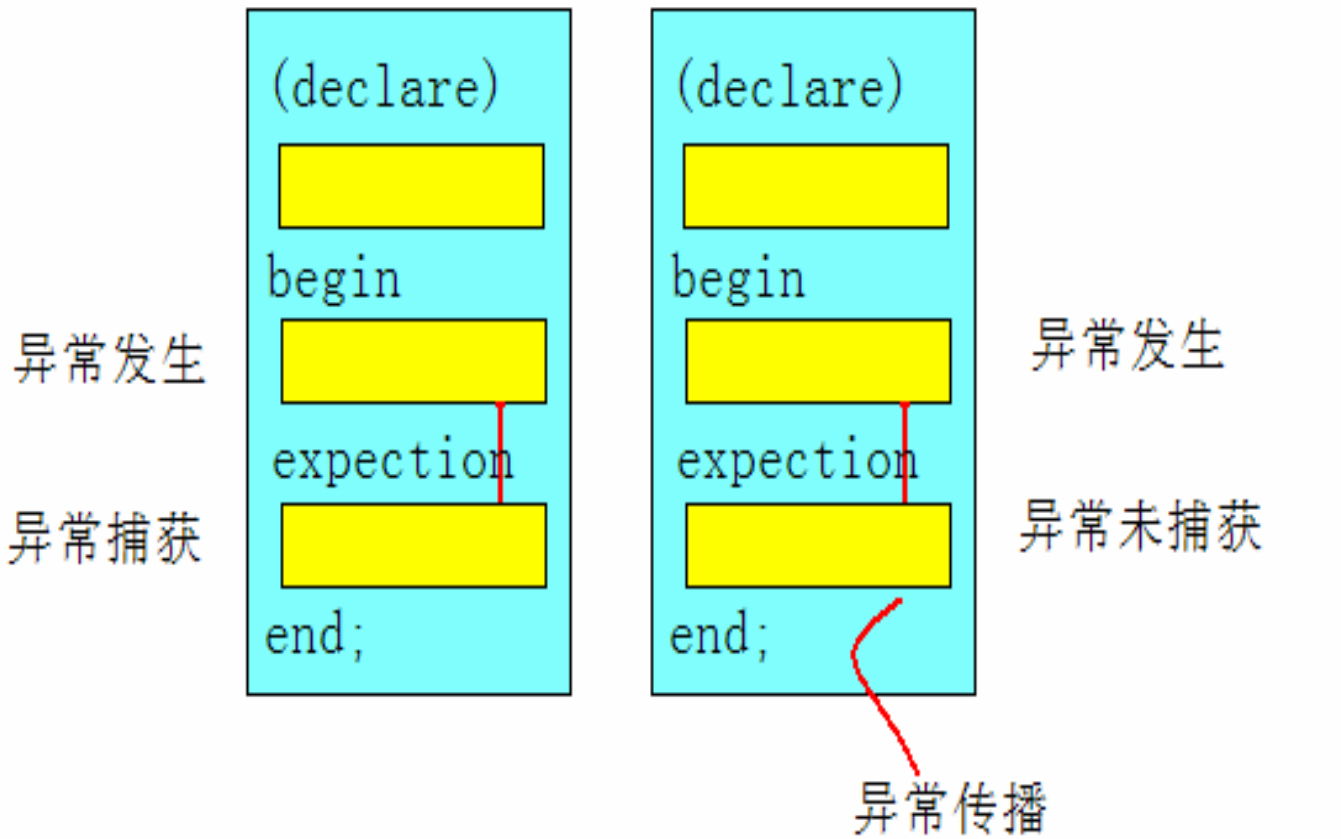
- SQLCODE
  - 返回错误代码
- SQLERRM
  - 返回与错误代码关联的消息

异常捕获中使用的函数

- 保存任何非预期的异常的错误编码和错误消息

```
. . .
  v_error_code      NUMBER;
  v_error_message   VARCHAR2(255);
BEGIN
. . .
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        v_error_code := SQLCODE;
        v_error_message := SQLERRM;
       INSERT INTO err_logs VALUES (v_error_code,  v_error_message);
END;
```

