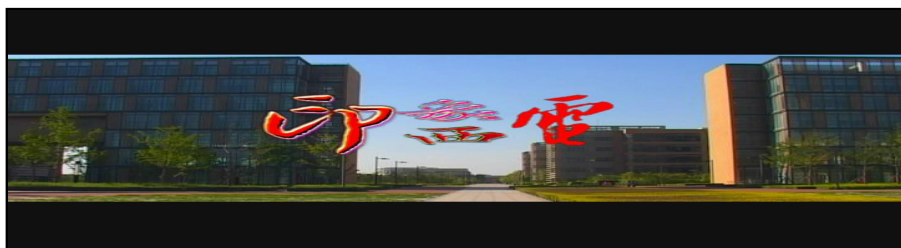

Oracle 教程

2009.5

Louis



目录

引言.....	3
第一章 数据库基础.....	6
第二章 Oracle 入门.....	6
第三章 查询基础.....	11
第四章 单行函数.....	14
第五章 分组函数.....	19
第六章 多表查询.....	21
第七章 子查询.....	24
第八章 高级查询.....	40
第九章 数据字典.....	43
第十章 Oracle 数据类型.....	44
第十一章 Oracle 体系结构(DBA).....	45
第十二章 DDL(改变表结构).....	46
第十三章 DML(改变数据结构).....	48
第十四章 约束.....	49
第十五章 视图.....	55
第十六章 索引.....	56
第十七章 序列、同义词.....	65
第十八章 PL SQL.....	67
第十九章 游标、函数.....	79
第二十章 存储过程.....	86
第二十一章 触发器.....	90
第二十二章 事务（数据库系统概论）.....	99
第二十三章 用户管理.....	100
第二十四章 备份 恢复 SQLLoader.....	104
第二十五章 数据库设计范式.....	106
第二十六章 数据库设计工具.....	107
第二十七章 对象关系数据库系统.....	112
第二十八章 其他数据库.....	113

引言

SUN 2008 初 10 亿美元收购 MySQL

Oracle 2009 年 4 月 74 亿美元收购 SUN

Sun 与 Oracle 合并的未来

1, 如果云计算对企业来说变得越来越重要, 那么数据将是云计算的核心。而讲到数据, 也就意味着数据库。就如塔克商学院数字策略中心主任 M. Eric Johnson 所说的那样, Sun 已经清楚展现了一个真实的云计算环境。就那些将云计算停留在理论阶段的对手来说, 收购 Sun 对 Oracle 来说将获得竞争优势。

2, Johnson 主任还指出, Sun 在很多需求旺盛的方面“保持领先地位”。Sun 的技术可以帮助数据库提高性能, 尤其是对大型数据库。Oracle 和 Sun 的结合之后, 对任何一个信息官来说都具有吸引力。

3, Oracle 软件和 Java 的紧密结合, 使得 Oracle 的软件能跨平台使用。就如 IDC 研究副总裁 Jean Bozman 指出那样, 真正云计算的第一步是“将服务器和存储单元垂直化”, 可以适用于不同的硬件平台。凭借 Java, Oracle 可以拓展跨平台的控制力, 而这正是企业 IT 部门想要的。实际上, Oracle 的 Oracle Fusion Middleware 正是基于 Java。

4, 有意思的是, Oracle Fusion Middleware 扩大了对服务器的需求, 其中原因正是由于它是基于 Java。而现在, Oracle 通过购得 Sun 服务器资源, 可以进一步降低价格, 从而进一步提高 Oracle Fusion Middleware 的竞争力。

5, 由于 Oracle 大部分收入来自数据库软件许可, 因此有必要的话, 可以用便宜的服务器硬件作为促销的手段, 这样对那些硬件服务器竞争对手来说是不小的打击。

6, Sun 已经售出 160 万到 200 万台服务器, 还有更多的运行 Sun Solaris 操作系统的其它公司的服务器。这对 Oracle 捆绑销售自家产品也很有帮助。

7, Oracle 收购 Sun, 当然也包括开源数据库 MySQL。对于那些预算不多但又需要 Oracle 产品的用户, Oracle 可以用 MySQL 来满足这些用户的需求。这可以使 Oracle 进入中小企业业务, 更有机会在这方面超过 SAP。此外, 还提供了和微软 SQLServer 的竞争产品, 而不需要降低 Oracle 自有的产品的定位。

Oracle 和 Sun 的合作可能会影响 Oracle 和一些合作伙伴的关系，比如 Oracle 和惠普的关系可能就会出现。不过，不像 IBM，事实上，每个硬件公司都绕不开 Oracle，因为它是数据库的主力军。他们离不开 Oracle，因此，他们也只能选择忍气吞声。

而且，与 Sun 和 IBM 联合不同，Oracle 和 Sun 不存在硬件的冲突，那些 Sun 的服务器用户不必担心合并会导致放弃自己在使用的服务器产品线。

总而言之，Oracle 以相对较低的价格收购 Sun 是明智之举，将会使得合并后的 Oracle 成为各大公司数据中心的枢纽。

云计算

李开复（现任 Google 全球副总裁、中国区总裁）打了一个很形象的比喻：钱庄。最早人们只是把钱放在枕头底下，后来有了钱庄，很安全，不过兑现起来比较麻烦。现在发展到银行可以到任何一个网点取钱，甚至通过 ATM，或者国外的渠道。就像用电不需要家家装备发电机，直接从电力公司购买一样。“云计算”带来的就是这样一种变革——由谷歌、IBM 这样的专业网络公司来搭建计算机存储、运算中心，用户通过一根网线借助浏览器就可以很方便的访问，把“云”做为资料存储以及应用服务的中心。

GOOGLE

（一）原理：

云计算(Cloud Computing)是分布式处理(Distributed Computing)、并行处理(Parallel Computing)和网格计算(Grid Computing)的发展，或者说是这些计算机科学概念的商业实现。

云计算的基本原理是，通过使计算分布在大量的分布式计算机上，而非本地计算机或远程服务器中，企业数据中心的运行将更与互联网相似。这使得企业能够将资源切换到需要的应用上，根据需求访问计算机和存储系统。

这可是一种革命性的举措，打个比方，这就好比是从古老的单台发电机模式转向了电厂集中供电的模式。它意味着计算能力也可以作为一种商品进行流通，就像煤气、水电一样，取用方便，费用低廉。最大的不同在于，它是通过互联网进行传输的。

云计算的蓝图已经呼之欲出：在未来，只需要一台笔记本或者一个手机，就可以通过网络服务来实现我们需要的一切，甚至包括超级计算这样的任务。从这个角度而言，最终用户才是云计算的真正拥有者。

云计算的应用包含这样的一种思想，把力量联合起来，给其中的每一个成员使用。

（二）云计算有哪些好处？

- 1、安全，云计算提供了最可靠、最安全的数据存储中心，用户不用再担心数据丢失、病毒入侵等麻烦。
- 2、方便，它对用户端的设备要求最低，使用起来很方便。
- 3、数据共享，它可以轻松实现不同设备间的数据与应用共享。
- 4、无限可能，它为我们使用网络提供了几乎无限多的可能。

（三）云计算最有利于中小企业？

云计算技术将使得中小企业的成本大大降低。如果说“云”给大型企业的 IT 部门带来了实惠，那么对于中小型企业而言，它可算得上是上天的恩赐了。过去，小公司人力资源不足，IT 预算吃紧，那种动辄数百万美元的 IT 设备所带来的生产力对它们而言真是如梦一般遥远，而如今，“云”为它们送来了大企业级的技术，并且先期成本极低，升级也很方便。

这一新兴趋势的重要性毋庸置疑，不过，它还仅仅是一系列变革的起步阶段而已。云计算不但抹平了企业规模所导致的优劣差距，而且极有可能让优劣之势易主。简单地说，当今世上最强大最具革新意义的技术已不再为大型企业所独有。“云”让每个普通人都能以极低的成本接触到顶尖的 IT 技术。

（四）“云”时代

目前，PC 依然是我们日常工作生活中的核心工具——我们用 PC 处理文档、存储资料，通过电子邮件或 U 盘与他人分享信息。如果 PC 硬盘坏了，我们会因为资料丢失而束手无策。

而在“云计算”时代，“云”会替我们做存储和计算的工作。“云”就是计算机群，每一群包括了几十万台、甚至上百万台计算机。“云”的好处还在于，其中的计算机可以随时更新，保证“云”长生不老。Google 就有好几个这样的“云”，其他 IT 巨头，如微软、雅虎、亚马逊（Amazon）也有或正在建设这样的“云”。

届时，我们只需要一台能上网的电脑，不需关心存储或计算发生在哪朵“云”上，但一旦有需要，我们可以在任何地点用任何设备，如电脑、手机等，快速地计算和找到这些资料。我们再也不用担心资料丢失。

第一章 数据库基础

1.1、数据库基本概念

数据库 (Database, DB)

数据库管理系统 (Database Management System, DBMS)

数据库管理员 (Database Administrator, DBA)

数据库系统 (Database System, DBS)

1.2、关系型数据库

Q:目前都有哪些主流的关系型数据库

A:Oracle Oracle、IBM DB2、MS SQL /Server、SyBase SyBase、IBM Informix、MySQL、Access

Q:XML,TXT 可以作为数据库吗?

1.3、E-R 模型 (Entry-Relation)

E-R 模型三要素: 实体、关系、属性

实体间联系 (1:1) (1: n) (n:m)

Q:学生与课程什么关系?

第二章 Oracle 入门

2.1、Oracle 概述

甲骨文, 四大创始人

Oracle 传奇人物



左起 Ed Oates、Bruce Scott、
Bob Miner、Larry Ellison



Larry Ellison 一生最大的目标，“财富榜超过 Bill Gate”

Bruce Scott 已离开 Oracle,创建了一套新的数据库 PointBase

2.2、Oracle 特点

全球化、跨平台的数据库

支持多用户、高性能的事务处理

强大的安全性控制和完整性控制

支持分布式数据库和分布处理

2.3、Oracle 版本

Oracle8i: I internet 表示 Oracle 公司要开始正式进入互联网

Oracle9i:与 Oracle8i 相关, 性能方面更佳, 管理更人性化

Oracle10g: g(grid)网格技术

Oracle11g: g(grid)网格技术

Q: 何为网格技术?

2009 年 1 月

淘宝网决定采用 Oracle 网络计算(Grid Computing)架构来,采用 Oracle 数据库和 Oracle 真正应用集群来重新打造并强化其基础架构和数据仓库环境

2.4、安装 Oracle 数据库

注意:

- 1、安装的时候,一定要关掉防火墙。否则可能造成安装不成功

- 2、全局数据库名 SID,类似于 MYSQL 中常用的 localhost.
- 3、字符集一定要选择正确。一旦选错，除非更改成该字符集的父亲。否则只能重装
- 4、安装完主要的用户为：
 - a) 普通用户：Scott/tiger(练习常用)
 - b) 普通管理员：System/system
 - c) 超级管理员：Sys/sys
- 5、安装完后的服务配置 (运行中输入：services.msc)

 OracleMTSRecoveryService	已禁用	本地系统
 OracleOraHome92Agent	已禁用	本地系统
 OracleOraHome92ClientCache	已禁用	本地系统
 OracleOraHome92HTTPServer	已禁用	本地系统
 OracleOraHome92PagingServer	已禁用	本地系统
 OracleOraHome92SNMPPeerEncapsulator	已禁用	本地系统
 OracleOraHome92SNMPPeerMasterAgent	已禁用	本地系统
 OracleOraHome92TNSListener	已启动 手动	本地系统
 OracleServiceCARE	已启动 手动	本地系统

开启该服务，会占用 tomcat 的 8080 端口。

查看端口号：tasklist|findstr "8080"

如果装完 Oracle 后，又改了机器名可能会导致 Listener 服务无法启动，解决方式：

修改 C:\oracle\ora92\network\admin\listener.ora 下的 HOST

2.5、卸载 Oracle 数据库

■ 卸载步骤：

1. 停止Oracle所有服务
2. 运行Oracle Universal Installer卸载Oracle
3. 修改注册表，删除Oracle相关信息
 - Oracle软件有关键-值
HKEY_LOCAL_MACHINE\SOFTWARE\Oracle
 - Oracle服务
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services
 - Oracle事件日志
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Appl
ication
4. 删除Oracle系统目录 C:\program files\oralce
5. 删除Oracle环境变量
6. 删除程序菜单项中的Oracle菜单
7. (重启系统后) 删除Oracle工作主目录 D:\oralce

2.6、Oracle 目录（Admin,oracle92,oradata）



Q:这三个文件夹各存放什么文件？

2.7、Oracle 客户端工具

2.7.1、SQLPLUS

2.7.2、SQLPLUSW

2.7.3、Sql Plus WorkSheet

2.7.4、iSqlplus (HTTP)

<http://localhost:7778/isqlplus>

2.7.5、PL/SQL Developer

7.14 以上版本，带有自动提示功能

2.7.6、Object Browser

国外用的比较多（日本）

2.8、Scott 案例下表分析

2.8.1、雇员表：EMP

雇员表 (EMP)			
No.	字段	类型	描述
1	EMPNO	NUMBER(4)	表示雇员编号, 是唯一编号
2	ENAME	VARCHAR2(10)	表示雇员姓名
3	JOB	VARCHAR2(9)	表示工作职位
4	MGR	NUMBER(4)	表示一个雇员的领导编号
5	HIREDATE	DATE	表示雇佣日期
6	SAL	NUMBER(7,2)	表示月薪, 工资
7	COMM	NUMBER(7,2)	表示奖金, 或者称为佣金
8	DEPTNO	NUMBER(2)	部门编号

2.8.2、部门表：Dept

部门表 (dept)			
No.	字段	类型	描述
1	DEPTNO	NUMBER(2)	部门编号, 是唯一编号
2	DNAME	VARCHAR2(14)	部门名称
3	LOC	VARCHAR2(13)	部门位置

2.8.3、工资等级表：Salgrade

工资等级表 (SALGRADE)			
No.	字段	类型	描述
1	GRADE	NUMBER	等级名称
2	LOSAL	NUMBER	此等级的最低工资
3	HISAL	NUMBER	此等级的最高工资

2.8.4、奖金表: Bonus

奖金表 (BONUS)			
No.	字段	类型	描述
1	ENAME	VARCHAR2(10)	雇员姓名
2	JOB	VARCHAR2(9)	雇员工作
3	SAL	NUMBER	雇员工资
4	COMM	NUMBER	雇员奖金 (佣金)

第三章 查询基础

3.1、入门语句

普通用户连接: Conn scott/tiger

超级管理员连接: Conn "sys/sys as sysdba"

Disconnect; 断开连接

Save c:\1.txt 把 SQL 存到文件

Ed c:\1.txt 编辑 SQL 语句

@ c:\1.txt 运行 SQL 语句

Desc emp; 描述 Emp 结构

Select * from tab; 查看该用户下的所有对象

Show user; 显示当前用户

如果在 sys 用户下: 查询 Select * from emp; 会报错, 原因: emp 是属于 scott, 所以此时必须使用: select * from scott.emp;

/ 运行上一条语句

3.2、SQL 简介

SQL 全名是结构化查询语言 (Structured Query Language), 是用于数据库中的标准数据查询语言, IBM 公司最早使用在其开发的数据库系统中。1986 年 10 月, 美国 ANSI 对 SQL 进行规范后, 以此作为关系式数据库管理系统的标准语言 (ANSI X3. 135-1986), 1987 年得到国际标准组织的支持下成为国际标准。不过各种通行的数据库系统在其实践过程中都对 SQL 规范作了某些编改和扩充。所以, 实际上不同数据库系统之间的 SQL 语言不能完全相互通用

DML 语句 (数据操作语言) Insert、Update、Delete、Merge

DDL 语句 (数据定义语言) Create、Alter、Drop、Truncate

DCL 语句 (数据控制语言) Grant、Revoke

事务控制语句 Commit、Rollback、Savepoint

3.3、简单的 Select 语句

3.4、使用算术表达式 + - / *

3.5、连接运算符 ||

3.6、使用字段别名 as

3.7、空值 is null

3.8、去除重复行 distinct

3.9、查询结果排序 order by asc(desc)

3.10、比较运算符 > < (!= or <>) between and

3.11、in 操作 not in

3.12、模糊查询 like

% 表示零或多个字符

_ 表示一个字符

对于特殊符号可使用 ESCAPE 标识符来查找

```
select * from emp where ename like '%*_%' escape '*'
```

上面的 escape 表示*后面的那个符号不当成特殊字符处理，就是查找普通的_符号

3.13、逻辑运算符 or and not

3.14、练习

选择在部门 30 中员工的所有信息

```
Select * from emp where deptno=30;
```

列出职位为 (MANAGER) 的员工的编号, 姓名

```
Select empno,ename from emp where job = 'Manager';
```

找出奖金高于工资的员工

```
Select * from emp where comm>sal;
```

找出每个员工奖金和工资的总和

```
Select sal+comm,ename from emp;
```

找出部门 10 中的经理(MANAGER)和部门 20 中的普通员工(CLERK)

```
Select * from emp where (deptno=10 and job='MANAGER') or (deptno=20 and job='CLERK');
```

找出部门 10 中既不是经理也不是普通员工, 而且工资大于等于 2000 的员工

```
Select * from emp where deptno=10 and job not in('MANAGER','CLERK') and sal>=2000;
```

找出有奖金的员工的的不同工作

```
Select distinct job from emp where comm is not null and comm>0
```

找出没有奖金或者奖金低于 500 的员工

```
Select * from emp where comm<500 or comm is null;
```

显示雇员姓名, 根据其服务年限, 将最老的雇员排在最前面

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

第四章 单行函数

4.1、字符函数

Upper

```
SELECT Upper ('abcde') FROM dual ;  
SELECT * FROM emp WHERE ename=UPPER('smith') ;
```

Lower

```
SELECT lower('ABCDE') FROM dual ;
```

Initcap

```
Select initcap(ename) from emp;
```

Concat

```
Select concat('a','b') from dual;  
Select 'a' || 'b' from dual;
```

Substr

```
Select substr('abcde',length('abcde')-2) from dual;  
Select substr('abcde',-3,3) from dual;
```

Length

```
Select length(ename) from emp;
```

Replace

```
Select replace(ename,'a','A') from emp;
```

Instr

```
Select instr('Hello World','or') from dual; 8 indexof
```

Lpad

```
lpad('Smith',10,'*') 左侧填充 lpad()*****Smith
```

Rpad

```
rpadd('Smith',10,'*') 右侧填充 rpad()Smith*****
```

Trim

```
trim(' Mr Smith ') 过滤首尾空格 trim() Mr Smith
```

4.2、数值函数

Round

```
select round(412,-2) from dual;
select round(412.313,2) from dual;
```

Mod

Trunc

```
select trunc(412.13,-2) from dual;
```

4.3、日期函数

Months_between()

```
select months_between(sysdate,hiredate) from emp;
```

Add_months()

```
select add_months(sysdate,1) from dual;
```

Next_day()

```
select next_day(sysdate,'星期一') from dual;
```

Last_day

```
select last_day(sysdate) from dual;
```

4.4、转换函数

To_char

```
select to_char(sysdate,'yyyy') from dual;
select to_char(sysdate,'fmyyyy-mm-dd') from dual;
select to_char(sal,'L999,999,999') from emp;
select to_char(sysdate,'D') from dual; //返回星期
```

To_number

```
select to_number('13')+to_number('14') from dual;
```

To_date

```
Select to_date('20090210','yyyymmdd') from dual;
```

4.5、通用函数

NVL()函数

```
select nvl(comm,0) from emp;
```

NULLIF()函数

如果表达式 exp1 与 exp2 的值相等则返回 null，否则返回 exp1 的值

NVL2()函数

```
select empno, ename, sal, comm, nvl2(comm, sal+comm, sal) total from emp;
```


COALESCE()函数

依次考察各参数表达式，遇到非 null 值即停止并返回该值。

```
select empno, ename, sal, comm, coalesce(sal+comm, sal, 0)总收入 from emp;
```

CASE 表达式

```
select empno, ename, sal,  
case deptno  
  when 10 then '财务部'  
  when 20 then '研发部'  
  when 30 then '销售部'  
else '未知部门'  
end 部门  
from emp;
```

DECODE()函数

和 case 表达式类似，decode()函数也用于实现多路分支结构

```
select empno, ename, sal,  
decode(deptno, 10, '财务部',  
20, '研发部',  
30, '销售部',  
'未知部门')  
部门  
from emp;
```

decode 与 case 哪个更好用呢？

单行函数嵌套

```
select empno, lpad(initcap(trim(ename)),10,' ') name, job, sal from emp;
```

4.6、练习

找出每个月倒数第三天受雇的员工（如：2009-5-29）

```
select * from emp where last_day(hiredate)-2=hiredate;
```

找出 25 年前雇的员工

```
select * from emp where hiredate<=add_months(sysdate,-25*12);
```

