
第二章 SQL 语言

一、 SQL 语言基础

1 什么是 SQL

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

2 SQL 能做什么?

- SQL 面向数据库执行查询
- SQL 可从数据库取回数据
- SQL 可在数据库中插入新的记录
- SQL 可更新数据库中的数据
- SQL 可从数据库删除记录
- SQL 可创建新数据库
- SQL 可在数据库中创建新表
- SQL 可在数据库中创建存储过程
- SQL 可在数据库中创建视图
- SQL 可以设置表、存储过程和视图的权限

3 SQL 标准

SQL 是 1986 年 10 月由美国国家标准局 (ANSI) 通过的数据库语言美国标准, 接着, 国际标准化组织 (ISO) 颁布了 SQL 正式国际标准。1989 年 4 月, ISO 提出了具有完整性特征的 SQL89 标准, 1992 年 11 月又公布了 SQL92 标准, 在此标准中, 把数据库分为三个级别: 基本集、标准集和完全集。在 1999 年推出 99 版标准。最新版本为 SQL2016 版。

比较有代表性的几个版本: SQL86、SQL92、SQL99。

除了 SQL 标准之外, 大部分 SQL 数据库程序都拥有它们自己的私有扩展!

4 SQL 语言结构

4.1 数据查询语言 (DQL: Data Query Language)

其语句, 也称为“数据检索语句”, 用以从表中获得数据, 确定数据怎样在应用程序给出。关键字 SELECT 是 DQL (也是所有 SQL) 用得最多的动词, 其他 DQL 常用的关键字有 WHERE, ORDER BY, GROUP BY 和 HAVING。这些 DQL 关键字常与其他类型的 SQL 语句一起使用。

select.....from.....where.....查询数据

4.2 数据操作语言（DML: Data Manipulation Language）

其语句包括动词 INSERT, UPDATE 和 DELETE。它们分别用于添加，修改和删除表中的行。

insert.....插入一条数据

update.....更新一条数据

delete.....删除一条数据

4.3 事务处理语言（TCL: Transaction Control Language）

它的语句能确保被 DML 语句影响的表的所有行及时得以更新。

commit.....事物提交

rollback....事物回滚

savepoint..设置回滚点

4.4 数据控制语言（DCL: Data Control Language）

它的语句通过 GRANT 或 REVOKE 获得许可，确定单个用户和用户组对数据库对象的访问。

grant...授予用户权限

revoke..撤销用户权限

4.5 数据定义语言（DDL: Data Definition Language）

定义数据库对象语言，其语句包括动词 CREATE 和 DROP 等。

create.....创建数据库对象

drop.....删除数据库对象

alter.....修改数据库对象

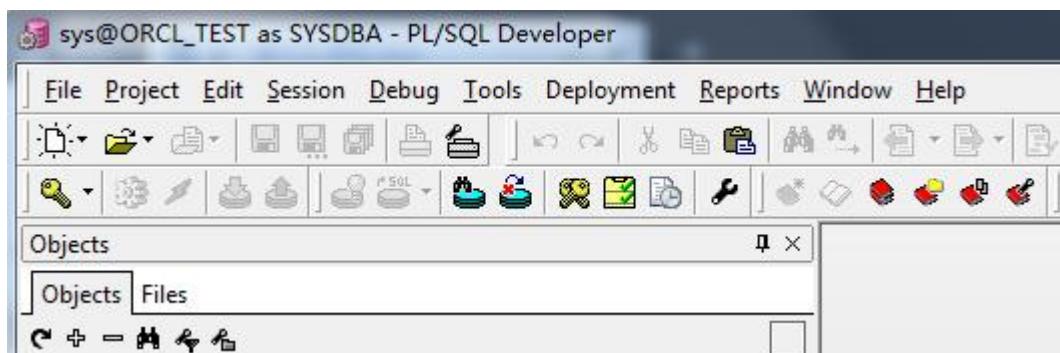
rename.....修改数据库对象名称

二、 Oracle 中的 HR 用户介绍

HR 用户是 Oracle 自带的一个示例用户。在该用户下提供了可供我们练习数据库操作时所使用的表与数据。

1 使用 HR 用户步骤

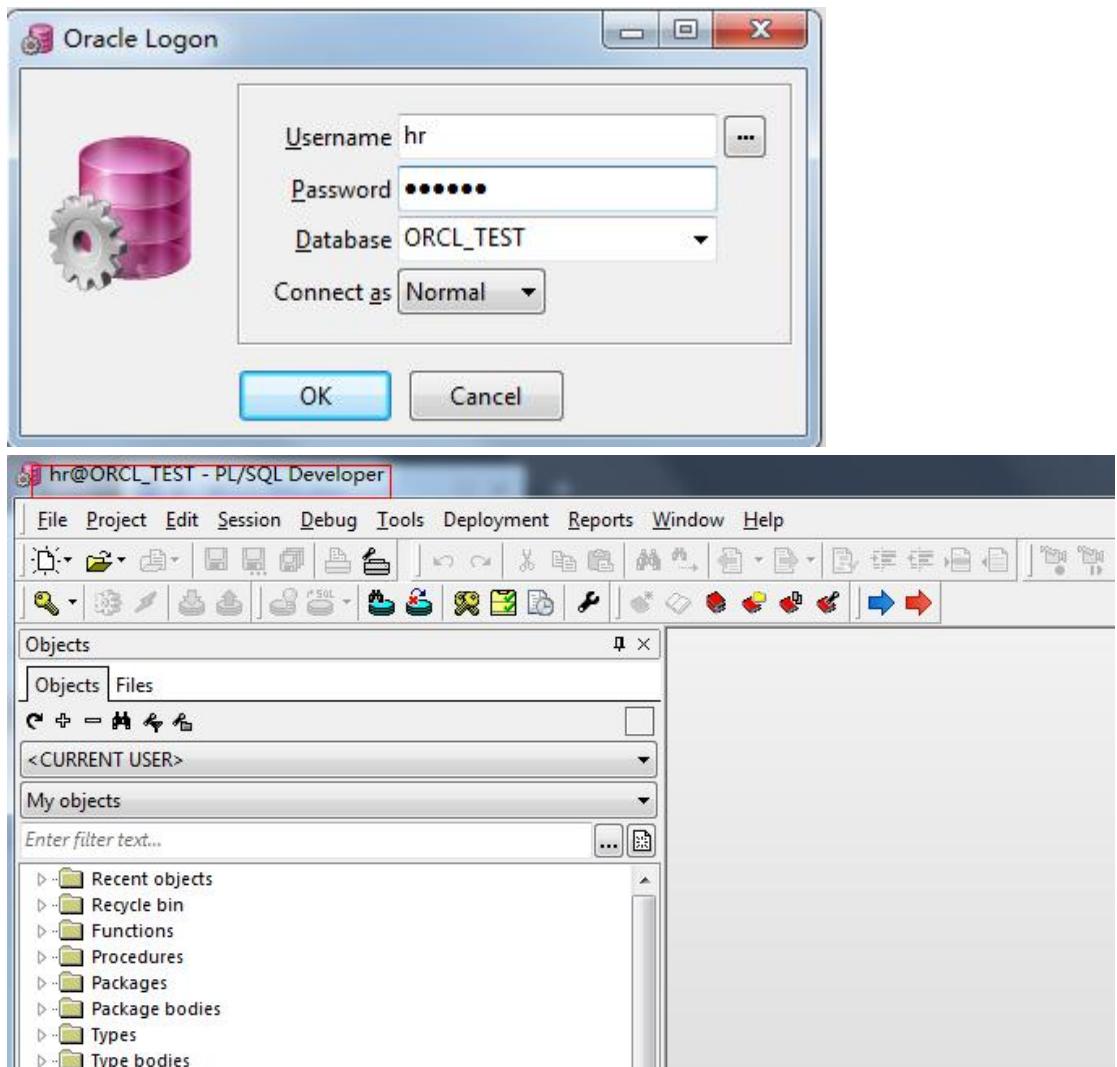
1.1 通过 sys 或 system 用户登录。



1.2 在 Users 中找到 HR 用户并设置登录密码。

A screenshot of the Oracle Database user management interface. On the left, there is a tree view of users and their objects. The "HR" user is selected. A context menu is open over the "HR" node, with the "Edit" option highlighted by a red box. Other options in the menu include New..., Duplicate..., Refresh, Copy comma separated, Properties, View, Drop, Browse, and Add to folder. Below the tree view, there is a navigation bar with tabs: General, Object privileges, Role privileges, System privileges, and Quotas. The "General" tab is selected. The main panel shows the "General" tab settings for the "HR" user. The fields are: Name (HR), Password (*****), Identified externally (unchecked), Default tablespace (USERS), Temporary tablespace (TEMP), Profile (DEFAULT). Below these fields are two checkboxes: "Password expire" (checked) and "Account locked" (unchecked).

1.3 切换 HR 用户登录



1.4 查看该用户下的表结构

The screenshot shows the PL/SQL Developer interface with the Objects Navigator expanded to show tables and the SQL editor displaying the description of the COUNTRIES table.

Objects Navigator:

- Tables folder:
 - COUNTRIES
 - DEPARTMENTS
 - EMPLOYEES
 - JOB_HISTORY
 - LOCATIONS
 - REGIONS
- Indexes folder.

SQL Editor:

```
SQL> desc countries;
Name          Type           Nullable Default Comments
-----        -----        -----
COUNTRY_ID    CHAR(2)        Primary key of countries table.
COUNTRY_NAME   VARCHAR2(40)  Y           Country name
REGION_ID     NUMBER         Y           Region ID for the country. Foreign key to region_id column in the departments table.
```

三、 DQL 语言

1 编写基本 SELECT 语句

SELECT 语句的作用是从数据库中返回信息。

1.1 SELECT 语句作用

1.1.1 列选择(投影操作)

能够使用 SELECT 语句的列选择功能选择表中的列，这些列是我们想要用查询返回的。当我们查询时，可在选择查询的表中指定的列。

1.1.2 行选择(选择操作)

能够使用 SELECT 语句的行选择功能选择表中的行，这些行是我们想要用查询返回的。能够使用不同的标准限制所看见的行。

1.1.3 连接(多表操作)

能够使用 SELECT 语句的连接功能来集合数据，这些数据虽然被存储在不同的表中，但是我们可以通过连接查询到该数据。

1.2 SELECT 语句基本结构

SELECT *|[DISTINCT] column|expression [alias],...} FROM table;

SELECT 确定哪些列。

FROM 确定哪张表。

1.2.1 基本 SELECT 语句

在最简单的形式中，SELECT 语句必须包含下面的内容：

一个 SELECT 子句，指定被显示的列

一个 FROM 子句，指定表，该表包含 SELECT 子句中的字段列表

1.2.2 在语法中

SELECT 是一个或多个字段的列表

* 选择所有的列

DISTINCT 关键字表示禁止重复

column|expression 选择指定的字段或表达式

alias 给所选择的列不同的标题

FROM table 指定包含列的表

1.3 选择操作(投影操作)

1.3.1 选择所有列

```
SELECT * FROM table;
```

1.3.2 选择指定的列

```
SELECT columnName,columnName FROM table;
```

1.4 SQL 语句语法要求

- SQL 语句对大小写不敏感
- SQL 语句可以写成一行或多行
- 关键字不能简写或分开折行
- 子句通常放在不同的行
- 缩进用于增强可读性

1.5 SELECT 语句中的算术表达式

用算术运算符创建数字和日期数据的表达式。(+ - * /)

注意：如果对日期进行计算，我们只能对 DATE 和 TIMESTAMP 数据类型使用加和减操作。

1.5.1 运算符的优先级



- 乘法和除法比加法和减法的优先级高
- 相同优先级的运算符从左到右计算
- 圆括号用于强制优先计算，并且使语句更清晰

1.5.2 示例

计算 employees 表中的员工全年薪水加 100 以后的薪水是多少？

```
select employee_id , first_name ,salary *12 +100 from employees;
```

计算 employees 表中的员工薪水加 100 以后的全年薪水是多少？

```
select employee_id,first_name ,(salary+100)*12 from employees;
```

1.6 定义空值

空值是一个未分配的、未知的，或不适用的值
空值不是 0，也不是空格

1.6.1 空值

如果一行中的某个列缺少数据值，该值被置为空值。空值和 0 或者空格不相同。0 是一个数字，而空格是一个字符。任何数据类型的列都可以包含空值。可是，某些约束，如，NOT NULL 和 PRIMARY KEY，防止在列中使用空。

1.6.2 算术表达式中的空值

包含空值的算术表达式计算结果为空。

1.6.3 示例

```
select last_name ,12*salary*commission_pct from employees;
```

1.7 定义列别名

列别名：

- 改变列标题的名字
- 紧跟在列名后面 – 在列名和别名之间可以有选项 AS 关键字
- 如果别名中包含有空格、或者特殊字符、或者大小写敏感，要求用双引号

在 SELECT 列表中的列名后面指定别名，列名和别名之间用空格分开。默认情况下，别名标题用大写字母显示。如果别名中包含空格或者特殊字符（例如 # 或 &），或者大小写敏感，将别名放在双引号 (“”) 中。

1.7.1 示例

查询 employees 表中的 last_name,commission_pct 并将佣金列名改为 comm。

方式一：`select last_name ,commission_pct as comm from employees;`
方式二：`select last_name name ,commission_pct comm from employees;`

计算所有员工的全年薪水，将列名修改“Annual Salary”。

方式一：`select last_name name ,12*salary as "Annual Salary" from employees;`

方式二：`select last_name name ,12*salary "Annual Salary" from employees;`

1.8 连字运算符

连字运算符：

- 连接列或者字符串到其它的列
- 用两个竖线表示 (||)
- 构造一个字符表达式的合成列

我们能够用连字运算符 (||) , 进行列与列之间、列与算术表达式之间或者列与常数值之间的连接，来创建一个字符表达式。连字运算符两边的列被合并成一个单个的输出列。

1.8.1示例

将 employees 表中的 LAST_NAME 和 JOB_ID 合并到一个单个的输出列中，并且指定列别名 Employees。

```
select last_name || job_id as "Employee" from employees;
```

1.9 文字字符串

文字字符串是包含在 SELECT 列表中的一个字符串，一个数字或者一个日期

- 日期和字符的文字字符串值必须用单引号括起来
- 每个文字字符串在每行输出一次

文字字符串不是列名或别名。对每个返回行打印一次。任意格式文本的文字字符串能够被包含在查询结果中，并且作为 SELECT 列表中的列处理。日期和字符文字 必须 放在单但引号 ('') 中；数字不需要。

1.9.1示例

显示所有雇员的名字和工作代码，使用 is a 链接查询结果。列标题用 Employee Details。

```
select last_name || ' is a ' || job_id as "Employee Details" from employees;
```

1.10 去除重复行

在 SELECT 语句中用 DISTINCT 关键字除去相同的行。为了在结果中除去相同的行，在 SELECT 子句中的 SELECT 关键字后面紧跟 DISTINCT 关键字。

1.10.1 示例

在 Employees 表中查询 department_id 并去除重复数据。

```
select distinct department_id from employees;
```

1.11基本 select 语句小节练习

1.11.1 在下面的语句中有 4 个编码错误，你能找出他们吗？

```
SELECT employee_id, last_name  sal x 12 ANNUAL SALARY  
FROM employees;
```

答案：1 在 employees 表中没有 sal 列，应该是 salary。

2 乘法运算符是 “*” 而不是 “x” 。

3 如果别名中包含空格，则需要在别名两侧添加双引号。

4 在 last_name 后少一个逗号。

1.11.2 显示 DEPARTMENTS 表的结构。选择表中的所有数据。

答案：显示表结构： desc departments;

查询所有数据： select * from departments;

1.11.3 创建一个查询，显示每个雇员的 last name, job code, hire date, 和 employee 号，employee 号显示在第一列，给 HIRE_DATE 列指定一个别名 STARTDATE 。

答案： select employee_id ,last_name,job_id,hire_date startdate from employees;

1.11.4 创建一个查询从 EMPLOYEES 表中显示唯一的工作代码。

答案： select distinct job_id from employees;

1.11.5 显示 last_name, 用 job_ID 连接，用逗号和空格分开，用 Employee and Title 作为列名。

答案： select last_name || ', ' || job_id from employees;

2 约束和排序数据

2.1 用选择限制行

SELECT *|[DISTINCT] column|expression [alias],...} FROM table [WHERE condition(s)];

WHERE 子句跟着 FROM 子句

WHERE 限制查询满足条件的行

condition 由列名、表达式、常数和比较操作组成

限制选择的行

你能够用 WHERE 子句限制从查询返回的行。一个 WHERE 子句包含一个必须满足的条件，WHERE 子句紧跟着 FROM 子句。如果条件是 true，返回满足条件的行。

WHERE 子句能够比较列值、文字值、算术表达式或者函数，WHERE 子句由三个元素组成：

列名

比较条件

列名、常量或值列表

2.1.1示例

查询雇员的 last_name, job ID 和 department 号，要求这些雇员的 department_id 是 90。

```
select last_name,job_id,department_id      from employees where  
department_id = 90;
```

2.2 字符串和日期

- 字符串和日期的值放在单引号中
- 字符值区分大小写，日期值是格式敏感的
- 日期的默认格式是 DD-MON-RR
- 中文版 Oracle 与英文版 Oracle 对于日期的月份格式有区别。中文版的用 1 月 英文版为月份的简写如：January Jan.

在 WHERE 子句中字符串和日期必须包含在单引号 (‘ ’) 中。但是，数字常数不应该包含在单引号中。所有的字符搜索是大小写敏感的。

Oracle 数据库以内部数字格式存储日期，表示为：世纪、年、月、日、小时、分和秒。默认的日期显示是 DD-MON-RR。

2.2.1示例

查询员工表中名字为 King 的员工的工作编号。

```
select job_id from employees where  
last_name = 'King';
```

查询 2006 年 1 月 24 日入职的员工的姓名、部门编号以及工作 ID。

```
select last_name, department_id, job_id  
from employees where hire_date = '24-1 月-06';
```

2.3 比较条件

条件运算符

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

注：不等于也可使用 != 和 ^= 来表示。

2.3.1示例

查询员工薪水小于等于 3000 的员工的姓名与薪水。

```
select last_name, salary from employees  
where salary <= 3000;
```

2.4其它比较条件

操作	含义
BETWEEN ... AND ...	在两个值之间 (包含)
IN (set)	匹配一个任意值列表
LIKE	匹配一个字符模板
IS NULL	是一个空值

2.4.1 使用 BETWEEN 条件

BETWEEN 条件: 可以用 BETWEEN 范围条件显示基于一个值范围的行。指定的范围包含一个下限和一个上限。BETWEEN ... AND ... 实际上是由 Oracle 服务器转变为 AND 条件: (a >= 下限) AND (a <= 上限), 所以使用 BETWEEN ... AND ... 并没有性能的提高, 只是逻辑上简单。

2.4.1.1 示例

查询薪水在\$2,500 和\$3,500(包含 2500 与 3500) 之间的那些雇员的姓名以及薪水。

```
select last_name, salary from employees  
where salary between 2500 and 3500;
```

2.4.2 使用 IN 条件

IN 条件: 用 IN 条件在指定的一组值中进行选择。IN (...) 实际上是由 Oracle 服务器转变为一组 OR 条件: a = value1 OR a = value2 OR a = value3, 所以使用 IN (...) 并没有得到性能的提高, 只是逻辑上简单。

2.4.2.1 示例

查询所有经理号为 100、101 或 201 的雇员的 employee numbers, last names, salary。

```
select employee_id, last_name, salary from  
employees where manager_id in(100,101,201);
```

2.4.3 使用 LIKE 条件

- 使用 LIKE 条件执行有效搜索串值的通配符搜索
- 搜索条件既可以包含文字也可以包含数字：

 % 表示零个或多个字符

 _ 表示一个占位符

LIKE 条件：我们也许不总能知道要搜索的确切的值，但能够选择那些用 LIKE 条件匹配一个字符模板的行。字符模板匹配运算涉及通配符查询。有两个符号 % 和 _ 可以用来构造搜索串。

2.4.3.1 示例一

查询 EMPLOYEES 表中名字以一个大写字母 S 开始的雇员的名字。

```
select last_name from employees where  
last_name like 'S%';
```

2.4.3.2 示例二

查询所有在 2005 年进入本公司的雇员的名字和受雇日期。

```
select last_name, hire_date from employees  
where hire_date like '%05';
```

ESCAPE：可以用 ESCAPE 标识符搜索实际的 % 和 _ 符号。使用 ESCAPE 选项，该选项指定换码符是什么。如果你想要搜索包含 ‘SA_’ 的字符串可以使用 ESCAPE 对表示该符号为转义符号。LIKE '%SA_%' ESCAPE '\';

2.4.3.3 示例三

查询员工表中工作 ID 中包含 SA_ 的员工姓名以及工作 ID。

```
select last_name, job_id from employees  
where job_id like 'SA\_%' escape '\';
```

2.4.4 使用 NULL 条件

NULL 条件：NULL 条件中包括 IS NULL 条件和 IS NOT NULL 条件。IS NULL 条件用于空值测试。空值的意思是难以获得的、未指定的、未知的或者不适用的。因此，不能用 = 判断，因为 null 不能等于或不等于任何值。IS NOT NULL 测试不是空值。

2.4.4.1 示例

查询所有没有佣金的雇员的 last name, job ID 和 commission。

```
select last_name, job_id, commission_pct  
from employees where commission_pct is null;
```

查询所有有佣金的雇员的 last name, job ID 和 commission。

```
select last_name, job_id, commission_pct  
from employees where commission_pct is not  
null;
```

2.5 逻辑条件关系

运算	含义
AND	如果两个组成部分的条件都为真，返回 TRUE
OR	如果两个组成部分中的任一个条件为真， 返回 TRUE
NOT	如果跟随的条件为假， 返回 TRUE

逻辑条件：逻辑条件组合两个比较条件的结果来产生一个基于这些条件的单个的结果，或者逆转一个单个条件的结果。当所有条件的结果为真时，返回行。SQL 的三个逻辑运算符是：AND 、 OR 、 NOT 。

2.5.1 AND

AND: AND 要求两个条件同时为真。

2.5.1.1 示例

查询工作岗位包含字符串 MAN 并且收入大于等于\$10,000 的那些雇员的编号、工作 ID、名字以及薪水。

```
select employee_id, job_id, last_name, salary  
from employees where job_id like '%MAN%' and  
salary >= 10000;
```

2.5.2 OR

OR: OR 操作要求两者之一为真即可。

2.5.2.1 示例

查询任何 job ID 中包含 MAN 或者收入大于等于\$10,000 的雇员编号、工作 ID、名字以及薪水。

```
select employee_id, job_id, last_name, salary  
from employees where job_id like '%MAN%' or  
salary >= 10000;
```

2.5.3 NOT

NOT: 取反。NOT 运算符也可以用于另一个 SQL 运算符，例如，BETWEEN、LIKE、IN 和 NULL。

2.5.3.1 示例

查询那些工作岗位不是 IT_PROG、ST_CLERK 或 SA REP 的雇员的名字和工作岗位。

```
select last_name, job_id from employees  
where job_id not  
in ('IT_PROG', 'ST_CLERK', 'SA REP');
```

2.6 优先规则

求值顺序	
1	算术运算
2	连字操作
3	比较操作
4	IS [NOT] NULL, LIKE, [NOT] IN
5	[NOT] BETWEEN
6	NOT 逻辑条件
7	AND 逻辑条件
8	OR 逻辑条件

优先规则：优先规则定义表达式求值和计算的顺序，表中列出了默认的优先顺序。你可以用圆括号括住你想要先计算的表达式来覆盖默认的优先顺序。

2.6.1示例一

查询工作岗位是 SA REP 或者工作岗位是 AD PRES 并且薪水大于 15000 的员工姓名、工作 ID 以及薪水。

```
select      last_name, job_id, salary      from
employees    where    job_id    =    'SA REP'    or
job_id='AD PRES' and salary > 15000;
```