异常的传播



异常的传播

- PL/SQL中错误处理的步骤:
  - 步骤1：如果当前块中有该异常的处理器，则执行该异常处理语句块，然后控制权传递到外层语句块
  - 步骤2：如果没有当前异常的处理器，把该异常传播给外层块。然后在外层执行步骤1。如果此语句在最外层语句块，则该异常将被传播给调用环境
- 没有处理的异常将沿检测异常调用程序传播到外面，当异常被处理并解决或到达程序最外层传播停止。异常是自里向外逐级传递的

本章小结

- 异常的基本概念
- 异常的处理
- 异常的传播

练习

- 1.根据员工号，获得员工到目前为止参加工作年限（保留到整数），员工号不存在时提示“此员工号不存在”
- 2.编写PL/SQL块，使用SELECT语句将管理者编号为空的员工的姓名及工作编号显示出来，如果符合条件的员工多于一人，则返回字符串“最高管理者人员过多！”字符串，如果找到没有符合条件的记录，则返回字符串“没有最高管理者，请指定”
- 3.获得每个部门的平均工资，如果平均工资大于15000，视为用户定义的异常，提示“该部门的平均工资过高”

第十四章　存储过程和函数

Neusoft 东软

本章要点

- 存储过程概念、创建及调用
- 存储过程参数模式
- 存储过程的删除
- 函数概念概念、创建及调用
- 过程和函数的区别
- 函数的删除

PL/SQL块

- PL/SQL包括匿名块和命名块。
  - 匿名块不存储到数据库中。
  - 命名块可被独立编译并且存储在数据库中，可以有参数。可以从任何数据库客户端或者工具引用和运行他们，比如SQL*PLUS，Pro*C。
  - 命名块主要包括存储过程、函数、包、触发器。

PL/SQL块

- 存储过程与函数有如下优点：
  - 可重用性：一旦命名并保存在数据库中以后，任何应用都可以调用
  - 抽象和数据隐藏：用户只需知道存储过程与函数对外提供的功能，而无需知道其内部实现
  - 安全性：通过存储过程与函数提供数据对象的操作权限，而不必给出存储过程与函数涉及到每个对象的权限，提高了安全性

存储过程

- 存储过程就是命了名的PL/SQL块，可以有零个或多个参数，没有返回值，以编译后的形式存放在数据库中，然后由开发语言调用或者PL/SQL块中调用。
- 是一种用来执行某些操作的子程序。

存储过程

- 存储过程创建语法：

```
CREATE [OR REPLACE] PROCEDURE
[schema.]procedure_name [(argument [in|out|in out] type…)]
IS | AS
 [本地变量声明]
BEGIN
      执行语句部分
[EXCEPTION]
      错误处理部分
END[procedure_name];
```

存储过程：基本规则

- 在Sql*Plus 中使用CREATE OR REPLACE子句创建存储程序单元。
- 在头部定义所有参数。
- 在IS之后，声明本地变量，不需要使用DECLARE开始声明
- BEGIN开始程序的执行主体
- oracle图形化界面“Enterprise Manager Console”来创建存储过程 。

存储过程参数

- 参数可以为任何合法的PL/SQL类型
- 参数模式：IN， OUT， IN OUT。
  - IN，就是从调用环境通过参数传入值，在过程中只能被读取，不能改变。
  - OUT，由过程赋值并传递给调用环境。不能是具有默认值的变量，也不能是常量，过程中要给OUT参数传递返回值。
  - IN OUT ，具有IN 参数和OUT 参数两者的特性，在过程中即可传入值，也可传出值。

创建存储过程：in参数模式

- 创建测试表：

```
Create table t_test
(f_1 varchar2 (20),f_2 number(3,0));
```

- 创建过程：

```
CREATE OR REPLACE PROCEDURE
 up_ins_test
(p_f1 varchar2 ,p_f2 number default 10)
IS
BEGIN
      INSERT INTO t_test (f_1,f_2) VALUES(p_f1,p_f2);
END up_ins_test;
```

调用过程

- 在Sql*Plus中调用过程
  - 在Sql*plus的提示符下执行带有参数的过程

```
SQL> exec up_ins_test (p_f1=> 'abc',p_f2=> 100);
```

- 在过程中调用过程

```
CREATE OR REPLACE PROCEDURE process_ins
        (p_f1 varchar2 , p_f2 number)
IS
BEGIN
         change_salary (p_f1, p_f2);
         ...
END;
```

