第 2 章

Oracle数据库基本概念

本章主要向读者介绍 Oracle 数据库的基本概念，目的是让读者对什么是数据库或者 Oracle 数据库是什么样的数据库有一个初步的印象。在之后的章节中，本书将逐步深入地介绍 Oracle 数据库的各个细节部分。虽然 Oracle 数据库发展了很多年，技术也在不断更新，但是作为当下使用最广泛的数据库软件，Oracle 数据库的本质并没有发生变化。

2.1 关于关系数据库

每个组织都有其必须存储和管理的信息，以满足其需求。例如，公司必须收集和维护其雇员的人力资源记录。此信息对需要它的人必须是可用的。信息系统是一个正式的系统，用于存储和处理信息。信息系统可能是一组文件柜，其中包含许多文件夹，以及如何存储和检索文件夹的规则。但是，大多数公司现在使用数据库来自动化其信息系统。数据库是信息的一个有组织的集合，被作为一个整体来看待。数据库的目的是收集、存储和检索相关的信息，以供数据库应用程序使用。整个系统一般被称为 Database Management System（DBMS，数据库管理系统）。

通常，一个 DBMS 具有以下元素：

- 内核代码，此代码为 DBMS 管理内存和存储。
- 元数据的存储库，此存储库通常称为数据字典。
- 查询语言，此语言使应用程序能够访问数据。

实际上企业所使用的数据库泛指数据库应用程序，是一个与数据库进行交互，以访问和操作其数据的软件程序。第一代的数据库管理系统包括以下类型：

- 层次型：层次数据库把数据组织在树状结构中。每个父记录都有一个或多个子记录，类似于文件系统的结构。
- 网络型：网络数据库类似于层次数据库，但有一个区别，即记录之间是多对多的关系，而不是一对多的关系。

- 关系模型：E.F.Codd 在他 1970 年发表的论文《大型共享数据库数据的关系模型》中，定义了一个基于数学集合理论的关系模型。目前，最广泛接受的数据库模型就是关系模型。

关系数据库是一个符合关系模型的数据库。关系模型有以下主要方面：

- 结构：定义良好的对象，用于存储或访问数据库的数据。
- 操作：清楚定义的操作，使应用程序可以处理数据库中的数据和结构。
- 完整性规则：完整性规则用于管理在数据库中的数据和结构上的操作。

关系数据库实际上就是将数据存储在一组简单的关系中。关系是一个元组的集合，一个元组是一些属性值的无序集合。表是一个关系的二维表示，关系由行（元组）和列（属性）的形式构成。表中的每一行具有相同的列集。关系数据库是一个将数据存储在关系（表）中的数据库。例如，关系数据库可以在一个雇员表、部门表和薪金表中存储有关公司雇员的信息。

2.2 Oracle 数据库的发展史

Oracle 数据库的当前版本是超过 30 年的创新发展的结果。Oracle 数据库发展过程中的重要事件包括：

- 创立 Oracle 公司。1977 年，拉里·埃利森、鲍勃·穆勒、和爱德·奥茨成立了“软件开发实验室”咨询公司，其后又叫作“关系软件”（RSI）公司，1983 年，RSI 公司成为 Oracle 系统公司，再后来又成为 Oracle 公司。
- 第一个商用 RDBMS。1979 年，RSI 公司发布了 Oracle V2（版本 2），这是第一个商用的基于 SQL 的 RDBMS，它是关系数据库发展史中的一个里程碑事件。
- 可移植版本的 Oracle 数据库。Oracle 版本 3 发布于 1983 年，是第一个可以同时在大型机、小型机和个人电脑上运行的关系数据库。该数据库用 C 语言编写，使其可以被移植到多种平台上。
- 并发控制、数据分发和可扩展性等增强功能。版本 4 引入了多版本读一致性。版本 5 发布于 1985 年，支持客户端/服务器计算和分布式数据库系统。版本 6 增强了磁盘 I/O、行锁定、可扩展性以及备份和恢复。并且，版本 6 推出了 PL/SQL 语言第一版，这是专门针对 SQL 的过程化扩展。
- PL/SQL 存储程序单元。Oracle 7 发布于 1992 年，引入了 PL/SQL 存储过程和触发器。
- 对象和分区。

Oracle 8 发布于 1997 年，作为一种对象-关系数据库，支持许多新的数据类型。此外，Oracle 8 支持对大型表进行分区。

因特网计算，Oracle 8i 数据库发布于 1999 年，提供了互联网协议的原生支持和服务器端

的 Java 支持。Oracle 8i 是为因特网计算而设计的，这使得数据库可以在多层环境中部署。

Oracle 真正应用集群（Oracle RAC），Oracle 9i 数据库在 2001 年引入了 Oracle RAC，使得多个实例可以同时访问单个数据库。此外，Oracle XML 数据库（Oracle XML DB）引入了存储和查询 XML 的能力。

- 网格计算。Oracle 数据库 10g 在 2003 年引入了网格计算。此版本使得各个公司可以通过构建基于低成本服务器的网格基础设施来虚拟化计算资源。一个关键的目标是使数据库可以自我管理和自我优化。Oracle 自动存储管理（Oracle ASM）通过虚拟化和简化数据库存储管理，有助于实现这一目标。
- 可管理性、可诊断性和可用性。Oracle 数据库 11g，发布于 2007 年，引入了大量的新功能，使管理员和开发人员可以快速适应不断变化的业务需求。这种适应性的关键在于通过整合信息和尽可能使用自动化来简化信息基础架构。
- 2013 年 6 月 26 日 Oracle Database 12c 版本正式发布。与之前 10g、11g 里的 g 代表 grid 类似，12c 版本中的“c”是 cloud，代表云计算的意思。

2.3 认识数据库对象

常见的数据库对象包括表、索引、视图。

1. 表

对于初学者来说，对表的概念也有一定的认识。因为使用者对数据库的操作，90%以上是对表的操作。

什么是数据库表呢？它用于描述一个实体，例如雇员。可以使用一个表名（如 employees ）和一个列集来定义表。当用户创建表时，应该给出每一列的列名、数据类型和宽。

对于关系型数据来说，表其实就是许多行数据的集合。每条数据对应表中的一行。表中列用来标识实体的属性，而行用来标识实体的实例。例如，雇员实体的属性对应雇员 ID 列和姓氏列。行标识一个特定的雇员，可以选择性地为每个表列指定规则。这些规则称为完整性约束。例如，“非空”即是一个完整性约束。此约束强制每一行中的该列都包含一个值。

2. 索引

在关系数据库中，索引是一种与表有关的数据库结构，同时也是一个可选的数据结构，可以在表中的一个或多个列上创建索引。索引可以提高数据检索的性能。在处理一个请求时，数据库可以使用可用索引有效地找到请求的行。当应用程序经常查询某一特定行或特定范围的行时，索引很有用。

索引在逻辑和物理上都独立于数据。因此，可以删除和创建索引，而对表或其他索引没有任何影响。在删除索引后，所有应用程序可以继续运行。用户可以将索引的作用想象为一本图书的目录，根据目录中的页码快速找到所需的内容，所以在实际的企业生产环境中，索引的使

用是必不可少的。

当检索表中少量的行时，使用 Oracle 索引能够更快速地访问表中的这些行。索引存储了进行索引的列的值，同时存储包含索引值的行的物理 Rowid，唯一的例外是索引组织表（IOT），它使用主键作为逻辑 ROWID。一旦在索引中找到匹配值，索引中的 ROWID 就会指向表行的确切位置：哪个文件、文件中的哪个块以及块中的哪一行。可以在一列或多个列上创建索引。索引条目存储在 B-树结构中，因此遍历索引以找到行的键值只需要使用非常少的 O 操作。

在唯一索引的情况下，使用索引可能有两个目的：提高搜索行的速度，以及在索引列上实施唯一或主键约束。在插入、更新或删除表行的内容时，自动更新索引中的条目。删除表时，在该表上创建的所有索引也自动被删除。

3. 视图

视图允许用户查看单独表或多个连接表中数据的自定义表示。视图也称为“存储查询”，用户无法看到视图底层的查询细节。普通的视图不存储任何数据，而只存储定义，并且在每次访问视图时都运行底层的查询。

普通视图的扩展称为“物化视图”，允许同时存储查询的结果和查询的定义，从而加快处理速度，另外还有其他优点。对象视图类似于传统的视图，它可以隐藏底层表连接的细节，并且允许在数据库中进行面向对象的开发和处理，而底层的表仍然保持数据库关系表的格式。

2.4 表

比较常见的表类型有规则表（regular table），严格意义上来说又叫 heap table（堆表），这是最普通的一张表，其他类型还有 partition table、index-organized table、cluster 三种表类型，本节的重点就是讲解一些普通的表。

2.4.1 堆表

对于一张普通的表，它存放数据的规则是无序，假设把数据的存储空间看成学生宿舍楼一个连一个的房间，并不是第一个来的人就一定先在第一个房间。先来的人只要发现某个房间还有床位是空的就可以入住。

那么如何让他变成有序的呢？我可以专业创建一列来记录顺序。宿管在一楼门口发号码，进来一个同学，发一个号码，上面标注几号房间几号床位。这样所有入住的同学都是有序的。

堆表是数据库中最常见的表类型，以堆的形式进行组织。换句话说，表中的行没有按照任何特定的顺序存储，在 create table 命令中，可以指定子句来定义以堆表的形式组织的表，但是堆表属于默认值，所以一般用户在创建数据表时不需要添加这个关键字。在堆表中，每一行包含一列或者多列，每一列都有一种数据类型和一个长度，从 Oracle 8i 版本开始，列也可以包含用户定义的对象类型。

【示例 2-1】下面举一个简单的例子。

```
SQL> create table t
 (a int,
 b varchar2(50),
 c varchar2(50));           //创建一个表

Table created.

insert into t(a) values(1);
insert into t(a) values(2);
insert into t(a) values(3);
insert into t(a) values(4);
insert into t(a) values(5);
insert into t(a) values(6);

insert into t(b) values(111);
insert into t(b) values(222);
insert into t(b) values(222);
insert into t(b) values(333);
insert into t(b) values(444);.........
select a from t;

        A
----------
        1
        2
        3
        4
        5
        6//上面查询插入的结果是无序的，如何变成的有序的呢？加上 order by
SQL>
select a from t order by a desc;

        A
----------

        6
        5
        4
        3
        2
        1
```