2.6.2示例二

查询工作岗位是 SA REP 或者是 AD PRES 并且他们的薪水大于 15000 的员工姓名、工作 ID 以及薪水。

```
select      last_name, job_id, salary      from
employees    where    (job_id    =    'SA REP'    or
job_id='AD PRES') and salary > 15000;
```

2.7 使用 ORDER BY 排序

2.7.1 ORDER BY 子句作用

用 ORDER BY 子句排序

ASC: 升序排序, 默认

DESC: 降序排序

ORDER BY 子句: 用于对结果集进行排序处理, 提供了升序排序(ASC)与降序排序(DESC)如果不指定排序规则默认为升序排序。在排序中也可以使用没有包括在 SELECT 子句中的列排序。如果未使用 ORDER BY 子句, 排序次序就未定义, 并且 Oracle 服务器可能对于相同查询的两次执行取回行的顺序不同。

2.7.2 ORDER BY 语法

ORDER BY 子句在 SELECT 语句的最后

ORDER BY 后侧指定需要排序列

ASC 以升序排序行 (这是默认排序)

DESC 以降序排序行

2.7.3 升序规则

对于数字值, 小的值在前面显示—例如, 1-999。

对于日期, 早的日期在前面显示—例如, 01-1-92 在 01-1-95 前面。

对于字符值, 依字母顺序显示—例如, A 第一, Z 最后。

对于空值, 升序排序时显示在最后, 降序排序时显示在最前面

2.7.3.1 示例

```
select
last_name, hire_date, salary, commission_pct
from employees order by commission_pct;
```

2.7.4 列号排序

可以使用投影的列的序号指定排序列, 但是不推荐此种做法。

2.7.4.1 示例

```
select
```

```
last_name, hire_date, salary, commission_pct  
from employees order by 2;
```

2.7.5用列别名排序

可以使用列的别名指定排序列。

2.7.5.1示例

```
select last_name as  
name, hire_date, salary, commission_pct from  
employees order by name;
```

2.7.6多列排序

多列排序：可以用多列排序查询结果。在 ORDER BY 子句中，多个指定的列名之间用逗号分开。如果想要对某个列倒序排序需则在该列名后面指定 DESC。

2.7.6.1示例：

```
select hire_date, salary from employees  
order by hire_date, salary desc;
```

2.7.7SELECT语句的执行顺序如下：

FROM 子句
WHERE 子句
SELECT 子句
ORDER BY 子句

2.8约束与排序小节练习

2.8.1创建一个查询，显示收入超过 \$12,000 的雇员的名字和薪水。

答案: `select last_name, salary from employees`

```
where salary > 12000;
```

2.8.2 创建一个查询,显示雇员号为 176 的雇员的名字和部门号。

答案： `select last_name, department_id from employees where employee_id = 176;`

2.8.3 显示所有薪水不在 5000 和 12000 之间的雇员的名字和薪水。

答案: `select last_name, salary from employees where salary not between 5000 and 12000;`

2.8.4 显示受雇日期在 2002 年 2 月 20 日 和 2007 年 5 月 1 日 之间的雇员的名字、岗位和受雇日期。按受雇日期顺序排序查询结果。

答案: `select last_name, job_id, hire_date from employees where hire_date between '22-2月-02' and '01-5月-07' order by hire_date;`

2.8.5 显示所有在部门 20 和 50 中的雇员的名字和部门号,并以名字按字母顺序排序。

答案： `select last_name, department_id from employees where department_id in (20, 50) order by last_name;`

2.8.6 列出收入在 \$5,000 和 \$12,000 之间，并且在部门 20 或 50 工作的雇员的名字和薪水。将列标题分别显示为 Employee 和 Monthly Salary。

答 案 : select last_name "Employee", salary
"Monthly Salary" from employees where salary
between 5000 and 12000 and department_id in
(20, 50);

2.8.7 显示每一个在 2004 年受雇的雇员的名字和受雇日期。

答 案 : select last_name , hire_date from
employees where hire_date like '%04';

2.8.8 显示所有没有主管经理的雇员的名字和工作岗位。

答案: select last_name, job_id from employees
where manager_id is null;

2.8.9 显示所有有佣金的雇员的名字、薪水和佣金。以薪水和佣金的降序排序数据。

答 案 : select last_name , salary ,
commission_pct from employees where
commission_pct is not null order by salary
desc , commission_pct desc;

2.8.10 显示所有名字中第三个字母是 a 的雇员的名字。

答案: `select last_name from employees where last_name like '_ _a%';`

2.8.11 显示所有名字中有一个 a 和一个 e 的雇员的名字。

答案: `select last_name from employees where last_name like '%a%' and last_name like '%e%';`

2.8.12 显示所有工作是销售代表(SA_REP)或者普通职员(ST_CLERK)，并且薪水不等于 \$2,500、\$3,500 或 \$7,000 的雇员的名字、工作和薪水。

答案: `select last_name, job_id, salary from employees where job_id in ('SA_REP', 'ST_CLERK') and salary not in (2500, 3500, 7000);`

2.8.13 显示所有佣金总计为 20% 的雇员的名字、薪水和佣金。

答案:

法一: `select last_name, salary, commission_pct from employees where commission_pct = 0.2;`

法二: `select last_name, salary, commission_pct from employees where commission_pct = .20;`

3 Oracle 函数

3.1 函数介绍

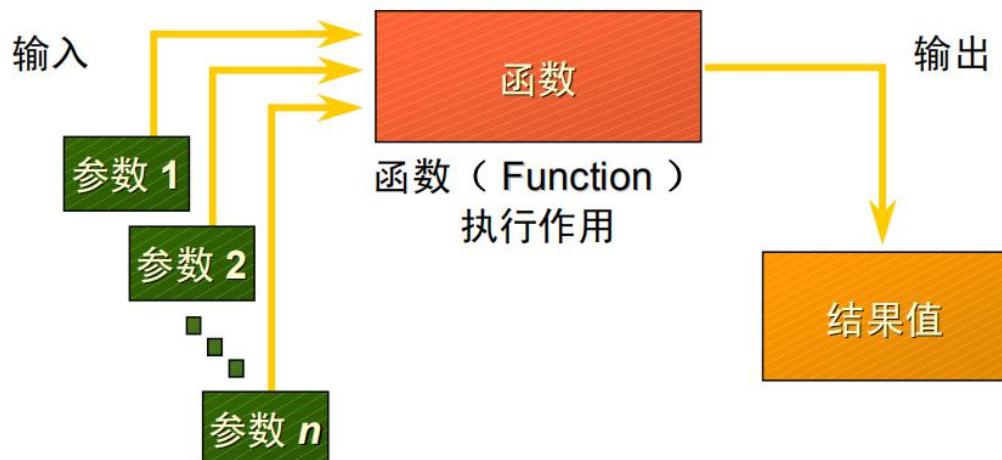
3.1.1 什么是函数

函数：是数据库产品中提供的能够处理查询结果的方法。

函数能够用于下面的目的：

- 执行数据计算
- 修改单个数据项
- 格式化显示的日期和数字
- 转换列数据类型
- 函数有输入参数，并且总有一个返回值。

SQL 函数

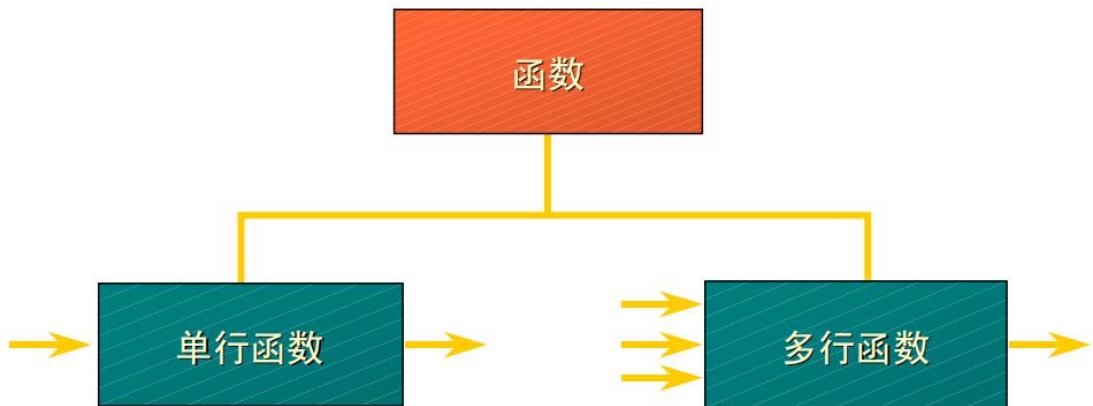


3.1.2 函数类型

单行函数：这些函数仅对单个行进行运算，并且每行返回一个结果。

多行函数(聚合函数)：这些函数能够操纵成组的行，每个行组给出一个结果，这些函数也被称为组函数。

SQL 函数的两种类型



3.1.3 函数语法

function_name(arg1,arg2,.....)

function_name: 是函数的名字。

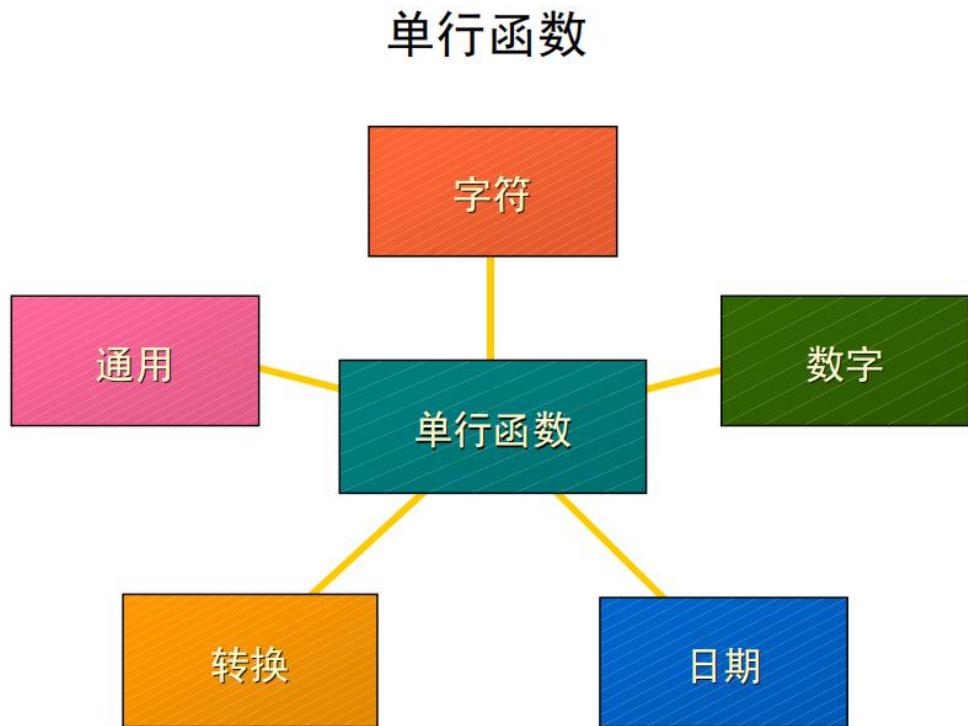
arg1, arg2: 是由函数使用的任意参数。参数可以是一个列名、用户提供的常数、变量值、或者一个表达式。

3.2 单行函数

3.2.1 单行函数的特性包括:

- 作用于每一个返回行，每行返回一个结果
- 可能需要一个或多个参数
- 可以修改结果集的数据类型
- 可以嵌套
- 可能返回一个与参数不同类型的数据值
- 能够用在 SELECT、WHERE 和 ORDER BY 子句中

3.2.2 单行函数分类



字符函数: 接受字符输入, 可以返回字符或者数字值

数字函数: 接受数字输入, 返回数字值

日期函数: 对 DATE 数据类型的值进行运算 (除了 MONTHS_BETWEEN 函数返回一个数字, 所有日期函数都返回一个 DATE 数据类型的值。)

转换函数: 从一个数据类型到另一个数据类型转换一个值

通用函数:

- NVL
- NVL2
- NULLIF
- COALESCE
- CASE
- DECODE

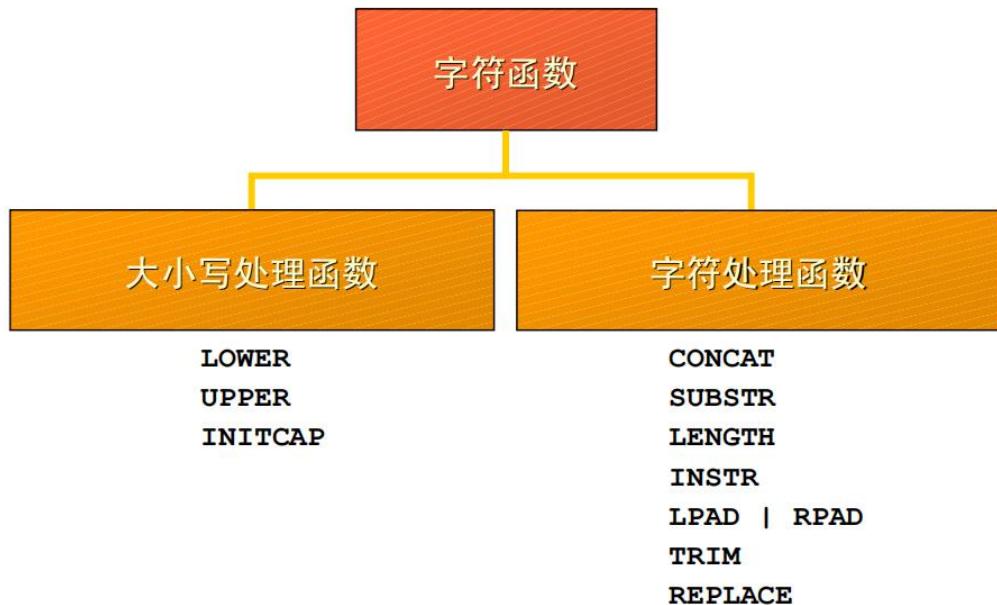
3.3 字符函数

字符函数: 单行字符函数接受字符数据作为输入, 既可以返回字符值也可以返回数字值。

3.3.1 字符函数分类

- 大小写处理函数
- 字符处理函数

字符函数



3.3.2 大小写处理函数

函数	结果
LOWER('SQL Course')	sql course
UPPER('SQL Course')	SQL COURSE
INITCAP('SQL Course')	Sql Course

LOWER: 转换大小写混合的字符串为小写字符串。

UPPER: 转换大小写混合的字符串为大写字符串。

INITCAP: 将每个单词的首字母转换为大写，其他字母为小写。

大小写处理函数需要一个参数，参数类型为字符串类型，返回一个字符串。

3.3.3示例一

查询员工表，使用“ The Job id for ”链接转换大写格式后的员工姓名，并使用“ is ”字符串链接他们工作 ID，要求将工作 ID 转换小写格式。修改列名为“EMPLOYEE DETAILS”。

```
select      'The job id for '
           || upper(last_name) || ' is ' || lower(job_id)
"EMPLOYEE DETAILS" from employees;
```

3.3.4示例二

显示雇员 higgins 的雇员号、姓名和部门号

```
select employee_id, last_name, department_id      from
employees          where      last_name      =
initcap('higgins');
```

3.4字符处理函数

dual 表: dual 是一张只有一个字段, 一行记录的表。dual 表也称之为'伪表', 因为他不存储主题数据。如果我们不需要从具体的表来取得表中数据, 而是单纯地为了得到一些我们想得到的信息, 并要通过 select 完成时, 就要借助 dual 表来满足结构化查询语言的格式。

函数	结果
CONCAT('Hello', 'World')	HelloWorld
SUBSTR('HelloWorld', 1, 5)	Hello
LENGTH('HelloWorld')	10
INSTR('HelloWorld', 'W')	6
LPAD(salary, 10, '*')	*****24000
RPAD(salary, 10, '*')	24000*****
TRIM('H' FROM 'HelloWorld')	elloWorld

CONCAT(arg1,arg2): 连接值在一起 (CONCAT 函数有两个输入参数)

arg1: 字符串类型。字符串拼接的值。

arg2: 字符串类型。字符串拼接的值。

SUBSTR(arg1,arg2,arg3): 截取子串。

arg1: 字符串类型。原字符串,

arg2: 整数类型。开始位置(开始位置可以是一个负数, -1 表示原串的最后一位, -2 则表示倒是第二位以此类推),

arg3: 整数类型。截取个数。

SUBSTR(arg1,arg2): 截取子串。

arg1: 字符串类型。原字符串。

arg2 开始位置(开始位置可以是一个负数, -1 表示原串的最后一位, -2 则表示倒是第二位以此类推)截取到末尾。

LENGTH(arg1): 以数字值显示一个字符串的长度。

arg1: 字符串类型。计算长度的字符串。

INSTR(arg1,arg2): 找到一个给定字符的数字位置。

arg1: 字符串类型。原字符串。

arg2: 字符串类型。查找内容。

INSTR(arg1,arg2,arg3,arg4): 指定查找位置以及出现的次数。

arg1: 字符串类型。原字符串。

arg2: 字符串类型。查找内容。

arg3: 整数类型。开始位置。

arg4: 整数类型。第几次出现。

LPAD(arg1,arg2,arg3): 用给定的字符左填充字符串到给定的长度。

arg1: 字符串类型。原字符串。

arg2: 整数类型。总长度。

arg3: 字符串类型。填充的子字符串。

RPAD(arg1,arg2,arg3): 用给定的字符右填充字符串到给定的长度。

arg1: 字符串类型。原字符串。

arg2: 整数类型。总长度。

arg3: 字符串类型。填充的子字符串。

TRIM(arg1): 从一个字符串中去除头(leading)或尾(trailing)或头尾两侧(both)的字符 (默认认为头尾两侧) 如果 trim_character 或 trim_source 是一个文字字符, 必须放在单引号中。

arg1 需要操作的字符串。FROM 为关键字。

格式 1: 需要去掉的内容 FROM 原字符串

格式 2: leading|trailing|both 需要去掉的内容 FROM 原字符串。

去掉头尾两侧方法一:

```
select trim('H' from HelloWorldH') from  
dual;
```

去掉头尾两侧方法二:

```
select trim(both 'H' from 'HelloWorldH')  
from dual;
```

去掉头:

```
select trim(leading 'H' from 'HelloWorldH')  
from dual;
```

去掉尾:

```
select trim(trailing 'H' from 'HelloWorldH')  
      from dual;
```

REPLACE(arg1,arg2,arg3): REPLACE 函数是用另外一个值来替代串中的某个值。

arg1: 字符串类型。原字符串。

arg2: 字符串类型。需要替换的子串。

arg3: 字符串类型。替换的内容。

3.4.1示例一

显示所有工作岗位名称从第 4 个字符位置开始包含字符串 REP 的雇员的信息，将雇员的姓和名连接显示在一起，还显示雇员名的长度，以及名字中字母 a 的位置。

```
select  
concat(first_name,last_name),length(last_na  
me),instr(last_name,'a') from employees where  
substr(job_id,4)='REP';
```

3.4.2示例二

显示名字是以 n 结束的雇员的数据，将雇员的姓和名连接显示在一起，还显示雇员名的长度，以及名字中字母 a 的位置。

```
select  
concat(first_name,last_name),length(last_na  
me),instr(last_name,'a') from employees where  
substr(last_name,-1)='n';
```

3.4.3示例三

将手机号中间四位用星号代替。

```
select  
replace('13622329860',substr('13622329860',
```

```
4,4),'*'*') from dual;
```

3.5 数字函数

- **ROUND:** 四舍五入指定小数的值

ROUND(45.926, 2)  **45.93**

- **TRUNC:** 截断指定小数的值

TRUNC(45.926, 2)  **45.92**

- **MOD:** 返回除法的余数

MOD(1600, 300)  **100**

ROUND(arg1,arg2): 四舍五入指定小数的值。

arg1: 数字类型。原数字。

arg2: 整数类型。小数点保留的位数，可以是一个负数。负数则表示指定整数的位置。

ROUND(arg1): 四舍五入保留整数。

arg1: 数字类型。原数字。

arg2: 整数类型。小数点保留的位数。

TRUNC(arg1,arg2): 截断指定小数的值，不做四舍五入处理。

arg1: 数字类型。原数字。

arg2: 整数类型。小数点保留的位数，可以使一个负数。负数则表示指定整数的位置。

TRUNC(arg1): 四舍五入保留整数。

arg1: 数字类型。原数字。

arg2: 整数类型。小数点保留的位数。

MOD(arg1,arg2): 取余。

arg1: 数字类型。被除数。

arg2: 数字类型。除数。

3.5.1示例一

计算所有是销售代表(SA REP)的雇员的工资被 5000 除后的余数。

```
select last_name, salary, mod(salary, 5000)  
from employees where job_id = 'SA REP';
```

3.6 日期处理

3.6.1 日期的使用

3.6.1.1 SYSDATE 函数

SYSDATE 是一个日期函数，它返回当前数据库服务器的日期和时间。

3.6.1.2 用日期计算

- 从日期加或者减一个数，结果是一个日期值
- 两个日期相减，得到两个日期之间的天数
- 用小时数除以 24，可以加小时到日期上

运算	结果	说明
date + number	日期	加一个天数到一个日期上
date - number	日期	从一个日期上减一个天数
date - date	天数	用一个日期减另一个日期
date + number/24	日期	加一个小时数到一个日期上

3.6.1.3 用日期做算术运算

3.6.1.3.1示例

显示所有在部门 90 中的雇员的名字和从业的周数。雇员的总工作时间以周计算。

```
select last_name, (sysdate-hire_date)/7  
from employees where department_id = 90;
```

3.6.2 日期函数

函数	说明
MONTHS_BETWEEN	两个日期之间的月数
ADD_MONTHS	加日历月到日期
NEXT_DAY	下个星期几是几号
LAST_DAY	指定月的最后一天
ROUND	四舍五入日期
TRUNC	截断日期

MONTHS_BETWEEN(date1,date2): 计算 date1 和 date2 之间的月数。其结果可以是正的也可以是负的。如果 date1 大于 date2, 结果是正的。反之，结果是负的。

date1: 日期类型。

date2: 日期类型。

ADD_MONTHS(date, n): 添加 n 个日历月到 date。n 的值必须是整数，但可以是负的。

date: 日期类型。

n: 整数

NEXT_DAY(date, 'char'): 计算在 date 之后的下一个周('char')的指定天的日期。char 的值可能是一个表示一天的数或者是一个字符串。如果使用数字表示星期，1 是从星期日开始。数字范围为：1-7。

date: 日期类型。

char: 数字或字符串。

LAST_DAY(date): 计算包含 date 的月的最后一天的日期。

date: 日期类型。

ROUND(date,'fmt'): 返回用格式化模式 fmt 四舍五入到指定单位的 date , 如果格式模式 fmt 被忽略, date 被四舍五入到最近的天。

date: 日期类型。

fmt: 字符串类型。

TRUNC(date, 'fmt'): 返回用格式化模式 fmt 截断到指定单位的带天的。如果格式模式 fmt 被忽略, date 被截断到最近的天。

date: 日期类型。

fmt: 字符串类型。

3.6.2.1示例一

查询所有受雇在 15 年 (180 个月) 以内的雇员的 employee_id, hire_date, 显示他们已被雇用的月, 从受雇日期开始加 6 个月的试用期后的日期, 受雇日期后的第一个星期五是几号, 以及受雇月的最后一天是几号。

