数据库的原理与设计

第1章　数据库基本知识

1.1　信息、数据与数据处理

1.1.1　数据与信息

数据的概念包括两个方面，即数据内容和数据形式。数据内容是指所描述客观事物的具体特性，也就是通常所说的数据的“值”；数据形式则是指数据内容存储在媒体上的具体形式，也就是通常所说的数据的“类型”。数据主要有数字、文字、声音、图形和图像等多种形式。

信息是指数据经过加工处理后所获取的有用知识。信息是以某种数据形式表现的。

数据和信息是两个相互联系、但又相互区别的概念；数据是信息的具体表现形式，信息是数据有意义的表现。

1.1.2　数据处理

数据处理就是将数据转换为信息的过程。数据处理的内容主要包括：数据的收集、整理、存储、加工、分类、维护、排序、检索和传输等一系列活动的总和。数据处理的目的是从大量的数据中，根据数据自身的规律和及其相互联系，通过分析、归纳、推理等科学方法，利用计算机技术、数据库技术等技术手段，提取有效的信息资源，为进一步分析、管理和决策提供依据。数据处理也称信息处理。

1.1.3　数据处理的发展

1．人工管理阶段

早期的计算机主要用于科学计算，计算处理的数据量很小，基本上不存在数据管理的问题。从 50 年代初，开始将计算机应用于数据处理。

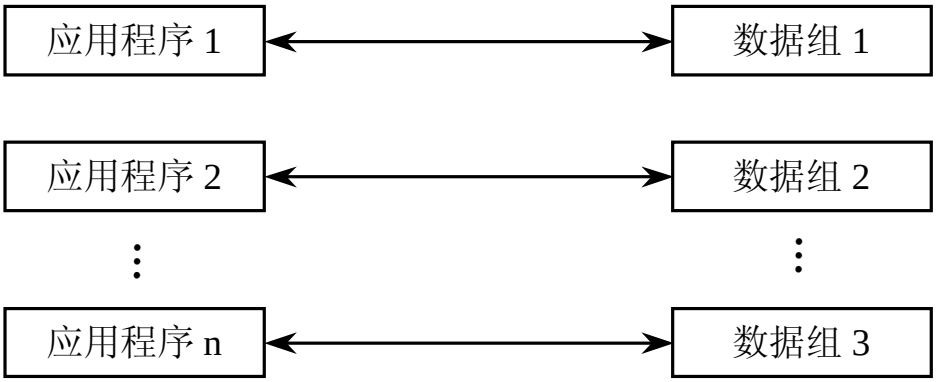


图1-1 人工管理阶段程序与数据的关系

2．文件管理阶段

从 50 年代后期开始至 60 年代末为文件管理阶段，应用程序通过专门管理数据的软件即文件系统管理来使用数据。

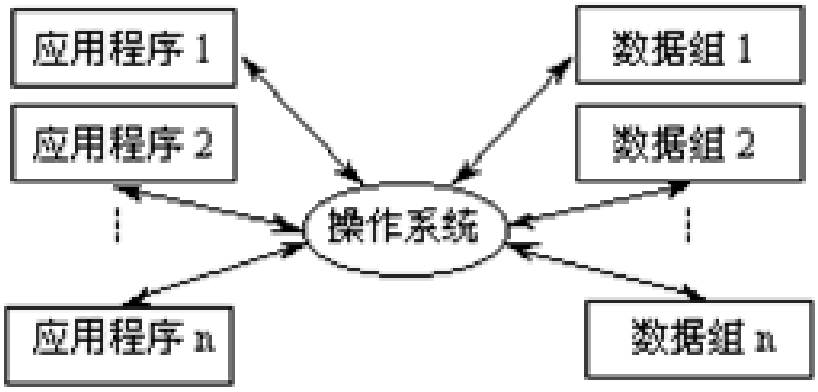


图1-2 文件管理阶段程序与数据的关系

3．数据库管理阶段

数据库管理阶段是 60 年代末在文件管理基础上发展起来的。

图1-3 应用程序与数据之间的关系

1.1.4 数据库技术的发展

（1）1969 年 IBM 公司研制、开发了数据库管理系统商品化软件 IMS（Information Management System），IMS 的数据模型是层次结构的。

（2）美国数据系统语言协会 CODASYL（Conference On Data System Language）下属的数据库任务组 DBTG（Data Base Task Group）对数据库方法进行系统的讨论、研究，提出了若干报告，成为 OBTG 报告。OBTG 报告确定并且建立了数据库系统的许多概念、方法和技术。

（3）1970 年 IBM 公司 San Jose 研究实验室的研究员 E.F.Codd 发表了著名的“大型共享系统的关系数据库的关系模型”论文，为关系数据库技术奠定了理论基础。

自 20 世纪 70 年代开始，数据库技术有了很大的发展，表现为：

（1）数据库方法，特别是 OBTG 方法和思想应用于各种计算机系统，出现了许多商品化数据库系统，它们大都是基于网状模型和层次模型的。

（2）这些商用系统的运行，使数据库技术日益广泛地应用到企业管理、事务处理、交通运输、信息检索、军事指挥、政府管理和辅助决策等各个方面，深入到生产、生活的各个领域。数据库技术成为实现和优化信息系统的基本技术。

（3）关系方法的理论研究和软件系统的研制取得了很大的成果。

1.1.5 数据库新技术

1．分布式数据库

分布式数据库系统（Distributed DataBase System，DDBS）是在集中式数据库基础上发展起来的，是数据库技术与计算机网络技术、分布处理技术相结合的产物。分布式数据库系统的主

（1）数据是分布的。

（2）数据是逻辑相关的。

（3）结点的自治性。

2．面向对象数据库

面向对象数据库系统（Object-Oriented DataBase System，OODBS）是将面向对象的模型、方法和机制，与先进的数据库技术有机地结合而形成的新型数据库系统。它从关系模型中脱离出来，强调在数据库框架中的发展类型、数据抽象、继承和持久性；它的基本设计思想是，一方面把面向对象语言向数据库方向扩展，使应用程序能够存取并处理对象；另一方面扩展数据库系统，使其具有面向对象的特征，提供一种综合的语义数据建模概念集，以便对现实世界中复杂应用的实体和联系建模。

3．多媒体数据库

多媒体数据库系统（Multi-Media DataBase System，MDBS）是数据库技术与多媒体技术相结合的产物。

（1）数据量大。

（2）结构复杂。

（3）时序性。

（4）数据传输的连续性。

从实际应用的角度考虑，多媒体数据库管理系统（MDBMS）应具有如下基本功能：

（1）应能够有效地表示多种媒体数据，对不同媒体的数据，如文本、图形、图像、声音等能够按应用的不同，采用不同的表示方法。

（2）应能够处理各种媒体数据，正确识别和表现各种媒体数据的特征、各种媒体间的空间或时间的关联。

（3）应能够像其他格式化数据一样对多媒体数据进行操作。

（4）应具有开放功能，提供多媒体数据库的应用程序接口等。

4．数据仓库

数据仓库可以提供对企业数据方便访问和具有强大分析能力的工具，从企业数据中获得有价值的信息，发掘企业的竞争优势，提高企业的运营效率和指导企业决策。数据仓库作为决策

库技术、联机分析处理（On-Line Analysis Processing，OLAP）技术和数据挖掘（Data Mining，DM）技术。

1.2 数据库系统

1.2.1 数据库系统的组成

1．计算机硬件

2．数据库管理系统

3．数据库

4．应用程序

5．数据库用户

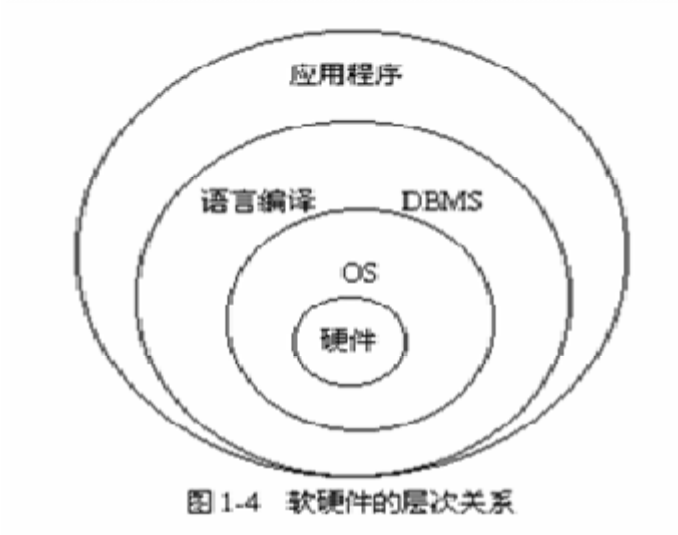


图 1-4 软硬件的层次关系

1.2.2 数据库系统体系结构

1．模式

模式又称概念模式或逻辑模式，对应于概念级。它是由数据库设计者综合所有用户的数据，按照统一的观点构造的全局逻辑结构。

2．外模式

外模式又称子模式，对应于用户级。它是某个或某几个用户所看到的数据库的数据视图，是与某一应用有关的数据的逻辑表示。

内模式又称存储模式，对应于物理级。它是数据库中全体数据的内部表示或底层描述，是数据库最低一级的逻辑描述，它描述了数据在存储介质上的存储方式和物理结构，对应着实际存储在外存储介质上的数据库。

4．数据库系统的二级映射

数据库系统的三级模式是数据在三个级别（层次）上的抽象，使用户能够逻辑地、抽象地处理数据而不必关心数据在计算机中的物理表示和存储。

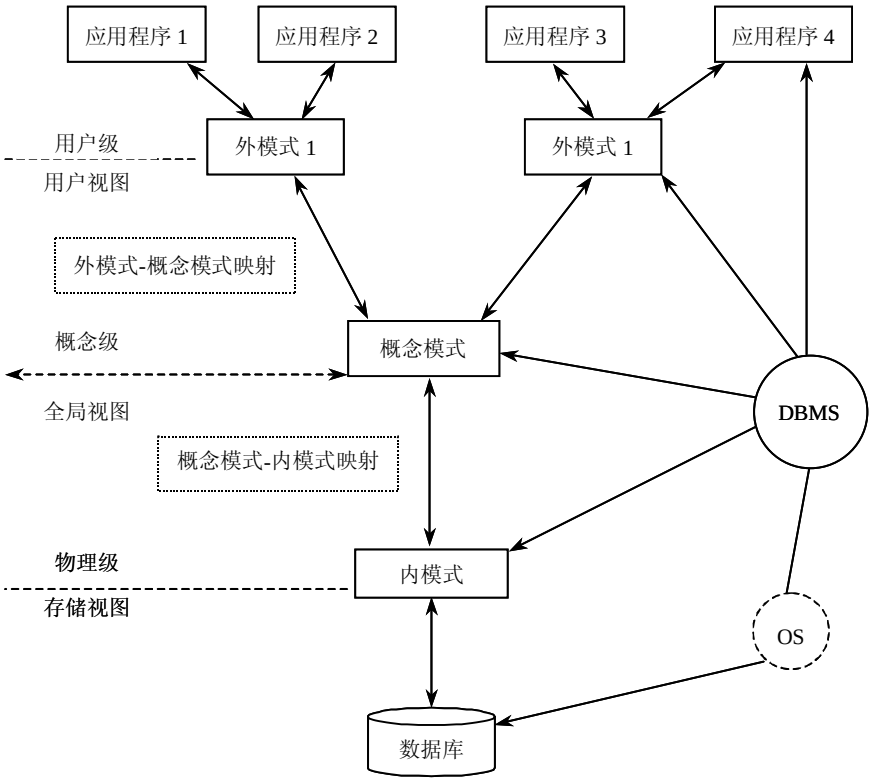


图1-5 数据库系统的体系结构

1.2.3 数据库管理系统的功能

1．数据库定义（描述）功能

2．数据库操纵功能

3．数据库运行管理功能

4．数据组织、存储和管理

5．数据库的建立和维护

1.2.4 数据库管理系统的组成

1．数据定义语言及其编译处理程序

2．数据操作语言及其编译程序

3．数据库运行控制程序

4．实用程序

1.2.5 数据库系统的特点

1．数据共享

2．减少数据冗余

3．具有较高的数据独立性

（1）物理数据独立：

（2）逻辑数据独立：

4．增强了数据安全性和完整性保护

1.3 数据模型

1.3.1 现实世界的数据描述

1.3.2 数据模型

1.3.3 关系的基本概念及其特点

1.3.1 现实世界的数据描述

1．信息处理的三个层次

（1）现实世界。

（2）信息世界。

（3）数据世界。

2．信息世界中的基本概念

（1）实体。

（2）属性。

（3）域。

（5）实体集。

（6）实体联系。

3．实体模型

实体模型又称概念模型，它是反映实体之间联系的模型。数据库设计的重要任务就是建立实体模型，建立概念数据库的具体描述。在建立实体模型时，实体要逐一命名以示区别，并描述它们之间的各种联系。实体模型只是将现实世界的客观对象抽象为某种信息结构，这种信息结构并不依赖于具体的计算机系统，E-R 图是目前常用的概念模型的表示方法。

1.3.2 数据模型

1．层次模型（Hierarchical Model）

用树形结构表示实体和实体间联系的数据模型称为层次模型。层次模型的基本特点：

（1）有且仅有一个结点无父结点，称其为根结点。

（2）其他结点有且只一个父结点。

2．网状模型（Network Model）

用网状结构表示实体和实体之间关系的数据模型称为网状模型。网状模型的基本特点：

（1）一个以上结点无父结点。

（2）至少有一结点有多于一个的父结点。

3．关系模型（Relational Model）

用二维表来表示实体和实体间联系的数据模型称为关系模型。例如，在关系模型中可用如表 1-1 的形式表示学生对象。关系不但可以表示实体间一对多的联系，也可以方便地表示多对多的联系。

表1-1 学生基本情况表

学号	姓名	性别	班级名	系别代号	地址	出生日期	是否团员	备注
011110	李建国	男	计0121	01	湖北武汉	1984-9-28	是	
011103	李宁	女	电0134	02	江西九江	1985-5-6	否	
011202	赵娜	女	英0112	03	广西南宁	1984-2-21	否	
011111	赵琳	女	计0121	01	江苏南京	1985-11-18	是	
021405	罗宇波	男	英0112	03	江苏南通	1985-12-12	否	

1.3.3 关系的基本概念及其特点

1．关系的基本概念

（1）关系：一个关系就是一张二维表，通常将一个没有重复行、重复列的二维表看成一个关系，每个关系都有一个关系名。

（2）元组：二维表的每一行在关系中称为元组。

（3）属性：二维表的每一列在关系中称为属性，每个属性都有一个属性名，属性值则是各个元组在该属性上的取值。

（4）域：属性的取值范围称为域。

2．关系模型的主要优点

关系模型具有如下优点：

（1）数据结构单一。关系模型中，不管是实体还是实体之间的联系，都用关系来表示，而关系都对应一张二维数据表，数据结构简单、清晰。

（2）关系规范化，并建立在严格的理论基础上。关系中每个属性不可再分割，构成关系的基本规范。同时关系是建立在严格的数学概念基础上，具有坚实的理论基础。

（3）概念简单，操作方便。

第2章　关系数据库

2.1　关系数据库概述

所谓关系数据库就是采用关系模型作为数据的组织方式，换句话说就是支持关系模型的数据库系统。

关系模型由三个部分构成：关系数据结构、关系数据操作和完整性约束。

1. 关系数据结构

关系模型的数据结构非常简单，实际上就是一张二维表，但这种简单的二维表却可以表达丰富的语义，可以很方便地描述出现实世界的实体以及实体之间的各种联系。

用户定义完整性

2. 关系数据操作

关系数据操作采用集合操作方式，即操作的对象和结果都是集合。关系数据操作包括查询和更新两个部分：

查询：选择、投影、连接、除、并、交、差等。

更新：增加、删除以及修改

3. 完整性约束

完整性约束条件是关系数据模型的一个重要组成部分，是为了保证数据库中的数据一致性。完整性约束分为三类：实体完整性、参照完整性和用户定义完整性。

2.2　关系数据结构

2.2.1　关系

1．域

定义：域是一组具有相同数据类型的值的集合。域中所包含的值的个数叫做域的基数。域是需要命名的。

2. 笛卡尔积

定义：给定一组域 D1，D2，D3，…Dn，则这些域的笛卡尔积为：D1×D2×D3×…×Dn={(d1,d2,d3,…,dn)|di∈Dj,i=1,2,…,n}，其中：

（2）元组的每一个值 di 叫做一个分量。

（3）笛卡尔积的基数为：$m=\prod_{i=1}^{n} mi$

3．关系

D1×D2×…×Dn 的子集叫作在域 D1,D2,…,Dn 上的关系，用 R（D1,D2,…,Dn）表示。其中 R 表示关系的名字，n 是关系的目或度（degree）。

4．码的定义

（1）码（Key）。

（2）侯选码（Candidate Key）。

（3）主码（Primary Key）。

（4）主属性（Prime Attribute）。

（5）非主属性（Non-Key Attribute）。

5．关系的三种类型

（1）基本关系：基本关系通常又称为基本表或基表，指的是实实在在存在的表。

（2）导出表：导出表是从一个或几个基本表进行查询而得到的结果所对应的表。

（3）视图：视图是由基本表或其他视图表导出的表，是虚表，不对应实际存储的数据。

6．基本关系的 6 条性质

（1）列是同质的，即每一列中的分量是同一类型的数据，来自同一个域。

（2）不同的列可出自同一个域，称其中的每一列为一个属性，不同的属性要给予不同的属性名。

（3）列的顺序无所谓，即列的次序可以任意交换。

（4）任意两个元组不能完全相同。

（5）行的顺序无所谓，即行的次序可以任意交换。

（6）分量必须取原子值，也就是说每一个分量都必须是不可分的数据项。

表2-2 图书关系BOOK

图书号 Bookid	图书名 Bookname	编者 Editor	价格 Price	出版社 Publish	出版年月 PubDate	库存数 Qty
TP2001--001	数据结构	李国庆	22.00	清华大学出版社	2001-01-08	20
TP2003--002	数据结构	刘娇丽	18.9	中国水利水电出版社	2003-10-15	50
TP2002--001	高等数学	刘自强	12.00	中国水利水电出版社	2002-01-08	60
TP2003--001	数据库系统	汪 洋	14.00	人民邮电出版社	2003-05-18	26
TP2004--005	数据库原理与应用	刘淳	24	中国水利水电出版社	2004-07-25	100

表2-3 读者关系READER

借书卡号 Cardid	读者姓名 Name	性别 Sex	工作单位 Dept	读者类别 Class
T0001	刘勇	男	计算机系	1
S0101	丁钰	女	人事处	2
S0111	张清蜂	男	培训部	3
T0002	张伟	女	计算机系	1

表2-4 借书关系BORROW

图书号 Bookid	借书卡号 Cardid	借书日期 Bdate	还书日期 Sdate
TP2003--002	T0001	2003-11-18	2003-12-09
TP2001--001	S0101	2003-02-28	2003-05-20
TP2003--001	S0111	2004-05-06	
TP2003--002	S0101	2004-02-08	

2.2.2 关系模式

所谓关系模式就是对关系的描述。描述的内容包括：

元组集合结构：有哪些属性、属性来自哪些域，属性与域之间的映象关系（属性的长度和类型）。

元组集合的语义。

完整性约束条件：属性间的相互关系，属性的取值范围限制。

2.2.3 关系数据库

所有支持关系数据库模型的实体及实体之间联系的关系集合就构成了一个关系数据库。

关系数据库有型与值之分，型称为关系数据库的模式，值称为关系数据库的值。关系数据库模式与关系数据库的值通常统称为关系数据库。

2.3 关系的完整性

1. 实体完整性
2. 参照完整性
3. 用户定义完整性

表2-5 职工

职工姓名 Name	部门编号 DeptNo
刘勇	01
丁钰	02
张清蜂	

表2-6 部门

部门编号 DeptNo	部门名称 DeptName
01	计算机系
02	人事处
03	电子系

2.4 关系代数

2.4.1 传统的集合运算

1. 并∪
2. 差-
3. 交∩
4. 广义笛卡尔积×

表2-7 关系代数运算符

运算符		含义
集合运算符	∪	并
	-	差
	∩	交
	×	广义笛卡尔积
专门的关系运算符	σ	选择
	∏	投影
	w v	连接
	÷	除

续表

运算符		含义
比较运算符	>	大于
	≥	大于或等于
	<	小于
	≤	小于或等于
	=	等于
	≠	不等于
逻辑运算符	¬	非
	∧	与
	∨	或

2.4.2 专门的关系运算

1 选择 selection

选择又称为限制，它是在关系 R 中选择满足给定条件的元组，组成一个新的关系。记作：

$\sigma_F(R) = \{t|t \in R \wedge F(t) = TRUE\}$

2．投影（projection）

从关系 R 上选取若干属性列 A，并删除重复行，组成新的关系。记作：

$\Pi_A(R) = \{ t[A] | t \in R\}$

投影操作是从列的角度进行的运算。

例 2-3　查询关系 BOOK 中所有图书的书名和对应的出版社。

$\Pi_{Bookname,Publish}(Book)$

结果如下：

Bookname	Publish
数据结构	清华大学出版社
数据结构	中国水利水电出版社
高等数学	中国水利水电出版社
数据库系统	人民邮电出版社
数据库原理与应用	中国水利水电出版社

例 2-4　查询“中国水利水电出版社”出版的所有藏书的书名和库存数量。

$\Pi_{Bookname,\ Qty}(\sigma_{Publish='中国水利水电出版社'}(Book))$

结果如下：

Bookname	Qty
数据结构	50
高等数学	60
数据库原理与应用	100

3．连接（join）

连接也称为 θ 连接。它是从两个关系 R 和 S 的笛卡尔积 R×S 中选取属性间满足一定条件的元组，构成新的关系。记作：

$$R \bowtie S = \{\widehat{t_r t_s} \mid t_r \in R \wedge t_s \in S \wedge X=Y\}$$
$$X=Y$$

表2-9（a） R关系

Bookid	Bookname	Publish
TP2001--001	数据结构	清华大学出版社
TP2003--002	数据结构	中国水利水电出版社
TP2002--001	高等数学	中国水利水电出版社
TP2003--001	数据库系统	人民邮电出版社
TP2004--005	数据库原理与应用	中国水利水电出版社

表2-9（b） S关系

Cardid	Bookid
T0001	TP2003--002
S0101	TP2001--001
S0111	TP2003--001
S0101	TP2003--002

表2-9（c） R和S的等值连接

R. Bookid	BookName	Publish	S. Bookid	Cardid
TP2001—001	数据结构	清华大学出版社	TP2001—001	S0101
TP2003—002	数据结构	中国水利水电出版社	TP2003—002	T0001
TP2003—002	数据结构	中国水利水电出版社	TP2003—002	S0101
TP2003—001	数据库系统	人民邮电出版社	TP2003—001	S0111

4．除（division）

为了说明除法运算，先得给出象集的概念。

象集的定义：给定一个关系 R（X，Z），X 和 Z 为属性组。定义当 t（X）= 时，在 R 中的象集为：

$$Zx=\{t[Z]|t\in R,t[X]= x\}$$

它表示 R 中属性组 X 上值为的诸元组在 Z 上分量的集合。

2.5 关系数据库管理系统

（1）（最小）关系系统。即满足上面最基本的条件，支持关系数据结构，支持选择、投影和连接操作。

（2）关系完备系统。支持关系数据结构和所有的关系代数操作。

（3）全关系。这类系统支持关系模型的所有特征，而且支持数据结构中域的概念及实体完整性和参照完整性。

第3章　关系数据库标准语言SQL

3.1　SQL概述

1．SQL 语言

SQL 语言是 1974 年由 BOYCE 和 CHAMBERLIN 提出的。1975 年至 1979 年 IBM 公司 SANJOSE RESEARCH LABORATORY 研究的关系数据库管理系统原型系统 SYSTEM R 实现了这种语言，由于它功能丰富，语言简洁，使用方便，被众多计算机公司和软件公司所采用，经各公司不断修改、扩充和完善，SQL 语言最终发展为关系数据库的标准语言。

2．扩展 SQL 语言

尽管 ASNI 和 ISO 已经针对 SQL 制定了一些标准，但标准 SQL 语言只能完成数据库的大部分操作，不适合为关系数据库编写各种类型的程序，各家厂商针对其各自的数据库软件版本做了某些扩充和修改，一般都根据需要增加了一些非标准的 SQL 语言。经扩充后的 SQL 语言称为扩展 SQL 语言。

3.2　数据定义

SQL 数据定义功能包括定义基本表、定义视图和定义索引等，如表 3-1 所示。由于视图是基于基本表的虚表，索引是基于基本表的，因此 SQL 通常不提供修改视图和索引语句，用户如果要修改视图或索引，只能先将它们删除，然后重新创建。

1．基本表定义

CREATE TABLE 〈表名〉(属性列表 [完整性定义])

（1）总体说明：

1）其中表名是要定义的基本表的名称。一个表可以由一个或多个属性列组成。

2）创建表时通常还可以定义与该表有关的完整性约束条件。

3）数据类型：SQL Server 2000 常用数据类型。

（2）完整性定义：

1）实体完整性定义语法：

[CONSTRAINT 约束名] PRIMARY KEY[(属性列表)]

2）参照完整性定义语法：

[CONSTRAINT 约束名] FOREIGN KEY(列名)REFERENCES <被参照表表名>(被参照表列名)。

列值非空：[CONSTRAINT 约束名] NOT NULL。

列值惟一：[CONSTRAINT 约束名] UNIQUE[(属性列表)]。

逻辑表达式：[CONSTRAINT 约束名] CHECK(表达式)。

例 3-1 创建图书信息表、读者信息表和借阅表。

（1）创建图书信息表。

```
CREATE TABLE BOOK
(BOOKID CHAR(20) PRIMARY KEY,
BOOKNAME VARCHAR(60) NOT NULL,
EDITOR CHAR(8),
PRICE NUMERIC(5,2),
PUBLISH CHAR(30),
PUBDATE DATETIME,
QTY INT)
```

（2）创建读者信息表。

```
CREATE TABLE READER
(CARDID CHAR(10) PRIMARY KEY,
NAME CHAR(8),
SEX CHAR(2),
DEPT CHAR(20),
CLASS INT)
--读者类型：1 代表教师，2 代表学生，3 代表临时读者。
```

