1. Oracle 基础

1.1. Oracle 简介

Oracle 是殷墟（yīn Xu）出土的甲骨文（oracle bone inscriptions）的英文翻译的第一个单词，在英语里是“神谕”的意思。Oracle 公司成立于 1977 年，总部位于美国加州，是世界领先的信息管理软件开发商，因其复杂的关系数据库产品而闻名。Oracle 数据库产品为财富排行榜上的前 1000 家公司所采用，许多大型网站也选用了 Oracle 系统。

Oracle 数据库是 Oracle（中文名称叫甲骨文）公司的核心产品，Oracle 数据库是一个适合于大中型企业的数据库管理系统。在所有的数据库管理系统中（比如：微软的 SQL Server，IBM 的 DB2 等），Oracle 的主要用户涉及面非常广，包括：银行、电信、移动通信、航空、保险、金融、电子商务和跨国公司等。

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

WebLogic 是美国 bea 公司出品的一个 application server 确切的说是一个基于 Javaee 架构的中间件，BEA WebLogic 是用于开发、集成、部署和管理大型分布式 Web 应用、网络应用和数据库应用的 Java 应用服务器。2008 年 1 月 16 日，全球最大的数据库软件公司甲骨文（Oracle）宣布已经同 BEA 达成协议，以 85 亿美元收购 BEA。

2008 年 1 月 16 日，Sun 宣布已经与 MySQL AB 达成协议，以大约 10 亿美元收购 MySQL AB。

2009 年 04 月 20 日，甲骨文宣布，该公司将以每股 9.5 美元的价格收购 Sun。该交易价值约为 74 亿美元。

Oracle 的官方网站：http://www.oracle.com

1.2. Oracle 相关的参考文档

文档：
http://www.oracle.com/technetwork/database/database10g/documentation/index.html

在线：
http://www.oracle.com/pls/db102/homepage

Linux 上安装 Oracle 10g：
http://69520.blog.51cto.com/59520/91156

Oracle 数据库监听配置：
http://blog.csdn.net/tianlesoftware/article/details/4861572

1.3. Oracle 中的一些概念

1.3.1. Oracle 数据库

位于硬盘上实际存放数据的文件，这些文件组织在一起，成为一个逻辑整体，即为 Oracle 数据库。因此在 Oracle 看来，“数据库”是指硬盘上文件的逻辑集合，必须要与内存里实例合作，才能对外提供数据管理服务。

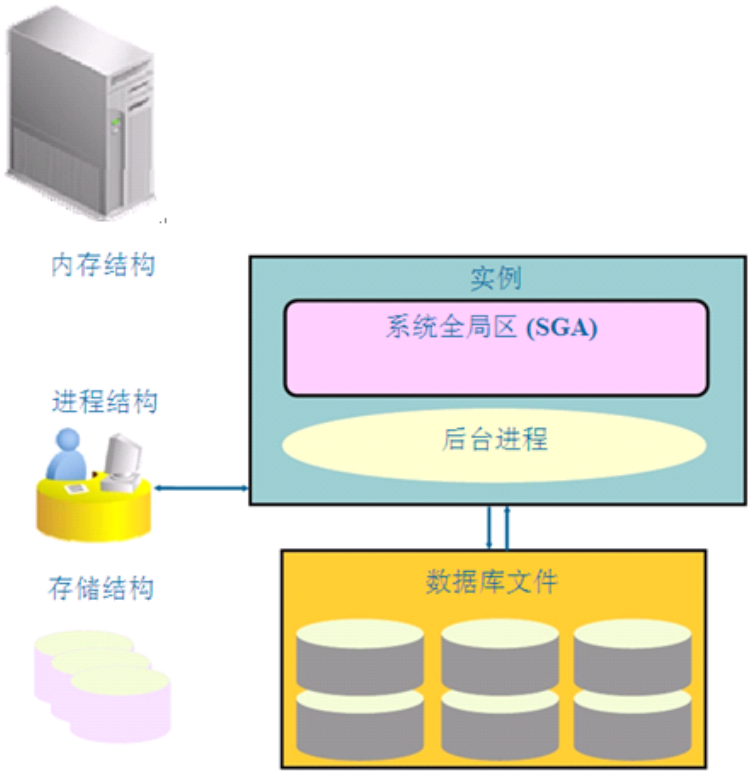
1.3.2. Oracle 实例

位于物理内存里的数据结构。它由一个共享的内存池和多个后台进程所组成，共享的内存池可以被所有进程访问。用户如果要存取数据库(也就是硬盘上的文件) 里的数据，必须通过实例才能实现，不能直接读取硬盘上的文件。

1.3.3. Oracle 服务器

一个 Oracle 服务器：是一个数据管理系统（RDBMS），它提供开放的，全面的，近乎完整的信息管理。由一个 Oracle 实例 和一个 Oracle 数据库组成。

实例可以操作数据库，在任何时刻一个实例只能与一个数据库关联。大多数情况下，一个数据库上只有一个实例对其进行操作（也可以有多个实例）。



1.3.4. 数据库的逻辑和物理结构

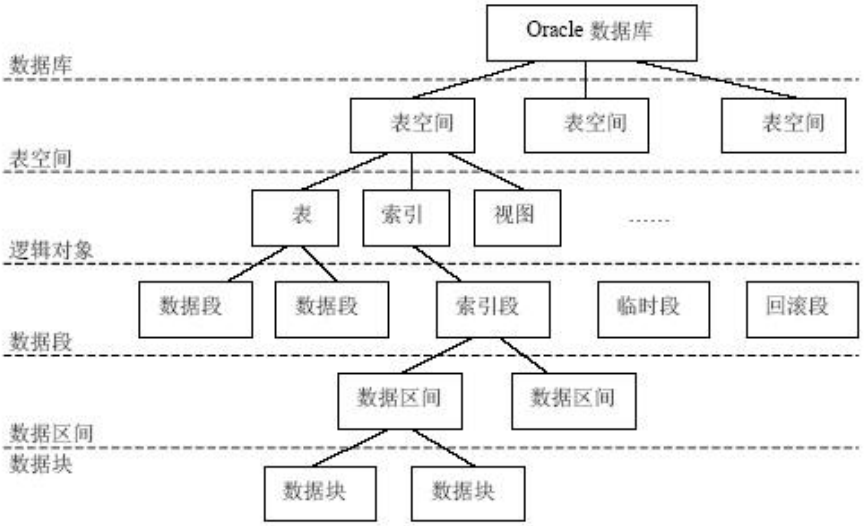


图 2.48 数据库的逻辑结构

- 表空间由多个数据文件组成
- 数据文件只能属于一个表空间
- 表空间为逻辑概念，数据文件为物理概念

- 段存在于表空间中
- 段是区的集合
- 区是数据块的集合
- 数据块会被映射到磁盘块

2． 搭建 Oracle 环境（安装与配置）

2.1. 系统需求

内存需求：
　　1 GB
磁盘空间需求：
　　Oracle 软件需要 1.5 GB 到 3.5 GB
操作系统：
　　根据手册文档而定

2.2. 安装 Oracle 服务器端

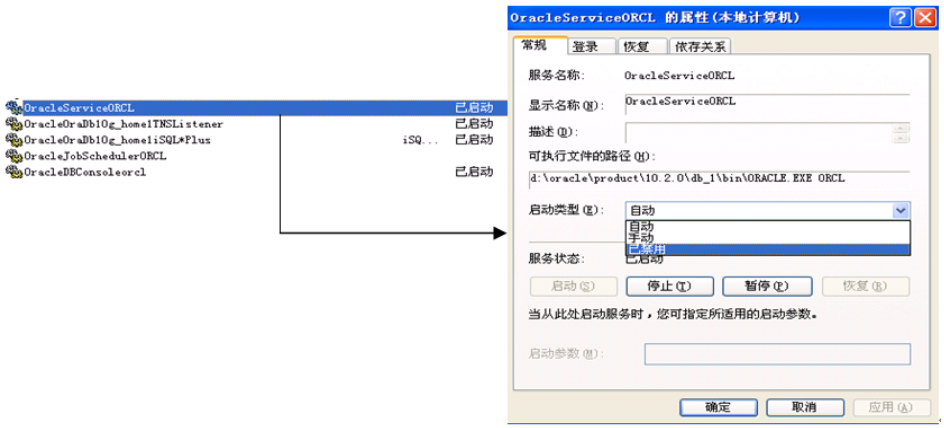
请参见相关文档：
1，《在 Windows XP 中安装 Oracle10g》
2，《在 Win7 或 Vista 中安装 Oracle11g》
3，《在 Linux 中安装 Oracle》
4，《虚拟机中安装 Oracle》

　　全局数据库名是数据库在服务器网络中的唯一标识。

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

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

Oracle 数据库是一个庞大的软件，启动它会占有大量的内存和 CPU 资源。如果不想让 Oracle 数据库自动启动，可做如下设置：



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

2.3. 使用自带工具 SqlPlus 操作 Oracle 数据库的基础

2.3.1. 操作 Oracle 的工具

- Sqlplusw.exe，命令行程序。
- iSql*Plus，Web 程序。
- 其他图形化工具。

使用 iSQL*Plus 可以：

- 描述表结构。
- 编辑 SQL 语句。
- 执行 SQL 语句。
- 将 SQL 保存在文件中并将 SQL 语句执行结果保存在文件中。

- 在保存的文件中执行语句。
- 将文本文件装入 SQL*Plus 编辑窗口。
- 以本机为例:http://localhost:5560/isqlplus/

2.3.2. SQL 语句说明

- SQL 语言大小写不敏感。
- SQL 可以写在一行或者多行（使用时最后要以分号结尾，表示一条 SQL 语句）。
- 关键字不能被缩写也不能分行
- 格式:
  各子句一般要分行写。
  使用缩进提高语句的可读性。

2.3.3. 一些 SQL*Plus 命令

说明：命令不区分大小写。

2.3.3.1. 登录、注销

- 登录普通用户：
  方式一：执行 sqlplus --> 输入用户名 --> 输入密码。
  方式二：执行 sqlplus {用户名} --> 输入密码。
  方式三：执行 sqlplus {用户名}/{密码}。
  例子：
  sqlplus --> 输入 scott --> 输入 tiger。
  sqlplus scott --> 输入密码。
  sqlplus scott/tiger。

- 登录管理员：
  执行 sqlplus / as sysdba

- 退出：
  exit

说明：用户名不区分大小写，密码区分。

2.3.3.2. 用户锁定、解锁、修改密码

- 解锁用户：
  alter user 用户名 account unlock;
- 锁定用户：
  alter user 用户名 account lock;
- 修改密码：
  alter user 用户名 identified by 新密码;
- 修改管理员密码：
  alter user sys identified by 新密码;

2.3.3.3. 加载脚本文件

- 加载脚本文件：
  例：@c:/a.sql

2.3.3.4. 查看与设置参数

- 查看参数
  格式：
  show {show 选项}
  例子：
  show pagesize
  show linesize

作用：

显示参数目前的值。

- 设置参数的值

格式：

set {set 选项} {新值}

例子：

set linesize 110

set pagesize 30

作用：

设置参数的值，通过这种方式设置的参数值只对本次登录有效。

- 永久保存设置的参数

如果希望能永久保存设置的参数，可以去修改文件：

oracleHome\product\10.2.0\db_2\sqlplus\admin\glogin.sql。

2.3.3.5. 中止正在执行的命令

在命令行的 SqlPlus 中，中止一个正在执行的命令是 Ctrl+/, Ctrl + C，如果直接按 Ctrl+C 会退出 SqlPlus 程序。

在 sqlplus.exe（单独运行的程序）中，中止一个正在执行的命令是 Ctrl + C。

2.3.3.6. Oracle 启动和关闭

必须是 sys 用户，命令为：

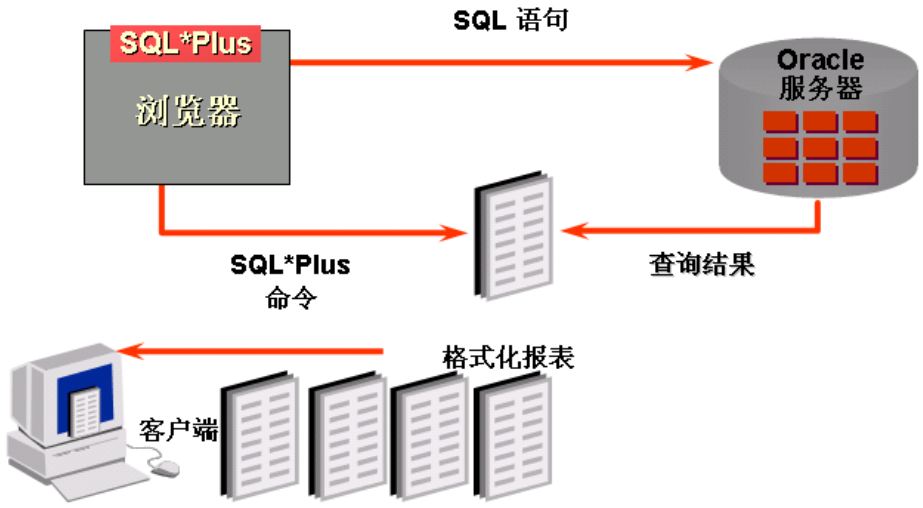
启动：

startup open

关闭：

shutdown immediate

2.3.4. SQL 语句与 SqlPlus 命令



SQL	SQL*Plus
• 一种语言 • ANSI 标准 • 关键字不能缩写 • 使用语句控制数据库中的表的定义信息和表中的数据	• 一种环境 • Oracle 的特性之一 • 关键字可以缩写 • 命令不能改变数据库中的数据的值

2.4. Oracle 相关的服务

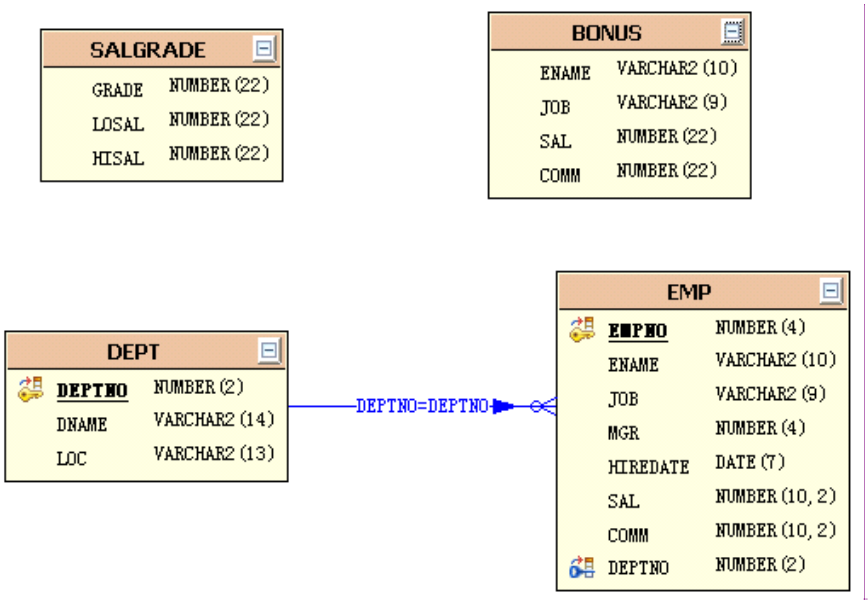
OracleService+服务名，该服务是数据库启动的基础，只有该服务启动了，Oracle 数据库才能正常启动。这是必须启动的服务。

OracleOraDb10g_home1TNSListener，该服务是服务器端为客户端提供的监听服务，只有该服务在服务器上正常启动，客户端才能连接到服务器。该监听服务接收客户端发出的请求，然后将请求传递给数据库服务器。一旦建立了连接，客户端和数据库服务器就能直接通信了。

OracleOraDb10g_home1iSQL*Plus，该服务提供了用浏览器对数据库中数据操作的方式。该服务启动后，就可以使用浏览器进行远程登录并进行数据库操作了。如下图所示：

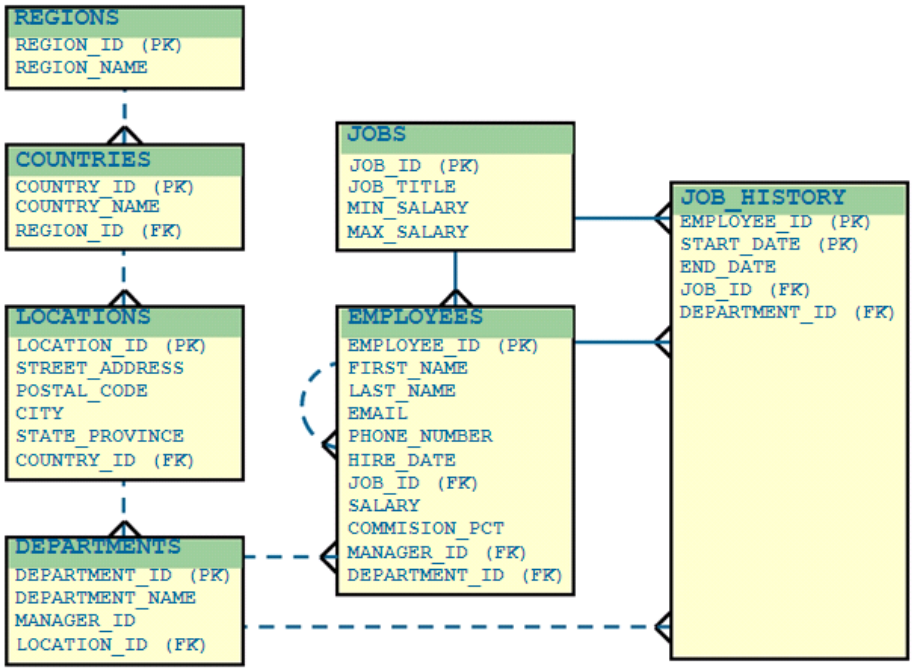
2.5. 贯穿这门课程的方案（scott 方案）

Scott 方案：



批注 [t1]: Bonus 是满怀奖金表。DEPT（部门表），EMP （员工表 ），SALGRADE(工资等级表）

Hr 方案：



3. 查询数据（基本查询）

3.1. 查询（select .. form ..）

3.1.1. 语法

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

- SELECT 标识 选择哪些列。
- FROM 标识从哪个表中选择。

3.1.2. 简单查询（全部列与指定列）

3.1.2.1. 选择全部列

```
SELECT *
FROM   departments;
```

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

8 rows selected.

3.1.2.2. 选择特定的列

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

DEPARTMENT_ID	LOCATION_ID
10	1700
20	1800
50	1500
60	1400
80	2500
90	1700
110	1700
190	1700

8 rows selected.

3.1.3. 使用数学运算符

- 乘除的优先级高于加减

- 优先级相同时，按照从左至右运算
- 可以使用括号改变优先级

示例 1：基本使用

```
SELECT last_name, salary, salary + 300
FROM   employees;
```

LAST_NAME	SALARY	SALARY+300
King	24000	24300
Kochhar	17000	17300
De Haan	17000	17300
Hunold	9000	9300
Ernst	6000	6300
...		
Hartstein	13000	13300
Fay	6000	6300
Higgins	12000	12300
Gietz	8300	8600

20 rows selected.

示例 2：优先级

```
SELECT last_name, salary, 12*salary+100
FROM   employees;
```

LAST_NAME	SALARY	12*SALARY+100
King	24000	288100
Kochhar	17000	204100
De Haan	17000	204100
Hunold	9000	108100
Ernst	6000	72100
...		
Hartstein	13000	156100
Fay	6000	72100
Higgins	12000	144100
Gietz	8300	99700

20 rows selected.

示例 3：使用括号

```
SELECT last_name, salary, 12*(salary+100)
FROM   employees;
```

LAST_NAME	SALARY	12*(SALARY+100)
King	24000	289200
Kochhar	17000	205200
De Haan	17000	205200
Hunold	9000	109200
Ernst	6000	73200
...		
Hartstein	13000	157200
Fay	6000	73200
Higgins	12000	145200
Gietz	8300	100800

20 rows selected.

3.1.4. 空值（null）与对空值的处理

- 空值是无效的，未指定的，未知的或不可预知的值
- 空值不是空格或者 0。
- 包含空值的数学表达式的值都为空值

示例：

```
SELECT last_name, job_id, salary, commission_pct
FROM     employees;
```

LAST_NAME	JOB_ID	SALARY	COMMISSION_PCT
King	AD_PRES	24000	
Kochhar	AD_VP	17000	
...			
Zlotkey	SA_MAN	10500	.2
Abel	SA_REP	11000	.3
Taylor	SA_REP	8600	.2
...			
Gietz	AC_ACCOUNT	8300	

20 rows selected.

示例：空值在数学运算中的使用

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

LAST_NAME	12*SALARY*COMMISSION_PCT
King	
Kochhar	
...	
Zlotkey	25200
Abel	39600
Taylor	20640
...	
Gietz	

20 rows selected.

3.1.5. 列的别名

定义：

- 重命名一个列。
- 便于计算。
- 紧跟列名，**也可以在列名和别名之间加入关键字'AS'，别名使用双引号，以便在别名中包含空格或特殊的字符并区分大小写。**
- AS 可以省略

示例 1：

```
SELECT last_name AS name, commission_pct comm
FROM     employees;
```

NAME	COMM
King	
Kochhar	
De Haan	

...

20 rows selected.

示例 2：

```
SELECT last_name "Name", salary*12 "Annual Salary"
FROM     employees;
```

Name	Annual Salary
King	288000
Kochhar	204000
De Haan	204000

...
20 rows selected.

3.1.6. 使用连接符（连接字符串或列）

连接符：

批注 [t2]: 也可以使用 concat()函数

- 把列与列，列与字符连接在一起。
- 用 '||'表示。
- 可以用来'合成'列。

示例：

```
SELECT last_name||job_id AS "Employees"
FROM   employees;
```

Employees
KingAD_PRES
KochharAD_VP
De HaanAD_VP
HunoldIT_PROG
ErnstIT_PROG
LorentzIT_PROG
MourgosST_MAN
RajsST_CLERK

...
20 rows selected.

3.1.7. 对字符串的处理

- **字符串可以是 SELECT 列表中的一个字符，数字，日期。**
- **日期和字符只能在单引号中出现。**
- **每当返回一行时，字符串被输出一次。**

```
SELECT last_name || ' is a ' || job_id    AS "Employee Details"
FROM     employees;
```

Employee Details
King is a AD_PRES
Kochhar is a AD_VP
De Haan is a AD_VP
Hunold is a IT_PROG
Ernst is a IT_PROG
Lorentz is a IT_PROG
Mourgos is a ST_MAN
Rajs is a ST_CLERK

...
20 rows selected.

3.1.8. 删除重复的行

- 默认情况下，查询会返回全部行，包括重复行

```
SELECT department_id
FROM     employees;
```

DEPARTMENT_ID
90
90
90
60
60
60
50
50
50

...
20 rows selected.

- 在 `SELECT` 子句中使用关键字 `'DISTINCT'` 删除重复行。

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

DEPARTMENT_ID
10
20
50
60
80
90
110

8 rows selected.

