

# 第 1 章

---

## 走进 Oracle

### 主要内容

- ✓ Oracle 安装
- ✓ Oracle 创建用户和角色
- ✓ 客户端链接 Oracle 服务器

# 1. Oracle 简介

---

在第一学期我们已经接触过关系型数据库 SQL Server，对数据库、表、记录、表的增删改查操作等这些基本的概念已经了解。Oracle 是基于对象的关系型数据库，Oracle 也是用表的形式对数据存储和管理，并且在 Oracle 的操作中添加了一些面向对象的思想。

Oracle 数据库是 Oracle（中文名称叫甲骨文）公司的核心产品，Oracle 数据库是一个适合于大中型企业的数据库管理系统。在所有的数据库管理系统中（比如：微软的 SQL Server，IBM 的 DB2 等），Oracle 的主要用户涉及面非常广，包括：银行、电信、移动通信、航空、保险、金融、电子商务和跨国公司。Oracle 产品是免费的，可以在 Oracle 官方网站上下载到安装包，另一方面 Oracle 服务是收费的。

Oracle 公司成立以来，从最初的数据库版本到 Oracle7、Oracle8i、Oracle9i，Oracle10g 到 Oracle11g，虽然每一个版本之间的操作都存在一定的差别，但是 Oracle 对数据的操作基本上都遵循 SQL 标准。因此对 Oracle 开发来说版本之间的差别不大。

很多人没有学习 Oracle 就开始发怵，因为人们在误解 Oracle，认为 Oracle 太难学了，认为 Oracle 不是一般人用的数据库，其实任何数据库对应用程序研发人员来说，都是大同小异，因为目前多数数据库都支持标准的 SQL。在 Oracle 这本书中，我们能学习到：

- Oracle 的安装
- Oracle 数据管理
- 常用子查询及常用函数
- PL/SQL 编程
- Oracle 基本管理

由于在第一学期已经接触了 SQL Server，Oracle 数据库的概念不是很难，主要是实践，因此在本书的学习中，认真的完成上机练习是学习好本书的关键。

接下来我们先从 Oracle 安装开始，接触一些 Oracle 中基本的概念。

## 2. Oracle 安装

Oracle 数据库产品是免费的，我们可以从 Oracle 的官方网站(<http://www.oracle.com>)下载到程序安装包，Oracle 在 Windows 下的安装非常方便，安装开始后，一直点击安装程序的“下一步”即可。

1. 下载 Oracle10g 后，解压到一个文件夹下，单击“setup.exe”文件即可启动安装界面。如下图：

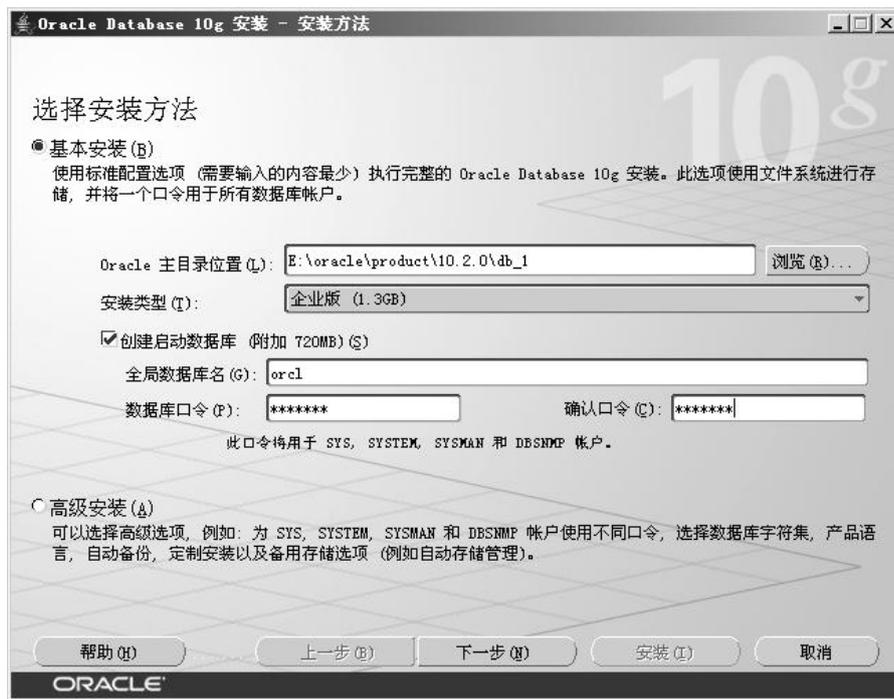


图 1 Oracle 安装启动界面

Oracle 主目录位置就是 Oracle 准备安装的位置，称为“Oracle\_Home”，一般 Oracle 根据当前计算机的硬盘大小默认给出一个合适的位置。Oracle 安装时可以只安装 Oracle 软件，然后单独创建数据库，也可以在上图中选中“创建启动数据库”复选框，在安装 Oracle 产品时，同时创建一个数据库，对初学者来说，推荐这样安装。填写全局数据库名，以及管理员的密码。**全局数据库名是数据库在服务器网络中的唯一标识。**

2. 点击“下一步”，就会出现如下图内容，开始对 Oracle 服务器进行环境检查，主要查看服务器是否符合 Oracle 安装的条件，比如操作系统是否支持、系统内存是否符合 Oracle 安装的最低要求等。

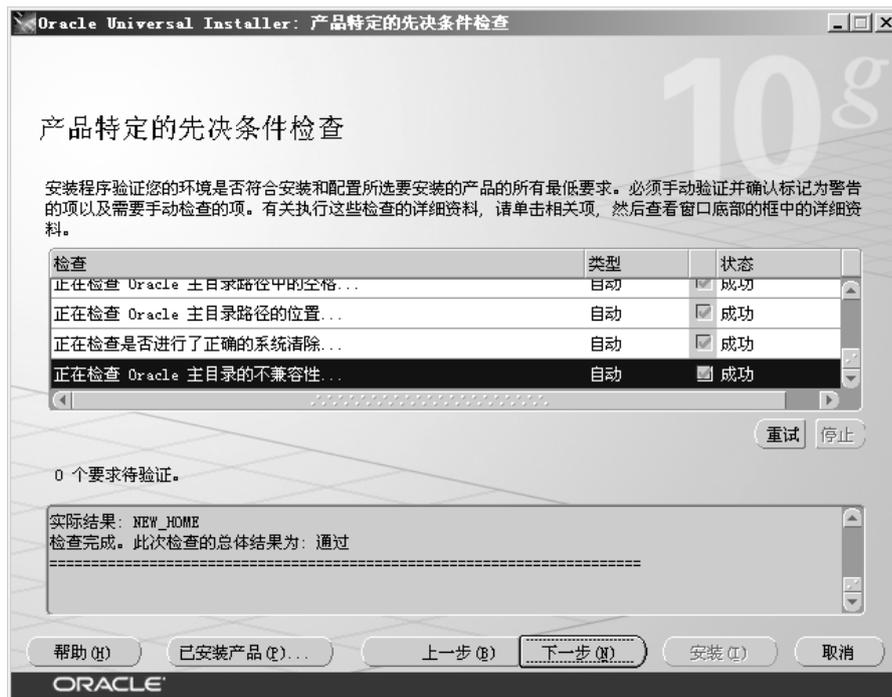


图 2 Oracle 安装前环境检查

- Oracle 检查通过后，单击“下一步”，就会列出所有安装 Oracle 过程中的默认选项。



图 3 Oracle 默认安装设置

- 单击“安装”按钮，进入安装界面，这一过程经历时间比较长，根据计算机的性能不同有很大差别。



图 4 Oracle 安装

5. 上图完成后，进入了各种 Oracle 工具的安装阶段，包括网络配置向导，iSQL\*plus 等（后面课程中讲解）。如下图所示：



图 5 Oracle 各种工具的安装

6. 接下来自动启动 DBCA (Database Configuration Assistant) 进入创建默认数据库阶段。



图 6 DBCA 下安装数据库

Oracle 中的数据库主要是指存放数据的文件，这些文件在 Oracle 安装完成后，在计算机硬盘上都能找到，包括数据文件、控制文件和数据库日志文件。

数据库创建后会有一系列为该数据库提供服务的内存空间和后台进程，称为该数据库的实例。每一个数据库至少会有一个实例为其服务。实例中的内存结构称为系统全局区 (SGA)，系统会根据当前计算机系统的性能给 SGA 分配非常可观的内存空间。

Oracle 创建数据库不能像 SQL Server 那样用一个简单的 CREATE DATABASE 命令就能完成，在创建数据库的过程中还需要配置各种参数。虽然有 DBCA 工具向导，但是仍然需要进行比较麻烦的配置。



提示

虽然一个 Oracle 数据库服务器中可以安装多个数据库，但是一个数据库需要占用非常大的内存空间，因此一般一个服务器只安装一个数据库。每一个数据库可以有很多用户，不同的用户拥有自己的数据库对象（比如：数据库表），一个用户如果访问其他用户的数据库对象，必须由对方用户授予一定的权限。不同的用户创建的表，只能被当前用户访问。因此在 Oracle 开发中，不同的应用程序只需使用不同的用户访问即可。

7. 数据库创建完毕后，需要设置数据库的默认用户。Oracle 中为管理员预置了两个用户分别是 SYS 和 SYSTEM。同时 Oracle 为程序测试提供了一个普通用户 scott，口令管理中，可以对数据库用户设置密码，设置是否锁定。Oracle 客户端使用用户名和密码登录 Oracle 系统后才能对数据库操作。

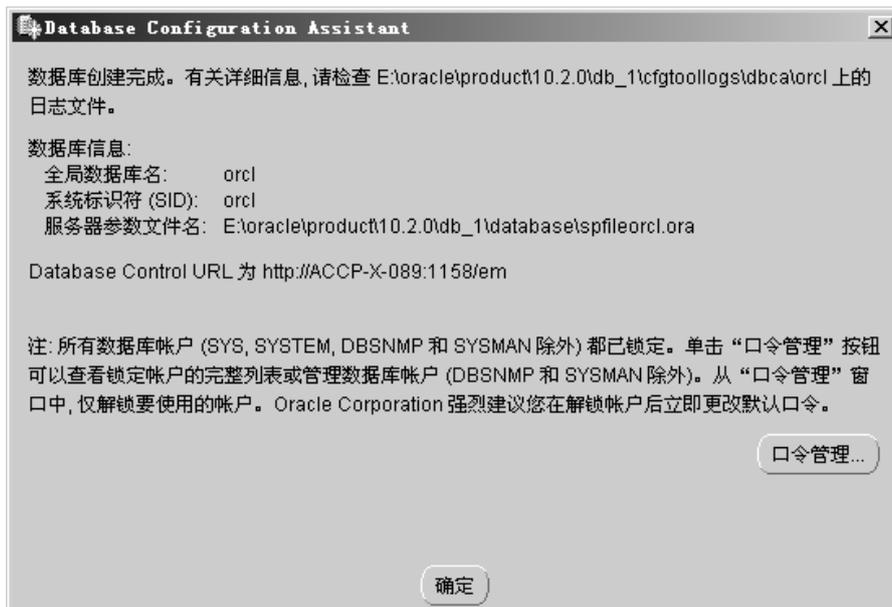


图 7 DBCA 下的口令管理



图 8 为 system,sys,scott 用户设置密码

默认的用户中, SYS 和 SYSTEM 用户是没有锁定的, 安装成功后可以直接使用, SCOTT 用户默认为锁定状态, 因此不能直接使用, 需要把 SCOTT 用户设定为非锁定状态才能正常使用。

这一步完成后, Oracle 系统安装成功。



提示

Oracle 数据库中，默认情况下，所有系统的数据，SQL 关键字等都是大写的，在操作过程中，Oracle 会自动把这些内容转换为大写，因此用户操作时不需考虑大小写问题，一般情况下，为了良好的程序风格，程序中建议关键字用大写，非关键字可以使用小写。

## 3. Oracle 客户端工具

Oracle 服务器安装成功后，就可以通过客户端工具连接 Oracle 服务器了，可以到 Oracle 官方下载 Oracle 专用的客户端软件，大多客户端工具都是基于 Oracle 客户端软件的。接下来介绍几种常用的 Oracle 客户端工具。

### ◇ SQL\*Plus 工具

该工具是 Oracle 系统默认安装下，自带的一个客户端工具。在 Windows 命令行中输入“sqlplusw”命令，就能够启动该工具了。



图 9 SQL\*Plus 工具

输入用户名和密码后，如果 SQL\*Plus 与数据库服务器在同一台计算机上，并且当前服务器下只有一个数据库实例，那么“主机字符串”可以不用填写。



提示

SQL\*Plus 工具虽然是 Oracle 自带的工具，但是在现实开发中，基于该环境对开发不方便，因此很少使用。

SQL\*Plus 连接成功后就如图所示：

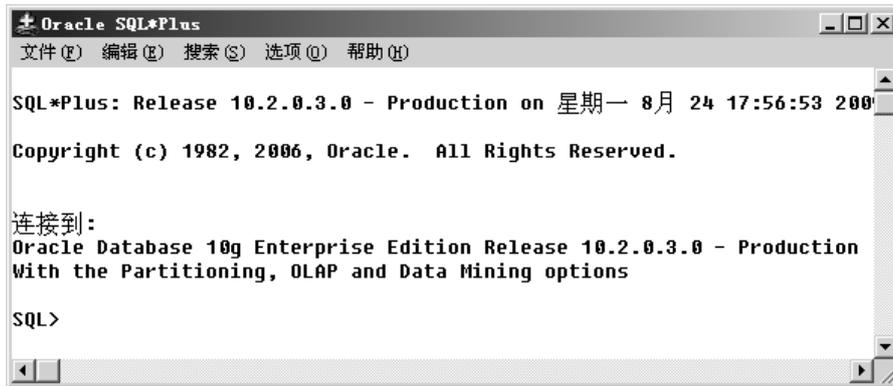


图 10 SQL\*Plus 工具登录后

## ◇ SQL\*Plus 命令行工具

该命令行工具，提供了与数据库交互的能力和维持数据库的能力，包括了 Oracle 自带的 SQL\*Plus 工具的全部功能，在 Oracle 管理中经常使用。在命令行中输入：“`sqlplus /nolog`”即可启动该工具。如下图：

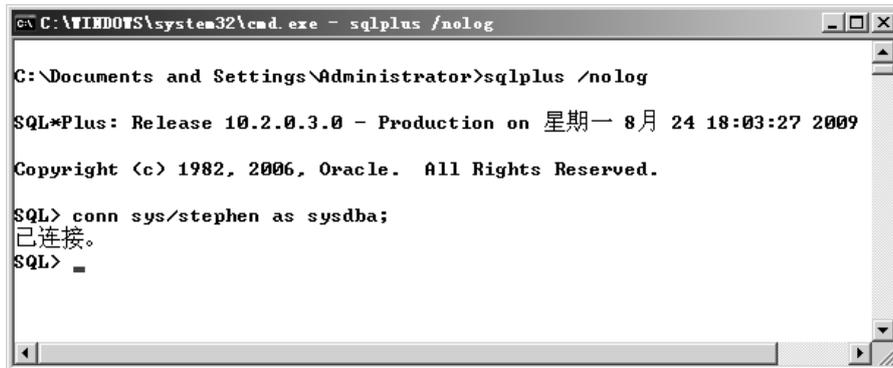


图 11 启动 SQL\*Plus 命令行工具

输入“`sqlplus /nolog`”命令后，只是启动了一个客户端进程，并没有与服务器连接，连接到 Oracle 服务器的命令是：

`conn 用户名/密码 as 连接身份@服务器连接字符串`

说明：

1. 连接身份：表示该用户连接后拥有的权限。

- **sysdba:** 即数据库管理员，权限包括：打开数据库服务器、关闭数据库服务器、备份数据库、恢复数据库、日志归档、会话限制、管理功能、创建数据库。**sys** 用户必须用 **sysdba** 身份才能登录，**system** 用户可以用普通身份登录。
- **sysoper:** 即数据库操作员，权限包括：打开数据库服务器、关闭数据库服务器、备份数据库、恢复数据库、日志归档、会话限制。
- **normal:** 即普通用户，权限只有查询某些数据表的数据。默认的身份是 **normal** 用户。

2. 客户端工具可以根据“服务器连接字符串”对服务器进行连接，有了连接字符串后客户端就可以像操作本机一样操作远程数据库，因此“服务器连接字符串”的配置也叫**本地网络服务配置**，如果 **SQL\*Plus** 工具启动在服务器上，并且服务器上只有一个数据库实例的情况下，连接字符串可以缺省，在**连接字符串中包括连接服务器的协议，服务器的地址，服务器的端口等设置，Oracle 服务名等**，该配置文件在 Oracle 安装目录下的：**network/ADMIN/tnsnames.ora**。该文件是一个文本文件，用记事本打开后如下所示：

```

# tnsnames.ora Network Configuration File:
# E:\oracle\product\10.2.0\db_1\network\admin\tnsnames.ora
# Generated by Oracle configuration tools.

ORCL =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = MyHost-X-089)(PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = orcl)
    )
  )

```

图 12 服务器连接字符串配置

## ◇ 配置本地网络服务名

本地网络服务名，即客户端与服务器的连接字符串，本地网络服务名是客户端的配置，Oracle 客户端安装后，可以使用客户端自带的网络配置向导（Net Configuration Assistant）进行配置：

1. 启动 Net Configuration Assistant。选择“本地 Net 服务名配置”选项。如下图所示：



图 13 启动 Net Configuration Assistant

2. 选择“下一步”，本步骤可以对本地网络服务名进行添加，删除，测试是否正常连接等操作，选择“添加”选项。



图 14 Net Configuration Assistant

3. 点击“下一步”，填写服务名，该服务名就是 Oracle 安装时（图 1），为数据库取的全局数据库名。



图 15 服务名配置

4. 点击“下一步”，选择服务需要的协议，默认是 TCP 协议。推荐使用默认的 TCP 协议。



图 16 选择协议

5. 点击“下一步”，输入主机名，主机名可以是计算机名称，也可以是一个 IP 地址，主机如果是本机，可以使用本机计算机名称、“localhost”、“127.0.0.1”、或者本机的 IP 地址。

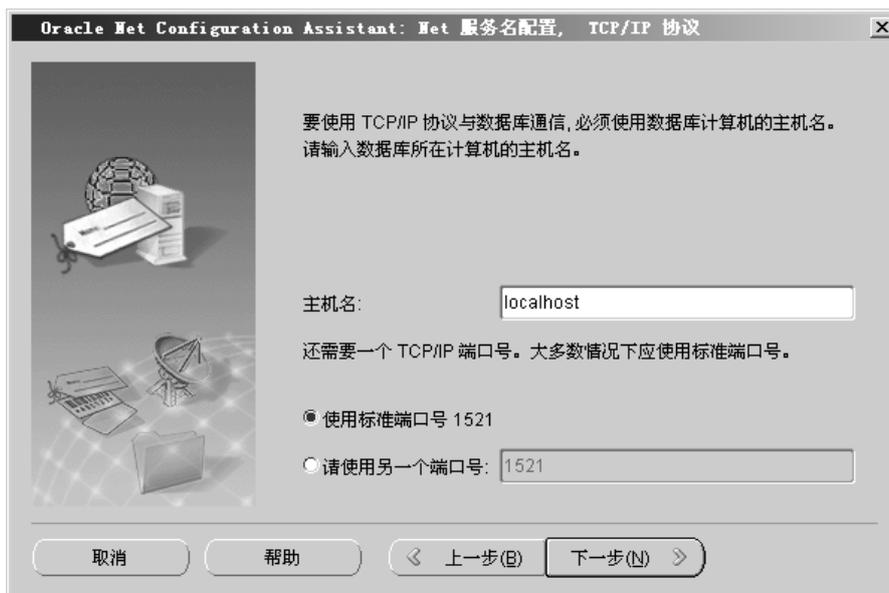


图 17 输入主机名和端口

6. 单击“下一步”，选择“是，进行测试”选项。进入下图界面。



图 18 测试成功

在测试时，默认采用的用户名和密码是 system/manager 进行测试，如果用户 system 的密码不是“manager”，有可能测试通不过，更改登录后，输入正确的用户名和密码后再进行测试即可。

7. 测试成功后，单击“下一步”，出现如下界面，这一步是为本地网络服务命名，即图 12 中的服务器连接字符串名。



图 19 为网络服务名命名

点击“下一步”，配置就完成了，进入 tnsnames.ora 文件中查看，就出现了如图 12 中的内容。

## ◇ PL/SQL Developer 工具

在实际 Oracle 开发中，经常使用一个功能强大的第三方工具：“PL/SQL Developer”工具。PL/SQL Developer 基本上可以实现 Oracle 开发中的任何操作。它运行在客户端时必须先安装 Oracle 客户端，并且通过网络配置向导配置网络服务名后才能正常与服务器连接。

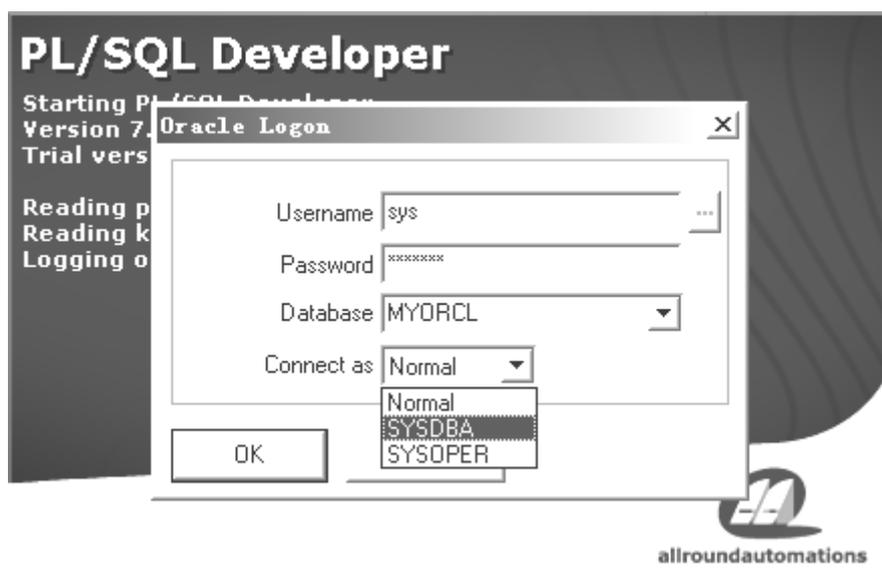


图 20 PL/SQL Developer

## 4. Oracle 服务

Oracle 在 windows 中安装完成后，会安装很多服务，下面介绍几个主要的服务。



图 21 Oracle 服务

- OracleService+服务名，该服务是数据库启动的基础，只有该服务启动了，Oracle 数据库才能正常启动。这是必须启动的服务。
- OracleOraDb10g\_home1TNSListener，该服务是服务器端为客户端提供的监听服务，只有该服务在服务器上正常启动，客户端才能连接到服务器。该监听服务接收客户端发出的请求，然后将请求传递给数据库服务器。一旦建立了连接，客户端和数据库服务器就能直接通信了。
- OracleOraDb10g\_home1iSQL\*Plus，该服务提供了用浏览器对数据库中数据操作的方式。该服务启动后，就可以使用浏览器进行远程登录并进行数据库操作了。如下图所示：



图 22 iSQL\*Plus

- OracleDBConsole+服务名，Oracle10g 中的一个新服务。在 Oracle9i 之前，Oracle 官方提供了一个基于图形界面的企业管理器（EM），从 Oracle10g 开始，Oracle 提供了一个基于 B/S 的企业管理器，在操作系统的命令行中输入命令：emctl start dbconsole，就可以启动 OracleDbConsole 服务，如下图所示：

```
Microsoft Windows [版本 5.2.3790]
(C) 版权所有 1985-2003 Microsoft Corp.

C:\>emctl start dbconsole
Oracle Enterprise Manager 10g Database Control Release 10.2.0.1.0
Copyright (c) 1996, 2006 Oracle Corporation. All rights reserved.
http://MyHost-X-089:1158/em/console/aboutApplication
Starting Oracle Enterprise Manager 10g Database
Control ...OracleDBConsoleorcl
服务正在启动 .....
OracleDBConsoleorcl 服务已经启动成功。
```