（3）创建借阅表。

```
CREATE TABLE BORROW
(BOOKID CHAR(20),
CARDID CHAR(10),
BDATE DATETIME NOT NULL,
SDATE DATETIME NOT NULL,
PRIMARY KEY(BOOKID,CARDID,BDATE),
CONSTRAINT FK_BOOKID FOREIGN KEY(BOOKID) REFERENCES BOOK(BOOKID),
CONSTRAINT FK_CARDID FOREIGN KEY(CARDID) REFERENCES READER (CARDID))
```

随着应用环境和应用需求的变化，有时需要修改已建立好的基本表，包括增加新列、增加新的完整性约束条件、修改原有的列定义或删除已有的完整性约束条件等。SQL 语言用 ALTER TABLE 语句修改基本表，其一般格式为：

ALTER TABLE <表名>

ALTER COLUMN <列名> <新的类型>[NULL| NOT NULL]

ADD <新列名> <数据类型> [完整性约束]

ADD <表级完整定义>

DROP CONSTRAINT <完整性约束名>

DROP COLUMN <列名>

例 3-2　在图书信息表中增加一列出版时期（PUBDATE），并将 BOOKID 列宽改为 15。

```
ALTER TABLE BOOK
  ADD PUBDATE DATETIME
GO
ALTER TABLE BOOK
 ALTER COLUMN BOOKID CHAR(15)
```

例 3-3　删除借阅表中的参照完整性。

```
ALTER TABLE BORROW
 DROP CONSTRAINT FK_BOOKID
GO
ALTER TABLE BORROW
 DROP CONSTRAINT FK_CARDID
```

3．删除基本表

当某个基本表不再需要时，可以使用 SQL 语句 DROP TABLE 进行删除。其一般格式为：

DROP TABLE <表名>

4．建立索引

在 SQL 语言中，建立索引使用 CREATE INDEX 语句，其一般格式为：

CREATE [UNIQUE] [CLUSTER] INDEX <索引名>

ON <表名> (<列名> [<ASC|DESC>] [,<列名> [<ASC|DESC>]]…)；

5．删除索引

的时间，但如果数据修改频繁，系统会花费许多时间来维护索引。这时，可以删除一些不必要的索引。

3.3 数据查询

建立数据库的目的是为了查询数据，因此，可以说数据库查询是数据库的核心操作。SQL 语言提供了 SELECT 语句进行数据库的查询，该语句具有灵活的使用方式和丰富的功能。其一般格式为：

SELECT [ALL | DISTINCT] <目标列表达式> [, <目标列表达式>…]
FROM <表名或视图名 [别名]> [, <表名或视图名>[别名]]…
［WHERE <条件表达式>］
[GROUP BY <分组表达式> [HAVING <条件表达式>]]
[ORDER BY <排序列名> [ASC | DESC]]；

3.3.1 单表查询

1．选择表中的若干列

选择表中的全部列或部分列，这类运算又称为投影。其变化方式主要表现在 SELECT 子句的<目标表达式>上。

例 3-6 查询所有读者的卡号和姓名。

SELECT CARDID, NAME
FROM READER

例 3-7 查询所有图书信息。

SELECT *
FROM BOOK

说明：*代表所有列

2．选择表中满足条件的记录

查询满足指定条件的元组可以通过 WHERE<条件表达式>子句实现。条件表达式是操作数与运算符的组合，操作数可以包括常数、变量和字段等。常用运算符如表 3-2 所示。

（1）比较运算：

（2）确定范围：

（3）集合运算：

（5）空值判断：

（6）逻辑运算：

3．对查询结果排序

如果没有指定查询结果的显示顺序，DBMS 将按其最方便的顺序（通常是元组在表中的先后顺序）输出查询结果。用户也可以用 ORDER BY 子句指定按照一个或多个属性列的升序（ASC）或降序（DESC）重新排列查询结果，其中升序 ASC 为缺省值。

4．使用集函数

为了进一步方便用户，增强检索功能，SQL 提供了许多集函数，主要包括：

COUNT ([DISTINCT |ALL] *)　统计元组个数。

COUNT ([DISTINCT | ALL] <列名>)　统计一列中值的个数。

SUM ([DISTINCT | ALL] <列名>)　计算一列值的总和（此列必须是数值型）。

AVG（[DISTINCT | ALL] <列名>）　计算一列值的平均值（此列必须是数值型）。

MAX（[DISTINCT | ALL] <列名>）　计算一列值的最大值。

MIN（[DISTINCT | ALL] <列名>）　计算一列值的最小值。

5．分组统计

GROUP BY<分组表达式>子句可以将查询结果中的各行按一列或多列取值相等的原则进行分组。分组一般与集函数一起使用。

对查询结果分组的目的是为了细化集函数的作用范围。如果未对查询结果分组，集函数将作用于整个查询结果，即对查询结果中的所有记录进行计算。如果有分组，集函数将作用于每一个分组，即集函数对每个组分别进行计算。

3.3.2　多表查询

1．等值与非等值连接查询

当用户的一个查询请求涉及到数据库的多个表时，必须按照一定的条件把这些表连接在一起，以便能够共同提供用户需要的信息。用来连接两个表的条件称为连接条件或连接谓词，其一般格式为：

［<表名 1> .］<列名 1> <比较运算符>［<表名 2> .］<列名 2>

```
SELECT NAME
FROM READER,BORROW
WHERE READER.CARDID=BORROW.CARDID AND SDATE IS NULL
```

结果如下：

```
NAME
----------
丁钰
张清蜂
----------
```

例 3-25　查询所有读者信息及借阅情况。

```
SELECT READER.* , BORROW.*
FROM READER , BORROW
WHERE READER.CARDID=BORROW.CARDID
```

该查询结果中将包含 READER 和 BORROW 表中的所有列。

2．自然连接

如果按照两个表中的相同属性进行等值连接，且目标列中去掉了重复的属性列，但保留所有不重复的属性列，则称为自然连接。

例 3-26　自然连接 READER 和 BORROW 表。

```
SELECT READER.CARDID, NAME, SEX, DEPT, CLASS , BOOKID, BDATE, SDATE
FROM READER,BORROW
WHERE READER.CARDID=BORROW.CARDID
```

查询结果如下：

CARDID	NAME	SEX	DEPT	CLASS	BOOKID	BDATE	SDATE
T0001	刘勇	男	计算机系	1	TP2003--002	2003-11-18	2003-12-09
S0101	丁钰	女	人事处	2	TP2001--001	2003-02-28	2003-05-20
S0111	张清蜂	男	培训部	3	TP2003--001	2004-05-06	NULL
S0101	丁钰	女	人事处	2	TP2003--002	2004-02-08	NULL

3．自身连接

连接操作不仅可以在两个表之间操作，也可以是一个表与其自己进行连接，这种操作称为自身连接。

例 3-27　查询书名相同而出版社不同的所有图书的书名。

```
SELECT DISTINCT B1.BOOKNAME
FROM BOOK B1,BOOK B2
WHERE B1.BOOKNAME=B2.BOOKNAME
        AND B1.PUBLISH<>B2.PUBLISH
```

查询结果为：

```
BOOKNAME
------------
数据结构
--------------
```

4．外连接

在通常的连接操作中，只有满足连接条件的元组才能作为结果输出。如例 3-26，如果某读者还没有借书记录，在结果集中就看不到该读者的信息。外连接又分为左外连、右外连和全外连。

左外连：查询结果中不仅包含符合连接条件的行，而且包含左表中所有数据行。

右外连：查询结果中不仅包含符合连接条件的行，而且包含右表中所有数据行。

全外连：查询结果中不仅包含符合连接条件的行，而且包含两个连接表中所有数据行。

5．连接查询综合实例

例 3-29　查询借书期限超过 2 个月的所有读者的姓名、所借书籍名和借书日期。

```
SELECT NAME,BOOKNAME,BDATE
FROM BOOK,READER,BORROW
WHERE BOOK.BOOKID=BORROW.BOOKID
AND READER.CARDID=BORROW.CARDID
AND DATEDIFF(MM, BDATE,GETDATE()) >2
```

例 3-30　按读者姓名查询指定读者的借还书历史记录。假设读者姓名为“刘勇”。

```
SELECT BOOK.NAME,BDATE,SDATE
FROM BORROW,BOOK,READER
WHERE BOOK.BOOKID=BORROW.BOOKID
AND READER.CARDID=BORROW.CARDID
AND READER.NAME='刘勇'
```

3.3.3 嵌套查询

1．带 IN 谓词的子查询

带有 IN 谓词的子查询是指父查询与子查询之间用 IN 进行连接，判断某个属性列值是否在子查询的结果中。由于在嵌套查询中，子查询的结果往往是一个集合，所以谓词 IN 是嵌套查询中最经常使用的谓词。

例 3-32　查询借了“数据库系统”书籍的所有读者的姓名。

```
SELECT NAME
FROM READER
WHERE CARDID IN
  (SELECT CARDID
     FROM BORROW
    WHERE BOOKID IN
      (SELECT BOOKID
         FROM BOOK
        WHERE BOOKNAME='数据库系统'))
```

2．带有比较运算符的子查询

带有比较运算符的子查询是指父查询与子查询之间用比较运算符进行连接。当用户能确切知道内层查询返回的是单值时，可以用>、<、=、>=、<=、!＝或<>等比较运算符。

3．带有 ANY 或 ALL 谓词的子查询

如果用户不能确切知道子查询的返回结果为单值时，可以使用带有 ANY 或 ALL 谓词的子查询，但 ANY 或 ALL 谓词必须与比较运算符一起使用。

例 3-34　查询所有正借阅“中国水利水电出版社”出版的书籍的读者姓名。

```
SELECT NAME
FROM READER
WHERE CARDID =ANY(
  SELECT CARDID
    FROM BORROW
    WHERE SDATE IS NULL AND BOOKID =ANY(
```

```
SELECT BOOKID
FROM BOOK
WHERE PUBLISH='中国水利水电出版社'))
```

4．带有 EXISTS 谓词的子查询

EXISTS 代表存在。带有 EXISTS 量词的子查询不返回任何实际数据，它只产生逻辑真值 true 或逻辑假值 false。若内层查询结果非空，则外层的 WHERE 子句返回真值，否则返回假值。

例 3-35　查询借阅了书号为 TP2004--005 图书的所有读者姓名。

```
SELECT NAME
FROM READER
WHERE EXISTS (
 SELECT *
FROM BORROW
WHERE BORROW.CARDID=READER.ID AND BOOKID='TP2004--005')
```

3.4　数据更新

3.4.1　插入数据

1．插入单个元组

插入单个元组的 INSERT 语句的语法格式为：

INSERT INTO<表名>[(<属性列 1>[,<属性列 2>…])]

VALUES(<常量 1>[,<常量 2>…])

其功能是将新元组插入指定表中。

2．插入子查询结果

插入子查询结果的 INSERT 语句的格式为：

INSERT INTO<表名>[(<属性列 1>[,<属性列 2>…])]

子查询；

其功能是以批量插入，一次将子查询的结果全部插入指定表中。子查询结果中列数应与 INTO 子句中的属性列数相同，否则会出现语法错误。

```
CREATE TABLE BOOKQTY
(BOOKID CHAR(20),
   QTY INT)
GO
INSERT INTO BOOKQTY
SELECT BOOKID, COUNT(*)
FROM BORROW
WHERE SDATE IS NULL
GROUP BY BOOKID
```

3.4.2 修改数据

修改操作又称为更新操作，其语句格式为：

UPDATE <表名>

SET<列名>=<表达式>[，<列名>=<表达式>]…

[WHERE<条件>]；

其功能是修改指定表中满足 WHERE 条件的元组。其中 SET 子句用于指定修改方法，即用<表达式>的值取代相应的属性列值。如果省略 WHERE 子句，则表示要修改表中的所有元组。

例 3-39　读者还书操作。设读者卡号为 T0001，书号为 TP2003--002。

```
UPDATE BORROW
SET SDATE=GETDATE()
WHERE BOOKID='TP2003--002' AND CARDID='T0001'
GO
UPDATE BOOK
SET QTY=QTY+1
WHERE BOOKID='TP2003--002'
```

3.4.3 删除数据

删除数据指删除表中的某些记录，删除语句的一般格式为：

DELETE

FROM<表名>

[WHERE<条件>]；

子句，表示删除表中的全部元组，但表的结构仍在。也就是说，DELETE 语句删除的是表中的数据，而不是表的结构。

例 3-41　删除卡号为 T0035 的读者的所有借书记录，然后删除该读者信息。

```
DELETE
FROM BORROW
WHERE CARDID='T0035'
GO
DELETE
FROM READER
WHERE CARDID ='T0035'
```

3.5　视图

1．建立视图

SQL 语言用 CREATE VIEW 命令建立视图，其一般格式为：

CREATE VIEW<视图名>[(<列名 1>[，<列名 2>]…)]

AS<子查询>

[WITH CHECK OPTION];

其中子查询可以是任意复杂的 SELECT 语句，但通常不允许含有 ORDER BY 子句和 DISTINCT 短语。

例 3-44　建立读者类别为学生（CLASS=1）的读者视图。

```
CREATE VIEW S_READER
AS
SELECT *
FROM READER
WHERE CLASS=1
```

例 3-45　创建教师读者视图，并要求进行修改和插入操作时仍保证视图只有教师记录。

```
CREATE VIEW T_READER
AS
SELECT CARDID, NAME, SEX, DEPT
FROM READER
```

WITH CHECK OPTION

2．删除视图

视图建好后，若导出此视图的基本表被删除了，该视图将失效，但一般不会被自动删除。删除视图通常需要显式地使用 DROP VIEW 语句进行。该语句的格式为：

DROP VIEW<视图名>；

一个视图被删除后，由该视图导出的其他视图也将失效，用户应该使用 DROP VIEW 语句将他们一一删除。

例 3-49　删除例 3-36 创建的视图 T_BORROW。

```
DROP VIEW T_BORROW
```

3．查询视图

视图定义后，用户就可以像对基本表进行查询一样对视图进行查询了。也就是说，在 3.3 节中介绍的对基本表的各种查询操作一般都可以用于查询视图。

例 3-50　查询读者“刘伟”的借书信息。

```
SELECT NAME, BOOKNAME, BDATE
FROM V-BORROW
WHERE NAME='刘伟'
```

4．更新视图

更新视图包括插入（INSERT）、删除（DELETE）和修改（UPDATE）三类操作。由于视图是不实际存储数据的虚表，因此对视图的更新，最终要转换为对基本表的更新。

例 3-52　通过视图 T_READER 修改读者（T0078）为“计算机系”。

```
UPDATE T_READER
SET DEPT='计算机系'
WHERE CARDID='T0078'
```

由于 T_READER 定义时带了 WITH CHECK OPTION 子句，所以 DBMS 实际执行的操作相当于：

```
UPDATE READER
SET DEPT='计算机系'
WHERE CARDID='T0078' AND CLASS=2
```

5．视图的用途

（1）视图能够简化用户的操作。

（2）视图使用户能以多种角度看待同一数据。

（3）视图对重构数据库提供了一定程度的逻辑独立性。

（4）视图能够对机密数据提供安全保护。

3.6　数据控制

SQL 语言提供了数据控制功能，能够在一定程度上保证数据库中数据的安全性和完整性，并提供了一定的并发控制及恢复能力。

数据库的完整性是指数据库中数据的正确性与相容性。SQL 语言定义完整性约束条件的功能主要体现在 CREATE TABLE 语句中，可以在该语句中定义码、取值惟一的列、参照完整性及其他一些约束条件。

1．授权

SQL 语言用 GRANT 语句向用户授予操作权限，GRANT 语句的一般格式为：

GRANT <权限 1> [,<权限>2]…

[ON <对象名>]

TO <用户 1> [,<用户 2>]…

[WITH GRANT OPTION]

其功能是将指定对象的指定操作权限授予指定的用户。

例 3-54　授予用户 USER1 创建表的权限，并允许他将此权限授予其他用户。

```
USE BOOKSYS
GRANT CREATE TABLE TO USER1
WITH GRANT OPTION
```

例 3-55　授予用户 USER1 有更新读者表（READER）中读者姓名字段的权限。

```
GRANT UPDATE ON READER（NAME）TO USER1
```

2．权限收回

授予的权限可以由 DBA 或其授权者用 REVOKE 语句收回，REVOKE 语句的一般格式为：

REVOKE <权限 1> [,<权限 2>]…

[ON <对象名>]

其功能是从指定的用户中收回指定的权限。

例 3-58　回收 USER1 对 BOOK 表的查询与修改权限。

REVOKE SELECT, UPDATE ON BOOK FROM USER1

如果要收回 USER1 的所有权限可以使用下面的命令：

REVOKE ALL PRIVILEGES ON BOOK FROM USER1

3．拒绝语句

有时要对用户在某对象上的操作权限暂时禁用，或取消某用户从角色中继承下来的权限，可以使用拒绝语句，其语法格式如下：

DENY〈ALL | 权限〉ON<对象> TO〈用户〉

例 3-59　如果按例 3-56 将 BOOK 表的 SELECT 权限授予了 PUBLIC，因为所有用户都是 PUBLIC 的成员，所以 USER1 也获得了对 BOOK 表的 SELECT 操作权限，如果要收回这个权限，必须用 DENY 语句。

第4章　关系数据库设计理论

4.1　数据依赖

4.1.1　关系模式中的数据依赖

关系是一张二维表，它是所涉及属性的笛卡尔积的一个子集。从笛卡尔积中选取哪些元组构成该关系，通常是由现实世界赋予该关系的元组语义来确定的。元组语义实质上是一个 N 目谓词（其中 N 是属性集中属性的个数）。使该 N 目谓词为真的笛卡尔积中的元素（或者说凡符合元组语义的元素）的全体就构成了该关系。

关系模式是对关系的描述，为了能够清楚地刻画出一个关系，它需要由五部分组成，即应该是一个五元组：

R（U, D, DOM, F）

其中：R 为关系名，U 为组成该关系的属性名集合，D 为属性组 U 中属性所来自的域，DOM 为属性向域的映像集合，F 为属性间数据的依赖关系集合。

4.1.2　数据依赖对关系模式的影响

关系数据库设计理论的中心问题是数据依赖性。所谓数据依赖是实体属性值之间相互联系和相互制约的关系，是现实世界属性间相互联系的抽象，是数据内在的性质，是语义的体现。现在人们已经提出了许多类型的数据依赖，其中函数依赖（Functional Dependency，简称为 FD）和多值依赖（Multivalued Dependency，简称为 MVD）是与数据库设计理论中最重要的两种数据依赖类型。

从上述事实可以得到属性组 U 上一组函数依赖 F（如图 4.1 所示）：

F＝｛Cardid→Class, Class→Maxcount,（Bookid, Cardid, Bdate）→Sdate｝

如果仅仅考虑函数依赖这一种数据依赖，就得到一个描述“学校图书管理”的关系模式 Book＜U, F＞。但这个关系模式存在 4 个问题

(1)存在较大数据冗余（Date Redundancy）。

(2)更新异常（Update Anomalies）。

(3)插入异常（Insertion Anomalies）。

(4)删除异常（Deletion Anomalies）。

4.1.3 有关概念

1．函数依赖

（1）函数依赖是指关系模式 R 的所有元组均要满足的约束条件，而不仅仅指 R 中某个或某些元组满足的约束条件特例。

（2）函数依赖并不一定具有可逆性。例如一般认为 Cardid→Class，即由于读者的卡号具有惟一性，因此读者的卡号可确定读者的类型，而反之则不行。

（3）若 X→Y，则 X 称为这个函数依赖的决定属性集（Determinant）。

（4）函数依赖和别的数据之间的依赖关系一样，是语义范畴的概念。

（5）数据库设计者可以对描述现实世界的关系模式作强制性的规定。

（6）若 X→Y，并且 Y→X，则记为 *X*←→*Y*。

（7）若 Y 不函数依赖于 X，则记为 X　　Y。

2．平凡函数依赖与非平凡函数依赖

对于任意一种关系模式，平凡函数依赖都是必然成立的，它不反映新的语义，因此在本章中，若不特别声明，总是讨论非平凡函数依赖。

5．码

码是关系模式中的一个重要概念，候选码能惟一标识一个元组（二维表中的一行），是关系模式中一组最重要的属性。另一方面，主码又和外部码一同提供了表示关系间联系的手段。

4.2 范式

4.2.1 第一范式（*1NF*）

简单地说，第一范式要求关系中的属性必须是原子项，即不可再分的基本类型，集合、数组和结构不能作为某一属性出现，严禁关系中出现“表中有表”的情况。

在任何一个关系数据库系统中，第一范式是关系模式的一个最起码的要求。不满足第一范式的数据库模式不能称为关系数据库。

BRB 关系存在以下 4 个问题：

（1）插入异常。

（2）删除异常。

（4）修改复杂。

4.2.2 第二范式（2NF

BRB 关系模式之所以出现上述问题，其原因是 Class、Readername 等非主属性对码的部分函数依赖。为了消除这些部分函数依赖，可以采用投影分解法，把 BRB 关系分解为两个关系模式：借阅和读者。

BORROW(Bookid, Cardid, Bdate, Sdate)

READER(Cardid, Readername, Class, Maxcount)

其中：READER 关系模式的码为(Cardid)，BORROW 关系模式的码为(Bookid, Cardid, Bdate)。他们的函数依赖如图 4-3 所示：

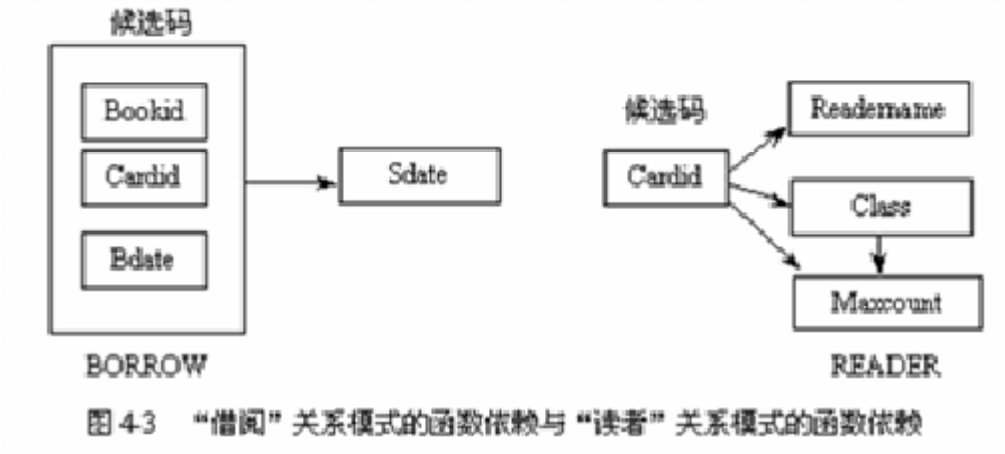


图 4-3 “借阅”关系模式的函数依赖与“读者”关系模式的函数依赖

READER 关系还是存在一些问题：

（1）插入异常。

（2）删除异常。

（3）仍有较大数据冗余

（4）修改复杂。

4.2.3 第三范式（3NF）

“读者”关系模式 READER 出现上述问题的原因是该关系模式含有传递函数依赖。为了消除该传递函数依赖，可以采用投影分解法，把“读者”关系模式 READER 分解为两个关系模式：读者和读者类别。

READER(Cardid, Readername, Class)

其中“读者”关系模式 READER 中的码为 Cardid，“读者类别”关系模式 READERCLASS 中的码为 Class。这两个关系模式的函数依赖如图 4-4 所示。

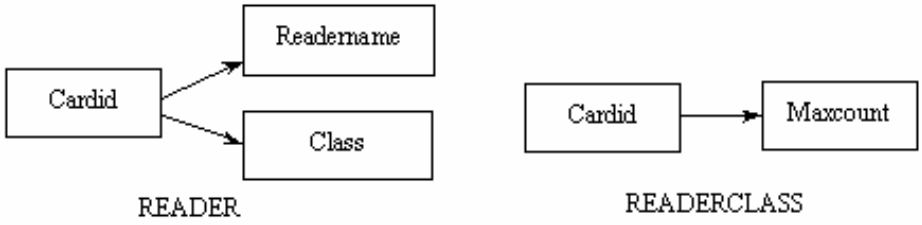


图 4-4　READER 的函数依赖与 READERCLASS 的函数依赖

显然，在分解后的关系模式中既没有非主属性对码的部分函数依赖，也没有非主属性对码的传递函数依赖，这在一定程度上解决了上述 4 个问题。

（1）“读者类别”READERCLASS 关系模式中可以插入暂无读者信息的“读者类型”相关信息。

（2）如果删除一个类型中的所有读者，只是删除 READER 关系中的相应元组，READERCLASS 关系中关于该类别的相关信息（如 Maxcount）将仍保存。

（3）每个“读者类型”对应的 Maxcount 信息只在 READERCLASS 关系中存储一次。

（4）当图书管理部门要修改某“读者类别”对应的 Maxcount 值时，只需修改 READERCLASS 关系中一个相应元组的 Maxcount 属性值。

例如，关系模式 STJ(S, T, J)，S 表示学生，T 表示教师，J 表示课程。假设每一个教师只教一门课，每门课由若干教师教，某一学生选定某门课程，就确定了一个固定的教师，于是有如下函数依赖关系，如图 4-5 所示。

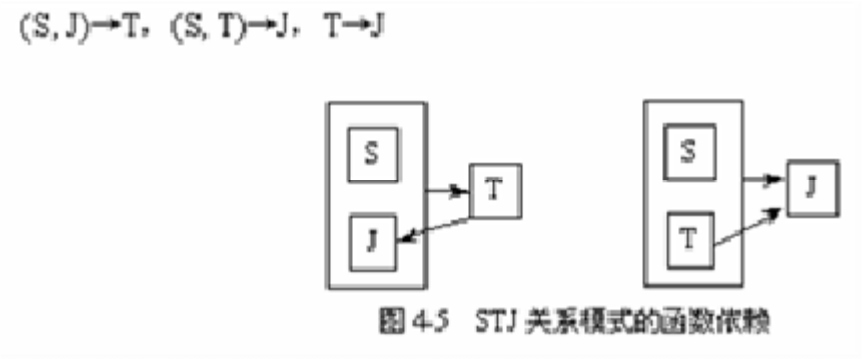


图 4-5　STJ 关系模式的函数依赖

属于 3NF 的 STJ 关系模式也存在一些问题：

（1）插入异常。

（2）删除异常。

（4）修改复杂。

4.2.4 BC范式（*BCNF*）

BCNF（Boyce Codd Normal Form）是由 Boyce 和 Codd 联合提出的，比 3NF 更进一步。通常认为 BCNF 是修正的第三范式。

STJ 关系模式出现上述问题的原因在于主属性 J 依赖于 T，即主属性 J 部分依赖于码(S, T)。解决这一问题仍然可以采用投影分解法，将 STJ 关系分解为两个关系模式：