3.2. 过滤（Where）

3.2.1. 语法

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

- 使用 WHERE 子句，将不满足条件的行过滤掉。
- WHERE 子句紧随 FROM 子句。

示例：

```
SELECT employee_id, last_name, job_id, department_id
FROM     employees
WHERE    department_id = 90 ;
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
100	King	AD_PRES	90
101	Kochhar	AD_VP	90
102	De Haan	AD_VP	90

3.2.2. 处理字符串与日期

- 字符和日期要包含在单引号中。

- **字符大小写敏感，日期格式敏感。**
- 默认的日期格式是 DD-MON-RR。

示例：

```
SELECT last_name, job_id, department_id
FROM    employees
WHERE   last_name = 'Whalen';
```

批注 [t3]: 赋值使用 := 符号

3.2.3. 比较运算

3.2.3.1. 简单运算符

操作符	含义
=	等于 (不是 ==)
>	大于
>=	大于、等于
<	小于
<=	小于、等于
<>	不等于 (也可以是 !=)

示例：

```
SELECT last_name, salary
FROM    employees
WHERE   salary <= 3000;
```

LAST_NAME	SALARY
Matos	2600
Vargas	2500

3.2.3.2. 其他运算符

操作符	含义
BETWEEN ...AND...	在两个值之间 (包含边界)
IN(set)	等于值列表中的一个
LIKE	模糊查询
IS NULL	空值

3.2.3.2.1. BETWEEN

- 使用 BETWEEN 运算来显示在一个区间内的值

```
SELECT last_name, salary
FROM   employees
WHERE  salary BETWEEN 2500 AND 3500;
```

Lower limit Upper limit

LAST_NAME	SALARY
Rajs	3500
Davies	3100
Matos	2600
Vargas	2500

3.2.3.2.2. IN

- 使用 IN 运算显示列表中的值。

示例：

```
SELECT employee_id, last_name, salary, manager_id
FROM   employees
WHERE  manager_id IN (100, 101, 201);
```

EMPLOYEE_ID	LAST_NAME	SALARY	MANAGER_ID
202	Fay	6000	201
200	Whalen	4400	101
205	Higgins	12000	101
101	Kochhar	17000	100
102	De Haan	17000	100
124	Mourgos	5800	100
149	Zlotkey	10500	100
201	Hartstein	13000	100

8 rows selected.

3.2.3.2.3. LIKE

- 使用 LIKE 运算选择类似的值
- 选择条件可以包含字符或数字：
  % 代表零个或多个字符(任意个字符)。
  _ 代表一个字符。

示例 1：

```
SELECTfirst_name
FROM   employees
WHERE first_name LIKE 'S%';
```

示例 2：‘%’和‘-’可以同时使用。

```
SELECT last_name
FROM   employees
WHERE  last_name LIKE '_o%';
```

LAST_NAME
Kochhar
Lorentz
Mourgos

示例 3：可以使用 **ESCAPE** 标识符选择‘%’和 ‘_’ 符号。

- 回避特殊符号的：**使用转义符**。例如：将[%]转为[\%]、[_]转为[_]，然后再加上[ESCAPE ‘\’] 即可

```
SELECT job_id
FROM     jobs
WHERE    job_id LIKE 'IT\_%' escape '\';
```

3.2.4. 对空值（null）的处理

使用 **IS (NOT) NULL** 判断空值。

示例：

```
SELECT last_name, manager_id
FROM     employees
WHERE    manager_id IS NULL;
```

LAST_NAME	MANAGER_ID
King	

3.2.5. 逻辑运算

操作符	含义
AND	逻辑并
OR	逻辑或
NOT	逻辑否

3.2.5.1. AND

- **AND 要求并的关系为真。**

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary >=10000  AND  job_id LIKE '%MAN%';
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
149	Zlotkey	SA_MAN	10500
201	Hartstein	MK_MAN	13000

3.2.5.2. OR

- **OR 要求或关系为真**

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary >= 10000  OR   job_id  LIKE '%MAN%';
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
100	King	AD_PRES	24000
101	Kochhar	AD_VP	17000
102	De Haan	AD_VP	17000
124	Mourgos	ST_MAN	5800
149	Zlotkey	SA_MAN	10500
174	Abel	SA_REP	11000
201	Hartstein	MK_MAN	13000
205	Higgins	AC_MGR	12000

8 rows selected.

3.2.5.3. NOT

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

LAST_NAME	JOB_ID
King	AD_PRES
Kochhar	AD_VP
De Haan	AD_VP
Mourgos	ST_MAN
Zlotkey	SA_MAN
Whalen	AD_ASST
Hartstein	MK_MAN
Fay	MK_REP
Higgins	AC_MGR
Gietz	AC_ACCOUNT

10 rows selected.

3.2.6. 优先级

优先级	
1	算术运算符
2	连接符
3	比较符
4	IS [NOT] NULL, LIKE, [NOT] IN
5	[NOT] BETWEEN
6	NOT
7	AND
8	OR

- 可以使用括号改变优先级顺序

3.3. 排序

3.3.1. 语法

- 使用 ORDER BY 子句排序
  ASC（ascend）：**升序（可以省略）**
  DESC（descend）：**降序**
- ORDER BY 子句在 SELECT 语句的结尾。
- 排序的规则
  - 可以按照 select 语句中的列名排序
  - 可以按照别名列名排序
  - 可以按照 select 语句中的列名的顺序值排序
  - 如果要按照多列进行排序，则规则是先按照第一列排序，如果相同，则按照第二列排序；以此类推。

示例 1：

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

LAST_NAME	JOB_ID	DEPARTMENT_ID	HIRE_DATE
Zlotkey	SA_MAN	80	29-JAN-00
Mourgos	ST_MAN	50	16-NOV-99
Grant	SA_REP		24-MAY-99
Lorentz	IT_PROG	60	07-FEB-99
Vargas	ST_CLERK	50	09-JUL-98
Taylor	SA_REP	80	24-MAR-98
Matos	ST_CLERK	50	15-MAR-98
Fay	MK_REP	20	17-AUG-97
Davies	ST_CLERK	50	29-JAN-97

...
20 rows selected.

示例 2：降序排序

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

3.3.2. 按别名排序

```
SELECT employee_id, last_name, salary*12 annsal
FROM   employees
ORDER BY annsal;
```

EMPLOYEE_ID	LAST_NAME	ANNSAL
144	Vargas	30000
143	Matos	31200
142	Davies	37200
141	Rajs	42000
107	Lorentz	50400
200	Whalen	52800
124	Mourgos	69600
104	Ernst	72000
202	Fay	72000
178	Grant	84000

. . .
20 rows selected.

3.3.3. 多个列排序

- 按照 ORDER BY 列表的顺序排序。
- 可以使用不在 SELECT 列表中的列排序。

示例：

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

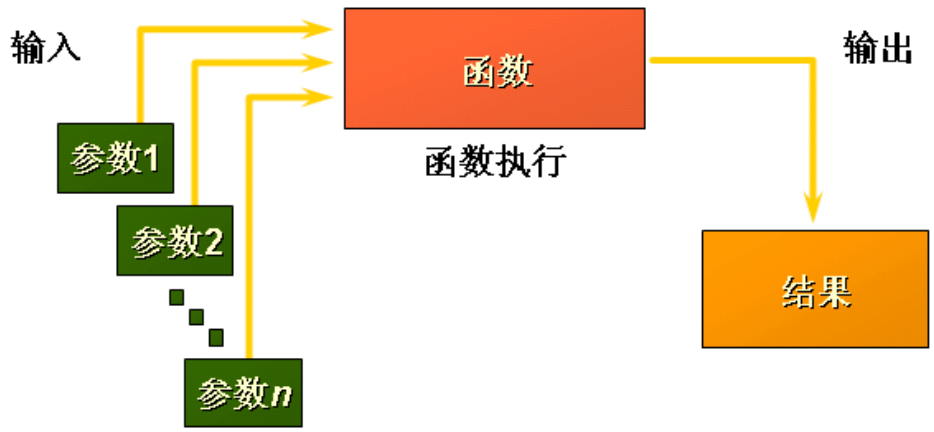
LAST_NAME	DEPARTMENT_ID	SALARY
Whalen	10	4400
Hartstein	20	13000
Fay	20	6000
Mourgos	50	5800
Rajs	50	3500
Davies	50	3100
Matos	50	2600
Vargas	50	2500

. . .
20 rows selected.

4． SQL 函数

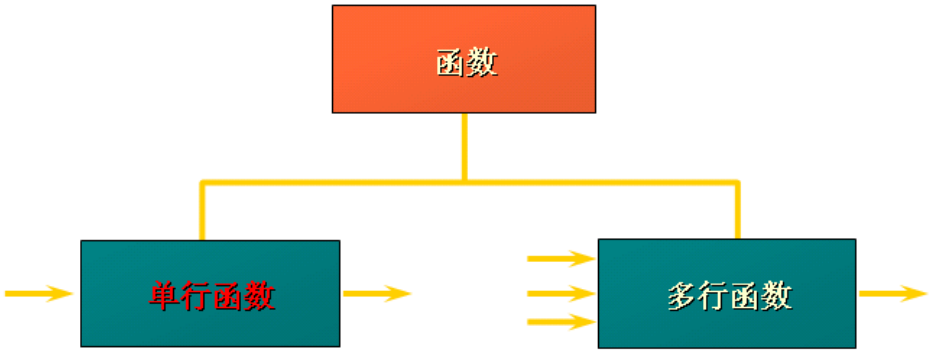
4.1. 函数基础

4.1.1． SQL 函数



注意：:函数可以没有参数，但必须要有返回值

4.1.2． 两种 SQL 函数：单行函数与多行函数



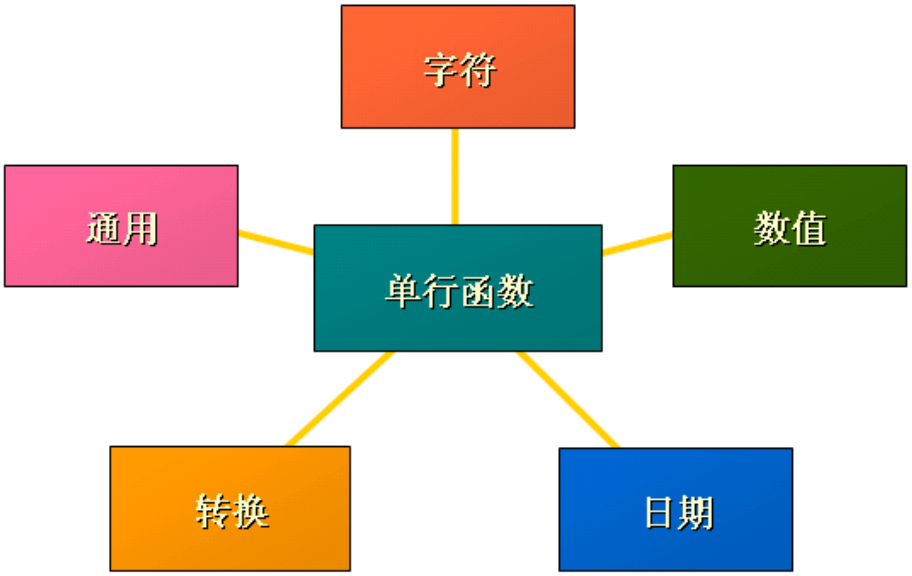
4.2. 单行函数

4.2.1. 单行函数说明

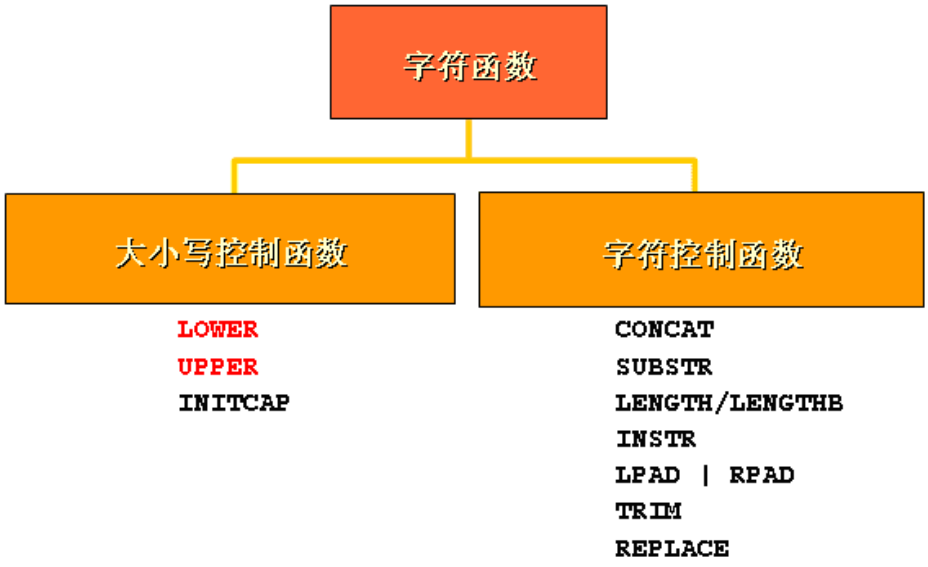
单行函数：

- 操作数据对象
- 接受参数返回一个结果
- **只对一行进行变换**
- **每行返回一个结果**
- 可以转换数据类型
- 可以嵌套
- 参数可以是一列或一个值

```
function_name [(arg1, arg2,...)]
```



4.2.2. 字符函数



4.2.3. 大小写控制函数

- 这类函数用于改变字符的大小写。

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

示例：显示员工 Higgins 的信息

```
SELECT employee_id, last_name, department_id
FROM    employees
WHERE   last_name = 'higgins';

结果：no rows selected
```

```
SELECT employee_id, last_name, department_id
FROM    employees
WHERE   LOWER(last_name) = 'higgins';
```

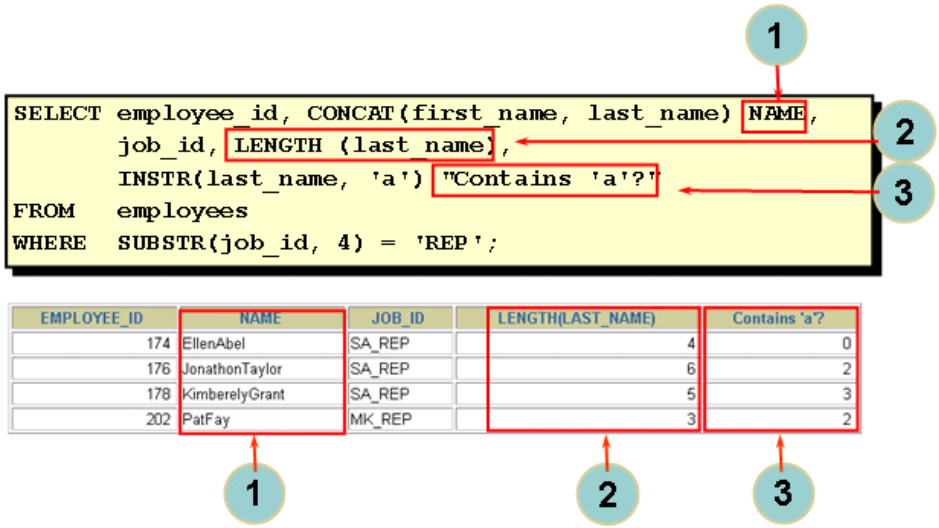
EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
205	Higgins	110

4.2.4. 字符控制函数

这类函数用于控制字符：

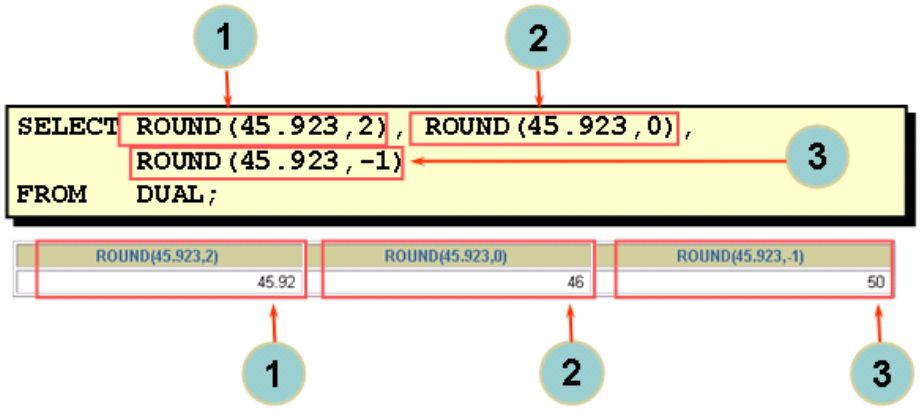
函数	结果
CONCAT('Hello', 'World')	HelloWorld
SUBSTR('HelloWorld',1,5)	Hello
LENGTH('HelloWorld')	10
INSTR('HelloWorld', 'W')	6
LPAD(salary,10,'*')	*****24000
RPAD(salary, 10, '*')	24000*****
TRIM('H' FROM 'HelloWorld')	elloWorld
replace('abcd', 'b', 'm')	amcd

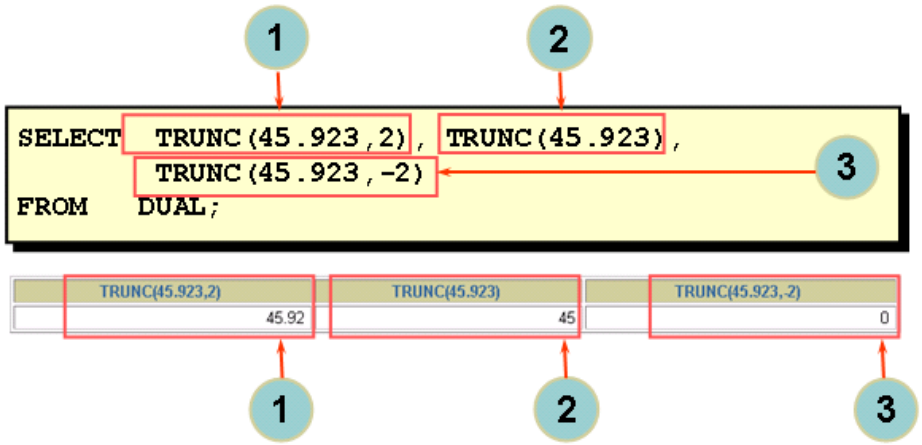
示例：



4.2.5. 数字函数

- ROUND: 四舍五入
  例：ROUND(45.926, 2)，结果 45.93
- TRUNC: 截断
  例：TRUNC(45.926, 2)，结果 45.92
- MOD: 求余
  例：MOD(1600, 300) ，结果 100





```
SELECT last_name, salary, MOD(salary, 5000)
FROM   employees
WHERE  job_id = 'SA_REP';
```

LAST_NAME	SALARY	MOD(SALARY,5000)
Abel	11000	1000
Taylor	8600	3600
Grant	7000	2000

4.2.6. 日期

- Oracle 中的日期型数据实际含有两个值: 日期和时间。
- 默认的日期格式是 DD-MON-RR，例如下面的 hire_date 的值：

LAST_NAME	HIRE_DATE
Gietz	07-JUN-94
Grant	24-MAY-99

函数 SYSDATE 返回:

- 日期
- 时间

4.2.7. 日期的数学运算

- 在日期上加上或减去一个数字结果仍为日期。
- **两个日期相减返回日期之间相差的天数**。
- 可以用数字除 24 来向日期中加上或减去小时。

```
SELECT last_name, (SYSDATE-hire_date)/7 AS WEEKS
FROM   employees
WHERE  department_id = 90;
```

LAST_NAME	WEEKS
King	744.245395
Kochhar	626.102538
De Haan	453.245395

4.2.8. 日期函数

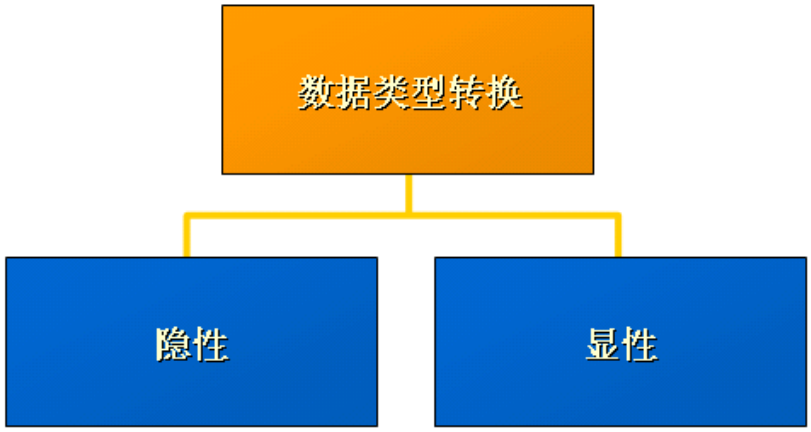
函数	描述
MONTHS_BETWEEN	两个日期相差的月数
ADD_MONTHS	向指定日期中加上若干月数
NEXT_DAY	指定日期的下一个日期
LAST_DAY	本月的最后一天
ROUND	日期四舍五入
TRUNC	日期截断

- MONTHS_BETWEEN ('01-SEP-95','11-JAN-94')
  → 19.6774194
- ADD_MONTHS ('11-JAN-94',6) → '11-JUL-94'
- NEXT_DAY ('01-SEP-95','FRIDAY')
  → '08-SEP-95'
- LAST_DAY('01-FEB-95') → '28-FEB-95'

假设SYSDATE = '25-JUL-95':

- ROUND(SYSDATE,'MONTH') → 01-AUG-95
- ROUND(SYSDATE ,'YEAR') → 01-JAN-96
- TRUNC(SYSDATE ,'MONTH') → 01-JUL-95
- TRUNC(SYSDATE ,'YEAR') → 01-JAN-95

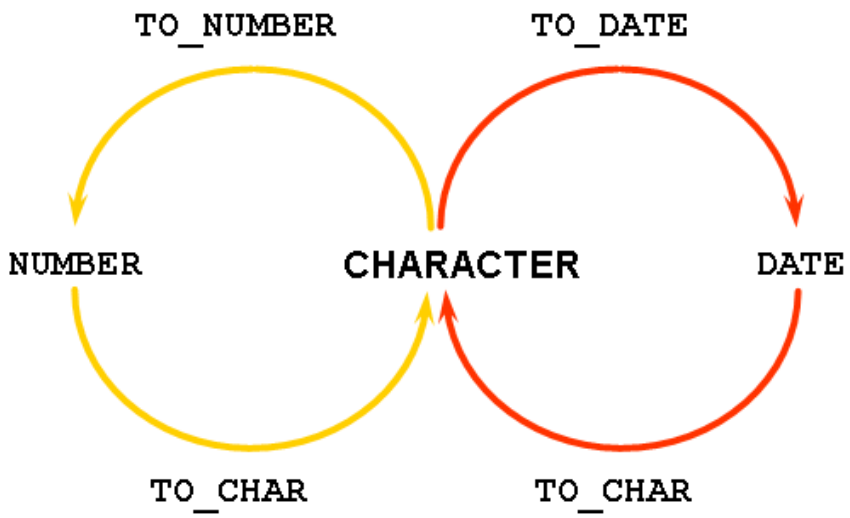
4.2.9. 转换函数



- **隐式数据类型转换**
  Oracle 自动完成下列转换：

源数据类型	目标数据类型
VARCHAR2 or CHAR	NUMBER
VARCHAR2 or CHAR	DATE
NUMBER	VARCHAR2
DATE	VARCHAR2

- 显式数据类型转换



4.2.10. TO_CHAR 函数对日期的转换

语法：TO_CHAR(*date*, '*format_model*')
格式：

- 必须包含在单引号中而且大小写敏感。
- 可以包含任意的有效的日期格式。
- 日期之间用逗号隔开。

- 日期格式的元素

格式	说明	举例
YYYY	Full year in numbers	2011
YEAR	Year spelled out(年的英文全称)	twenty eleven
MM	Two-digit value of month 月份（两位数字）	04
MONTH	Full name of the month（月的全称）	4 月
DY	Three-letter abbreviation of the day of the week(星期几)	星期一
DAY	Full name of the day of the week	星期一
DD	Numeric day of the month	02

- 时间格式

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

- 使用双引号向日期中添加字符

DD "of" MONTH	12 of OCTOBER