所有员工名字前加上 Dear ,并且名字首字母大写

```
select 'Dear ' || initcap(ename) from emp;
```

找出姓名为 5 个字母的员工

```
select * from emp where length(ename)=5;
```

找出姓名中不带 R 这个字母的员工

```
select * from emp where ename not like '%R%';
```

显示所有员工的姓名的第一个字

```
select substr(ename,0,1) from emp;
```

显示所有员工, 按名字降序排列, 若相同, 则按工资升序排序

假设一个月为 30 天, 找出所有员工的日薪, 不计小数

找到 2 月份受雇的员工

```
select * from emp where to_char(hiredate,'fmmm')='2';
```

列出员工加入公司的天数(四舍五入)

分别用 case 和 decode 函数列出员工所在的部门, deptno=10 显示'部门 10',

deptno=20 显示'部门 20'

deptno=30 显示'部门 30'

deptno=40 显示'部门 40'

否则为'其他部门'

第五章 分组函数

5.1、COUNT

如果数据库表的没有数据，count(*)返回的不是 null，而是 0

5.2、Avg, max, min, sum

5.3、分组函数与空值

分组函数省略列中的空值

```
select avg(comm) from emp;
```

```
select sum(comm) from emp;
```

可使用 NVL()函数强制分组函数处理空值

```
select avg(nvl(comm, 0)) from emp;
```

5.4、GROUP BY 子句

出现在 SELECT 列表中的字段或者出现在 order by 后面的字段，如果不是包含在分组函数中，那么该字段必须同时在 GROUP BY 子句中出现。

包含在 GROUP BY 子句中的字段则不必须出现在 SELECT 列表中。

可使用 where 子句限定查询条件

可使用 Order by 子句指定排序方式

如果没有 GROUP BY 子句，SELECT 列表中不允许出现字段（单行函数）与分组函数混用的情况。

```
select empno, sal from emp; //合法
```

```
select avg(sal) from emp; //合法
```

```
select empno, initcap(ename), avg(sal) from emp; //非法
```

不允许在 WHERE 子句中使用分组函数。

```
select deptno, avg(sal)
```

```
from emp
```

```
where avg(sal) > 2000;
```

```
group by deptno;
```

5.5、HAVING 子句

```
select deptno, job, avg(sal)
```

```
from emp
```

```
where hiredate >= to_date('1981-05-01','yyyy-mm-dd')
group by deptno,job
having avg(sal) > 1200
order by deptno,job;
```

5.6、分组函数嵌套

```
select max(avg(sal))
from emp
group by deptno;
```

5.7、练习

分组统计各部门下工资>500 的员工的平均工资、

```
Select avg(sal) from emp where sal>500 group by deptno ;
```

统计各部门下平均工资大于 500 的部门

```
select deptno,avg(sal) from emp group by deptno having avg(sal)>500 ;
```

算出部门 30 中得到最多奖金的员工奖金

```
Select max(comm) from emp where deptno = 30 ;
```

算出部门 30 中得到最多奖金的员工姓名

```
select ename from emp where comm = (select max(comm) from emp where
deptno=30) ;
```

算出每个职位的员工数和最低工资

```
Select job,min(sal),count(*) from emp group by job;
```

算出每个部门, 每个职位的平均工资和平均奖金(平均值包括没有奖金), 如果平均奖金大于 300, 显示“奖金不错”, 如果平均奖金 100 到 300, 显示“奖金一般”, 如果平均奖金小于 100, 显示“基本没有奖金”, 按部门编号降序, 平均工资降序排列

```
Select avg(sal),avg(nvl(comm.,0)) case when avg(nvl(comm.,0))>300 then ‘奖金不错’  
when avg(nvl(comm.,0))<100 and avg(nvl(comm.,0))>300 then ‘奖金一般’ end 奖金状况  
from emp group by job order by job desc,avg(sal) desc;
```

列出员工表中每个部门的员工数, 和部门 no

```
Select count(*),deptno from emp group by deptno;
```

得到工资大于自己部门平均工资的员工信息

```
select * from emp e1,(select deptno,avg(sal) as avgsal from emp group by deptno) e2  
where e1.deptno=e2.deptno and e1.sal > e2.avgsal;
```

分组统计每个部门下, 每种职位的平均奖金 (也要算没奖金的人) 和总工资(包括奖金)

```
select deptno,job,avg(nvl(comm.,0)),sum(sal+nvl(comm.,0)) from emp group by deptno,job;
```

第六章 多表查询

6.1、笛卡尔集(Cross Join)

```
Select * from emp,dept;
```

6.2、等值连接(Equijoin)(Natural join..on)

```
select empno, ename, sal, emp.deptno, dname from emp, dept  
where emp.deptno = dept.deptno;
```

6.3、非等值连接(Non-Equijoin)

```
select ename,empno,grade from emp,salgrade where sal between losal and  
hisal;
```

6.4、自连接(Self join)

```
select e.empno,e.ename,m.empno,m.ename from emp e,emp m where e.mgr =  
m.empno;  
select e.empno,e.ename,m.empno,m.ename from emp e,emp m where m.mgr =  
e.empno;
```

6.5、左外联接 (Left Outer Join)

```
select s.sid,s.sname,s1.sid,s1.sname from student s,student1 s1 where  
s.sid=s1.sid(+);
```

```
select empno,ename,dname from emp left outer join dept on emp.deptno =  
dept.deptno;
```

6.6、右外联接 (Right Outer Join)

```
select s.sid,s.sname,s1.sid,s1.sname from student s,student1 s1 where  
s.sid(+)=s1.sid;
```

```
select empno,ename,dname from emp right outer join dept on emp.deptno =  
dept.deptno;
```

6.7、满外联接 (Full Outer Join)

```
select empno,ename,dname from emp full outer join dept on emp.deptno =  
dept.deptno;
```

1 2

2 4

3 6

4

5

内连接

2 2

4 4

满连接

1
2 2
3
4 4
5
6

左连接

1
2 2
3
4 4
5

右连接

2 2
4 4
6

6.8、集合操作

- UNION: 并集, 所有的内容都查询, 重复的显示一次
- UNION ALL: 并集, 所有的内容都显示, 包括重复的
- INTERSECT: 交集: 只显示重复的
- MINUS: 差集: 只显示对方没有的 (跟顺序是有关系的)

首先建立一张只包含 20 部门员工信息的表:

```
CREATE TABLE emp20 AS SELECT * FROM emp WHERE deptno=20 ;
```

1、验证 UNION 及 UNION ALL

```
UNION: SELECT * FROM emp UNION SELECT * FROM emp20 ;
```

使用此语句重复的内容不再显示了

```
UNION ALL: SELECT * FROM emp UNION ALL SELECT * FROM emp20 ;
```

重复的内容依然显示

2、验证 INTERSECT

```
SELECT * FROM emp INTERSECT SELECT * FROM emp20 ;
```

只显示了两个表中彼此重复的记录。

3、MINUS: 返回差异的记录

```
SELECT * FROM emp MINUS SELECT * FROM emp20 ;
```

只显示了两张表中的不同记录

满链接也可以用以下的方式来表示:

```
select t1.id,t2.id from table1 t1,table t2 where t1.id=t2.id(+)
union
select t1.id,t2.id from table1 t1,table t2 where t1.id(+)=t2.id
```

第七章 子查询

7.1、单行子查询

```
select * from emp
where sal > (select sal from emp where empno = 7566);
```

7.2、子查询空值/多值问题

如果子查询未返回任何行，则主查询也不会返回任何结果

```
(空值)select * from emp where sal > (select sal from emp where empno = 8888);
```

如果子查询返回单行结果，则为单行子查询，可以在主查询中对其使用相应的单行记录比较运算符

```
(正常)select * from emp where sal > (select sal from emp where empno = 7566);
```

如果子查询返回多行结果，则为多行子查询，此时不允许对其使用单行记录比较运算符

```
(多值)select * from emp where sal > (select avg(sal) from emp group by deptno);//非法
```

7.3、多行子查询

```
select * from emp where sal > any(select avg(sal) from emp group by deptno);
```

```
select * from emp where sal > all(select avg(sal) from emp group by deptno);
```

```
select * from emp where job in (select job from emp where ename = 'MARTIN' or ename = 'SMITH');
```

7.4、TopN 查询

```
select * from emp where rownum=1 or rownum=2;
select *
```



```
from (select * from emp order by sal desc)
where rownum <= 5;
```

Q:如何理解 (select * from emp where rownum<=5 order by sal desc)

7.5、分页查询

```
select * from (select rownum no,e.* from
(select * from emp order by sal desc) e where rownum<=5 ) where no>=3;

select *
from
(select rownum no,e.* from (select * from emp order by sal desc) e)
where
no>=3 and no<=5;
```

7.6、exists

EXISTS 的执行流程

```
select * from t1 where exists ( select null from t2 where y = x )
```

可以理解为:

```
for x in ( select * from t1 )
loop
    if ( exists ( select null from t2 where y = x.x )
    then
        OUTPUT THE RECORD
    end if
end loop
```

对于 in 和 exists 的性能区别:

如果子查询得出的结果集记录较少, 主查询中的表较大且又有索引时应该用 in,反之如果外层的主查询记录较少, 子查询中的表大, 又有索引时使用 exists。

其实我们区分 in 和 exists 主要是造成了驱动顺序的改变 (这是性能变化的关键), 如果是 exists, 那么以外层表为驱动表, 先被访问, 如果是 IN, 那么先执行子查询, 所以我们会以驱动表的快速返回为目标, 那么就会考虑到索引及结果集的关系了

另外 IN 是不对 NULL 进行处理

如:

```
select 1 from dual where null in (0,1,2,null)
```

为空

7.7、练习

列出员工表中每个部门的员工数，和部门 no

```
select deptno,count(*) from emp group by deptno;
```

列出员工表中每个部门的员工数（员工数必须大于 3），和部门名称

```
select d.* ,ed.cou from dept d, (select deptno,count(*) cou from emp group  
by deptno having count(*)>3) ed where d.deptno=ed.deptno;
```

找出工资比 jones 多的员工

```
select * from emp where sal>=(select sal from emp where  
lower(ename)='jones');
```

列出所有员工的姓名和其上级的姓名

```
select e1.ename as lower ,e2.ename as upper from emp e1,emp e2 where e1.mgr  
= e2.empno;  
select e1.ename as lower ,e2.ename as upper from emp e1,emp e2 where e1.mgr = e2.empno(+);
```

以职位分组，找出平均工资最高的两种职位

```
Select * from ( select avg(sal) from emp order by job desc ) where rownum <3;
```

查找出不在部门 20，且比部门 20 中任何一个人工资都高的员工姓名、部门名称

```
Select e.ename,d.dname from emp e,dept d where e.deptno!=20 and e.sal>(select max(sal) from  
emp where deptno=20) and e.deptno=d.deptno
```

得到平均工资大于 2000 的工作职种

```
select job from emp group by job having avg(sal) > 2000;
```

分部门得到工资大于 2000 的所有员工的平均工资，并且平均工资还要大于 2500

```
select deptno,avg(sal) from emp where sal>2000 group by deptno having avg(sal)>2500;
```

得到每个月工资总数最少的那个部门的部门编号，部门名称，部门位置

```
select * from dept
where
deptno = (
select e.deptno from
(select deptno,sum(sal) from emp group by deptno order by sum(sal))
e
where rownum=1
);
```

分部门得到平均工资等级为 2 级（等级表）的部门编号

```
select new.dno from salgrade sa,
(select deptno as dno,avg(sal) as avgsal from emp group by deptno) new
where sa.grade=4 and new.avgsal between sa.losal and sa.hisal;
```

查找出部门 10 和部门 20 中，工资最高第 3 名到工资第 5 名的员工的员工名字，部门名字，部门位置

```
select emp.ename,dept.dname,dept.loc
from
emp,
dept,
(select rownum no,new.* from
(select * from emp
where emp.deptno = 10 or deptno = 20 order by emp.sal desc) new)
e
where emp.deptno = dept.deptno and e.no >=3 and e.no <=5 and e.empno =
emp.empno;
```

查找出收入（工资加上奖金），下级比自己上级还高的员工编号，员工名字，员工收入

```
select e.ename,e.empno,e.sal+nvl(e.comm,0)
from emp e,emp m
where e.mgr = m.empno and (e.sal+nvl(e.comm,0)) >(m.sal+nvl(m.comm,0));
```

查找出工资等级不为 4 级的员工的员工名字，部门名字，部门位置

```
select ename,dname,loc,sal
from dept d,(select emp.deptno,emp.ename,emp.sal
              from emp,salgrade
              where sal between losal and hisal and grade !=4) new
where d.deptno = new.deptno;
```

```
select
e.ename,d.dname,d.loc
from emp e,
dept d,
(select * from salgrade where grade=4) s
where e.deptno=d.deptno and
(e.sal<s.losal or e.sal>s.hisal);
```

查找出职位和'MARTIN' 或者'SMITH'一样的员工的平均工资

```
select avg(sal)
from emp
where job in (select job
              from emp
              where ename='MARTIN' or ename='SMITH');
```

查找出不属于任何部门的员工

```
select * from emp where deptno is null or deptno not in(select deptno from dept);
```

按部门统计员工数，查出员工数最多的部门的第二名到第五名（列出部门名字，部门位置）

```
select dept.dname, dept.loc
from (select rownum no, deptno from
      (select count(*) employeeSum, deptno from emp group by deptno order by employeeSum
      desc)) e,dept where e.no between 2 and 5 and e.deptno = dept.deptno;
```

查询出 king 所在部门的部门号\部门名称\部门人数

方法一：

```
select t.countno,d.dname,d.loc
      from
dept d,
```

```
(select count(*) countno,deptno
from
emp
where
deptno in (select deptno from emp where ename = 'KING') group by deptno
) t
where
d.deptno = t.deptno;
```

方法二:

```
select t2.countnum,d.dname,d.loc
from
dept d,
(select t.countnum,t.deptno
from
(select count(empno) countnum,deptno from emp group by deptno) t
where
t.deptno in (select deptno from emp where lower(ename) = 'king')
) t2
where
t2.deptno = d.deptno;
```

方法三:

```
select e.deptno as 部门号,d.dname as 部门名称,
(
select count(*) from (select deptno from emp where deptno in (select deptno from emp
where ename='KING') )
) as 部门人数
from emp e, dept d
where e.deptno = d.deptno and e.ename = 'KING';
```

查询出 king 所在部门的工作年限最大的员工名字

```
select ename, hiredate from emp
where hiredate in (select min(hiredate) from emp where deptno in (select deptno from emp
where ename='KING'));
```

查询出工资成本最高的部门的部门号和部门名称

```
select d.deptno,d.dname,t.sum_sal
from
dept d,
(select deptno,sum(sal) sum_sal from emp group by deptno having sum(sal) =
(select max(sum(sal)) from emp group by deptno)
) t
```

```
where
d.deptno = t.deptno;
-----
select d.deptno,d.dname,t.sum_sal
from
dept d,
(select * from (select deptno,sum(sal) sum_sal from emp group by deptno order by sum_sal
desc) where rownum<=1
) t
where
d.deptno = t.deptno;
```

7.8、面试题

面试题一（厦门）

Table: （员工 emp1）

id	name
1	a
2	b
3	c
4	d

Table:(性别 sext)

id	sex
1	男
4	女
5	男

找出忘记填写性别的员工（用 Oracle 的两种方式）

```
select id ,name from emp1 e where e.id not in(select id from sext);
select id from emp1 minus select id from sext;
select * from emp1 e where e.id <> all(select id from sext);
select e.* from emp1 e,(select id from emp1 minus select id from sext) s where e.id = s.id;
select e.id,e.name from emp1 e,sext s where e.id=s.id(+) and s.sex is null;
select * from emp1 left outer join sext on emp1.id = sext.id where sext.sex is null;
select * from emp1 e where not exists(select * from sext s where e.id
= s.id);
select * from emp1 where id not in (select emp1.id from emp1, sext where emp1.id = sext.id);
select name from emp1 where id not in (select id from emp1 intersect select id from sext);
SELECT
*
```

```
FROM emp1 e
WHERE
(SELECT
COUNT(*)
FROM
(SELECT id FROM emp1 UNION ALL SELECT id FROM sext) t
WHERE t.id = e.id) <2;
```

面试题二（上海）

表一(AAA)

商品名称 mc	商品总量 sl
A	100
B	120

表二(BBB)

商品名称 mc	出库数量 sl
A	10
A	20
B	10
B	20
B	30

用一条 Transact-SQL 语句算出商品 A,B 目前还剩多少？

```
select
AAA.mc,
sl-e.sum_sl as leave
from
AAA,
(select sum(sl) sum_sl,mc from BBB group by mc) e
where
AAA.mc = e.mc
```

```
select AAA.mc,AAA.sl-(select sum(BBB.sl) from BBB where BBB.mc=AAA.mc)
from AAA;
```

面试题三（上海）

人员情况表（employee）中字段包括，员工号（ID），姓名（name），年龄（age），文化程度（wh）：包括四种情况（本科以上，大专，高中，初中以下），现在我要根据年龄字段查询统计出：表中文化程度为本科以上，大专，高中，初中以下，各有多少人，占总人数多少。结果如下：

学历	年龄	人数	百分比
本科以上	20	34	14
大专	20	33	13
高中	20	33	13

初中以下	20	100	40
本科以上	21	50	20

.....

Transact-SQL 查询语句如何写？

```
select wh,age,trunc(count(*)/(select count(*) from employee)*100)
from employee
group by wh,age;
```

面试题四（上海）

1. Here's two table STUDENT and SCORE_RANK, write a SQL, list all student's name who ranks 'A'

Table STUDENT

COLUMN NAME COLUMN TYPE Comment

ID char(9) not nullable Student's ID

NAME Varchar(30) not nullable Student's Name

SCORE Int nullable Student's score

Table SCORE_RANK

COLUMN NAME COLUMN TYPE Comment

LO_SCORE Int not nullable Low score

HI_SCORE Int not nullable High score

RANK Char(1) nullable rank

Select name from student,score_rank where score between lo_score and hi_score and rank='A';

2. Here's two table STUDENT and SCORES, a student who have 3 or more courses rank 'A' is a 'GOOD LEARNER', write a SQL, list all 'GOOD LEARNER' 's name.

Table STUDENT

COLUMN NAME COLUMN TYPE Comment

ID char(9) not nullable Student's ID

NAME Varchar(30) not nullable Student's Name

Table SCORE

COLUMN NAME COLUMN TYPE Comment

STUDENT_ID char(9) not nullable Student's ID

COURSE_ID Int not nullable Course ID

SCORE Int not nullable Stuend's score of this course

Select name from student,(Select student_id from score,rank where score between lo_score and


```
hi_score and rank='A' group by student_id having count(course_id)>=3) e where  
id=e.student_id;
```

面试题五（福州）

四张表学生表 student (sid,sname),
教师表 teacher(tid,tname),
课程表 course(cid,cname, ctype),
选课表 choose_course(ccid,sid,tid,cid)

```
insert into student values(1,'小明');  
insert into student values(2,'小花');
```

```
insert into teacher values(1,'陈红');  
insert into teacher values(2,'陈白');
```

```
insert into course values(1,'语文','文科');  
insert into course values(2,'数学','理科');
```

```
--小明选了陈红老师的语文  
insert into choose_course values(1,1,1,1);  
--小明选了陈红老师的数学  
insert into choose_course values(2,1,1,2);  
--小花选了陈红老师的数学  
insert into choose_course values(3,2,1,2);  
--小明选了陈白老师的语文  
insert into choose_course values(1,1,2,1);  
--小花选了陈红老师的语文  
insert into choose_course values(4,2,1,1);
```

1.查找陈红老师教的学生是那些？

```
select *  
from student stu  
where stu.sid in  
        (select distinct cc.sid  
         from teacher t,choose_course cc  
         where t.tid = cc.tid and t.tname='陈红');
```

```
-----  
select distinct s.sid,s.sname  
        from  
        teacher t,choose_course cc ,student s
```

where

```
t.tid = cc.tid  
and t.tname='陈红'  
and s.sid = cc.sid;
```

2.找学生小明所有的文科老师?

```
select *  
from teacher  
where tid in  
(  
    select distinct tid  
    from choose_course cc,student s,course c  
    where cc.sid = s.sid  
    and cc.cid = c.cid  
    and s.sname = '小明'  
    and c.ctype = '文科'  
);
```

3.找出没有选修陈红老师的学生

```
select *  
from student stu  
where stu.sid not in  
    (select cc.sid  
     from teacher t,choose_course cc  
     where t.tid = cc.tid and t.tname='陈红');
```

4.教的学生最少的老师?

```
select t.tid,t.tname,t2.cou from teacher t,  
(  
    select tid,cou from  
    (select tid,count(distinct sid) cou  
     from choose_course group by tid order by cou) t1  
    where rownum=1) t2  
where  
t.tid = t2.tid;
```

```
select t.* from teacher t where t.tid =  
(  
    select tid from  
    (select tid,count(distinct sid) cou  
     from choose_course group by tid order by cou) t1
```

```
where rownum=1);
```

面试题六（厦门）

8:00--12:00 为迟到, 12:00--18:00 为早退

打卡表 card

```
SQL> create table card(  
    cid number(20),  
    ctime date,  
    cuser number(20));
```

人员表 person

```
create table person(  
    pid number(20),  
    name varchar2(10)  
)
```

--插入人员表的数据

```
insert into person values(1,'a');  
insert into person values(2,'b');
```

