第 1 章 ORACLE 数据库系统概述

数据库技术产生于 20 世纪 60 年代中期,已有三十余年的历史,而关系数据库从 1970 年 E.F.Codd 发表《大型共享数据库数据的关系模型》至今已有 28 年了，从这样两个时间概念可以看出关系数据库在数据库领域中所起到的历史作用。70 年代是关系数据库理论研究和系统开发的时代，80 年代推出了成功的关系数据库产品，目前，关系数据库已成为主导的数据库产品，ORACLE 数据库系统在其中发挥了重要的作用。另外，DB2、Informix、SQL Server、Sybase、Ingres 系统等也各有千秋。展望关系数据库技术的前景，形势更加喜人，在继承和发展关系数据库技术基础上出现的对象关系数据库技术也日渐成熟，ORACLE 推出的 V8.0 版在此方向上迈出了坚实的一步，可以这样说，关系数据库技术有过辉煌的昨天和今天，将会有更为美好的明天。

1.1 背景及基本概念

1.1.1 关系数据模型

数据库管理系统可以看成是对大量、持久的数据资源实现可靠管理和共享使用的一个软件系统，该软件的结构和功能与数据模型紧密相关。

数据模型是一组描述现实数据管理问题的概念以及概念间的相互联系，它可以与程序设计语言作一类比，语言中最核心的是语言中所提供的基本数据类型及由基本类型构造复杂数据结构的手段，其二是语言的控制结构。在数据库技术发展的历史上，出现过层次的数据模型、网状的数据模型、关系数据模型、面向对象数据模型、逻辑数据模型、对象关系数据模型，等等。以层次或网状模型建立的数据库系统，能摆脱程序员对物理实现的许多依赖，而仅面对逻辑的数据记录和逻辑的存取路径。这种暴露存取路径的导航是那一代系统的主要特征之一，1973 年 C.W.Bachman 在图灵奖的领奖台上所做的演讲题目就是《作为导航员的程序员》。能否隐蔽存取路径，不需要用户导航，使用户与系统在更高的抽象层次上进行对话，以克服底层的变化对上层逻辑结构的影响？为此，E.F.Codd 于 1970 年发表了《大型共享数据库数据的关系模型》一文，作者在该文中陈列了当时三种主要的依赖现象，即次序依赖(Ordering Dependent)、索引依赖(Indexing Dependent)、存取路径依赖(Access Path Dependent)，提出了关系概念，并据此提出了对关系进行操作的关系代数。由于关系的抽象性和关系代数的描述

需要全本请访问 www.ertongbook.com

性，使得这样的系统可以实现高度的数据独立性，同时使程序员摆脱对数据的底层管理，把存储、检索、修改等的实现方法和效率问题交给了系统，而用户只需关注于他的应用逻辑，可大大提高软件开发能力和生产率。E.F.Codd 在 1981 年获得图灵奖，他作了题为《提高生产率的现实基础》的演讲。

关系数据库管理系统是根据关系模型而构造的系统，它允许终端用户和应用程序员对数据库进行存取时仅以关系的方式进行，即不需了解底层的物理结构，仅在关系和关系操作的层面上来感知数据库。关系模型是为数据及数据间的相互关系进行建模的一种方法，包括如何表示数据以及怎样操纵这种表示，进一步讲，关系模型由模型描述、数据操纵、模型控制三部分构成，相应的语言为 DDL、DML 和 DCL。

关系模型是由 Codd 最先形成的，他以很慎重的态度使用一些术语，如关系术语本身，在那时的数据处理范围内是不太熟悉的，问题还在于，当时许多熟知的术语很混乱，它们不够准确，难以满足 Codd 所提出的那种形式理论的要求。例如，考虑术语记录(Record) 在不同的时期，记录可以表示一个记录实例或一个记录型，或像 COBOL 风格的记录(允许重复组出现),或是一个平面记录(没有重复组),一个逻辑记录或一个物理记录，一个存贮记录或一个虚记录，等等。这样，抽象形式的关系模型就没有采用术语记录，相反，它使用术语元组(Tuple，是 N-Tuple 的缩写),在 Codd 提出这一概念时就给予了精确定义。

1.关系数据结构

关系模型中最小的数据单元是单个数据值，这些值也称为原子值，就该模型而言，它是不可再分的。域是这种同一类型值的一个集合，如: Supplier numbers 的域是所有正确的提供者编号(number)的集合。如果两属性的值取自于同一域，则它们间的比较、连接、并等才有定义，否则，将可能无意义。例如:

```
select s.*, sp.*
    from s, sp
        where s.s#=sp.s# ;    /*可能有意义*/
```

而

```
select s.*, sp.*
    from s,  sp
        where s.status=sp.qty /* 则可能变得无意义 */
```

域的本质是概念上的，数据库中可以或不一定保存真实值集，在数据库定义时加以说明，每个属性的定义应包含它所引用的值域。

现在来定义关系。域 D1, D2, …, Dn 上的关系由一个头和体构成，头由固定的属性集 A1, A2, …, An 构成，并使属性 Ai 和域 Di 一一对应，体是可随时间而变化的元组集，每个元组又由属性值对集构成，(Ai, Vi)(i=1, …n)、每个(Ai, Vi)对对应于头中的 Ai, Vi 是从与属性 Ai 相关的 Di 中赋予的值。

根据对关系的定义，我们在纸上把关系表示成一个表只是为了便利，一个表与一个关系事实上是不一样的,例如,表中的行有一个次序(从上到下),而元组则没有,表的列同样有一个序，而属性则没有。一些术语的对比如表 1.1。

