第 3 章　关系数据库标准语言 SQL

本章导读

本章主要讨论具有关系代数和关系演算二者优点的关系数据库标准语言 SQL。SQL 是国际化标准组织通过的关系数据库的标准语言，目前，几乎所有的关系数据库如 Oracle、SQL Server、My SQL、Access 等都支持标准语言 SQL。它是实现数据库操作的一个最常用的途径，即使是在应用程序中，对数据库的操作也是通过嵌入到语句中的 SQL 语句完成的。因此，学好 SQL 语言是学好该课程的前提，也是本书的重点。本章内容是基于对关系数据库操纵的基础上进行的，即对数据的定义、检索、更新和控制四个方面。通过本章的学习，读者要牢固掌握 SQL，达到举一反三的目的，同时通过实践，体会面向过程的语言和 SQL 的区别，体会关系数据库系统为数据库应用系统的开发提供良好环境,减轻用户负担,提高用户生产率的原因。并且在使用具体的 SQL 时，能有意识地与关系代数、关系演算等语言进行比较，了解他们各自的特点。

本章要点

- SQL 的特点及 SQL 语言的基本概念
- 数据定义：定义表、删除表、修改表，建立和删除索引
- 用 SQL 语句表达单表查询，包括选择表中的若干列和若干元组、分组及进行计算
- 用 SQL 语句表达连接查询，包括等值连接、自身连接、外连接、复合条件连接等
- 用 SQL 语句表达嵌套查询，包括带 IN 谓词、比较运算符和 EXISTS 谓词的子查询等
- 用 SQL 语句表达数据的更新，包括插入、删除和修改
- 用 SQL 语句定义视图、查询视图、更新视图和删除视图
- 用 SQL 语句实现数据控制

3.1　SQL 概述

SQL（Structured Query Language，结构化查询语言）是关系数据库的标准语言，它是介于关系代数和关系演算之间的一种语言。目前，几乎所有的关系数据库管理系统都支持 SQL，该语言是一种综合性的数据库语言，可以实现对数据的定义、检索、操纵和控制等功能。

目前，居于主流的关系数据库 Oracle、SQL Server、My SQL、Sysbase 以及便于移动的小型数据库 Access 等，虽然其使用存在差别，但是核心部分是相同的，即都采用标准数据库语言 SQL 作为它们的操纵语言。

3.1.1　SQL 的发展

1970 年，美国 IBM 研究中心的 E.F.Codd 提出了关系模型，并连续发表了多篇论文，人们对关系数据库的研究也日渐加深。1972 年 IBM 公司开始研制实验型关系数据库管理系统 SYSTEM R，并且为其配置了 SQUARE（Specifying Queries As Relational Expression）查询语言。1974 年，Boyce 和 Chamberlin 在 SQUARE 语言的基础上进行了改进，产生了

SEQUEL（Structured English Query Language）语言，后来 SEQUEL 简称为 SQL，即“结构式查询语言”。

SQL 是一个综合的、功能极强的语言，且具有使用方便灵活、语言简洁、易学的优点，所以很快被业界接受。1986 年 10 月，经美国国家标准局（American National Standard Insitute，ANSI）的数据库委员会 X3H2 批准，SQL 被作为关系数据库语言的美国标准，同年公布了标准 SQL 文本（简称 SQL-86）。1987 年 6 月，国际标准化组织（International Organization for Standardization，ISO）也通过了这一标准。随着数据库技术的不断发展，SQL 标准也被不断的丰富和发展。ANSI 在 1989 年 10 月又颁布了增强完整性特征的 SQL 89 标准。1992 年发布了 SQL（1992）标准（被称为 SQL 2），1999 年发布了 SQL（1999）（被称为 SQL 3）。

本章的论述主要遵循 SQL 2 标准，但有些标准在具体的系统中还没有实现，且由于各数据库厂商的 SQL 产品在支持标准 SQL 2 语法的同时，在功能上都作了相应的扩充，在实现上略有不同。因此，在使用具体的 DBMS 时，请参阅系统提供的参考手册。