```
SELECT last_name,
       TO_CHAR(hire_date, 'DD Month YYYY')
       AS HIREDATE
FROM   employees;
```

LAST_NAME	HIREDATE
King	17 June 1987
Kochhar	21 September 1989
De Haan	13 January 1993
Hunold	3 January 1990
Ernst	21 May 1991
Lorentz	7 February 1999
Mourgos	16 November 1999

...

20 rows selected.

4.2.11. TO_CHAR 函数对数字的转换

语法：TO_CHAR(*number*, '*format_model*')

下面是在 TO_CHAR 函数中经常使用的几种格式:

9	数字
0	零
$	美元符
L	本地货币符号
.	小数点
,	千位符

批注 [t4]: L 不区分大小与，写为小写的'l'也可以。

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

SALARY
$6,000.00

4.2.12. TO_NUMBER 和 TO_DATE 函数

- 使用 TO_NUMBER 函数将字符转换成数字:

```
TO_NUMBER(char[, 'format_model'])
```

- 使用 TO_DATE 函数将字符转换成日期:

```
TO_DATE(char[, 'format_model'])
```

4.2.13. 通用函数

这些函数**适用于任何数据类型，同时也适用于空值**:

- NVL (expr1, expr2)
- NVL2 (expr1, expr2, expr3)
- NULLIF (expr1, expr2)

- COALESCE (expr1, expr2, ..., expr*n*)

NVL 函数将空值转换成一个已知的值：

- 可以使用的数据类型有日期、字符、数字。
- 函数的一般形式:
  - NVL(commission_pct,0)
  - NVL(hire_date,'01-JAN-97')
  - NVL(job_id,'No Job Yet')

```
SELECT last_name, salary, NVL(commission_pct, 0),
   (salary*12) + (salary*12*NVL(commission_pct, 0)) AN_SAL
FROM employees;
```

LAST_NAME	SALARY	NVL(COMMISSION_PCT,0)	AN_SAL
King	24000	0	288000
Kochhar	17000	0	204000
De Haan	17000	0	204000
Hunold	9000	0	108000
Ernst	6000	0	72000
Lorentz	4200	0	50400
Mourgos	5800	0	69600
Rajs	3500	0	42000

...

20 rows selected.

NVL2 (expr1, expr2, expr3) : expr1 不为 NULL，返回 expr2；为 NULL，返回 expr3。

```
SELECT last_name,  salary, commission_pct,
       NVL2(commission_pct,
            'SAL+COMM', 'SAL') income
FROM   employees WHERE department_id IN (50, 80);
```

LAST_NAME	SALARY	COMMISSION_PCT	INCOME
Zlotkey	10500	.2	SAL+COMM
Abel	11000	.3	SAL+COMM
Taylor	8600	.2	SAL+COMM
Mourgos	5800		SAL
Rajs	3500		SAL
Davies	3100		SAL
Matos	2600		SAL
Vargas	2500		SAL

8 rows selected.

NULLIF (expr1, expr2)： 相等返回 NULL，不等返回 expr1

NULLIF (expr1, expr2)：相等返回NULL，不等返回expr1

```
SELECT first_name, LENGTH(first_name) "expr1",
       last_name,  LENGTH(last_name)  "expr2",
       NULLIF(LENGTH(first_name), LENGTH(last_name)) result
FROM   employees;
```

FIRST_NAME	expr1	LAST_NAME	expr2	RESULT
Steven	6	King	4	6
Neena	5	Kochhar	7	5
Lex	3	De Haan	7	3
Alexander	9	Hunold	6	9
Bruce	5	Ernst	5	
Diana	5	Lorentz	7	5
Kevin	5	Mourgos	7	5
Trenna	6	Rajs	4	6
Curtis	6	Davies	6	

...

20 rows selected.

- **使用 COALESCE 函数**
  - COALESCE 与 NVL 相比的优点在于 COALESCE 可以同时处理交替的多个值。

- 如果第一个表达式为空,则返回下一个表达式，对其他的参数进行 COALESCE 。
- **即：找第一个不为空的值。**

```
SELECT   last_name,
         COALESCE(commission_pct, salary, 10) comm
FROM     employees
ORDER BY commission_pct;
```

LAST_NAME	COMM
Grant	.15
Zlotkey	.2
Taylor	.2
Abel	.3
King	24000
Kochhar	17000
De Haan	17000
Hunold	9000

...

20 rows selected.

4.2.14. 条件表达式

- 在 SQL 语句中使用 IF-THEN-ELSE 逻辑
- 使用两种方法:
  - CASE 表达式：SQL99 的语法，类似 Basic，比较繁琐
  - DECODE 函数：Oracle 自己的语法，类似 Java，比较简介

- **CASE 表达式**

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

```
SELECT last_name, job_id, salary,
       CASE job_id WHEN 'IT_PROG'  THEN  1.10*salary
                   WHEN 'ST_CLERK' THEN  1.15*salary
                   WHEN 'SA_REP'   THEN  1.20*salary
       ELSE       salary END     "REVISED_SALARY"
FROM   employees;
```

LAST_NAME	JOB_ID	SALARY	REVISED_SALARY
...			
Lorentz	IT_PROG	4200	4620
Mourgos	ST_MAN	5800	5800
Rajs	ST_CLERK	3500	4025
...			
Gietz	AC_ACCOUNT	8300	8300

20 rows selected.

- **DECODE 函数**

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

```
SELECT last_name, job_id, salary,
       DECODE(job_id, 'IT_PROG',  1.10*salary,
                      'ST_CLERK', 1.15*salary,
                      'SA_REP',   1.20*salary,
              salary)
       REVISED_SALARY
FROM   employees;
```

LAST_NAME	JOB_ID	SALARY	REVISED_SALARY
...			
Lorentz	IT_PROG	4200	4620
Mourgos	ST_MAN	5800	5800
Rajs	ST_CLERK	3500	4025
...			
Gietz	AC_ACCOUNT	8300	8300

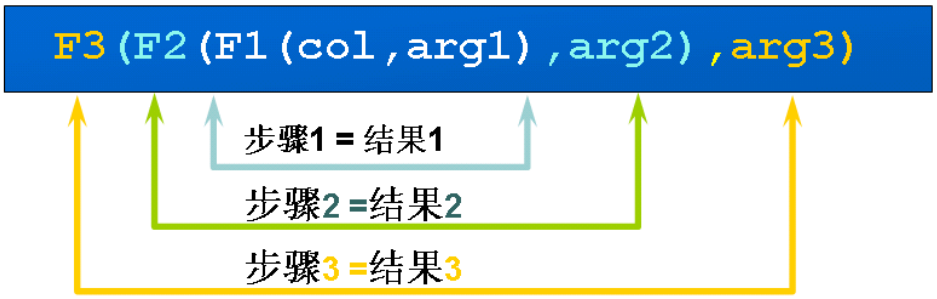
20 rows selected.

使用 decode 函数的一个例子：根据 80 号部门员工的工资，显示税率

```
SELECT last_name, salary,
       DECODE (TRUNC(salary/2000, 0),
                     0, 0.00,
                     1, 0.09,
                     2, 0.20,
                     3, 0.30,
                     4, 0.40,
                     5, 0.42,
                     6, 0.44,
                        0.45) TAX_RATE
FROM   employees
WHERE  department_id = 80;
```

4.2.15. 函数嵌套

- 单行函数可以嵌套。
- 嵌套函数的执行顺序是由内到外。



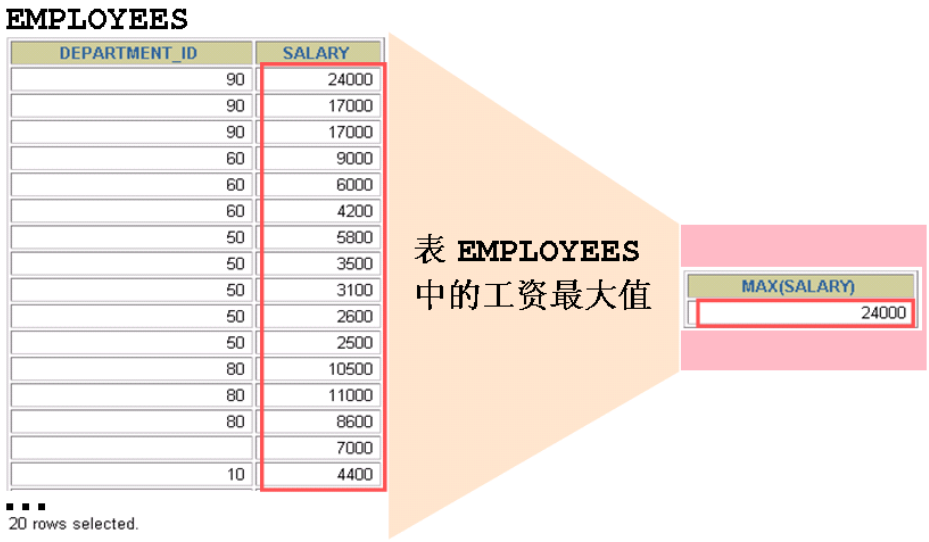

```
SELECT last_name,
       NVL(TO_CHAR(manager_id), 'No Manager')
FROM   employees
WHERE  manager_id IS NULL;
```

LAST_NAME	NVL(TO_CHAR(MANAGER_ID),'NOMANAGER')
King	No Manager

4.3. 分组函数（多行函数）

4.3.1. 什么是分组函数

分组函数作用于一组数据，并对一组数据返回一个值。



4.3.2. 组函数使用

- 类型：
  - AVG 平均值
  - COUNT 数量
  - MAX 最大值
  - MIN 最小值
  - SUM 总和

- 组函数语法

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

4.3.3. AVG（平均值）和 SUM （合计）函数

可以对数值型数据使用 AVG 和 SUM 函数。

```
SELECT AVG(salary), MAX(salary),
       MIN(salary), SUM(salary)
FROM   employees
WHERE  job_id LIKE '%REP%';
```

AVG(SALARY)	MAX(SALARY)	MIN(SALARY)	SUM(SALARY)
8150	11000	6000	32600

4.3.4. MIN（最小值） 和 MAX（最大值）函数

可以对任意数据类型的数据使用 MIN 和 MAX 函数。

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

MIN(HIRE_	MAX(HIRE_
17-JUN-87	29-JAN-00

4.3.5. COUNT（计数）函数

- COUNT(*) 返回表中记录总数。

```
SELECT COUNT(*)
FROM   employees
WHERE  department_id = 50;
```

COUNT(*)
5

- COUNT(*expr*) 返回 *expr* **不为空**的记录总数。

```
SELECT COUNT(commission_pct)
FROM   employees
WHERE  department_id = 80;
```

COUNT(COMMISSION_PCT)
3

- COUNT(**DISTINCT** expr) 返回 ***expr*** **非空且不重复**的记录总数

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

COUNT(DISTINCTDEPARTMENT_ID)
7

4.3.6. 组函数忽略空值。

- **组函数忽略空值。**

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

AVG(COMMISSION_PCT)
.2125

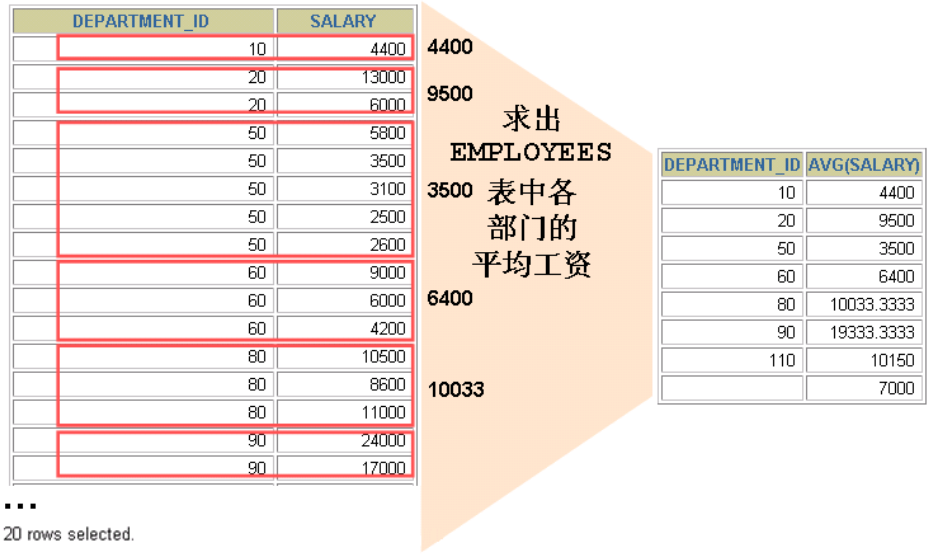
- **在组函数中使用 NVL 函数，NVL 函数使分组函数无法忽略空值。**

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

AVG(NVL(COMMISSION_PCT,0))
.0425

4.3.7. 分组数据

- **EMPLOYEES**



- **GROUP BY 子句语法**

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

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

可以使用 GROUP BY 子句将表中的数据分成若干组

- **在 SELECT 列表中所有未包含在组函数中的列都应该包含在 GROUP BY 子句中。**

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

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

8 rows selected.

- 包含在 GROUP BY 子句中的列不必包含在 SELECT 列表中

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

AVG(SALARY)
4400
9500
3500
6400
10033.3333
19333.3333
10150
7000

- 使用多个列分组

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.

使用多个列
进行分组

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.

- **在 GROUP BY 子句中包含多个列**

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

DEPT_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.

- **非法使用组函数**

所用包含于 SELECT 列表中，而未包含于组函数中的列都必须包含于 GROUP BY 子句中。

```
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
```

GROUP BY 子句中缺少列

- **不能在 WHERE 子句中使用组函数（注意）。**
- **可以在 HAVING 子句中使用组函数。**

```
SELECT    department_id, AVG(salary)
FROM      employees
WHERE     AVG(salary) > 8000
GROUP BY department_id;
```

```
WHERE  AVG(salary) > 8000
       *
ERROR at line 3:
ORA-00934: group function is not allowed here
```

WHERE 子句中不能使用组函数

4.3.8. 过滤分组（Having）

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
...	
20	6000
110	12000
110	8300

20 rows selected.

部门最高工资比 10000 高的部门

DEPARTMENT_ID	MAX(SALARY)
20	13000
80	11000
90	24000
110	12000

使用 HAVING 子句过滤分组:

1. 行已经被分组。
2. 使用了组函数。
3. 满足 HAVING 子句中条件的分组将被显示。

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

示例：

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

DEPARTMENT_ID	MAX(SALARY)
20	13000
80	11000
90	24000
110	12000

4.3.9. 组函数嵌套

- 显示平均工资的最大值：

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

MAX(AVG(SALARY))
19333.3333

5. 多表查询

5.1. 基础（相关说明与笛卡尔积）

5.1.1. 从多个表中获取数据

EMPLOYEES

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
101	Kochhar	90
...		
202	Fay	20
205	Higgins	110
206	Gietz	110

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
10	Administration	1700
20	Marketing	1800
50	Shipping	1500
60	IT	1400
80	Sales	2500
90	Executive	1700
110	Accounting	1700
190	Contracting	1700

EMPLOYEE_ID	DEPARTMENT_ID	DEPARTMENT_NAME
200	10	Administration
201	20	Marketing
202	20	Marketing
...		
102	90	Executive
205	110	Accounting
206	110	Accounting

5.1.2. 笛卡尔集

emp

empno	ename	sal	deptno
1	Tom	1000	10
2	Mary	2000	20
3	Mike	3000	10

dept

deptno	dname
10	开发部
20	行政部

emp*dept

empno	ename	sal	deptno	deptno	dname
1	Tom	1000	10	10	开发部
2	Mary	2000	20	10	开发部
3	Mike	3000	10	10	开发部
1	Tom	1000	10	20	行政部
2	Mary	2000	20	20	行政部
3	Mike	3000	10	20	行政部

连接条件:
emp.deptno=dept.deptno

EMPLOYEES (20行)

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
101	Kochhar	90
...		
202	Fay	20
205	Higgins	110
206	Gietz	110

20 rows selected.

DEPARTMENTS (8行)

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
10	Administration	1700
20	Marketing	1800
50	Shipping	1500
60	IT	1400
80	Sales	2500
90	Executive	1700
110	Accounting	1700
190	Contracting	1700

8 rows selected.

笛卡尔集:
20x8=160行

EMPLOYEE_ID	DEPARTMENT_ID	LOCATION_ID
100	90	1700
101	90	1700
102	90	1700
103	60	1700
104	60	1700
107	60	1700
...		

- **笛卡尔集会在下面条件下产生:**
  - 省略连接条件
  - 连接条件无效
  - 所有表中的所有行互相连接
- 为了避免笛卡尔集， 可以在 WHERE 加入有效的连接条件。
- 在实际运行环境下，应避免使用笛卡尔全集。

5.2. 连接的类型

5.2.1. Oracle 连接:

- Equijoin：等值连接
- Non-equijoin：不等值连接
- Outer join：外连接
- Self join：自连接

5.2.2. SQL: 1999

- Cross joins
- Natural joins
- Using clause
- Full or two sided outer joins

5.3. Oracle 的连接查询

5.3.1. 语法

使用连接在多个表中查询数据。

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

- **在 WHERE 字句中写入连接条件。**
- **在表中有相同列时，在列名之前加上表名前缀**

5.3.2. 等值连接

EMPLOYEES

EMPLOYEE_ID	DEPARTMENT_ID
200	10
201	20
202	20
124	50
141	50
142	50
143	50
144	50
103	60
104	60
107	60
149	80
174	80
176	80

...

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME
10	Administration
20	Marketing
20	Marketing
50	Shipping
50	Shipping
50	Shipping
50	Shipping
50	Shipping
60	IT
60	IT
60	IT
80	Sales
80	Sales
80	Sales

...

外键　　主键

```
SELECT employees.employee_id, employees.last_name,
       employees.department_id, departments.department_id,
       departments.location_id
FROM   employees, departments
WHERE  employees.department_id = departments.department_id;
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	LOCATION_ID
200	Whalen	10	10	1700
201	Hartstein	20	20	1800
202	Fay	20	20	1800
124	Mourgos	50	50	1500
141	Rajs	50	50	1500
142	Davies	50	50	1500
143	Matos	50	50	1500
144	Vargas	50	50	1500

...

19 rows selected.

5.3.3. 多个连接条件与 AND 操作符

EMPLOYEES

LAST_NAME	DEPARTMENT_ID
Whalen	10
Hartstein	20
Fay	20
Mourgos	50
Rajs	50
Davies	50
Matos	50
Vargas	50
Hunold	60
Ernst	60

...

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME
10	Administration
20	Marketing
20	Marketing
50	Shipping
50	Shipping
50	Shipping
50	Shipping
50	Shipping
60	IT
60	IT

...

5.3.4. 区分重复的列名

- **使用表名前缀在多个表中区分相同的列**。
- 在不同表中具有相同列名的列可以用**表的别名**加以区分。

5.3.5. 使用表的别名

- 使用别名可以简化查询。
- 使用表名前缀可以提高执行效率。

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e , departments d
WHERE  e.department_id = d.department_id;
```

- 如果使用了表的别名，则不能再使用表的真名。

5.3.6. 连接多个表

EMPLOYEES

LAST_NAME	DEPARTMENT_ID
King	90
Kochhar	90
De Haan	90
Hunold	60
Ernst	60
Lorentz	60
Mourgos	50
Rajs	50
Davies	50
Matos	50
Vargas	50
Zlotkey	80
Abel	80
Taylor	80

...
20 rows selected.

DEPARTMENTS

DEPARTMENT_ID	LOCATION_ID
10	1700
20	1800
50	1500
60	1400
80	2500
90	1700
110	1700
190	1700

8 rows selected.

LOCATIONS

LOCATION_ID	CITY
1400	Southlake
1500	South San Francisco
1700	Seattle
1800	Toronto
2500	Oxford

- **连接 *n* 个表,至少需要 n-1 个连接条件**。 例如：连接三个表，至少需要两个连接条件。

5.3.7. 非等值连接

EMPLOYEES

LAST_NAME	SALARY
King	24000
Kochhar	17000
De Haan	17000
Hunold	9000
Ernst	6000
Lorentz	4200
Mourgos	5800
Rajs	3500
Davies	3100
Matos	2600
Vargas	2500
Zlotkey	10500
Abel	11000
Taylor	8600

...
20 rows selected.

JOB_GRADES

GRA	LOWEST_SAL	HIGHEST_SAL
A	1000	2999
B	3000	5999
C	6000	9999
D	10000	14999
E	15000	24999
F	25000	40000

EMPLOYEES表中的列工资应在JOB_GRADES表中的最高工资与最低工资之间

```
SELECT e.last_name, e.salary, j.grade_level
FROM   employees e, job_grades j
WHERE  e.salary
       BETWEEN j.lowest_sal AND j.highest_sal;
```

LAST_NAME	SALARY	GRA
Matos	2600	A
Vargas	2500	A
Lorentz	4200	B
Mourgos	5800	B
Rajs	3500	B
Davies	3100	B
Whalen	4400	B
Hunold	9000	C
Ernst	6000	C

...

20 rows selected.

5.3.8. 外连接

DEPARTMENTS

DEPARTMENT_NAME	DEPARTMENT_ID
Administration	10
Marketing	20
Shipping	50
IT	60
Sales	80
Executive	90
Accounting	110
Contracting	190

8 rows selected.

EMPLOYEES

DEPARTMENT_ID	LAST_NAME
90	King
90	Kochhar
90	De Haan
60	Hunold
60	Ernst
60	Lorentz
50	Mourgos
50	Rajs
50	Davies
50	Matos
50	Vargas
80	Zlotkey

...

20 rows selected.

190号部门没有员工

- 外连接语法
- 使用外连接可以查询不满足连接条件的数据。
- 外连接的符号是 (+)。

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

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

示例：

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

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Hartstein	20	Marketing
Fay	20	Marketing
Mourgos	50	Shipping
Rajs	50	Shipping
Davies	50	Shipping
Matos	50	Shipping
...		
Gietz	110	Accounting
		Contracting

20 rows selected.

5.3.9. 自连接

EMPLOYEES (WORKER)

EMPLOYEE_ID	LAST_NAME	MANAGER_ID
100	King	
101	Kochhar	100
102	De Haan	100
103	Hunold	102
104	Ernst	103
107	Lorentz	103
124	Mourgos	100

...

EMPLOYEES (MANAGER)

EMPLOYEE_ID	LAST_NAME
100	King
101	Kochhar
102	De Haan
103	Hunold
104	Ernst
107	Lorentz
124	Mourgos

...

WORKER 表中的MANAGER_ID 和 MANAGER 表中的 MANAGER_ID相等

示例：