--插入打卡的数据

```
insert into card values(1,to_date('20090719080200','yyyymmddhh24miss'),1);  
insert into card values(2,to_date('20090719180200','yyyymmddhh24miss'),1);  
insert into card values(3,to_date('20090719090200','yyyymmddhh24miss'),2);  
insert into card values(4,to_date('20090719170200','yyyymmddhh24miss'),2);
```

```
insert into card values(5,to_date('20090720080200','yyyymmddhh24miss'),1);  
insert into card values(6,to_date('20090720160200','yyyymmddhh24miss'),1);  
insert into card values(7,to_date('20090720070200','yyyymmddhh24miss'),2);  
insert into card values(8,to_date('20090720200200','yyyymmddhh24miss'),2);
```

--分析： 先分组统计出每个人，每天的上班时间和下班时间 即（id,day,mindate,maxdate）

```
select p.pid as id,  
    to_char(c.ctime,'yyyymmdd') as day,  
    to_char(min(c.ctime),'hh24mi') as mdate,  
    to_char(max(c.ctime),'hh24mi') as maxdate  
from card c,person p where c.cuser = p.pid group by p.pid,to_char(c.ctime,'yyyymmdd');
```

--把上面的分析做成一个视图，判断上班时间为否为迟到 和 下班时间为否为早退
-- 如 果 判 断 前 10 天 的 打 卡 记 录 ， 就 改 成
to_char(c.ctime,'yyyymmdd')<=to_char(sysdate-10,'yyyymmdd')

```
select p.name as person_name,
e1.day as work_day,
e1.mindate as AM,
e1.maxdate as PM,
--判断迟到
case
when e1.mindate between '0800' and '1200' then 'yes'
else 'no'
end as later,
--判断早退
case
when e1.maxdate between '1201' and '1800' then 'yes'
else 'no'
end as leave_early
from
--员工表
person p,
--上面那张视图表
(select
p.pid as id,
to_char(c.ctime,'yyyymmdd') as day,
to_char(min(c.ctime),'hh24mi') as mindate,
to_char(max(c.ctime),'hh24mi') as maxdate
from card c,person p
where
c.cuser = p.pid and
to_char(c.ctime,'yyyymmdd')<=to_char(sysdate-1,'yyyymmdd')
group by p.pid,to_char(c.ctime,'yyyymmdd')
) e1
where p.pid = e1.id;
```

面试题七（福州）

删除一张表重复记录（ID 是自增唯一，重复记录：其他字段都是一样）

非常经典的一道面试题（可能存在很多数据，要求性能比较高）

```
1 louis 20
2 louis 20
3 jimmy 30
```

4 louis 20

delete from aa where id not in(select min(id) from aa group by name,age);

```
select a1.id
from a a1,
      a a2
where a1.id>a2.id and a1.name=a2.name and a1.age=a2.age and a1.sex=a2.sex
```

面试题八（福州）

用一条 SQL 语句 查询出每门课都大于 80 分的学生姓名

name	kecheng	fenshu
张三	语文	81
张三	数学	75
李四	语文	76
李四	数学	90
王五	语文	81
王五	数学	100
王五	英语	90

```
select Distinct name from TEST A Where Not Exists(Select * from TEST Where Name
=A.Name And fenshu<=80)
```

```
select name
from test
group by name
having min(fenshu)>80
```

```
select name from test where name not in(select name from test where fenshu<=80)
```

面试题九（福州）

有一表 table1 有两个字段 FID, Fno, 写一个 SQL 语句列出该表中一个 FID 对应多个不同的 Fno 的纪录。

类如:

101	1001
101	1001
102	1002
102	1003

103 1004
104 1005
104 1006
105 1007
105 1007
105 1007
105 1008

结果:

102 1002
102 1003
104 1005
104 1006
105 1007
105 1008

```
select distinct fid,fno from table1 where fid in  
(select fid from table1 group by fid having count(distinct fno)>1)
```

面试题十（福州）

有员工表 empinfo

```
(  
Fempno varchar2(10) not null pk,  
Fempname varchar2(20) not null,  
Fage number not null,  
Fsalary number not null  
);
```

假如数据量很大约 1000 万条；写一个你认为最高效的 SQL，用一个 SQL 计算以下四种人：

fsalary>9999 and fage > 35

fsalary>9999 and fage < 35

fsalary<9999 and fage > 35

fsalary<9999 and fage < 35

每种员工的数量；

```
select sum(case when tt.fsalary>9999 and fage > 35  
              then 1  
              else 0  
            end) as "fsalary>9999 and fage > 35",  
       sum(case when tt.fsalary>9999 and fage < 35  
              then 1  
              else 0  
            end) as "fsalary>9999 and fage < 35",
```

```
sum(case when tt.fsalary<9999 and fage > 35
      then 1
      else 0
    end) as "fsalary<9999 and fage > 35",
sum(case when tt.fsalary<9999 and fage < 35
      then 1
      else 0
    end) as "fsalary>9999 and fage < 35"
from empinfo tt;
```

面试题十一（上海）

表内容：

2005-05-09 胜

2005-05-09 胜

2005-05-09 负

2005-05-09 负

2005-05-10 胜

2005-05-10 负

2005-05-10 负

如果要生成下列结果，该如何写 sql 语句？

胜 负

2005-05-09 2 2

2005-05-10 1 2

```
1)select rq, sum(case when shengfu='胜' then 1 else 0 end)'胜',sum(case when shengfu='负' then
      1 else 0 end)'负' from #tmp group by rq
```

```
3)select a.col001,a.a1 胜,b.b1 负 from
```

```
(select col001,count(col001) a1 from temp1 where col002='胜' group by col001) a,
```

```
(select col001,count(col001) b1 from temp1 where col002='负' group by col001) b
```

```
where a.col001=b.col001
```

面试题十二（上海）

请用一个 sql 语得出结果

从 table1,table2 中取出如 table3 所列格式数据

table1

月份 mon 部门 dep 业绩 yj

一月份 01 10

一月份 02 10

一月份 03 5

二月份 02 8

二月份 04 9

三月份 03 8

table2

部门 dep 部门名称 dname

01 国内业务一部
02 国内业务二部
03 国内业务三部
04 国际业务部
05 其他部门

table3 (result)

部门 dep 部门名称 一月份 二月份 三月份

01 国内业务一部 10 0 0
02 国内业务二部 10 8 0
03 国内业务三部 0 5 8
04 国际业务部 0 0 9
05 其他部门

```
select a.dep,a.dname  
sum(case when b.mon='一月份' then b.yj else 0 end) as '一月份',  
sum(case when b.mon='二月份' then b.yj else 0 end) as '二月份',  
sum(case when b.mon='三月份' then b.yj else 0 end) as '三月份',  
from table2 a left join table1 b on a.dep=b.dep
```

面试题十二（上海）

华为一道面试题

一个表中的 Id 有多个记录，把所有这个 id 的记录查出来，并显示共有多少条记录数。

```
select id, Count(*) from tb group by id having count(*)>1
```

第八章 高级查询

8.1、随机返回 5 条记录

```
Select * from (select ename,job from emp order by dbms_random.value()) where rownum<=5
```


8.3、处理空值排序

```
select * from emp order by comm desc nulls last(first);
```

8.4、查询跳过表中的偶数行

```
select ename from (select row_number() over (order by ename) rn,ename  
from emp) x where mod(rn,2)=1;
```

8.5、查询员工信息与其中工资最高最低员工

```
select ename,sal,max(sal) over(), min(sal) over() from emp;  
SQL> select ename,sal,max(sal) over(), min(sal) over() from emp;
```

ENAME	SAL	MAX(SAL)OVER()	MIN(SAL)OVER()
SMITH	800.00	5000	800
ALLEN	1600.00	5000	800
WARD	1250.00	5000	800
JONES	2975.00	5000	800
MARTIN	1250.00	5000	800
BLAKE	2850.00	5000	800
CLARK	2450.00	5000	800
SCOTT	3000.00	5000	800
KING	5000.00	5000	800
TURNER	1500.00	5000	800
ADAMS	1100.00	5000	800
JAMES	950.00	5000	800
FORD	3000.00	5000	800
MILLER	1300.00	5000	800

```
14 rows selected
```

8.5、连续求和

```
select ename,sal,sum(sal) over(), sum(sal) over(order by ename) from  
emp;
```

sum(sal) over(order by ename)指的是连续求和.是以 ename 来排序的。若有两个这样的窗口函数，以后面的排序为主。

```
SQL> select ename,sal,sum(sal) over(), sum(sal) over(order by ename) from emp;
```

ENAME	SAL	SUM(SAL)OVER()	SUM(SAL)OVER(ORDERBYENAME)
ADAMS	1100.00	29025	1100
ALLEN	1600.00	29025	2700
BLAKE	2850.00	29025	5550
CLARK	2450.00	29025	8000
FORD	3000.00	29025	11000
JAMES	950.00	29025	11950
JONES	2975.00	29025	14925
KING	5000.00	29025	19925

8.5、分部门连续求和

```
select deptno,sal ,sum(sal) over (partition by deptno order by ename)
as s from emp;分部门连续求和
```

sum(sal) over (partition by deptno) 分部门求和

```
SQL> select deptno,sal ,sum(sal) over (partition by deptno order by ename) as s from emp;
```

DEPTNO	SAL	S
10	2450.00	2450
10	5000.00	7450
10	1300.00	8750
20	1100.00	1100
20	3000.00	4100
20	2975.00	7075
20	3000.00	10075
20	800.00	10875

8.6、得到当前行上一行或者下一行的数据

```
select ename,sal,lead(sal) over(order by sal) aaa ,lag(sal) over(order
by sal) bbb from emp;
```

month person income

月份 人员 收入

要求用一个 SQL 语句，统计每个月及上月和下月的总收入

要求列表输出为

月份 当月收入 上月收入 下月收入

```
select month,sum(income) ,lead(sum(income)) over(order by sum(income)) from
table group by month;
```

8.7、根据子串分组

```
Select to_char(hiredate,'yyyy'),avg(sal) from emp group by to_char(hiredate,'yyyy');
```

8.8、确定一年内的天数

```
select    add_months(trunc(sysdate,'y'),12)-trunc(sysdate,'y')    from
dual;
```

8.9、查询 EMP 员工表下每个部门工资前二名的员工信息

```
select deptno, ename, sal
from emp e1
where
    (select count(1)
     from emp e2
     where e2.deptno=e1.deptno and e2.ename!=e1.ename and e2.sal>e1.sal)
    <2
order by deptno, sal desc;
```

```
-----
select * from
(select deptno,ename,sal,row_number() over (partition by deptno
      order by sal desc) rn
 from emp)
  where rn<3;
```

8.9、student,course,choose_course, 找出选了所有课程的学生信息

```
select distinct sname from student s
where not exists(select * from course c where not exists(select * from
choose_course cc where s.sid=cc.sid and c.cid = cc.cid));
```

```
select student_id,count(distinct cid) from choose_course group by
student_id having count(distinct cid) = (select count(distinct cid) from
course);
```

第九章 数据字典

9.1、查询某用户下所有表

```
select table_name from all_tables where owner='SCOTT';
```

9.2、查询 EMP 表中所有字段（列）

```
select * from all_tab_columns where table_name='EMP';
```

9.3、列出表的索引列

```
select * from sys.all_ind_columns where table_name='EMP';  
select * from sys.all_ind_columns where  
upper(table_name)='CAREUSERHAM';
```

9.4、列出表中约束

```
select * from all_constraints where table_name='EMP'
```

9.5、在 oracle 中描述数据字典视图

```
select table_name ,comments from dictionary where table_name like '%TABLE%';
```

第十章 Oracle 数据类型

10.1、Oracle 主要数据类型 number(4,2) 24.223

Char,nchar,vchar2,nvarchar2,number(),date,lob(binary 二进制流的大对象),clob（文件大对象）

注意：

- 1、由于 char 是以固定长度的，所以它的速度会比 varchar2 快得多!但程序处理起来要麻烦一点，要用 trim 之类的函数把两边的空格去掉
- 2、Varchar2 一般适用于英文和数字，Nvarchar2 适用中文和其他字符，其中 N 表示 Unicode 常量，可以解决多语言字符集之间的转换问题
- 3、Number(4,2) 指的是整数占 2 位，小数占 2 位(99.994 可以。99.995 不行，因为是四舍五入)
- 4、Number 默认为 38 位

10.2、char,nchar,varchar2,nvarchar2 区别

10.3、Java 对 blob 字段的操作

第十一章 Oracle 体系结构(DBA)

11.1、数据库（Database）

一系列物理文件（数据文件，控制文件，联机日志等）的集合或与之对应的逻辑结构（表空间，段等）被称为数据库

物理存储结构

数据文件、重做日志文件、控制文件

Desc v\$logfile;

Select member from v\$logfile;

V\$controlfile

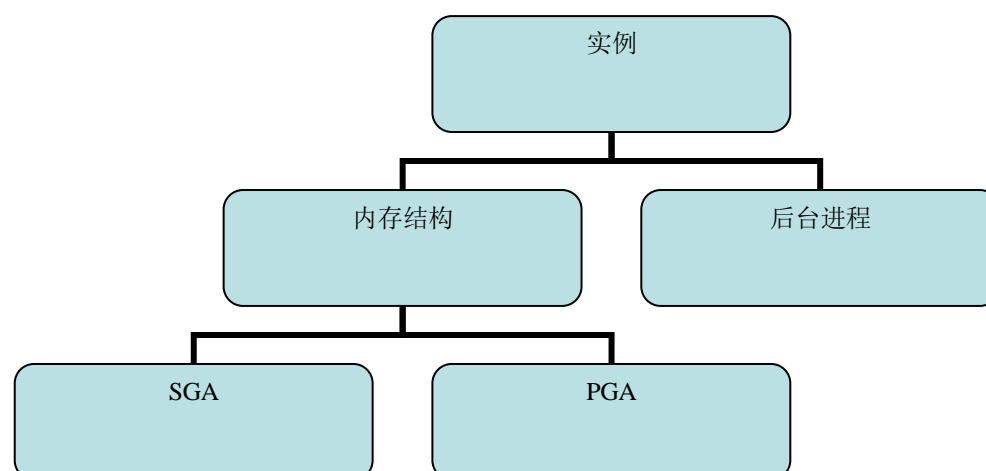
V\$datafile;

逻辑存储结构

表空间、段、区、块

11.2、数据库实例（Database Instance）

ORACLE 内存结构和后台进程被成为数据库的实例



11.3、Oracle 实时应用集群（RAC, Real Application Clusters）

11.4、数据库服务名（ Database Service_Name）

11.5、网络服务名（Net Service Name）

11.6、监听器（Monitor）

（参考：oracle 体系结构）

（参考：Oracle DBA 面试题）

第十二章 DDL(改变表结构)

12.1、创建表

```
Create table student(  
Sid number(10),  
Sname varchar2(10)  
) tablespace tt;
```

以上 tablespace 不是必须的。默认不写，则创建在登录的用户所在的表空间上

12.2、使用子查询创建表

```
create table myemp as select * from emp;  
create table myemp as select * from emp where deptno=10;  
create table myemp as select * from emp 1=2;
```

12.3、添加字段

```
Alter table student add age number(5);
```

12.4、修改字段

```
Alter table student modify age number(10);  
alter table table2 rename column result to result2;
```

12.5、删除字段

```
Alter table student drop column age;
```

12.6、清空表中数据

```
Truncate table student;
```

正常情况下删除数据,如果发现删除错了,则可以通过 **rollback** 回滚。如果使用了截断表,则表示所有的数据不可恢复了.**所以速度很快** (更详细的说明可查看 **Oracle 体系结构**)

12.7、删除表

```
Drop table student;
```

12.8、重命名表

```
Rename student to student1;
```

12.9、练习

1.scott 下面创建一个表

emp1

empno number(10)

ename varchar2(50)

create table emp1(empno number(10),ename varchar2(50));

2.添加一个字段

sal number(10,2)

alter table emp1 add sal number(10,2);

3.修改字段 ename varchar(100)

alter table emp1 modify ename varchar(100);

4.删除字段 sal

alter table emp1 drop column sal;

5.把表 emp1 改成 emp2

rename emp1 to emp2;

6.删除表 emp2

```
drop table emp2;
```

7.创建一个和 emp 结构一样的表，并同时插入工资大于 1000 的数据

```
create table emp3 as select * from emp where sal>1000;
```

8.清空表数据（用 truncate）

```
truncate table emp3;
```

第十三章 DML(改变数据结构)

13.1、insert 语句

表间数据拷贝 insert into dept1(id, name) select deptno, dname from dept;

13.2、update 语句

将编号为 7779 用户的工作换成编号为 7566 的雇员的工作和所属上级。

```
UPDATE myemp SET(job,mgr) = (SELECT job,mgr FROM myemp WHERE  
empno=7566) WHERE empno=7779 ;
```

如果子查询中返回的是空，则目标字段也更新成 NULL.

13.3、delete 语句

```
Delete from emp;
```

13.4、merge 语句

```
create table test1(eid number(10), name varchar2(20),birth date,salary number(8,2));
```

```
insert into test1 values (1001, '张三', '20-5 月-70', 2300);
```

```
insert into test1 values (1002, '李四', '16-4 月-73', 6600);
```

```
select * from test1;
```

```
create table test2(eid number(10), name varchar2(20),birth date,salary number(8,2));
```

```
select * from test2;
```

```
merge into test2
```

```
using test1
```

```
on(test1.eid = test2.eid )
```

```
when matched then
```

```
update set name = test1.name, birth = test1.birth, salary = test1.salary
```

```
when not matched then
```



```
insert (eid, name, birth, salary) values(test1.eid, test1.name, test1.birth, test1.salary);
```

```
select * from test2;
```

13.5、练习

1.往 emp 表中插入 empno,ename,sal 数据 (111,'1',1000)(222,'2',2000)

```
insert into emp(empno,ename,sal) values(111,'1',1000);
```

```
insert into emp(empno,ename,sal) values(222,'2',2000);
```

2.把 empno=111 的员工 comm 改成 100

```
update emp set comm=100 where empno=111;
```

3.往 dept 表中插入 dept 表中 deptno=100 的数据

```
insert into dept select * from dept where deptno=100;
```

4.删除 empno=111 的数据

```
delete from emp where empno=111;
```

5.为所有人长工资，标准是：10 部门长 10%；20 部门长 15%；30 部门长 20% 其他部门长 18%（要求用 DECODE 函数）

```
update emp
```

```
set sal=decode(deptno,'10',sal*(1+0.1), '20',sal*(1+0.15), '30',sal*(1+0.2),sal*(1+0.18));
```

6.根据工作年限长工资，标准是：为公司工作了几个月就长几个百分点。

```
update emp set sal= round(sal * (1+(sysdate - hiredate)/365/12/100),2);
```

第十四章 约束

约束就是指对插入数据的各种限制，例如：人员的姓名不能为空，人的年龄只能在 0~150 岁之间。约束可以对数据库中的数据进行保护。

约束可以在建表的时候直接声明，也可以为已建好的表添加约束。

14.1、NOT NULL：非空约束

例如：姓名不能为空

```
CREATE TABLE person
(
  pid NUMBER ,
  name VARCHAR(30) NOT NULL
);
-- 插入数据
INSERT INTO person(pid,name) VALUES (11,'张三');
-- 错误的数据，会受到约束限制，无法插入
```

```
INSERT INTO person(pid) VALUES (12);
```

14.2、PRIMARY KEY：主键约束

- 不能重复，不能为空
- 例如：身份证号不能为空。

现在假设 pid 字段不能为空，且不能重复。

```
DROP TABLE person ;
CREATE TABLE person
(
  pid NUMBER PRIMARY KEY ,
  name VARCHAR(30) NOT NULL
);
-- 插入数据
INSERT INTO person(pid,name) VALUES (11,'张三');
-- 主键重复了
INSERT INTO person(pid,name) VALUES (11,'李四');
```

14.3、UNIQUE：唯一约束，值不能重复（空值除外）

人员中有电话号码，电话号码不能重复。

```
DROP TABLE person ;
CREATE TABLE person
(
  pid NUMBER PRIMARY KEY NOT NULL ,
  name VARCHAR(30) NOT NULL ,
  tel VARCHAR(50) UNIQUE
);
-- 插入数据
INSERT INTO person(pid,name,tel) VALUES (11,'张三','1234567');
-- 电话重复了
INSERT INTO person(pid,name,tel) VALUES (12,'李四','1234567');
```

14.4、CHECK：条件约束，插入的数据必须满足某些条件

例如：人员有年龄，年龄的取值只能是 0~150 岁之间

```
DROP TABLE person ;
CREATE TABLE person
(
  pid NUMBER PRIMARY KEY NOT NULL ,
```

```
name VARCHAR(30) NOT NULL ,
tel VARCHAR(50) NOT NULL UNIQUE ,
age NUMBER CHECK(age BETWEEN 0 AND 150)
);
-- 插入数据
INSERT INTO person(pid,name,tel,age) VALUES (11,'张三','1234567',30);
-- 年龄的输入错误
INSERT INTO person(pid,name,tel,age) VALUES (12,'李四','2345678',-100);
```