*启动EM服务的命令*

*进入EM的地址*

*服务器地址可以是机器名，域名，或者IP地址*

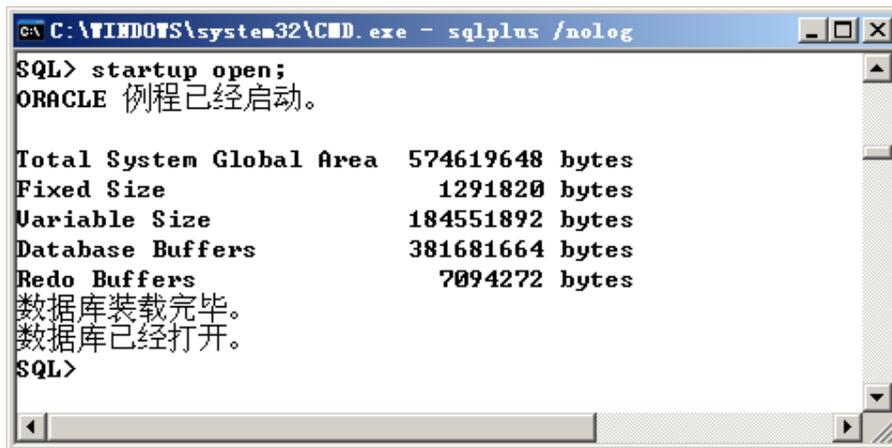
图 23 EM 服务的启动

服务启动之后，就可以在浏览器中输入上图中进入 EM 的地址，使用 B/S 方式管理 Oracle 服务器。

## 5. Oracle 启动和关闭

OracleService 启动后，就可以对数据库进行管理了，Oracle 的启动和关闭是最基本的命令，在 SQL\*Plus 中，启动 Oracle 必须是 sys 用户，命令格式是：

startup open

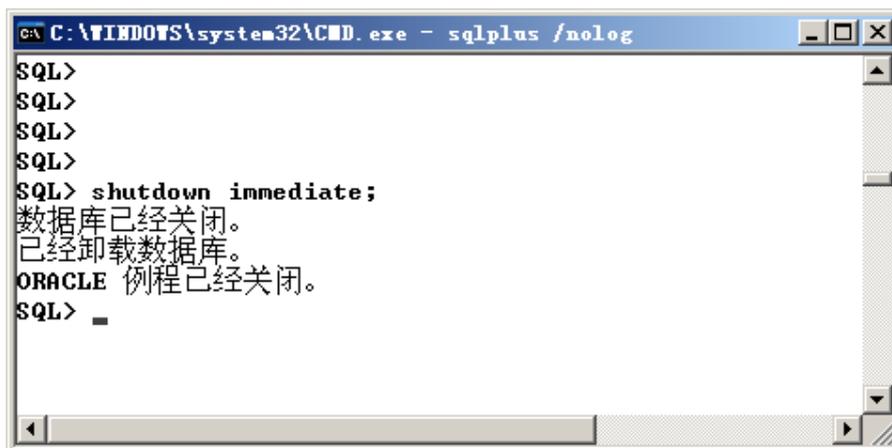


```
C:\WINDOWS\system32\CMD.exe - sqlplus /nolog
SQL> startup open;
ORACLE 例程已经启动。

Total System Global Area  574619648 bytes
Fixed Size                  1291820 bytes
Variable Size              184551892 bytes
Database Buffers          381681664 bytes
Redo Buffers                7094272 bytes
数据库装载完毕。
数据库已经打开。
SQL>
```

图 24 Oracle 服务启动

Oracle 服务关闭用命令: shutdown immediate



```
C:\WINDOWS\system32\CMD.exe - sqlplus /nolog
SQL>
SQL>
SQL>
SQL>
SQL> shutdown immediate;
数据库已经关闭。
已经卸载数据库。
ORACLE 例程已经关闭。
SQL> _
```

图 25 Oracle 服务关闭

## 6. Oracle 用户和权限

---

Oracle 中，一般不会轻易在一个服务器上创建多个数据库，在一个数据库中，不同的项目由不同的用户访问，每一个用户拥有自身创建的数据库对象，因此用户的概念在 Oracle 中非常重要。Oracle 的用户可以用 CREATE USER 命令来创建。其语法是：

### 语法结构：创建用户

---

```
CREATE USER 用户名 IDENTIFIED BY 口令 [ACCOUNT LOCK|UNLOCK]
```

---

## 语法解析：

LOCK|UNLOCK 创建用户时是否锁定，默认为锁定状态。锁定的用户无法正常的登录进行数据库操作。

## 代码演示：创建用户

```
SQL> CREATE USER jerry
2 IDENTIFIED BY tom
3 ACCOUNT UNLOCK;
```



注意

Oracle 在 SQL\*Plus 中的命令以分号 (;) 结尾，代表命令完毕并执行，系统同时会把该命令保存在缓存中，缓存中只保存最近执行过的命令，如果重新执行缓存中的命令，直接使用左斜杠符号 (/)。如果命令不以分号结尾，该命令只是写入缓存保存起来，但并不执行。

尽管用户成功创建，但是还不能正常的登录 Oracle 数据库系统，因为该用户还没有任何权限。如果用户能够正常登录，至少需要 CREATE SESSION 系统权限。

Oracle 用户对数据库管理或对象操作的权利，分为系统权限和数据库对象权限。系统权限比如：CREATE SESSION，CREATE TABLE 等，拥有系统权限的用户，允许拥有相应的系统操作。数据库对象权限，比如对表中的数据进行增删改操作等，拥有数据库对象权限的用户可以对所拥有的对象进行对应的操作。

还有一个概念就是数据库角色 (role)，数据库角色就是若干个系统权限的集合。下面介绍几个常用角色：

- CONNECT 角色，主要应用在临时用户，特别是那些不需要建表的用户，通常只赋予他们 CONNECT role。CONNECT 是使用 Oracle 的简单权限，拥有 CONNECT 角色的用户，可以与服务器建立连接会话 (session，客户端对服务器连接，称为会话)。
- RESOURCE 角色，更可靠和正式的数据库用户可以授予 RESOURCE role。RESOURCE 提供给用户另外的权限以创建他们自己的表、序列、过程 (procedure)、触发器 (trigger)、索引 (index) 等。
- DBA 角色，DBA role 拥有所有的系统权限----包括无限制的空间限额和给其他用户授予各种权限的能力。用户 SYSTEM 拥有 DBA 角色。

一般情况下，一个普通的用户 (如 SCOTT)，拥有 CONNECT 和 RESOURCE 两个角色即可进行常规的数据库开发工作。

可以把某个权限授予某个角色，可以把权限、角色授予某个用户。系统权限只能由 DBA 用户授权，对象权限由拥有该对象的用户授权，授权语法是：

### 语法结构：授权

---

GRANT 角色|权限 TO 用户（角色）

---

### 代码演示：授权

---

```
SQL> GRANT CONNECT TO jerry;
```

授权成功。

```
SQL> GRANT RESOURCE TO jerry;
```

授权成功。

```
SQL>
```

---

### 语法结构：其他操作

---

//回收权限

```
REVOKE 角色|权限 FROM 用户（角色）
```

//修改用户的密码

```
ALTER USER 用户名 IDENTIFIED BY 新密码
```

//修改用户处于锁定（非锁定）状态

```
ALTER USER 用户名 ACCOUNT LOCK|UNLOCK
```

---

## 7. 本章总结

---

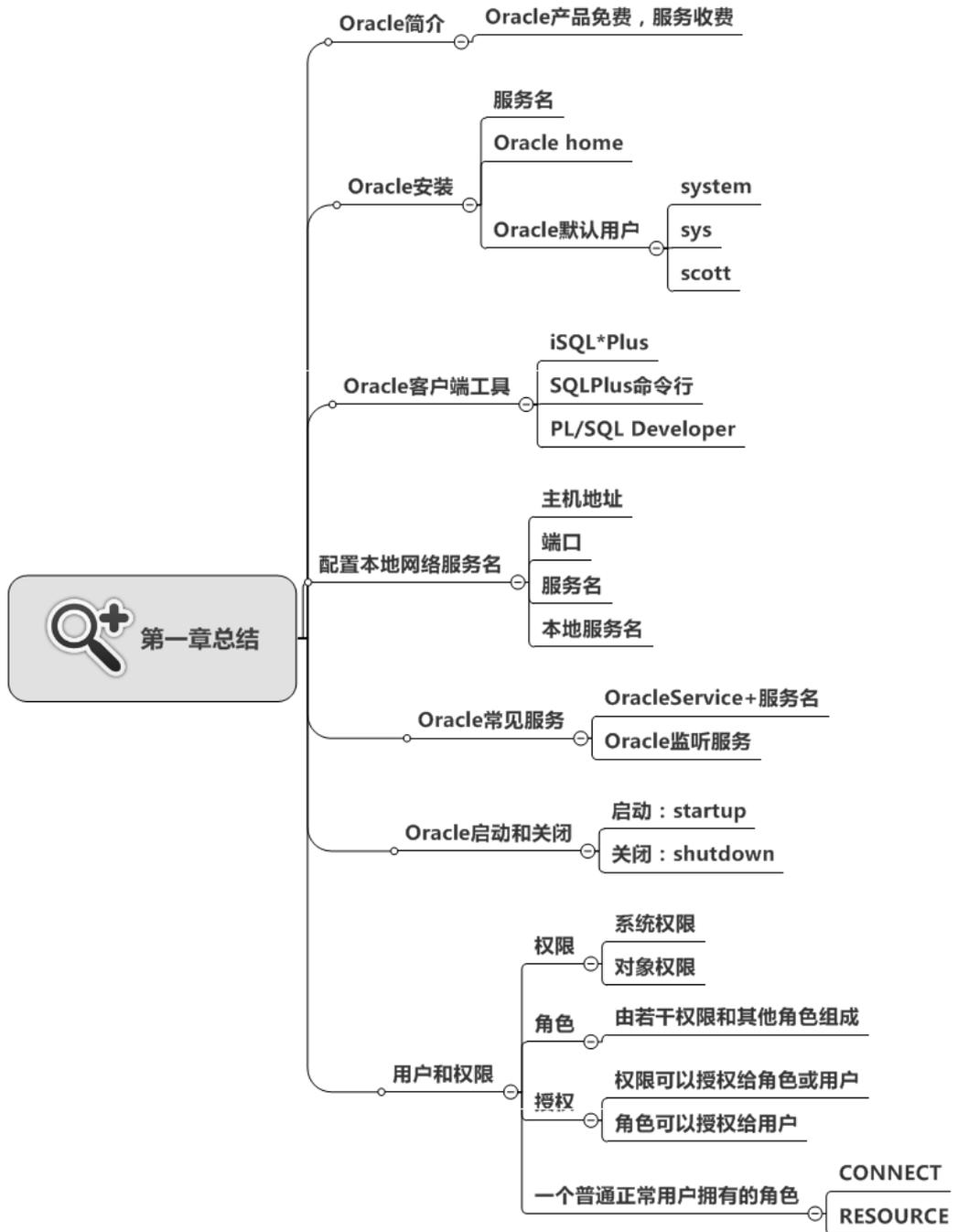
- Oracle 是基于对象的关系型数据库，Oracle 产品免费，服务收费。
- Oracle 安装后默认会有两个管理员用户（system，sys）和一个普通用户 Scott。
- Sql\*plus 是 Oracle 管理和数据操作的客户端工具。
- 客户端链接服务器前，服务器要启动监听服务，并且客户端工具要安装 Oracle 客户端，并且在客户端要建立本地网络服务名。
- Oracle 服务和监听启动后才能对数据库进行操作。
- 用 startup 命令启动数据库，用 shutdown 命令关闭数据库。
- Oracle 的角色包括了一系列系统权限和普通对象权限，可以把权限授权给角色，把权限或者角色授权给用户。

## 8. 本章练习

---

1. 描述 Oracle 安装过程中的关键点。
2. 描述创建本地网络服务名的步骤。
3. 描述 Oracle 主要服务的作用。
4. Oracle 使用什么命令才能启动和关闭。
5. 什么是 Oracle 权限和角色？他们的关系是什么？
6. 创建一个用户，并授权 CONNECT 和 RESOURCE。

# 章节知识结构图





# 第 2 章

---

## SQL 数据操作和查询

### 主要内容

- ✓ Oracle 数据类型
- ✓ SQL 建表和约束
- ✓ SQL 对数据增删改
- ✓ SQL 查询
- ✓ Oracle 伪列

# 1. SQL 简介

---

在第一学期的 SQL Server 学习中，已经知道，SQL 是结构化查询语言（Structured Query Language），专门用于数据存取、数据更新及数据库管理等操作。并且已经学习了用 SQL 语句对数据库的表进行增删改查的操作。

在 Oracle 开发中，客户端把 SQL 语句发送给服务器，服务器对 SQL 语句进行编译、执行，把执行的结果返回给客户端。Oracle SQL 语句由如下命令组成：

- 数据定义语言（DDL），包括 CREATE（创建）命令、ALTER（修改）命令、DROP（删除）命令等。
- 数据操纵语言（DML），包括 INSERT（插入）命令、UPDATE（更新）命令、DELETE（删除）命令、SELECT ... FOR UPDATE（查询）等。
- 数据查询语言（DQL），包括基本查询语句、Order By 子句、Group By 子句等。
- 事务控制语言（TCL），包括 COMMIT（提交）命令、SAVEPOINT（保存点）命令、ROLLBACK（回滚）命令。
- 数据控制语言（DCL），GRANT（授权）命令、REVOKE（撤销）命令。

目前主流的数据库产品（比如：SQL Server、Oracle）都支持标准的 SQL 语句。数据定义语言，表的增删改操作，数据的简单查询，事务的提交和回滚，权限的授权和撤销等，Oracle 与 SQL Server 在操作上基本一致。

# 2. Oracle 数据类型

---

Oracle 数据库的核心是表，表中的列使用到的常见数据类型如下：

类型	含义
CHAR(length)	存储固定长度的字符串。参数 length 指定了长度，如果存储的字符串长度小于 length，用空格填充。默认长度是 1，最长不超过 2000 字节。
VARCHAR2(length)	存储可变长度的字符串。length 指定了该字符串的最大长度。默认长度是 1，最长不超过 4000 字符。
NUMBER(p, s)	既可以存储浮点数，也可以存储整数，p 表示数字的最大位数（如果是小数包括整数部分和小数部分和小数点，p 默认是 38 为），s 是指小数位数。
DATE	存储日期和时间，存储纪元、4 位年、月、日、时、分、秒，存储时间

	从公元前 4712 年 1 月 1 日到公元后 4712 年 12 月 31 日。
TIMESTAMP	不但存储日期的年月日，时分秒，以及秒后 6 位，同时包含时区。
CLOB	存储大的文本，比如存储非结构化的 XML 文档
BLOB	存储二进制对象，如图形、视频、声音等。

表 1 Oracle 的部分数据类型

对应 NUMBER 类型的示例：

格式	输入的数字	实际的存储
NUMBER	1234.567	1234.567
NUMBER (6, 2)	123.4567	123.46
NUMBER (4, 2)	12345.67	输入的数字超过了所指定的精度，数据库不能存储

表 2 Number 示例

对于日期类型，可以使用 sysdate 内置函数可以获取当前的系统日期和时间，返回 DATE 类型，用 systimestamp 函数可以返回当前日期、时间和时区。

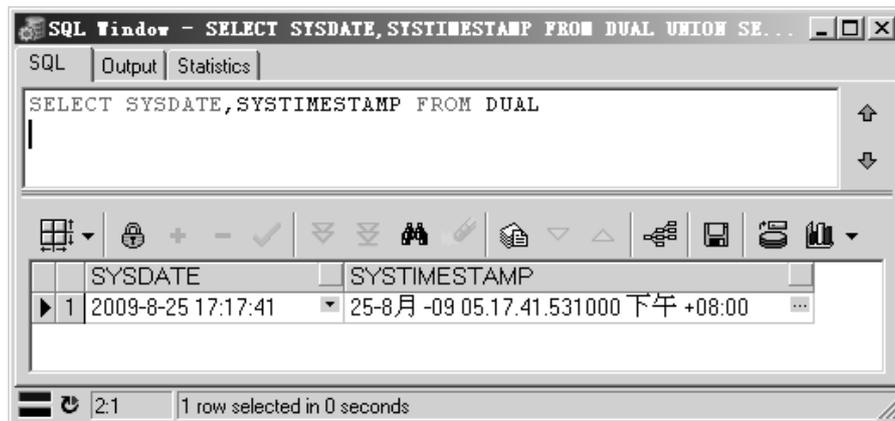


图 1 sysdate 和 sysTimestamp

Oracle 的查询中，必须使用“select 列... from 表”的完整语法，当查询单行函数的时候，from 后面使用 DUAL 表，dual 表在系统中只有一行一列，该表在输出单行函数时为了 select...from 的语法完整性而使用。

### 3. 创建表和约束

Oracle 创建表同 SQL Server 一样，使用 CREATE TABLE 命令来完成。创建约束则使用如下命令：

## 语法格式：ALTER TABLE 命令

---

ALTER TABLE 表名 ADD CONSTRAINT 约束名 约束内容。

---

不论创建表还是约束，与 SQL Server 基本相同，注意：在 Oracle 中 default 是一个值，而 SQL Server 中 default 是一个约束，因此 Oracle 的 default 设置可以在建表的时候创建。

### 案例 1：创建一个学生信息（INFOS）表和约束

#### 代码演示：Oracle 创建表和约束

---

```
CREATE TABLE INFOS
(
  STUID VARCHAR2(7) NOT NULL,    --学号 学号= 'S' +班号+2位序号
  STUNAME VARCHAR2(10) NOT NULL, --姓名
  GENDER VARCHAR2(2) NOT NULL,   --性别
  AGE NUMBER(2) NOT NULL,        --年龄
  SEAT NUMBER(2) NOT NULL,       --座号
  ENROLLDATE DATE,              --入学时间
  STUADDRESS VARCHAR2(50) DEFAULT '地址不详', --住址
  CLASSNO VARCHAR2(4) NOT NULL   --班号 班号=学期序号+班级序号
)
/ ①
ALTER TABLE INFOS ADD CONSTRAINT PK_INFOS PRIMARY KEY(STUID) ②
/
ALTER TABLE INFOS ADD CONSTRAINT CK_INFOS_GENDER
CHECK(GENDER = '男' OR GENDER = '女') ③
/
ALTER TABLE INFOS ADD CONSTRAINT CK_INFOS_SEAT
CHECK(SEAT >=0 AND SEAT <=50) ④
/
ALTER TABLE INFOS ADD CONSTRAINT CK_INFOS_AGE
CHECK(AGE >=0 AND AGE<=100) ⑤
/
ALTER TABLE INFOS ADD CONSTRAINT CK_INFOS_CLASSNO
CHECK(((CLASSNO >='1001' AND CLASSNO<='1999') OR
(CLASSNO >='2001' AND CLASSNO<='2999')) ⑥
/
```

---

---

```
ALTER TABLE INFOS ADD CONSTRAINTS UN_STUNAME UNIQUE(STUNAME) ⑦  
/
```

---

### 代码解析：

- ① 在 Oracle 代码中，“/” 执行缓存区中的语句，由于缓冲区中只存储一条刚刚保存过语句，由于每条语句没有用分号结尾，只是保存在缓冲区，因此每条语句后面都有单独一行“/”。
- ② 创建一个主键约束。
- ③ 与 ④ ⑤ ⑥ ⑦一起创建各种 check 约束。其中⑦是唯一约束，表示该列值是唯一的，列中的值不能重复。

Oracle 中创建外键约束与 SQL Server 相同。比如：现有成绩表定义如下：

### 案例 2：创建一个成绩表（SCORES）表和约束

#### 代码演示：Oracle 创建表和约束

---

```
CREATE TABLE SCORES  
(  
    ID NUMBER,          --ID ①  
    TERM VARCHAR2(2),   --学期 S1或S2  
    STUID VARCHAR2(7) NOT NULL, --学号  
    EXAMNO VARCHAR2(7) NOT NULL, --考号 E+班号+序号  
    WRITTENSORE NUMBER(4,1) NOT NULL, --笔试成绩  
    LABSCORE NUMBER(4,1) NOT NULL --机试成绩  
)  
ALTER TABLE SCORES  
    ADD CONSTRAINT CK_SCORES_TERM CHECK(TERM = 'S1' OR TERM ='S2')  
/  
ALTER TABLE SCORES  
    ADD CONSTRAINT FK_SCORES_INFOS_STUID FOREIGN KEY(STUID) REFERENCES  
INFOS(STUID) ②  
/
```

---

### 代码解析：

- ① SQL Server 中可以使用 identify 创建自动增长列，但是 Oracle 中的自动增长需要借助序列（Sequence）完成，在后面章节中讲解。
- ② Oracle 中的外键约束定义。

## 4. 数据操纵语言 ( DML )

数据操纵语言 (DML) 用于对数据库的表中数据进行添加、修改、删除和 SELECT...For UPDATE(后面专门学习该查询)操作。对比一期学习过的 SQL Server 操作，接下来一一介绍在 Oracle 中的操作。

### ◇ 简单查询

数据查询是用 SELECT 命令从数据库的表中提取信息。SELECT 语句的语法是：

#### 语法结构：简单查询

```
SELECT *|列名|表达式 FROM 表名 WHERE 条件 ORDER BY 列名
```

#### 语法解析：

1. \*表示表中的所有列。
2. 列名可以选择若干个表中的列名，各个列表中间用逗号分隔。
3. 表达式可以是列名、函数、常数等组成的表达式。
4. WHERE 子句是查询的条件。
5. ORDER BY 要求在查询的结果中排序，默认是升序。



图2 数据查询

Oracle 中可以把查询的结果根据结果集中的表结构和数据形成一张新表。

### 语法结构：根据结果集创建表

---

```
CREATE TABLE 表名 AS SELECT 语句
```

---

### 代码演示：根据结果集创建表

---

```
SQL> CREATE TABLE INFOS1 AS SELECT * FROM INFOS;  
TABLE CREATED
```

---

使用上面命令创建的新表中，不存在任何约束，并且把查询的数据一起插入到新表中。如果只复制表结构，只需使查询的条件不成立（比如 `where 1=2`），就不会查询出任何数据，从而复制一个表结构。

### 代码演示：复制表结构

---

```
SQL> CREATE TABLE INFOS2 AS SELECT * FROM INFOS WHERE 1=2;  
TABLE CREATED
```

---

## ✧ 数据插入

用 INSERT 命令完成对数据的插入。

### 语法结构：根据结果集创建表

---

```
INSERT INTO 表名(列名 1, 列名 2.....) VALUES (值 1, 值 2.....)
```

---

### 语法解析：

1. 列名可以省略。当省略列名时，默认是表中的所有列名，列名顺序为表定义中列的先后顺序。
2. 值的数量和顺序要与列名的数量和顺序一致。值的类型与列名的类型一致。

### 代码演示：向 INFOS 表和 SCORES 表中插入数据

---

```
SQL> INSERT INTO INFOS VALUES (  
1 's100102', '林冲', '男', 22, 2,  
2 TO_DATE('2009-8-9 06:30:10','YYYY-MM-DD HH24:MI:SS '),  
3
```

---

---

```
4 '西安', '1001'
5 )
6 /
1 row inserted
SQL> INSERT INTO INFOS VALUES (
        's100104','阮小二','男',26,3,SYSDATE,default,'1001'); ③
1 row inserted
SQL>COMMIT; ④
```

---

### 代码解析：

- ① 表名后面缺省了列名，默认是表 `Infos` 中的所有列名，`values` 中的值要与表中列一一对应，包括顺序和数据类型的对应。在 `SQL*Plus` 中一条语句可以写在多行，那么从第二行开始，`sqlplus` 会为每一行前面给出行号。
- ② 在 `Oracle` 中，日期是国际化的，不同的区域安装的数据库，默认的日期格式不同，因此为了程序便于移植，日期的输入要使用 `TO_DATE` 函数对日期格式化后输入，采用格式化字符串对日期进行格式化时，格式化字符串中字符不区分大小写，常见的格式化字符如下：
  1. `yyyy` 表示四位年份
  2. `mm` 表示两位月份，比如 3 月表示为 `03`
  3. `dd` 表示两位日期
  4. `hh24` 表示小时从 `0-23`，`hh12` 也表示小时从 `0-11`。
  5. `mi` 表示分钟
  6. `ss` 表示秒
- ③ 在遇到存在默认值的列时，可以使用 `default` 值代替。
- ④ `commit` 是把用户操作（添加、删除、修改操作）提交，只有提交操作后，数据才能真正更新到表中，否则其他用户无法查询到当前用户操作的结果。

在 `Oracle` 中，一个 `INSERT` 命令可以把一个结果集一次性插入到一张表中。使用的语句是：`INSERT INTO 表 SELECT 子句`，如下示例：

### 代码演示：INSERT 向表中插入一个结果集

---

```
SQL> INSERT INTO INFOS2 SELECT * FROM INFOS;
5 rows inserted
```

---

在这种语法下，要求结果集中每一列的数据类型必须与表中的每一列的数据类型一致，结果集中的列的数量与表中的列的数量一致。比如表 `INFOS2`，该表的结构与 `INFO` 表一样，那么可以把 `INFO` 表中的所有记录一次性插入到 `INFOS2` 表中。

