第4章 关系数据库标准语言 SQL

关系代数从数学表达的角度描述了选择、投影、连接等运算，这些运算的本质是对关系中的数据进行查询，因此，关系代数是抽象的关系查询语言。SQL（Structured Query Language，结构化查询语言）简单易学，功能丰富，是用户操作关系数据库的通用语言。它虽然叫结构化查询语言，但并不是说 SQL 只支持查询操作，它实际包含了数据定义、数据操纵和数据控制等与数据库有关的全部功能。本章节在介绍 SQL 的某些查询功能时，会联合关系代数进行描述，以帮助初学者更好地融会贯通。

另外，各个数据库厂商都有各自的 SQL 软件产品，尽管绝大多数产品对 SQL 语言的支持很相似，但它们之间也存在着一定的差异，这些差异不利于初学者的学习。因此，我们在本章介绍 SQL 时上主要介绍标准的 SQL 语言。

本章的学习要点：

*SQL 特点

*SQL 查询语句

*SQL 更新语句

*SQL 定义语句

本章的学习难点：

*相关子查询

4.1 SQL 概述

4.1.1 SQL 的产生与发展

最早的 SQL 原型是 IBM 的研究人员于 20 世纪 70 年代开发的，其原型被命名为 SEQUEL(Structured English Query Language)，因此很多人把在这个原型之后推出的 SQL 读成“sequel”，但根据 ANSI SQL 委员会的规定，其正式发音应该是“ess cue ell”。

1986 年 10 月，美国 ANSI（American National Standard Institute，美国国家标准学会）采用 SQL 作为**关系数据库管理系统**的标准语言（ANSI X3. 135-1986），后为 ISO（International Organization for Standardization，国际标准化组织）采纳为国际标准。此后 ANSI 不断修改和完善 SQL 标准，其扩充过程经历了 SQL/86、SQL/89、SQL/92、SQL99、SQL2003、SQL2006、SQL2008。可以发现 SQL 标准的内容越来越多，包括了 SQL 框架、SQL 基础部分、SQL 宿主语言绑定、SQL 调用接口、SQL 永久存储模块、SQL 外部数据管理、SQL 对象语言绑定、SQL 信息定义模式和 XML 相关规范等多个部分。

SQL 是由 ANSI 制定的用来访问和操作数据库系统的标准语言。不幸地是，存在着很多不同版本的 SQL 语言，但是为了与 ANSI 标准相兼容，它们必须以相似的方式共同地来支持一些主要的关键词（比如 SELECT、UPDATE、DELETE、INSERT、WHERE 等等）。本章我们重点介绍标准的 SQL 语句，在本书第二部分将介绍 SQL Server 支持的相关 SQL 语句。

说明： 除了 SQL 标准之外，大部分 SQL 数据库程序都拥有它们自己的私有扩展！

4.1.2 SQL 语言功能概述

SQL 语言从功能角度来看主要包括数据定义、数据操纵和数据控制，表 4-1 列出了对应这几个部分的命令。

表 4-1 SQL 语言包含的主要动词

SQL 功能	命令
数据定义	CREATE，DROP，ALTER
数据操纵	SELECT，INSERT，UPDATE，DELETE
数据控制	GRANT，REVOKE

以上各部分的简单介绍如下：

（1） 数据定义语言（Data Definition Language， DDL）

数据定义用来定义数据库的逻辑结构，包括定义表、视图和索引。数据定义是定义结构，不涉及具体的数据。

（2） 数据操纵语言（Data Manipulation Language， DML）

数据操纵语言包括数据查询和数据更新两大类操作。数据查询是其核心部分，可以按照一定的条件来获取特定的数据部分呈现给用户；数据更新包括插入、删除和修改操作。数据操纵就是对数据库中的具体数据进行存取操作。

（3） 数据控制语言（Data Control Language， DCL）

数据控制主要包括数据库的安全性和完整性的控制，以及对事务控制的描述。

本章重点介绍数据定义、操纵语言，数据控制语言详见本书第 7 章内容。

4.1.3 SQL 的特点

SQL 之所以能够被用户和业界所接受并成为国际标准，是因为它是一个综合的、功能强大的且又简捷易学的语言。其特点主要体现在下面 5 个方面。

1．一体化

数据库系统的主要功能是通过数据库支持的数据语言来实现的。非关系型的数据语言一般分为数据定义语言和数据操纵语言。它们分别用于定义数据库模式、外模式、内模式和进行数据的存取与处置。这些语言各有各的语法。当用户数据库投入使用后，如果需要修改模式，必须停止现有的数据库的运行，转储数据，修改模式并编译后再重装数据库，十分麻烦。

SQL 语言则集数据定义、操纵和控制功能于一体，语言风格统一，可以独立完成数据库生命周期中的全部活动。

2．高度非过程化

SQL 是一种第四代语言(4GL)，用户只需要提出“做什么”，不需要具体指明“怎么做”，像存取路径的选择和具体处理操作过程均由系统自动完成，不但大大减轻了用户负担，而且有利于提高数据独立性。

3．面向集合

SQL 的数据查询功能涵盖了关系代数的选择、投影、连接等基本运算，这些运算包含传统的集合运算。SQL 语言也采用集合操作方式，不仅操作对象查找结果可以是元组的集合，而且一次插入、删除、更新操作的对象也可以是元组的集合。

4．能以多种方式使用

SQL 语言可以直接以命令方式交互使用，也可以嵌入到到某种高级程序设计语言(如 C、Cobol、Fortran)中去使用。前一种方式适合于非计算机专业人员使用，后一种方式适合于专业计算机人员使用。尽管使用方式不向，但 SQL 语言的语法基本上是—致的。

5．语言简洁，易学易用

尽管 5QL 的功能很强，但语言十分简洁，核心功能只用了 9 个命令（见表 4-1）。SQL

的语法接近英语口语，所以，用户很容易学习和使用。

4.2 数据准备--曲库

本章用一个简化的曲库例子来讲解 SQL 的具体应用。曲库中主要包括以下 3 个表。

（1）歌曲表（表 4-2）

描述了歌词的词曲作者信息，包括歌曲编号、歌曲名称、词作者、曲作者、语言类别。对应的关系模式表示为 Songs(SongID，Name，Lyricist，Composer，Lang)；

（2）歌手表（表 4-3）

描述了歌手的信息，包括歌手编号、歌手姓名、性别、出生日期和籍贯。对应的关系模式表示为 Singers(SingerID，Name，Gender，Birth ，Nation)；

（3）曲目表（表 4-4）

描述了歌手演唱歌曲的信息，包括歌曲编号、歌手编号、专辑名称、曲风类别、发行量和时间。对应的关系模式为 Track(SongID， SingerID，Album ，Style，Circulation，PubYear)；

表 4-2 Songs

SongID	Name	Lyricist	Composer	Lang
S0001	传奇	左右	李健	中文
S0002	后来	施人诚	玉城千春	中文
S0101	Take Me Home, Country Roads	John Denver	John Denver	英文
S0102	Beat it	Michael Jackson	Michael Jackson	英文
S0103	Take a bow	Madonna	Madonna	英文

表 4-3 Singers

SingerID	Name	Gender	Birth	Nation
GC001	王菲	女	1969-07-01	中国
GA001	Michael_Jackson	男	1958-06-11	美国
GC002	李健	男	1974-10-02	中国
GC003	李小东	男	1970-08-12	中国
GA002	John_Denver	男	1943-02-16	美国