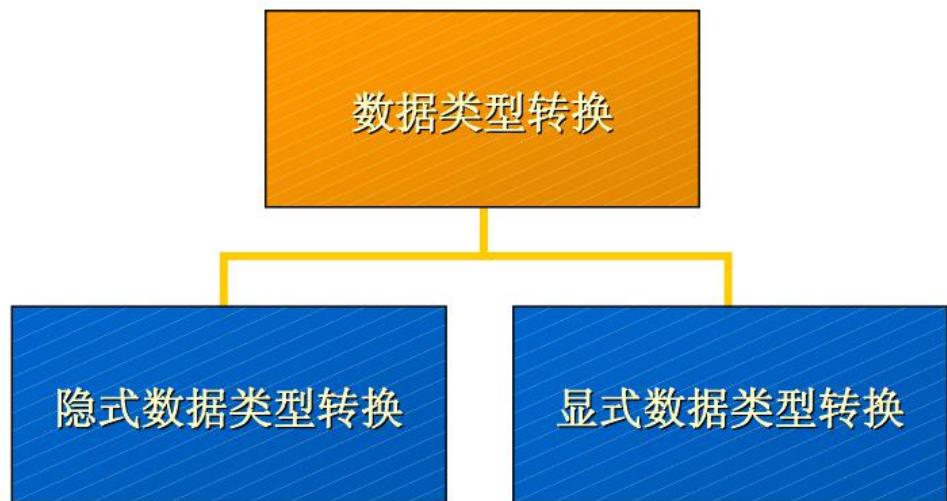
```
select  
employee_id ,hire_date,months_between(sysda  
te,hire_date),add_months(hire_date,6),next_  
day(hire_date,'星期五'),last_day(hire_date)  
from employees  
where  
months_between(sysdate,hire_date) < 180;
```

3.6.2.1.2示例二

查询受雇日期, 找出 2002 年开始工作的哪些人。用 ROUND 和 TRUNC 函数显示开始的月份。

```
select  
round(hire_date,'month'),trunc(hire_date,'m  
onth') from employees where hire_date like  
'%02';
```

3.7 数据类型转换



3.7.1 隐式数据类型转换

隐式转换：当源数据的类型和目标数据的类型不同的时候，如果没有转换函数，就会发生隐式转换，也称自动转换。

3.7.1.1 对于直接赋值转换

从	到
VARCHAR2 or CHAR	NUMBER
VARCHAR2 or CHAR	DATE
NUMBER	VARCHAR2
DATE	VARCHAR2

3.7.1.2 对于表达式赋值

从	到
VARCHAR2 or CHAR	NUMBER
VARCHAR2 or CHAR	DATE

3.7.1.3 隐式转换的问题

3.7.1.3.1 性能影响:

隐式转换的最大问题就是转换时会导致索引的无效，进而可能导致全表扫描。当表的数据量很大的时候，产生会很大的性能问题。比如说，VARCHAR2 和 NVARCHAR2 隐式数据类型转换导致的性能问题。

3.7.1.3.2 不便于阅读:

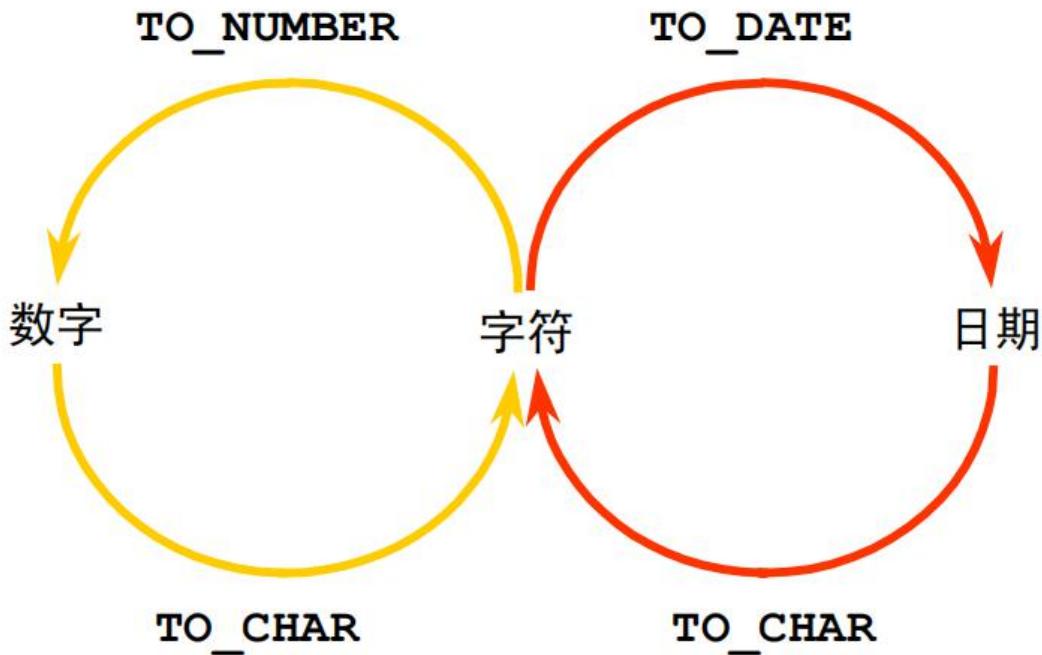
由于隐式转换使得数据库编程人员和 DBA 难以了解到究竟发生了怎样的类型转换，而且如果代码很多很长的话要查出错误就需要费很大的劲。

3.7.2 显示数据类型转换

通过数据库中的转换函数完成数据类型的转换。

3.7.3 转换函数

显式数据类型转换



TO_CHAR(arg1, 'fmt'): 将一个日期或者数字转换为字符类型。带格式化样式 fmt。

arg1: 数字或者日期类型。需要转换的数据。

fmt: 转换格式。

3.7.3.1 to_char 日期转换

日期格式模板的元素

YYYY	数字全写年
YEAR	年的拼写
MM	月的两数字值
MONTH	月的全名
MON	月的三字母缩写
DY	周中天的三字母缩写
DAY	周中天的全名
DD	月的数字天

元素	说明
SCC 或 CC	世纪, 带 - 服务器前缀 B.C. 日期
日期中的年 YYYY 或 SYYYY	年, 带 - 服务器前缀 B.C. 日期
YYY 或 YY 或 Y	年的最后 3、2 或 1 个数字
Y,YYY	年, 在这个位置带逗号
IYYY, IYY, IY, I	基于 ISO 标准的 4、3、2 或 1 位数字年
SYEAR 或 YEAR	拼写年; 带 - 服务器前缀 B.C. 日期
BC 或 AD	B.C.A.D.指示器
B.C.或 A.D.	带周期的 B.C./A.D.指示器
Q	四分之一年
MM	月: 两位数字值
MONTH	9 位字符长度的带空格填充的月的名字
MON	3 字母缩写的月的名字
RM	罗马数字月
WW 或 W	年或月的周
DDD 或 DD 或 D	年、月或周的天
DAY	9 位字符长度的带空格填充的天的名字
DY	3 字母缩写的天的名字
J	儒略日, 从公元前 4713 年 12 月 31 日开始的天数

时间格式模板元素

元素	说明
AM 或 PM	正午指示
A.M.或 P.M.	带句点的正午指示
HH 或 HH12 或 HH24	天的小时, 或小时(1-12), 或小时(0-23)
MI	分钟(0-59)
SS	秒(0-59)
SSSS	午夜之后的秒(0-86399)

- 日期的时间部分, 时间元素格式

HH24 : MI : SS AM	15 : 45 : 32 PM
-------------------	-----------------

- 加字符串, 将它们括在双引号中

DD "of" MONTH	12 of OCTOBER
---------------	---------------

- 数字前缀拼出数字

ddspth	fourteenth
--------	------------

其它格式

元素	说明
/.,	在结果中使用标点符号
"of the"	在结果中使用引文串

指定后缀来影响数字显示

元素	说明
TH	序数(例如, DDTH 显示为 4TH)
SP	拼写出数字(例如, DDSP 显示为 FOUR)
SPTH or THSP	拼写出序数(例如, DDSPTH 显示为 FOURTH))

3.7.3.1示例

显示所有雇员的名字和受雇日期, 受雇日期以 2007 年 8 月 10 日 12:00:00 AM 显示。

select

```
last_name,to_char(hire_date,'yyyy" 年 "mm" 月
"dd"日" hh:mi:ss am') from employees;
```

3.7.3.2 to_char 数字转换

数字格式模板

9	表示一个数
0	强制显示为零
\$	放置一个浮动美元符号
L	使用浮动本地货币符号
.	打印一个小数点
,	打印一个千位指示

FM: 代表去掉返回结果中的前后空格。

3.7.3.2.1示例

查询雇员 Whalen，显示他的薪水，在薪水前添加美元符号与千位符。

```
select
last_name, salary ,to_char(salary,'fm$999,99
9,999.00') from employees where last_name =
'Whalen';
```

3.7.3.3 to_number 字符串到数字转换

TO_NUMBER(‘arg1’ ,fmt’): 将字符串转换为数值型的格式。带格式化样式 fmt。

arg1: 字符串类型。需要转换的数据。

fmt: 转换格式。

9	表示一个数
0	强制显示为零
\$	放置一个浮动美元符号
L	使用浮动本地货币符号
.	打印一个小数点
,	打印一个千位指示

3.7.3.3.1示例

将¥34,346.56 转换为数字类型。

```
select          to_number('
```

```
¥34,346.56', '1999,999.99') from dual;
```

3.7.4 to_date 字符到日期转换

TO_DATE('arg1','fmt'): 将字符串转换为日期格式。带格式化样式 fmt。

arg1: 字符串类型。需要转换的数据。

fmt: 转换格式。

3.7.4.1.1示例

将 2019 年 3 月 9 日 11 点 30 分转换为 Date 类型。

```
select to_date('2019 年 3 月 9 日 11 点 30 分',
               'yyyy"年"MM"月"DD"日"HH"点"MI"分"') from dual;
```

3.7.5 函数嵌套

- 单行函数能够被嵌套任意层次
- 嵌套函数的计算是从最里层到最外层



3.7.5.1 示例

显示受雇日期 6 个月后的下一个星期五的日期。结果日期将应该是：星期，月，日，年。使用 Next 6 Month Review 作为列别名。结果按受雇日期排序。

```
select
  to_char(next_day(add_months(hire_date, 6), '星期五'), 'day,MM"月",DD"日",yyyy"年") as
  "Next 6 Month Review" from employees order by
```

```
hire_date;
```

3.8通用函数

通用函数：可用于任意数据类型，并且适用于空值。

- NVL(expr1, expr2)
- NVL2(expr1, expr2, expr3)
- NULLIF(expr1, expr2)
- COALESCE(expr1, expr2, ..., exprn)

函数	说明
NVL	转换空值为一个实际值
NVL2	如果 expr1 非空，NVL2 返回 expr2；如果 expr1 为空，NVL2 返回 expr3。参数 expr1 可以是任意数据类型。
NULLIF	比较两个表达式，如果相等返回空；如果不相等，返回第一个表达式
COALESCE	返回表达式列表中的第一个非空表达式

NVL(expr1, expr2) 函数：转换一个空值到一个实际的值。

expr1,expr2: 可用的数据类型可以是日期、字符和数字。两个参数的数据类型必须匹配。

expr1: 是包含空值的源值或者表达式。

expr2: 是用于转换空值的目的值。

3.8.1示例一

计算所有员工的年薪，如果有佣金包含佣金。

```
select last_name, salary, commission_pct,  
12*salary*nvl(commission_pct, 1)           from  
employees;
```

3.8.2示例二

计算雇员的年报酬，你需要用 12 乘以月薪，再加上年薪乘以佣金百分比。显示雇员的名字、薪水、佣金和计算完后的薪水，新的薪水列名为 AN_SAL。

```
select  
last_name, salary, commission_pct, 12*salary+1  
2*salary*nvl(commission_pct, 0)  an_sal  from
```

```
employees;
```

NVL2(expr1, expr2, expr3) 函数: NVL2 函数检查第一个表达式, 如果第一个表达式不为空, 那么 NVL2 函数返回第二个表达式; 如果第一个表达式为空, 那么第三个表达式被返回。

expr1: 是可能包含空的源值或表达式。

expr2: expr1 非空时的返回值。

expr3: expr1 为空时的返回值。

3.8.3示例三

查询雇员信息, 如果有佣金的显示 SAL+COMM 如果没有佣金则显示 SAL。

```
select
last_name, salary, commission_pct, nvl2(to_char(commission_pct), 'sal_comm', 'sal')      from
employees;
```

NULLIF(expr1, expr2)函数: 比较两个表达式, 如果相等, 函数返回空, 如果不相等, 函数返回第一个表达式。第一个表达式不能为 NULL。

expr1 是对于 expr2 的被比较原值

expr2 是对于 expr1 的被比较原值。(如果它不等于 expr1, expr1 被返回)。

3.8.4示例四

查询雇员, 显示他们的 first_name 与长度, 长度列命名为 expr1。last_name 与长度, 长度命名为 expr2。判断他们的 first_name 与 last_name 的长度, 如果长度相同返回空, 否则返回 first_name 的长度。判断结果列命名为 result。

```
select
first_name, length(first_name) "expr1" ,last_
name, length(last_name) "expr2" ,nullif(lengt
h(first_name), length(last_name)) "result"
from employees;
```

COALESCE(expr1, expr2, ... exprn) 函数：返回列表中的第一个非空表达式。
expr1 如果它非空，返回该表达式。
expr2 如果第一个表达式为空并且该表达式非空，返回该表达式。
exprn 如果前面的表达式都为空，返回该表达式。

3.8.5示例五

查询雇员表，如果 COMMISSION_PCT 值是非空，显示它。如果 COMMISSION_PCT 值是空，则显示 SALARY。如果 COMMISSION_PCT 和 SALARY 值都是空，那么显示 10。

```
select  
first_name,coalesce(commission_pct,salary,1  
0) from employees order by first_name;
```

3.9条件表达式

- 在 SQL 语句中提供 IF-THEN-ELSE 逻辑的使用
- 两种用法：
 - CASE 表达式
 - DECODE 函数

3.9.1CASE 表达式

使得 IF-THEN-ELSE 条件判断容易实现

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

CASE 表达式：CASE 表达式可以在 SQL 语句中使用 IF-THEN-ELSE 逻辑。如果没有 WHEN ... THEN 满足条件，并且 ELSE 子句存在，Oracle 返回 else_expr。否则，Oracle 返回 null。所有的表达式 (expr、comparison_expr 和 return_expr) 必须是相同的数据类型，

3.9.1.1 示例

查询雇员，显示 last_name,job_id,salary 如果 JOB_ID 是 IT_PROG，薪水增加 10%；如果 JOB_ID 是 ST_CLERK，薪水增加 15%；如果 JOB_ID 是 SA REP，薪水增加 20%。

对于所有其他的工作角色，不增加薪水。

```
select      last_name, job_id, salary, CASE
job_id WHEN 'IT_PROG' THEN salary * 1.1 WHEN
'ST_CLERK' THEN salary * 1.15 WHEN 'SA REP'
THEN salary * 1.2 END from employees;
```

3.9.2 DECODE 函数

使得 CASE 或者 IF-THEN-ELSE 条件判断容易实现：

```
DECODE(col|expression, search1, result1
      [, search2, result2,...,]
      [, default])
```

DECODE 函数：DECODE 函数以一种类似于在多种语言中使用的 IF-THEN-ELSE 逻辑的方法判断一个表达式。DECODE 函数在比较表达式 (expression) 和每个查找 (search) 值后，如果表达式与查找相同，返回结果。如果省略默认值，当没有查找值与表达式相匹配时返回一个空值。

3.9.2.1 示例

使用 DECODE 函数完成（显示 last_name, job_id, salary 如果 JOB_ID 是 IT_PROG，薪水增加 10%；如果 JOB_ID 是 ST_CLERK，薪水增加 15%；如果 JOB_ID 是 SA REP，薪水增加 20%。对于所有其他的工作角色，不增加薪水。）

```
select      last_name, job_id, salary,
decode(job_id, 'IT_PROG', salary*1.1, 'ST_CLER
K', salary*1.15, 'SA REP', salary * 1.2)  from
employees;
```

3.10 单行函数小节练习

3.10.1 写一个查询显示当前日期，列标签显示为 Date。

答案：

```
select sysdate "Date" from dual;
```

3.10.2 对每一个雇员，显示 **employee number**、**last_name**、**salary** 和 **salary** 增加 15%，并且表示成整数，列标签显示为 **New Salary**。

答案：

```
select  
employee_id, last_name, salary, round(salary*1.  
15) "New Salary" from employees;
```

3.10.3 查询所有名字开始字母是 J、A 或 M 的雇员，用首字母大写，其它字母小写显示雇员的 **last names**，显示名字的长度，用雇员的 **last_names** 排序结果。

答案：

```
select  
initcap(last_name), length(last_name)      from  
employees  where  last_name  like  'J%'  or  
last_name  like  'M%'  or  last_name  like  'A%'  
order by last_name;
```

3.10.4 对每一个雇员，显示其的 last name，并且计算从雇员受雇日期到今天的月数，列标签 MONTHS_WORKED。接受雇月数排序结果，四舍五入月数到最靠近的整数月。

答案：

```
select  
last_name, round(months_between(sysdate,hire  
_date)) month_worked from employees order by  
months_between(sysdate,hire_date);
```

3.10.5 写一个查询对每个雇员做计算：<雇员的 last name> earns <salary> monthly but wants <3 倍 salary>。要求薪水中包含\$与千位符。列标签 Dream Salaries。

答案：

```
select      last_name      ||      'earns'      ||  
to_char(salary,'fm$999,999.00')  ||  'monthly  
but                  wants'          ||  
to_char(3*salary,'fm$999,999.00') as "Dream  
Salaries" from employees;
```

3.10.6 创建一个查询显示所有雇员的 last name 和 salary。

对 salary 格式化为 15 个字符长度，用 \$ 左填充，列标签 SALARY。

答案：

```
select last_name, lpad(salary ,15,'$')
salary from employees;
```

3.10.7 显示每一个雇员的 last name、hire date 和 salary 和入职日期，该日期是服务六个月后的第一个星期一，列标签 REVIEW。格式化日期显示看起来象“2019 年 3 月 9 日 星期六”的样子。

答案：

```
select
last_name,hire_date,salary ,to_char(next_da
y(add_months(hire_date,6),'星期一'),'yyyy"年
"MM"月"DD"日"DAY') REVIEW from employees;
```

3.10.8 显示 last name、hire date 和 雇员开始工作的星期，列标签 DAY，用星期一作为周的起始日排序结果。

答案：

```
select
last_name,hire_date,to_char(hire_date,'DAY')
day from employees order by to_char(hire_date
-1,'d');
```

3.10.9 创建一个查询显示雇员的 last names 和 commission (佣金) 比率。如果雇员没有佣金，显示 “No Commission”，列标签 COMM。

答案：

```
select  
last_name, nvl(to_char(commission_pct), 'No  
Commission') from employees;
```

3.10.10 用 DECODE 函数，写一个查询，按照下面的数据显示所有雇员的基于 JOB_ID 列值的级别。

工作	级别
AD_PRES	A
ST_MAN	B
IT_PROG	C
SA REP	D
ST_CLERK	E
不在上面的	0

答案：

```
select  
job_id, decode(job_id, 'AD_PRES', 'A', 'ST_MAN',  
'B', 'IT_PROG', 'C', 'SA REP', 'D', 'ST_CLERK', '  
E', '0') from employees;
```

3.10.11 用 CASE 语法，实现前面的查询。

答案：

```

select job_id, CASE job_id WHEN 'AD_PRES'
THEN 'A' WHEN 'ST_MAN' THEN 'B' WHEN 'IT_PROG'
THEN 'C' WHEN 'SA REP' THEN 'D' WHEN 'ST_CLERK'
THEN 'E' ELSE '0' END from employees;

```

四、 多表查询

1 什么是多表查询

多表查询：当查询的数据并不是来源一个表时，需要使用**多表链接操作**完成查询。根据不同表中的数据之间的关系查询相关联的数据。



多表链接方式：

内连接：连接两个表，通过相等或不等判断链接列，称为内连接。在内连接中典型的联接运算有 = 或 <> 之类的比较运算符。包括等值联接和自然联接

等值连接

非等值连接

自连接

SQL99: 交叉链接(CROSS JOIN)

SQL99: 内连接(INNER JOIN)

SQL99: 自然链接(NATURAL JOIN)

外连接：在两个表之间的连接，返回内连接的结果，同时还返回不匹配行的左（或

右) 表的连接, 称为左(或右)连接。返回内连接的结果, 同时还返回左和右连接, 称为全连接。

左外链接
右外链接
全外链接

子查询: 当一个查询是另一个查询的条件时, 称之为子查询。

2 笛卡尔乘积

2.1 什么是笛卡尔乘积

笛卡尔乘积是指在数学中, 两个集合 X 和 Y 的笛卡尔积 (Cartesian product), 又称直积, 表示为 $X * Y$, 第一个对象是 X 的成员而第二个对象是 Y 的所有可能有序对的其中一个成员。

2.2 如何避免出现笛卡尔乘积

当一个连接条件无效或被遗漏时, 其结果是一个笛卡尔乘积 (Cartesian product), 其中所有行的组合都被显示。第一个表中的所有行连接到第二个表中的所有行。一个笛卡尔乘积会产生大量的行, 其结果没有什么用。应该在 WHERE 子句中始终包含一个有效的连接条件。

2.3 示例

```
select * from employees,departments;
```

3 多表连接语法

3.1 语法结构

使用一个连接从多个表中查询数据。

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

- 在 WHERE 子句中写连接条件。
- 当多个表中有相同的列名时, 将表名或者表的别名作为列名的前缀。

3.2 定义连接

当数据从多表中查询时, 要使用连接 (join) 条件。一个表中的行按照存在于相应列中

的值被连接到另一个表中的行。

3.3 原则

- 在写一个连接表的 SELECT 语句时, 在列名前面用表名或者表别名可以使语义清楚, 并且加快数据库访问。
- 为了连接 n 个表在一起, 你最少需要 $n-1$ 个连接条件。例如, 为了连接 4 个表, 最少需要 3 个连接条件。

4 等值连接

4.1 什么等值连接

等值连接也被称为简单连接 (simple joins) 或内连接 (inner joins)。是通过等号来判断连接条件中的数据值是否相匹配。

4.2 抢择矩阵 (decision matrix)

是通过行与列来分析一个查询的方式。

例如, 如果你想显示同一个部门中所有名字为 Taylor 的雇员的名字和部门名称, 可以写出下面的决策矩阵:

投影列	源表	条件
last_name	employees	last_name = 'Taylor'
department_name	departments	employees.department_id = departments.department_id

4.2.1示例一

查询所有雇员名字以及他们所在的部门名称。

```
select last_name, department_name from
employees , departments where
employees.department_id =
departments.department_id;
```

4.3 使用 AND 操作符附加搜索条件

除连接之外, 还可以要求用 WHERE 子句在连接中限制一个或多个表中的行。

4.3.1示例二

显示同一个部门中所有名字为 Taylor 的雇员的名字和部门号。

```
select    last_name,department_name    from  
employees      ,      departments      where  
employees.department_id      =  
departments.department_id  and  last_name  =  
'Taylor';
```

4.4使用表别名

使用表别名简化查询语句的长度。

4.4.1表别名

可以使用表别名代替表名。就象列别名给列另一个名字一样。表别名有助于保持 SQL 代码较小，因此使用的存储器也少。

4.4.2使用表别名原则

- 表别名最多可以有 30 个字符，但短一些更好。
- 如果在 FROM 子句中表别名被用于指定的表，那么在整个 SELECT 语句中都可以使用表别名。
- 表别名应该是有意义的。
- 表别名只对当前的 SELECT 语句有效。

4.4.3示例

使用表别名方式改写显示同一个部门中所有名字为 Taylor 的雇员的名字和部门号。

```
select    em.last_name,de.department_name  
from    employees    em,departments    de    where  
em.department_id    =    de.department_id    and  
em.last_name = 'Taylor';
```

4.5 多于两个表的连接

为了连接 n 个表，你最少需要 $n-1$ 个连接条件。例如，为了连接 3 个表，最少需要两个连接。

4.5.1示例一

查询每个雇员的 last name、departmentname 和 city(city 来源于 locations 表)。

```
select  
    em.last_name, de.department_name      , lo.city  
from employees em ,departments de,locations lo  
where em.department_id = de.department_id and  
de.location_id = lo.location_id;
```

4.5.2示例二

查询 Taylor 的雇员 ID、部门名称、和工作的城市。

```
select          em.employee_id,  
    em.last_name, de.department_name      , lo.city  
from employees em ,departments de,locations lo  
where em.department_id = de.department_id and  
de.location_id      =      lo.location_id      and  
em.last_name = 'Taylor';
```

5 非等值连接

5.1非等值连接

一个非等值连接是一种不使用相等(=)作为连接条件的查询。如!=、>、<、>=、<=、BETWEEN AND 等都是非等值链接的条件判断。

5.2 创建案例表

```
create table JOB_GRADES  
(  
    lowest_sal NUMBER,  
    highest_sal NUMBER,  
    gra           VARCHAR2(10)  
)
```

	LOWEST_SAL	HIGHEST_SAL	GRA	ROWID
1	1000	2999	A	...
2	3000	5999	B	...
3	6000	9999	C	...
4	10000	14999	D	...
5	15000	24999	E	...
6	25000	40000	F	...
▶				...