表 1.1 术语对比

Formal relational term	Informal equivalents
relation	Table
Tuple	Record, row
Attribute	Field, column

2. 关系数据完整性

上述关系定义的一个重要结果是每个关系都有一个主码，我们知道，关系是一集合，而集合意味着它不会有重复元组，意味着任何时刻，关系中没有两个元组是完全一样的。R 是具有 A1, …, An 的关系，K=(Ai, Aj, …, Ak)是 R 的候选码，当且仅当 K 满足如下两个不依时间而变化的特性：

①唯一性。任何时刻不能有 R 的两个元组具有完全相同的 Ai 值、Aj 值、…. Ak 值。

②最少性。不可能从 K 中剔除某 Ai 后仍然满足唯一性。每个关系至少有一个候选码，候选码中可指定一个为主码，其余为可选码(Alternate Key)。

外码(Foreign Key)是表 R2 的属性或属性集，它的值需与某关系 R1 的主码相匹配(R1 与 R2 可指同一个)。

关系数据完整性包括实体完整性、参照完整性和用户定义完整性。实体完整性要求 基本关系的主码中的属性值不能为空值。参照完整性指如基表 R2 包含一个外码 FK，则它的值应等于 R1 中的某 PK 值或完全为空值(所有参与 FK 值的属性全为空值)。R1 与 R2 可以指同一个表。基表是不依赖于其它关系表而存在的命名表。

3. 关系数据操纵

关系模型的操纵部分由称为关系代数的算子构成，它还包括赋值操作符。关系运算有并运算、交运算、差运算、广义笛卡尔积、选择、投影、除运算等，关系运算的特点是集合运算，无论是操作的对象还是操作的结果都是集合。

关系模型建立在严格数学概念的基础上，概念简单、清晰，易于用户理解和使用。现实实体及实体间的关系均以关系来表示，数据独立性强， 数据的物理存储和存取路径对用户透明，操作语言具有非过程化和说明性的特点，可大大降低编程难度。

1.1.2 SQL 语言

1. SQL 语言的诞生

SQL 发音为 ess-cue-ell，但许多人称之为 SEQUEL，是 Structured English Query Language 的缩写，它包括了数据模式定义，数据操纵和控制数据库中数据访问、完整性等各方面的功能。今天，它在关系数据库中得到了如此广泛的应用，主要应归功于：IBM 首先设计并把它加以实现；它是非过程化的，具有第四代语言的若干特性；SQL 语言的标准化工作在较早的时候就得到了重视，并推出了几个标准化版本。

E.F.Codd 在 1970 年发表的《大型共享数据库数据的关系模型》中引入了关系模型，该模型的一个重要方面是它的数学基础，1970～1979 年，Codd 还发表了另外九篇文章来进一步发展关系模型,在 1979年,他发表了《表达更多语义的扩展关系模型》(*Extending the Relational Model to Capture More Meaning*)，把已有的关系模型加以扩展，称为 RM/T。这一系列思想导致了大学、工业研究实验室的广泛研究和实验，并导致了众多关系产品在市场上的出现。在众多的研究中，一个重要方面就是涉及关系语言和其原型的实现。关系语言是实现抽象关系模型某些或全部特性的语言，多种关系语言是在 70 年代中期创造的,其中之一就是"结构英语查询语言",即 SEQUEL(是 Structured English Query Language 的缩写），是由 D. D. Chamberlin 和 IBM San Jose Research Laboratory (1974)的其他研究人员定义的，在 IBM SEQUEL-XRM 原型中首次实现。在 SEQUEL-XRM 实验的经验基础上，SEQUEL 的改进版 SEQUEL/2 于 1976～1977 年推出,取名为 SQL,并开始新的 System R 的开发,实现了 SEQUEL/2 语言的大部分，该原型在 1977 年运行，随后被安装到多台机器上。由于 SYSTEM R 的成功，促成了 70 年代后期 IBM 基于该技术的产品开发,特别是一些实现 SQL 语言的产品,事实上，ORACLE 早于 IBM 的产品走上市场。1981 年，IBM 又宣布了基于 DOS/VSE 环境的 SQL/DS，再后又宣布了它在 VM/CMS 上的 SQL/DS(1982),到 1983 年,MVS 上的又一产品 DB2(与 SQL/DS 基本兼容)也宣布走上市场,在随后的几年里,许多公司都宣布了基于 SQL 的产品,如 DG/SQL, Sybase 以及 Ingres 的 SQL 接口， Rdb/VMS 的 SQL 接口等，这样，从微机到大型机都有了 SQL 方言的产品在运行，SQL 已成为关系数据库领域事实上的标准。

2. SQL-86 与 SQL-89

在短时间里，涌现出了众多的关系数据库产品，导致了各种 SQL 方言的出现，即使是 IBM 的几个 SQL 版本也不完全兼容。为此，美国国家标准局于 1978 年授权成立 X3H2 技术委员会，起初是为了制定基于 CODASYL 的网络数据库语言 NDL 标准，该工作完成之后,授权制定关系数据库语言标准(RDL)。该委员会制定的 SQL语言与 IBM 的 SEQUEL/2 大体相同，另外，ISO 也着手制定类似的标准。由于标准需求的迫切性，ANSI 在 1986 年发表了 ANSI X3.135-1986。由于一些程序上的问题，ISO 推迟到 1987 年发表了 ISO 9075-1987 ,这个标准就是所谓的 SQL-86 。由 于 SQL-86 缺少参照完整性，导致了 ANSI X3.135-1989, Database Language SQL With Integrity Enhancement 及 ISO/IEC 9075: 1989 的出现。