ST(S, T)

TJ(T, J)

由 BCNF 的定义可以看到，每个 BCNF 的关系模式都具有如下 3 个性质：

（1）所有非主属性都完全函数依赖于每个候选码。

（2）所有主属性都完全函数依赖于每个不包含它的候选码。

（3）没有任何属性完全函数依赖于非码的任何一组属性。

4.2.5 多值依赖与第四范式（*4NF*）

例如，有关系模式 Teach(C,T,B)，C 表示课程，T 表示教师，B 表示参考书。假设该关系如图 4-6 所示。

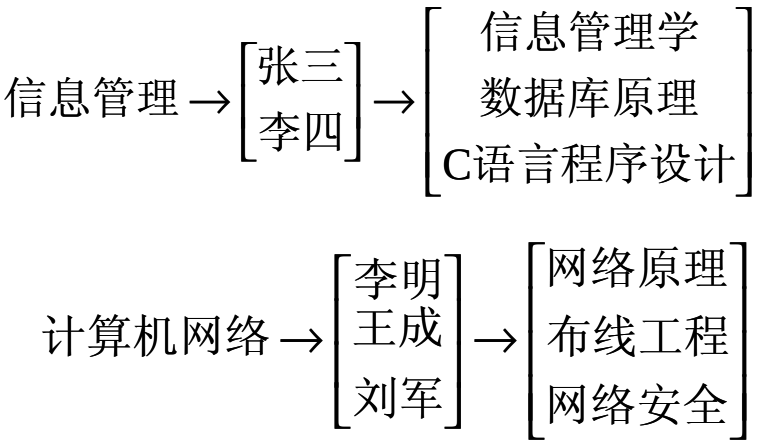


图4-6 课程－教师－参考书之间的关系

Teach 具有惟一候选码(C,T,B)，即全码，因而 TeachBCNF。但 Teach 模式中存在一些问题：

1 数据冗余度大

（2）增加操作复杂。

（3）删除操作复杂。

（4）修改操作复杂。

1．多值依赖

（1）多值依赖具有对称性。

（2）多值依赖具有传递性。

（3）函数依赖可以看作是多值依赖的特殊情况。

（4）若 X→→Y，X→→Z，则 X→→Y∩Z。

（5）若 X→→Y，X→→Z，则 X→→Y-Z，X→→Z-Y。

（6）多值依赖的有效性与属性集的范围有关。

（7）若多值依赖 X→→Y 在 R（U）上成立，对于 Y，并不一定有 X→→成立。

2．第四范式（4*NF*）

通俗地说，一个关系模式如果已满足 *BCNF*，且没有非平凡且非函数依赖的多值依赖，则关系模式属于 4*NF*。一个关系模式 R4*NF*，则必有 R*BCNF*。4*NF* 就是限制关系模式的属性之间不允许有非平凡且非函数依赖的多值依赖。

前面讨论过的关系模式 Teach 中存在非平凡的多值依赖 C→→T，且 C 不是候选码，因此 Teach 不属于 4NF。这正是它之所以存在数据冗余度大、插入和删除操作复杂等弊病的根源。可以用投影分解法把 Teach 分解为如下两个 4NF 关系模式以减少数据冗余：

CT(C,T)

CB(C,B)

4.3 关系模式的分解

4.3.1 关系模式规范化的步骤

规范化的基本思想是对已有的关系模式进行分解来实现的，它逐步消除数据依赖中不合适的部分，把低一级的关系模式分解为多个高一级的关系模式，使模式中的各关系模式达到某种程度的“分离”。即采用“一事一地”的模式设计原则，让一个关系描述一个概念、一个实体或者实体间的一种联系，若多于一个概念就把它“分离”出去。因此所谓规范化实质上是概念的单一化。

（1）对 1NF 关系进行投影，消除原关系中非主属性对码的部分函数依赖，将 1NF 关系转换为若干个 2NF 关系。

（2）对 2NF 关系进行投影，消除原关系中非主属性对码的传递函数依赖，从而产生一组 3NF 关系。

（3）对 3NF 关系进行投影，消除原关系中主属性对码的部分函数依赖和传递函数依赖（也就是说，使决定属性都成为投影的候选码），得到一组 BCNF 关系。

（4）对 BCNF 关系进行投影，消除原关系中非平凡且非函数依赖的多值依赖，从而产生一组 4NF 关系。

（5）对 4NF 关系进行投影，消除原关系中不是由候选码所蕴含的连接依赖，即可得到一组 5NF 关系。

4.3.2 关系模式的分解

关系模式的规范化过程是通过对关系模式的分解来实现的，但是把低一级的关系模式分解为若干个高一级的关系模式的方法并不是惟一的。在这些分解方法中，只有能够保证分解后的关系模式与原关系模式等价的方法才有意义。

READER 关系模式有下列函数依赖：

Cardid→Class

Class→Maxcount

Cardid→Maxcount

第一种分解方法是将 READER 分解为下面 3 个关系模式：

R1（Cardid） R2（Dept） R3（Class）

分解后的关系为：

R1

Cardid
T0001
T0101
S0111
S0102

R2

Class
1
3
2

R3

Maxcount
10
5

第二种分解方法是将 READER 分解为下面两个关系模式：

RM(Cardid, Maxcount)　　CM(Class, Maxcount)

分解后的关系为：

RM

Cardid	Maxcount
T0001	10
T0101	10
S0111	5
S0102	5

CM

Class	Maxcount
1	10
3	5
2	5

第三种分解方法是将 READER 关系分解为下面两个关系模式：

RC(Cardid, Class)，RM(Cardid, Maxcount)

分解后的关系为：

RC

Cardid	Class
T0001	1
T0101	1
S0111	3
S0102	2

RM

Cardid	Maxcount
T0001	10
T0101	10
S0111	5
S0102	5

第四种分解方法是将 READER 分解为下面两个关系模式：

RC(Cardid, Class)，CM(Class, Maxcount)。

这种分解方法保持了函数依赖。

判断关系模式的一个分解是否与原关系模式等价可以有三种不同的标准：

（1）分解具有无损连接性。

（2）分解要保持函数依赖。

（3）分解既要保持函数依赖，又要具有无损连接。

第5章　数据库安全性和完整性

5.1　数据库的安全性

5.1.1　安全性控制的一般方法

实际上，安全性问题并不是数据库系统所独有的，所有计算机系统中都存在这个问题，只是由于数据库系统中存放了大量数据，并为许多用户直接共享，使安全性问题更为突出而已。所以，在计算机系统中，安全措施一般是一级一级层层设置的，例如，图 5-1 就是一种很常用的安全模型。

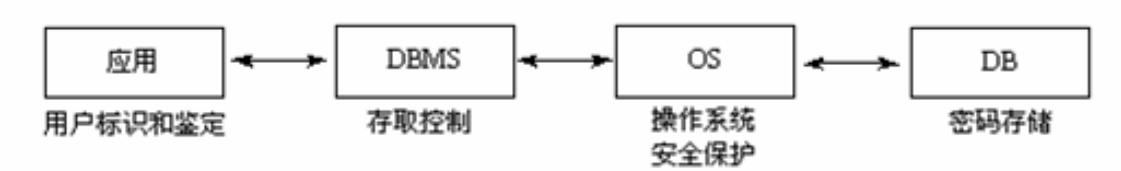


图 5-1　安全模型

1．用户标识和鉴定

用户标识和鉴定是系统提供的最外层安全保护措施。其方法是由系统提供一定的方式让用户标识自己的名字或身份。系统内部记录着所有合法用户的标识，每次用户要求进入系统时，由系统将用户提供的身份标识与系统内部记录的合法用户标识进行核对，通过鉴定后才提供机器使用权。用户标识和鉴定的方法有很多种，而且在一个系统中往往是多种方法并举，以获得更强的安全性。

标识和鉴定一个用户最常用的方法是用一个用户名或者用户标识号来标明用户身份，系统鉴别此用户是否是合法用户。若是，则可进入下步的核实；若不是，则不能进入系统。

2．存取控制

在数据库中，为了保证用户只能访问他有权存取的数据，必须预先对每个用户定义存取权限。对于通过鉴定进入系统的用户（即合法用户），系统根据他的存取权限定义对他的各种操作请求进行控制，确保他只执行合法操作。

存取权限由数据对象和操作类型两个要素组成。定义一个用户的存取权限就是要定义这个用户可以在哪些数据对象上进行哪些类型的操作。在数据库系统中，定义存取权限称为授权（Authorization）。这些授权定义经过编译后存放在数据字典中。

进行存取的控制，不仅可以通过授权与收回权力来实现，还可以通过定义用户的外模式来提供一定的安全保护功能。在关系系统中，就是为不同的用户定义不同的视图，通过视图机制把要保密的数据对无权存取这些数据的用户隐藏起来，从而自动地对数据提供一定程度的安全保护。

4．审计

用户识别和鉴定、存取控制、视图等安全性措施均为强制性机制，将用户操作限制在规定的安全范围内。但实际上任何系统的安全性措施都不可能是完美无缺的，蓄意盗窃、破坏数据的人总是想方设法打破控制。所以，当数据相当敏感，或者对数据的处理极为重要时，就必须以审计技术作为预防手段，监测可能的不合法行为。

审计追踪使用的是一个专用文件或数据库，系统自动将用户对数据库的所有操作记录在上面，利用审计追踪的信息，就能重现导致数据库现有状况的一系列事件，以找出非法存取数据的人。

5．数据加密

对于高度敏感性数据，例如财务数据、军事数据和国家机密，除以上安全性措施外，还可以采用数据加密技术，以密码形式存储和传输数据。这样企图通过不正常渠道获取数据，例如，利用系统安全措施的漏洞非法访问数据，或者在通信绕路上窃取数据，那么只能看到一些无法辨认的二进制代码。用户正常检索数据时，首先要提供密码钥匙，由系统进行译码后，才能得到可识别的数据。

5.1.2　数据库用户的种类

数据库用户按其操作权限的大小可分为三类：

1．数据库系统管理员

2．数据库对象拥有者

3．普通用户

5.2　SQL Server数据库的安全性

数据库的安全是数据库技术的重要组成部分，Microsoft 公司推出的 SQL Server 2000 企业版运行在 Windows 2000 Server 或 Windows 2000 Advanced Sever 操作系统上，也可以运行在 Windows NT SQL Server

2000 也可以使用自己的安全管理技术。

SQL Server 的安全性管理包括以下几个方面：数据库登录管理、数据库用户管理、数据库角色管理以及数据库权限的管理。

5.3 完整性

数据库的完整性是指数据的正确性和相容性。

数据库是否具备完整性关系到数据库系统能否真实地反映现实世界，因此维护数据库的完整性是非常重要的。

数据的完整性与安全性是数据库保护的两个不同方面。

5.3.1 完整性约束条件

完整性约束条件作用的对象可以有列级、元组级和关系级三种粒度。其中对列的约束主要指对其取值类型、范围、精度和排序等的约束条件。对元组的约束是指对记录中各个字段间的联系的约束。对关系的约束是指对各记录间或关系之间的联系的约束。

1．静态列级约束

静态列级约束是对一个列的取值域的说明，这是最常见最简单同时也最容易实现的一类完整性约束，包括以下几方面：

（1）对数据类型的约束，包括数据的类型、长度、单位和精度等。

（2）对数据格式的约束。

（3）对取值范围或取值集合的约束。

（4）对空值的约束。

（5）其他约束。

2．静态元组约束

一个元组是由若干个列值组成的，静态元组约束就是规定组成一个元组的各个列之间的约束关系。

静态元组约束只局限在单个元组上，因此比较容易实现。例如在图书借阅表中可以规定：还书日期>借书日期。

3　静态关系约束

在一个关系的各个元组之间或者若干关系之间常常存在各种联系或约束。常见的静态关系约束有以下四种：

（1）实体完整性约束。

（2）参照完整性约束。

（3）函数依赖约束。

（4）统计约束。

4．动态列级约束

动态列级约束是修改列定义或列值时要满足的约束条件，包括以下两方面：

（1）修改列定义时的约束。

（2）修改列值时的约束。

5．动态元组约束

动态元组约束是指修改某个元组时需要参照其旧值，并且新旧值之间需要满足某种约束条件。

6．动态关系约束

动态关系约束是加在关系变化前后状态上的限制条件。例如，事务一致性、原子性等约束条件。

5.3.2 完整性控制

1．完整性约束的定义

DBMS 的完整性控制机制应具有三个方面的功能：

（1）定义功能：即提供定义完整性约束条件的机制。

（2）检查功能：即检查用户发出的操作请求是否违背了完整性约束条件。

（3）如果发现用户的操作请求使数据违背了完整性约束条件，则采取一定的动作来保证数据的完整性。

2．对违背了完整性约束条件的操作应采取的措施

对于违反实体完整性规则和用户定义的完整性规则的操作一般都是采用拒绝执行的方式进行处理。而对于违反参照完整性的操作，并不都是简单的拒绝执行，有时还需要采取另一种方

（1）删除被参照关系的元组时的考虑。

1）级联删除（Cascades）。

2）受限删除（Restricted）。

3）置空值删除（Nullifies）。

（2）修改被参照关系中主码的考虑。

1）级联修改（Cascades）。

2）受限修改（Restricted）。

3）置空值修改（Nullifies）。

（3）外码是否可以接受空值。

外码是否可以接受空值是由其语义来决定的，在上面提到的职工-部门数据库中，“职工”关系包含有外码“部门号”，某一元组的这一列若为空值，表示这一职工尚未分配到任何具体的部门工作。这和应用环境的语义是相符的，因此“职工”表的“部门号”列应允许空值，但在学生-选课数据库中，“学生”关系为被参照关系，其主码为“学号”。“选课”为参照关系，外码为“学号”。

5.3.3 SQL Server的完整性

1．实体完整性

数据库中最重要的约束就是实体完整性约束，在 SQL Server 中是通过说明某个属性或属性集构成关系的主码来实现关系的实体完整性。主码意味着对于关系的任意两个元组，不允许在说明为主码的属性或属性组上取相同的值，也不允许主码中属性取空值。

主码约束在建表语句中说明，可用两种方法说明主码：第一种方法是在列定义中用关键字 PRIMARY KEY；第二种方法是在表级完整性定义中使用[CONSTRAINT 约束名] PRIMARY KEY[(属性列表)]子句。

2．参照完整性

外码的定义是在表定义时通过 FOREIGN KEY 子句来完成的。其语法格式如下：

[CONSTRAINT 约束名] FOREIGN KEY（列名）REFERENCES <被参照表表名>（被参照表列名）[ON <DELETE|UPDATE> < CASCADES | RESTRICTED | NULLIFIES >]

其中：[ON <DELETE|UPDATE> < CASCADES | RESTRICTED | NULLIFIES >]选项是说明当被参照关系某元组被删除或主码被修改时，参照关系中相应元组的处理办法。

3 用户定义的完整性

列值非空（NOT NULL 短语）。

列值惟一（UNIQUE 短语）。

检查列值是否满足一个布尔表达式（CHECK 短语）。

第6章　数据库设计

6.1　数据库设计概述

数据库设计是指对一个给定的应用环境，构造最优的、最有效的数据库模式，建立数据库及其应用系统，使之能够高效率地存取数据，满足各种用户的应用需求。数据库设计通常是在一个通用的 DBMS 支持下进行的，本书都是以关系数据库—Oracle 为基础来设计数据库的。

数据库的设计工作通常分阶段进行，不同的阶段完成不同的设计内容。数据库规范设计方法通常将数据库的设计分为 6 个阶段，如图 8-1 所示。

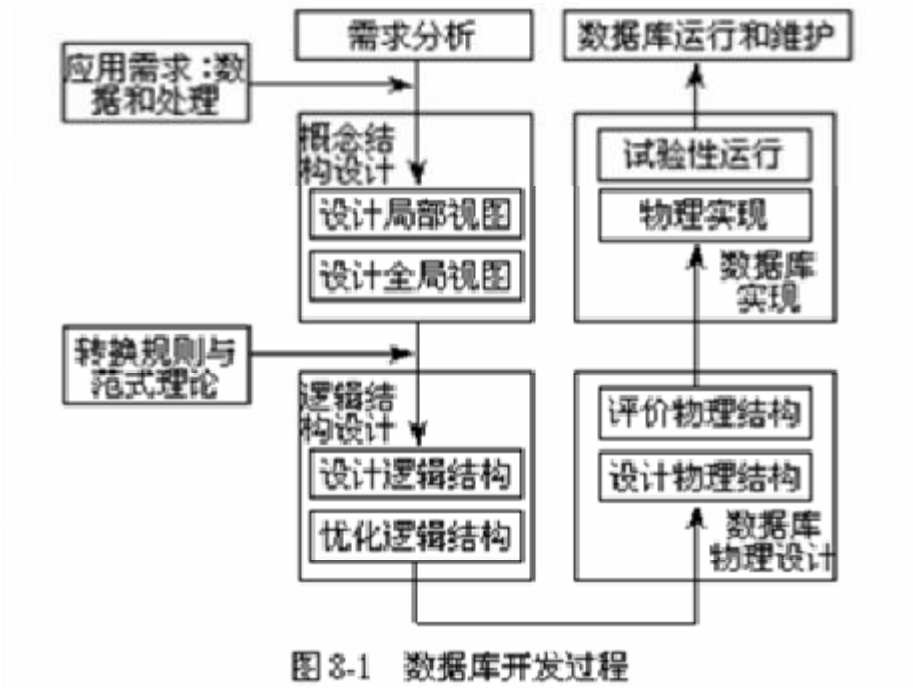


图 8-1　数据库开发过程

数据库的设计分为 6 个阶段

（1）需求分析。收集和分析用户对系统的信息需求和处理需求，得到设计系统所必须的需求信息，建立系统说明文档。

（2）概念结构设计。概念结构设计是整个数据库设计的关键。它通过对用户的需求进行综合、归纳与抽象，形成一个独立于具体 DBMS 的概念模型。

（3）逻辑结构设计。在概念模型的基础上导出一种 DBMS 支持的逻辑数据库模型（如关系型、网络型或层次型），该模型应满足数据库存取、一致性及运行等各方面的用户需求。

（4）物理结构设计。从一个满足用户需求的已确定的逻辑模型出发，在限定的软、硬件环境下，利用 DBMS 提供的各种手段设计数据库的内模式，即设计数据的存储结构和存取方法。

（5）数据库实施。运用 DBMS 提供的数据语言及宿主语言，根据逻辑设计和物理设计的结果建立数据库，编制与调试应用程序，组织数据入库，并进行试运行。

（6）数据库运行和维护。

6.2 需求分析

6.2.1 需求分析的任务

根据需求分析的目标，需求分析这一阶段的任务主要有两项：

（1）确定设计范围。通过详细调查现实世界要处理的对象（组织、部门和企业等），弄清现行系统（手工系统或计算机系统）的功能划分、总体工作流程，明确用户的各种需求。

（2）数据收集与分析。需求分析的重点是在调查研究的基础上，获得数据库设计所必须的数据信息。

6.2.2 需求分析的基本步骤

1．调查与初步分析用户的需求，确定系统的边界

（1）首先调查组织机构情况。

（2）然后调查各部门的业务活动情况。

（3）在熟悉了业务活动的基础上，协助用户明确对新系统的各种要求，包括信息要求、处理要求、安全性与完整性要求，这是调查的又一个重点。

（4）最后对前面调查的结果进行初步分析，确定新系统的边界，确定哪些功能由计算机完成或将来由计算机完成，哪些活动由人工完成。

2．分析和表达用户的需求

（1）数据流图。

数据流图（Data Flow Diagram，简称 DFD）是一种最常用的结构化分析工具，它用图形的方式来表达数据处理系统中信息的变换和传递过程。

（2）数据字典。

1）数据项条目：数据项是不可再分的数据单位，它直接反映事物的某一特征。

3）数据流条目：数据流是数据结构在系统内传输的路径。

4）数据文件条目：数据文件是数据项停留或保存的地方，也是数据流的来源和去向之一。

5）处理过程条目。

6.2.3 需求分析应用实例

现要开发高校图书管理系统。经过可行性分析和初步的需求调查，确定了系统的功能边界，该系统应能完成下面的功能：

（1）读者注册。

（2）读者借书。

（3）读者还书。

（4）图书查询。

具体的数据流图和数据字典如下。

1．数据流图

通过对系统的信息及业务流程进行初步分析后，首先抽象出该系统最高层的数据流图，即把整个数据处理过程看成是一个加工的顶层数据流图，如图 8-5 所示。

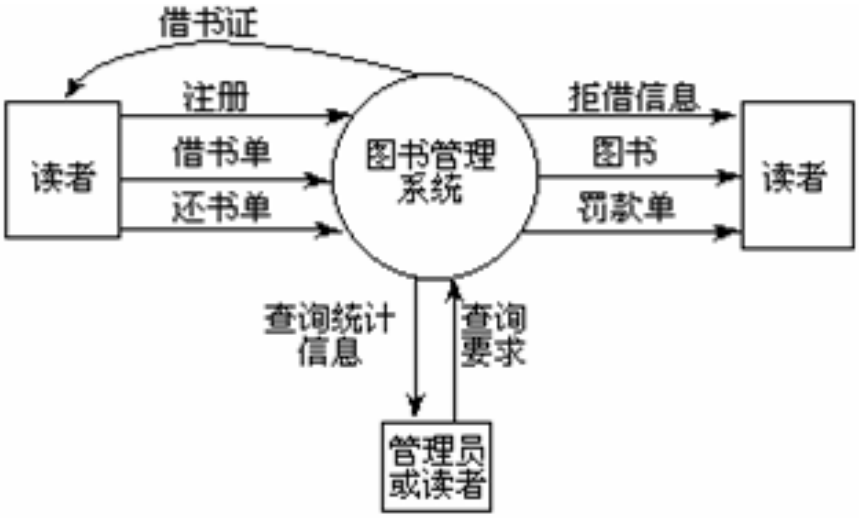


图 8-5 图书管理系统顶层数据流图

顶层数据流图反映了图书管理系统与外界的接口，但未表明数据的加工要求，需要进一步细化。根据前面图书管理系统功能边界的确定，再对图书管理系统顶层数据流图中的处理功能做进一步分解，可分解为读者注册、借书、还书和查询四个子功能，这样就得到了图书管理系

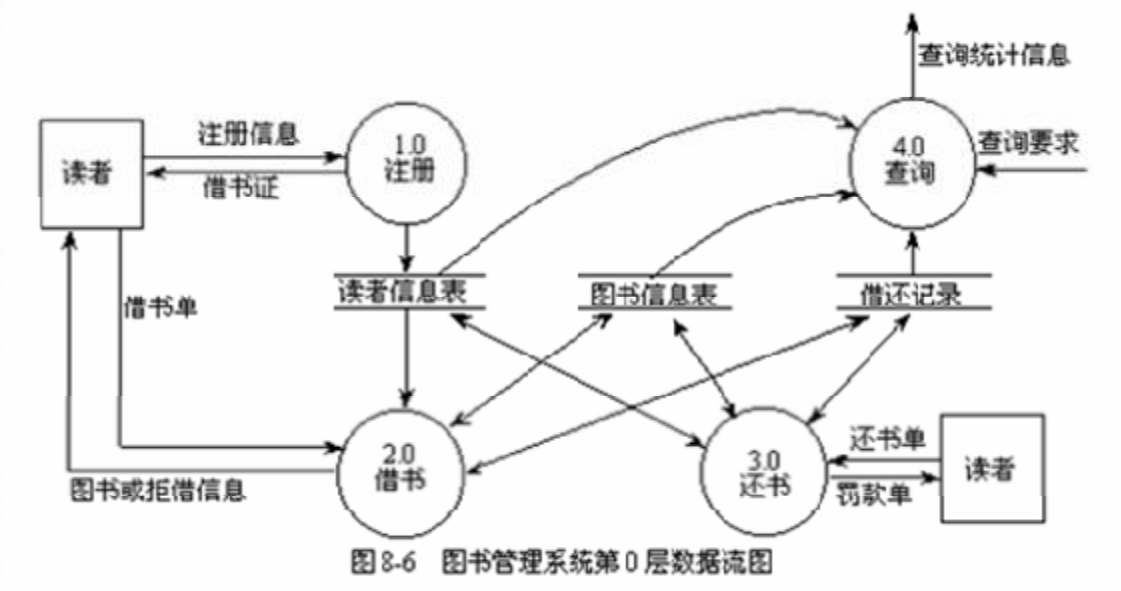
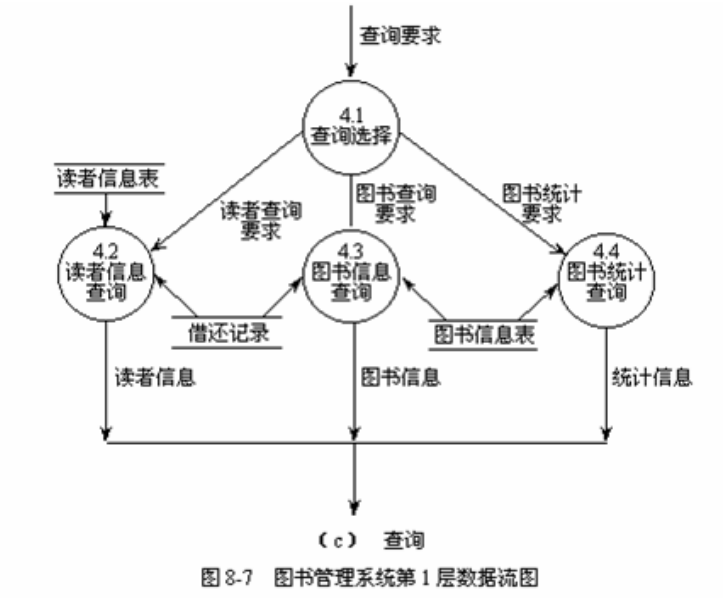