14.5、Foreign Key: 外键

例如：有以下一种情况：

- 一个人有很多本书：

|- **Person** 表

|- **Book** 表：而且 **book** 中的每一条记录表示一本书的信息，一本书的信息属于一个人

```
CREATE TABLE book
(
  bid NUMBER PRIMARY KEY NOT NULL ,
  name VARCHAR(50) ,
  -- 书应该属于一个人
  pid NUMBER
);
```

如果使用了以上的表直接创建，则插入下面的记录有效：

```
INSERT INTO book(bid,name,pid) VALUES(1001,'JAVA',12) ;
```

以上的代码没有任何错误，但是没有任何意义，因为一本书应该属于一个人，所以在此处的 **pid** 的取值应该与 **person** 表中的 **pid** 一致。

此时就需要外键的支持。修改 **book** 的表结构

```
DROP TABLE book ;
CREATE TABLE book
(
  bid NUMBER PRIMARY KEY NOT NULL ,
  name VARCHAR(50) ,
  -- 书应该属于一个人
  pid NUMBER REFERENCES person(pid) ON DELETE CASCADE
  -- 建立约束：book_pid_fk, 与 person 中的 pid 为主-外键关系
  --CONSTRAINT book_pid_fk FOREIGN KEY(pid) REFERENCES person(pid)
);

INSERT INTO book(bid,name,pid) VALUES(1001,'JAVA',12) ;
```

14.6、级联删除

那么再分析：

如果假设一个人的人员信息没有了，那么此人所拥有的书还应该存在吗？最好，如果 **person** 中的一条数据没了，则对应在 **book** 中的数据也应该同时消失。

在之前的结构上执行 **delete** 语句，删除 **person** 表中的一条记录：

```
DELETE FROM person WHERE pid=11;
```

提示不能删除的错误：因为 **book** 中存在着此项的关联，如果 **person** 表中的一条数据删除了，则肯定会直接影响到 **book** 表中数据的完整性，所以不让删除。

如果非要删除，则应该先删除 **book** 表中的对应数据，之后再删除 **person** 表中的对应数据。

此时如果想完成删除 **person** 表的数据同时自动删除掉 **book** 表的数据操作，则必须使用级联删除。

在建立外键的时候必须指定级联删除（**ON DELETE CASCADE**）。

```
CREATE TABLE book
(
  bid NUMBER PRIMARY KEY NOT NULL ,
  name VARCHAR(50) ,
  -- 书应该属于一个人
  pid NUMBER ,
  -- 建立约束：book_pid_fk, 与 person 中的 pid 为主-外键关系
  CONSTRAINT book_pid_fk FOREIGN KEY(pid) REFERENCES person(pid) ON DELETE
CASCADE
);
```

```
DROP TABLE book ;
DROP TABLE person ;
CREATE TABLE person
(
  pid NUMBER ,
  name VARCHAR(30) NOT NULL ,
  tel VARCHAR(50),
  age NUMBER
);
CREATE TABLE book
(
  bid NUMBER ,
  name VARCHAR(50) ,
  pid NUMBER
);
```

以上两张表中没有任何约束，下面使用 `alter` 命令为表添加约束

1、为两个表添加主键：

·person 表 pid 为主键：

```
ALTER TABLE person ADD CONSTRAINT person_pid_pk PRIMARY KEY(pid) ;
```

·book 表 bid 为主键：

```
ALTER TABLE book ADD CONSTRAINT book_bid_pk PRIMARY KEY(bid) ;
```

2、为 person 表中的 tel 添加唯一约束：

```
ALTER TABLE person ADD CONSTRAINT person_tel_uk UNIQUE(tel) ;
```

3、为 person 表中的 age 添加检查约束：

```
ALTER TABLE person ADD CONSTRAINT person_age_ck CHECK(age BETWEEN 0 AND 150) ;
```

4、为 book 表中的 pid 添加与 person 的主-外键约束，要求带级联删除

```
ALTER TABLE book ADD CONSTRAINT person_book_pid_fk FOREIGN KEY (pid) REFERENCES person(pid) ON DELETE CASCADE ;
```

Q:如何用 `alter` 添加非空约束

A:用 `check` 约束

14.7、删除约束：

```
ALTER TABLE book DROP CONSTRAINT person_book_pid_fk ;  
alter table student drop unique(tel) ;
```

14.8、启用约束

```
ALTER TABLE book enable CONSTRAINT person_book_pid_fk ;
```

14.9、禁用约束

```
ALTER TABLE book disable CONSTRAINT person_book_pid_fk ;
```

14.10、练习

1.创建一张表 student

id number

name varchar2(10)

age number(10)

tel varchar2 (10)

给 id 字段添加主键约束

给 name 字段添加非空约束

给 age 字段添加 check 约束 (age 必须大于 18 岁)

给 tel 添加唯一 非空 约束

```
create table student(  
id number primary key,  
name varchar2(10) not null,  
age number(10) check(age > 18),  
tel varchar2(10) unique not null  
);
```

2. 创建一张学员兴趣爱好表 hobby

```
id number(10)  
hobby_name varchar2(10)  
sidnumber --学生 id  
给 sid 字段添加外键约束，并且要带级联删除
```

```
create table hobby(  
id number(10),  
hobby_name varchar2(10),  
sid number references student(id) on delete cascade  
);
```

3. 删除掉 student 表中 tel 字段的唯一约束（先写出查看该表约束的 sql）

```
select constraint_name, constraint_type from all_constraints where user_table = upper('student');  
alter table student drop constraint unique(tel);
```

4. 手动添加 student 表中 tel 字段的唯一约束（约束名为：MY_CONSTRAINT_1）

```
alter table student add constraint MY_CONSTRAINT_1 unique(tel);
```

5. 禁用约束 MY_CONSTRAINT_1

```
alter table student disable constraint MY_CONSTRAINT_1;
```

6. 启用约束 MY_CONSTRAINT_1

```
alter table student enable constraint MY_CONSTRAINT_1;
```

第十五章 视图

视图：是一个封装了各种复杂查询的语句，就称为视图。

15.1、创建视图

CREATE VIEW 视图名字(字段) AS 子查询

建立一个只包含 20 部门雇员信息的视图（雇员的编号、姓名、工资）

```
CREATE VIEW empv20 (empno,ename,sal) AS SELECT empno,ename,sal FROM emp
WHERE deptno=20 ;
```

例如：将之前的一个复杂语句包装成视图

显示部门内最低工资比 20 部门最低工资要高的部门的编号及部门内最低工资：

```
SELECT deptno,MIN(sal) FROM emp GROUP BY deptno HAVING MIN(sal)>(SELECT
MIN(sal) FROM emp WHERE deptno=20) ;
```

此时就可以将上面的复杂查询语句建立一张视图，之后查询视图即可。

15.1、高级视图

如果要创建一个同名的视图，则必须先将之前的视图删除掉，再进行创建：

```
DROP VIEW empv20 ;
```

有些时候如果先删除再创建操作会比较麻烦，所以有时候最好的方式：如果视图存在则先自动删除，之后自动创建。

```
CREATE OR REPLACE VIEW empv20 (deptno,msal) AS (SELECT deptno,MIN(sal) FROM
emp GROUP BY deptno HAVING MIN(sal)>(SELECT MIN(sal) FROM emp WHERE
deptno=20)) ;
```

例如，还是创建一个只包含 20 部门的视图

```
CREATE OR REPLACE VIEW empv20 (empno,ename,sal,deptno) AS SELECT
empno,ename,sal,deptno FROM emp WHERE deptno=20 ;
```

现在直接更新视图里的数据

将 7369 的部门编号修改为 30。此操作在视图中完成。

```
update empv20 SET deptno=30 where empno=7369 ;
```

此时，提示更新完成。

默认情况下创建的视图，如果更新了，则会自动将此数据从视图中删除，之后会更新原本的数据。

思考：

如果能这样做的话，肯定存在问题，因为视图最好还是不要更新。

在建立视图的时候有两个参数：

·WITH CHECK OPTION → 保护视图的创建规则

```
CREATE OR REPLACE VIEW empv20 (empno,ename,sal,deptno)
AS SELECT empno,ename,sal,deptno FROM emp WHERE deptno=20
WITH CHECK OPTION CONSTRAINT empv20_ck;
```

再执行更新操作：

update empv20 SET deptno=30 where empno=7369 ; → 此处更新的是部门编号，失败

└ 之前是按照部门编号建立的视图，所以不能修改部门编号

update empv20 SET ename='tom' where empno=7369 ; → 可以更新，更新的是名字，成功

·WITH READ ONLY（只读，不可修改），视图最好不要轻易的修改

```
CREATE OR REPLACE VIEW empv20 (empno,ename,sal,deptno)
AS SELECT empno,ename,sal,deptno FROM emp WHERE deptno=20
WITH READ ONLY;
```

现在任意的字段都不可更改，所以现在的视图是只读的。

如果视图的基表有多行查询（比如：group by,distinct）那么该视图也是只读的

15.1、查看视图

Select text from user_views;查看视图的创建语句

第十六章 索引

16.1、索引

select * from user_indexes 查询现有的索引

select * from user_ind_columns 可获知索引建立在那些字段上

索引

- 什么是索引(Index)?
 - 一种用于提升查询效率的数据库对象;
 - 通过快速定位数据的方法, 减少磁盘 I/O 操作;
 - 索引信息与表独立存放;
 - Oracle 数据库自动使用和维护索引。
- 索引分类
 - 唯一性索引
 - 非唯一索引
- 创建索引的两种方式
 - 自动创建 - 在定义主键或唯一键约束时系统会自动在相应的字段上创建唯一性索引。
 - 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。

16.2、索引优缺点

建立索引的优点

1. 大大加快数据的检索速度;
2. 创建唯一性索引, 保证数据库表中每一行数据的唯一性;
3. 加速表和表之间的连接;
4. 在使用分组和排序子句进行数据检索时, 可以显著减少查询中分组和排序的时间。

索引的缺点

1. 索引需要占物理空间。
2. 当对表中的数据进行增加、删除和修改的时候, 索引也要动态的维护, 降低了数据的维护速度。

16.3、创建索引的原则

创建索引: 创建索引一般有以下两个目的: 维护被索引列的唯一性和提供快速访问表中数据的策略。

--在 select 操作占大部分的表上创建索引;

--在 where 子句中出现最频繁的列上创建索引;

--在选择性高的列上创建索引 (补充索引选择性, 最高是 1, eg: primary key)

--复合索引的主列应该是最有选择性的和 where 限定条件最常用的列, 并以此类推第二

列.....。

--小于 5M 的表，最好不要使用索引来查询，表越小，越适合用全表扫描。

16.4、使用索引的原则

--查询结果是所有数据行的 5%以下时，使用 index 查询效果最好；

--where 条件中经常用到表的多列时，使用复合索引效果会好于几个单列索引。因为当 sql 语句所查询的列，全部都出现在复合索引中时，此时由于 Oracle 只需要查询索引块即可获得所有数据，当然比使用多个单列索引要快得多；

--索引利于 select，但对经常 insert, delete 尤其 update 的表，会降低效率。

eg: 试比较下面两条 SQL 语句(emp 表的 deptno 列上建有 unique index):

语句 A: SELECT dname, deptno FROM dept WHERE deptno NOT IN
(SELECT deptno FROM emp);

语句 B: SELECT dname, deptno FROM dept WHERE NOT EXISTS
(SELECT deptno FROM emp WHERE dept.deptno = emp.deptno);

这两条查询语句实现的结果是相同的，但是执行语句 A 的时候，ORACLE 会对整个 emp 表进行扫描，没有使用建立在 emp 表上的 deptno 索引，执行语句 B 的时候，由于在子查询中使用了联合查询，ORACLE 只是对 emp 表进行的部分数据扫描，并利用了 deptno 列的索引，所以语句 B 的效率要比语句 A 的效率高。

----where 子句中的这个字段，必须是复合索引的第一个字段；

eg: 一个索引是按 f1, f2, f3 的次序建立的，若 where 子句是 f2 = : var2, 则因为 f2 不是索引的第 1 个字段，无法使用该索引。

---- where 子句中的这个字段，不应该参与任何形式的计算：任何对列的操作都将导致表扫描，它包括数据库函数、计算表达式等等，查询时要尽可能将操作移至等号右边。

----应尽量熟悉各种操作符对 Oracle 是否使用索引的影响：以下这些操作会显式（explicitly）地阻止 Oracle 使用索引： is null ; is not null ; not in; !=; like ; numeric_col+0;date_col+0; char_col||' '; to_char; to_number, to_date 等。

Eg:

Select jobid from mytabs where isReq='0' and to_date (updatedate) >= to_Date ('2001-7-18', 'YYYY-MM-DD'); --updatedate 列的索引也不会生效。

16.4、创建索引

```
create index abc on student(sid,sname);  
create index abc1 on student(sname,sid);
```

这两种索引方式是不一样的

索引 abc 对 Select * from student where sid=1; 这样的查询语句更有效

索引 abc1 对 Select * from student where sname='louis'; 这样的查询语句更有效

因此建立索引的时候，字段的组合顺序是非常重要的。一般情况下，需要经常访问的字段放在组合字段的前面

16.5、索引的存储

索引和表都是独立存在的。在为索引指定表空间的时候，不要将被索引的表和索引指向同一个表空间，这样可以避免产生 IO 冲突。使 Oracle 能够并行访问存放在不同硬盘中的索引数据和表数据，更好的提高查询速度。

16.6、删除索引

```
drop index PK_DEPT1;
```

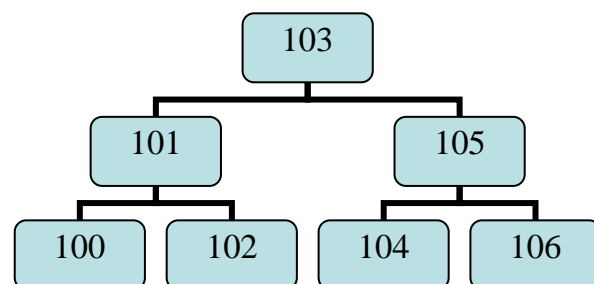
16.7、索引类型

B 树索引 (B-Tree Index)

创建索引的默认类型，结构是一颗树，采用的是平衡 B 树算法：

- 右子树节点的键值大于等于父节点的键值
- 左子树节点的键值小于等于父节点的键值

比如有数据:100,101,102,103,104,105,106



位图索引(BitMap Index)

如果表中的某些字段取值范围比较小，比如职员性别、分数列 ABC 级等。只有两个值。这样的字段如果建 B 树索引没有意义，不能提高检索速度。这时我们推荐用位图索引

```
Create BitMap Index student on(sex);
```

16.8、管理索引

1) 先插入数据后创建索引

向表中插入大量数据之前最好不要先创建索引，因为如果先建立索引。那么在插入每行数据的时候都要更改索引。这样会大大降低插入数据的速度。

2) 设置合理的索引列顺序

3) 限制每个表索引的数量

4) 删除不必要的索引

5) 为每个索引指定表空间

6) 经常做 insert, delete 尤其是 update 的表最好定期 exp/imp 表数据，整理数据，降低碎

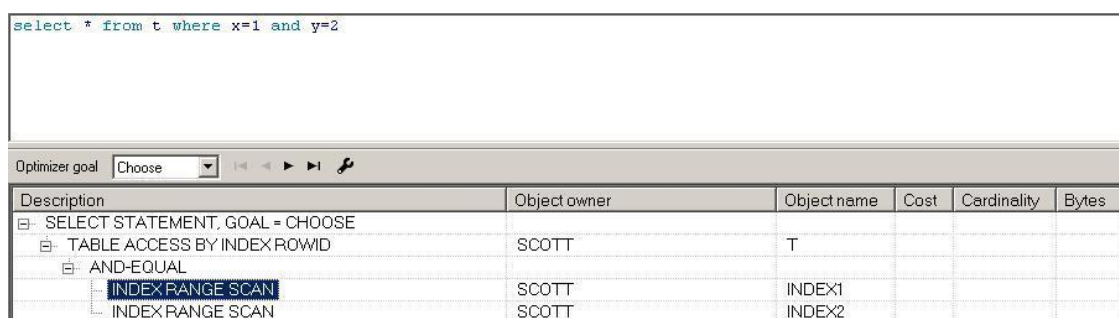
片（缺点：要停应用，以保持数据一致性，不实用）；有索引的最好定期 rebuild 索引（rebuild 期间只允许表的 select 操作，可在数据库较空闲时间提交），以降低索引碎片，提高效率

16.8、索引问题

- 1: 针对一个表的查询语句能否会用到两个索引？
- 2: 如果能用到，那么其实现原理是怎样的？
- 3: 效率如何？其代价如何，比如额外开销等。

回答：

1. 一个表的查询语句可以同时用到两个索引。如下图：



Description	Object owner	Object name	Cost	Cardinality	Bytes
SELECT STATEMENT, GOAL = CHOOSE					
TABLE ACCESS BY INDEX ROWID	SCOTT	T			
AND-EQUAL					
INDEX RANGE SCAN	SCOTT	INDEX1			
INDEX RANGE SCAN	SCOTT	INDEX2			

2. 索引是以独立于表存在的一种数据库对象，它是对基表的一种排序（默认是 B 树索引就是二叉树的排序方式），比如：

t 表(x,y,z) ,在 x,y,z 上分别都建立了索引(index1,index2,index3)，那在查询 select * from t where x=1 and y=2;的时候，会分别用到 index1,index2。

原理是先到 index1 索引表中查到符合 x=1 条件的记录，然后到 index2 索引表中查到 y=2 条件的记录。

3. 这样的查询效率，肯定是大于没有索引情况的全表扫描（table access full），但是有两个问题。

问题一：建立索引将占用额外的数据库空间，更重要的是增删改操作的时候，索引的排序也必须改变，加大的维护的成本

问题二：如果经常查询 x=?和 y=?，那推荐使用组合 index(x,y)，这种情况下组合索引的效率是远高于两个单独的索引的。

同时在使用组合索引的时候，大家一定要注意一个细节：建立组合索引 index(x,y,z)的时候，那在查询条件中出现 x,xy,xyz,yzx 都是可以用到该组合索引，但是 y,yz,z 是不能用到该索引的。关于这段话的原文如下：

A leading portion of an index is a set of one or more columns that were specified first and consecutively in the list of columns in the CREATE INDEX statement that created the index. Consider this CREATE INDEX statement:

```
CREATE INDEX comp_ind
```

ON table1(x, y, z);

x, xy, and xyz combinations of columns are leading portions of the index

yz, y, and z combinations of columns are not leading portions of the index

第十七章 SQL 优化

删除一张表的重复记录（ID 是自增唯一主键，重复记录：其他字段都是一样）
(数据量很大,性能要求很高)

表名: T

Id name age

1 louis 20

2 louis 20

3 jimmy 30

4 louis 20

做法一:

Delete from t where id not in (Select min(id) from t Group by name,age);

做法二:

```
delete from t where id in (Select distinct a2.id from t a1,t a2 where  
a1.id>a2.id and a1.name=a2.name and a1.age=a2.age);
```

做法三:

```
delete from t a1 where not exists(select *  
from t a2  
where a1.id>a2.id and a1.name=a2.name and a1.age=a2.age  
);
```

前提数据量>100, 0000

以上三种做法，均可。但是第三种做法的性能最佳。第一种用 **not in** 没办法用到索引.第三种方式也不会用到索引

数据量	1000	100000	100,0000
方法一	0.047	3.77	72
方法二	0.286	5.77	65

第二种方式快于第一种方式。

SQL 优化的实质就是在结果正确的前提下，用优化器可以识别的语句，**充份利用索引**，执行过程中访问尽量少的数据块，减少表扫描的 I/O 次数，尽量**避免**全表扫描和其他额外开销。

oracle 数据库常用的两种优化器: RBO (rule-based-optimizer) 和 CBO(cost-based-optimizer)。目前更多地采用 CBO(cost-based-optimizer)基于开销的优化器。在 CBO 方式下, Oracle 会根据表及索引的状态信息来选择计划; 在 RBO 方式下, Oracle 会根据自己内部设置的一些

规则来决定选择计划，例如 oracle 会根据以下优先级来选择执行计划（越靠前，rank 越低，越快）：

17.1、尽量少用 IN 操作符

基本上所有的 IN 操作符都可以用 EXISTS 代替，在选择 IN 或 EXIST 操作时，要根据主子表数据量大小来具体考虑

17.2、尽量用 NOT EXISTS 或者外连接替代 NOT IN 操作符

因为 NOT IN 不能应用表的索引

17.3、尽量不用“<>”或者“!=”操作符

不等于操作符是永远不会用到索引的，因此对它的处理只会产生全表扫描。比如：a<>0 改为 a>0 or a<0

17.4、在设计表时，把索引列设置为 NOT NULL

判断字段是否为空一般是不会应用索引的，因为 B 树索引是不索引空值的。

17.5、尽量不用通配符“%”或者“_”作为查询字符串的第一个字符

当通配符“%”或者“_”作为查询字符串的第一个字符时，索引不会被使用。比如用 T 表中 Column1 LIKE ‘%5400%’ 这个条件会产生全表扫描，如果改成 Column1 ‘X5400%’ OR Column1 LIKE ‘B5400%’ 则会利用 Column1 的索引进行两个范围的查询，性能肯定大大提高。