Oracle 的简单查询和 SQL Server 一样都可以在查询列中使用常量，如图：

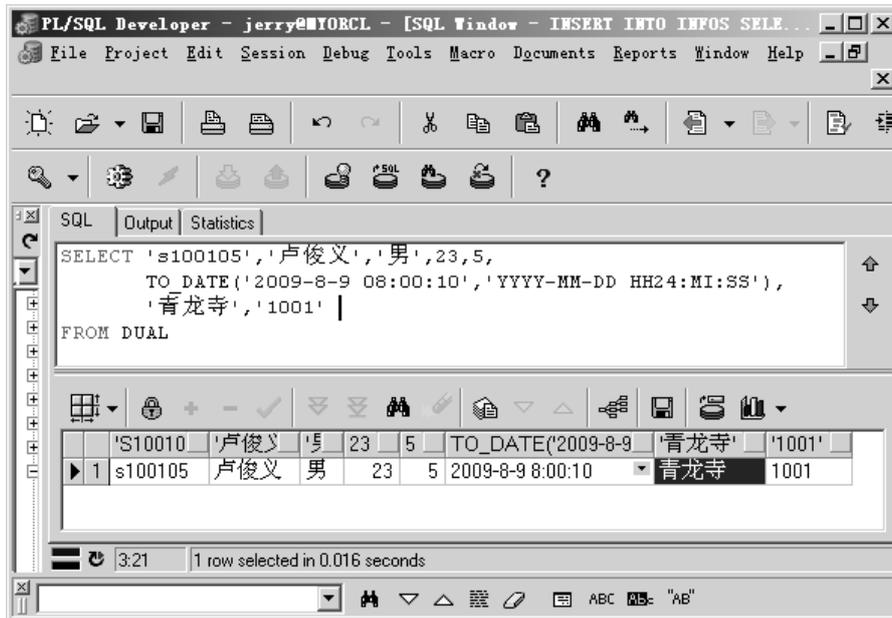


图3 Select 中的常量

可以使用刚才的做法，把该结果集中的数据插入到表 INFOS 中。

### 代码演示：INSERT 向表中插入一个常量结果集

```
SQL> INSERT INTO INFOS  
      SELECT 's100106', '卢俊义', '男', 23, 5,  
            TO_DATE('2009-8-9 08:00:10', 'YYYY-MM-DD HH24:MI:SS'),  
            '青龙寺', '1001'  
      FROM DUAL;  
1 rows inserted  
SQL> COMMIT;
```

### ◇ 更新数据

Oracle 在表中更新数据的语法是：

### 语法结构：UPDATE 操作

```
UPDATE 表名 SET 列名 1=值, 列名 2=值..... WHERE 条件
```

### 代码演示：UPDATE 操作

---

```
SQL> UPDATE INFOS SET CLASSNO='1002',STUADDRESS='山东莱芜'
      WHERE STUNAME='阮小二';
1 rows updated
SQL> commit;
```

---

## ❖ 删除数据

Oracle 在表中删除数据的语法是：

### 语法结构：DELETE 操作

---

```
DELETE FROM 表名 WHERE 条件
```

---

### 代码演示：DELETE 操作

---

```
SQL> DELETE FROM INFOS WHERE STUID='s100103';
1 ROW DELETED
SQL> COMMIT;
```

---

## ❖ TRUNCATE

在数据库操作中，TRUNCATE 命令（是一个 DDL 命令）可以把表中的所有数据一次性全部删除，语法是：

### 语法结构：TRUNCATE

---

```
TRUNCATE TABLE 表名
```

---

TRUNCATE 和 DELETE 都能把表中的数据全部删除，他们的区别是：

1. TRUNCATE 是 DDL 命令，删除的数据不能恢复；DELETE 命令是 DML 命令，删除后的数据可以通过日志文件恢复。
2. 如果一个表中数据记录很多，TRUNCATE 相对 DELETE 速度快。

由于 TRUNCATE 命令比较危险，因此在实际开发中，TRUNCATE 命令慎用。



提示

Oracle 默认安装中，已经创建了一个 SCOTT 用户，默认密码是：tiger，该用户下有四张表分别是：雇员表（EMP），部门表（DEPT），工资登记表和奖金表，请参考本章后面的附表。接下来很多操作都是在该用户下完成的。

## 5. 操作符

---

Oracle 开发中，依然存在算术运算，关系运算，和逻辑运算。

### ◇ 算术运算

Oracle 中的算术运算符，没有 C# 中的算术运算符丰富，只有 +、-、\*、/ 四个，其中除号 (/) 的结果是浮点数。求余运算只能借助函数：MOD(x,y)：返回 x 除以 y 的余数。

**案例 3：**每名员工年终奖是 2000 元，请显示基本工资在 2000 元以上的员工的月工资，年总工资。

该案例的表请参见本章练习的附表 1、附表 2、附表 3，这三张表是 ORACLE 10g 自带的。

### 代码演示：查询中的算术运算

---

```
SQL> SELECT ENAME,SAL,(SAL*12+2000) FROM EMP WHERE SAL>2000;
```

ENAME	SAL	(SAL*12+2000)
JONES	2975	37700
BLAKE	2850	36200
CLARK	2450	31400
SCOTT	3000	38000
KING	5000	62000
FORD	3000	38000

6 rows selected

---

### ◇ 关系运算和逻辑运算

Oracle 中 Where 子句中经常见到关系运算和逻辑运算，常见的关系运算有：

运算符	说明	运算符	说明
=	等于	>	大于
<>或者!=	不等于	<=	小于或者等于
<	小于	>=	大于或者等于

表 3 Oracle 的关系运算符

逻辑运算符有三个：AND、OR、NOT

关系运算和逻辑运算与前面 SQL Server 学习过的一致。

## ◇ 字符串连接操作符 (||)

在 Oracle 中，字符串的连接用双竖线 (||) 表示。比如，在 EMP 表中，查询工资在 2000 元以上的姓名以及工作。

### 代码演示：字符串连接

---

```
SQL> SELECT (ENAME || 'is a ' || JOB) AS "Employee Details" ①
2 FROM EMP
3 WHERE SAL>2000;
Employee Details
-----
JONESis a MANAGER
BLAKEis a MANAGER
CLARKis a MANAGER
SCOTTis a ANALYST
KINGis a PRESIDENT
FORDis a ANALYST
6 rows selected
```

---

### 代码解析：

- ① Oracle 中字符串可以用单引号，也可以用双引号，在别名中存在空格时，必须用双引号。在表名、列名时用双引号。

## 6. 高级查询

在第一期学习过 SQL 的简单查询和连接查询。现在学习一些新的 SQL 操作符。

## ◇ 消除重复行

在 Oracle 查询中结果中，可能出现若干行相同的情况，那么可以使用 DISTINCT 消除重复行。具体的用法如示例：

### 代码演示：DISTINCT 消除重复行

---

```
SQL> SELECT DISTINCT DEPTNO FROM EMP;
DEPTNO
-----
      30
      20
      10
```

---

## ◇ NULL 操作

如果某条记录中有缺少的数据值，就是空值（NULL 值）。空值不等于 0 或者空格，空值是指未赋值、未知或不可用的值。任何数据类型的列都可以包括 NULL 值，除非该列被定义为非空或者主键。

### 代码演示：EMP 中的 NULL 值

---

```
SQL> SELECT ENAME, JOB, SAL, COMM FROM EMP WHERE SAL < 2000;

ENAME      JOB          SAL      COMM
-----
SMITH      CLERK        800
ALLEN      SALESMAN     1600     300
WARD       SALESMAN     1250     500
MARTIN     SALESMAN     1250     1400
TURNER     SALESMAN     1500      0
ADAMS      CLERK        1100
JAMES      CLERK         950

7 rows selected
```

---

在查询条件中 NULL 值用 IS NULL 作条件，非 NULL 值用 NOT IS NULL 做条件。

**案例 4**：查询 EMP 表中没有发奖金的员工。

### 代码演示：NULL 值查询

---

```
SQL> SELECT ENAME,JOB,SAL,COMM FROM EMP  
2 WHERE SAL<2000 AND COMM IS NULL;
```

ENAME	JOB	SAL	COMM
SMITH	CLERK	800	
ADAMS	CLERK	1100	
JAMES	CLERK	950	
MILLER	CLERK	1300	

---

### ◇ IN 操作

在 Where 子句中可以使用 IN 操作符来查询其列值在指定的列表中的行。比如：查询出工作职责是 SALESMAN、PRESIDENT 或者 ANALYST 的员工。条件有两种表示方法：

1. WHERE job = 'SALESMAN' OR job = 'PRESIDENT' OR job = 'ANALYST'
2. WHERE job IN ('SALESMAN', 'PRESIDENT', 'ANALYST')

### 代码演示：IN 操作

---

```
SQL> SELECT ENAME,JOB,SAL FROM EMP  
2 WHERE job IN ('SALESMAN', 'PRESIDENT', 'ANALYST');
```

ENAME	JOB	SAL
ALLEN	SALESMAN	1600
WARD	SALESMAN	1250
MARTIN	SALESMAN	1250
SCOTT	ANALYST	3000
KING	PRESIDENT	5000
TURNER	SALESMAN	1500
FORD	ANALYST	3000

7 rows selected

---

对应 IN 操作的还有 NOT IN，用法一样，结果相反。

## ◇ BETWEEN...AND...

在 WHERE 子句中，可以使用 BETWEEN 操作符来查询列值包含在指定区间内的行。比如，查询工资从 1000 到 2000 之间的员工。可以使用传统方法：

```
WHERE SAL>=1000 AND SAL<=2000
```

也可以使用：

```
WHERE SAL BETWEEN 1000 AND 2000
```

BWTWEEN 操作所指定的范围也包括边界。

### 代码演示：BETWEEN 操作

```
SQL> SELECT ename,job,sal FROM EMP WHERE sal BETWEEN 1000 AND 2000;
```

ENAME	JOB	SAL
ALLEN	SALESMAN	1600
WARD	SALESMAN	1250
MARTIN	SALESMAN	1250
TURNER	SALESMAN	1500
ADAMS	CLERK	1100
MILLER	CLERK	1300

6 rows selected

## ◇ LIKE 模糊查询

在一些查询时，可能把握不准需要查询的确切值，比如百度搜索时输入关键字即可查询出相关的结果，这种查询称为模糊查询。模糊查询使用 LIKE 关键字通过字符匹配检索出所需要的数据行。字符匹配操作可以使用通配符“%”和“\_”：

- %：表示零个或者多个任意字符。
- \_：代表一个任意字符。

语法是：LIKE '字符串'[ESCAPE '字符']。匹配的字符串中，ESCAPE 后面的“字符”作为转义字符。与一期 SQLServer 中 ESCAPE 用法相同。

通配符表达式	说明
'S%'	以 S 开头的字符串。

'_S%'	第二个字符时 S 的字符串。
'%30\%%%' escape '\'	包含“30%”的字符串，“\”指转义字符，“\%”在字符串中表示一个字符“%”。

表 4 通配符示例

**案例 5：**显示员工名称以 J 开头以 S 结尾的员工的姓名、工资和工资。

### 代码演示：LIKE 操作

```
SQL> SELECT ENAME, JOB, SAL FROM EMP WHERE ENAME LIKE 'J%S';
```

ENAME	JOB	SAL
JONES	MANAGER	2975.00
JAMES	CLERK	950.00

## ◇ 集合运算

集合运算就是将两个或者多个结果集组合成为一个结果集。集合运算包括：

- INTERSECT(交集)，返回两个查询共有的记录。
- UNION ALL(并集)，返回各个查询的所有记录，包括重复记录。
- UNION(并集)，返回各个查询的所有记录，不包括重复记录。
- MINUS(补集)，返回第一个查询检索出的记录减去第二个查询检索出的记录之后剩余的记录。

当使用集合操作的时候，要注意：查询所返回的列数以及列的类型必须匹配，列名可以不同。

**案例 6：**查询出 dept 表中哪个部门下没有员工。只需求出 dept 表中的部门号和 emp 表中的部门号的补集即可。

### 代码演示：求补运算

```
SQL> SELECT DEPTNO FROM DEPT
2 MINUS
3 SELECT DEPTNO FROM EMP;
DEPTNO
```

前面学习过可以通过 insert into ...select 把一个结果集插入到另一张结构相同的表中，因此可以使用 union 把若干条记录一次性插入到一张表中。

### 代码演示：用 union 插入多条数据

```
SQL> INSERT INTO DEPT
  2  SELECT 50,'公关部','台湾' FROM DUAL
  3  UNION
  4  SELECT 60,'研发部','西安' FROM DUAL
  5  UNION
  6  SELECT 70,'培训部','西安' FROM DUAL
  7  /
3 rows inserted
```

### ◇ 连接查询

在 SQL Server 中已经学习过内联接(inner join)、外联接(outer join)，外联接又分为左外联接(left outer join)和右外联接(right outer join)。Oracle 中对两个表或者若干表之间的外联接用 (+) 表示。

**案例 7：**请查询出工资大于 2000 元的，员工姓名，部门，工作，工资。

由于部门名称在 dept 中，其他的信息在 emp 表中，需要内联接才能完成。

### 代码演示：内联接

```
SQL> SELECT e.ENAME,e.JOB,e.SAL,d.DNAME
  2  FROM emp e,dept d
  3  WHERE e.deptno=d.deptno
  4  AND e.SAL>2000;
```

ENAME	JOB	SAL	DNAME
JONES	MANAGER	2975	RESEARCH
BLAKE	MANAGER	2850	SALES
CLARK	MANAGER	2450	ACCOUNTING

---

SCOTT	ANALYST	3000	RESEARCH
KING	PRESIDENT	5000	ACCOUNTING
FORD	ANALYST	3000	RESEARCH

6 rows selected

---

也可以使用 SQL/92 标准中的内联接:

### 代码演示：内联接

---

```
SELECT e.ENAME,e.JOB,e.SAL,d.DNAME
FROM EMP e INNER JOIN DEPT d ON e.DEPTNO=d.DEPTNO
WHERE e.SAL>2000
```

---

这里 INNER JOIN 中，关键字 INNER 可以省略。

**案例 8：**请查询出每个部门下的员工姓名，工资。

### 案例分析：

Emp 表用外键 deptno 引用 Dept 表中的 deptno,在 Dept 表中如果有某些部门没有员工,那么用内联接,没有员工的部门将无法显示,因此必须以 Dept 表为基准的外联接。

### 代码演示：外联接

---

```
SQL> SELECT e.ENAME,e.JOB,e.SAL,d.DNAME
2 FROM EMP e ,DEPT d
3 WHERE e.DEPTNO(+)=d.DEPTNO ①
4 /
```

ENAME	JOB	SAL	DNAME
SMITH	CLERK	800	RESEARCH
ALLEN	SALESMAN	1600	SALES
WARD	SALESMAN	1250	SALES
JONES	MANAGER	2975	RESEARCH
MARTIN	SALESMAN	1250	SALES
BLAKE	MANAGER	2850	SALES
CLARK	MANAGER	2450	ACCOUNTING
SCOTT	ANALYST	3000	RESEARCH

---

---

KING	PRESIDENT	5000	ACCOUNTING
TURNER	SALESMAN	1500	SALES
ADAMS	CLERK	1100	RESEARCH
JAMES	CLERK	950	SALES
FORD	ANALYST	3000	RESEARCH
MILLER	CLERK	1300	ACCOUNTING

公关部

研发部

培训部

OPERATIONS

---

18 rows selected

---

### 代码解析：

- ① (+): Oracle 专用的联接符，在条件中出现在左边指右外联接，出现在右边指左外联接。

也可以使用 SQL/92 标准的写法：

### 代码演示：外联接

---

```
SELECT e.ENAME,e.JOB,e.SAL,d.DNAME  
FROM EMP e RIGHT OUTER JOIN DEPT d ON e.DEPTNO=d.DEPTNO
```

---

这里 RIGHT OUTER JOIN 中，关键字 OUTER 可以省略。



提示

虽然 Oracle 自身的联接查询语法比较好写，同时容易理解，但是为了程序便于移植，推荐使用 SQL/92 表中的联接查询。同时也可以与 SQL Server 获得一致。

## 7. 本章总结

---

- Oracle SQL 语句中有数据操纵语言 (DML)、数据定义语言 (DDL)、数据控制语言 (DCL)、事务控制语言 (TCL) 等等。
- DML 语句包括增删改查语句, DDL 语句包括数据库对象创建、修改和删除语句, 数据控制命令包括 GRANT、REVOKE 等, 事务控制命令有 COMMIT、ROLLBACK 等。
- 数据库中建表常用的类型有: 数字类型 number(p,s), 可变字符串 varchar2(length), 日期 date。
- Oracle 中 default 是一个值, 在 Oracle 中不存在 default 约束。
- Oracle 的增删改语句与 SQL Server 基本一致, 都是使用 INSERT、UPDATE、DELETE 完成。
- Oracle 高级查询中要注意: DISTINCT、NULL、IN、BETWEEN...AND...。
- 集合操作有: UNION、UNION ALL、INTERSECT、MINUS。
- 联接查询有内联接和外联接。

## 8. 本章练习

---

1. 创建一查询，显示与 Blake 在同一部门工作的雇员的项目和受雇日期，但是 Blake 不包含在内。
2. 显示位置在 Dallas 的部门内的雇员姓名、变化以及工作。
3. 显示被 King 直接管理的雇员的姓名以及工资。
4. 创建一查询，显示能获得与 Scott 一样工资和奖金的其他雇员的姓名、受雇日期以及工资。

附表 1: Scott 表中的 EMP 表: 员工表

序号	列名	类型	说明
1	EMPNO	NUMBER(4)	员工编号, EMP 表主键
2	ENAME	VARCHAR2(10)	员工姓名
3	JOB	VARCHAR2(9)	员工工作
4	MGR	NUMBER(4)	员工的领导编号, 引用 EMPNO
5	HIREDATE	DATE	入职日期
6	SAL	NUMBER(7,2)	员工工资
7	COMM	NUMBER(7,2)	员工奖金
8	DEPTNO	NUMBER(2)	员工部门编号, 是表 DEPT 的外键。

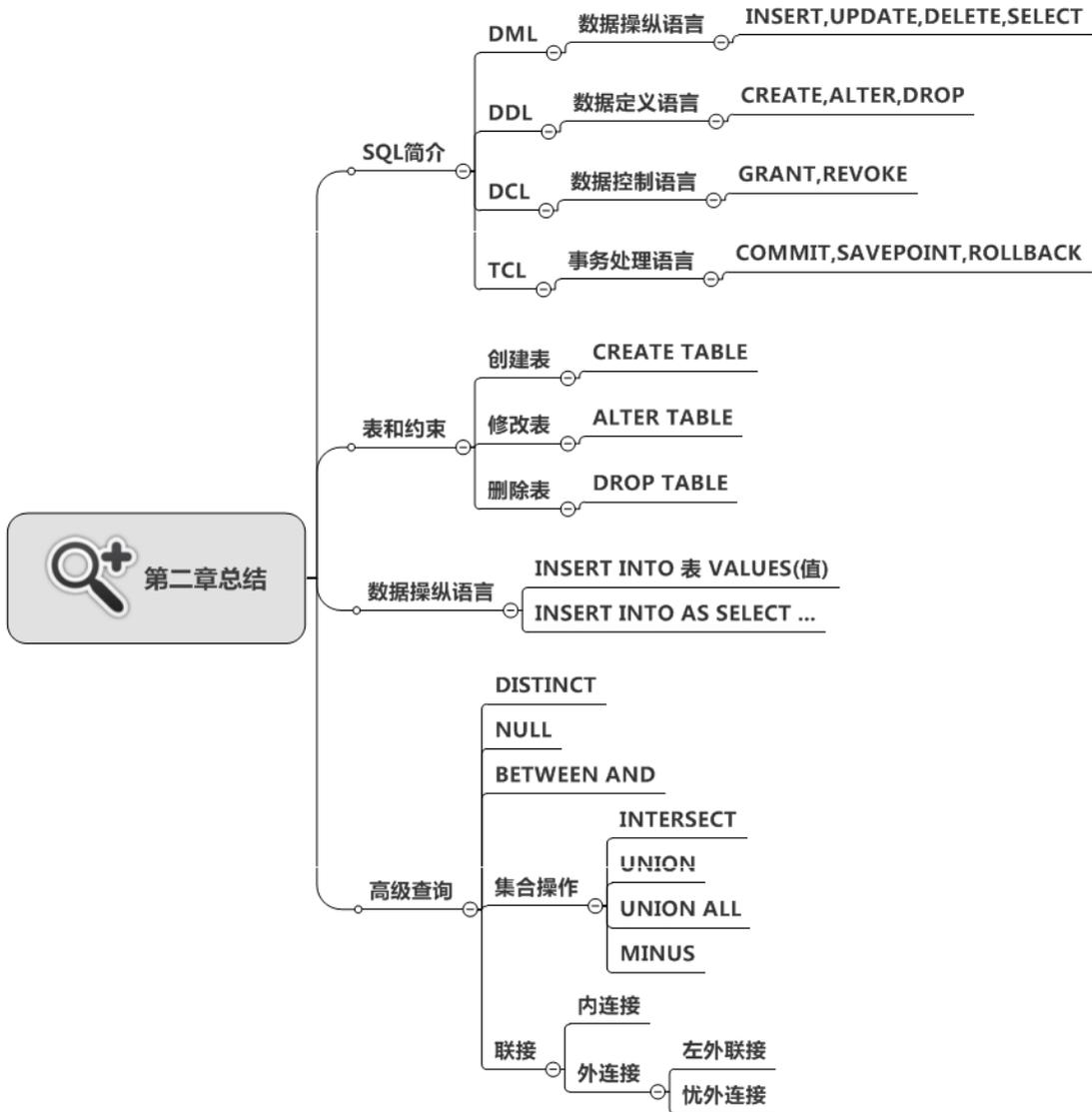
附表 2: Scott 表中的 DEPT 表: 部门表

序号	列名	类型	说明
1	DEPTNO	NUMBER(2)	部门编号, 主键
2	DNAME	VARCHAR2(14)	部门名称
3	LOC	VARCHAR2(13)	部门位置

附表 3: Scott 表中的 SALGRADE 表: 工资等级表

序号	列名	类型	说明
1	GRADE	NUMBER	等级
2	LOSAL	NUMBER	此等级下最低工资
3	HISAL	NUMBER	此等级下最高工资

# 章节知识结构图





# 第 3 章

---

## 子查询和常用函数

### 主要内容

- ✓ 子查询
- ✓ 伪列
- ✓ 锁的概念

# 1. 子查询

---

子查询在 SELECT、UPDATE、DELETE 语句内部可以出现 SELECT 语句。内部的 SELECT 语句结果可以作为外部语句中条件子句的一部分，也可以作为外部查询的临时表。子查询的类型有：

1. 单行子查询：不向外部返回结果，或者只返回一行结果。
2. 多行子查询：向外部返回零行、一行或者多行结果。

**案例 1：**查询出销售部（SALES）下面的员工姓名，工作，工资。

## 案例分析

该问题可以用联接查询实现，由于所需的结果信息都在 Emp 表中，可以先从 Dept 表中查询出销售部对应的部门号，然后根据当前部门号再到 Emp 表中查询出符合该部门的员工记录即可。从销售表中查询出的结果可以作为 Emp 表中查询的条件，SQL 语句实现如下：

## 代码演示：单行子查询

---

```
SQL> SELECT ENAME,JOB,SAL FROM EMP
  2  WHERE DEPTNO=(SELECT DEPTNO FROM DEPT WHERE DNAME='SALES') ①
  3  /
```

ENAME	JOB	SAL
ALLEN	SALESMAN	1600
WARD	SALESMAN	1250
MARTIN	SALESMAN	1250
BLAKE	MANAGER	2850
TURNER	SALESMAN	1500
JAMES	CLERK	950

6 rows selected

---

## 代码解析：

① 内部查询的结果作为外部查询的条件。

需要注意：

- 如果内部查询不返回任何记录,则外部条件中字段 DEPTNO 与 NULL 比较永远为假,也就是说外部查询不返还任何结果。
- 在单行子查询中外部查询可以使用=、>、<、>=、<=、<>等比较运算符。
- 内部查询返回的结果必须与外部查询条件中的字段 (DEPTNO) 匹配。
- 如果内部查询返回多行结果则出现错误。

**案例 2 :** 查询出 Emp 表中比任意一个销售员(“SALESMAN”)工资低的员工姓名、工作、工资。

### 案例分析

销售员在 Emp 表中有很多条记录,每个人工资不相等,如果返回“比任意员工的工资还低”的条件,返回比“最高工资还低”即可。如果用子查询做,子查询中就会返回多条记录。用普通的关系符(>、<等)运行就会出错。这时候需要用关键字 ANY。ANY 放在比较运算符后面,表示“任意”的意思。

### 代码演示 : ANY 子查询

---

```
SQL> SELECT ENAME,JOB,SAL FROM EMP
  2  WHERE SAL<ANY (SELECT SAL FROM EMP WHERE JOB='SALESMAN') ①
  3  /
```

ENAME	JOB	SAL
SMITH	CLERK	800
JAMES	CLERK	950
ADAMS	CLERK	1100
WARD	SALESMAN	1250
MARTIN	SALESMAN	1250
MILLER	CLERK	1300
TURNER	SALESMAN	1500

7 rows selected

---

### 代码解析 :

- ① <any:比子查询结果中任意的值都小,也就是说,比子查询结果中最大值还小,那么同理>any 表示比子查询结果中最小的还大。

**案例 3 :** 查询出比所有销售员的工资都高的员工姓名,工作,工资。

## 案例分析

ANY 可以表示任意的，但本案例中要求比所有销售员工资都高，那么就要使用另外一个关键字 ALL。ALL 与关系操作符一起使用，表示与子查询中所有元素比较。

## 代码演示：ALL 子查询

```
SQL> SELECT ENAME, JOB, SAL FROM EMP
2  WHERE SAL > ALL (SELECT SAL FROM EMP WHERE JOB = 'SALESMAN') ①
3  /
```

