Day01-Oracle 基础

学习目标：

- 数据库简介
- **Oracle** 用户管理
- **Oracle** 权限角色

1. 数据库简介

1.1 数据库(DataBase)

数据库是一种软件产品，是用于存放数据，管理数据的存储仓库，是有效组织在一起的数据集合。

1.2 常用数据库软件

- **大型数据库**
  - ✓ **Oracle** Oracle 是著名 Oracle(甲骨文)公司的数据库产品，它是世界上第一个商品化的关系型数据库管理系统，也是第一个推出和数据库结合的第四代语言开发工具的数据库产品。

    Oracle 公司的软件产品丰富，包括 Oracle 服务器产品，Oracle 开发工具和 Oracle 应用软件。其中最著名的就是 Oracle 数据库 ， 目前最新的版本是 Oracle11g。
  - ✓ **DB2** 是 IBM 的关系型数据库管理系统，DB2 有很多不同的版本， 可以运行在从掌上产品到大型机不同的终端机器上。DB2 在高端数据库的主要竞争对手是 Oracle。
  - ✓ **Sybase** 是美国 Sybase 公司研制的一种关系型数据库系统 ， 是较早采用 C/S 技术的数据库厂商，是一种典型的 UNIX 或 Windows NT 平台上客户机/服务器环境下 的大型数据库系统 ， 在国内大中型系统中具有广泛的应用。
- **中小型数据库**
  - ✓ **Sql Server Microsoft SQL Server 是运行在 Windows NT 服务器上 ， 支持 C/S 结构的数据库 管理系统。它采用标准 SQL 语言。**
  - ✓ **Mysql** 是一个小型关系型数据库管理系统 ， 开发者为瑞典 MySQL AB 公司。在 2008 年 1 月 16 号被 Sun 公司收购。而 2009 年 SUN 又被 Oracle

收购。MySQL 体积小、速度快、总体拥有成本低 ， 尤其是开放源码，许多中小型网站为了降低网站总体拥有成本而选择了 MySQL 作为网站数据库。

- **小型数据库**

  - ✓ **Access** Microsoft Office Access(前名 Microsoft Access)是由微软发布的关联式数据库管理系统，是 Microsoft Office 的成员之一。

1.3 数据库分类（关系型和非关系数据库）

当前主流的关系型数据库有 Oracle、DB2、Microsoft SQL Server、Microsoft Access、MySQL 等。

非关系型数据库有 NoSql、MonGOdb。

关系型数据库通过**外键关联**来建立表与表之间的关系，非关系型数据库通常指数据以**对象的形式存储在数据库中**，而对象之间的关系通过每个对象**自身的属性**来决定

1.4 SQL(Structured Query Language)

SQL(Structured Query Language)语言是用来在关系数据库上执行数据操作、检索及维护所使用的标准语言 ，是一个综合的、通用的关系数据库语言。大多数数据库都使用相同或者相似的语言来操作和维护数据库。

SQL 语言可以用来查询数据 ， 操纵数据 ， 定义数据 ， 控制数据 ， 使软件开发人员、数据库管理员都可以通过 SQL 语言对数据库执行特定的操作。

1.5 DBA

DBA: Database Administrator 数据库管理员

1.6 Table(表)

表是数据库存储的基本单元，对应于现实世界中的实体对象，比如部门、职员等， 表是一个二维结构，由行和列组成，横向为行(Row)，也叫记录(Record)，用来表示实体的数据，比如一个职员的相关信息。纵向为列(Column)，也叫作字段(Filed)，用来表示实体的属性，比如职员的薪水。

2. Oracle 用户管理

2.1 sys 和 system 用户

当我们每创建一个数据库实例的时候，就会自动的生成三个用户：

sys 用户：**超级管理员（群主）**，权限最大， 有 create database 的权限

system 用户：**普通管理员用户（普通群管理员）**，没有 create database 的权限

scott 用户：**普通用户（普通的群成员）** 在默认的情况下，scott 用户是锁定状态(lock user)，一般我们启用它。如果安装的时候，忘记对 scott 用户解锁，比如 scott 可以通过 system 来对该用户解锁。

解锁步骤：

1、先使用 system 登录，运行 sqlplus

2、使用命令：alter user scott account unlock

- **sys 和 system 的主要区别：**

地位	**sys** :Oracle 的一个超级用户	**system:**Oracle 默认的系统管理员，拥有 DBA 权限
作用	主要用来维护系统信息和管理实例	通常用来管理 Oracle 数据库的用户、权限和存储等
登录身份	只能以 SYSDBA 或 SYSOPER 角色登录	只能以 Normal 方式登录

- **normal 、sysdba、 sysoper 有什么区别：**

Normal：是普通用户

Sysdba：拥有最高的系统权限

Sysoper：主要用来启动、关闭数据库;

2.2 oracle 后台服务

OracleDBConsoleorcl 网络企业管理器服务

OracleOraDb10g_home1TNSListener 监听服务(监听端口 1521)(设置成手动)

OracleServiceORCL 数据库实例服务(设置成手动)

2.3 oracle 开发工具

1）slqplus：工具是 oracle 自带的工具软件

在**开始--程序--oracle oradb_home11g--application development--sql plus**

在**运行栏中输入**：sqlplus 即可，也可以使用 sqlplus 进入 oracle 数据库 oracle 管理工具介绍

2）pl/sql developer：属于第三方软件，主要用于开发，测试，优化 oracle pl/sql 的存储过程比如：触发器，此软件 oracle 不带，需要单独安装。

3. sql*plus 常用命令

1）**conn[ect]** (可用于切换用户使用)

用法：conn 用户名/密码@网络服务名[as sysdba/sysoper]

当用特权用户身份连接时，必需带上 as sysdba 或 as sysoper

2）**disc[onnect]**

说明：该命令用来断开与当前数据库的连接，但不退出 sqlplus 窗口

3）**passw[ord]**

说明：该命令用于修改用户的密码，如果要想修改其它用户的密码，需要 sys/system

特别说明：如果给自己改密码则可以不带用户名。如果给别人修改密码需带用户名**(必需是 system 或者 sys 用户才可以修改)**

4）**show user**

说明：显示当前用户名

5）exit

说明：该命令会断开与数据库的连接，同时会退出 sql*plus

4. 创建用户

在 oracle 中要创建一个新的用户使用 create user 语句，一般是具有 dba(数据库管理员)的权限才能使用。

1） 创建用户

命令：**create user 用户名 identified by 密码;**

2） **给用户修改密码**

如果给别人修改密码则需要具有 dba 的权限，或是拥有 alter user 的系统权限

命令：**alter user 用户名 identified by 新密码**

3）**修改自己的密码**

如果给自己修改密码可以直接使用

命令：password 用户名

问题：创建好的用户无法正常登录？

oracle 中用户建立后是无法正常登录的，只有在数据库管理员(DBA)对用户分配相应的权限后，用户才可以登录。

5. oracle 用户分配权限和角色

概述：创建的新用户是没有任何权限的，甚至连登录的数据库的权限都没有，需要为其指定相应的权限。

给一个用户赋权限使用使令 grant，回收权限使用命令 revoke

赋权限基本语法：grant 权限/角色 to 用户名;

回收权限基本语法：revoke 权限/角色 from 用户名;

5.1 分配权限

语法：grant create 权限 to 用户名;

例如：给小明赋予会话的权限

```
sql> grant create session to xiaoming;
```

5.2 分配角色

也可以按**角色**对用户分配权限

语法：grant 角色名 to 用户名;

例如：授予小明 resource 角色

```
sql> grant resource to xiaoming;
```

5.3 oracle 中权限的概念

权限分为系统权限与对象权限。

系统权限：是数据库管理相关的权限：create session(登录权限)、create table(创建表权限)、create index(创建索引权限)、create view(创建视图权限)、create sequence(创建序列权限)、create trriger(创建触发器权限)

对象权限：是用户操作数据对象相关的权限。比如对表的增删改查（insert 增、delete 删、update 改、select 查）；

5.4 角色

在 oracle 中角色分为：预定义角色和自定义角色；

预定义角色：把常用的权限集中起来，形成角色。常见的角色有：dba、connect、resource 等；

connnect 角色：是授予用户的最基本的权利，能够连接到 oralce 数据库中，并在对其他用户的表有访问权限时，做 SELECT、UPDATE、INSERTT 等操作。Create session--建立会话；Alter session--修改会话；Create view--建立视图。Create sequence--建立序列等权限

resource 角色：具有创建表、序列、视图等权限。Create table--建表；Create trigger--建立促发器；Create procedure--建立过程；Create sequence—建立序列；Create type--建立类型等权限；

dba 角色：是授予系统管理员的，拥有该角色的用户就能成为系统管理员了，它拥有所有的系统权限。

自定义角色：按需定制一定权限形成角色，可以作为预定义角色的补充。来满足用户的需求。

6. 删除用户

概述：一般以 dba 的身份去删除某个用户，如果用其它用户去删除用户则需要具有 drop user 的权限。

命令：drop user 用户名 [cascade]可选参数 cascade

在删除用户时，**注意：** 如果要删除的用户，已经创建了表，那么就需要在删除的时候带一个参数 cascade; Cascade **有级联的作用**

练习

1、创建 xiaoming，并赋予 connect 和 resource

创建用户基本语法：create user 用户名 identified by 密码;(掌握)

```
sql>create user xiaoming identified by m123;
```

给用户授权基本语法：grant 权限/角色 to 用户名; (掌握)

```
sql>grant connect to xiaoming;
sql>grant resource to xiaoming;
```

2、切换用户

切换用户基本语法：connect 用户名/密码; (掌握)

```
sql>conn xiaoming/m123;
```

3、xiaoming 修改密码

修改密码基本语法：password 用户名;

```
sql>password xiaoming;
```

管理员修改密码基本语法：alter user 用户名 identified by 新密码; (掌握)

(sys 或 system 用户修改其它用户的方法)

```
sql>alter user xiaoming identified by xiaoming123;
```

4、使用 xiaoming 建表

建表基本语法：create table 表名(字段属性);

```
sql>create table users(id number);
```

5、添加数据

添加数据基本语法：insert into 表名 values(值);

```
sql>insert into users values(1);
```

6、查询数据

查询表内容基本语法：select * from 表名;

```
sql>select * from users;
```

7、删除表

删除表基本语法：drop table 表名;

```
sql>drop table users;
```

8、回收权限(需 sys 或 system 用户)

回收权限基本语法：revoke 权限/角色名 from 用户名;

```
sql>revoke connect from xiaoming;
sql>revoke resource from xiaoming;
```

9、删除用户

删除用户基本语法：drop user 用户名 [cascade];

当我们删除一个用户的时候，如果这个用户自己已经创建过数据对象，那么我们在删除该用户时，需要加[cascade]，表示把这个用户删除的同时，把该用户创建的数据对象一并删除。

```
sql>drop user xiaoming;
```

Day02-表的操作

学习目标：

- **Oracle 数据类型**
- 表空间（**Oracle** 独有）
- 表结构操作

Oracle 数据类型

1、 char(size)

存放字符串，它最大可以存放 2000 个字符，是定长。

例子：

```
create table test1(name char(32));
```

说明：

test1 表中的 name 字段最多存放 32 个字符，

不足 32 个字符 oracle 会用空格补齐,如果超过会报错。

2、 varchar2(size)

存放字符串，它最大可以存放 4000 个字符，是变长。

举例说明：

```
create table test2(name varchar2(16));
//test2 表中的 name 字段最多存放 16 个字符，实际有几个字符就占几个字符的空间，如果超过会报错。
```

注意：如果我们的数据的长度是固定的，比如编号(8 位)，最好使用 char 来存放，因为这样存取的速度

就会提高。如果存放的数据长度是变化的，则使用 varchar2 来存放。

3、 nchar(n)

以 Unicode 编码来存放字符串，它最大可以存放 2000 个字符，是定长。

举例说明：

create table test3(name nchar(32));

//与 char 类似，只是以 Unicode 编码存放字符串

4、 nvarchar(n)

以 Unicode 编码来存放字符串，它最大可以存放 4000 个字符，是变长。

举例说明：

create table test4(name nvarchar2(16));

//与 varchar2 类似，只是以 Unicode 编码存放字符串

5、 clob 型

字符型大对象，它最大可以存放 8TB，是变长。（nvarchar(4000)满足不了需求）

与字符型的 char/varchar2/nchar/nvarchar2 的使用方式一样。

6、 blob 型

二进制数据，可存放图片、声音，它最大可以存放 8TB，是变长。

与字符型的 char/varchar2/nchar/nvarchar2 的使用方式一样。

说明：一般情况下很少使用数据库来存放媒体类文件，一般只使用数据库来记录媒体类文件的 URL 地址。如果考虑到文件的安全性，可以存放到数据库中来保存。

7、 number 型

存放整数，也可存放小数，是变长。number(p,s)//p 代表整个位数，s 代表小数位

说明：

number(5,2)： 表示一个小数有 5 位有效数，2 位小数。范围-999.99～999.99

如果数值超出了位数限制就会被截取多余的位数。但在一行数据中的这个字段输入 575.316，则真正保存到字段中的数值是 575.32。

number(5) ： 等价于 number(5,0) 表示一个五位整数，范围-99999～99999。

输入 57523.316，真正保存的数据是 57523

8、 date 类型

用于表示时间，(年/月/日/时/分/秒)，是定长。

举例说明：

create table test5(birthday date);

//添加时个要使用默认格式 insert into test5 values('11-11 月-11');

如使用 insert into test5 values('2011-11-11');则报错。

说明：oracle 日期有默认格式为：DD-MON-YYYY,日-月-年；如果我们希望使用自己习惯的日期添加，也可以，但是需要借助 oracle 函数来添加。

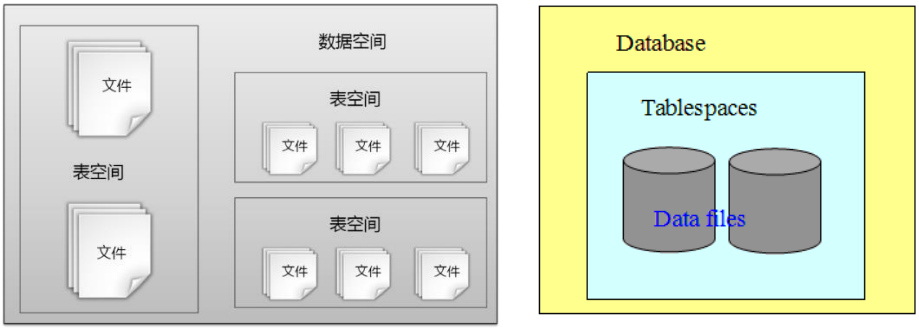
insert into test5 values(to_date('2015-5-3','yyyy-mm-dd'))

表空间（Oracle 独有）

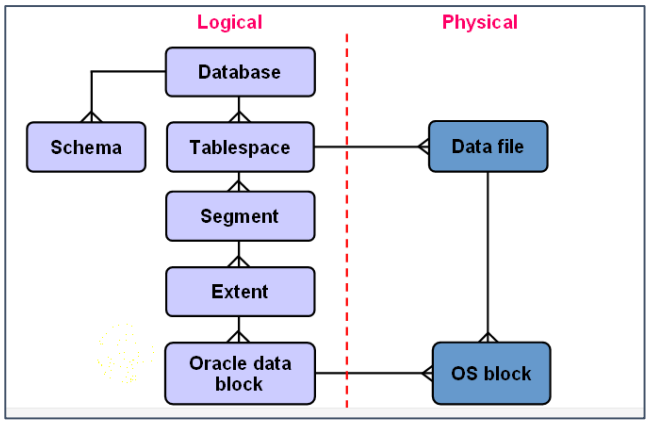
表空间概述：

表空间：表存在的空间，一个表空间是指向具体的数据文件。创建表空间的目的：主要是为了提高数据库的管理性能。

表空间属性：一个数据库可以包含**多个表空间**，**一个表空间**只能属于**一个数据库**；**一个表空间**包含多个**数据文件**，**一个数据文件**只能属于**一个表空间**；**表空间**可以划分成更细的逻辑存储单元



从逻辑的角度来看，一个数据库（database）下面可以分多个表空间（tablespace）；一个表空间下面又可以分多个段（segment）；一个数据表要占一个段（segment），一个索引也要占一个段（segment ）。 一个段（segment）由多个区间（extent）组成，一个区间又由一组连续的数据块（data block）组成。这连续的数据块是在逻辑上是连续的，有可能在物理磁盘上是分散。



那么从物理的角度上看，一个表空间由多个数据文件组成，数据文件是实实在在存在的磁盘上的文件。这些文件是由 oracle 数据库操作系统的 block 组成的。

Segment（段）：段是指占用数据文件空间的通称，或数据库对象使用的空间的集合；段可以有表段、索引段、回滚段、临时段和高速缓存段等。

Extent（区间）：分配给对象（如表）的任何连续块叫区间；区间也叫扩展，因为当它用完已经分配的区间后，再有新的记录插入就必须在分配新的区间（即扩展一些块）；一旦区间分配给某个对象（表、索引及簇），则该区间就不能再分配给其它的对象.

