

ORACLE 数据库管理

目 录

第 1 章 数据库基础	1
1.1 基本概念	1
1.1.1 数据	1
1.1.2 数据库	1
1.1.3 数据库管理系统	1
1.1.4 数据库系统	1
1.2 关系数据库	1
1.2.1 关系模型	2
1.2.2 关系数据库	2
1.3 常见的数据库对象	3
1.3.1 表	3
1.3.2 索引	3
1.3.3 视图	3
1.3.4 存储过程	3
1.4 范式	4
1.4.1 第一范式	4
1.4.2 第二范式	4
1.4.3 第三范式	4
1.5 DML语言	5
1.5.1 查询数据	5
1.5.2 插入数据	8
1.5.3 修改数据	9
1.5.4 删除数据	9
第 2 章 ORACLE SERVER构成	10
2.1 概述	10
2.2 ORACLE实例	10
2.2.1 ORACLE进程	11
2.2.2 ORACLE内存结构	13

2.3 ORACLE数据库	14
2.3.1 数据库物理结构	15
2.3.2 数据库逻辑结构	16
2.4 模式和模式对象	19
2.4.1 表	19
2.4.2 视图	20
2.4.3 聚集	20
2.4.4 索引	20
2.4.5 程序单元	21
第 3 章 用户管理	22
3.1 创建用户	22
3.2 撤销和改变用户	23
3.3 用户权限	23
3.3.1 系统权限	23
3.3.2 对象权限	26
3.4 角色	27
3.4.1 创建角色	27
3.4.2 授予和取消角色	28
3.4.3 启用角色	28
3.5 查询与权限和角色有关的视图	28
3.6 概要文件	29
3.6.1 创建profile	30
3.6.2 查询用户和profile相关信息的数据字典视图	31
第 4 章 启动和关闭数据库	33
4.1 启动数据库	33
4.1.1 正常启动	33
4.1.2 非加载启动	34
4.1.3 加载启动	35
4.1.4 约束启动	35
4.1.5 强制数据库启动	36
4.2 更改数据库的可用性	36

4.2.1 将数据库装入实例	36
4.2.2 打开一个关闭的数据库	36
4.3 关闭数据库	36
4.3.1 用normal选项关闭	37
4.3.2 用immediate 选项关闭	37
4.3.3 用transactional 选项关闭	38
4.3.4 用abort选项关闭	38
4.4 通过诊断文件监控数据库运行	38
4.4.1 后台进程运行跟踪文件	38
4.4.2 警告日志文件	38
4.4.3 用户跟踪文件	39
第 5 章 ORACLE NET	40
5.1 ORACLE NET体系结构.....	40
5.1.1 客户机/服务器结构	40
5.1.2 瘦客户机结构	41
5.2 ORACLE NET服务器端.....	43
5.2.1 监听器进程	43
5.2.2 listener.ora文件	43
5.3 UNIX环境下的监听器进程	44
5.3.1 启动和停止监听器进程	44
5.3.2 查看监听程序运行状况	44
5.3.3 lsnrctl命令的参数	44
5.4 连接描述符	46
第 6 章 数据字典.....	48
6.1 内部RDBMS (X\$) 表	48
6.2 数据字典表	49
6.3 动态性能视图	49
6.4 数据字典视图	49
6.5 数据字典举例	49
6.5.1 数据字典总体信息	49
6.5.2 模式对象信息	49

6.5.3 空间分配信息	49
6.5.4 数据库结构信息	50
第 7 章 管理数据库存储	51
7.1 ORACLE数据库结构	51
7.2 ORACLE块	51
7.2.1 ORACLE块结构	51
7.2.2 理解PCTFREE与PCTUSED	52
7.3 管理表存储区（数据段）	54
7.3.1 创建表	54
7.3.2 STORAGE子句	55
7.4 管理表空间	55
7.4.1 创建表空间	55
7.4.2 表空间的区管理方式	56
7.4.3 调整表空间的大小	57
7.5 获取存储结构的信息	58
7.5.1 查询DBA_SEGMENTS	59
7.5.2 查询DBA_EXTENTS	59
7.5.3 查询DBA_FREE_SPACE	60
第 8 章 备份和恢复	62
8.1 逻辑备份	62
8.1.1 Export实用程序	62
8.1.2 Import实用程序	66
8.2 脱机备份	71
8.3 联机备份	72
8.3.1 归档模式和非归档模式	72
8.3.2 联机数据库备份	73
8.4 数据库恢复	75
8.4.1 实例恢复	75
8.4.2 介质恢复概述	75
8.4.3 无归档日志的数据库恢复	76
8.4.4 有归档日志的数据库完全恢复	77

8.4.5 有归档日志的数据库不完全恢复	79
----------------------------	----

第1章 数据库基础

1.1 基本概念

1.1.1 数据

所谓数据（Data），就是描述事物的符号，在我们的日常生活中数据无所不在，数字、文字、图表、图像、声音等都是数据。人们通过数据来认识世界，交流信息。

1.1.2 数据库

数据库（DB即Database），顾名思义，就是数据存放的地方。在计算机中，数据库是数据和数据库对象的集合。所谓数据库对象是指表（Table）、视图（View）、存储过程（Stored Procedure）、触发器（Trigger）等。

1.1.3 数据库管理系统

数据库管理系统（DBMS 即 Database Management System），是用于管理数据的计算机软件。数据库管理系统使用户能方便地定义和操纵数据，维护数据的安全性和完整性，以及进行多用户下的并发控制和恢复数据库。

1.1.4 数据库系统

数据库系统（DBS 即 Database System），狭义地讲是由数据库、数据库管理系统和用户构成。广义地讲是由计算机、硬件、操作系统、数据库管理系统以及在它支持下建立起来的数据库应用程序、用户和维护人员组成的一个整体。

1.2 关系数据库

数据库这一概念提出后，先后出现了几种数据模型。其中基本的数据模型有三种：层次模型系统、网状模型系统和关系模型系统。60年代末期提出的关系模型具有数据结构简单灵活、易学易懂且具有雄厚的数学基础等特点，从70年代开始流行，发展到现在已成为数据库的标准。目前广泛使用的数据库软件都是基于关系模型的关系数据库管理系统。

1.2.1 关系模型

关系模型（RM即Relational Model）把世界看作是由实体Entity 和联系Relationship 构成的。

所谓实体就是指现实世界中具有区别于其它事物的特征或属性并与其它实体有联系的对象。在关系模型中实体通常是以表的形式来表现的。表的每一行描述实体的一个实例，表的每一列描述实体的一个特征或属性。

所谓联系就是指实体之间的关系，即实体之间的对应关系。联系可以分为三种：

- （1） 一对一的联系。如：一个人只有一种性别，一个人->性别为一对一的联系；
- （2） 一对多的联系。如：相同性别的人有许多个，性别->人为一对多的联系；
- （3） 多对一的联系。如：很多人有同一个性别，人->性别为多对一的联系。

通过联系就可以用一个实体的信息来查找另一个实体的信息。关系模型把所有的数据都组织到表中。表是由行和列组成的，行表示数据的记录，列表示记录中的域。

1.2.2 关系数据库

关系数据库（RDB 即 Relational Database），就是基于关系模型的数据库。

1. 关系数据库管理系统

关系数据库管理系统（RDBMS 即 Relational Database Management System），就是管理关系数据库的计算机软件。

2. 关键字

关键字是关系模型中的一个重要概念，它是逻辑结构，不是数据库的物理部分。

（1） 候选关键字（Candidate Key）

如果一个属性集能唯一地标识表的一行而又不含多余的属性，那么这个属性集称为候选关键字。

（2） 主关键字（Primary Key）

主关键字是被挑选出来作表的行的唯一标识的候选关键字。一个表只有一个主关键字。主关键字又可以称为主键。

（3） 公共关键字（Common Key）

在关系数据库中关系之间的联系是通过相容或相同的属性或属性组来表示的。如果两个关系中具有相容或相同的属性或属性组，那么这个属性或属性组被称为这两个关系的公共关键字。

（4） 外关键字（Foreign Key）

如果公共关键字在一个关系中是主关键字，那么这个公共关键字被称为另一个关系的外关键字。由此可见，外关键字表示了两个关系之间的联系。以另一个关系的外关键字作主关键字的表被称为主表，具有此外关键字的表被称为主表的从表，外关键字又称作外键。

1.3 常见的数据库对象

数据库对象是数据库的组成部分，常见的有以下几种：

1.3.1 表

数据库中的表（Table）与我们日常生活中使用的表格类似，它也是由行（Row）和列（Column）组成的。列由同类的信息组成，每列又称为一个字段，每列的标题称为字段名。行包括了若干列信息项，一行数据称为一个或一条记录，它表达有一定意义的信息组合。一个数据库表由一条或多条记录组成，没有记录的表称为空表。每个表中通常都有一个主关键字，用于唯一地确定一条记录。

1.3.2 索引

索引（Index）是根据指定的数据库表列建立起来的顺序。它提供了快速访问数据的途径，并且可监督表的数据，使其索引所指向的列中的数据不重复。

1.3.3 视图

视图（View）看上去同表似乎一模一样，具有一组命名的字段和数据项，但它其实是一个虚拟的表，在数据库中并不单独存储视图的数据，视图的数据来源于数据库中的表。视图是由查询数据库表产生的，它限制了用户能看到和修改的数据。由此可见，视图可以用来控制用户对数据的访问，并能简化数据的显示，即通过视图只显示那些需要的数据信息。

1.3.4 存储过程

存储过程（Stored Procedure）是为完成特定的功能而汇集在一起的一组 SQL 程序语句，经编译后存储在数据库中。

1.4 范式

构造数据库必须遵循一定的规则，在关系数据库中这种规则就是范式（NF即 Normal Form）。范式是符合某一种级别的关系模式的集合。关系数据库中的关系必须满足一定的要求，即满足不同的范式。目前关系数据库有六种范式：第一范式（1NF）、第二范式（2NF）、第三范式（3NF）、第四范式（4NF）、第五范式（5NF）和第六范式（6NF）。满足最低要求的范式是第一范式1NF，在第一范式的基础上进一步满足更多要求的称为第二范式2NF，其余范式依次类推。一般说来数据库只需满足第三范式3NF就行了。

1.4.1 第一范式

在任何一个关系数据库中第一范式 1NF 是对关系模式的基本要求，不满足第一范式 1NF 的数据库就不是关系数据库。所谓第一范式 1NF，是指数据库表的每一列都是不可分割的基本数据项，同一列中不能有多值，即实体的某个属性不能有多值或者不能有重复的属性。如果出现重复的属性就可能需要定义一个新的实体。新的实体由重复的属性构成新实体与原实体之间为一对多关系。在第一范式 1NF 中，表的每一行只包含一个实例的信息。

1.4.2 第二范式

第二范式 2NF 是在第一范式 1NF 的基础上建立起来的，即满足第二范式 2NF 必须先满足第一范式 1NF。第二范式 2NF 要求数据库表中的每个实例或行必须可以被唯一地区分，为实现区分通常需要为表加上一个列以存储各个实例的唯一标识，这个唯一属性列被称为主关键字或主键、主码。

第二范式 2NF 要求实体的属性完全依赖于主关键字。所谓完全依赖是指自身不能存在仅依赖主关键字一部分的属性，如果存在，那么这个属性和主关键字的这一部分应该分离出来形成一个新的实体，新实体与原实体之间是一对多的关系。为实现区分通常需要为表加上一个列以存储各个实例的唯一标识。例如，员工信息表中加上了员工编号 emp_id 列，因为每个员工的员工编号是唯一的，因此每个员工可以被唯一区分。简而言之，第二范式就是非主属性非部分依赖于主关键字。

1.4.3 第三范式

满足第三范式 3NF 必须先满足第二范式 2NF，简而言之第三范式 3NF 要求一个数据库表中不包含已在其它表中已包含的非主关键字信息。例如，存在一个部门信息表，其中每个部门有部门编号 dept_id、部门名称、部门简介等信息，那么员工信息表中列出部门编号后就不能再将部门名称、部门简介等与部门有关的信息再

加入员工信息表中。如果不存在部门信息表则根据第三范式 3NF 也应该构建它，否则就会有大量的数据冗余。简而言之，第三范式就是属性不依赖于其它非主属性。

1.5 DML 语言

1.5.1 查询数据

SQL 语言中最主要、最核心的部分是它的查询功能即 **SELECT** 语句。查询语言用来对已经存在于数据库中的数据按照特定的组合、条件表达式或次序进行检索。**SELECT** 语句的完整语法如下：

```
SELECT [ALL|DISTINCT]] select_list  
[INTO new_table_]  
FROM table_source  
[WHERE search_condition]  
[GROUP BY group_by_expression]  
[HAVING search_condition]  
[ORDER BY order_expression [ASC|DESC]]
```

1. 简单查询

- (1) 用 **SELECT** 子句来指定查询所需的列，多个列之间用逗号分开。例如：

```
select p_id, p_name, cost from products
```

- (2) 可以使用符号* 来选取表的全部列。例如：

```
select * from products
```

- (3) 在查询结果中添加列。例如：

```
select p_id, p_name, quantity, cost, cost*quantity as sum_cost from products
```

- (4) 使用 **WHERE** 子句。例如：

```
select e_name from employee where e_wage between 2000 and 3000
```

- (5) 使用 **DISTINCT** 关键字。例如：

```
select distinct dept_id from employee where e_wage > 7000
```

(6) 使用 IN 关键字。例如：

```
select e_name from employee where dept_id in ('1001','1002')
```

(7) 使用通配符。例如：

```
select e_name, dept_id from employee where e_name like 'wang_'
```

(8) 使用 ORDER 子句。例如：

```
select p_id, p_name, cost, quantity from products where dept_id = '1003'

order by cost desc, quantity
```

(9) 选取前几行数据。

在 SELECT 语句中使用 TOP n 或 TOP n PERCENT 来选取查询结果的前 n 行或前百分之 n 的数据，此语句经常和 ORDER 子句一起使用。例如：

```
select top 3 e_name, e_wage from employee order by e_wage desc
```

(10) 使用 GROUP 子句。例如：

```
select title_id, copies_sold=sum(qty) from sales group by title_id
```

GROUP BY 的特征是：能按列或表达式分组、一般同集合函数一起用、为每组产生一个值。此外，在包含 GROUP BY 子句的查询语句中，SELECT 子句后的所有字段列表，除集合函数外，都应包含在 GROUP BY 子句中，否则将出错。

(11) 使用 HAVING 子句。例如：

```
select dept_id, count(*) from employee where e_wage >= 6000

group by dept_id having count(*) > 1
```

HAVING 子句用来选择特殊的组，它将组的一些属性与常数值进行比较，如果一个组满足 HAVING 子句中的逻辑表达式，它就可以包含在查询结果中。

2. 连接查询

在数据库应用中，经常要涉及从两个或更多的表中查询数据，这就需要使用连接查询。

(1) 非限制连接 (CROSS JOIN)

非限制连接 (CROSS JOIN)，就是指不带 WHERE 子句的查询。在数学上，就是表的笛卡尔积。若 R 表和 S 表非限制连接，而且 R 表有 X 行，S 表有 Y 行，那么结果集是 X * Y 行。即：R 表的一行对应着 S 表的所有行。在应用中，非限制

连接产生的是无意义结果集，但在数据库的数学模式上有重要的作用。例如，查询出版社和书的所有组合：

```
SELECT PUB_NAME, TITLE FROM TITLES, PUBLISHERS
```

（2）自然连接（INNER JOIN）也叫内连接。例如，求每本杂志上刊登的文章：

```
SELECT PUB_NAME, TITLE FROM TITLES,PUBLISHERS
```

```
WHERE TITLES.PUB_ID=PUBLISHERS.PUB_ID
```

或写成：

```
SELECT PUB_NAME,TITLE FROM TITLES INNER JOIN PUBLISHERS
```

```
ON TITLES.PUB_ID=PUBLISHERS.PUB_ID
```

其中，INNER JOIN 是 SQL Server 的缺省连接，可简写为 JOIN。在 JOIN 后面指定哪些表作连接，ON 后面指定了连接的条件。

（3）外连接

外连接（OUTER JOIN）允许限制一张表中的行，而不限制另一张表中的行。例如，显示所有的书名（无销售记录的书也包括在内，“*”在左边表示不限制左边表的数据）：

```
SELECT TITLES.TITLE_ID,title=convert(char(38),TITLE),QTY FROM
TITLES,SALES WHERE TITLES.TITLE_ID *= SALES.TITLE_ID
```

或写成：

```
SELECT TITLES.TITLE_ID,title=convert(char(38),TITLE),QTY
```

```
FROM TITLES LEFT OUTER JOIN SALES ON TITLES.TITLE_ID =
SALES.TITLE_ID
```

3. 子查询

子查询是指，一条 SELECT 语句作为另一条 SELECT 语句的一部分。外层的 SELECT 语句叫外部查询，内层的 SELECT 语句叫内部查询（或子查询）。例如，查询有销售记录的所有书信息，包括书的编号、书名（最长显示 28 个字符）、类型和价格：

```
SELECT title_id,convert(char(28),title),type, price FROM TITLES
```

```
WHERE TITLE_ID IN (SELECT TITLE_ID FROM SALES)
```

4. 存在性查询

使用 **EXISTS** 来决定数据是否在查询列表中存在。**EXISTS** 表示一个子查询至少返回一行时条件成立。例如，求已销售的书的信息：

```
SELECT TITLE_ID,TITLE FROM TITLES WHERE EXISTS (SELECT * FROM SALES WHERE
TITLES.TITLE_ID =SALES.TITLE_ID)
```

5. 在查询的基础上创建新表

SELECT INTO 的作用是，在查询的基础上创建新表。若建永久表，必须设置“**SELECT INTO / BULKCOPY**”。若建临时表，必须在表前设置#（局部临时表）或##（全局临时表）。例如：

```
SELECT TITLE=SUBSTRING (TITLE,1,40), MONTHLY = YTD_SALES/12 INTO
#PHONYTABLE FROM TITLES
```