表 4-4 Track

SongID	SingerID	Album	Style	Circulation	PubYear
S0102	GA001	SPIRIT	摇滚	20	1983
S0101	GA002	MEMORY	乡村	10	1971
S0001	GC001	流金岁月	流行	5	2003
S0001	GC002	想念你	流行	8	2010
S0002	GA001	GOD BLESS	爵士	NULL	1975

4.3 数据定义

数据库中存在多种数据对象，以 SQL Server 为例，它包含表、视图、索引、存储过程、函数等。**表 4-5** 列出了几乎所有关系数据库管理系统都支持的 3 类主体对象。

表 4-5 SQL 的数据定义语句

操作对象	操作方式		
	创建	删除	修改
表	CREATE TABLE	DROP TABLE	ALTER TABLE
视图	CREATE VIEW	DROP VIEW	
索引	CREATE INDEX	DROP INDEX	

表是关系数据库中的基本对象，关系数据库中的数据都存储在表中。视图是一个虚表，在数据库中只存储视图的定义，而不存放视图的数据，这些数据仍存放在基本表中。索引不是关系模型中的概念，它属于物理实现的范畴，主要是为了能够加快查找速度。视图和索引都要依赖于表才能够存在。

说明： SQL 标准通常不提供修改视图和索引的语句，用户如果要修改这些对象，只能将它们先删除，然后重新建立。

本章主要介绍表的定义，视图和索引的相关定义请参见本书第 11 章的内容。

1. 定义表

（1）定义格式

SQL 语言使用 CREATE TABLE 定义表，其一般格式如下：

```
CREATE TABLE  <表名>  (
    <列名> <数据类型> [列级完整性约束条件]
    [,<列名> <数据类型> [列级完整性约束条件] ]…
    [, <表级完整性约束条件>]
)
```

其中：

◆ <表名>，是所定义的基本表的名字，这个名字最好能表达表的应用语义。比如，歌曲表的表名可以是 Songs。

◆ <列名>是表中所包含的列的名字，<数据类型>指明列的数据类型，一个表可以包含多个列，也就包含多个列定义。

◆ 在定义表的同时还可以定义与表有关的完整性约束条件，这些完整性约束条件都会存储在系统的数据字典中。如果完整性约束只涉及到表中的多个列，则这些约束条件必须在表级处定义，否则既可以定义在列级也可以定义在表级。

说明： SQL 对大小写不敏感！

注意，考虑到编码问题，最好表名和列名都不要用中文命名！

上述格式中的（）是定义的一部分，表示定义的作用范围，必须书写；<>和[]不是定义语句的一部分，只表示特定含义：<>表示括在里面的内容必须包括，[]表示括在里面的内容可以省略。

（2）数据类型

关系数据库的表由列组成，列指明了要存储的数据的含义，同时也要指明存储的数据的类型。因此，在定义表结构时，必然要指明每个列的数据类型。

每种数据库产品所支持的数据类型并不完全相同，而且与标准的 SQL 也有差异。我们这里主要介绍一些主要的数据类型，如**表 4-6** 所示。

表 4-6 常用数据类型

数据类型	**描述**
INT SMALLINT	长整数 短整数
NUMERIC（size，d） REAL DOUBLE FLOAT（size）	定点数，由 size 位数字组成，"d" 规定小数点右侧的最大位数。 取决于机器精度的浮点数 取决于机器精度的双精度浮点数 浮点数，精度至少为 size 位数字
CHAR（size） VARCHAR（size）	容纳固定长度的字符串（可容纳字母、数字以及特殊字符）。 size 规定字符串的最大长度。 容纳可变长度的字符串（可容纳字母、数字以及特殊的字符）。
DATE（yyyy-mm-dd) TIME	日期，包含年、月、日 时间，包含时、分、秒

（3）示例

例 4-1 建立歌手表 Singers

```
CREATE TABLE Singers
(       SingerID CHAR(10) PRIMARY KEY,                  /*定义列级主键*/
        Name VARCHAR(8) NOT NULL,                       /*定义列值不能为空*/
        Gender VARCHAR(2) CHECK(Gender IN (‘男’,’女’)),
        Birth DATE,
        Nation VARCHAR(20)
)
```

例 4-2 建立曲目表 Track

```
CREATE TABLE Track
(       SongID CHAR(10) ,
        SingerID CHAR(15),
        Album VARCHAR(50),
        Style VARCHAR(20),
        Circulation INT,
        PubYear INT,
        PRIMARY KEY(SongID，SingerID)                   /*定义表级主键*/
        FOREIGN KEY(SongID) REFERENCES Songs(SongID),       /*定义外键*/
        FOREIGN KEY(SingerID) REFERENCES Singers(SingerID),   /*定义外键*/
);
```

说明：　完整性约束条件在本书第 7 章有进一步的描述，上述例子包含的约束主要包括：PRIMARY KEY　定义主键（码）；FOREIGN KEY　定义外键；NOT NULL　定义列值不为空

2. 修改表

表建立好后，一般不会再修改，但随着应用环境和需求的变化，偶尔也要修改已建立好的表，SQL 语言用 ALTER TABLE 语句修改表，其一般格式为：

```
ALTER TABLE <表名>
[ ADD    <新列名>  <数据类型>  <完整性约束>]          /*增加新列*/
[DROP COLUMN <列名>]                                /*删除列*/
[ALTER COLUMN <列名>  <数据类型>]                    /*修改列定义*/
[ADD  <完整性约束>]                                  /*添加约束*/
[DROP  <完整性约束>]                                 /*删除约束*/
```

其中<表名>是要修改的表，ADD 子句用于增加新列和新的完整性约束条件，DROP 子句用于删除指定的完整性约束条件，ALTER COLUMN 子句用于修改原有的列定义，包括修改列名和数据类型。

例 4-3 建立歌曲表 Songs 增加一列“翻唱歌曲编号”（DupSongID），其数据类型是字符型。

```
ALTER TABLE Songs ADD DupSongID CHAR(10);
```

说明：　不论基本表中原来是否有数据，新增加的一列值全部为空（NULL）。

例 4-4 删除例 4-3 中刚增加的 “翻唱歌曲编号”（DupSongID）列。

```
ALTER TABLE Songs DROP COLUMN DupSongID;
```

例 4-5 将歌手表中出生日期修改成只存放出生年份的整型。

```
ALTER TABLE Singers ALTER COLUMN Birth INT;
```

例 4-6 增加歌曲名称必须唯一的约束。

```
ALTER TABLE Songs ADD UNIQUE(Name);
```

例 4-7 删除歌手姓名不能取空值的约束。

```
ALTER TABLE Singers DROP NOT NULL(Name);
```

3. 删除表

当某个表不再需要时，可以使用 DROP TABLE 语句删除它。其一般格式为：

```
DROP TABLE <表名>
```

说明：　表一旦被删除，表中的数据及在表上建立的索引和视图都将被自动删除。因此执行删除表的操作时要格外小心！

例 4-8 删除曲目表。

```
DROP TABLE Track;
```

4.4 数据查询

4.4.1 查询语句的基本结构

数据查询是数据库的核心操作，SQL 使用 SELECT 语句进行数据库的查询，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 > [，<表名或视图名>…]
[WHERE < 条件表达式>]
[GROUP BY < 列名列表> [HAVING < 条件表达式>]]
[ORDER BY < 列名> [ASC | DESC]]；