ENAME	JOB	SAL
JONES	MANAGER	2975
BLAKE	MANAGER	2850
CLARK	MANAGER	2450
SCOTT	ANALYST	3000
KING	PRESIDENT	5000
FORD	ANALYST	3000

6 rows selected

## 代码解析：

- ① >ALL：比子查询结果中所有值还要大，也就是说，比子查询结果中最大值还要大。  
<ALL 表示比最小值还要小。

对于子查询还可以使用 IN 和 NOT IN 操作符进行操作。

## 2. Oracle 中的伪列

在 Oracle 的表的使用过程中，实际表中还有一些附加的列，称为伪列。伪列就像表中的列一样，但是在表中并不存储。伪列只能查询，不能进行增删改操作。接下来学习两个伪列：ROWID 和 ROWNUM。

### ✧ ROWID

表中的每一行在数据文件中都有一个物理地址，ROWID 伪列返回的就是该行的物理地址。使用 ROWID 可以快速的定位表中的某一行。ROWID 值可以唯一的标识表中的一行。由于 ROWID 返回的是该行的物理地址，因此使用 ROWID 可以显示行是如何存储的。

## 代码演示：ROWID

```
SQL> SELECT ROWID,ENAME FROM EMP WHERE SAL>2000;
```

ROWID	ENAME
AAAMgzAAEAAAAAgAAD	JONES
AAAMgzAAEAAAAAgAAF	BLAKE
AAAMgzAAEAAAAAgAAG	CLARK
AAAMgzAAEAAAAAgAAH	SCOTT
AAAMgzAAEAAAAAgAAI	KING
AAAMgzAAEAAAAAgAAM	FORD

6 rows selected

## ◇ ROWNUM

在查询的结果集中，ROWNUM 为结果集中每一行标识一个行号，第一行返回 1，第二行返回 2，以此类推。通过 ROWNUM 伪列可以限制查询结果集中返回的行数。



注意

ROWNUM 与 ROWID 不同，ROWID 是插入记录时生成，ROWNUM 是查询数据时生成。  
ROWID 标识的是行的物理地址。ROWNUM 标识的是查询结果中的行的次序。

**案例 4：**查询出员工表中前 5 名员工的姓名，工作，工资。

## 代码演示：ROWNUM

```
SQL> SELECT ROWNUM,ENAME,JOB,SAL FROM EMP WHERE ROWNUM<=5;
```

ROWNUM	ENAME	JOB	SAL
1	SMITH	CLERK	800
2	ALLEN	SALESMAN	1600
3	WARD	SALESMAN	1250
4	JONES	MANAGER	2975
5	MARTIN	SALESMAN	1250

**案例 5：**查询出工资最高的前 5 名员工的姓名、工资和工资。

### 案例分析

“工资最高的前 5 名”需要先降序排序，再取前 5 名，但是生成 ROWNUM 操作比排序要早，排序时已经连同 ROWNUM 一起排序了，因此不能直接在案例 1 的语句中直接加上 Order by 就行，而是需要对排序的结果重新做二次查询，产生新的 ROWNUM 才能作为查询的条件依据。

### 代码演示：ROWNUM 应用

---

```
SQL> SELECT ROWNUM,T.* FROM ①  
2   (SELECT ENAME,JOB,SAL  
3   FROM EMP ORDER BY SAL DESC) T ②  
4 WHERE ROWNUM<=5  
5 /
```

ROWNUM	ENAME	JOB	SAL
1	KING	PRESIDENT	5000
2	SCOTT	ANALYST	3000
3	FORD	ANALYST	3000
4	JONES	MANAGER	2975
5	BLAKE	MANAGER	2850

---

### 代码解析：

① T 是子查询②的别名，这里的 ROWNUM 是第二次查询后的 ROWNUM。

**案例 6：**查询出表 EMP 中第 5 条到第 10 条之间的记录。

### 案例分析

这是分页的应用。在查询条件中，如果查询条件中 ROWNUM 大于某一正整数，则不返回任何结果。

### 代码演示：ROWNUM 分页

---

```
SQL> SELECT * FROM  
2   (SELECT ROWNUM R,ENAME,JOB,SAL ①  
3   FROM EMP WHERE ROWNUM<=10) ②
```

---

---

```

4 WHERE R>5 ③
5 /

```

R	ENAME	JOB	SAL
6	BLAKE	MANAGER	2850
7	CLARK	MANAGER	2450
8	SCOTT	ANALYST	3000
9	KING	PRESIDENT	5000
10	TURNER	SALESMAN	1500

---

### 代码解析：

- ① 内部查询中得到 ROWNUM 并且用别名 R 记录，供外层条件③使用。
- ② 内部查询的 ROWNUM，与外出的 ROWNUM 列是平等的两列。
- ③ 使用的 R 是内层产生的 ROWNUM，在外层看来，内层查询的 ROWNUM 是正常的一列。

## 3. Oracle 函数

---

Oracle SQL 提供了用于执行特定操作的专用函数。这些函数大大增强了 SQL 语言的功能。函数可以接受零个或者多个输入参数，并返回一个输出结果。Oracle 数据库中主要使用两种类型的函数：

1. 单行函数：对每一个函数应用在表的记录中时，只能输入一行结果，返回一个结果，比如：MOD(x,y)返回 x 除以 y 的余数（x 和 y 可以是两个整数，也可以是表中的整数列）。常用的单行函数有：
  - 字符函数：对字符串操作。
  - 数字函数：对数字进行计算，返回一个数字。
  - 转换函数：可以将一种数据类型转换为另外一种数据类型。
  - 日期函数：对日期和时间进行处理。
2. 聚合函数：聚合函数同时可以对多行数据进行操作，并返回一个结果。比如 SUM(x) 返回结果集中 x 列的总合。

### ◇ 字符函数

字符函数接受字符参数，这些参数可以是表中的列，也可以是一个字符串表达式。下表列出了常用的字符函数。

函数	说明
----	----

ASCII(x)	返回字符 x 的 ASCII 码。
CONCAT(x,y)	连接字符串 x 和 y。
INSTR(x, str [,start] [,n])	在 x 中查找 str，可以指定从 start 开始，也可以指定从第 n 次开始。
LENGTH(x)	返回 x 的长度。
LOWER(x)	x 转换为小写。
UPPER(x)	x 转换为大写。
LTRIM(x[,trim_str])	把 x 的左边截去 trim_str 字符串，缺省截去空格。
RTRIM(x[,trim_str])	把 x 的右边截去 trim_str 字符串，缺省截去空格。
TRIM([trim_str FROM] x)	把 x 的两边截去 trim_str 字符串，缺省截去空格。
REPLACE(x,old,new)	在 x 中查找 old，并替换为 new。
SUBSTR(x,start[,length])	返回 x 的字符串，从 start 处开始，截取 length 个字符，缺省 length，默认到结尾。

表 1 字符函数

示例	示例结果
SELECT ASCII('a') FROM DUAL	97
SELECT CONCAT('Hello', ' world') FROM DUAL	Hello world
SELECT INSTR('Hello world', 'or') FROM DUAL	8
SELECT LENGTH('Hello') FROM DUAL	5
SELECT LOWER('hEllo') FROM DUAL;	hello
SELECT UPPER('hello') FROM DUAL	HELLO
SELECT LTRIM('===HELLO===', '=') FROM DUAL	HELLO===
SELECT '=='  LTRIM(' HELLO===') FROM DUAL	==HELLO===
SELECT RTRIM('===HELLO===', '=') FROM DUAL	===HELLO
SELECT '='  TRIM(' HELLO ')  '=' FROM DUAL	=HELLO=
SELECT TRIM('=') FROM '===HELLO===') FROM DUAL	HELLO
SELECT REPLACE('ABCDE','CD','AAA') FROM DUAL	ABAAAE
SELECT SUBSTR('ABCDE',2) FROM DUAL	BCDE
SELECT SUBSTR('ABCDE',2,3) FROM DUAL	BCD

表 2 字符函数示例

## ◇ 数字函数

数字函数接受数字参数，参数可以来自表中的一列，也可以是一个数字表达式。

函数	说明	示例
----	----	----

ABS(x)	x 绝对值	ABS(-3)=3
ACOS(x)	x 的反余弦	ACOS(1)=0
COS(x)	余弦	COS(1)=1.57079633
CEIL(x)	大于或等于 x 的最小值	CEIL(5.4)=6
FLOOR(x)	小于或等于 x 的最大值	FLOOR(5.8)=5
LOG(x,y)	x 为底 y 的对数	LOG(2,4)=2
MOD(x,y)	x 除以 y 的余数	MOD(8,3)=2
POWER(x,y)	x 的 y 次幂	POWER(2,3)=8
ROUND(x[,y])	x 在第 y 位四舍五入	ROUND(3.456,2)=3.46
SQRT(x)	x 的平方根	SQRT(4)=2
TRUNC(x[,y])	x 在第 y 位截断	TRUNC(3.456,2)=3.45

表 3 数字函数

**说明：**

1. ROUND(x[,y])，四舍五入。

在缺省 y 时，默认 y=0；比如：ROUND(3.56)=4。

y 是正整数，就是四舍五入到小数点后 y 位。ROUND(5.654,2)=5.65。

y 是负整数，四舍五入到小数点左边 |y| 位。ROUND(351.654,-2)=400。

2. TRUNC(x[,y])，直接截取，不四舍五入。

在缺省 y 时，默认 y=0；比如：TRUNC(3.56)=3。

y 是正整数，就是四舍五入到小数点后 y 位。TRUNC(5.654,2)=5.65。

y 是负整数，四舍五入到小数点左边 |y| 位。TRUNC(351.654,-2)=300。

## ◇ 日期函数

日期函数对日期进行运算。常用的日期函数有：

1. ADD\_MONTHS(d,n)，在某一个日期 d 上，加上指定的月数 n，返回计算后的新日期。  
d 表示日期，n 表示要加的月数。

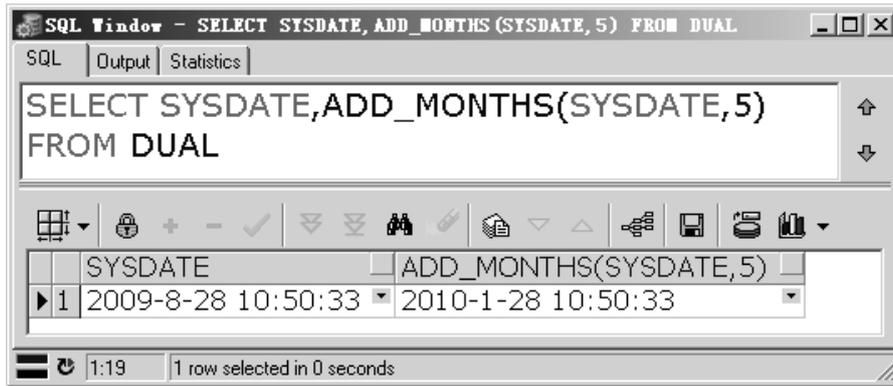


图1 ADD\_MONTHS 函数示例

2. LAST\_DAY(d), 返回指定日期当月的最后一天。

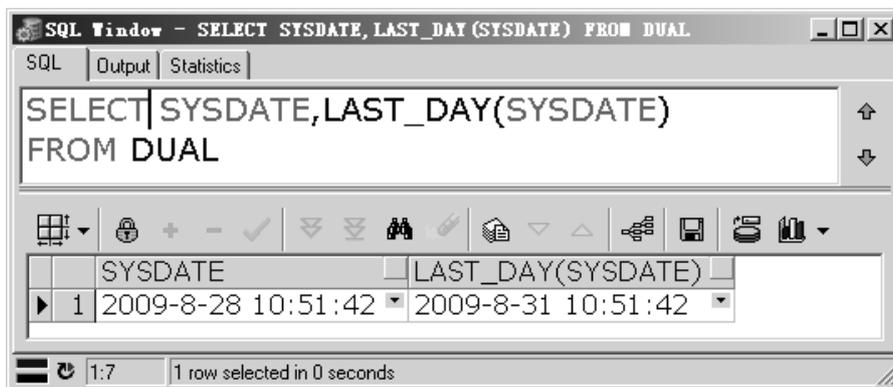


图2 LAST\_DAY 函数示例

3. ROUND(d[,fmt]), 返回一个以 fmt 为格式的四舍五入日期值, d 是日期, fmt 是格式模型。默认 fmt 为 DDD, 即月中的某一天。

- 如果 fmt 为 “YEAR” 则舍入到某年的 1 月 1 日, 即前半年舍去, 后半年作为下一年。
- 如果 fmt 为 “MONTH” 则舍入到某月的 1 日, 即前月舍去, 后半月作为下一月。
- 默认为 “DDD”, 即月中的某一天, 最靠近的天, 前半天舍去, 后半天作为第二天。
- 如果 fmt 为 “DAY” 则舍入到最近的周的周日, 即上半周舍去, 下半周作为下一周周日。

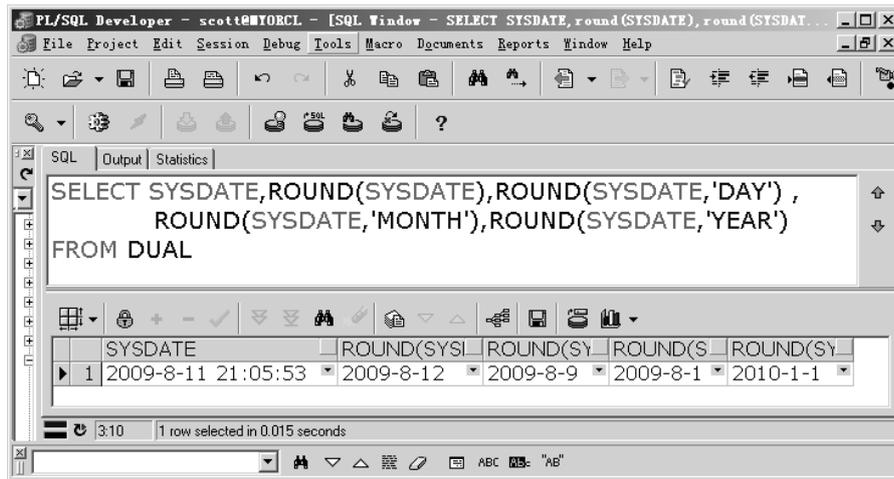


图3 ROUND 函数示例

与 ROUND 对应的函数是 TRUNC(d[,fmt])对日期的操作，TRUNC 与 ROUND 非常相似，只是不对日期进行舍入，直接截取到对应格式的第一天。

4. EXTRACT(fmt FROM d)，提取日期中的特定部分。

fmt 为：YEAR、MONTH、DAY、HOUR、MINUTE、SECOND。其中 YEAR、MONTH、DAY 可以为 DATE 类型匹配，也可以与 TIMESTAMP 类型匹配；但是 HOUR、MINUTE、SECOND 必须与 TIMESTAMP 类型匹配。

HOUR 匹配的结果中没有加上时区，因此在中国运行的结果小 8 小时。

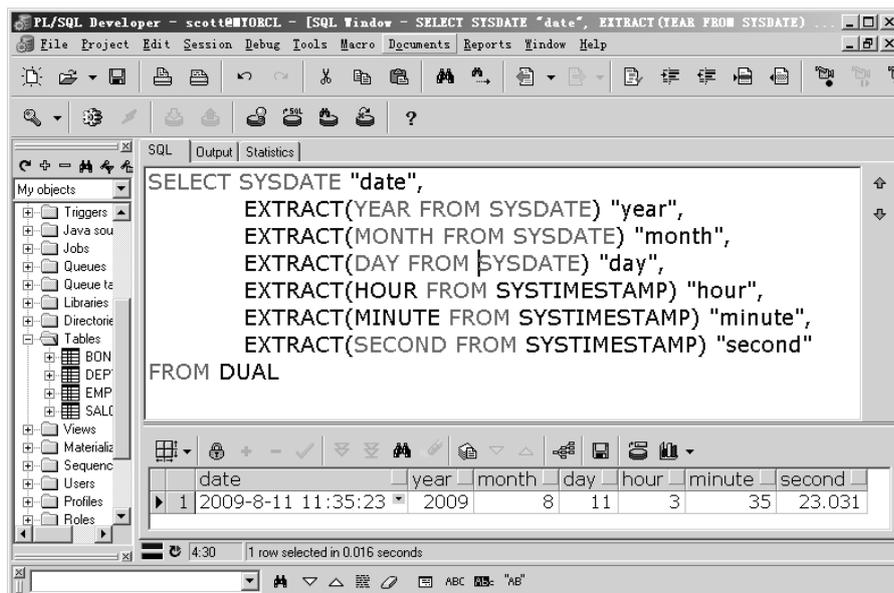


图4 EXTRACT 函数示例

## ◇ 转换函数

转换函数将值从一种数据类型转换为另外一种数据类型。常用的转换函数有：

### 1. TO\_CHAR(d|n[,fmt])

把日期和数字转换为制定格式的字符串。fmt 是格式化字符串，日期的格式化字符串前面已经学习过。

### 代码演示：TO\_CHAR 对日期的处理

---

```
SQL> SELECT TO_CHAR(SYSDATE,'YYYY"年"MM"月"DD"日" HH24:MI:SS') "date" ①  
2 FROM DUAL;
```

```
date  
-----  
2009年08月11日 12:06:00
```

---

### 代码解析：

① 在格式化字符串中，使用双引号对非格式化字符进行引用。

针对数字的格式化，格式化字符有：

参数	示例	说明
9	999	指定位置处显示数字。
.	9.9	指定位置返回小数点
,	99,99	指定位置返回一个逗号
\$	\$999	数字开头返回一个美元符号
EEEE	9.99EEEE	科学计数法表示
L	L999	数字前加一个本地货币符号
PR	999PR	如果数字式负数则用尖括号进行表示

表 4 数字格式化字符

### 代码演示：TO\_CHAR 对数字的处理

---

```
SQL> SELECT TO_CHAR(-123123.45,'L9.9EEEEPR') "date"  
2 FROM DUAL  
3 /  
date  
-----
```

---

---

<¥1.2E+05>

---

2. TO\_DATE(x [,fmt])

把一个字符串以 fmt 格式转换为一个日期类型，前面已经学习过。

3. TO\_NUMBER(x[,fmt])

把一个字符串以 fmt 格式转换为一个数字。fmt 格式字符参考表 3。

### 代码演示：TO\_NUM 函数

---

```
SQL> SELECT TO_NUMBER('-12,345.67','$99,999.99') "NUM"  
2 FROM DUAL  
3 /
```

**NUM**

-----  
-12345.67

---

### ◇ 其他单行函数

1. NVL(x,value)

如果 x 为空，返回 value，否则返回 x。

**案例 7：**对工资是 2000 元以下的员工，如果没有发奖金，每人奖金 100 元。

### 代码演示：NVL 函数

---

```
SQL> SELECT ENAME,JOB,SAL,NVL(COMM,100) FROM EMP WHERE SAL<2000;
```

ENAME	JOB	SAL	NVL(COMM,100)
SMITH	CLERK	800	100
ALLEN	SALESMAN	1600	300
WARD	SALESMAN	1250	500
MARTIN	SALESMAN	1250	1400
TURNER	SALESMAN	1500	50
ADAMS	CLERK	1100	100
JAMES	CLERK	950	100

7 rows selected

---

## 2. NVL2(x,value1,value2)

如果 x 非空，返回 value1，否则返回 value2。

**案例 8：**对 EMP 表中工资为 2000 元以下的员工，如果没有奖金，则奖金为 200 元，如果有奖金，则在原来的奖金基础上加 100 元。

### 代码演示：NVL2 函数

```
SQL> SELECT ENAME,JOB,SAL,NVL2(COMM,comm+100,200) "comm"  
2 FROM EMP WHERE SAL<2000;
```

ENAME	JOB	SAL	comm
SMITH	CLERK	800	200
ALLEN	SALESMAN	1600	400
WARD	SALESMAN	1250	600
MARTIN	SALESMAN	1250	1500
TURNER	SALESMAN	1500	150
ADAMS	CLERK	1100	200
JAMES	CLERK	950	200
MILLER	CLERK	1300	200

8 rows selected

## ◇ 聚合函数

聚合函数同时对一组数据进行操作，返回一行结果，比如计算一组数据的总和，平均值等。

名称	作用	语法
AVG	平均值	AVG(表达式)
SUM	求和	SUM(表达式)
MIN、MAX	最小值、最大值	MIN(表达式)、MAX(表达式)
COUNT	数据统计	COUNT(表达式)

表 5 聚合函数

**案例 9：**求本月所有员工的基本工资总和。

### 代码演示：SUM 函数

---

```
SQL> select sum(sal) from emp;
```

```
SUM(SAL)
```

```
-----
```

```
29025
```

---

**案例 10**：求不同部门的平均工资。

**代码演示：AVG 函数下的分组查询**

---

```
SQL> SELECT DEPTNO,AVG(SAL) FROM EMP GROUP BY DEPTNO;
```

```
DEPTNO  AVG(SAL)
```

```
-----  -
```

```
30      1566.66666
```

```
20      2175
```

```
10      2916.66666
```

---

## 4. 本章总结

---

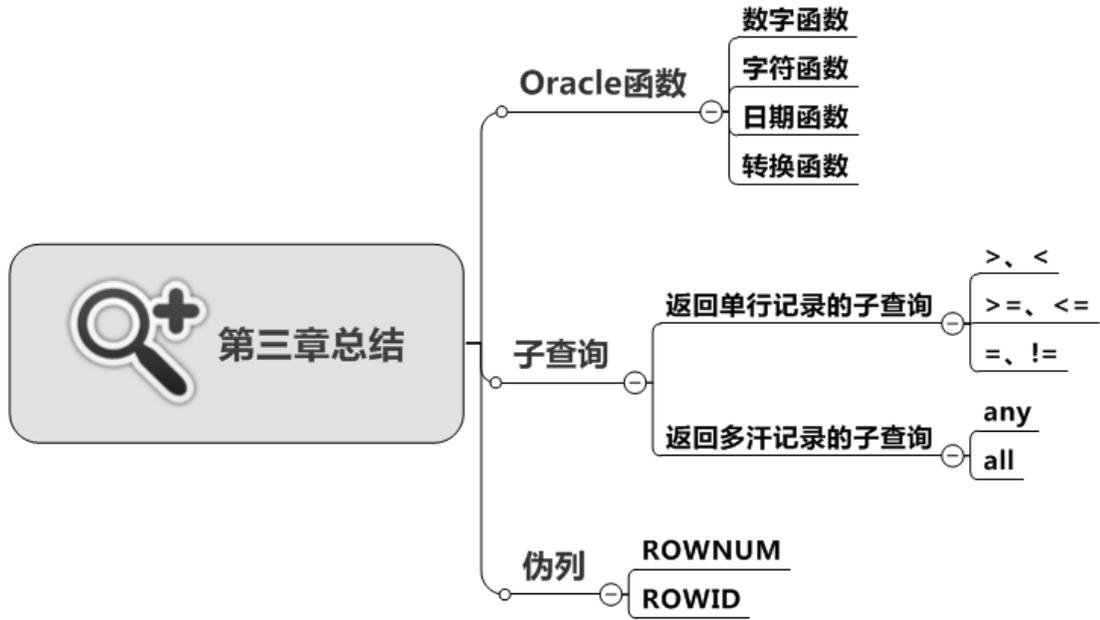
- Oracle 常用函数有字符相关的函数、数字相关的函数、日期相关的函数、转换函数等。
- EXTRACT 函数能够获取日期中的某个字段的值。
- TO\_CHAR 函数能够把数字和日期转换成固定的字符串格式。TO\_DATE 函数能够把固定格式的字符串转换为日期类型。
- 子查询中有返回单行的子查询和返回多行的子查询。
- Oracle 中存在 ROWID、ROWNUM 等伪列。

## 5. 本章练习

---

1. 描述 TO\_CHAR 和 TO\_DATE 函数的用法。
2. 描述 EXTRACT 函数的用法。
3. 你知道有哪些关于日期函数的用法？

# 章节知识结构图





# 第4章

---

## 表空间、数据库对象

### 主要内容

- ✓ 同义词概念
- ✓ 序列的应用
- ✓ 视图的概念
- ✓ 索引的概念
- ✓ 表空间的概念

# 1. Oracle 数据库对象

---

数据库对象是数据库的组成部分，常常用 CREATE 命令进行创建，可以使用 ALTER 命令修改，用 DROP 执行删除操作。前面已经接触过的数据库对象有表、用户等。

今天将学习更多的 Oracle 数据库对象：

- 同义词：就是给数据库对象一个别名。
- 序列：Oracle 中实现增长的对象。
- 视图：预定义的查询，作为表一样的查询使用，是一张虚拟表。
- 索引：对数据库表中的某些列进行排序，便于提高查询效率。

## 2. 同义词

---

同义词（Synonym）是数据库对象的一个别名，Oracle 可以为表、视图、序列、过程、函数、程序包等指定一个别名。同义词有两种类型：

- 私有同义词：拥有 CREATE SYNONYM 权限的用户（包括非管理员用户）即可创建私有同义词，创建的私有同义词只能由当前用户使用。
- 公有同义词：系统管理员可以创建公有同义词，公有同义词可以被所有用户访问。