17.6、Where 子句中避免在索引列上使用计算

如果索引不是基于函数的，那么当在 Where 子句中对索引列使用函数时，索引不再起作用。因此 Where 子句中避免在索引列上使用计算。

比如：

substr(no,1,4)=’5400’，优化处理：no like ‘5400%’
trunc(hiredate)=trunc(sysdate)，优化处理：hiredate >=trunc(sysdate) and hiredate <trunc(sysdate+1)

17.7、用“>=”替代“>”

大于或小于操作符一般情况下是不用调整的，因为它有索引就会采用索引查找，但有的情况下可以对它进行优化，如一个表有 100 万记录，一个数值型字段 A， 30 万记

录的 A=0, 30 万记录的 A=1, 39 万记录的 A=2, 1 万记录的 A=3。那么执行 A>2 与 A>=3 的效果就有很大的区别了, 因为 A>2 时 ORACLE 会先找出为 2 的记录索引再进行比较, 而 A>=3 时 ORACLE 则直接找到=3 的记录索引

17.8、利用 SGA 共享池, 避开 parse 阶段

同一功能同一性能不同写法 SQL 的影响

如一个 SQL 在 A 程序员写的为

```
Select * from zl_yhjbqk
```

B 程序员写的为

```
Select * from dlyx.zl_yhjbqk (带表所有者的前缀)
```

C 程序员写的为

```
Select * from DLYX.ZLYHJBQK (大写表名)
```

D 程序员写的为

```
Select * from DLYX.ZLYHJBQK (中间多了空格)
```

以上四个 SQL 在 ORACLE 分析整理之后产生的结果及执行的时间是一样的, 但是从 ORACLE 共享内存 SGA 的原理, 可以得出 ORACLE 对每个 SQL 都会对其进行一次分析, 并且占用共享内存, 如果将 SQL 的字符串及格式写得完全相同则 ORACLE 只会分析一次, 共享内存也只会留下一次的分析结果, 这不仅可以减少分析 SQL 的时间, 而且可以减少共享内存重复的信息, ORACLE 也可以准确统计 SQL 的执行频率。

不同区域出现的相同的 Sql 语句要保证查询字符完全相同, 建议经常使用变量来代替常量, 以尽量使用重复 sql 代码, 以利用 SGA 共享池, 避开 parse 阶段, 防止相同的 Sql 语句被多次分析, 提高执行速度。

因此使用存储过程, 是一种很有效的提高 share pool 共享率, 跳过 parse 阶段, 提高效率的办法。

17.9、WHERE 后面的条件顺序要求

WHERE 后面的条件, 表连接语句写在最前, 可以过滤掉最大数量记录的条件居后。

比如:

```
Select * from zl_yhjbqk where dy_dj = '1KV 以下' and xh_bz=1
```

```
Select * from zl_yhjbqk where xh_bz=1 and dy_dj = '1KV 以下'
```

以上两个 SQL 中 dy_dj (电压等级) 及 xh_bz (销户标志) 两个字段都没进行索引, 所以执行的时候都是全表扫描, 第一条 SQL 的 dy_dj = '1KV 以下' 条件在记录集内比率为 99%, 而 xh_bz=1 的比率只为 0.5%, 在进行第一条 SQL 的时候 99% 条记录都进行 dy_dj 及 xh_bz

的比较,而在进行第二条 SQL 的时候 0.5% 条记录都进行 dy_dj 及 xh_bz 的比较,以此可以得出第二条 SQL 的 CPU 占用率明显比第一条低。

17.10、使用表的别名,并将之作为每列的前缀

当在 Sql 语句中连接多个表时,使用表的别名,并将之作为每列的前缀。这样可以减少解析时间

17.11、进行了显式或隐式的运算的字段不能进行索引

比如:

ss_df+20>50, 优化处理: ss_df>30

'X' || hbs_bh > 'X5400021452', 优化处理: hbs_bh > '5400021452'

sk_rq+5=sysdate, 优化处理: sk_rq=sysdate-5

hbs_bh=5401002554, 优化处理: hbs_bh=' 5401002554', 注: 此条件对 hbs_bh 进行隐式的 to_number 转换, 因为 hbs_bh 字段是字符型。

17.12、用 UNION ALL 代替 UNION

UNION 是最常用的集操作,使多个记录集联结成为单个集,对返回的数据行有唯一性要求,所以 oracle 就需要进行 SORT UNIQUE 操作(与使用 distinct 时操作类似),如果结果集又比较大,则操作会比较慢;

UNION ALL 操作不排除重复记录行,所以会快很多,如果数据本身重复行存在可能性较小时,用 union all 会比用 union 效率高很多!

17.13、其他操作

尽量使用 packages: Packages 在第一次调用时能将整个包 load 进内存,对提高性能有帮助。

尽量使用 cached sequences 来生成 primary key : 提高主键生成速度和使用性能。

很好地利用空间: 如用 VARCHAR2 数据类型代替 CHAR 等

使用 Sql 优化工具: sqlexpert; toad; explain-table; PL/SQL; OEM

17.14、通过改变 oracle 的 SGA 的大小

SGA: 数据库的系统全局区。

SGA 主要由三部分构成: 共享池、数据缓冲区、日志缓冲区

1、共享池又由两部分构成：共享 SQL 区和数据字典缓冲区。共享 SQL 区专门存放用户 SQL 命令，oracle 使用最近最少使用等优先级算法来更新覆盖；数据字典缓冲区（library cache）存放数据库运行的动态信息。数据库运行一段时间后，DBA 需要查看这些内存区域的命中率以从数据库角度对数据库性能调优。通过执行下述语句查看：

```
select (sum(pins - reloads)) / sum(pins) "Lib Cache" from v$librarycache;
```

--查看共享 SQL 区的重用率，最好在 90% 以上，否则需要增加共享池的大小。

```
select (sum(gets - getmisses - usage - fixED)) / sum(gets) "Row Cache" from v$rowcache;
```

--查看数据字典缓冲区的命中率，最好在 90% 以上，否则需要增加共享池的大小。

2、数据缓冲区：存放 sql 运行结果抓取到的 data block;

```
SELECT name, value FROM v$sysstat WHERE name IN ('db block gets', 'consistent gets', 'physical reads');
```

--查看数据库数据缓冲区的使用情况。查询出来的结果可以计算出来数据缓冲区的使用命中率 = $1 - (\text{physical reads} / (\text{db block gets} + \text{consistent gets}))$ 。命中率应该在 90% 以上，否则需要增加数据缓冲区的大小。

3、日志缓冲区：存放数据库运行生成的日志。

```
select name,value from v$sysstat where name in ('redo entries','redo log space requests');
```

--查看日志缓冲区的使用情况。查询出的结果可以计算出日志缓冲区的申请失败率：申请失败率 = $\text{requests} / \text{entries}$ ，申请失败率应该接近于 0，否则说明日志缓冲区开设太小，需要增加 ORACLE 数据库的日志缓冲区。

第十八章 序列、同义词

17.1、创建序列

```
Create sequence myseq
```

```
Start with 1
```

```
Increment by 1
```

```
Order
```

```
cache 20
```

```
Nocycle;
```

17.2、NextVal,CurrVal

```
Select myseq.nextval from dual;
```

```
Select myseq.currval from dual;
```

(必须先有 nextval，才能有 currval)

查询完之后就已经自增 1 了

Insert into table1 values(myseq.nextval) 这时候已经是 2 了

17.3、Cycle, Cache

而用了 nocycle, 就可以确保当该序列用于多张表的时候, ID 是唯一的

用 cycle 时, 用法如下:

```
create sequence myseq2 start with 1 increment by 1 cycle maxvalue 3  
nocache ;
```

这样到 3 之后, 要会重新从 1 开始

如果指定 CACHE 值, ORACLE 就可以预先在内存里面放置一些 sequence, 这样存取的快些。cache 里面的取完后, oracle 自动再取一组到 cache。使用 cache 或许会跳号, 比如数据库突然不正常 down 掉 (shutdown abort), cache 中的 sequence 就会丢失。所以可以在 create sequence 的时候用 nocache 防止这种情况

不能改变当前值, 但是可以改变增量

```
Alter sequence myseq increment by 3;
```

17.4、同义词

在任何一个用户下, 都可以直接访问 dual, 而不需要加上前缀的用户名如: scott.emp

```
Select * from dual;
```

为什么? 因为同义词的存在

Dual 其实是 sys 用户下的一张表

```
select table_name from user_tables where lower(table_name) = 'dual';
```

作用:

很方便的操作不同用户下的对象

能使两个应用程序使用不同的名字指向同一张表

使用不同的用户指向同一张表的。

Create synonym dept for soctt.dept; (这样创建的同义词是私有的, 只有创建者才能用)

```
Drop synonym dept;
```

Create public synonym dept for soctt.dept; (这样创建的同义词才是公有的)

```
Drop public synonym dept;
```

17.5、练习

- 1: 创建一个包含 1982 年 3 月 31 日以后入职的所有雇员的视图
- 2: 创建一个包含佣金高于其薪金的雇员视图
- 3: 创建一个包含所有雇员的雇员编号、雇员名称、部门名称和薪金的视图
- 4: 创建一个有手下的雇员的视图，包括雇员编号，雇员名称
- 5: 创建一个包含各种工作的薪金总和的视图
- 8: 创建一个序列，从 50 开始，每次增加 10
- 9: 用上面的序列给 dept 表加一条数据
- 9: 为 emp 表创建一个同义词 emp1
- 10: 在上面创建的同义词中插入值，并观察对基表的影响
- 11: 在 emp 表的 empno 字段上创建一个索引，并检查是否可以创建
- 12: 在 emp 表的 sal 上创建一个索引
- 13: 列出您所创建的全部视图、同义词、序列和索引。
- 14: 删除你所创建的任何视图的基表，然后尝试查询视图，并观察查询的输出情况
- 15: 删除您创建的所有视图、同义词、序列和索引。

第十九章 PL SQL

18.1、PL/SQL 块

PL/SQL 块是在 SQL 语言之上发展起来的一种应用，可以集中的处理各种复杂的 SQL 操作。

组成：

DECLARE:

声明部分

BEGIN

编写主题

EXCEPTION

捕获异常

END ;

/

看一个简单的 PL/SQL 块

DECLARE

i NUMBER ;

BEGIN

i := 30 ;

DBMS_OUTPUT.put_line('I 的内容为: '||i);

END ;

/

此时，直接执行程序即可。

执行之后发现没有任何的输出。因为 Oracle 在系统设置中默认设置了输出不显示，如果要显示的话，输入以下命令：

set serveroutput on

DECLARE

i NUMBER ;

BEGIN

i:= 1/0;

EXCEPTION

when ZERO_DIVIDE then

dbms_output.put_line('error');

END ;

/

PL/SQL 块还可以接收用户的输入信息，例如：现在要求用户输入一个雇员编号，之后根据输入的内容进行查询，查询雇员的姓名。

- 用户的输入信息使用“&”完成。

DECLARE

eno NUMBER ;

en VARCHAR2(30) ;

BEGIN

-- 输入的信息保存在 eno 里

eno := &no ;

-- 之后根据 eno 的值，对数据库进行查询操作

SELECT ename INTO en FROM emp WHERE empno=eno ;

DBMS_OUTPUT.put_line('编号为: '||eno||'雇员的姓名为: '||en);

EXCEPTION

WHEN no_data_found THEN

DBMS_OUTPUT.put_line('没有此雇员');

END ;

/

在以上的查询中再进一步：可以根据雇员的编号查出姓名及其领导的姓名和所在的部门，进行显示。

```
DECLARE
    eno emp.empno%TYPE ;
    en emp.ename%TYPE ;
    mn emp.ename%TYPE ;
    dn dept.dname%TYPE ;
    dept dept %rowtype ;
BEGIN
    -- 输入的信息保存在 eno 里
    eno := &no ;
    -- 之后根据 eno 的值，对数据库进行查询操作
    SELECT e.ename,m.ename,d.dname INTO en,mn,dn FROM emp e,dept d,emp m WHERE
    e.empno=7369 AND e.mgr=m.empno AND e.deptno=d.deptno ;
    DBMS_OUTPUT.put_line('编号为: '||eno||'雇员的姓名为: '||en);
    DBMS_OUTPUT.put_line('编号为: '||eno||'雇员的上级姓名为: '||mn);
    DBMS_OUTPUT.put_line('编号为: '||eno||'雇员所在的部门: '||dn);
    DBMS_OUTPUT.put_line(dept.deptno);
EXCEPTION
    WHEN no_data_found THEN
        DBMS_OUTPUT.put_line('没有此雇员');
END ;
/
```

说明：

- emp.empno%TYPE ;：表示以 emp 表中的 empno 字段的类型定义变量
- e.ename,m.ename,d.dname INTO en,mn,dn：一次可以同时放进去多个值
- dept dept %rowtype ; 表示 dept 是一行数据，类似于 Hibernate 的 PO

18.2、Loop 循环 (do...while)

PL/SQL 之中也包含了：循环、分支等条件控制语句

Loop 循环(do...while)

格式：

```
LOOP
    循环的语句 ;
    EXIT WHEN 终止条件 ;
    循环条件必须更改 ;
END LOOP ;
循环输出 1~10。
DECLARE
    cou NUMBER ;
BEGIN
    -- 必须给一个初始值
```

```
cou := 1 ;  
LOOP  
  DBMS_OUTPUT.put_line('cou = '||cou) ;  
  EXIT WHEN cou>10 ;  
  cou := cou + 1 ;  
END LOOP ;  
END ;  
/
```

此循环是先执行一次之后再进行判断

18.3、while 循环

格式:

```
while(判断循环的条件) loop  
  循环的语句 ;  
  循环条件的改变 ;  
End loop ;
```

使用此语句修改上面的程序:

```
DECLARE  
  cou NUMBER ;  
BEGIN  
  -- 必须给一个初始值  
  cou := 1 ;  
  WHILE(cou<10) LOOP  
    DBMS_OUTPUT.put_line('cou = '||cou) ;  
    cou := cou + 1 ;  
  END LOOP ;  
END ;  
/
```

此语句，是先判断，之后如果条件满足则执行，与 while 循环类似。

18.4、for 循环

格式:

```
FOR 变量名称 in 变量的初始值..结束值 LOOP  
  循环语句 ;  
END LOOP ;  
DECLARE  
  cou NUMBER ;  
BEGIN  
  FOR cou IN 1..10 LOOP  
    DBMS_OUTPUT.put_line('cou = '||cou) ;
```

```
END LOOP ;  
END ;  
/
```

18.5、IF 语句

条件判断
格式：
IF 条件 THEN
 满足条件时，执行此语句
END IF ;
DECLARE
 cou NUMBER ;
BEGIN
 cou := 11 ;
 IF cou>10 THEN
 DBMS_OUTPUT.put_line('cou = '||cou) ;
 END IF ;
END ;
/

18.6、IF...ELSE 语句

如果 IF 满足了，则执行，否则执行 ELSE
DECLARE
 cou NUMBER ;
BEGIN
 cou := 1 ;
 IF cou>10 THEN
 DBMS_OUTPUT.put_line('cou = '||cou) ;
 ELSE
 DBMS_OUTPUT.put_line('条件不满足') ;
 END IF ;
END ;
/

18.7、IF...ELSIF...ELSE 语句

```
DECLARE  
    cou NUMBER ;
```

```
BEGIN
cou := 1 ;
IF cou>10 THEN
  DBMS_OUTPUT.put_line('cou = '||cou) ;
ELSIF cou<5 THEN
  DBMS_OUTPUT.put_line('值小于 5') ;
ELSE
  DBMS_OUTPUT.put_line('条件不满足') ;
END IF ;
END ;
/
```

18.8、GOTO 语句

无条件跳转语句

```
DECLARE
eno emp.empno%TYPE ;
sal emp.sal%TYPE ;
BEGIN
eno := &en ;
SELECT sal INTO sal FROM emp WHERE empno=eno ;
IF sal>3500 THEN
  goto po1 ;
ELSIF sal>2000 THEN
  goto po2 ;
ELSE
  goto po3 ;
END IF ;
<<po1>>
  DBMS_OUTPUT.put_line('高工资。。。') ;
<<po2>>
  DBMS_OUTPUT.put_line('中等工资。。') ;
<<po3>>
  DBMS_OUTPUT.put_line('底工资。。。') ;
END ;
/
```


18.9、练习

问题 1：输入一个雇员的编号，如果其工资高于 3500，则显示高工资，工资大于 2000，则显示中等工资，工资小于 2000 的则认为是低等工资。

```
DECLARE
```

```
eno emp.empno%TYPE ;
```

```
sal emp.sal%TYPE ;
```

```
BEGIN
```

```
eno := &en ;
```

```
SELECT sal INTO sal FROM emp WHERE empno=eno ;
```

```
IF sal>3500 THEN
```

```
DBMS_OUTPUT.put_line('高工资。。。') ;
```

```
ELSIF sal>2000 THEN
```

```
DBMS_OUTPUT.put_line('中等工资。。。') ;
```

```
ELSE
```

```
DBMS_OUTPUT.put_line('底工资。。。') ;
```

```
END IF ;
```

```
END ;
```

```
/
```

问题 2: 输入一个雇员编号, 根据它所在的部门涨工资, 规则:

- 10 部门上涨 10%
- 20 部门上涨 20%
- 30 部门上涨 30%

所有部门的上涨工资, 最不能超过 5000, 如果超过 5000, 则工资就为 5000。

DECLARE

eno emp.empno%TYPE ;

dno emp.deptno%TYPE ;

sal emp.sal%TYPE ;

BEGIN

eno := &en ;

SELECT deptno,sal INTO dno,sal FROM emp WHERE empno=eno ;

IF dno=10 THEN

IF sal*1.1>5000 THEN

UPDATE emp SET sal=5000 WHERE empno=eno ;

ELSE

UPDATE emp SET sal=sal*1.1 WHERE empno=eno ;

END IF ;

ELSIF dno=20 THEN

IF sal*1.2>5000 THEN

UPDATE emp SET sal=5000 WHERE empno=eno ;

ELSE

UPDATE emp SET sal=sal*1.2 WHERE empno=eno ;

END IF ;

ELSIF dno=30 THEN

IF sal*1.3>5000 THEN

```
UPDATE emp SET sal=5000 WHERE empno=eno ;

ELSE

UPDATE emp SET sal=sal*1.3 WHERE empno=eno ;

END IF ;

ELSE

null ;

END IF ;

END ;

/

-----

declare
    eno emp.empno%TYPE;
    dno emp.deptno%TYPE;
    esal emp.sal%TYPE;
begin
    eno:=&no;
    select deptno,sal into dno,esal from emp where empno=eno;
    if dno=10 then
        esal := esal + esal*0.1;
    elsif dno=20 then
        esal := esal + esal*0.2;
    elsif dno=30 then
        esal := esal + esal*0.3;
    end if;

    if esal>5000 then
        esal:=5000;
    end if;

    update emp set sal=esal where empno=eno;

end;

/
```

问题 3: 接收部门编号, 显示部门名和地理位置

```
declare
sno dept.deptno%type;
dname dept.dname%type;
loc dept.loc%type;
begin
sno:=&no;
select dname,loc into dname,loc from dept where deptno=sno;
dbms_output.put_line(dname||' '||loc);
exception
when no_data_found then
dbms_output.put_line('not find data');
end;
```

问题 4: 接收雇员号, 显示 该雇员的工资和提成, 没有提成的用 0 替代。(用%type 实现)

```
declare
sno emp.empno%type;
sal emp.sal%type;
comm emp.comm%type;
begin
sno:=&no;
select sal,nvl(comm,0) into sal,comm from emp where empno=sno;
dbms_output.put_line(sal||' '||comm);
exception
when no_data_found then
dbms_output.put_line('not find data');
end;
```

问题 5: 接收雇员号, 显示 该雇员的所有信息, 没有提成的用 0 替代。(用%rowtype 实现)

```
declare
sno emp.empno%type;
e1 emp%rowtype;
begin
sno:=&no;
select * into e1 from emp where empno=sno;
dbms_output.put_line(e1.empno||' '||e1.ename||' '||e1.job||' '||e1.mgr||' '||e1.hiredate||' '||e1.sal||'
'||nvl(e1.comm,0)||' '||e1.deptno);
exception
when no_data_found then
dbms_output.put_line('not find date');
```

end;

问题 6: 接收一个雇员名, 判断他的 job, 根据 job 不同, 为他增加相应的 sal (用 if-elsif 实现)

```
declare
sname emp.ename%type;
ejob emp.job%type;
esal emp.sal%type;
begin
sname:=&no;
select job,sal into ejob,esal from emp where ename=sname;
if(ejob='ANALYST') then
    esal:=esal+esal*0.1;
elsif(ejob='CLERK') then
    esal:=esal+esal*0.2;
elsif(ejob='MANAGER') then
    esal:=esal+esal*0.2;
elsif(ejob='PRESIDENT') then
    esal:=esal+esal*0.2;
elsif(ejob='SALESMAN') then
    esal:=esal+esal*0.2;
end if;
update emp set sal=esal;
exception
WHEN NO_data_found then
dbms_output.put_line('not find data');
end;
```