创建表空间：

- **创建表空间必须使用 system 用户创建，语法如下：**

```
create tablespace ts_hp（表空间名）
    datafile 'E:\HP.DBF'   //指定指向的数据文件路径
    size 5m     //表空间大小为 5m
    autoextend on next 2m     //自动扩展，每次扩展 2m
    maxsize unlimited;       //最大扩展量没有限制
```

- **创建用户时要指定默认的表空间：**

创建用户指定默认表空间

```
create user zhangsan identified by 123 default tablespace hy_ts;
```

修改用户默认表空间：

```
alter user xiaoming default tablespace ts_hp
```

- **查看系统表空间：**

查看表空间：

```
select * from    v$tablespace
```

查看用户默认表空间：

```
select username,default_tablespace from dba_users where username='SCOTT';
```

注意：这里的用户名 SCOTT 必须是大写

表结构的操作

1. 创建表：

语法：

```
create   table   表的名称（字段名称 1   数据类型, 字段名称 2  数据类型）;
```

举例：创建学生表，字段和类型如下所示：

字段	字段类型	说明
id	整型	number
name	字符型	varchar2
sex	字符型	char（2）
brithday	日期型	date
score	小数型	number(3,1)
resume（简历）	大文本型	clob

代码：

```
create table students(
        id number,
        name varchar2(64),
        sex char(2),
        brithday date,
        score number(3,1),
        resume clob
);
```

2. 修改表

使用 alter table 语句添加、修改或删除列的语法

- **添加列基本语法：**

语法：

alter table TABLENAME add(columnname datatype);

alter table 表名 add(列名(字段名) 列类型(字段类型));

例如：给学生表添加班级编号

alter table students add(class_id number);

- **修改列基本语法：**

语法：

alter table TABLENAME **modify**(columnname datatype);

alter table 表名 **modify**(列名(字段名) 列类型(字段类型));

例如：学生姓名变成 varchar2(30)

alter table students modify(name varchar2(30));

3. 删除列：

- **删除多列语法：**

语法：

alter table TABLENAME **drop(columnname,columnname2,...);**

alter table 表名 drop(列名 1,列名 2,...);

- **删除单列语法：**

语法：

alter table TABLENAME **drop column** COLUMNNAME;

alter table 表名 **drop column** 列名(字段名);

例如：删除学生表的 score 字段

alter table students **drop column** score;

alter table students **drop(score)**;

4. 修改表的名称：

语法：

rename OldTableName to NewTableName;

rename 表名 to 新表名;

例如：把学生表名 students 修改成 stu

SQL>rename students to stu;

5. 修改列名：

语法：

alter table 表名 rename column 旧列名 to 新列名;

例如：把学生表 resume 修改成 intro

SQL>alter table stu rename column resume to intro;

6. 查看表结构基本语法：

语法：

desc TABLENAME;

desc 表名;

7. oracle 表的管理--添加数据

语法：使用 insert 语句向表中插入数据

insert into table[(column [,column...])] values(value [,value...]);

insert into 表名[(列名 [,列名 2...])] values(值 [,值 2...]);

插入全部数据：

insert into 表名 values(值 [,值 2...]);

注意事项：

1、插入的数据应与字段的数据类型相同。

2、数据的大小应在列的规定范围内，

例如：不能将一个长度为 80 的字符串加入到长度为 40 的列中。

3、在 values 中列出的数据位置必须与被加入的列的排列位置相对应。

4、字符和日期型数据应包含在单引号中。

5、插入空值，不指定或 insert into table values(null);

6、给表的所有列添加数据时，可以不带列名直接添加 values 值。

例如：向 students 添加数据

```
insert into students (id,name,sex,brithday,fellowship,resume) values
   (1,'张三','男','11-11 月-01',23.34,'hello');

insert into students values(2,'李四','男','11-11 月-02',67.34,'hello2');
insert into students values(3,'王五','女','11-11 月-03',671.34,'hello3');
```

Day03-Oracle 约束

学习目标：

- 什么是约束

- 约束的功能
- 约束分类（重点）
- 案例商店售货系统（掌握）
- 约束维护

1.什么是约束？

1.1 约束定义

约束：是强加在表上的规则或条件。确保数据库满足业务规则。保证数据的完整性。当对表进行 DML 或 DDL 操作时，如果此操作会造成表中的数据违反约束条件或规则的话，系统就会拒绝执行这个操作。约束可以是列一级别的 也可以是表级别的。定义约束时没有给出约束的名字，ORACE 系统将为该约束自动生成一个名字，其格式为 SYS_Cn，其中 n 为自然数(强烈建议各位在创建表或增加约束时，给约束定义名称。).

1.2 约束功能

约束的功能：实现一些业务规则，防止无效的垃圾数据进入数据库，维护数据库的完整性(完整性指正确性与一致性)。从而使数据库的开发和维护都更加容易。

2.约束分类

约束分为： 非空（NOT NULL）约束、 唯一（UNIQUE）约束、主键（PRIMARY KEY）约束、外键（FOREIGN KEY）约束、条件（CHECK）约束。

2.1 not null 非空

非空（NOT NULL）约束：顾名思义，所约束的列不能为 NULL 值。否则就会报错

举例：

```
create table user1(id number,name varchar2(30) not null);
insert into user1 values(001,'');//会报错
```

2.2 unique 唯一

唯一（UNIQUE）约束：在表中每一行中所定义的这列或这些列的值都不能相同。必须保证唯一性。否则就会违法约束条件。

用于指定列的值不能重复，可以为 null

举例：

```
create table user2(id number unique,name varchar2(30));
insert into user2 values(1,111);//id 输入重复的值是会报错
```

注意：oracle 中 unique 可以为 null，而且允许多行为 null

2.3 primary key 主键

主键（PRIMARY KEY）约束：唯一的标识表中的每一行，不能重复，不能为空。 创建主键或唯一约束后，ORACLE 会自动创建一个与约束同名的索引（UNIQUENES 为 UNIQUE 唯一索引）。需要注意的是：每个表只能有且有一个主键约束。

举例：

```
create table user3(id number primary key,name varchar2(30));
insert into user3 values(1,111);
insert into user3 values(1,111);//报错，唯一性
insert into user3 values(null,111);//报错，不能为 null
```

特别说明 primary key 与 unique 的区别：

1、一张表可以有多个 unique(唯一)约束；

2、一张表只能有一个主键；

3、设置为主键的列不能有 null 值；

2.4 foreign key 外键 references

外键（FOREIGN KEY）约束：用来维护从表（Child Table）和主表（Parent Table）之间的引用完整性.能够维护数据库的数据一致性，数据的完整性。防止错误的垃圾数据入库；

用于定义主表和从表之间的关系，外键约束要定义在从表上，主表则必需具有主键约束或是 unique 约束，当定义外键约束后，要求外键列数据必需在主表的主键列存在或是为 null

```
create table class(id number primary key,name varchar2(32));
create table stus(id number primary key,
                  name varchar2(36) not null,
                  classid number references class(id)
                 );
```

特别说明：froeign key 外键的细节

1、外键指向主键列；

2、外键可以指向 unique 列；

3、建表时先建主表，再建从表；删除表先删从表，再删主表。

4、外键列属性值要与主键或 unique 列属性值的类型保持一致

5、外键列的值，必需在主键列中存在。但外键列的值允许为 null

2.5 check 检查

条件（CHECK）约束：表中每行都要满足该约束条件。条件约束既可以在表一级定义也可以在列一级定义。在一列上可以定义任意多个条件约束。

举例：

```
create table user4(id number primary key,
sal number check(sal>=1000 and sal<=2000),
sex char(2) check(sex in('男','女')));
insert into user4 values(1,1000,'男');//sal 列的值不满足 1000 至 2000，报错。
```

3.案例：（商店售货系统）（掌握创建约束）

商店售货系统表设计案例：

现有一个商店的数据库，记录客户及其购物情况，由下面三个表组成：

商品 goods 表：

商品号 goodsId（主键），商品名 goodsName（非空），

单价 price（大于零），商品类别 category，供应商 provider；

客户 customer:

客户号 customerId（主键），姓名 name（非空），住址 address，

邮箱 email，性别 sex（默认男，check），身份证 cardId（唯一）；

购买 purchase：

客户号 customerId（外键），商品号 goodsId（外键），

购买数量 nums（1 到 30 个）；

请用 SQL 语言完成下列功能：

1. 建表，在定义中要求声明：

(1).每个表的主外键；

(2).客户的姓名不能为空值；

(3).单价必须大于 0，购买数量必须在 1 到 30 之间；

(4).邮箱不能够重复；

(5).客户的性别必须是 男 或者 女，默认是男；

- **商品表：**

商品 goods 表：

商品编号：**goodsId**　商品名称：**goodsName**

单价：**price**　　商品类别：**category**

供应商：**provider**

代码：

```
create table goods(goodsId number primary key,
                   goodsName varchar2(36),
                   price number check(price>0),
                   category varchar2(64),
                   provider varchar2(64)
                   );
```

- **客户表：**

客户 customer 表：

客户编号：**customerId**　姓名：**name，**

住址：**address**　　邮箱：**email，**

性别：**sex**　　身份证：**idCard**

代码：

```
create table customer(customerId number primary key,
                      name varchar2(32) not null,
                      address varchar2(64),
                      email varchar2(64) unique,
                      sex char(2) default'男' check(sex in('男','女')),
                      idCard varchar2(20)
                     );
```

- **购买表：**

购买 purchase 表：

客户编号：customerId　　商品编号：goodsId

购买数量：nums

代码：

```
create table purchase(customerId number references customer(customerId),
                      goodsId number references goods(goodsId),
                      nums number check(nums>=1 and nums<=30),
                      primary key(customerId,goodsId)
                      );
```

4.维护约束（掌握）

4.1 修改约束

如果在建表时忘记建立必要的约束，则可以在建表后使用 alter table 命令为表增加约束；

注意：

1）增加 not null 约束使用 modify（因为字段（列）默认都是可以为空）
2）增加其它四种约束使用 add。

- **添加或修改not null约束**

语法：

alter table 表名 modify 字段名 not null;

- **添加或修改unique(唯一)、primary key(主键)、foreign key(外键)和check(检查)约束**

语法：

alter table 表名 add constraint 约束名 约束种类(字段);

- **修改售货系统表设计方案**

1、增加 purchase 表主键；

alter table purchase add constraint PK_PURCHASE_PID primary key (cId,gId);

2、客户的姓名不能为空值；--增加商品名也不能为空

alter table goods modify goodsName not null;//添加非空约束要用 modify

3、邮箱不能够重复；--增加身份证也不重复

alter table customer add constraint unique_cardId unique(cardId);

alter table customer add constraint UK_EMAIL unique(email);

4、增加客户的住址只能是(伊滨区，老城区，洛龙区)

alter table customer add constraint check_address check(address in('伊滨区','老城区','洛龙区'));

4.2 删除约束

当不再需要某个约束时，可以删除。

删除约束基本语法：

alter table 表名 drop constratint 约束名称;

约束名称指的是：一个表的每一个约束都对应一个名称。约束名称用户没有设置时，系统会自动分配一个名称。

特别说明：

在删除主键约束的时候，可能有错误，

比如：alter table 表名 drop primary key;

这是因为如果在两张表存在主从关系，那么在删除主要的主键约束时，

必需带上 cascade 选项。

基本语法：

alter table 表名 drop primary key cascade;

4.3 约束命名规范

约束名称：建议自己定义一套命名规则，否则使用系统生成的约束名，很难能把它和对应的表、字段联系起来。

约束命名规则：

非空约束：	**NN_表名_列名**
唯一约束：	**UK_表名_列名**
主键约束：	**PK_表名**
外键约束：	**FK_表名_列名**
条件约束：	**CK_表名_列名**

4.4 显示约束信息（了解）

1）显示表约束信息

通过查询数据字典视图 user_constraints，可以显示当前用户所有的约束的信息。

语法：

```
select constraint_name, constraint_type, status, validated
  from user_constraints
  where table_name = '表名';
```

注意：表名要大写

2）当然也有更容易的方法，直接用 pl/sql developer 查看即可。

4.5 表级定义 列级定义（了解）

■ 列级定义

列级定义：是在定义列的同时定义约束。

举例：如果在 department 表定义主键约束

```
create table department(
        dept_id number(12) constraint pk_department primary key,
        name varchar2(12), loc varchar2(12));
```

■ 表级定义

表级定义：指在定义了所有列后，再定义约束。

举例：建立 employee 表时定义主键约束和外键约束为例：

```
create table employee(emp_id number(4),
        name varchar2(15),
        dept_id number(2),
        constraint pk_employee primary key (emp_id),
        constraint fk_department foreign key (dept_id) references department4(dept_id)
        );
```

特别说明：not null 约束不可以出现在表级定义中，not null 约束只能在列级上定义。

一般情况下，我们使用列级定义即可。但是如果遇到定义复合主键(两列一起被定义为

主键)时，需要用到表级定义。

4.6 联合主键（了解）

举例，将 id 与 name 定义为复合主键：

```
create table test(id number,
name varchar2(64),nums number,
constraint pk_id_name primary key(id,name));
```

特别说明：不推荐大家使用复合主键。

Day04-Oracle 查询基础

学习目标：

- 修改数据
- 删除数据
- 简单查询
- 条件查询
- 分组查询

oracle 修改数据

使用 update 语句修改表中数据。

Update 语句基本语法：

```
update 表名 set 列名=表达式 [,列名 2=表达式 2,...][where 条件];
```

注意事项：

1、update 语法可以用新值更新原有表行中的各列；

2、set 子句指示要修改哪些列和要给予哪些值；

3、where 子句指定应更新哪些行。如没有 where 子句，则更新所有的行。**（特别小心）**

对 students 中的数据进行修改

将张三的性别改成女

SQL>update students set sex='女' where name='张三';

把张三的奖学金改为 10

SQL>update students set fellowship=10 where name='张三';

把所有人的奖金都指高 10%

SQL>update students set fellowship=fellowship*1.1;

将没有奖学金同学的奖学金改成 10 元

SQL>update students set fellowship=10 where fellowship is null;

特别注意：当修改空记录时应用 is null 而不能使用=null 或=''

oracle 表的管理--删除数据

基本语法：

delete from TABLENAME [where where_definition];

delete from 表名 [where 条件表达式];

注意事项：

1、如果不使用 where 子句，将删除表中所有的数据。(特别注意)

2、delete 语句不能删除某一列的值(可使用 update)。

3、使用 delete 语句仅删除记录，不删除表本身。如要删除表，使用 drop table 语句。

4、同 insert 和 update 一样，从一个表中删除记录将引起其它表的参照完整性问题，在修改数据库数据时，头脑中应始终不要忘记这个潜在的问题。

删除的几种方法比较：

delete from 表名;

删除所有记录，表结构还在，写日志，可以恢复的，速度慢

drop table 表名;

删除表的结构和数据

delete from student where xh='A001';

删除一条记录

truncate table 表名;

删除表中的所有记录，表结构还在，不写日志，无法找回删除的记录，速度快。

设置保存点

savepoint 保存点名称;

回滚

rollback to 保存点名称;

特别注意：设置保存点及回滚操作是配合 delete 语句使用，用来找回使用 delete 删除的数据。而通过 truncate 删除的表数据是无法通过此方法找回的。

建议：

在使用 delete 删除表数据前使用 savepoint 设置保存点，防止数据误删除。

oracle 表基本查询基础

scott 用户存在的几张表**(emp,dept,salgrade)**为大家演示如何使用 select 语句，select 语句在软件编程中非常的有用，希望大家好好的掌握。