图 8-6　图书管理系统第 0 层数据流图

从图书管理系统第 0 层数据流图中可以看出，在图书管理的不同业务中，借书、还书、查询这几个处理较为复杂，使用到不同的数据较多，因此有必要对其进行更深层次的分析，即构建这些处理的第 1 层数据流图。下面的图 8-7 分别给出了借书、还书、查询子功能的第 1 层数据流图。



（c）　查询

图 8-7　图书管理系统第 1 层数据流图

2．数据字典

（1）数据项描述。

（2）数据结构描述

（4）数据存储说明。

（5）处理过程说明。

6.3 概念结构设计

6.3.1 概念结构设计的方法和步骤

1．自顶向下设计法

2．自底向上设计法

3．由里向外设计法

4．混合策略设计法

6.3.2 局部视图设计

局部视图设计是根据系统的具体情况，在多层的数据流图中选择一个适当层次的数据流图，作为设计分 E-R 图的出发点，并让数据流图中的每一个部分都对应一个局部应用。选择好局部应用之后，就可以对每个局部应用逐一设计分 E-R 图了。局部 E-R 图的设计分为如下的几个步骤，如图 8-10 所示。

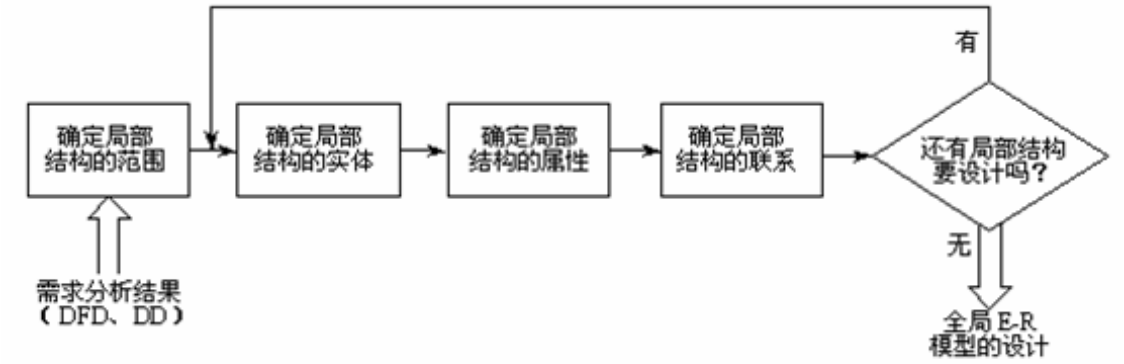


图 8-10 局部 E-R 模型设计

1．确定实体类型和属性

实体和属性之间没有严格的区别界限，但对于属性来讲，可以用下面的两条准则作为依据：

（1）作为属性必须是不可再分的数据项，也就是属性中不能再包含其他的属性。

（2）属性不能与其他实体之间具有联系。

2．确定实体间的联系

对一，一对多或多对多），接下来要确定哪些联系是有意义的，哪些联系是冗余的，并消除冗余的联系。所谓冗余的联系是指无意义的或可以从其他联系导出的联系。

3．画出局部 E-R 图

确定了实体及实体间的联系后，可用 E-R 图描述出来。形成局部 E-R 图之后，还必须返回去征求用户意见，使之如实地反映现实世界，同时还要进一步规范化，以求改进和完善。每个局部视图必须满足：

（1）对用户需求是完整的。

（2）所有实体、属性、联系都有惟一的名字。

（3）不允许有异名同义、同名异义的现象。

（4）无冗余的联系。

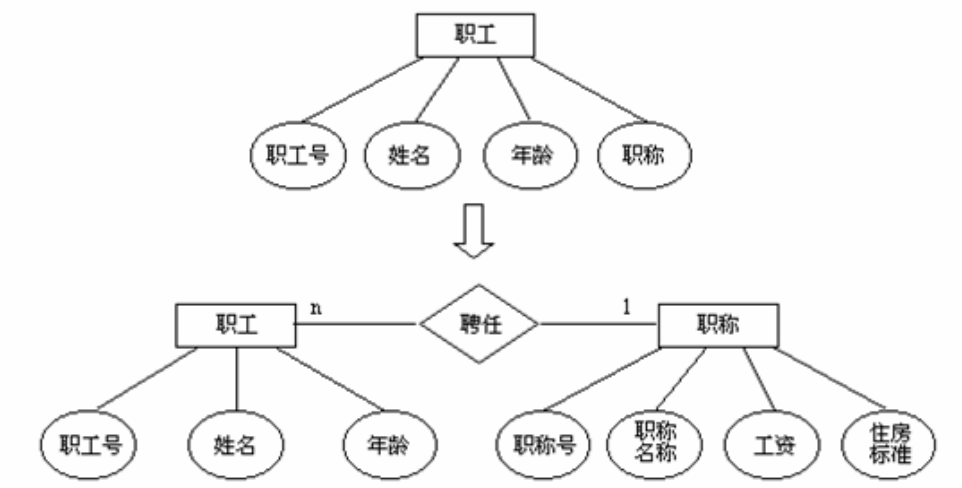


图 8-11　“职称”由属性变为实体示意图

6.3.3　视图的集成

各个局部视图建立好后，还需要对它们进行合并，集成为一个整体的数据概念结构，即总 E-R 图。集成局部 E-R 图型，设计全局 E-R 模型的步骤如图 8-12 所示。

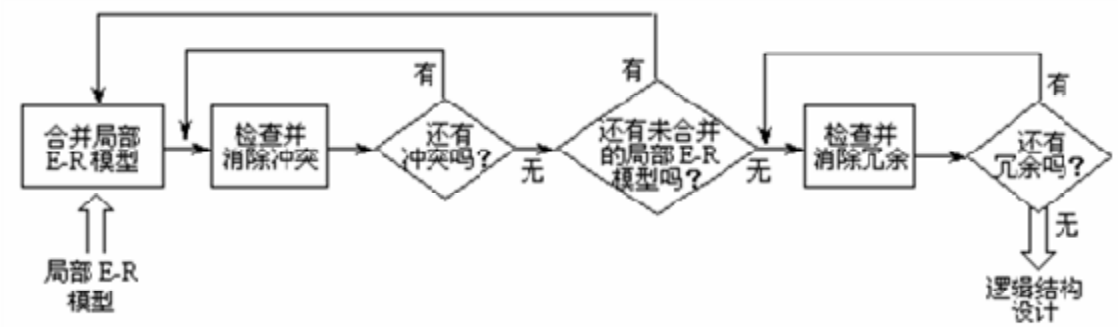


图 8-12　全局 E-R 模型设计

1．合并局部 E-R 图，生成初步 E-R 图

（1）属性冲突。

（2）命名冲突。

（3）结构冲突。

2．修改和重构初步 E-R 图，消除冗余，生成基本 E-R 图

（1）用分析的方法消除冗余。分析方法是消除冗余的主要方法。

（2）用规范化理论消除冗余。

6.3.4　概念结构设计实例

1．标识图书管理系统中的实体和属性

参照数据字典中对数据存储的描述，可初步确定三个实体的属性为：

读者：{卡号，姓名，性别，部门，类别、办卡日期，卡状态}

图书：{书号，书名，作者，价格，出版社，库存数量}

借还记录：{卡号，书名，借书日期，还书日期}

其中有下划线的属性为实体的码。

2．确定实体间的联系

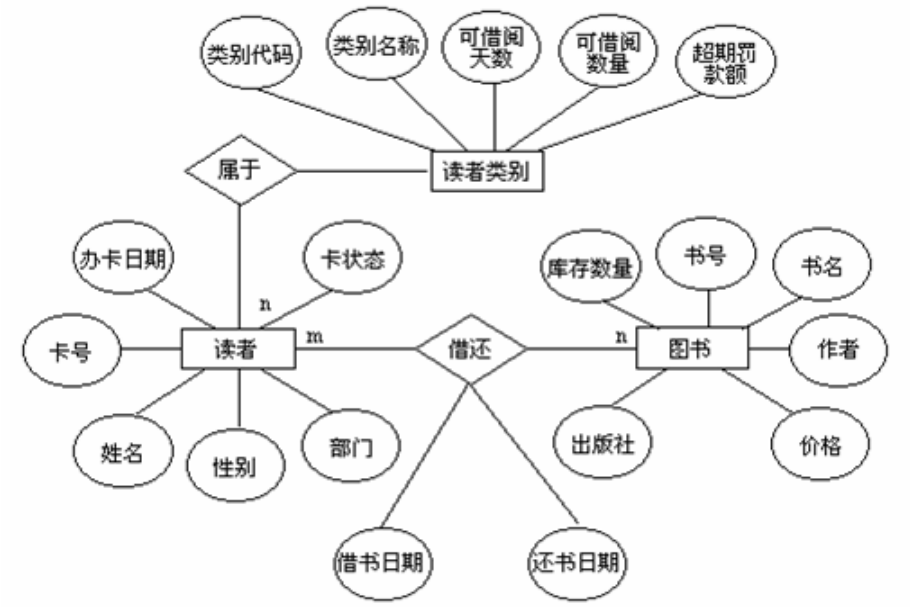


图 8-17　图书管理系统初步 E-R 图

6.4　逻辑结构设计

6.4.1　逻辑结构设计的任务和步骤

逻辑结构设计的主要目标是将概念结构转换为一个特定的 DBMS 可处理的数据模型和数据库模式。该模型必须满足数据库的存取、一致性及运行等各方面的用户需求。逻辑结构的设计过程如图 8-18 所示。

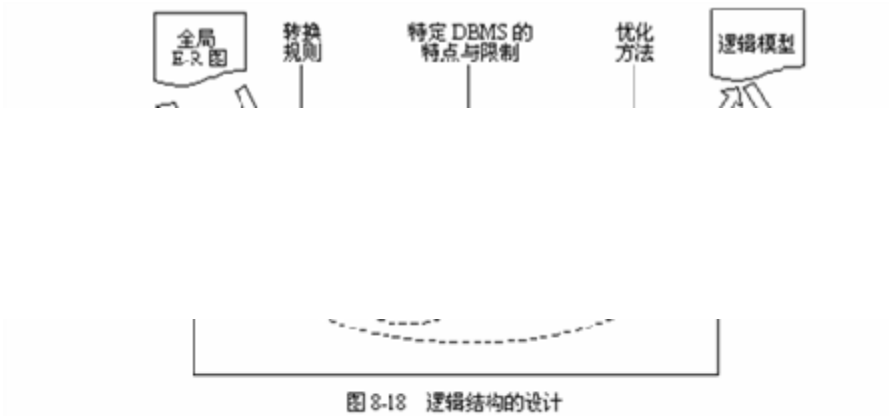


图 8-18　逻辑结构的设计

从图 8-18 中可以看出，概念模型向逻辑模型的转换过程分为 3 步进行：

（1）把概念模型转换为一般的数据模型。

（3）通过优化方法将其转化为优化的数据模型。

6.4.2 概念模型转换为一般的关系模型

1．实体的转换规则

将 E-R 图中的每一个常规实体转换为一个关系，实体的属性就是关系的属性，实体的码就是关系的码。

2．实体间联系的转换规则

（1）一个 1：1 联系可以转换为一个独立的关系模式，也可以与任意一端所对应的关系模式合并。

（2）一个 1 : n 联系可以转换为一个独立的关系模式，也可以与 n 端所对应的关系模式合并。

（3）一个 m : n 联系转换为一个关系模式。转换的方法为：与该联系相连的各实体的码以及联系本身的属性均转换为关系的属性，新关系的码为两个相连实体码的组合。

（4）三个或三个以上实体间的多元联系转换为一个关系模式。

3．关系合并规则

为了减少系统中的关系个数，如果两个关系模式具有相同的主码，可以考虑将它们合并为一个关系模式。合并的方法是将其中一个关系模式的全部属性加入到另一个关系模式中，然后去掉其中的同义属性，并适当调整属性的次序。

6.4.3 逻辑结构设计综合实例

下面仍以图书管理系统的基本 E-R 模型（图 8-17）为例，说明基本 E-R 模型转换成初始关系模型的规则：

（1）将图 8-17 中的实体转换成关系模式。

（2）将图 8-17 中的 1 : n 联系“属于”转换为关系模型。

（3）将图 8-17 中的 m : n 联系“借还”转换为关系模型。

（4）将具有相同码的关系合并。

6.4.4 将一般的关系模型转换为Oracle下的关系模型

下面就将图书管理系统中的关系设计成 Oracle 下相应的表，如下所示。

（2）DZCLASS（读者类别表）。

（3）BOOK（图书表）。

（4）BORROW（借还表）。

6.4.5 数据模型的优化

（1）确定各属性之间的数据依赖。

（2）对各个关系模式之间的数据依赖进行极小化处理，消除冗余的联系。

（3）判断每个关系的范式，根据实际需要确定最合适的范式。

（4）根据需求分析阶段得到的处理要求，分析这些模式是否适用于用户的应用环境，从而确定是否要对某些模式进行分解或合并。

（5）对关系模式进行必要的分解，以提高数据的操作效率和存储空间的利用率。

6.4.6 设计用户外模式

在定义外模式时可以考虑以下因素：

（1）使用更符合用户习惯的别名。

（2）对不同级别的用户定义不同的外模式，以保证数据的安全。

（3）简化用户对系统的使用。

6.5 数据库的物理设计

1．确定数据库的物理结构

（1）存储结构的设计。

1）顺序存储。

2）散列存储。

3）索引存储。

（2）存取方法设计。

（3）存放位置的设计。

2．评价物理结构

评价物理数据库的方法完全依赖于所选用的 DBMS，主要是从定量估算各种方案的存储空间、存取时间和维护代价入手，对估算结果进行权衡、比较，选择出一个较优的合理的物理结

6.6 数据库实施

（1）建立实际的数据库结构。

利用给定的 DBMS 所提供的命令，建立数据库的模式、外模式和内模式。对于关系数据库来讲，就是创建数据库、建立数据库中所包含的各个基本表、视图和索引等。

（2）将原始数据装入数据库。

装入数据的过程是非常复杂的。这是因为原始数据一般分散在企业各个不同的部门，而且它们的组织方式、结构和格式都与新设计的数据库系统中的数据有不同程度的区别。

6.7 数据库运行与维护

在数据库运行阶段，对数据库经常性的维护工作是由 DBA 完成的，它包括以下工作：

（1）数据库的转储和恢复。

（2）数据库安全性、完整性控制 DBA 必须对数据库的安全性和完整性控制负起责任。

（3）数据库性能的监督、分析和改进。

（4）数据库的重组织和重构造。

另外，数据库系统的应用环境是不断变化的，常常会出现一些新的应用，也会消除一些旧的应用，这将导致新实体的出现和旧实体的淘汰，同时原先实体的属性和实体间的联系也会发生变化。

小结 ：

本章介绍了数据库设计的全过程。设计一个数据库应用系统需要经历需求分析、概念设计、逻辑结构设计、物理设计、实施和运行维护六个阶段。

概念结构设计用于设计某个企业或组织所关心的信息结构，是对现实世界的第一层抽象，它独立于机器特点，独立于具体的数据库管理系统，它用 E-R 模型来描述。逻辑设计是将概念设计的结果 E-R 模型转换为数据的组织模型，对于关系数据库来说，是转换为关系表。

第7章　数据库优化

7.1　SQL优化

7.1.1　SQL优化流程概述

SQL 优化是数据库优化的一个方面，而且是最主要的一个方面，一般来说绝大部分的优化工作最终可以归结为 SQL 优化。SQL 优化的目的就是为了提高数据的访问速度，而提高数据访问速度的方法主要就是正确的使用索引。但是相反的一面是，使用索引未必能提高数据访问速度，这句话的意思包括两层含义：

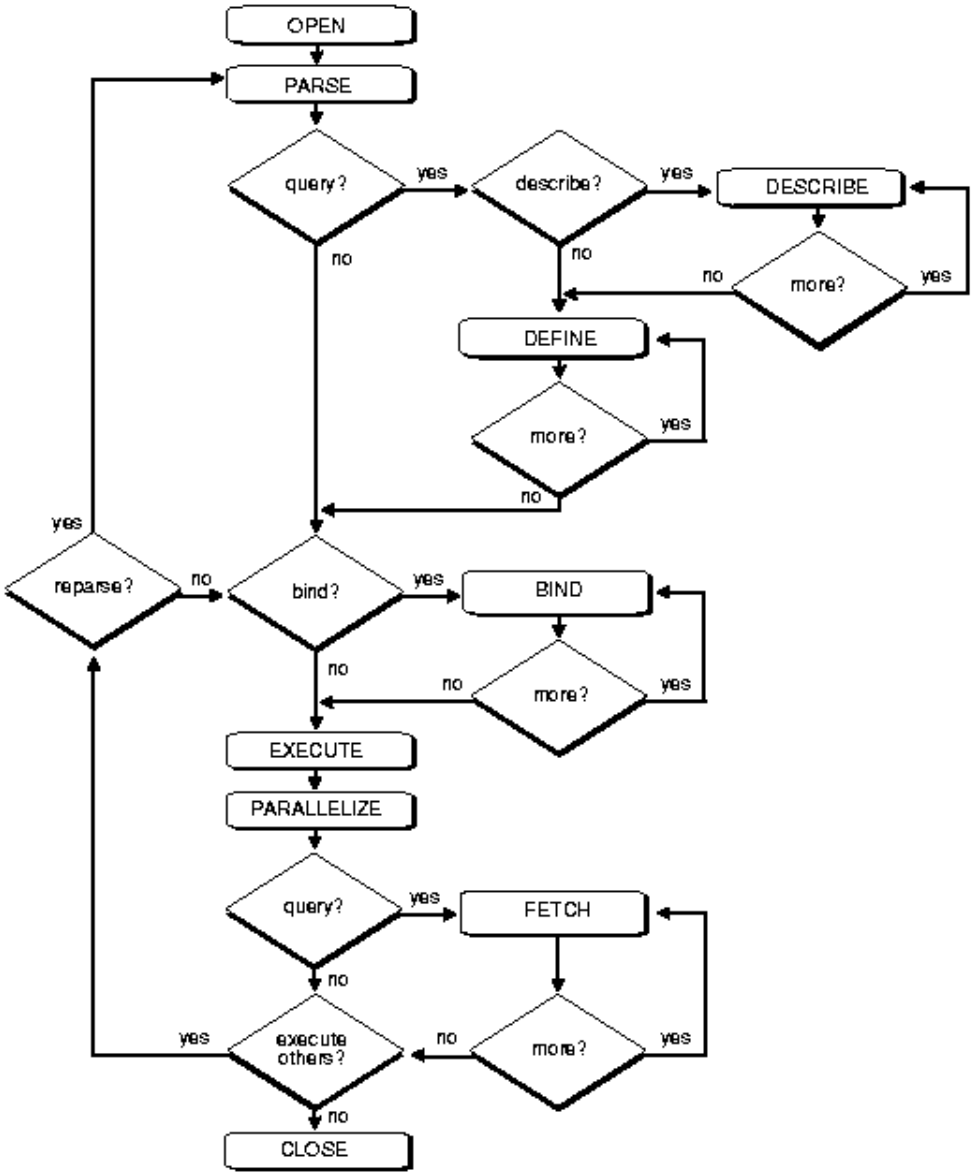
1．有时候使用索引，访问速度更慢；

2．索引建立不正确。

要做好 SQL 的优化工作，我们有必要了解数据库处理 SQL 的机理。

7.1.2　SQL语句的处理流程

按照 Oracle 官方文档 SQL 语句的处理流程总共有九步，如下图 7-1 所示。



从图上来看，具体的九个步骤是：

1．Create a Cursor 创建游标

游标是什么？实际上就是为了本次任务在内存中创建的一种数据类型，游标的名字可以理解为句柄或者指针，它指向这段内存。

由程序接口调用创建一个游标（cursor）。任何 SQL 语句都会创建它，特别在运行 DML 语句时，都是自动创建游标的，不需要开发人员干预。多数应用中，游标的创建是自动的。然而，在预编译程序(pro*c)中游标的创建，可能是隐含的，也可能显式的创建。在存储过程中也是这样的。

2．Parse the Statement 分析语句

在语法分析期间，SQL 语句从用户进程传送到 Oracle，SQL 语句经语法分析后，SQL 语句本身与分析的信息都被装入到共享 SQL 区。在该阶段中，可以解决许多类型的错误。语法分析分别执行下列操作：

（1）翻译 SQL 语句，验证它是合法的语句，即书写正确

（2）实现数据字典的查找，以验证是否符合表和列的定义

（3）在所要求的对象上获取语法分析锁，使得在语句的语法分析过程中不改变这些对象的定义

（4）验证为存取所涉及的模式对象所需的权限是否满足

（5）决定此语句最佳的执行计划

（6）将它装入共享 SQL 区

（7）对分布的语句来说，把语句的全部或部分路由到包含所涉及数据的远程节点

以上任何一步出现错误，都将导致语句报错，中止执行。

只有在共享池中不存在等价 SQL 语句的情况下，才对 SQL 语句作语法分析。在这种情况下，数据库内核重新为该语句分配新的共享 SQL 区，并对语句进行语法分析。进行语法分析需要耗费较多的资源，所以要尽量避免进行语法分析，这是优化的技巧之一。

语法分析阶段包含了不管此语句将执行多少次，而只需分析一次的处理要求。Oracle 只对每个 SQL 语句翻译一次，在以后再次执行该语句时，只要该语句还在共享 SQL 区中，就可以避免对该语句重新进行语法分析，也就是此时可以直接使用其对应的执行计划对数据进行存取。这主要是通过绑定变量(bind variable)实现的，也就是我们常说的共享 SQL，后面会给出共享 SQL 的概念。

虽然语法分析验证了 SQL 语句的正确性，但语法分析只能识别在 SQL 语句执行之前所能发现的错误(如书写错误、权限不足等)。因此，有些错误通过语法分析是抓不到的。例如，在数据转换中的错误或在数据中的错（如企图在主键中插入重复的值）以及死锁等均是只有在语句执行阶段期间才能遇到和报告的错误或情况。

3．Describe Results of a Query 描述查询的结果集

描述阶段只有在查询结果的各个列是未知时才需要；例如，当查询由用户交互地输入需要输出的列名。在这种情况要用描述阶段来决定查询结果的特征（数据类型，长度和名字）。

4．Define Output of a Query 定义查询的输出数据

在查询的定义阶段，你指定与查询出的列值对应的接收变量的位置、大小和数据类型，这样我们通过接收变量就可以得到查询结果。如果必要的话，Oracle 会自动实现数据类型的转换。这是将接收变量的类型与对应的列类型相比较决定的。

5．Bind Any Variables 绑定变量

此时，Oracle 知道了 SQL 语句的意思，但仍没有足够的信息用于执行该语句。Oracle 需要得到在语句中列出的所有变量的值。

此过程称之为将变量值捆绑进来。程序必须指出可以找到该数值的变量名（该变量被称为捆绑变量，变量名实质上是一个内存地址，相当于指针）。应用的最终用户可能并没有发觉他们正在指定捆绑变量，因为 Oracle 的程序可能只是简单地指示他们输入新的值，其实这一切都在程序中自动做了。

因为你指定了变量名，在你再次执行之前无须重新捆绑变量。你可以改变绑定变量的值，而 Oracle 在每次执行时，仅仅使用内存地址来查找此值。

6．Parallelize the Statement 并行执行语句

ORACLE 可以在 SELECT, INSERT, UPDATE, MERGE, DELETE 语句中执行相应并行查询操作，对于某些 DDL 操作，如创建索引、用子查询创建表、在分区表上的操作，也可以执行并行操作。并行化可以导致多个服务器进程(oracle server processes)为同一个 SQL 语句工作，使该 SQL 语句可以快速完成，但是会耗费更多的资源，所以除非很有必要，否则不要使用并行查询。

7．Run the Statement 运行语句

到了现在这个时候，Oracle 拥有所有需要的信息与资源，因此可以真正运行 SQL 语句了。如果该语句为 SELECT 查询或 INSERT 语句，则不需要锁定任何行，因为没有数据需要被改变。然而，如果语句为 UPDATE 或 DELETE 语句，则该语句影响的所有行都被锁定，防止该用户提交或回滚之前，别的用户对这些数据进行修改。这保证了数据的一致性。

对于某些语句，你可以指定执行的次数，这称为批处理(array processing)。指定执行 N 次，则绑定变量与定义变量被定义为大小为 N 的数组的开始位置，这种方法可以减少网络开销，也是优化的技巧之一。

在 fetch 阶段，行数据被取出来，每个后续的存取操作检索结果集中的下一行数据，直到最后一行被取出来。上面提到过，批量的 fetch 是优化的技巧之一。

9．Close the Cursor 关闭游标

SQL 语句处理的最后一个阶段就是关闭游标。

7.1.3 执行计划

上面的 SQL 执行步骤中，在分析语句部分讲到 Oracle 最终会生成执行计划。那么执行计划怎么理解呢？执行计划就是用户和数据库服务程序都能识别的执行 SQL 语句的顺序指令集合。汇编语句就是指令集合，可以这样理解，只是执行计划还处于二进制编码的上层，没有那么底层。

执行计划是 SQL 优化中最为复杂也是最为关键的部分，只有知道了 ORACLE 在内部到底是如何执行该 SQL 语句后，我们才能知道优化器选择的执行计划是否为最优的。所以我们面临的问题主要是：如何得到执行计划；如何分析执行计划，从而找出影响性能的主要问题。下面先从分析树型执行计划开始介绍，然后介绍如何得到执行计划，再介绍如何分析执行计划。