问题 7: 用 loop 循环结构, 为 dept 表增加 50-90 这些部门

```
declare
no number;
begin
no :=50;
while (no <= 90) loop
insert into dept(deptno) values(no);
no := no+10;
end loop;
end;
```

问题 8: 接收一个雇员名, 显示该雇员的所有内容, (用%rowtype 实现), 当没有这个雇员时(no_data_found),

用异常来显示错误提示

```
declare
name emp.ename%type;
e1 emp%rowtype;
begin
name:=&name;
select * into e1 from emp where ename=name;
dbms_output.put_line(e1.empno||' '||e1.job||' '||e1.mgr||' '||e1.hiredate||' '||e1.sal||' '||e1.comm||'
'||e1.deptno);
exception
when no_data_found then
dbms_output.put_line('没有这个雇员');
end;
/
```

问题 9: 编写一个 PL/SQL 程序块以计算某个雇员的年度薪水总额

```
declare
name emp.ename%type;
sal number;
begin
name := &name;
select (sal*12) into sal from emp where ename = name;
dbms_output.put_line('薪水'||sal);
exception
when no_data_found then
dbms_output.put_line('没有这个雇员');
end;
/
```

问题 10: 编写一个 PL/SQL 程序块以向 emp 表添加 10 新雇员编号 (7901-7910)

```
declare
eno emp.empno%type;
begin
eno := 7911;
loop
insert into emp(empno) values(eno);
exit when eno = 7920;
```

```
eno := eno+1;  
end loop;  
end;  
/
```

问题 11：接受 2 个数相除，并显示结果，如果除数为 0，则显示错误提示；

```
declare  
a number;  
b number;  
begin  
a := &a;  
b := &b;  
dbms_output.put_line(a||'除以'||b||'='||a/b);  
exception  
when zero_divide then  
dbms_output.put_line('被除数不能为 0! ');  
end;  
/
```

第二十章 游标、函数

19.1、游标

游标是一种 PL/SQL 控制结构；可以对 SQL 语句的处理进行显示控制，便于对表的行数据逐条进行处理。

游标并不是一个数据库对象，只是存留在内存中

操作步骤：

- 声明游标
- 打开游标
- 取出结果，此时的结果取出的是一行数据
- 关闭游标

到底那种类型可以把一行的数据都装进来

- 此时使用 ROWTYPE 类型，此类型表示可以把一行的数据都装进来。

例如：查询雇员编号为 7369 的信息（肯定是一行信息）。

```
DECLARE  
eno emp.empno%TYPE ;  
empInfo emp%ROWTYPE ;  
BEGIN  
eno := &en ;
```

```
SELECT * INTO empInfo FROM emp WHERE empno=eno ;
DBMS_OUTPUT.put_line('雇员编号: '||empInfo.empno);
DBMS_OUTPUT.put_line('雇员姓名: '||empInfo.ename);
END ;
/
```

使用 for 循环操作游标（比较常用）

```
DECLARE
-- 声明游标
CURSOR mycur IS SELECT * FROM emp where empno=-1;
empInfo emp%ROWTYPE ;
cou NUMBER ;
BEGIN
-- 游标操作使用循环，但是在操作之前必须先将游标打开
FOR empInfo IN mycur LOOP
    cou := mycur%ROWCOUNT ;
    DBMS_OUTPUT.put_line(cou||'雇员编号: '||empInfo.empno);
    DBMS_OUTPUT.put_line(cou||'雇员姓名: '||empInfo.ename);
END LOOP ;
END ;
/
```

编写第一个游标，输出全部的信息。

```
DECLARE
-- 声明游标
CURSOR mycur IS SELECT * FROM emp ; -- List (EmpPo)
empInfo emp%ROWTYPE ;
BEGIN
-- 游标操作使用循环，但是在操作之前必须先将游标打开
OPEN mycur ;
-- 使游标向下一行
FETCH mycur INTO empInfo ;
-- 判断此行是否有数据被发现
WHILE (mycur%FOUND) LOOP
    DBMS_OUTPUT.put_line('雇员编号: '||empInfo.empno);
    DBMS_OUTPUT.put_line('雇员姓名: '||empInfo.ename);
    -- 修改游标，继续向下
    FETCH mycur INTO empInfo ;
END LOOP ;
END ;
/
```

也可以使用另外一种方式循环游标：LOOP...END LOOP;

```
DECLARE
-- 声明游标
```



```
CURSOR mycur IS SELECT * FROM emp ;
empInfo emp%ROWTYPE ;
BEGIN
-- 游标操作使用循环，但是在操作之前必须先将游标打开
OPEN mycur ;
LOOP
-- 使游标向下一行
FETCH mycur INTO empInfo ;
EXIT WHEN mycur%NOTFOUND ;
DBMS_OUTPUT.put_line('雇员编号: '||empInfo.empno) ;
DBMS_OUTPUT.put_line('雇员姓名: '||empInfo.ename) ;
END LOOP ;
END ;
/
```

注意 1:

在打开游标之前最好先判断游标是否已经是打开的。

通过 ISOPEN 判断，格式：游标%ISOPEN

```
IF mycur%ISOPEN THEN
```

```
    null ;
```

```
ELSE
```

```
    OPEN mycur ;
```

```
END IF ;
```

注意 2:

可以使用 ROWCOUNT 对游标所操作的行数进行记录。

```
DECLARE
```

```
-- 声明游标
```

```
CURSOR mycur IS SELECT * FROM emp ;
```

```
empInfo emp%ROWTYPE ;
```

```
cou NUMBER ;
```

```
BEGIN
```

```
-- 游标操作使用循环，但是在操作之前必须先将游标打开
```

```
IF mycur%ISOPEN THEN
```

```
    null ;
```

```
ELSE
```

```
    OPEN mycur ;
```

```
END IF ;
```

```
LOOP
```

```
-- 使游标向下一行
```

```
FETCH mycur INTO empInfo ;
```

```
EXIT WHEN mycur%NOTFOUND ;
```

```
cou := mycur%ROWCOUNT ;
```

```
DBMS_OUTPUT.put_line(cou||'雇员编号: '||empInfo.empno) ;
```

```
DBMS_OUTPUT.put_line(cou||'雇员姓名: '||empInfo.ename) ;
```

```
END LOOP ;
```

```
END ;
```

```
/
```

一次性上涨全部雇员的工资。根据它所在的部门涨工资，规则：

- 10 部门上涨 10%
- 20 部门上涨 20%
- 30 部门上涨 30%

所有部门的上涨工资，最不能超过 5000，如果超过 5000，则工资就为 5000。

```
declare
  cursor mycur is select * from emp;
  empInfo emp%rowtype;
  s emp.sal%type;
begin
  for empInfo in mycur loop
    if empInfo.deptno = 10 then
      s := empInfo.sal*1.1;
    elsif empInfo.deptno = 20 then
      s := empInfo.sal*1.2;
    elsif empInfo.deptno = 30 then
      s := empInfo.sal* 1.3;
    end if;

    if s > 5000 then
      s := 5000;
    end if;

    update emp set sal = s where empno = empInfo.empno;
  end loop;
end;
/
```

19.2、函数

函数就是一个有返回值的过程。

定义一个函数：此函数可以根据雇员的编号查询出雇员的年薪

```
CREATE OR REPLACE FUNCTION myfun(eno emp.empno%TYPE) RETURN NUMBER
AS
  rsal NUMBER ;
BEGIN
  SELECT (sal+nvl(comm,0))*12 INTO rsal FROM emp WHERE empno=eno ;
  RETURN rsal ;
END ;
/
```

直接写 SQL 语句，调用此函数：

```
SELECT myfun(7369) FROM dual ;
```

19.3、练习

- 1: 使用游标 和 loop 循环来显示所有部门的名称
- 2: 使用游标 和 loop 循环来显示所有部门的地理位置（用%found 属性）
- 3:接收用户输入的部门编号，用 for 循环和游标，打印出此部门的所有雇员的所有信息
- 4: 向游标传递一个工种，显示此工种的所有雇员的所有信息
- 5: 用更新游标来为雇员加佣金：
- 6:编写一个 PL/SQL 程序块，对名字以 ‘A’ 或 ‘S’ 开始的所有雇员按他们的基本薪水的 10%给他们加薪
- 7: 编写一个 PL/SQL 程序块，对所有的 salesman 增加佣金 500
- 8: 编写一个 PL/SQL 程序块，以提升 2 个资格最老的职员为高级职员（工作时间越长，资格越老）
- 9: 编写一个 PL/SQL 程序块，对所有雇员按他们的基本薪水的 20%为他们加薪，如果增加的薪水大于 300 就取消加薪

写一个函数 输入一个员工名字，判断该名字在员工表中是否存在。存在返回 1，不存在返回 0

```
create or replace function empfun(en emp.ename%type) return number
as
is_exist number;
begin
select count(*) into is_exist from emp where ename=upper(en);
return is_exist;
end;
/
```

- 1.写一个函数，传入员工编号，返回所在部门名称

```
create or replace function myfun(eno emp.empno%type) return varchar
as
name varchar(30);
begin
select d.dname into name from emp e,dept d where e.deptno = d.deptno and e.empno = eno;
return name;
end;
/
```

- 2.写一个函数，传入时间，返回入职时间比这个时间早的所有员工的平均工资

```
create or replace function getAvgSal(hdate emp.hiredate%type) return number
as
esal number;
```

```
begin
select avg(sal) into esal from emp where hdate>emp.hiredate;
return esal;
end;
/
```

删除一张表重复记录（ID 是自增唯一，重复记录：其他字段都是一样）

非常经典的一道面试题（可能存在很多数据，要求性能比较高）

aa

```
1 louis 20
2 louis 20
3 jimmy 30
4 louis 20
```

```
-----
delete from aa where id not in(select min(id) from aa group by name,age);
```

```
-----
delete test where id in
(select distinct t2.id
 from test t1,test t2
 where t1.id<t2.id and t1.name=t2.name and t1.age=t2.age);
```

```
-----
delete test where id in
(select distinct t1.id
 from test t1,test t2
 where t1.id<t2.id and t1.name=t2.name and t1.age=t2.age);
```

```
-----
create table tmp as select distinct name ,age from aa;
truncate table aa;
insert into aa select * from tmp;
drop table tmp;
```

```
-----
create table tmp as select * from aa where 1=2;
DECLARE
CURSOR mycur IS select * from aa order by id;
```

```
a1 aa%rowtype;
cou number;
begin
for a1 in mycur loop

select count(*) into cou from tmp
where name=a1.name and age=a1.age;
```

```
if cou=0 then
  insert into tmp values(a1.id,a1.name,a1.age);
else
  delete from aa where id = a1.id;
end if;
end loop;
end;
-----
DECLARE
  CURSOR mycur IS select * from aa order by id;

a1 aa%rowtype;
cou number;
begin
  for a1 in mycur loop

    select count(*) into cou from tmp
      where name=a1.name and age=a1.age;

    if cou=0 then
      insert into tmp values(a1.id,a1.name,a1.age);
    end if;
  end loop;

  delete from aa;
  insert into aa select * from tmp;
end;
```

- 1: 创建一个可以向 dept 表中插入一行的过程
- 2: 创建一个过程, 传入雇员名, 返回该雇员的薪水 (薪金+佣金)
- 3: 创建一个过程, 传入雇员号, 和薪水及增长比例 (10%=0.1)。其中薪水为 in out 参数! 更新薪水为新薪水用一个 PL/SQL 程序 块来调用此过程, 其传入到过程中的参数都是用户输入得到的
- 4: 编写一个函数, 以 deptno 为标准, 返回此部门所有雇员的整体薪水
- 5: 编写一个函数, 接收 deptno, 和 name (out), 返回此部门的名称和地址
- 6: 编写一个过程以显示所指定雇员名的雇员部门名和位置
- 7: 编写一个给特殊雇员加薪 10% 的过程, 这之后, 检查如果已经雇佣该雇员超过了 60 个月, 则给他额外加薪 300。

8: 编写一个函数一检查所指定雇员的薪水是否在有效范围内, 不同职位的薪水范围为:

clerk 1500-2500

salesman 2501-2500

analyst 3501-4500

others 4501-n

如果薪水在此范围内, 则显示消息 “salaly is ok!” , 否则, 更新薪水为该范围内的最小值

9: 编写一个函数以显示该雇员在此组织中的工作天数

第二十一章 存储过程

20.1、过程（存储过程）

与过程相比, 存储过程是存在数据库中的一个对象

如果编译错误。可以用 show errors or show errors procedure myproc

现在定义一个简单的过程, 就是打印一个数字

```
CREATE OR REPLACE PROCEDURE myproc
```

```
AS
```

```
  i NUMBER ;
```

```
BEGIN
```

```
  i := 100 ;
```

```
  DBMS_OUTPUT.put_line('i = '||i) ;
```

```
END ;
```

```
/
```

执行过程: exec 过程名字

下面编写一个过程, 要求, 可以传入部门的编号, 部门的名称, 部门的位置, 之后调用此过程就可以完成部门的增加操作。

```
CREATE OR REPLACE PROCEDURE myproc(dno dept.deptno%TYPE,name
dept.dname%TYPE,dl dept.loc%TYPE)
```

```
AS
```

```
  cou NUMBER ;
```

```
BEGIN
```

```
-- 判断插入的部门编号是否存在, 如果存在则不能插入
```

```
SELECT COUNT(deptno) INTO cou FROM dept WHERE deptno=dno ;
```

```
IF cou=0 THEN
```

```
-- 可以增加新的部门
```

```
INSERT INTO dept(deptno,dname,loc) VALUES(dno,name,dl) ;
```

```
DBMS_OUTPUT.put_line('部门插入成功! ');  
ELSE  
DBMS_OUTPUT.put_line('部门已存在, 无法插入! ');  
END IF;  
END;  
/
```

过程的参数类型:

- IN: 值传递, 默认的
- IN OUT: 带值进, 带值出
- OUT: 不带值进, 带值出

IN OUT 类型:

```
CREATE OR REPLACE PROCEDURE myproc(dno IN OUT dept.deptno%TYPE,name  
dept.dname%TYPE,dl dept.loc%TYPE)  
AS  
cou NUMBER;  
BEGIN  
-- 判断插入的部门编号是否存在, 如果存在则不能插入  
SELECT COUNT(deptno) INTO cou FROM dept WHERE deptno=dno;  
IF cou=0 THEN  
-- 可以增加新的部门  
INSERT INTO dept(deptno,dname,loc) VALUES(dno,name,dl);  
DBMS_OUTPUT.put_line('部门插入成功! ');  
-- 修改 dno 的值  
dno := 1;  
ELSE  
DBMS_OUTPUT.put_line('部门已存在, 无法插入! ');  
dno := -1;  
END IF;  
END;  
/
```

编写 PL/SQL 块验证过程:

```
DECLARE  
deptno dept.deptno%TYPE;  
BEGIN  
deptno := 12;  
myproc(deptno,'开发','南京');  
DBMS_OUTPUT.put_line(deptno);  
END;  
/
```

OUT 类型

不带任何值进, 只把值带出来。

```
CREATE OR REPLACE PROCEDURE myproc(dno OUT dept.deptno%TYPE)  
AS  
I number
```

```
BEGIN
  I:= dno;
END ;
/
执行上面的存储过程
DECLARE
  deptno dept.deptno%TYPE ;
BEGIN
  deptno :=12
  myproc(deptno) ;
  DBMS_OUTPUT.put_line(deptno) ;
END ;
/
```

20.2、练习

下面编写一个存储过程，要求，可以传入部门的编号，部门的名称，部门的位置，之后调用此过程就可以完成部门的增加操作。

```
create or replace procedure myproc(dno dept.deptno%type,
dn dept.dname%type,dl dept.loc%type)
as
  cou number;
begin
  select count(*) into cou from dept where deptno = dno;
  if cou=0 then
    insert into dept values (dno,dn,dl);
    dbms_output.put_line('增加部门成功');
  else
    dbms_output.put_line('部门已存在');
  end if;
end;
/
```

1. 创建三张表 dept10,dept20,dept30，表结构和 dept 一致（不拷贝数据）

```
create table dept10 as select * from dept where 1=2;
create table dept20 as select * from dept where 1=2;
create table dept30 as select * from dept where 1=2;
```

2. 编写一个存储过程 mypro,

- i. 把 dept 表中 depto=10 的数据，存到 dept10,
- ii. 把 dept 表中 depto=20 的数据，存到 dept20
- iii. 把 dept 表中 depto=30 的数据，存到 dept30
- iv. 执行该存储过程


```
create or replace procedure myproc
```

```
as
```

```
begin
```

```
insert into dept10 select * from dept where deptno=10;
```

```
insert into dept20 select * from dept where deptno=20;
```

```
insert into dept30 select * from dept where deptno=30;
```

```
end;
```

```
/
```

```
create or replace procedure mypro
```

```
as
```

```
cursor mycur is select * from dept;
```

```
empInfo emp%ROWTYPE ;
```

```
begin
```

```
for empInfo in mycur loop
```

```
if empInfo.deptno = 10 then
```

```
insert into dept10 values(empInfo.deptno, empInfo.dname, empInfo.loc);
```

```
elsif empInfo.deptno = 20 then
```

```
insert into dept20 values(empInfo.deptno, empInfo.dname, empInfo.loc);
```

```
elsif empInfo.deptno = 30 then
```

```
insert into dept30 values(empInfo.deptno, empInfo.dname, empInfo.loc);
```

```
end if;
```

```
end loop;
```

```
end;
```

```
exec myproc;
```

3. 删除 mypro 存储过程

```
drop procedure myproc;
```

4. 写一个存储过程 (给一个用户名, 判断该用户名是否存在)

```
create or replace procedure findName(name emp.ename%type,en out number)
```

```
as i number;
```

```
begin
```

```
select count(*) into i from emp where ename=name;
```

```
if i=1 then
```

```
en:=i;
```

```
dbms_output.put_line('用户存在');
```

```
else
```

```
en:=0;
```

```
dbms_output.put_line('用户不存在');
```

```
end if;
```

```
end;
```

```
/
```

5. 执行该存储过程

```
DECLARE
deptno dept.deptno%TYPE ;
BEGIN
findName(upper('aaa'),deptno) ;
DBMS_OUTPUT.put_line(deptno) ;
END ;
/
```

6. 编写一个存储过程，批量插入 1000 条数据（只插入 ID 为奇数的数据）

```
create table test(i number(10));

create or replace procedure add1
as
i number(10);
begin
for i in 1..1000 loop
if mod(i,2) = 1 then
insert into test values(i);
end if;
end loop;
end;
```

第二十二章 触发器

存放在数据库中，并被隐含执行的存储过程。在 Oracle8i 之前，只允许给予表或者视图的 DML 的操作，而从 Oracle8i 开始，不仅可以支持 DML 触发器，也允许给予系统事件和 DDL 的操作

21.1、语句触发器

Before 语句触发器

例如：禁止工作人员在休息日改变雇员信息

```
create or replace trigger tr_src_emp
before insert or update or delete on emp
```

```
begin
if to_char(sysdate,'DY','nls_date_language=AMERICAN') in( 'SAT','SUN') then
raise_application_error(-20001,'can't modify user information in weekend');
end if;
end;
/
```

使用条件谓语

```
create or replace trigger tr_src_emp
before insert or update or delete on emp
begin
if to_char(sysdate,'DY') in( '星期六','星期天') then
case
when inserting then
raise_application_error(-20001,'fail to insert');
when updating then
raise_application_error(-20001,'fail to update');
when deleting then
raise_application_error(-20001,'fail to delete');
end case;
end if;
end;
/
```

after 语句触发器

例如：为了统计在 EMP 表上的增、删、改的次数。先建一张表

Create table audit_table(

Name varchar2(20),ins int,upd int,del int,starttime date,endtime date);

然后建立触发器

```
Create or replace trigger tr_audit_emp
After insert or update or delete on emp
Declare
v_temp int;
Begin
Select count(*) into v_temp from audit_table
Where name='EMP';
If v_temp=0 then
Insert into audit_table values('EMP',0,0,0,sysdate,null);
End if;
```

```
Case
When inserting then
Update audit_table set ins=ins+1,endtime=sysdate where name='EMP';
When updating then
Update audit_table set upd=upd+1,endtime=sysdate where name='EMP';
When deleting then
Update audit_table set del= del +1,endtime=sysdate where name='EMP';
End case;
End;
/
```

21.2、行触发器

执行 DML 操作时，每作用一行就触发一次触发器。

Bofre 行触发器

例如：确保员工工资不能低于原有工资

```
Create or replace trigger tr_emp_sal
before update of sal on emp
for each row
begin
if :new.sal<:old.sal then
raise_application_error(-20010,'sal should not be less');
end if;
end;
/
```

after 行触发器

例如：统计员工工资变化

```
Create table audit_emp_change(
Name varchar2(10),
Oldsal number(6,2),
Newsal number(6,2),
Time date);
```

```
Create or replace trigger tr_sal_sal
after update of sal on emp
for each row
declare
v_temp int;
```

```
begin
select count(*) into v_temp from audit_emp_change where name=:old.ename;
if v_temp=0 then
insert into audit_emp_change values(:old.ename,:old.sal,:new.sal,sysdate);
else
update audit_emp_change set oldsal=:old.sal,newsal=:new.sal,time=sysdate
where name=:old.ename;
end if;
end;
/
```