3.1.2 SQL 的特点

虽然 SQL 是结构化查询语言的简称，但是 SQL 的功能远不止查询这么简单，而是集数据定义、数据查询、数据更新和数据控制功能于一体。其主要特点包括：

（1）综合统一。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。在非关系数据模型的数据库中，其数据语言分为数据定义语言、数据存储有关的描述语言和数据操纵语言，其中数据定义又按照模式和外模式进行了划分，这些语言的划分导致的结果就是：正在使用的数据库，如果要修改模式，就必须停止现有数据库的运行，转储数据，修改模式并编译后再重装数据库，为用户的使用带来很多不必要的麻烦。而 SQL 语句集数据定义语言、数据操纵语言、数据控制语言于一体，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。非关系数据模型的数据操纵语言是“面向过程”的，即是“过程化”的语言，用户不但要知道“做什么”，而且还应该知道“怎样做”，对于 SQL，用户只需要提出“做什么”，无需具体指明“怎么做”，例如，存取位置、存取路径选择、具体处理操作过程等均由系统自动完成。这种高度非过程化的特性大大减轻了用户的负担，使得用户更能集中精力考虑要“做什么”和所要得到的结果，并且存取路径对用户来说是透明的，有利于提高数据的独立性。

（3）面向集合的操作方式。在非关系数据模型中，采用的是面向记录的操作方式，即操作对象是一条记录。操作过程非常冗长复杂。而 SQL 语言采用的是面向集合的操作方式，且操作对象和操作结果都可以是元组的集合。

（4）统一的语法结构提供两种使用方式。SQL 可用于所有用户，通过自含式语言和嵌入式语言两种方式对数据库进行访问，前者是用户直接通过键盘键入 SQL 命令，后者是将 SQL 语句嵌入到高级语言（如 C、C++、VB、VC++、ASP.NET、Java 等）程序中。这两种方式使用的是统一的语法结构。

（5）语言简洁，易学易用。尽管 SQL 的功能很强，但语言十分简洁，SQL 完成核心功能只用了九个动词，且容易学习，易于使用：

1）数据定义：CREATE（创建），DROP（移除），ALTER（修改）。

2）数据查询：SELECT（查询）。

3）数据操纵：INSERT（插入），UPDATE（更新），DELETE（删除）。

4）数据控制：GRANT（授权），REVOKE（取消授权）。

3.1.3　SQL 体系结构

关系数据库的体系结构把关系数据库分为外模式、模式和内模式三级模式结构。SQL 语言也支持关系数据库三级模式结构，利用 SQL 语言可以实现对三级模式的定义、修改和数据的操纵功能，并在此基础上形成了 SQL 体系结构，只是术语与传统关系模型术语不同，其中外模式对应于视图和部分基本表，模式对应于基本表，内模式对应于存储文件，如图 3-1 所示。

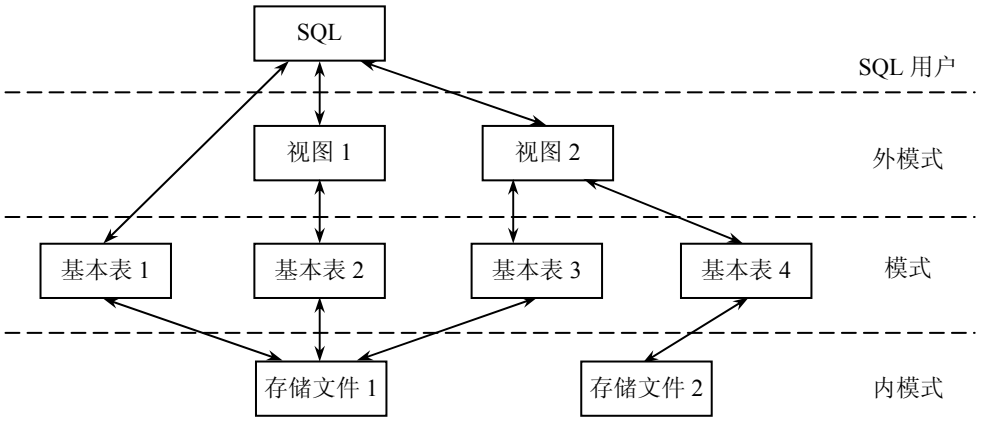

图 3-1　SQL 体系结构

在图 3-1 中对应的几个基本概念如下：

（1）存储文件是指数据表存储在硬盘上的结构，与外部存储器上的一个物理文件对应。存储文件的逻辑结构组成了关系数据库的内模式，存储文件的物理结构是任意的，对用户是透明的。