5.3示例

查询所有雇员的薪水级别。

```
select      em.last_name,em.salary,gr.gra  
from   employees  em  ,job_grades  gr  where  
em.salary    between     gr.lowest_sal      and  
gr.highest_sal;
```

6 自连接

6.1什么是自连接

使用一个表连接它自身的操作。

6.2示例

查询每个雇员的经理的名字以及雇员的名字。

```
select

worker.last_name,manager.last_name      from
employees  worker,employees  manager  where
worker.manager_id = manager.employee_id;
```

7 外连接(OUTER JOIN)

7.1什么是外连接

外连接是指查询出符合连接条件的数据同时还包含孤儿数据。左外链接包含左表的孤儿数据，右外连接包含右表的孤儿数据，全外连接包含两个表中的孤儿数据。

7.2孤儿数据(Orphan Data)

孤儿数据是指被连接的列的值为空的数据。

7.3外连接类型

左外(LEFT OUTER JOIN): 包含左表的孤儿数据。

右外(RIGHT OUTER JOIN): 包含右表的孤儿数据。

全外(FULL OUTER JOIN): 包含两个表中的孤儿数据。

7.4SQL99 中的外连接

SQL99 外连接语法格式:

用 LEFT OUTER JOIN | RIGHT OUTER JOIN | FULL OUTER JOIN 定义连接类型，用 ON 子句创建连接条件。

7.4.1左外链接(LEFT OUTER JOIN)

7.4.1.1 示例

用左外链接查询雇员名字以及他们所在的部门名称，包含那些没有部门的雇员。

```
select e.last_name,d.department_name from
employees e left outer join departments d on
e.department_id = d.department_id;
```

7.4.2右外链接(RIGHT OUTER JOIN)

7.4.2.1示例

用右外链接查询雇员名字以及他们所在的部门名称，包含那些没有雇员的部门。

```
select e.last_name,d.department_name from
employees e right outer join departments d on
e.department_id = d.department_id;
```

7.4.3全外链接(FULL OUTER JOIN)

7.4.3.1示例

查询所有雇员和部门，包含那些没有雇员的部门以及没有部门的雇员。

```
select e.last_name,d.department_name from
employees e full outer join departments d on
e.department_id = d.department_id;
```

7.5Oracle扩展的外连接

在 Oracle 数据库中对外连接中的左外与右外连接做了扩展，可以简化外连接的语法。通过在连接条件的后侧使用(+)来表示是否显示孤儿数据，有(+)表示不显示孤儿数据而另一侧则显示孤儿数据。**但是该种写法仅能在 Oracle 数据库中使用。**

7.5.1示例一

查询雇员名字以及他们所在的部门名称，包含那些没有雇员的部门。

```
select e.last_name,d.department_name from
employees e ,departments d where
e.department_id(+) = d.department_id;
```

7.5.2示例二

查询雇员名字以及他们所在的部门名称，包含那些没有部门的雇员。

```
select e.last_name, d.department_name from
employees      e      ,departments      d      where
e.department_id = d.department_id(+);
```

8 SQL99 中的交叉连接

- CROSS JOIN 子句导致两个表的交叉乘积
- 该连接和两个表之间的笛卡尔乘积是一样的

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

8.1示例

查询 Employees 表与 Departments 表的笛卡尔乘积。

```
select * from employees cross join
departments;
```

9 SQL99 中的自然连接(NATURAL JOIN)

- NATURAL JOIN 子句基于两个表之间有相同名字的所有列。
- 它从两个表中选择在所有的匹配列中有相等值的行。
- 如果有相同名字的列的数据类型不同，返回一个错误。

9.1使用自然连接需要注意

- 1.如果做自然连接的两个表的有多个字段都满足有相同名称个类型，那么他们会被作为自然连接的条件。
- 2.如果自然连接的两个表仅是字段名称相同，但数据类型不同，那么将会返回一个错误。
- 3.由于 oracle 中可以进行这种非常简单的 natural join，我们在设计表时对具有相同含义的字段需要考虑到使用相同的名字和数据类型。

9.2示例

查询部门 ID， 部门名称以及他们所在的城市。

```
select
```

```
d.department_id,d.department_name,l.city  
from departments d natural join locations l;
```

9.3用 USING 子句创建连接

- 当有多个列匹配时，用 USING 子句匹配唯一的列。
- 如果某列在 USING 中使用，那么在引用该列时不要使用表名或者别名。
- NATURAL JOIN 和 USING 子句是相互排斥的。

9.3.1示例

查询 location_id 为 1800 的部门名称以及他们所在的城市名称，指定 location_id 为连接列。

```
select      d.department_name,l.city      from  
departments      d          join      locations      l  
using(location_id)  where location_id = 1800;
```

10 SQL99 中的内连接(INNER JOIN)

内连接(INNER JOIN): 内连接通过 INNER JOIN 来建立两个表的连接。在内连接中使用 INNER JOIN 作为表的连接，用 ON 子句给定连接条件。INNER JOIN 语句在性能上其他语句没有性能优势。

10.1示例

查询雇员 id 为 202 的雇员名字，部门名称，以及工作的城市。

等值连接:

```
select e.last_name,d.department_name,l.city  
from employees e,departments d ,locations l  
where e.department_id = d.department_id and  
d.location_id = l.location_id and  
e.employee_id = 202;
```

内连接:

```
select e.last_name,d.department_name,l.city
```

```
from employees e inner join departments d on
e.department_id = d.department_id inner join
locations l on d.location_id = l.location_id
where e.employee_id = 202;
```

在内连接中使用 USING 子句定义等值连接

```
select e.last_name,d.department_name,l.city
from employees e inner join departments d
using(department_id) inner join locations l
using(location_id) where e.employee_id = 202;
```

11 多表查询小节练习

11.1写一个查询显示所有雇员的 last name、department number、and department name。

答案(等值):

```
select
e.last_name,e.department_id,d.department_na
me from employees e ,departments d where
e.department_id = d.department_id;
```

答案(内连接):

```
select
e.last_name,e.department_id,d.department_na
me from employees e inner join departments d
on(e.department_id = d.department_id);
```

11.2查询部门编号 80 中的所有工作岗位的唯一列表，在输出中包括部门编号、地点编号。

答案:

```
select distinct e.job_id,d.location_id
from employees e,departments d
where e.department_id = d.department_id and
e.department_id = 80;
```

11.3写一个查询显示所有有佣金的雇员的 **last name**、**department name**、**location ID** 和城市。

答案:

```
select
e.last_name,d.department_name,l.location_id,
l.city
from employees e,departments
d,locations l
where e.department_id =
d.department_id and d.location_id =
l.location_id and e.commission_pct is not
null;
```

11.4显示所有在其 **last names** 中有一个小写 a 的雇员的 **last name** 和 **department name**。

答案:

```
select e.last_name,d.department_name from
employees e,departments d
where
```

```
e.department_id      =      d.department_id      and  
e.last_name like '%a%';
```

11.5 使用内连接写一个查询显示那些工作在 **Toronto** 的所有雇员的 **last name**、**job**、**department number** 和 **department name**。

答案：

```
select  
e.last_name,e.job_id,e.department_id,d.depa  
rtment_name from employees e inner join  
departments d on(e.department_id      =  
d.department_id) inner join locations l  
on(d.location_id      =      l.location_id) where  
lower(l.city) ='toronto';
```

11.6 显示雇员的 **last name** 和 **employee number** 连同他们的经理的 **last name** 和 **manager number**。列标签分别为 **Employee**、**Emp#**、**Manager** 和 **Mgr#**。

答案：

```
select          emp.last_name  
"Employee",emp.employee_id  
"Emp#",manager.last_name  
"Manager" ,manager.employee_id "Mgr#" from  
employees    emp   ,employees manager where
```

```
emp.manager_id = manager.employee_id;
```

11.7查询所有雇员的经理包括 King，他没有经理。显示雇员的名字、雇员 ID、经理名、经理 ID、用雇员号排序结果。

答案：

```
select emp.last_name  
      , "Employee", emp.employee_id  
      , "Emp#", manager.last_name  
      , "Manager" , manager.employee_id "Man#" from  
employees emp left outer join employees  
manager          on (emp.manager_id  
                     =  
                     manager.employee_id);
```

11.8创建一个查询显示所有与被指定雇员工作在同一部门的雇员(同事)的 last names、department numbers。给每列一个适当的标签。

答案：

```
select e.last_name, e.department_id from  
employees e ,employees c where e.department_id  
= c.department_id and e.employee_id <>  
c.employee_id;
```

11.9显示 JOB_GRADES 表的结构。创建一个查询显示所有雇员的 name、job、department name、salary 和 grade。

答案：

```
select e.last_name, e.job_id, d.department_name, e.salary, j.grade
  from employees e, departments d, job_grades j
 where e.department_id = d.department_id
   and e.salary between j.lowest_sal and j.highest_sal;
```

11.10 创建一个查询显示那些在雇员 **Davies** 之后入本公司工作的雇员的 **name** 和 **hire date**

答案:

```
select e.last_name, e.hire_date
  from employees e, employees d
 where d.last_name = 'Davies' and d.hire_date < e.hire_date;
```

11.11 显示所有雇员的 **names** 和 **hire dates**, 他们在他们的经理之前进入本公司, 连同他们的经理的名字和受雇日期一起显示。列标签分别为 **Employee**、**Emp Hired**、**Manager** 和 **Mgr Hired**。

答案:

```
select e.last_name, e.hire_date, m.last_name, m.hire_date
  from employees e, employees m
 where e.manager_id = m.employee_id and e.hire_date < m.hire_date;
```

五、组函数(聚合函数)

1 组函数介绍

1.1 什么是组函数

组函数操作行集，给出每组的结果。组函数不象单行函数，组函数对行的集合进行操作，对每组给出一个结果。这些集合可能是整个表或者是表分成的组。

EMPLOYEES

DEPARTMENT_ID	SALARY
90	24000
90	17000
90	17000
60	9000
60	6000
60	4200
50	5800
50	3500
50	3100
50	2600
50	2500
80	10500
80	11000
80	8600
	7000
10	4400

20 rows selected.

在 EMPLOYEES 表中的最高薪水

MAX(SALARY)
24000

1.2 组函数与单行函数区别

单行函数对查询到每个结果集做处理，而组函数只对分组数据做处理。

单行函数对每个结果集返回一个结果，而组函数对每个分组返回一个结果。

1.3 组函数的类型

- AVG 平均值
- COUNT 计数
- MAX 最大值
- MIN 最小值
- SUM 合计

1.4 组函数的语法

```
SELECT      [column,] group function(column), ...
FROM        table
[WHERE      condition]
[GROUP BY   column]
[ORDER BY   column];
```

1.5 使用组函数的原则

- 用于函数的参数的数据类型可以是 CHAR、VARCHAR2、NUMBER 或 DATE。
- 所有组函数忽略空值。为了用一个值代替空值，用 NVL、NVL2 或 COALESCE 函数。

2 组函数的使用

2.1 使用 AVG 和 SUM 函数

AVG(arg)函数：对分组数据做平均值运算。
arg:参数类型只能是数字类型。

SUM(arg)函数：对分组数据求和。
arg:参数类型只能是数字类型。

2.1.1示例

求雇员表中的的平均薪水与薪水总额。

```
select avg(salary) , sum(salary) from
employees;
```

2.2 使用 MIN 和 MAX 函数

MIN(arg)函数：求分组中最小数据。
arg:参数类型可以是字符、数字、日期。

MAX(arg)函数：求分组中最大数据。
arg:参数类型可以是字符、数字、日期。

2.2.1示例

求雇员表中的最高薪水与最低薪水。

```
select      min(salary),max(salary)      from  
employees;
```

2.3使用 COUNT 函数

COUNT 函数：返回一个表中的行数。

COUNT 函数有三种格式：

- COUNT(*)
- COUNT(expr)
- COUNT(DISTINCT expr)

2.3.1COUNT(*)

返回表中满足 SELECT 语句标准的行数，包括重复行，包括有空值列的行。如果 WHERE 子句包括在 SELECT 语句中，COUNT(*) 返回满足 WHERE 子句条件的行数。

2.3.1.1 示例一

返回查询结果的总条数。

```
select count(*) from employees;
```

2.3.2COUNT(expr)函数

返回在列中的由 expr 指定的非空值的数。

2.3.2.1 示例二

显示部门 80 中有佣金的雇员人数。

```
select      count(commission_pct)      from  
employees e where e.department_id = 80;
```

2.3.3COUNT(DISTINCT expr):

使用 DISTINCT 关键字禁止计算在一列中的重复值。

2.3.3.1 示例三

显示 EMPLOYEES 表中不重复的部门数。

```
select count (distinct department_id) from  
employees ;
```

2.4 组函数和 Null 值

所有组函数忽略列中的空值。

在组函数中使用 NVL 函数来处理空值。

2.4.1示例一

计算有佣金的员工的佣金平均值。

```
select      avg (commission_pct)      from  
employees;
```

2.4.2示例二

计算所有员工的佣金的平均值。

```
select  avg (nvl (commission_pct, 0))  from  
employees;
```

3 创建数据组(GROUP BY)

3.1 什么是创建数据组

可以根据需要将查询到的结果集信息划分为较小的组，用 GROUP BY 子句实现。

EMPLOYEES

DEPARTMENT_ID	SALARY
10	4400
20	13000
20	6000
50	5800
50	3500
50	3100
50	2500
50	2600
60	9000
60	6000
60	4200
80	10500
80	8600
80	11000
90	24000
90	17000

20 rows selected.

4400
9500 在
EMPLOYEES
3500 表中
每个
部门的
平均
薪水
6400
10033
10033

DEPARTMENT_ID	AVG(SALARY)
10	4400
20	9500
50	3500
60	6400
80	10033.3333
90	19333.3333
110	10150
	7000

3.2 GROUP BY 子句语法

```
SELECT      column, group_function(column)
FROM        table
[WHERE      condition]
[GROUP BY   group_by_expression]
[ORDER BY   column];
```

用 GROUP BY 子句划分表中的行到较小的组中

GROUP BY 子句: GROUP BY 子句可以把表中的行划分为组。然后可以用组函数返回每一组的摘要信息。

3.3 使用分组原则

- 如果在 SELECT 子句中包含了组函数，就不能选择单独的结果，除非单独的列出现在 GROUP BY 子句中。如果未能在 GROUP BY 子句中包含一个字段列表，你会收到一个错误信息。
- 使用 WHERE 子句，你可以在划分行成组以前过滤行。
- 在 GROUP BY 子句中必须包含列。
- 在 GROUP BY 子句中你不能用列别名。
- 默认情况下，行以包含在 GROUP BY 列表中的字段的升序排序。可以用 ORDER BY 子句覆盖这个默认值。

3.4 GROUP BY 子句的使用

我们可以根据自己的需要对数据进行分组，在分组时，只要将需要做分组的列的列名添加到 GROUP BY 子句后侧就可以。GROUP BY 列不必在 SELECT 列表中。

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

3.4.1示例一

求每个部门的平均薪水。

```
select department_id , avg(salary) from
employees e group by e.department_id;
```

3.5 多于一个列的分组

EMPLOYEES

DEPARTMENT_ID	JOB_ID	SALARY
90	AD_PRES	24000
90	AD_VP	17000
90	AD_VP	17000
60	IT_PROG	9000
60	IT_PROG	6000
60	IT_PROG	4200
50	ST_MAN	5800
50	ST_CLERK	3500
50	ST_CLERK	3100
50	ST_CLERK	2600
50	ST_CLERK	2500
80	SA_MAN	10500
80	SA_REP	11000
80	SA_REP	8600
20	MK_REP	6000
110	AC_MGR	12000
110	AC_ACCOUNT	8300

20 rows selected.

在 EMPLOYEES
表中
先按照部门，
再按工作岗位
分组
相加薪水

DEPARTMENT_ID	JOB_ID	SUM(SALARY)
10	AD_ASST	4400
20	MK_MAN	13000
20	MK_REP	6000
50	ST_CLERK	11700
50	ST_MAN	5800
60	IT_PROG	19200
80	SA_MAN	10500
80	SA_REP	19600
90	AD_PRES	24000
90	AD_VP	34000
110	AC_ACCOUNT	8300
110	AC_MGR	12000
	SA_REP	7000

13 rows selected.

3.5.1示例一

显示在每个部门中付给每个工作岗位的合计薪水的报告。

```
select          department_id,job_id,
sum(salary) from        employees      group      by
```

```
department_id, job_id order by department_id;
```

3.6 GROUP BY 子句的执行顺序

先进行数据查询，在对数据进行分组，然后执行组函数。

3.7 非法使用 Group 函数的查询

- 在 SELECT 列表中的任何列必须在 GROUP BY 子句中。
- 在 GROUP BY 子句中的列或表达式不必在 SELECT 列表中。

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

```
SELECT department_id, COUNT(last_name)
      *
ERROR at line 1:
ORA-00937: not a single-group group function
```

列未包含在 GROUP BY 子句中

3.8 约束分组结果

3.8.1 什么是 HAVING 子句

HAVING 语句通常与 GROUP BY 语句联合使用，用来过滤由 GROUP BY 语句返回的记录集。

HAVING 语句的存在弥补了 WHERE 关键字不能与聚合函数联合使用的不足。

3.8.2 HAVING 子句语法

```
SELECT    department_id, MAX(salary)
FROM      employees
GROUP BY department_id
HAVING    MAX(salary)>10000 ;
```

3.8.3示例一

显示那些最高薪水大于 \$10,000 的部门的部门号和最高薪水。

```
select e.department_id,max(e.salary) from
employees e group by e.department_id having
max(e.salary) > 10000;
```

3.8.4示例二

查询那些最高薪水大于 \$10,000 的部门的部门号和平均薪水。

```
select e.department_id,avg(e.salary) from
employees e group by e.department_id having
max(e.salary) > 10000;
```

3.9嵌套组函数

在使用组函数时我们也可以根据需要来做组函数的嵌套使用。

3.9.1示例

显示部门中的最大平均薪水。

```
select max(avg(e.salary)) from employees
e group by e.department_id;
```

4 组函数小节练习

4.1组函数在多行上计算，对每个组产生一个结果。True/False

答案: True

4.2组函数在计算中包含空值。True/False

答案: False 组函数会忽略空值，如果需要空值参与计算，需要使用 nvl 函数处理空值。

4.3在分组计算中，**WHERE** 子句对行的限制在计算的前面。

True/False

答案: True

4.4显示所有雇员的最高、最低、合计和平均薪水，列标签分别为：

Maximum、**Minimum**、**Sum** 和 **Average**。四舍五入结果为最近的整数。

答案:

```
select  
max(salary),min(salary),sum(salary),avg(sal  
ary) from employees;
```

4.5修改上题显示每中工作类型的最低、最高、合计和平均薪水。

答案:

```
select  
max(salary),min(salary),sum(salary),avg(sal  
ary) from employees group by job_id;
```

4.6写一个查询显示每一工作岗位的人数。

答案:

```
select job_id, count(*) from employees  
group by job_id;
```

4.7确定经理人数，不需要列出他们，列标签是 **Number of Managers**。

答案:

```
select count(distinct manager_id) from
```

```
employees ;
```

4.8写一个查询显示最高和最低薪水之间的差。列标签是
DIFFERENCE。

答案:

```
select max(salary) - min(salary) from  
employees;
```

4.9显示经理号和经理付给雇员的最低薪水。排除那些经理未知的人。排除最低薪水小于等于 \$6,000 的组。按薪水降序排序输出。

答案:

```
select e.manager_id ,min(e.salary) from  
employees e where e.manager_id is not null  
group by e.manager_id having min(e.salary) >  
6000 order by min(e.salary) desc;
```

4.10写一个查询显示每个部门的名字、地点、人数和部门中所有雇员的平均薪水。四舍五入薪水到两位小数。

答案:

```
select  
d.department_name,d.location_id,count(*) ,a  
vg(e.salary) from employees e ,departments d  
where e.department_id = d.department_id group  
by d.department_name ,d.location_id;
```

4.11 创建一个查询显示雇员总数，和在 **2001、2002、2003** 和受雇的雇员人数。创建适当的列标题。

答案：

```
select          count(*)  
total,sum(decode(to_char(hire_date,'yyyy'),  
'2000',1,0))"2000" ,sum(decode(to_char(hire  
_date,'yyyy'),'2001',1,0))"2001",sum(decode  
(to_char(hire_date,'yyyy'),'2002',1,0))"200  
2",sum(decode(to_char(hire_date,'yyyy'),'20  
03',1,0))"2003" from employees e;
```

4.12 创建一个混合查询显示工作岗位和工作岗位的薪水合计，并且合计部门 **20、50、80** 和 **90** 的工作岗位的薪水。给每列一个恰当的列标题。

答案：

```
select  
job_id,sum(salary),sum(decode(department_id,  
20,salary))"Dep  
20",sum(decode(department_id,50,salary))"De  
p  
50",sum(decode(department_id,80,salary))"De  
p  
80" ,sum(decode(department_id,90,salary))"D
```

```
ep 90"from employees group by job_id;
```

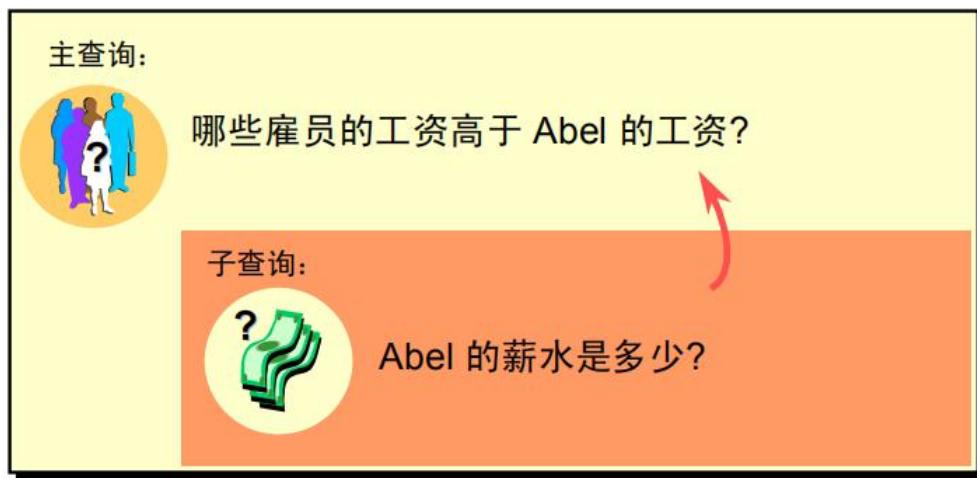
六、 子查询

1 子查询介绍

1.1 什么是子查询

子查询是一个 SELECT 语句，它是嵌在另一个 SELECT 语句中的子句。

谁的薪水比 Abel's 多？



可以用组合两个查询的方法解决这个问题，放置一个查询到另一个查询中。内查询或子查询返回一个值给外查询或主查询。使用一个子查询相当于执行两个连续查询并且用第一个查询的结果作为第二个查询的搜索值。

1.2 子查询语法

子查询语法

```
SELECT      select_list
FROM        table
WHERE       expr operator
           (SELECT      select_list
            FROM       table);
```

- 子查询（内查询）在主查询之前执行一次
- 子查询的结果被用于主查询（外查询）

可以将子查询放在许多的 SQL 子句中，包括：

-
- WHERE 子句
 - HAVING 子句
 - FROM 子句

2 使用子查询

2.1 使用子查询的原则

- 子查询放在圆括号中。
- 将子查询放在比较条件的右边。
- 在单行子查询中用单行运算符，在多行子查询中用多行运算符。

2.1.1示例

谁的薪水比 Abel 高。

用内连接实现：

```
select      em.last_name,em.salary      from  
employees      abel,employees      em      where  
abel.last_name  =  'Abel'  and  em.salary  >  
abel.salary;
```

用子查询实现：

```
select      em.last_name,em.salary      from  
employees em where em.salary >(select m.salary  
from employees m where m.last_name = 'Abel');
```

3 子查询的类型

- 单行子查询



- 多行子查询



- 单行子查询：子查询语句只返回一行的查询
- 多行子查询：子查询语句返回多行的查询

3.1 单行子查询

- 仅返回一行
- 使用单行比较符

主查询对子查询结果的单行比较运算符：

运算符	含义
=	等于
>	大于
>=	大于或等于
<	小于
<=	小于或等于
<>	不等于

3.1.1示例一

显示那些 job ID 与雇员 141 相同的雇员的名字与 job ID。

```
select      em.last_name,em.job_id      from
employees em where  em.job_id = (select job_id
from employees e where e.employee_id = 141);
```

3.1.2示例二

显示 job ID 与雇员 141 相同，并且薪水 高于雇员 143 的那些雇员。

```
select e.last_name,e.job_id,e.salary from
employees e where e.job_id = (select em.job_id
from employees em where em.employee_id = 141)
and  e.salary >  (select  emp.salary   from
employees emp where emp.employee_id = 143);
```

3.2在子查询中使用组函数

在子查询中也可使用组函数。

3.2.1示例

显示所有其薪水等于最低薪水的雇员的 last name、job ID 和 salary。

```
select  em.last_name,em.job_id,em.salary
from employees em where em.salary =(select
min(salary)  from employees);
```

3.3带子查询的 HAVING 子句

可以在 WHERE 子句中使用子查询，也可以在 HAVING 子句中使用子查询。

3.3.1示例

显示所有其最低薪水小于 部门 50 的最低薪水的部门号和最低薪水。

```
select    em.department_id,min(em.salary)
from employees em group by em.department_id
having min(em.salary) > (select min(e.salary)
from employees e where e.department_id = 50);
```

3.4什么是子查询错误?