```
SELECT worker.last_name || ' works for '
       || manager.last_name
FROM   employees worker, employees manager
WHERE  worker.manager_id = manager.employee_id ;
```

WORKER.LAST_NAME||'WORKSFOR'||MANAGER.LAST_NAME
Kochhar works for King
De Haan works for King
Mourgos works for King
Zlotkey works for King
Hartstein works for King
Whalen works for Kochhar
Higgins works for Kochhar
Hunold works for De Haan
Ernst works for Hunold

...

19 rows selected.

5.4. SQL99 标准的连接查询

5.4.1. 使用连接从多个表中查询数据

使用连接从多个表中查询数据：

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

5.4.2. 叉集

- 使用 **CROSS JOIN** 子句使连接的表产生叉集。
- 叉集和笛卡尔集是相同的。

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

LAST_NAME	DEPARTMENT_NAME
King	Administration
Kochhar	Administration
De Haan	Administration
Hunold	Administration

...

160 rows selected.

5.4.3. 自然连接

- NATURAL JOIN 子句，**会以两个表中具有相同名字的列为条件创建等值连接**。
- 在表中查询满足等值条件的数据。
- 如果只是列名相同而数据类型不同，则会产生错误。

示例：

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

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID	CITY
60	IT	1400	Southlake
50	Shipping	1500	South San Francisco
10	Administration	1700	Seattle
90	Executive	1700	Seattle
110	Accounting	1700	Seattle
190	Contracting	1700	Seattle
20	Marketing	1800	Toronto
80	Sales	2500	Oxford

8 rows selected.

5.4.4. 使用 USING 子句创建连接

- 在 NATURAL JOIN 子句创建等值连接时，可以**使用 USING 子句指定等值连接中需要用到的列**。
- 使用 USING 可以在有多个列满足条件时进行选择。
- **不要给选中的列中加上表名前缀或别名**。
- **NATURAL JOIN 和 USING 子句经常同时使用**。

```
SELECT e.employee_id, e.last_name, d.location_id
FROM   employees e JOIN departments d
USING (department_id) ;
```

EMPLOYEE_ID	LAST_NAME	LOCATION_ID
200	Whalen	1700
201	Hartstein	1800
202	Fay	1800
124	Mourgos	1500
141	Rajs	1500
142	Davies	1500
143	Matos	1500
144	Vargas	1500
103	Hunold	1400

...

19 rows selected.

5.4.5. 使用 ON 子句创建连接

- 自然连接中是以具有相同名字的列为连接条件的。
- **可以使用 ON 子句指定额外的连接条件**。

- 这个连接条件是与其它条件分开的。
- **ON 子句使语句具有更高的易读性**。

- **ON 子句**

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

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	LOCATION_ID
200	Whalen	10	10	1700
201	Hartstein	20	20	1800
202	Fay	20	20	1800
124	Mourgos	50	50	1500
141	Rajs	50	50	1500
142	Davies	50	50	1500
143	Matos	50	50	1500

...

19 rows selected.

- **使用 ON 子句创建多表连接**

```
SELECT employee_id, city, department_name
FROM   employees e
JOIN   departments d
ON     d.department_id = e.department_id
JOIN   locations l
ON     d.location_id = l.location_id;
```

EMPLOYEE_ID	CITY	DEPARTMENT_NAME
103	Southlake	IT
104	Southlake	IT
107	Southlake	IT
124	South San Francisco	Shipping
141	South San Francisco	Shipping
142	South San Francisco	Shipping
143	South San Francisco	Shipping
144	South San Francisco	Shipping

...

19 rows selected.

5.4.6. 内连接和外连接(2)

- 在 SQL: 1999 中，内连接只返回满足连接条件的数据
- 两个表在连接过程中除了返回满足连接条件的行以外还返回左（或右）表中不满足条件

的行 ， 这种连接称为左（或右） 外联接。

- 两个表在连接过程中除了返回满足连接条件的行以外还返回两个表中不满足条件的行 ，这种连接称为**满外联接**。

5.4.7. 左外联接

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

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Fay	20	Marketing
Hartstein	20	Marketing
...		
De Haan	90	Executive
Kochhar	90	Executive
King	90	Executive
Gietz	110	Accounting
Higgins	110	Accounting
Grant		

20 rows selected.

5.4.8. 右外联接

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

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
King	90	Executive
Kochhar	90	Executive
...		
Whalen	10	Administration
Hartstein	20	Marketing
Fay	20	Marketing
Higgins	110	Accounting
Gietz	110	Accounting
		Contracting

20 rows selected.

5.4.9. 满外联接

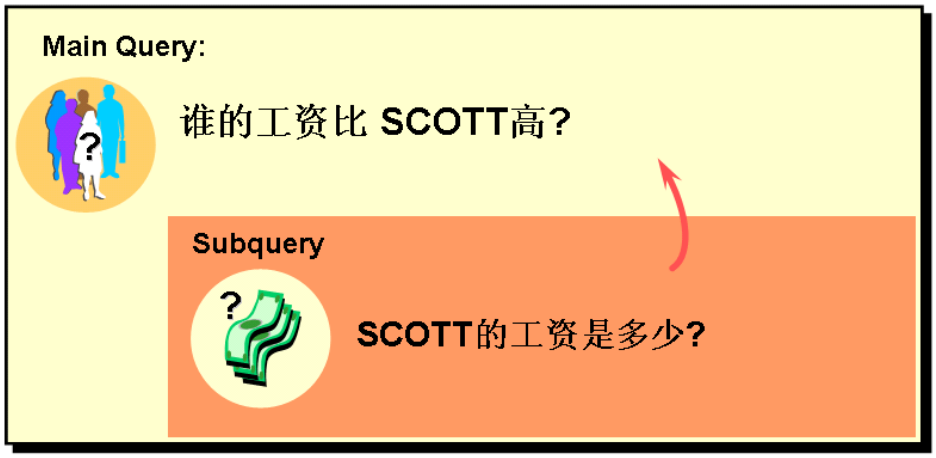
```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e
FULL OUTER JOIN departments d
ON    (e.department_id = d.department_id) ;
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Fay	20	Marketing
...		
De Haan	90	Executive
Kochhar	90	Executive
King	90	Executive
Gietz	110	Accounting
Higgins	110	Accounting
Grant		
		Contracting

21 rows selected.

6. 子查询

谁的工资比 SCOTT 高?



6.1. 子查询语法

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

- 子查询 (内查询) 在主查询之前一次执行完成。
- 子查询的结果被主查询使用 (外查询)。

示例：

```
SELECT last_name
FROM   employees 11000
WHERE  salary >
                (SELECT salary
                 FROM   employees
                 WHERE  last_name = 'Abel');
```

LAST_NAME
King
Kochhar
De Haan
Hartstein
Higgins

6.2. 注意事项

- 子查询要包含在括号内。
- 将子查询放在比较条件的右侧。
- 单行操作符对应单行子查询，多行操作符对应多行子查询。

6.3. 子查询类型

- 单行子查询

- 多行子查询

6.4. 单行子查询

- 只返回一行。
- 使用单行比较操作符。

操作符	含义
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to

- 执行单行子查询

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  job_id =
                  (SELECT job_id
                   FROM   employees
                   WHERE  employee_id = 141)
AND    salary >
                  (SELECT salary
                   FROM   employees
                   WHERE  employee_id = 143);
```

ST_CLERK

2600

LAST_NAME	JOB_ID	SALARY
Rajs	ST_CLERK	3500
Davies	ST_CLERK	3100

- 在子查询中使用组函数

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  salary =
                  (SELECT MIN(salary)
                   FROM   employees);
```

2500

LAST_NAME	JOB_ID	SALARY
Vargas	ST_CLERK	2500

- 子查询中的 HAVING 子句
  - 首先执行子查询。
  - 向主查询中的 HAVING 子句返回结果。

```
SELECT   department_id, MIN(salary)
FROM     employees
GROUP BY department_id
HAVING   MIN(salary) >
                        (SELECT MIN(salary)
                         FROM   employees
                         WHERE  department_id = 50);
```

2500

6.5. 非法使用子查询

- 多行子查询使用单行比较符

```
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
```

6.6. 子查询中的空值问题

```
SELECT last_name, job_id
FROM   employees
WHERE  job_id =
                (SELECT job_id
                 FROM   employees
                 WHERE  last_name = 'Haas');
```

```
no rows selected
```

- 子查询不返回任何行

```
SELECT emp.last_name
FROM   employees emp
WHERE  emp.employee_id NOT IN
                              (SELECT mgr.manager_id
                               FROM   employees mgr);

no rows selected
```

6.7. 多行子查询

- 返回多行。
- 使用多行比较操作符。

操作符	含义
IN	等于列表中的任何一个
ANY	和子查询返回的任意一个值比较
ALL	和子查询返回的所有值比较

- 在多行子查询中使用 ANY 操作符

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

9000, 6000, 4200

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
124	Mourgos	ST_MAN	5800
141	Rajs	ST_CLERK	3500
142	Davies	ST_CLERK	3100
143	Matos	ST_CLERK	2600
144	Vargas	ST_CLERK	2500

...

10 rows selected.

- 在多行子查询中使用 ALL 操作符

```
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';
```

9000, 6000, 4200

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
141	Rajs	ST_CLERK	3500
142	Davies	ST_CLERK	3100
143	Matos	ST_CLERK	2600
144	Vargas	ST_CLERK	2500

6.8. 练习题

- 找到员工表中工资最高的前三名，如下格式：

```
ROWNUM      EMPNO ENAME      SAL
------ ---------- ---------- ----------
     1       7839 KING             5000
     2       7788 SCOTT            3000
     3       7902 FORD             3000
```

- 找到员工表中薪水大于本部门平均薪水的员工。

```
EMPNO ENAME      SAL     AVGSAL
----- ---------- ---------- ----------
 7499 ALLEN            1600 1566.66667
 7566 JONES            2975       2175
 7698 BLAKE            2850 1566.66667
 7788 SCOTT            3000       2175
 7839 KING             5000 2916.66667
 7902 FORD             3000       2175
```

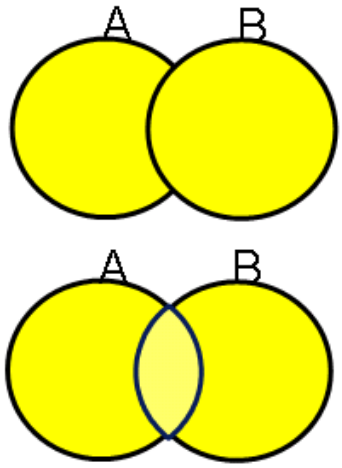
- 统计每年入职的员工个数。

```
 Total       1980       1981       1982       1987
------- ---------- ---------- ---------- ----------
    14          1         10          1          2
```

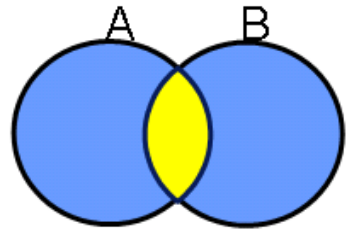
7. 集合运算

7.1. 集合运算的类型与集合运算符

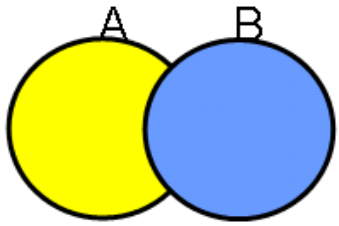
7.1.1. UNION/UNION ALL 并集



7.1.2. INTERSECT 交集

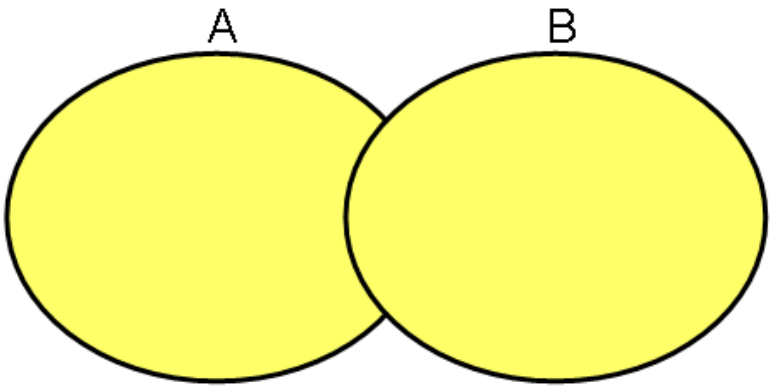


7.1.3. MINUS 差集



7.2. 并集（Union 与 Union All）

7.2.1. 并集（Union）



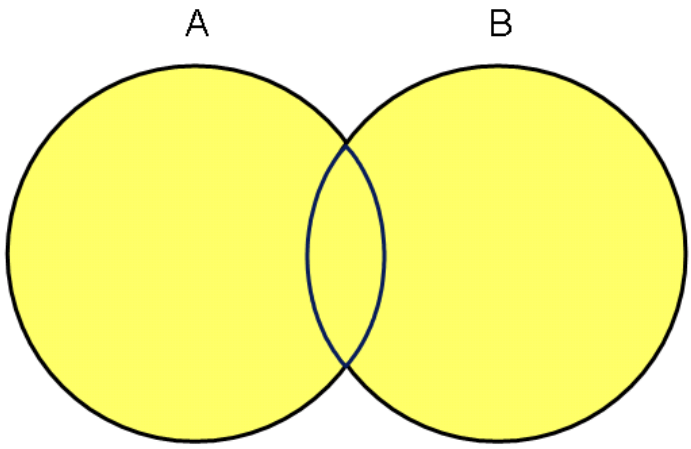
UNION 运算符返回两个集合去掉重复元素后的所有记录。

- 示例：显示员工当前和之前的工作情况，每次记录显示一次。

```
SELECT employee_id, job_id
FROM   employees
UNION
SELECT employee_id, job_id
FROM   job_history;
```

EMPLOYEE_ID	JOB_ID
100	AD_PRES
101	AC_ACCOUNT
...	
200	AC_ACCOUNT
200	AD_ASST
...	
205	AC_MGR
206	AC_ACCOUNT

7.2.2. 并集（Union All）



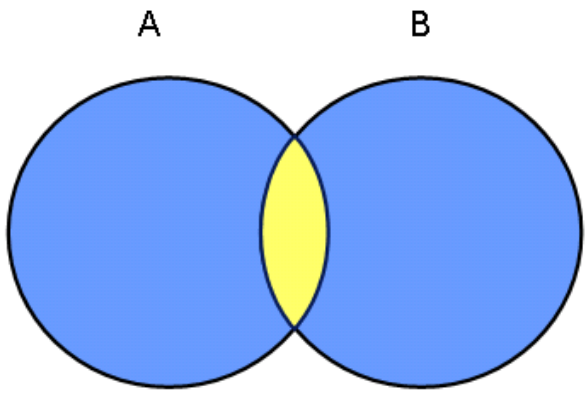
UNION ALL 返回两个集合的所有记录，包括重复的。

- 示例：使用 Union All：显示所有员工当前和之前的部门信息。

```
SELECT employee_id, job_id, department_id
FROM   employees
UNION ALL
SELECT employee_id, job_id, department_id
FROM   job_history
ORDER BY  employee_id;
```

EMPLOYEE_ID	JOB_ID	DEPARTMENT_ID
100	AD_PRES	90
101	AD_VP	90
...		
200	AD_ASST	10
200	AD_ASST	90
200	AC_ACCOUNT	90
...		
205	AC_MGR	110
206	AC_ACCOUNT	110

7.3. 交集（Intersect）



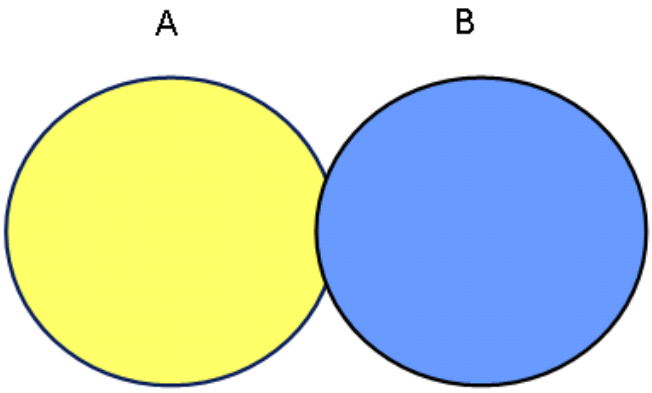
INTERSECT 运算符返回同时属于两个集合的记录。

- 示例：使用 INTERSECT 运算符，显示薪水同时位于级别 1（700~1300）和级别 2（1201~1400）的员工信息。

```
select ename,sal from emp
where sal between 700 and 1300
INTERSECT
select ename,sal from emp
where sal between 1201 and 1400;
```

ENAME	SAL
MARTIN	1250
MILLER	1300
WARD	1250

7.4. 差集（Minus）



MINUS 返回属于第一个集合，但不属于第二个集合的记录。

- 使用 Minus 运算符：显示薪水位于级别 1（700~1300），但不属于级别 2（1201~1400）的员工信息。

```
select ename,sal from emp
where sal between 700 and 1300
minus
select ename,sal from emp
where sal between 1201 and 1400;
```

ENAME	SAL
ADAMS	1100
JAMES	950
SMITH	800

7.5. 集合运算的注意事项

- select 语句中参数类型和个数要一致。
- 可以使用括号改变集合执行的顺序
- 如果有 order by 子句，必须放到每一句查询语句后
- 集合运算采用第一个语句的表头作为表头

8. 多表查询

8.1. 基础（相关说明与笛卡尔积）

8.1.1. 从多个表中获取数据

EMPLOYEES

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
101	Kochhar	90
...		
202	Fay	20
205	Higgins	110
206	Gietz	110

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
10	Administration	1700
20	Marketing	1800
50	Shipping	1500
60	IT	1400
80	Sales	2500
90	Executive	1700
110	Accounting	1700
190	Contracting	1700

EMPLOYEE_ID	DEPARTMENT_ID	DEPARTMENT_NAME
200	10	Administration
201	20	Marketing
202	20	Marketing
...		
102	90	Executive
205	110	Accounting
206	110	Accounting

8.1.2. 笛卡尔集

emp

empno	ename	sal	deptno
1	Tom	1000	10
2	Mary	2000	20
3	Mike	3000	10

dept

deptno	dname
10	开发部
20	行政部

emp*dept

empno	ename	sal	deptno	deptno	dname
1	Tom	1000	10	10	开发部
2	Mary	2000	20	10	开发部
3	Mike	3000	10	10	开发部
1	Tom	1000	10	20	行政部
2	Mary	2000	20	20	行政部
3	Mike	3000	10	20	行政部

连接条件:
emp.deptno=dept.deptno

EMPLOYEES (20行)

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
101	Kochhar	90
...		
202	Fay	20
205	Higgins	110
206	Gietz	110

20 rows selected.

DEPARTMENTS (8行)

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
10	Administration	1700
20	Marketing	1800
50	Shipping	1500
60	IT	1400
80	Sales	2500
90	Executive	1700
110	Accounting	1700
190	Contracting	1700

8 rows selected.

笛卡尔集:
20x8=160行

EMPLOYEE_ID	DEPARTMENT_ID	LOCATION_ID
100	90	1700
101	90	1700
102	90	1700
103	60	1700
104	60	1700
107	60	1700
...		

- **笛卡尔集会在下面条件下产生:**
  - 省略连接条件
  - 连接条件无效
  - 所有表中的所有行互相连接
- 为了避免笛卡尔集， 可以在 WHERE 加入有效的连接条件。
- 在实际运行环境下，应避免使用笛卡尔全集。

8.2. 连接的类型

8.2.1. Oracle 连接:

- Equijoin：等值连接
- Non-equijoin：不等值连接
- Outer join：外连接
- Self join：自连接

8.2.2. SQL: 1999

- Cross joins
- Natural joins
- Using clause
- Full or two sided outer joins

8.3. Oracle 的连接查询

8.3.1. 语法

使用连接在多个表中查询数据。

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

- 在 WHERE 字句中写入连接条件。
- 在表中有相同列时，在列名之前加上表名前缀

8.3.2. 等值连接

```
SELECT employees.employee_id, employees.last_name,
       employees.department_id, departments.department_id,
       departments.location_id
FROM   employees, departments
WHERE  employees.department_id = departments.department_id;
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	LOCATION_ID
200	Whalen	10	10	1700
201	Hartstein	20	20	1800
202	Fay	20	20	1800
124	Mourgos	50	50	1500
141	Rajs	50	50	1500
142	Davies	50	50	1500
143	Matos	50	50	1500
144	Vargas	50	50	1500

...

19 rows selected.

8.3.3. 多个连接条件与 AND 操作符

EMPLOYEES

LAST_NAME	DEPARTMENT_ID
Whalen	10
Hartstein	20
Fay	20
Mourgos	50
Rajs	50
Davies	50
Matos	50
Vargas	50
Hunold	60
Ernst	60

...

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME
10	Administration
20	Marketing
20	Marketing
50	Shipping
50	Shipping
50	Shipping
50	Shipping
50	Shipping
60	IT
60	IT

...

批注 [t5]: 没有写 SQL 示例

8.3.4. 区分重复的列名

- **使用表名前缀在多个表中区分相同的列**。
- 在不同表中具有相同列名的列可以用**表的别名**加以区分。

8.3.5. 使用表的别名

- 使用别名可以简化查询。
- 使用表名前缀可以提高执行效率。

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e , departments d
WHERE  e.department_id = d.department_id;
```

- 如果使用了表的别名，则不能再使用表的真名。

8.3.6. 连接多个表

EMPLOYEES

LAST_NAME	DEPARTMENT_ID
King	90
Kochhar	90
De Haan	90
Hunold	60
Ernst	60
Lorentz	60
Mourgos	50
Rajs	50
Davies	50
Matos	50
Vargas	50
Zlotkey	80
Abel	80
Taylor	80

...

20 rows selected.

DEPARTMENTS

DEPARTMENT_ID	LOCATION_ID
10	1700
20	1800
50	1500
60	1400
80	2500
90	1700
110	1700
190	1700

8 rows selected.

LOCATIONS

LOCATION_ID	CITY
1400	Southlake
1500	South San Francisco
1700	Seattle
1800	Toronto
2500	Oxford

- **连接 *n* 个表,至少需要 n-1 个连接条件**。 例如：连接三个表，至少需要两个连接条件。

8.3.7. 非等值连接

EMPLOYEES

LAST_NAME	SALARY
King	24000
Kochhar	17000
De Haan	17000
Hunold	9000
Ernst	6000
Lorentz	4200
Mourgos	5800
Rajs	3500
Davies	3100
Matos	2600
Vargas	2500
Zlotkey	10500
Abel	11000
Taylor	8600

...

20 rows selected.

JOB_GRADES

GRA	LOWEST_SAL	HIGHEST_SAL
A	1000	2999
B	3000	5999
C	6000	9999
D	10000	14999
E	15000	24999
F	25000	40000

EMPLOYEES表中的列工资应在JOB_GRADES表中的最高工资与最低工资之间

```
SELECT e.last_name, e.salary, j.grade_level
FROM   employees e, job_grades j
WHERE  e.salary
       BETWEEN j.lowest_sal AND j.highest_sal;
```

LAST_NAME	SALARY	GRA
Matos	2600	A
Vargas	2500	A
Lorentz	4200	B
Mourgos	5800	B
Rajs	3500	B
Davies	3100	B
Whalen	4400	B
Hunold	9000	C
Ernst	6000	C

...