（2）基本表是本身独立存在的表，在 SQL 中一个关系就对应一个基本表，一个或多个基本表对应一个存储文件，一个表可以带若干索引，索引也存放在存储文件中。

（3）视图是从一个或者几个基本表导出的表，是一个虚表，它本身并不独立存储在数据库中，数据库中只存放对视图的定义而不存放该视图中的数据，需要读取视图中数据时，再从相应的表中读取。

（4）SQL 用户可以是应用程序，也可以是终端用户。用户可以用 SQL 语言对视图和基本表进行查询。在用户眼中，视图和基本表都是关系。

3.2　数据定义

SQL 语言的数据定义功能包括定义表、定义视图和定义索引，这里的定义实质上还包括对数据结构的定义、修改和删除。基本表和视图都是表，但基本表是实际存储在数据库中的表，视图是虚表，它是从基本表或其他视图中导出的表。

由于视图是基于基本表的虚表，索引又是依附于基本表的，大多数 RDBMS 支持的 SQL 语言，通常没有提供对视图和索引定义的修改操作，用户如果想修改这些对象的定义，只能先将其删除，然后再重建，但有些新的关系数据库管理系统软件提供了 SQL 语言修改视图的功能。

本节主要讲解对基本表和索引的定义，对于视图的定义将在 3.5 节中进行介绍。

3.2.1 创建、修改和删除基本表

1. 定义基本表

建立表的第一步就是定义基本表的结构。SQL 语言使用 CREATE TABLE 语句定义基本表，其基本格式如下：

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>]);

说明：

（1）< >中是 SQL 语句必须定义的部分，[]中是 SQL 语句可选择的部分。

（2）黑体（这里即为 CREATE TABLE）表示是 SQL 的关键字。

（3）<表名>是所要定义的基本表的名称，一个表可以由一个或若干个属性（列）组成。

（4）定义表的各个属性（列）时需要指明其数据类型及长度，表 3-1 列出了主要数据类型，需要说明的是，不同的 RDBMS 中支持的数据类型不完全相同。一个属性选用哪种数据类型要根据实际情况来决定，一般从取值范围和要做哪些运算两个方面来考虑，例如图书的库存量属性，可以选择数据类型 char(2)，但考虑到当被学生借出后需要将库存量进行减 1 的操作，因此要采用整数作为其库存量。

表 3-1 SQL2 提供的主要数据类型

类型	数据类型举例及缩写	说明
Character	CHAR(n) VARCHAR(n)	CHAR 数据类型可以存储字符集中的任意字符组合，具有固定的字符长度。VARCHAR(可变长度)数据类型允许字符长度变化，能够自动删除后继的空格
Numeric	INT SMALLINT NUMERICA(p [,d)] DECIMAL(p [,d)] FLOAT[(n)] REAL DOUBLE PRECISION	INT 和 SMALLINT 分别表示整型和短整型；NUMERICA(p [,d)]和 DECIMAL(p [,d)]表示定点数，由 p 位数字（不包括符号、小数点）组成，小数后面有 d 位数字；FLOAT[(n)]表示浮点数，精度至少为 n 位数字；REAL 和 DOUBLE PRECISION 分别表示取决于机器精度的浮点数和双精度浮点数
Boolean	BOOLEAN	BOOLEAN 数据类型存储 TRUE、FALSE 或 UNKNOWN
Temporal	DATE TIME TIMESTAMP INRERVAL	这些数据类型存储日期时间值。DATE 和 TIME 分别存储日期和时间。TIMESTAMP 类型存储着按机器当前运行时间计算出来的值。INRERVAL 指定一个时间间隔，它是一个相对值，用于增加或减少一个日期、时间或时间戳类型数据的绝对值
Bit string	BIT(n) BITVARYING(n)	这两种数据类型可以存储二进制和十六进制数据，BIT 数据类型长度固定，而 BITVARYING 数据类型具有可变长度
Binary	Binary large objects（BLOB）	BLOB 数据类型以十六进制格式存储二进制字符串的值