创建一个较为复杂的单表：

```
CREATE TABLE employee
      (EMPNO NUMBER(4) NOT NULL,
       ENAME VARCHAR2(10),
       JOB VARCHAR2(9),
       MGR NUMBER(4),
```

```
      HIREDATE DATE,
      SAL NUMBER(7, 2),
      COMM NUMBER(7, 2),
      DEPTNO NUMBER(2));

INSERT INTO employee VALUES
      (7369, 'SMITH',  'CLERK',    7902,
      TO_DATE('17-11-1980', 'DD-MM-YYYY'),  800, NULL, 20);
INSERT INTO employee VALUES
      (7499, 'ALLEN',  'SALESMAN',  7698,
      TO_DATE('20-11-1981', 'DD-MM-YYYY'), 1600,  300, 30);
INSERT INTO employee VALUES
      (7521, 'WARD',   'SALESMAN',  7698,
      TO_DATE('22-11-1981', 'DD-MM-YYYY'), 1250,  500, 30);
INSERT INTO employee VALUES
      (7566, 'JONES',  'MANAGER',   7839,
      TO_DATE('2-11-1981', 'DD-MM-YYYY'),  2975, NULL, 20);
INSERT INTO employee VALUES
      (7654, 'MARTIN', 'SALESMAN',  7698,
      TO_DATE('28-11-1981', 'DD-MM-YYYY'), 1250, 1400, 30);
INSERT INTO employee VALUES
      (7698, 'BLAKE',  'MANAGER',   7839,
      TO_DATE('1-11-1981', 'DD-MM-YYYY'),  2850, NULL, 30);
INSERT INTO employee VALUES
      (7782, 'CLARK',  'MANAGER',   7839,
      TO_DATE('9-11-1981', 'DD-MM-YYYY'),  2450, NULL, 10);
INSERT INTO employee VALUES
      (7788, 'SCOTT',  'ANALYST',   7566,
      TO_DATE('09-11-1982', 'DD-MM-YYYY'), 3000, NULL, 20);
INSERT INTO employee VALUES
      (7839, 'KING',   'PRESIDENT', NULL,
      TO_DATE('17-11-1981', 'DD-MM-YYYY'), 5000, NULL, 10);
INSERT INTO employee VALUES
      (7844, 'TURNER', 'SALESMAN',  7698,
      TO_DATE('8-11-1981', 'DD-MM-YYYY'),  1500,    0, 30);
INSERT INTO employee VALUES
      (7876, 'ADAMS',  'CLERK',    7788,
      TO_DATE('12-11-1983', 'DD-MM-YYYY'), 1100, NULL, 20);
INSERT INTO employee VALUES
      (7900, 'JAMES',  'CLERK',    7698,
      TO_DATE('3-10-1981', 'DD-MM-YYYY'),   950, NULL, 30);
INSERT INTO employee VALUES
      (7902, 'FORD',   'ANALYST',   7566,
      TO_DATE('3-11-1981', 'DD-MM-YYYY'),  3000, NULL, 20);
INSERT INTO employee VALUES
      (7934, 'MILLER', 'CLERK',    7782,
      TO_DATE('23-12-1982', 'DD-MM-YYYY'), 1300, NULL, 10);
```

2.4.2　临时表

临时表就是用来暂时保存临时数据（或叫中间数据）的一个数据库对象，和普通表有些类似，然而又有很大区别。它只能存储在临时表空间，而非用户的表空间。Oracle 临时表是会话或事务级别的，只对当前会话或事务可见。每个会话只能查看和修改自己的数据。

目前所有使用 Oracle 作为数据库支撑平台的应用大部分都是数据量比较庞大的系统，即一般情况下都是在百万级以上的数据量。当然在 Oracle 中创建分区是一种不错的选择，但是当发现应用有多张表关联并且这些表大部分都比较庞大，在关联的时候会发现其中的某一张或者某几张表关联之后得到的结果集非常小并且查询得到这个结果集的速度非常快，那么这个时候就应考虑在 Oracle 中创建“临时表”了。

对临时表的概念也可以这样理解，在 Oracle 中创建一张表，这个表不用于其他的什么功能，主要用于自己的软件系统一些特有功能，用完之后表中的数据就没用了。Oracle 的临时表创建之后基本不占用表空间，如果没有指定临时表（包括临时表的索引）存放的表空间时，插入到临时表的数据是存放在 Oracle 系统的临时表空间（Temp）中的。

Oracle 临时表有两种类型：会话级的临时表和事务级的临时表。

1. ON COMMIT DELETE ROWS

它是临时表的默认参数，表示临时表中的数据仅在事务（transaction）过程中有效，当事务提交（commit）后，临时表的暂时段将被自动截断（truncate），但是临时表的结构以及元数据还存储在用户的数据字典中。在临时表完成它的使命后，最好将其删除，否则数据库会残留很多临时表的表结构和元数据。

会话级的临时表的数据和当前会话有关系，当前 SESSION 不退出的情况下，临时表中的数据就还存在，临时表的数据只有退出当前 SESSION 时才被截断（truncate table）。

【示例 2-2】会话级别的临时表创建

```
create global temporary table tmp_test(id number,name varchar2(32)) on commit
preserve rows;
```

2. ON COMMIT PRESERVE ROWS

它表示临时表的内容可以跨事务而存在，不过，当该会话结束时，临时表的暂时段将随着会话的结束而被丢弃，临时表中的数据自然也就随之丢弃，但是临时表的结构以及元数据还存储在用户的数据字典中。在临时表完成它的使命后，最好将其删除，否则数据库会残留很多临时表的表结构和元数据。

事务级的临时表（默认）与事务有关，当进行事务提交或者事务回滚时，临时表的数据将自行截断，即当 commit 或 rollback 时，数据就会被截断，其他的特性和会话级的临时表一致。

【示例 2-3】事务级临时表的创建方法

```
create global temporary table tmp_test(id number,name varchar2(32)) on commit
delete rows;
```

2.4.3 索引组织表

索引组织表（index organized table，IOT）就是存储在一个索引结构中的表。

创建索引，可以更有效地查找表中的特定行，然而创建索引将带来额外的一些系统开销，因为数据库必须同时维护表的数据行和索引条目。如果表包含的列并不是很多，并且对表的访问主要集中在某一行上，那么应该怎么做呢？

在这种情况下索引组织表可能就是正确的解决方案。索引组织表是以 b 树索引的形式存储表中的行，其中 b 树索引的每个节点都包含作为键的列以及一个或多个非索引列。索引组织表最明显的优点在于只需要维护一个存储结构，而不是两个。类似的，表中主键的值只在索引组织表中存储一次，而在普通表中则需要存储两次。使用索引组织表也有一些缺点，有些表，例如记录事件的表可能不需要主键，或者在某些情况下不需要任何键，而索引组织表则必须有主键，同时索引组织表不可以是集群的成员。最后，如果表中有大量的列并且在检索表中的行时需要频繁地访问许多列，那么索引组织表可能就不是最佳的解决方案。

那么到底 IOT 表有什么意义呢？

使用堆组织表时，用户必须为表和表主键上的索引分别留出空间。IOT 不存在主键的空间开销，因为索引就是数据，数据就是索引，二者已经合二为一。但是，IOT 带来的好处并不止于节约了磁盘空间的占用，更重要的是大幅度降低了 I/O，减少了访问缓冲区缓存。尽管从缓冲区缓存获取数据比从硬盘读要快得多，但缓冲区缓存并不免费，而且也绝对不是廉价的。每个缓冲区缓存获取都需要缓冲区缓存的多个闩，而闩是串行化设备，会限制应用的扩展能力。

IOT 适用的场合有：

- 完全由主键组成的表。这样的表如果采用堆组织表，则表本身完全是多余的开销，因为所有的数据全部同样也保存在索引里，此时，堆表是没用的。
- 代码查找表，如果只会通过一个主键来访问一个表，这个表就非常适合实现为 IOT。
- 如果想保证数据存储在某个位置上，或者希望数据以某种特定的顺序物理存储，IOT 就是一种合适的结构。

IOT 提供如下好处：

- 提高缓冲区缓存效率，因为给定查询在缓存中需要的块更少。
- 减少缓冲区缓存访问，这会改善可扩缩性。
- 获取数据的工作总量更少，因为获取数据更快。
- 每个查询完成的物理 I/O 更少，因为对于任何给定的查询，需要的块更少，而且对地址记录的一个物理 I/O 很可能可以获取所有地址（而不只是其中一个地址，但堆表实现就只是获取一个地址）。如果经常在一个主键或唯一键上使用 BETWEEN 查询也是如此，因为相近的记录存在一起，查询时引入的逻辑 IO 和物理 IO 都会更少。

2.4.4 集群表

如果经常同时访问两个或多个表，例如一个订单表和一个项目表，那么创建集群表可能是

一个较好的方法，它可以改进应用这些表的查询性能。

在具有相关行是项目表和订单表共同拥有时，订单表和项目表的相关信息可以存储在同一个数据块中，从而减少检索的 IO 开销，还可以减少存储两个表共有的列所需的存储空间。

两个表共有的列称为集群键值。集群键值存储在集群索引中，针对集群索引的操作，非常类似于传统的索引。通过集群键值访问集群表时，可以改进对集群表的查询效率，共同的列只需存储一次，而不必针对每个行重复存储。相对于表执行的查询语句数量，如果需要频繁地对表执行插入更新和删除操作，则集群表的优点会减弱，此外经常对集群中的单个表进行查询，集群表的特点也不会得到更好的体现。

2.4.5　分区表

对表进行分区，或对索引进行分区，可帮助建立更加易于管理的大型表。可以将表分区为较小的部分，从应用程序的观点来看，分区是透明的。也就是说，在终端用户的 SQL 中不需要对任何特定分区进行显式的引用，用户唯一能够观察到的是在 WHERE 子句后面使用符合分区方案的筛选条件。对分区表进行查询，用户会发现 SQL 运行得更为快速，从 DBA 的角度看，对表进行分区有很多优点，如果表的一个分区位于已损坏的磁盘卷上，用户仍然可以查询表的其他分区。

DBA 是 Database Administrator 的英文缩写，即数据库管理员。

分区有三种类型：范围分区、散列分区以及从 Oracle 9i 开始引入的列表分区。从 Oracle 11g 开始，也可以根据父与子的关系进行分区，可以由应用程序控制分区，并且可以对基本分区类型进行很多组合，包括列表-散列、列表-列表、列表-范围和范围-范围等组合分区类型，分区表中的每一行只能存在于一个分区中，分区键用于对行数据指定正确的分区，分区键可以是组合键，最多可组合表中的 16 个列。

一般来说，推荐对于任何大于 2GB 的表，应尽量考虑对其进行分区并且表中包含历史数据，新的数据被增加到新的分区中。

表分区的优点：

- 改善查询性能：对分区对象的查询可以仅搜索自己关心的分区，提高检索速度。
- 增强可用性：如果表的某个分区出现故障，表在其他分区的数据仍然可用。
- 维护方便：如果表的某个分区出现故障，需要修复数据，只修复该分区即可。
- 均衡 I/O：可以把不同的分区映射到磁盘以平衡 I/O，改善整个系统性能。

下面罗列表分区的几种类型及操作方法。

1. 范围分区

范围分区将数据基于范围映射到每一个分区，这个范围是你在创建分区时指定的分区键决定的。这种分区方式是最为常用的，并且分区键经常采用日期。例如，你可能会将销售数据按

照月份进行分区。

当使用范围分区时，请考虑以下几个规则：

- 每一个分区都必须有一个 VALUES LESS THEN 子句，它指定了一个不包括在该分区中的上限值。分区键的任何值等于或者大于这个上限值的记录都会被加入下一个高一些的分区中。
- 除了第一个分区，其他分区都会有一个隐式的下限值，这个值就是此分区的前一个分区的上限值。
- 在最高的分区中，MAXVALUE 被定义。MAXVALUE 代表了一个不确定的值。这个值高于其他分区中任何分区键的值，也可以理解为高于任何分区中指定的 VALUE LESS THEN 的值，同时包括空值。