常见的参数传递方法

- 使用名称

```
SQL> exec up_ins_test (p_f2=> 100,p_f1=> ‘abc’);
```

- 使用位置

```
SQL> exec up_ins_test (‘abc’, 100)
```

- 使用混合方式

```
SQL> exec up_ins_test (‘abc’, p_f2=> 100);
```

创建存储过程：out参数模式

- 使用employees表，编写搜索过程，输入EMPLOYEE_ID,返回LAST_NAME

```
CREATE OR REPLACE PROCEDURE up_GetEmpName
(p_empid in employees.employee_id%type,
      p_ename out employees.last_name%type)
AS
BEGIN
   SELECT last_nam  INTO p_ename
    FROM employees  WHERE employee_id = p_empid;
EXCEPTION
WHEN NO_DATA_FOUND THEN
          p_ename := ‘null’;
 END up_GetEmpName;
```

删除存储过程

- 和其他数据库对象一样，删除存储过程时使用DROP语句，语法如下：

```
SQL> DROP PROCEDURE procedure_name
```

- 例如，删除过程up_GetEmpName

```
SQL> DROP PROCEDURE up_GetEmpName;
```

函数

- 函数是有返回值的命名的PL/SQL块
- 函数以编译后的形式存放在数据库中用来重复执行
- 函数作为表达式的一部分被调用

创建函数

- 创建函数语法：

```
CREATE [OR REPLACE] FUNCTION
 [schema.] function_name [(argument [in|out|in out] type…)]
RETURN returning_datatype
IS | AS
 [本地变量声明]
BEGIN
      执行语句部分
[EXCEPTION]
      错误处理部分
END[function_name];
```

创建函数例子

```
CREATE OR REPLACE FUNCTION uf_GetEmpName
(p_empid in number)
return employees.last_name%type
AS
    p_ename employees.last_name%type;
BEGIN
  SELECT last_name INTO p_ename
  FROM employees   WHERE employee_id = p_empid;
   return p_ename
EXCEPTION
   WHEN NO_DATA_FOUND THEN
        return 'error';
END uf_GetEmpName;
```

在SQL语句中调用函数

- 好处
  - 可以实现用简单的SQL语句不能实现的计算
  - 提高查询的效率
- 使用规则
  - 只能使用函数，而不是过程
  - 从SQL语句中调用的函数中不允许DML语句
  - 形参必须为IN
  - 必须返回Oracle支持数据类型，不能使用PL/SQL数据类型
  - 必须有EXECUTE权限

在SQL语句中调用函数

- 任何有效的SQL子句中
- SELECT命令的选择列表
- WHERE和HAVING条件子句
- ORDER BY, 和GROUP BY子句
- INSERT命令的VALUES 子句
- UPDATE 命令的SET 子句

在SQL语句中调用函数示例

```
SQL> SELECT  uf_GetEmpName(employee_id), salary
  2  FROM    employees
  3  WHERE   department_id = 50;
```

在Sql*Plus中调用函数

```
SQL> declare
  2  v_name employees.last_name%type;
  3  begin
  4  v_name:=uf_GetEmpName(100);
  5  end;
  6   /
```