上述基本格式里我们使用了一些特殊符号来表达“实际需要”，这些符号的含义如下。

（1） []：表示[]中的内容是可选的，比如[WHERE < 条件表达式>]，表示可以使用 WHERE 也可以不使用。

（2） <>：表示<>中的内容必须出现。比如< 表名或视图名 >，这个部分表示从哪个地方获取数据，是不可或缺的。

（3） | ：表示选择其一，例如 < 列名 2> [ASC | DESC]]，表示列名 2 后只能用 ASC 或者 DESC 其中之一来进行结果的排序，前者代表升序，后者代表降序。

（4） [，…]：表示括号中的内容可以重复出现 1 至多次。

SELECT 语句由 SELECT 子句、FROM 子句、WHERE 子句、GROUP BY 子句、HAVING 子句和 ORDER BY 子句组成，其中 SELECT 语句和 FROM 子句是必须出现的，其它子句可以根据实际需要选用。

需要**注意**的是，关系代数的操作结果最终还是关系，因此关系代数的运算结果需要满足“元组不能重复”的特性；但是 SELECT 查询结果里是允许重复元组出现的。所以我们说：

SELECT 子句所完成的功能**类似于**关系代数中的投影运算，而 WHERE 子句的功能**类似于**关系代数中的选择运算。

简单地说，一个 SELECT 语句可以对应于下面的关系代数式：

$$\pi_{A1，A2，。。。，An}（\sigma_{F1\wedge F2\wedge。。。\wedge Fn}（R_1\times R_2\times ...\times R_n））$$

4.4.2 查询语句的执行过程

在讲解 SELECT 语句之前，我们需要介绍一下 SELECT 语句的执行过程，这将非常有利于帮助大家来理解后面章节的内容。

例如，为了从表 4-2 中选出所有的中国歌手并显示他们的名字和性别，我们可以采用下面的 SELECT 语句：

SELECT Name, Gender
FROM Singers
WHERE Nation=’中国’

其执行过程如下：

（1）从 Singers 表中取出一条元组 T。

（2）用 WHERE 子句后面的条件来检查元组 T 是丢弃还是保留。如果 T 被保留，则转去执行第（3）步，否则转向第（1）步。

假设，T 的内容为（GC001，王菲，女，1969-0701，中国），由于检查到 T 的 Nation 列

的值等于“中国”，则保留 T 元组，转去执行第（3）步；若 T 的内容为（GA001，Michael_Jackson，男，1958-06-11，美国），由于 T 的 Nation 列的值等于“美国”，则丢弃该元组，转去执行第（1）步；

（3）根据需要投影的信息，挑选 T 的某些列值（有时候需要进一步处理，比如涉及到计算列时）形成最后的结果。比如，此例只要挑选 t 中的姓名和性别这两个列中的值。接着执行第（1）步。

上述过程一直持续，直到把 Singers 表中的所有元组都遍历完毕才结束。

总之， SELECT 语句的直接操作对象是**元组**，SELECT 语句的执行是根据 WHERE 子句中的条件表达式，从 FROM 子句后的基本表或视图中找出满足条件的元组，并按 SELECT 子句中规定的列选出元组中的分量形成结果表。

SELECT 语句既可以完成简单的单表查询，也可以完成复杂的连接查询和嵌套查询。下面将以 4.2 节中的曲库为例来说明它的各种用法。

4.4.3 单表查询

单表查询是指查询仅涉及到一张表。

1. 对列的操作

选择表中的全部列或部分列，这个操作类似于关系代数的投影运算。

（1）查询指定列

在很多情况下，用户只对表中的一部分列感兴趣，这时可以在 SELECT 子句的<目标列表达式>中指定要查询的列。

例 4-9 查询曲库中所有歌手的姓名和籍贯。

分析：歌手的信息存放在表 Singers 中，所以本题要在关系 Singers 中投影出姓名和籍贯，对应的关系代数式为：

$$\pi_{\text{Name，Nation}}（\text{Singers}）$$

在 SQL 语句中，用 FROM 子句指定要操纵的表，SELECT 子句给出要投影的列，本例对应的 SQL 语句如下：

```
SELECT    Name, Nation
FROM    Singers;
```

SELECT 语句的执行结果是一个新表，这个表没有名字，是个临时表，它的关系模式由 SELECT 子句里的属性列（这里是 Name、Nation）构成，查询结果如下：

Name	Nation
王菲	中国
李健	中国
刘若英	中国
Michael Jackson	美国
John Denver	美国

注意， SELECT 子句后的目标列的次序可以与实际表的次序不一致，比如本例中如果要先显示籍贯再显示歌手姓名，其对应的 SQL 语句就改成如下形式：

```
SELECT    Nation, Name,
FROM    Singers;
```

其查询结果也同时变化成如下形式：

Nation	Name
中国	王菲
中国	李健
中国	刘若英
美国	Michael Jackson
美国	John Denver

说明：　从关系的性质角度来讲，这两个查询结果是等价的，但从编程角度来讲，这里存在差别。比如，程序员要访问查询结果的第一条元组（Rows[0]）的 Nation 列，对于前一种情况(Nation 列处在第 2 列)，程序员需要访问的是 Rows[0][1],但是对于后一种情况（Nation 列处在第 1 列），程序员却要访问 Rows[0][0]。

（2）查询全部列

如果要显示表中的所有属性列，可以有两种方式，一种是在 SELECT 子句中列出所有列；另一种是用符号“*”表示所有的列。第一种方式可以按照需要改变显示次序。

例 4-10 查询曲库里的所有歌曲的详细信息。

所有的歌曲信息都存放在表 Songs 中，所以 From 子句后要跟 Songs。

方法一：

```
SELECT　SongID,　Name,　Lyricist, Composer, Lang
FROM　Songs;
```

方法二：

```
SELECT　*
FROM　Songs;
```

其查询结果如下所示：

SongID	Name	Lyricist	Composer	Lang
S0001	传奇	左右	李健	中文
S0002	后来	施人诚	玉城千春	中文
S0101	Take Me Home, Country Roads	John Denver	John Denver	英文
S0102	Beat it	Michael Jackson	Michael Jackson	英文
S0103	Take a bow	Madonna	Madonna	英文

（3）列别名

有时候，我们希望查询结果中的某些列名不同于基本表中的列名，这时，可以在 SELECT 子句中增加列别名。

SQL 语句使用 AS 关键词对列设置“别名”。AS 使用格式如下：

旧名 AS 别名

例 4-11 查询曲库里的所有歌曲的详细信息，请把表中的各个列名：SongID，Name，Lyricist，Composer，Lang 分别替换成：歌曲编号、歌曲名称、词作者、曲作者、语言类别。

```
SELECT　SongID AS 歌曲编号, Name AS 歌曲名称, Lyricist AS 词作者,
        Composer AS 曲作者, Lang AS 语言类别
FROM　Songs;
```

执行结果如下：

歌曲编号	歌曲名称	词作者	曲作者	语言类别
S0001	传奇	左右	李健	中文
S0002	后来	施人诚	玉城千春	中文
S0101	Take Me Home, Country Roads	John Denver	John Denver	英文
S0102	Beat it	Michael Jackson	Michael Jackson	英文
S0103	Take a bow	Madonna	Madonna	英文