假设有一个 CUSTOMER 表，表中有数据 200000 行，将此表通过 CUSTOMER_ID 进行分区，每个分区存储 100000 行，将每个分区保存到单独的表空间中，这样数据文件就可以跨越多个物理磁盘了。

【示例 2-4】创建表和分区

```
create table customer
(
   customer_id number not null primary key,
   first_name  varchar2(30) not null,
   last_name   varchar2(30) not null,
   phonevarchar2(15) not null,
   emailvarchar2(80),
   status      char(1)
)
partition by range (customer_id)
(
   partition cus_part1 values less than (100000) tablespace cus_ts01,
   partition cus_part2 values less than (200000) tablespace cus_ts02
);
```

【示例 2-5】按时间分区

```
create table sales (
 product_id varchar2(5)
 , sales_date date
 , sales_cost number(10)
 , status varchar2(20)
 ) partition by range (sales_date) subpartition by list (status) (
 partition p1 values less than(to_date('2003-01-01', 'yyyy-mm-dd')) tablespace
rptfact2009(subpartition p1sub1 values ('active') tablespace rptfact2009,
subpartition p1sub2 values ('inactive') tablespace rptfact2009)
 , partition p2 values less than(to_date('2003-03-01', 'yyyy-mm-dd')) tablespace
rptfact2009(subpartition p2sub1 values ('active') tablespace rptfact2009,
subpartition p2sub2 values ('inactive') tablespace rptfact2009)
 );
```

2. 列表分区

列表分区的特点是某列的值只有几个，基于这样的特点可以采用列表分区。

【示例 2-6】列表分区

```
create table problem_tickets
(
   problem_id   number(7) not null primary key,
   description  varchar2(2000),
   customer_id  number(7) not null,
   date_entered date not null,
   status       varchar2(20)
)
partition by list (status)
(
     partition prob_active   values ('active') tablespace prob_ts01,
     partition prob_inactive values ('inactive') tablespace prob_ts02
);
```

3. 散列分区

这类分区是在列值上使用散列算法，以确定将行放入哪个分区中。当列的值没有合适的条件时，建议使用散列分区。散列分区为通过指定分区编号来均匀分布数据的一种分区类型，因为通过在 I/O 设备上进行散列分区，使得这些分区大小一致。

【示例 2-7】散列分区

```
create table hash_table
(
  col number(8),
  inf varchar2(100)
)
partition by hash (col)
(
  partition part01 tablespace hash_ts01,
  partition part02 tablespace hash_ts02,
  partition part03 tablespace hash_ts03
);
```

散列分区最主要的机制是根据 hash 算法来计算具体某条记录应该插入哪个分区中，hash 算法中最重要的是 hash 函数，Oracle 中如果要使用 hash 分区，只需指定分区的数量即可。建议分区的数量采用 2 的 n 次方，这样可以使各个分区间数据分布更加均匀。

4. 组合范围散列分区

这种分区是基于范围分区和列表分区的，表首先按某列进行范围分区，然后按某列进行列表分区，分区之中的分区被称为子分区。

【示例 2-8】组合范围散列分区

```
create table sales (
 product_id varchar2(5)
 , sales_date date
 , sales_cost number(10)
 , status varchar2(20)
 ) partition by range (sales_date) subpartition by list (status) (
 partition p1 values less than(to_date('2003-01-01', 'yyyy-mm-dd')) tablespace
rptfact2009(subpartition p1sub1 values ('active') tablespace rptfact2009,
subpartition p1sub2 values ('inactive') tablespace rptfact2009)
 , partition p2 values less than(to_date('2003-03-01', 'yyyy-mm-dd')) tablespace
rptfact2009(subpartition p2sub1 values ('active') tablespace rptfact2009,
subpartition p2sub2 values ('inactive') tablespace rptfact2009)
 );
```

5. 复合范围散列分区

这种分区是基于范围分区和散列分区的，表首先按某列进行范围分区，然后按某列进行散列分区。

【示例 2-9】复合范围散列分区

```
CREATE TABLE RANGE_HASH_TEST (
 TRANSACTION_ID NUMBER PRIMARY KEY
 ,ITEM_ID NUMBER(8) NOT NULL
 ,ITEM_DESCRIPTION VARCHAR2(300)
 ,TRANSACTION_DATE DATE
 ) PARTITION BY RANGE (TRANSACTION_DATE) SUBPARTITION BY HASH (TRANSACTION_ID)
SUBPARTITIONS 3 STORE IN (
 DINYA_SPACE01
 ,DINYA_SPACE02
 ,DINYA_SPACE03
 ) (
 PARTITION PART_01 VALUES LESS THAN(TO_DATE('2006-01-01', 'YYYY-MM-DD'))
 ,PARTITION PART_02 VALUES LESS THAN(TO_DATE('2010-01-01', 'YYYY-MM-DD'))
 ,PARTITION PART_03 VALUES LESS THAN(MAXVALUE)
 );
```

2.5 索引

Oracle 中有一些可用的索引类型，每种索引都适合于特定的表类型、访问方法或应用程序环境。下面几个小节将介绍最常见的索引类型的重点内容和特性。

（1）唯一索引。唯一索引是最常见的 B 树索引形式，经常用于实施表的主键约束，其作用在于确保索引列中不存在重复的值。可以在 T 表中 a 列上创建唯一索引。

（2）非唯一索引。非唯一索引帮助提高表访问的速度，而不会实施唯一性。例如，可以

在 t 表的 b 列上创建非唯一索引，从而提高按姓查找的速度。但是，对于任何给定的值，确实可以有许多重复的值，如果在 create index 语句中没有指定其他任何关键字，则默认在列上创建非唯一 B 树索引。

（3）反向键索引。反向键索引是特殊类型的索引，一般用于 OLTP（联机事务处理）环境中。在反向键索引中，反向每个列的索引键值中的所有字节。在 create index 命令中，使用 reverse 关键字指定反向键索引。

【示例 2-10】创建反向键索引

```
create index index_reverse on t(a) reverse;
```

插入到表中的内容分布在索引的所有叶键上，从而减少一些插入新行的写入程序之间的争用，如果在发出订单后不久就查询或修改订单，则反向键索引也可减少 OLTP 环境中这些“热点”的潜在性。

（4）基于函数的索引。基于函数的索引类似于标准的 B 树索引，不同之处在于它将被声明为表达式的列的变换形式存储在索引中，而不是存储列自身。当名称和地址可以作为混合内容存储在数据库中时，基于函数的索引就非常有用了，如果搜索标准是“Smith”，在包含值“Smith”的列上进行普通索引就不会返回任何值。另一方面，如果索引以全大写字母的形式存储姓，那么所有对姓的搜索都可以使用大写字母。

【示例 2-11】在 employee 表的 ename 列上创建基于函数的索引

```
create index  up_name on employee(upper(ename));
```

因此，使用如下查询的搜索将使用前面所创建的索引，而不是进行完整的表扫描。

【示例 2-12】使用索引

```
select EMPNO,ename from employee where upper(ename)= 'SMITH';

    EMPNO ENAME
---------- ----------
     7369 SMITH
```

（5）位图索引。在索引的叶节点上，位图索引结构与 B 树索引存在着较大的区别。它只存储索引列每个可能值（基数）的一个位串，位串的长度与索引表中的行数相同。与传统索引相比，位图索引除了可以节省大量的空间外，还可以大大缩短响应时间，因为在需要访问表自身之前，Oracle 就可以从包含多个 where 子句的查询中快速删除潜在的行。对于多个位图可以使用逻辑 and 和 or 操作来确定访问表中的哪些行。虽然位图索引可用于表中的任何列，但在索引列具有较低基数或大量不同的值时，使用位图索引才最有效。例如，PERS 表中的 Gender 列将有 NULL、M 或 F 值。

Gender 列上的位图索引将只有 3 个位图存储在索引中。另一方面，last name 列上的位图索引将有和表中行数基本相同的位图串数量。如果执行完整的表扫描而不是使用索引，则查找特定姓的查询将很可能花费较少的时间。在这种情况下，使用传统的 B 树非唯一索引将更有

意义。位图索引的一种变体称为“位图连接索引”，这种索引在某个表列上创建一个位图索引，此列常常根据相同的列与一个或多个其他的表相连接。这就在数据仓库环境中提供了大量的优点，在一个事实表和一维或多维表上创建位图连接索引，实质上等于预先连接这些表，从而在执行实际的连接时节省 CPU 和 IO 资源。

现在简单总结几点索引的作用：

- 索引是数据库对象之一，用于加快数据的检索，类似于书籍的索引。在数据库中索引可以减少数据库程序查询结果时需要读取的数据量，类似于在书籍中利用索引可以不用翻阅整本书即可找到想要的信息。
- 索引是建立在表上的可选对象；索引的关键在于通过一组排序后的索引键来取代默认的全表扫描检索方式，从而提高检索效率。
- 索引在逻辑上和物理上都与相关的表和数据无关，当创建或者删除一个索引时，不会影响基本的表。
- 索引一旦建立，在表上进行 DML 操作时（例如在执行插入、修改或者删除相关操作时），Oracle 会自动管理索引，索引删除，不会对表产生影响。
- 索引对用户是透明的，无论表上是否有索引，SQL 语句的用法不变。
- Oracle 创建主键时会自动在该列上创建索引。

2.5.1 索引的使用

1. 创建索引

【示例 2-13】创建索引

```
create [unique] | [bitmap] index index_name  --unique 表示唯一索引
on table_name([column1 [asc|desc], column2    --bitmap，创建位图索引
[asc|desc], …] | [express])
[tablespace tablespace_name]
[pctfree n1]                                  --指定索引在数据块中空闲空间
[storage (initial n2)]
[nologging]                                   --表示创建和重建索引时允许对表做 dml 操作，默认
情况下不应该使用
[noline]
[nosort];                                     --表示创建索引时不进行排序，默认不适用，如果数据
已经是按照该索引顺序排列的可以使用
```

2. 修改索引

【示例 2-14】重命名索引

```
alter index up_name rename to low_name;
```

【示例 2-15】合并索引（表使用一段时间后在索引中会产生碎片，此时索引效率会降低，可以选择重建索引或者合并索引，合并索引方式更好些，无须额外存储空间，代价较低）

```
alter index up_name coalesce;
```

【示例 2-16】重建索引

```
alter index up_name rebuild;
```

3. 删除索引

【示例 2-17】删除索引

```
drop index up_name;
```

4. 查看索引

【示例 2-18】查看索引

```
select index_name,index-type, tablespace_name, uniqueness from all_indexes where
table_name =' EMPLOYEE ';
```

2.5.2　索引建立的原则

事物都有两面性，一方面索引可以提高查询效率，另一方面，Oracle 数据库自动维护索引同时消耗数据库性能。

用户在创建索引时也有一些事项需要注意：

（1）如果有两个或者以上的索引，其中有一个唯一性索引，而其他是非唯一，这种情况下 Oracle 将使用唯一性索引而完全忽略非唯一性索引，至少要包含组合索引的第一列（即如果索引建立在多个列上，只有它的第一个列被 where 子句引用时，优化器才会使用该索引）；