删除函数

- 和其他数据库对象一样，删除时使用DROP语句，语法如下：

```
SQL> DROP FUNCTION  function_name
```

- 例如我们把刚创建的函数删除

```
SQL> DROP FUNCTION uf_GetEmpName;
```

过程和函数的区别

- 函数有一个返回值，而过程没有返回值。
- 过程的调用：作为一个独立执行语句在PL/SQL中调用

```
过程名称(参数1,参数2);
```

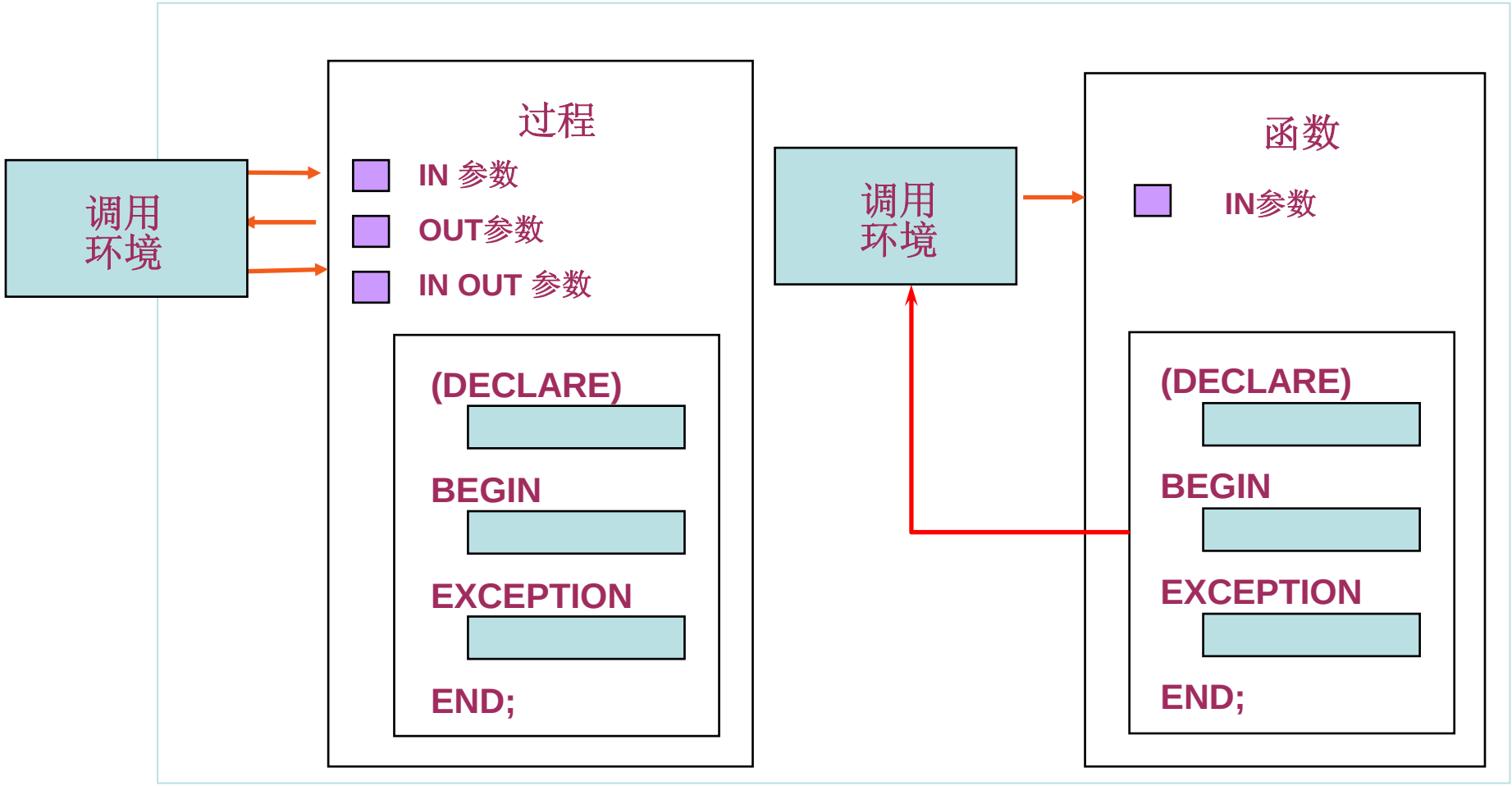
- 函数的调用：以合法的表达式的方式调用，如给变量赋值

```
V_sal := 函数名称(参数1,参数2) ;
```

或作为SQL语句的子句，如：

```
SELECT函数名称(参数1,参数2) FROM dual;
```

函数和过程的比较



函数和过程的比较

过程	函数
完成某些特定的任务或者工作	完成复杂的计算
作为一个独立PL/SQL语句来执行	不能独立执行，必须作为表达式的一部分来调用
程序头部使用PROCEDURE说明	程序头部使用FUNCTION说明
程序头部不需要描述返回类型	程序头部必须描述函数返回值的类型
可以有RETURN语句	必须包含RETURN语句
可以不返回值，也可以返回一个值 或多个值，通过OUT或者IN OUT模式参数来返回	必须返回一个值，通过RETURN语句返回，返回值必须与函数头部声明的返回类型一致