可以看到，与**例 4-10** 的查询结果不同，查询后的列名具有一定的语义，可读性增强。同时，列别名还隐藏了真正的表结构的列名，有利于提高数据的安全性。列别名还有一个频繁使用的场所——跟计算列的联合使用。

（4）查询经过计算的列

SELECT 子句的<目标列表达式>不仅可以是表中真实存在的列，也可以是一个表达式。表达式是由运算符将常量、列、函数连接而成的有意义的式子。表达式列也可称为“计算列”，计算列在表中并不存在，是一个虚拟的列，它没有对应的列名，所以“计算列”往往要跟“列别名”联合起来使用。

◆ **常量和计算表达式：**

例 4-12 查询曲目表中歌曲编号、歌手编号和发行量，发行量的计量单位是万。

方法一：

```
SELECT SongID, SingerID, Circulation*10000 as NewCirculation
FROM    Track
```

方法二：

```
SELECT SongID, SingerID, Circulation, '万' as Unit
FROM    Track
```

方法一的执行结果对应（a），方法二的执行结果对应（b）。

分析：曲目表中的“发行量”列没有体现计量单位，为了体现“万”，方法一用了一个计算表达式 volume*10000，因此它的查询结果中第 3 列不再对应原来列的值，而是在原来列的值的基础上都乘上了 10000。。

方法二的查询结果中“NewCirculation”列中的数据跟原来的“Circulation”列的内容相同，但是比原表多了一个“Unit”列，这一列的值是一个字符串常量，所有元组对应此列的值都将用这个常量“万”去填充。

SongID	SingerID	NewCirculation
S0102	GA001	200000
S0101	GA002	100000
S0001	GC001	80000
S0001	GC002	50000

(a)

SongID	SingerID	NewCirculation	*Unit*
S0102	GA001	20	万
S0101	GA002	10	万
S0001	GC001	8	万
S0001	GC002	5	万

(b)

说明：　在 SELECT 语句中字符串常量一般都要用单引号括起来，比如前面的'万'。

◆ **函数**

例 4-13：查询歌手表中歌手的名字、出生日期和年龄，假设现在的年度是 2012 年。

分析：歌手表中并没有年龄这个字段，但是有出生日期字段，为了得到每个歌手的年龄，需要用现在年度减去他的出生年份。

```
SELECT Name as 姓名, Birth 出生日期, 2012-Year（Birth） as 年龄
FROM Singers
```

这里 **Year** 是一个日期函数，可以把一个日期型变量的年份解析出来，所以其执行结果如下所示：

姓名	出生日期	年龄
王菲	1969-07-01	43
Michael_Jackson	1958-06-11	54
李健	1974-10-02	38
李小东	1970-08-12	42
John_Denver	1943-02-16	69

（5）过滤选择列中的重复元组

两条本来并不完全相同的元组，投影到某些列上后，可能变成相同的行了，可以用 DISTINCT 关键字过滤它们。

例 4-14 列出曲目表 Track 中的所有的曲风类别。

```
SELECT  Style
FROM  Track;
```

查询结果是如（a）所示。可以看到“流行”style 出现了 2 次。

去掉结果表中的重复行，必须指定 **DISTINCT** 关键词：

```
SELECT  DISTINCT Style
FROM  Track;
```

执行结果如（b）所示，每种 style 只出现 1 次。

Style
摇滚
乡村
流行
流行

(a)

Style
摇滚
乡村
流行

(b)

2. 对元组的操作--WHERE 子句

WHERE 子句用于表达关系代数中选择运算的选择条件。WHERE 子句常用的运算符如表 4-7 所示。

表 4-7 常用查询条件的运算符

查询条件	谓词
比较	= ，>，<，>=，<=，!=，<>
确定范围	BETWEEN AND ，NOT BETWEEN AND
字符匹配	LIKE， NOT LIKE
确定集合	IN ，NOT IN
空值	IS NULL， IS NOT NULL
逻辑运算符	AND， OR ，NOT

（1）比较大小

比较运算符包括：=（等于），>（大于），<(小于)，>=（大于等于），<=（小于等于），!=或者<>（不等于）。

例 4-15 查询中国歌手的详细信息。

歌手的详细信息都在表 Singer 中，Nation 列表示歌手的国籍，歌手表中既有中国歌手又有美国歌手，题目只需要查询中国歌手的信息，因此需要对表里的元组进行选择操作，选择条件是 Nation='中国'，因为需要歌手的详细信息，即需要投影所有列，所以对应的关系代数为：

$$\sigma_{\text{Nation='中国'}}(\text{Singers})$$

对应的 SQL 语句为：

```
SELECT   *
FROM     Singers
WHERE    Nation='中国'
```

例 4-16 查询出生日期早于 1967 年的歌手的姓名和出生日期。

与**例 4-15** 不一样的地方是不需要显示歌手的详细信息，只要姓名和出生日期两个字段，另外查询条件也限定了出生日期<1967，因此关系代数为：

$$\pi_{\text{Name,BirthYear}}(\sigma_{\text{Birth<1967}}(\text{Singers}))$$

对应的 SQL 语句为：

```
SELECT Name, Birth
FROM   Singers
Where Birth<1967
```

（2）确定范围

谓词 BETWEEN…AND 用于查询列值在指定范围内]的元组，其中 BETWEEN 后面跟的是范围下限，AND 后跟的是范围的上限。

例 4-17 查询出生日期在 1970~1980 之间的歌手的姓名和性别。

关系代数：

$$\pi_{\text{Name,Gender}}(\sigma_{\text{Birth}\geq 1970 \wedge \text{Birth}\leq 1980}(\text{Singers}))$$

SQL 语句为：

```
SELECT Name, Gender
FROM Singers
WHERE Birth BETWEEN 1970 AND 1980;
```

与 BETWEEN…AND 相对的谓词表达式是 NOT BETWEEN…AND。 如果要求查询出生年份不在 1970~1980 之间的歌手的姓名与性别，则对应的关系代数和 SQL 分别如下所示：

$$\pi_{\text{Name,Gender}}（\sigma_{\text{Birth<1970}\vee\text{Birth>1980}}（\text{Singers}））$$

```
SELECT Name, Gender
FROM Singers
WHERE Birth NOT BETWEEN 1970 AND 1980;
```

（3）字符匹配

谓词 LIKE 可以用来进行字符串的匹配。 其一般语法格式如下：

列名 [NOT] LIKE ‘<匹配串>’ [ESCAPE ‘<换码字符>’]

其含义是查找列值与<匹配串>相匹配的元组。通常，<匹配串>中可以使用通配符“%”（百分号）和“_”（下横线）。其中：

%：代表任意长度（长度可以为 0）的字符串。例如，a%b 表示以 a 开头、以 b 结尾的任意长度的字符串。字符串 ab、axb、agxb 都满足该匹配串。

_：代表任意单个字符。例如，a_b 表示以 a 开头，以 b 结尾，中间夹一个字符的任意字符串。如 axb、a!b 等都满足要求。

例 4-18 查询歌手编号以“GC”开头的歌手姓名、性别。

```
SELECT Name, Gender
FROM Singers
WHERE SingerID like ‘GC%’
```

例 4-19 查询姓“李”且全名为 2 个汉字的歌手姓名。

```
SELECT Name
FROM Singers
WHERE Name like ‘李_’
```

因为这里查询的是李姓歌手且全名只能为 2 个汉字的歌手，所以本例用的是’李_’，所以只会筛选出表 4-3 中的名字列是“李健”的元组；而如果用的是’李%’，则“李健”和“李小东”两个歌手会全部查询出来。