创建同义词的语法是：

### 语法结构：同义词

---

```
CREATE [OR REPLACE] [PUBLIC] SYNONYM [schema.]synonym_name  
FOR [schema.]object_name
```

---

### 语法解析：

- ① CREATE [OR REPLACE: ]表示在创建同义词时，如果该同义词已经存在，那么就用新创建的同义词代替旧同义词。
- ② PUBLIC: 创建公有同义词时使用的关键字，一般情况下不需要创建公有同义词。
- ③ Oracle 中一个用户可以创建表、视图等多种数据库对象，一个用户和该用户下的所有数据库对象的集合称为 Schema（中文称为模式或者方案），用户名就是 Schema 名。一个数据库对象的全称是：用户名.对象名，即 schema.object\_name。

如果一个用户有权限访问其他用户对象时，就可以使用全称来访问。比如：

## 代码演示：System 用户访问 Scott 用户的 Emp 表

---

```
SQL> conn system/manager@orcl;
```

```
Connected to Oracle Database 10g Enterprise Edition Release 10.2.0.3.0
```

```
Connected as system
```

```
SQL> SELECT ENAME,JOB,SAL FROM SCOTT.EMP WHERE SAL>2000; ①
```

ENAME	JOB	SAL
JONES	MANAGER	2975
BLAKE	MANAGER	2850
CLARK	MANAGER	2450
SCOTT	ANALYST	3000
KING	PRESIDENT	5000
FORD	ANALYST	3000

```
6 rows selected
```

---

### 代码解析：

- ① 管理员用户可以访问任何用户的数据库对象，SYSTEM 用户访问 SCOTT 用户的 EMP 表时，必须使用 **SCOTT.EMP**。

**案例 1** :创建一个用户 XiaoMei，该用户拥有 CONNECT 角色和 RESOURCE 角色。为 SCOTT 用户的 EMP 表创建同义词，并通过同义词访问该 EMP 表。

## 代码演示：创建同义词并访问

---

```
SQL> CONN system/manager@orcl;
```

```
Connected to Oracle Database 10g Enterprise Edition Release 10.2.0.3.0
```

```
Connected as system
```

```
SQL> CREATE USER XiaoMei IDENTIFIED BY XiaoMei; ①
```

```
User created
```

```
SQL> GRANT CONNECT TO XiaoMei;
```

```
Grant succeeded
```

```
SQL> GRANT RESOURCE TO XiaoMei;
```

```
Grant succeeded
```

```
SQL> GRANT CREATE SYNONYM TO XiaoMei;
```

```
Grant succeeded
```

---

---

```
SQL> CONN XiaoMei/XiaoMei@ORCL;
Connected to Oracle Database 10g Enterprise Edition Release 10.2.0.3.0
Connected as XiaoMei
```

```
SQL> CREATE SYNONYM MyEmp FOR SCOTT.EMP; ②
Synonym created
SQL> SELECT * FROM MYEMP; ③
SELECT * FROM MYEMP
ORA-00942: 表或视图不存在
```

```
SQL> CONNECT SCOTT/tiger@ORCL //连接到Scott
Connected to Oracle Database 10g Enterprise Edition Release 10.2.0.3.0
Connected as SCOTT
```

```
SQL> GRANT ALL ON EMP TO XiaoMei; ④
Grant succeeded
```

```
SQL> CONNECT XiaoMei/XiaoMei@ORCL;
Connected to Oracle Database 10g Enterprise Edition Release 10.2.0.3.0
Connected as XiaoMei
```

```
SQL> SELECT ENAME,JOB,SAL FROM MyEmp WHERE SAL>2000; ⑤
```

ENAME	JOB	SAL
JONES	MANAGER	2975
BLAKE	MANAGER	2850
CLARK	MANAGER	2450
SCOTT	ANALYST	3000
KING	PRESIDENT	5000
FORD	ANALYST	3000

6 rows selected

---

### 代码解析：

- ① 在管理员用户下创建用户 XiaoMei，对用户 XiaoMei 授予 CONNECT 和 RESOURCE 角色。为了 XiaoMei 能够创建序列，必须授予系统权限：CREATE SYNONYM。
- ② 在 XiaoMei 用户下，为 SCOTT.EMP 创建私有同义词 MyEmp，同义词 MyEmp 只能在 XiaoMei 用户下使用。访问 MyEmp 就是访问 SCOTT.EMP 对象。

- ③ 访问 MyEmp 对象出错：对象不存在。因为 XiaoMei 如果访问 MyEmp，就相当于访问 SCOTT.EMP 对象，那么 SCOTT 用户必须对 XiaoMei 授予相应的权限。
- ④ SCOTT 用户下，把 EMP 表的所有权限（增删改查）授予 XiaoMei。
- ⑤ 对 MyEmp 执行查询操作。MyEmp 就可以像在本地的表一样使用。

删除同义词使用的语法是：

### 语法结构：删除同义词

---

```
DROP [PUBLIC] SYNONYM [schema.]synonym_name
```

---

#### 语法解析：

- ① PUBLIC：删除公共同义词。
- ② 同义词的删除只能被拥有同义词对象的用户或者管理员删除。
- ③ 此命令只能删除同义词，不能删除同义词下的源对象。

## 3. 序列

---

序列(Sequence)是用来生成连续的整数数据的对象。序列常常用来作为主键中增长列，序列中的可以升序生成，也可以降序生成。创建序列的语法是：

### 语法结构：创建序列

---

```
CREATE SEQUENCE sequence_name  
[START WITH num]  
[INCREMENT BY increment]  
[MAXVALUE num|NOMAXVALUE]  
[MINVALUE num|NOMINVALUE]  
[CYCLE|NOCYCLE]  
[CACHE num|NOCACHE]
```

---

#### 语法解析：

- ① START WITH：从某一个整数开始，升序默认值是 1，降序默认值是-1。
- ② INCREMENT BY：增长数。如果是正数则升序生成，如果是负数则降序生成。升序默认值是 1，降序默认值是-1。
- ③ MAXVALUE：指最大值。
- ④ NOMAXVALUE：这是最大值的默认选项，升序的最大值是： $10^{27}$ ，降序默认值是-1。
- ⑤ MINVALUE：指最小值。

- ⑥ **NOMINVALUE**: 这是默认值选项，升序默认值是 1，降序默认值是 $-10^{26}$ 。
- ⑦ **CYCLE**: 表示如果升序达到最大值后，从最小值重新开始；如果是降序序列，达到最小值后，从最大值重新开始。
- ⑧ **NOCYCLE**: 表示不重新开始，序列升序达到最大值、降序达到最小值后就报错。默认 **NOCYCLE**。
- ⑨ **CACHE**: 使用 **CACHE** 选项时，该序列会根据序列规则预生成一组序列号。保留在内存中，当使用下一个序列号时，可以更快的响应。当内存中的序列号用完时，系统再生成一组新的序列号，并保存在缓存中，这样可以提高生成序列号的效率。Oracle 默认会生产 20 个序列号。
- ⑩ **NOCACHE**: 不预先在内存中生成序列号。

**案例 2** : 创建一个从 1 开始，默认最大值，每次增长 1 的序列，要求 **NOCYCLE**，缓存中有 30 个预先分配好的序列号。

### 代码演示：生成序列号

---

```
SQL> CREATE SEQUENCE MYSEQ
2 MINVALUE 1
3 START WITH 1
4 NOMAXVALUE
5 INCREMENT BY 1
6 NOCYCLE
7 CACHE 30
8 /
```

Sequence created

---

序列创建之后，可以通过序列对象的 **CURRVAL** 和 **NEXTVAL** 两个“伪列”分别访问该序列的当前值和下一个值。

### 代码演示：序列使用

---

```
SQL> SELECT MYSEQ.NEXTVAL FROM DUAL;
NEXTVAL
-----
1
SQL> SELECT MYSEQ.NEXTVAL FROM DUAL;
NEXTVAL
```

---

---

```
-----  
2  
SQL> SELECT MYSEQ.CURRVAL FROM DUAL;  
CURRVAL  
-----  
2
```

---

使用 ALTER SEQUENCE 可以修改序列，在修改序列时有如下限制：

1. 不能修改序列的初始值。
2. 最小值不能大于当前值。
3. 最大值不能小于当前值。

使用 DROP SEQUENCE 命令可以删除一个序列对象。

#### 代码演示：序列修改和删除

---

```
SQL> ALTER SEQUENCE MYSEQ  
2 MAXVALUE 10000  
3 MINVALUE -300  
4 /  
SEQUENCE ALTERED  
SQL> DROP SEQUENCE MYSEQ;  
SEQUENCE DROPPED
```

---

## 4. 视图

---

视图（View）实际上是一张或者多张表上的预定义查询，这些表称为基表。从视图中查询信息与从表中查询信息的方法完全相同。只需要简单的 SELECT...FROM 即可。

视图具有以下优点：

1. 可以限制用户只能通过视图检索数据。这样就可以对最终用户屏蔽建表时底层的基表。
2. 可以将复杂的查询保存为视图。可以对最终用户屏蔽一定的复杂性。
3. 限制某个视图只能访问基表中的部分列或者部分行的特定数据。这样可以实现一定的安全性。
4. 从多张基表中按一定的业务逻辑抽出用户关心的部分，形成一张虚拟表。

## 语法结构：创建视图

---

```
CREATE [OR REPLACE] [{FORCE|NOFORCE}] VIEW view_name
AS
SELECT 查询
[WITH READ ONLY CONSTRAINT]
```

---

### 语法解析：

1. OR REPLACE：如果视图已经存在，则替换旧视图。
2. FORCE：即使基表不存在，也可以创建该视图，但是该视图不能正常使用，当基表创建成功后，视图才能正常使用。
3. NOFORCE：如果基表不存在，无法创建视图，该项是默认选项。
4. WITH READ ONLY：默认可以通过视图对基表执行增删改操作，但是有很多在基表上的限制（比如：基表中某列不能为空，但是该列没有出现在视图中，则不能通过视图执行 insert 操作），WITH READ ONLY 说明视图是只读视图，不能通过该视图进行增删改操作。现实开发中，基本上不通过视图对表中的数据进行增删改操作。

### 案例 3：基于 EMP 表和 DEPT 表创建视图

### 代码演示：视图

---

```
SQL> CREATE OR REPLACE VIEW EMPDETAIL
2 AS
3 SELECT EMPNO,ENAME,JOB,HIREDATE,EMP.DEPTNO,DNAME
4 FROM EMP JOIN DEPT ON EMP.DEPTNO=DEPT.DEPTNO
5 WITH READ ONLY
6 /
```

VIEW CREATED

```
SQL> SELECT * FROM EMPDETAIL; ①
```

EMPNO	ENAME	JOB	HIREDATE	DEPTNO	DNAME
7369	SMITH	CLERK	17-12月-80	20	RESEARCH
7499	ALLEN	SALESMAN	20-2月-81	30	SALES
7521	WARD	SALESMAN	22-2月-81	30	SALES
7566	JONES	MANAGER	02-4月-81	20	RESEARCH
7654	MARTIN	SALESMAN	28-9月-81	30	SALES

---

7698	BLAKE	MANAGER	01-5月 -81	30	SALES
7782	CLARK	MANAGER	09-6月 -81	10	ACCOUNTING
7788	SCOTT	ANALYST	19-4月 -87	20	RESEARCH
7839	KING	PRESIDENT	17-11月-81	10	ACCOUNTING
7844	TURNER	SALESMAN	08-9月 -81	30	SALES
7876	ADAMS	CLERK	23-5月 -87	20	RESEARCH
7900	JAMES	CLERK	03-12月-81	30	SALES
7902	FORD	ANALYST	03-12月-81	20	RESEARCH
7934	MILLER	CLERK	23-1月 -82	10	ACCOUNTING

14 ROWS SELECTED

### 代码解析：

① 对视图可以像表一样进行查询。该视图中隐藏了员工的工资。

删除视图可以使用“DROP VIEW 视图名称”，删除视图不会影响基表的数据。

## 5. 索引

当我们在某本书中查找特定的章节内容时，可以先从书的目录着手，找到该章节所在的页码，然后快速的定位到该页。这种做法的前提是页面编号是有序的。如果页码无序，就只能从第一页开始，一页页的查找了。

数据库中索引（Index）的概念与目录的概念非常类似。如果某列出现在查询的条件中，而该列的数据是无序的，查询时只能从第一行开始一行一行的匹配。**创建索引就是对某些特定列中的数据排序，生成独立的索引表。**在某列上创建索引后，如果该列出现在查询条件中，Oracle 会自动的引用该索引，**先从索引表中**查询出符合条件记录的 **ROWID**，由于 ROWID 是记录的物理地址，因此可以根据 ROWID 快速的定位到具体的记录，表中的数据非常多时，引用索引带来的查询效率非常可观。



提示

- ✧ 如果表中的某些字段经常被查询并作为查询的条件出现时，就应该考虑为该列创建索引。
- ✧ **当从很多行的表中查询少数行时**，也要考虑创建索引。有一条基本的准则是：当任何单个查询要检索的行少于或者等于整个表行数的 10% 时，索引就非常有用。

Oracle 数据库会为表的主键和包含唯一约束的列自动创建索引。索引可以提高查询的效率，但是在数据增删改时需要更新索引，因此索引对增删改时会有负面影响。

### 语法结构：创建索引

---

```
CREATE [UNIQUE] INDEX index_name ON  
table_name(column_name[,column_name...])
```

---

#### 语法解析：

1. UNIQUE:指定索引列上的值必须是唯一的。称为唯一索引。
2. index\_name: 指定索引名。
3. tabl\_name: 指定要为哪个表创建索引。
4. column\_name: 指定要对哪个列创建索引。我们也可以对多列创建索引；这种索引称为组合索引。

**案例 4：**为 EMP 表的 ENAME 列创建唯一索引，为 EMP 表的工资列创建普通索引，把 JOB 列先变为小写再创建索引。

### 代码演示：创建索引

---

```
SQL> CREATE UNIQUE INDEX UQ_ENAME_IDX ON EMP(ENAME); ①  
Index created  
SQL> CREATE INDEX IDX_SAL ON EMP(SAL); ②  
Index created  
SQL> CREATE INDEX IDX_JOB_LOWER ON EMP(LOWER(JOB)); ③  
Index created
```

---

#### 代码解析：

- ① 为 SCOTT.EMP 表的 ENAME 列创建唯一索引。
- ② 为 SCOTT.EMP 表的 SAL 列创建索引。
- ③ 在查询中可能经常使用 job 的小写作为条件的表达式，因此创建索引时，可以先对 JOB 列中的所有值转换为小写后创建索引，而这时需要使用 lower 函数，这种索引称为基于函数的索引。

在 select 语句查询时，Oracle 系统会自动为查询条件上的列应用索引。索引就是对某一列进行排序，因此在索引列上，重复值越少，索引的效果越明显。

Oracle 可以为一些列值重复非常多且值有限的列（比如性别列）上创建位图索引。关于 Oracle 更多的索引类型（比如反向键索引等），请参考 Oracle 官方文档。

## 6. 表空间

在数据库系统中，存储空间是较为重要的资源，合理利用空间，不但能节省空间，还可以提高系统的效率和工作性能。Oracle 可以存放海量数据，所有数据都在数据文件中存储。而数据文件大小受操作系统限制，并且过大的数据文件对数据的存取性能影响非常大。同时 Oracle 是跨平台的数据库，Oracle 数据可以轻松的不同平台上移植，那么如何才能提供统一存取格式的大容量呢？Oracle 采用表空间来解决。

表空间只是一个逻辑概念，若干操作系统文件（文件可以不是很大）可以组成一个表空间。表空间统一管理空间中的数据文件，一个数据文件只能属于一个表空间。一个数据库空间由若干个表空间组成。如图所示：

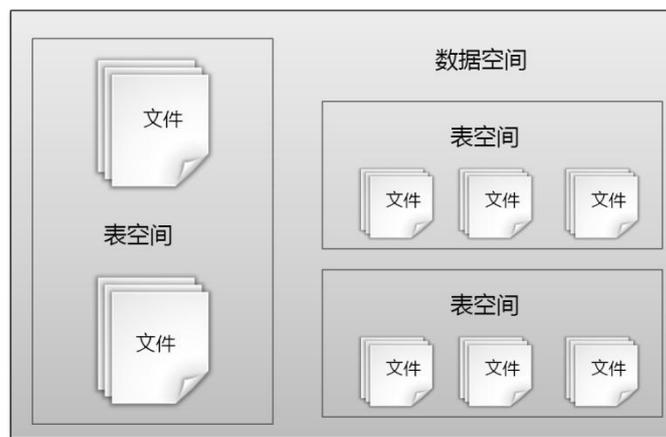


图 1 数据空间、表空间和数据文件

Oracle 中所有的数据（包括系统数据），全部保存在表空间中，常见的表空间有：

- 系统表空间：存放系统数据，系统表空间在数据库创建时创建。表空间名称为 SYSTEM。存放数据字典和视图以及数据库结构等重要系统数据信息，在运行时如果 SYSTEM 空间不足，对数据库影响会比较大，虽然在系统运行过程中可以通过命令扩充空间，但还是会影响数据库的性能，因此有必要在创建数据库时适当的把数据文件设置大一些。
- TEMP 表空间：临时表空间，安装数据库时创建，可以在运行时通过命令增大临时表空间。临时表空间的重要作用和数据排序。比如当用户执行了诸如 Order by 等命令后，服务器需要对所选取数据进行排序，如果数据很大，内存的排序区可能装不下太大数据，就需要把一些中间的排序结果写在硬盘的临时表空间中。
- 用户表自定义空间：用户可以通过 CREATE TABLESPACE 命令创建表空间。

创建表空间需要考虑数据库对分区（Extent，一个 Oracle 分区是数据库文件中一段连续的空间，Oracle 分区是 Oracle 管理中最小的单位）的管理，比如当一个表创建后先申请一个

分区，在 Insert 执行过程中，如果分区数据已满，需要重新申请另外的分区。如果一个数据库中的分区大小不一，创建表空间时需要考虑一系列问题。因此在 Oracle8i 之后，创建表空间都推荐使用“本地管理表空间”，这种表空间中的分区是一个固定大小的值，创建表空间的语法是：

### 语法结构：创建表空间

---

```
CREATE TABLESPACE 空间名称
DATAFILE '文件名1' SIZE 数字M
[,'文件名2' SIZE 数字....]
EXTENT MANAGEMENT LOCAL
UNIFORM SIZE 数字M
```

---

### 语法解析：

1. 文件名包括完整路径和文件名，每个数据文件定义了文件的初始大小，初始大小一般以“M”为单位。一个表空间中可以有多个数据文件。
2. **EXTENT MANAGEMENT LOCAL** 指明表空间类型是：**本地管理表空间**。本地管理表空间要求 Oracle 中的**数据分区（Extent）**大小统一。
3. **UNIFORM SIZE**：指定每个分区的统一大小。

**案例 5**：创建一个表空间，包含两个数据文件大小分别是 10MB，5MB，要求 extent 的大小统一为 1M。

### 代码演示：创建表空间

---

```
SQL> CREATE TABLESPACE MYSPACE
2 DATAFILE 'D:/A.ORA' SIZE 10M,
3          'D:/B.ORA' SIZE 5M
4 EXTENT MANAGEMENT LOCAL
5 UNIFORM SIZE 1M
6 /
```

Tablespace created

---

必须是**管理员用户**才能创建表空间，当表空间的空间不足时可以使用 ALTER TABLESPACE 命令向表空间中追加数据文件扩充表空间。

### 代码演示：扩充表空间

---

```
SQL> ALTER TABLESPACE MYSPACE
  2  ADD DATAFILE 'D:/C.ORA' SIZE 10M
  3  /
```

Tablespace altered

---

表空间可以在不使用时删除，使用 `DROP TABLESPACE` 命令。

数据库的所有数据全部在某一表空间中存放，在创建用户时，可以为用户指定某一表空间，那么该用户下的所有数据库对象（比如表）默认都存储在该空间中。

### 代码演示：为某一用户指定默认表空间

---

```
SQL> CREATE USER ACONG IDENTIFIED BY ACONG
  2  DEFAULT TABLESPACE MYSPACE
  3  /
```

User created

---

在创建表时，表中数据存放在用户的默认表空间中，也可以通过 `tablespace` 子句为表指定表中数据存放在其他表空间中。

### 代码演示：为表指定表空间

---

```
SQL> CREATE TABLE SCORES
  2  (
  3      ID NUMBER ,
  4      TERM VARCHAR2(2),
  5      STUID VARCHAR2(7) NOT NULL,
  6      EXAMNO VARCHAR2(7) NOT NULL,
  7      WRITTENSORE NUMBER(4,1) NOT NULL,
  8      LABSCORE NUMBER(4,1) NOT NULL
  9  )
 10  TABLESPACE MYSPACE
 11  /
```

Table created

---

创建索引时也可以为索引指定表空间。

## 代码演示：为索引指定表空间

---

```
SQL> CREATE INDEX UQ_ID ON SCORES(ID)  
2 TABLESPACE MYSPACE;
```

Index created

---

表和索引一旦创建，表空间无法修改。

## 7. 本章总结

---

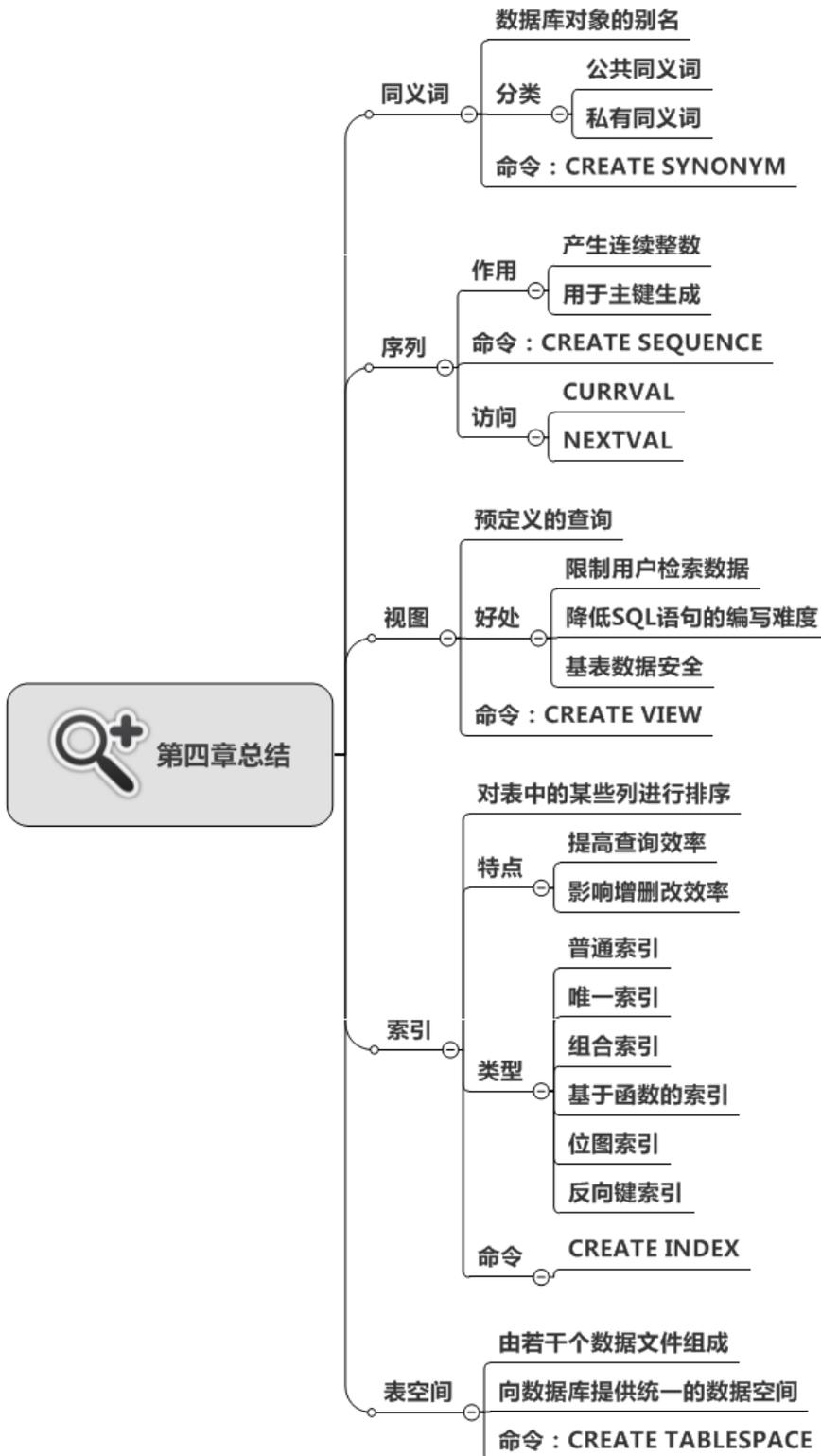
- Oracle 数据库对象都是使用 `CREATE` 命令创建的。
- 同义词就是数据库对象的一个别名。同义词的类型有公有同义词和私有同义词。只有管理员可以创建共有同义词。创建同义词的命令是：`CREATE SYNONYM`。
- 序列能够产生一个连续不重复的整数。经常作为数据库的主键生成器。创建序列的命令是 `CREATE SEQUENCE`。
- 序列的访问使用两个“伪列”，`CURRVAL` 表示序列的当前值，`NEXTVAL` 表示序列的下一个值。
- 视图就是一个预处理的查询语句，可以从若干表中过滤数据。
- 索引就是在查询中经常使用的列进行排序。常见的索引有：普通索引、唯一索引、组合索引以及基于函数的索引。此外还有位图索引、反向键索引等。
- 表空间是数据库的一个逻辑概念，表空间由若干个数据文件组成。为数据库对象和数据提供统一的空间管理。