相关数据字典

- 一般来说，关于我们创建的函数和过程的一些信息，我们都可以通过图形化界面清晰的看到，也可以在user_source这个数据字典视图中查询。

本章小结

- 存储过程概念、创建及调用
- 存储过程参数模式
- 存储过程的删除
- 函数概念概念、创建及调用
- 函数的删除

练习

- 1.创建存储过程PrintEmp，将EMPLOYEES表中所有员工的编号和姓名显示出来。
- 2.创建存储过程PTEST，接受两个数相除并且显示结果，如果第二个数是0，则显示消息“not to DIVIDE BYZERO！”，不为0则显示结果。
- 3.创建一个函数Emp_Avg：根据员工号，返回员工所在部门的平均工资。
- 4.创建一个过程Update_SAL，通过调用上题3中的函数，实现对每个员工工资的修改：如果该员工所在部门的平均工资小于1000，则该员工工资增加500；大于等于1000而小于5000，增加300；大于等于5000，增加100。

练习（续）

- 5.创建存储过程，根据员工编号计算该员工年薪（工资+奖金），并将计算出的年薪值传递到调用环境。
- 6.编写一个函数，根据员工编号计算该员工进入公司的月数。
- 7.为employees表中工作满20年以上的员工加薪20%。

练习（续）

- 8.编写函数计算个人所得税：个人收入（除去各项保险及住房公积金）外超过1600元的，需交个人所得税。用月收入减去1600，看其余额是多少。

  余额在500元及500元以下的，用余额乘以5%。

  余额在501—2000元的，用余额乘以10%-25元。

  余额在2001—5000元的，用余额乘以15%-125元。

  余额在5000—20000元的，用余额乘以20%-375元。

第十五章 用户、权限和角色

Neusoft 东软

本章要点

- 用户的概念及管理
- 权限的概念及管理
- 角色的概念及管理

用户

- 用户是数据库的使用者。用户相关的信息包括用户的用户名称和密码、用户的配置信息（包括用户的状态，用户的默认表空间等）、用户的权限、用户对应的方案中的对象等。
- 用户一般是由DBA来创建和维护的。创建用户后，用户不可以执行任何Oracle操作（包括登陆），只有赋予用户相关的权限，用户才能执行相关权限允许范围内的相关操作。对用户授权可以直接授权，也可以通过角色来间接授权。

创建用户

- 创建用户语法

```
CREATE USER user
IDENTIFIED BY   password;
```

- 执行该语句的用户需要有“创建用户”的权限，一般为系统的DBA用户。

创建用户示例

- 以SYSTEM用户登陆。

```
conn system
请输入口令:  ******
已连接。
```

```
CREATE USER test IDENTIFIED BY test;
```

- 用户被创建后，没有任何权限，包括登陆。

登陆权限

- 用户如果想登陆，至少需要有“CREATE SESSION”的权限。

```
GRANT CREATE SESSION TO test;
```

用户建表相关权限

- 对新建用户，默认情况，用户对于其所拥有对象具有所有的数据的增删改查权限，但没有定义的权限（如创建表等）。用户要想创建对象，需要有对象的创建权限 CREATE TABLE、CREATE SEQUENCE等
- 以test用户身份建表：

```
CREATE TABLE emp1(id NUMBER,last_name
    VARCHAR2(20),salary NUMBER);
返回权限不足。
```

用户建表相关权限

- 赋予test用户的创建表的权限。

```
Conn system/oracle;
GRANT CREATE TABLE TO test;
```

- test用户身份执行建表操作：

```
CREATE TABLE emp1(id NUMBER,last_name
    VARCHAR2(20),salary NUMBER);
返回错误“表空间‘SYSTEM’中无权限”
```

- 建表的相关权限：
  - CREATE TABLE
  - 空间使用权限，需要DBA来分配。

用户空间分配及管理

- 用户空间分配和管理：
  - 给用户分配表空间的配额；
  - 给用户指定一个默认的表空间，如果没有特殊指定，则对象都是在该用户默认表空间中创建的。如果不指定默认表空间，则系统缺省默认表空间是SYSTEM，默认情况下，SYSTEM表空间也没有给任何普通用户分配配额（SYS和SYSTEM用户除外。