1.5.2 插入数据

INSERT 语句可以实现往数据库表中插入记录。有两种方法可以向一张表中插入数据。一种是用 **VALUES** 选择，直接给各字段赋值；二是通过一条查询语句，把从其他表或视图选取的数据插入。**INSERT** 语句的完整语法如下：

```
INSERT [INTO ]
{ TABLE_NAME | VIEW_NAME } [ ( COLUMN_LIST ) ]
{ VALUES | VALUES_LIST | SELECT_STATEMENT }
```

1. 插入所有值

当向表中所有列都插入新数据时，可以省略列名表，但是必须保证 **VALUES** 后的各数据项位置同表定义时顺序一致。例如：

```
INSERT AUTHORS VALUES ( '123-45-6789 ','YANG','ZH','900555_1212')
```

2. 插入部分值

当向表中插入部分数据时，应在列名表处写出各个字段的顺序。例如：

```
INSERT PUBLISHERS (PUB_ID, PUB_NAME) VALUES ('9975', 'UNBOUND PRESS')
```

3. 插入缺省值选项

方法一：为所有列插入缺省值

语法：**INSERT TABLENAME DEFAULT VALUES**

这种语法要求所有列必须具有 IDENTITY 属性、TIMESTAMP 类型、允许 NULL 或赋有 DEFAULT 值。

方法二：为列插入缺省值

该列必须具有 TIMESTAMP 类型、允许 NULL 或赋有 DEFAULT 值。

```
INSERT shippers(companyname,phone)
```

```
VALUES ('yangzh coffee co.', DEFAULT )
```

4. 用 SELECT 插入多行

可以用查询语句从其他表或视图中选出要插入的数据，并将它插入到表中。语法如下：

```
INSERT TABLE_NAME
```

```
SELECT COLUMN_LIST
```

```
FROM TABLE_LIST
```

```
WHERE SEARCH_CONDITIONS
```

1.5.3 修改数据

UPDATE 语句实现更新数据库。其完整语法如下：

```
UPDATE { TABLE_NAME | VIEW_NAME }
```

```
SET [ { TABLE_NAME | VIEW_NAME } ]
```

```
{ COLUMN_LIST | VARIABLE_LIST } = expression
```

```
[WHERE CLAUSE]
```

例如：

```
update discounts set discount=discount+0.10 where lowqty>=100
```

1.5.4 删除数据

DELETE 实现删除数据库表中的记录，其完整语法如下：

```
DELETE [FROM] TABLE_NAME WHERE SEARCH_CONDITIONS
```

例如：

```
delete sales where datediff(year,ord_date,getdate())>=3
```

第2章 ORACLE SERVER 构成

2.1 概述

ORACLE SERVER 由 ORACLE 实例和 ORACLE 数据库构成。如图 2.1-1 所示。

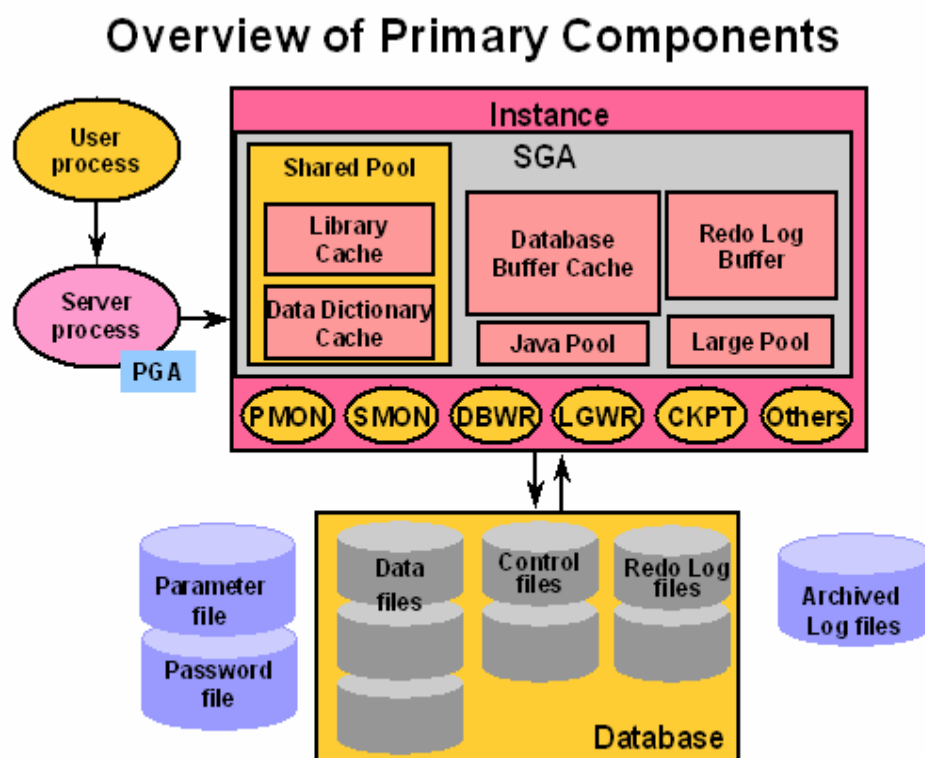


图 2.1-1 ORACLE SERVER 的构成

2.2 ORACLE 实例

一个 ORACLE 实例为存取和控制一数据库的软件机制。每一次在数据库服务器上启动一数据库时，称为系统全局区（SYSTEM GLOBAL AREA）的一内存区（简称 SGA）被分配，有多个 ORACLE 后台进程被启动。该 SGA 和 ORACLE 后台进程的结合作为一个 ORACLE 实例。ORACLE 实例为管理数据库的数据，为数据库的一个或多个用户而工作。在 ORACLE SERVER 启动时，首先是实例启动，然后由实例装配（MOUNT）一数据库。

2.2.1 ORACLE 进程

ORACLE 客户和服务交互过程中的进程分为用户进程和 ORACLE 进程。当一用户运行一 ORACLE 客户应用程序，如 PRO*C 程序或一个 ORACLE 工具（如 SQL*PLUS），为用户运行的应用建立一个用户进程。ORACLE 进程又分为两类：服务器进程和后台进程。服务器进程用于处理连接到该实例的用户进程的请求。当应用和 ORACLE 是在同一台机器上运行，而不再通过网络，一般将用户进程和它相应的服务器进程组合成单个的进程，可降低系统开销。然而，当应用和 ORACLE 运行在不同的机器上时，用户进程经过一个分离服务器进程与 ORACLE 通信。它可执行下列任务：

- (1) 对应用所发出的 SQL 语句进行语法分析和执行
- (2) 从磁盘数据文件中读入必要的数据库块到 SGA 的共享数据库缓冲区（该块不在缓冲区时）
- (3) 将结果返回给应用程序处理。

系统为了使性能最好和协调多个用户，在多进程系统中使用一些附加进程，称为后台进程。在许多操作系统中，后台进程是在实例启动时自动地建立。一个 ORACLE 实例有多个后台进程，分别为：DBWR（数据库写进程）、LGWR（日志写进程）、CKPT（检查点进程）、SMON（系统监控进程）、PMON（进程监控进程）、ARCH（归档进程）、RECO（恢复进程）。每个后台进程与 ORACLE 数据库的不同部分交互，其中前 5 个进程是必须的，后 2 个进程是可选的。

下面对后台进程的功能作简单介绍。

1. DBWR 进程

该进程的作用是将缓冲区写入数据文件，是负责缓冲存储区管理的一个 ORACLE 后台进程。当缓冲区中的一缓冲区被修改，它被标志为“弄脏”，DBWR 的主要任务是将“弄脏”的缓冲区写入磁盘，使缓冲区保持“干净”。由于缓冲存储区的缓冲区填入数据库或被用户进程弄脏，未用的缓冲区的数目减少。当未用的缓冲区下降到很少，以致用户进程要从磁盘读入块到内存存储区时无法找到未用的缓冲区时，DBWR 将管理缓冲存储区，使用户进程总可得到未用的缓冲区。

ORACLE 采用 LRU（LEAST RECENTLY USED）算法（最近最少使用算法）保持内存中的数据块是最近使用的，使 I/O 最小。在下列情况下 DBWR 要将弄脏的缓冲区写入磁盘。

- (1) 当一个服务器进程将一缓冲区移入“弄脏”表，该弄脏表达到临界长度时，该服务进程将通知 DBWR 进行写。该临界长度是为参数 DB-BLOCK-WRITE-BATCH 的值的一半。
- (2) 当一个服务器进程在 LRU 表中查找 DB-BLOCK-MAX-SCAN-CNT 缓冲区时，没有查到未用的缓冲区，它停止查找并通知 DBWR 进行写。
- (3) 出现超时（每次 3 秒），DBWR 将通知本身。
- (4) 当出现检查点时，LGWR 将通知 DBWR。

在前两种情况下，DBWR 将弄脏表中的块写入磁盘，每次可写的块数由初始参数 DB-BLOCK-WRITE-BATCH 所指定。如果弄脏表中没有该参数指定块数的缓冲区，DBWR 从 LUR 表中查找另外一个弄脏缓冲区。

如果 DBWR 在三秒内未活动，则出现超时。在这种情况下 DBWR 对 LRU 表查找指定数目的缓冲区，将所找到任何弄脏缓冲区写入磁盘。如果数据库空运转，DBWR 最终将全部缓冲区存储区写入磁盘。

在出现检查点时，LGWR 指定一修改缓冲区表必须写入到磁盘。DBWR 将指定的缓冲区写入磁盘。

在有些平台上，一个实例可有多个 DBWR。在这样的实例中，一些块可写入一磁盘，另一些块可写入其它磁盘。参数 DB-WRITERS 控制 DBWR 进程个数。

2. LGWR 进程

该进程将日志缓冲区写入磁盘上的一个日志文件，它是负责管理日志缓冲区的一个 ORACLE 后台进程。LGWR 进程将自上次写入磁盘以来的全部日志项输出。在下列情况下 LGWR 进程将日志缓冲区内容写入磁盘。

- (1) 当用户进程提交一事务时写入一个提交记录。
- (2) 每三秒将日志缓冲区输出。
- (3) 当日志缓冲区的 1/3 已满时将日志缓冲区输出。
- (4) 当 DBWR 将修改缓冲区写入磁盘时则将日志缓冲区输出。

LGWR 进程同步地写入到活动的镜像在线日志文件组。如果组中一个文件被删除或不可用，LGWR 可继续地写入该组的其它文件。

日志缓冲区是一个循环缓冲区。当 LGWR 将日志缓冲区的日志项写入日志文件后，服务器进程可将新的日志项写入到该日志缓冲区。LGWR 通常写

得很快，可确保日志缓冲区总有空间可写入新的日志项。当一事务提交时，被赋予一个系统修改号（SCN），它同事务日志项一起记录在日志中。

3. CKPT 进程

当该进程被唤醒执行时，对全部数据文件的标题进行修改，指示该检查点，并通知 DBWR 将脏数据块写入数据文件。

4. SMON 进程

该进程的作用是，在 ORACLE 实例启动时执行实例恢复，还负责清理不再使用的临时段。

5. PMON 进程

该进程在用户进程出现故障时执行进程恢复，负责清理私有存储区和释放该进程所使用的资源。例如它要重置活动事务表的状态，释放封锁，将该故障的进程的 ID 从活动进程表中移去。

6. RECO 进程

该进程是在具有分布式选项时所使用的一个进程，自动地解决在分布式事务中的故障。

7. ARCH 进程

该进程将已填满的在线日志文件拷贝到指定的存储设备。当日志是为 ARCHIVELOG 使用方式、并可自动地归档时 ARCH 进程才存在。

2.2.2 ORACLE 内存结构

ORACLE 在内存存储下列信息：

- (1) 执行的程序代码
- (2) 连接的会话信息
- (3) 程序执行期间所需数据和共享的信息
- (4) 存储在外存储上的缓冲信息

ORACLE 具有下列基本的内存结构：

- 软件代码区
- 系统全局区，包括数据库缓冲存储区、日志缓冲区和共享池
- 程序全局区，包括栈区和数据区

■ 排序区

1. 软件代码区

用于存储正在执行的或可以执行的程序代码。软件区是只读，可安装成共享或非共享。ORACLE 系统程序是可共享的，以致多个 ORACLE 用户可存取它，而不需要在内存有多个副本。用户程序可以共享也可以不共享。

2. 系统全局区

为一组由 ORACLE 分配的共享的内存结构，可包含一个数据库实例的数据或控制信息。如果多个用户同时连接到同一实例时，在实例的 SGA 中数据可为多个用户所共享，所以又称为共享全局区。当实例启动时，SGA 的存储自动地被分配；当实例关闭时，该存储被回收。SGA 划分成：数据库缓冲存储区、日志缓冲区、共享池、请求和响应队列、数据字典存储区和其它各种信息。

3. 程序全局区

程序全局区 PGA (PROCESS GLOBAL AREA) 是一个内存区，包含单个进程的数据和控制信息，所以又称为进程全局区。

4. 排序区

排序需要内存空间，ORACLE 利用该内存排序数据，这部分空间称为排序区。排序区存在于请求排序的用户进程的内存中。

2.3 ORACLE 数据库

一个 ORACLE 数据库是数据的集合，被处理成一个单位。每个 ORACLE 数据库有一个物理结构和一个逻辑结构。

数据库物理结构是由构成数据库的操作系统文件所决定的。数据库的文件为数据库信息提供真正的物理存储。每一个 ORACLE 数据库是由三种类型的文件组成，即数据文件、日志文件和控制文件。

逻辑数据库结构是用户所涉及的数据库结构，一个 ORACLE 数据库的逻辑结构由下列因素决定：

- (1) 一个或多个表空间
- (2) 数据库模式对象

逻辑存储结构例如表空间用于支配一个数据库的物理空间如何使用，模式对象及它们之间的联系组成了一个数据库的关系设计。

2.3.1 数据库物理结构

ORACLE 数据库由三种类型的物理文件组成，即数据文件、日志文件和控制文件，如图 2.3-1 所示。

Physical Structure

The physical structure includes three types of files:

- **Control files**
- **Data files**
- **Online redo log files**

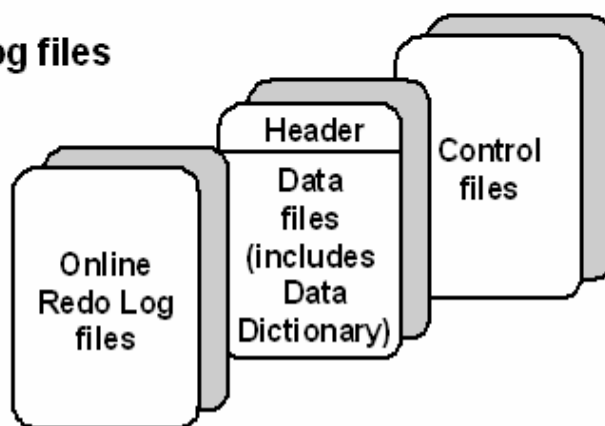


图 2.3-1 ORACLE 数据库的物理结构

1. 数据文件

每一个 ORACLE 数据库有一个或多个物理的数据文件(data file)。一个数据库的数据文件包含全部数据库数据。逻辑数据库结构的数据也是物理地存储在数据库的数据文件中。数据文件有下列特征：

- (1) 一个数据文件仅与一个数据库相联系
- (2) 一旦建立数据文件不能改变大小

(3) 一个表空间由一个或多个数据文件组成

数据文件中的数据在需要时可以读取并存储在 ORACLE 内存存储区中。例如：用户要存取数据库一表的某些数据，如果请求信息不在数据库的内存存储区内，则从相应的数据文件中读取并存储在内存。当修改和插入新数据时，不必立刻写入数据文件。为了减少磁盘输出的总数，提高性能，数据存储在内存，然后由 ORACLE 后台进程 DBWR 决定如何将其写入到相应的数据文件。

2. 日志文件

每一个数据库有由两个或多个日志文件(redo log file)构成的日志文件组，每一个日志文件用于收集数据库日志。

日志的主要功能是记录对数据库所做的修改，所以对数据库作的全部修改均被记录在日志中。日志文件主要是保护数据库以防止故障，为了防止日志文件本身的故障，ORACLE 允许镜像日志(mirrored redo log) 以便可在不同磁盘上维护多个相同的日志副本。

日志文件中的信息仅在系统故障或介质故障恢复数据库时使用。

3. 控制文件

每一个 ORACLE 数据库至少有一个控制文件(control file)，它记录数据库的物理结构，包含的主要信息是：

- (1) 数据库名
- (2) 数据库数据文件和日志文件的名字和位置
- (3) 数据库建立日期

每一次 ORACLE 数据库的实例启动时，它的控制文件用于标识数据库和日志文件，当着手数据库操作时它们必须被打开。当数据库的物理组成更改时，ORACLE 自动更改该数据库的控制文件。

2.3.2 数据库逻辑结构

数据库的逻辑结构包含表空间(table space)、段(segment)、区(extent)、数据块(block)，如图 2.3-2 所示。

1. 表空间

一个数据库划分为一个或多个逻辑单位，该逻辑单位称为表空间。一个表空间可将相关的逻辑结构组合在一起。DBA 可利用表空间作下列工作：

- (1) 控制数据库数据的磁盘分配
- (2) 将确定的空间份额分配给数据库用户
- (3) 通过使单个表空间在线或离线，控制数据的可用性
- (4) 执行部分数据库备份或恢复操作
- (5) 为提高性能，跨越设备分配数据存储

数据库、表空间和数据文件之间的关系如图 2.3-2 所示。

Logical Structure

- Dictates how the physical space of a database is used
- Hierarchy consisting of tablespaces, segments, extents, and blocks

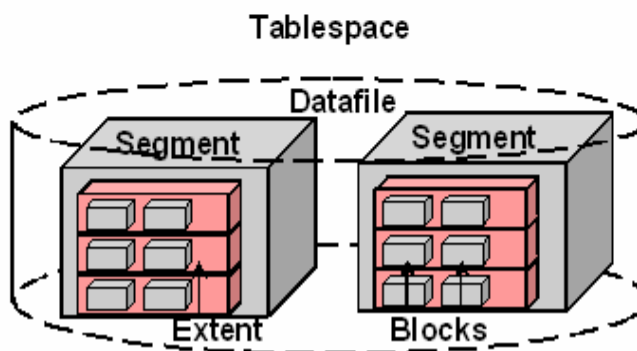


图 2.3-2 ORACLE 数据库的逻辑结构

每个数据库可逻辑划分为一个或多个表空间。每一个表空间是由一个或多个数据文件组成，该表空间物理地存储表空间中全部逻辑结构的数据。DBA 可以建立新的表空间，可为表空间增加数据文件或可删除数据文件，设置或更改缺省的段存储位置。