ANSI SQL-86 规定了模式定义语言(SQL-DDL）和模块语言与数据操纵语言(SQL-DML)的语法和语义。定义了建立数据库的基本语句规范，包括对关系数据库模式的基表，视图等的建立（但没有包括对所创建的对象的删除语句，也没包含建立索引，聚集等检索辅助设施）。定义了基本的操作和权限管理语句，如对表或表中某些列的查询(SELECT)，插入新行(INSERT),删 除(DELETE) 修 改(UPDATE)等操作权限的管理。另外，还定义了游标申请和关闭(OPEN、CLOSE)语句,以及与游标相关的 DELETE 语句、FETCH 语句、INSERT 语句、UPDATE 语句以及 SELECT 语句等。该标准还包含了用于事务管理的 COMMIT 和 ROLLBACK 语句等。

3.第二代 SQL 语言 SOL2

在完成了 SQL-89 的工作后，ANSI 和 ISO 又立即着手 SQL 的进一步完善和扩展工作，导致了 SQL-92 或 SQL2 的产生，在 1992 年发表了 ANSI 3.135-1992, Database Language SQL 及 ISO/IEC 9075:1992, Database Language SQL。

SQL-92 在 SQL-86 和 SQL-89 的基础上进行了许多方面的扩展。如增加了一些新的数据类型，包括可变长度的字符串类型，位串(BIT)及可变长度位串，复合字符集，时间(DATE/TIME)等。新增了差、交、外连接、自然连接等新的关系算子以及一些连接谓词，如 UNIQE, MATCH 和 OVERLAP，等等。

4. SQL3

SQL2 在 SQL-89 的基础上往前迈进了一大步，但还远未达到完美的境界，依然缺少某些非常重要的功能设施，如触发器(trigger)和规则(rule)。ANSI 和 ISO 正积极开展 SQL3 的研制工作。SQL3 中将加入一些面向对象的新成份，如元组型、引用、抽象数据类型等，以满足对象关系数据库的要求。目前，ORACLE 在此方面已经做出了许多工作，最近推出了具有对象关系管理功能的 V8.0 版本。

1.2 ORACLE 关系数据库系统发展概述

1.2.1 发展历程

1977 年，ORACLE 公司成立，致力于研制关系数据库管理系统。1979 年，ORACLE 推出了第一个商业化的关系型数据库管理系统。1983 年，采用 ANSI C 重写 ORACLE 内核，使得它易于移植到各种平台，成为第一个具有真正开放性的数据库管理系统。1984 年，推出 ORACLE PC 版解决了 64k 内存的限制问题。同时，推出与数据库结合的第四代语言开发工具系列。1986 年，ORACLE 公司推出 V5.1，它的开放型产品 SQL*STAR 是具有分布查询功能的关系型数据库系统，满足局部自治和场地透明原则。1988 年，推出 V6.0，并具有 ORACLE*tps(事务处理子系统)选件，是一个高性能，有容错能力的数据库管理系统，适用于联机事务处理和大型数据库应用环境。1991 年，推出 V6.2，它能并行运行在多个处理机上，如 DEC 的 VAX Cluster 上，支持两种运行方式，即排它方式和共享方式。1992 年，ORACLE 协同服务器 V7.0 问世，它在数据存取语言、操作系统、用户接口、网络通信协议方面严格遵循工业标准，支持分布事务和分布处理。1997 年，ORACLE 推出对象关系数据库版 V8.0。

1.2.2 ORACLE 数据库的特点

关系系统的优点概括地说就是它的简单性，对用户来讲，这种简单性也就意味着它的易用性和高的生产开发效率。ORACLE 数据库具有以下一些特点：

需要全本请访问 www.ertongbook.com

- 大数据库和空间的管理。它支持数据库的规模可达到几百个 GB。允许用户控制空间的使用；
- ORACLE 支持大量用户的联机访问，保证数据的一致性，并使竞争干扰降到最低程度；
- 事务处理的高性能；
- 高可用性，ORACLE 支持全天 24 小时不间断使用；
- ORACLE 可以选择控制数据的可用性，如在数据库级或在数据库的某个部分上进行；
- 广泛采用工业标准；
- 完善的安全性措施，如各种权限控制和审计措施；
- 增强用户完整性的管理，如触发器机制；
- 具有分布的客户/服务器处理环境；
- 完整的分布式数据库功能；
- 很好的可移植性和兼容性；
- 系统的可连接性。

1.3 ORACLE 的产品结构及基本安装过程

1.3.1 ORACLE 产品结构

ORACLE 数据库系统的层次结构如用图 1.1 所示，最内层是数据库的数据，数据

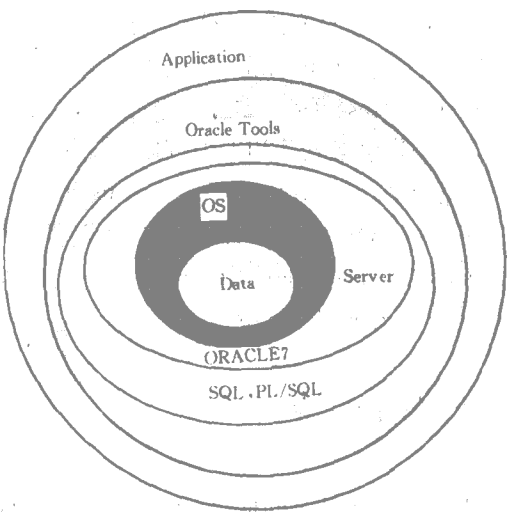


图 1.1 ORACLE 数据库系统的层次结构

以操作系统的文件形式存在。ORACLE 数据库管理系统建立在操作系统之上，其外是 SQL 语言或 PL/SQL 语言，再外层是开发工具和应用。在工具层，包括若干工具系列，如应用开发工具,就有 CASE*Dictionary、CASE*Designer、CASE*Generator、SQL*Plus、SQL*Forms、SQL*Menu、SQL*ReportWriter、SQL*TextRetrieval Pro*C、Pro*Ada 等，数据库连接产品，如 SQL*Net、SQL*Connect、SQL Palm Link 等。