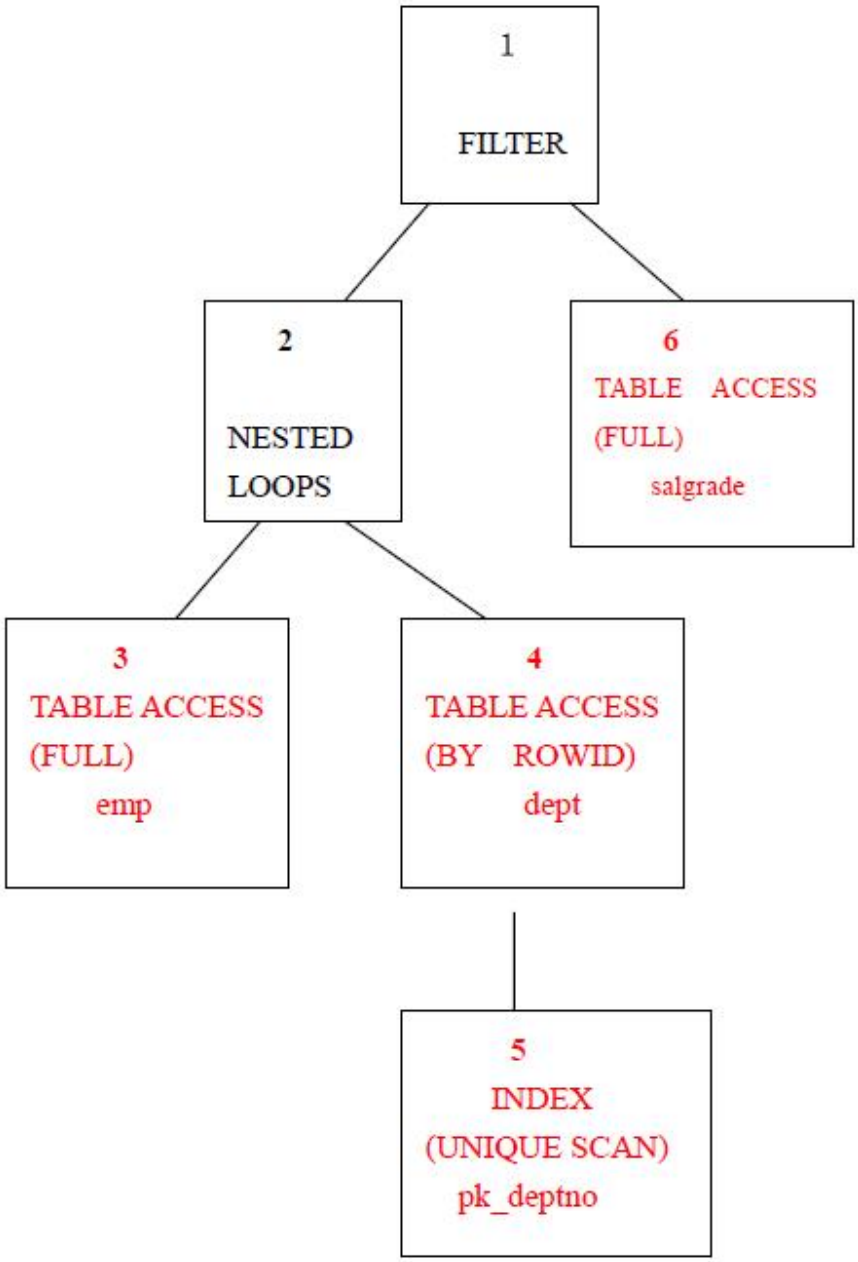
举例：

这个例子显示关于下面 SQL 语句的执行计划。

```
SELECT ename, job, sal, dname
FROM emp, dept
WHERE emp.deptno = derpt.deptno
AND NOT EXISTS
( SELECT *
FROM salgrade
WHERE emp.sal BETWEEN losal AND hisal );
```

此语句查询薪水不在任何建议薪水范围内的所有雇员的名字，工作，薪水和部门名。

下图 7-2 显示了一个执行计划的图形表示：



我们来解释一下图 7-2 各个步骤的含义：

（1）第 3 步和第 6 步分别的从 EMP 表和 SALGRADE 表读所有的行。

（2）第 5 步在 PK_DEPTNO 索引中查找由步骤 3 返回的每个 DEPTNO 值。它找出与 DEPT 表中相关联的那些行的 ROWID。

（3）第 4 步从 DEPT 表中检索出 ROWID 为第 5 步返回的那些行。

（4）第 2 步实现嵌套的循环操作(相当于 C 语句中的嵌套循环)，接收从第 3 步和第 4 步来

（5）第 1 步完成一个过滤器操作。它接收来自第 2 步和第 6 步的行源，消除掉第 2 步中来的，在第 6 步有相应行的那些行，并将来自第 2 步的剩下的行返回给发出语句的用户或应用。

要注意一点，执行计划中的步骤不是按照它们编号的顺序来实现的：Oracle 首先实现图 7-2 树结构图形里作为叶子出现的那些步骤(例如步骤 3、5、6)。由每一步返回的行称为它下一步骤的行源。然后 Oracle 实现父步骤。

举例来说，为了执行图 5-1 中的语句，Oracle 以下列顺序实现这些步骤：

（1）首先，Oracle 实现步骤 3，并一行一行地将结果行返回给第 2 步。

（2）对第 3 步返回的每一行，Oracle 实现这些步骤：

1）Oracle 实现步骤 5，并将结果 ROWID 返回给第 4 步。

2）Oracle 实现步骤 4，并将结果行返回给第 2 步。

3）Oracle 实现步骤 2，将接受来自第 3 步的一行和来自第 4 步的一行，并返回给第 1 步一行。

4）Oracle 实现步骤 6，如果有结果行的话，将它返回给第 1 步。

5）Oracle 实现步骤 1，如果从步骤 6 返回行，Oracle 将来自第 2 步的行返回给发出 SQL 语句的用户。

注意 Oracle 对由第 3 步返回的每一行实现步骤 5， 4， 2， 6 一次。

有时语句执行时，并不是象上面说的那样一步一步有先有后的进行，而是可能并行运行，如在实际环境中，3、5、4 步可能并行运行，以便取得更好的效率。

7.1.4 访问路径

前面讲执行计划的时候，我们知道执行计划的每个叶子就代表访问数据，相对应的就存在一个访问路径，也就是访问数据的方式。Oracle 中总共有 3 大类访问路径。

1．全表扫描（Full Table Scans, FTS）

为实现全表扫描，Oracle 读取表中所有的行，并检查每一行是否满足语句的 WHERE 限制条件。Oracle 顺序地读取分配给表的每个数据块，直到读到表的最高水线处(high water mark, HWM，标识表的最后一个数据块)。一个多块读操作可以使一次 I/O 能读取多块数据块(db_block_multiblock_read_count 参数设定)，而不是只读取一个数据块，这极大的减少了 I/O 总次数，提高了系统的吞吐量，所以利用多块读的方法可以十分高效地实现全表扫描，而且只有在全表扫描的情况下才能使用多块读操作。在这种访问模式下，每个数据块只被读 次 由于 HWM 标识最后 块被读入的数据 而 delete 操作不影响 HWM 值，所以

个表的所有数据被 delete 后，其全表扫描的时间不会有改善，一般我们需要使用 truncate 命令来使 HWM 值归为 0。幸运的是 oracle 10G 后，可以人工收缩 HWM 的值。

由 FTS 模式读入的数据被放到高速缓存的 Least Recently Used (LRU)列表的尾部，这样可以使其快速交换出内存，从而不使内存重要的数据被交换出内存。

使用 FTS 的前提条件：在较大的表上不建议使用全表扫描，除非取出数据的比较多，超过总量的 5% -- 10%，或你想使用并行查询功能时。

使用全表扫描的例子：

~~~~~~~~~~~~~~~~~~~~~~~~~~

```
SQL> explain plan for select * from dual;
Query Plan
----------------------------------------
SELECT STATEMENT [CHOOSE] Cost=
TABLE ACCESS FULL DUAL
```

2．通过 ROWID 的表存取（Table Access by ROWID 或 rowid lookup）

行的 ROWID 指出了该行所在的数据文件、数据块以及行在该块中的位置，所以通过 ROWID 来存取数据可以快速定位到目标数据上，是 Oracle 存取单行数据的最快方法。

为了通过 ROWID 存取表，Oracle 首先要获取被选择行的 ROWID，或者从语句的 WHERE 子句中得到，或者通过表的一个或多个索引的索引扫描得到。Oracle 然后以得到的 ROWID 为依据定位每个被选择的行。

这种存取方法不会用到多块读操作，一次 I/O 只能读取一个数据块。我们会经常在执行计划中看到该存取方法，如通过索引查询数据。

使用 ROWID 存取的方法：

```
SQL> explain plan for select * from dept where rowid = 'AAAAyGAADAAAAATAAF';
Query Plan
----------------------------------
SELECT STATEMENT [CHOOSE] Cost=1
TABLE ACCESS BY ROWID DEPT [ANALYZED]
```

3．索引扫描（Index Scan 或 index lookup）

我们先通过 index 查找到数据对应的 rowid 值(对于非唯一索引可能返回多个 rowid 值)，然后根据 rowid 直接从表中得到具体的数据，这种查找方式称为索引扫描或索引查找(index
~~~~~~~~~~~~~~~~~~~~~~~~~~

情况下该次 i/o 只会读取一个数据库块。

在索引中，除了存储每个索引的值外，索引还存储具有此值的行对应的 ROWID 值。索引扫描可以由 2 步组成：(1) 扫描索引得到对应的 rowid 值。 (2) 通过找到的 rowid 从表中读出具体的数据。每步都是单独的一次 I/O，但是对于索引，由于经常使用，绝大多数都已经 CACHE 到内存中，所以第 1 步的 I/O 经常是逻辑 I/O，即数据可以从内存中得到。但是对于第 2 步来说，如果表比较大，则其数据不可能全在内存中，所以其 I/O 很有可能是物理 I/O，这是一个机械操作，相对逻辑 I/O 来说，是极其费时间的。所以如果多大表进行索引扫描，取出的数据如果大于总量的 5% -- 10%，使用索引扫描会效率下降很多。

如下列所示：

```
SQL> explain plan for select empno, ename from emp where empno=10;
Query Plan
-----------------------------------
SELECT STATEMENT [CHOOSE] Cost=1
TABLE ACCESS BY ROWID EMP [ANALYZED]
INDEX UNIQUE SCAN EMP_I1
```

注意 TABLE ACCESS BY ROWID EMP 部分，这表明这不是通过 FTS 存取路径访问数据，而是通过 rowid lookup 存取路径访问数据的。在此例中，所需要的 rowid 是由于在索引查找 empno 列的值得到的，这种方式是 INDEX UNIQUE SCAN 查找，后面给予介绍，EMP_I1 为使用的进行索引查找的索引名字。

但是如果查询的数据能全在索引中找到，就可以避免进行第 2 步操作，避免了不必要的 I/O，此时即使通过索引扫描取出的数据比较多，效率还是很高的，因为这只会在索引中读取。所以上面我在介绍基于规则的优化器时，使用了 select count(id) from SWD_BILLDETAIL where cn <'6'，而没有使用 select count(cn) from SWD_BILLDETAIL where cn <'6'。因为在实际情况中，只查询被索引列的值的情况极为少，所以，如果我在查询中使用 count(cn)，则不具有代表性。

```
SQL> explain plan for select empno from emp where empno=10; -- 只查询 empno 列值
Query Plan
-----------------------------------
SELECT STATEMENT [CHOOSE] Cost=1
INDEX UNIQUE SCAN EMP_I1
```

进一步讲，如果 sql 语句中对索引列进行排序，因为索引已经预先排序好了，所以在执行计划中不需要再对索引列进行排序

where empno > 7876 order by empno;

Query Plan

SELECT STATEMENT [CHOOSE] Cost=1

TABLE ACCESS BY ROWID EMP [ANALYZED]

INDEX RANGE SCAN EMP_I1 [ANALYZED]

从这个例子中可以看到：因为索引是已经排序了的，所以将按照索引的顺序查询出符合条件的行，因此避免了进一步排序操作。

根据索引的类型与 where 限制条件的不同，有 4 种类型的索引扫描：

(1) 索引唯一扫描(index unique scan)

通过唯一索引查找一个数值经常返回单个 ROWID。如果该唯一索引有多个列组成(即组合索引)，则至少要有组合索引的引导列参与到该查询中，如创建一个索引：create index idx_test on emp(ename, deptno, loc)。则 select ename from emp where ename = 'JACK' and deptno = 'DEV'语句可以使用该索引。如果该语句只返回一行，则存取方法称为索引唯一扫描。而 select ename from emp where deptno = 'DEV'语句则不会使用该索引，因为 where 子句种没有引导列。如果存在 UNIQUE 或 PRIMARY KEY 约束（它保证了语句只存取单行）的话，Oracle 经常实现唯一性扫描。

使用唯一性约束的例子：

SQL> explain plan for

select empno,ename from emp where empno=10;

Query Plan

SELECT STATEMENT [CHOOSE] Cost=1

TABLE ACCESS BY ROWID EMP [ANALYZED]

INDEX UNIQUE SCAN EMP_I1

(2) 索引范围扫描(index range scan)

使用一个索引存取多行数据，同上面一样，如果索引是组合索引，如(1)所示，而且 select ename from emp where ename = 'JACK' and deptno = 'DEV'语句返回多行数据，虽然该语句还是使用该组合索引进行查询，可此时的存取方法称为索引范围扫描。在唯一索引上使用索引范围扫描的典型情况下是在谓词(where 限制条件)中使用了范围操作符(如>、<、<>、

使用索引范围扫描的例子：

SQL> explain plan for select empno,ename from emp

where empno > 7876 order by empno;

Query Plan

SELECT STATEMENT [CHOOSE] Cost=1

TABLE ACCESS BY ROWID EMP [ANALYZED]

INDEX RANGE SCAN EMP_I1 [ANALYZED]

在非唯一索引上，谓词 col = 5 可能返回多行数据，所以在非唯一索引上都使用索引范围扫描。

使用 index rang scan 的 3 种情况：

(a) 在唯一索引列上使用了 range 操作符(> < <> >= <= between)

(b) 在组合索引上，只使用部分列进行查询，导致查询出多行

(c) 对非唯一索引列上进行的任何查询。

(3) 索引全扫描(index full scan)

与全表扫描对应，也有相应的全索引扫描。在某些情况下，可能进行全索引扫描而不是范围扫描，需要注意的是全索引扫描只在 CBO 模式下才有效。CBO 根据统计数值得知进行全索引扫描比进行全表扫描更有效时，才进行全索引扫描，而且此时查询出的数据都必须从索引中可以直接得到。

全索引扫描的例子：

An Index full scan will not perform single block i/o's and so it may prove to be inefficient.

e.g.

Index BE_IX is a concatenated index on big_emp (empno, ename)

SQL> explain plan for select empno, ename from big_emp order by empno,ename;

Query Plan

SELECT STATEMENT [CHOOSE] Cost=26

INDEX FULL SCAN BE_IX [ANALYZED]

(4) 索引快速扫描(index fast full scan)

扫描索引中的所有的数据块，与 index full scan 很类似，但是一个显著的区别就是它不对查询出的数据进行排序，即数据不是以排序顺序被返回　在这种存取方法中，可以使用

多块读功能，也可以使用并行读入，以便获得最大吞吐量与缩短执行时间。

索引快速扫描的例子：

BE_IX 索引是一个多列索引：big_emp (empno,ename)

SQL> explain plan for select empno,ename from big_emp;

Query Plan

SELECT STATEMENT [CHOOSE] Cost=1

INDEX FAST FULL SCAN BE_IX [ANALYZED]

只选择多列索引的第 2 列：

SQL> explain plan for select ename from big_emp;

Query Plan

SELECT STATEMENT [CHOOSE] Cost=1

INDEX FAST FULL SCAN BE_IX [ANALYZED]

7.1.5 表连接

在多表查询的时候，主要使用表连接（Join）来获取数据。

Join 是一种试图将两个表结合在一起的谓词，一次只能连接 2 个表，表连接也可以被称为表关联。在后面的叙述中，我们将会使用”row source”来代替”表”，因为使用 row source 更严谨一些，并且将参与连接的 2 个 row source 分别称为 row source1 和 row source 2。Join 过程的各个步骤经常是串行操作，即使相关的 row source 可以被并行访问，即可以并行的读取做 join 连接的两个 row source 的数据，但是在将表中符合限制条件的数据读入到内存形成 row source 后，join 的其它步骤一般是串行的。有多种方法可以将 2 个表连接起来，当然每种方法都有自己的优缺点，每种连接类型只有在特定的条件下才会发挥出其最大优势。

row source(表)之间的连接顺序对于查询的效率有非常大的影响。通过首先存取特定的表，即将该表作为驱动表，这样可以先应用某些限制条件，从而得到一个较小的 row source，使连接的效率较高，这也就是我们常说的要先执行限制条件的原因。一般是在将表读入内存时，应用 where 子句中对该表的限制条件。

根据 2 个 row source 的连接条件的中操作符的不同，可以将连接分为等值连接(如 WHERE A.COL3 = B.COL4)、非等值连接(WHERE A.COL3 > B.COL4)、外连接(WHERE A.COL3 = B.COL4(+))。上面的各个连接的连接原理都基本一样，所以为了简单期间，下面以等值连接为例进行介绍　在后面的介绍中，都以：

FROM A, B
WHERE A.COL3 = B.COL4;
为例进行说明，假设 A 表为 Row Soruce1，则其对应的连接操作关联列为 COL 3；B 表为 Row Soruce2，则其对应的连接操作关联列为 COL 4。

目前为止，无论连接操作符如何，典型的连接类型共有 3 种：

1．排序--合并连接(Sort Merge Join (SMJ))

SMJ 的内部连接过程如下：

（1）首先生成 row source1 需要的数据，然后对这些数据按照连接操作关联列(如 A.col3)进行排序。

（2）随后生成 row source2 需要的数据，然后对这些数据按照与 sort source1 对应的连接操作关联列(如 B.col4)进行排序。

（3）最后两边已排序的行被放在一起执行合并操作，即将 2 个 row source 按照连接条件连接起来。

下面是连接步骤的图形表示：

```
  MERGE
  / \
  SORT SORT
  | |
  Row Source 1 Row Source 2
```

如果 row source 已经在连接关联列上被排序，则该连接操作就不需要再进行 sort 操作，这样可以大大提高这种连接操作的连接速度，因为排序是个极其费资源的操作，特别是对于较大的表。 预先排序的 row source 包括已经被索引的列(如 a.col3 或 b.col4 上有索引)或 row source 已经在前面的步骤中被排序了。尽管合并两个 row source 的过程是串行的，但是可以并行访问这两个 row source(如并行读入数据，并行排序).

SMJ 连接的例子：

```
SQL> explain plan for
select /*+ ordered */ e.deptno, d.deptno
from emp e, dept d
where e.deptno = d.deptno
order by e.deptno, d.deptno;
Query Plan
```

SELECT STATEMENT [CHOOSE] Cost=17

MERGE JOIN

SORT JOIN

TABLE ACCESS FULL EMP [ANALYZED]

SORT JOIN

TABLE ACCESS FULL DEPT [ANALYZED]

排序是一个费时、费资源的操作，特别对于大表。基于这个原因，SMJ 经常不是一个特别有效的连接方法，但是如果 2 个 row source 都已经预先排序，则这种连接方法的效率也是蛮高的。

2．嵌套循环(Nested Loops (NL))

这个连接方法有驱动表(外部表)的概念。其实，该连接过程就是一个 2 层嵌套循环，所以外层循环的次数越少越好，这也就是我们为什么将小表或返回较小 row source 的表作为驱动表(用于外层循环)的理论依据。但是这个理论只是一般指导原则，因为遵循这个理论并不能总保证使语句产生的 I/O 次数最少。有时不遵守这个理论依据，反而会获得更好的效率。如果使用这种方法，决定使用哪个表作为驱动表很重要。有时如果驱动表选择不正确，将会导致语句的性能很差、很差。

内部连接过程：

Row source1 的 Row 1 ---------- -- Probe -> Row source 2

Row source1 的 Row 2 ---------- -- Probe -> Row source 2

Row source1 的 Row 3 ---------- -- Probe -> Row source 2

…….

Row source1 的 Row n ---------- -- Probe -> Row source 2

从内部连接过程来看，需要用 row source1 中的每一行，去匹配 row source2 中的所有行，所以此时保持 row source1 尽可能的小与高效的访问 row source2(一般通过索引实现)是影响这个连接效率的关键问题。这只是理论指导原则，目的是使整个连接操作产生最少的物理 I/O 次数，而且如果遵守这个原则，一般也会使总的物理 I/O 数最少。但是如果不遵从这个指导原则，反而能用更少的物理 I/O 实现连接操作，那尽管违反指导原则吧！因为最少的物理 I/O 次数才是我们应该遵从的真正的指导原则，在后面的具体案例分析中就给出这样的例子。

在上面的连接过程中，我们称 Row source1 为驱动表或外部表。Row Source2 被称为被

在 NESTED LOOPS 连接中，Oracle 读取 row source1 中的每一行，然后在 row sourc2 中检查是否有匹配的行，所有被匹配的行都被放到结果集中，然后处理 row source1 中的下一行。这个过程一直继续，直到 row source1 中的所有行都被处理。这是从连接操作中可以得到第一个匹配行的最快的方法之一，这种类型的连接可以用在需要快速响应的语句中，以响应速度为主要目标。

如果 driving row source(外部表)比较小，并且在 inner row source(内部表)上有唯一索引，或有高选择性非唯一索引时，使用这种方法可以得到较好的效率。NESTED LOOPS 有其它连接方法没有的的一个优点是：可以先返回已经连接的行，而不必等待所有的连接操作处理完才返回数据，这可以实现快速的响应时间。

如果不使用并行操作，最好的驱动表是那些应用了 where 限制条件后，可以返回较少行数据的的表，所以大表也可能称为驱动表，关键看限制条件。对于并行查询，我们经常选择大表作为驱动表，因为大表可以充分利用并行功能。当然，有时对查询使用并行操作并不一定会比查询不使用并行操作效率高，因为最后可能每个表只有很少的行符合限制条件，而且还要看你的硬件配置是否可以支持并行(如是否有多个 CPU，多个硬盘控制器)，所以要具体问题具体对待。

NL 连接的例子：

```
SQL> explain plan for
select a.dname,b.sql
from dept a,emp b
where a.deptno = b.deptno;
Query Plan
------------------------
SELECT STATEMENT [CHOOSE] Cost=5
NESTED LOOPS
TABLE ACCESS FULL DEPT [ANALYZED]
TABLE ACCESS FULL EMP [ANALYZED]
```

3．哈希连接(Hash Join)

这种连接是在 oracle 7.3 以后引入的，从理论上来说比 NL 与 SMJ 更高效，而且只用在 CBO 优化器中。

较小的 row source 被用来构建 hash table 与 bitmap，第 2 个 row source 被用来被 hansed，并与第一个 row source 生成的 hash table 进行匹配，以便进行进一步的连接。Bitmap 被用来

比较大而不能全部容纳在内存中时，这种查找方法更为有用。这种连接方法也有 NL 连接中所谓的驱动表的概念，被构建为 hash table 与 bitmap 的表为驱动表，当被构建的 hash table 与 bitmap 能被容纳在内存中时，这种连接方式的效率极高。

HASH 连接的例子：

```
SQL> explain plan for
select /*+ use_hash(emp) */ empno
from emp, dept
where emp.deptno = dept.deptno;
Query Plan
---------------------------
SELECT STATEMENT [CHOOSE] Cost=3
HASH JOIN
TABLE ACCESS FULL DEPT
TABLE ACCESS FULL EMP
```

要使哈希连接有效，需要设置 HASH_JOIN_ENABLED=TRUE，缺省情况下该参数为 TRUE，另外，不要忘了还要设置 hash_area_size 参数，以使哈希连接高效运行，因为哈希连接会在该参数指定大小的内存中运行，过小的参数会使哈希连接的性能比其他连接方式还要低。

7.1.6 SQL语句日常书写规则

当优化器采用 CBO 以后，SQL 语句书写规则似乎变得不重要了。但其实不然，首先正确的书写规则基于我们对数据库的正确认识，你可以认为它就是一种技能。其次，尽管采用 CBO，但是如果书写时正确，那就减少了优化器工作，提高了效率。以下是常用的一些规则：

1．尽量避免对索引列进行计算

错误的例子：

```
WHERE sa*1.1>950
```

正确的例子：

```
WHERE sa>950/1.1
```

2．比较值与索引列的数据类型一致

emp： NUMBER 型

WHERE emp=’123’

正确的例子：

emp_char：CHAR 型

WHERE emp_char=’123’

3．避免使用 NULL

错误的例子：

WHERE comm IS NOT NULL

正确的例子：

WHERE comm>=0

4．对于复合索引，SQL 语句必须使用主索引列

错误的例子：

复合索引(deptno,job)

WHERE job=’MANAGER’

正确的例子：

复合索引(deptno,job)

WHERE deptno=20

5．ORDER BY 子句

规则：

（1）子句中，列的顺序与索引列的顺序一致。

（2）子句中，列应为非空列。

6．查询列与索引列次序（WHERE）一致

错误的例子：

SELECT empno,job FROM emp WHERE job=’MANAGER’ AND empno<100;

SELECT empno,job FROM emp WHERE empno<100 AND job='MANAGER';

7．尽量少用嵌套查询

错误的例子：

SELECT * FROM emp WHERE empno IN (SELECT empno FROM OnWork);

正确的例子：

SELECT emp.* FROM emp t1,OnWork t2 WHERE t1.empno = t2.empno;

8．多表连接时，使用表的别名来引用列

错误的例子：

SELECT ab02.aab001,ab01.aab004 FROM ab02 ,ab01
WHERE ab02.aab001 = ab01.aab001

正确的例子：

SELECT t1.aab004,t2.aab001 FROM ab02 t1,ab01 t2
WHERE t1.aab001=t2.aab001

9．用 NOT EXISTS 代替 NOT IN

错误的例子：

SELECT * FROM ab01
WHERE aab001 NOT IN
(SELECT aab001 FROM ab02 WHERE aae140='3');

正确的例子：

SELECT * FROM ab01 t
WHERE NOT EXISTS
(SELECT 1 FROM ab02 WHERE aab001=t.aab001 AND aae140='3');

10．用多表连接代替 EXISTS 子句

错误的例子：

SELECT * FROM ab01 t
WHERE EXISTS

正确的例子：

```
SELECT t1.* FROM ab01 t1,ab02 t2
WHERE t1.aab001 = t2.aab001 AND t2.aae140=‘3’;
```

11．少用 DISTINCT，用 EXISTS 代替

错误的例子：

```
SELECT DISTINCT ac01.aac016 aac016 FROM ac01,ac02
WHERE ac01.aac001=ac02.aac001
AND ac01.aab001 = '100659‘
AND NVL(ac01.aac016,'0') <> '107'
AND NVL(ac01.aac008,'0') = '1'
AND ac02.aae140 = '3' AND ac02.aac031 = '1';
```