每一个 ORACLE 数据库包含有一个名为 **SYSTEM** 的表空间，该表空间在数据库建立时自动建立。在该表空间中总包含有整个数据库的数据字典表。最小的数据库可只需要 **SYSTEM** 表空间。该表空间必须总是在线。表和存储的 PL/SQL 程序单元（过程、函数、包和触发器）的全部存储数据是存储在 **SYSTEM** 表空间中。

通过增加表空间的数据文件来扩大表空间，表空间的大小为组成该表空间的数据文件大小的和。

DBA 可以使 ORACLE 数据库中除 **SYSTEM** 表空间外的任何其他表空间在在线或离线（离线的该表空间不能有活动的回滚段）。表空间通常是在线，以致它所包含的数据对数据库用户是可用的。当表空间为离线时，其数据不可使用。在下列情况下，DBA 可以使其离线：

- (1) 使部分数据不可用，而剩余的部分允许正常存取
- (2) 执行离线的表空间备份
- (3) 为了修改或维护一应用，使它和它的一组表临时不可用

包含有正在活动的回滚段的表空间不能被离线，仅当回滚段不正在使用时，该表空间才可离线。

在数据字典中记录表空间的状态，在线还是离线。如果在数据库关闭时一表空间为离线，那么在下次数据库装配和重新打开后，它仍然保持离线。

当出现某些错误时，一个表空间可自动地由在线改变为离线。通过使用多个表空间，将不同类型的数据分开，更方便 DBA 来管理数据库。

ORACLE 数据库中一表空间是由一个或多个物理数据文件组成，一个数据文件只可与一个表空间相联系。

ORACLE 通过段、区和数据块等逻辑数据结构可更细地控制磁盘空间的使用。

2. 段

段（SEGMENT）包含表空间中一种指定类型的逻辑存储结构，是由一组区组成。在 ORACLE 数据库中有几种类型的段：数据段、索引段、回滚段和临时段。

数据段：对于每一个非聚集的表有一数据段，表的所有数据存放在该段。每一聚集有一个数据段，聚集中每一个表的数据存储在段中。

索引段：每一个索引有一索引段，存储索引数据。

回滚段：是由 DBA 建立，用于临时存储要撤消的信息，这些信息用于生成读一致性数据库信息、在数据库恢复时使用、回滚未提交的事务。

临时段：当一个 SQL 语句需要临时工作区时，由 ORACLE 建立。当语句执行完毕，临时段的区退回给系统。

ORACLE 对所有段的空间分配，以区为单位。

3. 区

一个区（EXTENT）是数据库存储空间分配的一个逻辑单位，它由连续的 ORACLE 数据块所组成。每一个段是由一个或多个区组成。当一段中的所有空间已完全使用时，ORACLE 为该段分配一个新的区。

为了维护的目的，在数据库的每一段含有段标题块说明段的特征以及该段中的区目录。

4. 数据块

数据块（data block）是 ORACLE 管理数据文件中存储空间单位，为数据库使用的 I/O 的最小单位，其大小可不同于操作系统的标准 I/O 块大小。

2.4 模式和模式对象

一个模式(schema)为模式对象(schema object)的一个集合，每一个数据库用户对应一个模式。模式对象为直接引用数据库数据的逻辑结构，模式对象包含如表、视图、索引、聚集、序列、同义词、数据库链、过程和包等结构。模式对象是逻辑数据存储结构，每一种模式对象在磁盘上没有一个相应文件存储其信息。一个模式对象逻辑地存储在数据库的一个表空间中，每一个对象的数据物理地包含在表空间的一个或多个数据文件中。

2.4.1 表

表（table）为数据库中数据存储的基本单位，其数据按行、列存储。每个表具有一表名和列的集合。每一列有一个列名、数据类型、宽度或精度、比例。一行是对应单个记录的列信息的集合。

2.4.2 视图

一个视图（view）是一个或多个表（或其他视图）中的数据的一种定制的表达，是用一个查询定义，所以可认为是一个存储的查询（stored query）或是一个虚表（virtual table）。视图可在使用表的许多地方使用。

由于视图是由表导出的，视图和表存在许多类似。视图可以被查询，而在修改、插入或删除时具有一定的限制，在视图上执行的全部操作真正地影响视图的基本表中的数据，受到基本表的完整性约束和触发器的限制。

视图与表不同，一个视图不分配任何存储空间，视图不真正地包含数据。由查询定义的视图相应于视图引用表中的数据。视图只在数据字典中存储其定义。

引入视图有下列好处：

- 通过限制对表的行预定义集合的存取，为表提供附加的安全性
- 隐藏数据复杂性
- 为用户简化命令
- 为基本表的数据提供另一种观点
- 可将应用隔离基本表定义的修改
- 用于不用视图无法表示的查询
- 可用于保存复杂查询

2.4.3 聚集

聚集（cluster）是存储表数据的可选择的方法。一个聚集是一组表，将具有同一公共列值的行存储在一起，并且它们经常一起使用。这些公共列构成聚集码。例如：EMP 表与 DEPT 表共享 DEPTNO 列，所以 EMP 表和 DEPT 表可聚集在一起，聚集码的列为 DEPTNO 列。

2.4.4 索引

索引(index)是与表和聚集相关的一种选择结构。索引是为提高数据检索的性能而建立，利用它可快速地确定指定的信息。ORACLE 索引为表数据提供快速存取路径。索引适用于一定范围的行查询或指定行的查询。

索引可建立在一表的一列或多列上，一旦建立，由 ORACLE 自动维护和使用，对用户是完全透明的。索引是逻辑地和物理地独立于数据，它们的建立或删除对表

没有影响，应用可继续处理。索引数据的检索性能几乎保持常数，而当一表上存在许多索引时，修改、删除和插入操作的性能会下降。

索引有唯一索引和非唯一索引。唯一索引保证表中没有两行在定义索引的列上具有重复值。

组合索引是在表的某个列上所建立的一索引。组合索引可加快 **SELECT** 语句的检索速度，在其 **WHERE** 子句中可引用组合索引的全部或主要部分。所以在定义中给出列的次序，将经常存取的或选择最多的列放在首位。

在建立索引时，将在表空间自动地建立一索引段。

2.4.5 程序单元

程序单元（**program unit**）是指存储过程、函数和包（**package**）。一个过程和函数，是由 **SQL** 语句和 **PL/SQL** 语句组合在一起，为执行某一个任务的一个可执行单位。一个过程或函数可被建立，在数据库中存储其编译形式，可由用户或数据库应用所执行。过程和函数差别在函数总返回单个值给调用者，而过程没有值返回给调用者。

包提供相关的过程、函数、变量和方法，允许管理者和应用开发者利用方法编写程序来提供更多的功能和提高性能。

第3章 用户管理

3.1 创建用户

创建用户的目的是，建立一个安全、有用的帐户，并且这个帐户要有充分的权限和正确的缺省设置值。可以使用 `create user` 命令来创建一个新的数据库帐户。该帐户创建后，在授权前它没有任何效力，用户甚至不能注册。

所有用户帐户所需的设置值都可以在一个 `create user` 命令中指定。这些设置包括表 3.1-1 列出的所有参数。

表 3.1-1 create user 命令的参数

参 数	使 用
Username	用户名
Password	帐户的口令，也可以直接与操作系统主帐户名相连，或者通过一个网络验证服务验证。对于基于主机的验证，使用 <code>identified externally</code> 命令。对于基于网络的验证，使用 <code>identified globally as</code> 命令
Default tablespace	缺省表空间用来存储在该模式下创建的对象。这个设置并不授予用户创建对象的权力；而只是设置一个缺省值
Temporary tablespace	这个表空间只用来存储排序事务处理时所用的临时段
Quota[on tablespace]	用户在规定的表空间存储对象，最多可达到这个定额规定的总尺寸
Profile	赋予用户一个环境文件。如果没有指定环境文件，就使用缺省环境文件。环境文件用于限制对系统资源的使用和执行口令管理规则
Default Role	设置缺省角色供用户使用

下面是一个 `create user` 命令的例子：

```
Create user user1
Identified by password
Default tablespace users
Temporary tablespace temp
Quoto 100M on users;
```

在这个例子中，创建了一个叫作 `user1` 的用户，口令为 `password`，缺省表空间为 `users`，临时表空间为 `temp`，使用缺省概要文件 `DEFAULT`。

授予 `user1` 的定额为 `user` 表空间中的 `100MB`，用户 `user1` 现在可以在 `users` 表空间中创建最多 `100MB` 的数据段。

除用户名外，`create user` 命令中的全部参数都可以由 `alter user` 命令来更改。

3.2 撤销和改变用户

可以用 `drop user` 命令从数据库中撤销一个用户。这个命令只有一个参数即 `cascade`，在撤销该用户之前，它撤销用户模式中的所有对象。如果用户拥有对象，就必须指定 `cascade` 以撤销用户。例如：

```
drop user user1 cascade;
```

引用被撤销的用户模式中对象的任何视图、同义词、过程、函数或数据包都被标上 `invalid`。如果以后用相同的名字创建另一个用户，他从具有相同名称的前任用户继承不到任何东西。

具有 `alter user` 系统权限的用户可以使用命令 `alter user` 改变用户的属性。例如：

```
Alter user user1
```

```
Identified by new_password
```

```
Default tablespace data_ts
```

```
Temporary tablespace temp
```

```
Quoto 200M on users
```

```
Profile profile_name
```

3.3 用户权限

ORACLE 的用户权限包括两种，即系统权限和对象权限。用户权限是指执行某一 SQL 语句或访问另一用户对象的权利。

3.3.1 系统权限

ORACLE 有 100 多种系统权限，每一种系统权限允许用户执行某一个数据库操作或某一类数据库操作。系统权限应谨慎地授予数据库用户或角色。常见的系统权限如表 3.3-1 所示。

表 3.3-1 ORACLE 的常见系统权限

特 权	所能实现的操作
分析	
ANALYZE ANY	分析数据库中的任何表、簇或索引
审计	
AUDIT ANY	审计数据库中的任何模式对象
AUDIT SYSTEM	启用与停用语句和特权的审计选项
簇	
CREATE CLUSTER	在自有的模式中创建一个簇
CREATE ANY CLUSTER	在任何一个模式中创建一个簇；操作类似于 CREATE ANY TABLE
ALTER ANY CLUSTER	改变数据库中的任何一个簇
DROP ANY CLUSTER	删除数据库中的任何一个簇
数据库	
ALTER DATABASE	改变数据库；不管操作系统的特权，经由 Oracle 把文件添加到操作系统中
数据库链接	
CREATE DATABASE LINK	在自有模式中创建专用数据库链接
索引	
CREATE ANY INDEX	在任何表的任何模式中创建一条索引
ALTER ANY INDEX	改变数据库中的任何索引
DROP ANY INDEX	删除数据库中的任何索引
库	
CREATE LIBRARY	在自有模式中创建调出库
CREATE ANY LIBRARY	在任何模式中创建调出库
DROP LIBRARY	删除自有模式中的调出库
DROP ANY LIBRARY	删除任何模式中的调出库
特权	
GRANT ANY PRIVILEGE	授予任何系统特权（不包括对象特权）
过程	
CREATE PROCEDURE	在自有模式中创建存储的过程、函数和包
CREATE ANY PROCEDURE	在任何模式中创建存储的过程、函数和包（这要求用户还要有 ALTER ANY TABLE、BACKUP ANY TABLE、DROP ANY TABLE、SELECT ANY TABLE、INSERT ANY TABLE、UPDATE ANY TABLE、DELETE ANY TABLE 或 GRANT ANY TABLE 特权）
ALTER ANY PROCEDURE	编译任何模式中的任何存储的过程、函数或包
DROP ANY PROCEDURE	删除任何模式中的任何存储的过程、函数或包
EXECUTE ANY PROCEDURE	执行任何过程或函数（独立的或成组的），或在任何模式中引用任何公共包变量
环境资源文件	
CREATE PROFILE	创建环境资源文件
ALTER PROFILE	改变数据库中的任何环境资源文件
DROP PROFILE	删除数据库中的任何环境资源文件
ALTER RESOURCE COST	设置所有的用户会话中使用的资源开销

公共数据库链接	
CREATE PUBLIC DATABASE LINK	创建公共数据库链接
DROP PUBLIC DATABASE LINK	删除公共数据库链接
公共同义词	
CREATE PUBLIC SYNONYM	创建公共同义词
DROP PUBLIC SYNONYM	删除公共同义词
角色	
CREATE ROLE	创建角色
ALTER ANY ROLE	改变数据库中的任何一个角色
DROP ANY ROLE	删除数据库中的任何一个角色
GRANT ANY ROLE	授权数据库中的任何一个角色
回滚段	
CREATE ROLLBACK SEGMENT	创建回滚段
ALTER ROLLBACK SEGMENT	改变回滚段
DROP ROLLBACK SEGMENT	删除回滚段
会话	
CREATE SESSION	连接到数据库
ALTER SESSION	发出ALTER SESSION语句
RESTRICTED SESSION	当数据库利用 STARTUP RESTRICT 启动时进行连接 (OSOPER与 OSDBA角色包含此特权)
序列	
CREATE SEQUENCE	在自有模式中创建序列
CREATE ANY SEQUENCE	在任何模式中创建任何序列
ALTER ANY SEQUENCE	在任何模式中改变任何序列
DROP ANY SEQUENCE	在任何模式中删除任何序列
SELECT ANY SEQUENCE	在任何模式中引用任何序列
快照	
CREATE SNAPSHOT	在自有模式中创建快照 (用户还必须具有 CREATE TABLE 特权)
CREATE ANY SNAPSHOT	在任何模式中创建快照 (用户还必须具有 CREATE ANY TABLE 特权)
ALTER SNAPSHOT	改变任何模式中的任何快照
DROP ANY SNAPSHOT	删除任何模式中的任何快照
同义词	
CREATE SYNONYM	在自有模式中创建同义词
CREATE ANY SYNONYM	在任何模式中创建任何同义词
DROP ANY SYNONYM	在任何模式中删除任何同义词
系统	
ALTER SYSTEM	发出ALTER SYSTEM语句
表	
CREATE TABLE	在自有模式中创建表。还使被授权者能在自有模式下的表中创建索引, 包括那些用于完整性约束的索引 (被授权者必须有表空间的定额或 UNLIMITED TABLESPACE 特权)
CREATE ANY TABLE	在任何模式中创建表 (假如被授权者有 CREATE ANY TABLE 特权并在另一个用户模式中创建了一张表, 那么拥有者必须在那个表空间上有空间定额。表的拥有者不必具有 CREAT [ANY] TABLE 特权)
ALTER ANY TABLE	改变任何模式中的任何表并编译任何模式中的任何视图
BACKUP ANY TABLE	在任何模式中使用表的导出工具执行一个增量导出操作
DROP ANY TABLE	删除或截断任何模式中的任何表
LOCK ANY TABLE	锁定任何模式中的任何表或视图

COMMENT ANY TABLE	对任何模式中的任何表、视图或列进行注释
SELECT ANY TABLE	对任何模式中的任何表、视图或快照进行查询
INSERT ANY TABLE	把行插入到任何模式中的任何表或视图中
UPDATE ANY TABLE	修改任何模式中的任何表或视图中的行
DELETE ANY TABLE	删除任何模式中的任何表或视图中的行
表空间	
CREATE TABLESPACE	创建表空间；不管用户有何操作系统特权，经由 Oracle 把文件添加到操作系统中
ALTER TABLESPACE	改变表空间；不管用户有何操作系统特权，经由 Oracle 把文件添加到操作系统中
MANAGE TABLESPACE	使任何表空间脱机，使任何表空间联机。开始和结束对任何表空间的备份
DROP TABLESPACE	删除表空间
UNLIMITED TABLESPACE	使用任何没有数量限制的表空间。此特权忽略了所分配的任何具体定额。假如被取消的话，被授权者的模式对象仍然保留，但是进一步的表空间分配被拒绝，除非这一分配是具体的表空间定额允许的。此系统特权仅可以授予用户，而不授予角色。一般而言，应分配具体的表空间定额，而不授予此系统特权
事务	
FORCE TRANSACTION	强迫提交或回滚本地数据库中悬而未决的自有的分布式事务
FORCE ANY TRANSACTION	强迫提交或回滚本地数据库中悬而未决的任何分布式事务
触发器	
CREATE TRIGGER	在自有模式中创建触发器
CREATE ANY TRIGGER	在任何模式中创建与任何模式的任何表相关的任何触发器
ALTER ANY TRIGGER	启用、停用或编译任何模式中的任何触发器
DROP ANY TRIGGER	删除任何模式中的任何触发器
用户	
CREATE ANY USER	创建用户；分配任意表空间上的定额，设置缺省和临时表空间，指定一个环境资源文件（在 CREATE USER 语句中）
BECOME ANY USER	成为另一个用户（这是任何一个执行完全数据库导入的用户所需要的）
ALTER USER	改变其他用户；修改任意用户的口令或验证方法，分配表空间定额，设置缺省或临时表空间，在 ALTER USER 语句中指定环境资源文件与缺省角色（不必改变自有口令）
DROP USER	删除另一个用户
视图	
CREATE VIEW	在自有模式中创建视图
CREATE ANY VIEW	在任意模式中创建视图。要在另一个用户模式中创建视图，你必须具有 CREATE ANY VIEW 特权，拥有者必须在该视图引用的对象上具有所需的特权
DROP ANY VIEW	删除任意模式中的任意视图

系统权限的授予命令为 GRANT，例如把创建任何表视图的权限授予 scott 用户：

```
GRANT create any view to scott;
```

系统权限的回收命令为 REVOKE，例如将 create any view 权限从 scott 用户手中收回：

```
REVOKE create any view from scott;
```