20 rows selected.

8.3.8. 外连接

DEPARTMENTS

DEPARTMENT_NAME	DEPARTMENT_ID
Administration	10
Marketing	20
Shipping	50
IT	60
Sales	80
Executive	90
Accounting	110
Contracting	190

8 rows selected.

EMPLOYEES

DEPARTMENT_ID	LAST_NAME
90	King
90	Kochhar
90	De Haan
60	Hunold
60	Ernst
60	Lorentz
50	Mourgos
50	Rajs
50	Davies
50	Matos
50	Vargas
80	Zlotkey

...

20 rows selected.

190号部门没有员工

- 外连接语法
- 使用外连接可以查询不满足连接条件的数据。
- 外连接的符号是 (+)。

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

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

示例：

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

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Hartstein	20	Marketing
Fay	20	Marketing
Mourgos	50	Shipping
Rajs	50	Shipping
Davies	50	Shipping
Matos	50	Shipping
...		
Gietz	110	Accounting
		Contracting

20 rows selected.

8.3.9. 自连接

EMPLOYEES (WORKER)

EMPLOYEE_ID	LAST_NAME	MANAGER_ID
100	King	
101	Kochhar	100
102	De Haan	100
103	Hunold	102
104	Ernst	103
107	Lorentz	103
124	Mourgos	100
...		

EMPLOYEES (MANAGER)

EMPLOYEE_ID	LAST_NAME
100	King
101	Kochhar
102	De Haan
103	Hunold
104	Ernst
107	Lorentz
124	Mourgos
...	

WORKER 表中的MANAGER_ID 和 MANAGER 表中的 MANAGER_ID相等

示例：

```
SELECT worker.last_name || ' works for '
       || manager.last_name
FROM   employees worker, employees manager
WHERE  worker.manager_id = manager.employee_id ;
```

WORKER.LAST_NAME||'WORKSFOR'||MANAGER.LAST_NAME
Kochhar works for King
De Haan works for King
Mourgos works for King
Zlotkey works for King
Hartstein works for King
Whalen works for Kochhar
Higgins works for Kochhar
Hunold works for De Haan
Ernst works for Hunold

...

19 rows selected.

8.4. SQL99 标准的连接查询

8.4.1. 使用连接从多个表中查询数据

使用连接从多个表中查询数据：

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

8.4.2. 叉集

- 使用 **CROSS JOIN** 子句使连接的表产生叉集。
- 叉集和笛卡尔集是相同的。

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

LAST_NAME	DEPARTMENT_NAME
King	Administration
Kochhar	Administration
De Haan	Administration
Hunold	Administration

...

160 rows selected.

8.4.3. 自然连接

- NATURAL JOIN 子句，**会以两个表中具有相同名字的列为条件创建等值连接**。
- 在表中查询满足等值条件的数据。
- 如果只是列名相同而数据类型不同，则会产生错误。

示例：

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

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID	CITY
60	IT	1400	Southlake
50	Shipping	1500	South San Francisco
10	Administration	1700	Seattle
90	Executive	1700	Seattle
110	Accounting	1700	Seattle
190	Contracting	1700	Seattle
20	Marketing	1800	Toronto
80	Sales	2500	Oxford

8 rows selected.

8.4.4. 使用 USING 子句创建连接

- 在 NATURAL JOIN 子句创建等值连接时，可以**使用 USING 子句指定等值连接中需要用到的列**。
- 使用 USING 可以在有多个列满足条件时进行选择。
- **不要给选中的列中加上表名前缀或别名**。
- **NATURAL JOIN 和 USING 子句经常同时使用**。

```
SELECT e.employee_id, e.last_name, d.location_id
FROM   employees e JOIN departments d
USING (department_id) ;
```

EMPLOYEE_ID	LAST_NAME	LOCATION_ID
200	Whalen	1700
201	Hartstein	1800
202	Fay	1800
124	Mourgos	1500
141	Rajs	1500
142	Davies	1500
143	Matos	1500
144	Vargas	1500
103	Hunold	1400

...

19 rows selected.

8.4.5. 使用 ON 子句创建连接

- 自然连接中是以具有相同名字的列为连接条件的。
- **可以使用 ON 子句指定额外的连接条件。**
- 这个连接条件是与其它条件分开的。
- **ON 子句使语句具有更高的易读性。**

- **ON 子句**

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

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	LOCATION_ID
200	Whalen	10	10	1700
201	Hartstein	20	20	1800
202	Fay	20	20	1800
124	Mourgos	50	50	1500
141	Rajs	50	50	1500
142	Davies	50	50	1500
143	Matos	50	50	1500

...

19 rows selected.

- **使用 ON 子句创建多表连接**

```
SELECT employee_id, city, department_name
FROM   employees e
JOIN   departments d
ON     d.department_id = e.department_id
JOIN   locations l
ON     d.location id = l.location id;
```

EMPLOYEE_ID	CITY	DEPARTMENT_NAME
103	Southlake	IT
104	Southlake	IT
107	Southlake	IT
124	South San Francisco	Shipping
141	South San Francisco	Shipping
142	South San Francisco	Shipping
143	South San Francisco	Shipping
144	South San Francisco	Shipping

...

19 rows selected.

8.4.6. 内连接和外连接(2)

- 在 SQL: 1999 中，内连接只返回满足连接条件的数据
- 两个表在连接过程中除了返回满足连接条件的行以外还返回左（或右）表中不满足条件的行 ，这种连接称为左（或右） 外联接。
- 两个表在连接过程中除了返回满足连接条件的行以外还返回两个表中不满足条件的行 ，这种连接称为**满外联接**。

8.4.7. 左外联接

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

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Fay	20	Marketing
Hartstein	20	Marketing
...		
De Haan	90	Executive
Kochhar	90	Executive
King	90	Executive
Gietz	110	Accounting
Higgins	110	Accounting
Grant		

20 rows selected.

8.4.8. 右外联接

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

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
King	90	Executive
Kochhar	90	Executive
...		
Whalen	10	Administration
Hartstein	20	Marketing
Fay	20	Marketing
Higgins	110	Accounting
Gietz	110	Accounting
		Contracting

20 rows selected.

8.4.9. 满外联接

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

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Fay	20	Marketing
...		
De Haan	90	Executive
Kochhar	90	Executive
King	90	Executive
Gietz	110	Accounting
Higgins	110	Accounting
Grant		
		Contracting

21 rows selected.

9． 处理数据（DML）

9.1. 数据控制语言（DML）

DML(Data Manipulation Language – 数据操作语言) 可以在下列条件下执行:
向表中插入数据
修改现存数据
删除现存数据
事务是由完成若干项工作的 DML 语句组成的

9.2. 插入数据

9.2.1. INSERT 语句语法

- 使用 INSERT 语句向表中插入数据。
- 使用这种语法一次只能向表中插入**一条**数据。

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

9.2.2. 插入数据

- 为每一列添加一个新值。
- 按列的默认顺序列出各个列的值。
- 在 INSERT 子句中随意列出列名和他们的值。
- **字符和日期型数据应包含在单引号中**。

例：

```
INSERT INTO departments(department_id, department_name,
                        manager_id, location_id)
VALUES      (70, 'Public Relations', 100, 1700);
1 row created.
```

9.2.3. 向表中插入空值

- 隐式方式: 在列名表中省略该列的值。

```
INSERT INTO   departments (department_id,
                           department_name )
VALUES        (30, 'Purchasing');
1 row created.
```

- 显式方式: 在 VALUES 子句中指定空值。

```
INSERT INTO   departments
VALUES        (100, 'Finance', NULL, NULL);
1 row created.
```

9.2.4. 插入指定的值

- **例 1：**

```
INSERT INTO employees (employee_id,
                  first_name, last_name,
                  email, phone_number,
                  hire_date, job_id, salary,
                  commission_pct, manager_id,
                  department_id)
VALUES           (113,
                  'Louis', 'Popp',
                  'LPOPP', '515.124.4567',
                  SYSDATE, 'AC_ACCOUNT', 6900,
                  NULL, 205, 100);
1 row created.
```

说明：SYSDATE 记录当前系统的日期和时间。

- **例 2：加入新员工**

```
INSERT INTO employees
VALUES      (114,
             'Den', 'Raphealy',
             'DRAPHEAL', '515.127.4561',
             TO_DATE('FEB 3, 1999', 'MON DD, YYYY'),
             'AC_ACCOUNT', 11000, NULL, 100, 30);
1 row created.
```

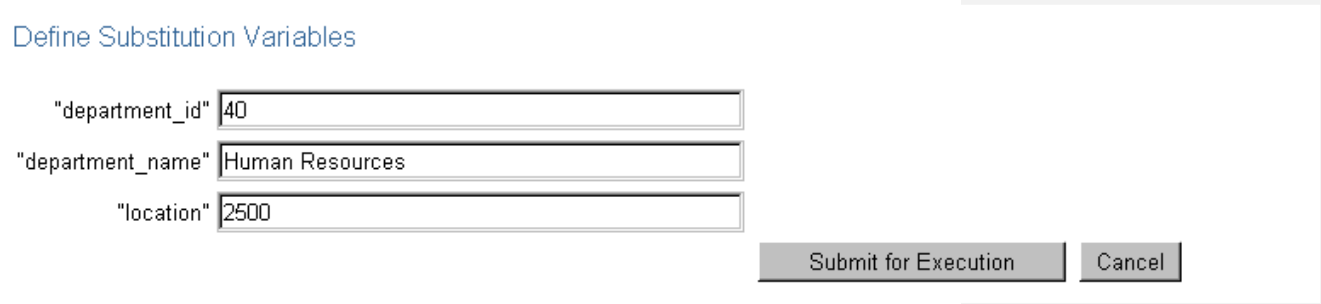
效果如下：

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_P
114	Den	Raphealy	DRAPHEAL	515.127.4561	03-FEB-99	AC_ACCOUNT	11000	

9.2.5. 创建脚本 （使用 & 变量指定列值）

- 在 SQL 语句中使用 & 变量指定列值。
- & 变量放在 VALUES 子句中。

```
INSERT INTO departments
            (department_id, department_name, location_id)
VALUES      (&department_id, '&department_name', &location);
```




```
1 row created.
```

9.2.6. 从其它表中拷贝数据（Insert Into ... Select ...）

9.3. 更新数据

9.3.1. 语法

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

9.3.2. 一般用法示例

- 使用 WHERE 子句指定需要更新的数据。

```
UPDATE employees
SET    department_id = 70
WHERE  employee_id = 113;
1 row updated.
```

- 如果省略WHERE子句，则表中的所有数据都将被更新。

```
UPDATE   copy_emp
SET      department_id = 110;
22 rows updated.
```

9.3.3. 在 UPDATE 语句中使用子查询

- 例：更新 114 号员工的工作和工资使其与 205 号员工相同。

```
UPDATE   employees
SET      job_id  = (SELECT  job_id
                    FROM    employees
                    WHERE   employee_id = 205),
         salary  = (SELECT  salary
                    FROM    employees
                    WHERE   employee_id = 205)
WHERE    employee_id    =  114;
1 row updated.
```

- 在 UPDATE 中使用子查询，使更新基于另一个表中的数据。

```
UPDATE  copy_emp
SET     department_id  =  (SELECT department_id
                          FROM employees
                          WHERE employee_id = 100)
WHERE   job_id         =  (SELECT job_id
                          FROM employees
                          WHERE employee_id = 200);
1 row updated.
```

9.3.4. 更新中的数据完整性错误

```
UPDATE employees
SET    department_id = 55
WHERE  department_id = 110;
```

```
UPDATE employees
       *
ERROR at line 1:
ORA-02291: integrity constraint (HR.EMP_DEPT_FK)
violated - parent key not found
```

不存在 55 号部门

9.4. 删除数据

9.4.1. 语法

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

9.4.2. 一般用法示例

- 使用WHERE 子句指定删除的记录。

```
DELETE FROM departments
WHERE  department_name = 'Finance';
1 row deleted.
```

- 如果省略WHERE子句，则表中的全部数据将被删除。

```
DELETE FROM  copy_emp;
22 rows deleted.
```

9.4.3. 在 DELETE 中使用子查询

- 在 DELETE 中使用子查询，使删除基于另一个表中的数据。

```
DELETE FROM employees
WHERE  department_id =
                       (SELECT department_id
                        FROM   departments
                        WHERE  department_name LIKE '%Public%');
1 row deleted.
```

9.4.4. 删除中的数据完整性错误

```
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
```

You cannot delete a row that contains a primary key that is used as a foreign key in another table.

9.4.5. Delete 和 Truncate

- 都是删除表中的数据
- Delete 操作可以 rollback，可以闪回
- Delete 操作可能产生碎片，并且不释放空间
- Truncate 是清空表

9.5. 数据库事务

9.5.1. 事务基础

数据库事务由以下的部分组成:

- **一个或多个 DML 语句**
- 一个 DDL(Data Definition Language – 数据定义语言) 语句
- 一个 DCL(Data Control Language – 数据控制语言) 语句

数据库事务:

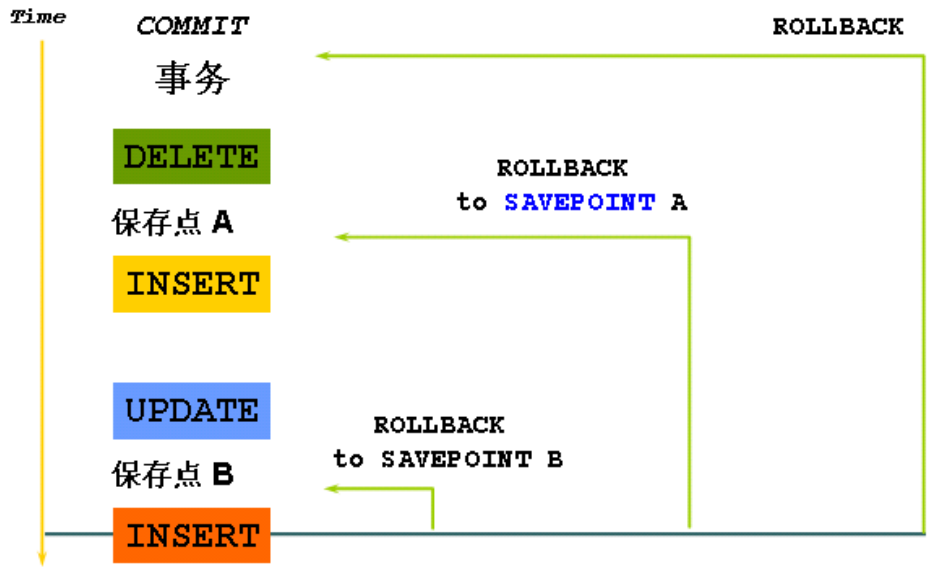
- 以第一个 DML 语句的执行作为开始
- 以下面的其中之一作为结束:
  - **显示结束: commit rollback**
  - **隐式结束(自动提交): DDL 语言,DCL 语言, exit(事务正常退出)**

- **隐式回滚(系统异常终了)： 关闭窗口，死机，掉电**

使用 COMMIT 和 ROLLBACK 语句,我们可以：

- 确保数据完整性。
- 数据改变被提交之前预览。
- 将逻辑上相关的操作分组。

9.5.2. 控制事务



- **回滚到保留点**
  - 使用 **SAVEPOINT** 语句在当前事务中创建保存点。
  - 使用 **ROLLBACK TO SAVEPOINT** 语句回滚到创建的保存点。

```
UPDATE...
SAVEPOINT update_done;
Savepoint created.
INSERT...
ROLLBACK TO update_done;
Rollback complete.
```

9.5.3. 事务进程

- 自动提交在以下情况中执行:
  - DDL 语句。
  - DCL 语句。
  - 不使用 COMMIT 或 ROLLBACK 语句提交或回滚，正常结束会话 exit。
- 会话异常结束或系统异常会导致自动回滚。

- **提交或回滚前的数据状态**
  - 改变前的数据状态是可以恢复的
  - 执行 DML 操作的用户可以通过 SELECT 语句查询之前的修正
  - 其他用户不能看到当前用户所做的改变，直到当前用户结束事务。
  - DML **语句所涉及到的行被锁定， 其他用户不能操作**。

- **提交后的数据状态**
  - 数据的改变已经被保存到数据库中。
  - 改变前的数据已经丢失。
  - 所有用户可以看到结果。
  - 锁被释放， 其他用户可以操作涉及到的数据。
  - 所有保存点被释放。

- **提交数据示例：**
  - 改变数据

```
DELETE FROM employees
WHERE  employee_id = 99999;
1 row deleted.

INSERT INTO departments
VALUES (290, 'Corporate Tax', NULL, 1700);
1 row inserted.
```

  - 提交改变

```
COMMIT;
Commit complete.
```

- **数据回滚后的状态**

使用 ROLLBACK 语句可使数据变化失效:

- 数据改变被取消。

- 修改前的数据状态被恢复。
- 锁被释放。

```
DELETE FROM copy_emp;
22 rows deleted.
ROLLBACK;
Rollback complete.
```

9.5.4. 数据库的事务隔离级别

- 对于同时运行的多个事务，当这些事务访问数据库中相同的数据时，如果没有采取必要的隔离机制，就会导致各种并发问题:
  - **脏读**: 对于两个事物 T1, T2, T1 读取了已经被 T2 更新但还**没有被提交**的字段. 之后, 若 T2 回滚, T1 读取的内容就是临时且无效的.
  - **不可重复读**: 对于两个事物 T1, T2, T1 读取了一个字段, 然后 T2 **更新**了该字段. 之后, T1 再次读取同一个字段, 值就不同了.
  - **幻读**: 对于两个事物 T1, T2, T1 从一个表中读取了一个字段, 然后 T2 在该表中**插入**了一些新的行. 之后, 如果 T1 再次读取同一个表, 就会多出几行.
- 数据库事务的隔离性: 数据库系统必须具有隔离并发运行各个事务的能力, 使它们不会相互影响, 避免各种并发问题.
- 一个事务与其他事务隔离的程度称为隔离级别. 数据库规定了多种事务隔离级别, 不同隔离级别对应不同的干扰程度, 隔离级别越高, 数据一致性就越好, 但并发性越弱

- 数据库提供的 4 种事务隔离级别:

隔离级别	描述
READ UNCOMMITTED (读未提交数据)	允许事务读取未被其他事物提交的变更. 脏读, 不可重复读和幻读的问题都会出现
READ COMMITED (读已提交数据)	只允许事务读取已经被其它事务提交的变更. 可以避免脏读, 但不可重复读和幻读问题仍然可能出现
REPEATABLE READ (可重复读)	确保事务可以多次从一个字段中读取相同的值. 在这个事务持续期间, 禁止其他事物对这个字段进行更新. 可以避免脏读和不可重复读, 但幻读的问题仍然存在.
SERIALIZABLE(串行化)	确保事务可以从一个表中读取相同的行. 在这个事务持续期间, 禁止其他事务对该表执行插入, 更新和删除操作. 所有并发问题都可以避免, 但性能十分低下.

- **Oracle 支持的 2 种事务隔离级别：READ COMMITED, SERIALIZABLE. Oracle 默认的**事务隔离级别为: READ COMMITED
- **Mysql 支持 4 中事务隔离级别. Mysql 默认的事务隔离级别为: REPEATABLE READ**

10.管理表结构（DML）

10.1. 显示表结构

使用 DESCRIBE 命令，表示表结构

```
DESC[RIBE] tablename
```

示例：

```
DESCRIBE employees
```

Name	Null?	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
FIRST_NAME		VARCHAR2(20)
LAST_NAME	NOT NULL	VARCHAR2(25)
EMAIL	NOT NULL	VARCHAR2(25)
PHONE_NUMBER		VARCHAR2(20)
HIRE_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
SALARY		NUMBER(8,2)
COMMISSION_PCT		NUMBER(2,2)
MANAGER_ID		NUMBER(6)
DEPARTMENT_ID		NUMBER(4)

10.2. 数据类型

数据类型	描述
VARCHAR2(*size*)	可变长字符数据
CHAR(*size*)	定长字符数据
NUMBER(*p,s*)	可变长数值数据
DATE	日期型数据
LONG	可变长字符数据，最大可达到 2G
CLOB	字符数据，最大可达到 4G

RAW and LONG RAW	原始的二进制数据
BLOB	二进制数据，最大可达到 4G
BFILE	存储外部文件的二进制数据，最大可达到 4G

10.3. 新建表（Create Table）

10.3.1. 建表规则

表名和列名：

- 必须**以字母开头**
- 必须在 1-30 个字符之间
- 必须只能包含 A-Z, a-z, 0-9, _, $, 和 #
- 必须不能和用户定义的其他对象重名
- 必须不能是 Oracle 的保留字
- Oracle 默认存储是都存为大写
- 数据库名只能是 1~8 位，datalink 可以是 128 位，和其他一些特殊字符

10.3.2. CREATE TABLE 语句语法

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

- 必须指定：
  - 表名
  - 列名，数据类型，数据类型的大小

- 要求必须具备以个资源才可以建表：
  - CREATE TABLE 权限
  - 存储空间

10.3.3. Default 值

- 执行 insert 操作时，可以为其指定默认值

```
... hire_date DATE DEFAULT SYSDATE, ...
```

- 值、表达式和 SQL 语句都可以作为默认值
- 其他的列名或者是伪列都是非法的
- 默认值的类型必须和该列的类型一致

```
CREATE TABLE hire_dates
        (id           NUMBER(8),
        hire_date DATE DEFAULT SYSDATE);
Table created.
```

10.3.4. 建表示例

- 语句

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

- 确认

```
DESCRIBE dept
```

Name	Null?	Type
DEPTNO		NUMBER(2)
DNAME		VARCHAR2(14)
LOC		VARCHAR2(13)

10.3.5. 使用子查询创建表

- 使用 AS *subquery* 选项，**将创建表和插入数据结合起来**

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

- 指定的列和子查询中的列要一一对应
- 通过列名和默认值定义列

- **使用子查询创建表举例**

```
CREATE TABLE  dept80
  AS
    SELECT   employee_id, last_name,
             salary*12 ANNSAL,
             hire_date
    FROM     employees
    WHERE    department_id = 80;
Table created.
```

```
DESCRIBE dept80
```

Name	Null?	Type
EMPLOYEE_ID		NUMBER(6)
LAST_NAME	NOT NULL	VARCHAR2(25)
ANNSAL		NUMBER
HIRE_DATE	NOT NULL	DATE

10.4. 修改表结构（Alter Table）

10.4.1. 语法说明

使用 ALTER TABLE 语句可以:

- 追加新的列
- 修改现有的列
- 删除一个列

- 使用 ALTER TABLE 语句追加，修改，或删除列的语法

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

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

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

```
ALTER TABLE table_name rename column old_column_name
to new_column_name
```

10.4.2. 示例

10.4.2.1. 追加一个新列

- 使用 ADD 子句追加一个新列

```
ALTER TABLE dept80
ADD         (job_id VARCHAR2(9));
Table altered.
```

- 新列是表中的最后一列

EMPLOYEE_ID	LAST_NAME	ANNSAL	HIRE_DATE	JOB_ID
149	Zlotkey	126000	29-JAN-00	
174	Abel	132000	11-MAY-96	
176	Taylor	103200	24-MAR-98	

10.4.2.2. 修改一个列

- 可以修改列的数据类型, 尺寸, 和默认值

```
ALTER TABLE   dept80
MODIFY        (last_name VARCHAR2(30));
Table altered.
```

- 对默认值的修改只影响今后对表的修改

10.4.2.3. 删除一个列

使用 DROP COLUMN 子句删除不再需要的列.

```
ALTER TABLE  dept80
DROP COLUMN  job_id;
Table altered.
```

10.5. 清空表（Truncate）

- **TRUNCATE TABLE** 语句:
  - 删除表中所有的数据
  - 释放表的存储空间

```
TRUNCATE TABLE detail_dept;
Table truncated.
```

- TRUNCATE 语句**不能回滚**
- 可以使用 DELETE 语句删除数据

10.6. 删除表（Drop Table）

- 数据和结构都被删除
- 所有正在运行的相关事物被提交

- 所有相关索引被删除
- DROP TABLE 语句不能回滚，但是可以闪回

```
DROP TABLE dept80;
Table dropped.
```

10.7. 改变对象的名称

- 执行 RENAME 语句改变表，视图，序列，或同义词的名称

```
RENAME dept TO detail_dept;
Table renamed.
```

- 必须是对象的拥有者

10.8. 约束

10.8.1. 约束说明

- 约束是表一级的限制
- 如果存在依赖关系，约束可以防止错误的删除数据
- 约束的类型：
  - NOT NULL
  - UNIQUE
  - PRIMARY KEY
  - FOREIGN KEY
  - CHECK

- 约束规则
  - 用户可以自定义约束，也可以使用 Oracle Server 的 sys_cn 格式命名约束
  - 约束创建的时机：
    - 创建表的时候，同时创建约束
    - 表结构创建完成后
  - 约束可以定义在列一级，或者是表一级
  - 通过数据字典查看约束

10.8.2. 非空约束

- 保证列的值不能为空:

EMPLOYEE_ID	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID
100	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000	90
101	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	17000	90
102	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000	90
103	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	9000	60
104	Ernst	BERNST	590.423.4568	21-MAY-91	IT_PROG	6000	60
178	Grant	KGRANT	011.44.1644.429263	24-MAY-99	SA_REP	7000	
200	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	4400	10

20 rows selected.