限制行触发器

```
Create or replace trigger tr_sal_sal
after update of sal on emp
for each row
when (old.job='SALESMAN')
declare
v_temp int;
begin
select count(*) into v_temp from audit_emp_change where name=:old.ename;
if v_temp=0 then
insert into audit_emp_change values(:old.ename,:old.sal,:new.sal,sysdate);
else
update audit_emp_change set oldsal=:old.sal,newsal=:new.sal,time=sysdate
where name=:old.ename;
end if;
end;
/
```

21.3、注意事项

编写 DML 触发器的时，触发器代码不能从触发器所对应的基表中读取数据。

例如：如果要基于 EMP 表建立触发器。那么该触发器的执行代码不能包含对 EMP 表的查询操作。

```
Create or replace trigger tr_emp_sal
Before update of sal on emp
For each row
declare
Maxsal number(6,2);
Begin
```

```
Select max(sal) into maxsal from emp;  
If :new.sal>maxsal then  
Raise_application_error(-21000,'error');  
End if;  
End;  
/
```

创建的时候不会报错。但是一旦执行就报错了

```
update emp set sal=sal*1.1 where deptno=30
```

21.4、触发器的主要用途

控制数据安全

例如：在非工作时间不能对 EMP 表做操作

```
create or replace trigger tr_emp_time  
before insert or update or delete on emp  
begin  
if to_char(sysdate,'HH24') not between '9' AND '17' THEN  
raise_application_error(-20101,'not work time');  
end if;  
end;  
/
```

实现数据统计

例如：上面提到的记载员工的工资变化等

实现数据的完整性

例如：如果只是限制员工的工资不能低于 800，可以选用 check 约束

```
Alter table emp add constraint ck_sal check(sal>=800);
```

但如果是限定新工资不能低于其原来工资，也不能高于 20%。则通过约束是无法实现的。这时可通过触发器来实现

```
Create or replace trigger tr_check_sal  
Before update of sal on emp  
For each row  
When(new.sal<old.sal or new.sal>1.2*old.sal)  
Begin
```

```
Raise_application_error(-20931,'ddd');  
End;  
/
```

实现参照完整性

约束可实现级联删除，却不能实现级联更新
可通过触发器实现

```
Create or replace trigger tr_update_cascade  
after update of deptno on dept  
for each row  
begin  
update emp set deptno=:new.deptno where deptno=:old.deptno;  
end;  
/
```

21.5、系统事件触发器

系统事件触发器是指基于 Oracle 系统事件（例如 LOGON 和 STARTUP）所建立的触发器。通过使用系统事件触发器，提供了跟踪系统或数据库变化的机制。下面介绍一些常用的系统事件属性函数，以及建立各种事件触发器的方法。

1. 常用事件属性函数

建立系统事件触发器时，应用开发人员经常需要使用事件属性函数。常用的事件属性函数如下：

- ora_client_ip_address：用于返回客户端的 IP 地址。
- ora_database_name：用于返回当前数据库名。
- ora_des_encrypted_password：用于返回 DES 加密后的用户口令。
- ora_dict_obj_name：用于返回 DDL 操作所对应的数据库对象名。
- ora_dict_obj_name_list(name_list OUT ora_name_list_t)：用于返回在事件中被修改的对象名列表。
- ora_dict_obj_owner：用于返回 DDL 操作所对应的对象的所有者名。
- ora_dict_obj_owner_list(owner_list OUT ora_name_list_t)：用于返回在事件中被修改对象的所有者列表。
- ora_dict_obj_type：用于返回 DDL 操作所对应的数据库对象的类型。
- ora_grantee(user_list OUT ora_name_list_t)：用于返回授权事件的授权者。
- ora_instance_num：用于返回例程号。
- ora_is_alter_column(column_name IN VARCHAR2)：用于检测特定列是否被修改。
- ora_is_creating_nested_table：用于检测是否正在建立嵌套表。
- ora_is_drop_column(column_name IN VARCHAR2)：用于检测特定列是否被删除。
- ora_is_servererror(error_number)：用于检测是否返回了特定 Oracle 错误。

- ora_login_user: 用于返回登录用户名。
- ora_sysevent: 用于返回触发触发器的系统事件名。

案例启动和关闭触发器

记载例程启动和关闭的事件和时间，首先建立事件表 event_table。示例如下：

```
SQL> conn sys/oracle as sysdba
SQL> create table event_table(event varchar2(30),time date);
```

在建立了事件表 event_table 之后，就可以在触发器中引用该表了。注意，例程启动触发器和例程关闭触发器只有特权用户才能建立，并且例程启动触发器只能使用 AFTER 关键字，而例程关闭触发器只能使用 BEFORE 关键字。示例如下：

```
CREATE OR REPLACE TRIGGER tr_startup
AFTER STARTUP ON DATABASE
BEGIN
    INSERT INTO event_table VALUES(ora_sysevent,SYSDATE);
END;
/
CREATE OR REPLACE TRIGGER tr_shutdown
BEFORE SHUTDOWN ON DATABASE
BEGIN
    INSERT INTO event_table VALUES(ora_sysevent,SYSDATE);
END;
/
```

在建立了触发器 tr_startup 之后，当打开数据库之后，会执行该触发器的相应代码；在建立了触发器 tr_shutdown 之后，在关闭例程之前，会执行该触发器的相应代码，但 SHUTDOWN ABORT 命令不会触发该触发器。示例如下：

```
SQL> SHUTDOWN
SQL> STARTUP
SQL> SELECT event,to_char(time,'YYYY/MM/DD HH24:MI') time
   2 FROM event_table;
EVENT                                TIME
-----
SHUTDOWN                            2003/12/18 09:25
STARTUP                             2003/12/18 09:26
```

登录和退出触发器

3. 建立登录和退出触发器

为了记载用户登录和退出事件，可以分别建立登录和退出触发器。为了记载登录用户和退出用户的名称、时间和 IP 地址，应该首先建立专门存放登录和退出的信息表 LOG_TABLE。示例如下：

```
SQL> conn sys/oracle as sysdba
SQL> CREATE TABLE log_table(
   2  username VARCHAR2(20),logon_time DATE,
   3  logoff_time DATE,address VARCHAR2(20));
```

在建立了 LOG_TABLE 表之后，就可以在触发器中引用该表了。注意，登录触发器和退出触发器一定要以特权用户身份建立，并且登录触发器只能使用 AFTER 关键字，而退出触发

器只能使用 BEFORE 关键字。示例如下：

```
CREATE OR REPLACE TRIGGER tr_logon
AFTER LOGON ON DATABASE
BEGIN
    INSERT INTO log_table (username,logon_time,address)
        VALUES(ora_login_user,SYSDATE,ora_client_ip_address);
END;
/
CREATE OR REPLACE TRIGGER tr_logoff
BEFORE LOGOFF ON DATABASE
BEGIN
    INSERT INTO log_table (username,logoff_time,address)
        VALUES(ora_login_user,SYSDATE,ora_client_ip_address);
END;
/
```

在建立了触发器 tr_logon 之后，当用户登录到数据库之后，会执行其触发器代码；在建立了触发器 tr_logoff 之后，当用户断开数据库连接之前，会执行其触发器代码。示例如下：

```
SQL> conn scott/tiger@orcl
SQL> conn system/manager@orcl
SQL> conn sys/oracle@orcl as sysdba
SQL> select * from log_table;
USERNAME          LOGON_TIME          LOGOFF_TIM          ADDRESS
-----
SCOTT
SYSTEM            18-12 月-03          18-12 月-03          127.0.0.1
SYSTEM
SYS               18-12 月-03          18-12 月-03          127.0.0.1
SYS
SCOTT             18-12 月-03          18-12 月-03          127.0.0.1
```

DDL 触发器

为了记载系统所发生的 DDL 事件（CREATE，ALTER，DROP 等），可以建立 DDL 触发器。为了记载 DDL 事件信息，应该建立专门的表，以便存放 DDL 事件信息。示例如下：

```
SQL> conn sys/oracle as sysdba
SQL> CREATE TABLE event_ddl(
2   event VARCHAR2(20),username VARCHAR2(10),
3   owner VARCHAR2(10),objname VARCHAR2(20),
4   objtype VARCHAR2(10),time DATE);
```

在建立了表 event_ddl 之后，就可以在触发器中引用该表了。为了记载 DDL 事件，应该建立 DDL 触发器。注意，当建立 DDL 触发器时，必须要使用 AFTER 关键字。示例如下：

```
CREATE OR REPLACE TRIGGER tr_ddl
AFTER DDL ON scott.schema
BEGIN
    INSERT INTO event_ddl VALUES(
        ora_sysevent,ora_login_user,ora_dict_obj_owner,
        ora_dict_obj_name,ora_dict_obj_type,SYSDATE);
```

```
END;
```

```
/
```

在建立了触发器 `tr_ddl` 之后，如果在 SCOTT 方案对象上执行了 DDL 操作，则会将该信息记载到表 `event_ddl` 中。示例如下：

```
SQL> conn scott/tiger
SQL> CREATE TABLE temp(col1 int);
SQL> DROP TABLE temp;
SQL> select event,owner,objname from event_ddl;
EVENT          OWNER      OBJNAME
-----
CREATE          SCOTT      TEMP
DROP            SCOTT      TEMP
```

21.6、管理触发器

1. 显示触发器信息

建立触发器时，Oracle 会将触发器信息写入到数据字典中，通过查询数据字典视图 `USER_TRIGGERS`，可以显示当前用户所包含的所有触发器信息。示例如下：

```
SQL> conn scott/tiger
SQL> SELECT trigger_name,status FROM user_triggers
2 WHERE table_name='EMP';
TRIGGER_NAME          STATUS
-----
TR_AUDIT_EMP           ENABLED
TR_CHECK_SAL           ENABLED
TR_EMP_SAL             ENABLED
TR_SAL_CHANGE          ENABLED
TR_SEC_EMP             ENABLED
```

2. 禁止触发器

禁止触发器是指使触发器临时失效。当触发器处于 `ENABLE` 状态时，如果在表上执行 DML 操作，则就会触发相应的触发器。如果基于 `INSERT` 操作建立了触发器，当使用 `SQL*Loader` 装载大批量数据时会触发触发器。为了加快数据装载速度，应该在装载数据之前禁止触发器。方法如下：

```
SQL> ALTER TRIGGER tr_check_sal DISABLE;
```

3. 激活触发器

激活触发器是指使触发器重新生效。当使用 `SQL*Loader` 装载了数据之后，为了使被禁止的触发器生效，应该激活触发器。方法如下：

```
SQL> ALTER TRIGGER tr_check_sal ENABLE;
```

4. 禁止或激活表的所有触发器

如果在表上同时存在多个触发器，那么使用 `ALTER TABLE` 命令可以一次禁止或激活所有触发器，示例如下：

```
SQL> ALTER TABLE emp DISABLE ALL TRIGGERS;
SQL> ALTER TABLE emp ENABLE ALL TRIGGERS;
```

5. 重新编译触发器

当使用 ALTER TABLE 命令修改表结构（例如增加列、删除列）时，会使得其触发器转变为 INVALID 状态。在这种情况下，为了使得触发器继续生效，需要重新编译触发器。示例如下：

```
SQL> ALTER TRIGGER tr_check_sal COMPILE;
```

6. 删除触发器

当触发器不再需要时，可以使用 DROP TRIGGER 命令删除触发器。注意，在表上的触发器越多，对于 DML 操作的性能影响也越大，所以一定要适度使用触发器。删除触发器的示例如下：

```
SQL> DROP TRIGGER tr_check_sal;
```

第二十三章 事务（数据库系统概论）

22.1、事务控制

在 oracle 中每个连接都会产生一个 Session,一个 session 对数据库的修改，不会立刻反映到数据库的真实数据上，是允许回滚的。只有当提交了，才变成持久数据了

可能出现死锁的情况

原子性（Atomicity）

一致性（Consistency）

隔离性（Isolation ）

持久性（Durability ）

原子性和一致性的差别？

原子性：只一个事务中，包含若干个数据操作，这些操作是一个整体，要么一起完成，要么一起不完成，不能只完成其中的一部分。比

如你去银行转账，从一个账户转账到另一个账户，这是一个完整的事务，包括两个操作，从你第一个账户读数，增加到第二个账户，并

减去第一个账户中的钱，如果这些操作有一个失败了，整个事务都必须还原成最开始的状态。

一致性：是指数据库从一个完整的状态跳到另一个完整的状态，是用于保护数据库的完整性的。比如你修改数据库的某个外键值，如果

没有和相应的主键对应，就违反了数据库的一致性。另外，还有读一致性，如：你刚写入一个数到数据库中，但还没有提交，这时候有

人要读这个数，就涉及完整性问题，要保证读取的数据在整个数据库中是处于和其他数据一致的一个状态。

第二十四章 用户管理

注意:

如果要创建用户只能在管理员下完成:

· 超级管理员: sys/sys

· 普通管理员: system/system

用户: s c o t t / t i g e r

23.1、创建用户

CREATE USERS 用户名 IDENTIFIED BY 密码。

| - create user test identified by admin

| - 用户名: test

| - 密码: admin

23.2、删除用户

Drop user test;

如果该用户下面已经存在表等一些数据库对象。则必须用级联删除

Drop user test cascade;

23.3、创建 Session 权限

一般在数据库中, 一个用户的连接称为建立一个 session, 如果一个新的用户想访问数据库, 则必须授予创建 session 的权限 —— 用户授权

GRANT 权限 TO 用户。

给 test 用户以创建 session 的权限: GRANT create session TO test ;

以上用户虽然可以连接了, 但是不能有任何的操作(比如创建表)

23.4、用户角色

角色是权限的集合

在 Oracle 中提供了两个角色, 可以直接将这两个角色给用户:

·CONNECT 角色:

·RESOURCE 角色:

现在将这两个角色给 test 用户

GRANT CONNECT,RESOURCE TO test ;

这时有创建表的权限了，但是还是不能创建表。主要是因为该用户还没有对表空间操作的权限

Q:ORA-01950: no privileges on Tablespace 'USERS'"

A: GRANT UNLIMITED TABLESPACE TO youruser;

1. CONNECT, RESOURCE, DBA

这些预定义角色主要是为了向后兼容。其主要是用于数据库管理。oracle 建议用户自己设计数据库管理和安全的权限规划，而不要简单的使用这些预定角色。将来的版本中这些角色可能不会作为预定义角色。

2. DELETE_CATALOG_ROLE, EXECUTE_CATALOG_ROLE, SELECT_CATALOG_ROLE
这些角色主要用于访问数据字典视图和包。

3. EXP_FULL_DATABASE, IMP_FULL_DATABASE
这两个角色用于数据导入导出工具的使用。

23.5、锁住一个用户

```
·ALTER USER 用户名 ACCOUNT LOCK|UNLOCK  
|- ALTER USER test ACCOUNT LOCK ;  
|- ALTER USER test ACCOUNT UNLOCK ;
```

23.6、密码失效

提示用户第一次连接的时候需要修改密码，让用户的密码到期

```
|- ALTER USER test PASSWORD expire ;
```

23.7、对象授权

以上的所有操作只针对于 test 用户。如果 test 要访问其他用户呢？例如，访问 emp 表
此时如果要想让其可以访问，则必须把 scott 用户下的 emp 表的查询权限给 test。

GRANT 权限（select、update、insert、delete） ON schema.table TO 用户

```
|- GRANT select ON scott.emp TO test ;  
|- Grant all on scott.emp to test; --将表相关的所有权限付给 test  
|- Grant update(ename) on emp to test; 可以控制到列(还有 insert)
```

23.8、权限回收

REVOKE 权限 ON schema.table FROM 用户
|- REVOKE select ON scott.emp FROM test ;

23.9、查看权限

```
select * from user_sys_privs;
```

23.10、权限传递

Grant create session to test with admin option;(可以就可以实现权限传递)

Q: 如果权限 sys->test->test1 ，这时断掉 test 的权限， test1 还会有权限吗？

A:在 oracle9i 是，答案是还会有。

23.11、角色

角色就是一堆权限的集合

Create role myrole;

Grant create table to myrole;

Drop role myrole; 删除角色

23.12、某真实项目实例（BaoBao121）

Linux 下的操作（Windows 下类似）

1:建立相应的目录并变更其所有者以及组

##建目录

#mkdir /opt/oracle/oradata/baobaodata

##变更所有者

#chown oracle /opt/oracle/oradata/baobaodata/

##变更组

#chgrp oinstall /opt/oracle/oradata/baobaodata/

2:创建表空间

##转到 oracle 用户执行操作，注意要有中划线

#su - oracle

##进入 SQL 操作

#sqlplus /nolog

SQL>connect / as sysdba

##执行创建表空间

SQL>CREATE TABLESPACE baobao121

DATAFILE '/opt/oracle/oradata/baobaodata/baobao121.dbf' SIZE 200M

AUTOEXTEND ON NEXT 50M

EXTENT MANAGEMENT LOCAL

SEGMENT SPACE MANAGEMENT AUTO;

SQL>create temporary tablespace baobao121_temp

tempfile '/opt/oracle/oradata/baobaodata/baobao121_temp.dbf' size 50M

AUTOEXTEND ON NEXT 50M

EXTENT MANAGEMENT LOCAL;

3: 增加相关的用户及权限

##增加用户

SQL>create user baobao121 identified by baobao121

DEFAULT TABLESPACE baobao121

temporary tablespace baobao121_temp;

##增加权限

SQL>grant all privileges to baobao121;

##测试连接

SQL>conn baobao121/baobao121;

SQL>show user;

显示应为 USER is "BAOBAO121"

第二十五章 备份 恢复 SQLLoader

24.1、imp 导入 import

24.2、exp 导出 export

24.3、SQL Loader

在 Oracle 数据库中，我们通常在不同数据库的表间记录进行复制或迁移时会用以下几种方法：

1. A 表的记录导出为一条条分号隔开的 insert 语句，然后执行插入到 B 表中
2. 建立数据库间的 dblink，然后用 create table B as select * from A@dblink where ...，或 insert into B select * from A@dblink where ...
3. exp A 表，再 imp 到 B 表，exp 时可加查询条件
4. 程序实现 select from A ...，然后 insert into B ...，也要分批提交
5. 再就是本篇要说的 Sql Loader(sqlldr) 来导入数据，效果比起逐条 insert 来很明显

第 1 种方法在记录多时是个噩梦，需三五百条的分批提交，否则客户端会死掉，而且导入过程很慢。如果要不产生 REDO 来提高 insert into 的性能，就要下面那样做：

```
alter table B nologging;
```

sql loader 的用法：

1. 只使用一个控制文件，在这个控制文件中包含数据(推荐)
2. 使用一个控制文件(作为模板) 和一个数据文件

Csv 文件如下： (dept1.csv)

```
"  ", "DEPTNO", "DNAME", "LOC"
"1", "10", "ACCOUNTING", "NEW YORK"
"2", "20", "RESEARCH", "DALLAS"
"3", "30", "SALES", "CHICAGO"
"4", "40", "OPERATIONS", "BOSTON"
"5", "50", "sdsaf", "adf"
"6", "12", "aaa", "aaa"
```

Ctl 文件如下： (dept1.ctl)

方式一:

```
Load data
Infile c:\dept1.csv
truncate
Into table dept1
(
Deptno position(1:2),
Dname position(3:5),
Loc position(6:8)
)
```

方式二:

```
OPTIONS (skip=1,rows=128) -- sqlldr 命令显示的选项可以写到这里边来,skip=1 用来跳过数据中的第一行
LOAD DATA
INFILE "c:\dept1.csv" --指定外部数据文件, 可以写多个 INFILE "another_data_file.csv" 指定多个数据文件
--这里还可以使用 BADFILE、DISCARDFILE 来指定坏数据和丢弃数据的文件,
append --操作类型, 用 truncate table 来清除表中原有记录
INTO TABLE dept1 -- 要插入记录的表
Fields terminated by "," -- 数据中每行记录用 "," 分隔
Optionally enclosed by '"' -- 数据中每个字段用 '"' 框起, 比如字段中有 "," 分隔符时
trailing nullcols --表的字段没有对应的值时允许为空
(
virtual_column FILLER, --跳过由 PL/SQL Developer 生成的第一列序号
deptno ,--"dept_seq.nextval", --这一列直接取序列的下一值, 而不用数据中提供的值
dname ,--"Hi '||upper(:dname)",--,还能用 SQL 函数或运算对数据进行加工处理
loc
)
```

说明: 在操作类型 **truncate** 位置可用以下中的一值:

- 1) insert --为缺省方式, 在数据装载开始时要求表为空
- 2) append --在表中追加新记录
- 3) replace --删除旧记录(用 **delete from table** 语句), 替换成新装载的记录
- 4) truncate --删除旧记录(用 **truncate table** 语句), 替换成新装载的记录