（2）限制表中索引的数量。

- 创建索引耗费时间，并且随数据量的增大而增大。
- 索引会占用物理空间。
- 当对表中的数据进行增加、删除和修改时，索引也要动态维护，降低了数据的维护速度。

2.6　视图

下面各小节将介绍一般数据库用户、开发人员或 DBA 创建并使用基本视图的基础知识。

2.6.1　普通视图

通常称之为“视图”，不会占据任何内存空间，只有它的定义（查询）存储在数据字典中。视图底层查询的表称为“基表”，视图中的每个基表都可以进一步定义为视图。

视图有许多优点，它可以隐藏数据复杂性：高级分析人员可以定义包含 EMPLOYEE 表的视图，这样上层管理部门可以更容易地使用 select 语句检索相关信息，这种检索表面上看起来是使用表，但实际上是包含查询的视图，该查询连接 EMPLOYEE 表。视图也可以用于实施安

全性。 EMPLOYEE 表上的视图 EMPINFO 只能检索雇员名和雇员编号，并且该视图应定义为只读，从而防止更新该表。

【示例 2-19】创建普通视图

```
create view empinfo as select EMPNO, ENAME from employee with read only;
```

如果没有 read only 子句，则可以更新某行或添加行到视图中，甚至在包含多个表的视图上进行这些操作。视图中有一些构造可防止对其进行更新，例如使用 distinct 操作符、聚集函数或 group by 子句。当 Oracle 处理包含视图的查询时，它替换用户 select 语句中的底层查询定义，并且处理结果查询，就好像视图不存在一样。因此，在使用视图时，基表上任何已有索引的优点并没有改变。

2.6.2 物化视图

在某些方面，物化视图非常类似于普通视图：视图的定义存储在数据字典中，并且该视图对用户隐藏底层基查询的细节。但是，相似之处仅限于此。

物化视图也在数据库段中分配空间，用于保存执行基查询得到的结果集。物化视图可用于将表的只读副本复制到另一个数据库，该副本具有和基表相同的列定义和数据。这是物化视图最简单的实现。为了减少刷新物化视图时的响应时间，可以创建物化视图日志以刷新物化视图。否则，在需要刷新时就必须进行完全刷新，即必须获取基查询的全部结果以刷新物化视图。

物化视图日志为以增量方式更新物化视图提供了方便。在数据仓库环境中，物化视图可存储来自于 group by rollup 或 group by cube 查询的聚集数据，如果设置适当的初始参数值，例如 query rewrite enable，并且查询自身允许查询重写（使用 query rente 子句），则任何与物化视图执行相同类型的聚集操作的查询将自动使用物化视图，而不是运行初始的查询。无论物化视图的类型是什么，在基表中提交事务或根据需要刷新它时，系统都会自动对物化视图进行刷新。

物化视图在很多方面类似于索引，它们都直接和表联系并且占用空间，在更新基表时必须刷新它们，它们的存在对用户而言实际上是透明的。通过使用可选的访问路径来返回查询结果，它们可以帮助优化查询。

2.6.3 对象视图

面向对象（OO）的应用程序开发环境已经变得越来越流行，Oracle 10g 数据库完全支持数据库中本地化对象和方法的实现。然而，从纯粹的关系数据库环境向纯粹的 OO 数据库环境迁移并不是容易的过程，很少有组织愿意花费时间和资源从头开始构建新的系统，而 Oracle 10g 使用对象视图使这种变迁变得更为容易。

对象视图允许面向对象的应用程序查看作为对象集合的数据，这种对象集合具有属性和方法，而遗留系统仍然可以对 INVENTORY 表运行批处理作业。对象视图可以模仿抽象数据类型、对象标识符（OID）以及纯粹的 OO 数据库环境能够提供的引用。和普通视图一样，可以

在视图定义中使用 instead of 触发器来允许针对视图的 DML，这里使用的是 pl / sql 代码块，而不是用户或应用程序提供的实际 DML 语句。

2.7 小结

本章主要介绍了 Oracle 数据库的基本概念和一般的数据库对象，这部分的内容其实不单单是针对 Oracle 数据库，表或者索引的概念在各类关系型数据库中都是很重要的基本概念，作为数据库管理人员，日常需要处理很多与表和索引相关的工作，需要读者重点掌握。通过章的学习，对数据库有了初步的认识，在下一章中，将重点介绍 Oracle 数据库的体系结构，这部分的内容会是学习 Oracle 数据库的重中之重，在 DBA 的生涯中，所有的知识点都会是围绕其体系结构展开的。

第 3 章

Oracle数据库体系结构

对于一门技术的学习，尤其是像 Oracle Database 这种知识体系极其庞杂的技术来讲，从宏观上了解其体系结构是至关重要的。同时，未必是专业 DBA 人员才需要了解其体系结构（对于数据库专业人员来讲，这些固然是必备知识了），一般的技术人员如果对其有较深入的了解，也是大有益处的，毕竟技术思想很多时候都是相通的。

大多数的 Oracle 数据库使用者在实际工作中经常会遇到下面几个问题：

- 通常说 Oracle 数据库是什么？
- 如何理解 Oracle 实例？
- Oracle 实例由哪些部分组成，它们之间的作用是什么？
- 如何理解 Oracle 的物理结构？
- Oracle 的物理结构由哪些部分组成，它们之间的作用是什么？
- 如何理解 Oracle 的逻辑结构？
- Oracle 的逻辑结构由哪些部分组成，它们之间的作用是什么？

本章就从不同维度，如 Oracle 的内存结构、进程结构、存储结构等方面做相应描述。以上疑惑，读者可以从本章的学习中得到答案。

3.1 体系结构概述

众所周知，Oracle Database 是一款关系型数据库管理系统，同类的产品还有 MySQL、SQL Server 等。很多时候，多数人会把那个承载核心数据的系统笼统地称为数据库服务器，但从严格意义上来讲 Oracle 数据库是指 Oracle 数据库服务器（Oracle Server），由 Oracle 实例（Oracle Instance）和 Oracle 数据库（Oracle Database）组成。

（1）实例是数据库启动时初始化的一组进程和内存结构。实例启动时，系统首先在服务器内存中分配系统全局区（System Global Area），构成 Oracle 内存结构，然后启动必需的常驻内存的操作系统进程，组成 Oracle 的进程结构，内存结构和进程结构即构成 Oracle 实例。

（2）数据库指的是用户存储数据的一些物理文件，包括数据文件、重做日志文件、控制

文件、参数文件、密码文件、归档日志文件、备份文件、告警日志文件、跟踪文件等。其中，数据文件、控制文件、重做日志文件和参数文件是必需的，其他文件可选。一个实例只能对应/操作一个数据库，一个数据库可以由一个或多个实例操作（比如 RAC）。

从实例和数据库的概念上来看，实例是暂时的，不过是一组逻辑划分的内存结构和进程结构，会随着数据库的关闭而消失，而数据库其实就是一堆物理文件（控制文件、数据文件、日志文件等），它是永久存在的（除非磁盘损坏）。数据库和实例通常是一对一的，这种结构称为单实例体系结构。当然还有一些复杂的分布式结构，一个数据库可以对多个实例，像 Oracle 的 RAC。

从图 3-1 中可以清楚地看到每个组件之间的关联情况，这些组件包括内存组件和物理文件部分，组成了全部的 Oracle Database Server。

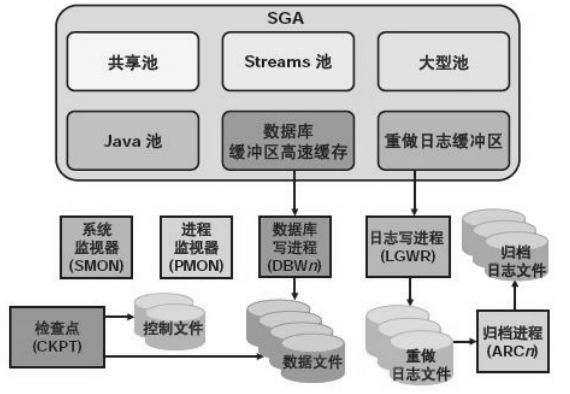

图 3-1　Oracle 数据库体系结构示意图

3.2　Oracle 数据库的连接

连接和会话都与用户进程密切相关，但意义却大不相同。

连接是用户进程和 Oracle DB 实例之间的通信路径。通信路径是使用可用的进程间通信机制（在一台同时运行用户进程和 Oracle DB 的计算机上）或网络软件（多台不同的计算机运行数据库应用程序和 Oracle DB 并通过网络进行通信时）建立的，如图 3-2 所示。

会话代表登录到数据库实例的当前用户的状态。例如，当某个用户启动 SQL*Plus 时，该用户必须提供有效的用户名和口令，然后系统会为该用户建立一个会话。会话从用户建立连接时开始，一直持续到用户断开连接或退出数据库应用程序时为止。

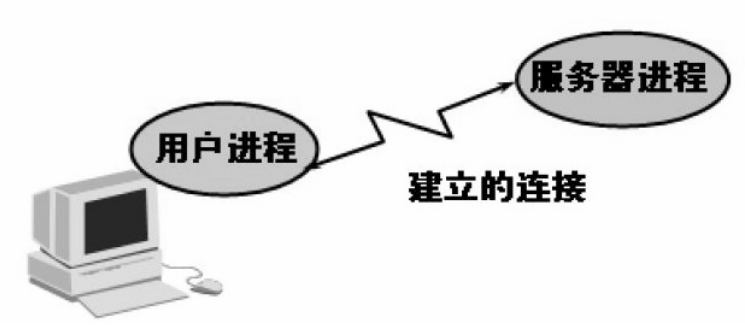

图 3-2　用户进程连接示意图

一个 Oracle DB 用户可以使用相同用户名创建多个会话，并让这些会话并存。例如，用户名/口令为 HR/HR 的用户可以多次连接到同一个 Oracle DB 实例。用户进程可以是一般的客户端软件，像 Oracle 的 sqlplus、sql developer，或者是一些驱动程序等都属于用户进程。

服务器进程有时会称为前台进程，当然是相对于后台进程（后面会提到数据库写入器、日志写入器等）来说的，服务器进程的主要作用就是处理连接到当前实例的用户进程的请求，对客户端发来的 sql 进行执行并返回执行结果。在专有服务器结构中，用户进程和服务器进程是一对一的，也就是说，当监听程序监听到客户端来了一个请求，会为其分配一个对应的服务器进程。还有一种结构为共享服务器，这种结构就不是一个用户进程对应一个服务器进程了，会通过调度程序进行协调处理，关于共享服务器连接，本文就不再赘述了。

上面描述了一些在进行数据库连接操作时大致的交互流程是什么样的。下面就来看看 Oracle 的实例内存结构。

3.3 实例内存区

由于内存结构和进程结构关系较紧密，进程会作用到对应的内存区域，比如数据库写入器作用到数据库缓冲区缓存中，日志写入器会作用到日志缓冲区，所以内存结构和进程结构会相互配合地进行描述。

Oracle 实例内存结构由 SGA（系统全局区）和 PGA（用户全局区）两部分组成。SGA 是一块共享的内存区域，也是最大的一块内存区域；PGA 则是用户会话专有的内存区域，每个会话在服务器端都有一块专有的内存区域就是 PGA。本文主要对 SGA 进行分析描述。SGA 组成如图 3-3 所示。