1.3.2 ORACLE 安装

ORACLE 数据库系统可以安装在许多计算机平台上，对不同的平台，安装过程中的参数设置与过程可能有一些不同。安装 ORACLE 系统的基本过程如图 1.2 所示。

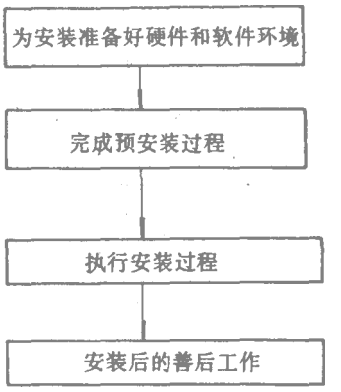


图 1.2 基本安装过程

1.为安装准备好硬件和软件环境

硬件环境要求：包括根据 ORACLE 数据库管理系统所需的条件来选择合适的硬件配置，如内存与磁盘容量、CPU 的指标、终端的型号，还包括支持客户/服务器运行方式或分布式数据库所需的网卡、读取软件介质的相应的 I/O 设备，如磁带机或光驱等。

软件环境要求：主要包括操作系统版本、语言编译器及版本(如 Ada 编译器、C 编译器、Fortran 编译器等)和网络软件(如 TCP/IP 协议)等。

2.设置安装环境

预安装指实际安装之前系统管理员所需完成的一些准备工作，如建立系统帐号和目录，如组号。应当注意，系统管理员注册名 ORACLE 所选择的组号应与/etc/group 文件中名为 dba 的组号一致,名字 dba 被编码到 rdbms/lib/config.c 中，不能改变，如改变则在安装时要重新连接。ORACLE 用户比 dba 组中的其他成员拥有更多的特权，它是系统文件的拥有者，有权改变文件的属主及存取方式，还可以为其他 DBA 创建不同的注册名，只要它与 dba 同组即可。

预安装过程中的另一个主要工作是改变系统的某些参数。在 UNIX 操作系统中，如信号量参数 SEMMNI 表示系统中的信号量集个数；SEMMSL 表示系统中每个信号量集中的最大可能的信号量个数；SEMMNS 表示系统中最多可能的信号量个数。另如系统中有关共享存储器方面的参数设置，如 SHMMAX 表示一个共享存储段的最大字节数；SHMSEG 表示用户进程可连接的共享段数目；SHMMNS 表示系统中可分配的共享存储段数目；AHMMNI 表示系统共享存储段的数目，等等。

3.执行安装过程

执行 ORACLE 提供的 INSTALL 安装程序，在安装中提供一些参数后，安装程序

将自动完成系统的安装。完成的工作包括：

重新连接和安装每个产品的执行文件；

初始化数据库；

装载系统字典；

装载在线帮助文件；

装载在线演示数据。

系统中，一些常见的文件后缀的含义见表 1.2

表 1.2 常见的文件后缀

.ora	系统配置文件	.msg	消息文件的文本形式
.dbf	数据库运行时使用的数据文件	.ctl	SQL*Loader 的控制文件
.bsq	数据字典定义文件	.mk	Makefile 文件
.sh	Shell 脚本文件	.dat	用于 SQL*Loader 的数据文件
.trc	ORACLE 追踪文件	.rpt	SQL*Report 中用于 rpt 的输入文件
.sql	SQL 语言文件	.rpf	SQL*Report 中用于 rpf 的输入文件
.frm	SQL*FORMS 产生的二进制文件	.DIST	产品清单记录文件
.a	ORACLE 库文件	.key	一些码值文件
.inp	SQL*FORMS 中的输入文件	.update	产品修改脚本文件
.lst	SQL*PLUS 产生的 SPOOL 文件	.verify	检测产品是否成功安装的脚本文件
.log	SQL*DBA 产生的 SPOOL 文件	.install	产品安装脚本文件
.msb	原始消息文件		

第 2 章 ORACLE 数据库管理系统软件结构

不管 ORACLE 数据库系统在哪个平台上运行，一个正在运行的可供用户操作和使用的数据库总是和一个 ORACLE 实例联系在一起的，ORACLE 实例由称为系统全局区(SGA，System Gloabal Area)的一个内存区域和 ORACLE 进程两部分组成。SGA 用于存放数据库实例的数据和控制信息，以实现对数据库中数据的管理和操作。进程是操作系统实现资源共享和调度控制的单元，对于 ORACLE 数据库管理系统来说，进程由用户进程、服务器进程和后台进程所组成。

在 ORACLE 数据库管理系统中，每当启动一个 ORACLE 数据库时,系统首先启动实例，分配 SGA 并启动一个或多个 ORACLE 进程，然后由该实例安装一个数据库，使该数据库与一个已启动的数据库相联。在安装数据库时，实例要查询控制文件。在没打开数据库之前，数据库还处于关闭状态，用户尚不能对其存取，只有 DBA 能对其存取。当打开数据库之后，用户才能对其进行正常操作。操作完成后，关闭数据库，卸下数据库，最后关闭实例。