正确的例子：

```
SELECT aac016 FROM ac01 t
WHERE aab001 = '100659'
AND NVL(aac016,'0') <> '107'
AND NVL(aac008,'0') = '1'
AND EXISTS (SELECT 1 FROM ac02 WHERE aac001=t.aac001
AND ac02.aae140 = '3' AND ac02.aac031 = '1');
```

12．使用 ROWID 提高检索速度

对 SELECT 得到的单行记录，需进行 DELETE、UPDATE 操作时，使用 ROWID 将会使效率大大提高。例如：

```
SELECT rowid INTO v_rowid FROM t1
WHERE con1 FOR UPDATE OF col2;
……
……
UPDATE t1 SET col2=……
WHERE rowid=v_rowid;
```

13．查询的 WHERE 过滤原则，

```
SELECT info
FROM taba a,tabb b,tabc c
WHERE a.acol between :alow and :ahigh
AND b.bcol between :blow and :bhigh
AND c.ccol between :clow and :chigh
AND a.key1 = b.key1
AND a.key2 = c.key2
```

7.1.7 SQL语句优化的基本原则

其实在概述部分已经大致上说，这里单独列出来，一方面通过前面的知识，更容易理解；另一方面也是为了强调一下。

SQL 语句优化的基本原则就是正确的使用索引。

不该使用索引的，不要使用索引。

因为有些时候我们查询的表本身尺寸很小，通过 Full Table Scan 可能一次 IO 就出来了，而如果使用索引，则至少要两次 IO。

要正确的使用索引。这点很好理解，因为盲目的建立索引，实际上很多时候用不上，徒增磁盘空间。

7.1.8 SQL语句的优化工具

1．Autotrace

Autotrace 允许在执行 SQL 的结果后面显示执行计划以及 SQL 语句的统计信息。它的使用很简单，就是在 sqlplus 中执行以下命令：

```
Sql> set autotrace on
Sql> select * from dual;
```

执行完语句后，会显示执行计划与统计信息。

这个工具有一个缺点，在用该方法查看执行时间较长的 sql 语句时，需要等待该语句执行成功后，才返回执行计划，使优化的周期大大增长。

如果不想执行语句而只是想得到执行计划可以采用：

```
Sql> set autotrace traceonly
```

这样，就只会列出执行计划，而不会真正的执行语句，大大减少了优化时间。但是又会存在

2．Explain plan

Explain plan 的使用很简单。如下：

Sql> explain plan for select ……

注意，用此方法时，并不执行 sql 语句，所以只会列出执行计划，不会列出统计信息，并且执行计划只存在 plan_table 中。所以该语句比起 set autotrace traceonly 可用性要差。需要用下面的命令格式化输出：

```
set linesize 150
set pagesize 500
col PLANLINE for a120
SELECT EXECORD EXEC_ORDER, PLANLINE
FROM (SELECT PLANLINE, ROWNUM EXECORD, ID, RID
FROM (SELECT PLANLINE, ID, RID, LEV
FROM (SELECT lpad(' ',2*(LEVEL),rpad(' ',80,' '))||
OPERATION||' '|| -- Operation
DECODE(OPTIONS,NULL,'','('||OPTIONS || ') ')|| -- Options
DECODE(OBJECT_OWNER,null,'','OF '"'|| OBJECT_OWNER||'.')|| -- Owner
DECODE(OBJECT_NAME,null,'',OBJECT_NAME|| '" ')|| -- Object Name
DECODE(OBJECT_TYPE,null,'','('||OBJECT_TYPE|| ') ')|| -- Object Type
DECODE(ID,0,'OPT_MODE:')|| -- Optimizer
DECODE(OPTIMIZER,null,'','ANALYZED','', OPTIMIZER)||
DECODE(NVL(COST,0)+NVL(CARDINALITY,0)+NVL(BYTES,0),
0,null,' (COST='||TO_CHAR(COST)||',CARD='||
TO_CHAR(CARDINALITY)||',BYTES='||TO_CHAR(BYTES)||')')
PLANLINE, ID, LEVEL LEV,
(SELECT MAX(ID)
FROM PLAN_TABLE PL2
CONNECT BY PRIOR ID = PARENT_ID
AND PRIOR STATEMENT_ID = STATEMENT_ID
START WITH ID = PL1.ID
AND STATEMENT_ID = PL1.STATEMENT_ID) RID
FROM PLAN_TABLE PL1
CONNECT BY PRIOR ID = PARENT_ID
AND PRIOR STATEMENT_ID = STATEMENT_ID
```

```
AND STATEMENT_ID = 'aaa')
ORDER BY RID, -LEV))
ORDER BY ID;
```

上面这 2 种方法只能为在本会话中正在运行的语句产生执行计划，即我们需要已经知道了哪条语句运行的效率很差，我们是有目的只对这条 SQL 语句去优化。其实，在很多情况下，我们只会听一个客户抱怨说现在系统运行很慢，而我们不知道是哪个 SQL 引起的。此时有许多现成的语句可以找出耗费资源比较多的语句。如下：

```
Sql>SELECT ADDRESS, substr(SQL_TEXT,1,20) Text,
    buffer_gets, executions, buffer_gets/executions AVG
    FROM v$sqlarea
    WHERE executions>0 AND buffer_gets > 100000
    ORDER BY 5;
```

从而对找出的语句进行进一步优化。当然我们还可以为一个正在运行的会话中运行的所有 SQL 语句生成执行计划，这需要对该会话进行跟踪，产生 trace 文件，然后对该文件用 tkprof 程序格式化一下，这种得到执行计划的方式很有用，因为它包含其它额外信息，如 SQL 语句执行的每个阶段(如 Parse、Execute、Fetch)分别耗费的各个资源情况(如 CPU、DISK、elapsed 等)。

3．dbms_system

使用 dbms_system 包可以跟踪另一个会话发出的 sql 语句，并记录所使用的执行计划，而且还提供其它对性能调整有用的信息。因其使用方式与上面 2 种方式有些不太一样。这种方法是对 SQL 进行调整比较有用的方式之一，有些情况下非它不可。

使用 dbms_system 前有必要设置以下参数：

timed_statistics：收集跟踪信息时，是否将收集时间信息，如果收集，

则可以知道一个 sql 的各个执行阶段耗费的时间情况

user_dump_dest：存放跟踪数据的文件的位置

max_dump_file_size：放跟踪数据的文件的最大值，防止由于无意的疏忽，

使跟踪数据的文件占用整个硬盘，影响系统的正常运行

（1）设置参数：

```
SQL> exec sys.dbms_system.set_bool_param_in_session( -
```

serial# => 3, -
parnam => 'timed_statistics', -
bval => true);

SQL> alter system set user_dump_dest='c:\temp';

SQL> exec sys.dbms_system.set_int_param_in_session(
sid => 8,
serial# => 3,
parnam => 'max_dump_file_size',
intval => 2147483647)

（2）启动跟踪功能

SQL> exec sys.dbms_system.set_sql_trace_in_session(8, 3, true);
注意，只有跟踪的 session 再次发出 sql 语句后，才会产生 trc 文件。

（3）让系统运行一段时间，以便可以收集到跟踪数据。

（4）关闭跟踪功能

SQL> exec sys.dbms_system.set_sql_trace_in_session(8,3,false);

6) 使用 tkprof 格式化 trace 文件中的数据

在命令提示符下，运行下面的命令：

tkprof dsdb2_ora_18468.trc dsdb2_trace.out SYS=NO EXPLAIN=SCOTT/TIGER

4．STA

在 10g 还可以使用 STA 来优化 SQL 语句。STA 的使用非常简单。如下：

（1）创建优化任务并执行：

```
DECLARE
    my_task_name VARCHAR2(30);
    my_sqltext CLOB;
BEGIN
    my_sqltext := 'select a.table_name user_tables a';
    my_task_name := DBMS_SQLTUNE.CREATE_TUNING_TASK(
```

```
        sql_text => my_sqltext,
        user_name => 'SA',
        scope => 'COMPREHENSIVE',
        time_limit => 60,
        task_name => 'TEST_sql_tuning_task',
        description => 'Task to tune a query on a specified PRODUCT');
        dbms_sqltune.Execute_tuning_task (task_name => 'TEST_sql_tuning_task');
END;
```

DBMS_SQLTUNE.CREATE_TUNING_TASK 就是用来创建优化任务的函数。其中，

sql_text 是需要优化的语句，

user_name 是该语句通过哪个用户执行，

scope 是优化范围（limited 或 comprehensive），

time_limit 优化过程的时间限制，

task_name 优化任务名称，

description 优化任务描述。

dbms_sqltune.Execute_tuning_task 是执行优化的函数。

如果需要再次执行它，则首先要使用 dbms_sqltune.drop_tuning_task 删除任务后再重建任务。

（2）查看优化建议结果：

```
SQL> set long 10000
SQL> set linesize 100
SQL> SELECT
DBMS_SQLTUNE.REPORT_TUNING_TASK('TEST_sql_tuning_task')
FROM DUAL;

DBMS_SQLTUNE.REPORT_TUNING_TAS
--------------------------------------------------------------------------------
GENERAL INFORMATION SECTION
--------------------------------------------------------------------------------
Tuning Task Name                 : TEST_sql_tuning_task
```

Scope : COMPREHENSIVE
Time Limit(seconds) : 60
Completion Status : COMPLETED
Started at : 11/24/2005 14:46:23
Completed at : 11/24/2005 14:46:24
Number of SQL Restructure Findings: 1

Schema Name: DEMO
SQL ID : f5k29d73zdpsd
SQL Text : select a.table_name, b.object_id from bigtab b, smalltab a

---

FINDINGS SECTION (1 finding)

1- Restructure SQL finding (see plan 1 in explain plans section)

An expensive cartesian product operation was found at line ID 1 of the execution plan.

Recommendation

- Consider removing the disconnected table or view from this statement or add a join condition which refers to it.

Rationale

A cartesian product should be avoided whenever possible because it is an expensive operation and might produce a large amount of data.

EXPLAIN PLANS SECTION

```
----------
Plan hash value: 3479921507

-------------------------------------------------------------------------------
| Id  | Operation                    | Name        | Rows  | Bytes | Cost (%CPU)| Time
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |             |  1474M|    32G|  4316K  (2)| 14:23:21
|   1 |  MERGE JOIN CARTESIAN        |             |  1474M|    32G|  4316K  (2)| 14:23:21
|   2 |   TABLE ACCESS FULL          | SMALLTAB    |  1223 | 23237 |    11   (0)| 00:00:01
|   3 |   BUFFER SORT                |             |  1205K|  5887K|  4316K  (2)| 14:23:21
|   4 |    TABLE ACCESS FULL         | BIGTAB      |  1205K|  5887K|  3530   (2)| 00:00:43
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
```

优化建议结果分为三部分，下面简单介绍一下每个部分。

（1）和所有 Oracle 的报告一样，第一部分总是基础信息。

```
GENERAL INFORMATION SECTION
-------------------------------------------------------------------------------
Tuning Task Name                 : TEST_sql_tuning_task
Tuning Task Owner                : DEMO
Scope                            : COMPREHENSIVE
Time Limit(seconds)              : 60
Completion Status                : COMPLETED
Started at                       : 11/24/2005 14:46:23
Completed at                     : 11/24/2005 14:46:24
Number of SQL Restructure Findings: 1

-------------------------------------------------------------------------------
Schema Name: DEMO
SQL ID       : f5k29d73zdpsd
SQL Text     : select a.table_name, b.object_id from bigtab b, smalltab a
```

这部分信息包括：任务名称、任务所有者、任务范围、任务执行时间限制（前面几个信息

要重新构造的问题语句数量。接下来就是 schema 名称、SQL ID 和 SQL 内容。

（2）优化器发现的需要优化的语句，在我们的例子中，肯定只有一条：

FINDINGS SECTION (1 finding)

1- Restructure SQL finding (see plan 1 in explain plans section)

An expensive cartesian product operation was found at line ID 1 of the execution plan.

Recommendation

- Consider removing the disconnected table or view from this statement or add a join condition which refers to it.

Rationale

A cartesian product should be avoided whenever possible because it is an expensive operation and might produce a large amount of data.

这部分内容可以细分为：

1）是在查询计划中发现的问题：在语句 1 的查询计划中发现有很大的笛卡尔积操作（显然，我们的语句是一个外连接操作，做的就是迪卡尔积操作）；

2）是针对问题给出的建议：考虑将不需要做连接查询的表或视图去掉，或者增加一个连接条件；

3）导致问题的根本原因：由于迪卡尔积是一个会产生很大数据量的消耗资源的操作，所以需要尽量避免迪卡尔积操作。

（3）SQL 语句的查询计划：

EXPLAIN PLANS SECTION

1- Original

Plan hash value: 3479921507

```
-------------------------------------------------------------------------------
| Id  | Operation                    | Name        | Rows  | Bytes | Cost (%CPU)| Time
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |             |  1474M|    32G|  4316K  (2)| 14:23:21
|   1 |  MERGE JOIN CARTESIAN        |             |  1474M|    32G|  4316K  (2)| 14:23:21
|   2 |   TABLE ACCESS FULL          | SMALLTAB    |  1223 | 23237 |    11   (0)| 00:00:01
|   3 |   BUFFER SORT                |             |  1205K|  5887K|  4316K  (2)| 14:23:21
|   4 |    TABLE ACCESS FULL         | BIGTAB      |  1205K|  5887K|  3530   (2)| 00:00:43
-------------------------------------------------------------------------------
```

这里给出的是语句优化前的原始查询计划。如果优化器认为这条语句需要重写，它还会给出重写后的查询计划。

7.1.9 稳定执行计划

1．Hints

基于代价的优化器是很聪明的，在绝大多数情况下它会选择正确的优化器。但有时它也聪明反被聪明误，选择了很差的执行计划，使某个语句的执行变得奇慢无比。此时就需要 DBA 进行人为的干预，告诉优化器使用我们指定的存取路径或连接类型生成执行计划，从而使语句高效的运行。例如，如果我们认为对于一个特定的语句，执行全表扫描要比执行索引扫描更有效，则我们就可以指示优化器使用全表扫描。在 ORACLE 中，是通过为语句添加 hints(提示)来实现干预优化器优化的目的。

Hints 是 oracle 提供的一种机制，用来告诉优化器按照我们的告诉它的方式生成执行计划。我们可以用 Hints 来实现：

1) 使用的优化器的类型

2) 基于代价的优化器的优化目标，是 all_rows 还是 first_rows。

3) 表的访问路径，是全表扫描，还是索引扫描，还是直接利用 rowid。

5) 表之间的连接顺序

6) 语句的并行程度

我们可以使用注释(comment)来为一个语句添加 hints，一个语句块只能有一个注释，而且注释只能放在 SELECT, UPDATE, or DELETE 关键字的后面。语法如下：

{DELETE|INSERT|SELECT|UPDATE} /*+ hint [text] [hint[text]]*/ ……

注意：

1) DELETE、INSERT、SELECT 和 UPDATE 是标识一个语句块开始的关键字，包含提示的注释只能出现在这些关键字的后面，否则提示无效。

2) “+”号表示该注释是一个 hints，该加号必须立即跟在”/*”的后面，中间不能有空格。

3) hint 是下面介绍的具体提示之一，如果包含多个提示，则每个提示之间需要用一个或多个空格隔开。

4) text 是其它说明 hint 的注释性文本

如果你没有正确的指定 hints，Oracle 将忽略该 hints，并且不会给出任何错误。

常用的 Hint：

（1）指示优化器的方法与目标的 hints：

1）ALL_ROWS -- 基于代价的优化器，以吞吐量为目标

2）FIRST_ROWS(n) -- 基于代价的优化器，以响应时间为目标

3）CHOOSE -- 根据是否有统计信息，选择不同的优化器

4）RULE -- 使用基于规则的优化器

例子：

```
SELECT /*+ FIRST_ROWS(10) */ employee_id, last_name, salary, job_id
FROM employees
WHERE department_id = 20;

SELECT /*+ CHOOSE */ employee_id, last_name, salary, job_id
FROM employees
WHERE employee_id = 7566;

SELECT /*+ RULE */ employee_id, last_name, salary, job_id
FROM employees
```

（2）指示存储路径的 hints：

1）FULL /*+ FULL (table) */ 指定该表使用全表扫描

2）ROWID /*+ ROWID (table) */ 指定对该表使用 rowid 存取方法，该提示用的较少

3）INDEX /*+ INDEX (table [index]) */ 使用该表上指定的索引对表进行索引扫描

4）INDEX_FFS /*+ INDEX_FFS (table [index]) */ 使用快速全表扫描

5）NO_INDEX /*+ NO_INDEX (table [index]) */

不使用该表上指定的索引进行存取，仍然可以使用其它的索引进行索引扫描

例子：

```
SELECT /*+ FULL(e) */ employee_id, last_name
FROM employees e
WHERE last_name LIKE :b1;

SELECT /*+ROWID(employees)*/ *
FROM employees
WHERE rowid > 'AAAAtkAABAAAFNTAAA' AND employee_id = 155;

SELECT /*+ INDEX(A sex_index) use sex_index because there are few
male patients */ A.name, A.height, A.weight
FROM patients A
WHERE A.sex = ’m’;

SELECT /*+NO_INDEX(employees emp_empid)*/ employee_id
FROM employees
WHERE employee_id > 200;
```

（3）指示连接顺序的 hints:

1）ORDERED /*+ ORDERED */ 按 from 字句中表的顺序从左到右的连接

2）STAR /*+ STAR */ 指示优化器使用星型查询

例子：

```
SELECT /*+ORDERED */ o.order_id, c.customer_id, l.unit_price * l.quantity
FROM customers c, order_items l, orders o
WHERE c.cust_last_name = :b1
```

AND o.order_id = l.order_id;

（4）指示连接类型的 hints：

1）USE_NL /*+ USE_NL (table [,table, ...]) */ 使用嵌套连接

2）USE_MERGE /*+ USE_MERGE (table [,table, ...]) */ 使用排序- -合并连接

3）USE_HASH /*+ USE_HASH (table [,table, ...]) */ 使用 HASH 连接

注意：如果表有 alias(别名)，则上面的 table 指的是表的别名，而不是真实的表名。

例子：

```
select /*+ USE_NL (A C)*/ A.col4
from C , A , B
where C.col3 = 5 and A.col1 = B.col1 and A.col2 = C.col2
and B.col3 = 10;
```

注意：

对于 USE_NL 与 USE_HASH 提示，建议同 ORDERED 提示一起使用，否则不容易指定那个表为驱动表。

2．Stored Outlines

Stored Outlines 是从 8i 开始 Oracle 提供的一个新对象，用来稳定 SQL 的执行计划，它实际上就是由一系列的 Hints 组成。

Stored Outlines 的创建有两种方式：

系统自动创建

1）指定一个分类 category，开始监控用户的 Hints：

```
alter session set create_stored_outlines = demo;
```

2）执行 SQL 语句，自动创建 Stored Outlines：

```
select /*+ and_equal(so_demo, sd_i1, sd_i2) */ v1
from so_demo
where n1 = 1
and n2 = 2;
```

3）停止监控：

```
alter session set create_stored_outlines = false;
```

用户自定义：

```
Sql> create or replace outline so_fix
```

```
select /*+ and_equal(so_demo, sd_i1, sd_i2) */ v1
from so_demo
where n1 = 1
and n2 = 2;
```

Stored Outlines 就是一系列有系统自动创建的 Hints。通过以下方法可以看到这些 Hints：

```
Sql> select name, category, used, sql_text
from user_outines
where category = 'DEMO';

NAME CATEGORY USED
------------------------------ ------------------------------ -------
SQL_TEXT
--------------------------------------------------------------------------------
SYS_OUTLINE_020503165427311 DEMO UNUSED
SELECT V1 FROM SO_DEMO WHERE N1 = :b1 AND N2 = :b2

Sql> select name, stage, hint
from user_outline_hints
where name = ' SYS_OUTLINE_020503165427311';

NAME STAGE HINT
------------------------------ ---------- ------------------------------
SYS_OUTLINE_020503165427311 3 NO_EXPAND
SYS_OUTLINE_020503165427311 3 ORDERED
SYS_OUTLINE_020503165427311 3 NO_FACT(SO_DEMO)
SYS_OUTLINE_020503165427311 3 FULL(SO_DEMO)
SYS_OUTLINE_020503165427311 2 NOREWRITE
SYS_OUTLINE_020503165427311 1 NOREWRITE
```

建好 Stored Outlines，就可以启用它。如下：

```
Sql> alter session set use_stored_outline = demo;
```

而用户在发出上述的 SQL 语句时，执行计划将不会改变。

存储概要有着巨大的好处。当你不能修改源代码或者索引策略时，存储概要是令第三方的

1．调查与初步分析用户的需求，确定系统的边界

（1）首先调查组织机构情况。

（2）然后调查各部门的业务活动情况。

（3）在熟悉了业务活动的基础上，协助用户明确对新系统的各种要求，包括信息要求、处理要求、安全性与完整性要求，这是调查的又一个重点。

（4）最后对前面调查的结果进行初步分析，确定新系统的边界，确定哪些功能由计算机完成或将来由计算机完成，哪些活动由人工完成。

2．分析和表达用户的需求

（1）数据流图。

数据流图（Data Flow Diagram，简称 DFD）是一种最常用的结构化分析工具，它用图形的方式来表达数据处理系统中信息的变换和传递过程。如图 8-4 所示，数据流图有 4 种基本符号。

（2）数据字典。

1）数据项条目：数据项是不可再分的数据单位，它直接反映事物的某一特征。

2）数据结构条目：反映了数据之间的组合关系。

3）数据流条目：数据流是数据结构在系统内传输的路径。

4）数据文件条目：数据文件是数据项停留或保存的地方，也是数据流的来源和去向之一。

5）处理过程条目。

7.1.10 SQL语句的优化步骤

通过上面的说明，我们现在大致可以总结 SQL 语句优化的一般步骤：

1．收集对象的统计信息

这个步骤在上面没有提到，这里简单说一下。

在基于性能的优化器下，Oracle 如何知道哪种执行计划的效率高？答案是根据查询对象的统计信息。所以，要得到高效的执行计划，首要条件是保证查询对象的统计信息准确。

如何保证呢？有三种方法：

系统自动统计：

自动统计是 10g 才引入的新特性，只需要设置参数 TIME_STATISTICS。如下：

DBMS_STATS 包：

```
Sql> DBMS_STATS.GATHER_TABLE_STATS (
    Ownname => 'username',
    Tabname => 'tablename',
    method_opt => 'for all indexed columns',
    cascade => TRUE,);
```

Analyze 命令：

```
Sql> analyze table tablename compute statistics for all;
```

2．察看 SQL 语句的执行计划：

如何察看？可以使用 7.1.8 小节所列出的工具察看。

3．根据执行计划，察看是否用到索引，然后相应的增加索引。

4．多 CPU 的情况下，且在机器性能允许的情况下，可以考虑使用并行查询。

7.2 DB优化

7.2.1 DB优化流程概述

DB 优化是数据库优化工作的重要方面，它和 SQL 语句优化出发点是不同的。SQL 优化的出发点是提高 SQL 的查询速度，而 DB 优化的出发点是为了减少资源竞争，从 CPU、内存、IO 到网络。这个也很好理解，因为 DB 是一个系统软件，而且通常 DB 服务器的资源只供 DB 使用，所以，有必要减少这些资源的竞争。

7.2.2 OS性能收集工具

1．Top

top 命令是 Linux 下常用的性能分析工具，能够实时显示系统中各个进程的资源占用状况，类似于 Windows 的任务管理器。

```
top - 01:06:48 up  1:22,  1 user,  load average: 0.06, 0.60, 0.48
Tasks:  29 total,   1 running,  28 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.3% us,  1.0% sy,  0.0% ni, 98.7% id,  0.0% wa,  0.0% hi,  0.0% si
Mem:    191272k total,   173656k used,    17616k free,    22052k buffers
Swap:   192772k total,        0k used,   192772k free,   123988k cached

  PID  USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 1379  root      16   0  7976 2456 1980 S  0.7  1.3   0:11.03 sshd
```

```
1    root     16   0  1992   632   544 S   0.0   0.3    0:00.90 init
2    root     34  19     0     0     0 S   0.0   0.0    0:00.00 ksoftirqd/0
3    root     RT   0     0     0     0 S   0.0   0.0    0:00.00 watchdog/0
```

2．Vmstat

vmstat 是用来实时查看内存使用情况，反映的情况比用 top 直观一些。如果直接使用，只能得到当前的情况，最好用个时间间隔来采集。用法是：

vmstat T

其中 T 用具体的时间表示，单位是秒。

例如：vmstat 5 表是每隔 5 秒采集一次。

```
procs -----------memory---------- ---swap-- -----io---- --system-- ----cpu----
r b    swpd    free    buff cache    si    so    bi    bo    in    cs us sy id wa
1 0     144 186164 105252 2386848     0     0    18   166    83     2 48 21 31 0
2 0     144 189620 105252 2386848     0     0     0   177 1039 1210 34 10 56 0
0 0     144 214324 105252 2386848     0     0     0    10 1071   670 32 5 63 0
0 0     144 202212 105252 2386848     0     0     0   189 1035   558 20 3 77 0
2 0     144 158772 105252 2386848     0     0     0   203 1065 2832 70 14 15 0
```

3．Iostat

iostat 工具将对系统的磁盘操作活动进行监视。它的特点是汇报磁盘活动统计情况，同时也会汇报出 CPU 使用情况。同 vmstat 一样，iostat 也有一个弱点，就是它不能对某个进程进行深入分析，仅对系统的整体情况进行分析。

```
tty:         tin          tout     avg-cpu:  % user     % sys     % idle   % iowait
                0.0           0.8                8.4       2.6       88.5       0.5
                0.0          80.2                4.5       3.0       92.1       0.5
                0.0          40.5                7.0       4.0       89.0       0.0
                0.0          40.5                9.0       2.5       88.5       0.0
                0.0          40.5                7.5       1.0       91.5       0.0
                0.0          40.5               10.0       3.5       80.5       6.0
```

```
r/s    w/s    kr/s    kw/s wait actv wsvc_t asvc_t  %w  %b device
0.2    0.2     0.0     1.6  0.0  0.0    0.0    8.7   0   0 c1t0d0
0.0    0.0     0.0     0.0  0.0  0.0    0.0    0.0   0   0 c0t0d0
0.0    1.6     0.0    12.8  0.0  0.0    0.0    0.4   0   0 c3t50060E8000542270d4
```