## 8. 本章练习

---

1. 产生一个用于 DEPT 表的主键值的序列，序列起始值是 100，最大值是 500，增长步长是 10。
2. 用序列产生 DEPT 表的主键，向 DEPT 表中插入 3 条记录。
3. 为 DEPT 表创建一个同义词。
4. 创建一个视图包括 EMP 表的 EMPNO,ENAME,JOB，部门表的 DNAME 列，只能包含销售部的记录。
5. 为 EMP 表的 ENAME 列创建唯一索引。
6. 为 EMP 表的 SAL 列创建一个普通索引。
7. 以学期和学生姓名为名称比如 (S2XiaoMei) 创建一个表空间，该表空间是以学生姓名为用户的默认表空间。

# 章节知识结构图





# 第 5 章

---

## PL/SQL 程序设计

### 主要内容

- ✓ PL/SQL 数据类型
- ✓ PL/SQL 条件和循环控制
- ✓ 动态执行 SQL
- ✓ PL/SQL 中的异常处理

# 1. PL/SQL 简介

从第一学期到现在，在数据库上一直使用单一的 SQL 语句进行数据操作，没有流程控制，无法开发复杂的应用。Oracle PL/SQL 语言（Procedural Language/SQL）是结合了结构化查询与 Oracle 自身过程控制为一体的强大语言，PL/SQL 不但支持更多的数据类型，拥有自身的变量声明、赋值语句，而且还有条件、循环等流程控制语句。过程控制结构与 SQL 数据处理能力无缝的结合形成了强大的编程语言，可以创建过程和函数以及程序包。

PL/SQL 是一种块结构的语言，它将一组语句放在一个块中，一次性发送给服务器，PL/SQL 引擎分析收到 PL/SQL 语句块中的内容，把其中的过程控制语句由 PL/SQL 引擎自身去执行，把 PL/SQL 块中的 SQL 语句交给服务器的 SQL 语句执行器执行。如图所示：

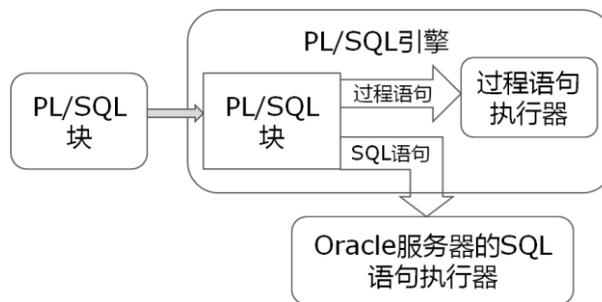


图 1 PL/SQL 体系结构

PL/SQL 块发送给服务器后，先被编译然后执行，对于有名称的 PL/SQL 块（如子程序）可以单独编译，永久的存储在数据库中，随时准备执行。PL/SQL 的优点还有：

## ➤ 支持 SQL

SQL 是访问数据库的标准语言，通过 SQL 命令，用户可以操纵数据库中的数据。PL/SQL 支持所有的 SQL 数据操纵命令、游标控制命令、事务控制命令、SQL 函数、运算符和伪列。同时 PL/SQL 和 SQL 语言紧密集成，PL/SQL 支持所有的 SQL 数据类型和 NULL 值。

## ➤ 支持面向对象编程

PL/SQL 支持面向对象的编程，在 PL/SQL 中可以创建类型，可以对类型进行继承，可以在子程序中重载方法等。

## ➤ 更好的性能

SQL 是非过程语言，只能一条一条执行，而 PL/SQL 把一个 PL/SQL 块统一进行编译后执行，同时还可以把编译好的 PL/SQL 块存储起来，以备重用，减少了应用程序和服务器之间的通信时间，PL/SQL 是快速而高效的。

### ➤ 可移植性

使用 PL/SQL 编写的应用程序，可以移植到任何操作系统平台上的 Oracle 服务器，同时还可以编写可移植程序库，在不同环境中重用。

### ➤ 安全性

可以通过存储过程对客户机和服务器之间的应用程序逻辑进行分隔，这样可以限制对 Oracle 数据库的访问，数据库还可以授权和撤销其他用户访问的能力。

## 2. PL/SQL 块

---

PL/SQL 是一种块结构的语言，一个 PL/SQL 程序包含了一个或者多个逻辑块，逻辑块中可以声明变量，变量在使用之前必须先声明。除了正常的执行程序外，PL/SQL 还提供了专门的异常处理部分进行异常处理。每个逻辑块分为三个部分，语法是：

### 语法结构：PL/SQL 块的语法

---

```
[DECLARE
    --declaration statements] ①
BEGIN
    --executable statements ②
[EXCEPTION
    --exception statements] ③
END;
```

---

### 语法解析：

- ① 声明部分：声明部分包含了变量和常量的定义。这个部分由关键字 **DECLARE** 开始，如果不声明变量或者常量，可以省略这部分。
- ② 执行部分：执行部分是 PL/SQL 块的指令部分，由关键字 **BEGIN** 开始，关键字 **END** 结尾。所有的可执行 PL/SQL 语句都放在这一部分，该部分执行命令并操作变量。其他的 PL/SQL 块可以作为子块嵌套在该部分。PL/SQL 块的执行部分是必选的。注意 **END** 关键字后面用分号结尾。

- ③ 异常处理部分：该部分是可选的，该部分用 **EXCEPTION** 关键字把可执行部分分成两个小部分，之前的程序是正常运行的程序，一旦出现异常就跳转到异常部分执行。

PL/SQL 是一种编程语言，与 Java 和 C# 一样，除了有自身独有的数据类型、变量声明和赋值以及流程控制语句外，PL/SQL 还有自身的语言特性：

PL/SQL 对大小写不敏感，为了良好的程序风格，开发团队都会选择一个合适的编码标准。比如有的团队规定：关键字全部大些，其余的部分小写。

PL/SQL 块中的每一条语句都必须以分号结束，SQL 语句可以是多行的，但分号表示该语句结束。一行中可以有多条 SQL 语句，他们之间以分号分隔，但是不推荐一行中写多条语句。

PL/SQL 中的特殊符号说明：

类型	符号	说明
赋值运算符	:=	Java 和 C# 中都是等号，PL/SQL 的赋值是：=
特殊字符		字符串连接操作符。
	--	PL/SQL 中的单行注释。
	/*,*/	PL/SQL 中的多行注释，多行注释不能嵌套。
	<<, >>	标签分隔符。只为了标识程序特殊位置。
	..	范围操作符，比如：1..5 标识从 1 到 5
算术运算符	+, -, *, /	基本算术运算符。
	**	求幂操作，比如：3**2=9
关系运算符	>, <, >=, <=, =	基本关系运算符，=表示相等关系，不是赋值。
	<>, !=	不等关系。
逻辑运算符	AND, OR, NOT	逻辑运算符。

表 1 PL/SQL 中的特殊符号和运算符

## ◇ 变量声明

PL/SQL 支持 SQL 中的数据类型，PL/SQL 中正常支持 **NUMBER, VARCHAR2, DATE** 等 Oracle SQL 数据类型。声明变量必须指明变量的数据类型，也可以声明变量时对变量初始化，变量声明必须在声明部分。声明变量的语法是：

### 语法格式：声明变量

---

变量名 数据类型[:=初始值]

---

### 语法解析：

数据类型如果需要长度，可以用括号指明长度，比如：varchar2(20)。

### 代码演示：声明变量

---

```
SQL> DECLARE
2     sname VARCHAR2(20) :='jerry'; ①
3 BEGIN
4     sname:=sname||' and tom'; ②
5     dbms_output.put_line(sname); ③
6 END;
7 /jerry
```

PL/SQL procedure successfully completed

---

### 代码解析：

- ① 声明一个变量 `sname`，初始化值是“jerry”。字符串用单引号，如果字符串中出现单引号可以使用两个单引号（''）来表示，即单引号同时也具有转义的作用。
- ② 对变量 `sname` 重新赋值，赋值运算符是“:=”。
- ③ `dbms_output.put_line` 是输出语句，可以把一个变量的值输出，在 SQL\*Plus 中输出数据时，可能没有结果显示，可以使用命令：`set serveroutput on` 设置输出到 SQL\*Plus 控制台上。

对变量赋值还可以使用 `SELECT...INTO` 语句从数据库中查询数据对变量进行赋值。但是查询的结果只能是一行记录，不能是零行或者多行记录。

### 代码演示：变量赋值

---

```
SQL> DECLARE
2     sname VARCHAR2(20) DEFAULT 'jerry'; ①
3 BEGIN
4     SELECT ename INTO sname FROM emp WHERE empno=7934; ②
5     dbms_output.put_line(sname);
6 END;
7 /
```

MILLER

PL/SQL procedure successfully completed

---

### 代码解析：

- ① 变量初始化时，可以使用 `DEFAULT` 关键字对变量进行初始化。

- ② 使用 `select...into` 语句对变量 `sname` 赋值，要求查询的结果必须是一行，不能是多行或者没有记录。

## ◇ 声明常量

常量在声明时赋予初值，并且在运行时不允许重新赋值。使用 `CONSTANT` 关键字声明常量。

### 代码演示：声明常量

---

```
SQL> DECLARE
2     pi CONSTANT number :=3.14;    --圆周率长值 ①
3     r number DEFAULT 3;    --圆的半径默认值3 ②
4     area number;    --面积。
5 BEGIN
6     area:=pi*r*r;    --计算面积
7     dbms_output.put_line(area);    --输出圆的面积
8 END;
9 /
```

28.26

---

PL/SQL procedure successfully completed

---

### 代码解析：

- ① 声明常量时使用关键字 `CONSTANT`，常量初值可以使用赋值运算符 (`:=`) 赋值，也可以使用 `DEFAULT` 关键字赋值。

在 `SQL*Plus` 中还可以声明 `Session`（会话，也就是一个客户端从连接到退出的过程称为当前用户的会话。）全局级变量，该变量在整个会话过程中均起作用，类似的这种变量称为宿主变量。宿主变量在 `PL/SQL` 引用时要用“:变量名”引用。

### 代码演示：宿主常量

---

```
SQL> var emp_name varchar(30); ①
SQL> BEGIN
2 SELECT ename INTO :emp_name FROM emp WHERE empno=7499; ②
3 END;
```

---

---

```
PL/SQL procedure successfully completed
```

```
emp_name
```

```
-----
```

```
ALLEN
```

```
SQL> print emp_name; ③
```

```
emp_name
```

```
-----
```

```
ALLEN
```

---

### 代码解析：

- ① 可以使用 var 声明宿主变量。
- ② PL/SQL 中访问宿主变量时要在变量前加“:”。
- ③ 在 SQL\*Plus 中，使用 print 可以输出变量中的结果。

## 3. PL/SQL 数据类型

---

前面在建表时，学习过 Oracle SQL 的数据类型，PL/SQL 不但支持这些数据类型，还具备自身的数据类型。PL/SQL 的数据类型包括标量数据类型，引用数据类型和存储文本、图像、视频、声音等非结构化的大数据类型（LOB 数据类型）等。下面列举一些常用的类型。

### ◇ 标量数据类型

标量数据类型的变量只有一个值，且内部没有分量。标量数据类型包括数字型，字符型，日期型和布尔型。这些类型有的是 Oracle SQL 中定义的数据类型，有的是 PL/SQL 自身附加的数据类型。字符型和数字型又有子类型，子类型只与限定的范围有关，比如 NUMBER 类型可以表示整数，也可以表示小数，而其子类型 POSITIVE 只表示正整数。

类型	说明
VARCHAR2(长度)	可变长度字符串，Oracle SQL 定义的数据类型，在 PL/SQL 中使用时最常 32767 字节。在 PL/SQL 中使用没有默认长度，因此必须指定。
NUMBER(精度，小数)	Oracle SQL 定义的数据类型，见第二章。
DATE	Oracle SQL 定义的日期类型，见第二章。
TIMESTAMP	Oracle SQL 定义的日期类型，见第二章。
CHAR(长度)	Oracle SQL 定义的日期类型，固定长度字符，最长 32767 字节，默认

	长度是 1，如果内容不够用空格代替。
LONG	Oracle SQL 定义的数据类型，变长字符串基本类型，最长 32760 字节。在 Oracle SQL 中最长 2147483647 字节。
BOOLEAN	PL/SQL 附加的数据类型，逻辑值为 TRUE、FALSE、NULL
BINARY_INTEGER	PL/SQL 附加的数据类型，介于 $-2^{31}$ 和 $2^{31}$ 之间的整数。
PLS_INTEGER	PL/SQL 附加的数据类型，介于 $-2^{31}$ 和 $2^{31}$ 之间的整数。类似于 BINARY_INTEGER，只是 PLS_INTEGER 值上的运行速度更快。
NATURAL	PL/SQL 附加的数据类型，BINARY_INTEGER 子类型，表示从 0 开始的自然数。
NATURALN	与 NATURAL 一样，只是要求 NATURALN 类型变量值不能为 NULL。
POSITIVE	PL/SQL 附加的数据类型，BINARY_INTEGER 子类型，正整数。
POSITIVEN	与 POSITIVE 一样，只是要求 POSITIVE 的变量值不能为 NULL。
REAL	Oracle SQL 定义的数据类型，18 位精度的浮点数
INT,INTEGER,SMALLINT	Oracle SQL 定义的数据类型，NUMBERDE 的子类型，38 位精度整数。
SIGNTYPE	PL/SQL 附加的数据类型，BINARY_INTEGER 子类型。值有：1、-1、0。
STRING	与 VARCHAR2 相同。

表 2 PL/SQL 中标量数据类型。

## ◇ 属性数据类型

当声明一个变量的值是数据库中的一行或者是数据库中某列时，可以直接使用属性类型来声明。Oracle 中存在两种属性类型：%TYPE 和%ROWTYPE。

### ➤ % ROWTYPE

引用数据库表中的一行作为数据类型，即 RECORD 类型（记录类型），是 PL/SQL 附加的数据类型。表示一条记录，就相当于 C# 中的一个对象。可以使用“.”来访问记录中的属性。

### 代码演示：

---

```
SQL> DECLARE
2     myemp EMP%ROWTYPE; ①
3 BEGIN
4     SELECT * INTO myemp FROM emp WHERE empno=7934; ②
5     dbms_output.put_line(myemp.ename); ③
6 END;
7 /
MILLER
```

---

---

PL/SQL procedure successfully completed

---

### 代码解析：

- ① 声明一个 myemp 对象，该对象表示 EMP 表中的一行。
- ② 从 EMP 表中查询一条记录放入 myemp 对象中。
- ③ 访问该对象的属性可以使用“.”。

#### ➤ %TYPE

引用某个变量或者数据库的列的类型作为某变量的数据类型。

### 代码演示：%TYPE 应用

---

```
SQL> DECLARE
2     sal emp.sal%TYPE; ①
3     mysal number(4):=3000;
4     totalsal mysal%TYPE; ②
5 BEGIN
6     SELECT SAL INTO sal FROM emp WHERE empno=7934;
7     totalsal:=sal+mysal;
8     dbms_output.put_line(totalsal);
9 END;
10 /
4300
PL/SQL procedure successfully completed
```

---

### 代码解析：

- ① 定义变量 sal 为 emp 表中 sal 列的类型。
- ② 定义 totalsal 是变量 mysal 的类型。

%TYPE 可以引用表中的某列作的类型为变量的数据类型，也可以引用某变量的类型作为新变量的数据类型。

## 4. PL/SQL 条件控制和循环控制

---

PL/SQL 程序可通过条件或循环结构来控制命令执行的流程。PL/SQL 提供了丰富的流程控制语句，与 C#一样也有三种控制结构：

- 顺序结构
- 条件结构

➤ 循环结构

## ◇ 条件控制

C#中的条件控制使用关键字 `if` 和 `switch`。PL/SQL 中关于条件控制的关键字有 `IF-THEN`、`IF-THEN-ELSE`、`IF-THEN-ELSIF` 和多分枝条件 `CASE`。

➤ IF-THEN

该结构先判断一个条件是否为 `TRUE`，条件成立则执行对应的语句块，与 C#中的 `if` 语句很相似，具体语法是：

C#中 if 语法	PL/SQL 中 IF 语法
<pre>if (条件){     //条件结构体 }</pre>	<pre>IF 条件 THEN     --条件结构体 END IF;</pre>

表 3 PL/SQL 中条件语法

### 说明：

- ① 用 `IF` 关键字开始，`END IF` 关键字结束，注意 `END IF` 后面有一个分号。
- ② 条件部分可以不使用括号，但是必须以关键字 `THEN` 来标识条件结束，如果条件成立，则执行 `THEN` 后到对应 `END IF` 之间的语句块内容。如果条件不成立，则不执行条件语句块的内容。
- ③ C#结构用一对大括号来包含条件结构体的内容。PL/SQL 中关键字 `THEN` 到 `END IF` 之间的内容是条件结构体内容。
- ④ 条件可以使用关系运算符符合逻辑运算符。

**案例 1：**查询 JAMES 的工资，如果大于 900 元，则发奖金 800 元。

### 代码演示：IF-THEN 应用

---

```
DECLARE
    newSal emp.sal % TYPE;
BEGIN
    SELECT sal INTO newSal FROM emp
    WHERE ename='JAMES';
    IF newSal>900 THEN ①
```

---

---

```

UPDATE emp
SET comm=800
WHERE ename='JAMES';
END IF;
COMMIT ; ②
END;

```

---

### 代码解析：

- ① 先判断条件，如果条件为 TRUE，则执行条件结构体内部的内容。
- ② 在 PL/SQL 块中可以使用事务控制语句，该 COMMIT 同时也能把 PL/SQL 块外没有提交的数据一并提交，使用时需要注意。

### ➤ IF-THEN-ELSE

### 语法格式：IF-THEN-ELSE

C#中 if 语法	PL/SQL 中 IF 语法
<pre> if (条件){     //条件成立结构体 } else{     //条件不成立结构体 } </pre>	<pre> IF 条件 THEN     --条件成立结构体 ELSE     --条件不成立结构体 END IF; </pre>

表 4 PL/SQL 中条件语法

### 语法解析：

把 ELSE 与 IF-THEN 连在一起使用，如果 IF 条件不成立则执行就会执行 ELSE 部分的语句。

**案例 2：**查询 JAMES 的工资，如果大于 900 元，则发奖金 800 元，否则发奖金 400 元。

### 代码演示：IF-THEN-ELSE 应用

---

```

DECLARE
    newSal emp.sal % TYPE;
BEGIN
    SELECT sal INTO newSal FROM emp
    WHERE ename='JAMES';
    IF newSal>900 THEN

```

---

---

```

UPDATE emp
SET comm=800
WHERE ename='JAMES';
ELSE
UPDATE emp
SET comm=400
WHERE ename='JAMES';
END IF;
END;

```

---

➤ IF-THEN-ELSIF

### 语法格式：IF-THEN-ELSIF

C#中 if 语法	PL/SQL 中 IF 语法
<pre> if (条件 2){     //条件成立结构体 } else if(条件 2){     //条件不成立结构体 } else{     //以上条件都不成立结构体 } </pre>	<pre> IF 条件 1 THEN     --条件 1 成立结构体 ELSIF 条件 2 THEN     --条件 2 成立结构体 ELSE     --以上条件都不成立结构体 END IF; </pre>

表 5 PL/SQL 中多分枝条件语法

### 语法解析：

PL/SQL 中的再次条件判断中使用关键字 ELSIF，而 C#使用 else if。

**案例 3：**查询 JAMES 的工资，如果大于 1500 元，则发放奖金 100 元，如果工作大于 900 元，则发奖金 800 元，否则发奖金 400 元。

### 代码演示：IF-THEN-ELSIF 应用

---

```

DECLARE
    newSal emp.sal % TYPE;
BEGIN
    SELECT sal INTO newSal FROM emp

```

---

---

```
WHERE ename='JAMES';
IF newSal>1500 THEN
    UPDATE emp
    SET comm=1000
    WHERE ename='JAMES';
ELSIF newSal>1500 THEN
    UPDATE emp
    SET comm=800
    WHERE ename='JAMES';
ELSE
    UPDATE emp
    SET comm=400
    WHERE ename='JAMES';
END IF;
END;
```

---

➤ CASE

CASE 是一种选择结构的控制语句，可以根据条件从多个执行分支中选择相应的执行动作。也可以作为表达式使用，返回一个值。类似于 C# 中的 switch 语句。语法是：

### 语法格式：CASE

---

```
CASE [selector]
    WHEN 表达式 1 THEN 语句序列 1;
    WHEN 表达式 2 THEN 语句序列 2;
    WHEN 表达式 3 THEN 语句序列 3;
    .....
    [ELSE 语句序列 N];
END CASE;
```

---

### 语法解析：

如果存在选择器 selector，选择器 selector 与 WHEN 后面的表达式匹配，匹配成功就执行 THEN 后面的语句。如果所有表达式都与 selector 不匹配，则执行 ELSE 后面的语句。

**案例 4：**输入一个字母 A、B、C 分别输出对应的级别信息。

**代码演示：**CASE 中存在 selector，不返回值

---

```
DECLARE
    v_grade CHAR(1):=UPPER('&p_grade'); ①
BEGIN
    CASE v_grade ②
        WHEN 'A' THEN
            dbms_output.put_line('Excellent');
        WHEN 'B' THEN
            dbms_output.put_line('Very Good');
        WHEN 'C' THEN
            dbms_output.put_line('Good');
        ELSE
            dbms_output.put_line('No such grade');
    END CASE;
END;
```

---

### 代码解析：

- ① & grade 表示在运行时由键盘输入字符串到 grade 变量中。
- ② v\_grade 分别于 WHEN 后面的值匹配，如果成功就执行 WHEN 后的程序序列。

CASE 语句还可以作为表达式使用，返回一个值。

### 代码演示：CASE 中存在 selector，作为表达式使用

---

```
DECLARE
    v_grade CHAR(1):=UPPER('&grade');
    p_grade VARCHAR(20) ;
BEGIN
    p_grade := ①
    CASE v_grade
        WHEN 'A' THEN
            'Excellent'
        WHEN 'B' THEN
            'Very Good'
        WHEN 'C' THEN
            'Good'
        ELSE
            'No such grade'
    END;
    dbms_output.put_line('Grade: ' ||v_grade||',the result is '||p_grade);
```

---

---

```
END;
```

---

### 代码解析：

① CASE 语句可以返回一个结果给变量 p\_grade

PL/SQL 还提供了搜索 CASE 语句。也就是说，不使用 CASE 中的选择器，直接在 WHEN 后面判断条件，第一个条件为真时，执行对应 THEN 后面的语句序列。

### 代码演示：搜索 CASE

---

```
DECLARE
    v_grade CHAR(1):=UPPER('&grade');
    p_grade VARCHAR(20) ;
BEGIN
    p_grade :=
    CASE
        WHEN v_grade='A' THEN
            'Excellent'
        WHEN v_grade='B' THEN
            'Very Good'
        WHEN v_grade='C' THEN
            'Good'
        ELSE
            'No such grade'
    END;
    dbms_output.put_line('Grade:' ||v_grade||',the result is '||p_grade);
END;
```

---

## ◇ 循环结构

PL/SQL 提供了丰富的循环结构来重复执行一些列语句。Oracle 提供的循环类型有：

1. 无条件循环 LOOP-END LOOP 语句
2. WHILE 循环语句
3. FOR 循环语句

在上面的三类循环中 EXIT 用来强制结束循环，相当于 C#循环中的 break。

➤ LOOP 循环

LOOP 循环是最简单的循环，也称为无限循环，LOOP 和 END LOOP 是关键字。

### 语法格式：LOOP 循环

---

```
LOOP
    --循环体
END LOOP;
```

---

#### 语法格式：

1. 循环体在 LOOP 和 END LOOP 之间，在每个 LOOP 循环体中，首先执行循环体中的语句序列，执行完后再重新开始执行。
2. 在 LOOP 循环中可以使用 EXIT 或者[EXIT WHEN 条件]的形式终止循环。否则该循环就是死循环。

**案例 5：**执行 1+2+3+...+100 的值

### 代码演示：LOOP 循环

---

```
DECLARE
    counter number(3):=0;
    sumResult number:=0;
BEGIN
    LOOP
        counter := counter+1;
        sumResult := sumResult+counter;
        IF counter>=100 THEN ①
            EXIT;
        END IF;
        -- EXIT WHEN counter>=100; ②
    END LOOP;
    dbms_output.put_line('result is :'||to_char(sumResult));
END;
```