执行命令

Sqlldr scott/tiger control=dept1.ctl

看看日志文件, 坏数据文件, 从中可让你更好的理解 **Sql Loader**, 里面有对控制文件的解析、列出每个字段的类型、加载记录的统计、出错原因等信息

第二十六章 数据库设计范式

25.1、第一范式：字段要设计的不可再分

Name 字段 可拆分成 FirstName+LastName

25.2、第二范式：两个表的关系，在第三张关系表中体现

比如学生和课程表。为多对多的关系。这种关系需要在第三张表中体现

25.3、第三范式：多张表中，只存关系，不存具体信息（具体开发中用的最多）

比如： emp,dept

如果一对多用第三张表（关系表）来表示，则会出现问题。（一个员工可能属于多个部门，显然这是不符合现实逻辑的）

25.4、总范式 数据库表关联越少越好，SQL 语句复杂度越低越好

三种范式其实只供参与。数据库设计原则：数据库表关联越少越好，SQL 语句复杂度越低越好。所以有时候，违反了第三范式，但是简化了查询语句。加快的检索速度。

例如：我们开发的某日本项目中，

serviceoffice (serviceofficeid) (1——>N) careuserham (careuid serviceofficeid)
careuserham (careuid serviceofficeid) (1——>N) helperassign (assigned careuid servicedate
serviceofficeid)

这里的 **helperassign** 表中的 **serviceofficeid** 就是冗余字段，不符合第三范式。但是因为加上这个字段，使的检索速度加快不下 **10 倍**

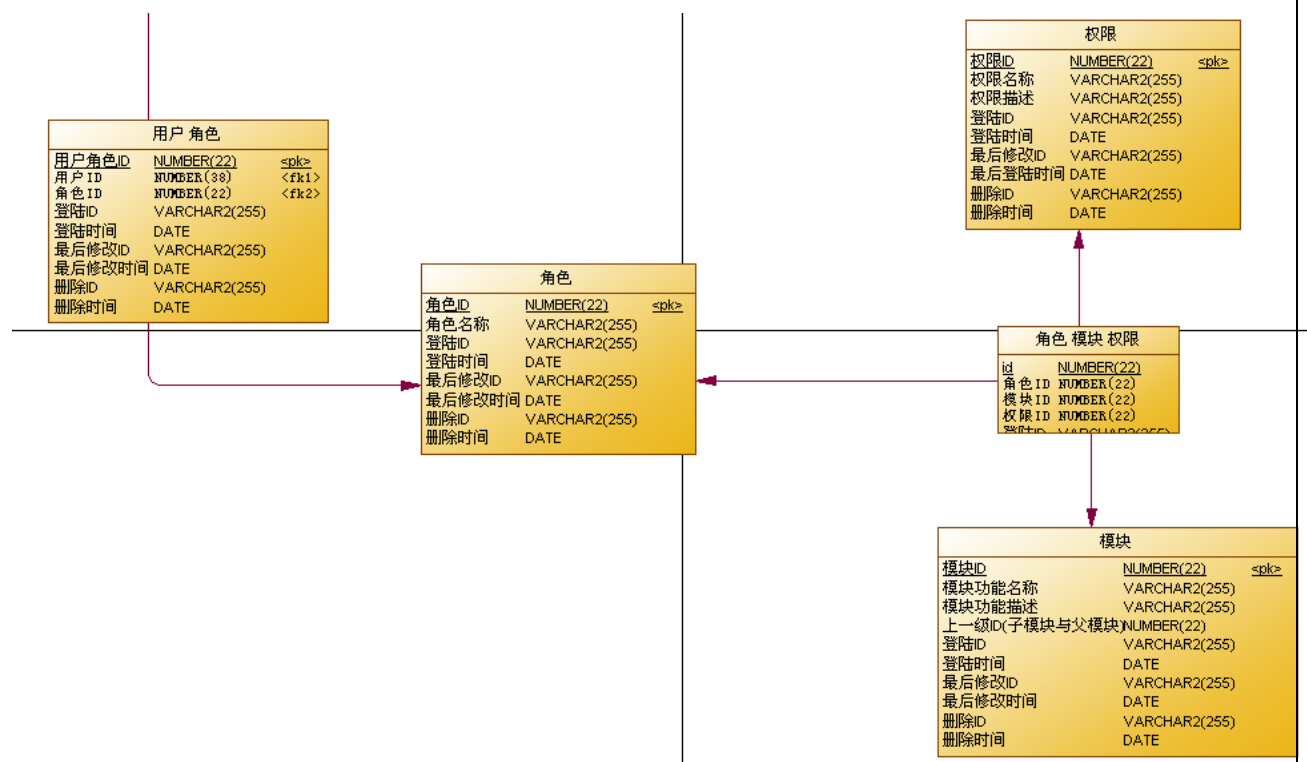
第二十七章 数据库设计工具

26.1、PowerDesigner

26.2、ErWin

26.3、Rose

26.4、权限系统设计

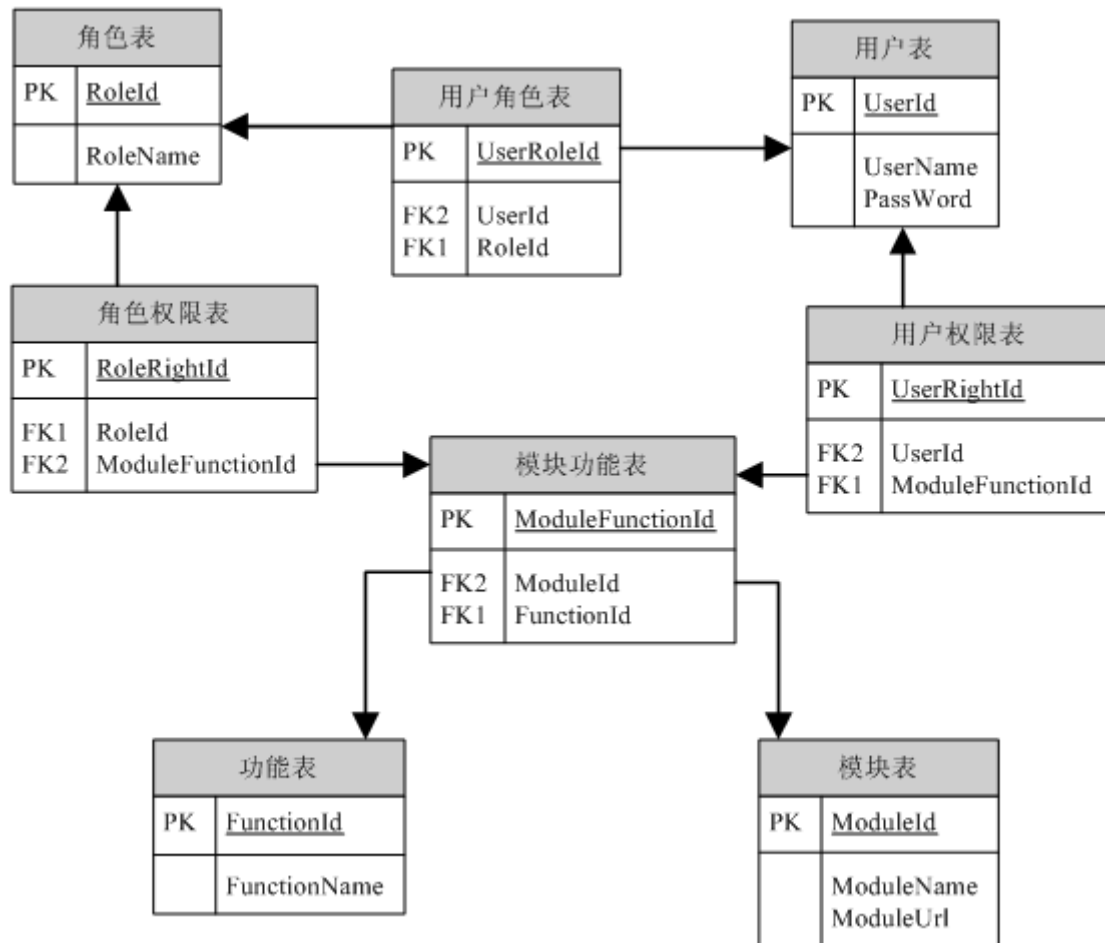


首先，我来介绍一下系统的主要组成部分，因为是权限系统，所以它的组成主要由模块权限、角色权限、用户权限三大部分组成，下面将详细介绍一下三大权限的作用以及它们互相存在的依赖性。

模块权限部份，说白了就是管理系统中的功能模块，而在这些模块中，它们有着各式各样的具体的操作，这些具体的操作权限就是模块权限，例如现新加了一个用户管理的功能模块，在这个模块设计中需要进行浏览、添加、修改、删除、审核、查询等一些具体的操作，在系统运行的过程中，不是所有的用户都会拥有这些权限的，根据需要，就产生了下面的角色权限。

角色权限部分，它就是一个身份，拥有这个身份的用户在系统中能做什么，不能做什么，用户都得依照这个身份，无法过界，这也是一个权限的范围限定，在一个管理系统有很多的用户，我们不能将模块中的权限逐个的分配给用户，现在角色的出现就解决了这个问题，它就像一个权限组，将模块的权限指派给角色，让拥有该角色的用户可以拥有对模块对应的操作权限，然而，一个系统中用户可能会成千上万不等，但我想角色最多不过几十个，将模块权限授权给几十个角色比授权给上万个用户轻松多了，角色虽然是权限组，有限定的作用范围，但是也有会出现意外情况的时候，如果一个用户有操作用户模块的角色，但对它有个特殊的要求，就是不能操作用户模块中的删除或其它功能，还有就是它还有操作另外模块的权限，而它拥有的角色只能访问用户模块，怎么办呢？难道要为这一个用户再建立一个角色吗？我的回答是 No.

现在我介绍我们系统中的最后一个权限，它就是用户权限，我想很多朋友以前在开发管理系统时都用过这种权限模式，虽然这种方法比较原始，但它能解决我们上面所遇到的问题，当然，还需要做一些小小的改进，在角色中，我们只告诉用户能做什么，没有告诉的就不能做，以角色做权限批量判断这点已经够了，而用户单独的权限判断有点特殊，它得跳出角色。它的权限优先于角色的权限，它拥有对某权限允许和禁止操作的功能，例如一个用户的角色权限可以操作用户模块所有的功能，而该用户拥有了该模块某一功能的禁止权限，那该用户不能对这个模块进行该功能操作，反之，用户角色没有授权这个模块功能，而用户权限被授予了这个功能模块的允许权限，那它就能操作该功能，流程图如下



功能表: Function

序号	列名	数据类型	长度	小数位	标识	主键	允许空	默认值	说明
1	FuctionId	int	4	0		是	否		功能Id
2	FunctionName	nvarchar	50	0			是		功能名称

模块表: Module

序号	列名	数据类型	长度	小数位	标识	主键	允许空	默认值	说明
1	ModuleId	int	4	0	是	是	否		模块 Id 主键

2	ModuleName	nvarchar	50	0			是		模块名称
3	ModuleUrl	nvarchar	50	0			是		模块路径

模块功能表: ModuleFuntion

序号	列名	数据类型	长度	小数位	标识	主键	允许空	默认值	说明
1	ModuleFuntionId	int	4	0		是	否		模块功能 Id 主键
2	ModuleId	int	4	0			是		模块 Id
3	FunctionId	int	4	0			是		功能 Id

角色表: Role

序号	列名	数据类型	长度	小数位	标识	主键	允许空	默认值	说明
1	RoleId	int	4	0	是	是	否		角色 Id 主键
2	RoleName	nvarchar	50	0			是		角色名称

角色权限表: RoleRight

序号	列名	数据类型	长度	小数位	标识	主键	允许空	默认值	说明
1	RoleRightId	int	4	0	是	是	否		角色权限 Id 主键
2	RoleId	int	4	0			是		角色 Id
3	ModuleFunctionId	int	4	0			是		模块功能 Id

用户权限表: UserRight

序号	列名	数据类型	长度	小数位	标识	主键	允许空	默认值	说明
1	UserRightId	int	4	0	是	是	否		用户权限 Id 主键
2	UserId	int	4	0			是		用户 Id
3	ModuleFuctionId	int	4	0			是		功能模块 Id

用户角色表: UserRole

序号	列名	数据类型	长度	小数位	标识	主键	允许空	默认值	说明
1	UserRoleId	int	4	0		是	否		用户角色 Id 主键
2	UserId	int	4	0			是		用户 Id
3	RoleId	int	4	0			是		角色 Id

用户表: Users

序号	列名	数据类型	长度	小数位	标识	主键	允许空	默认值	说明
1	UserId	int	4	0	是	是	否		用户 Id 主键 自动递增
2	UserName	nvarchar	50	0			是		用户名
3	PassWord	nvarchar	50	0			是		密码

26.5、订餐系统设计

26.6、工学结合项目系统设计

第二十八章 对象关系数据库系统

27.1、数据库发展历程

层次型数据库

网状型数据库

关系型数据库（主流）(19 世纪 70 年代)

对象关系型数据库（20 世纪 80 年代开始）

定义：ORDBS 是面向对象的数据库模型(Object Oriented Data Model 简称 OO 模型)和关系型数据模型的结合产物

27.2、对象关系型数据库发展方向

- 1、以面向对象的程序设计语言为基础，研究持久化的程序设计语言。支持 OO 模型（Hibernate）
- 2、建立新的面向对象数据库系统 OODBS，支持 OO 数据模型
- 3、以关系数据库和 SQL 为基础，把面向对象技术融入到数据库系统的 ORDBS

从纯粹的数据库系统角度来说，第三种发展较为卓著。它在传统的关系数据库的基础上吸收了 OO 模型的主要思想，同时又保持了关系数据库系统的优点。成功开发了诸如：Postgres, Illusta 等原型系统。本章课程主要以第三种发展方向来阐述

27.3、SQL3

SQL 全名是结构化查询语言（Structured Query Language），是用于数据库中的标准数据查询语言，IBM 公司最早使用在其开发的数据库系统中。1986 年 10 月，美国 ANSI 对 SQL 进行规范后，以此作为关系式数据库管理系统的标准语言（ANSI X3. 135-1986），1987 年得到国际标准组织的支持下成为国际标准

SQL3 是 1999 年发布的新的 SQL 标准。最显著的特点就是：提供了面向对象的扩展克服的原来关系型数据库数据类型单一的缺点。扩展的类型 LOB, Boolean, 集合类型 Array. 用户定义的 Distinct 类型
现在最新的标准是 SQL99。ORACLE 不但对标准的 SQL 完全兼容，而且有自己更为方便的增强 SQL

27.3.1、大对象 Lob (Large Object) 类型

Lob 主要分为 Blob(Binary Large Object), 用于存储音频, 图像数据。Clob(Character Large Object) 主要用于存储长字符串数据

Lob 对象可以想其他数据类型那样被查询, 修改, 插入, 但是必须为 LOB 提供足够大的缓冲区。

应用程序通常并不传输整个 Lob 类型的值, 而是通过 LOB 定位器访问大对象值

27.3.2、Boolean 类型

除了 not,and,or 还增加了 every,any

27.3.3、集合类型 (Array)

```
Create table sales(  
Item_no char(20), //商品号  
Qty integer array[12]; //整数数组。存放 12 个月的销售额  
);
```

插入值的方式:

```
Insert into sales(item_no qty) values('nike10029',array[20,10,20,10,1,12,101,1,1,1,1,1]);
```

查询三月份的销售额大于 100 的商品号

```
Select item_no from sales where qty[3]>100;
```

27.3.4、Distinct 类型

第二十九章 其他数据库

28.1 MYSQL

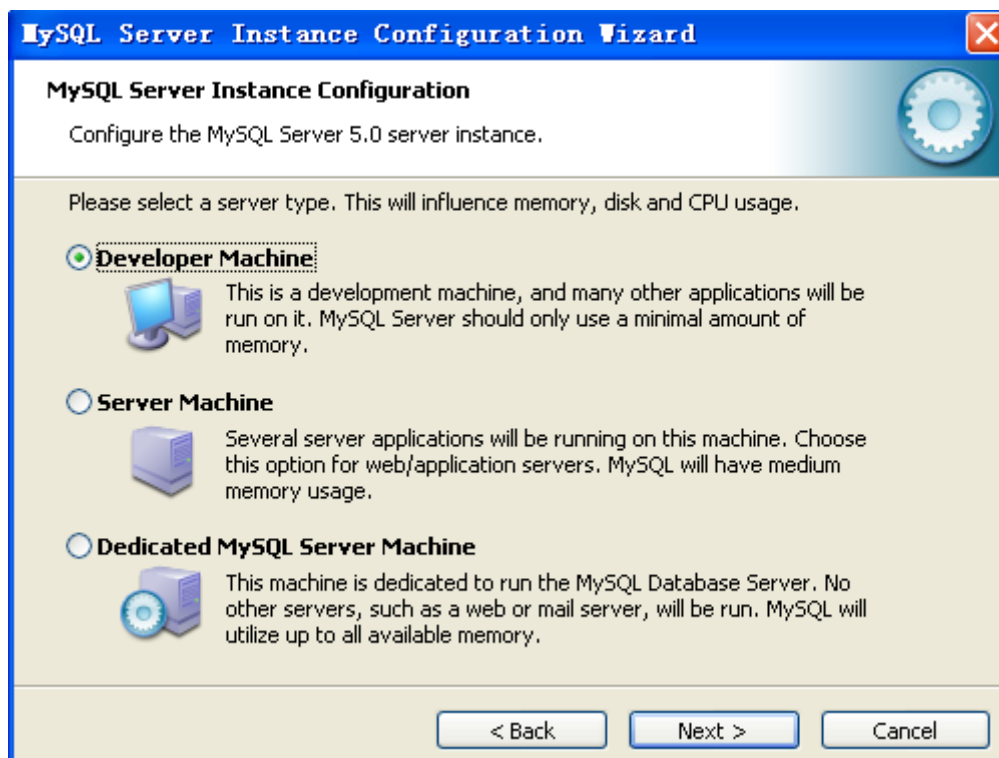
28.1.1 MYSQL 简介

服务器端下载:

下载地址: www.mysql.com

Title: The world's most popular open source database

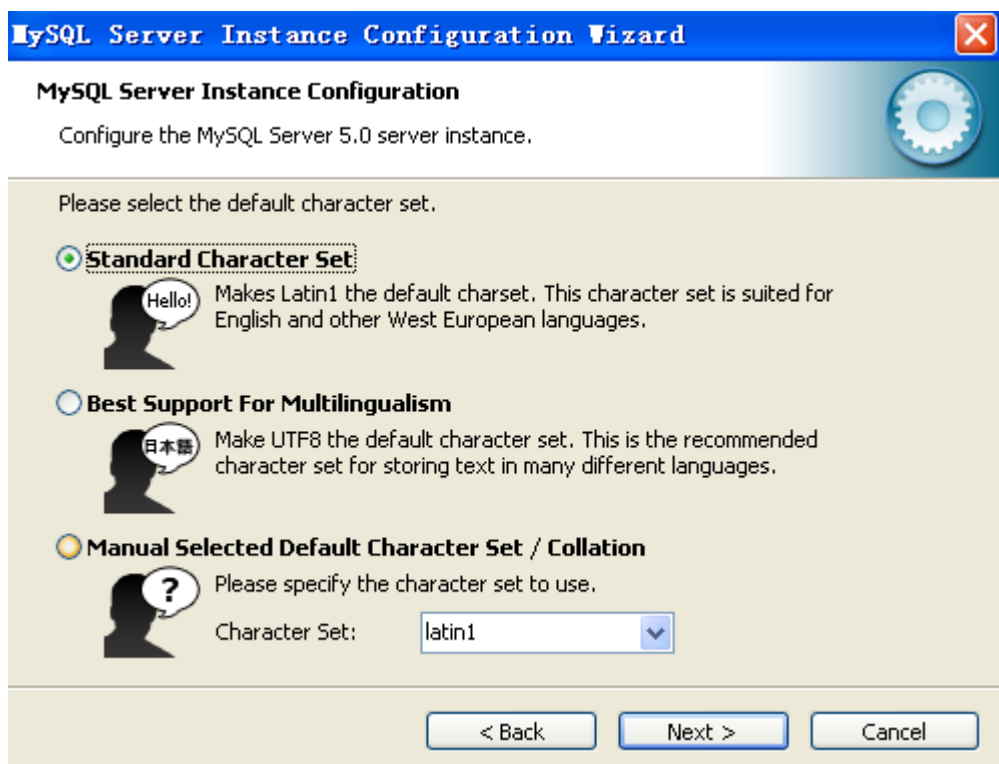
安装的时候,



第一个选项为：把 MYSQL 安装在开发的机器上，只用少量内存

第二个选项为：把 MYSQL 安装在服务器上，将会应用大部分内存

第三个选项为：把 MYSQL 安装在专门的服务器上，这时 MYSQL 将用掉所有可用的内存



编码格式一定要注意。默认开发国内的项目一般选择 GB2312



Enable root access from remote machines 指的是是否可以从远程客户端访问该数据库

28.1.2 客户端简介

Mysql Commang Line client(命令行的客户端)

Navicat for MySQL (推荐)

28.1.3 Mysql 与 Oracle 的区别

<http://majie.javaeye.com/blog/126561>

特别注意：数据类型的区别，一些 SQL 命令的区别，比如:Limit 等

Select * from user limit 2,3 //指的是从 2 条以后开始，找出 3 条

MYSQL 数据库里，分页的实现非常简单。而在 Oracle 中，分页的实现显得异常繁琐

28.2 DB2