NOT NULL约束　　NOT NULL约束　　没有定义NOT NULL 约束

10.8.3. 唯一性约束

UNIQUE约束

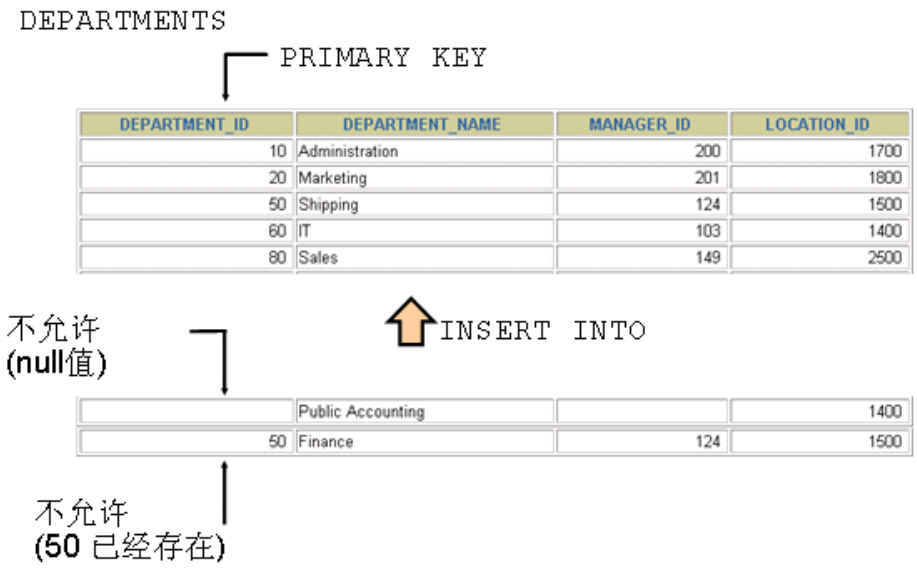
EMPLOYEES

EMPLOYEE_ID	LAST_NAME	EMAIL
100	King	SKING
101	Kochhar	NKOCHHAR
102	De Haan	LDEHAAN
103	Hunold	AHUNOLD
104	Ernst	BERNST

INSERT INTO

208	Smith	JSMITH	插入成功
209	Smith	JSMITH	插入失败

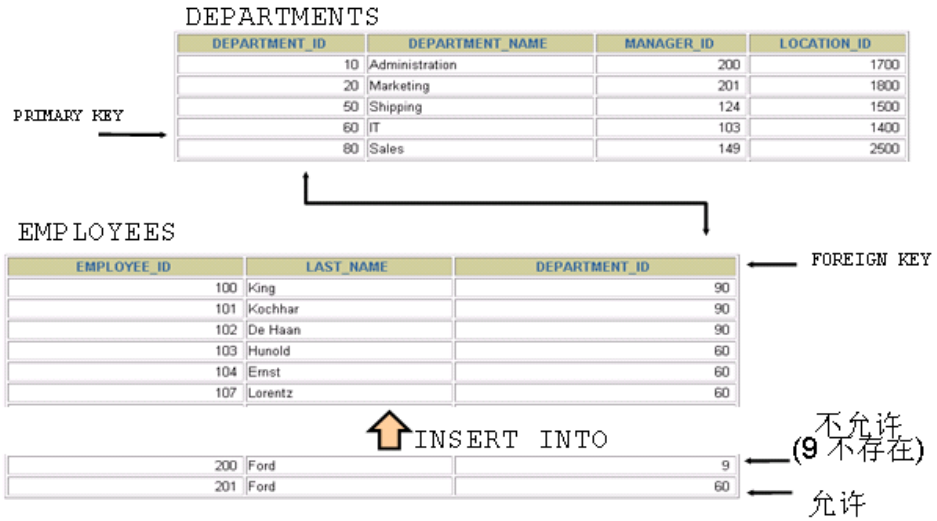
10.8.4. 主键约束



DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500

	Public Accounting		1400
50	Finance	124	1500

10.8.5. 外键约束



DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
101	Kochhar	90
102	De Haan	90
103	Hunold	60
104	Ernst	60
107	Lorentz	60

200	Ford	9
201	Ford	60

- FOREIGN KEY: 在子表中，定义了一个表级的约束
- REFERENCES: 指定表和父表中的列
- ON DELETE CASCADE: 当删除父表时，级联删除子表记录
- ON DELETE SET NULL: 将子表的相关依赖记录的外键值置为 null

10.8.6. check 约束

- 定义每一行记录所必须满足的条件
- 下面的表达式可以使用在 check 约束中:
  - 引用 CURRVAL, NEXTVAL, LEVEL, 和 ROWNUM
  - 调用 SYSDATE, UID, USER, 和 USERENV 函数
  - 另一个表的查询记录

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

10.8.7. 一个用到以上所有约束的示例

```
CREATE TABLE employees
    ( employee_id    NUMBER(6)
        CONSTRAINT     emp_employee_id   PRIMARY KEY
    , first_name     VARCHAR2(20)
    , last_name      VARCHAR2(25)
        CONSTRAINT     emp_last_name_nn  NOT NULL
    , email          VARCHAR2(25)
        CONSTRAINT     emp_email_nn      NOT NULL
        CONSTRAINT     emp_email_uk      UNIQUE
    , phone_number   VARCHAR2(20)
    , hire_date      DATE
        CONSTRAINT     emp_hire_date_nn  NOT NULL
    , job_id         VARCHAR2(10)
        CONSTRAINT     emp_job_nn        NOT NULL
    , salary         NUMBER(8,2)
        CONSTRAINT     emp_salary_ck     CHECK (salary>0)
    , commission_pct NUMBER(2,2)
    , manager_id     NUMBER(6)
    , department_id  NUMBER(4)
        CONSTRAINT     emp_dept_fk       REFERENCES
           departments (department_id));
```

10.8.8. 违反约束

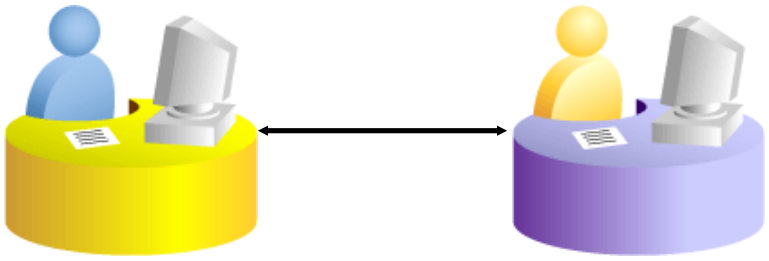
- 不能删除有外键约束的记录

```
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
```

10.9. 查询其他用户的表

- 其他用户的表不属于本用户的空间
- 如果要查询其他用户下的表，要使用其他用户的用户名作为前缀。



●USERA

●SELECT *

●USERB

●SELECT *

11.其他数据库对象

11.1. 视图

11.1.1. 视图说明

表EMPLOYEES：

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALA
100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	240
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	170
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	170
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	90
104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-91	IT_PROG	60
107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-99	IT_PROG	42
124	Kevin	Mourgos	KMOURGOS	650.123.5234	16-NOV-99	ST_MAN	58
141	Trenna	Rajs	TRAJS	650.121.8009	17-OCT-95	ST_CLERK	35
142	Curtis	Davies	CDAVIES	650.121.2994	29-JAN-97	ST_CLERK	31
143	Randall	Matos	RMATOS	650.121.2874	15-MAR-98	ST_CLERK	26
144	Peter	Vargas	PVARGAS	650.121.2004	09-JUL-98	ST_CLERK	25
149	Eleni	Zlotkey	EZLOTKEY	011.44.1344.429018	29-JAN-00	SA_MAN	105
174	Ellen	Abel	EABEL	011.44.1644.429267	11-MAY-96	SA_REP	110
176	Jonathon	Taylor	JTAYLOR	011.44.1644.429265	24-MAR-98	SA_REP	86
178	Kimberely	Grant	KGRANT	011.44.1644.429263	24-MAY-99	SA_REP	70
200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	44
201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-96	MK_MAN	130
202	Pat	Fay	PFAY	603.123.6666	17-AUG-97	MK_REP	60
205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	120
206	William	Gietz	WGIETZ	515.123.8181	07-JUN-94	AC_ACCOUNT	83

EMPLOYEE_ID	LAST_NAME	SALARY
149	Zlotkey	10500
174	Abel	11000
176	Taylor	8600

20 rows selected.

- 视图是一种虚表.
- **视图建立在已有表的基础上**, 视图赖以建立的这些表称为**基表**。
- 向视图提供数据内容的语句为 SELECT 语句, 可以将视图理解为**存储起来的 SELECT 语句**.
- 视图向用户提供基表数据的另一种表现形式

- 视图的优点
  - 简化复杂查询
  - 限制数据访问
  - 同样的数据，可以有不同的显示方式

- 注意：不建议通过视图对表进行修改
  视图不能提高性能

11.1.2. 创建视图

- 使用下面的语法格式创建视图

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

  - FORCE：　　子查询不一定存在
  - NOFORCE：　　子查询存在（默认）
  - WITH READ ONLY:只能做查询操作
- 子查询可以是复杂的 SELECT 语句

- 创建视图举例

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

- 描述视图结构

```
DESCRIBE empvu80
```

- 创建视图时在子查询中给列定义别名

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

- 在选择视图中的列时应使用别名

11.1.3. 创建复杂视图

复杂视图举例：查询各个部门的最低工资，最高工资，平均工资

```
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.
```

11.1.4. 查询视图

```
SELECT *
FROM  salvu50;
```

ID_NUMBER	NAME	ANN_SALARY
124	Mourgos	69600
141	Rajs	42000
142	Davies	37200
143	Matos	31200
144	Vargas	30000

11.1.5. 修改视图

- 使用 CREATE **OR REPLACE** VIEW 子句**修改视图**

```
CREATE OR REPLACE VIEW empvu80
  (id_number, name, sal, department_id)
AS SELECT  employee_id, first_name || ' ' || last_name,
           salary, department_id
   FROM    employees
   WHERE   department_id = 80;
View created.
```

- CREATE VIEW 子句中各列的别名应和子查询中各列相对应

11.1.6. 视图中使用 DML 的规定

- 可以在简单视图中执行 DML 操作
- 当视图定义中包含以下元素之一时不能使用 delete:
  - 组函数
  - GROUP BY 子句
  - DISTINCT 关键字
  - ROWNUM 伪列

当视图定义中包含以下元素之一时不能使用 update :

- 组函数
- GROUP BY 子句
- DISTINCT 关键字
- ROWNUM 伪列
- 列的定义为表达式

当视图定义中包含以下元素之一时不能使用 insert :

- 组函数
- GROUP BY 子句
- DISTINCT 关键字
- ROWNUM 伪列
- 列的定义为表达式
- 表中非空的列在视图定义中未包括

11.1.7. 屏蔽 DML 操作

- 可以使用 **WITH READ ONLY** 选项屏蔽对视图的 DML 操作
- 任何 DML 操作都会返回一个 Oracle server 错误

```
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.
```

11.1.8. 删除视图

删除视图只是删除视图的定义，并不会删除基表的数据

```
DROP VIEW view;
```

```
DROP VIEW empvu80;
View dropped.
```

11.2. 序列

11.2.1. 什么是序列

序列: 可供多个用户用来产生唯一数值的数据库对象

- 自动提供唯一的数值
- 共享对象
- **主要用于提供主键值**
- 将序列值装入内存可以提高访问效率

11.2.2. 创建序列

- **CREATE SEQUENCE 语句**

定义序列:

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

例：

- 创建序列 DEPT_DEPTID_SEQ 为表 DEPARTMENTS 提供主键
- 不使用 CYCLE 选项

```
CREATE SEQUENCE dept_deptid_seq
                INCREMENT BY 10
                START WITH 120
                MAXVALUE 9999
                NOCACHE
                NOCYCLE;
Sequence created.
```

11.2.3. 查询序列

- 查询数据字典视图 **USER_SEQUENCES** 获取序列定义信息

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

- 如果指定 NOCACHE 选项，则列 LAST_NUMBER 显示序列中下一个有效的值

11.2.4. NEXTVAL 和 CURRVAL 伪列

- NEXTVAL 返回序列中下一个有效的值，任何用户都可以引用
- CURRVAL 中存放序列的当前值
- **NEXTVAL 应在 CURRVAL 之前指定** ，二者应同时有效

11.2.5. 序列应用举例

```
INSERT INTO departments(department_id,
            department_name, location_id)
VALUES      (dept_deptid_seq.NEXTVAL,
            'Support', 2500);
1 row created.
```

- 序列 DEPT_DEPTID_SEQ 的当前值

```
SELECT   dept_deptid_seq.CURRVAL
FROM     dual;
```

11.2.6. 使用序列

- 将序列值装入内存可提高访问效率
- 序列在下列情况下出现**裂缝**:
  - 回滚
  - 系统异常
  - 多个表同时使用同一序列
- 如果不将序列的值装入内存(NOCACHE), 可使用表 USER_SEQUENCES 查看序列当前的有效值

11.2.7. 修改序列

修改序列的增量, 最大值, 最小值, 循环选项, 或是否装入内存

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

- **修改序列的注意事项**
  - 必须是序列的拥有者或对序列有 ALTER 权限
  - 只有将来的序列值会被改变
  - 改变序列的初始值只能通过删除序列之后重建序列的方法实现

11.2.8. 删除序列

- 使用 DROP SEQUENCE 语句删除序列
- 删除之后，序列不能再次被引用

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

11.3. 索引

11.3.1. 索引说明

- 一种独立于表的模式对象，可以存储在与表不同的磁盘或表空间中
- 索引被删除或损坏，不会对表产生影响，其影响的只是**查询**的速度
- 索引一旦建立，Oracle 管理系统会对其进行自动维护，而且由 Oracle 管理系统决定何时使用索引. 用户不用在查询语句中指定使用哪个索引
- 在删除一个表时，所有基于该表的索引会自动被删除
- 通过指针**加速 Oracle 服务器的查询速度**
- 通过快速定位数据的方法，减少磁盘 I/O

11.3.2. 创建索引

- 自动创建: 在定义 PRIMARY KEY 或 UNIQUE 约束后系统自动在相应的列上创建**唯一性**索引
- 手动创建: 用户可以在其它列上创建非唯一的索引，以加速查询

- 在一个或多个列上创建索引

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

- 在表 EMPLOYEES的列 LAST_NAME 上创建索引

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

11.3.3. 什么时候创建索引

以下情况可以创建索引:

- 列中数据值分布范围很广
- 列经常在 WHERE 子句或连接条件中出现
- 表经常被访问而且数据量很大 ， 访问的数据大概占数据总量的 2%到 4%
- **什么时候不要创建索引**

11.3.4. 下列情况不要创建索引：

- 表很小
- 列不经常作为连接条件或出现在 WHERE 子句中
- 查询的数据大于 2%到 4%
- 表经常更新

11.3.5. 查询索引

- 可以使用数据字典视图 USER_INDEXES 和 USER_IND_COLUMNS 查看索引的信息

```
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';
```

11.3.6. 删除索引

- 使用DROP INDEX 命令删除索引

```
DROP INDEX index;
```

- 删除索引UPPER_LAST_NAME_IDX

```
DROP INDEX upper_last_name_idx;
Index dropped.
```

- 只有索引的拥有者或拥有DROP ANY INDEX权限的用户才可以删除索引

11.4. 同义词

11.4.1. 同义词说明

使用同义词访问相同的对象:

- 方便访问其它用户的对象
- 缩短对象名字的长度

```
CREATE [PUBLIC] SYNONYM synonym
FOR    object;
```

11.4.2. 创建和删除同义词

- 为视图DEPT_SUM_VU 创建同义词

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

- 删除同义词

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

12. PL/SQL 程序设计

12.1. 什么是 PL/SQL

- PL/SQL（Procedure Language/SQL）
- PLSQL 是 Oracle 对 sql 语言的过程化扩展
- 指在 SQL 命令语言中增加了过程处理语句（如分支、循环等），使 SQL 语言具有过程处理能力。

- 把 SQL 语言的数据操纵能力与过程语言的数据处理能力结合起来，使得 PLSQL 面向过程但比过程语言简单、高效、灵活和实用。
- Plsql(oracle), Transact-sql(SQL server)

12.2. HelloWorld

- 写一段 PL/SQL 程序，在屏幕上打印“Hello World!”

```
SQL> declare
  2  begin
  3     dbms_output.put_line('Hello World');
  4  end;
  5  /
```

- 注意：如果要在屏幕上输出信息，需要将 serveroutput 开关打开：
  set serveroutput on

```
--打印 Hello World
set serveroutput on

 declare
    --变量说明
 begin
    --程序体
    dbms_output.put_line('Hello World');
 end;
 /
```

12.3. PL/SQL 程序结构

```
declare
      说明部分     （变量说明，光标申明，例外说明 ）
begin
      语句序列     （DML 语句）…
exception
      例外处理语句
end;
/
```

- 如果没有变量，就可以不写 declare 段
- PL/SQL 对大小写不敏感。
- 赋值是使用冒号等号":="（中间不能有空格）
- 注释使用“--”或是“/* ... */”（就是 SQL 注释）
- 注意最后的 end 后面有个分号。

12.4. 变量与赋值

12.4.1. 声明变量

- 说明变量 (char, varchar2, date, number, boolean, long)

```
var1        char(15) ;
married     boolean :=true ;
psal        number(7,2);
my_name     emp.ename%type;
emp_rec     emp%rowtype;
```

说明变量名、数据类型和长度后用分号结束说明语句。

引用型变量，即my_name的类型与emp表中ename列的类型一样

记录型变量

- 记录变量分量的引用：

```
emp_rec.ename := 'ADAMS';
```

✱ 属性类型有两种：
  - ✱ %TYPE - 引用变量和数据库列的数据类型
  - ✱ %ROWTYPE - 提供表示表中一行的记录类型

✱ 使用属性类型的优点：
  - ✱ 不需要知道被引用的表列的具体类型
  - ✱ 如果被引用对象的数据类型发生改变，PL/SQL 变量的数据类型也随之改变

12.4.2. 赋值语句

- var1:='this is a argument';
- emp_rec.sal:= sal *2 + nvl(comm, 0);
- sum_sal:=sum_sal+v_sal;

- FETCH c1 INTO e_eno , e_sal ;
- Select sal into psal from emp where ....