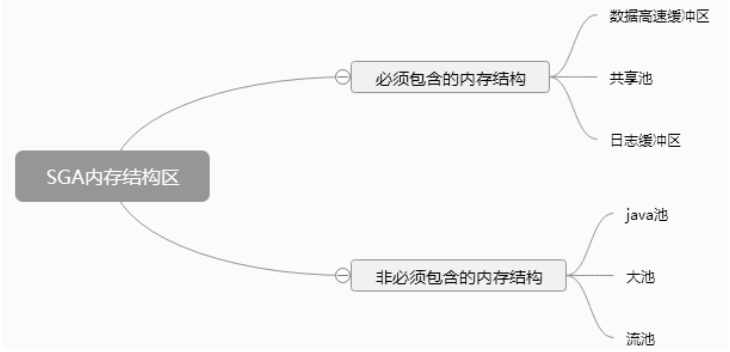

图 3-3　SGA 内存结构

3.3.1　数据库高速缓冲区

缓冲区缓存是 Oracle 用来执行 SQL 语句的工作区域，在更新数据时，用户会话不会直接去更新磁盘上的数据。

【示例 3-1】查询语句

```
select ename,salary from emp where name='SAM';
```

如示例 3-1 这样一条简单的查询语句，Oracle 是如何处理的呢？首先，当用户提交了该条 sql 语句，由对应的用户进程（比如常用的 sql developer）将其发送给服务器，监听程序监听到该条请求，会为其建立一个对应的服务器进程，然后服务器进程会先扫描缓冲区中有没有包含关键行（"SAM"）的数据块，如果有，这就算一次缓存命中了，然后相关行会传输到 PGA 进行进一步处理，最终经过格式化后展示给用户；如果没有命中，那么服务器进程会首先将对应行复制到缓冲区内，然后返回给客户端。

DML（insert，update，delete）操作同理，加入用户发送一条 update 语句，服务进程依然先去扫描缓冲区，如果缓存命中，则直接更新，数据变脏；如果没有命中，由服务器进程将对应数据块先从磁盘上复制到缓冲区内，再进行更新操作。

如果缓冲区存储的块和磁盘上的块不一致，该缓冲区就叫作“脏缓冲区”，脏缓冲区最终会由数据库写入器（DBWn）写入到磁盘中去。

数据库写入器是 Oracle 的一个后台进程。所谓后台进程，是相对于前台进程（服务器进程）来讲的。DBWn 的"n"意味着一个实例是可以有多个数据库写入器的。简而言之，DBWn 的作用就是将变脏了的缓冲区从数据库缓冲区缓存中写入到磁盘中的数据文件中去。

数据库缓冲区缓存区域和数据库写入器是比较重要的概念，别的数据库产品像 MySQL 也都有对应的实现，只不过叫法不一样罢了。了解时，要时刻意识到会话是不会直接更新磁盘数据的，会话的更新、插入、删除包括查询等都是先作用到缓冲区上，随后，DBWn 会将其中的脏缓冲区转储到磁盘上去。

那么 DBWn 什么时候写入呢？DBWn 是一个比较懒的进程，会尽可能少地进行写入，在以下四种情况下会执行写入：

- 没有任何可用缓冲区（不得不写）。
- 脏缓冲区过多。
- 3 秒超时（最晚 3 秒会执行一次写入）。
- 遇到检查点，即 checkPoint（检查点）。检查点是一个 Oracle 事件，遇到检查点，DBWn 会执行写入。比如，实例有序关闭时会有检查点，DBWn 会将所有脏缓冲区写入到磁盘上去，这很容易理解，要保持数据文件的一致性。

从上述 DBWn 的几个写入时机可以看出，DBWn 的写入不直接依赖于会话的更新操作。不是一有脏缓冲区，它就执行写入。而且，DBWn 执行写入跟 commit 操作也没有任何关系，不要以为 commit 操作的影响结果会实时流入到磁盘中去。

DBWn 采用极懒算法进行写入，原因应该要清楚：频繁的磁盘 IO 对系统的压力很大，如果 DBWn 很积极地去写入磁盘，那对系统性能的影响就太大了，换个角度想，如果 DBWn 很勤快地写磁盘，那么数据库缓冲区存在的意义也就不大了。

当然，说到这里，用户可能会意识到一个问题，DBWn 如此懒地进行数据转储，如果在某一时刻，数据库缓冲区缓存内存在着大量的脏缓冲区（在生产环境中，这是常态），也就是有大量的未 commit 和已 commit 的数据还在内存中，没有持久化到磁盘中，然后突然系统断电了，这种情况下，数据是不是就丢掉了？数据当然不会丢失，这就引出了重做日志（redo log）的概念。接下来，就来谈谈对应重做日志的内存结构和后台进程。

3.3.2 日志缓冲区

当用户执行一些 DML 操作（INSERT、UPDATE、DELETE）时，数据块发生改变，产生的变更向量则会写入到重做日志文件中去。有了这些记录，当系统由于断电等因素突然宕掉，数据库缓冲区缓存内的大量脏数据还没来得及写入到数据文件中去，在重新启动时，会有一个实例恢复的过程，在此过程中就应用了重做日志记录来使数据保持一致；或者数据库遭遇了物理损坏，比如磁盘损坏了，此时可以通过 Oracle 的备份恢复工具（如 RMAN）进行数据恢复，原理就是提取备份集然后应用重做日志文件中的变更记录。

日志缓冲区是一块比较小的内存区域，用来短期存储将写入到磁盘中的重做日志文件中的变更向量的。日志缓冲区存在的意义依然是为了减少磁盘 IO，减少用户的等待时间。试想下，如果每一次用户 DML 操作都要进行等待重做记录被写入到磁盘中去，用户体验会比较差。

日志的写入工作是由日志写入器来负责完成的。顾名思义，日志写入器（LGWR）就是把日志缓冲区内的内容写入到磁盘的重做日志文件中去，相比数据库写入器（DBWn），日志写入器就勤快多了。

以下三种情况 LGWR 会执行写入：

（1）Commit 时写入：因为 DBWn 的写入和 commit 没有任何关系，如果 commit 时数据库没有任何记录，那数据就真的丢失了，Oracle 的重做日志就是为了保证数据安全而存在的，commit 时，会话会先挂起，等待 LGWR 将这些记录写入到磁盘上的重做日志文件中，才会通知用户提交完成。所以，LGWR 在 commit 时执行写入，是为了确保事务永不丢失。

（2）日志缓冲区的占用率达到 1/3。

（3）DBWn 要写入脏缓冲区前，这个写入是为了数据回滚考虑的。DBWn 完全可能写入还没提交的事务（参照上面提到的写入时机），那如何保证事务回滚呢？

首先要知道，DBWn 除了写入实际的数据，还会写入撤销数据。简单说，事务回滚需要撤销数据，在写入撤销数据前，会先写入针对撤销数据的日志记录，若用户要进行事务回滚，就可以应用这些日志记录来构造撤销数据，然后进行回滚。

下面对这两块最重要的内存区域和对应的后台进程做个总结：数据库缓冲区缓存和日志缓冲区都是为了提高性能，避免频繁 IO 而存在的。日志缓冲区相比数据库缓冲区缓存要小得多，并且不能进行自动管理，对于日志缓冲区的修改需要重启实例，数据库缓冲区缓存可进行自动管理。作用在数据库缓冲区缓存上的 DBWn 进程，为了避免频繁的磁盘 IO 导致系统性能下降，会尽可能少地执行写入，且 DBWn 的写入和 commit 操作没有任何关系。作用在日志缓冲区上的 LGWR 进程，则会非常积极地进行写入，一般情况下，它几乎是实时地将重做日志记录转储到磁盘中去。LGWR 是 Oracle 体系结构中最大的瓶颈之一。DML 的速度不可能超过 LGWR 将变更向量写入磁盘的速度。

下面继续讲解其他的内存区域和后台进程。

3.3.3　共享池

共享池是最复杂的 SGA 结构，它有许多子结构，下面来看看常见的几个共享池组件。

（1）库缓存：库缓存这块内存区域会按已分析的格式缓存最近执行的代码。这样，同样的 sql 代码多次执行时，就不用重复地去进行代码分析，可以在很大程度上提高系统性能。

（2）数据字典缓存：存储 Oracle 中的对象定义（表、视图、同义词、索引等数据库对象）。这样在分析 sql 代码时，就不用频繁去磁盘上读取数据字典中的数据了。

（3）PL/SQL 区：缓存存储过程、函数、触发器等数据库对象，这些对象都存储在数据字典中，通过将其缓存到内存中，可以在重复调用时提高性能。

（4）大池：大池是一个可选的内存区域。前面提到专有服务器连接和共享服务器连接，如果数据库采用了共享服务器连接模式，则要使用到大池；RMAN（Oracle 的高级备份恢复工具）备份数据也需要大池。

3.3.4　Java 池和流池

Java 池和流池因为实际的企业环境中使用的比较少，这里不做过多的介绍。Oracle 的很多选项是使用 Java 写的，Java 池用作实例化 Java 对象所需的堆空间。从重做日志中提取变更记录的进程和应用变更记录的进程会用到流池（如实例不正常关闭，譬如断电导致实例关闭，在重启时，Oracle 会自动执行实例恢复过程，在此过程需要提取重做日志记录和应用重做日志两个动作）。

3.4 后台进程

下面本章将介绍有关于 Oracle 数据库的进程相关知识，重点介绍与日常管理工作紧密相关的后台进程。

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，一般会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。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建立的。一个 Oracle 实例可以有许多后台进程，但它们不是一直存在。

数据库实例有内存结构和后台进程。应用与数据库的所有操作和交互都由数据库实例完成，SGA 可以理解为交互平台，后台进程则可以理解为 SGA 与数据库交互的桥梁。PMON、SMON、DBWRn、LGWRn、CKPT 进程为必需的后台进程，ARCHn、LCKn 等为可选后台进程，如图 3-4 所示。

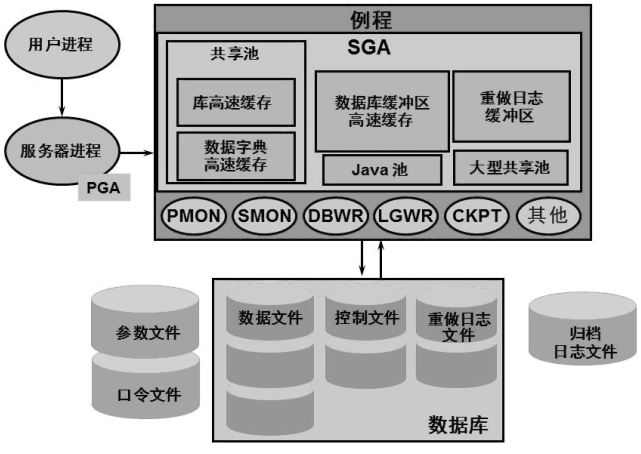

图 3-4　数据库后台进程机构

【示例 3-2】Oracle 数据库各个进程的启动顺序

```
PMON started with pid=2, OS id=18042
```

```
DIAG started with pid=3, OS id=18044
PSP0 started with pid=4, OS id=18051
LMON started with pid=5, OS id=18053
LMD0 started with pid=6, OS id=18055
LMS0 started with pid=7, OS id=18057
MMAN started with pid=8, OS id=18061
DBW0 started with pid=9, OS id=18063
LGWR started with pid=10, OS id=18065
CKPT started with pid=11, OS id=18067
SMON started with pid=12, OS id=18069
RECO started with pid=13, OS id=18071
CJQ0 started with pid=14, OS id=18073
MMON started with pid=15, OS id=18075
MMNL started with pid=16, OS id=18077
```