1）.EMP 员工表

EMP表

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	1980/12/17	800.00		20
7499	ALLEN	SALESMAN	7698	1981/2/20	1600.00	300.00	30
7521	WARD	SALESMAN	7698	1981/2/22	1250.00	500.00	30
7566	JONES	MANAGER	7839	1981/4/2	2975.00		20
7654	MARTIN	SALESMAN	7698	1981/9/28	1250.00	1400.00	30
7698	BLAKE	MANAGER	7839	1981/5/1	2850.00		30
7782	CLARK	MANAGER	7839	1981/6/9	2450.00		10
7788	SCOTT	ANALYST	7566	1987/4/19	3000.00		20
7839	KING	PRESIDENT		1981/11/17	5000.00		10
7844	TURNER	SALESMAN	7698	1981/9/8	1500.00	0.00	30
7876	ADAMS	CLERK	7788	1987/5/23	1100.00		20
7900	JAMES	CLERK	7698	1981/12/3	950.00		30
7902	FORD	ANALYST	7566	1981/12/3	3000.00		20
7934	MILLER	CLERK	7782	1982/1/23	1300.00		10

EMPNO：员工号 ENAME：姓名 JOB：职位 MGR：直接上级工号

HIREDATE：入职时间 SAL：工资 COMM：奖金 DEPTNO:部门编号

2）DEPT 部门表

DEPT表

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

DEPTNO：部门编号

DNAME：部门名称

LOC：部门所在城市

3）SALGRADE 工资级别表

SALGRADE表

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

geade：工资级别

losal：级别最低工资

hisal：级别最高工资

基本 select 语句

基本语法：

select [是否剔除重复数据] *|{字段名(列名),字段名 2(列名 2),字段名 3(列名 3)..} from 表名 [where {条件}];

注意事项：

1、select 指定查询哪些列的数据；

2、column 指定列名；

3、*代表查询所有列；

4、from 指定查询哪张表；

5、distinct 可选，指显示结果时，是否剔除重复数据；

6、where 条件。

简单的查询语句

1）查询所有列

SQL>select * from 表名;

2）查询指定列

SQL>select 列 1,列 2,列 3,.. from 表名;

3）如何取消重复行

SQL>select distinct deptno,job from emp;

4）查询 SMITH 的薪水，工作，所在部门

SQL>select sal,job,deptno from emp where ename='SMITH';

特别注意：

oracle 对 sql 语句不区分大小写，但对查询内容区分大小写。这与 sqlserver 是有区别的，sqlserver 对查询内容不区分大小写。

使用算数表达式

1）显示每个雇员的年工资

SQL>select ename,sal*13+nvl(comm,0)*13 from emp;

2）使用列的别名

SQL>select ename "姓名",sal*13+nvl(comm,0)*13 "年收入" from emp;

SQL>select ename 姓名,sal*13+nvl(comm,0)*13 年收入 from emp;

SQL>select ename as "姓名",sal*13+nvl(comm,0)*13 as "年收入" from emp;

特别注意：

oracle 在使用别名时，可以用双引号或不使用或使用 as 来表明别名。但不能使用单引号。sqlserver 是可以使用双引号、单引号。

如何处理 null 值

nvl 函数：oracle 提供的函数，是用于处理 null 值使用的。

例子：查询年薪

SQL>select ename,sal*13+nvl(comm,0)*13 from emp;

nvl(值 1,值 2) 解释：nvl 值 1 为 null 时则取值 2，值 1 不为 null 时则取值 1 原值。

如何连接字符串(||)

在查询的时候，希望把多列内容做为一列内容返回可以使用||连接符。

例子：查询年薪

SQL>select ename ||'年收入'||(sal*13+nvl(comm,0)*13) "雇员的年收入" from emp;

使用 where 子句

1）如何显示工资高于 3000 的员工

SQL>select ename,sal from emp where sal>3000;

2）如何查找 1982.1.1 后入职的员工

SQL>select ename,hiredate from emp where hiredate>'1-1 月-82';

也可以使用 to_char 函数转换日期类型后再进行日期比较，如下：

SQL>select ename,hiredate from emp where to_char(hiredate,'yyyy-mm-dd')>'1982-1-1';

字符对比还是有一定出入的。不推荐使用。

3）如何显示工资在 2000 到 2500 的员工情况

SQL>select * from emp where sal>=2000 and sal<=2500;

SQL>select * from emp where sal between 2000 and 2500;

说明： between 是指定区间内取值，如：between 2000 and 2500，取 2000 至 2500 内的值，同时包含 2000 和 2500

如何使用 like 操作符

%:表示任意 0 到多个字符

_:表示任意单个字符

1）如何显示首字符为 S 的员工姓名和工资

SQL>select ename,sal from emp where ename like 'S%';

2）如何显示第三个字符为大写 O 的所有员工的姓名和工资

SQL>select ename,sal from emp where ename link '__O%';

在 where 条件中使用 in

如何显示 empno 为 123,345,800...的雇员情况

SQL>select * from emp where empno=123 or empno=345 or emp=800;

SQL>select * from emp where empno in(123,345,800);

使用 is null 的操作符

如何显示没有上级的雇员的情况

SQL>select * from emp where mgr is null;

使用逻辑操作符号

查询工资高于 500 或是岗位为 manager 的雇员，同时还要满足他们的姓名首写字母为大写的 J

SQL>select * from emp where (sal>500 or job='MANAGER') and (ename like 'J%');

使用 order by 子句

1）如何按照工资的从低到高的顺序显示雇员的信息

SQL>select * from emp order by sal asc;

注意：asc 写或不写都是升序排序即从小到大排序，desc 则是降序排序从大到小排序。

2）按照部门号升序而雇员的入职时间降序排列

SQL>select * from emp order by deptno,hiredate desc;

3）使用列的别名排序

SQL>select ename,sal*12 "年薪" from emp order by "年薪" asc;

别名需要使用“”号圈中。

Oracle 分组查询

在实际应用中经常需要执行复杂的数据统计，经常需要显示多张表的数据；要用到分组函数 max, min, avg, sum, count。

Max(),min()最大最小

如何显示所有员工中最高工资和最低工资

SQL>select max(sal) "最高工资",min(sal) "最低工资" from emp;

请查询最高年工资

SQL>select max(sal*13+nvl(comm,0)*13) "最高年工资",min(sal*13+nvl(comm,0)*13) "最低年工资" from emp;

Avg()求平均

显示所有员工的平均工资和工资总和

SQL>select avg(sal) "平均工资",sum(sal) "工资总和" from emp;

特别注意：

avg(sal)不会把 sal 为 null 的行进行统计，因此我们要注意，如果，你希望为空值也考虑，则我们可以这样做

SQL>selec sum(sal)/count(*) from emp;

count(*)求总数

计算共有多少员工

SQL>select count(*) "共有员工" from emp;

练习题：

请显示工资最高的员工的名字，工作岗位

SQL>select ename,job from emp where sal=(select max(sal) from emp);

特别注意：select 语句执行的顺序是从右向左执行，正好和书写的方式相反。

SQL>select ename,job from emp where sal=(select max(sal) from emp);

oracle 会先执行 select max(sal) from emp 这个语句，得出最大工资后。再执行 where 条件前的语句。

请显示工资高于平均工资的员工信息

SQL>select * from emp where sal>(select avg(sal) from emp);

SQL>select * from emp where sal>(select sum(sal)/count(*) from emp);

group by 和 having 子句

group by：用于对查询的结果分组统计；
having 子句：用于限制(过滤)分组显示结果。

1）如何显示每个部门的平均工资和最高工资

SQL>select avg(sal) "平均工资",max(sal) "最高工资",deptno "部门编号" from emp group by deptno;

2）显示每个部门的每种岗位的平均工资和最低工资

SQL>select avg(sal) "平均工资",min(sal) "最低工资",job "职位",deptno "部门编号" from emp group by deptno,job order by deptno;

3）显示部门平均工资低于 2000 的部门号和它的平均工资

SQL>select avg(sal) "平均工资",deptno "部门编号" from emp group by deptno having avg(sal)<2000;

对数据分组的总结：

1、分组函数(avg...)只能出现在选择列表、having、order by 子句中；

2、如果在 select 语句中同时包含有 group by/having/order by 那么他们的顺序是 group by/having/order by；

3、在选择列中如果有列、表达式和分组函数，那么这些列和表达式必需有一个出现在 group by 子句中，否则会出错。

如 select deptno,avg(sal),max(sal) from emp group by deptno having avg(sal)<2000;

这里 deptno 就一定要出现在 group by 中。

Day05-Oracle 常用函数

学习目标：

- 字符函数
- 数学函数
- 日期函数
- 转换函数

- 系统函数
- 聚合函数

Oralce 函数概述

Oracle 数据库的强大，体现在对用户管理，pl/sql 编程，函数丰富。

函数的分类：单行函数，多行函数；

单行函数：对每个行输入值进行计算，得到相应的计算结果，返回给用户，也就是说，每行作为一个输入参数，经过函数计算得到每行的计算结果。**单行函数分为：**字符函数、数字函数、日期函数、转换函数、系统函数。比如 length、to_date、to_char；

多行函数：对多行输入值进行计算，得到多行对应的单个结果。比如 max,min 等分组函数；

字符函数

Ascii(掌握)

返回与指定的字符对应的十进制数；

SQL>select ascii('A') A,ascii('a') a,ascii('0') zero,ascii(' ') space from dual;

结果：

```
         A          A       ZERO      SPACE
---------- ---------- ---------- ----------
        65         97         48         32
```

chr(掌握)

给出整数，返回对应的字符；

结果：

SQL>select chr(54740) zhao,chr(65) chr65 from dual;

```
ZHAO CHR65
---- -----
赵   A
```

concat(掌握)

连接两个字符串；

SQL>select concat('hello','world') from dual;

结果：

```
CONCAT('HELLO','WORLD')
-----------------------
Helloworld
```

特别说明：concat(字串 1 或字段 1,字串 2 或字段 2)也可以书写成 字串 1 或字段 1 || 字串 2 或字段 2

initcap(掌握)

返回字符串并将字符串的第一个字母变成大写；

SQL>select initcap('smith') upp from dual;

结果：

```
UPP
-----
Smith
```

instr(C1,C2,I,J) (掌握)

在一个字符串中搜索指定的字符，返回发现指定的字符的位置；

C1 被搜索的字符串；

C2 希望搜索的字符串；

I 搜索的开始位置，默认为 1；

J 第 j 次出现的位置，默认为 1；

SQL>select instr('oracle training','ra',1,2) instring from dual;

结果：

```
   INSTRING
----------
         9
```

length(掌握)

返回字符串的长度；

SQL>select ename, length(ename) , job , length(job), sal, length(to_char(sal))
from emp where ename='SMITH';

结果：

```
ENAME LENGTH(ENAME) JOB LENGTH(JOB) SAL LENGTH(TO_CHAR(SAL))
SMITH         5         CLERK         5      800.00        3
```

特别说明：在 oracle 中单个汉字、字母、还是特殊符号都认为是长度为 1

lower(掌握)

返回字符串，并将所有的字符小写；

SQL>select lower('AaBbCcDd') "lower-AaBbCcDd" from dual;

结果：

```
lower-AaBbCcDd
--------------
aabbccdd
```

upper(掌握)

返回字符串，并将所有的字符大写；

SQL>select upper('AaBbCcDd') "upper-AaBbCcDd" from dual;

结果：

upper-AaBbCcDd

AABBCCDD

rpad 和 lpad(粘贴字符)

rpad 在列的右边粘贴字符 rpad('显示内容'或字段,显示长度,'填充占位符')

lpad 在列的左边粘贴字符 lpad('显示内容'或字段,显示长度,'填充占位符')

SQL>select lpad(rpad('htf',10,'*'),17,'*') from dual;

LPAD(RPAD('htf',10,'*'),17,'*'

*******htf*******

ltrim 和 rtrim

ltrim 删除左边出现的字符串 ltrim('原内容'或字段,'要删除的字符串')

rtrim 删除右边出现的字符串 rtrim('原内容'或字段,'要删除的字符串')

SQL>select ltrim(rtrim(' han teng fei ',' '),' ') from dual;

结果：

LTRIM(RTRIM('hantengfei',''),

---

han teng fei

SQL>select rtrim('han teng fei**','*') from dual;**

结果：

RTRIM('**HANTENGFEI**','*')

**han teng fei

SQL>select ltrim('han teng fei**','*') from dual;**

结果：

LTRIM('**HANTENGFEING**','*')

---

han teng fei**

substr(string,start,count) (掌握)

取子字符串，从 start 开始，取 count 个

SQL>select substr('13088888888',3,8) from dual;

结果：

SUBSTR('13088888888',3,8)

08888888

replace('string','s1','s2') (掌握)

string　　希望被替换的字符或变量

s1　被替换的字符串

s2　要替换的字符串

SQL>select replace('he love you','he','i') from dual;

结果：

REPLACE('HELOVEYOU','HE','I')

---

i love you

trim('s' from 'string')

如果不指定参数，默认为空格符。

SQL>select trim(0 from 0009872348900) "trim example" from dual;

结果：

trim example

98723489

例子：

- 问题：将所有员工的名字按小写的方式显示

  SQL> select lower(ename) from emp;
- 问题：将所有员工的名字按大写的方式显示。

  SQL> select upper(ename) from emp;
- 问题：显示正好为 5 个字符的员工的姓名。

  SQL> select * from emp where length(ename)=5;
- 问题：显示所有员工姓名的前三个字符。

  SQL> select substr(ename,1,3) from emp;
- 问题：以首字母大写,后面小写的方式显示所有员工的姓名。

  SQL> select upper(substr(ename,1,1)) || lower(
  substr(ename,2,length(ename)-1)) from emp;
- 问题：以首字母小写后面大写方式显示所有员工姓名。

  SQL> select lower(substr(ename,1,1)) ||
  　　upper(substr(ename,2,length(ename)-1)) from emp;
- 问题：显示所有员工的姓名，用“我是老虎”替换所有“A”

  SQL> select replace(ename,'A', '我是老虎') from emp;

案例 1：

问题：

某公司印了一批充值卡，卡的密码是随机生成的，现在出现这个问题：**卡里面的“O 和 0”（哦和零）“i 和 1”（哎和一），**用户反映说看不清楚，公司决定，把存储在数据库中的密码中所有的**“哦”都改成“零”，把所有的“i”都改成“1”；**

请编写 SQL 语句实现以上要求；数据库**表名：Card；**密码**字段名：PassWord**

分析：

1）这是更新语句，需要使用 UPDATE 语句

2）牵涉到字符串的替换，需要使用到 Oracle 中的函数 Replace

答案：

1）UPDATE Card SET PassWord = Replace(PassWord ,'O','0')

UPDATE Card SET PassWord = Replace(PassWord ,'i','1')

2）UPDATE Card SET PassWord = Replace(Replace(PassWord ,'O','0'),'i','1')

案例 2：

问题：

在数据库表中有以下字符数据，如：

13-1、13-2、13-3、13-10、13-100、13-108、13-18、13-11、13-15、14-1、14-2

现在希望通过SQL语句进行排序，并且首先要按照前半部分的数字进行排序，然后再按照后半部分的数字进行排需，输出要排成这样：

13-1、13-2、13-3、13-10、13-11、13-15、13-18、13-100、13-108、14-1、14-2

请编写 SQL 语句实现以上要求；数据库**表名：SellRecord；字段名：ListNumber**

分析：

1）查询语句：使用 SELECT 语句

2）排序：ORDER BY：在 ORDER BY 的排序列中，需要重新计算出排序的数字

3）前半部分的数字：

找到“-”符号的位置

取其左半部分

使用 Convert 函数将其转换为数字：

to_number(substr(list_number,1,instr(list_number,'-')-1))

4）后半部分的数字：

找到“-”符号的位置

把从第一个位置到该位置的全部字符替换为空格

使用 Convert 函数将其转换为数字：

to_number(substr(list_number,instr(list_number,'-')+1))

答案：

```
SELECT ListNumber
FROM   SellRecord
ORDER BY  to_number(substr(list_number,1,instr(list_number,'-')-1)),
          to_number(substr(list_number,instr(list_number,'-')+1))
```