同一机器上可以同时运行多个实例，每个实例访问它自己的物理数据库。在松耦合系统中，使用 ORACLE 的并行服务器可以使多个实例安装同一个数据库，这样，这些实例就可以共享这一物理数据库，当 Trusted ORACLE 版本在 OS MAC 模式下运行时，一个实例可以安装多个数据库。

2.1 进程结构概述

ORACLE 实例有两种形式：单进程 ORACLE(也称为单用户 ORACLE)，在这样的数据库系统中，所有的 ORACLE 代码和应用程序代码合为一个进程，单进程 ORACLE 实例仅允许一个用户存取。MS-DOS 下的单进程实例结构如图 2.1。

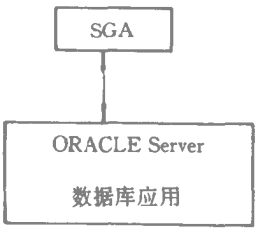


图 2.1 单进程 ORACLE 实例

多进程 ORACLE(也称多用户 ORACLE)使用多个进程来分别执行 ORACLE 的不同功能部分，每个与 ORACLE 系统相联的用户使用一个单独的用户进程，允许多个用户进程对 ORACLE 数据库的共享使用，这也是数据库系统更为常见的情况。

当用户运行一个应用程序（如 PRO*C 程序)或

ORACLE 工具(如 DQL*DBA)时就创建相应的一个用户进程。ORACLE 进程又分为服务器进程和后台进程两种。

通常，当 ORACLE DBMS 和应用程序运行在同一机器平台上，而不是通过网络相互连接时，用户进程和服务器进程被合二为一结合为一个单独的进程，以减少系统开销。然而，当应用程序和 ORACLE DBMS 分别运行于不同的机器上时，一个用户进程就和一个相应的数据库服务进程相连接(在多线索情况下稍有不同),服务器进程处理与之相连的用户进程的请求，它与用户进程相通讯， 完成用户对 ORACLE 的请求服务，如: 分析和执行 SQL 语句；将数据库中的数据块读入 SGA 数据库缓冲区；把 SQL 语句的执行结果返回给用户进程。

ORACLE 后台进程包括:

写数据库进程 DBWR(Database Writer);

写日志进程 LGWR(Log Writer);

检测点进程 CKPT(Checkpoint);

系统监控进程 SMON(System Monitor);

进程监控进程 PMON(Process Monitor);

日志归档进程 ARCH(Archiver);

系统恢复进程 RECO(Recover);

封锁进程 LCKn(Lock);

请求分发进程 Dnnn(Dispatcher)。

ORACLE 实例的多进程结构如图 2.2。

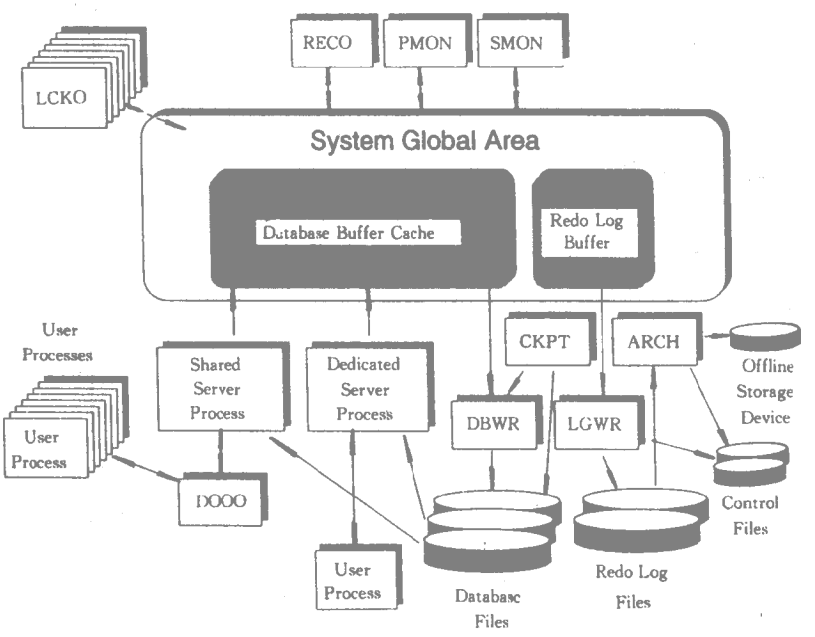


图 2.2 多进程结构

2.2 多进程结构

2.2.1 DBWR 进程

DBWR 进程负责数据库缓冲区的管理，把所有数据库文件缓冲区中被修改数据写入到数据文件。DBWR 维持一个自由缓冲区队列和“脏缓冲块”队列，以便为用户进程提供所需的自由缓冲块。为此，要完成的工作主要有：

①把数据缓冲区中的脏缓冲块写入数据库的数据文件中。

②使用 LRU(最近最少使用算法）使最近使用最多(Most-Recently-Used 简称为 MRU)的数据块保留在内存中，这样，对这些数据进行再次访问时，直接对内存中的数据进行即可，而不必从外存中读入，可有效地减少 I/O 次数。