如果用户要查询的字符串本身就含有%或_，这时就要使用’ESCAPE<换码字符>’对通配符进行转义了。

例 4-20 查询以”John_”开头的歌手的详细信息。

```
SELECT *
FROM Singers
WHERE Name like ‘John\_%’ ESCAPE ’\’;
```

这样的匹配串中紧跟在\后面的字符“_”不再具有通配符的含义，转义为普通的“_”。

（4）确定集合

谓词 IN 用来查找某个列值属于指定集合的元组，格式为：

列名 IN 集合

如果一个元组在指定列上的值出现在集合中，则该元组满足选择条件。在 SQL 中，集合用（）表示，里面的元素用“,”分割。

例 4-21 查询曲目表中属于“乡村”,“爵士”和“摇滚”曲风的专辑详细信息。

```
SELECT *
FROM Track
WHERE Style    IN    (‘乡村’, ‘爵士’, ‘摇滚’);
```

其查询结果如下：

SongID	SingerID	Album	Style	Circulation	PubYear
S0102	GA001	SPIRIT	摇滚	20	1983
S0101	GA002	MEMORY	乡村	10	1971
S0002	GA001	GOD BLESS	爵士	NULL	1975

与 IN 相对的谓词是 NOT IN， 用于查找属性值不属于指定集合的元组。

例 4-22 查询曲目表中既不属于“乡村”、“爵士”也不是“摇滚”曲风的专辑详细信息。

```
SELECT    SongID, SingerID
FROM    Track
WHERE    Style NOT IN (‘乡村’, ‘爵士’, ‘摇滚’);
```

其查询结果如下：

SongID	SingerID	Album	Style	Circulation	PubYear
S0001	GC001	流金岁月	流行	5	2003
S0001	GC002	想念你	流行	8	2010

（5）空值的判断

谓词 IS NULL 用于判断某列的值是否为空值，格式如下：

列名 IS NULL

例 4-23 查询曲目表中缺少发行量信息的记录。

```
SELECT *
FROM    Track
WHERE Circulation IS NULL;
```

其查询结果如下：

SongID	SingerID	Album	Style	Circulation	PubYear
S0002	GA001	GOD BLESS	爵士	NULL	1975

说明： “IS NULL”不能用“=NULL”代替。

（6）逻辑运算

逻辑运算符 AND 和 OR 可以用来联结多个查询条件。AND 的优先级高于 OR，但用户可以用括号来改变优先级。

例 4-24 查询曲目表中曲风为“摇滚”或者发行量大于 10，并且发布时间早于 1980 年的曲目信息。

$$\pi_{SongID,SingerID,Album,Style,Circulation,PubYear}$$

$$(\sigma_{(Style='摇滚' \vee Circulation>10 \wedge PubYear<1980)} (Singers))$$

```
SELECT *
FROM Track
WHERE Style='摇滚' OR (Circulation>10  and  PubYear<1980);
```

3. 排序--ORDER BY 子句

由 SELECT-FROM-WHERE 构成的 SELECT 查询语句完成对表的选择和投影操作，得到一个新的结果表，还可以对得到的新表做进一步的操作。

ORDER BY 子句用于对查询结果进行排序。可以按照一个或者多个属性列进行升序（ASC）或者降序(DESC)排列，默认将按照升序排列。

注意： ORDER BY 子句往往处在 SELECT 语句的最下方

例 4-25 查询曲目表中的歌手编号、歌曲编号和曲风类别，将查询结果按照发行时间降序排列。

```
SELECT SongID, SingerID, Style
FROM  Track
ORDER BY PubYear DESC;
```

例 4-26 查询曲目表中的歌手编号、歌曲编号和曲风类别，将查询结果按照曲风类别和发行时间降序排列。

```
SELECT SongID, SingerID, Style
FROM Track
ORDER BY Style DESC, PubYear DESC;
```

例 4-25 和**例 4-26** 的查询结果分别对应下图（a）和（b）。

(a)中的结果仅按 PubYear 列降序排列；(b)中的结果首先要按照 style 字段降序排列（拼音次序），在 style 等同的情况下再按 PubYear 降序排列，参见（b）中 style=“流行”的元组。

SongID	SingerID	Style	PubYear
S0001	GC002	流行	2010
S0001	GC001	流行	2003
S0102	GA001	摇滚	1983
S0002	GA001	爵士	1975
S0101	GA002	乡村	1971

(a)

SongID	SingerID	Style	PubYear
S0102	GA001	摇滚	1983
S0101	GA002	乡村	1971
S0001	GC002	流行	2010
S0001	GC001	流行	2003
S0002	GA001	爵士	1975

(b)

4. 汇总--聚集函数

我们有时要对查询结果做一些简单的统计工作，SQL 提供了如表 4-8 所示的若干聚集函数来完成此项任务。

表 4-8 聚集函数

函数	描述
COUNT（[DISTINCT|ALL]*）	计算元组的个数
COUNT（[DISTINCT|ALL]<列名>）	对一列中的值计算个数
SUM（[DISTINCT|ALL]<列名>）	求某一列值的总和（此列的值必须是数值）
AVG（[DISTINCT|ALL] <列名>）	求某一列值的平均值（此列的值必须是数值）
MAX（[DISTINCT|ALL] <列名>）	求某一列值中的最大值
MIN（[DISTINCT|ALL] <列名>）	求某一列值中的最小值

如果指定 DISTINCT 标识符，则表示计算时要取消指定列中的重复值。如果不指定 DISTINCT 或指定 ALL（ALL 为缺省），则表示不取消重复值。

例 4-27 查询 Track 表中的记录总数，最早发行时间、最近发行时间和整个发行量的总和。

SELECT Count(*) as 记录总数, MIN(PubYear) as '最早发行量', MAX(PubYear) as '最近发行时间', SUM(Circulation) as '发行量总和'

FROM Track

对表 4-3 的执行结果如下：

记录总数	最早发行时间	最近发行时间	发行量总和
5	1971	2010	43

此例出现了 4 个聚集函数，但每个聚集函数只返回一个值，所以执行结果只有 1 行但是有 4 个统计数字。

说明： 表 4-4 中 Circulation 字段的值有 NULL 值，在聚集函数遇到空值时，除 COUNT(*)外，都跳过空值而值处理非空值。

注意，聚集函数不能直接出现在 WHERE 子句中。

例如，查询年龄最大的歌手姓名，如下写法是错误的：

SELECT Name FROM Singers WHERE Birth=MIN(Birth)

正确的写法要使用子查询，请参见**例 4-34**。

5. 分组--GROUP BY 子句

聚合函数是对查询结果（一个元组集）中的值进行统计。在有的查询中，我们要把具有相同特征的元组分成若干子集，然后需要再对每个子集中的值进行统计，此时就要用到 SELECT 句型中的分组子句“GROUP BY”，格式为

GROUP BY <列名列表>

列名列表描述了分组的依据，同一组元组在这些列上的值一定相同。列名列表又称为**分组列**。

例 4-28 统计歌曲表(表 4-2)中各个语种的歌曲数量。

SELECT Lang 语种, COUNT(*) as 数量

FROM Songs

GROUP BY Lang

执行结果如下：

语种	数量
英文	3
中文	2

此例执行时，先根据 Lang 属性值进行分组，获得“中文”和“英文”两组值，然后再分别对每组统计个数。

需要注意的是，下列查询语句是**错误**的：

```
SELECT    SongID, COUNT(*) as 数量
FROM Songs
GROUP BY Lang
```