每个后台进程与 Oracle 数据库的不同部分交互。其中，SMON、PMON、DBWn、CKPT、LGWR 是五个必需的 Oracle 后台进程。

3.4.1　进程监视器

进程监视器（Process Monitor，PMON）主要监视服务器进程。在专有服务器体系模式下，用户进程和服务器进程是一对一的关系，如果某个会话发生异常，PMON 会销毁对应的服务器进程、回滚未提交的事务并回收会话专有的 PGA 内存区域。例如，如果因某些原因专用服务“故障”或被“杀死”，PMON 就负责处理（恢复或回滚工作）和释放资源。PMON 将发出未提交工作的回滚、释放锁和释放分配给故障进程的 SGA 资源。

除了在异常中断之后的清理外，PMON 监控其他 Oracle 后台进程，如果有必要（和有可能）就重新启动。如果共享服务或一个分配器故障（崩溃），PMON 将插手并且重启另一个（在清理故障进程之后）。PMON 将观察所有 Oracle 进程，只要合适或重启它们或中止进程。例如，在数据库日志写进程事件中的 LGWR 故障、实例故障。这是一个严重的错误，最安全的处理方法就是去立即终止实例，让正常的恢复处理数据。

这是很少发生的事情，应该立即报告 Oracle 支持。

PMON 为实例做的另一件事是去使用 Oracle TNS 监听器登记。当一个实例开启时，PMON 进程投出众所周知的端口地址，除非指向其他，来看监听器是否正在开和运行着。众所周知，默认端口是使用 1521。现在，如果监听器在一些不同端口开启会发生什么？这种情况下，机制是相同的，除了监听器地址需要被 LOCAL_LISTENER 参数明确指定。如果监听器运行在库实例开启时，PMON 和监听器通信，传到它相关参数，譬如服务器名和实例的负载度量。如果监听器没被开启，PMON 将周期性地试着和它联系来登记自己。

3.4.2　系统监视器

系统监控后台进程（System Monitor Process，SMON）有时也被叫作 system cleanup process，这么叫的原因是它负责完成很多清理（cleanup）任务。安装和打开数据库，实例恢复是由此

进程完成的。

这个进程对于 Oracle 数据库来说，可以利用一句话来概括，即人小鬼大。其负责的内容并不是很多，但是对于数据的安全与数据库的性能却有很关键的作用。如随着表空间中的数据不断的建立、删除、更新等，在表空间中难免会产生碎片。由于这些碎片的存在，数据库的性能会逐渐降低，而系统监视进程 SMON 正好可以解决这些碎片。SMON 进程会将各个表空间的空闲碎片合并在一起，让数据库系统更加容易分配，从而提高数据库的性能。

另外，在数据库运行的过程中，会因为断电或者其他的原因而发生故障。此时由于数据高速缓存中的脏缓存块还没有来得及写入到数据文件中，从而导致数据的丢失。在数据库启动时，系统监视进程 SMON 会在下一次启动例程时自动读取重做日志文件并对数据库进行恢复。也就是说，进行将已提交的事务写入数据文件（已经写入到日志文件中而没有写入到数据文件中的数据）、回退未提交的事务操作。可见，SMON 进程在 Oracle 数据库中是一个比较小但是却非常重要的角色。

在管理这个进程时，主要需要注意两个问题。

（1）启动的时机。一般情况下，例程重新启动时，会启动这个系统监视进程。然后在这个例程运行期间，这个进程也会被系统定期唤醒，然后会检查是否有工作需要其完成。最重要的是，在有需要时，数据库管理员可以通过其他进程来启动这个 SMON 系统监视进程来完成一些特定的工作。

（2）表空间配置对这个进程的影响。在表空间管理中，有一个参数叫作 PCTINCREASE。如果将这个参数设置为 0 的话，则这个 SMON 系统监视进程对于这个表空间的作用就要打折扣了。在设置为 0 的情况下，SMON 进程就不会对这个表空间中的空闲碎片进行整理、合并操作。也就是说，需要数据库管理员通过数据的导出导入等手工操作才能够解决表空间的碎片问题。显然这会增加数据库管理员的工作量。为此建议，除非有特别的需要，不要将这个参数设置为 0。让 SMON 进程自动对表空间中的碎片进行管理，自动合并表空间中的空闲碎片。不将某个表空间中的这个参数设置为 0 的话，也不会影响到系统监视进程的其他用途，如不会影响到在例程非正常关闭时对数据的恢复操作。即这个参数设置为 0 ，在有需要时仍然可以利用重做日志文件中的记录来恢复相关的数据。

3.4.3 检查点管理进程

检查点管理进程（Checkpoint Process，CKPT）负责发起检查点信号。

【示例 3-3】手动设置检查点

```
SQL>alter system checkpoint;
```

检查点可强制 DBWn 写入脏缓冲区，当数据库崩溃后，由于大量脏缓冲区未写入数据文件，在重新启动时，需要由 SMON 进行实例恢复，实例恢复需要提取和应用重做日志记录，提取的位置就是从上次检查点发起的位置开始的（检查点之前的数据已经被强制写入到数据文件中），这个位置称为 RBA（Redo Byte Address）。CKPT 会不断将这个位置更新到控制文件

中去（以确定实例恢复需要从哪儿开始提取日志记录）。

3.4.4　数据库写进程

上文在讲解 Oracle 内存结构时，已经就数据库进程（Database Writer，DBWn）的作用做了介绍。该进程负责将脏数据块写入磁盘。它是一个非常重要的进程，随着内存的不断增加，一个 DBWn 进程可能不够用了。所以从 Oracle 8i 起就可以为系统配置多个 DBWn 进程。初始化参数 db_writer_processe 决定了启动多少个 DBWn 进程。每个 DBWn 进程都会分配一个 cache buffers lru chain latch。

DBWn 作为一个后台进程，只有在某些条件满足了才会触发。这些条件包括：

- 当进程在 LRU 链表扫描以查找可以覆盖的 buffer header 时，如果已经扫描的 buffer header 的数量到达一定的限度时，触发 DBWn 进程。
- 如果脏数据块的总数超过一定限度，也将触发 DBWn 进程。
- 发生检查点（包括增量检查点（Incremental Checkpoint）和完全检查点（Complete Checkpoint））时触发 DBWn。
- 每隔三秒钟启动一次 DBWn。

3.4.5　日志写进程

日志写进程（Log Writer，LGWR）也是一种后台进程，主要负责将日志缓冲内容写到磁盘的在线重做日志文件或组中。DBWn 将 dirty 块写到磁盘之前，所有与 buffer 修改相关的 redo log 都需要由 LGWR 写入磁盘的在线重做日志文件（组）。如果未写完，那么 DBWn 会等待 LGWR，也会产生一些相应的等待事件（例如，log file prarllel write，后面单独作为话题再聊）。总之，这样做的目的就是为了当 crash 时，可以有恢复之前操作的可能，也是 Oracle 在保持交易完整性方面的一个机制。

该进程有如下几方面的特点：

（1）LGWR 写日志是顺序写，这就解释了一个 Orace Server 只能有一个 LGWR 进程，不能像 DBWR 那样可以有多个，否则就无法保证顺序写的机制，而且可能会产生锁的问题。

（2）用户进程每次修改内存数据块时，都会在日志缓冲区（redo buffer）中构造一个相应的重做条目（redo entry），它记录了被修改数据块修改之前和之后的值。

（3）LGWR 将 redo entry 写入联机日志文件的情况可以概括为两种：后台写和同步写，或者说异步写和同步写。

① 后台写的条件如下：

- 每 3 秒 LGWR 启动一次。
- DBWR 启动时如果发现 dirty 块对应的 redo entry 还没写入联机日志文件，则 DBWR 触发 LGWR 进程并等待 LGWR 完成后继续。
- redo entry 数量达到整个 log buffer 的 1/3 时，触发 LGWR。

- redo entry 的数量达到 1MB。

② 同步写的条件是：执行 commit 时，必须等待 log buffer 进行 flushing 操作（可能产生 log file sync 等待事件），写入磁盘中的联机日志文件。一般上述 1/3 满的条件触发 LGWR，几乎强制 LGWR 实时写，因此当需要执行 commit 时，可能没有任何 redo entry 需要写入了。

（4）3 秒触发 LGWR 的规则，事实上，这个超时是 DBWR 的，但是因为 LGWR 总在 DBWR 调用之前执行，因此效果上也相当于 LGWR 的超时是 3 秒即调用。

3.4.6 管理监控进程

管理监控进程（Manageability Monitor，MMON）是数据库的自我监视和自我调整的支持进程。实例在运行中，会收集大量有关实例活动和性能的统计数据，这些数据会收集到 SGA 中，MMON 定期从 SGA 中捕获这些统计数据，并将其写入到数据字典中，便于后续对这些快照进行分析。（默认情况下，MMON 每隔一个小时收集一次快照）。

3.4.7 归档进程

归档进程（Archiver，ARCn）是可选的。在重做日志文件管理中，有归档与非归档两种模式。如果数据库配置为归档模式，这个进程就是必需的。所谓归档，就是将重做日志文件永久保存（生产库一般都会配置为归档模式）到归档日志文件中。归档日志文件和重做日志文件的作用是一样的，只不过重做日志文件会不断被重写，而归档日志文件则保留了关于数据更改的完整的历史记录。

在日志进行切换时，如果不对原先的日志文件进行归档，而直接覆盖的话，就叫作非归档模式。相反，在写入下一个日志文件时，会先对目标日志文件进行归档，这就叫作归档模式。归档进程 ARCH 就是负责在重做日志文件切换后将已经写满的重做日志文件复制到归档日志文件中，以防止循环写入重做日志文件时将其覆盖。

所以说，只有数据库运行在归档模式时，这个 ARCH 进程才会被启动。在任何一种操作模式下，重做日志文件都会被循环使用。所以当 LGWR 进程在进行日志切换，需要用到下一个日志文件时，数据库会被暂时挂起，进行目标日志文件的归档工作。直到这个目标重做日志文件归档完毕后，数据库才会恢复正常。所以说，归档日志的操作有时候也会影响数据库的性能，特别是当需要进行频繁的大批量数据更改时。

那么有什么方法可以提高归档作业的效率呢？如下一些建议可供数据库管理员参考。

（1）增加归档进程的个数。

在默认情况下，一个例程只会启动一个归档进程 ARCH。当 ARCH 进程正在归档一个重做日志文件时，任何其他的进程都不能够访问这个重做日志文件。在 Oracle 数据库中，可以根据需要启动多个归档进程 ARCH。在 Oracle 数据库中，启动多个归档进程时分为手工与自动两个方式。为了提高重做日志文件归档的速度，当用户进程发生比较长时间的等待时，LGWR 进程会根据时机情况来自动启动多个归档进程。在 Oracle 数据库中，其最多可以启动十个归