（5）完整性约束条件，分为列级的完整性约束和表级的完整性约束，如果完整性约束条

件涉及到该表的多个属性列，则必须定义在表级上，否则既可以定义在列级也可以定义在表级。在关系模型中，完整性约束包括实体完整性、参照完整性和用户定义完整性，这三种完整性约束条件都可以在表的定义中给出。其中，实体完整性定义表的主关键字（Primary Key），属于列级的完整性，可以定义在列级也可以定义在表级；参照完整性定义外关键字（Foreign Key），属于表级的完整性约束，只能定义在表级；用户定义完整性根据具体应用对关系模式提出要求：数据类型、数据格式、取值范围、空值约束等，一般定义在列级，当然也可以定义在表级。这些完整性约束条件跟表结构一起被存入系统的数据字典中，当用户操作表中数据时，由 DBMS 自动检查该操作是否违背了这些完整性约束条件。

在学生图书管理系统的 XSBOOK 中，有三个表分别为 STUDENT 表、BOOK 表以及 BORROW 表，下面分别通过几个例子加以说明。

【例 3-1】建立一个 STUDENT 表。

```
CREATE TABLE STUDENT
  ( 借书证号  char(8)  PRIMARY KEY,
    姓名      char(8)  UNIQUE,
    专业名    char(12), NOT NULL
    性别      char(2)  DEFAULT('女'),
    出生时间  datetime,
    借书数    int,
    照片      image );
```

该语句执行后，会在指定的 XSBOOK 数据库中建立 STUDENT 表，该表由 7 个属性列组成，借书证号是主关键字，姓名有列级的约束条件必须唯一，不能有重复值，专业名不能为空，在性别属性列，用户定义了其默认值为“女”。在本例中，均是列级的完整性约束条件，当然也可以放在表级的约束条件处。

【例 3-2】建立一个 BOOK 表。

```
CREATE TABLE BOOK
  (  ISBN    char(16)  PRIMARY KEY,
    书名     char(26) NOT NULL,
    作者     char(8) NOT NULL,
    出版社   char(20),
    价格     float,
    复本量   int DEFAULT(0),
    库存量   int );
```

该表同样也是只有列级的完整性约束条件。

【例 3-3】建立一个 BORROW 表。

```
CREATE TABLE BORROW
  ( 借书证号  char(8),
    ISBN     char(16),
    借书时间  datetime NOT NULL,
    应还时间  datetime NOT NULL,
    PRIMARY  KEY (借书证号,ISBN),
    FOREIGN  KEY 借书证号 REFERENCES STUDENT(借书证号),
    FOREIGN  KEY ISBN REFERENCES BOOK(ISBN));
```

该例中，借书时间和应还时间是列级的完整性约束条件，而对于主关键字的定义，因为涉及两个属性列，所以只能放在表级的完整性约束的位置。最下面的两句是对外键的定义，即 BORROW 表中“借书证号”和 ISBN 属性列分别参照了 STUDENT 表和 BOOK 表中的“借书

证号”和“ISBN 属性列”，当然本例中对于外键的定义只涉及到一个属性列，故也可以将 REFERENCES STUDENT（借书证号）和 REFERENCES BOOK（ISBN）分别置放在借书证号和 ISBN 属性列的后面，即：

```
CREATE TABLE BORROW
  ( 借书证号  char(8) REFERENCES STUDENT(借书证号),
    ISBN     char(16) REFERENCES BOOK(ISBN),
    借书时间  datetime NOT NULL,
    应还时间  datetime NOT NULL,
    PRIMARY  KEY (借书证号,ISBN));
```

通过以上三个例子，可以发现：

（1）主关键字的定义。对主关键字的定义可以有两种方法，分别为：

- 在列出关系模式的属性时，在属性及其类型后加上保留字 PRIMARY KEY，表示该属性为主码属性，此时是作为列级的完整性约束。
- 在列出关系模式的所有属性后，再附加一个声明：PRIMARY KEY (<属性 1>[,<属性 2>,…])，此时是作为表级的完整性约束。

一个关系可能有多个候选关键字，但在定义基本表时只能定义一个主关键字。一个关系的主关键字由一个或几个属性构成，当主关键字由一个属性列组成时，可以采用上述的两种方法定义，如果关键字由多个属性构成时必须采用第二种方法。