数学函数

ceil（向上取整）(掌握)

返回大于或等于给出数字的最小整数；

SQL>select ceil(3.14159265) from dual;

结果：

```
CEIL(3.14159265)
----------------
               4
```

floor（向下取整）(掌握)

对给定的数字取整数；

SQL>select floor(2345.67) from dual;

结果：

```
FLOOR(2345.67)
--------------
          2345
```

trunc(掌握)

按照指定的精度截取一个数；

SQL>select trunc(124.1666,-2),trunc(124.16666,2) from dual;

结果：

```
TRUNC(124.1666,-2) TRUNC(124.16666,2)
---------- ----------
               100             124.16
```

round 和 trunc(掌握)

按照指定的精度进行舍入；

round 函数为四舍五入

trunc（直接截取）

SQL>select round(55.5),round(-55.4),trunc(55.5),trunc(-55.5) from dual;

结果：

```
ROUND(55.5) ROUND(-55.4) TRUNC(55.5) TRUNC(-55.5)
----------- ------------ ----------- ------------
         56          -55          55          -55
```

abs

返回指定值的绝对值；

SQL>select abs(100),abs(-100) from dual;

结果：

```
   ABS(100)    ABS(-100)
---------- ----------
        100          100
```

以下为了解内容（知道有）

acos

给出反余弦的值；

```
SQL>select acos(-1) from dual;
结果：
   ACOS(-1)
----------
3.14159265
```

asin

给出反正弦的值；

```
SQL>select asin(0.5) from dual;
  ASIN(0.5)
----------
0.52359877
```

atan

返回一个数字的反正切值；

```
SQL>select atan(1) from dual;
     ATAN(1)
----------
0.78539816
```

cos

```
返回一个给定数字的余弦；
SQL>select cos(-3.14159265) from dual;
COS(-3.14159265)
----------------
                -1
```

cosh

```
返回一个数字反余弦值；
SQL>select cosh(20) from dual;
   COSH(20)
----------
242582597.
```

exp

```
返回一个数字 e 的 n 次方根；
SQL>select exp(2),exp(1) from dual;
      EXP(2)         EXP(1)
---------- ----------
7.38905609 2.71828182
```

ln

```
返回一个数字的对数值；
SQL>select ln(1),ln(2),ln(2.7182818) from dual;
        LN(1)          LN(2) LN(2.7182818)
---------- ---------- -------------
         0 0.69314718 0.99999998953
```

log(n1,n2)

```
返回一个以 n1 为底 n2 的对数；
SQL>select log(2,1),log(2,4) from dual;
   LOG(2,1)     LOG(2,4)
---------- ----------
         0              2
```

mod(n1,n2)

```
返回一个 n1 除以 n2 的余数；(取模函数)
SQL>select mod(10,3),mod(3,3),mod(2,3) from dual;
 MOD(10,3)     MOD(3,3)     MOD(2,3)
```

```
---------- ---------- ----------
         1          0          2
```

power

```
返回 n1 的 n2 次方根；
SQL>select power(2,10),power(3,3) from dual;
POWER(2,10) POWER(3,3)
----------- ----------
       1024         27
```

sign

```
取数字 n 的符号，大于 0 返回 1，小于 0 返回-1，等于 0 返回 0；
SQL>select sign(123),sign(-100),sign(0) from dual;
 SIGN(123) SIGN(-100)       SIGN(0)
---------- ---------- ----------
         1         -1          0
```

sin

```
返回一个数字的正弦值；
SQL>select sin(1.57079) from dual;
SIN(1.57079)
------------
0.9999999999
```

sinh

```
返回双曲正弦的值；
SQL>select sin(20),sinh(20) from dual;
    SIN(20)     SINH(20)
---------- ----------
0.91294525 242582597.
```

sqrt

```
返回数字 n 的根；
SQL>select sqrt(64),sqrt(10) from dual;
   SQRT(64)     SQRT(10)
---------- ----------
         8 3.16227766
```

tan

返回数字的正切值；

```
SQL>select tan(20),tan(10) from dual;
     TAN(20)       TAN(10)
---------- ----------
2.23716094 0.64836082
```

tanh

返回数字的 n 的双曲正切值；

```
SQL>select tanh(20),tan(20) from dual;
   TANH(20)       TAN(20)
---------- ----------
                1 2.23716094
```

日期函数

介绍 :日期函数用于处理 date 类型的数据。
默认情况下日期格式是 dd-mon-yy 即 12-7 月-78

add_months(掌握)

add_months(日期值,增加(减少)值)
增加或减去月份；

SQL>select to_char(add_months(to_date('199912','yyyymm'),2),'yyyymm') from dual;

结果：

```
TO_CHAR(ADD_MONTHS(TO_DATE('19
------------------------------
200002
```

SQL>select hiredate,add_months(hiredate,2) from emp where ename='SMITH';

结果：

```
HIREDATE       ADD_MONTHS(HIREDATE,2)
---------- ---------------------
1980/12/17    1981/2/17
```

请查找最近 350 个月入职的员工

SQL>select ename,hiredate from emp where add_months(hiredate,350)>=sysdate;

结果：

ENAME HIREDATE

---------- -----------

SCOTT 1987/4/19

ADAMS 1987/5/23

last_day(掌握)

返回日期的最后一天；

SQL>select to_char(sysdate,'yyyy-mm-dd'),to_char((sysdate)+1,'yyyy-mm-dd') from dual;

结果：

TO_CHAR(SYSDATE,'YYYY-MM-DD') TO_CHAR((SYSDATE)+1,'YYYY-MM-DD

结果：

---------- ----------

2017-03-01 2017-03-02

SQL>select to_char(last_day(sysdate),'yyyy-mm-dd') from dual;

结果：

TO_CHAR(LAST_DAY(SYSDATE),'YYYY-MM-DD

2017-03-31

months_between(date2,date1) (掌握)

给出 date2-date1 的月份，共有多少个月；

例子：

SQL>select months_between('19-12 月-1999','19-3 月-1999') mon_between from dual;

结果：

MON_BETWEEN

9

SQL>select months_between(to_date('2000-05-20','yyyy-mm-dd'),to_date('2005-05-20','yyyy-mm-dd')) mon_betw from dual;

结果：

MON_BETW

-60

next_day(date,'day') (掌握)

给出日期 date 和星期 X（day）之后计算下一个星期的日期；

SQL>select next_day('1-3 月-2017','星期五') next_day from dual;

结果：

NEXT_DAY

2017/3/3

sysdate(掌握)

用来得到系统的当前日期;

SQL>select to_char(sysdate,'day') from dual;

结果:

TO_CHAR(SYSDATE,'DAY')

星期四

例子:

- 问题:查找已经入职 8 个月多的员工

```
SQL> select * from emp where sysdate>=add_months(hiredate,8);
```

- 问题:显示满 10 年服务年限的员工的姓名和受雇日期。

```
SQL> select ename, hiredate from emp
      where sysdate>=add_months(hiredate,12*10);
```

- **问题:对于每个员工,显示其加入公司的天数。 (掌握)**

```
SQL> select floor(sysdate-hiredate) "入职天数",ename from emp;
SQL> select trunc(sysdate-hiredate) "入职天数",ename from emp;
```

- 问题:找出各月倒数第 3 天受雇的所有员工。

```
SQL> select hiredate,ename from emp
      where last_day(hiredate)-2=hiredate;
```

转换类型函数

to_char(date,'format') (掌握)

日期类型转换成字符串格式(主要用于将日期以习惯的格式输出显示)

SQL>select to_char(sysdate,'yyyy/mm/dd hh24:mi:ss') from dual;

结果:

TO_CHAR(SYSDATE,'YYYY/MM/DDHH2

2014/04/24 16:19:34

to_char(掌握)

可以使用 select ename,hiredate,sal from emp where deptno=10;

显示信息,可是在某些情况下,这个并不能满足你的需求。

日期是否可以显示时/分/秒?

select to_char(hiredate,'yyyy-mm-dd hh24:mi:ss') from dual;

薪水是否可以显示指定的货币符号？

select to_char(sal,'$9999.99') from emp;

特别说明：

日期格式：

yy:两位数字的年份 2004--04

yyyy:四位数字的年份 2004 年

mm:两位数字的月份 8 月--08

dd:两位数字的天数 30 号--30

hh24:二十四小时制 8 点--20

hh12:十二小时制 8 点--08

mi,ss--显示分钟\秒

day 显示星期几

month 显示几月

year 显示年

数字格式：

9:显示数字，并忽略前面 0

0:显示数字，如位数不足，则用 0 补齐

.:在指定位置显示小数点

,:在指定位置显示逗号

$:在数字前加美元符号

L:在数字前加本地货币符号

C:在数字前加国际货币符号

G:在指定位置显示组分隔符

D:在指定位置显示小数点符号(.)

说明：,逗号.和小数点可以合在一起使用，G 分隔符和 D 小数点符可以合在一起使用，但,.不能和 GD 综合使用，否则报错。

to_date(string,'format')

将字符串转换成日期(主要用于将日期按习惯的格式输入到 oracle 数据库中)

to_number(掌握)

将给出的数字类型的字符转换为数字；

SQL>select to_number('1999') year from dual;

结果：

```
      YEAR
----------
      1999
```

例子：

- 问题：显示薪水的时候，把本地货币单位加在前面

```
SQL> select ename, to_char(hiredate, 'yyyy-mm-dd hh24:mi:ss'),
to_char(sal,'L99999.99') from emp;
```

- 问题：显示 1980 年入职的所有员工

```
SQL> select * from emp where to_char(hiredate, 'yyyy')=1980;
```

- 问题：显示所有 12 月份入职的员工

```
SQL> select * from emp where to_char(hiredate, 'mm')=12;
```

系统函数

decode 函数类似于 java 的 switch case 分支语句

```
SQL>select ename||decode(deptno,
10,'在 10 号部门',
20,'在 20 号部门',
30,'在 30 号部门')
from emp where ename='SCOTT' order by deptno;
结果：
ENAME||DECODE(DEPTNO,10,'在 10?
------------------------------
SCOTT 在 20 号部门
```

聚合函数

聚合函数：一组值进行计算，并返回计算后的值 ，具有统计数据的作用(必须掌握)

avg(distinct|all)

all 表示对所有的值求平均值,distinct 只对不同的值求平均值

```
SQL>select chr(54740) zhao,chr(65) chr65 from dual;
AVG(DISTINCTSAL)
----------------
2064.58333333333

SQL> select avg(all sal) from emp;
AVG(ALLSAL)
-----------
2073.214285
```

max(distinct|all)

求最大值,ALL 表示对所有的值求最大值,DISTINCT 表示对不同的值求最大值,相同的只取一次

```
SQL>select max(distinct sal) from emp;
```

```
MAX(DISTINCTSAL)
----------------
            5000
```

min(distinct|all)

求最小值,ALL 表示对所有的值求最小值,DISTINCT 表示对不同的值求最小值,相同的只取一次

```
SQL>select min(all sal) from emp;
MIN(ALLSAL)
-----------
        800
```

COUNT(DISTINCT|ALL)

求记录、数据个数。 ALL 对所有记录，数组做统计， DISTINCT 只对不同值统计(相同值只取一次)

```
SELECT COUNT(SAL) FROM EMP;
COUNT(SAL)
----------
        14
SELECT COUNT(DISTINCT SAL) FROM EMP;
COUNT(DISTINCTSAL)
------------------
                12
```

/*/**

分组查询

group by

主要用来对一组数进行统计

```
SQL> select deptno,count(*),sum(sal) from emp group by deptno;
DEPTNO   COUNT(*)   SUM(SAL)
------ ---------- ----------
    30          6       9400
    20          5      10875
    10          3       8750
```

分组后过滤

having

对分组统计再加限制条件

```
SQL> select deptno,count(*),sum(sal) from emp group by deptno having
count(*)>=5;
DEPTNO   COUNT(*)   SUM(SAL)
------ ---------- ----------
    30          6       9400
20          5      10875
```

```
SQL> select deptno,count(*),sum(sal) from emp group by
 deptno having count(*)>=5;
DEPTNO   COUNT(*)   SUM(SAL)
------ ---------- ----------
    30          6       9400
    20          5      10875
```

Oracle06-复杂查询

多表查询

多表查询：是指基于两个和两个以上的表或是视图的查询，在实际应用中，查询单个表可能不能满足你的需求，(如显示 sales 部门位置和其员工的姓名)，这种情况下需要使用到(dept 表和 emp 表)

例如：显示雇员名，雇员工资及所在的部门的名字[笛卡尔集]

```
SQL>select e.ename,e.sal,d.dname from emp e,dept d where d.deptno=e.deptno;
```

注意：笛卡尔集，在多表查询的时候，如果不带任何条件，则会出现笛卡尔集，避免笛卡尔集多表查询的条件是，至少不能少于表的个数-1

规定：多表查询的条件是至少不能少于表的个数-1

- 如何显示部门为 10 的部门名、员工名和工资

```
SQL>select d.dname,e.ename,e.sal,e.deptno from emp e,dept d where d.deptno=e.deptno and e.deptno=10;
```

- 显示各个员工的姓名、工资及其工资的级别

```
SQL>select e.ename,e.sal,s.grade from emp e,salgrade s where e.sal between s.losal and s.hisal;
```

注意：在多表查询时，不同的表中列名相同时要加表名，不同时可不加。(为增强可读性，建议都加表名或别名)

- 显示雇员名，雇员工资及所在部门的名字，并按部门排序。

```
SQL>select e.ename,e.sal,d.dname from emp e,dept d where
e.deptno=d.deptno order by d.dname;
```

自连接

自连接：是指在同一张表的连接查询(把一张表看作两张表)

- 显示员工的上级领导的姓名

```
SQL>select e2.ename from emp e1,emp e2 where e1.mgr=e2.empno;
```

- 比如显示'FORD'的上级

```
SQL>select e1.ename "员工姓名",e2.ename "领导姓名" from emp e1,emp e2 where e1.mgr=e2.empno and
e1.ename='FORD';
```

1） 限制固定行数(rownum 字段限制固定行数

- 如何查询前五名员工的姓名和工资？

```
select    ename, sal , rownum from emp where rownum
 <=5;
```

2） 按照多列进行排序

- 如何查询员工的姓名和工资，并按照部门和岗位进行降序排列？

```
select ename, sal, deptno, job from emp order by deptno, job desc;
```

- 如何查询前五名员工的姓名和工资？

```
select    ename, sal , rownum from emp where rownum<=5;
```

扩展要求：

- 显示各员工的姓名和他的上级领导姓名

```
SQL> select e1.ename "员工姓名",e2.ename "领导姓名"
    from emp e1,emp e2
    where e1.mgr=e2.empno;
```

疑惑：这里我们看到 king 没有显示，因为 king 没有上级。如果我们希望把没有上级的员工也显示出来，则需要使用到外连接。外连接包括左外连接和右外连接。此处提到外连接，后面会详细讲解。

左外连接：

1）select 列名,.. from 表名 1 left join 表名 2 on 条件;

SQL>select e1.ename "员工姓名",e2.ename "直接上级领导" from emp e1 left join emp e2 on e1.mgr=e2.empno;

2）使用(+)在右边也可以实现左外连接。

SQL>select e1.ename "员工姓名",e2.ename "领导姓名" from emp e1,emp e2 where e1.mgr=e2.empno(+);

右外连接：

1)select 列名,.. from 表名 1 right join 表名 2 on 条件;

```
select e1.ename "员工姓名",e2.ename "直接上级领导"
from emp e2 right
join emp e1
on e1.mgr=e2.empno;
```

2）使用(+)在左边也可以实现右外连接。

```
select e1.ename "员工姓名",e2.ename "领导姓名"
from emp e1,emp e2
where e2.empno(+)=e1.mgr;
```

左外连接和右外连接在这里提到，后面会详细讲解。

子查询

子查询：是指嵌入在其它 sql 语句中的 select 语句，也叫嵌套查询；分为**单行子查询、多行子查询、多列子查询；**

1） 单行子查询是指只返回一行数据的子查询语句。

请思考：如果显示与 smith 同一部门的所有员工？

```
SQL>select * from emp
      where deptno=(select deptno from emp where ename='SMITH');
```