3.3.2 对象权限

对象特权允许对数据库对象的存取与维护，与系统权限类似，对象权限可以直接授予用户，也可以授予角色。

对 象 特 权	所能实现的操作
ALTER	ALTER对象（表或序列）
DELETE	DELETE FROM对象（表或视图）
EXECUTE	EXECUTE对象（过程或函数）；引用公共包变量
INDEX	CREATE INDEX ON对象（仅有表）
INSERT	INSERT INTO对象（表或视图）
REFERENCES	在对象（仅有表）上定义一个 FOREIGN KEY 完整性约束的 CREATE 或 ALTER TABLE 语句
SELECT	SELECT...FROM对象（表、视图或快照）；使用序列的 SQL 语句
UPDATE	UPDATE对象（表或视图）

例如给用户 scott 授予表 emp 的 select 和 insert 权限：

GRANT select insert ON emp TO scott;

将 emp 表上的 Select 权限从 scott 手中回收：

REVOKE select ON emp FROM scott;

3.4 角色

角色是将权限和角色组成组，这样可将他们同时授权给用户或同时从用户中取消。每个用户可启用或禁用角色。

ORACLE 数据库自动定义了一些系统角色，例如 CONNECT 和 RESOURCE 角色分别是最终用户和开发人员所要求的基本系统权限。

通常用户设计和定义自己的角色来满足数据库安全管理的需要。

3.4.1 创建角色

具有 create role 系统权限的用户可以使用 create role 语句创建角色。角色刚创建时还没有权限，需要给角色赋予权限或角色。角色不包含在任何用户的模式中。例如：

create role role_name

identified by password

其中 identified by 指定了角色的授权方式，启用该角色时需要提供。用户可使用 alter role 语句设置或改变角色授权方式。为了改变角色的授权方式，用户需要拥有 alter any role 的系统权限或已用 admin option 授权该角色。

3.4.2 授予和取消角色

使用 grant 语句将角色授予用户或另外一个角色。例如：

```
grant role_name to user1;
```

```
grant role_name to user2 with admin option;
```

经过执行如上两条语句，user1 具有了 role_name 角色。user2 不但具有 role_name 角色，还可以授权、取消或删除 role_name 角色。

取消角色通过 revoke 命令完成。例如：

```
revoke role_name from user1
```

3.4.3 启用角色

如果授权一个口令保护的角色给用户，用户可以在 set role 语句中通过提供正确的口令启用或禁用一个角色。例如：

set role role_name identified by password 可以启用角色 role_name。命令 set role none 实现禁用所有角色。

3.5 查询与权限和角色有关的视图

有关已经授予权限和角色的信息保存在数据字典中，可以通过数据字典视图访问这些信息。这些数据字典名称及含义见表 3.5-1。

表 3.5-1 与权限和角色有关的数据字典视图

数据字典视图	内 容
DBA_ROLES	角色名及其口令状态
DBA_ROLES_PRIVS	已被授予角色的用户
DBA_SYS_PRIVS	已被授予系统权限的用户
DBA_TAB_PRIVS	已被授予表中权限的用户
DBA_COL_PRIVS	已被授予列中权限的用户
ROLE_ROLE_PRIVS	已被授给其他角色的角色
ROLE_SYS_PRIVS	已被授给角色的系统权限
ROLE_TAB_PRIVS	已被授给角色的表权限

例如，可能想要显示哪些系统权限已被授给哪些角色。在这种情况下，下列查询将会显示这些信息：

```

select
    Role,                /*Name of the role*/
    Privilege,            /*System privilege*/
    Admin_Option         /*Was admin option granted?*/
from ROLE_SYS_PRIVS;

```

若要检索授予用户的表权限，需找出两种类型的授权：给用户的显式授权和通过角色的授权。如下面清单所示，若要查看显式授予的权限，可查询 DBA_TAB_PRIVS 视图：

```

select
    Grantee,              /*Recipient of the grant*/
    Owner,                /*Owner of the object*/
    Table_Name,           /*Name of the object*/
    Grantor,              /*User who made the grant*/
    Privilege,            /*Privilege granted*/
    Grantable             /*Was admin option granted?*/
from DBA_TAB_PRIVS;

```

若要查看通过角色授予的表权限，可在 DBA_ROLE_PRIVS 中查找用户的记录并与角色的表权限（列在 ROLE_TAB_PRIVS 中）进行比较。

```

select
    DBA_ROLE_PRIVS.Grantee, /*Recipient of the grant*/
    ROLE_TAB_PRIVS.Owner,   /*Owner of the object*/
    ROLE_TAB_PRIVS.Table_Name, /*Name of the object*/
    ROLE_TAB_PRIVS.Privilege, /*Privilege granted*/
    ROLE_TAB_PRIVS.Grantable /*Was admin option granted?*/
from DBA_ROLE_PRIVS, ROLE_TAB_PRIVS
where DBA_ROLE_PRIVS.Granted_Role = ROLE_TAB_PRIVS.Role
      and DBA_ROLE_PRIVS.Grantee = 'some username';

```

这个查询将针对一个特定用户，检索由角色授予的表权限。

3.6 概要文件

创建用户时可以指定概要文件 profile。用户的 Profile 文件限制数据库的使用情况和实例使用的资源，创建用户时如果不指定 profile，系统会给用户分配一个缺省的 profile。

3.6.1 创建 profile

创建 profile 的语法如下：

```
CREATE PROFILE profile_name LIMIT  
[SESSIONS_PER_USER max_value]  
[CPU_PER_SESSION max_value]  
[CPU_PER_CALL max_value]  
[CONNECT_TIME max_value]  
[IDLE_TIME max_value]  
[LOGICAL_READS_PER_SESSION max_value]  
[LOGICAL_READS_PER_CALL max_value]  
[COMPOSITE_LIMIT max_value]  
[PRIVATE_SGA max_bytes]
```

其中，

profile_name 是 profile 的名字；max_value 是整数值、UNLIMITED 或 DEFAULT；max_bytes 是整数值，后面紧跟 K 或 M，也可以是 UNLIMITED 或 DEFAULT。命令参数的含义见表 3.6-1。

表 3.6-1 create profile 中各参数的含义

资 源	描 述
SESSION_PER_USER	在一个实例中，一个用户可以同时拥有的会话数量
CPU_PER_SESSION	一个会话可以使用的 CPU时间，以百分之一秒为单位
CPU_PER_CALL	语法分析、执行或获取可以使用的CPU时间，以百分之一秒为单位
CONNECT_TIME	一个会话可以连接到一个数据库的分钟数
IDLE_TIME	一个会话可以连接到一个数据库而没有激活使用的分钟数
LOGICAL_READS_PER_SESSION	可以在一个会话中读取的数据库块数
LOGICAL_READS_PER_CALL	在语法分析、执行或获取期间可以读取的数据库块数
PRIVATE_SGA	在SGA的SQL共享池中，一个会话可以分配的私有空间量(对于MTS)
COMPOSITE_LIMIT	一个基于前面的限制的复合限制
FAILED_LOGIN_ATTEMPTS	将引起一个帐户被锁定的连续注册失败的次数
PASSWORD_LIFE_TIME	一个口令在其终止前可以使用的天数
PASSWORD_REUSE_TIME	一个口令在能够被重新使用之前所必须经过的天数
PASSWORD_REUSE_MAX	一个口令在能够被重新使用之前必须改变的次数
PASSWORD_LOCK_TIME	如果超过FAILED_LOGIN_ATTEMPTS设置值，一个帐户将被锁定的天数
PASSWORD_GRACE_TIME	以天为单位的“宽限时间”。在宽限期内，在口令达到PASSWORD_LOGIN_TIME设置值时，仍能对其修改
PASSWORD_VERIFY_FUNCTION	一个函数名，用于判断口令的复杂性：由 Oracle提供一个口令并可以编辑

例如通过下面语句创建一个有关密码属性设置的 profile 文件：

```
CREATE PROFILE grace_5 LIMIT
FAILED_LOGIN_ATTEMPTS 3
PASSWORD_LOCK_TIME UNLIMITED
PASSWORD_LIFE_TIME 30
PASSWORD_REUSE_TIME 30
PASSWORD_GRACE_TIME 5;
```

3.6.2 查询用户和 profile 相关信息的数据字典视图

下面有关数据库用户和 profile 的视图非常有用。

- (1) all_users
列举当前可见的所有数据库用户。
- (2) dba_users
描述数据库的所有用户。
- (3) User_users

和 `dba_users` 有相同的列，但只描述当前用户。

(4) `dba_ts_quotas`

描述所有用户的表空间配额。

(5) `user_ts_quotas`

和 `dba_ts_quotas` 有相同的列，但只描述当前用户的表空间配额的相关信息。

(6) `user_password_limits`

描述指定给用户的 `profile` 中的口令限制情况。

(7) `user_resource_limits`

描述指定给用户的 `profile` 中的资源限制情况。

(8) `dba_profiles`

显示所有 `profile` 文件及其限制。

(9) `resource_cost`

列举每个资源的价值。

(10) `v$session`

列举每个当前会话的信息，包括用户名。

第4章 启动和关闭数据库

4.1 启动数据库

启动 ORACLE 和关闭 ORACLE 的命令分别是 startup 和 shutdown，启动以及关闭的过程如图 4.1-1 所示。

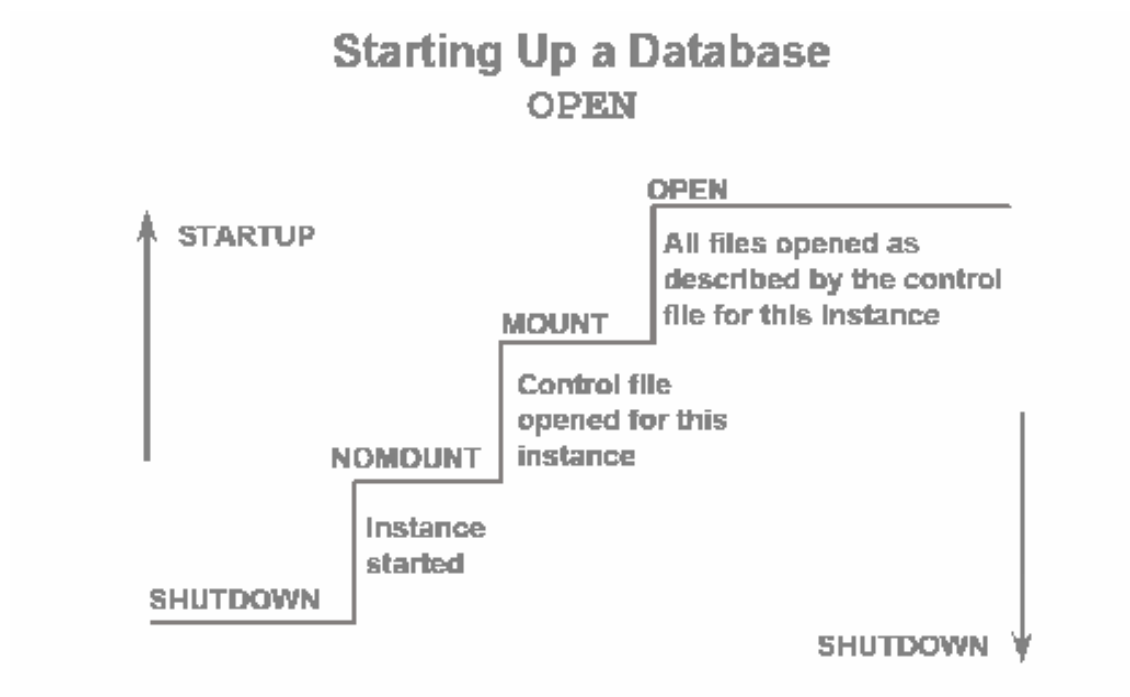


图 4.1-1 启动和关闭 ORACLE 的过程

4.1.1 正常启动

正常启动 ORACLE SERVER 的命令是 startup normal，因为这是缺省的启动模式，所以 normal 参数可以省略。如图 4.1-2 所示。在 UNIX 系统中，以 oracle 用户登陆后运行 ORACLE 的服务器管理器程序 svrmgrl，然后以 internal 用户连接执行 startup 即可以启动 ORACLE SERVER。

```
Oracle Server Manager Release 3.1.5.0.0 - Production
(c) Copyright 1997, Oracle Corporation. All Rights Reserved.
Oracle8i Enterprise Edition Release 8.1.5.0.0 - Production
With the Partitioning and Java options
PL/SQL Release 8.1.5.0.0 - Production
SVRMGR> connect internal/oracle
SVRMGR> startup
ORACLE instance started.
Total System Global Area      8030448 bytes
Fixed Size                    44584 bytes
Variable Size                  7510728 bytes
Database Buffers               409600 bytes
Redo Buffers                   65536 bytes
Database mounted.
Database opened.
SVRMGR>
```

图 4.1-2 正常启动 ORACLE

4.1.2 非加载启动

startup nomount 命令可以实现非加载启动。

非加载启动模式用于重建控制文件或重建数据库。由于数据库未被打开，所以不允许用户访问。启动服务器管理器后，按如下步骤操作：

1. 输入 “connect internal ”。
2. 输入 “startup nomount ”， 从 Oracle 返回的信息如图 4.1-2 所示。

```
ORACLE instance started.
Total System Global Area      8030448 bytes
Fixed Size                    44584 bytes
Variable Size                  7510728 bytes
Database Buffers               409600 bytes
Redo Buffers                   65536 bytes
SVRMGR>
```

图 4.1-2 非加载模式启动 ORACLE

执行非加载启动，ORACLE SERVER 会进行如下动作：

- (1) 读取初始化参数文件 `initsid.ora`
- (2) 启动 ORACLE 后台进程
- (3) 根据初始化参数文件分配 SGA
- (4) 根据初始化参数文件打开跟踪文件

4.1.3 加载启动

`startup mount` 命令可以实现加载启动。

加载启动模式用于改变数据库的归档状态或执行恢复操作。此时数据库尚未被打开，因而用户不能访问。启动服务器管理器后，操作步骤如下：

1. 输入 “`connect internal` ”。
2. 输入 “`startup mount` ”， 从 Oracle 返回的信息如图 4.1-3 所示。

```
ORACLE instance started.  
Total System Global Area      8030448 bytes  
Fixed Size                     44584 bytes  
Variable Size                  7510728 bytes  
Database Buffers               409600 bytes  
Redo Buffers                   65536 bytes  
Database mounted.  
SVRMGR>
```

图 4.1-3 加载模式启动 ORACLE

执行加载模式启动时，ORACLE SERVER 首先完成非加载模式启动的过程，然后进一步打开和读取控制文件。

4.1.4 约束启动

`startup restrict` 命令实现用约束模式启动数据库。该模式启动数据库只运行具有 `CREATE SESSION` 和 `RESTRICTED SESSION` 系统权限的用户访问数据库，而非约束模式下用户具有 `CREATE SESSION` 系统权限就可以访问数据库。

使用 `alter system enable restricted session` 和 `alter system disable restricted session` 命令可以实现在数据库启动之后限制对打开数据库的访问（将数据库更改为约束模式或非约束模式）。

4.1.5 强制数据库启动

在特殊的环境下，用户可能在试图启动数据库时遇到问题。例如下面两种情况：

- (1) 用户不能用 `shutdown normal`、`shutdown immediate`、`shutdown transactional` 命令来关闭数据库
- (2) 在启动实例时遇到问题

如果上述问题之一发生，用户可以使用带 `startup force` 命令启动数据库。如果一个实例正在运行，`startup force` 将在重新启动前用 `abort` 模式将数据库关闭。

4.2 更改数据库的可用性

4.2.1 将数据库装入实例

如果执行了 `startup nomount` 启动实例，则数据库还没有被装载，这时可以通过 `alter database mount` 命令装载数据库，即将数据库与实例关连。

4.2.2 打开一个关闭的数据库

在数据库被装载但没有被打开的情况下，通过执行命令 `alter database open` 可以将数据库打开。

4.3 关闭数据库

图 4.3-1 表示了一个银行帐号到另一个银行帐号的基金传送过程中的不同 `shutdown` 命令的影响。

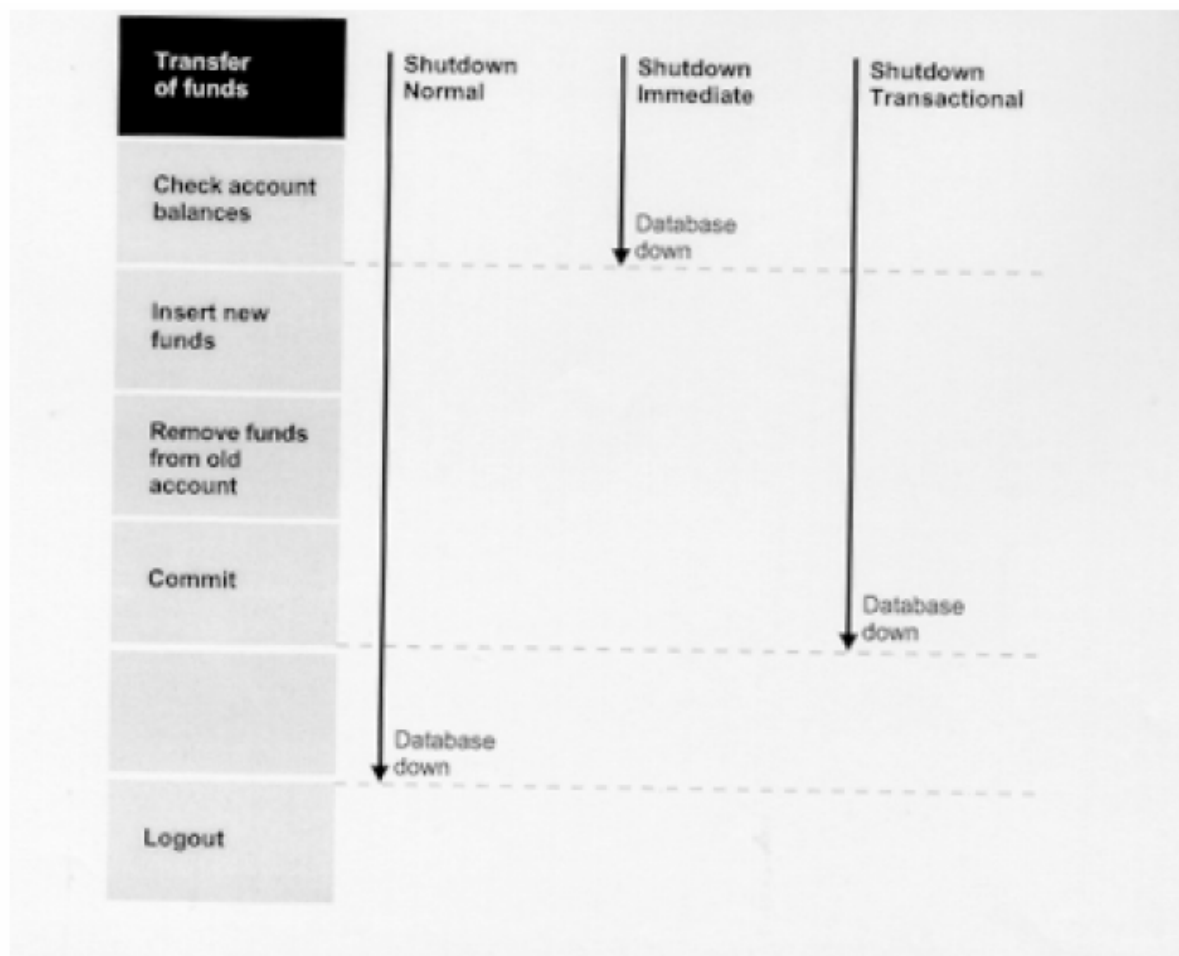


图 4.3-1 以不同形式关闭数据库的影响

4.3.1 用 normal 选项关闭

通过 `shutdown normal` 命令正常关闭数据库。具体过程是：

- (1) 在语句发出后不允许新的连接
- (2) 在数据库关闭之前，ORACLE 等待所有当前与数据库连接的用户断开连接
- (3) 数据库的下一次启动不需要进行实例恢复