（2）外部关键字的定义。外部关键字即外码的定义是建立参照完整性的约束，它是关系模式的另一种重要约束。其格式为：REFERENCES <表名><属性>，对于外码的定义同样有两种方式，同定义主关键字一样，对外码的定义同样也是有两种方式，且同样是只涉及一个属性时两种方法均可用，而如果涉及多个属性则只能将其定义为表级的完整性约束。

（3）关于缺省值。可以在定义属性时增加保留字 DEFAULT 和一个合适的值，例如：

```
年龄 SMALLINT DEFAULT 18
```

2. 修改基本表

随着应用环境和应用需求的变化，有时需要修改已经建立好的基本表，如增加列、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，其语法格式如下：

```
ALTER TABLE <表名>
  [ ADD      [COLUMN] <新列名><数据类型> [完整性约束]]
  [ DROP     [COLUMN] <列名>[RESTRICT | CASCADE]]
  [ MODIFY   [COLUMN] <列名><新数据类型>];
```

ALTER TABLE 允许对表进行很多有用的修改操作。这一通用命令允许用户添加、删除列（ADD、DROP COLUMN）或约束条件。此处，CASCADE 方式表示，在基本表中删除该列时，所有引用到此列的视图和约束也要一起被自动删除，而 RESTRICT 方式表示，在没有视图或约束引用到该属性列时，才能在基本表中删除该列，否则拒绝删除操作。

需要说明的是，这是 SQL99 的语法格式，而在实际的 RDBMS 中会有所改变，请读者在具体使用时参照系统的帮助文档。

【例 3-4】在 STUDENT 表中增加“年龄”属性列，类型为 SMALLINT 型。

```
ALTER TABLE STUDENT ADD 年龄 SMALLINT;
```

需要说明的是，新增加的列不能定义为 NOT NULL。基本表在增加一列后，原有元组在新增加的列上的值都被定义为空值（NULL）。

【例 3-5】在 STUDENT 表中删除“出生时间”属性。

```
ALTER TABLE STUDENT DROP 出生时间;
```

【例 3-6】在 BOOK 表中将“书名”属性列的长度修改为 20 位。

```
ALTER TABLE BOOK MODIFY 书名 char(20);
```

3. 删除基本表

当某个基本表不在需要时，需要将其删除，以释放其所占的资源，删除基本表可以使用 DROP TABLE 语句实现，其格式如下：

DROP TABLE 表名 [RESTRICT | CASCADE]

此处 RESTRICT 和 CASCADE 选项的使用与前面句法中的语义相同。需要注意的是，一旦对一个基本表执行了此删除操作后，该表中所有的数据也就丢失了，所以对于删除表的操作，用户一定要慎用。

【例 3-7】假设已经存在一个表，表名为“临时表”，现将其删除，并将与该表有关的其他数据库对象一起删除。

```
DROP TABLE 临时表 CASCADE;
```

前面曾经提到过，不同的数据库产品对于 SQL 语言的支持会有所不同，对于 RESTRICT 和 CASCADE 选项，目前居于主流的 Oacle 9i 数据库只有 CASCADE 选项，而 SQL Server 数据库这两个选项都没有。

3.2.2　创建和删除索引

建立索引是加快查询速度的有效手段。索引实际上是根据关系（表）中某些字段的值建立一棵树型结构的文件。索引文件中存储的是按照某些字段的值排列的一组记录号，每个记录号指向一个待处理的记录，所以，索引实际上可以理解为根据某些字段的值进行逻辑排序的一组指针。在日常生活中，经常会遇到索引，如图书目录、词典索引等，通过索引可以大大提高查询的速度，但索引的功能仅限于查询时起作用。

目前，很多 DBMS 直接使用主键的概念建立主索引，方法是建立基本表时直接定义主键，即建立了主索引，一个表只能有一个主索引，同时用户还可以建立其他索引，不同的 DBMS 略有区别，如 VFP 有主索引、侯选索引、普通索引和唯一索引四种类型的索引；Access 中有重复索引和非重复索引；SQL Server 中则是聚簇索引、非聚簇索引和唯一索引。

SQL 语言支持用户根据应用环境的需要，在基本表上建立一个或多个索引，以提供多种存取路径，加快查找速度。一般来说，只有数据库管理员 DBA 和表的属主，即建立表的人员才能进行索引的创建和删除。