12.5. IF 语句

12.5.1. 语法

- 只有 IF 的情况：

```
IF   条件  THEN 语句 1;
     语句 2;
END IF;
```

- 带 ELSE 的情况：

```
IF  条件   THEN   语句序列 1;
    ESLE    语句序列 2;
END   IF;
```

- IF ... ELSE IF ... ELSE 的情况：

```
IF   条件  THEN 语句;
   ELSIF  语句  THEN  语句;
   ELSE    语句;
END  IF;
```

注意：是 **ELSIF**，不是 **ELSEIF**。

12.5.2. 示例

- 要求：判断用户输入的数字。
- 提示：
  - 从键盘输入：`accept num prompt '请输入一个数字';`
  - 得到键盘输入的值：`pnum number := &num;`

```
declare
 --保存用户输入的数字( 隐式转换)
 pnum  number := 5; -- 设置不同的值进行测试
begin
  --判断
  if pnum=0 then dbms_output.put_line('您输入的是 0');
    elsif  pnum=1 then dbms_output.put_line('您输入的是 1');
    elsif  pnum=2 then dbms_output.put_line('您输入的是 2');
    else   dbms_output.put_line('其他数字');
  end if;
end;
/
```

12.6. 循环语句

12.6.1. 语法

- Loop

```
Loop
    EXIT [when   条件];
    ...
End loop
```

- For

```
FOR   i   IN   1..3
LOOP
    语句序列 ;
END    LOOP;
```

- While

```
WHILE  total  <= 25000
LOOP
    ...
    total := total + salary;
END  LOOP;
```

12.6.2. 示例

```
declare
  pnum number := 1;
begin
  loop
    -- 退出条件
    exit when pnum > 10;

    --打印
    dbms_output.put_line(pnum);

    --加一
    pnum := pnum + 1;

  end loop;
end;
/
```

```
begin
   for i in 1 .. 10
   loop
          dbms_output.put_line( i );
   end loop;
end;
/
```

```
declare
   v_num number := 1;
begin
   while v_num <= 10
   loop
          dbms_output.put_line( v_num  );
          v_num := v_num + 1;
   end loop;
end;
/
```

批注 [t6]:

12.7. 光标(Cursor)

12.7.1. 使用光标

- 说明光标语法：
  CURSOR 光标名 [(参数名 数据类型[,参数名 数据类型]...)]
  IS SELECT 语句;
- 用于存储一个查询返回的多行数据
  例如：
  cursor c1 is select ename from emp;

- 打开光标：open c1; (打开光标执行查询)
- 取一行光标的值：fetch c1 into pjob; (取一行到变量中)
- 关闭光标：close c1;(关闭游标释放资源)

- 注意： 上面的 pjob 必须与 emp 表中的 job 列类型一致：
  - 定义：pjob emp.empjob%type;

12.7.2. 带参数的光标

```
cursor c2(jobc varchar2)
is
select ename, sal from emp
where job=jobc;
```

执行语句：

```
Open c2('clerk');
```

12.7.3. 带参数的光标示例

- 写一段 PL/SQL 程序，为部门号为 10 的员工涨工资。

12.7.4. 示例：按员工的工种长工资,总裁 1000 元，经理长 800 元其，他人员长 400 元。要真正的修改数据

```
declare
  cursor c1 is select empno,empjob from emp; --定义光标保存查询结果
  pno emp.empno% TYPE;
  pjob emp.empjob% TYPE;
begin
  open c1;--打开游标，即执行查询

  --循环开始
  loop
     fetch c1 into pno, pjob; --取一条记录中的员工编号和工种，并付给pno pjob
     --循环退出条件
     exit when c1%notfound;

     --判断员工工种，执行加薪
     if pjob = 'PRESIDENT' then update emp set sal = sal + 1000 where empno = pno;
     elsif pjob = 'MANAGER' then update emp set sal = sal + 800 where empno = pno;
     else update emp set sal = sal + 400 where empno = pno;
     end if;

  end loop;

  close c1;--关闭游标
  commit; --提交修改
end;
/
```

12.8. 例外（异常）

- 例外是程序设计语言提供的一种功能，用来增强程序的健壮性和容错性。

12.8.1. Oracle 的异常处理

- 系统定义例外
  - No_data_found （没有找到数据）
  - Too_many_rows （select …into 语句匹配多个行）
  - Zero_Divide （ 被零除）
  - Value_error （算术或转换错误）
  - Timeout_on_resource （在等待资源时发生超时）
- 用户定义的例外
- 演示：系统定义例外（被 0 除）

12.8.2. 演示：用户定义例外及处理例外

- 在 declare 节中定义例外
  - out_of exception ;
- 在可行语句中引起例外
  - raise out_of ；
- 在 Exception 节处理例外
  - when Out_of then …

```
DECLARE
My_job   char(10);
v_sal   emp.sal%type;
No_data   exception;
cursor c1 is select
distinct job from emp
order by job;
```

```
begin
open c1;
Fetch c1 into v_job;
IF c1%notFOUND then raise no_data;
end if;
...
EXCEPTION
WHEN no_data  THEN insert into emp
values('fetch语句没有获得数据或数据已经处理完');
END;
```

12.9. 实例

12.9.1. 实例 1：按员工的工种长工资,总裁 1000 元，经理长 800 元，其他人员长 400 元

12.9.2. 实例 2：统计每年入职的员工个数

12.9.3. 实例 3：为员工涨工资，从最低工资调起，工资总额不能超过 5 万元

```
declare
   cursor c1 is select empno,sal from emp order by sal;
   salTotal NUMBER; --记录工资总额
   empCount NUMBER := 0; --涨工资的人数

   pempno emp.empno% TYPE; --记录员工的编号
   psal   emp.sal%type;    --记录员工的工资
begin
  --得到当前总工资
  select sum(sal) into salTotal from emp;
  --打开游标
  open c1;
  --执行循环
  while salTotal <= 50000
  loop
      fetch c1 into pempno, psal;--取出一条记录
      exit when c1%notfound;
      update emp set sal = sal * 1.1 where empno = pempno; --执行加薪
      --记录涨工资后的总额
      salTotal := salTotal + psal*0.1;
      --记录涨工资的人数
      empCount := empCount + 1;
  end loop;
  close c1;
  commit;

  dbms_output.put_line('涨工资人数:' || empCount || '  工资总额:' || salTotal);
end;
/
```

12.9.4. 案例 4：实现按部门分段（6000 以上、(6000，3000)、3000 元以下)统计各工资段的职工人数、以及各部门的工资总额

题目：

用 PL/SQL 语言编写一程序，实现按部门分段（6000 以上、(6000，3000)、3000 元以下)统计各工资段的职工人数、以及各部门的工资总额(工资总额中不包括奖金)，参考如下格式：

部门	小于3000数	3000-6000	大于6000	工资总额
10	2	1	0	8750
20	3	2	0	10875
30	6	0	0	9400

- 提示：可以创建一张新表用于保存数据

```
create table msg1
(deptno    number,
  emp_num1 number,
  emp_num2 number,
  emp_num3 number,
  sum_sal number);
```

13. 存储过程和存储函数

13.1. 存储过程与存储函数说明

指存储在数据库中供所有用户程序调用的子程序叫存储过程、存储函数。

- 什么时候用存储过程/存储函数
  原则：如果只有一个返回值，用存储函数；否则，就用存储过程。

13.2. 存储过程

13.2.1. 创建存储过程

用 CREATE PROCEDURE 命令建立存储过程。语法如下：

```
create [or replace] PROCEDURE 过程名[(参数列表)]
AS
    变量声明
PLSQL 子程序体;
```

13.2.2. 调用存储过程

- 方法一：

```
set serveroutput on
begin
 raisesalary(7369);
end;
/
```

- 方法二：
  exec[ute] 存储过程名

```
set serveroutput on
exec raisesalary(7369);
```

13.2.3. 存储过程示例

- 为指定的职工在原工资的基础上长 10%的工资，并打印涨工资前和涨工资后的工资

```
/*
为指定的职工在原工资的基础上长10%的工资,并打印涨工资前和涨工资后的工资

可能用到的sql语句
update emp set sal = sal * 1.1 where empno = empid;

*/

create or replace procedure raiseSalary(empid in number)
as
  pSal emp.sal%type; --保存员工当前工资
begin
  --查询该员工的工资
  select sal into pSal from emp where empno=empid;
  --给该员工涨工资
  update emp set sal = sal * 1.1 where empno = empid;

  --打印涨工资前后的工资
  dbms_output.put_line('员工号:' || empid || ' 涨工资前:' || psal || ' 涨工资后' || psal * 1.1);
end;
/
```

13.3. 存储函数

13.3.1. 创建存储函数

函数（Function）为一命名的存储程序，可带参数，并返回一计算值。函数和过程的结构类似，但必须有一个 RETURN 子句，用于返回函数值。函数说明要指定函数名、结果值的类型，以及参数类型等。

语法如下：

```
CREATE [OR REPLACE] FUNCTION 函数名(参数列表)
RETURN  函数值类型
AS
    变量声明
PLSQL 子程序体;
```

13.3.2. 调用存储函数

```
declare
    v_sal number;
begin
    v_sal:=queryEmpSalary(7934);
    dbms_output.put_line('salary is:' || v_sal);
end;
/
```

或写成：

```
begin
    dbms_output.put_line('salary is:' || queryEmpSalary(7934));
 end;
```

13.3.3. 存储函数示例

```
/*
查询某职工的总收入。
*/

create or replace function queryEmpSalary(empid in number)
  RETURN NUMBER
as
  pSal number; --定义变量保存员工的工资
  pComm number; --定义变量保存员工的奖金
begin
  select sal,comm into pSal, pcomm from emp where empno = empid;
  return psal*12+ nvl(pcomm,0);
end;
/
```

13.4. 过程和函数中的 in 和 out

13.5. 在 Java 语言中调用存储过程与存储函数

13.5.1. JDBC 调用存储过程

- 存储过程：

```
create or replace procedure testprocedure(eid in number, empname out varchar,empsal out NUMBER )
as
begin
  select ename,sal into empname, empsal from emp where empno=eid;
end;

```

- Java 程序：

```
//创建CallableStatement
CallableStatement call = conn.prepareCall("{call testprocedure(?,?,?)}");
//设置参数
call.setInt(1, 7369);
call.registerOutParameter(2, oracle.jdbc.OracleTypes.VARCHAR);
call.registerOutParameter(3, oracle.jdbc.OracleTypes.NUMBER);
//执行存储过程
call.execute();
//输出结果
String name = call.getString(2);
int sal = call.getInt(3);
System.out.println(name);
System.out.println(sal);
```

13.5.2. JDBC 调用存储函数

- 存储函数：

```
create or replace function testfunction(eid in number, empname out varchar,empsal out NUMBER )
return NUMBER
as
begin
  select ename,sal into empname, empsal from emp where empno=eid;
  return empsal * 12;
end;
```

- Java 程序

```
//创建CallableStatement
CallableStatement call = conn.prepareCall("{?=call testfunction(?,?,?)}");
//设置参数
call.registerOutParameter(1,oracle.jdbc.OracleTypes.NUMBER);
call.setInt(2, 7369);
call.registerOutParameter(3, oracle.jdbc.OracleTypes.VARCHAR);
call.registerOutParameter(4, oracle.jdbc.OracleTypes.NUMBER);
//执行存储函数
call.execute();
//输出结果
int annualSal = call.getInt(1);
String name = call.getString(3);
int sal = call.getInt(4);
System.out.println(annualSal);
System.out.println(name);
System.out.println(sal);
```

13.5.3. JDBC 调用带光标类型 out 参数的存储过程或函数

14. 触发器

14.1. 说明

- 数据库触发器是一个与表相关联的、存储的 PL/SQL 程序。每当一个特定的数据操作语句(Insert,update,delete)在指定的表上发出时，Oracle 自动地执行触发器中定义的语句序列。

- 触发器的类型
  - 语句级触发器
    在指定的操作语句操作之前或之后执行一次，不管这条语句影响了多少行 。

  - 行级触发器（FOR EACH ROW）
    触发语句作用的每一条记录都被触发。在行级触发器中使用 old 和 new 伪记录变量, 识别值的状态。

- 触发器可用于
  - 数据确认
  - 实施复杂的安全性检查
  - 做审计，跟踪表上所做的数据操作等

- 数据的备份和同步

14.2. 创建触发器

```
CREATE  [or REPLACE] TRIGGER  触发器名
    {BEFORE | AFTER}
    {DELETE | INSERT | UPDATE [OF  列名]}
    ON  表名
    [FOR EACH ROW [WHEN(条件) ] ]
PLSQL 块
```

14.3. 触发语句与伪记录变量的值

触发语句	:old	:new
Insert	所有字段都是空(null)	将要插入的数据
Update	更新以前该行的值	更新后的值
delete	删除以前该行的值	所有字段都是空(null)

14.4. 示例 1：限制非工作时间向数据库插入数据

触发器(禁止在非工作时间向emp表中插入数据).sql

```
1  create or replace
2  trigger securityEmp
3  before insert on emp
4  declare
5
6  begin
7
8    if   to_char(sysdate,'day') in ('星期四','星期六','星期天')
9      or to_number(to_char(sysdate,'hh24')) not between 8 and 18  then
10       raise_application_error(-20001,'不能在非工作时间插入数据.');
11   end if;
12 end;
```

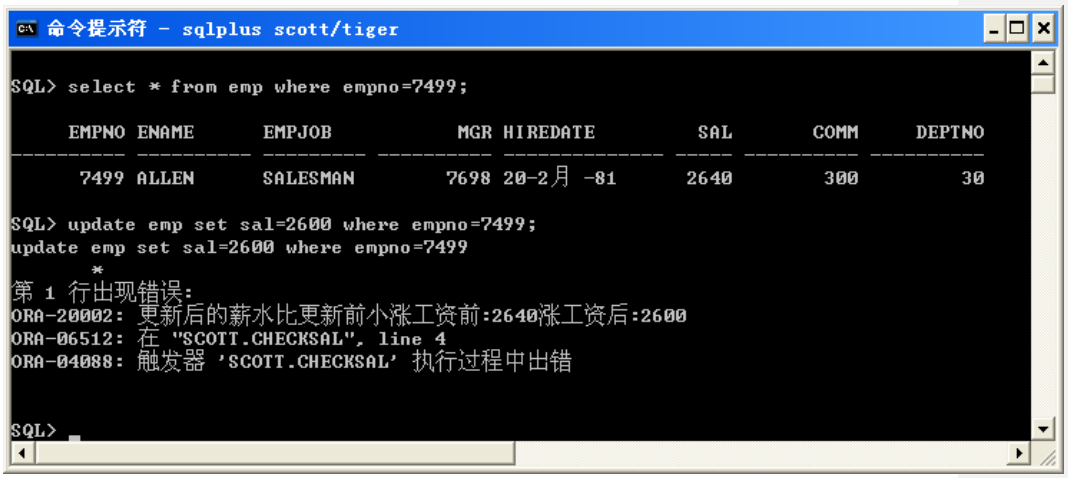
-20000到-20999之间

14.5. 示例 2：确认数据（检查 emp 表中 sal 的修改值不低于原值）

```
/*
示例2：确认数据（检查emp表中sal 的修改值不低于原值）
*/

create or replace trigger checkSal
before update of sal on emp
for each row
declare
begin
  if :new.sal <:old.sal THEN
    raise_application_error(-20002,'更新后的薪水比更新前小');
  end if;
end;
```

- 运行效果



14.6. 查询触发器、过程及函数

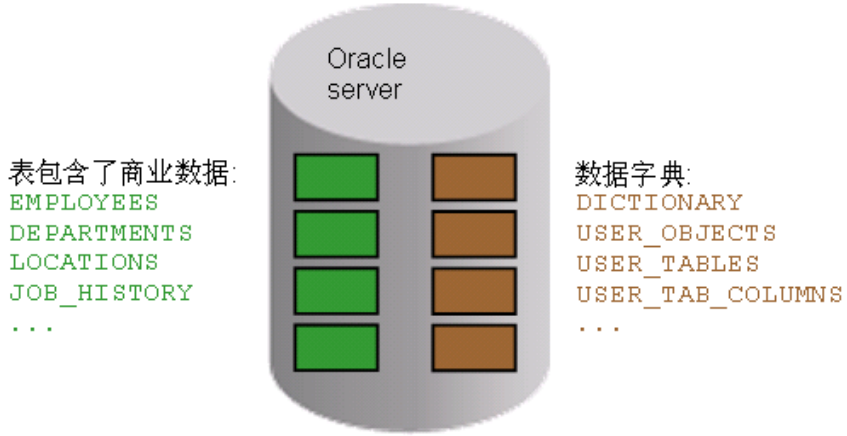
- Select * from user_triggers;

- Select * from user_source;

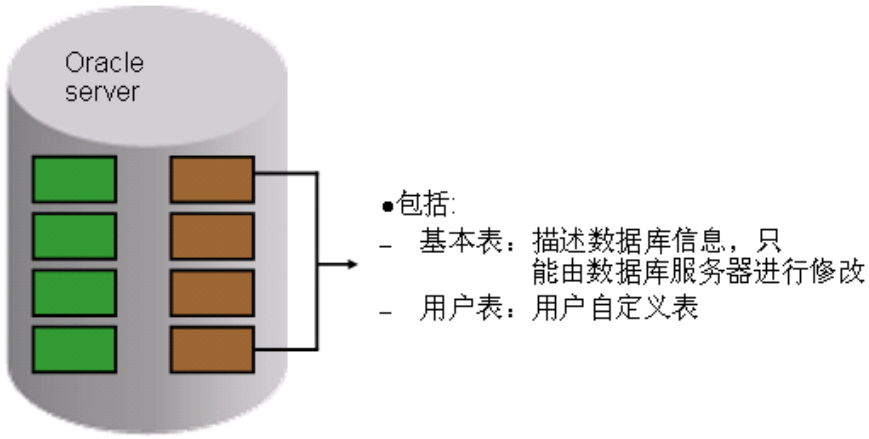
15.数据字典

15.1. 数据字典说明

15.1.1. 有 2 种作用的表



15.1.2. 数据字典的结构



15.1.3. 数据字典命名规则

前缀	说明
USER	用户自己的
ALL	用户可以访问到的
DBA	管理员视图
V$	性能或参数设置相关的数据

15.1.4. 如何使用数据字典视图

DESCRIBE DICTIONARY

Name	Null?	Type
TABLE_NAME		VARCHAR2(30)
COMMENTS		VARCHAR2(4000)

```
SELECT *
FROM     dictionary
WHERE    table_name = 'USER_OBJECTS';
```

TABLE_NAME	COMMENTS
USER_OBJECTS	Objects owned by the user

从 DICTIONARY 开始，这个数据对象包含了数据字典的表名和说明

15.1.5. USER_OBJECTS 和 ALL_OBJECTS

USER_OBJECTS:

- 通过查询 USER_OBJECTS 可以确定当前用户所有创建的对象
- 可以获得如下信息:
    - Date created
    - Date of last modification
    - Status (valid or invalid)

ALL_OBJECTS:

- 可以通过查询 ALL_OBJECTS 来确定当前用户能访问的数据对象

15.1.6. USER_OBJECTS 视图

```
SELECT object_name, object_type, created, status
FROM      user_objects
ORDER BY object_type;
```

OBJECT_NAME	OBJECT_TYPE	CREATED	STATUS
REG_ID_PK	INDEX	10-DEC-03	VALID
DEPARTMENTS_SEQ	SEQUENCE	10-DEC-03	VALID
REGIONS	TABLE	10-DEC-03	VALID
LOCATIONS	TABLE	10-DEC-03	VALID
DEPARTMENTS	TABLE	10-DEC-03	VALID
JOB_HISTORY	TABLE	10-DEC-03	VALID
JOB_GRADES	TABLE	10-DEC-03	VALID
EMPLOYEES	TABLE	10-DEC-03	VALID
JOBS	TABLE	10-DEC-03	VALID
COUNTRIES	TABLE	10-DEC-03	VALID
EMP_DETAILS_VIEW	VIEW	10-DEC-03	VALID

15.1.7. 表与列的信息

DESCRIBE user_tables

Name	Null?	Type
TABLE_NAME	NOT NULL	VARCHAR2(30)
TABLESPACE_NAME		VARCHAR2(30)
CLUSTER_NAME		VARCHAR2(30)
IOT_NAME		VARCHAR2(30)

```
SELECT table_name
FROM      user_tables;
```

TABLE_NAME
JOB_GRADES
REGIONS
COUNTRIES
LOCATIONS
DEPARTMENTS

15.1.8. 列的信息

DESCRIBE user_tab_columns

Name	Null?	Type
TABLE_NAME	NOT NULL	VARCHAR2(30)
COLUMN_NAME	NOT NULL	VARCHAR2(30)
DATA_TYPE		VARCHAR2(106)
DATA_TYPE_MOD		VARCHAR2(3)
DATA_TYPE_OWNER		VARCHAR2(30)
DATA_LENGTH	NOT NULL	NUMBER
DATA_PRECISION		NUMBER
DATA_SCALE		NUMBER
NULLABLE		VARCHAR2(1)
COLUMN_ID		NUMBER
DEFAULT_LENGTH		NUMBER
DATA_DEFAULT		LONG

```
SELECT column_name, data_type, data_length,
          data_precision, data_scale, nullable
FROM     user_tab_columns
WHERE    table_name = 'EMPLOYEES';
```

COLUMN_NAME	DATA_TYPE	DATA_LENGTH	DATA_PRECISION	DATA_SCALE	NUL
EMPLOYEE_ID	NUMBER	22	6	0	N
FIRST_NAME	VARCHAR2	20			Y
LAST_NAME	VARCHAR2	25			N
EMAIL	VARCHAR2	25			N
PHONE_NUMBER	VARCHAR2	20			Y
HIRE_DATE	DATE	7			N
JOB_ID	VARCHAR2	10			N
SALARY	NUMBER	22	8	2	Y
COMMISSION_PCT	NUMBER	22	2	2	Y
MANAGER_ID	NUMBER	22	6	0	Y
DEPARTMENT_ID	NUMBER	22	4	0	Y

15.1.9. 约束

- USER_CONSTRAINTS:当前用户表上的约束
- USER_CONS_COLUMNS 当前用户创建的列约束

DESCRIBE user_constraints

Name	Null?	Type
OWNER	NOT NULL	VARCHAR2(30)
CONSTRAINT_NAME	NOT NULL	VARCHAR2(30)
CONSTRAINT_TYPE		VARCHAR2(1)
TABLE_NAME	NOT NULL	VARCHAR2(30)
SEARCH_CONDITION		LONG
R_OWNER		VARCHAR2(30)
R_CONSTRAINT_NAME		VARCHAR2(30)
DELETE_RULE		VARCHAR2(9)
STATUS		VARCHAR2(8)

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

CONSTRAINT_NAME	CON	SEARCH_CONDITION	R_CONSTRAINT_NAME	DELETE_RULE	STATUS
EMP_LAST_NAME_NN	C	"LAST_NAME" IS NOT NULL			ENABLED
EMP_EMAIL_NN	C	"EMAIL" IS NOT NULL			ENABLED
EMP_HIRE_DATE_NN	C	"HIRE_DATE" IS NOT NULL			ENABLED
EMP_JOB_NN	C	"JOB_ID" IS NOT NULL			ENABLED
EMP_SALARY_MIN	C	salary > 0			ENABLED
EMP_EMAIL_UK	U				ENABLED
EMP_EMP_ID_PK	P				ENABLED
EMP_DEPT_FK	R		DEPT_ID_PK	NO ACTION	ENABLED
EMP_JOB_FK	R		JOB_ID_PK	NO ACTION	ENABLED
EMP_MANAGER_FK	R		EMP_EMP_ID_PK	NO ACTION	ENABLED

DESCRIBE user_cons_columns

Name	Null?	Type
OWNER	NOT NULL	VARCHAR2(30)
CONSTRAINT_NAME	NOT NULL	VARCHAR2(30)
TABLE_NAME	NOT NULL	VARCHAR2(30)
COLUMN_NAME		VARCHAR2(4000)
POSITION		NUMBER

SELECT constraint_name, column_name
FROM user_cons_columns
WHERE table_name = 'EMPLOYEES';

CONSTRAINT_NAME	COLUMN_NAME
EMP_EMAIL_UK	EMAIL
EMP_SALARY_MIN	SALARY
EMP_JOB_NN	JOB_ID
EMP_HIRE_DATE_NN	HIRE_DATE

15.1.10. 视图

DESCRIBE user_views

Name	Null?	Type
VIEW_NAME	NOT NULL	VARCHAR2(30)
TEXT_LENGTH		NUMBER
TEXT		LONG

SELECT DISTINCT view_name FROM user_views;

VIEW_NAME
EMP_DETAILS_VIEW

SELECT text FROM user_views
WHERE view_name = 'EMP_DETAILS_VIEW';

TEXT
SELECT e.employee_id, e.job_id, e.manager_id, e.department_id, d.locat ion_id, l.country_id, e.first_name, e.last_name, e.salary, e.commissio n_pct, d.department_name, j.job_title, l.city, l.state_province, c.cou ntry_name, r.region_name FROM employees e, departments d, jobs j, loca tions l, countries c, regions r WHERE e.department_id = d.department_id AN D d.location_id = l.location_id AND l.country_id = c.country_id AND c.region _id = r.region_id AND j.job_id = e.job_id WITH READ ONLY

15.1.11. 序列

DESCRIBE user_sequences

Name	Null?	Type
SEQUENCE_NAME	NOT NULL	VARCHAR2(30)
MIN_VALUE		NUMBER
MAX_VALUE		NUMBER
INCREMENT_BY	NOT NULL	NUMBER
CYCLE_FLAG		VARCHAR2(1)
ORDER_FLAG		VARCHAR2(1)
CACHE_SIZE	NOT NULL	NUMBER
LAST_NUMBER	NOT NULL	NUMBER

♦ 通过 USER_SEQUENCES 查询序列信息

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

SEQUENCE_NAME	MIN_VALUE	MAX_VALUE	INCREMENT_BY	LAST_NUMBER
LOCATIONS_SEQ	1	9900	100	3300
DEPARTMENTS_SEQ	1	9990	10	280
EMPLOYEES_SEQ	1	1.0000E+27	1	207