将 SMITH 排除在外不显示

```
SQL>select * from emp
      where deptno=(select deptno from emp where ename='SMITH') and ename <> 'SMITH';
SQL>select * from emp
      where deptno=(select deptno from emp where ename='SMITH') and ename != 'SMITH';
```

2） 多行子查询指返回多行**数据的子查询。**

请思考：**如何查询和部门 10 的工作相同的雇员的名字、岗位、工资、部门号**

```
SQL>select ename,job,sal,deptno from emp
       where job in (select distinct job from emp where deptno=10);
```

特别注意：多行子查询是不能使用=号的，=号是单行子查询(由于只返回一个结果所以使用=号)，多行子查询返回的不是一个结果所以要使用 in。

a)行子查询中使用 all 操作符

请思考：如何显示工资比部门 30 的所有员工的工资高的员工的姓名、工资和部门号

```
SQL>select ename,sal,deptno from emp where sal>all(select sal from emp where deptno=30);
```

扩展要求：大家想想还有没有别的查询方法？

```
SQL>select ename,sal,deptno from emp where sal>(select max(sal) from emp where deptno=30);
```

b)行子查询中使用 any 操作符

请思考：如何显示工资比部门 30 的任意一个员工的工资高(只要比 30 号部门中任意的一个工资高就满足条件)的员工的姓名、工资和部门号

```
SQL>select ename,sal,deptno from emp
      where sal>any(select sal from emp where deptno=30);
```

扩展要求：大家想想还有没有别的查询方法？

```
SQL>select ename,sal,deptno from emp where sal>(select min(sal) from emp where deptno=30);
```

3） 多列子查询

单行子查询是指子查询只返回单列、单行数据；

多行子查询是指返回单列多行数据，都是针对单列而言的；

多列子查询：是指查询返回多个列数据的子查询语句。

请思考：如何查询与 smith 的部门和岗位完全相同的所有雇员？

- **单表多列查询：**

SQL>select * from emp where deptno=(select deptno from emp where ename='SMITH') and job=(select job from emp where ename='SMITH');

- **多列查询(优化方法，在 oracle 下推荐使用)**

SQL>select * from emp where (deptno,job)=(select deptno,job from emp where ename='SMITH');

特别注意：查询的列要与返回的列名相对应，顺序不能出错。否则报错。

4） from 子句中使用子查询

当在 from 子句中使用子查询时，该子查询会被作为一个临时表来对待，而且必需给予查询指定别名。

请思考：如何显示高于自己部门平均工资的员工的信息

这里要用到数据查询的小技巧，把一个子查询当作一个临时表使用。

```
SQL>select e.ename, e.sal, t1.myavg, e.deptno
      from emp e, (select avg(sal) myavg, deptno from emp group by deptno) t1
      where e.deptno = t1.deptno and e.sal > t1.myavg
      order by e.deptno;
```

下面方法不推荐使用(无法取出子查询(临时表)的值)

SQL>select e.* from emp e where e.sal>(select avg(sal) from emp where deptno=e.deptno);

请思考：查找每个部门工资最高的人的详细资料

思路：得到所有的员工，进行筛选，每拿到一个员工，判断该员工的工资是否是他们部门的最高工资。

SQL>select e.*,t1.mysal from emp e,(select max(sal) mysal,deptno from emp group by deptno) t1 where e.deptno=t1.deptno and e.sal>=t1.mysal order by e.deptno;

SQL>select * from emp where sal in (select max(sal) from emp group by deptno) order by deptno;

5） 用查询结果创建新表（非常好用，希望大家记住）

基本语法：

create table 新建表名 (列名 1,列名 2,列名 3,..) as select 列名 1,列名 2,列名 3,.. from 已有表名;

说明：

1、新建表名中的列名要与 select 中的列名一一对应，否则会报错；

2、as 关键字不可少；

Sql>create table t1 (id ,name) as select empno,ename from emp;

自我复制数据(蠕虫复制)

基本语法：**insert into 表名 (列名 1,列名 2,列名 3,..) select 列名 1,列名 2,列名 3,.. from 表名;**

sql>insert into t2(empno,ename) select empno,ename from emp;

说明：表名后的列名要与 select 后的列名一致，否则会报错。

合并查询（掌握）

有时在实际应用中，为了合并多个 select 语句的结果，可以使用集合操作符 union,union all,intersect,minus

3） union 取并集

该操作符用于取得两个结果集的并集。当使用该操作符时，会自动去掉结果集中重复行。

select ename,sal,job from emp where sal>2500 **union**

select ename,sal,job from emp where job='MANAGER';

解释：union 并集，将多条查询结果进行合并，去除相同重复的查询结果。

2）union all 取所有

该操作符与 union 相似，但是它不会取消重复行，而且不会排序。

select ename,sal,job from emp where sal>2500 **union all**

select ename,sal,job from emp where job='MANAGER';

解释：union all 的用法与 union 相似，但 union all 不会对多条查询结果进行合并、排序。只会将查询到的所有内容都显示出来，而不象 union 对查询结果进行合并过滤。

3）intersect 取交集

使用该操作符用于取得两个结果集的交集。

select ename,sal,job from emp where sal>2500 **intersect**

select ename,sal,job from emp where job='MANAGER';

解释：intersect 是将多条查询结果，重复部分提取并显示出来。

4）minus 取差集

使用该操作符用于取得两个结果集的差集，它只会显示存在第一个集合中，而不存在第二个集合中的数据。

select ename,sal,job from emp where sal>2500 minus select ename,sal,job from emp where job='MANAGER';

解释：minus 取差集，是将前一条查询结果与第二条查询结果进行比较，去除满足条件 2，所得到的结果。

Oracle 表之间的连接

Oracle 表之间的连接分为三种：**自连接**（同一张表内的连接）；**内连接**(自然连接)；外连接；外连接有分为三种：**左外连接** (左边的表不加限制)，**右外连接**(右边的表不加限制)，**全外连接**(左右两表都不加限制)；

内连接(inner join - on)

内连接:是利用 where 子句对两张表形成的笛卡尔集进行筛选，我们前面学习的查询都是内连接，也是开发过程中用的最多的连接查询，又叫自然连接（nature join）。

内连接基本语法：

select 列名 1,.. from 表 1 inner join 表 2 on 条件;

说明：内连接只有两张表同时满足条件才会被查询到。

举例：显示员工的信息和部门名称

select e.*,d.dname from emp e,dept d where e.deptno=d.deptno;

等价于

select e.*,d.dname from emp e inner join dept d on e.deptno=d.deptno;

外连接

外连接分为三种：左外连接、右外连接、完全外连

1、**左外连接**（left join - on)(如果左侧的表完全显示我们就说是左外连接)

基本语法：

select 列名 1,列名 2,.. from 表 1 left join 表 2 on 条件;

或者

select 列名 1,列名 2,.. from 表 1,表 2 where 条件 1=条件 2(+);

2、**右外连接**（right join - on)(如果右侧的表完全显示我们就说是右外连接)

基本语法：

select 列名 1,列名 2,.. from 表 1 right join 表 2 on 条件;

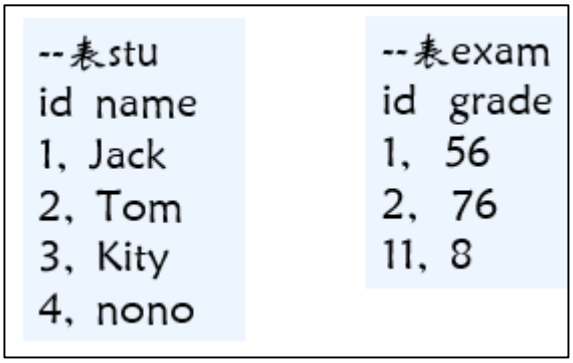
或者

select 列名 1,列名 2,.. from 表 1,表 2 where 条件 1(+)=条件 2;

3、**完全外连**（full outer join - on)(完全显示两个表，没有匹配的记录置为空)

基本语法：

select 列名 1,列 2,.. from 表 1 full outer join 表 2 on 条件;



创建两张表做测试使用 stu/exam 表

```
create table stu (id number,name varchar2(32));
        insert into stu values(1,'jack');
        insert into stu values(2,'tom');
        insert into stu values(3,'kity');
        insert into stu values(4,'nono');
```

	ID	NAME
1	1	jack
2	2	tom
3	3	kity
4	4	nono

```
 create table exam (id number,grade number);
        insert into exam values(1,56);
        insert into exam values(2,76);
        insert into exam values(11,8);
```

	ID	GRADE
1	1	56
2	2	76
3	11	8

外连接

1、内连接案例(显示两表 ID 匹配的)

基本语法：select 字段 1,字段 2... 表名 1 inner join 表名 2 on 条件;

```
SQL>select s.*,e.* from stu s inner join exam e on s.id=e.id;
```

结果：

	ID	NAME	ID	GRADE
1	1	jack	1	56
2	2	tom	2	76

2、**左连接**(显示所有人的成绩，如果没有成绩，也要显示该人的姓名和 id 号，成绩显示为空)

SQL>select s.id,s.name,e.grade from stu s left join exam e on s.id=e.id;

SQL>select s.id,s.name,e.grade from stu s,exam e where s.id=e.id(+);

显示结果：

	ID	NAME	GRADE
1	1	jack	56
2	2	tom	76
3	4	nono	
4	3	kity	

3、**右外连接**(显示所有成绩，如果没有名字匹配，显示空)

SQL>select s.id,s.name,e.grade from stu s right join exam e on s.id=e.id;

SQL>select s.id,s.name,e.grade from stu s,exam e where s.id(+)=e.id;

显示结果：

	ID	NAME	GRADE
1	1	jack	56
2	2	tom	76
3			8

4、**完全外连接**(显示所有成绩和所有人的名字，如果没有相应的匹配值，则显示空)

SQL>select s.id,s.name,e.grade from stu s full outer join exam e on s.id=e.id;

显示结果：

	ID	NAME	GRADE
1	1	jack	56
2	2	tom	76
3	3	kity	
4	4	nono	
5			8

oracle 表内连接和外连接

一个小练习

为加深大家对外连接的理解，我们做一个小练习

列出部门名称和这些部门的员工信息，同时列出那些没有员工的部门。

SQL>select d.dname,e.* from dept d left join emp e on d.deptno=e.deptno;

SQL>select d.dname,e.* from dept d,emp e where d.deptno=e.deptno(+);

SQL>select d.dname,e.* from emp e right join dept d on d.deptno=e.deptno;

SQL>select d.dname,e.* from emp e,dept d where e.deptno(+)=d.deptno;

练习题 3：

1、假定 sales 部门有 10 个雇员，其中 9 个雇员的工资为 1000，另一个雇员的工资为 null，那么使用 avg 函数取得该部门的平均工资时，结果应该是多少？ A

A、1000 B、900

2、以下哪条语句是正确的？C

A、select deptno,sum(sal) from emp;

B、select deptno,avg(sal) from emp where avg(sal)>2000 group by deptno;

C、select deptno,avg(sal) from emp group by deptno having avg(sal)>2000;

3、如果要显示所有的部门及其雇员信息，应该使用哪个语句？B

A、select a.dname,b.ename from dept a,emp b where a.deptno=b.deptno;

B、select a.dname,b.ename from dept a,emp b where a.deptno=b.deptno(+);

C、select a.dname,b.ename from dept a,emp b where a.deptno(+)=b.deptno;

5、以下哪个集合操作符不会执行排序操作？B

A、union B、union all C、intersect D、minus

6、使用分组函数和数据分组子句

(1)显示所有雇员的平均工资、总计工资、最高工资、最低工资。

SQL>select avg(sal),sum(sal),max(sal),min(sal) from emp;

(2)显示每种岗位的雇员总数、平均工资。

SQL>select count(*),avg(sal),job from emp group by job;

(3)显示雇员总数，以及获得补助的雇员数。

SQL>select count(*),count(comm) from emp;

(4)显示管理者的总人数

SQL>select count(distinct mgr) from emp;

(5)显示雇员工资的最大差额。

SQL>select max(sal)-min(sal) from emp;

(6)显示每个部门每个岗位的平均工资、每个部门的平均工资、每个岗位的平均工资。

SQL>select avg(sal),deptno,job from emp group by cube(deptno,job);

特别注意：cube 立方体函数的使用，此函数用于分组统计。

7、使用连接查询完成

(1)显示部门 20 的部门名，以及该部门的所有雇员名、雇员工资及岗位

SQL>select e.deptno,d.dname,e.ename,e.sal,e.job from emp e,dept d where e.deptno=d.deptno and e.deptno=20;

(2)显示获得补助的所有雇员名、补助以及所在部门名

SQL>select e.ename,e.comm,d.dname from emp e,dept d where e.deptno=d.deptno and e.comm is not null;

(3)显示在 dallas 工作的所有雇员名、雇员工资及所在部门名

SQL>select e.ename,e.sal,d.dname from emp e,dept d where e.deptno=d.deptno and d.loc='DALLAS';

(4)显示雇员 scott 的管理者名称

SQL>select ename from emp where empno=(select mgr from emp where ename='SCOTT');

(5)查询 emp 表和 salgrade 表显示部门 20 的雇员名、工资及其工资级别

SQL>select e.ename,e.sal,s.grade from emp e,salgrade s where e.deptno=20 and e.sal between losal and hisal;

(6)显示部门 10 的所有雇员名、部门名以及其他部门名

SQL>select e.ename,d.dname from emp e right join dept d on e.deptno=d.deptno and e.deptno=10;

(7)显示部门 10 的所有雇员名、部门名以及其他雇员名

SQL>select e.ename,d.dname from emp e left join dept d on e.deptno=d.deptno and e.deptno=10;

(8)显示部门 10 的所有雇员名、部门名以及其他部门名和雇员名

SQL>select e.ename,d.dname from emp e full outer join dept d on e.deptno=d.deptno and e.deptno=10;

8、使用子查询完成

(1)显示 blake 同部门的所有雇员，但不显示 blake

SQL>select * from emp where deptno=(select deptno from emp where ename='BLAKE') and ename!='BLAKE';

(2)显示超过平均工资的所有雇员名，工资及其部门号

SQL>select ename,sal,deptno from emp where sal>(select avg(sal) from emp);

(3)显示超过部门平均工资的所有雇员名、工资及部门号

SQL>select e.ename,e.sal,e.deptno,t1.mysal from emp e,(select avg(sal) mysal,deptno from emp group by deptno) t1 where e.deptno=t1.deptno and e.sal>t1.mysal order by e.deptno;

(4)显示高于 clerk 岗位所有雇员工资的所有雇员名、工资及岗位

SQL>select ename,sal,job from emp where sal>(select sum(sal) from emp where job='CLERK');

(5)显示工资、补助与 scott 完全一致的所有雇员名、工资及补助

SQL>select e.ename,e.sal,nvl(e.comm,0) ecomm,e.deptno from emp e,(select sal,nvl(comm,0) tcomm from emp where ename='SCOTT') t1 where e.sal=t1.sal and nvl(e.comm,0)=t1.tcomm;

Day08-Oracle 分页查询和视图

数据库中的两个伪列：rownum 与 rowid：

Rownum：顾名思义就是行数/行号；

Rowid：就是编码/编号/唯一识别号，所以他是类似“AAAR8gAAEAAAAErAAK”的编号，注意他是没有先后顺序的，也就是说他和数据入库时间没有任何关系，打个比方：他就像磁盘、内存存储数据用的是 16 进制的地址一样。

他们都是伪列，可以理解成表中的一个列只是他们并不是你创建的。同样是伪列区别是什么呢？

Rowid

Rowid：是你录入数据时有数据库自动为这条记录添加的唯一的 18 位编号是一个物理编号用于找到这条记录（顺便说一句这也是为什么数据优调的时候强调尽量使用 rowid 的原因），他是不会随着查询而改变的 除非在表发生移动(比如表空间变化，数据导入/导出以后)，才会发生变化。

Rowid 特点：

1）首先是一种数据类型，唯一标识一条记录物理位置的一个 id，
基于 64 位编码的 18 个字符显示。

2）未存储在表中，可以从表中查询，但不支持插入，更新，删除它们的值。

Rowid 的组成

rowid 确定了每条记录是在 Oracle 中的哪一个数据对象，数据文件、块、行上。

ROWID 的格式如下：

数据对象编号	文件编号	块编号	行编号
OOOOOO	FFF	BBBBBB	RRR

Rownum

Rownum：是根据 sql 查询后得到的结果自动加上去的，但是他却不受到 sql 中 order by 排序的影响，因为他和 rowid 的顺序一样是系统按照记录插入时的顺序给记录排的号（顺序的、无跳跃）。