③实现延迟写策略，进一步优化 I/O。

在下列情况下，DBWR 将脏缓冲块写入数据文件中：

- 当服务器进程将脏缓冲块移入脏表时，如果脏表的长度达到某个阈值，则该服务器进程通知 DBWR 执行写操作,阈值为初始参数 DB BLOCK WRITE BATCH 值的一半。
- 一个服务器进程在查找 LRU 表时，如果扫描了 DB BLOCK MAX SCAN CNT 个缓冲块还没有找到自由块，就停止查找，并通知 DBWR 进程进行写操作。这时，因为没有可用的足够多的自由缓冲区，DBWR 必须进行脏块清理，腾出更多的自由缓冲区。
- 每当 DBWR 在 3 秒内未活动时，则通知 DBWR 进程在 LRU 表中查找下一组尚未查找的缓冲区(该组缓冲区的数目是初始化参数 DB BLOCK WRITE BATCH 值的 2 倍），并将找到的脏缓冲块写入磁盘，因此，如果数据库是空运转时，则 LRU 表中的全部脏缓冲块最终都会写入磁盘。
- 发生检测点时，LGWR 将通知空闲的(Idle)DBWR 进程对脏缓冲块进行写操作。

一些平台上，DBWR 进程可有多个，初始化参数 DB_WRITERS 控制 DBWR 的个数，在有多个 DBWR 进程的情况下，可将一些缓冲区块写入一个磁盘，而将另一些缓冲区块写入另一个磁盘。

2.2.2 LGWR 进程

LGWR 进程负责对日志缓冲区的管理，它把日志缓冲区中的日志项写入磁盘中的日志文件中，在下列情况下，该进程进行写操作：

①当用户进程提交一个事务时，LGWR 写入一个提交记录，并把日志缓冲区中的日志项写入日志文件；

需要全本请访问 www.ertongbook.com

②当日志缓冲池填满达到 1/3 时，LGWR 执行写操作，把日志缓冲区内容写入日志文件；

③当 LGWR 在 3 秒内未活动时，LGWR 就要执行一次写操作。

每个实例只有一个 LGWR 进程，若实例中配置镜像日志文件组时，LGWR 要同时完成对活动镜像日志组文件的写入操作，如组中有某个文件被破坏或不可用时，LGWR 则将日志写入组中其它文件，并在 LGWR 跟踪文件和警告(Alert)文件中记载该错误。当组中所有文件遭受破坏或由于还未归档不能使用时，LGWR 将不能继续正常工作。

在数据库操作活动频繁，LGWR 需要不断将日志缓冲区写入日志文件时，它使用一种组提交(Group Commit)机制来减少磁盘 I/O 次数，例如，在事务 A 提交一个事务时，LGWR 必须把日志的重做项(Redo Entry)写入磁盘,与此同时,用户 B 也发出事务提交语句,这时，LGWR 要把前一个用户 A 的重做项写入磁盘后才能开始把用户 B 的日志项写入磁盘，在用户 A 的重做日志项写完后,正在等待(还未提交)的事务 B 的所有重做项可以一次写入磁盘，这就比一次只写一个重做项的方式减少了 I/O 次数,当然,在提交事务较多时，LGWR 可把提交记录一次写入磁盘。

2.2.3 CKPT 进程

在数据库检测点出现时，DBWR 进程将数据库缓冲区的所有脏缓冲块写入磁盘上的数据文件中。而 CKPT 进程则在检测点完成后用于修改数据文件和控制文件的标题。通常情况下，该修改工作由 LGWR 进程完成，但是，如果使用 LGWR 会明显降低系统性能时,可启动 CKPT 进程来完成。在许多情况下，不使用 CKPT 进程，只有当数据库有许多数据文件，而使用 LGWR 又降低系统性能时才使用 CKPT。初始化参数 CHECKPOINT_PROCESS 控制 CKPT 进程能否被使用。取值为 TRUE 时，表示能使用，缺省值为 FALSE。

2.2.4 SMON 进程

SMON 进程负责完成实例启动时的自动恢复、清理不再使用的临时段以及合并相邻的自由区间(Free Extent)，形成更大的自由区间块。在具有并行服务器的环境下，对有故障的实例进行恢复。SMON 进程按照一定的规律被唤醒，检查是否需要完成一些相应的处理。其他进程需要时也可调用它。

2.2.5 PMON 进程

PMON 进程主要用于实现用户进程故障恢复，清理内存区和释放该进程所拥有的资源,具体为:

清理不正常结束的连接；
回滚未提交的事务；
释放由结束进程所保持的块；
释放分配给失败进程的 SGA 资源；
发现死锁，并通过回滚该事务来自动解除死锁。

PMON 进程也像 SMON 进程一样，按照一定的规律被唤醒，检查是否需要完成一些相应的处理。

2.2.6 RECO 进程

RECO 进程是在具有分布式选项时才有的一个进程，用于自动解决分布事务故障，初始参数 DISTRIBUTED_TRANSACTION 参数必须大于 0，如该参数设置为 0，实例启动时将不启动 RECO 进程。一个结点上的 RECO 进程将自动连接到未正常结束的分布事务的所有其它结点，当远程数据库系统或网络不正常而不能建立连接时，RECO 将在一段时间内再进行连接，每次时间间隔以指数形式增加。一旦连接建立，RECO 就可自动地处理未正常提交的事务，当这种分布式事务被处理后，数据库未决事务表(Pending Table)DBA_2PC_PENDING 中相应的行就被清除。

2.2.7 ARCH 进程

ARCH 归档进程把已填满的在线日志文件拷贝到一个指定的存储设备上，仅当数据库运行在归档模式时可以进行自动归档，或日志文件组开关(Switch)出现时，或用手工归档时就要进行 ARCH 操作，把日志文件归档，以便对介质故障发生时进行数据库系统恢复。参数文件中可设置 LOG_ARCHIVE_START 参数，可以使 ARCH 进程在实例启动时就启动。当然，实例启动后，数据库系统管理员还可以手工启动和关闭该进程。