- 查询数据库表空间。以SYSTEM用户身份：

```
SELECT * FROM v$tablespace;
```

指定默认表空间

- 以SYSTEM用户身份执行:

```
ALTER USER test
DEFAULT TABLESPACE example;
```

- 以test用户身份执行

```
CREATE TABLE emp1(id NUMBER,last_name
    VARCHAR2(20),salary NUMBER);
返回错误“ 表空间‘EXAMPLE’中无权限”。
```

- 以SYSTEM用户身份执行，给test分配配额。

```
ALTER USER test
QUOTA 10m ON example;
```

默认表空间及配额

- 以test用户身份执行建表命令。

```
CREATE TABLE emp1(id NUMBER,last_name
    VARCHAR2(20),salary NUMBER);
表已创建。
```

- 表空间“example”上分配了10m的空间使用权限，用户在该表空间上只有10M的使用权限，如果超过该限度，用户的相关操作执行失败。

修改用户密码

```
ALTER USER user IDENTIFIED BY 新密码;
```

- DBA可以修改任何普通用户的密码，而不需要知道用户的旧密码。
- 在sqlplus下执行password命令来修改登陆用户自己的密码，提示会输入旧密码和新密码。

用户状态

- 用户状态：OPEN、EXPIRED、LOCKED。
  - OPEN表正常状态，为用户帐号初始创建后状态。
  - EXPIRED表示密码过期，用户下次登陆的时候需要修改密码；
  - LOCKED表示该帐户已被锁定，不能执行任何Oracle相关操作（即使拥有相关的权限）。

- 状态管理语句：

```
ALTER USER user PASSWORD EXPIRE;--密码过期
ALTER USER user ACCOUNT LOCK[UNLOCK];--帐户锁定/解锁
```

删除用户

- 删除用户语法

```
DROP USER user [CASCADE]
```

- CASCADE表示系统先自动删除该用户下的所有对象，然后再删除该用户的定义。
- 已经登陆的用户是不允许被删除的。

数据字典视图

- 与用户信息相关的数据字典视图有
  - DBA_USERS是关于用户的属性信息
  - DBA_TS_QUOTAS是用户的相关表空间的配额信息。
- 数据字典视图一般是以SYSTEM用户身份执行:

```
SELECT username,account_status,default_tablespace FROM
    dba_users;
```

```
SELECT * FROM dba_ts_quotas;
```

权限

- Oracle中存在两种权限
  - 系统权限（SYSTEM PRIVILEGE）：允许用户在数据库中执行指定的行为，一般可以理解成比较通用的一类权限。
  - 对象权限（OBJECT PRIVILEGE）：允许用户访问和操作一个指定的对象，该对象是一个确切存储在数据库中的命名对象。

系统权限

- 包含100多种系统权限，其主要作用：
  - 执行系统端的操作，比如CREATE SESSION是登陆的权限，CREATE TABLESPACE创建表空间的权限
  - 管理某类对象，比如CREATE TABLE是用户建表的权限
  - 管理任何对象，比如CREATE ANY TABLE，ANY关键字表明该权限“权力”比较大，可以管理任何用户下的表，所以一般只有DBA来使用该权限，普通用户是不应该拥有该类权限的。

表的系统权限

- CREATE TABLE（建表）
- CREATE ANY TABLE（在任何用户下建表）
- ALTER ANY TABLE（修改任何用户的表的定义）
- DROP ANY TABLE（删除任何用户的表）
- SELECT ANY TABLE（从任何用户的表中查询数据）
- UPDATE ANY TABLE（更改任何用户表的数据）
- DELETE ANY TABLE（删除任何用户的表的记录）。

表的系统权限（续）

- 除ANY权限外，当前用户下表的操作权只有CREATE TABLE，而没有DROP TABLE，SELECT TABLE，ALTER TABLE、CREATE INDEX、DROP INDEX等权限。
- 当用户拥有了CREATE TABLE权限后，也同时获得了该用户下任何表的DROP、UPDATE、SELECT、DELETE、INSERT、TRUNCATE等权限。
- 从安全的角度来说，任何含ANY关键字的权限不应该被分配给普通用户。

索引及会话系统权限

- 索引:
  - CREATE ANY INDEX（在任何用户下创建索引）
  - ALTER ANY INDEX（修改任何用户的索引定义）
  - DROP ANY INDEX（删除任何用户的索引）
- 会话: （SESSION）
  - CREATE SESSION（创建会话，登陆权限）
  - ALTER SESSION（修改会话）