299.6 0.0 2995.6 0.0 0.0 1.0 0.0 3.3 0 99 rmt/0

4．Netstat

netstat 命令是一个监控 TCP/IP 网络的非常有用的工具，它可以显示路由表、实际的网络连接以及每一个网络接口设备的状态信息，在计算机上执行 netstat 后，其输出结果为：

Active Internet connections (w/o servers)

Proto Recv-Q Send-Q Local Address Foreign Address State

tcp 0 2 210.34.6.89:telnet 210.34.6.96:2873 ESTABLISHED

tcp 296 0 210.34.6.89:1165 210.34.6.84:netbios-ssn ESTABLISHED

tcp 0 0 localhost.localdom:9001 localhost.localdom:1162 ESTABLISHED

tcp 0 0 localhost.localdom:1162 localhost.localdom:9001 ESTABLISHED

tcp 0 80 210.34.6.89:1161 210.34.6.10:netbios-ssn CLOSE

Active UNIX domain sockets (w/o servers)

Proto RefCnt Flags Type State I-Node Path

unix 1 [ ] STREAM CONNECTED 16178 @000000dd

unix 1 [ ] STREAM CONNECTED 16176 @000000dc

unix 9 [ ] DGRAM 5292 /dev/log

unix 1 [ ] STREAM CONNECTED 16182 @000000df

7.2.3 性能视图

1．V$sysstat

记录的是数据库系统的统计信息。 这些信息对于数据库诊断来说及其重要。

2．V$sesstat

记录 session 从 login 到 logout 的详细资源使用统计。

3．V$process

记录所有服务器进程的信息。

4．V$lock

记录数据库中当前存在的所有对象锁。

5．V$latch

记录数据库中所有内存锁的统计信息。

6．V$session_longops

7．V$filestat

记录所有数据文件的 IO 情况。

8．V$tempstat

记录所有临时文件的 IO 情况。

7.2.4 Statspack

Statspack 是 Oracle 提供的比较全面的而且比较成熟和可靠的诊断数据库性能的工具。它的使用非常简单，先使用脚本 spcreate.sql 安装 statspack 工具，然后使用脚本 spreport.sql 脚本生成 statspack 报告。

在读报告的时候，我们首先需要看清楚，留意 3 个内容，这份报告所对应的数据库版本，cluster 方式，以及报告的时间段。尤其需要注意的就是时间段，脱离了时间段的 statspck 将是毫无意义的，甚至会得出错误的结果。

1．报表头信息

报表头信息，数据库实例相关信息，包括数据库名称、ID、版本号及主机明等信息。另外，重点还需要关注一下报告产生的时间跨度（在这里是 14 分钟），以及并发数（在这里是 272）。

```
DB Name         DB Id     Instance     Inst Num Release     Cluster Host
------------ ----------- ------------ -------- ----------- ------- ------------
ORA92         1924035339 ora92               1 9.2.0.6.0   NO      jsdxh_db02

              Snap Id     Snap Time      Sessions Curs/Sess Comment
            --------- ------------------ -------- --------- -------------------
Begin Snap:        13 14-Jul-07 00:18:52      274  55,345.0
  End Snap:        14 14-Jul-07 00:32:55      272  55,823.8
   Elapsed:               14.05 (mins)

Cache Sizes (end)
~~~~~~~~~~~~~~~~~
               Buffer Cache:     5,120M      Std Block Size:          8K
           Shared Pool Size:       400M          Log Buffer:      2,048K
```

2．实例负载信息

这部分主要是实例的负载信息，主要有以下一些参数：

（1）Redo size:每秒产生的日志大小(单位字节)，可标志数据变更频率，数据库任务的繁重与否。

（2）Logical reads:平决每秒产生的逻辑读的 block 数。Logical Reads= Consistent Gets + DB Block Gets

（3）Block changes:每秒 block 变化数量，数据库事物带来改变的块数量。

（4）Physical reads:平均每秒数据库从磁盘读取的 block 数。

（5）Physical writes:平均每秒数据库写磁盘的 block 数。

（6）User calls:每秒用户调用次数。

（7）Parses:每秒解析次数，包括 fast parse，soft parse 和 hard parse 三种数量的综合。

（8）Sorts:每秒产生的排序次数。

（9）Logons:每秒登陆的次数。

（10）Executes:每秒执行次数。

（11）Transactions:每秒产生的事务数，反映数据库任务繁重与否。

（12）% Blocks changed per Read：在每一次逻辑读中更改的块的百分比。

（13）Rollback per transaction %:看回滚率是不是很高，因为回滚很耗资源 ,如果回滚率过高,可能说明你的数据库经历了太多的无效操作 ,过多的回滚可能还会带来 Undo Block 的竞争。该参数计算公式如下: Round(User rollbacks / (user commits + user rollbacks) ,4)* 100% 。

（14）Recursive Call %:递归调用的百分比，如果有很多 PL/SQL，那么这个值就会比较高。

（15）Rows per Sort:平均每次排序操作的行数。

Load Profile

~~	Per Second	Per Transaction
Redo size:	422,086.46	4,706.23
Logical reads:	23,200.54	258.68
Block changes:	3,080.59	34.35
Physical reads:	31.46	0.35
Physical writes:	104.38	1.16
User calls:	409.32	4.56
Parses:	227.20	2.53

Sorts:		213.87	2.38
Logons:		0.85	0.01
Executes:		1,191.32	13.28
Transactions:		89.69	
% Blocks changed per Read:	13.28	Recursive Call %:	80.21
Rollback per transaction %:	0.03	Rows per Sort:	2.84

3．实例有效性信息

通过实例有效性统计数据，我们可以获得大概的一个整体印象，主要有以下一些参数：

（1）Buffer Nowait %：在缓冲区中获取 Buffer 的未等待比率。Buffer Nowait 的这个值一般需要大于 99%。否则可能存在争用，可以在后面的等待事件中进一步确认。

（2）Redo NoWait %：在 Redo 缓冲区获取 Buffer 空间的未等待比率。当 redo buffer 达到 1M 时，就需要写到 redo log 文件，所以一般当 redo buffer 设置超过 1M，不太可能存在等待 buffer 空间分配的情况。当前，一般设置为 2M 的 redo buffer，对于内存总量来说，应该不是一个太大的值。

（3）Buffer Hit %：数据块在数据缓冲区中的命中率，通常应在 95%以上。否则，小于 95%，需要调整重要的参数，小于 90%可能是要加 db_cache_size。

（4）In-memory Sort %：在内存中的排序率。如果低于 95%，可以通过适当调大初始化参数 PGA_AGGREGATE_TARGET 或者 SORT_AREA_SIZE 来解决。

（5）Soft Parse %：sql 在共享区的命中率，小于<95%,需要考虑绑定，如果低于 80%，那么就可以认为 sql 基本没有被重用。

（6）Execute to Parse %：一个语句执行和分析了多少次的度量。该值<0 通常说明 shared pool 设置或者语句效率存在问题，造成反复解析，reparse 可能较严重,或者是可能同 snapshot 有关，通常说明数据库性能存在问题。

（7）Latch Hit %：要确保>99%，否则存在严重的性能问题。当该值出现问题的时候，我们可以借助后面的等待时间和 latch 分析来查找解决问题。

（8）Parse CPU to Parse Elapsd %：如果该比率为 100%，意味着 CPU 等待时间为 0，没有任何等待。

（9）% Non-Parse CPU：如果这个值比较小，表示解析消耗的 CPU 时间过多。

（10）Memory Usage %：正在使用的共享池的百分率。这个数字应该长时间稳定在 75%～90%。如果这个百分比太低，表明共享池设置过大。如果这个百分率太高，会产生对大量的硬解析。在一个大小合适的系统中，共享池的使用率将处于 75%到略低于 90%的范围内。

量。

（12）% Memory for SQL w/exec>1：这是与不频繁使用的 SQL 语句相比，频繁使用的 SQL 语句消耗内存多少的一个度量。

```
Instance Efficiency Percentages (Target 100%)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

            Buffer Nowait %:   99.98       Redo NoWait %:    100.00
            Buffer  Hit   %:   99.87    In-memory Sort %:    100.00
            Library Hit   %:   99.67        Soft Parse %:     96.82
         Execute to Parse %:   80.93         Latch Hit %:     96.10
Parse CPU to Parse Elapsd %:    6.93     % Non-Parse CPU:     99.88
Shared Pool Statistics        Begin   End
                              ------  ------
             Memory Usage %:   32.87   33.12
    % SQL with executions>1:   80.00   82.69
  % Memory for SQL w/exec>1:   77.62   80.70
```

4．Top5 及其他等待事件信息

通过实例有效性统计数据，虽然可以获得一个整体印象，但是我们并不能由此来确定数据运行的性能。当前性能问题的确定，主要还是依靠下面的等待事件来确认。下面介绍常用的等待事件：

对于常见的等待事件，说明如下：

（1）db file scattered read 该事件通常与全表扫描或者 fast full index scan 有关。

（2）db file sequential read：该事件说明在单个数据块上大量等待，该值过高通常是由于表间连接顺序很糟糕（没有正确选择驱动行源），或者使用了非选择性索引。

（3）buffer busy wait：当缓冲区以一种非共享方式或者如正在被读入到缓冲时,就会出现该等待。

（4）latch free：当闩锁丢失率高于 0.5%时，需要调整这个问题。

（5）Enqueue 队列是一种锁，保护一些共享资源，防止并发的 DML 操作。

（6）Free Buffer：这个等待事件表明系统正在等待内存中的可用空间，这说明当前 Buffer 中已经没有 Free 的内存空间。

（7）Log file single write：该事件仅与写日志文件头块相关 通常发生在增加新的组成员和增进

（8）log file parallel write：从 log buffer 写 redo 记录到 redo log 文件，主要指常规写操作(相对于 log file sync)。

（9）log file sync：当一个用户提交或回滚数据时，LGWR 将会话的 redo 记录从日志缓冲区填充到日志文件中，用户的进程必须等待这个填充工作完成。

（11）logfile switch：通常是因为归档速度不够快。

（12）DB File Parallel Write：文件被 DBWR 并行写时发生。解决办法：改善 IO 性能。

（13）DB File Single Write：当文件头或别的单独块被写入时发生，这一等待直到所有的 I/O 调用完成。解决办法：改善 IO 性能。

（14）DB FILE Scattered Read：散列读取数据文件时产生。解决办法：改善 IO 性能。

（15）DB FILE Sequential Read：顺序读取数据文件时产生。解决办法：改善 IO 性能。

（16）Direct Path Read：一般直接路径读取是指将数据块直接读入 PGA 中。一般用于排序、并行查询和 read ahead 操作。这个等待可能是由于 I/O 造成的。

（17）direct path write：直接路径写该等待发生在，系统等待确认所有未完成的异步 I/O 都已写入磁盘。对于这一写入等待，我们应该分散负载，加快其写入操作。

（18）control file parallel write：当 server 进程更新所有控制文件时，这个事件可能出现。

（19）control file sequential read/ control file single write：控制文件连续读/控制文件单个写对单个控制文件 I/O 存在问题时，这两个事件会出现。如果等待比较明显，检查单个控制文件，看存放位置是否存在 I/O 瓶颈。

```
Top 5 Timed Events
~~~~~~~~~~~~~~~~~~                                                 % Total
Event                                            Waits    Time (s) Ela Time
-------------------------------------------- ------------ ----------- --------
CPU time                                                         361     54.14
log file sync                                      74,324         101     15.22
enqueue                                               729          88     13.28
db file sequential read                             7,303          65      9.76
SQL*Net message from dblink                           482          20      3.05

Wait Events for DB: ORA92  Instance: ora92  Snaps: 13 -14
-> s  - second
```

```
-> ms - millisecond -    1000th of a second
-> us - microsecond - 1000000th of a second
-> ordered by wait time desc, waits desc (idle events last)
```

Event	Waits	Timeouts	Total Wait Time (s)	Avg wait (ms)	Waits /txn
log file sync	74,324	0	101	1	1.0
enqueue	729	0	88	121	0.0
db file sequential read	7,303	0	65	9	0.1
SQL*Net message from dblink	482	0	20	42	0.0
db file parallel write	725	0	14	19	0.0
db file scattered read	2,415	0	6	3	0.0
process startup	8	0	4	440	0.0
latch free	1,307	1,300	2	1	0.0
log file parallel write	67,042	0	2	0	0.9
control file sequential read	269	0	1	3	0.0
single-task message	24	0	1	33	0.0
control file parallel write	325	0	1	2	0.0
buffer busy waits	3,368	0	1	0	0.0
log file switch completion	19	0	0	20	0.0
direct path read	288	0	0	0	0.0
LGWR wait for redo copy	1,032	0	0	0	0.0
SQL*Net more data to client	1,390	0	0	0	0.0
kksfbc child completion	1	1	0	10	0.0
log file sequential read	2	0	0	5	0.0
direct path write	128	0	0	0	0.0

5. SQL 统计信息

这部分分别按照逻辑读、物理读、执行次数、调用、解析次数、共享内存占用和多版本缓存的排序，列出了最严重的几个 SQL。这里就不给出例子了，大家可以自己研究一下。

6．实例的活动信息

这部分数据主要是从 V$SYSSTAT 表中统计出来的， 其中有一些条目对我们的调优还是很有参考价值的。比如：

（1）consistent gets，db block gets 和 physical reads：这三个值，我们也可以计算得到buffer hit ratio，计算的公式如下： buffer hit ratio = 100*(1-physical reads /(consistent gets+ db block gets))。

（2）dirty buffers inspected：脏数据从 LRU 列表中老化，A value here indicates that the DBWR is not keeping up。如果这个值大于 0，就需要考虑增加 DBWRs。

（3）parse count (hard)/ parse count (total)：通过他们可以计算 soft parse 率为：

100-100*(parse count (hard)/parse count (total)) =100-100*(1-6090/191531)=96.82

（4）sorts (disks)：磁盘排序一般不能超过 5%。

（5）table fetch by rowid：这是通过索引或者 where rowid=语句来取得的行数，当然这个值越大越好。

（6）table fetch continued row：这是发生行迁移的行。当行迁移的情况比较严重时，需要对这部分进行优化。

```
Instance Activity Stats for DB: ORA92  Instance: ora92  Snaps: 13 -14
Statistic                                     Total     per Second    per Trans
--------------------------------- ----------------- -------------- ------------
.............
consistent gets                       16,616,758       19,711.5        219.8
consistent gets - examination          1,168,584        1,386.2         15.5
current blocks converted for CR                0            0.0          0.0
cursor authentications                         2            0.0          0.0
data blocks consistent reads - un         63,873           75.8          0.8
db block changes                       2,596,938        3,080.6         34.4
db block gets                          2,941,398        3,489.2         38.9
deferred (CURRENT) block cleanout        130,783          155.1          1.7
dirty buffers inspected                      166            0.2          0.0
enqueue conversions                          485            0.6          0.0
enqueue deadlocks                              0            0.0          0.0
```

enqueue requests	318,825	378.2	4.2
enqueue timeouts	0	0.0	0.0

Instance Activity Stats for DB: ORA92 Instance: ora92 Snaps: 13 -14

Statistic	Total	per Second	per Trans
enqueue waits	728	0.9	0.0
exchange deadlocks	30	0.0	0.0
execute count	1,004,280	1,191.3	13.3
free buffer inspected	188	0.2	0.0
free buffer requested	116,422	138.1	1.5
hot buffers moved to head of LRU	17,750	21.1	0.2
.....			
no work - consistent read gets	14,240,381	16,892.5	188.4
opened cursors cumulative	84,306	100.0	1.1
parse count (failures)	6,074	7.2	0.1
parse count (hard)	6,090	7.2	0.1
parse count (total)	191,531	227.2	2.5
.....			

Instance Activity Stats for DB: ORA92 Instance: ora92 Snaps: 13 -14

Statistic	Total	per Second	per Trans
session pga memory max	1,185,992,768	1,406,871.6	15,686.5
session uga memory	3,015,076,014,672	##############	############
session uga memory max	175,484,416	208,166.6	2,321.0
shared hash latch upgrades - no w	675,962	801.9	8.9
shared hash latch upgrades - wait	3,460	4.1	0.1
sorts (disks)	0	0	0
sorts (memory)	180,293	213.9	2.4
sorts (rows)	511,574	606.9	6.8

```
switch current to new buffer                59,534          70.6          0.8
table fetch by rowid                     2,094,274       2,484.3         27.7
table fetch continued row                      408           0.5          0.0
......
```

7．I/O 统计信息

下面两个报表是面向 I/O 的。在这里主要关注 Av Rd(ms)列 (reads per millisecond)的值，一般来说，大部分磁盘系统的这个值都能调整到 14ms 以下。如果该值超过 1000ms，基本可以肯定存在 I/O 的性能瓶颈。

```
Tablespace IO Stats for DB: ORA92  Instance: ora92  Snaps: 13 -14
->ordered by IOs (Reads + Writes) desc
Tablespace
------------------------------
                 Av      Av     Av                    Av        Buffer Av Buf
         Reads Reads/s Rd(ms) Blks/Rd       Writes Writes/s      Waits Wt(ms)
-------------- ------- ------ ------- ------------ -------- ---------- ------
......
SYSTEM
            55       0   11.5     5.0           17        0        717    0.1
TOOLS
             1       0   40.0     1.0            1        0          0    0.0
USERS
             1       0   40.0     1.0            1        0          0    0.0
          -------------------------------------------------------------
File IO Stats for DB: ORA92  Instance: ora92  Snaps: 13 -14
->ordered by Tablespace, File

Tablespace               Filename
------------------------ ----------------------------------------------------
                 Av      Av     Av                    Av        Buffer Av Buf
         Reads Reads/s Rd(ms) Blks/Rd       Writes Writes/s      Waits Wt(ms)
```

```
ICD_DXH_DAT                    /dev/rlv_data001
              377         0     9.4      1.0          1,640          2          321     0.2
                                 /dev/rlv_data002
              327         0     9.0      1.0          1,630          2          169     0.0
                                 /dev/rlv_data003
              313         0    10.0      1.0          1,718          2           87     0.0
```

8．Buffer Pool 统计信息

这部分是 buffer pool 的详细情况。

```
Buffer Pool Statistics for DB: ORA92  Instance: ora92  Snaps: 13 -14
-> Standard block size Pools  D: default,  K: keep,  R: recycle
-> Default Pools for other block sizes: 2k, 4k, 8k, 16k, 32k
                                                          Free    Write  Buffer
     Number of Cache      Buffer    Physical   Physical  Buffer Complete    Busy
P      Buffers Hit %       Gets       Reads     Writes   Waits    Waits   Waits
--- ---------- ----- ---------- ---------- ---------- ------- --------  ------
D     635,200  99.9  19,556,815     26,990     88,450       0        0   3,368
```

9．实例的恢复情况统计信息

这部分主要是关于实例的恢复的一些统计信息，也不需要怎么关注。

```
Instance Recovery Stats for DB: ORA92  Instance: ora92  Snaps: 13 -14
-> B: Begin snapshot,  E: End snapshot
  Targt Estd                                     Log File   Log Ckpt   Log Ckpt
  MTTR  MTTR   Recovery    Actual     Target       Size     Timeout    Interval
   (s)   (s)   Estd IOs  Redo Blks  Redo Blks  Redo Blks  Redo Blks  Redo Blks
- ----- ----- ---------- ---------- ---------- ---------- ---------- ----------
B   300    70      54311     452198     450720     450720    1224858
E   300    69      53127     452947     450720     450720    1472619
```

10．Buffer Pool 调整的 Advisory

这是 oracle 的对 buffer pool 的大小的调整建议。

-> Only rows with estimated physical reads >0 are displayed

-> ordered by Block Size, Buffers For Estimate (default block size first)

P	Size for Estimate (M)	Size Factr	Buffers for Estimate	Est Physical Read Factor	Estimated Physical Reads
D	512	.1	63,520	9.85	245,880,558
D	1,024	.2	127,040	5.41	134,932,093
D	1,536	.3	190,560	3.38	84,471,707
D	2,048	.4	254,080	2.41	60,240,471
D	2,560	.5	317,600	1.86	46,399,611
D	3,072	.6	381,120	1.54	38,365,243
D	3,584	.7	444,640	1.32	33,017,978
D	4,096	.8	508,160	1.18	29,353,901
D	4,608	.9	571,680	1.07	26,763,133
D	5,120	1.0	635,200	1.00	24,962,078
D	5,632	1.1	698,720	0.95	23,661,399
D	6,144	1.2	762,240	0.91	22,672,122
D	6,656	1.3	825,760	0.88	21,902,502
D	7,168	1.4	889,280	0.85	21,277,585
D	7,680	1.5	952,800	0.83	20,755,944
D	8,192	1.6	1,016,320	0.81	20,331,009
D	8,704	1.7	1,079,840	0.80	19,949,127
D	9,216	1.8	1,143,360	0.78	19,563,065
D	9,728	1.9	1,206,880	0.77	19,226,351
D	10,240	2.0	1,270,400	0.76	18,948,622

11. Buffer Pool 等待情况统计

这部分列出的 buffer 等待往往引起 data block 的比较大的等待。

Buffer wait Statistics for DB: ORA92 Instance: ora92 Snaps: 13 -14

-> ordered by wait time desc, waits desc

```
Class                    Waits   Time (s) Time (ms)
------------------ ---------- --------- ---------
data block               3,086         0         0
undo block                 196         0         0
undo header                 87         0         0
1st level bmb                3         0         0
2nd level bmb                1         0         0
```

12. PGA 统计信息

这部分主要说明 PGA 的使用情况，我们可以根据具体情况来设置参数 PGA_AGGREGATE_TARGET 的值。

```
PGA Aggr Target Stats for DB: ORA92  Instance: ora92  Snaps: 13 -14
-> B: Begin snap   E: End snap (rows dentified with B or E contain data
   which is absolute i.e. not diffed over the interval)
-> PGA cache hit % - percentage of W/A (WorkArea) data processed only in-memory
-> Auto PGA Target - actual workarea memory target
-> W/A PGA Used    - amount of memory used for all Workareas (manual + auto)
-> %PGA W/A Mem    - percentage of PGA memory allocated to workareas
-> %Auto W/A Mem   - percentage of workarea memory controlled by Auto Mem Mgmt
-> %Man W/A Mem    - percentage of workarea memory under manual control
PGA Cache Hit % W/A MB Processed Extra W/A MB Read/Written
--------------- ---------------- ------------------------
          100.0            2,421                        0
                                              %PGA  %Auto   %Man
  PGA Aggr  Auto PGA   PGA Mem    W/A PGA     W/A    W/A    W/A   Global Mem
  Target(M) Target(M)  Alloc(M)   Used(M)     Mem    Mem    Mem    Bound(K)
- --------- --------- ---------- ---------- ------ ------ ------ ----------
B       500       354      216.3        0.0     .0     .0     .0     25,600
E       500       355      214.4        0.0     .0     .0     .0     25,600

PGA Aggr Target Histogram for DB: ORA92  Instance: ora92  Snaps: 13 -14
```

Low Optimal	High Optimal	Total Execs	Optimal Execs	1-Pass Execs	M-Pass Execs
8K	16K	246,058	246,058	0	0
16K	32K	104	104	0	0
32K	64K	1	1	0	0
64K	128K	3	3	0	0
128K	256K	2	2	0	0
256K	512K	2	2	0	0
512K	1024K	1	1	0	0
2M	4M	4	4	0	0

13. PGA 调整的 Advisory

这部分给出了 PGA_AGGREGATE_TARGET 参数的调整建议。

```
PGA Memory Advisory for DB: ORA92  Instance: ora92  End Snap: 14
-> When using Auto Memory Mgmt, minimally choose a pga_aggregate_target value
   where Estd PGA Overalloc Count is 0
```

PGA Target Est (MB)	Size Factr	W/A MB Processed	Estd Extra W/A MB Read/ Written to Disk	Estd PGA Cache Hit %	Estd PGA Overalloc Count
63	0.1	4,398,132.4	17,448.1	100.0	34,943
125	0.3	4,398,132.4	4,267.6	100.0	47
250	0.5	4,398,132.4	435.8	100.0	0
375	0.8	4,398,132.4	382.9	100.0	0
500	1.0	4,398,132.4	0.0	100.0	0
600	1.2	4,398,132.4	0.0	100.0	0
700	1.4	4,398,132.4	0.0	100.0	0
800	1.6	4,398,132.4	0.0	100.0	0
900	1.8	4,398,132.4	0.0	100.0	0

1,500	3.0	4,398,132.4	0.0	100.0	0
2,000	4.0	4,398,132.4	0.0	100.0	0
3,000	6.0	4,398,132.4	0.0	100.0	0
4,000	8.0	4,398,132.4	0.0	100.0	0

14. 队列的统计信息

这部分详细列出了 Enqueue 各个子类的等待情况。

```
Enqueue activity for DB: ORA92  Instance: ora92  Snaps: 13 -14
-> Enqueue stats gathered prior to 9i should not be compared with 9i data
-> ordered by Wait Time desc, Waits desc
```

Eq	Requests	Succ Gets	Failed Gets	Waits	Avg Wt Time (ms)	Wait Time (s)
TX	81,127	81,263	0	674	134.32	91
SQ	2,032	2,032	0	53	.25	0
HW	107	107	0	1	2.00	0

15. 回滚段统计信息

这里列出了回滚段的使用情况。从 9i 开始，回滚段一般都是自动管理的，一般情况下，这里我们不需要太重点关注。

```
Rollback Segment Stats for DB: ORA92  Instance: ora92  Snaps: 13 -14
->A high value for "Pct Waits" suggests more rollback segments may be required
->RBS stats may not be accurate between begin and end snaps when using Auto Undo
  managment, as RBS may be dynamically created and dropped as needed
```