在用分组语句时，SELECT 后跟的列只能是聚集函数或者是出现在 GROUP BY 之后的分组列。因为 SongID 既不是聚集函数，也没有在 GROUP BY 之后出现，以上语句就不能正确执行。

6. HAVING 子句

对分组进行选择操作由 HAVING 子句完成。虽然也具有选择作用，但是应当要把 HAVING 子句与 WHERE 子句区别开来，WHERE 子句对 SELECT-FROM 语句查询的元组进行选择，从中选出满足条件的元组。HAVING 子句作用与 SELECT-FROM-GROUP BY 语句后得到的分组，再进一步进行选择满足条件的元组。它们的作用对象不同。

HAVING 子句的格式同 WHERE 子句:

HAVING <条件表达式>

但其条件表达式是由分组列、聚集函数和常数构成的有意义的式子。

例 4-29 统计歌曲表中歌曲数量超出 2 个的各个语种的歌曲数量。

与**例 4-28 例 4-27 例 4-28** 不同，按照语种分组后，本例不显示所有语种对应的歌曲数量，它只显示这样的分组：包含的歌曲数量大于 2。

```
SELECT    Lang 语种, COUNT(*) as 数量
FROM Songs
GROUP BY Lang
HAVING COUNT(*)>2
```

其执行结果如下所示：

语种	数量
英文	3

4.4.4 多表查询

前面章节的内容介绍了单表查询，单表查询的特点是 FROM 子句后面值出现一个表名。进行数据库设计时，由于规范化、数据的一致性和完整性的要求，每个表中的数据都是有限的，所以数据库中的各个表都不是孤立的，存在一定的关系。这时就需要将多个表连接在一起，进行组合查询数据。

1. SQL 中的笛卡尔积和连接

SQL 在一个查询中建立几个表的联系的方法非常简单，只要在 FROM 子句中列出所有

涉及到的表就可以了。从概念上讲，FROM 子句先对这些表做笛卡尔积操作，得到一个临时表，以后的选择、投影等操作都是针对这个临时表，从而将多表查询转换为单表查询。连接查询中的 WHERE 子句中用来连接两个表的的条件称为连接条件或者连接谓词，其一般格式为：

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

其中比较运算符主要有:= 、>、 <、 >=、<=、!=或者<>等。当比较运算符为“=”时为等值连接，若在目标列中去掉相同的属性，则为自然连接。

连接谓词中的列名称为连接属性，连接属性必须是可以比较的，但不必相同。当连接表具有相同的列名时，为了不引起混淆，引用时采用下面的方式：

<表名>.<列名或*>

连接查询是关系数据库中最主要的查询，主要包括内连接、外连接和交叉连接等。由于交叉连接使用得很少，并且其查询结果也没有太大的实际意义，所以不作介绍。本书重点介绍内连接和外连接。

2. 内连接

使用内连接时，如果两个表的相关字段满足连接条件，则从这两个表中提取数据并组合成新的元组。在非 ANSI 标准的实现中，连接是在 WHERE 子句中执行的（即在 WHERE 子句中指定表间的连接条件）；在 ANSI SQL-92 中，连接是在 JOIN 子句中执行的。前者称为 Theta 连接，后者称为 ANSI 连接。

ANSI 内连接的语法格式：

FROM 表 1 [INNER] JOIN 表 2 ON <连接条件>

例 4-30 列出曲库中所有演唱过歌曲的歌手名，歌曲名和发行量。

分析：歌手名、歌曲名和发行量分别存放在歌曲表、歌手表和曲目表中，所以此例涉及到 3 个表，其中歌曲表和曲目表以公共字段 SongID 相关联，歌手表和曲目表以公共字段 SingerID 字段相关联。根据各属性语义可以写出下列关系代数：

$$\pi_{\text{Songs.Name,Singers.Name,Circulation}}(\text{Songs} \bowtie \text{Track} \bowtie \text{Singers})$$

其 Theta 方式的 SQL 语句非常简单：

```
SELECT Songs.Name AS 歌曲名, Singers.Name AS 歌手名,Circulation AS 发行量
FROM Songs,Track,Singers
WHERE Songs.SongID=Track.SongID AND Track.SingerID=Singers.SingerID
```

其 ANSI 方式的 SQL 语句为：

```
SELECT Songs.Name AS 歌曲名, Singers.Name AS 歌手名,Circulation AS 发行量
FROM Songs JOIN Track ON Songs.SongID=Track.SongID
JOIN Singers ON Track.SingerID=Singers.SingerID
```

两种方式的查询结果都如下所示：

歌曲名	歌手名	发行量
Beat it	Michael_Jackson	20
Take Me Home, Country Roads	John_Denver	10
传奇	王菲	5
传奇	李健	8
后来	Michael_Jackson	NULL

说明：　本例出现了 2 个 Name 列，在写 SQL 语句时必须通过添加表名前缀来区分这些列。

3. 自身连接

连接操作不仅可以在两个表之间进行，也可以与其自身进行连接，称为表的自身连接。

例 4-31 列出所有互为翻唱的歌曲。假设互为翻唱歌曲的条件是：歌名相同，词作者相同但曲作者不同的元组。

分析：所有的歌曲信息都在歌曲表中，我们要对比两首歌曲是否互为翻唱，就需要 Songs 表与其自身连接。因为相连接的 2 张表相同，为了区分它们，需要用到表别名的概念。

可以为表提供别名，其格式如下：

FROM <源表名> [AS] <表别名>

此题为参加连接的两个 Songs 表分别取名为 First 和 Second。（为了便于对比，省略了表 4-2 所示的部分字段）

First 表

Name	Lyricist	Composer
传奇	左右	李健
传奇	王菲	李健
后来	施人诚	玉城千春
Take a bow	Madonna	Madonna

Second 表

Name	Lyricist	Composer
传奇	左右	李健
传奇	王菲	李健
后来	施人诚	玉城千春
Take a bow	Madonna	Madonna

完成该查询的 Theta 形式的 SQL 语句为：

```
SELECT    DISTINCT First.Name, First.Lyricist, First.Composer
FROM    Songs AS First, Songs AS Second
WHERE    First.Name= Second.Name AND First.Composer= Second. Composer
AND First.Lyricist<>Second.Lyricist
```

ANSI 形式的 SQL 语句为：

```
SELECT DISTINCT First.Name, First.Lyricist, First.Composer
FROM Songs AS First JOIN Songs AS Second
ON First.Name= Second.Name
WHERE First.Composer= Second. Composer    AND First.Lyricist<>Second.Lyricist
```

查询结果如下：

Name	Lyricist	Composer
传奇	左右	李健
传奇	王菲	李健

这两条记录，名字相同，词作者相同但是曲作者不同，所以是互为翻唱歌曲。

4. 外连接

在内连接操作中，只有满足连接条件的元组才能作为结果输出，但有时我们也希望输出那些不满足连接条件的元组信息，比如查看所有歌曲被演唱的情况，包括曲目表中有歌手演唱过的歌曲，还有那些暂时没人演唱的歌曲。如果用内连接实现（通过 Songs 表和 Track 表的内连接），则只能查询到有歌手演唱过的歌曲，因为内连接的结果要满足连接条件：