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

```
ERROR at line 4:
ORA-01427: single-row subquery returns more than
one row
```

子查询错误：使用子查询的一个常见的错误是单行子查询返回了多行。

3.5多行子查询

- 返回多于一行
- 使用多行比较符

主查询对子查询的多行比较运算符

操作	含义
IN	等于列表中的任何成员
ANY	比较子查询返回的每个值
ALL	比较子查询返回的全部值

在条件中也可使用 NOT 取反。

3.6 在多行子查询中使用 IN 运算符

3.6.1示例

查找各部门收入为部门最低的那些雇员。显示他们的名字，薪水以及部门 ID。

```
select
e.last_name, e.department_id, e.salary    from
employees   e    where    e.salary    in(select
min(em.salary)  from  employees  em  group  by
em.department_id);
```

3.7 在多行子查询中使用 ANY 运算符

```
SELECT employee_id, last_name, job_id, salary
FROM   employees      9000, 6000, 4200
WHERE  salary < ANY
       (SELECT salary
        FROM   employees
        WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

<ANY 意思是小于最大值。 >ANY 意思是大于最小值。

3.7.1示例

显示工作岗位不是 IT_PROG 的雇员，并且这些雇员的薪水少于 IT_PROG 工作岗位的雇员的 ID、名字、工作岗位和薪水。

```
select  
e.employee_id, e.last_name, e.job_id, e.salary  
from employees e where e.job_id <> 'IT_PROG'  
and e.salary < any (select em.salary from  
employees em where em.job_id = 'IT_PROG') ;
```

3.8 在多行子查询中使用 ALL 运算符

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

<ALL 意思是小于最小值。>ALL 意思是大于最大值。

ANY 与 ALL 的区别:

ANY: >ANY 表示至少大于一个值, 即大于最小值。

ALL: >ALL 表示大于每一个值, 既大于最大值。

3.8.1示例

显示那些薪水低于工作岗位 IT_PROG 的最低薪水, 并且工作岗位不是 IT_PROG 的所有雇员。

4 子查询小结练习

4.1 写一个查询显示与 Zlotkey 在同一部门的雇员的 last name 和 hire date, 结果中不包括 Zlotkey。

答案:

```
select em.last_name, em.hire_date from  
employees em where em.department_id = (select
```

```
e.department_id    from    employees    e    where  
e.last_name = 'Zlotkey');
```

4.2 创建一个查询显示所有其薪水高于平均薪水的雇员的雇员号和名字。按薪水的升序排序。

答案:

```
select em.employee_id,em.last_name  from  
employees  em   where  em.salary > (select  
avg(salary)    from    employees)    order    by  
em.salary;
```

4.3 写一个查询显示所有工作在有任一雇员的名字中包含一个 u 的部门的雇员的雇员号和名字。

答案:

```
select e.employee_id,e.last_name  from  
employees e where e.department_id in(select  
em.department_id  from  employees  em  where  
em.last_name like '%u%');
```

4.4 显示所有部门地点号 (department location ID) 是 1700 的雇员的 last name、department number 和 job ID。

答案:

```
select  
em.last_name,em.department_id,em.job_id  
from  employees  em  where  em.department_id
```

```
in(select d.department_id from departments d  
where d.location_id = 1700);
```

4.5 显示每个向 King 报告的雇员的名字和薪水。

答案:

```
select      e.last_name,e.salary      from  
employees   e  where  e.manager_id  in(select  
em.employee_id  from  employees   em  where  
em.last_name = 'King');
```

4.6 显示在 Executive 部门的每个雇员的 department number、last name 和 job ID。

答案:

```
select  
em.department_id,em.last_name,em.job_id  
from employees em where em.department_id =  
(select d.department_id from departments d  
where d.department_name ='Executive');
```

4.7 查询显示所有收入高于平均薪水并且工作在有任一雇员的名字中带有一个 u 的部门的雇员的 employee numbers、last names 和 salaries。

答案:

```
select  
emp.employee_id,emp.last_name,emp.salary
```

```
from employees emp where emp.salary > (select
avg(salary)      from      employees)      and
emp.department_id in(select em.department_id
from  employees  em  where  em.last_name  like
'%u%') ;
```

七、 操纵数据(DML)

数据操纵语言：当添加、更新或者删除数据库中的数据时需要使用 DML 语句。DML 依据的一个集合构成了一个被称为事务的逻辑单元。

当完成下面操作时，DML 语句被执行：

- 添加新行到表中
- 修改表中的行
- 删除表中的行

1 添加一个新行到表中

1.1 INSERT 语句语法

```
INSERT INTO    table [(column [, column...])]
VALUES          (value [, value...]);
```

用该语法一次只能插入一行

1.2 指定列添加

1.2.1示例

向 departments 表中添加一条数据。

Department_Id=280

Department_name = Teaching

Manager_id=180

Location_id =2000

```
insert           into
departments(department_id,department_name,m
anager_id,location_id)
```

```
values(280,'Teaching',180,2000);
```

1.3 完全列添加

1.3.1示例

向 departments 表中添加一条数据。

```
Department_Id=290  
Department_name = Development  
Manager_id=149  
Location_id =2000
```

```
select      *      from      departments      where  
department_id = 290;
```

1.4 插入带空值的行

隐式方法：省略字段列表中的列。

```
insert          into  
departments(department_id,department_name)  
values(300,'A');
```

显式方法：在 VALUES 子句中指定 NULL 关键字。

```
insert          into  
departments(department_id,department_name,m  
anager_id,location_id)  
values(310,'A',null,null);
```

1.5 插入日期值

1.5.1示例

添加一个新的雇员：

```
insert           into

employees (employee_id,first_name,last_name,
email,phone_number,hire_date,job_id,salary,
commission_pct,manager_id,department_id)
values(300,'Old','Lu' , 'sdfsdf', '23423423',
sysdate,'AD_PRES',23423,null,204,290);
```

使用默认日期格式:

```
insert           into

employees (employee_id,first_name,last_name,
email,phone_number,hire_date,job_id,salary,
commission_pct,manager_id,department_id)
values(301,'Old','Lu' , 'sdfsddfd', '2342342
3','01/3          月
/2019','AD_PRES',23423,null,204,290);
```

指定日期格式:

```
insert           into

employees (employee_id,first_name,last_name,
email,phone_number,hire_date,job_id,salary,
commission_pct,manager_id,department_id)
values(302,'Old','Lu' , 'sdfsddf', '23423423
',to_date('2019-03-01','yyyy-MM-dd'),'AD_PR
```

```
ES',23423,null,204,290);
```

1.6从另一个表中复制行

- 用一个查询写 INSERT 语句
- 不用 VALUES 子句
- 在查询中列的数目要匹配 INSERT 子句中列的数目

1.6.1创建表

```
create table EMP
(
    id      NUMBER not null,
    name    VARCHAR2(50),
    salary   NUMBER(8,2),
    commission NUMBER(2,2)
)
```

1.6.2示例

将 employees 表中的 job_id 中含有 REP 的工作岗位的数据插入到 emp 表中。

```
insert into emp(id,name,salary,commission)
select
employee_id,last_name,salary,commission_pct
from employees where job_id like '%REP%';
OR
insert      into      emp      select
employee_id,last_name,salary,commission_pct
from employees where job_id like '%REP%';
```

1.7 使用默认值

在 INSERT 中的 DEFAULT

```
insert      into      emp(id, name)
values(302, 'Kevin');

insert      into      emp
values(301, 'OldLu', default, null);
```

2 改变表中的数据

2.1 UPDATE 语句的语法

```
UPDATE      table
SET          column = value [, column = value, ...]
[WHERE        condition];
```

用 UPDATE 语句修改已存在的行

2.1.1示例

更新 emp 表中 ID 为 170 的数据，将名字修改为 OldLu，薪水修改 20000。

```
update emp e set e.name = 'OldLu', e.salary
= 20000 where e.id = 170;
```

2.2用查询更新列

2.2.1示例

更新 emp 表中的雇员 165 的薪水，使其和雇员 156 相同。

```
update emp e set e.salary = (select salary
from emp where id = 156) where e.id = 165;
```

3 从表中删除行

3.1 DELETE 语句

```
DELETE [FROM] table  
[WHERE condition];
```

使用 DELETE 语句从表中删除已存在的行。

3.1.1示例

删除 emp 表中 id 为 302 的雇员。

```
delete from emp where id = 302;
```

3.2 删除基于另一个表的行

3.2.1示例

删除 emp 表中薪水与 IT_PROG 岗位相同的薪水。

```
delete from emp where salary in(select  
distinct salary from employees where job_id =  
'IT_PROG');
```

3.3 删除表中的全部数据

如果在删除语句中没有给定任何条件，那么数据将会删除该表中的所有数据。

3.3.1示例

删除表中的所有数据。

```
delete emp;
```

3.4删除行：完整性约束错误

```
DELETE FROM departments  
WHERE department_id = 60;
```

```
DELETE FROM departments  
*  
ERROR at line 1:  
ORA-02292: integrity constraint (HR.EMP_DEPT_FK)  
violated - child record found
```

不能删除包含主键的行，该主键被用做另一个表的外键

八、 事务处理语言(TCL)

1 什么是事务

指作为单个逻辑工作单元执行的一系列操作，要么完全地执行，要么完全地不执行。

2 什么是事务特性

2.1原子性(ATOMICITY)

事务中的操作要么都不做，要么就全做。

2.2一致性(CONSISTENCY)

一个事务应该保护所有定义在数据上的不变的属性(例如完整性约束)。在完成了一个成功的事务时，数据应处于一致的状态。

2.3隔离性(ISOLATION)

一个事务的执行不能被其他事务干扰。

2.4 持久性(DURABILITY)

一个事务一旦提交，它对数据库中数据的改变就应该是永久性的。

3 使用事务

commit.....事物提交

rollback....事物回滚

savepoint..设置回滚点

3.1 事务类型

3.1.1 显式事务

需要我们手动的提交或回滚。

DML 语言中的所有操作都是显示事务操作。

3.1.2 隐式事务

数据库自动提交不需要我们做任何处理，同时也不具备回滚性。

DDL、DCL 语言都是隐式事务操作

3.2 事务提交

Commit

当我们执行了一个 DML 语言后，此时的数据并不会持久化到数据文件中。需要使用 commit 来确认提交。

3.3 事务回滚

Rollback

当我们执行了一个 DML 语言后，也可以使用 rollback 来撤销当前对表的操作。

3.4 设置回滚点

Savepoint

Rollback to

九、 数据定义语言(DDL)

create.....创建数据库对象

drop.....删除数据库对象
alter.....修改数据库对象
rename.....修改数据库对象名称

1 创建表(CREATE TABLE)

- 创建表

```
CREATE TABLE dept
  (deptno NUMBER(2),
   dname  VARCHAR2(14),
   loc    VARCHAR2(13));
Table created.
```

1.1示例

```
create table dept (deptno NUMBER(2), dname
VARCHAR2(14), loc VARCHAR2(13));
```

1.2 Oracle 数据库中的表

- 用户表:
 - 由用户创建和维护的表的集合。
 - 包含用户信息。
- 数据字典:
 - 由 Oracle 服务器创建和维护的表的集合。
 - 包含数据库信息。

1.2.1常见的数据字典表

- 查看本用户所拥有的表的名称

```
SELECT table_name
FROM user_tables ;
```

- 查看本用户所拥有的不同的对象类型

```
SELECT DISTINCT object_type
FROM user_objects ;
```

- 查看本用户所拥有的表、视图、同义词和序列

```
SELECT *
FROM user_catalog ;
```

1.3用查询创建表

1.3.1示例

```
create      table      dept80      as      select  
employee_id, last_name  name, salary, hire_date  
from employees e where e.department_id = 80;
```

2 修改表(ALTER TABLE)

ALTER TABLE 语句可以修改表的信息。

- 添加一个新列
- 修改一个已存在的列
- 删除一个列

2.1添加一个新的列

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

2.1.1示例

```
alter table dept add(salary number(8,2));
```

2.2修改一个列

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

2.2.1示例一

修改数据类型

```
alter      table      dept      modify(dname
```

```
varchar2(40));
```

2.2.2示例二

修改默认值

```
alter table dept modify(salary number(8,2)  
default 1000);
```

```
SQL> alter table dept modify(salary  
number(8,2) default null);
```

2.2.3示例三

修改列名

```
alter table dept rename column dname to  
name;
```

2.3删除一个列

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

2.3.1示例

```
alter table dept drop column salary;
```

3 修改名称(RENAME)

3.1示例

```
rename dept80 to dept90;
```

4 截断表(TRUNCATE TABLE)

特点：

删除表中的所有的数据，但是保留表结构。

在截断表时不能给定条件

截断表时隐式事务。

4.1示例

```
truncate table dept90;
```

5 删除表(DROP TABLE)

5.1示例

```
drop table dept90;
```

十、 定义约束

1 数据库中的约束类型

- 非空约束(NOT NULL)
- 唯一性约束(UNIQUE)
- 主键约束(PRIMARY KEY)
- 外键约束(FOREIGN KEY)
- 用户自定义约束(CHECK)

约束	说明
NOT NULL	指定列不能包含空值
UNIQUE	指定列的值或者列的组合的值对于表中所有的行必须是唯一的
PRIMARY KEY	表的每行的唯一性标识
FOREIGN KEY	在列和引用表的一个列之间建立并且强制一个外键关系
CHECK	指定一个必须为真的条件

1.1定义约束原则

创建一个约束：在创建表的同时，或者在创建表之后都可以定义约束。可以给约束起名字，但是约束名不能相同，必须是唯一的。如果没有为约束起名字，Oracle 服务器将用默认格式 SYS_Cn 产生一个名字，这里 n 是一个唯一的整数，所以约束名是唯一的。

2 定义 NOT NULL 约束

```
CREATE TABLE employees (
    employee_id      NUMBER(6),
    last_name        VARCHAR2(25) NOT NULL, ← 由系统指定
    salary           NUMBER(8,2),
    commission_pct   NUMBER(2,2),
    hire_date        DATE
        CONSTRAINT emp_hire_date_nn ← 用户指定
                                    NOT NULL,
    ...
)
```

2.1 创建表时定义 NOT NULL 约束

2.1.1示例

```
create table dept80(id number, name
varchar2(20) not null, salary number
constraint dept80_notn not null);
```

2.2 修改表定义 NOT NULL 约束

2.2.1示例

```
alter table dept80 modify location_id not
null;
```

3 定义 UNIQUE 约束

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

3.1 创建表时定义 UNIQUE 约束

3.1.1示例

```
create table dept90(id number constraint
dept90_uk unique, name varchar2(20));
```

3.2 修改表定义 UNIQUE 约束

3.2.1示例

```
alter table dept90 modify(name unique);
```

4 定义 PRIMARY KEY 约束

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

4.1 创建表时定义 PRIMARY KEY 约束

4.1.1示例

```
create table dept70(id number constraint  
dept70_pk primary key);
```

4.2 修改表定义 PRIMARY KEY 约束

4.2.1示例

```
alter table dept60 modify(id constraint  
dept60_pk primary key);
```

4.3 创建联合主键

4.3.1示例

```
create table dept50(id number, name  
varchar2(20), constraint dept50_pk primary  
key(id, name));
```

5 定义 FOREIGN KEY 约束

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

5.1 创建表时定义 FOREIGN KEY

5.1.1示例

```
create table dept40(id number,d_id  
number,constraint dept40_fk foreign key(d_id)  
references dept60(id));
```

5.2 修改表定义 FOREIGN KEY

5.2.1示例

```
alter table dept50 add constraint  
dept50_fk foreign key(d_id) references  
dept60(id);
```

6 定义 CHECK 约束

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

6.1 创建表时定义 CHECK 约束

6.1.1示例

```
create table dept30(id number,salary  
number(8,2) constraint dept30_ck check  
(salary > 1000));
```

6.2 修改表定义 CHECK 约束

6.2.1示例

```
alter table dept50 add constraint  
dept50_ck check(salary > 1000);
```

7 禁用与启用约束

7.1查看约束

```
SELECT constraint_name, constraint_type,  
       search_condition  
  FROM user_constraints  
 WHERE table_name = 'EMPLOYEES';
```

7.1.1示例

```
select  
constraint_name,constraint_type,search_cond  
ition from user_constraints where table_name  
= 'DEPT40';
```

7.2禁用约束

```
ALTER TABLE employees  
DISABLE CONSTRAINT emp_emp_id_pk CASCADE;  
Table altered.
```

7.2.1示例

```
alter table dept40 disable constraint
```

```
dept40_ck;
```

级联禁用

```
alter table dept60 disable constraint  
dept60_pk cascade;
```

7.3启用约束

```
ALTER TABLE employees  
ENABLE CONSTRAINT emp_emp_id_pk;  
Table altered.
```

7.3.1示例