表空间系统权限

- 表空间
  - CREATE TABLESPACE（创建表空间）
  - ALTER TABLESPACE（修改表空间）
  - DROP TABLESPACE（删除表空间）
  - UNLIMITED TABLESPACE（不限制任何表空间的配额）
- 注意： 表空间的所有权限都不应该分配给普通用户。

系统特权

- 系统特权权限SYSDBA和SYSOPER
  - SYSOPER的权限：启动停止数据库，恢复数据库等
  - SYSDBA的权限：所有SYSOPER功能的管理权限；创建数据库等权限。
- 注意： 以系统特权权限登陆的用户一般都是特权用户，或称为超级用户。以SYSDBA身份登陆的用户在ORACLE中是权限最大的用户，可以执行数据库的所有操作。这些特权权限是不应该随便赋予给普通用户的。

系统权限授予及回收

- 授予用户系统权限语法

```
GRANT 系统权限 TO user [WITH ADMIN OPTION]
```

- WITH ADMIN OPTION选项则可以授予普通用户管理权限的权限。

系统权限授予及回收

- 授予test用户CREATE SESSION权限

```
GRANT create session TO test WITH ADMIN OPTION;
```

- 以test身份执行CREATE SESSION权限管理：

```
GRANT create session TO neu;
```

系统权限授予及回收

- 回收系统权限

  REVOKE 系统权限 FROM *user*;
- 只能回收使用GRANT授权过的权限，权限被回收后，用户就失去了原权限的使用权和管理权（如果有管理权限的话）。
- 注意：使用WITH ADMIN OPTION选项授予的权限在回收时候的级联回收策略如下：
  - 如果用户A授予权限给用户B，同时带有选项WITH ADMIN

```
CREATE INDEX        emp_last_name_idx
ON  employees(last_name);
Index created.
```

系统权限授予及回收

- 回收系统权限

```
REVOKE 系统权限 FROM user;
```

- 只能回收使用GRANT授权过的权限，权限被回收后，用户就失去了原权限的使用权和管理权（如果有管理权限的话）。
- 系统权限不会级联回收。

对象权限

- 对象权限的种类不是很多，但数量较大，因为具体对象的数量很多。
- 对象权限的分类

权限分类＼对象类型	表(Table)	视图(View)	序列(Sequence)	存储(Procedure)
SELECT(选择)	○	○	○	
INSERT(插入)	○	○		
UPDATE(更改)	○	○		
DELETE(删除)	○	○		
ALTER(修改)	○		○	
INDEX(索引)	○			
REFERENCE(引用)	○	○		
EXECUTE(执行)				○

授予对象权限

```
GRANT 对象权限种类 ON 对象名[(列名列表)] TO user [WITH
    GRANT OPTION];
```

- 授予对象权限的用户是对象的拥有者(OWNER)或其他有对象管理权限的用户(常为DBA)。
- WITH GRANT OPTION，表示把对象的管理权限赋给其他用户。

- 以neu身份执行授权命令:

```
GRANT select on employees To test;
```

回收对象权限

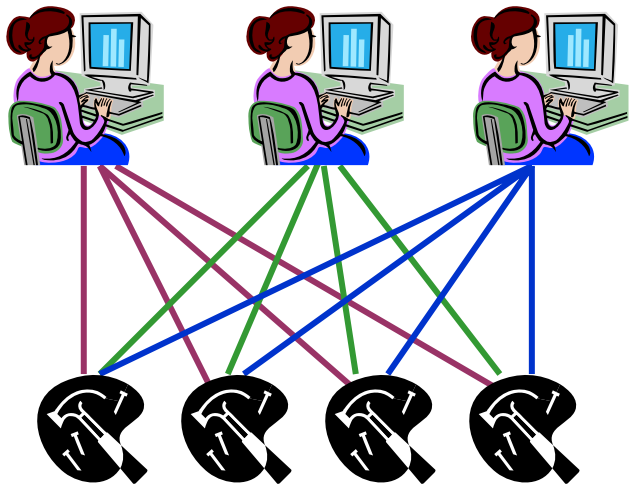
- 回收对象权限语法：

```
REVOKE 对象权限种类 ON 对象名[(列名列表)] FROM user
```