1. 建立索引

创建索引可以用 CREATE INDEX 语句实现，其格式如下：

CREATE [UNIQUE][CLUSTER] INDEX <索引名> ON <表名>(<列名>[<次序>][,<列名>[<次序>]]…);

其中，<表名>是指要建立索引的基本表的名字，<索引名>是用户自己为建立的索引起的名字。索引可以建立在该表的一列或多列上，各列名之间用逗号分隔，这种由两列或多列属性组成的索引称为复合索引（Composite Index）。每个列名后面还可以指定<次序>，即索引值的排列次序，可选 ASC（升序）或 DESC（降序），缺省值为 ASC。

【例 3-8】为 STUDENT、BOOK 和 BORROW 表建立索引。

```
CREATE  INDEX  STU_IDX_LNO  ON   STUDENT (借书证号);
```

```
CREATE INDEX COU_IDX_BNO ON BOOK (ISBN);
CREATE INDEX SC_IDX_LNO_BNO ON BORROW (借书证号,ISBN);
```

2. 删除索引

删除索引使用的语句是 DROP INDEX，其格式如下：

DROP INDEX <索引名>;

DROP INDEX 命令可以删除当前数据库内的一个或几个索引。当一个索引被删除后，该索引先前占有的存储空间就会被收回。但是，DROP INDEX 不会影响 PRIMARY KEY 和 UNIQUE 约束条件，这些约束条件必须用 ALTER TABLE…DROP 命令来完成。

【例 3-9】删除 STUDENT 表中在借书证号上建立的索引。

```
DROP INDEX STU_IDX_LNO;
```

需要说明的是，DBMS 一般会自动建立 PRIMARY KEY 和 UNIQUE 列上的索引。此外，系统在存取数据时会自动选择合适的索引作为存取路径，用户不必也不能显式地选择索引。

虽然采用了索引技术可以提高数据查询的速度，但另一方面，增加了数据插入、删除和修改的复杂性以及维护索引的时间开销。过多的索引文件还会占用一定的文件目录和存储空间，从而使系统负担加重。根据实际需要，有时可以删除一些不必要的索引。因此，是否使用索引，对哪些属性建立索引，数据库设计人员必须全面考虑，权衡折中。下面给出几点使用索引的技巧：

（1）对于记录少的表使用索引，其性能不会有任何提高。

（2）索引列中有较多的不同的数据和空值时，会大大提高索引的性能。

（3）当查询要返回的数据很少时，索引可以优化你的查询（比较好的情况是少于全部数据的 25%），相反，如果要返回的数据很多，索引会加大系统开销。

（4）索引可以提高数据的返回速度，但是它使得数据的更新操作变慢，因此不要对经常需要更新或修改的字段创建索引，更新索引的开销会降低用户所期望获得的性能。

（5）索引会占用数据库的空间，因此在设计数据库的可用空间时要考虑索引所占用的空间。

（6）不要将索引与表存储在同一个驱动器上，分开存储会去掉访问的冲突从而使结果返回得更快。

3.3 数据查询

建立数据库的目的就是为了对数据库进行操作，以便能够从中提取有用的信息。在 3.2 节中已经对数据库及其表结构进行了定义，从本节开始将开始介绍对数据库的操作，其中数据库查询是数据操作中的核心操作，SQL 提供了 SELECT 语句对数据库进行查询操作。其标准语法是：

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 1> [HAVING <条件表达式>]]

[ORDER BY <列名 2> [ASC|DESC]];

说明：

该语句的基本语义为，根据 WHERE 子句中的条件表达式，从 FROM 子句指定的基本表

或视图中找出满足条件的元组，并按 SELECT 子句中指出的目标属性列，选出元组中的分量形成结果表。

实际上，语句中的 SELECT 子句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。进行数据库查询时，并非上述语句中的每个子句都会用到，最简单的情况下，查询只需要一个 SELECT 和一个 FROM 子句。如果有 GROUP BY 子句选项，则将结果按<列名 1>的值进行分组，该属性列值相等的元组为一个组，通常会在每组中使用聚集函数。如果 GROUP BY 子句带有 HAVING 短语，则结果只有满足指定条件的组。ORDER BY 子句是将查询的结果进行排序显示，ASC 表示升序，DESC 表示降序，默认为升序排列。可选项[ALL | DISTINCT]的含义是，如果没有指定 DISTINCT 短语，则缺省为 ALL，即保留结果中取值重复的行，相反，如果指定了 DISTINCT 短语，则可消除重复的行。