```
alter table dept40 enable constraint  
dept40_ck;
```

十一、 创建数据库其他对象

1 视图(View)

1.1什么是视图

可以通过创建表的视图来表现数据的逻辑子集或数据的组合。视图是基于表或另一个视图的逻辑表，一个视图并不包含它自己的数据，它象一个窗口，通过该窗口可以查看或改变表中的数据。视图基于其上的表称为基表。

EMPLOYEES Table:

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY
100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	17000
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	9000
104	Elischa	Eversmann	EEVERSM	515.123.4560	01-JAN-95	SA_MAN	10500
105	Mark	Tobias	MTOBIA	515.123.4561	01-JAN-95	SA_REP	11000
106	Adam	Cronin	ACRONIN	515.123.4562	01-JAN-95	SA_REP	11000
107	Janet	Ranganathan	JRANGAN	515.123.4563	01-JAN-95	SA_REP	11000
108	Michael	Schoen	MSCHOEN	515.123.4564	01-JAN-95	SA_REP	11000
109	Susan	Mavris	SMAVRIS	515.123.4565	01-JAN-95	SA_REP	11000
110	Pat	Bergman	PBERGM	515.123.4566	01-JAN-95	SA_REP	11000
111	James	Frederickson	JFREDER	515.123.4567	01-JAN-95	SA_REP	11000
112	Wendy	Colmenares	WCOLMEN	515.123.4568	01-JAN-95	SA_REP	11000
113	Matthew	Vandemoer	MVANDEM	515.123.4569	01-JAN-95	SA_REP	11000
114	Victoria	Chen	VCHEN	515.123.4570	01-JAN-95	SA_REP	11000
115	Mike	Plavcan	MPLAVCAN	515.123.4571	01-JAN-95	SA_REP	11000
116	Patricia	Gunderson	PGUNDERSON	515.123.4572	01-JAN-95	SA_REP	11000
117	Thomas	Ernst	TERNST	515.123.4573	01-JAN-95	SA_REP	11000
118	Robert	King	RKING	515.123.4574	01-JAN-95	SA_REP	11000
119	David	Driscoll	DDRISCOLL	515.123.4575	01-JAN-95	SA_REP	11000
120	John	Matsumoto	JMATSUMO	515.123.4576	01-JAN-95	SA_REP	11000
121	Michael	Schoen	MSCHOEN	515.123.4577	01-JAN-95	SA_REP	11000
122	Victoria	Chen	VCHEN	515.123.4578	01-JAN-95	SA_REP	11000
123	Mike	Plavcan	MPLAVCAN	515.123.4579	01-JAN-95	SA_REP	11000
124	Patricia	Gunderson	PGUNDERSON	515.123.4580	01-JAN-95	SA_REP	11000
125	Thomas	Ernst	TERNST	515.123.4581	01-JAN-95	SA_REP	11000
126	Robert	King	RKING	515.123.4582	01-JAN-95	SA_REP	11000
127	David	Driscoll	DDRISCOLL	515.123.4583	01-JAN-95	SA_REP	11000
128	John	Matsumoto	JMATSUMO	515.123.4584	01-JAN-95	SA_REP	11000
129	Michael	Schoen	MSCHOEN	515.123.4585	01-JAN-95	SA_REP	11000
130	Victoria	Chen	VCHEN	515.123.4586	01-JAN-95	SA_REP	11000
131	Mike	Plavcan	MPLAVCAN	515.123.4587	01-JAN-95	SA_REP	11000
132	Patricia	Gunderson	PGUNDERSON	515.123.4588	01-JAN-95	SA_REP	11000
133	Thomas	Ernst	TERNST	515.123.4589	01-JAN-95	SA_REP	11000
134	Robert	King	RKING	515.123.4590	01-JAN-95	SA_REP	11000
135	David	Driscoll	DDRISCOLL	515.123.4591	01-JAN-95	SA_REP	11000
136	John	Matsumoto	JMATSUMO	515.123.4592	01-JAN-95	SA_REP	11000
137	Michael	Schoen	MSCHOEN	515.123.4593	01-JAN-95	SA_REP	11000
138	Victoria	Chen	VCHEN	515.123.4594	01-JAN-95	SA_REP	11000
139	Mike	Plavcan	MPLAVCAN	515.123.4595	01-JAN-95	SA_REP	11000
140	Patricia	Gunderson	PGUNDERSON	515.123.4596	01-JAN-95	SA_REP	11000
141	Thomas	Ernst	TERNST	515.123.4597	01-JAN-95	SA_REP	11000
142	Robert	King	RKING	515.123.4598	01-JAN-95	SA_REP	11000
143	David	Driscoll	DDRISCOLL	515.123.4599	01-JAN-95	SA_REP	11000
144	John	Matsumoto	JMATSUMO	515.123.4600	01-JAN-95	SA_REP	11000
145	Michael	Schoen	MSCHOEN	515.123.4601	01-JAN-95	SA_REP	11000
146	Victoria	Chen	VCHEN	515.123.4602	01-JAN-95	SA_REP	11000
147	Mike	Plavcan	MPLAVCAN	515.123.4603	01-JAN-95	SA_REP	11000
148	Patricia	Gunderson	PGUNDERSON	515.123.4604	01-JAN-95	SA_REP	11000
149	Thomas	Ernst	TERNST	515.123.4605	01-JAN-95	SA_REP	11000
150	Robert	King	RKING	515.123.4606	01-JAN-95	SA_REP	11000
151	David	Driscoll	DDRISCOLL	515.123.4607	01-JAN-95	SA_REP	11000
152	John	Matsumoto	JMATSUMO	515.123.4608	01-JAN-95	SA_REP	11000
153	Michael	Schoen	MSCHOEN	515.123.4609	01-JAN-95	SA_REP	11000
154	Victoria	Chen	VCHEN	515.123.4610	01-JAN-95	SA_REP	11000
155	Mike	Plavcan	MPLAVCAN	515.123.4611	01-JAN-95	SA_REP	11000
156	Patricia	Gunderson	PGUNDERSON	515.123.4612	01-JAN-95	SA_REP	11000
157	Thomas	Ernst	TERNST	515.123.4613	01-JAN-95	SA_REP	11000
158	Robert	King	RKING	515.123.4614	01-JAN-95	SA_REP	11000
159	David	Driscoll	DDRISCOLL	515.123.4615	01-JAN-95	SA_REP	11000
160	John	Matsumoto	JMATSUMO	515.123.4616	01-JAN-95	SA_REP	11000
161	Michael	Schoen	MSCHOEN	515.123.4617	01-JAN-95	SA_REP	11000
162	Victoria	Chen	VCHEN	515.123.4618	01-JAN-95	SA_REP	11000
163	Mike	Plavcan	MPLAVCAN	515.123.4619	01-JAN-95	SA_REP	11000
164	Patricia	Gunderson	PGUNDERSON	515.123.4620	01-JAN-95	SA_REP	11000
165	Thomas	Ernst	TERNST	515.123.4621	01-JAN-95	SA_REP	11000
166	Robert	King	RKING	515.123.4622	01-JAN-95	SA_REP	11000
167	David	Driscoll	DDRISCOLL	515.123.4623	01-JAN-95	SA_REP	11000
168	John	Matsumoto	JMATSUMO	515.123.4624	01-JAN-95	SA_REP	11000
169	Michael	Schoen	MSCHOEN	515.123.4625	01-JAN-95	SA_REP	11000
170	Victoria	Chen	VCHEN	515.123.4626	01-JAN-95	SA_REP	11000
171	Mike	Plavcan	MPLAVCAN	515.123.4627	01-JAN-95	SA_REP	11000
172	Patricia	Gunderson	PGUNDERSON	515.123.4628	01-JAN-95	SA_REP	11000
173	Thomas	Ernst	TERNST	515.123.4629	01-JAN-95	SA_REP	11000
174	Robert	King	RKING	515.123.4630	01-JAN-95	SA_REP	11000
175	David	Driscoll	DDRISCOLL	515.123.4631	01-JAN-95	SA_REP	11000
176	John	Matsumoto	JMATSUMO	515.123.4632	01-JAN-95	SA_REP	11000
177	Michael	Schoen	MSCHOEN	515.123.4633	01-JAN-95	SA_REP	11000
178	Victoria	Chen	VCHEN	515.123.4634	01-JAN-95	SA_REP	11000
179	Mike	Plavcan	MPLAVCAN	515.123.4635	01-JAN-95	SA_REP	11000
180	Patricia	Gunderson	PGUNDERSON	515.123.4636	01-JAN-95	SA_REP	11000
181	Thomas	Ernst	TERNST	515.123.4637	01-JAN-95	SA_REP	11000
182	Robert	King	RKING	515.123.4638	01-JAN-95	SA_REP	11000
183	David	Driscoll	DDRISCOLL	515.123.4639	01-JAN-95	SA_REP	11000
184	John	Matsumoto	JMATSUMO	515.123.4640	01-JAN-95	SA_REP	11000
185	Michael	Schoen	MSCHOEN	515.123.4641	01-JAN-95	SA_REP	11000
186	Victoria	Chen	VCHEN	515.123.4642	01-JAN-95	SA_REP	11000
187	Mike	Plavcan	MPLAVCAN	515.123.4643	01-JAN-95	SA_REP	11000
188	Patricia	Gunderson	PGUNDERSON	515.123.4644	01-JAN-95	SA_REP	11000
189	Thomas	Ernst	TERNST	515.123.4645	01-JAN-95	SA_REP	11000
190	Robert	King	RKING	515.123.4646	01-JAN-95	SA_REP	11000
191	David	Driscoll	DDRISCOLL	515.123.4647	01-JAN-95	SA_REP	11000
192	John	Matsumoto	JMATSUMO	515.123.4648	01-JAN-95	SA_REP	11000
193	Michael	Schoen	MSCHOEN	515.123.4649	01-JAN-95	SA_REP	11000
194	Victoria	Chen	VCHEN	515.123.4650	01-JAN-95	SA_REP	11000
195	Mike	Plavcan	MPLAVCAN	515.123.4651	01-JAN-95	SA_REP	11000
196	Patricia	Gunderson	PGUNDERSON	515.123.4652	01-JAN-95	SA_REP	11000
197	Thomas	Ernst	TERNST	515.123.4653	01-JAN-95	SA_REP	11000
198	Robert	King	RKING	515.123.4654	01-JAN-95	SA_REP	11000
199	David	Driscoll	DDRISCOLL	515.123.4655	01-JAN-95	SA_REP	11000
200	John	Matsumoto	JMATSUMO	515.123.4656	01-JAN-95	SA_REP	11000
201	Michael	Schoen	MSCHOEN	515.123.4657	01-JAN-95	SA_REP	11000
202	Victoria	Chen	VCHEN	515.123.4658	01-JAN-95	SA_REP	11000
203	Mike	Plavcan	MPLAVCAN	515.123.4659	01-JAN-95	SA_REP	11000
204	Patricia	Gunderson	PGUNDERSON	515.123.4660	01-JAN-95	SA_REP	11000
205	Thomas	Ernst	TERNST	515.123.4661	01-JAN-95	SA_REP	11000
206	Robert	King	RKING	515.123.4662	01-JAN-95	SA_REP	11000
207	David	Driscoll	DDRISCOLL	515.123.4663	01-JAN-95	SA_REP	11000
208	John	Matsumoto	JMATSUMO	515.123.4664	01-JAN-95	SA_REP	11000
209	Michael	Schoen	MSCHOEN	515.123.4665	01-JAN-95	SA_REP	11000
210	Victoria	Chen	VCHEN	515.123.4666	01-JAN-95	SA_REP	11000
211	Mike	Plavcan	MPLAVCAN	515.123.4667	01-JAN-95	SA_REP	11000
212	Patricia	Gunderson	PGUNDERSON	515.123.4668	01-JAN-95	SA_REP	11000
213	Thomas	Ernst	TERNST	515.123.4669	01-JAN-95	SA_REP	11000
214	Robert	King	RKING	515.123.4670	01-JAN-95	SA_REP	11000
215	David	Driscoll	DDRISCOLL	515.123.4671	01-JAN-95	SA_REP	11000
216	John	Matsumoto	JMATSUMO	515.123.4672	01-JAN-95	SA_REP	11000
217	Michael	Schoen	MSCHOEN	515.123.4673	01-JAN-95	SA_REP	11000
218	Victoria	Chen	VCHEN	515.123.4674	01-JAN-95	SA_REP	11000
219	Mike	Plavcan	MPLAVCAN	515.123.4675	01-JAN-95	SA_REP	11000
220	Patricia	Gunderson	PGUNDERSON	515.123.4676	01-JAN-95	SA_REP	11000
221	Thomas	Ernst	TERNST	515.123.4677	01-JAN-95	SA_REP	11000
222	Robert	King	RKING	515.123.4678	01-JAN-95	SA_REP	11000
223	David	Driscoll	DDRISCOLL	515.123.4679	01-JAN-95	SA_REP	11000
224	John	Matsumoto	JMATSUMO	515.123.4680	01-JAN-95	SA_REP	11000
225	Michael	Schoen	MSCHOEN	515.123.4681	01-JAN-95	SA_REP	11000
226	Victoria	Chen	VCHEN	515.123.4682	01-JAN-95	SA_REP	11000
227	Mike	Plavcan	MPLAVCAN	515.123.4683	01-JAN-95	SA_REP	11000
228	Patricia	Gunderson	PGUNDERSON	515.123.4684	01-JAN-95	SA_REP	11000
229	Thomas	Ernst	TERNST	515.123.4685	01-JAN-95	SA_REP	11000
230	Robert	King	RKING	515.123.4686	01-JAN-95	SA_REP	11000
231	David	Driscoll	DDRISCOLL	515.123.4687	01-JAN-95	SA_REP	11000
232	John	Matsumoto	JMATSUMO	515.123.4688	01-JAN-95	SA_REP	11000
233	Michael	Schoen	MSCHOEN	515.123.4689	01-JAN-95	SA_REP	11000
234	Victoria	Chen	VCHEN	515.123.4690	01-JAN-9		

简单视图：

- 数据仅来自一个表
- 不包含函数或数据分组
- 能通过视图执行 DML 操作

复杂视图：

- 数据来自多个表
- 包含函数或数据分组
- 不允许通过视图进行 DML 操作

1.5 创建简单视图

```
CREATE VIEW empvu80
AS SELECT employee_id, last_name, salary
      FROM employees
     WHERE department_id = 80;
View created.
```

1.5.1示例

创建一个视图，视图中包含部门 id 为 80 的员工的 id，名字以及薪水。

```
create view emp80 as select
e.employee_id,e.last_name,e.salary from
employees e;
```

1.6 用子查询中的列别名创建视图

如果在创建视图的查询语句中含有列别名，那么列别名将作为视图的列名。

1.6.1示例

创建一个视图，包含部门 id 为 50 的员工 id 使用 ID_NUMBER 命名该列，包含员工名字使用 NAME 命名该列，包含员工的年薪使用 ANN_SALARY 命名该列。

```
create view emp50 as select e.employee_id
id_number,e.last_name      name,12*e.salary
ann_salary from employees e;
```

1.7从视图中取回数据

1.7.1示例一

查询部门 id 为 80 的员工信息，包含他们的 id，名字以及薪水。

```
select * from emp80;
```

1.7.2示例二

查询部门 id 为 50 的员工信息，包含他们的 id 和薪水。

```
select e.id_number,e.ann_salary from emp50 e;
```

1.8创建复杂视图

```
CREATE VIEW dept_sum_vu
  (name, minsal, maxsal, avgsal)
AS SELECT d.department_name, MIN(e.salary),
          MAX(e.salary), AVG(e.salary)
        FROM employees e, departments d
       WHERE e.department_id = d.department_id
         GROUP BY d.department_name;
View created.
```

1.8.1示例

创建一个视图，包含每个部门的部门名称，部门最低薪水、部门最高薪水以及部门的平均薪水。

```
create view dept_name as select
d.department_name,min(e.salary)
min,max(e.salary) max ,avg(e.salary) avg from
employees e, departments d where
e.department_id = d.department_id group by
```

```
d.department_name;
```

1.9 在定义视图时指定列名

1.9.1示例

创建一个视图，包含每个部门的部门名称、部门最低薪水、部门最高薪水以及部门的平均薪水。将部门名称命名为 name、最低薪水命名为 minsal、最高薪水命名为 maxsal、平均薪水命名为 avgsal。

```
create view  
dept_name1(name,minsal,maxsal,avgsal) as  
select  
d.department_name,min(e.salary) ,max(e.sala  
ry) ,avg(e.salary) avg from employees e,  
departments d where e.department_id =  
d.department_id group by d.department_name;
```

1.10 视图中 DML 操作的执行规则

如果视图中包含下面的部分就不能修改数据：

- 组函数
- GROUP BY 子句
- DISTINCT 关键字
- 用表达式定义的列

1.10.1 示例

删除 emp80 视图中雇员 ID 为 190 的雇员。

```
delete from emp80 e where e.employee_id =  
190;
```

1.11 拒绝 DML 操作

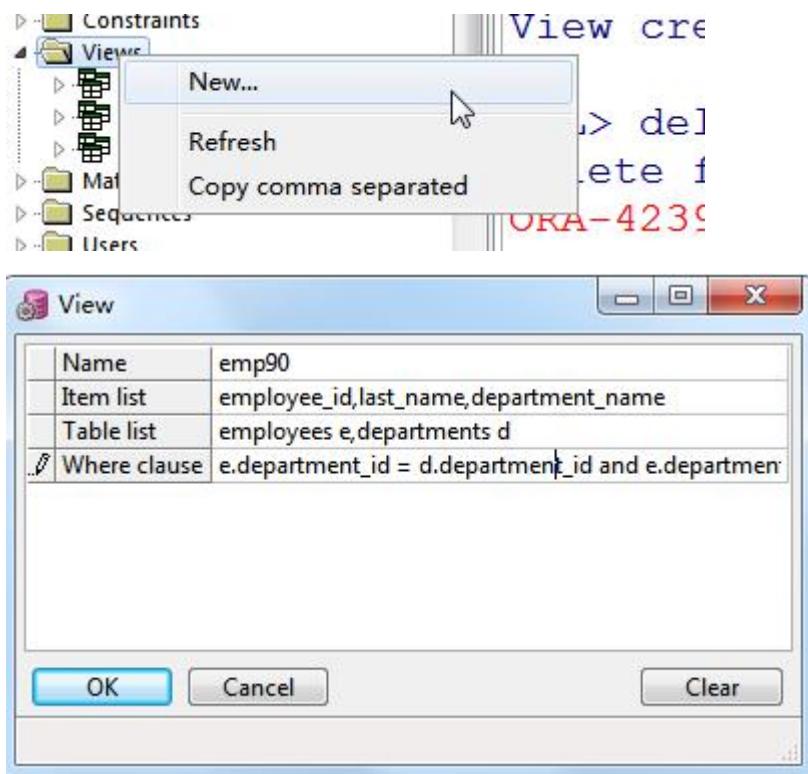
```
CREATE OR REPLACE VIEW empvu10
  (employee_number, employee_name, job_title)
AS SELECT employee_id, last_name, job_id
  FROM employees
 WHERE department_id = 10
 WITH READ ONLY;
View created.
```

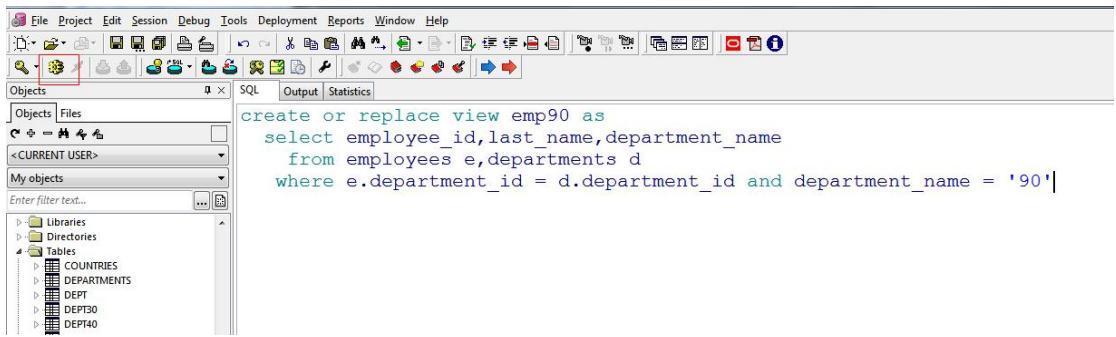
1.11.1 示例

创建一个简单视图，包含 employees 表中的所有数据，单该视图拒绝 DML 操作。

```
create view v_emp as select * from employees
with read only;
```

1.12 通过工具创建视图





1.13删除视图

```
DROP VIEW view;
```

删除视图不会丢失数据，因为视图是基于数据库中的基本表的。

1.13.1 示例

删除名称为 emp90 的视图。

```
drop view emp90;
```

1.14内建视图

1.14.1 是什么内建视图

- 内建视图是一个带有别名 (或相关名) 的可以在 SQL 语句中使用的子查询。
- 一个主查询的在 FROM 子句中指定的子查询就是一个内建视图。

内建视图：内建视图由位于 FROM 子句中命名了别名的子查询创建。该子查询定义一个可以在主查询中引用数据源。

1.14.1.1 示例

显示那些雇员低于他们部门最高薪水的雇员的名字、薪水、部门号和他们部门最高的薪水。

```
select
    em.last_name, em.salary, em.department_id,
    e.maxsal   from   employees   em   , (select
    e.department_id, max(e.salary)  maxsal  from
```

```
employees e group by e.department_id)e where  
em.department_id      =      e.department_id      and  
em.salary < e.maxsal;
```

1.15 Top-N 分析

1.15.1 什么是“Top-N”分析

Top-N 查询在需要基于一个条件，从表中显示最前面的 n 条记录或最后面的 n 条记录时是有用的。该结果可以用于进一步分析，例如，用 Top-N 分析你可以执行下面的查询类型：

- 在中挣钱最多的三个人
- 公司中最新的四个成员
- 销售产品最多的两个销售代表
- 过去 6 个月中销售最好的 3 种产品

1.15.2 执行“Top-N”分析

Top-N 查询使用一个带有下面描述的元素的一致的嵌套查询结构：

- 子查询或者内建视图产生数据的排序列表，该子查询或者内建视图包含 ORDER BY 子句来确保排序以想要的顺序排列。为了取回最大值，需要用 DESC 参数。
- 在最后的结果集中用外查询限制行数。外查询包括下面的组成部分：
 - ROWNUM 伪列，它为从子查询返回的每一行指定一个从 1 开始的连续的值
 - 一个 WHERE 子句，它指定被返回的 n 行，外 WHERE 子句必须用一个<或者<=操作。

1.15.3 示例一

从 EMPLOYEES 表中显示挣钱最多的 3 个人的名字及其薪水。

```
select    rownum    ,last_name,salary   from  
(select last_name, salary from employees order  
by salary desc) where rownum <=3;
```

1.15.4 示例二

显示公司中 4 个资格最老的雇员显示他们的入职时间与名字。

```
select rownum, e.last_name, e.hire_date
from (select last_name, hire_date from
employees order by hire_date) e where rownum
<= 4;
```

1.16 Oracle 的分页查询

1.16.1 什么是分页查询

当查询的结果集数据量过大时，可能会导致各种各样的问题发生，例如：服务器资源被耗尽，因数据传输量过大而使处理超时，等等。最终都会导致查询无法完成。解决这个问题的一个策略就是“分页查询”，也就是说不要一次性查询所有的数据，每次只查询一部分数据。这样分批次地进行处理，可以呈现出很好的用户体验，对服务器资源的消耗也不大。

分页查询原则：

在内建视图中通过 rownum 伪劣值的判断来指定获取数据的数量。

1.16.2 示例

查询雇员表中数据，每次只返回 10 条数据。

```
select * from (select rownum rn, e.* from
employees e) em where em.rn between 11 and 20;
```

2 序列(Sequence)

2.1 什么是序列

序列是用户创建的数据库对象，序列会产生唯一的整数。序列的一个典型的用途是创建一个主键的值，它对于每一行必须是唯一的。序列由一个 Oracle 内部程序产生并增加或减少。

序列是一个节省时间的对象，因为它可以减少应用程序中产生序列程序的代码量。序列号独立于表被存储和产生，因此，相同的序列可以被多个表使用。

2.2 创建序列

2.2.1 通过 DDL 语句创建序列

```
CREATE SEQUENCE sequence
  [INCREMENT BY n]
  [START WITH n]
  [{MAXVALUE n | NOMAXVALUE}]
  [{MINVALUE n | NOMINVALUE}]
  [{CYCLE | NOCYCLE}]
  [{CACHE n | NOCACHE}];
```

在语法中：

sequence 是序列发生器的名字

INCREMENT BY n 指定序列号之间的间隔，在这儿 n 是一个整数（如果该子句被省略，序列增量为 1）

START WITH n 指定要产生的第一个序列数（如果该子句被省略，序列从 1 开始）

MAXVALUE n 指定序列能产生的最大值

NOMAXVALUE 对于升序序列指定 10^{27} 为最大值，对于降序序列指定 -1 为最大值（这是默认选项）

MINVALUE n 指定最小序列值

NOMINVALUE 对于升序序列指定 1 为最小值，对于降序序列指定 -(10^{26}) 为最小值（这是默认选项）

CYCLE|NOCYCLE 指定序列在达到它的最大或最小值之后，是否继续产生（NOCYCLE 是默认选项）

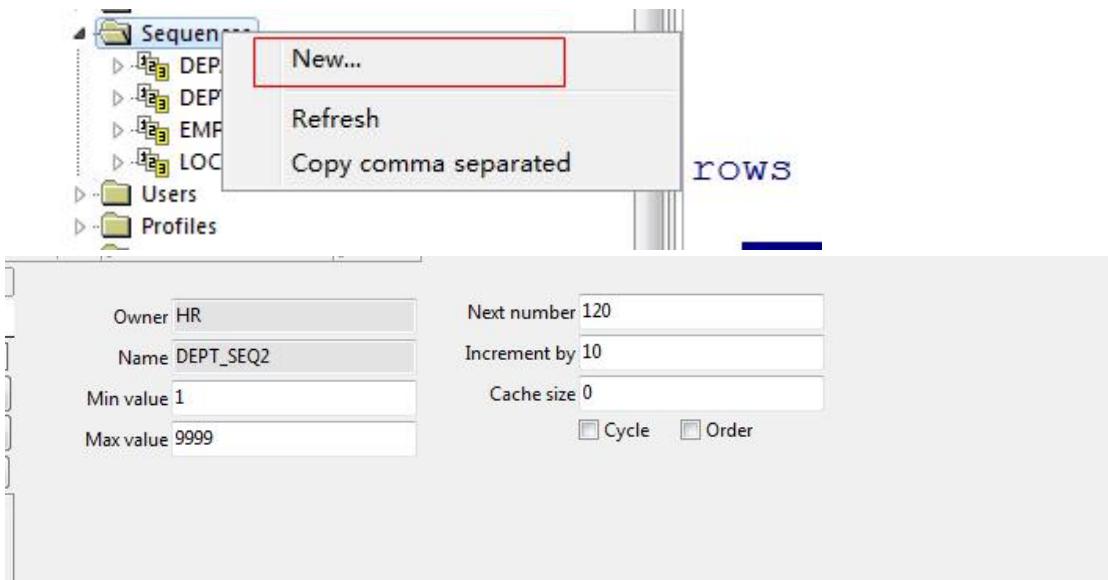
CACHE n|NOCACHE 指定 Oracle 服务器预先分配多少值，并且保持在内存中（默认情况下，Oracle 服务器缓冲 20 个值）

2.2.1.1 示例

创建一个序列名称为：dept_seq，增长间隔为 10，从 120 开始，最大值为 9999，不缓存。不循环使用。

```
create sequence dept_seq increment by 10
start with 120 maxvalue 9999 nocache nocycle;
```

2.2.2通过工具创建序列



2.3操作序列

2.3.1查询序列

```
SELECT    sequence_name, min_value, max_value,
          increment_by, last_number
FROM      user_sequences;
```

2.3.1.1示例

```
select
sequence_name,increment_by,max_value,min_va
lue,last_number from user_sequences;
```

2.3.2使用序列

NEXTVAL 和 CURRVAL 伪列

- NEXTVAL 返回下一个可用的序列值，它每次返回一个唯一的被引用值，即使对于不同的用户也是如此
- CURRVAL 获得当前的序列值
- 在 CURRVAL 获得一个值以前，NEXTVAL 对该序列必须发布

2.3.2.1 示例

在 location ID 2500 中插入一个新部门名称 Support。

```
insert          into  
departments(department_id,department_name,  
location_id)  
values(dept_seq.nextval,'Support',2500);
```

2.4 修改与删除序列

2.4.1 修改序列