Songs.SongID=Track.SongID。对于在 Songs 表中存在，但是没有在 Track 表中出现的歌曲，由于不满足该条件，则查询不出。这种情况就需要使用外连接来实现。

外连接之限制一张表中的数据必须满足连接条件，而另一张表中的数据可以不满足连接条件。

Theta 方式的外连接语法格式为：

左外连接：FROM 表 1，表 2 WHERE [表 1.]列名(+)=[表 2.]列名

右外连接：FROM 表 1，表 2 WHERE [表 1.]列名=[表 2.]列名(+)

ANSI 方式的外连接语法格式为：

FROM 表 1 LEFT | RIGHT [OUTER] 表 2 ON <连接条件>

SQL SERVER 支持 ANSI 方式的外连接，但 ORACLE 支持的是 Theta 方式的外连接。

例 4-32 查询歌曲被演唱的情况，包括被演唱的和没有被演唱的，最终投影出歌曲编号、歌曲名称、所在专辑名称、发行时间。

Theta 方式：

```
SELECT SongID, Name, Album, PubYear
FROM Songs, Track
WHERE Songs.SongID(+)=Track.SongID
```

ANSI 方式：

```
SELECT Songs .SongID, Name, Album, PubYear
FROM Songs
LEFT JOIN Track ON Songs.SongID=Track.SongID
```

查询结果：

SongID	Name	Album	PubYear
S0001	传奇	流金岁月	2003
S0001	传奇	想念你	2010
S0002	后来	GOD BLESS	1975
S0101	Take Me Home, Country Roads	MEMORY	1971
S0102	Beat it	SPIRIT	1983
S0103	Take a bow	NULL	NULL

其中最后一条歌曲在 Track 表中并没有记录，所以它在 Album 和 PubYear 列的值就用 NULL 去替代。

此查询也可以用右外连接去实现，如下所示：

```
SELECT Songs .SongID, Name, Album, PubYear
FROM Track
RIGHT JOIN Songs ON Songs.SongID=Track.SongID
```

该语句执行结果与左外连接相同。

5. 子查询

在实际应用中，经常有一些 SELECT 语句需要使用其他 SELECT 语句的查询结果，此时就需要用到子查询。

子查询就是嵌套在另一个查询语句（SELECT）中的查询语句（SELECT），因此，子查询也称为嵌套查询。外部的 SELECT 语句称为外围查询（父查询），内部的 SELECT 语句称为子查询。子查询的结果将作为外围查询的参数，这种关系就好像是函数调用嵌套，将嵌套函数的返回值作为调用函数的参数。

虽然子查询和连接可能都要查询多个表，但子查询和连接不一样。因为它们的语法格式不一样，使用子查询最符合自然的表达查询方式，书写更容易。

根据子查询的执行是否依赖父查询，我们可以把子查询分为无关子查询和相关子查询。

（1）无关子查询

无关子查询是子查询的查询条件不依赖于父查询。它的执行过程是由里向外逐层处理。即每个子查询都会在它的上一级父查询处理之前求解。子查询的结果用于建立其父查询的查找条件。

◆ **语法格式**

其格式如下：

```
SELECT <列名> [,<列名>] …
FROM <表名>
WHERE <表达式> <[NOT] IN | 其它比较运算符>
(
    SELECT <列名>
    FROM <表名>
    WHERE <条件>
)
```

在无关子查询中，[NOT]IN 表示进行的是集合运算，而“其它运算符”进行的是比较运算。

◆ **无关子查询的集合运算**

通过运算符 IN 或 NOT IN，子查询可以进行集合运算。这种形式的子查询是分步骤实现的，即先执行子查询，然后根据子查询返回的结果——某个集合，再执行外层父查询。

使用 IN 运算时，如果表达式的值与集合中的某个值相等，则此运算结果为真；如果表达式的值与集合中的所有值均不相等，则运算结果为假。

例 4-33 查询歌唱风格是“摇滚”的歌手信息。

```
SELECT *
FROM Singers
WHERE SingerID IN
(
    SELECT DISTINCT SingerID
    FROM Track
    WHERE Style =’摇滚’
)
```

说明：由关键字 IN 引入的子查询的 SELECT 后的列名只允许有 1 项内容，即只能是一个列名或者是表达式。

◆ **无关子查询的比较运算**

使用子查询进行比较运算时，通过比较运算符（=、<>、<、>、<=、>=），将一个表达式的值与子查询返回的值进行比较。如果比较运算的结果为真，则比较运算返回 **TRUE**。

例 4-34 查询年龄最小的歌手的信息

```
SELECT *
FROM Singers
WHERE Birth =
(
    SELECT MAX(Birth)
    FROM Singers
)
```

上面的例子先执行内部的子查询，把最大 Birth 的值取到后，再返回给父查询进行查询。

说明：　使用子查询进行比较测试时，要求子查询语句必须是返回单值的查询语句。

（2）相关子查询

相关子查询要使用到父查询的数据。

◆ **基本语法结构**

相关子查询通常使用 EXISTS 谓词，其形式为：

```
SELECT <列名> [,<列名>] …
FROM <表名>
WHERE
WHERE [NOT] EXISTS
(   SELECT *   FROM <表名>   WHERE <条件>    )
```

带有 EXISTS 谓词的子查询不返回任何数据，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

若内层查询结果为空，则外层的 WHERE 子句返回假值

由 EXISTS 引出的子查询，其目标列表达式通常都用“* ”，因为带 EXISTS 的子查询只返回真值或假值，给出列名无实际意义。

带 NOT EXISTS 谓词的含义是：

若内层查询结果非空，则外层的 WHERE 子句返回假值

若内层查询结果为空，则外层的 WHERE 子句返回真值

对于**例 4-33**，我们也能用下列相关子查询来实现：

```
SELECT *
FROM Singers X
WHERE EXISTS
(
    SELECT *
    FROM Track
    WHERE Track.SingerID=X.SingerID and   Style =’摇滚’
)
```

◆ **相关子查询执行过程**

相关子查询的执行过程比较复杂，总体执行过程如下：首先取外层父查询中的第一个元组，根据它与内层子查询的相关属性值处理内层子查询，若 WHERE 子句返回值为真，则取此元组放入结果表；然后，再取外层父查询的下一个元组；重复这一过程，直至外层表全部检查完为止。

我们采用**例 4-33** 的相关子查询来进一步描述这个过程：

①执行父查询，顺序扫描 Singers 表

②取出 Singers 表中第一个元组，存放到元组变量 X 中。

③执行父查询的 WHERE 子句

（i）把上一步得到的 X 的值传送到子查询里

（ii）执行子查询，得到一个集合

（iii）判断集合是否为空

④如果返回为空，那么 WHERE 条件为 false，不输出结果；如果不为空，则 WHERE 条件为 true，则输出 X.*

⑤按照上面步骤，继续处理 Singers 表的下一个元组，直到处理完 Singers 的所有元组。

◆ **一个复杂的相关子查询实例**

例 4-35 查询至少演唱过 Michael_Jackson 演唱过的全部歌曲的其他歌手的信息。

用 R 表示 Michael_Jackson 演唱过的全部歌曲的集合，S 表示歌手 x 演唱的全部歌曲的集合，如果 R⊆S 成立，则 x 就是我们要找的歌手。

由于 SQL 不提供“⊆”运算符，因此需要进行逻辑变换：如果 R⊆S 成立，则 R-S 为空集，即 NOT EXISTS(R-S)为真。所以对应的 SQL 语句如下：

```
SELECT SingerID
FROM    Singers    x
WHERE NOT EXISTS
(
      (                        --集合 R
          SELECT SongID
          FROM Track JOIN Singers ON Track. SingerID =Singers.SingerID
          Where    Singers.Name=' Michael_Jackson'
      )
      EXCEPT
      (                         --集合 S
          SELECT SongID
          FROM Track WHERE Track.SingerID=x.SingerID
      )
)   AND    x.Name<>' Michael_Jackson'
```