4.3.2 用 immediate 选项关闭

使用 `shutdown immediate` 命令立即关闭数据库。具体过程是：

- (1) 在语句发出后不允许新的连接
- (2) 当前未完成（未 commit）的事务回退

- (3) ORACLE 断开用户当前的数据库连接
- (4) 数据库的下一启动不需要进行实例恢复

4.3.3 用 transactional 选项关闭

使用 shutdown transactional 命令关闭数据库的具体过程是：

- (1) 在语句发出后不允许新的连接
- (2) 已经连接的客户不能启动新的事务，如果客户端试图启动一个新的事务，他们将被断开连接
- (3) 等待当前激活的所有事务提交完成
- (4) 客户事务提交完成后被断开连接
- (5) 数据库的下一启动不需要进行实例恢复

4.3.4 用 abort 选项关闭

使用 shutdown abort 命令关闭数据库的具体过程是：

- (1) ORACLE 处理的当前客户端 SQL 语句立即中断
- (2) 未提交的事务将不会被回退
- (3) ORACLE 立即断开所有用户的连接
- (4) 数据库的下一启动需要进行实例恢复

4.4 通过诊断文件监控数据库运行

ORACLE 实例运行时，诊断文件记载了一些有用的关键信息，这些信息可以供数据库管理员在日常维护和故障分析时使用。

4.4.1 后台进程运行跟踪文件

当 ORACLE 的后台进程遇到了错误，会记录在后台进程跟踪文件中。初始化参数文件的 background_dump_dest 参数指定了后台进程运行跟踪文件的位置。

4.4.2 警告日志文件

除了跟踪文件外，Oracle 还有一个称作警告日志（alert log）的文件 alertsid.log，警告日志记录数据库文件运行中主要事件的命令及结果。例如，表空间的创建、

重做日志的转换、操作系统的恢复、数据库的建立等信息都记录在警告日志中。警告日志是数据库每日管理的重要资源，当需要查找主要失败原因时，跟踪文件就非常有用。

应经常监控警告日志。警告日志的条目将通知你数据库操作期间遇到的任何问题。为使警告日志便于使用，最好是每天能自动对其重新命名。例如，如果警告日志称作 `alert_orcl.log`，可以对它重新命名，以便其文件名包括当前日期。下次 Oracle 要写该警告日志时，将找不到具有 `alert_orcl.log` 文件名的文件，因此数据库将创建一个新的文件名。这样，除了有以前的警告日志外，还有一个当前的警告日志文件。用这种方式区分警告日志条目就可以使对警告日志条目的分析更有效。

初始化参数文件中的 `background_dump_dest` 参数指定了 `alertsid.log` 的位置。

4.4.3 用户跟踪文件

用户跟踪文件由用户进程产生。初始化参数文件中的 `user_dump_dest` 参数指定了用户跟踪文件的位置。该文件可以记载设定跟踪的 SQL 语句的统计信息，也可以记载用户会话发生的错误等信息。

在会话级设定用户跟踪的命令为 `alter session set SQL_TRACE=TRUE`。在实例级设定用户跟踪的方法为修改初始化参数文件中的参数 `SQL_TRACE=TRUE`。

第5章 ORACLE NET

5.1 ORACLE NET 体系结构

ORACLE 使用 ORACLE NET 解决网络环境下的数据库应用。网络环境下的 ORACLE 数据库应用常用的有客户机/服务器结构和瘦客户机体系结构两种。

5.1.1 客户机/服务器结构

使用客户机 / 服务器结构（也称为双层体系结构）允许在两台机器之间分布负载。称为客户机的第一台机器支持发出数据请求的应用程序。后端机器称为服务器，数据库就驻留在该机器中。客户机负责表现数据，而数据库服务器则专用于支持查询，而不是应用程序。

当客户机向服务器发出数据库请求时，服务器接收并执行传送给它的 SQL 语句，然后把 SQL 语句的执行结果和要返回的错误信息返回客户机。客户机 / 服务器需要一个具有大型硬盘驱动器（2GB 以上）和大内存需求（通常超过 64MB 的 RAM）的稳固的工作站。由于所需的资源较多，使得这种客户机 / 服务器配置有时被叫做“胖客户机”体系结构，这种结构以及 ORACLE 实现原理如图 5.1-1 所示。

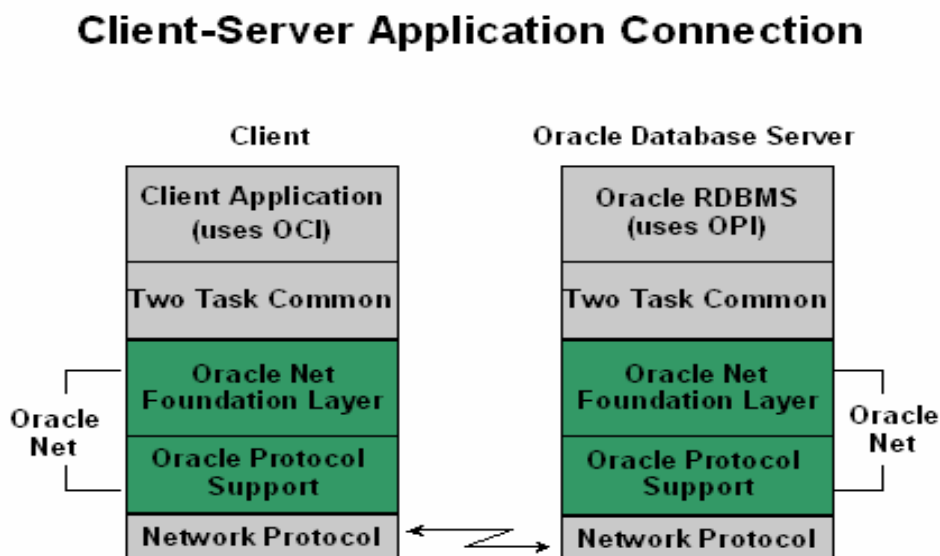


图 5.1-1 胖客户机结构的 ORACLE 实现原理

ORACLE NET 是驻留在 ORACLE 客户端和数据库服务端的 ORACLE 软件组件，它支持 ORACLE 客户端与数据库服务器之间的网络连接。当客户端发起连接请求时，请求的数据通过自顶向下的协议栈处理，然后利用网络协议如 TCP/IP 发送到数据库服务器，数据库服务器端对收到的数据进行自底向上的协议栈的处理。利用相同的过程，ORACLE 服务器将处理结果发给客户端。

客户端应用（例如 SQL*PLUS）调用 OCI（ORACLE CALL INTERFACE）发起请求，OCI 是 ORACLE 的软件组件，它被客户端所调用，是客户机应用请求和数据库服务器可以理解的 SQL 命令的桥梁。客户端应用相当于 OSI 模型的应用层。

TWO TASK COMMON（简称 TTC）提供了客户端服务器端之间字符集和数据类型的转换，相当于 OSI 模型的表示层。

ORACLE NET FOUNDATION 层负责建立和维持 ORACLE 客户端和服务端之间的会话连接，相当于 OSI 模型的会话层。

ORACLE PROTOCOL SUPPORT 层负责将 ORACLE NET 的功能与工业标准的协议如 TCP/IP 或命令管道（named pipe）等进行映射。

OPI（ORACLE PROGRAMING INTERFACE）是服务器端调用的接口，与客户端调用的 OCI 功能上对应。

5.1.2 瘦客户机结构

基于 SQL*Net 的更广泛使用的体系结构是“瘦客户机”结构，也称为三层体系结构。在这种配置中，应用程序被放置在与数据库服务器分开的服务器上并使用 Java 脚本文件来执行。客户机的资源需求变得很小，价格显著降低。应用程序变成独立于数据库，如图 5.1-2 所示。对这种结构的 ORACLE 实现原理如图 5.1-3 所示。

Web Client Application Connection: Application Web Server Middle-Tier

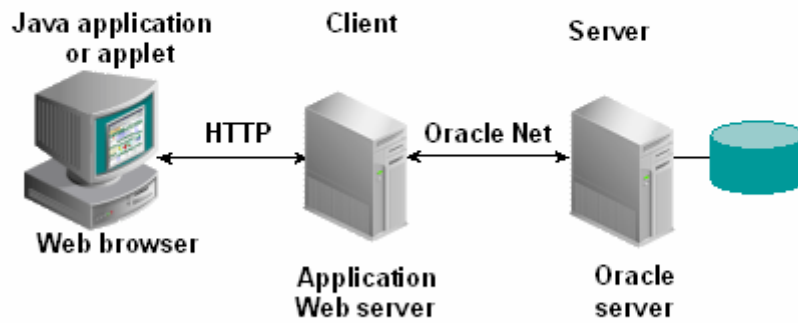


图 5.1-2 瘦客户机结构的 ORACLE 应用

Web Client Application Connection: Java Application Client

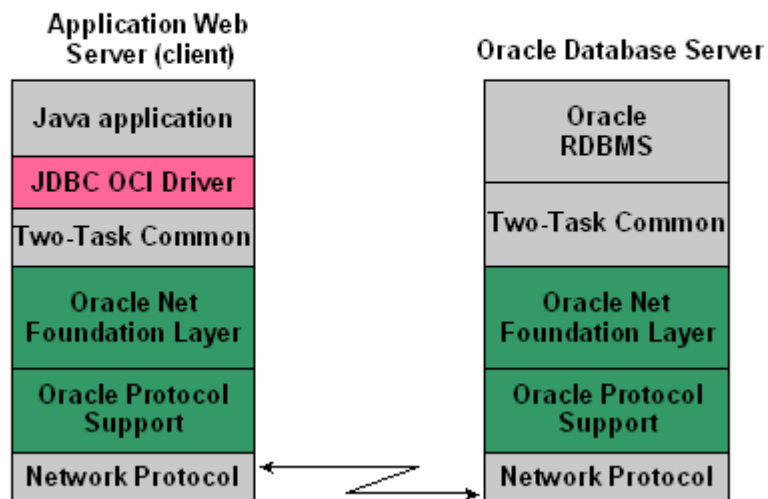


图 5.1-3 瘦客户机结构的 ORACLE 实现原理

5.2 ORACLE NET 服务器端

5.2.1 监听器进程

在 ORACLE 服务器端运行监听器进程。当有客户端请求到达服务器时，监听器进程完成如下任务：

- (1) 受理客户端的请求
- (2) 产生相应的服务器进程
- (3) 将受理的客户端连接转移到服务器进程去受理

这一过程如图 5.2-1 所示。

Spawn and Bequeath Connections

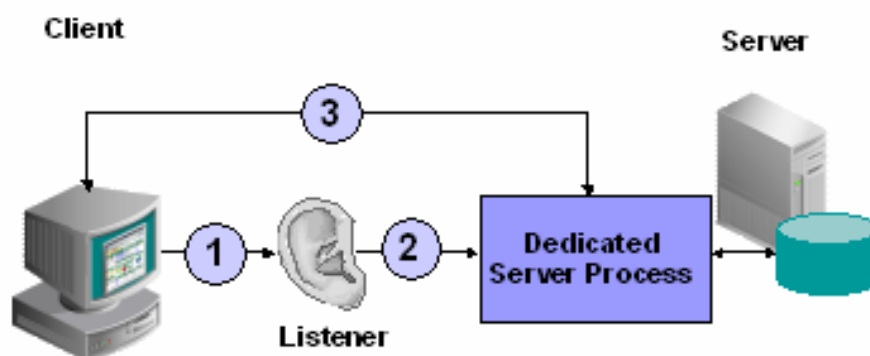


图 5.2-1 监听器进程的作用

5.2.2 listener.ora 文件

网络上的每一个数据库服务器都必须包含一个 listener.ora 文件，该文件列出机器中所有监听进程的名字和地址以及它们所支持的实例。监听程序进程接收来自 ORACLE NET 客户机的连接。

一个 listener.ora 文件包含四个部分：

标题节

地址列表

实例定义

操作参数

listener.ora 文件可以由 Net8 Assistant 工具自动生成。只要遵循其语法规则，就可对结果文件进行编辑。下面列出了一个 listener.ora 文件的例子：

```

1. LISTENER =
2. {ADDRESS_LIST =
3.   {ADDRESS= (PROTOCOL= TCP) (Host= stc-
      sun02) (Port= 1521))}
4. SID_LIST_LISTENER =
5.   {SID_LIST =
6.     {SID_DESC =
7.       {ORACLE_HOME= /home/oracle}
8.       {GLOBAL_DBNAME = ORCL.us.oracle.com}
9.       {SID_NAME = ORCL}}}
```

5.3 UNIX 环境下的监听器进程

5.3.1 启动和停止监听器进程

监听器进程由 lsnrctl (Listener Control Utility, 监听器控制实用程序) 控制，通过 lsnrctl 命令来执行。使用 lsnrctl start 命令来启动缺省的监听器（缺省监听器命名为 listener）。

如果想启动另一个监听器，则可以在 lsnrctl 命令中包括该监听器的名称作为第二个参数。例如，如果创建了一个叫做 my_listener 的监听器，则可通过命令 lsnrctl start my_listener 来启动它。

5.3.2 查看监听程序运行状况

执行 lsnrctl status 命令可以查看监听器的运行状况。

5.3.3 lsnrctl 命令的参数

lsnrctl 命令的常用参数如下：

1. change_password

给监听程序设置新口令，系统会提示输入监听程序的旧口令。

2. exit

退出 lsnrctl。

3. help

显示 lsnrctl 命令选项的列表，也可以通过 help set 和 help show 命令查看附加选项。

4. quit

退出 lsnrctl。

5. reload

允许在启动监听程序之后修改该监听程序。

6. save_config

创建现有的 listener.ora 文件的备份，然后用已由 lsnrctl 更改的参数来更新 listener.ora 文件。

7. set

设置参数值，这些参数包括：

(1) connect_timeout

以秒为单位，监听程序启动之后等待合法连接请求的时间。

(2) current_listener

改变其参数正被设置或显示的监听程序进程。

(3) log_directory

监听程序日志文件的目录。

(4) log_file

监听程序日志文件的名称。

(5) log_status

日志记录是 on 还是 off。

(6) password

监听程序口令。

(7) save_config_on_stop

当退出 lsnrctl 时把配置变化保存到 listener.ora 文件。

(8) startup_waittime

监听程序在响应 lsnrctl start 命令之前的休眠秒数。

(9) `trc_directory`

监听程序跟踪文件的目录。

(10) `trc_file`

监听程序跟踪文件的名称。

(11) `show`

显示当前参数设置，这些选项与除 `password` 命令外的 `set` 选项一样。

(12) `start`

启动监听程序。

(13) `status`

提供有关监听程序的状态信息，包括它的启动时间、参数文件名、日志文件和它支持的服务。该命令可用来查询远程服务器上的监听程序的状态。

(14) `stop`

停止监听程序。

(15) `trace`

设置监听程序的跟踪。

(16) `version`

显示监听程序、TNS 和协议适配器的版本信息。

5.4 连接描述符

客户机为了和服务器连接，都必须先和服务器上的监听进程联络，ORACLE 通过 `tnsnames.ora` 文件中的连接描述符来说明连接信息。

一般 `tnsnames.ora` 是建立在客户机上的。如果是客户机/服务器结构，整个网络上只有一台机器安装了 ORACLE 数据库服务器，那么只需在每个要访问 ORACLE 服务器的客户机上定义该文件，在服务器上无需定义。但是如果网络上有多台机器均安装了 ORACLE 数据库服务器，并且服务器之间有数据共享的要求，那么在每台服务器上都必须定义该文件。

`tnsnames.ora` 文件缺省放在 `/ORACLE_HOME/network/admin` 目录下。

下面给出 tnsnames.ora 文件的一个样例条目, 该例把 loc 的服务名赋予上面给出的连接描述符。

```
LOC = (DESCRIPTION=
      (ADDRESS=
        (PROTOCOL=TCP)
        (HOST=HQ)
        (PORT=1521))
      (CONNECT DATA=
        (SID=loc)))
```

这样, 想连接到 HQ 服务器上 loc 实例的用户现在可以使用 LOC 连接描述符:
sqlplus user1/password@LOC;