```
ALTER SEQUENCE dept_deptid_seq  
INCREMENT BY 20  
MAXVALUE 999999  
NOCACHE  
NOCYCLE;  
Sequence altered.
```

2.4.2 修改序列的原则

必须是被修改序列的所有者，或者有 ALTER 权限。

用 ALTER SEQUENCE 语句，只有以后的序列数会受影响。

用 ALTER SEQUENCE 语句，START WITH 选项不能被改变。为了以不同的数重新开始一个序列，该序列必须被删除和重新创建。

2.4.2.1 示例

将 dept_seq 序列中的增长量修改 20，最大值修改为 999999。

```
alter sequence dept_seq increment by 20  
maxvalue 999999 nocache nocycle;
```

2.4.3 删除序列

```
DROP SEQUENCE dept_deptid_seq;  
Sequence dropped.
```

2.4.3.1 示例

删除 dept_seq 序列

```
drop sequence dept_seq;
```

3 索引(Index)

3.1 什么是索引

在关系型数据库中，索引是一种单独的、物理的对数据库表中一列或多列的值进行排序的一种存储结构，它是某个表中一列或若干列值的集合和相应的指向表中物理标识这些值的数据页的逻辑指针清单。索引的作用相当于图书的目录，可以根据目录中的页码快速找到所需的内容。

索引提供对表中行的直接和快速访问，它的目的是用已索引的路径快速定位数据以减少磁盘 I/O。索引由 Oracle 服务器自动使用和维护，索引逻辑地和物理地独立于他们索引的表，这意味着索引可以在任何时候被创建或删除，并且不影响基表或其它的索引。当删除表时，相应的索引也被删除。

3.2 索引的类型

唯一性索引：当你在一个表中定义一个列为主键，或者定义一个唯一键约束时 Oracle 服务器自动创建该索引，索引的名字习惯上是约束的名字。

非唯一索引：由用户创建，例如，可以创建一个 FOREIGN KEY 列索引用于一个查询中的连接来改进数据取回的速度。

3.3 创建索引的方式

- **自动：**在一个表的定义中，当定义一个 PRIMARY KEY 或 UNIQUE 约束时，一个唯一索引被自动创建
- **手动：**用户能够在列上创建非唯一的索引来加速对行的访问。

3.4 使用索引

过多也件坏事

在表上建立更多的索引并不意味着更快地查询，在带索引的表上被提交的每个 DML 操作意味着索引必须更新；与表联系的索引越多，对 Oracle 数据库的影响越大，Oracle 数据库在每次 DML 操作之后必须更新所有的索引。

3.4.1 什么时候创建索引

- 一个列包含一个大范围的值
- 一个列包含很多的空值
- 一个或多个列经常同时在一个 WHERE 子句中或一个连接条件中被使用
- 表很大，并且经常的查询期望取回少于百分之 2 到 4 的行。

3.4.2什么时候不创建索引

- 表很小
- 不经常在查询中作为条件被使用的列
- 大多数查询期望取回多于表中百分之 2 到 4 的行
- 表经常被更新
- 被索引的列作为表达式的的一部分被引用

3.5操作索引

3.5.1非唯一性索引的类型

oracle 的非唯一性索引：单行索引，复合索引(组合索引)，函数索引。

3.5.2创建索引语法

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

3.5.2.1 创建单行索引

3.5.2.1.1示例

为 employees 表中的 last_name 创建一个索引并命名为 emp_index。

```
create index emp_index on  
employees(last_name);
```

3.5.2.2 创建复合索引

3.5.2.2.1示例

为 departments 表创建一个包括 manager_id 与 location_id 复合索引并命名为 dept_man_loc。

```
create index dept_man_loc on  
departments(manager_id,location_id);
```

3.5.2.3 创建函数索引

3.5.2.3.1示例

为 departments 表中的 department_name 创建一个带有大写函数的索引 dept_upper2。

```
create index dept_upper2 on  
departments(upper(department_name));
```

3.5.3查询索引

- **USER_INDEXES** 数据字典视图包含索引和它唯一的名字
- **USER_IND_COLUMNS** 视图包含索引名、表名和列名

```
SELECT    ic.index_name, ic.column_name,  
        ic.column_position col_pos, ix.uniqueness  
FROM      user indexes ix, user ind columns ic  
WHERE     ic.index_name = ix.index_name  
AND       ic.table_name = 'EMPLOYEES';
```

3.5.3.1 示例

```
select  
ic.INDEX_NAME ,ic.COLUMN_NAME,ic.COLUMN_POS  
ITION,ix.uniqueness      from      user_indexes  
ix ,user _ind _columns ic where ix.index_name =  
ic.INDEX_NAME      and      ic.TABLE_NAME      =  
'DEPARTMENTS' ;
```

3.5.4删除索引

```
DROP INDEX index;
```

3.5.4.1 示例

删除名称为 dept_upper 的索引。

```
drop index dept_upper;
```

4 同义词(Synonym)

4.1 什么是同义词

同义词可以除去对象名必须带的方案限制，并提供给你一个可替换表名、视图名、序列名和存储过程名或其它对象名。该方法对具有特别长的对象的名字很有用。

4.2 创建和删除同义词

4.2.1 创建同义词

```
CREATE SYNONYM d_sum  
FOR dept_sum_vu;  
Synonym Created.
```

4.2.1.1 示例

```
select * from em;
```

4.2.2 删除同义词

```
DROP SYNONYM d_sum;  
Synonym dropped.
```

4.2.2.1 示例

```
drop synonym em;
```

5 创建用户(User)

5.1 什么是 Oracle 用户

Oracle 用户是用来连接数据库和访问数据库对象的。

5.2 操作用户

5.2.1 创建用户

需要具备创建用户的权限可以使用 sys 或者 system 用户来创建新用户

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

5.2.1.1 使用默认表空间

5.2.1.1.1示例

创建一个用户名为 u_test，永久表空间使用 Oracle 默认的永久表空间。

```
create user u_test identified by  
oracle;
```

5.2.1.2 使用指定表空间

5.2.1.2.1示例

创建一个用户名为 u_bjsxt, 使用 bjsxt 表空间为他的表空间。

```
create user u_bjsxt identified by  
oracle default tablespace bjsxt temporary  
tablespace temp;
```

5.2.2 删除用户

5.2.2.1 示例

删除 u_test 用户

```
drop user u_test;
```

5.2.2.2 删除用户的同时将该用户下的其他对象一并删掉

```
drop user u_test cascade;
```

十二、 数据控制语言(DCL)

grant...授予用户权限

revoke..撤销用户权限

1 授予系统权限

```
GRANT create session, create table,  
       create sequence, create view  
TO      scott;  
Grant succeeded.
```

1.1 授予创建其他对象权限

- CREATE SESSION
- CREATE TABLE
- CREATE SEQUENCE
- CREATE VIEW
- CREATE PROCEDURE
- UNLIMITED TABLESPACE

1.1.1示例

为 u_bjsxt 用户分配创建表、创建视图、创建序列权限以及使用永久表空间权限。

```
grant create table ,create view ,create  
sequence ,unlimited tablespace to u_bjsxt;
```

2 撤销权限

Revoke 权限 from 用户。

2.1示例

撤销 u_bjsxt 用户创建表的权限。

```
revoke create table from u_bjsxt;
```

3 Oracle 中的角色

3.1 什么是角色

角色是命名的可以授予用户的相关权限的组，该方法使得授予、撤回和维护权限容易的多。一个用户可以使用几个角色，并且几个用户也可以被指定相同的角色。

3.2 创建角色并且授予权限给角色

3.2.1 创建角色

```
CREATE ROLE manager;
Role created.
```

3.2.1.1 示例

创建一个名称为 manager 的角色。

```
create role manager;
```

3.2.2 授予权限给一个角色

```
GRANT create table, create view
TO manager;
Grant succeeded.
```

3.2.2.1 示例一

向 manager 角色中添加创建会话、创建表、创建视图、创建序列。

```
grant create session,create table,create
view,create sequence to manager;
```

3.2.3 授予一个角色给用户

```
GRANT manager TO DEHAAN, KOCHHAR;
Grant succeeded.
```

3.2.3.1 示例一

创建一个名称为 newbjsxt 用户密码为 oracle。该用户使用 bjsxt 表空间。

```
create user newbjsxt identified by oracle
default tablespace bjsxt;
```

3.2.3.2 示例二

为用户分配可以无限制的使用永久表空间。

```
grant unlimited tablespace to newbjsxt;
```

3.2.3.3 示例三

将 manager 角色分配给 newbjsxt 用户。

```
grant manager to newbjsxt;
```

3.2.3.4 示例四

在 newbjsxt 用户中创建一个测试表，包含一个 id 列类型为整数类型。

```
create table test(id number);
```

3.2.4 撤销用户角色

3.2.4.1 示例

撤销 newbjsxt 用户的 manager 角色。

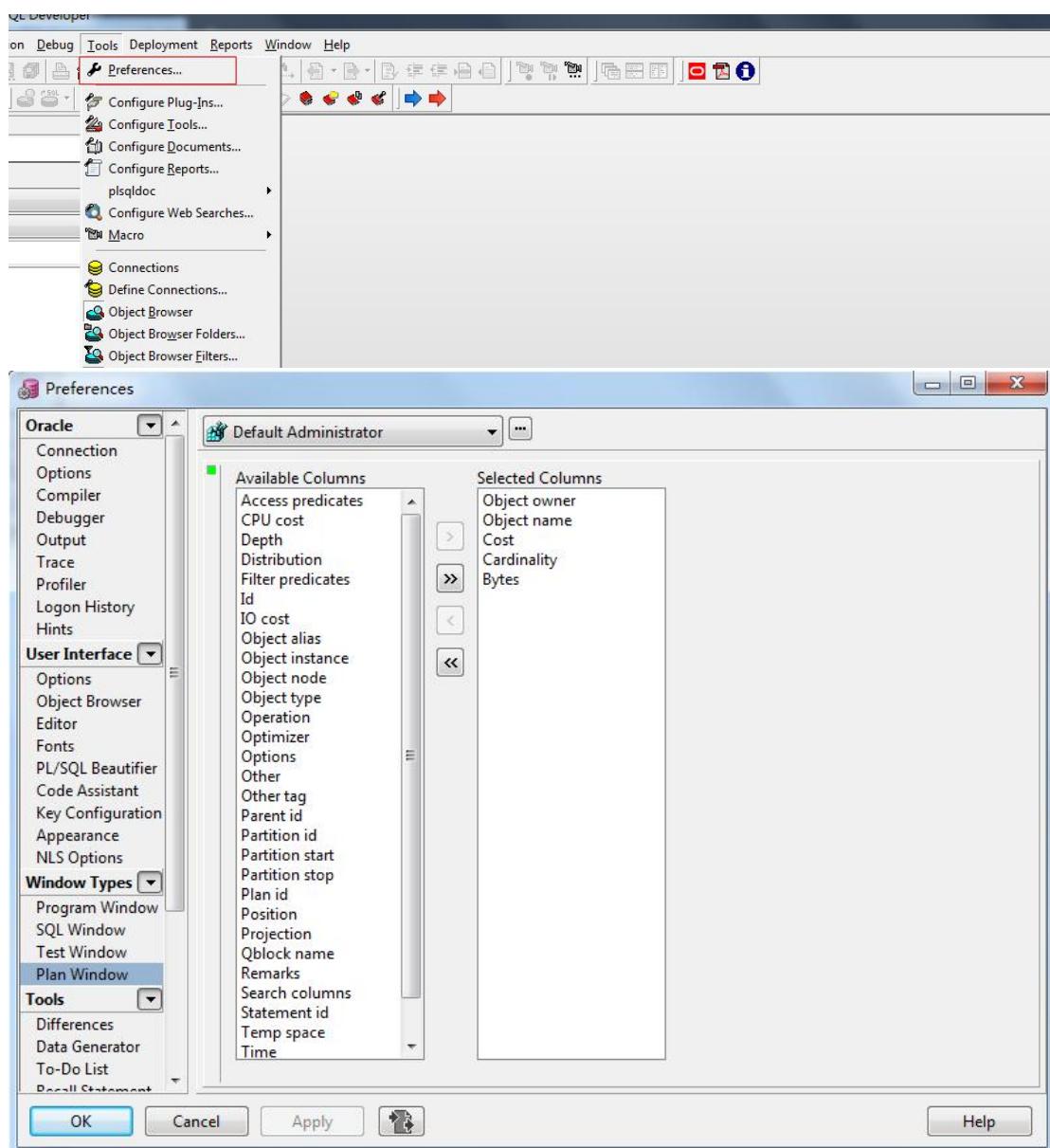
```
revoke manager from newbjsxt;
```

十三、通过 PL/SQL Developer 查看查询的执行计划

1 什么是执行计划

执行计划是一条查询语句在 Oracle 中的执行过程或访问路径的描述。

2 配置执行计划需要显示的项



3 执行计划的常用列字段解释

基数（Cardinality）：Oracle 估计的当前操作的返回结果集行数

字节（Bytes）：执行该步骤后返回的字节数

耗费（COST）、CPU 耗费：Oracle 估计的该步骤的执行成本，用于说明 SQL 执行的代价，理论上越小越好（该值可能与实际有出入）

时间（Time）：Oracle 估计的当前操作所需的时间：

4 使用执行计划

通过工具启动执行计划。选中需要查看执行计划的查询语句，在工具栏中选择

Tools--->Explain Plan

The screenshot shows the Oracle SQL Developer interface with the Explain Plan tool open. The query entered is:

```
select * from employees
```

The Explain Plan results are displayed in a table:

Description	Object owner	Object name	Cost	Cardinality	Bytes
SELECT STATEMENT, GOAL = ALL_ROWS	HR	EMPLOYEES	3	107	14,231
TABLE ACCESS FULL					

或者是选择需要查看执行计划的查询语句后按 F5。

5 查看执行计划

The screenshot shows the Oracle SQL Developer interface with the Explain Plan tool open. The query entered is:

```
select emp.employee_id,emp.last_name,emp.salary from employees emp where emp.salary > (se
```

The Explain Plan results are displayed in a table, with the HASH JOIN SEMI section highlighted by a red box:

Description	Object owner	Object name	Cost	Optimizer	IO cost	Cardinality	Bytes
SELECT STATEMENT, GOAL = ALL_ROWS			9	ALL_ROWS	8	5	150
HASH JOIN SEMI			6		5	5	150
TABLE ACCESS FULL	HR	EMPLOYEES	3	ANALYZED	3	5	95
SORT AGGREGATE						1	4
TABLE ACCESS FULL	HR	EMPLOYEES	3	ANALYZED	3	107	428
VIEW	HR	index\$join\$...	3			2	55
HASH JOIN							
INDEX FAST FULL SCAN	HR	EMP_DEPAR...	1	ANALYZED	1	5	55
INDEX FAST FULL SCAN	HR	EMP_NAME...	1	ANALYZED	1	5	55

5.1 执行顺序

缩进最多的最先执行；（缩进相同时，最上面的最先执行）。

5.2 表访问的几种方式（非全部）：

- TABLE ACCESS FULL (全表扫描)
- TABLE ACCESS BY INDEX ROWID (通过 ROWID 的表存取)
- TABLE ACCESS BY INDEX SCAN (索引扫描)

5.2.1 TABLE ACCESS FULL (全表扫描)

Oracle 会读取表中所有的行，并检查每一行是否满足 SQL 语句中的 Where 限制条件；
使用建议：数据量太大的表不建议使用全表扫描，除非本身需要取出的数据较多，占到表数据总量的 5%~10% 或以上

5.2.2 TABLE ACCESS BY INDEX ROWID（通过 ROWID 的表存取）

5.2.2.1 什么是 ROWID

ROWID 是由 Oracle 自动加在表中每行最后的一列伪列，既然是伪列，就说明表中并不会物理存储 ROWID 的值。

你可以像使用其它列一样使用它，只是不能对该列的值进行增、删、改操作。

一旦一行数据插入后，则其对应的 ROWID 在该行的生命周期内是唯一的，即使发生行迁移，该行的 ROWID 值也不变。

5.2.2.2 TABLE ACCESS BY INDEX ROWID

行的 ROWID 指出了该行所在的数据文件、数据块以及行在该块中的位置，所以通过 ROWID 可以快速定位到目标数据上，这也是 Oracle 中存取单行数据最快的方法；

5.2.3 TABLE ACCESS BY INDEX SCAN（索引扫描）

在索引块中，既存储每个索引的键值，也存储具有该键值的行的 ROWID。

5.2.3.1 索引扫描其实分为两步

- 1 扫描索引得到对应的 ROWID。
- 2 通过 ROWID 定位到具体的行读取数据。

5.2.3.2 五种索引扫描

- INDEX UNIQUE SCAN（索引唯一扫描）
- INDEX RANGE SCAN（索引范围扫描）
- INDEX FULL SCAN（索引全扫描）
- INDEX FAST FULL SCAN（索引快速扫描）
- INDEX SKIP SCAN（索引跳跃扫描）

5.2.3.2.1 INDEX UNIQUE SCAN（索引唯一扫描）

针对唯一性索引（UNIQUE INDEX）的扫描，每次至多只返回一条记录；
表中某字段存在 UNIQUE、PRIMARY KEY 约束时，Oracle 常实现唯一性扫描；

5.2.3.2.2 INDEX RANGE SCAN（索引范围扫描）

使用一个索引存取多行数据；

发生索引范围扫描的三种情况：

- 1 在唯一索引列上使用了范围操作符（如：> < <> >= <= between）；
- 2 在组合索引上，只使用部分列进行查询（查询时必须包含前导列，否则会走全表扫描）；
- 3 对非唯一索引列上进行的任何查询；

5.2.3.2.3 INDEX FULL SCAN (索引全扫描)

进行全索引扫描时，查询出的数据都必须从索引中可以直接得到；

5.2.3.2.4 INDEX FAST FULL SCAN (索引快速扫描)

扫描索引中的所有的数据块，与 INDEX FULL SCAN 类似，但是一个显著的区别是它不对查询出的数据进行排序（即数据不是以排序顺序被返回）；

5.2.3.2.5 INDEX SKIP SCAN (索引跳跃扫描)

表有一个复合索引，且在查询时有除了前导列（索引中第一列）外的其他列作为条件；

5.2.4 Oracle 的优化器

5.2.4.1 Oracle 的优化器种类

- RBO (Rule-Based Optimization) 基于规则的优化器
- CBO (Cost-Based Optimization) 基于代价的优化器

5.2.4.2 RBO 优化器

RBO 有严格的使用规则，只要按照这套规则去写 SQL 语句，无论数据表中的内容怎样，也不会影响到你的执行计划。换句话说，RBO 对数据“不敏感”，它要求 SQL 编写人员必须要了解各项细则。RBO 一直沿用至 ORACLE 9i，从 ORACLE 10g 开始，RBO 已经彻底被抛弃。

5.2.4.3 CBO 优化器

CBO 是一种比 RBO 更加合理、可靠的优化器，在 ORACLE 10g 中完全取代 RBO。CBO 通过计算各种可能的执行计划的“代价”，即 COST，从中选用 COST 最低的执行方案作为实际运行方案。

6 执行计划的使用

6.1 单表

6.1.1 分析查询表中的所有数据

6.1.1.1 示例

查询 employees 表中的所有数据

```
select * from employees
```

Optimizer goal: All rows

Description	Object owner	Object name	Cost	Optimizer	IO cost	Cardinality	Bytes
SELECT STATEMENT, GOAL = ALL_ROWS			3	ALL_ROWS	3	107	7,383
TABLE ACCESS FULL	HR	EMPLOYEES	3	ANALYZED	3	107	7,383

6.1.2 分析主键作为条件的查询

6.1.2.1 示例

查询 employees 表中 employee_id 为 100 的雇员

```
select * from employees e where e.employee_id = 100
```

Optimizer goal: All rows

Description	Object owner	Object name	Cost	Optimizer	IO cost	Cardinality	Bytes
SELECT STATEMENT, GOAL = ALL_ROWS			1	ALL_ROWS	1	1	69
TABLE ACCESS BY INDEX ROWID	HR	EMPLOYEES	1	ANALYZED	1	1	69
INDEX UNIQUE SCAN	HR	EMP_EMP_I...	0	ANALYZED	0	1	

6.1.3 分析非主键列作为查询条件

6.1.3.1 示例

查询雇员名字为 Tarloy 的雇员

```
select * from employees e where e.last_name = 'Taylor'
```

Optimizer goal: All rows

Description	Object owner	Object name	Cost	Optimizer	IO cost	Cardinality	Bytes
SELECT STATEMENT, GOAL = ALL_ROWS			3	ALL_ROWS	3	1	69
TABLE ACCESS FULL	HR	EMPLOYEES	3	ANALYZED	3	1	69

6.1.4分析 like 条件

6.1.4.1 示例

查询雇员名字中含有 a 的雇员

```
select * from employees e where e.last_name like '%a%'
```

Optimizer goal All rows							
Description	Object owner	Object name	Cost	Optimizer	IO cost	Cardinality	Bytes
SELECT STATEMENT, GOAL = ALL_ROWS			3	ALL_ROWS	3	5	345
TABLE ACCESS FULL	HR	EMPLOYEES	3	ANALYZED	3	5	345

6.1.5分析非唯一性索引列作为条件的查询

6.1.5.1 示例一

将 last_name 创建索引

```
create index emp_name on  
employees(last_name);
```

6.1.5.2 示例二

查询雇员名字为 Taylor 的雇员。

```
select * from employees e where e.last_name = 'Taylor'
```

Optimizer goal All rows							
Description	Object owner	Object name	Cost	Optimizer	IO cost	Cardinality	Bytes
SELECT STATEMENT, GOAL = ALL_ROWS			2	ALL_ROWS	2	1	69
TABLE ACCESS BY INDEX ROWID	HR	EMPLOYEES	2	ANALYZED	2	1	69
INDEX RANGE SCAN	HR	EMP_NAME	1	ANALYZED	1	1	

6.1.6分析非唯一性索引中=、>、<、<>条件

6.1.6.1 示例一

对雇员薪水列做=、>、<、<>条件判断。

```
select * from employees e where e.salary = 3000
```

Optimizer goal All rows							
Description	Object owner	Object name	Cost	Optimizer	IO cost	Cardinality	Bytes
SELECT STATEMENT, GOAL = ALL_ROWS			3	ALL_ROWS	3	2	138
TABLE ACCESS FULL	HR	EMPLOYEES	3	ANALYZED	3	2	138

6.1.6.2 示例二

对薪水创建索引，使用薪水列做=、>、<、<>条件判断。

```
select * from employees e where e.salary = 3000
```

Optimizer goal All rows							
Description	Object owner	Object name	Cost	Optimizer	IO cost	Cardinality	Bytes
SELECT STATEMENT, GOAL = ALL_ROWS			2	ALL_ROWS	2	2	138
TABLE ACCESS BY INDEX ROWID	HR	EMPLOYEES	2	ANALYZED	2	2	138
INDEX RANGE SCAN	HR	EMP_SAL	1	ANALYZED	1	2	

```
select * from employees e where e.salary <> 3000
```

Optimizer goal All rows							
Description	Object owner	Object name	Cost	Optimizer	IO cost	Cardinality	Bytes
SELECT STATEMENT, GOAL = ALL_ROWS			3	ALL_ROWS	3	105	7,245
TABLE ACCESS FULL	HR	EMPLOYEES	3	ANALYZED	3	105	7,245

```
select * from employees e where e.salary < 3000
```

Optimizer goal All rows							
Description	Object owner	Object name	Cost	Optimizer	IO cost	Cardinality	Bytes
SELECT STATEMENT, GOAL = ALL_ROWS			2	ALL_ROWS	2	4	276
TABLE ACCESS BY INDEX ROWID	HR	EMPLOYEES	2	ANALYZED	2	4	276
INDEX RANGE SCAN	HR	EMP_SAL	1	ANALYZED	1	4	

```
select * from employees e where e.salary > 3000
```

Optimizer goal All rows							
Description	Object owner	Object name	Cost	Optimizer	IO cost	Cardinality	Bytes
SELECT STATEMENT, GOAL = ALL_ROWS			3	ALL_ROWS	3	103	7,107
TABLE ACCESS FULL	HR	EMPLOYEES	3	ANALYZED	3	103	7,107

6.2 多表

6.2.1 内连接

6.2.1.1 示例一

使用等值连接，查询所有部门以及所有部门下的雇员。