档进程。另外，如果数据库管理员在部署数据库时，估计日志归档作业会影响到数据库的性能，就可以手工来启动多个归档进程。这是通过初始化参数 LOG_ARCHIVE_MAX_PROCESSES 来确定的。可以将这个参数设置为大于 1 的数值（注意不能够超过 9 个归档进程）。如此的话，数据库在创建例程时就会启动多个归档进程。不过还是倾向于让数据库系统来自动管理这个进程。数据库管理员最好不要干涉。

另外，需要注意 ARCH 归档进程个数与 DBWR 进程个数的区别。默认情况下，DBWR 进程也只有一个。为了提高数据库的性能，可以根据情况增加 DBWR 进程的个数。不过其增加时受到 CPU 数量的限制，即一个 DBWR 进程需要使用一个独立的 CPU。如果想启动三个 DBWR 进程的话，就必须采用 3 个 CPU 处理器。对于 ARCH 归档进程来说，则没有这个限制。即使只有一个 CPU 处理器，其也可以启动三个甚至更多的 ARCH 进程。

（2）增加重做日志文件来延长归档日志进程启动的时间间隔。

通常情况下，只有当前一个重做日志文件写满、需要进行日志切换时，才会触发这个 ARCH 归档日志进程。所以如果重做文件比较大，其日志切换的时间间隔就会延长，ARCH 归档日志进程的启动时间间隔也会比较长。所以说，通过调整重做日志文件的大小，可以延长归档进程启动的时间间隔，从而降低因为归档进程启动而对数据库性能造成的负面影响。

（3）在数据库初始化的过程中，可能需要导入大量的数据。

此时会对数据库中的数据进行大量的插入、删除、更新等操作，从而导致重做日志文件切换频繁。这就会导致数据库需要频繁启动 ARCH 归档进程。

数据库大量的更新操作、重做日志文件（LGWR 进程）、归档重做日志文件（ARCH）进程之间就形成了一条无形的链条。由于“蝴蝶效应”，从而降低了数据库的性能。为此在必要时，需要砍断这根链条，以提高数据库的性能。例如，可以在数据大量导入、更新、删除时，不往日志文件中插入记录，或者临时增加重做日志文件的空间。如此的话，在进行这些操作时就可以避免进行重做日志切换或者延长重做日志切换的时间间隔，从而使 ARCH 归档日志进程也可以避免或者延长其时间间隔，从而提高数据库的性能。当数据库初始化完成之后，再将其恢复过来。这些临时性的调整虽然比较麻烦，但是可以提高数据库的性能。为此认为这是值得的。

Oracle 是如何保存数据的呢？众所周知，内存的数据处理速度是比物理磁盘快很多的，Oracle 数据库的所有数据更改都在内存中完成，当然，在某些情况下，某些读取类的操作是不会经过 SGA 内存区域的，而是选择直接从硬盘将数据获取并返回给用户。在实际的数据库管理工作中，管理员更多的工作是与内存中的数据相关。数据库是存放数据的容器，那么数据是如何放进这个容器中的呢？内存数据块写入数据文件其实是一个相当复杂的过程，在这个过程中，首先要保证安全。所谓安全，就是在写的过程中，一旦发生实例崩溃，要有一套完整的机制能够保证用户已经提交的数据不会丢失；其次，在保证安全的基础上，要尽可能地提高效率。众所周知，I/O 操作是最昂贵的操作，所以应该尽可能地将脏数据块收集到一定程度以后，再批量写入磁盘中。

直观上最简单的解决方法就是，每当用户提交时就将所改变的内存数据块交给 DBWn，由其写入数据文件。这样的话，一定能够保证提交的数据不会丢失，但是这种方式效率最为低

下，在高并发环境中，一定会引起 I/O 方面的争用。Oracle 当然不会采用这种没有伸缩性的方式。Oracle 引入了 CKPT 和 LGWR 这两个后台进程，这两个进程与 DBWn 进程互相合作，提供了既安全又高效的写脏数据块的解决方法。

用户进程每次修改内存数据块时，都会在日志缓冲区（log buffer）中构造一个相应的重做条目（redo entry），该重做条目描述了被修改的数据块在修改之前和修改之后的值，而 LGWR 进程则负责将这些重做条目写入联机日志文件。只要重做条目进入了联机日志文件，那么数据的安全就有保障了，否则这些数据都是有安全隐患的。LGWR 是一个必须和前台用户进程通信的进程。LGWR 承担了维护系统数据完整性的任务，它保证了数据在任何情况下都不会丢失。

假如 DBWR 在写脏数据块的过程中突然发生实例崩溃该怎么办呢？用户提交时，Oracle 是不一定会把提交的数据块写入数据文件的。那么实例崩溃时，必然会有一些已经提交但是还没有被写入数据文件的内存数据块丢失了。当实例再次启动时，Oracle 需要利用日志文件中记录的重做条目在 buffer cache 中重新构造出被丢失的数据块，从而完成前滚和回滚的工作，并将丢失的数据块找回来。于是这里就存在一个问题，就是 Oracle 在日志文件中找重做条目时，到底应该找哪些重做条目？换句话说，应该在日志文件中从哪个起点开始往后应用重做条目？注意，这里所指的日志文件可能不止一个日志文件。

因为需要预防随时可能的实例崩溃现象，所以 Oracle 在数据库的正常运行过程中会不断地定位这个起点，以便在不可预期的实例崩溃中能够最有效地保护并恢复数据。同时，这个起点的选择非常有讲究。首先，这个起点不能太靠近日志文件的头部，太靠近日志文件头部意味着要处理很多的重做条目，这样会导致实例再次启动时所进行恢复的时间太长；其次，这个起点也不能太靠近日志文件的尾部，太靠近日志文件的尾部说明只有很少的脏数据块没有被写入数据文件，也就是说前面已经有很多脏数据块被写入了数据文件，那也就意味着只有在 DBWn 进程很频繁地写数据文件的情况下，才能使得 buffer cache 中所残留的脏数据块的数量很少。但很明显，DBWn 写得越频繁，所占用写数据文件的 I/O 就越严重，那么留给其他操作（比如读取 buffer cache 中不存在的数据块等）的 I/O 资源就越少。这显然也是不合理的。

从这里也可以看出，这个起点实际上说明了在日志文件中位于这个起点之前的重做条目所对应的在 buffer cache 中的脏数据块已经被写入了数据文件，从而在实例崩溃以后的恢复中不需要去考虑，而这个起点以后的重做条目所对应的脏数据块实际还没有被写入数据文件。在实例崩溃以后的恢复中，需要从这个起点开始往后依次取出日志文件中的重做条目进行恢复。考虑到目前的内存容量越来越大，buffer cache 也越来越大，buffer cache 中包含几百万个内存数据块也是很正常的现象，如何才能最有效地来定位这个起点呢？

为了能够确定这个最佳的起点，Oracle 引入了名为 CKPT 的后台进程，通常也叫作检查点进程（Checkpoint Process）。这个进程与 DBWn 共同合作，从而确定这个起点。同时，这个起点也有一个专门的名字，叫作检查点位置（Checkpoint Position，该检查点位置记录在控制文件里）。

Oracle 为了在检查点的算法上更加具有可扩展性（也就是为了能够在巨大的 buffer cache 下依然有效工作），引入了检查点队列（Checkpoint Queue），该队列上串起来的都是脏数据块所对应的 buffer header。每次 DBWn 写脏数据块时，也是从检查点队列上扫描脏数据块，并将这些脏数据块实际写入数据文件的。当写完以后，DBWn 会将这些已经写入数据文件的脏

数据块从检查点队列上摘下来。

这样即便是在巨大的 buffer cache 下工作，CKPT 也能够快速地确定哪些脏数据块已经被写入了数据文件，而哪些还没有写入数据文件。显然，只要在检查点队列上的数据块就都是还没有写入数据文件的脏数据块。而且，为了更加有效地处理单实例和多实例（RAC）环境下的表空间的检查点处理，比如将表空间设置为离线状态或者为热备份状态等，Oracle 还专门引入了文件队列（File Queue）。文件队列的原理与检查点队列是一样的，只不过每个数据文件会有一个文件队列，该数据文件所对应的脏数据块会被串在同一个文件队列上；同时为了能够尽量减少实例崩溃后恢复的时间，Oracle 还引入了增量检查点（Incremental Checkpoint），从而增加了检查点启动的次数。

如果每次检查点启动的间隔时间过长的话，再加上内存很大，可能会使得恢复的时间过长。因为前一次检查点启动以后，标识出了这个起点。然后在第二次检查点启动之前，DBWn 可能已经将很多脏数据块写入了数据文件，而假如在第二次检查点启动之前发生实例崩溃，导致在日志文件中所标识的起点仍然是上一次检查点启动时所标识的，导致 Oracle 不知道这个起点以后的很多重做条目所对应的脏数据块实际上已经写入了数据文件，从而使得 Oracle 在实例恢复时重复地处理一遍，效率低下，浪费时间。

上面说到了有关 CKPT 的两个重要概念：检查点队列（包括文件队列）和增量检查点。检查点队列上的 buffer header 是按照数据块第一次被修改的时间先后顺序来排列的。越早修改的数据块的 buffer header 排在越前面，同时如果一个数据块被修改了多次的话，在该链表上也只出现一次。而且，检查点队列上的 buffer header 还记录了脏数据块在第一次被修改时所对应的重做条目在重做日志文件中的地址，也就是 LRBA（Low Redo Block Address）。Low 表示第一次修改时对应的 RBA。每个检查点都会由 checkpoint queue latch 来保护。

而增量检查点是从 Oracle 8i 开始出现的，是相对于 Oracle 8i 之前的完全检查点（Complete Checkpoint）而言的。完全检查点启动时，会标识出 buffer cache 中所有的脏数据块，然后以最高优先级启动 DBWn 进程将这些脏数据块写入数据文件。Oracle 8i 之前，日志切换时会触发完全检查点。到了 Oracle 8i 及以后，完全检查点只有在两种情况下才会被触发：

- 发出 alter system checkpoint 命令。
- 除了 shutdown abort 以外的正常关闭数据库。

日志切换不会触发完全检查点，而是触发增量检查点。自 Oracle 8i 所引入的增量检查点每隔三秒钟或发生日志切换时启动。它启动时只做一件事情：找出当前检查点队列上的第一个 buffer header，并将该 buffer header 中所记录的 LRBA（这个 LRBA 也就是 checkpoint position）记录到控制文件中去。如果是由日志切换所引起的增量检查点，则还会将 checkpoint position 记录到每个数据文件头中。也就是说，如果这个时候发生实例崩溃，Oracle 在下次启动时就会到控制文件中找到这个 checkpoint position 作为日志文件的起点，然后从这个起点开始向后依次取出每个重做条目进行处理。

上面所描述的概念，用一句话来概括，其实就是 DBWn 负责写检查点队列上的脏数据块，而 CKPT 负责记录当前检查点队列的第一个数据块所对应的重做条目在日志文件中的地址，而到底应该写哪些脏数据块、写多少脏数据块则要到检查点队列上才能确定。