第6章 数据字典

数据字典（data dictionary）是存储在数据库中的所有对象信息的知识库，Oracle 使用数据字典获取对象信息和安全信息，而用户和数据库系统管理员用它来查阅数据库信息。

它保存着数据库中数据库对象和段的信息，例如表、视图、索引、包以及过程，它还保存着关于如用户、权限、角色、约束等的信息。数据字典是只读的，用户决不要去尝试对任何数据字典表中的任何信息进行手工更新或改动。它由四部分组成：内部 RDBMS（X\$）表、数据字典表、动态性能(V\$) 视图和数据字典视图。

6.1 内部 RDBMS（X\$）表

Oracle 数据库的心脏就是即所谓的内部 RDBMS（X\$）表，这些表被 Oracle 用于跟踪内部数据库信息。X\$表是加密命名的、非文献性的表。大多数 X\$表被设计成不能被数据库系统管理员或用户直接使用。

要解密在某个 X\$表中贮存的信息，简单的方法是从一个已知的数据字典表（视图）倒推寻找。

SQL*Plus 自动跟踪特性对这项工作是非常有用的。例如，要想知道 V\$SGA STAT 中的信息到底存放在哪里，可以执行下面的分析步骤：

- （1）作为 SYS 用户（或者一个对 X\$和 V\$表有明确访问权限的帐户）登录到 SQL*Plus。如果所登录的模式中不存在 PLAN_TABLE，可以通过运行 \$ORACLE_HOME/rdbms/admin/UTLXPLAN.sql 语句来建立一个。
- （2）执行下列 SQL*Plus 命令：SET AUTOTRACE ON。
- （3）针对有你感兴趣内容的表执行一条查询，把 WHERE 子句设为一个永远不会为真的值，以便没有任何行能够返回：SELECT * FROM V\$sgastat WHERE 0=1;

于是自动追踪返回类似下列的输出：

```
Execution Plan
0  SELECT STATEMENT Optimizer=CHOOSE
1  0      FILTER
2  1      FIXED TABLE (FULL) OF 'X$KSMSS'
```


6.2 数据字典表

数据字典表（data dictionary table）存储表、索引、约束以及所有其他数据库结构的信息。它们属于 SYS，通过运行 SQL.BSQ 脚本来创建（在数据库创建时自动发生）。通过它们名字后面的美元（\$）符号（例如 tab\$、seg\$、cons\$等），可以很容易地将它们辨认出来。在数据字典视图中可以找到数据字典表中的大部分信息，但是一些应用或查询也可以从使用包含在基表中的信息中获益。

6.3 动态性能视图

动态性能（V\$）视图（dynamic performance（V\$）view）是 Oracle 数据库系统管理员的主要依靠，这些视图包含了大量数据库运行时的性能和统计信息。它们还具有易读性（与 X\$表相反），从而能够被数据库系统管理员用于诊断和解决问题。

6.4 数据字典视图

数据字典视图是在 X\$和数据字典表上创建的视图，也就意味着它们能被终端用户和数据库系统管理员使用和查询，它们被分成三类： DBA_、ALL_和 USER_视图。DBA_视图包含了数据库所有对象的信息。例如， DBA_TABLES 包含所有已创建表的信息， ALL_视图包含了用户查询表时可以访问的所有对象的信息， USER_视图包含了用户查询表时所拥有的全部对象的信息。

6.5 数据字典举例

6.5.1 数据字典总体信息

查询 dictionary 视图可以获得可查询数据字典的列表。命令 describe dictionary 可以获得 dictionary 的结构，从而使用 select 语句查询想要的列。查询 dict_columns 数据字典可以获得数据字典列的详细信息。

6.5.2 模式对象信息

DBA_TABLES 、 DBA_INDEXES 、 DBA_TAB_COLUMNS, 以及 DBA_CONSTRAINTS 描述了有关模式对象的信息。

6.5.3 空间分配信息

DBA_SEGMENTS 和 DBA_EXTENTS 提供了关于存储空间分配的信息。

6.5.4 数据库结构信息

DBA_TABLESPACES 和 DBA_DATA_FILES 提供了数据库结构方面的信息。

第7章 管理数据库存储

7.1 ORACLE 数据库结构

ORACLE 数据库结构分为物理结构和逻辑结构。其中，物理结构由控制文件、数据文件、重作日志文件组成。逻辑结构由表空间、段、区、ORACLE 块组成。这些概念之间的关系可以通过图 7.1-1 反映出来。

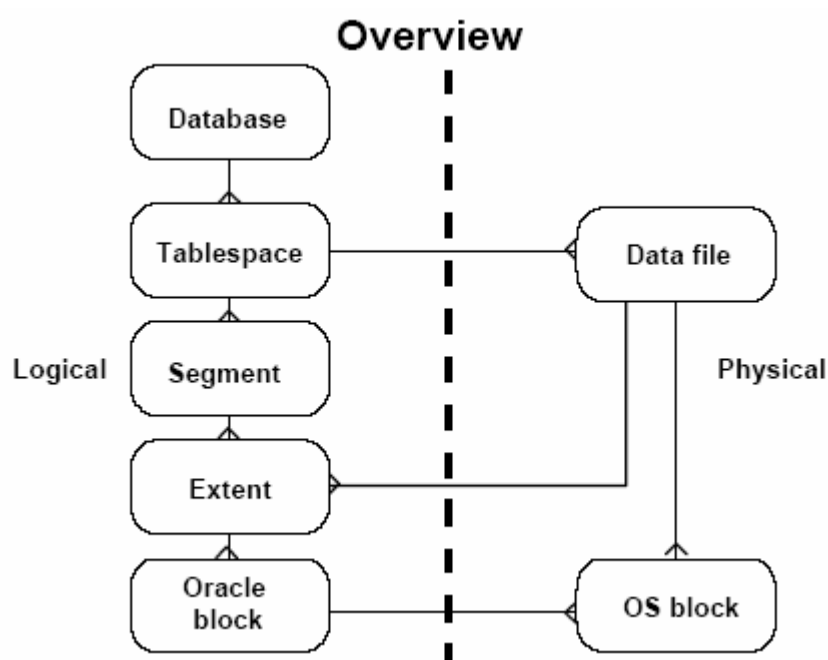


图 7.1-1 ORACLE 数据库结构

7.2 ORACLE 块

7.2.1 ORACLE 块结构

ORACLE 服务器可寻址的最小存储单元是 ORACLE 块。所有的数据库段都由 ORACLE 块组成，无论段是一个表、索引、簇或者其他对象，块结构是相同的。设计一个性能最优的数据库要从对 ORACLE 块适当的配置和管理开始。

每个 ORACLE 块都是由下面三部分组成的：

块头

数据存储区

自由空间区

块头包含有关块的信息，即什么类型的段数据存储在块中，什么段在块中有数据，块地址以及指向存储在其中的实际行的指针。块头由一个固定部分和一个可变部分组成，在块中块头通常是 85 到 100 字节。

在 ORACLE 块中，数据存储区和自由空间区是彼此直接相关的。数据区域是块中实际存储行的地方。保留的自由空间区是一个区域，被定义为总的可用空间的百分数，被保留用于存储有关在块中的行将来更新的信息。管理块的数据存储区和自由空间区 ORACLE 块管理所主要关注的主要问题。

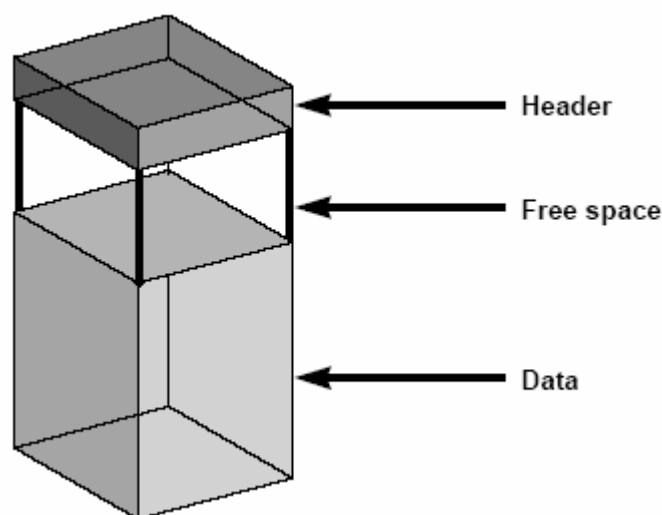


图 7.2-1 ORACLE 块结构

7.2.2 理解 PCTFREE 与 PCTUSED

PCTFREE 规定了在块中保留的用于更新操作所需的自由空间的百分比。PCTUSED 规定了最小的已使用空间的百分比。已使用空间百分比低于 PCTUSED 时，块被加入到自由列表中。自由列表中的块可以 insert 操作

PCTFREE 和 PCTUSED 是应用于段的两个存储参数。当 ORACLE 向数据库中写信息时，它必须首先在一个段的分配区中找到一个或更多块来存储信息。ORACLE 保留了块的一个列表，这些块对每个段来说都是自由的，称为自由列表。ORACLE 使用 PCTFREE 和 PCTUSED 参数的组合确定块何时、何时没有足够的空间接受新信息。

如果在块中自由空间的百分数比 **PCTFREE** 参数大得多，它就要被添加到段的自由表中以便它可以用来存储新信息，例如对表进行的插入操作。当自由空间的百分数低于 **PCTFREE** 时，块就被认为是满的，**ORACLE** 就把它从段的自由表中移出来。

当块中已经使用空间的百分数低于 **PCTUSED** 时，**ORACLE** 就把该块加到段的自由表中，这样它就可以再次被用来存储新信息。这种方法允许 **ORACLE** 保持足够的额外空间用于行增长，而不需要跨过超过一个块的空间。保持行被限制在一个单独的块中有助于使数据库以最高性能运行。

PCTFREE 和 **PCTUSED** 的值永远都不能等于 100%。如果一个段被这样配置，块很可能将被连续地从自由表中取出，并且在每个数据子处理中又被放置到自由表中。**Oracle** 为 **PCTFREE** 和 **PCTUSED** 设置的默认值分别是 10 和 40。

PCTFREE 和 **PCTUSED** 可以被微调，以满足段的存储需要。例如，一个永远都不会被修改的表可以安全地把 **PCTFREE** 设为 0，这在每个块中都节约了空间。另一方面，如果一个表包含了将来肯定要被改变的列，那么就可以用高一点的 **PCTFREE** 设置以避免行链接或者行转移。

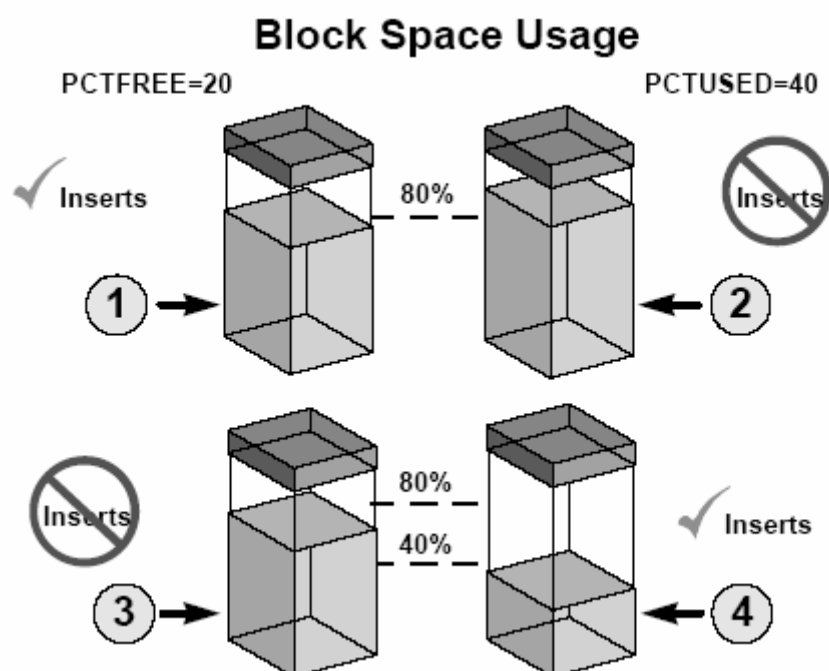


图 7.2-2 块空间使用参数的含义

图 7.2-2 表示了块空间使用参数 PCTFREE 和 PCTUSED 的含义：

- (1) 新行可以被插入到块中，直到块中的可用空间百分比等于或小于 20%。换句话说，当行占用了 80%或更多块的可用空间时，该块就不能够再进行插入操作了。
- (2) 剩余的 20%可以用来预留作为导致行增长的更新操作。例如原来一个字段原来是 NULL，后来更新操作使其变为一个非空的值，因此占用空间增加。
- (3) 由于行缩减，导致块的可用空间可能会大于 20%(即已使用空间小于 80%)，但是块还不会被立即加入到自由列表中，直到块已使用空间低于 40%时，该块才会被加入到自由列表中。
- (5) 当块的已使用空间低于 40%时，该块可以被插入新行。随着块的使用空间增加，循环又开始了。

7.3 管理表存储区（数据段）

7.3.1 创建表

创建表的语法如下：

```
CREATE TABLE [schema.] table  
(column datatype [ , column datatype ] ...)  
[TABLESPACE tablespace ]  
[ PCTFREE integer ]  
[ PCTUSED integer ]  
[ INITRANS integer ]  
[ MAXTRANS integer ]  
[ STORAGE storage-clause ]  
[LOGGING | NOLOGGING]
```

其中，

schema 是表的所有者；table 是表名；column 是列名；data type 是列的类型；TABLESPACE 是表所在的表空间；PCTFREE 和 PCTUSED 是控制块使用的存储参数，具体含义参见 7.2-2 节；INITRANS 表示每个块所预留的允许并发事务的入

口数，默认值是 1；MAXTRANS 表示每个块所允许的最大并发事务的入口数；STORAGE 子句定义了如何给表分配区，具体含义参见下面关于创建表例子的说明；LOGGING 表示表的创建以及后续对该表的操作都记入重作日志；NOLOGGING 表示对表的创建以及某些数据装载不记入重作日志。

7.3.2 STORAGE 子句

下面 SQL 创建了一个表：

```
CREATE TABLE employee(  
  id NUMBER(7),  
  last_name VARCHAR2(25),  
  dept_id NUMBER(7))  
PCTFREE 20 PCTUSED 50  
STORAGE(INITIAL 200K NEXT 200K  
PCTINCREASE 0 MAXEXTENTS 50)  
TABLESPACE data;
```

其中关于 STORAGE 子句的含义如下：

INITIAL 表示给表分配的最初的区容量；

NEXT 表示如果最初的区已经完全被表行填满了，那么该参数就是给表分配的下一个区容量；

PCTINCREASE 表示超尺寸的表的第三个（如果 MINEXTENTS 被设置为 1）和随后的区容量在前一个区容量基础上增长的百分数；

MINEXTENTS 表示在建立表示分配的区个数，对表来说通常是 1；

MAXEXTENTS 表示表能分配到的最大的区个数；

7.4 管理表空间

7.4.1 创建表空间

创建表空间的语法如下：

```
CREATE TABLESPACE tablespace
```

```

DATAFILE datafile_clause]

[, datafile_clause]...

[MINIMUM EXTENT integer[K|M]]

[LOGGING|NOLOGGING]

[DEFAULT storage_clause ]

[ONLINE|OFFLINE]

[PERMANENT|TEMPORARY]

[extent_management_clause]

```

其中，

tablespace 是表空间名称；DATAFILE 表示组成表空间的数据文件位置、名称、大小、能否重用以及能否自动扩展，其语法如下：

```

datafile_clause ::= filename

{ SIZE integer[K|M] [REUSE] | REUSE }

[ autoextend_clause ]

```

其中的 autoextend_clause 子句语法如下：

```

autoextend_clause ::= [ AUTOEXTEND { OFF|ON[NEXT integer[K|M]]

[ MAXSIZE UNLIMITED | integer[K|M]] } ]

```

关于 DATAFILE 子句的含义以及例子参见下面 7.4-3 节。

MINIMUM EXTENT 表示表空间中区的最小大小，值为整数单位可以指定为 K 或 M 字节；LOGGING 表示表空间中的对象（表、索引等）的变化记入重作日志；DEFAULT 表示表空间中所有对象的默认存储参数；ONLINE 使表空间在线；OFFLINE 使表空间离线；PERMANENT 表示表空间是永久表空间，可以存储永久对象；TEMPORARY 表示临时表空间，只可以存储例如排序操作等产生的临时数据；extent_management_clause 表示对表空间中区的管理方式。

7.4.2 表空间的区管理方式

extent_management_clause 子句的语法如下：

```

extent_management_clause ::=

[ EXTENT MANAGEMENT

```



```
{ DICTIONARY | LOCAL
```

```
{ AUTOALLOCATE | UNIFORM [SIZE integer[K|M]] } } ]
```

其中，

DICTIONARY 表示采用数据字典管理方式管理表空间的区；**LOCAL** 表示采用位图管理方式管理表空间的区。对于前者，**ORACLE** 会自动更新数据字典表来反映表空间中区分配的变化。在 **ORACLE 8.0** 以及之前的版本，这种方式是唯一的表空间区管理方式。

对于位图管理方式，**ORACLE** 利用数据文件中的位图记录数据块的使用和空闲情况。

AUTOALLOCATE 表示由系统管理存储空间的分配；**UNIFORM** 表示表空间使用唯一的区大小，区大小由 **SIZE** 指定。

例如下面 SQL 语句建立了一个位图管理的表空间：

```
CREATE TABLESPACE user_data  
  
DATAFILE '/DISK2/user_data_01.dbf'  
  
SIZE 500M  
  
EXTENT MANAGEMENT LOCAL  
  
UNIFORM SIZE 10M;
```