2.2.8 LCKn 进程

LCKn 进程用于并行服务器系统，主要完成实例之间的封锁，LCKn 进程最多可达 10 个(LCK0，LCK1，…，LCK9)。但在多数并行数据库服务器系统中，一个封锁进程就够了。

2.2.9 请求分发调度进程(Dispatcher Process, Dnnn)

请求分发调度进程允许用户进程对有限服务器进程的共享。当没有分发进程时，

一个用户进程需要一个相应的服务器进程，然而，在多进程服务器中，可以通过分发进程这个机制，减少共享服务器进程个数，满足同样多用户的服务请求，分发进程把用户请求放入队列之中，系统服务进程可以从队列获得请求信息，具体完成对请求的处理，把处理结果也放入队列，由分发进程把结果交给用户进程。

在多线索中的这种处理过程需要 SQL*NET 的支持，分发进程把它自身监听请求的地址信息告诉 SQL*NET 的监听进程。这样，SQL*NET 的监听进程就可以把来自用户进程的连接请求转发给合适的一个分发进程(分发进程可以有不止一个),当用户进程请求连接时，SQL*NET 的监听进程将捕获该信息，并对它进行分析，找到一个合适的分发进程，并返回该分发进程的监听地址，这样，就可以使用户进程自己连接到相应的分发进程上。

2.2.10 跟踪文件、警告文件与后台进程

每个服务进程和后台进程均可以把检测到的各种内部错误及其相应信息写入到相应的跟踪文件中，服务器进程仅在初始参数 SQL_REACE 置为 TRUE 时，才自动进行。另外，在会话控制语句中也可以该参数的值。如：

ALTER SESSION SET SQL_TRACE＝TRUE;

每个数据库同样还有一个警告文件，该文件包含数据库的各种信息和错误，如：

①内部错误(ORA-600)、块错误(ORA-1578)、死 锁 错误(ORA-60) ；

② 管理操作信息， 例如 CREATE/ALTER/DROP、DATABASE/TABLESPACE/ROLLBACK SEGMENT 等 SQL 语句,和 STARTUP、SHUTDOWN 、ARCHIVE LOG. RECOVER 等 SQL*DBA 语句；

③与共享进程和分发调度进程相联系的有关信息和错误；

④快照自动刷新过程中的一些错误。

2.3 多进程 ORACLE 实例的进程配置

ORACLE 实例中的进程有如下三种配置方案：

User/Server 进程结构；

专用服务器进程结构；

多线索服务器进程结构。

2.3.1 User/Server 进程结构

在这种进程结构中，用户的应用程序代码和 ORACLE 数据库服务进程代码合并为一个进程来执行，结构如图 2.3。

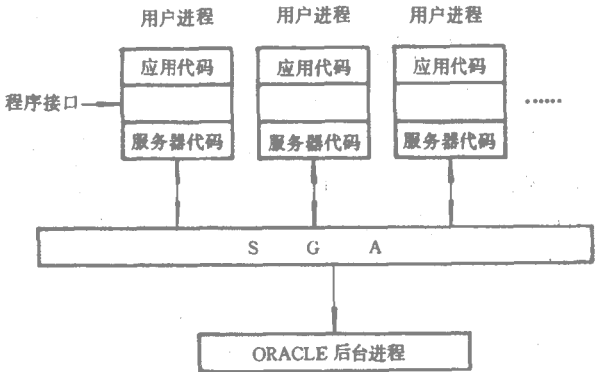


图 2.3 User/Server 进程结构

2.3.2 专用服务器进程结构

在这种进程结构中，ORACLE 系统中的每个用户都有一个单独的用户进程，即和一个与此用户进程相对应的 ORACLE 数据库服务进程，用户进程和服务进程成为一一对应关系，服务进程处理用户进程的请求，完成对数据库的操作。这种结构也称为两任务(Two-Task)ORACLE，结构如图 2.4。

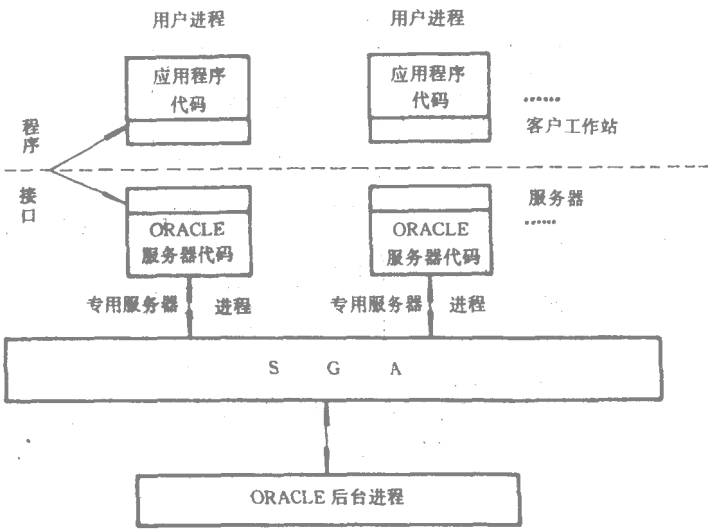


图 2.4 专用服务器进程结构

图中程序接口部分实现两进程之间的通信，可由 SQL*NET 或其他机制实现(如 UNIX 平台中的 IPC)。

2.3.3 多线索服务器进程结构

多线索服务器进程结构允许多个用户进程共享一个可用的服务器进程池，用户进程连接到一个分发调度进程，由分发调度进程把用户的请求分发给一个可用的服务器进程，并把结果返回给用户进程。这种结构的优点在于它扩大了用户数目，降低了系统开销。其结构如图 2.5 所示。