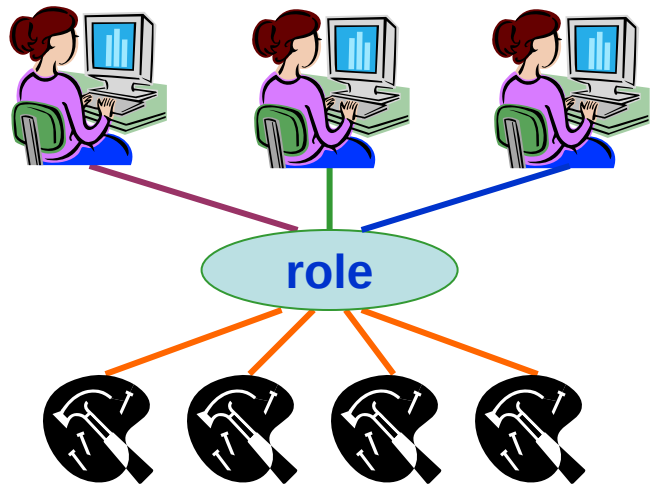
- 对象的权限会级联回收。
- 权限的查询
  - DBA_SYS_PRIVS：查询所有的系统权限的授权情况。
  - SESSION_PRIVS：能够查询出当前会话已经激活的所有系统权限。
  - DBA_TAB_PRIVS：查询出表的对象权限的授权情况。

角色

- 角色（ROLE）的目的就是为了简化权限的管理。



单独授予权限



使用角色授予权限

用户

权限

使用角色的好处

- 简化权限的管理，而且易于以后的维护，使得维护成本降低。
- 动态权限的管理
- 权限的可选择性

角色管理

- 创建角色语法：

```
CREATE ROLE role;
```

- 以SYSTEM的用户身份建立测试角色tr。

```
CREATE ROLE tr;
```

```
GRANT create sequence TO tr;
```

```
GRANT tr To test;
```

- 以test用户登陆，验证是否已拥有相关权限

```
SELECT * FROM session_privs;
```

收回角色

- 收回角色语法:

```
REVOKE 角色 FROM 用户;
```

- 删除角色:

```
DROP ROLE 角色;
```

预定义角色

- 常用预定义角色:
  - DBA角色。该角色中的权限通常赋给数据库管理员。
  - CONNECT角色。
  - RESOURCE角色。
- CONNECT和RESOURCE是相对较安全的角色，角色中包含的权限仅限于用户自己的对象范围，因此，可使用CONNECT和RESOURCE来简化权限管理。
- 两者区别是RESOURCE中没有登陆的权限，并增加了几种对象的创建权限。

查看预定义角色的权限

- 角色DBA中包含的系统权限

```
SELECT * FROM DBA_SYS_PRIVS
WHERE GRANTEE='DBA‘
```

- CONNECT角色的相关权限：

```
SELECT * FROM DBA_SYS_PRIVS
WHERE GRANTEE='CONNECT‘
```

- RESOURCE角色中的权限：

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

PUBLIC

- PUBLIC对象既不是用户，也不是角色，代表公众，公开，PUBLIC中拥有的所有权限，所有数据库的用户都会自动拥有。为安全起见，PUBLIC中不应该拥有任何权限。
- 给PUBLIC赋予权限

```
GRANT create session TO public;
```

- 所有的用户都会自动从public中获得登陆的权限。

角色相关的数据字典视图

- DBA_ROLES：数据库中的角色列表
- DBA_ROLE_PRIVS：查询把哪些角色赋予给哪些对象了(包括给用户、角色、PUBLIC)
- SESSION_ROLES：当前用户激活的角色。

本章小结

- 用户的管理
- 权限的管理
- 角色的管理

练习

- 1.建立新用户user_neu
- 2. 给用户user_neu授权,使其能够登陆到数据库，能够查询neu下的employees表，能修改employees表的salary,last_name两个字段
- 3.查询用户user_neu的权限
- 4.回收用户user_neu的登陆权限
- 5.回收用户user_neu的所有对象权限
- 6.建立角色role_neu

练习（续）

- 7.给角色role_neu授权,使其能够登陆到数据库
- 8.赋角色role_neu给用户user_neu
- 9.删除角色role_neu
- 10.删除用户user_neu