如果你想让 rownum 和 order by 一样的顺序 那么可以使用子查询，

例如：

select rownum,t.* from (select * from 表空间名 order by 字段名) t

这样的话 rownum 就是根据该字段进行排序的编号了，为什么会这样呢，

rownum 特点：

1. rownum 从 1 开始；

2.rownum 按照记录插入时的顺序给记录排序，

例如：id 主键是按照从小到大的顺序插入的，select 语句没有 group by 和 order by 的子句时，rownum 的顺序和 id 顺序基本一致。

3.对于 Oracle 的 rownum 来说，不支持 >,>=,=,between...and，只能用以上符号(<、<=、!=)，并非说用>,& amp; gt;=,=,between..and 时会提示 SQL 语法错误，而是经常是查不出一条记录来

例如：假设某个表 t1(c1) 有 20 条记录

如果用 select rownum,c1 from t1 where rownum < 10, 只要是用小于号，查出来的结果很容易地与一般理解在概念上能达成一致，应该不会有任何疑问的。

可如果用 select rownum,c1 from t1 where rownum > 10 (如果写下这样的查询语句，这时候在您的头脑中应该是想得到表中后面 10 条记录)，你就会发现，显示出来的结果要让您失望了，也许您还会怀疑是不谁删了一 些记录，然后查看记录数，仍然是 20 条啊？那问题是出在哪呢？

先 好好理解 rownum 的意义吧。因为 ROWNUM 是对结果集加的一个伪列，即先查到结果集之后再加上去的一个列 (强调：先要有结果集)。简单的说 rownum 是对符合条件结果的序列号。它总是从 1 开始排起的。所以你选出的结果不可能没有 1，而有其他大于 1 的值。所以您没办法期望得到下面的结果集：

11 aaaaaaaa

12 bbbbbbb

13 ccccccc

.....

rownum >10 没有记录，因为第一条不满足去掉的话，第二条的 ROWNUM 又成了 1，所以永远没有满足条件的记录。或者可以这样理解：

ROWNUM 是一个序列，是 oracle 数据库从数据文件或缓冲区中读取数据的顺序。它取得第一条记录则 rownum 值为 1，第二条为 2，依次类推。如果你 用>,>=,=,between...and 这些条件，因为从缓冲区或数据文件中得到的第一条记录的 rownum 为 1，则被删除，接着取下条，可是它的 rownum 还是 1，又被删除，依次类推，便没有了数据。

Oracle 分页查询

分页查询是我们学习任何一种数据库，必需掌握的一个要点。

mysql 数据库的分页查询说明：

select * from 表名 where 条件 limit 从第几条取,取几条;

oracle 数据库的分页查询说明：

oracle 每张表都有 rownum 默认字段，默认情况下是不显示的。但是是一直存在的。

分页查询语法模版：

```
select t2.*
from ( select t1.*, rownum rn
        from (select *
                from 表名) t1
       where rownum<=大范围(取到多少条数据)) t2
where rn>=小范围(从第几条数据开始取);
```

特别说明：

oracle 分页查询是通过三层筛选法进行查询的。每一次都可以带 where 条件来对要查询的信息进行筛选。

建议：

第一层：构建我们所要查询字段信息并排序；

第二层：构建 rownum 别名 rn

第三层：加 where 条件，rn>=M and rn <=N

例如：

按照入职时间的先后顺序，查询从第 7 至第 10 个人是谁？

分析：1）查询出 emp 中的所有信息，并按照 hiredate 字段排序；

```
select *  from emp order by hiredate;
```

2）在第一步查询的结果中添加 rownum 字段并起别名 rn

```
select rownum rn, t1.*
from (select *  from emp order by hiredate) t1
```

3）在第二步基础上查询所有并添加 where 过滤条件

```
select *
from  (select rownum rn, t1.*
       from (select *
             from emp
```

```
                                        order by hiredate) t1) t2
                        where rn>=7 and rn<=10;
或者：如下
SQL>select t2.ename,t2.hiredate,t2.rn
      from (select t1.*,rownum rn
                from (select *
                        from emp order by hiredate) t1
                where rownum<=10) t2
      where rn>=7;
SQL>select t2.*
      from (select t1.*,rownum rn
                from (select *
                      from emp) t1
                where rownum<=10) t2
where rn>=5;

看得更清楚一点
select t2.* from
(select t1.*,rownum rn from
       (select * from emp)
t1)
t2 where rn>=5 and rn<=10;
```

oracle 视图

介绍

视图是一个虚拟表，其内容由查询定义。同真实的表一样，视图包含一系列带有名称的列和行数据。但是，视图并不在数据库中以存储的数据值形式存在。行和列数据来自由定义视图的查询所引用的表，并且在引用视图时动态生成。

视图是 oracle 又一种数据对象，视图的主要的用处是简化操作，提高案例，满足不同用户的查询需求，视图不是一个真正存在的物理表，它是根据别的表动态生成的。

创建视图基本语法：

create view 视图名 as select 语句 [with read only]

特别说明：with read only 如果带上的话，只能查询，不能改写。

创建或修改视图基本语法：

create or replace view 视图名 as select 语句 [with read only]

特别说明：with read only 如果带上的话，只能查询，不能改写。

删除视图基本语法：

drop view 视图名;

案例：

创建和 emp 表(empno,ename,job)完全一致的视图，看看带 with read only 和不带的区别。

注意：当表结构过于复杂时，请使用视图。

```
create or replace view empview as
select empno,ename,job
from emp;
```

视图可以简化操作(可以将多表中的字段合并在一张视图中)

例：我们希望查询雇员的名字和部门编号和部门名称

```
create or replace view empdeptview as
select e.ename,e.deptno,d.dname
from emp e,dept d
where e.deptno=d.deptno
with read only;
```

视图与表的区别

1、表需要占用磁盘空间，而视图不需要；

2、视图不能添加索引；

3.使用视图可以简化复杂查询；

4.使用视图利于提高安全性。

5.对视图的修改将会影响实际的数据表

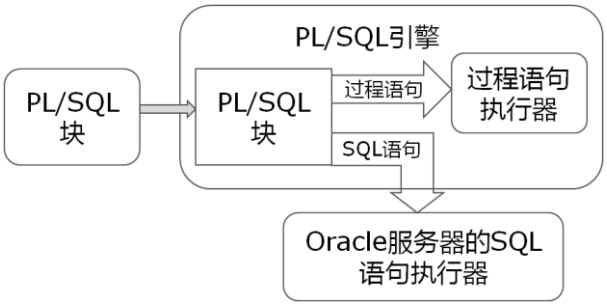
Day09-PL/sql 基础

PL/SQL 简介

到目前为止，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：



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。

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）声明部分
BEGIN
        --executable statements    2）执行部分
[EXCEPTION
        --exception statements]    3）异常处理部分
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。

- **举例：输出‘Hello World!’；**

```
begin
        dbms_output.put_line('hello world');
end;
```

Dbms_output 是 Oracle 所提供的包（类似 java 开发包），该包包含一些过程，**put_line** 就是该包下的一个**过程**。

- **plsql 说明：**

1）**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：

2）**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。

3）**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。**

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：

Plsql 代码规范：

注释：

a) 单行注释：--

b) 多行注释：/*……*/

标识符号和命名规范：

a) 当定义变量时，建议使用 v_作为前缀，如 v_sal；

b) 当定义常量时，建议使用 c_作为前缀，如 c_rate；

c) 当定义游标时，建议使用_cursor 作为后缀，如 emp_cursor；

d) 当定义例外时，建议使用 e_作为前缀，例如 e_error；

变量的命名规范：

a) 首字母必须是字母_、$、#

b) 变量名字的长度不能超过 30

声明变量

语法格式：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)。

■ **例子：声明变量**

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

说明：

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。

对变量 sname 重新赋值，赋值运算符是“:=”。

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。

■ **例子：从数据库中查询出结果赋给变量**

代码演示：变量赋值

```
DECLARE
        sname VARCHAR2(20) DEFAULT 'jerry';   --（1）
BEGIN
        SELECT ename INTO sname FROM emp WHERE empno=7934;   --（2）
        dbms_output.put_line(sname);
END;
```

输出结果：

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

说明：

（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。

（3）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

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。

例子：计算元的面积：

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

输出结果：

```
28.26
PL/SQL procedure successfully completed
```

说明：

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。

例子：全局常量

```
SQL> var emp_name varchar(30);    --声明全局变量 emp_name
SQL> BEGIN
        SELECT ename INTO :emp_name FROM emp WHERE empno=7499;    ②
        END;
      /
```

输出结果：

```
PL/SQL procedure successfully completed
emp_name
---------
ALLEN
```

通过 print 命令行输出变量：

```
SQL> print emp_name;
emp_name
```

```
---------
ALLEN
```

说明：

（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。

在 SQL*Plus 也可以像 Java 一样从控制台输入变量的值。

例子：从控制台输入变量值

格式：

&变量名

```
declare   --定义变量
        v_ename varchar2(10);
        v_empno number;
begin --执行部分
         v_empno =: &no;
        select ename into v_ename from emp where empno=v_empno;
        dbms_output.put_line('用户名'||v_ename);
exception
        when no_data_found then
        dbms_output.put_line('未发现该编号');
end;
```

说明：

（1）使用&给 empno 赋值。

（2）no_data_found ： PL/SQL 的预定义异常。

问题：每次运行一个“语句块”的时候，都要把语句输入一遍，麻烦不？

Day09 PL/sql 流程控制、存储过程

存储过程：

存储过程是 SQL 语句和可选控制流语句的预编译集合，以一个名称存储并作为一个单元处理。

存储过程**存储在数据库内**，可由应用程序通过一个调用执行，而且允许用户声明变量、有条件执行以及其它强大的编程功能。

存储过程在创建时即在服务器上进行编译，所以执行起来比单个 SQL 语句快。

存储过程的优点：

1. 存储过程只在创造时进行编译，以后每次执行存储过程都不需再重新编译，而一般 SQL 语句每执行一次就编译一次,所以使用存储过程可提高数据库执行速度。

2. 当对数据库进行复杂操作时(如对多个表进行 Update,Insert,Query,Delete 时），可将此复杂操作用存储过程封装起来与数据库提供的事务处理结合一起使用。

3. 存储过程可以重复使用,可减少数据库开发人员的工作量

4.安全性高,可设定只有某此用户才具有对指定存储过程的使用权

第一个案例(创建一个简单的存储过程)

例如：向 emp 表中插入数据（111,’张三’）

```
create procedure pro1 is
begin
    insert into emp(empno, ename)values(111,'张三');
end;
```

调用存储过程：

```
exec pro1
```

带参数的存储过程

```
create procedure pro2(in_empno number) is
begin
    delete from emp where empno = in_empno;
end;
```

调用：exec pro2(111);

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。

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

```
DECLARE
      sal emp.sal%TYPE;  ①
      mysal number(4):=3000;
      totalsal mysal%TYPE;  ②
BEGIN
      SELECT SAL INTO sal FROM emp WHERE empno=7934;
      totalsal:=sal+mysal;
```

```
        dbms_output.put_line(totalsal);
    END;
    /
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。

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

if (条件){
//条件结构体
}
	IF 条件 THEN
	--条件结构体
	END IF;

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

if (条件){
　　//条件成立结构体
}
else{
　　//条件不成立结构体
}
	IF 条件 THEN
	　　--条件成立结构体
	ELSE
	　　--条件不成立结构体
	END IF;

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

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
}
	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;

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

在程序顺序结构中有两个特殊的语句。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 sedond_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。

Day10-游标

游标：

游标：用来处理使用 select 语句从数据库中检索到的多行记录的工具

分类：

1） 显示游标

返回多条记录时，使用显示游标逐行读取

2） 隐式游标

PL/SQL 自动为 DML 语句创建隐式游标，包含一条返回记录

游标的属性

属性名称	说　明
%found	**用于检验游标是否成功，通常在 FETCH 语句前使用，当游标按照条件查询出一条记录时，返回 true**
%isopen	判断游标是否处于打开状态，视图打开一个已经打开或者已经关闭的游标，将会出现错误
%notfound	与%found 的作用相反，当按照条件无法查询到记录时，返回 true
%rowcount	**循环执行游标读取数据时，返回检索出的记录数据的行数**

游标的使用

1） 游标的声明

语法：

CURSOR cursor_name [(parameter [, parameter]……)]
[RETURN return_type] IS selectsql

说明：

CURSOR：用于声明一个游标
parameter：可选参数，用于指定参数类型、模式等
return：可选，指定游标的返回类型
selectsql:需要处理的 select 语句，不能含 INTO 子句

2）打开游标

语法： open test_cursor
使用 OPEN 语句开启一个游标

3）提取游标

语法：

FETCH cursor_name INTO variable_list

说明：

使用 FETCH 语句实现对游标内容的读取
variable_list 必须与从游标提取的结果集类型相同

4）关闭游标

语法：

close cursor_name

说明：

使用 CLOSE 语句关闭一个游标
关闭游标后，所有资源都将被释放，且不能再次被打开

示例：游标的使用

1）读取一条数据

```
例如：查询员工标号为：7369 的员工姓名和薪水；
DECLARE
    ename varchar2(20);
    sal emp.sal%type;
    --声明一个游标
    CURSOR c_emp is select ename, sal from emp where empno=7369 ;
BEGIN
    --打开游标
   OPEN c_emp;
   -- 判断游标是否返回记录
   if c_emp %NOTFOUND THEN
        dbms_output.put_line('没有找到相应的记录');
   else
   --从游标中读取记录
        fetch c_emp into ename,sal;
        dbms_output.put_line(ename||'薪水为'||lname);
   end if;
```

```
    CLOSE c_emp;
END;
```

2）读取多条记录：

使用 FETCH 语句只能读取一行记录，那么如何读取多行记录呢？

读取多行记录时，可以采用循环的方式

LOOP 循环

while 循环

FOR 循环

例如： 查询薪水小于 3000 的所有人的姓名和薪水？

适用 for 循环，自动打开游标，而无需使用 open 语句

```
DECLARE
        v_ename varchar2(20);
        v_sal emp.sal%type;
        CURSOR c_emps is select ename, sal from emp where sal<=3000 ;
BEGIN
      FOR c_emp IN c_emps LOOP
        V_ename:=c_emp.ename;
        V_sal:=c_emp.sal;
        dbms_output.put_line('员工姓名:'||v_ename||'薪水为'||v_sal);
      END LOOP;
END;
```

说明：

1.使用 for 循环时，自动打开游标，而无需使用 OPEN 语句

2.PL/SQL 会自动对变量进行隐式声明

3.当循环结束后，游标会总动关闭

练习

- **需求说明：**
  - 编写 PL/SQL 语句，实现使用游标从表中读取出‘SMITH’的相关信息（姓名，编号，部门名称）
- **提示：**
  - 使用 OPEN 打开游标
  - 对游标内容进行判断
  - 使用 FETCH 进行读取
  - 关闭游标

```
declare
    v_ename emp.ename%type;
```

```
    v_deptno      emp.deptno%type;
    v_dname    dept.dname%type;
   --声明一个游标
    cursor c_emp is select ename,emp.deptno,dname
                    from emp,dept
                    where ename='SMITH' and emp.deptno=dept.deptno;
begin
  --打开游标
   open c_emp;
    --判断游标是否返回记录
   if c_emp%NOTFOUND then
      dbms_output.put_line('没有找到相应的记录!');
      else
        --从游标中读取记录
        fetch c_emp into v_ename,v_deptno,v_dname ;
        dbms_output.put_line(v_ename||'部门编号：'||v_deptno||'部门名称：'||v_dname);
        end if;
  --关闭游标
   close c_emp;
   end;
```

- **需求说明：**
  - 在上一练习基础上，实现读取‘smith’所在部门员工信息并显示
  - **提示：**
  - 使用 FOR 循环实现游标的循环读取