其中包括网络监听进程、分发调度进程和共享服务器进程。网络监听进程属于 SQL*NET 产品，负责用户进程和分发调度进程之间的连接。分发调度进程负责把用户进程请求放入到 SGA 中的请求队列，在有可用的共享服务器进程时，分发调度进程就为之分配用户请求，服务器进程完成服务请求后，就把结果登录到 SGA 的响应队列，最后由分发调度进程将响应队列中的结果返回给用户。共享服务器进程具体处理用户对数据库操作的请求，在多线索服务器进程结构中，一个共享服务器进程可为任何用户的请求进行服务。值得注意的是，即使用户进程和共享服务器进程位于同一机器上，也要使用 SQL*NET 才能使两者产生上述连接关系。另外，通过分发调度进程连接的服务器进程不能完成诸如实例启动和关闭、介质恢复等工作，此时，需要连接到一个专用的服务器进程而不是一个分发进程，由 SRVR = DEDICATED 指明并以 Internal 用户进入即可 。

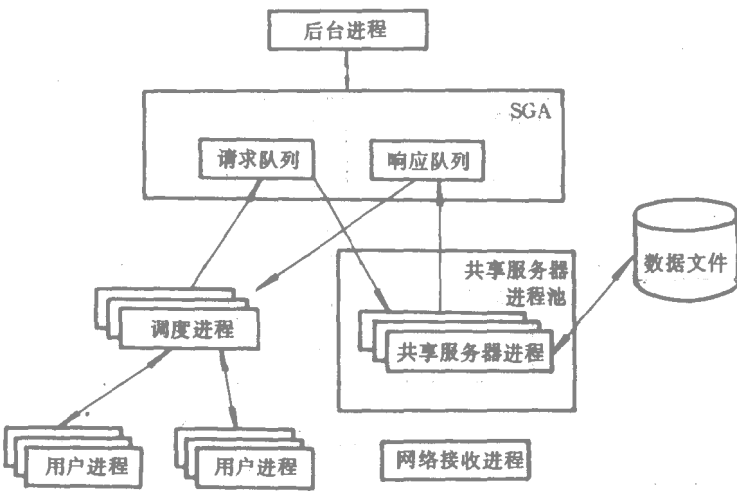


图 2.5 多线索服务器进程结构

在多线索服务器进程结构中，数据库管理员可通过设置有关的初始化参数来管理进程，如 MTS_DISPATCHERS、MTS_LISTENER_ADDRESS 等参数。

2.4 SGA(系统全局区)的结构

SGA 是 ORACLE 为一个实例分配的一组共享内存区，它包含该实例的数据和控制信息，在有多个用户与同一个实例相连接时，SGA 中的数据可为多个用户共享，所以，也把它称为共享全局区（Shared Global Area)，或称为系统全局区（System Global Area)，SGA 在实例启动时被自动分配，实例结束时收回。SGA 中所包含的共享信息分为多种类型，根据类型的不同，可将 SGA 分为数据库缓冲存储区、日志缓冲区、共享池、请求和响应队列(多线索服务器方式)等几个区域,如图 2.6 所示。

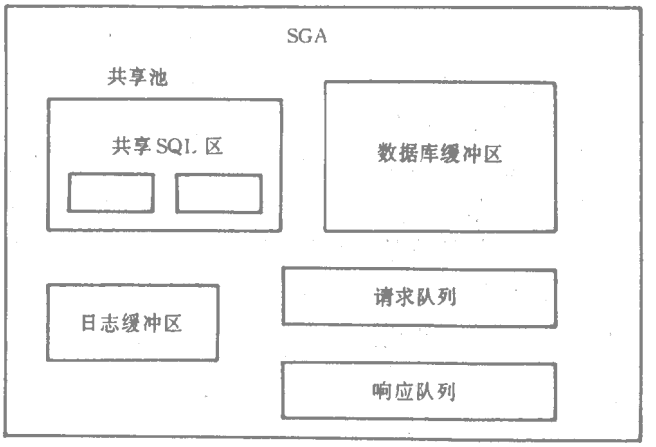


图 2.6 SGA 的结构

1 数据库缓冲存储区

数据库缓冲存储区用于存储数据库中数据块的副本，它由一组缓冲块所组成，这些缓冲块为所有的与该实例进程相连的用户进程所共享，缓冲块的数目由初始参数 DB_BLOCK_BUFFERS 的值确定，一个缓冲块的大小则由初始参数 DB BLOCK SIZE 确定。一个缓冲块的大小正好与一个数据块的大小一致，选择较大的数据块可以提高查询速度。

数据库缓冲存储区中的缓冲块被组织成两个队列：脏块队列和最近最少使用队列 LRU(Least-Recently-Used)，脏块队列包含已被修改但尚未写入数据库中的缓冲块。LRU 包含自由缓冲块(尚未使用的缓冲块)、当前正在存取的缓冲块以及已被弄脏但尚未移到脏块队列中的缓冲块。

当 ORACLE 进程访问 LRU 中的某些缓冲区时，进程就把该块移到 LRU 队列中 MRU(Most-Recently Used)端。当服务器进程需要将磁盘中的数据块读入数据库缓冲区时，它也要检索 LRU 队列，寻找一个尚未使用的缓冲块，当找到时就停止检索并将数据块读入该缓冲块，并把该块移至 LRU 的 MRU 端，然后将该块移至脏块队列。只在一种特殊的情况下，即用户进程对超过一定尺寸的表进行全表扫描时不是把它放入 LRU

需要全本请访问 www.ertongbook.com