位图管理方式相比数据字典管理方式有一些优点，即不需要频繁对数据字典表的操作导致回滚段管理和避免存储空间碎片等。

7.4.3 调整表空间的大小

调整表空间大小的方法有两种，即改变表空间现有数据文件的大小，或为表空间添加新的数据文件。改变表空间现有数据文件的大小可以是自动也可以是手工方式。

例如，下面 SQL 语句为表空间 **app_data** 新添加了一个 **200M** 的数据文件 **app_data_04.dbf**，并允许该数据文件自动扩展，扩展时以 **10M** 字节为单位增长（当数据文件需要分配新的区，但数据文件却没有足够空间时会导致数据文件自动扩展），扩展后数据文件大小不能超过 **500M**。

```
ALTER TABLESPACE app_data  
  
ADD DATAFILE '/DISK6/app_data_04.dbf'
```

```
SIZE 200M
```

```
AUTOEXTEND ON
```

```
NEXT 10M
```

```
MAXSIZE 500M;
```

也可以使用以下命令为已经添加到表空间的数据文件设置自动扩展属性：

```
ALTER DATABASE [database]
```

```
DATAFILE 'filename'[, 'filename']...autoextend_clause
```

7.5 获取存储结构的信息

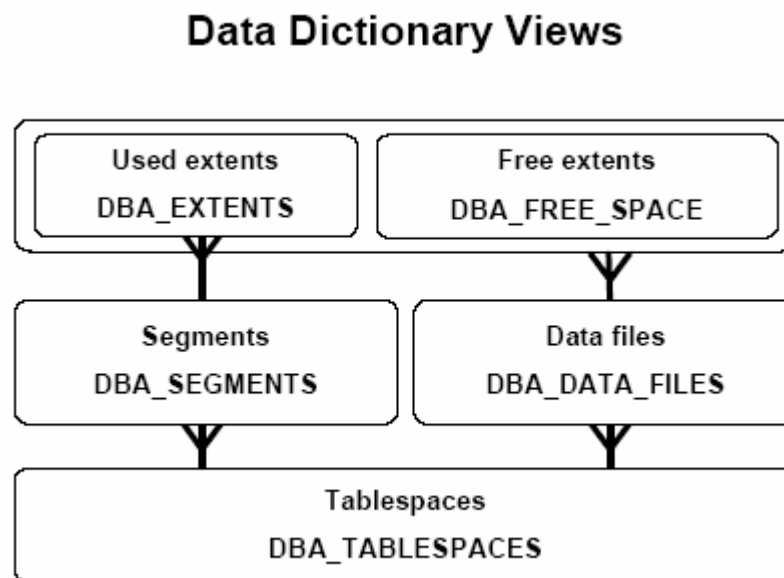


图 7.5-1 获取存储结构信息的数据字典

表空间、数据文件、段、空闲和已使用区之间的关系可以通过查询数据字典获得。这些数据字典及其关系如图 7.5-1 所示。

如果一个表空间被创建，则 DBA_TABLESPACES 会增加一条记录。

数据库中的每个数据文件在 DBA_DATA_FILES 中都有一条记录。

如果一个数据文件被增加到数据库，则在 DBA_FREE_SPACE 中对应有一条空闲区（除去文件头的剩余空间）记录。

如果一个段被创建，则在 DBA_SEGMENTS 会有一条相应的记录。

关于段的区空间分配情况可以从 DBA_SEGMENTS 查询获得，随着区的分配进行，DBA_FREE_SPACE 所显示的数据文件的可用空间也会动态变化。

7.5.1 查询 DBA_SEGMENTS

通过查询 DBA_SEGMENTS，可以获得如图 7.5-2 所示的一些有用信息。

Querying DBA_SEGMENTS

- | | |
|------------------------------|----------------------------|
| • General information | • Storage settings |
| – OWNER | – INITIAL_EXTENT |
| – SEGMENT_NAME | – NEXT_EXTENT |
| – SEGMENT_TYPE | – MIN_EXTENTS |
| – TABLESPACE_NAME | – MAX_EXTENTS |
| • Size | – PCT_INCREASE |
| – EXTENTS | • Other information |
| – BLOCKS | – Location |
| – BYTES | – Tuning |

图 7.5-2 查询 DBA_SEGMENTS

例如，

```
SQL> SELECT segment_name,tablespace_name,extents,blocks
      2 FROM dba_segments
      3 WHERE owner='SCOTT';
```

该查询可以获得用户 SCOTT 模式中的段的一些信息。

7.5.2 查询 DBA_EXTENTS

通过查询 DBA_EXTENTS，可以获得如图 7.5-3 所示的一些有用信息。

Querying DBA_EXTENTS

- | | |
|------------------|-------------------|
| • Identification | • Location |
| – OWNER | – TABLESPACE_NAME |
| – SEGMENT_NAME | – RELATIVE_FNO |
| – EXTENT_ID | – FILE_ID |
| • Size | – BLOCK_ID |
| – BLOCKS | |
| – BYTES | |

图 7.5-3 查询 DBA_SEGMENTS

利用 DBA_EXTENTS 查询可以获得关于一个段的所有区的情况。例如，

```
SQL> SELECT extent_id,file_id,block_id,blocks
2 FROM dba_extents
3 WHERE owner='SCOTT'
4 AND segment_name='EMP';
```

7.5.3 查询 DBA_FREE_SPACE

通过查询 DBA_FREE_SPACE，可以获得如图 7.5-4 所示的一些有用信息。

Querying DBA_FREE_SPACE

- **Location**
 - TABLESPACE_NAME
 - RELATIVE_FNO
 - FILE_ID
 - BLOCK_ID
- **Size**
 - BYTES
 - BLOCKS

图 7.5-4 查询 DBA_FREE_SPACE

例如，以表空间为分组条件查询表空间的空闲空间情况：

```
SQL> SELECT tablespace_name, count(*),  
2 max(blocks), sum(blocks)  
3 FROM dba_free_space  
4 GROUP BY tablespace_name;
```

第8章 备份和恢复

备份一个 ORACLE 数据库有三种标准方式：EXPORT(导出)、脱机备份和联机备份。导出方式是数据库的逻辑备份，其他两种备份方式都是物理文件备份。

8.1 逻辑备份

ORACLE 的 EXPORT 实用程序 exp 用来读取数据库（其中包括数据字典）和把输出写入一个叫作导出转储文件(export dump file)的二进制文件中。可以导出整个数据库、指定用户或指定表。在导出期间，可以选择是否导出与表相关的数据字典信息，如权限、索引和与其相关的约束条件。exp 所写的文件包括完全重建全部被选对象所需的命令。

利用 exp 进行导出备份具有三种方式： Full 方式、User 方式和 Table 方式。

8.1.1 Export 实用程序

exp 的选项及含义如下：

(1) Userid

执行导出的帐户的用户名/口令，如果这是 exp 命令后的第一个参数，则关键字 userid 就不必指定。

(2) Buffer

用于获取数据行的缓冲区尺寸，缺省值随系统而定，通常设为一个高值（大于 64000）。

(3) File

导出转储文件的名称。

(4) Filesize

一个导出转储文件的最大尺寸。如果 file 条目中列出了多个文件，将根据 filesize 设置值导出这些文件。

(5) Compress

一个 Y/N 标志，用于指定导出是否应把碎片段压缩成单个区。这个标志影响将存储到导出文件中的 storage 子句。

(6) Grants

一个 Y/N 标志，用于指定数据库对象的权限是否导出。

(7) Indexes

一个 Y/N 标志，用于指示表上的索引是否导出。

(8) Rows

一个 Y/N 标志，用于指示行是否导出。如果设置为 N，在导出文件中将只创建数据库对象的 DDL。

(9) Constraints

一个 Y/N 标志，用于指示表上的约束条件是否导出。

(10) Full

若设为 Y，执行 FULL 数据库导出。

(11) owner

导出数据库帐户的清单，可以执行这些帐户的 USER 导出。

(12) Tables

导出表的清单，可以执行这些表的 TABLE 导出。

(13) Recordlength

导出转储文件记录的长度，以字节为单位。除非是在不同的操作系统间转换导出文件，否则就使用缺省值。

(14) Inctype

要执行的导出类型(允许值为 complete(缺省)、cumulative 和 incremental)。

(15) Direct

一个 Y/N 标志，用于指示是否执行 DIRECT 导出。DIRECT 导出在导出期间绕过缓冲区，从而大大提高导出处理的效率。

(16) Record

用于 INCREMENTAL 导出，这个 Y/N 标志指示一个记录是否存储在记录导出的数据字典表中。

(17) Parfile

传递给 EXPORT 的一个参数文件名，这个文件可以包含 exp 所需的全部参数条目。

(18) Statistics

这个参数指示导出对象的 ANALYZE 命令是否应写到导出转储文件上。其有效值是 COMPUTE、ESTIMATE (缺省) 和 N。在较早的 ORACLE 版本中，这个参数叫作 ANALYZE。

(19) consistent

一个 Y/N 标志，用于指示是否应保留全部导出对象的读一致版本。在 EXPORT 处理期间，当相关的表被用户修改时需要这个标志。

(20) Log

一个要写导出日志的文件名。

(21) Feedback

表导出时显示进度的行数。缺省值是 0，所以在一个表全部导出前没有反馈显示。

(22) point_in_time_recover

一个 Y/N 标志，用于向 ORACLE 指示，是否正在导出用于表空间时间点恢复的元数据。

(23) recover_tablespaces

在表空间时间点恢复期间，其元数据应被导出的表空间。

(24) Query

导出时用于每个表的 where 子句。

(25) tablespaces

移动一个表空间时应导出其元数据的表空间。

ORACLE8i 中 exp 参数的缺省值如表 8.1-1 所示。

表 8.1-1 exp 的参数缺省值（ORACLE 8i）

关 键 字	Oracle8缺省值	关 键 字	Oracle8缺省值
userid	未定义	record	Y
buffer	由系统决定	parfile	未定义
file	EXPDAT.DMP	statistics	ESTIMATE
compress	Y	Triggers	Y
grants	Y	consistent	Y
indexes	Y	log	未定义
rows	Y	direct	N
constraints	Y	feedback	0
full	N	filesize	未定义
owner	当前用户	query	未定义
tables	未定义	transport_tablespace	未定义
recordlength	由系统决定	tablespaces	未定义
inctype	COMPLETE		

使用 exp help=Y 可以显示 exp 命令的参数。

1. Compress 参数

对于包含多个区的数据段，COMPRESS = Y 选项可修改 storage 子句的 initial 参数。因此，该段的总分配空间应压缩到一个区。使用该参数要注意两点：

- 压缩的空间是分配的空间而不是使用的空间。例如，一个分配有 3 个 100 M B 区的 300M B 空表，将压缩成一个 300M B 的空区。
- 若表空间具有多个数据文件，一个数据段可以分配的空间可能会大于最大的那个数据文件。在这种情况下，使用 COMPRESS 将改变 storage 子句使 initial 区大于任何一个数据文件尺寸。由于一个区不能跨越多个数据文件，所以在导入时创建对象将失败。

例如，下面的命令导出 myaccount1 和 myaccount2 模式。

```
exp system/manager file=mydata.dmp compress=y owner=(myaccount1,
myaccount2)
```

2. consistent 参数

在向导出转储文件写数据库的数据时，exp 一次读一个表。因此，尽管 exp 开始于一个指定的时间点，而各个表则读于不同的时间。在 exp 开始读时

存储在那个表中的数据就是要导出的数据。由于大多数表与其他表相关，所以如果用户在导出期间修改数据，就会引起导出的数据不一致。

假设一个事务修改表 A 和表 B，但在表 A 导出前未提交：

- (1) 导出开始
- (2) 对表 A 和表 B 开始事务处理
- (3) 表 A 导出
- (4) 提交事务
- (5) 表 B 导出

这样会导致导出转储文件包含了不一致的数据。

要避免这个问题可以有两个选择，一是应在无人修改表时安排导出；二是可以使用 `consistent` 参数。当 `consistent=y` 时，数据库保留一个回滚段保证导出一致性，但是这样也增加了回滚段的开销并且降低了导出性能。

8.1.2 Import 实用程序

- (1) Userid

需执行导入操作的帐户的用户名/口令。如果这是 `imp` 命令后的第一个参数，就不必指定 `userid` 关键字。

- (2) Buffer

取数据行用的缓冲区尺寸。缺省值随系统而定，该值通常设为一个高值（大于 100000 ）。

- (3) File

要导入的导出转储文件名。

- (4) Show

一个 Y/N 标志，指定文件内容显示而不是执行。

- (5) Ignore

一个 Y/N 标志，指定在发出 `create` 命令时遇到的错误是否忽略。若要导入的对象已存在，就使用这个标志。

- (6) Grants

一个 Y/N 标志，指定数据库对象上的权限是否导入。

(7) Indexes

一个 Y/N 标志，指定表上的索引是否导入。

(8) Constraints

一个 Y/N 标志，指定表上的约束条件是否导入。

(9) Rows

一个 Y/N 标志，确定行是否导入。若将其设为 N，就只对数据库对象执行 DDL。

(10) Full

一个 Y/N 标志，如果设置为 Y，就导入 Full 导出转储文件。

(11) Fromuser

应从导出转储文件中读取其对象的数据库帐户的列表（当 full=n 时）。

(12) Touser

导出转储文件中的对象将被导入到的数据库帐户的列表。ffromuser 和 touser 不必设置成相同的值。

(13) Tables

要导入的表的列表。

(14) Recordlength

导出转储文件记录的长度，以字节为单位。除非要在不同的操作系统间转换，否则都用缺省值。

(15) inctype

要被执行导入的类型（有效值是 complete [缺省]、cumulative 和 incremental）。

(16) Commit

一个 Y/N 标志，确定每个导入后 import 是否提交（其大小由 buffer 设置），如果设置为 N，在每个表导入后都要提交 import。对于大型表，commit=N 需要同样大的回滚段。

(17) Parfile

传递给 `import` 的一个参数文件名，这个文件可以包含这里所列出的全部参数的条目。

(18) `Indexfile`

这是个非常有效的选项，可以把所有的 `create table`、`create cluster` 和 `create index` 命令写到一个文件中，而不是运行它们。几乎所有的 `create index` 命令都要改为注释。这个文件在以 `index=N` 导入后就可以运行（进行少量修改）。这对把表和索引分别放在不同的表空间中非常有用。

(19) `Point_in_time_recover`

一个 Y/N 标志，确定导入是否是表空间时间点恢复的一部分。

(20) `Destroy`

一个 Y/N 标志，指示是否执行在 `Full` 导出转储文件中找到的 `create tablespace` 命令（从而破坏正在导入的数据库数据文件）。

(21) `Log`

`import` 日志将要写入的文件名。

(22) `Analyze`

一个 Y/N 标志，指示 `import` 是否应执行在导出转储文件中找到的 `analyze` 命令。

(23) `Feedback`

表导入时显示进展的行数。缺省值为 0，所以在没有完全导入一个表前不显示反馈。

(24) `Filesize`

如果参数 `filesize` 用在 `export` 上，这个标志就是对 `export` 指定的最大转储尺寸。

(25) `recalculate_statistics`

一个 Y/N 标志，确定是否应生成优化程序统计。

(26) `transport_tablespaces`

一个 Y/N 标志，指示可移植的表空间元数据被导入到数据库中。

(27) `Tablespaces`

要传送到数据库中的表空间名字或名字清单。

(28) Datafiles

要传送到数据库的数据文件清单。

(29) tts_owner

可移植表空间中数据拥有者的名字或名字清单。

ORACLE8i 中 imp 参数的缺省值如表 8.1-2 所示。

表 8.1-2 imp 的参数缺省值 (ORACLE 8i)

关键字	Oracle8缺省值	关键字	Oracle8缺省值
userid	未定义	parfile	未定义
buffer	随系统而定	indexfile	未定义
file	EXPDAT.DMP	destroy	N
show	N	log	未定义
ignore	N	charset	实例的NLS_LANG(过时)
grants	Y	point_in_time_recover	N(由Transport_Tablespaces代替)
indexes	Y	skip_unusable_indexes	N
constraints	Y	analyze	Y
rows	Y	feedback	0
full	N	tiod_novvalidate	无
fromuser	未定义	filesize	随系统而定，必须与Export值一致
touser	未定义	recalculate_statistics	N
tables	未定义	transport_tablespace	N
recordlength	随系统而定	tablespaces	无
inctype	未定义	datafiles	无
commit	N	tts_owners	无

1. 回滚段需求

在缺省情况下，数据库在每个表完成导入后就执行一次提交。若有一个 300 MB 数据的表，那么就需要至少这么容量的回滚段来容纳。对回滚段来说，这是个不必要的负担。若要减少回滚段尺寸，在设定 commit=y 的同时，为 buffer 设一个值。如下例所示，在每次达到 buffer 相应的数据后执行一次提交。在所示的第一个导入命令中，每装入一个表就执行一次提交。第二个命令是每当插入 64000 字节数据后就执行一次提交。

Imp system/manager file=expdat.dmp

Imp system/manager file=expdat.dmp buffer=64000 commit=y

为 buffer 设置大小的原则是使 buffer 要能处理导入的最大单行。在带有 LONG 和 LOB 数据类型的表中，应大于 64KB。如果不知道导出的最大行

的长度，可先设一个适当值（如 50000），并运行 `import`。若返回一个 `IMP-00020` 错误，就说明 `buffer` 容量不够大，需要加大后再进一步尝试 `import` 操作。

当使用 `commit=y` 时，注意为每个 `buffer` 数组执行 `commit` 操作。因此如果一个表的导入操作失败，这个表的一些行就可能已经导入和提交。那么就可能再次运行 `import` 操作前使用或删除部分装入。

2. 导入到不同帐户

若要通过 `export/import` 把对象从一个用户转换到另一个用户，可对对象拥有者执行一个 `user` 导出操作。在导入操作期间，将拥有者指定为 `fromuser`，将拥有对象的帐户指定为 `touser`。例如：

```
exp system/manager file=user1.dmp owner=user1 grants=N indexes=Y
compress=Y rows=Y
```

```
imp system/manager file=user1.dmp fromuser=user1 touser=user2 rows=Y
indexes=Y
```

3. 导入上次导入失败的结构

`ROWS` 参数在两种情况下非常有用：

- (1) 可用于只重建数据库结构而不重建表的数据，即使这些数据已被导出也是如此。
- (2) 在第一次导入操作期间未能成功创建对象时，这种情况下，通常需要使用 `ROWS` 参数进行多个导入操作。