```
select * from employees e,departments d where e.department_id = d.department_id
```

Optimizer goal All rows							
Description	Object owner	Object name	Cost	Optimizer	IO cost	Cardinality	Bytes
SELECT STATEMENT, GOAL = ALL_ROWS			6	ALL_ROWS	5	106	9,540
MERGE JOIN			6		5	106	9,540
TABLE ACCESS BY INDEX ROWID	HR	DEPARTME...	2	ANALYZED	2	27	
INDEX FULL SCAN	HR	DEPT_ID_PK	1	ANALYZED	1	27	567
SORT JOIN			4		3	107	7,383
TABLE ACCESS FULL	HR	EMPLOYEES	3	ANALYZED	3	107	7,383

6.2.1.2 示例二

使用子查询，查询所有部门以及所有部门下的雇员。

```
select * from employees e where e.department_id in (select d.department_id from depa...
```

Optimizer goal All rows							
Description	Object owner	Object name	Cost	Optimizer	IO cost	Cardinality	Bytes
SELECT STATEMENT, GOAL = ALL_ROWS			3	ALL_ROWS	3	106	7,314
TABLE ACCESS FULL	HR	EMPLOYEES	3	ANALYZED	3	106	7,314

6.2.2 外连接

6.2.2.1 示例一

查询所有雇员与雇员的部门名称，包含那些没有部门的雇员。

```
select * from employees e left outer join departments d on e.department_id = d.department_id
```

Description	Object owner	Object name	Cost	Optimizer	IO cost	Cardinality	Bytes
SELECT STATEMENT, GOAL = ALL_ROWS			7	ALL_ROWS	6	107	9,630
HASH JOIN RIGHT OUTER			7		6	107	9,630
TABLE ACCESS FULL	HR	DEPARTMENTS	3	ANALYZED	3	27	567
TABLE ACCESS FULL	HR	EMPLOYEES	3	ANALYZED	3	107	7,383

6.2.2.2 示例二

查询所有部门以及雇员，包含哪些没有雇员的部门。

```
from employees e right outer join departments d on e.department_id = d.department_id
```

Description	Object owner	Object name	Cost	Optimizer	IO cost	Cardinality	Bytes
SELECT STATEMENT, GOAL = ALL_ROWS			6	ALL_ROWS	5	106	9,540
MERGE JOIN OUTER			6		5	106	9,540
TABLE ACCESS BY INDEX ROWID	HR	DEPARTMENTS	2	ANALYZED	2	27	567
INDEX FULL SCAN	HR	DEPT_ID_PK	1	ANALYZED	1	27	
SORT JOIN			4		3	107	7,383
TABLE ACCESS FULL	HR	EMPLOYEES	3	ANALYZED	3	107	7,383

十四、 Oracle 的数据导入与导出

1 数据库导入导出需要注意

1. 目标数据库要与源数据库有着名称相同的表空间。
2. 目标数据在进行导入时，用户名尽量相同(这样保证用户的权限级别相同)。
3. 目标数据库每次在进行数据导入前，应做好数据备份，以防数据丢失。
4. 弄清是导入导出到相同版本还是不同版本(oracle10g 版本与 oracle11g 版本)。
5. 目标数据导入前，弄清楚是数据覆盖(替换)，还是仅插入新数据或替换部分数据表。
6. 确定目标数据库磁盘空间是否足够容纳新数据，是否需要扩充表空间。
7. 导入导出时注意字符集是否相同，一般 Oracle 数据库的字符集只有一个，并且固定，一般不改变。
8. 确定操作者的账号权限。

2 导出数据格式介绍

Dmp 格式：.dmp 是二进制文件，可跨平台，还能包含权限，效率好。

Sql 格式：.sql 格式的文件，可用文本编辑器查看，通用性比较好，效率不如第一种，适合小数据量导入导出。尤其注意的是表中不能有大字段（blob,clob,long），如果有，会报错。

Pde 格式：.pde 格式的文件，.pde 为 PL/SQL Developer 自有的文件格式，只能用 PL/SQL Developer 工具导入导出，不能用文本编辑器查看。

3 传统方式 exp(导出)和imp(导入):

3.1 命令执行方式

该命令需要在操作系统的命令窗口执行，而非 sql/plus
在使用导出或导入命令时，在命令的后侧不要添加分号。

3.2 命令格式

```
exp|imp 用户名 / 密码 @ 连接地址 : 端口 / 服务名 file= 路径 / 文件名 .dmp
full=y|tables(tablename,tablename...)|owner(username1,username2,username3)
exp: 导出命令, 导出时必写。
imp: 导入命令, 导入时必写, 每次操作, 二者只能选择一个执行。
username: 导出数据的用户名, 必写;
password: 导出数据的密码, 必写;
@: 地址符号, 必写;
SERVICENAME: Oracle 的服务名, 必写;
1521: 端口号, 1521 是默认的可以不写, 非默认要写;
file="文件名.dmp": 文件存放路径地址, 必写;
full=y: 表示全库导出。可以不写, 则默认为 no, 则只导出用户下的对象;
tables: 表示只导出哪些表;
owner: 导出该用户下对象;
full|tables|owner: 只能使用一种;
```

3.3 导出数据

```
exp 用户名/密码@oracle 的连接地址:端口/需要导出的服务名 file=路径/文件名.dmp
```

3.3.1示例

将 HR 用户下的对象导出。

```
exp hr/oracle@localhost:1521/orcl file=d:/1.dmp
```

3.4 导入数据

```
imp 用户名/密码@oracle 的连接地址:端口/需要导出的服务名 file=路径/文件名.dmp
```

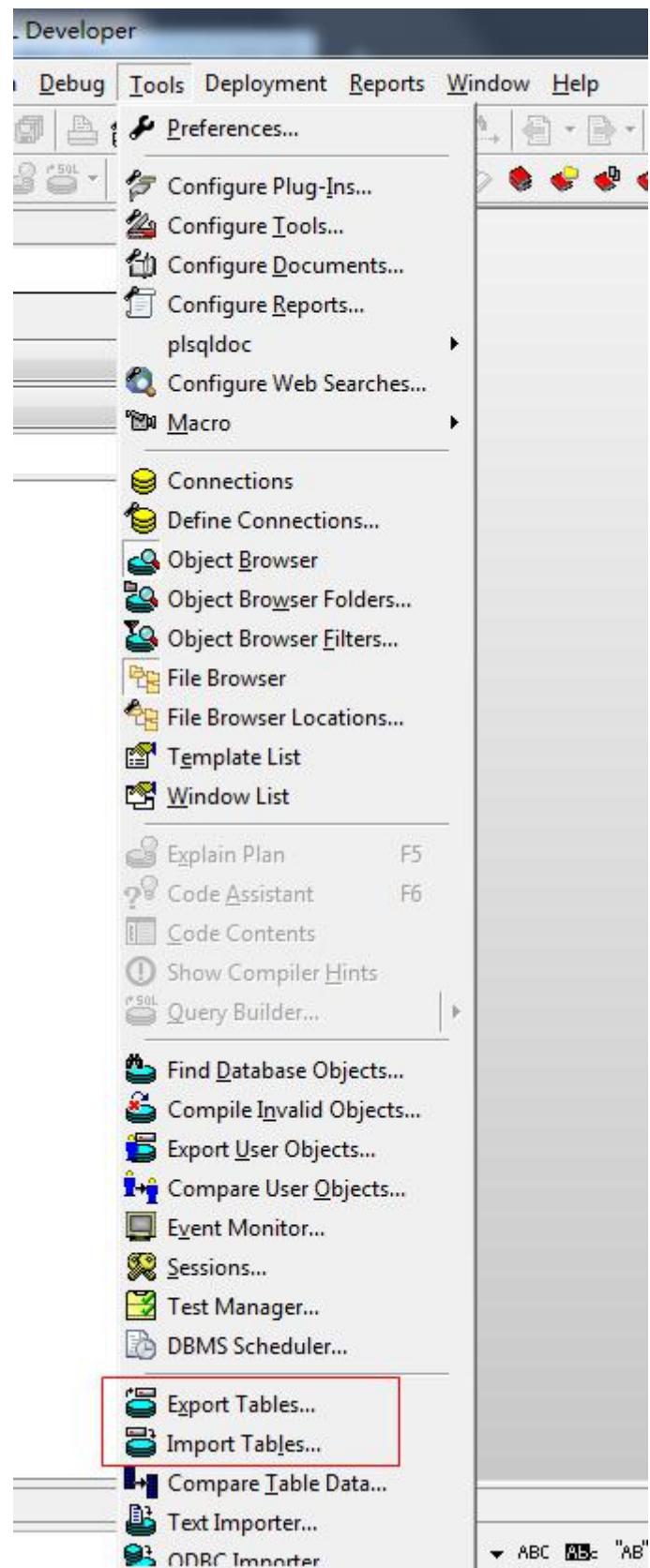
3.4.1示例

将导出的 dmp 文件导入到数据库中。

```
imp hr/oracle@localhost:1521/orcl file=d:/1.dmp
```

4 使用 PL/SQL Developer 实现数据的导入与导出

4.1 导出与导入表对象

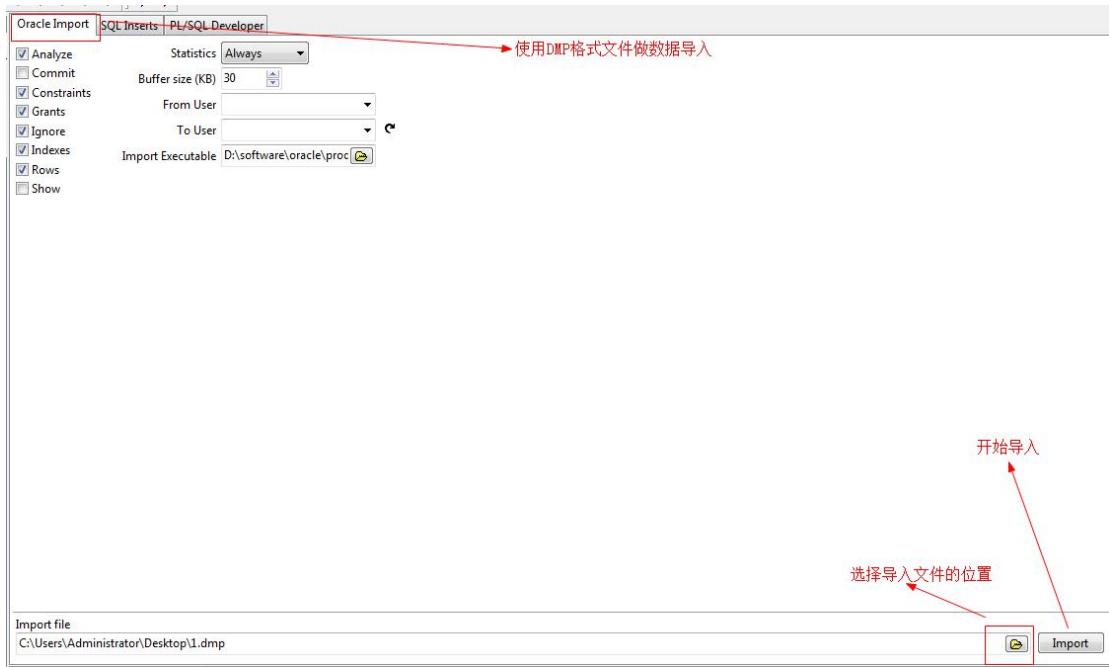


4.1.1 DMP 格式

4.1.1.1 导出 DMP 格式

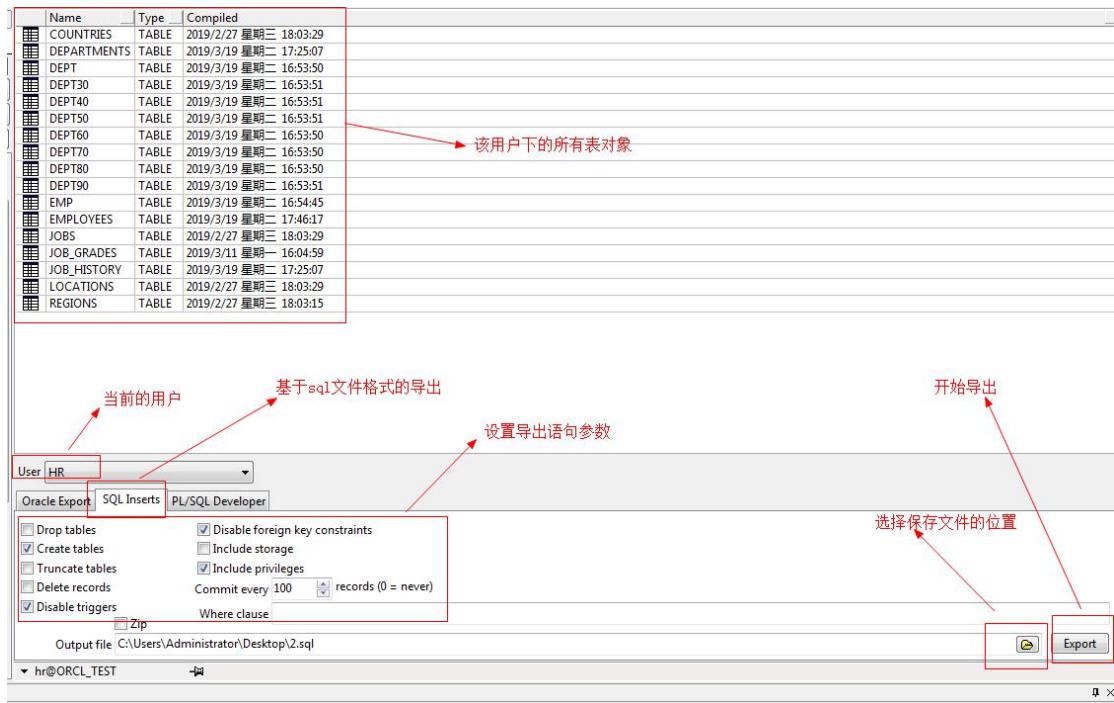


4.1.1.2 导入 DMP 格式



4.1.2 SQL 格式

4.1.2.1 导出 SQL 格式

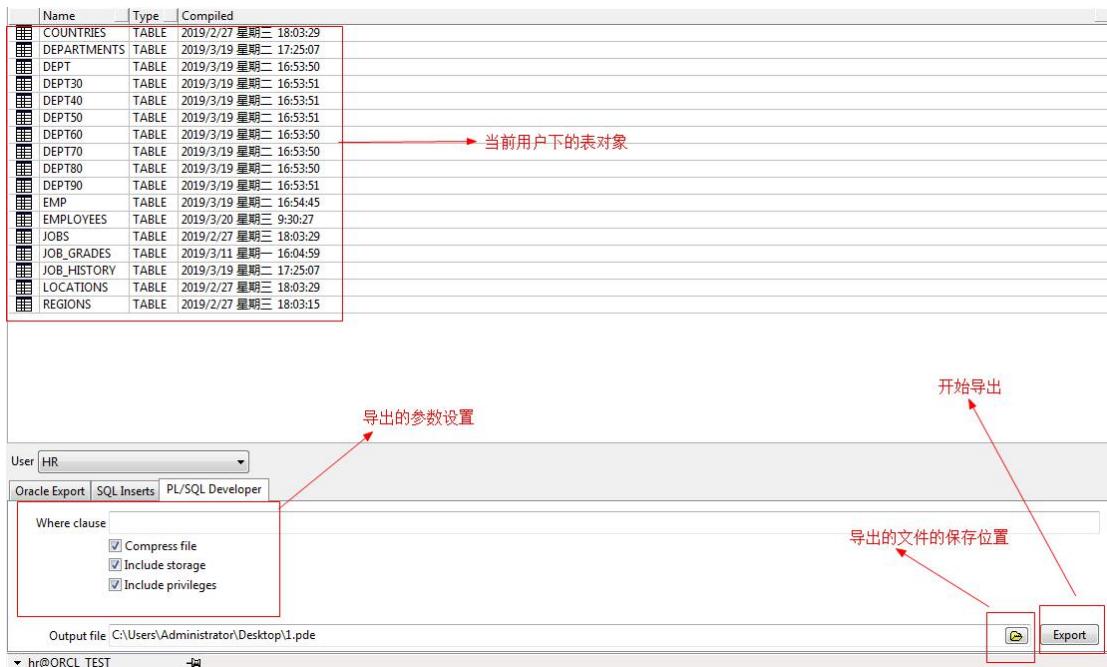


4.1.2.2 导入 SQL 格式

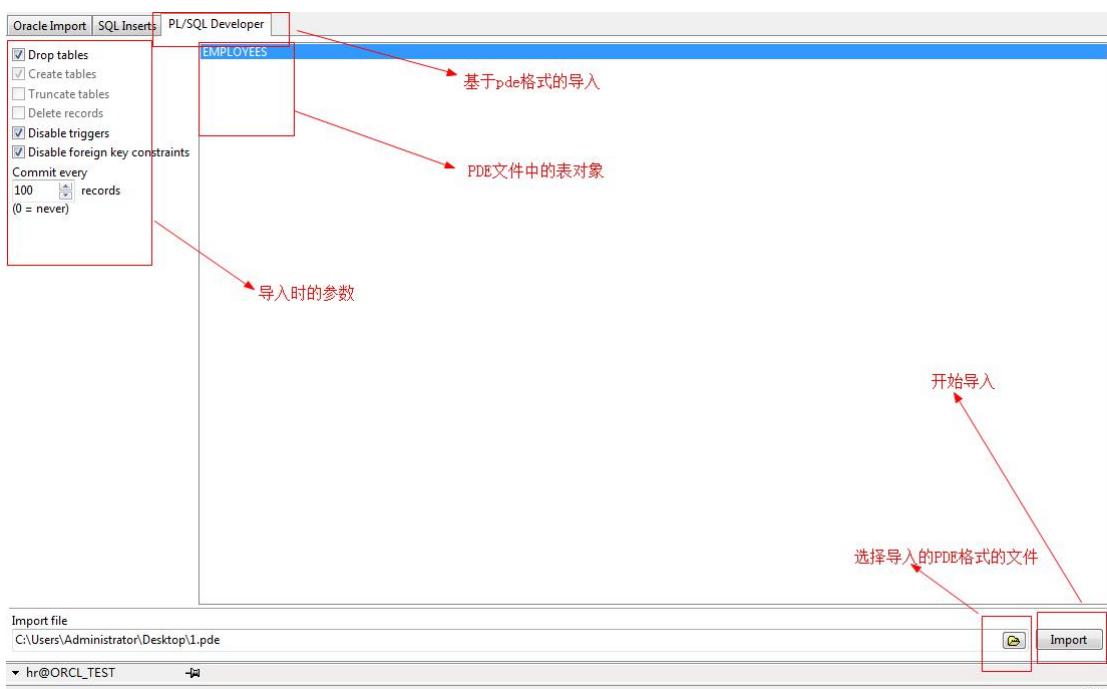


4.1.3 PDE 格式

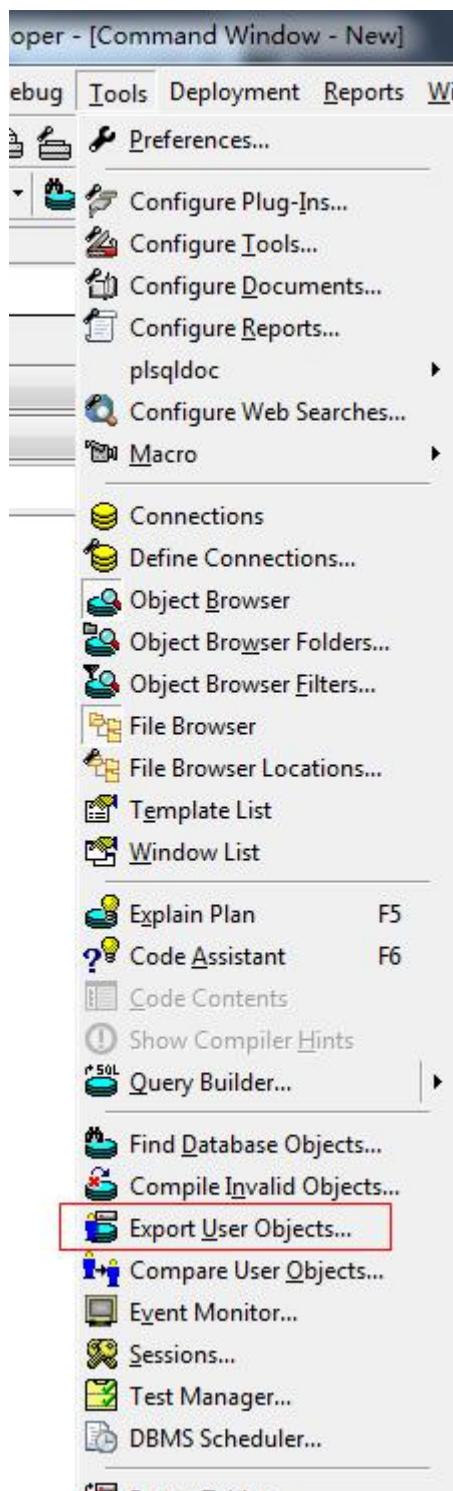
4.1.3.1 导出 PDE 格式



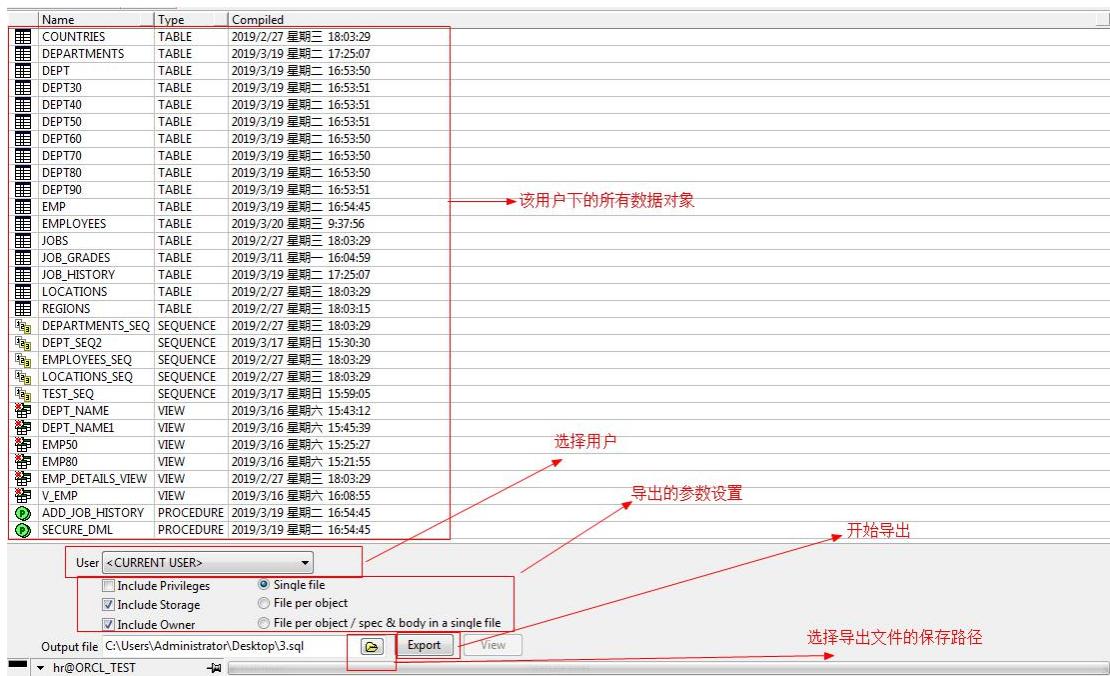
4.1.3.2 导入 PDE 格式



4.2 数据中其他对象的导入与导出



4.2.1 导出数据库对象



4.2.2 导入数据库其他对象