3.5 物理结构

Oracle 物理结构包含了数据文件、重做日志文件、控制文件、参数文件、密码文件、归档日志文件、备份文件、告警日志文件、跟踪文件等。其中，数据文件、控制文件、重做日志文件和参数文件是必需的，其他文件可选。

3.5.1 数据文件

每一个 Oracle 数据库都有一个或多个物理的数据文件（Data File）。数据文件包含全部数据库数据，逻辑数据库结构（如表、索引、视图、函数）的数据物理地存储在数据库的数据文件中。数据文件中的数据在需要时可以读取并存储在 Oracle 内存储区中。例如，用户要存取数据库一表的某些数据，若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，当修改或插入新数据时，为了减少磁盘输出的总数、提高性能，不必立刻写入数据文件，数据存储在内存，然后由 Oracle 后台进程 DBWRn 决定如何将其写入到相应的数据文件。

数据文件有下列特征：

一个数据文件仅与一个数据库联系。

一个表空间（数据库存储的逻辑单位）由一个或多个数据文件组成。

3.5.2 日志文件

每一个数据库实例有两组或以上日志文件组（Redo Log Files），为了防止日志文件本身的故障，每个日志文件组可以有一个或以上日志成员。日志的主要功能是记录对数据所做的修改。在出现故障时，如果不能将修改数据永久地写入数据文件，则可利用日志得到该修改，从而保证数据不丢失。

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。对于任何丢失的数据，在下一次数据库打开时，Oracle 都会自动地应用日志文件中的信息来恢复数据库数据文件。Oracle 日志文件有联机日志文件和归档日志文件两种。联机日志文件用来循环记录数据库改变的操作系统文件。归档日志文件是为避免联机日志文件重写时丢失重复数据而对联机日志文件所做的备份。Oracle 数据库可以选择归档（ARCHIVELOG）或非归档（NOARCHIVELOG）模式。

3.5.3　控制文件

每一个 Oracle 数据库都有一个控制文件（Control File）或同一个控制文件的多个备份，它记录数据库的物理结构信息，包括数据库名、数据库数据文件和日志文件的名字和位置、数据库建立日期等。由于控制文件记录数据库的物理结构信息，对数据库运行至关重要，为了安全起见，Oracle 建议保存两份以上的控制文件镜像于不同的存储设备。

当 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。当然，在数据恢复时，自然会使用控制文件以确定数据库物理文件的名字和位置。

3.5.4　参数文件

除了构成Oracle数据库物理结构的三类主要文件外，参数文件（Parameter Files）也是Oracle数据库较为重要的一种文件结构。参数文件记录了 Oracle 数据库的基本参数信息，主要包括数据库名、控制文件所在路径、进程等。在 Oracle 9i 之前，都只有 pfile 一种文本格式的参数文件。在 9i 之后，新增了服务器二进制参数文件 SPFILE。通过修改 pfile 以修改数据库参数，必须要求重启数据库后才能生效。通过修改 SPFILE 以修改数据库参数时，根据参数类型分为静态参数需要重启和动态参数无须重启立即生效，可以通过查询 v$parameter 视图确定参数类型。有多种参数文件类型存在，而 Oracle 的正常运行只使用一种参数文件，在 Oracle 启动过程中加载文件的顺序为 spfilesid.ora> spfile.ora> initsid.ora。

3.5.5　其他文件

Oracle 数据的运行除了以上重要的必需文件以外，还有虽然非必需但一样重要的其他文件（Other Files）结构，比如密码文件、归档日志文件、alter 告警日志文件、trace 跟踪文件等。

（1）密码文件的作用是主要进行 DBA 权限的身份认证，用于记录数据库账户管理员的密码，每个数据库必须要拥有密码文件。

（2）归档日志文件只有在数据库开启了归档模式时才会产生，作用是 redo log 的归档备份。

（3）alter 告警日志文件主要记录数据库行为，其中比较重要的信息是里面记录的告警或报错信息，这些对于数据库日常的优化和故障诊断是非常有帮助的。

（4）trace 跟踪文件比 alter 告警日志文件内容更加详细，有一些数据库报错信息会直接写入到 trace 文件中，在 alter 日志中只显示 trace 文件名，供 DBA 深入分析问题。

3.6　逻辑结构

Oracle 数据库的逻辑结构是一种层次结构，主要由表空间、段、区和数据块等概念组成。

逻辑结构是面向用户的，即用户利用 Oracle 开发应用程序使用的就是逻辑结构。数据库存储层次结构及其构成关系，结构对象也从数据块到表空间形成了不同层次的粒度关系，如图 3-5 所示。

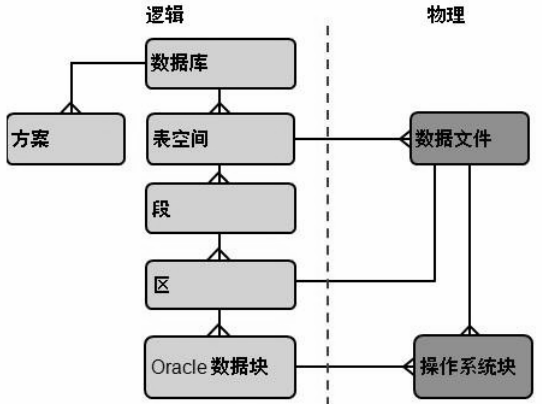

图 3-5　数据库逻辑结构

3.6.1　表空间

表空间（tablespace）是数据库的逻辑划分，任何数据库逻辑对象在存储时都必须存储在某个表空间中。从图 3-5 可以知道每个表空间由多个段组成。表空间一般是由一个或多个数据文件构成的，每个数据文件只能属于某一个表空间，也就是说表空间和数据文件是 1 对 N 的关系。每个数据库至少有一个表空间（system tablespace），表空间大小等于从属于它的所有数据文件大小的总和。在 Oracle 10g 中初始创建的只有 5 个表空间，即 system、temp、undotbs1、sysaux、users。

1. 系统表空间（system tablespace）

system 表空间是每个 Oracle 数据库都必须具备的，数据库创建时自动创建，用于存储数据库系统对象、数据字典、存储过程、触发器和系统回滚段及数据库管理所需的信息；系统表空间的名称是不可更改的，系统表空间必须在任何时候都可以用，也是数据库运行的必要条件。因此，系统表空间是不能脱机的。为避免系统表空间产生存储碎片以及争用系统资源的问题，建议创建独立的用户表空间来单独存储用户对象及数据。

2. 临时表空间（temp tablespace）

temp 表空间相对于其他表空间而言，临时表空间主要用于存储 Oracle 数据库运行期间所产生的临时数据，比如 SQL 排序等。数据库可以建立多个临时表空间。当数据库关闭后，临时表空间中所有数据将全部被清除。

3. 回滚表空间（undo tablespace）

回滚表空间是 Oracle 特有的概念，用于保存 Oracle 数据库变化前的记录，在对数据库中

的记录进行 DML 操作时，Oracle 数据库会将变化前的记录副本保存到回滚表空间中，在 rollback、实例恢复（前滚）、一致性读 CR 块的构造时会使用到 undo 信息，同时保证事务读一致性。在 Oracle 8i 中是 rollback tablespace，从 Oracle 9i 开始改为 undo tablespace。其中，temp 是临时表空间，undotbs1 是 undo 回滚表空间。

4. sysaux 表空间

sysaux 表空间是随着数据库的创建而创建的，它充当 system 的辅助表空间，主要存储除数据字典以外的其他对象，如果启用 EM 或 Gird Control，该表空间就用于存放 EM 采集的监控信息。

5. users 表空间

在创建数据库时自动创建的用户（users）表空间，一般用于维护账户使用的表空间。应用程序表空间一般另外根据应用需求创建。

3.6.2　段

段（Segment）是由多个数据区（Extent）构成的，是为特定的数据库对象（如数据段、索引段、回滚段、临时段）分配的一系列数据区。段内包含的数据区可以不连续，而且可以跨越多个数据文件，使用段的目的是用来保存特定对象。Oracle 数据库分为数据段、索引段、回滚段和临时段 4 种类型。

- 数据段：也称为表段，所包含的数据与表和簇相关。当创建一个表时，系统自动创建一个以该表的名字命名的数据段。
- 索引段：包含索引相关信息，创建索引时，系统自动创建一个以该索引的名字命名的索引段。
- 回滚段：包含回滚信息。DML 操作时，Oracle 数据库会将变化前的记录副本保存到回滚段中，在 rollback、实例恢复（前滚）、一致性读 CR 块的构造时会使用到回滚段信息，同时用于保证事务读一致性。创建数据库时，Oracle 会创建默认的回滚段，其管理方式既可以是自动的，也可以是手工的。
- 临时段：是 Oracle 在运行过程中自行创建的段，当一个 SQL 语句需要临时工作区（比如排序）时，由 Oracle 创建临时段，一旦语句执行完毕，临时段就会自动释放。

3.6.3　数据区

数据区（Extent）是一组连续的数据块。当一个表、回滚段或临时段创建或需要附加空间时，系统总是为之分配一个新的数据区。一个数据区不能跨越多个文件，因为它包含连续的数据块。使用区的目的是用来保存特定数据类型的数据，也是表中数据增长的基本单位。在 Oracle 数据库中，分配空间就是以数据区为单位的。一个 Oracle 对象包含至少一个数据区，设置一个表或索引的存储参数包含设置它的数据区大小。

3.6.4 数据块

数据块（Data Blocks）是 Oracle 最小的存储单位。Oracle 数据存放在“块”中。Oracle 每次请求数据时，都是以块为单位，也就是说，Oracle 每次请求的数据是块的整数倍。如果 Oracle 请求的数据量不到一块，也会读取整个块，因为“块”是 Oracle 读写数据的最小单位或者说最基本的单位。

特别需要注意的是，这里的“块”是 Oracle 的“数据块”，不是操作系统的“块”。Oracle 块的标准大小由初始化参数 DB_BLOCK_SIZE 指定，默认标准块大小为 8KB。具有标准大小的块称为标准块（Standard Block），和标准块的大小不同的块叫非标准块（Nonstandard Block）。同一个数据库实例可以同时存在多种不同的块大小，由初始化参数 DB_BLOCK_SIZE 指定一个标准块大小。操作系统每次执行 I/O 时，都是以操作系统的块为单位的。Oracle 每次执行 I/O 时，都是以 Oracle 的块为单位。Oracle 数据块大小一般是操作系统块的整数倍。

3.7 小结

在学习 Oracle 数据库的过程中，体系结构是重中之重，一开始要从宏观上掌握它的物理结构组成、文件组成和内存组成。掌握得越深入越好，在实际工作中，遇到疑难问题时，往往可以归结到体系结构中来解释。体系结构是对一个系统的框架描述，是设计系统的一个宏观工作。自从 Oracle 推出 ASM 功能模块之后，在学习数据库体系结构时，都会增加 ASM 体系结构的学习，这部分的内容将在下一章节做详细的介绍。ASM 是目前 Oracle 广泛使用的存储方式，也是学习 Oracle 数据库的一部分重要内容。