上面的 SQL 语句还可以进一步精简， R⊆S 的逻辑语义是：

$(\forall t)(t \in R \rightarrow t \in S) = (\nexists t)(t \in R \wedge t \notin S)$

如果把 EXISTS 理解为存在量词，则可以写出与上述 SQL 语句不同但执行结果相同的 SQL 语句：

```
SELECT SingerID
FROM    Singers    x
WHERE NOT EXISTS
(
   SELECT SongID        --集合 R
   FROM Track AS y
   JOIN Singers ON Track. SingerID =Singers.SingerID
   WHERE    Singers.Name=' Michael_Jackson' AND NOT EXISTS
                                                (
                                               SELECT SongID     --集合 S
                                               FROM Track
                                               WHERE SingerID=x.SingerID
                                               AND SongID=y.SongID
                                               )
)   AND    x.Name<>' Michael_Jackson'
```

（3）不同形式的查询间的替换

一些带 EXISTS 或 NOT EXISTS 谓词的相关子查询不能被其他形式的子查询等价替换，比如**例 4-35**。

但所有带 IN 谓词、比较运算符的无关子查询都能用带 EXISTS 谓词的子查询等价替换。

例 4-36 查询与“王菲”同一个国籍的歌手的详细信息。

```
SELECT    *
FROM Singers X
WHERE Nation IN
(
        SELECT    Nation
FROM    Singers    Y
        WHERE Name='王菲'
)
```

可以把上例用 EXISTS 改写成：

```
SELECT *
FROM Singers X
WHERE EXISTS (
        SELECT *
FROM    Singers    Y
        WHERE Y.Name='王菲'    AND    Y.SingerID=X.SingerID
)
```

4.4.5 集合查询

SELECT 查询操作的对象是集合，结果也是集合。集合操作主要包括并操作 UNION，交操作 INTERSECT 和差操作 EXCEPT。

注意：　参加集合操作的各查询结果的列数必须相同；对应项的数据类型也要相同。

例 4-37 查询中国歌手和出生日期晚于 1960 年的歌手。

```
SELECT *
FROM Singers
WHERE Nation='中国'
UNION
SELECT *
FROM Singers
WHERE Birth>=1960
```

使用 UNION 将多个查询合并起来时，系统会自动过滤掉重复元组。

例 4-38 查询中国歌手和出生日期晚于 1960 年的歌手的交集。

```
SELECT *
FROM Singers
WHERE Nation='中国'
INTERSECT
SELECT *
FROM Singers
WHERE Birth>=1960
```

这实际就是查询出生日期晚于 1960 年的中国歌手信息，等同于下面语句

```
SELECT *
FROM Singers
WHERE Nation='中国' AND Birth>=1960
```

例 4-39 查询中国歌手和出生日期晚于 1960 年的歌手的差集。

```
SELECT *
FROM Singers
WHERE Nation='中国'
EXCEPT
SELECT *
FROM Singers
WHERE Birth>=1960
```

也就是查询早于 1960 年出生的中国歌手

```
SELECT *
FROM Singers
WHERE Nation='中国' AND Birth<1960
```

4.5 数据更新

SQL 提供的数据库修改操作主要有元组插人、删除和修改，通称为数据库更新。

4.5.1 插入操作

元组插入语句的一般格式：

INSERT INTO <表名>
(<列名 1> [， <列名 2>] …)
VALUES (<常量 1> [, <常量 2>]…)

或者

INSERT INTO <表名>
(<列名 1> [， <列名 2>] …)
<子查询>

第一种格式是单元组的插入，第二种是多元组的插入。写 INSERT 语句时要注意下面几点：

- ◆ 在第一种格式中，VALUES 子句中列出的常量的次序要与列出的列名的次序相一致。
- ◆ 若表中的某些属性在插入语句中没有出现，则这些属性最常见的是设置为空值，也可以用默认值填入。但是，对于在表的定义中说明为 NOT NULL 的属性，则不能用 NULL 值。
- ◆ 若插入的语句中没有列出<列名>，则新纪录必须在每个分量上均有值，且这些值的排列次序要与表中定义的列的次序严格一致。

下面分别举例说明。

例 4-40 将歌手“杨坤”的信息插入到歌手表中。

```
INSERT INTO Singers(SingerID, Name, Gender, Nation, Birth)
VALUES('GA003', '杨坤', '男', '中国' , '1972-12-18')
```

例 4-41 假设另有一张歌手表 S，表结构与 Singer 一致，现在要求把 S 表中所有没有在 Singer 表里出现过的男歌星加入到 Singer 表中。

```
INSERT INTO Singers (SingerID, Name, Gender, Nation, Birth)
（
    SELECT   (SingerID, Name, Gender, Nation, Birth)
    FROM S
    WHERE S.Gender='男' and   NOT EXISTS (S.SingerID=Singers.SingerID)
)
```

4.5.2 删除操作

SQL 的删除操作是指从关系中删除元组，而不是从元组中删除某些属性值，也不是删除表结构。SQL 删除语句的格式：

DELETE FROM <表名>
［WHERE <条件>］

注意：　如果没有 WHERE 子句，将会删除表中所有数据，所以执行 DELETE 语句要非常小心！

例 4-42 删除歌手表中不是中国国籍的歌手

```
DELETE　FROM　Singers
WHERE　Nation<>'中国'
```

例 4-43 删除所有非中国歌手的曲目信息。

```
DELETE　FROM　Track
WHERE　SingerID　IN (
　　SELECT SingerID FROM Singers WHERE Nation<>'中国'）
```

4.5.3 修改操作

当需要修改关系中元组的某些值时，可以用 UPDATE 语句实现。SQL 的 UPDATE 语句的格式：

```
UPDATE <表名>
SET　<列名>=<值表达式>［，<列名>=<值表达式>］…
[WHERE <条件>]
```

该语句的功能是修改表中满足条件表达式的元组中的指定属性值，SET 子句指出元组中要修改哪些列，修改一个列就要给出一个相应的“<列名>=<值表达式>”，其含义是使列的新值等于<值表达式>的值。值表达式中可以出现常数、列名、系统支持的函数及运算符。

注意：　如果省略 WHERE 子句，将会修改表中所有的元组，所以进行此操作的时候也要非常小心！

例 4-44 将 Singers 表中歌手编号为“GC002”的 Birth 属性值修改为 1978。

```
UPDATE Singers
SET Birth＝1978
WHERE SingerID＝' GC002'
```

例 4-45 把 Track 表中所有流行歌曲的发行量（Circulation）提高 10％。

```
UPDATE Track
SET Circulation＝Circulation *1.1
WHERE Style='流行'
```

例 4-46 当 Track 表中的发行量（Circulation）低于平均发行量时，设置为 NULL。

```
UPDATE Track
SET Circulation＝NULL
WHERE Circulation<(SELECT AVG(Circulation）
FROM Track）
```