RBS No	Trans Table Gets	Pct Waits	Undo Bytes Written	Wraps	Shrinks	Extends
0	4.0	0.00	0	0	0	0
1	67,197.0	0.00	52,642,136	7	0	7
2	16,647.0	0.00	12,321,446	2	0	2

```
.....
    14        7,656.0    0.00       5,171,854        6        0        6
          -------------------------------------------------------------
Rollback Segment Storage for DB: ORA92  Instance: ora92  Snaps: 13 -14
->Optimal Size should be larger than Avg Active
RBS No    Segment Size      Avg Active    Optimal Size    Maximum Size
------ --------------- --------------- --------------- ---------------
     0         385,024          59,380                        385,024
     1     176,283,648       6,272,394                    176,283,648
     2      92,397,568       3,103,770                     92,397,568
     3      54,648,832       1,317,711                    116,514,816
.....
    14      45,211,648         967,324                     45,211,648
          -------------------------------------------------------------
Undo Segment Summary for DB: ORA92  Instance: ora92  Snaps: 13 -14
-> Undo segment block stats:
-> uS - unexpired Stolen,   uR - unexpired Released,   uU - unexpired reUsed
-> eS - expired   Stolen,   eR - expired   Released,   eU - expired   reUsed

Undo            Undo        Num  Max Qry     Max Tx Snapshot Out of uS/uR/uU/
 TS#          Blocks      Trans  Len (s)   Concurcy  Too Old  Space eS/eR/eU
---- -------------- ---------- -------- ---------- -------- ------ -------------
   1         13,853 ##########       56          1        0      0 0/0/0/0/0/0
          -------------------------------------------------------------

Undo Segment Stats for DB: ORA92  Instance: ora92  Snaps: 13 -14
-> ordered by Time desc

                     Undo      Num Max Qry   Max Tx  Snap   Out of uS/uR/uU/
End Time           Blocks    Trans Len (s)    Concy Too Old  Space eS/eR/eU
------------ ------------ -------- ------- -------- ------- ------ -------------
```

16．闩锁统计信息

这部分列出了 Latch 的统计信息。

Latch 是一种低级排队机制，用于防止对内存结构的并行访问，保护系统全局区(SGA)共享内存结构。Latch 是一种快速地被获取和释放的内存锁。下面对几个重要类型的 latch 等待加以说明：

（1）latch free：当 latch free 在报告的高等待事件中出现时，就表示可能出现了性能问题，就需要在这一部分详细分析出现等待的具体的 latch 的类型，然后再调整。

（2）cache buffers chain：cbc latch 表明系统存在热块，通过稀释数据块来减少热块。

（3）cache buffers lru chain：该 latch 用于扫描 buffer 的 LRU 链表。三种情况可导致争用：1）buffer cache 太小 ；2）buffer cache 的过度使用，或者太多的基于 cache 的排序操作；3）DBWR 不及时。解决方法：查找逻辑读过高的 statement，增大 buffer cache。

（4）Library cache and shared pool 争用：

降低共享池的 latch 争用，我们主要可以考虑如下的几个事件：

1）使用绑定变量，

2）使用 cursor sharing，

3)设置 session_cached_cursors 参数。该参数的作用是将 cursor 从 shared pool 转移到 pga 中。减小对共享池的争用。一般初始的值可以设置为 100，然后视情况再作调整。

4）设置合适大小的共享池

（5）Redo Copy：这个 latch 用来从 PGA 中 copy redo records 到 redo log buffer。latch 的初始数量是 2*COU_OUNT，可以通过设置参数_LOG_SIMULTANEOUS_COPIES 在增加 latch 的数量，减小争用。

（6）Redo allocation：该 latch 用于 redo log buffer 的分配。减小这种类型的争用的方法有 3 个：

1）增大 redo log buffer，

2）适当使用 nologging 选项，

3）避免不必要的 commit 操作。

（7）Row cache objects：该 latch 出现争用，通常表明数据字典存在争用的情况，这往往也预示着过多的依赖于公共同义词的 parse。解决方法：

1）增大 shared pool，

2）使用本地管理的表空间，尤其对于索引表空间。

```
Latch Activity for DB: ORA92  Instance: ora92  Snaps: 13 -14
```

willing-to-wait latch get requests
->"NoWait Requests", "Pct NoWait Miss" are for no-wait latch get requests
->"Pct Misses" for both should be very close to 0.0

Latch	Get Requests	Pct Get Miss	Avg Slps /Miss	Wait Time (s)	NoWait Requests	Pct NoWait Miss
Consistent RBA	67,061	0.0		0	0	
FIB s.o chain latch	4	0.0		0	0	
FOB s.o list latch	240	0.0		0	0	
SQL memory manager latch	1	0.0		0	281	0.0
SQL memory manager worka	296,916	0.0	0.0	0	0	
active checkpoint queue	1,340	0.0		0	0	
cache buffer handles	2,980	0.0	0.0	0	0	
cache buffers chains	44,382,255	5.4	0.0	0	93,328	0.0
cache buffers lru chain	148,064	0.0	0.0	0	131,731	0.0
.....						
library cache	4,755,237	1.1	0.0	0	0	
library cache load lock	274	0.0		0	0	
library cache pin	3,867,684	0.2	0.0	0	0	
.....						
redo allocation	1,477,895	0.2	0.0	0	0	
redo copy	0	0	1,344,659	0.1		
redo writing	203,129	0.0	0.0	0	0	
row cache enqueue latch	21,995	0.1	0.0	0	0	
row cache objects	30,913	1.9	0.0	0	0	
sequence cache	122,787	0.3	0.0	0	0	
session allocation	499,051	0.4	0.0	0	0	
session idle bit	901,267	0.0	0.0	0	0	
session switching	14	0.0		0	0	
session timer	291	0.0		0	0	

->"Get Requests", "Pct Get Miss" and "Avg Slps/Miss" are statistics for
  willing-to-wait latch get requests
->"NoWait Requests", "Pct NoWait Miss" are for no-wait latch get requests
->"Pct Misses" for both should be very close to 0.0

Latch	Get Requests	Pct Get Miss	Avg Slps /Miss	Wait Time (s)	NoWait Requests	Pct NoWait Miss
shared pool	2,819,295	0.1	0.0	0	0	
sim partition latch	0			0	252	0.0
simulator hash latch	1,046,772	0.0	0.0	0	0	
simulator lru latch	596	0.0		0	15,390	5.7
sort extent pool	17	0.0		0	0	
spilled msgs queues list	27	0.0		0	0	
transaction allocation	3,207	0.4	0.0	0	0	
transaction branch alloc	62	0.0		0	0	
undo global data	286,249	0.1	0.0	0	0	
user lock	1,432	0.6	0.0	0	0	

Latch Sleep breakdown for DB: ORA92 Instance: ora92 Snaps: 13 -14
-> ordered by misses desc

Latch Name	Get Requests	Misses	Sleeps	Spin & Sleeps 1->4
cache buffers chains	44,382,255	2,408,704	1,302	0/0/0/0/0
library cache	4,755,237	51,183	5	51178/5/0/0/0
session allocation	499,051	2,050	1	2049/1/0/0/0

Latch Miss Sources for DB: ORA92 Instance: ora92 Snaps: 13 -14
-> only latches with sleeps are shown

Latch Name	Where	NoWait Misses	Sleeps	Waiter Sleeps
cache buffers chains	kcbgtcr: kslbegin excl	0	859	741
……				
library cache	kglpnc: child	0	3	2
library cache	kglic	0	1	0
library cache	kglupc: child	0	1	0
session allocation	ksuxds: not user session	0	1	0

17．共享池统计信息

这部分列出了共享池的统计信息并给出了调整建议。主要参数有：

（1）Get Requests：get 表示一种类型的锁，语法分析锁。这种类型的锁在引用了一个对象的那条 SQL 语句的语法分析阶段被设置在该对象上。每当一条语句被语法分析一次时 ，Get Requests 的值就增加 1。

（2）pin requests：pin 也表示一种类型的锁，是在执行发生的加锁。每当一条语句执行一次，pin requests 的值就增加 1。

（3）reloads：reloads 列显示一条已执行过的语句因 Library Cache 使该语句的已语法分析版本过期或作废而需要被重新语法分析的次数。

（4）invalidations：失效发生在一条已告诉缓存的 SQL 语句即使已经在 library cache 中，但已被标记为无效并迎词而被迫重新做语法分析的时候。每当已告诉缓存的语句所引用的对象以某种方式被修改时，这些语句就被标记为无效。

（5）pct miss 应该不高于 1%。

（6）Reloads /pin requests <1%，否则应该考虑增大 SHARED_POOL_SIZE。

```
Dictionary Cache Stats for DB: ORA92  Instance: ora92  Snaps: 13 -14
->"Pct Misses"  should be very low (< 2% in most cases)
->"Cache Usage" is the number of cache entries being used
->"Pct SGA"     is the ratio of usage to allocated size for that cache
```

Cache	Get Requests	Pct Miss	Scan Reqs	Pct Miss	Mod Reqs	Final Usage

dc_database_links	528	0.0	0	0	2
dc_histogram_defs	22	86.4	0	0	4,295
dc_object_ids	69	29.0	0	0	403
dc_objects	233	36.1	0	0	895
dc_profiles	705	0.0	0	0	1
dc_rollback_segments	3,911	0.0	0	0	233
dc_segments	187	1.1	0	12	5,778
dc_sequences	1,978	0.0	0	1,978	9
dc_tablespace_quotas	12	0.0	0	12	4
dc_tablespaces	458	0.0	0	0	11
dc_user_grants	120	0.0	0	0	18
dc_usernames	56	0.0	0	0	12
dc_users	3,674	0.0	0	0	20

Library Cache Activity for DB: ORA92 Instance: ora92 Snaps: 13 -14

->"Pct Misses" should be very low

Namespace	Get Requests	Pct Miss	Pin Requests	Pct Miss	Reloads	Invali-dations
BODY	1,699	0.0	1,699	0.0	0	0
INDEX	54	0.0	54	0.0	0	0
SQL AREA	5,665	0.0	1,497,193	0.4	6,086	0
TABLE/PROCEDURE	92,424	0.1	429,557	0.1	0	0
TRIGGER	1,412	0.0	1,412	0.0	0	0

Shared Pool Advisory for DB: ORA92 Instance: ora92 End Snap: 14

-> Note there is often a 1:Many correlation between a single logical object
in the Library Cache, and the physical number of memory objects associated
with it. Therefore comparing the number of Lib Cache objects (e.g. in
v$librarycache), with the number of Lib Cache Memory Objects is invalid

Shared Pool	SP	Estd	Estd	Estd Lib LC Time

Estim (M)	Factr	Size (M)	Mem Obj	Saved (s)	Factr	Mem Obj Hits
224	.5	97	55,129	10,514,328	1.0	807,719,425
272	.7	112	68,387	10,514,329	1.0	807,719,480
320	.8	113	68,787	10,514,329	1.0	807,719,485
368	.9	113	68,787	10,514,329	1.0	807,719,485
416	1.0	113	68,787	10,514,329	1.0	807,719,485
464	1.1	113	68,787	10,514,329	1.0	807,719,485
512	1.2	113	68,787	10,514,329	1.0	807,719,485
560	1.3	113	68,787	10,514,329	1.0	807,719,485
608	1.5	113	68,787	10,514,329	1.0	807,719,485
656	1.6	113	68,787	10,514,329	1.0	807,719,485
704	1.7	113	68,787	10,514,329	1.0	807,719,485
752	1.8	113	68,787	10,514,329	1.0	807,719,485
800	1.9	113	68,787	10,514,329	1.0	807,719,485
848	2.0	113	68,787	10,514,329	1.0	807,719,485

18. SGA 内存分配

这部分列出了当前 SGA 的分配情况。

SGA Memory Summary for DB: ORA92 Instance: ora92 Snaps: 13 -14

SGA regions	Size in Bytes
Database Buffers	5,368,709,120
Fixed Size	754,760
Redo Buffers	2,371,584
Variable Size	3,137,339,392
sum	8,509,174,856

SGA breakdown difference for DB: ORA92 Instance: ora92 Snaps: 13 -14

Pool	Name	Begin value	End value	% Diff

```
java    free memory                    33,554,432       33,554,432     0.00
large   free memory                    16,777,216       16,777,216     0.00
shared 1M buffer                        1,056,768        1,056,768     0.00
```

……省略部分内容……

19. 资源限制统计信息

这部分是关于 oracle 资源限制的一个描述。

```
Resource Limit Stats for DB: ORA92  Instance: ora92  End Snap: 14
-> only rows with Current or Maximum Utilization > 80% of Limit are shown
-> ordered by resource name

                                 Current      Maximum     Initial
Resource Name                  Utilization  Utilization  Allocation      Limit
------------------------------ ----------- ----------- ---------- ----------
parallel_max_servers                     0           5          6          6
```

20. 初始化统计信息

这是报表的最后一部分，是关于常用重要的一些系统的初始化参数设置的汇总情况。

```
init.ora Parameters for DB: ORA92  Instance: ora92  Snaps: 13 -14
                                                                End value
Parameter Name                 Begin value                     (if different)
------------------------------ --------------------------------- -------------
aq_tm_processes                1
background_dump_dest           /opt/oracle/app/oracle/admin/ora9
compatible                     9.2.0.0.0
control_files                  /dev/rlv_ctrl1, /dev/rlv_ctrl2, /
core_dump_dest                 /opt/oracle/app/oracle/admin/ora9
cursor_sharing                 SIMILAR
db_block_size                  8192
db_cache_size                  5368709120
db_domain
```

```
db_name                          ora92
```

7.2.5 AWR

Active Workload Repository(活动负载档案库)是 Oracle 10g 引入的新特性，它是从 Statspack 发展起来的，和 Statspack 相比，报表更友好，而且最主要的是 AWR 可以为 RAC 提供了报告，而 Statspack 不行。

要使用 AWR，必须打开参数 STATISTICS_LEVEL，至少为 TYPICAL。

AWR 需要收集的数据本来是放到内存中（share pool），它通过一个新的后台进程 MMON 定期写到磁盘中。所以 10g 的 share pool 要求比以前版本更大，一般推荐比以前大 15-20%。

只要系统生成快照,MMON 进程就会自动运行 ADDM。ADDM 接受来自 AWR 的统计量及其他信息，自动生成的报告。

手工生成 awr 报告有两种方法：

（1）Oracle 提供的脚本：

$ORACLE_HOME/rdbms/admin/awrrpt.sql

如果是 RAC 环境下，可以执行$ORACLE_HOME/rdbms/admin/addmrpti.sql

报表的形式有两种：text 和 html。

（2）Oracle 提供的程序包：

生成 text 报表：DBMS_WORKLOAD_REPOSITORY.AWR_REPORT_TEXT

生成 html 报表：DBMS_WORKLOAD_REPOSITORY.AWR_REPORT_HTML

例子：

```
SQL>DBMS_WORKLOAD_REPOSITORY.awr_report_html(
l_dbid =>3265083610,l_inst_num =>1,l_bid =>314,l_eid =>315) from dual;
```

AWR 报表的内容和 Statspack 差不多，但是更为详细直观。这里就不多说了。

7.2.6 EM

EM 是 Oracle 提供的一个基于 BS 的集管理、维护与调优为一体的图形化工具。其实在维护与调优方面，EM 和 AWR 是紧密联系的，他们之间的纽带就是 ADDM。

ADDM，称为自动诊断监视器，当 Oracle 生成 AWR 快照时，MMON 进程就会自动启动 ADDM，然后 ADDM 接受来自 AWR 的统计量及其他信息，自动生成的 ADDM 报告，并将这些报告传送给 EM。这样我们通过浏览器就可以看到数据库存在的性能问题了。

7.2.7 DB优化主要关注点

Oracle 性能调整的关注点有很多，这里列出了主要的几点：

1．SGA

（1）Shared pool tunning

Shared pool 的优化应该放在优先考虑，因为一个 cache miss 在 shared pool 中发生比在 data buffer 中发生导致的成本更高，由于 dictionary 数据一般比 library cache 中的数据在内存中保存的时间长，所以关键是 library cache 的优化。

1）检查 v$librarycache 中 sql area 的 gethitratio 是否超过 90%，如果未超过 90%，应该检查应用代码，提高应用代码的效率。

Sql> Select gethitratio from v$librarycache where namespace='sql area';

2）保留大的对象在 shared pool 中。大的对象是造成内存碎片的主要原因，为了腾出空间许多小对象需要移出内存，从而影响了用户的性能。因此需要将一些常用的大的对象保留在 shared pool 中。

查找没有保存在 library cache 中的大对象：

```
Sql> Select * from v$db_object_cache
  where sharable_mem>10000 and type in
  ('PACKAGE','PROCEDURE','FUNCTION','PACKAGE BODY') and kept='NO';
```

将这些对象保存在 library cache 中：

Sql> Execute dbms_shared_pool.keep('package_name');

要使用该存储过程需要先安装 dbmspool.sql 脚本。

3）查找是否存在过大的匿名 pl/sql 代码块。两种解决方案：

1〉转换成小的匿名块调用存储过程

2〉将其保留在 shared pool 中

查找是否存在过大的匿名 pl/sql 块：

Sql> Select sql_text from v$sqlarea where command_type=47 and length(sql_text)>500;

4）Dictionary cache 的优化：

避免出现 Dictionary cache 的 misses，或者 misses 的数量保持稳定,只能通过调整 shared_pool_size 来间接调整 dictionary cache 的大小。

1) 统计 buffer cache 的 cache hit ratio>90%，如果低于 90％，需要增加 buffer cache 的值。

2）granule 大小的设置，db_cache_size 以字节为单位定义了 default buffer pool 的大小。

如果 SGA<128M，granule=4M,否则 granule＝16M，即需要调整 sga 的时候以 granule 为单位增加大小，并且 sga 的大小应该是 granule 的整数倍。

（3）redo log buffer

对应的参数是 log_buffer，缺省值与 OS 相关，一般是 500K。关于 redo log buffer，需要考虑以下几点：

1）检查 v$session_wait 是否存在 log buffer wait：

如果存在，可以增加 log buffer 的大小，也可以通过将 log 文件移到访问速度更快的磁盘来解决。

2）检查 v$sysstat 是否存在 redo buffer allocation retries：

如果 Redo buffer allocation retries 接近 0，小于 redo entries 的 1％，如果一直在增长，表明进程已经不得不等待 redo buffer 的空间。如果 Redo buffer allocation retries 过大，增加 log_buffer 的值。

3）检查日志文件上是否存在磁盘 IO 竞争现象：

如果存在竞争，可以考虑将 log 文件转移到独立的、更快的存储设备上或增大 log 文件。

察看方法：

```
Sql> Select event,total_waits,time_waited,average_wait from
     v$system_event where event like ‘log file switch completion%’;
```

4）检查点的设置是否合理：

检查 alert.log 文件中，是否存在 checkpoint not complete。如果存在等待，调整 log_checkpoint_interval、log_checkpoint_timeout 的设置。

5）检查 log archiver 的工作：

如果存在等待，检查保存归档日志的存储设备是否已满，增加日志文件组，调整 log_archiver_max_processes。察看方法：

```
Sql> Select event,total_waits,time_waited,average_wait from v$system_event
     where event like ‘log file switch (arch%’;
```

2．IO

IO IO

（1）分散磁盘的 IO：

1）将文件分散到不同的设备上

2）将数据文件与日志文件分开

3）减少与服务器无关的磁盘 IO

4）将表和索引分散到独立的表空间中

5）使用独立的回滚表空间

6）将大的数据库对象保存在各自独立的表空间中

7）创建一个或多个独立的临时表空间

（2）减少全表扫描：

如果 table scans (long tables)的值很高，说明大部分的 table access 没有经过索引查找，应该检查应用或建立索引，要确保有效的索引在正确的位置上。以下方法察看全表扫描：

Sql> Select name,value from v$sysstat where name like '%table scan%';

3．排序

服务器首先在 sort_area_size 指定大小的内存区域里排序，如果所需的空间超过 sort_area_size，排序会在临时表空间里进行。在专用服务器模式下，排序空间在 PGA 中，在共享服务器模式下，排序空间在 UGA 中。如果没有建立 large pool，UGA 处于 shared pool 中，如果建立了 large pool，UGA 就处于 large pool 中，而 PGA 不在 sga 中，它是与每个进程对应单独存在的。

排序优化的主要措施有：

（1）尽可能避免排序

（2）为需要排序的操作创建索引

（3）尽可能在内存中排序

（4）大型排序需要分配合适的临时空间

可以通过以下方法察看系统的排序量：

```
Sql> SELECT disk.Value disk,mem.Value mem,(disk.Value/mem.Value)*100 ratio
     FROM v$sysstat disk,v$sysstat mem
     WHERE mem.NAME='sorts (memory)' AND disk.NAME='sorts (disk)';
```

如果 Sort（disk）/ Sort（memory）<5%,如果超过 5%，增加 sort_area_size 的值。

4 Latch

Latch 是简单的、低层次的序列化技术，用以保护 SGA 中的共享数据结构，比如并发用户列表和 buffer cache 里的 blocks 信息。一个服务器进程或后台进程在开始操作或寻找一个共享数据结构之前必须获得对应的 latch，在完成以后释放 latch。不必对 latch 本身进行优化，如果 latch 存在竞争，表明 SGA 的一部分正在经历不正常的资源使用。

比如我们执行以下语句：

```
Sql> Select * from v$latch
        where name like  '%shared pool%'  or name like  '%library cache%';
```

如果 Latch 在 shared pool 或 library cache 上存在竞争，可能的原因有：

（1）不能共享的 sql,应检查他们是否相似，考虑使用绑定变量。确认方法：

```
Sql> Select sql_text from v$sqlarea where executions=1 order by upper(sql_text);
```

（2）共享 sql 被重新编译，考虑 library cache 的大小是否需要调整。确认方法：

```
Sql> SELECT sql_text,parse_calls,executions FROM v$sqlarea where parse_calls>5;
```

（3）library cache 不够大。考虑增大。

5．回滚段

Transaction 以轮循的方式使用 rollback segment 里的 extent，当前所在的 extent 满时就移动到下一个 extent。可能有多个 transaction 同时向同一个 extent 写数据，但一个 rollback segment block 中只能保存一个 transaction 的数据。

Oracle 在每个 Rollback segment header 中保存了一个 transaction table，包括了每个 rollback segment 中包含的事务信息，rollback segment header 的活动控制了向 rollbak segment 写入被修改的数据。rollback segment header 是经常被修改的数据库块，因此它应该被长时间留在 buffer cache 中，为了避免在 transaction table 产生竞争导致性能下降，应有多个 rollback segment 或应尽量使用 oracle server 自动管理的 rollback segment。

6．Lock

Lock 是 Oracle 的对象锁。一般来说，一个 transaction 至少获得两个锁：一个共享的表锁，一个专有的行锁。Oracle server 将所有的锁维护在一个队列里，队列跟踪了等待锁的用户、申请锁的类型以及用户的顺序信息。

引起锁等待的原因主要有：

（2）长时间运行的 transaction；

（3）未提交的修改；

（4）其他产品施加的高层次的锁。

解决锁等待的方法：锁的拥有者提交或回滚事务；杀死用户会话。

7．Block

Block 上面存在的问题主要有以下三个方面：

（1）block size：

block size 的大小和 block 被访问的次数密切相关。如何降低 block 的访问次数：

1）使用较大的 block size：容易产生行迁移和热点酷爱，但是可以提高数据查询的 IO，尤其是在全表扫描时。

2）使用较小的 block size：极少产生 block 竞争，有利于较小的行和随机访问。缺点是由于每个 block 的行数更少，可能需要读取更多的 index 块和数据块。

3）一般来说，在 OLTP 环境中，较小的 block size 更合适，而在 DSS 环境中，适宜选择较大的 block size。

（2）PCTFREE 和 PCTUSED：

PCTFREE 和 PCTUSED 能控制一个 segment 里所有数据块中 free space 的使用。

1）PCTFREE：一个数据块保留的用于块中已有记录的可能需要更新的自由空间 block size 的最小比例。

2）PCTUSED：在新记录被插入 block 之前这个 block 可以用于存储数据的空间所占 block size 的最小比例。

（3）Migration 和 Chaining：

如果一行的数据太大以至一个单独的 block 容纳不下，会产生两种现象：

1）Chaining：行数据太大以至一个空 block 容纳不下，oracle 会将这一行的数据存放在一个或多个 block 组成的 block chain 中，insert、update 都可能导致这个问题，在某些情况下 row chaining 是不能避免的。

2）Migration：一次 update 操作可能导致行数据增大，以至它所在的 block 容纳不下，oracle server 会去寻找一个有足够自由空间容纳整行数据的 block，如果这样的 block 存在，oracle server 把整行移到新的 block，在原位置保存一个指向新存放位置的镜像行，镜像行的 rowid 和原来的 rowid 一致。

Chaining、Migration 的弊端：insert、update 的性能降低，索引查询增加了 IO 次数。

如何消除镜像行：

1）产生 Migration 的原因可能是由于 PCTFREE 设置的太低以至没有保留足够的空间用于更新。可以通过增加 PCTFREE 的值避免行镜像产生。

2）运行 analyize table ... list chained rows;

复制镜像行到另一个表 tmp；

从源表中删除这些行；

从 tmp 中将这些行插回到源表中。

（4）索引 block：

在一个数据量变化很快的表上建索引会影响性能。因为索引的 block 只有完全清空时才能进入 free list，即使一个索引 block 里只含有一个条目，它也必须被维护。所以为了提高索引块的空间利用率，就需要定时的重建索引。

8．SQL

大部分性能问题都是由低效的 SQL 造成的。所以 SQL 这一块需要重点关注。Statspack 和 awr 都有专门部分列出低效的 SQL，也可以通过视图 V$session_longops 来察看执行时间比较长的 SQL。

至于具体的优化措施可以察看 7.2 章节的内容。

9．OS

操作系统优化时应该考虑的因素有：内存的使用；Cpu 的使用；IO 级别；网络流量。各个因素互相影响，正确的优化次序是内存、IO、CPU。

7.2.8 DB优化步骤

从上面讲述的内容可以看出 DB 的优化也是有迹可寻的。一般步骤是：

1．在系统有代表性的时段收集性能报告。

所谓代表性的时段，是指 DB 负载上升到需要我们优化的时段。

收集报告的一般工具如上所述主要是：statspack、awr、em 等。

2．分析性能报告，找到引起性能问题的主要原因。

3．针对不同的原因采取相应的措施。

如果是 OS 参数问题　那么修改 OS 参数

如果是内存问题，增加内存；

如果是 IO 问题，调整存储，文件分布等；

如果是 SQL 问题，转到 SQL 调优。

4．调整后，还需要验证这些措施的有效性和合理性。