```
create or replace procedure p3 is
    v_ename varchar2(20);
    v_deptno number(4);
    v_dname varchar2(20);

    cursor c_emps is select ename,dept.deptno,dname
                     from emp ,dept
                     where emp.deptno=dept.deptno
                          and emp.deptno = (select deptno
                                            from emp
                                            where ename='SMITH');
    begin
      for c_emp in c_emps loop
        v_ename:= c_emp.ename;
        v_deptno:=c_emp.deptno;
        v_dname:= c_emp.dname;
        dbms_output.put_line(v_ename||'部门编号'||v_deptno||'部门名称：'||v_dname);
```

```
        end loop;
    end;
```

Day12-触发器

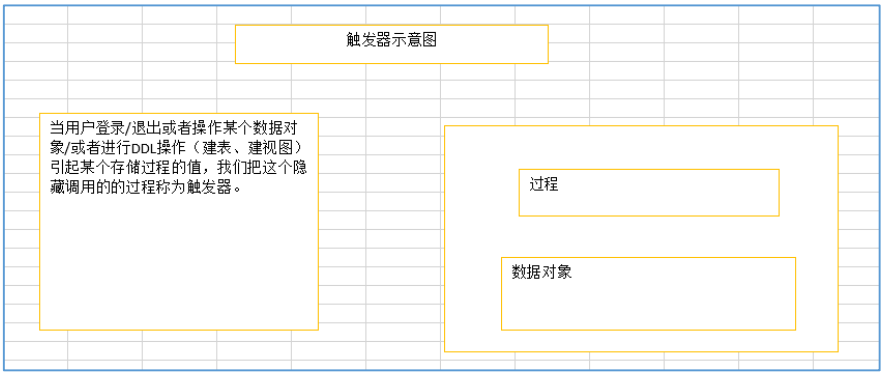
触发器：

先思考几个问题：

1. 当一个用户登录到 Oracle 时，在一张表中记录登录到 Oracle 的用户名和登录时间等信息，怎么办？
2. 禁止用户在星期天对某一张表进行删除操作，怎么办？
3. 当用户在删除一张表的时候，自动把删除的记录备份到另一张表里去……

很多关系型数据库都提供了一种技术，可以在用户进行某种操作的时候，自动地进行另外一种操作，我们把这种技术成为触发器技术。

触发器是指存放在数据库中，**被隐含执行的存储过程**，可以支持 **DML 触发器**，还支持**基于系统事件**（启动数据库、关闭数据库、登录数据库以及 **DDL 操作建立触发器**）。



触发器的分类：

1. DML 触发器（insert 、delete 、update 触发器）
2. DDL 触发器（create table、create view 、drop table……）
3. 系统触发器（与系统相关的触发器，比如用户登录、退出、启动数据库、关闭数据库）

如何创建触发器？

语法结构：

```
Create [or replace] trigger trigger_name      –触发器名字
{brfore| after}                                —在操作前还是后
{insert | delete | update [of column [, column ……]]} –进行增删改查的哪一种操作
On [schema.] table_name                        --在哪个用户的哪张表中
[for each row]                                 --每一行
[where condition]                              -- where 条件
Begin
   Trigger_body;
End;
```

after—是在记录操纵之后触发，是先完成数据的增删改，再触发，触发的语句晚于监视的增删改操作;

before—是在记录操纵之前触发，是先完成触发，再增删改，触发的语句先于监视的增删改.

DML 触发器：

案例：

- 在某张表（u1）添加一条数据的时候，提示“ 添加了一条数据”。

```
create table u1(id number,name varchar2(20));

create or replace trigger t1
after insert on u1
begin
   dbms_output.put_line('你在 u1 表中插入了一条数据！ ');
   end;

insert into u1 values(1,'张三');
```

- 在某张表(emp)修改多条数据的时候，提示多次“修改了数据”。

如果这样写，只会输出一句话。（语句级的写法）

```
create or replace trigger t1
after update on scott.emp
begin
      dbms_output.put_line('修改了一条数据');
end;

--执行更新操作
update emp set sal = sal +10;
```

应该这样写，才会输出多条语句:（变为行级写法）

```
create or replace trigger t1
after update on scott.emp
for each row--代表行级触发器
begin
      dbms_output.put_line('修改了一条数据');
end;

--执行更新操作
update emp set sal = sal +10;
```

- 案例：为了禁止工作人员在休息日改变员工的信息，开发人员可以建立 before 语句触发器，从而实现数据的安全。

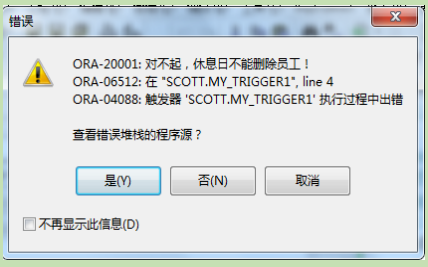
```
create or replace trigger t1
before delete on scott.emp
begin
    if to_char(sysdate,'day')in('星期五','星期日') then
          dbms_output.put_line('休息日不能删除员工信息！');
          raise_application_error(-20001,  '对不起，休息日不能删除员工！');
    end if;
end;

--执行删除操作，抛出错误信息。
     delete from emp where empno    =7499;
```

说明：

1）raise_application_error (error_number number, error_msg varchar2)系统已经定义好的一个存储过程，让系统报错。

2）Error_number [自定义]从-20000 到-20999 之间，这样就不会与 Oracle 的任何错误代码发生冲突。Error_msg_in[自定义]的长度不能超过 2K，否则截取 2K，是错误的提示信息;



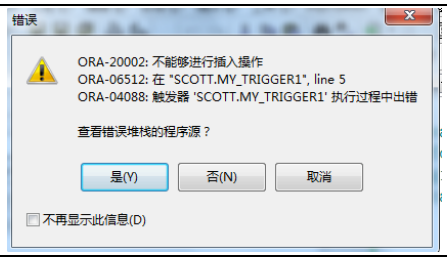
使用条件谓词：

当触发器中同时包含多个触发事件(insert 、update、delete)时，为了在

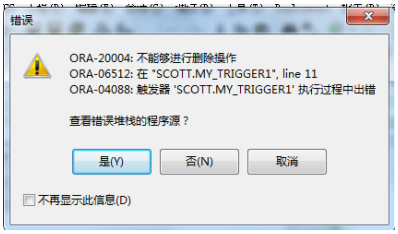
触发器代码中区分具体的触发事件，可以使用三个条件:inserting、updating、deleting。

案例：为了禁止工作人员在休息日改变员工的信息，开发人员可以建立 before 语句触发器，从而实现数据的安全。在给出提示时，明确提示用户是进行的 insert、update 还是 delete 操作。

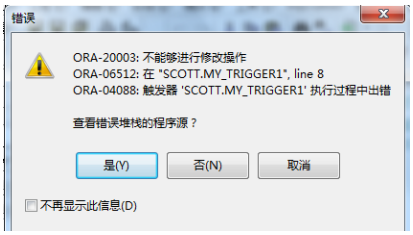
```
create or replace trigger t1
before delete or update or insert on scott.u1
begin
case
when inserting then
        dbms_output.put_line('请不要进行插入操作！');
        raise_application_error(-20002,'不能够进行插入操
                        作');
when updating then
        dbms_output.put_line('请不要进行修改操作！');
        raise_application_error(-20003,'不能够进行修改操
                        作');
when deleting then
        dbms_output.put_line('请不要进行删除操作！');
        raise_application_error(-20004,'不能够进行删除操
                        作');
end case;
end;
--执行插入操作
    insert into u1(empno, ename)values(1,'张三');
```




```
--删除操作
delete from emp where empno =7499;
```




```
--更新操作
update emp set ename ='张三'where empno =7499;
```



:old 和 :new 关键字

思考：

当触发器被触发时，要使用被插入、删除或修改的记录中的列值，有时要使用操作前、后列的值。

:old 修饰符访问操作完成前列的值

:new 修饰符访问操作

特性	INSERT	UPDATE	DELETE
OLD	NULL	有效	有效
NEW	有效	有效	NULL

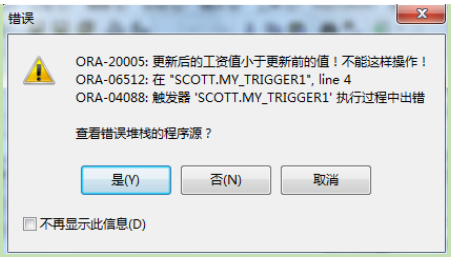
案例 4：

在修改 emp 表雇员的薪水时，显示雇员工资修改前和修改后的值。

如何确保在修改员工工资不能低于原有工资。

```
create or replace trigger t1
before update on scott.emp
for each row
begin
if :new.sal < :old.sal then
        dbms_output.put_line('更新后的工资值小于更新前的值！不能这样操作！');
        raise_application_error(-20005,'更新后的工资值 小于更新前的值！不能这样操作！');
else
        dbms_output.put_line('更新前工资为:'||:old.sal ||'更新后工资为：'||:new.sal);
endif;
end;

--执行 update 操作
update emp set sal =2 where empno =7499;
```



update emp set sal =7000where empno =7499;

更新前工资为：6000更新后工资为：7000

案例 5：

编写一个触发器，保证当用户在删除一张表（emp）记录的时候，自动将删除的记录备份到另一张表(u1)中。

```
create or replace trigger t2
before delete on scott.emp
for each row
begin
insert into u1(id, ename)values(:old.empno,:old.ename);
end;

delete from emp where empno =7499;
```

系统触发器：

系统事件是指基于 Oracle 事件（例如用户**登录** logon、logoff 和数据库的**启动**和**关闭** startup、shutdown）所建立的触发器，通过使用系统事件触发器，提供了跟踪系统或者是数据库变化的机制;

下面介绍一些常用的系统事件属性函数，和建立各种事件触发器的方法，在建立系统事件触发器时，我们需要使用事件属性函数，常用的时间属性函数如下：

```
ora_client_ip_address  //返回客户端的 ip
ora_database_name    //返回数据库名
ora_login_user        //返回登录用户名
ora_sysevent          //返回触发触发器的系统事件名
ora_des_encrypted_password  //返回用户 des(md5)加密后的密码
```

案例：建立登录或者退出触发器：

```
语法结构：
Create or replace trigger  触发器名字
After [before] logon [logoff] on database
Begin
//执行语句……;
End;
```

案例：完成登录触发器和退出触发器的创建。

为了记录用户的登录和退出事件，我们可以建立登录或者退出触发器。为了记录用户名称、时间、ip 地址，我们首先建立一张信息表：

打开命令行窗口：

Sqlplus system/123456@orcl as dba;

```
创建一张存储登录或者退出系统信息的表。
Create table log_table( username varchar2(20),
                        logon_time date,
                        logoff_time date, address varchar2(20));

--创建登录触发器：
Create or replace trigger mytrigger1
After logon on database
Begin
   Insert into log_table(username, logon_time , address)
   values(ora_login_user, sysdate, ora_client_ip_address);
End;

--创建退出触发器：
Create or replace trigger
Before logoff on database
Begin
   Insert into log_table(username, logoff_time, address)
  values(ora_login_user, sysdate , ora_client_ip_address);
End;
```

DDL 触发器：

DDL 简单说就是我们平常使用的 create、alter 和 drop 这些数据定义语句。

```
基本语法结构：
Create or replace trigger  触发器的名字
After ddl on  方案名.schema—如 scott.schema
Begin
--执行语句……;
```

```
End;
```

案例：

编写一个触发器，可以记录某个用户进行的 DDL 操作。

为了记录系统发生的 ddl 事件，应该建立一张表(my_ddl_record)用来存储相关的信息

注意：需要使用 system 用户登录: conn system/manager as sysdba;

```
Create table my_ddl_record (
        event varchar2(64),
        username varchar2(64),
        ddl_time date
);

--创建触发器
Create or replace trigger mytrigger2
After ddl on scott.schema
Begin
      Insert into my_ddl_record values(ora_sysevent, ora_login_user, sysdate);
End;
```

注意：在 Oracle 中，DML 的事务需要手动提交，DDL 语句不需要进行手动提交。我们在写 DML 语句时，如果没有手动提交，在退出控制台的时候也会自动提交。

触发器管理

1. 禁止触发器

   指让触发器临时失效。

   ```
   Alter trigger 触发器名 disable;
   ```

2. 激活触发器

   ```
   Alter trigger 触发器名 enable;
   ```

3. 删除触发器

   ```
   Drop trigger 触发器名;
   ```

Day13-函数、序列

pl/sql 函数

函数一般用于计算和返回一个值，可以经常需要进行的计算写成函数。函数的调用是表达式的一部分，而调用过程是一条 PL/SQL 语句。

函数的结构

声明部分：包括类型、变量、游标
执行部分：完成功能而编写的 SQL 语句或则是 PL/SQL 代码块
异常处理部分：

函数与过程在创建的形式是有些相似，也是编译后放在内存中供用户使用，只不过调用时函数时要用表达式，而不像过程只需要用过程名。另外，函数必须有一个返回值，而过程则没有。

创建函数：

函数用于返回特定的数据，当建立函数时，在函数头部必须包含 return **子句**，而在函数体内必须包含 return **语句返回的数据。**

基本语法：

```
create function  函数名(参数 1,...)
return  数据类型  is
        定义变量;
begin
        执行语句;
end;
/
```

举例：

```
--创建函数
CREATE OR REPLACE Function GETCOUNT(Major IN varchar2 )
--声明返回类型
return number is
f_count number;
BEGIN
--使用 INTO  语句将结果赋值给变量
select count(*) into f_count from students where major='Major' ;
--使用 RETURN  语句返回
return f_count;    --返回结果
END;
```

函数调用

方式一：

```
--调用函数--
```

```
DECLARE
        --声明变量接收函数的返回值
        v_count number;
BEGIN
      v_count:=GETCOUNT('MUSIC');
      Dbms_Output.put_line(v_count);
END;
```

方式二：

```
var  变量名  变量类型
call  函数名(参数值,...) into :变量名;
print  变量名
```

方式三：

```
select  函数名(参数,...) from dual;
```

删除函数：

语法

```
DROP FUNCTION  函数名
```

案例：

请编写一个函数，可以接收用户名并返回该用户的年薪。

```
create function inName_outSal(v_in_name varchar2)
   return number is
  v_annual_sal number;
begin
      select (sal+nvl(comm,0))*13 into v_annual_sal from emp where ename=v_in_name;
      return v_annual_sal;
end;
/
```

函数和过程的区别：

	过程	函数
标识符	PROCEDURE	FUNCTION
返回值	必须使用变量形参	用函数名直接返回
赋值	不能赋值并定义类型	可以定义类型，并直接赋值
调用方式	独立的过程调用句	以表达式方式调用
目的	完成一系列的数据处理	获得函数返回值

1、函数必须有返回值，而过程可以没有；

2、函数和过程在 java 中调用的方式不一样；

3、java 中调用 oracle 函数可以在 select 语句中直接调用，如：select 自定义的函数名(参数) from 表; 过程则是使用 CallableStatement 完成调用。

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：

START WITH：从某一个整数开始，升序默认值是 1，降序默认值是-1。

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。

MAXVALUE：指最大值。

NOMAXVALUE：这是最大值的默认选项，升序的最大值是：1027，降序默认值是-1。

MINVALUE：指最小值。

NOMINVALUE：这是默认值选项，升序默认值是 1，降序默认值是-1026。

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。

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。

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。

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：

不能修改序列的初始值。

最小值不能大于当前值。

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
```

Day14-索引

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：

UNIQUE:指定索引列上的值必须是唯一的。称为唯一索引。

index_name：指定索引名。

tabl_name：指定要为哪个表创建索引。

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：为 EMP 表的 ENAME 列创建创建唯一索引，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：

为 SCOTT.EMP 表的 ENAME 列创建唯一索引。

为 SCOTT.EMP 表的 SAL 列创建索引。

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。

索引--原理介绍

索引是用于加速数据存取的数据对象，合理的使用索引可以大大降低 I/O 次数，从而提高数据访问性能。

为什么添加了索引后，会加快查询速度呢？

索引可以理解成类似书的目录，不用查找所有的内容来进行匹配，而是通过目录对应来快速找到存放的地方，然后再将其取出。

单列索引

单列索引是基于单个列所建立的索引，语法：

```
create index index_name on table(columnname);
create index  索引名称  on  表名(列名);
```

复合索引

复合索引是基于两列或者多列的索引。在同一张表上可以有多个索引，但是要求列的组合必需不同，

语法：

```
create index index_name on table(columnname,columnname…);
create index  索引名称  on  表名(列名 1，列名 2,…);
```