---

#### 代码解析：

- ① LOOP 循环中可以使用 IF 结构嵌套 EXIT 关键字退出循环
- ② 注释行，该行可以代替①中的循环结构，WHEN 后面的条件成立时跳出循环。

➤ WHILE 循环

与 C#中的 while 循环很类似。先判断条件，条件成立再执行循环体。

### 语法格式：WHILE

C#中 while 语法	PL/SQL 中 WHILE 语法
while (条件){ //循环体体 }	WHILE 条件 LOOP --循环体 END LOOP;

表 5 PL/SQL 中 LOOP 语法

### 案例 6：WHILE 循环

#### 代码演示：WHILE 循环

---

```
DECLARE
    counter number(3):=0;
    sumResult number:=0;
BEGIN
    WHILE counter<100 LOOP
        counter := counter+1;
        sumResult := sumResult+counter;
    END LOOP;
    dbms_output.put_line('result is :'||sumResult);
END;
```

---

#### ➤ FOR 循环

FOR 循环需要预先确定的循环次数，可通过给循环变量指定下限和上限来确定循环运行的次数，然后循环变量在每次循环中递增（或者递减）。FOR 循环的语法是：

### 语法格式：FOR 循环

---

```
FOR 循环变量 IN [REVERSE] 循环下限..循环上限 LOOP LOOP
    --循环体
END LOOP;
```

---

#### 语法解析：

循环变量：该变量的值每次循环根据上下限的 REVERSE 关键字进行加 1 或者减 1。

REVERSE: 指明循环从上限向下限依次循环。

### 案例 7 : FOR 循环

#### 代码演示 : FOR 循环

---

```
DECLARE
    counter number(3):=0;
    sumResult number:=0;
BEGIN
    FOR counter IN 1..100 LOOP
        sumResult := sumResult+counter;
    END LOOP;
    dbms_output.put_line('result is :'||sumResult);
END;
```

---

## ◇ 顺序结构

在程序顺序结构中两个特殊的语句。GOTO 和 NULL

### ➤ GOTO 语句

GOTO 语句将无条件的跳转到标签指定的语句去执行。标签是用双尖括号括起来的标示符，在 PL/SQL 块中必须具有唯一的名称，标签后必须紧跟可执行语句或者 PL/SQL 块。GOTO 不能跳转到 IF 语句、CASE 语句、LOOP 语句、或者子块中。

### ➤ NULL 语句

NULL 语句什么都不做，只是将控制权转到下一行语句。NULL 语句是可执行语句。NULL 语句在 IF 或者其他语句语法要求至少需要一条可执行语句，但又不需要具体操作的地方。比如 GOTO 的目标地方不需要执行任何语句时。

### 案例 8 : GOGO 和 NULL

#### 代码演示 : GOTO 和 NULL

---

```
DECLARE
```

---

---

```
sumsal emp.sal%TYPE;
BEGIN
  SELECT SUM(sal) INTO sumsal FROM EMP;
  IF sumsal>20000 THEN
    GOTO first_label; ①
  ELSE
    GOTO second_label; ②
  END IF;
  <<first_label>> ③
  dbms_output.put_line('ABOVE 20000:' || sumsal);
  <<second_label>> ④
  NULL;
END;
```

---

### 代码解析：

- ① 跳转到程序 first\_label 位置，就是②的位置，first\_label 是一个标签，用两个尖括号包含。
- ② 无条件跳转到 second\_label 位置，就是④的位置。④处不执行任何内容，因此是一个 NULL 语句。

与 C#一样，在 PL/SQL 中，各种循环之间可以相互嵌套。

## 5. PL/SQL 中动态执行 SQL 语句

---

在 PL/SQL 程序开发中，可以使用 DML 语句和事务控制语句，但是还有很多语句（比如 DDL 语句）不能直接在 PL/SQL 中执行。这些语句可以使用动态 SQL 来实现。

PL/SQL 块先编译然后再执行，动态 SQL 语句在编译时不能确定，只有在程序执行时把 SQL 语句作为字符串的形式由动态 SQL 命令来执行。在编译阶段 SQL 语句作为字符串存在，程序不会对字符串中的内容进行编译，在运行阶段再对字符串中的 SQL 语句进行编译和执行，动态 SQL 的语法是：

### 语法格式：动态 SQL

---

```
EXECUTE IMMEDIATE 动态语句字符串
[INTO 变量列表]
[USING 参数列表]
```

---

## 语法解析：

如果动态语句是 SELECT 语句，可以把查询的结果保存到 INTO 后面的变量中。如果动态语句中存在参数，USING 为语句中的参数传值。

动态 SQL 中的参数格式是：[:参数名]，参数在运行时需要使用 USING 传值。

## 案例 9：动态 SQL

### 代码演示：动态 SQL

---

```
DECLARE
    sql_stmt    VARCHAR2(200); --动态SQL语句
    emp_id      NUMBER(4) := 7566;
    salary      NUMBER(7,2);
    dept_id     NUMBER(2) := 90;
    dept_name   VARCHAR2(14) := 'PERSONNEL';
    location    VARCHAR2(13) := 'DALLAS';
    emp_rec     emp%ROWTYPE;
BEGIN
    --无子句的execute immediate
    EXECUTE IMMEDIATE 'CREATE TABLE bonus1 (id NUMBER, amt NUMBER)'; ①
    ----using子句的execute immediate
    sql_stmt := 'INSERT INTO dept VALUES (:1, :2, :3)';
    EXECUTE IMMEDIATE sql_stmt USING dept_id, dept_name, location; ②
    ----into子句的execute immediate
    sql_stmt := 'SELECT * FROM emp WHERE empno = :id';
    EXECUTE IMMEDIATE sql_stmt INTO emp_rec USING emp_id; ③

    ----returning into子句的execute immediate
    sql_stmt := 'UPDATE emp SET sal = 2000 WHERE empno = :1
        RETURNING sal INTO :2';
    EXECUTE IMMEDIATE sql_stmt USING emp_id RETURNING INTO salary; ④

    EXECUTE IMMEDIATE 'DELETE FROM dept WHERE deptno = :num'
        USING dept_id; ⑤
END;
```

---

### 代码解析：

- ① 动态执行一个完整的 SQL 语句。
- ② SQL 语句中存在 3 个参数分别标识为：[:1、:2、:3]，因此需要用 USING 关键字对三

- 个参数分别赋值。
- ③ 对动态查询语句可以使用 INTO 子句把查询的结果保存到一个变量中，要求该结果只能是单行。
  - ④ 在 Oracle 的 insert, update, delete 语句都可以使用 RETURNING 子句把操作影响的行中的数据返回，对 SQL 语句中存在 RETURNING 子句时，在动态执行时可以使用 RETURNING INTO 来接收。
  - ⑤ 动态执行参数中可以是：[:数字]也可以是[:字符串]。

## 6. PL/SQL 的异常处理

在程序运行时出现的错误，称为异常。发生异常后，语句将停止执行，PL/SQL 引擎立即将控制权转到 PL/SQL 块的异常处理部分。异常处理机制简化了代码中的错误检测。PL/SQL 中任何异常出现时，每一个异常都对应一个异常码和异常信息。比如：

```
SQL> DECLARE
2   newSal emp.sal % TYPE;
3 BEGIN
4   SELECT sal INTO newSal FROM emp;
5 END;
6 /
```

ORA-01422: 实际返回的行数超出请求的行数

图 1 PL/SQL 中的异常

### ❖ 预定义异常

为了 Oracle 开发和维护的方便，在 Oracle 异常中，为常见的异常码定义了对应的异常名称，称为预定义异常，常见的预定义异常有：

异常名称	异常码	描述
DUP_VAL_ON_INDEX	ORA-00001	试图向唯一索引列插入重复值
INVALID_CURSOR	ORA-01001	试图进行非法游标操作。
INVALID_NUMBER	ORA-01722	试图将字符串转换为数字
NO_DATA_FOUND	ORA-01403	SELECT INTO 语句中没有返回任何记录。
TOO_MANY_ROWS	ORA-01422	SELECT INTO 语句中返回多于 1 条记录。
ZERO_DIVIDE	ORA-01476	试图用 0 作为除数。
CURSOR_ALREADY_OPEN	ORA-06511	试图打开一个已经打开的游标

表 6 PL/SQL 中预定义异常

PL/SQL 中用 EXCEPTION 关键字开始异常处理。具体语法是：

### 语法格式：异常处理

---

```
BEGIN
    --可执行部分
    EXCEPTION  -- 异常处理开始
        WHEN 异常名 1 THEN
            --对应异常处理
        WHEN 异常名 2 THEN
            --对应异常处理
        .....
        WHEN OTHERS THEN
            --其他异常处理
    END;
```

---

### 语法解析：

异常发生时，进入异常处理部分，具体的异常与若干个 WHEN 子句中指定的异常名匹配，匹配成功就进入对应的异常处理部分，如果对应不成功，则进入 OTHERS 进行处理。

### 案例 10：异常处理

### 代码演示：异常处理

---

```
SQL> DECLARE
2     newSal emp.sal % TYPE;
3 BEGIN
4     SELECT sal INTO newSal FROM emp;
5 EXCEPTION
6     WHEN TOO_MANY_ROWS THEN
7         dbms_output.put_line('返回的记录太多了');
8     WHEN OTHERS THEN
9         dbms_output.put_line('未知异常');
10 END;
11 /
返回的记录太多了
```

---

## ❖ 自定义异常。

除了预定义异常外，用户还可以在开发中自定义异常，自定义异常可以让用户采用与 PL/SQL 引擎处理错误相同的方式进行处理，用户自定义异常的两个关键点：

- 异常定义：在 PL/SQL 块的声明部分采用 **EXCEPTION** 关键字声明异常，定义方法与定义变量相同。比如声明一个 **myexception** 异常方法是：

```
myexception EXCEPTION;
```

- 异常引发：在程序可执行区域，使用 **RAISE** 关键字进行引发。比如引发 **myexception** 方法是：

```
RAISE myexception;
```

### 案例 11：自定义异常

#### 代码演示：自定义异常

---

```
SQL> DECLARE
2     sal emp.sal%TYPE;
3     myexp EXCEPTION; ①
4 BEGIN
5     SELECT sal INTO sal FROM emp WHERE ename='JAMES';
6     IF sal<5000 THEN
7         RAISE myexp; ②
8     END IF;
9 EXCEPTION
10    WHEN NO_DATA_FOUND THEN
11        dbms_output.put_line('NO RECORDSET FIND!');
12    WHEN MYEXP THEN ③
13        dbms_output.put_line('SAL IS TO LESS!');
14 END;
15 /
SAL IS TO LESS!
PL/SQL procedure successfully completed
```

---

#### 代码解析：

- ① 用 **EXCEPTION** 定义一个异常变量 **myexp**

- ② 在一定条件下用 RAISE 引发异常 myexp
- ③ 在异常处理部分，捕获异常，如果不处理异常，该异常就抛给程序执行者。

## ◇ 引发应用程序异常

在 Oracle 开发中，遇到的系统异常都有对应的异常码，在应用系统开发中，用户自定义的异常也可以指定一个异常码和异常信息，Oracle 系统为用户预留了自定义异常码，其范围介于-20000 到-20999 之间的负整数。引发应用程序异常的语法是：

RAISE\_APPLICATION\_ERROR(异常码, 异常信息)

### 案例 12：引发系统异常

#### 代码演示：引发应用系统异常

---

```
SQL> DECLARE
2     sal emp.sal%TYPE;
3     myexp EXCEPTION;
4 BEGIN
5     SELECT sal INTO sal FROM emp WHERE ename='JAMES';
6     IF sal<5000 THEN
7         RAISE myexp;
8     END IF;
9 EXCEPTION
10    WHEN NO_DATA_FOUND THEN
11        dbms_output.put_line('NO RECORDSET FIND!');
12    WHEN MYEXP THEN
13        RAISE_APPLICATION_ERROR(-20001,'SAL IS TO LESS!'); ①
14 END;
15 /
ORA-20001: SAL IS TO LESS! ②
ORA-06512: 在 line 14
```

---

#### 代码解析：

- ① 引发应用系统异常，指明异常码和异常信息。
- ② 在控制台上显示异常码和异常信息。

如果要处理未命名的内部异常，必须使用 OTHERS 异常处理器。也可以利用 PRAGMA EXCEPTION\_INIT 把一个异常码与异常名绑定。

PRAGMA 由编译器控制，PRAGMA 在编译时处理，而不是在运行时处理。EXCEPTION\_INIT 告诉编译器将异常名与 ORACLE 错误码绑定起来，这样可以通过异常名引用任意的内部异常，并且可以通过异常名为异常编写适当的异常处理器。PRAGMA EXCEPTION\_INIT 的语法是：

```
PRAGMA EXCEPTION_INIT(异常名,异常码)
```

这里的异常码可以是用户自定义的异常码，也可以是 Oracle 系统的异常码。

### 案例 13 : PRAGMA EXCEPTION\_INIT 异常

#### 代码演示 : PRAGMA EXCEPTION\_INIT 异常

---

```
<<outerseg>>
DECLARE
    null_salary EXCEPTION;
    PRAGMA EXCEPTION_INIT(null_salary, -20101); ①
BEGIN
    <<innerStart>> ②
    DECLARE
        curr_comm NUMBER;
    BEGIN
        SELECT comm INTO curr_comm FROM emp WHERE empno = &empno;
        IF curr_comm IS NULL THEN
            RAISE_APPLICATION_ERROR(-20101, 'Salary is missing'); ③
        ELSE
            dbms_output.put_line('有津贴');
        END IF;
    END;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        dbms_output.put_line('没有发现行');
    WHEN null_salary THEN
        dbms_output.put_line('津贴未知'); ④
    WHEN OTHERS THEN
        dbms_output.put_line('未知异常');
END;
```

---

#### 代码解析 :

- ① 把异常名称 `null_salary` 与异常码-20101 关联，该语句由于是预编译语句，必须放在声明部分。也就是说-20101 的异常名称就是 `null_salary`。
- ② 嵌套 PL/SQL 语句块
- ③ 在内部 PL/SQL 语句块中引发应用系统异常-20101。
- ④ 在外部的 PL/SQL 语句块中就可以用异常名 `null_salary` 进行捕获。

## 7. 本章总结

---

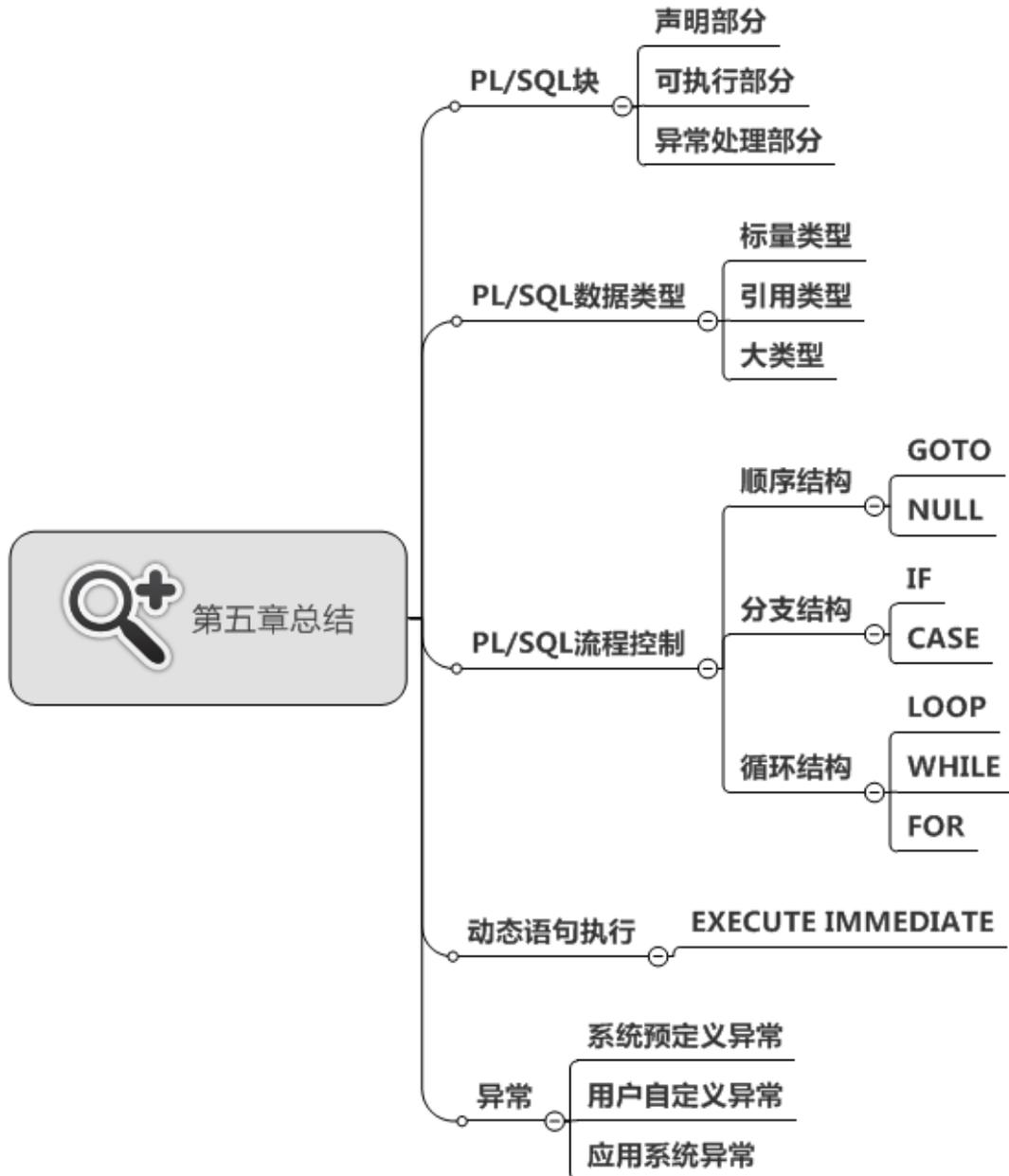
- PL/SQL 是一种块结构的语言，它将一组语句放在一个块中，一次性发送给服务器，PL/SQL 引擎把接收到 PL/SQL 语句块中的内容进行分析，把其中的过程控制语句由 PL/SQL 引擎自身去执行，把 PL/SQL 语句块中的 SQL 语句交给服务器的 SQL 语句执行器执行。
- PL/SQL 的数据类型包括标量数据类型，引用数据类型和存储文本、图像、视频、声音等非结构化得大数据类型（LOB 数据类型）等。
- Oracle 中存在两种属性类型：%TYPE 和%ROWTYPE。
- PL/SQL 程序可通过控制结构来控制命令执行的流程。PL/SQL 中提供三种程序结构：顺序结构、条件结构和循环结构。
- 在 PL/SQL 程序开发中，可以使用 DML 语句和事务控制语句，还可以动态执行 SQL 语句，动态执行 SQL 语句的命令是：EXECUTE IMMEDIATE。
- 在程序运行时出现的错误，称为异常。发生异常后，语句将停止执行，PL/SQL 引擎立即将控制权转到 PL/SQL 块的异常处理部分。PL/SQL 中任何异常出现时，每一个异常都对应一个异常码和异常信息。

## 8. 本章练习

---

1. PL/SQL 有哪些优点？
2. 请描述 PL/SQL 块的结构。
3. 请描述多分枝判断 CASE 的用法。
4. PL/SQL 中有哪些循环控制语句？如何使用它们？
5. 如何执行动态 SQL 语句？
6. 如何自定义异常，如何把自定义异常与异常码绑定？
7. 编写一个程序，输入一个整数，使用循环结构将该数字左右翻转，输出翻转后的结果。
8. 编写一个程序，在 EMP 表查找姓名为 ALLEN 员工，并获取 TOO\_MANY\_ROWS 和 NO\_DATA\_FOUND 异常。
9. 编写一个过程为班级每位同学创建一个用户，用户名和密码都是：“班级号+学号”，并为每位用户授权：CONNECT 和 RESOURCE。

# 章节知识结构图



# 第 6 章

---

## Oracle 应用于 .Net 平台

### 主要内容

- ✓ 回顾 ADO.NET
- ✓ 使用 ADO.NET 连接 Oracle
- ✓ 抽象工厂加入 Oracle

# 1. 回顾 ADO.NET

ADO.NET 是一组用于和数据源进行交互的面向对象类库集，它存在于 .Net Framework 中。通常情况下，数据源可以是各种类型的数据库，利用 ADO.NET 可以访问目前几乎所有的主流数据库，如 Oracle、SQL Server、DB2、Access 等，但数据源同样也能够是文本文件、Excel 文件或者 XML 文件，因此，ADO.NET 可以访问的数据源是很多的。

ADO.NET 由两部分组成：.Net 数据提供程序和数据集。

## ◇ .Net 数据提供程序

ADO.NET 提供了与常用的各种数据源进行交互的一些公共方法，但是对于不同的数据源由于它们各采用的协议是不一样的，所以会采用不同的类库，这些类库称为数据提供程序。主要的数据提供程序如表 1 所示：

数据提供程序	说明	对应的命名空间
SQL Server.NET	访问 SQL Server 数据库的提供程序，由于是针对 SQLServer 数据源而设计，所以采用本提供程序访问 SQL Server 数据源时比采用其它的提供程序，如 OLE DB.NET 访问 SQL Server 数据源，要快速得多	System.Data.SqlClient
OLE DB.NET	访问基于 OLE DB 协议构建的数据源	System.Data.OleDb
Oracle.NET	访问 Oracle 数据库	System.Data.OracleClient
ODBC.NET	访问基于 ODBC 协议构建的数据源	System.Data.Odbc

表 1 主要的 .NET 数据提供程序

ADO.NET 主要负责与数据库服务器建立连接通道，并基于此连接通道实现从数据库中检索数据，并将内存中的数据回送到数据库以提交更新或将在内存中拼写好的 SQL 语句提交到数据库服务器并执行以实现某种数据操作等等。

## ◇ 数据集

数据集可以被看作是一个存储在内存中的离线式的数据库，它很像我们最终存储数据的物理数据库，它将很多物理数据库的机制搬到了内存中。

ADO.NET 的主要部分已经在之前的章节介绍过了，例如：利用 SQL Server.NET 链接 SQL

Server 数据库。本章将主要介绍利用 ADO.NET 连接 Oracle 数据库，实现数据的存取。ADO.NET 的结构如图 1 所示。

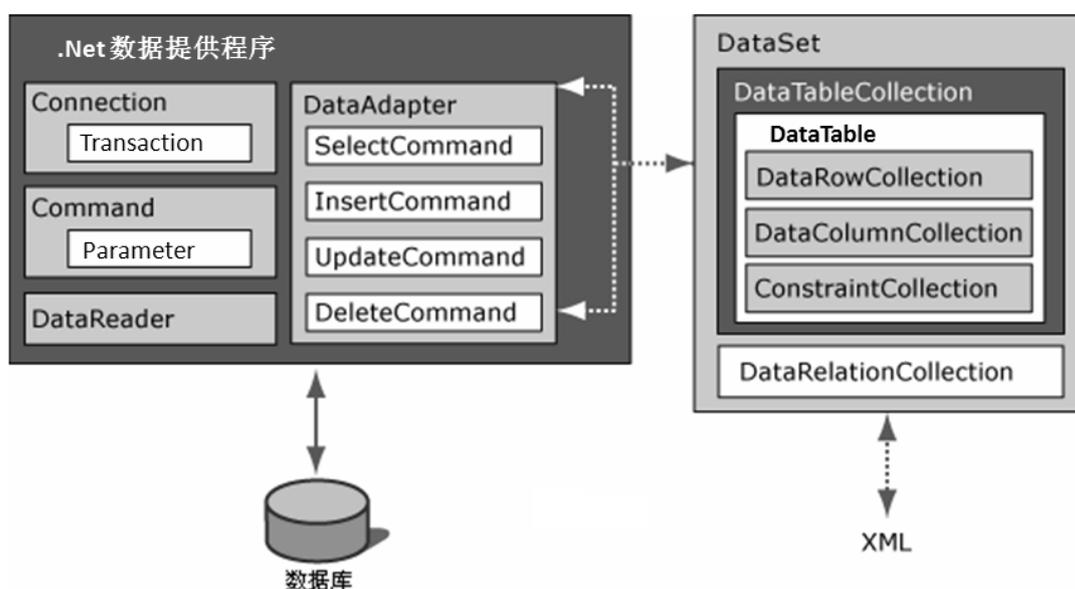


图 1 ADO.NET 结构图

## 2. 使用 ADO.NET 连接 Oracle

使用 ADO.NET 数据访问技术连接 Oracle 数据库和连接 Sql Server 数据库的步骤基本相同：

1. 使用 Connection 对象建立数据库连接。
2. 使用 Command 对象执行数据库操作。
3. 采用连线或者断线的方式进行数据的存取。
4. 使用 Connection 对象的 Close 方法关闭数据库连接。

连接 Sql Server 数据库和 Oracle 数据库所使用的数据提供程序是不一样的，连接 Sql Server 数据库使用的是 SqlServer.NET，包括 SqlConnection、SqlCommand 等数据访问类。而连接 Oracle 数据库则使用的是 Oracle.NET，包括 OracleConnection、OracleCommand 等数据访问类，包含在 System.Data.OracleClient 命名空间下，由于该命名空间默认并没有被添加到项目中来，所以在使用前需要如图 2 所示添加响应的引用才能使用。