- LAST_NUMBER 表示当没有使用 NOCAHCE 时，下一个可用的值

15.1.12. 同义词

DESCRIBE user_synonyms

Name	Null?	Type
SYNONYM_NAME	NOT NULL	VARCHAR2(30)
TABLE_OWNER		VARCHAR2(30)
TABLE_NAME	NOT NULL	VARCHAR2(30)
DB_LINK		VARCHAR2(128)

```
SELECT *
FROM   user_synonyms;
```

SYNONYM_NAME	TABLE_OWNER	TABLE_NAME	DB_LINK
EMP	ORA1	EMPLOYEES	

15.1.13. 给表添加注释

- 使用 COMMENT 语句给表或者列，添加注释:

```
COMMENT ON TABLE employees
IS 'Employee Information';
Comment created.
```

- 注释相关的视图:
    - ALL_COL_COMMENTS
    - USER_COL_COMMENTS
    - ALL_TAB_COMMENTS
    - USER_TAB_COMMENTS
- 查询表的注释
    - select * from user_tab_comments where table_name= ‘???';

15.1.14. 权限相关的数据字典

- DBA_TAB_PRIVS:
    - 包含数据库所有的对象权限信息
- DBA_SYS_PRIVS:
    - 包含数据库中所有的系统权限信息
- SESSION_PRIVS:
    - 包含当前用户可以使用的权限信息（Session 就是当前用户的会话）

15.2. 小结

- **DICTIONARY**
- USER_OBJECTS
- USER_TABLES
- USER_TAB_COLUMNS
- USER_CONSTRAINTS
- USER_CONS_COLUMNS
- USER_VIEWS

- USER_SEQUENCES
- USER_TAB_SYNONYMS
- 表的注释

16.Oracle 数据库管理

16.1. 闪回删除的表

16.1.1. 说明

- 闪回删除，实际上从系统的回收站中将已删除的对象，恢复到删除之前的状态。
- 系统的回收站只对普通用户有作用。

- 回收站是所有被删除对象及其相依对象的逻辑存储容器，例如当一个表被删除(drop)时，该表及其相依对象并不会马上被数据库彻底删除，而是被保存到回收站中。

- 回收站将用户执行的 drop 操作记录在一个系统表中，也就是将被删除的对象写到一个数据字典中。如果确定不再需要该对象，可以使用 purge 命令对回收站进行清空。

- 被删除的对象的名字可能是相同的，例如用户创建了一个 test 表，使用 drop 命令删除该表后，又创建了一个 test 表，这时，如果再次删除该表就会导致向回收站中添加了两个相同的表。

16.1.2. 闪回删除(flashback drop)的语法

```
FLASHBACK TABLE [schema.]<table_name>
TO
{[BEFORE DROP [RENAME TO table]]
 [ENABLE|DISABLE]TRIGGERS}
```

- schema：模式名，一般为用户名。
- ENABLE TRIGGERS：表示触发器恢复以后为 enable 状态，而默认为 disable 状态。
- TO BEFORE DROP：表示恢复到删除之前。
- RENAME TO table：表示更换表名。

16.1.3. 回收站中对象的命名规则

- 为了确保添加到回收站中的对象的名称都是唯一的，系统会对这些保存到回收站中的对象进行重命名，重命名的格式如下：

  BIN$globalUID$version

  - 其中： BIN 表示 RECYCLEBIN；globalUID 是一个全局唯一的、24 个字非长的对象，该标识与原对象名没有任何关系；version 指数据库分配的版本号。

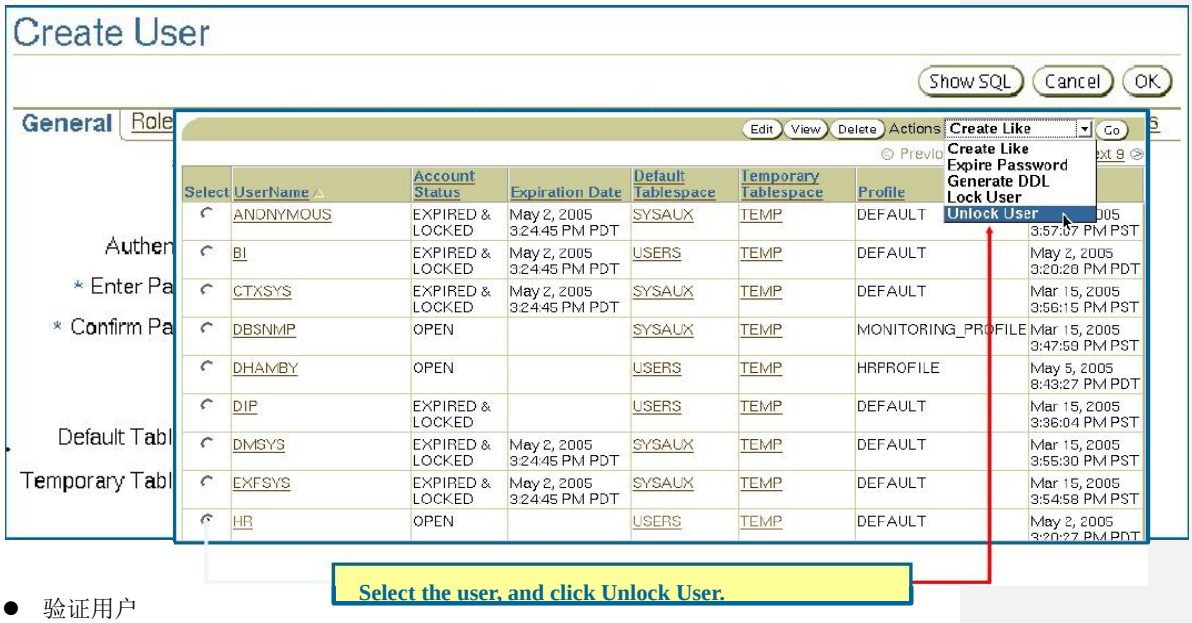
16.1.4. Oracle 回收站举例

16.2. 管理用户与权限

16.2.1. 预定义帐户: SYS 和 SYSTEM

- SYS 帐户(数据库拥有者):
  - 拥有 DBA 权限
  - 拥有 ADMIN OPTION 的所有权限
  - 拥有 startup, shutdown, 以及若干维护命令
  - 拥有数据字典
- SYSTEM 帐户拥有 DBA 权限.
- 这些帐户并非用于常规操作

16.2.2. 创建用户



- 验证用户
  - 密码验证方式(用户名/密码)
  - 外部验证方式(主机认证)
  - 全局验证方式(其他方式：生物认证方式、token 方式)

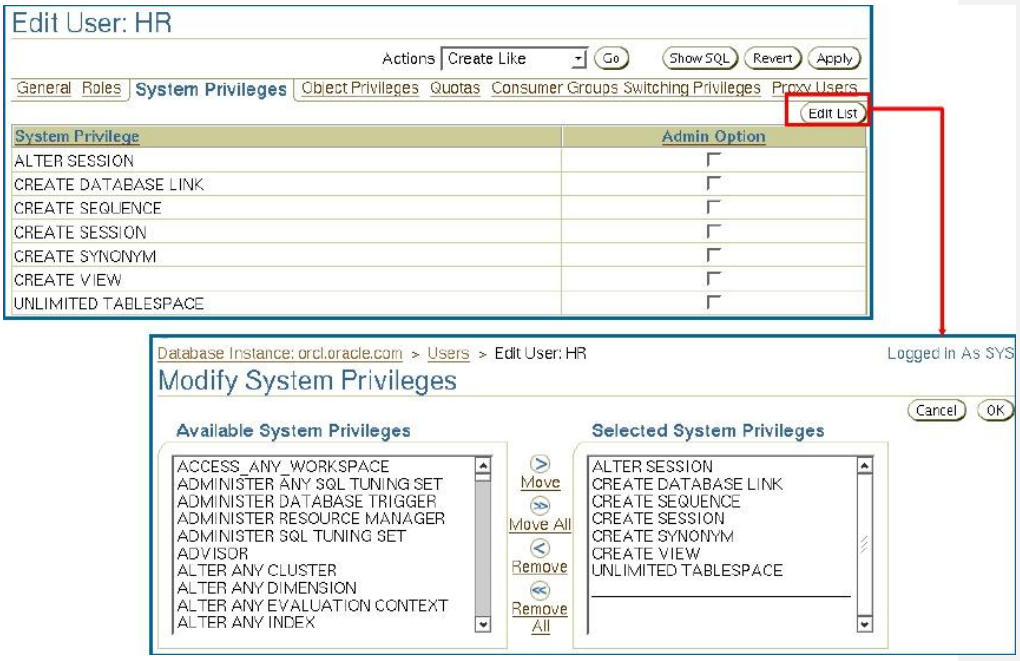
- 管理员验证
  - 操作系统安全:
    - DBA 必须拥有创建和删除文件的操作系统权限
    - 普通数据库用户不应具有拥有创建和删除文件的操作系统权限
  - 管理员安全:
    - SYSBA 和 SYSOPER 通过密码文件或操作系统实现连接审定.
    - 密码文件使用名称鉴别 DBA 用户.
    - OS 验证并不记录特定用户.
    - 对于 SYSDBA 和 SYSOPER 来说 OS 验证优先于密码文件认证.

- 解锁用户帐户和重置密码

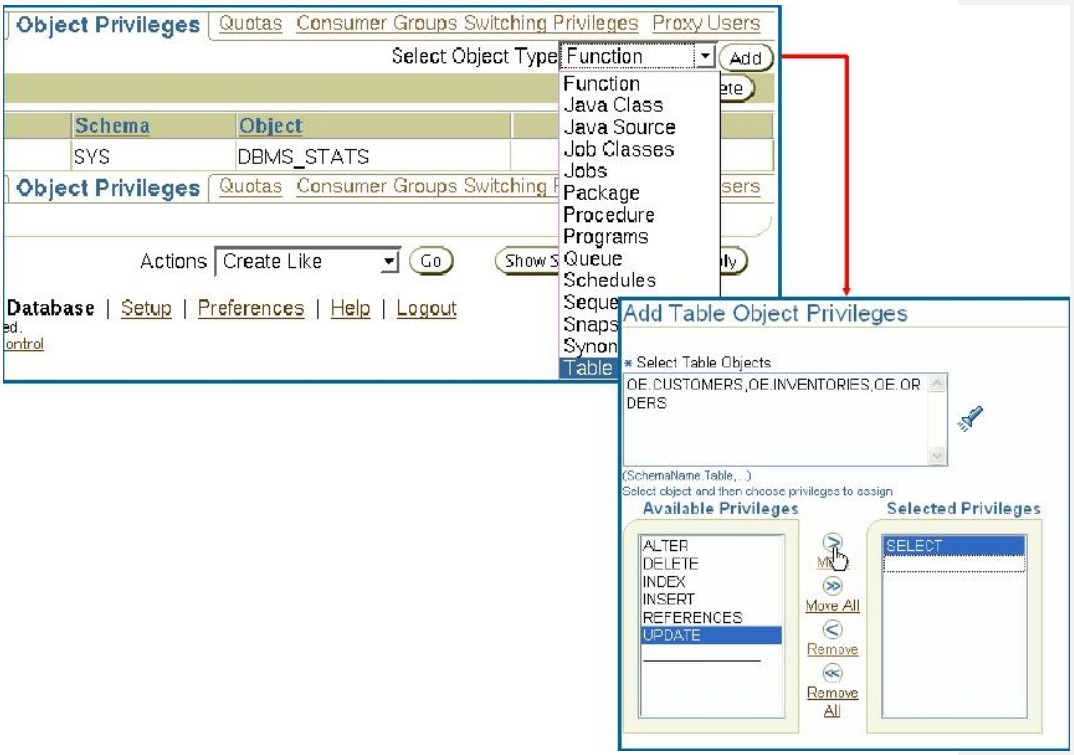
16.2.3. 权限

- 用户权限有两种:
  - System: 允许用户执行对于数据库的特定行为，例如：创建表、创建用户等
  - Object: 允许用户访问和操作一个特定的对象，例如：对其他方案下的表的查询

1.1.1.1. 系统权限

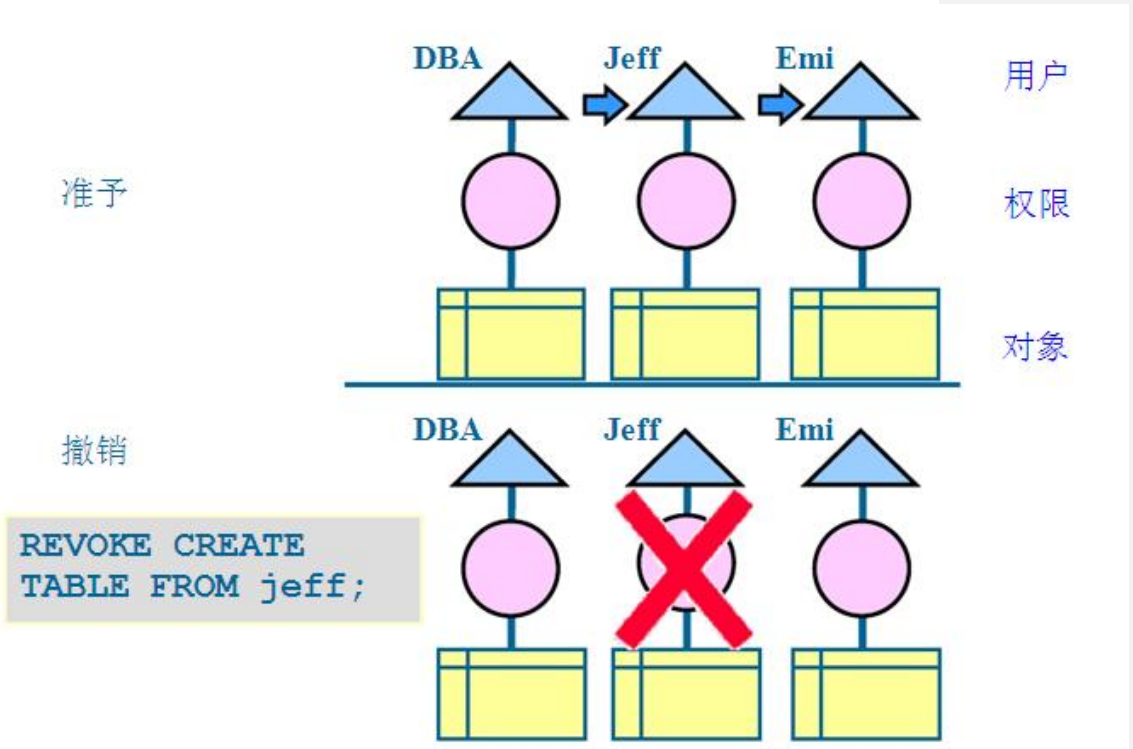


1.1.1.2. 对象权限

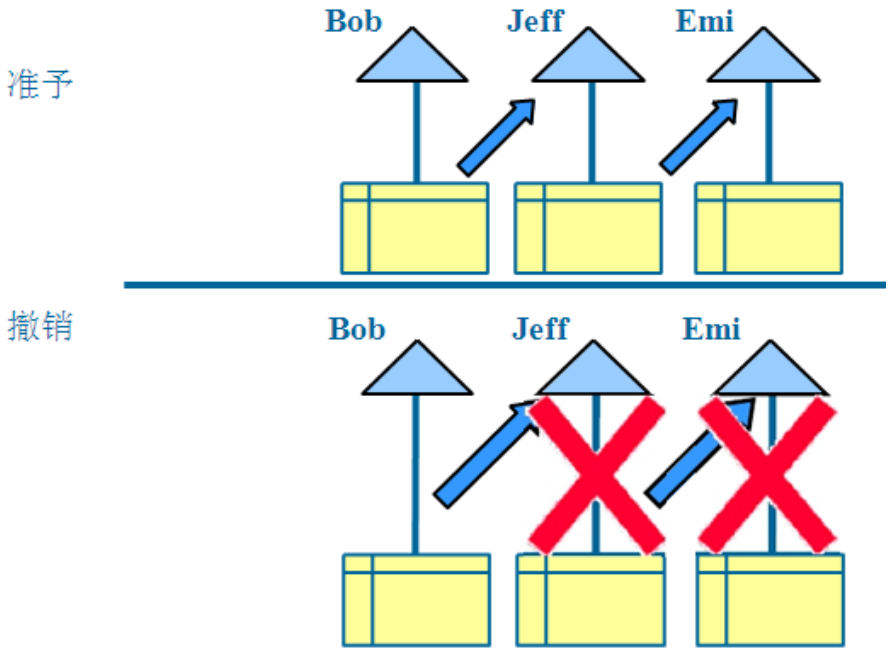


- 赋予对象权限:
  - 选择对象类型.
  - 选择对象.
  - 选择权限.

1.1.1.3. 使用 ADMIN OPTION 撤销系统权限



1.1.1.4. 使用 GRANT OPTION 撤销对象权限



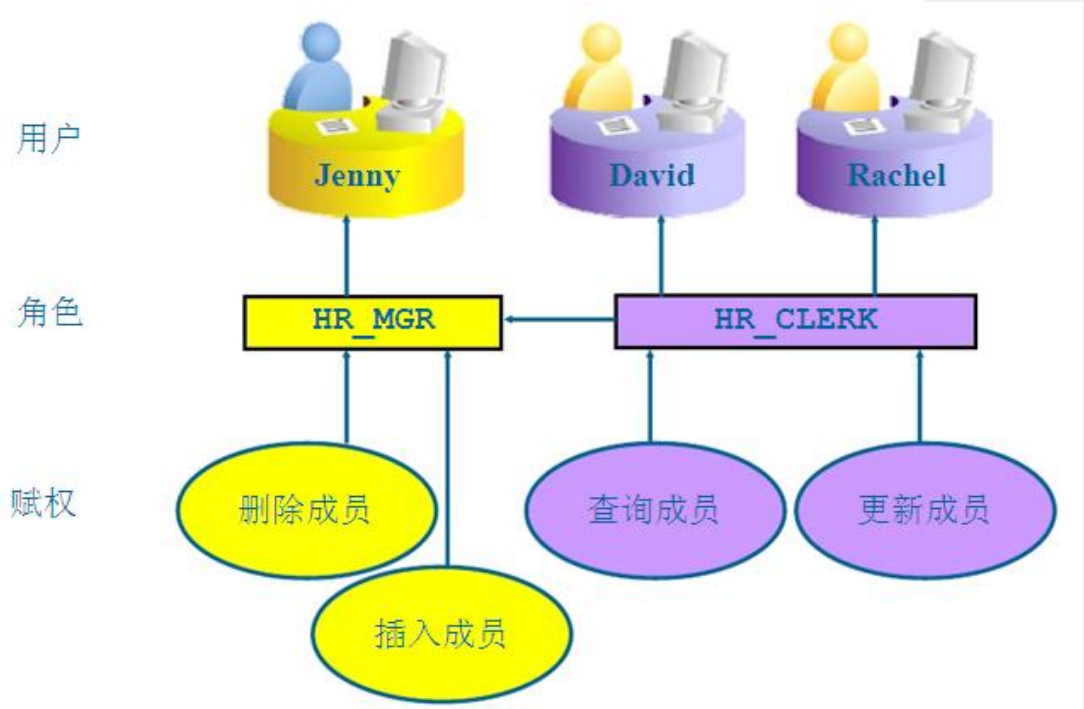
注意：grant option 只对 DML 语句起作用，例如 select

1.1.1.5. 角色

- 角色的优点

.1. 易于权限管理
.2. 动态权限管理
.3. 选择有效的权限

- 关联权限到角色，关联角色到用户



- 预定义角色和权限

角色	权限
CONNECT	CREATE SESSION
RESOURCE	CREATE CLUSTER, CREATE INDEXTYPE, CREATE OPERATOR, CREATE PROCEDURE, CREATE SEQUENCE, CREATE TABLE, CREATE TRIGGER, CREATE TYPE
SCHEDULER_ADMIN	CREATE ANY JOB, CREATE EXTERNAL JOB, CREATE JOB, EXECUTE ANY CLASS, EXECUTE ANY PROGRAM, MANAGE SCHEDULER
DBA	Most system privileges, several other roles. Do not grant to nonadministrators.
SELECT_CATALOG_ROLE	No system privileges, but HS_ADMIN_ROLE and over 1,700 object privileges on the data dictionary

- 创建角色

- Select Administration > Schema > Users & Privileges > Roles.

- 关联角色到用户

16.3. 数据的导入与导出（备份与恢复）

16.3.1. 说明

使用的是命令行的命令，不是 sqlplus 命令：

```
Imp.exe
Exp.exe
```

16.3.2. 数据导出：exp 命令

- 格式：

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

- 参数说明：
  - 参考帮助文档。

- exp 命令举例：

  - 表方式：将指定表的数据导出

```
exp scott/tiger@192.168.1.104:1521/orcl file=d:/temp/1.dmp log=d:/temp/log.log tables=emp,dept
```

  - 用户方式：将指定用户的所有对象及数据导出

```
exp scott/tiger@192.168.1.104:1521/orcl file=d:/temp/2.dmp log=d:/temp/log.log
```

16.3.3. 数据导入：imp 命令

- 格式：
  - IMP KEYWORD=value 或 KEYWORD=(value1,value2,...,valueN)

- 参数说明：
  - 参考帮助文档。

- Imp 命令举例：

  - 导入一张或几张表

```
imp itcast/password@192.168.1.104:1521/orcl file=d:/temp/1.dmp log=d:/temp/1.log tables=emp,dept fromuser=scott touser=itcast commit=y ignore
```

  - 导入用户下的表

```
imp itcast/password@192.168.1.104:1521/orcl file=d:/temp/user.dmp log=d:/temp/log.log fromuser=scott touser=itcast commit=y ignore=y
```

16.3.4. Exp 和 imp 的提示模式

- 导出：

```
emp scott/tiger@192.168.1.104:1521/orcl
```

- 导入：

```
imp scott/tiger@192.168.1.104:1521/orcl
```

```
declare
   --定义两个游标保存结果
   cursor c1 is select distinct deptno from dept;
   cursor c2(pdno number) is select sal from emp where deptno=pdno;

   --定义三个变量用于保存每个部门三个工资段的人数
   count1 NUMBER;
   count2 number;
   count3 number;

   --记录c1游标中的部门号
   pdeptno dept.deptno% TYPE;
   --记录c2游标中的薪水值
   psal emp.sal% TYPE;
begin
  open c1;--打开c1 获得所以部门号
    loop
      fetch c1 into pdeptno;--取一个部门号
      exit when c1%notfound;
      --计数器清零
      count1 := 0;
      count2 := 0;
      count3 := 0;
      --得到该部门的所有员工
      open c2(pdeptno);
      loop
        fetch c2 into psal; --得到该员工的工资
        exit when c2%notfound;
        if psal <=3000 then count1 := count1 + 1;
        elsif psal > 3000 and psal <=6000 then count2 := count2 + 1;
        else  count3 := count3 + 1;
        end if;
      end loop;
    close c2;

    --保存该部门的统计结果
    insert into salcount values(pdeptno,count1,count2,count3);
    commit;
  end loop;
 close c1;
end;
/
```