`Import` 进行导入操作时，有时会出问题，主要是因为数据库对象之间可能存在的相关性引起的。例如，如果 `import` 要在其所依赖的对象未创建以前试图创建该对象（如视图），就会引起错误。在这种情况下，`import` 可以改为 `rows=N` 和 `ignore=N` 重新运行，这样就仅导入第一次导入操作期间未能导入的结构。

下面例子显示这种用法。第一次导入操作试图导入整个导出转储文件。若这次 `import` 操作失败，第二次就试图导入第一次失败的那些结构。

```
Imp system/manager file=user1.dmp full=Y commit=Y buffer=64000
```

导入时，某些视图会因出现 `ORA-00942` 错误（表或视图不存在）而失败。现在采用

`IGNORE=N` 和 `ROWS=N` 第二次运行 `import`：

`Imp system/manager file=user1.dmp ignore=N rows=N commit=Y buffer=64000`

第二个命令中的 `IGNORE=N` 参数通知 `import` 忽略第一次导入时已创建的那些对象。它仅导入那些失败的对象。

4. 利用导入分隔表和索引

可以用两个 `import` 选项 `indexes` 和 `indexfile` 来重新组织表和索引的表空间。

在进行导入操作时使用 `indexfile` 选项,可以读取而不是导入导出转储文件,其建表和索引脚本文件会被写入一个输出文件中。可以编辑这个文件,以修改这里列出的表和索引的 `tablespace` 和 `storage` 参数。然后可以通过 `SQL*PLUS` 运行修改后的文件,以便在导入数据前预建所有对象或只创建指定的对象(如索引)。

创建索引文件时, `create index` 脚本文件只是文件中没有通过 `REM` 命令注释的那些命令。这个缺省功能允许数据库管理人员在进行导入操作时将用户的表和索引分送到分离的表空间中。具体过程是:首先创建索引文件并修改索引的 `tablespace` 子句,接着以 `indexes=N` 导入用户,这样用户索引就不会被导入。然后运行修改的索引文件以便在新的表空间中创建索引。

需要注意的是,索引文件可以包含多个用户的条目(如果导出了多个用户)。实际上,有用的是将索引文件分成多个文件,每个用户对应一个。这样就容易保持表空间分布的一致性。

8.2 脱机备份

冷备份是数据库文件的物理备份,通常在数据库通过一个 `shutdown normal` 或 `shutdown immediate` 命令正常关闭后进行。当数据库关闭时,其使用的各个文件都可以进行备份。这些文件构成一个数据库关闭时的一个完整映像。

冷备份通常要备份以下文件:

所有数据文件

所有控制文件

所有联机重做日志

初始化参数文件 `init.ora` (可选)

先执行以下 `SQL` 语句查看所有需要备份的文件:

```
SVRMGR> select * from v$datafile;
```

```
SVRMGR> select * from v$controlfile;
```

```
SVRMGR> select * from v$logfile;
```

记录下所有的这些文件的路径和文件名，同时连同初始化参数文件一起备份到 disk 或 tape。

8.3 联机备份

脱机备份只能在数据库关闭时使用。不过，当数据库打开时也能对数据库进行物理文件备份并且保证备份正确，但要求数据库运行在 ARCHIVELOG 方式下。这种备份叫作热备份、联机备份或 ARCHIVELOG 备份。

ORACLE 以循环方式写联机重做日志文件，写满第一个日志文件后，开始写第二个，然后第三个。当最后一个联机重做日志文件写满后，LGWR 后台进程开始重写第一个文件的内容。

当 ORACLE 运行在 ARCHIVELOG 方式时，ARCH 后台进程重写重做日志文件前将每个重做日志文件做一份拷贝。这些归档的重做日志文件通常写到一个磁盘设备中，也可以直接写入磁带设备。

8.3.1 归档模式和非归档模式

数据库分为归档日志执行模式和非归档日志执行模式。

在归档日志执行模式下，数据库后台进程 ARCH 在每次日志切换事件发生时，将刚填满的联机重做日志文件备份到 disk 或 tape 中成为脱机交易记录文件，当需要恢复数据时，只需要将最近一次的数据库文件的全备份再加上这些脱机重做日志文件即可恢复到问题发生的时间点。

在非归档日志执行模式下，数据库系统循环使用一组联机重做日志文件，所以只能保存近期的交易记录，这样要进行数据恢复时，只能恢复近期所做的交易。

1. 查看数据库当前模式

通过执行 SVRMGR>archive log list 命令查看数据库当前模式。

2. 从非归档日志模式切换到归档日志模式

修改启动参数文件，设置如下项：

```
log_archive_start=true
```

该参数为 true 则实例启动时会启动 ARCH 后台进程。


```
log_archive_dest=/vol4/arch/arch
```

该参数设置存放脱机交易记录文件的位置及文件名开头。

```
log_archive_format=_%s.log
```

该参数设置脱机交易记录文件的后缀名，%s 代表记录文件的 log sequence number。

然后执行以下命令：

```
SVRMGR>connect internal
```

```
SVRMGR>shutdown
```

```
SVRMGR>startup mount
```

```
SVRMGR>alter database archivelog
```

```
SVRMGR>alter database open
```

这样，数据库就运行在归档模式了。

命令 `alter database noarchivelog` 可以将数据库运行模式更改为非归档模式。

8.3.2 联机数据库备份

一旦数据库运行在 `archivelog` 方式，在打开并对用户可用时就可以进行备份。这一特性允许连续运转的数据库可以归档并能保证其恢复性。联机热备份应该安排在用户活动最少的时间段进行。

联机热备份包括三个过程，即逐个表空间地备份数据文件、备份归档重做日志文件和备份控制文件。

1. 逐个表空间地备份数据文件

该过程又可以分为四个步骤，即查询表空间包括哪些数据文件、设置表空间为备份状态、备份表空间的数据文件、将表空间恢复到正常状态。

(1) 查询表空间包括哪些数据文件

执行命令 `select tablespace_name,file_name from dba_data_files` 可以获得系统中所有数据文件以及其归属的表空间。

(2) 设置表空间为备份状态

执行命令 `alter tablespace tablespace_name begin backup` 将表空间 `tablespace_name` 设置为备份状态。

(3) 备份表空间的数据文件

执行操作系统的文件备份命令将表空间的数据文件备份。

(4) 将表空间恢复到正常状态

执行命令 `alter tablespace tablespace_name end backup` 将表空间 `tablespace_name` 恢复到正常状态。

2. 备份归档重做日志文件

该过程又可以分为三个步骤，即暂停归档进程 ARCH、备份归档重做日志文件。

(1) 暂停归档进程 ARCH

执行命令 `archive log stop` 暂停归档进程 ARCH。

(2) 记录归档目标目录中已归档日志文件的列表

可以查询 V\$LOG 动态字典视图。如果日志已完全归档，V\$LOG 的 archived 列将含有 YES 值。可以从 V\$LOG 选择最高的归档日志（使用 sequence# 列）并将其用作备份文件清单的基础。例如，如果 V\$LOG 表明 sequence#2334 是最后一个被归档的日志文件，就可以成功地备份归档重做日志目标目录中所有序号在 2334 以下的文件。如果试图备份 2335，可以在操作系统级成功备份，但由于这个文件还没有完全归档，这个备份可能只写入一半，因而在恢复操作期间不可能有用。

(3) 重新启动归档进程 ARCH

执行命令 `archive log start` 启动归档进程 ARCH。

(4) 备份归档重做日志文件

执行操作系统的文件备份命令备份归档重作日志文件。

(5) 从归档目标目录中删除已经备份的归档日志文件

3. 备份控制文件

执行命令 `alter database backup controlfile to destination/control.bak` 进行在线备份控制文件。也可以执行 `alter database backup controlfile to trace` 将 `create controlfile` 的命令写入到数据库的跟踪文件中。

8.4 数据库恢复

8.4.1 实例恢复

ORACLE 实例失败的情形包括数据库服务器异常宕机、后台进程异常关闭等。ORACLE 实例失败后，下次重起时恢复能通过 SMON 进程自动进行。

当一个实例失败后数据库启动时，ORACLE 会根据当前联机重作日志文件的检查点标志，同步所有的数据文件，即将日志中记录的自上次执行检查点以来的已经提交的事务前滚(roll forward 或 cache recovery)，而把未提交的事务回滚(roll back 或 transaction recovery)。

ORACLE 实例失败后，下次启动后通常检查数据库报警日志中的错误信息以查明原因。

8.4.2 介质恢复概述

介质失败也叫磁盘失败，常发生于一个磁盘上驻留的当前数据库文件变得无法被数据库读出时。介质失败通常由磁盘损坏或磁盘上读错误引起。

为了避免介质失败导致数据库故障，需要做好四种数据库文件的备份，即控制文件、联机重作日志文件、归档重作日志文件和数据文件。

每个数据库都应有其控制文件的多个拷贝，所有这些文件应存储在不同的设备上。要恢复一个丢失的控制文件，只要关闭数据库并从保留有控制文件的地方拷贝一份到正确的位置即可。

如果所有控制文件都丢失了，可以使用 `create controlfile` 命令。该命令允许为数据库创建一个新的控制文件，并指定数据库中的所有数据文件、联机重做日志文件和数据库参数。如果对使用的参数有疑问并且数据库运行在 ARCHIVELOG 模式，则可用 `alter database backup controlfile to trace` 将一个合适的 `create controlfile` 命令写入到跟踪文件。这时可根据需要编辑这个跟踪文件。除非所有控制文件都丢失了，否则不要使用 `create controlfile` 命令。

使用多路联机重作日志文件来避免联机重作日志文件的失败，具体方案可以采取通过使用联机重做日志组或在操作系统级镜像文件。由于联机重作日志文件被镜像，所以介质失败时它们不会丢失。

如果丢失的是归档重做日志文件，就无法恢复。为此，最重要的是使归档重做日志文件目标设备也保持镜像。

在 ORACLE 8i 中, 可以使用 LOG_ARCHIVE_DEST_ *n* 参数为归档重做日志文件指定最多 5 个位置 (LOG_ARCHIVE_DEST_1 、 LOG_ARCHIVE_DEST_2 等)。ARCH 进程将同时把文件写入全部指定的位置。每个归档目标都有一个通过 LOG_ARCHIVE_DEST_STATE_ *n* 参数设置的相应状态。一个归档目标的状态可以是 ENABLED (这种情况下可以在这里写文件) 或 DEFER (这种情况下目标区当前无效)。应为初始化参数文件中的每一个 LOG_ARCHIVE_DEST_ *n* 条目创建一个 LOG_ARCHIVE_DEST_STATE_ *n* 条目。

介质恢复分为无归档日志的恢复和有归档日志的恢复两种, 下面分别进行介绍。

8.4.3 无归档日志的数据库恢复

对于无归档日志的数据库恢复, 所有的控制文件和数据文件必须能够被恢复。重作日志文件和初始化参数文件根据具体情况可以进行恢复或者不恢复。这种恢复的好处是容易操作, 缺点是只能将数据库恢复到上次进行数据库完全备份的时刻, 因此有丢失数据的危险。

无归档日志的数据库恢复的过程比较简单, 如下所述:

```
SVRMGR>shutdown immediate;
```

将上一次全备份的所有数据库文件, 包括控制文件和全部数据文件恢复到原有目录。然后依次执行以下命令:

```
SVRMGR>connect internal;
```

```
SVRMGR>startup mount;
```

```
SVRMGR>recover database until cancel using backup controlfile;
```

```
SVRMGR>cancel;
```

```
SVRMGR>alter database open resetlogs;
```

如果无法将数据文件恢复到原来目录位置, 可将其放置于其他目录下, 只要在打开数据库前, 通过以下方式修改控制文件中的设置即可:

```
SVRMGR>startup mount;
```

```
SVGMGR>alter database rename file
```

```
2>' /vol2/ncm.dat' to ' /vol4/ncm.dat';
```

```
SVRMGR>alter database open;
```

数据库重起以后即恢复正常使用。

8.4.4 有归档日志的数据库完全恢复

有归档日志的数据库完全恢复的好处是：可以只恢复损坏或者丢失的数据文件，提交的事务能够得到完整的恢复，恢复能够在数据库打开状态下进行。缺点是：由于 ORACLE 要通过归档日志重作记录恢复数据文件，因此自从上次数据库全备份以来的所有的重作日志必须都被归档保存，否则只能恢复到某个时间点。

有归档日志地数据库完全恢复包括下面几个步骤：

1. 确定哪些数据文件需要恢复

```
SQL> SELECT * FROM v$recover_file;
```

FILE#	ONLINE	ONLINE_STATUS	ERROR	CHANGE#	TIME
-------	--------	---------------	-------	---------	------

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

2	OFFLINE	OFFLINE		288772	22-JUL-02
---	---------	---------	--	--------	-----------

ERROR 字段描述了数据文件需要恢复的原因，通常有两个值：NULL 和 OFFLINE NORMAL。NULL 表示未知原因，OFFLINE NORMAL 表示不需要恢复，是正常脱机。CHANGE#代表系统修改号 SCN（system change number），表示恢复必须从哪里开始。

2. 确定恢复过程需要哪些归档重作日志

```
SQL> SELECT * FROM v$recovery_log;
```

THREAD#	SEQUENCE#	TIME	ARCHIVE_NAME
---------	-----------	------	--------------

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

1	34	02-MAR-01	.../ORADATA/ARCHIVE1/arch_34.arc
---	----	-----------	----------------------------------

1	43	04-MAR-01	.../ORADATA/ARCHIVE1/arch_43.arc
---	----	-----------	----------------------------------

1	44	04-MAR-01	.../ORADATA/ARCHIVE1/arch_44.arc
---	----	-----------	----------------------------------

从该查询可以知道，为了恢复 datafile 2，从日志序号（SEQUENCE#）34 开始的归档日志被恢复过程所需要。

也可以查询 V\$ARCHIVED_LOG 得到所有归档的日志文件。3. 使用 recover 命令进行恢复

- (1) 在数据库 mount 状态下可以执行的 recover 命令

```
SQL>recover database;
```

```
SQL>recover datafile 'datafile_path/datafile_name';
```

- (2) 在数据库 open 状态下可以执行的 recover 命令

```
SQL>recover tablespace tablespace_name;
```

```
SQL>recover datafile 'datafile_path/datafile_name';
```

通常，执行 recover database 是优先考虑使用的恢复方式，但是这种方式需要先 shutdown 数据库，然后以 mount 方式启动才可以运行。在不能先 shutdown 数据库的情况下，可以执行 recover tablespace tablespace_name 命令以表空间为单位恢复。执行命令 recover datafile 'datafile_path/datafile_name' 以数据文件为单位恢复可以在数据库 open 或者 mount 方式下进行。

在介质恢复的过程中，ORACLE 可以自动方式或与用户交互的方式决定使用哪些归档重作日志文件和联机重作文件来重建同步的数据文件。如果重作日志文件的位置不在 LOG_ARCHIVE_DEST_n 指定的目录下，则需要通过以下方式通知恢复进程：

- 在恢复的提示符下，直接敲回车键或者输入以下命令之一：

```
{<RET>=suggested | filename | AUTO | CANCEL}
```

使用 ALTER SYSTEM ARCHIVE 命令：

```
SQL> ALTER SYSTEM ARCHIVE LOG START TO <new location>;
```

使用 RECOVER FROM <LOCATION> 命令：

```
SQL> RECOVER FROM '<new location>' DATABASE;
```

例如，下面的恢复过程，通过在执行恢复命令之前执行 set autorecovery on，从而按照 ORACLE 默认选项进行恢复。

```
SVRMGR>shutdown;
```

```
SVRMGR>connect internal;
```

```
SVRMGR>startup mount;
```

```
SVRMGR>set autorecovery on;
```

```
SVRMGR>recover database;
```

```
SVRMGR>alter database open;
```

下面的恢复过程，通过在恢复提示符下键入 AUTO 让 ORACLE 按照默认选项进行恢复。

```
SVRMGR> RECOVER datafile 4
```

```
ORA-00279: change 308810...07/22/02 17:00:14 needed for thread 1
```

```
ORA-00289: suggestion : /ORADATA/ARCHIVE1/arch_35.arc
```

```
ORA-00280: change 308810 for thread 1 is in sequence #35
```

```
Specify log: {<RET>=suggested | filename | AUTO | CANCEL}
```

```
AUTO
```

```
Log applied.
```

```
Media recovery complete.
```

以表空间和数据文件为单位进行介质恢复需要注意的是，必须保证表空间和数据文件都是脱机 OFFLINE 状态，才能执行 recover 命令。将表空间脱机的命令是：

```
SVRMGR>alter tablespace tablespace_name offline immediate;
```

恢复完成后要将表空间联机，命令是：

```
SVRMGR>alter tablespace tablespace_name online;
```

8.4.5 有归档日志的数据库不完全恢复

不完全恢复是指将数据库恢复到某个时间点或者某个日志序号。

执行不完全恢复需要注意以下几个问题：

- (1) 由于不完全恢复操作过程较为复杂，为保险起见，数据库管理员应先在关闭数据库的状态下做一次完全脱机备份后再开始。
- (2) 如果恢复到的时间点时刻的数据库结构和当前的不一样，则需要使用备份的控制文件（与恢复到的时间点时刻数据库结构一致的当时的控制文件）来做恢复。

- (3) 执行完不完全恢复以后，为使现有的控制文件中的日志序号与所有的数据文件一致，必须执行 `alter database open resetlogs` 命令来打开数据库，ORACLE 会将起始日志序号设为 1。起始日志序号重设为 1 后，以前所有的归档重作日志都将失效，所以应立即 `shutdown` 整个数据库再做一次完全数据库脱机备份。

```
SVRMGR>connect internal;
```

```
SVRMGR>startup mount;
```

```
SVRMGR>recover database until cacle;
```

或者可以指定一个备份控制文件进行恢复：

```
SVRMGR>recover database until cancel
```

```
2>using backup controlfile '/vol1/control01';
```

在接下去的交互式问答过程中，依次执行重做日志文件的交易，直到用户键入 CANCEL 时结束不完全恢复。

恢复完以后用以下命令同步控制文件中日志序号：

```
SVRMGR>alter database open resetlogs;
```