索引的使用原则：

1、在大表上建立索引才有意义；

2、在 where 子句或是连接条件上经常引用的列上建立索引；

3、在逻辑型类型字段上或者值就是固定几种的列上也不要建立索引。

索引的缺点

索引缺点分析

索引有一些先天不足：

1、建立索引，系统要占用大约为表的 1.2 倍的硬盘和内存空间来保存索引。

2、更新数据的时候，系统必需要有额外的时间来同时对索引进行更新，以维持数据和索引的一致性。实践表明，不恰当的索引不但于事无补，反而会降低系统性能。因为大量的索引在进行插入、修改和删除操作时比没有索引花费更多的系统时间。

比如在如下字段建立索引应是不恰当的：

1、很少或从不引用的字段；

2、逻辑型的字段，如男或女(是或否)等。

综上所述，提高查询效率是以消耗一定的系统资源为代价的，索引不能盲目的建立，这是考验一个 DBA 是否优秀的很重要的指标。

索引分类

简单了解

按照数据存储方式，可以分为 **B*树、反向索引、位图索引；**

按照索引列的个数分类，可以分为**单列索引、复合索引；**

按照索引列值的唯一性，可以分为**唯一索引和非唯一索引。**

此外还有**函数索引、全局索引、分区索引...**

oracle 权限和角色(作为扩展内容，自己看一下)

介绍

这一部分主要看看 oracle 中如何管理权限和角色，权限和角色的区别在哪里。当刚刚建立用户时，用户没有任何权限，也不能执行任何操作。如果要执行某种特定的数据库操作，则必需为其授予系统的权限；如果用户要访问其它方案的对象，则必需为其授予对象的权限，为了简化权限的管理，可以使用角色。

权限：

权限是指执行特定类型 sql 命令或是访问其它方案对象的权利，包括**系统权限和对角权限**两种。

系统权限介绍

系统权限是指执行特定类型 sql 命令的权利，它用于控制用户可以执行的一个或是一组数据库操作。比如当用户具有 create table 权限时，可以在其方案中建表，当用户具有 create any table 权限时，可以在任何方案中建表。oracle 提供了 160 多中系统权限。**常用的有：**

create session 连接数据库;

create view 建视图;

create procedure 建过程、函数、包;

create cluster 建簇;

create table 建表;

create public synonym 建同义词;

create trigger 建触发器;

显示系统权限

oracle 提供了 166 系统权限，而且 oracle 的版本越高，提供的系统权限就越多，我们可以**查询数据字典视图 system_privilege_map,可以显示所有系统权限**。

基本语法：

```
select * from system_privilege_map order by name;
```
oracle11GR2 中提供了 208 个系统权限。

授予系统权限

一般情况，授予系统权限是有 dba 完成的，如果用其它用户来授予系统权限，则要求用户必需具有 grant any privilege 的系统权限在授予系统权限时，可以带有 **with admin option 选项**，这样，**被授予权限的用户或是角色还可以将该系统权限授予其它的用户或是角色**。为了让大家快速入门，我们举例说明：

1、创建两个用户 ken,tom 初始阶段他们没有任何权限，如果登陆就会出错误的信息。

1.1、创建两个用户，并指定密码。

```
SQL>create user ken identified by ken;
SQL>create user tom identified by tom;
```

2、给用户 ken 授权

2.1、授予 create session 和 create table 权限时带 with admin option

授权基本语法：

grant 权限名称 to 用户名;

```
SQL>grant create session to ken with admin option;
SQL>grant create table to ken with admin option;
```

2.2、授予 create view 时不带 with admin option

```
SQL>grant create view to ken;
```

3、给用户 tom 授权

我们可以通过 ken 给 tom 授权，因为 with admin option 是加上的。当然也可以通过 dba 给 tom 授权，我们就用 ken 给 tom 授权：

```
SQL>grant create session to tom with admin option;
SQL>grant create table to tom with admin option;
SQL>grant create view to tom;//报错，由于 ken 被 dba 授权时没有带 with admin option 参数，所以 ken 没有权限对 tom 进行 create view 的授权。
```

回收系统权限

一般情况下，**回收系统权限是 dba 来完成的**，如果其它的用户来回收系统权限，要求该用户必需具有相应系统权限及转授系统权限的选项(with admin option)。回收系统权限使用 revoke 来完成，当回收了系统权限后，用户就不能执行相应的操作了，但是请注意，**系统权限级联收回问题？(不会级联回收权限)**

system==>>ken==>>tom

(create session)(create session)(create session)

用 system 执行如下操作：

revoke create session from ken;请思考 tom 还能登录吗？可以登录。

回收系统权限基本语法：
revoke 系统权限名 from 用户名;
特别说明：系统权限的回收不是级联回收。

对象权限介绍

访问其它方案对象的权利，用户可以直接访问自己方案的对象，但是如果要访问别的方案的对象，则必需具有对象的权限，比如 smith 用户要访问 scott.emp 表(scott:方案，emp：表)
则必需在 scott.emp 表上具有对象的权限。**常用的权限有：**
alter 修改、delete 删除、select 查询、insert 添加、update 修改、index 索引、references 引用、execute 执行。
查看 oracle 提供的所有的对象权限(仅 dba 用户可以查看)
select distinct privilege from dba_tab_privs;

重要查询文档
查看当前用户的表(自己的表)
```
select table_name from user_tables;
```

查询 oracle 中所有的系统权限，一般是 dba
```
select * from system_privilege_map order by name;
```

查询 oracle 中所有的角色，一般是 dba
```
select * from dba_roles;
```

查询 oracle 中所有对象权限，一般是 dba
```
select distinct privilege from dba_tab_privs;
```

查询数据库的表空间
```
select tablespace_name from dba_tablespaces;
```

查询当前用户具有什么样的系统权限
```
select * from user_sys_privs;
```

查询当前用户在别人的表上，具有什么样的对象权限
```
select * from user_tab_privs; (查看对表的权限)
select * from user_col_privs; (查看对表的列的权限)
```

查询某个用户具有怎样的角色
```
select * from dba_role_privs where grantee='用户名';
```

查看某个角色包括哪些系统权限。
```
select * from dba_sys_privs where grantee='DBA';
```
或者是：

```
select * from role_sys_privs where role='DBA';
```

查看某个角色包括的对象权限

```
select * from dba_tab_privs where grantee='角色名';
```

当某个角色具有什么样的系统权限或对象权限，也可通过 pl/sql developer 工具直接查看即可。

显示执行的错误信息
在执行完语句后，就执行下面的语句

```
show error;
```

显示 oracle 某个操作用时

```
set timing on;
```

授权对象权限

在 oracle9i 前，授予对象权限是由对象的所有者来完成的，如果用其它的用户来操作，则需要用户具有相应的(with grant option)权限，从 oracle9i 开始，sys,system 可以将任何对象上的对象权限授予其它用户，**授予对象权限是用 grant 命令来完成**的。

授权基本语法：

grant 对象权限 on 数据库对象 to 用户名 [with grant option] [,角色名]

特别说明：可以把权限直接赋给用户或角色。[with grant option]选项只能授予用户，不能授予角色。

我们看几个案例：

1、monkey 用户要操作 scott.emp 表，则必需授予相应的对象权限。

(1)、希望 monkey 可以查询 scott.emp 的表数据，怎样操作？

```
SQL>grant select on scott.emp to monkey;
```

(2)、希望 monkey 可以修改 scott.emp 的表数据，怎样操作？

```
SQL>grant update on scott.emp to monkey;
```

(3)、希望 monkey 可以删除 scott.emp 的表数据，怎样操作？

```
SQL>grant delete on scott.emp to monkey;
```

(4)、有没有更加简单的方法，一次把所有权限赋给 monkey？

```
SQL>grant all on scott.emp to monkey;
```

2、授权 alter 权限

如果 black 用户修改 scott.emp 表的结构，则必需授予 alter 对象权限

```
SQL>grant alter on scott.emp to black;
```

3、授予 execute 权限

如果用户想要执行其它方案的包/过程/函数，则需有 execute 权限。比如为了让 ken 可以执行包 dbms_transaction，可以授予 execute 权限。

```
SQL>grant execute on dbms_transaction to ken;
```

4、授予 index 权限

如果想在别的方案的表上建立索引，则必需具有 index 对象权限，如为了让 black 可以在 scott.emp 上建立索引，就给其 index 的对象权限

```
SQL>grant index on scott.emp to blake;
```

5、使用 with grant option 选项

该选项用于转授对象权限，但是该选项只能被授予用户，而不能授予角色。

例：由 blake 给 jones 授予 select 权限

先由 dba 给 blake 授予 select 权限

```
SQL>conn system/orcl;
SQL>grant select on scott.emp to blake with grant option;
SQL>conn blake/orcl;
SQL>grant select on scott.emp to jones;
```

回收对象权限

在 oracle9i 中，收回对象的权限可以由对象的所有者来完成，也可以用 dba 用户(sys,system)来完成。

这里要说明的是：收回对象权限后，用户就不能执行相应的 sql 命令，但是要注意的是**对象的权限是否会被级联收回？（级联回收）**

请看案例：

```
scott========>>blake========>>jones
select on emp   select on emp   select on emp
```

对象权限回收基本语法：

revoke 对象权限 on 数据库对象 from 用户名[,角色名];

特别说明：对象的权限回收是级联回收。

1、切换 system 用户

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

2、建立 blake 和 jones 用户

```
SQL>create user blake identified by blake;
SQL>create user jones identified by jones;
```

3、给 blake 和 jones 用户赋系统权限(登录权限)

```
SQL>grant create session to blake with admin option;
SQL>grant create session to jones;
```

4、切换 scott 用户给 blake 赋对象权限(查看权限)

```
SQL>conn scott/tiger;
SQL>grant select on emp to blake with grant option;
```

5、切换 blake 用户给 jones 赋对象权限(查看权限)

```
SQL>conn blake/blake;
SQL>grant select on scott.emp to jones;
```

6、切换 scott 用户回收 blake 对象权限

```
SQL>conn scott/tiger;
SQL>revoke select on emp from blake;
```

7、切换 blake 用户测试查看对象权限是否还能用。

```
SQL>conn blake/blake;
SQL>select * from scott.emp;//报错，blake 无查询权限。
```

8、切换 jones 用户测试查看对象权限是否还能用。

```
SQL>conn jones/jones;
SQL>select * from scott.emp;//报错，jones 无查询权限。
```

管理权限和角色--角色

介绍

角色就是相关权限的命令集合，使用角色的主要目的就是为了简化权限的管理。

请看一个问题：假定有用户 1，2，3 为了让他们都拥有权限。

1、连接数据库

2、在 scott.emp 表上 select,insert,update

如果采用直接授权操作，则需要进行 12 次授权。

角色分为预定义角色和自定义角色。

预定义角色

预定义角色是指 oracle 所提供的角色，每种角色都用于执行一些特定的管理任务，下面我们介绍常用的**预定义角色 connect,resource,dba**

特别说明：角色可以包含系统权限，也可以包含对象权限。

要查看角色有怎样的权限可以通过下列语句查看：

```
select * from dba_sys_privs where grantee='DBA';
```

注意：查询时角色的名称要大写(DBA、CONNECT、RESOURCE)，小写无法查询

使用 system 登录可以查询所有预定义角色：

```
select * from dba_roles;
```

如何知道某个用户具有什么样的角色：

```
select * from dba_role_privs where grantee='用户名';
```

通过角色给用户赋权限基本语法：
grant 角色名[,角色名 2,...] to 用户名;

1、connect 角色
connect 角色具有一般应用开发人员需要的大部分权限，只要给用户授予 connect 和 resource 角色就够了，connect 角色具有哪些系统权限呢？
connect 角色具有：
create session 创建连接权限
2、resource 角色
resource 角色具有应用开发人员所需要的其它权限，比如建立存储过程、触发器等。这里需要注意的是 resource 角色隐含了 unlimited tablespace 系统权限。
resource 角色具有：
create trigger 创建触发器
create sequence 创建序列
create type 创建类型权限
create procedure 创建过程
create cluster 创建集群
create operator 创建运营商
create indextype 创建索引类型
create table 创建表

3、dba 角色
dba 角色具有所有的系统权限，及 with admin option 选项，默认的 dba 用户为 sys 和 system 他们可以将任何系统权限授予其它用户，但是要注意的是 **dba 角色不具备 sysdba 和 sysoper 的特权(启动和关闭数据库)**

案例：
创建一个用户，然后赋给 connect 角色和 resource 角色
```
SQL>create user tempuser identified by tempuser;
SQL>grant connect,resource to tempuser;
```

创建一个用户 jack 并将其设为具有 dba 角色的用户
```
SQL>create user jack identified by jack;
SQL>grant dba to jack;
```

自定义角色

顾名思义就是自己定义的角色，根据自己的需要来定义，一般是 dba 来建立，如果使用别的用户来建立，则需要具有 **create role** 的系统权限。在建立角色时可以指定验证方式(不验证，数据库验证等)
1、建立角色(不验证)
如果角色是公用的角色，可以采用不验证的方式建立角色。
建立角色不验证基本语法：

create role 角色名 not identified;

2、建立角色(数据库验证)

采用这样的方式时，角色名、口令存放在数据库中。当激活该角色时，必需提供口令。在建立这种角色时，需要为其提供口令。

建立角色需数据库验证基本语法：

create role 角色名 identified by 口令;

角色授权

当建立角色时，角色没有任何权限，为了使得角色完成特定任务，必需为其授予相应的系统权限和对象权限。

(一)给角色授权

给角色授予权限和给用户授权没有太多区别，但是要**注意，系统权限的 unlimited tablespace 和对象权限的 with grant option 选项是不能授予角色的**。

给角色授权基本语法：

grant 对象权限 on 数据库对象 to 自定义角色名;

案例：

完成将 create session,select on scott.emp,insert on scott.emp,update on scott.emp 授予角色，然后将该角色授予 a,b,c 用户。

1、使用 system 用户创建自定义角色

```
SQL>conn system/orcl;
SQL>create role crud_scott not identified;
```

2、给自定义角色 crud_scott 授权

```
SQL>grant create session to crud_scott;
SQL>grant select on scott.emp to crud_scott;
SQL>grant insert on scott.emp to crud_scott;
SQL>grant update on scott.emp to crud_scott;
```

3、通过将授权过的角色 crud_scott 给用户进行授权

```
SQL>grant crud_scott to a;
SQL>grant crud_scott to b;
SQL>grant crud_scott to c;
```

(二)分配角色给某个用户

一般分配角色是由 DBA 来完成的，如果要以**其它用户身份分配角色，则要求用户必需具有 grant any role 的系统权限**。

通过角色名授权用户基本语法：

grant 角色名 to 用户名 [with admin option];

如果给用户赋权限时带了 with admin option 选项，被授权的用户可以继续将此权限授予其它用户。

删除角色

使用 drop role，一般是 dba 来执行，如用**其它用户则要求该用户具有 drop any role 系统权限**

删除角色基本语法：

drop role 角色名;

显示角色信息

1、显示所有角色

```
select * from dba_roles;
```

2、显示角色具有的系统权限

```
select privilege,admin_option from role_sys_privs where role='角色名';
```

3、显示角色具有的对象权限

通过查询数据字典视图 dba_tab_privs 可以查看角色具有的对象权限或是列的权限。

```
select * from dba_tab_privs where grantee='角色名';
```

4、显示用户具有的角色，及默认角色

当以用户的身份连接到数据库时，oracle 会自动的激活默认的角色，通过查询数据库字典视图 dba_role_privs 可以显示某个用户具有的所有角色及当前默认的角色

```
select granted_role,default_role from dba_role_privs where grantee='角色名';
```

精细访问控制(只做了解，不详细介绍)

是指用户可以使用函数、策略实现更加细微的案例访问控制。如果使用精细访问控制，则当在客户端发出 sql 语句(select,insert,update,delete)时，oracle 会自动在 sql 语句后追加谓语(where 子句)，并执行新的 sql 语句。通过这样的控制，可以使得不同的数据库用户在访问相同的表时，返回不同的数据信息。使用函数或策略是为了更好的保护数据安全性，为不同权限的用户提供不同的安全级别，可有效的保障信息安全。