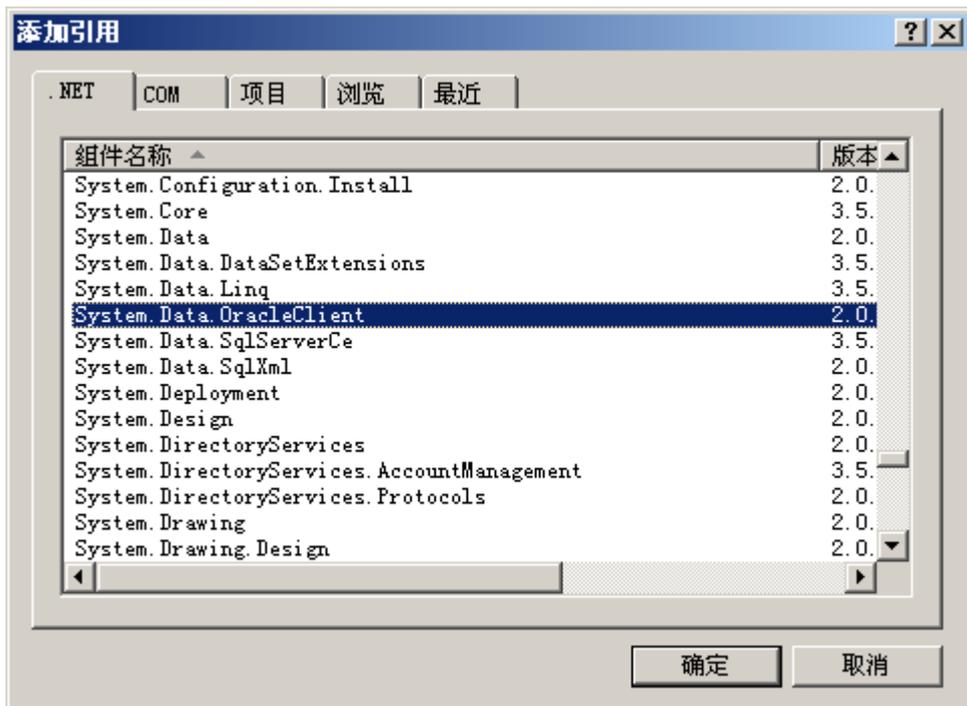


图 2 添加 System.Data.OracleClient 引用

**案例 1：**在 Oracle 数据库中在 System 用户下，创建 UserInfo 表，并插入一定数据，使用 ADO.NET 数据访问技术将 UserInfo 表中的数据检索出来，并显示在 ASP.NET 页面中。UserInfo 表中的数据如图 3 所示。

	USERID	USERNAME	USERPAGE
▶ 1	1	林冲	29
2	2	鲁智深	28
3	3	李逵	23
4	4	宋江	31

图 2 UserInfo 表中数据

### 实现步骤：

1. 使用 OracleConnection 对象建立与 Oracle 之间的连接。

### 代码演示：建立连接

---

```
//连接字符串
string connectionString = "Data Source=MYORCL;User
ID=System;Password=accp;Unicode=True";
//创建 Oracle 连接对象
OracleConnection con = new OracleConnection(connectionString);
```

---

---

```
//打开连接
con.Open();
```

---

2. 创建命令行对象，准备执行检索数据库操作。

### 代码演示：建立命令行对象

---

```
//建立 Sql 查询语句
string sql = "select * from userinfo";
//创建 Oracle 命令行对象
OracleCommand cmd = new OracleCommand(sql, con);
//执行命令行对象
OracleDataReader odr =
cmd.ExecuteReader(CommandBehavior.CloseConnection);
```

---

3. 创建实体类，并遍历结果集，将数据存放到集合中存储。

### 代码演示：创建实体类

---

```
//实体类
public class UserInfo
{
    int userID;

    public int UserID
    {
        get { return userID; }
        set { userID = value; }
    }
    string userName;

    public string UserName
    {
        get { return userName; }
        set { userName = value; }
    }
    string userAge;

    public string UserAge
```

---

---

```
{
    get { return userAge; }
    set { userAge = value; }
}
}
```

---

### 代码演示：遍历结果集

---

```
//遍历结果集
IList<UserInfo> users = new List<UserInfo>();
while (odr.Read())
{
    UserInfo user = new UserInfo();
    user.UserID = Convert.ToInt32(odr["UserID"]);
    user.UserName = Convert.ToString(odr["UserName"]);
    user.UserAge = Convert.ToInt32(odr["UserAge"]);

    users.Add(user);
}
```

---

4. 将集合中的数据绑定到 GridView 控件上显示出来。

### 代码演示：绑定数据

---

```
//绑定 DataGridView 控件
this.GridView1.DataSource = users;
this.GridView1.DataBind();
```

---

根据上面的操作，不难看出连接 Oracle 数据库与连接 SqlServer 数据库除了数据提供程序方面有所区别之外，从操作步骤和实现原理等方面看区别不大。按照上面步骤操作，在页面中运行的效果如图 3 所示。增删改操作和检索操作基本相似，在这里就不再叙述了。



图3 案例效果

### 3. 抽象工厂中加入 Oracle

在设计模式中介绍过抽象工厂设计模式（Abstract Factory），抽象工厂有四种关键角色：抽象工厂、实体工厂、抽象产品、实体产品。抽象工厂模式实现原理强调的是对象组合机制，由在“父工厂”内定义不同的“子工厂”对象来负责不同的目标对象的创建，也就是说利用实体产品由抽象产品来约束，而由实体工厂来创建，实体工厂则由抽象工厂约束，可以有效的发挥工厂模式管理清晰的优点。

**案例 2：**采用抽象工厂模式实现在学员信息管理系统中支持 Access、SQLServer 以及 Oracle 三套数据库的切换（以学员基本信息模块为例），以学员基本信息模块为例给出概要的实现，并实现展示所有学生信息功能。

**案例分析：**本例是抽象工厂课堂案例的延续，需要在项目中多添加一个 Oracle 数据库，实体产品是数据访问对象，三套数据库相当于有三套数据库访问对象，通过三个实体工厂管理三套数据库访问对象，最后使用抽象工厂管理三个实体工厂。

#### 实现步骤：

1. 在 Oracle 数据库中使用 Sql 语句新建 Infos 表，并添加约束和表数据，如图 3 所示：

	STUID	STUNAME	GENDER	AGE	SEAT	ENROLLDATE	STUADDRESS	CLASSNO
1	s100102	林冲	男	22	2	2009-8-9 6:30:10	西安	1001
2	s100104	阮小二	男	26	3	2009-9-16 10:25:29	山东莱芜	1002
3	s100106	卢俊义	男	23	5	2009-8-9 8:00:10	青龙寺	1001

图3 Student 表数据

2. 在 VS2008 中创建空白解决方案，命名为 Test.sln。
3. 在解决方案中添加表示层，并添加 StudentList.aspx 页面。
4. 在解决方案中添加模型层，根据上面的表结构新建 Student.cs 实体类。

### 代码演示：Student 类

---

```
public class Student
{
    string stuID;

    public string StuID
    {
        get { return stuID; }
        set { stuID = value; }
    }

    /**
     * 其他成员... ..
     */
}
```

---

5. 在解决方案中添加数据访问层 IDAL（抽象产品）。

### 代码演示：抽象产品

---

```
public interface IStudentService
{
    //获取所有学生信息
    IList<Student> GetAllStudents();
}
```

---

6. 在解决方案中添加数据访问层 DAL（实体产品），并利用文件夹将不同的实体产品分类，如图 4 所示。

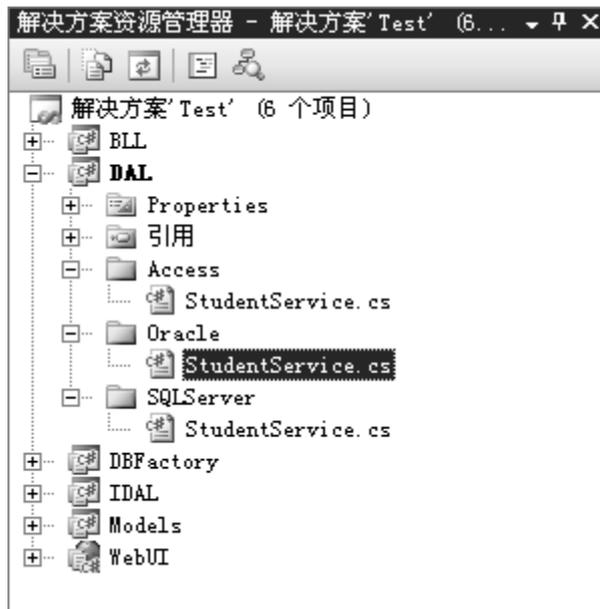


图 4 实体产品

### 代码演示：实体产品

---

```
/// <summary>
/// 获得所有学生信息
/// </summary>
/// <returns>所有学生信息集合</returns>
public IList<Student> GetAllStudents()
{
    //创建 SQL 语句
    string sql = "select * from sys.infos";
    //创建泛型集合
    IList<Student> students = new List<Student>();
    //执行 SQL 语句得到结果集
    OracleDataReader odr = dbh.ExecuteReader(sql);
    //遍历结果集
    while(odr.Read())
    {
        Student student = new Student();

        student.StuID = Convert.ToString(odr["StuID"]);
        student.StuName = Convert.ToString(odr["StuName"]);
        student.StuAddress = Convert.ToString(odr["StuAddress"]);
        student.Seat = Convert.ToInt32(odr["Seat"]);
        student.Gender = Convert.ToString(odr["Gender"]);
    }
}
```

---

---

```
        student.EnRollDate = Convert.ToDateTime(odr["EnRollDate"]);
        student.ClassNo = Convert.ToString(odr["ClassNo"]);
        //添加到泛型集合
        students.Add(student);
    }
    //返回
    return students;
}
```

---

7. 在解决方案中添加业务逻辑层，命名为 StudentManager.cs。

### 代码演示：BLL 层

---

```
/// <summary>
/// 获取所有学生信息
/// </summary>
/// <returns>所有学生信息集合</returns>
public IList<Student> GetAllStudents()
{
    //利用抽象工厂创建实体工厂
    Factory factory = Factory.CreateFactory();
    //利用工厂创建产品
    IStudentService iss = factory.GetStudentService();
    return iss.GetAllStudents();
}
```

---

8. 在解决方案中添加抽象工厂，并添加相应实体工厂，如图 5 所示。

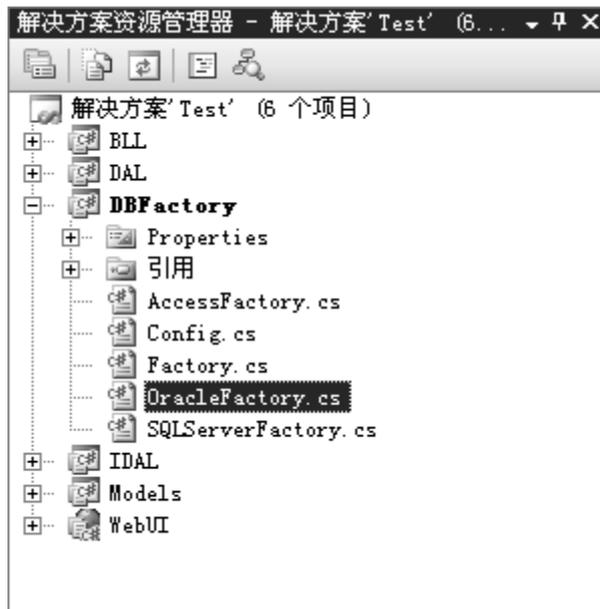


图 5 抽象工厂

### 代码演示：抽象工厂

---

```
//抽象工厂
public abstract class Factory
{
    public static Factory CreateFactory()
    {
        //采用反射技术得到配置文件中的配置信息
        string factoryType = Config.FactoryType;
        Factory factory =

        (Factory)System.Reflection.Assembly.Load("DBFactory").CreateInstance(factoryType);
        return factory;
    }

    //定义子类（实体工厂）的操作规则
    public abstract IStudentService GetStudentService();
}
```

---

9. 在表示层添加数据展示控件，通过设定属性绑定数据提取方法，实现案例目标。

## 4. 本章总结

---

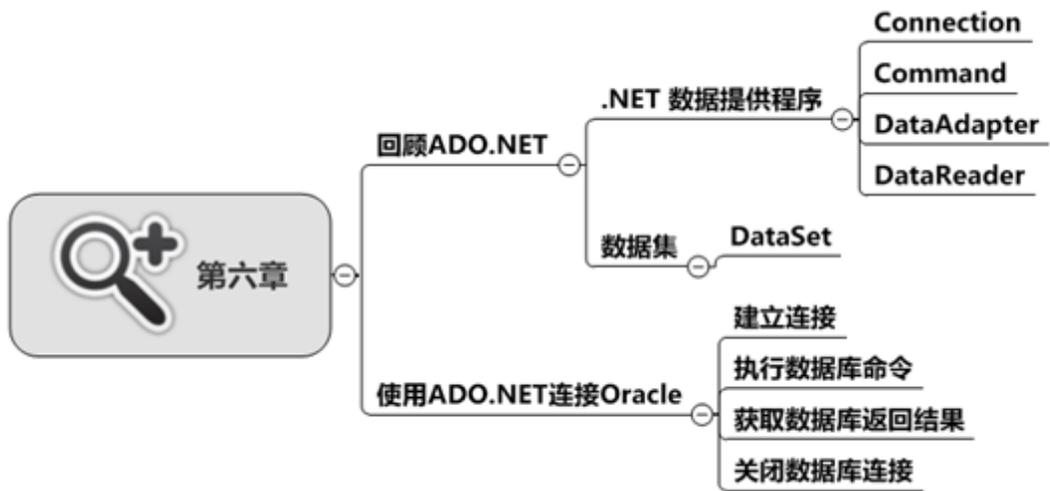
- ADO.NET 是一组用于和数据源进行交互的面向对象类库集,它存在于 .Net Framework 中。通常情况下,数据源可以是各种类型的数据库,利用 ADO.NET 可以访问目前几乎所有的主流数据库,如 Oracle、SQL Server、DB2、Access 等。
- 使用 ADO.NET 技术访问数据库的大致步骤如下:
  1. 使用 Connection 对象建立数据库连接。
  2. 使用 Command 对象执行数据库操作。
  3. 采用连线或者断线的方式进行数据的存取。
  4. 使用 Connection 对象的 Close 方法关闭数据库连接。
- 抽象工厂有四种关键角色:抽象工厂、实体工厂、抽象产品、实体产品。

## 5. 本章练习

---

1. 利用本章所学知识，完成如下功能：
  - 1) 在本章完成项目基础上，继续实现对 Oracle 数据库的增删改操作。
  - 2) 在 StudentList.aspx 页面添加查看详细链接，编写 GetStudentById 等方法。
  - 3) 在 StudentList.aspx 页面中添加删除链接，编写 DeleteStudent 等方法。
  - 4) 在 StudentList.aspx 页面中添加修改链接，编写 UpdateStudent 等方法。

# 章节知识结构图



# 附录

---

## 数据库导入导出

### 主要内容

- ✓ 导出
- ✓ 导入

# 1. Oracle 导入导出

---

Oracle 的备份是 Oracle 操作中常见的工作，常见的备份方案有：逻辑备份（IMP&EXP 命令进行备份）、物理文件备份（脱机及联机备份）、利用 RMAN(Recovery Manager)的增量物理文件系统备份。ORACLE 数据库的逻辑备份分为四种模式：表空间备份(tablespace)、表备份(table)、用户备份(user)和完全备份(full)。Oracle 的逻辑备份是使用 IMP&EXP 命令进行数据导入导出的操作。使用 EXP 命令导出或者使用 IMP 命令导入时，需要 Create Session 系统权限，但是如果导出其他的表，必须拥有权限：EXP\_FULL\_DATABASE。

调用导入导出命令时，首先要估计所需的空空间。EXP 命令导出的文件是二进制文件 (\*.dmp) 只能由对应的 IMP 命令进行读取恢复。导入导出的用途是：

- 备份与恢复
- Oracle 平台更换：可以在相同版本之间进行备份与恢复，Oracle 较低版本的 export 数据文件可以 import 到高版本的 Oracle 数据库中，但是 Oracle 的版本只能是相邻的，不能垮版本。

## 2. EXP 导出数据

---

EXP 命令可以在交互环境下导出数据数据库中的数据，也可以在非交互环境下执行命令。交互环境下的命令执行，是一步一步执行的过程。

### 代码演示：exp 的交互环境

---

```
D:\>exp scott/tiger@my_orcl ①
Export: Release 10.2.0.3.0 - Production on 星期一 10月 19 17:04:14 2009
Copyright (c) 1982, 2005, Oracle. All rights reserved.
连接到: Oracle Database 10g Enterprise Edition Release 10.2.0.3.0 - Production
With the Partitioning, OLAP and Data Mining options
输入数组提取缓冲区大小: 4096 > ②
  导出文件: EXPDAT.DMP > scott.dmp ③
(2)U(用户), 或 (3)T(表): (2)U > 2 ④
导出权限 (yes/no): yes > yes ⑤
导出表数据 (yes/no): yes > yes ⑥
压缩区 (yes/no): yes > no ⑦
已导出 ZHS16GBK 字符集和 AL16UTF16 NCHAR 字符集
. 正在导出 pre-schema 过程对象和操作
```

---

---

```

. 正在导出用户 SCOTT 的外部函数库名
. 导出 PUBLIC 类型同义词
. 正在导出专用类型同义词
. 正在导出用户 SCOTT 的对象类型定义
即将导出 SCOTT 的对象...
. 正在导出数据库链接
. 正在导出序号
. 正在导出簇定义
. 即将导出 SCOTT 的表通过常规路径...
.. 正在导出表                BONUS导出了                0 行
.. 正在导出表                DEPT导出了                10 行
.. 正在导出表                EMP导出了                14 行
.. 正在导出表                SALGRADE导出了                5 行
.. 正在导出表                TBLSTUDENT导出了                3 行
. 正在导出同义词
. 正在导出视图
. 正在导出存储过程
. 正在导出运算符
. 正在导出引用完整性约束条件
. 正在导出触发器
. 正在导出索引类型
. 正在导出位图, 功能性索引和可扩展索引
. 正在导出后期表活动
. 正在导出实体化视图
. 正在导出快照日志
. 正在导出作业队列
. 正在导出刷新组和子组
. 正在导出维
. 正在导出 post-schema 过程对象和操作
. 正在导出统计信息
成功终止导出, 没有出现警告。
D:\>

```

---

### 代码解析：

- ① Exp 是导出命令，该命令后面紧跟“用户名/密码@服务器网络连接”。
- ② Exp 程序导出时使用的缓冲区大小，缓冲区越大，导出速度越快。直接回车代表使用默认值 4096B。
- ③ Exp 命令会把所有要到的数据导出到一个 Dmp 文件中，该步骤是 Exp 询问导出的数据文件名称。
- ④ Exp 程序询问导出整个用户还是导出某个表。默认导出整个用户。
- ⑤ Exp 程序询问是否导出每张表的访问权限。默认导出访问权限。
- ⑥ Exp 程序询问是否导出表中的数据。默认导出数据库表中的数据。

- ⑦ Oracle 表中的数据可能来自不同的分区中的数据块，默认导出时会把所有的数据压缩在一个数据块上，IMP 导入时，如果不存在连续一个大数据块，则会导入失败。

也可以使用 Exp 命令时，设置各种参数，使准备就绪的 Exp 命令不需要与用户交互，按照参数的要求，Exp 命令会一次性执行导出工作。要指定参数，您可以使用关键字：

EXP KEYWORD=value 或 KEYWORD=(value1,value2,...,valueN)

例如: EXP SCOTT/TIGER GRANTS=Y TABLES=(EMP,DEPT,MGR)

参数名	说明
USERID	表示“用户名/密码”。
BUFFER	数据缓冲区大小。以字节为单位，一般在 64000 以上。
FILE	指定输出文件的路径和文件名。一般以.dmp 为后缀名，注意该文件包括完整路径，但是路径必须存在，导出命令不能自动创建路径。
COMPRESS	是否压缩导出，默认 yes。
GRANTS	是否导出权限，默认 yes
INDEXES	是否导出索引，默认 yes
DIRECT	是否直接导出，默认情况，数据先经过 Oracle 的数据缓冲区，然后再导出数据。
LOG	指定导出命令的日志所在的日志文件的位置。
ROWS	是否导出数据行，默认导出所有数据。
CONSTRAINTS	是否导出表的约束条件，默认 yes
PARFILE	可以把各种参数配置为一个文本键值形式的文件，该参数可以指定参数文件的位置。
TRIGGERS	是否导出触发器，默认值是 yes。
TABLES	表的名称列表，导出多个表可以使用逗号隔开。
TABLESPACES	导出某一个表空间的数据。
Owner	导出某一用户的数据。
Full	导出数据库的所有数据。默认值是 no。
QUERY	把查询的结果导出。

表 1 EXP 参数说明

### 代码演示：exp 的非交互环境

---

```
D:\>exp scott/tiger file=employee.dmp tables=(emp,dept)
Export: Release 10.2.0.3.0 - Production on 星期一 10月 19 17:38:25 2009
Copyright (c) 1982, 2005, Oracle. All rights reserved.
连接到: Oracle Database 10g Enterprise Edition Release 10.2.0.3.0 - Production
```

---

---

With the Partitioning, OLAP and Data Mining options

已导出 ZHS16GBK 字符集和 AL16UTF16 NCHAR 字符集

即将导出指定的表通过常规路径...

.. 正在导出表	EMP导出了	14 行
.. 正在导出表	DEPT导出了	10 行

成功终止导出, 没有出现警告。

D:\>

---

### 3. IMP 导入

---

IMP 程序导入就是把 Exp 导出的文件重新导入到数据库的过程。导入时也有一些重要的参数:

- Fromuser:指出导出时 dmp 文件中记载的用户信息。
- Touser:dmp 文件要导入到什么目标用户中。
- Commit:默认是 N,在缓冲区满时是否需要 commit,如果设为 N,需要较大的回滚段。
- Ignore: Oracle 在恢复数据的过程中,当恢复某个表时,该表已经存在,就要根据 ignore 参数的设置来决定如何操作。若 ignore=y, Oracle 不执行 CREATE TABLE 语句,直接将数据插入到表中,如果插入的记录违背了约束条件,比如主键约束,则出错的记录不会插入,但合法的记录会添加到表中。若 ignore=n, Oracle 不执行 CREATE TABLE 语句,同时也不会将数据插入到表中,而是忽略该表的错误,继续恢复下一个表。

#### 代码演示 : Imp 导入

---

```
D:\>imp system/manager file=employee.dmp fromuser=scott touser=employee  
commit=y
```

```
Import: Release 10.2.0.3.0 - Production on 星期一 10月 19 17:54:51 2009
```

```
Copyright (c) 1982, 2005, Oracle. All rights reserved.
```

```
连接到: Oracle Database 10g Enterprise Edition Release 10.2.0.3.0 - Production
```

```
With the Partitioning, OLAP and Data Mining options
```

```
经由常规路由 EXPORT:V10.02.01 创建的导出文件
```

```
警告: 这些对象由 SCOTT 导出, 而不是当前用户
```

```
已经完成 ZHS16GBK 字符集和 AL16UTF16 NCHAR 字符集中的导入
```

```
. 正在将 SCOTT 的对象导入到 EMPLOYEE
```

.. 正在导入表	"EMP"导入了	14 行
.. 正在导入表	"DEPT"导入了	10 行

```
即将启用约束条件...
```

```
成功终止导入, 没有出现警告。
```

```
D:\>
```

---

## 4. 常见问题

---

➤ 数据库对象已经存在

一般情况, 导入数据前应该彻底删除目标数据下的表, 序列, 函数/过程, 触发器等。

数据库对象已经存在, 按缺省的 `imp` 参数, 则会导入失败。

如果用了参数 `ignore=y`, 会把 `exp` 文件内的数据内容导入。

如果表有唯一关键字的约束条件, 不合条件将不被导入。

如果表没有唯一关键字的约束条件, 将引起记录重复。

➤ 数据库对象有主外键约束

不符合主外键约束时, 数据会导入失败。

解决办法: 先导入主表, 再导入依存表。

`disable` 目标导入对象的主外键约束, 导入数据后, 再 `enable` 它们。

➤ 权限不够

如果要把 A 用户的数据导入 B 用户下, A 用户需要有 `imp_full_database` 权限。

➤ 导入大表(大于 80M) 时, 存储分配失败

默认的 `EXP` 时, `compress = Y`, 也就是把所有的数据压缩在一个数据块上。

导入时, 如果不存在连续一个大数据块, 则会导入失败。

导出 80M 以上的大表时, 记得 `compress = N`, 则不会引起这种错误。

➤ `Imp` 和 `Exp` 使用的字符集不同

如果字符集不同, 导入会失败, 可以改变 `unix` 环境变量或者 `NT` 注册表里 `NLS_LANG` 相关信息。

➤ `Imp` 和 `Exp` 版本不能往上兼容

`Imp` 可以成功导入低版本 `Exp` 生成的文件, 不能导入高版本 `Exp` 生成的文件根据情况我们可以用。