在 3.2 节中，例 3-1、例 3-2 和例 3-3 所建立的图书管理系统中的 3 个表分别为：

（1）STUDENT（借书证号，姓名，专业，性别，出生时间，借书数，照片，办证日期）。

（2）BOOK（ISBN，书名，作者，出版社，价格，复本书，库存量）。

（3）BORROW（借书证号，ISBN，借书时间，应还时间）。

各表中的数据分别如表 3-2、表 3-3 和表 3-4 所示。

表 3-2　STUDENT 表

借书证号	姓名	专业名	性别	借书数	出生年份	办证日期
080101	吕亭亭	计算机	女	3	1988-01	2008-06
080102	张玉玲	计算机	女	1	1989-05	2008-07
080105	汪东升	网络工程	男	2	1988-06	2008-06
080208	陈艺	电子	女	2	1987-10	2008-09
080210	张彦	电子	男	0	1988-09	2008-07
080511	孙森茂	外语	男	1	1986-05	2008-10
⋮	⋮	⋮	⋮	⋮	⋮	⋮

表 3-3　BOOK 表

ISBN	书名	作者	出版社	价格	复本数	库存量
730200899X	版主答疑 -Delphi 高级编程技巧	岳庆生	机械工业出版社	49.0	7	5
781067224X	大学英语词汇记忆点津与考点要览	马德高	牛津大学出版社（港）	16.0	20	15
7871112133	数据库系统导论	C.J.Date	机械工业出版社	75.0	10	2
7800048381	英语网上文摘	董素华	科学出版社	5.0	20	3
7040100959	C++程序设计语言（特别版）	Special Stroustrup	人民交通出版社	55.67	8	1
7115101620	计算机网络	谢希仁	电子工业出版社	39.0	4	0
7040195835	数据库系统概论	王珊	高等教育出版社	33.8	8	2
7302050031	亲密接触 ASP.NET	杜亮	清华大学出版社	39	20	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮

表 3-4 BORROW 表

借书证号	ISBN	借书时间	应还时间
080101	7040100959	2008-09-01	2008-12-01
080101	7040195835	2008-09-01	2008-12-01
080101	7115101620	2008-09-01	2008-12-01
080102	730200899X	2008-08-25	2008-11-25
080105	781067224X	2008-09-10	2008-12-10
080208	781067224X	2008-10-09	2009-01-09
080208	7115101620	2008-09-28	2008-11-28
080515	7800048381	2008-10-23	2009-01-23
⋮	⋮	⋮	⋮

下面将查询分为简单查询、复杂查询和嵌套查询几类对该图书管理系统进行举例说明，通过这些例子可以看出查询语句的丰富功能和灵活的使用方式。

3.3.1 单表查询

单表查询是最简单的 SQL 查询，指只涉及一个表的查询，可分为以下几种操作：

（1）选择表中的若干列（关系代数中的投影运算）。

（2）选择表中的若干元组（关系代数中的选择运算）。

（3）使用聚集函数。

（4）GROUP BY 子句。

（5）ORDER BY 子句。

1. 选择表中的若干列

选择表中的若干列对应于关系代数中的投影运算，在 SQL 中利用 SELECT 子句来指定要投影的属性列。SELECT 子句既可以指定表中所有的属性列，也可以指定个别的读者感兴趣的属性列，还可以通过对列值进行算术运算得到表中不存在的信息。

（1）选择一个表中的指定列。使用 SELECT 子句选择一个表中的某些列，各列名之间用逗号分隔。

【例 3-10】在 STUDENT 表中查询出所有读者的姓名、专业名和借书数。

```
SELECT  姓名,专业名,借书数
FROM   STUDENT;
```

（2）查询全部列。使用 SELECT 子句查询一个表中的所有属性列且与表结构中的顺序相同时，可以使用通配符“*”代替所有的列。

【例 3-11】在 BORROW 表中找出所有的借阅信息。

```
SELECT  借书证号,ISBN,借书时间,应还时间
FROM   BORROW;
```

在该查询中，因为查询的是所有的属性列，且其顺序与表结构中的顺序相同，故也可以表示为

```
SELECT  *
FROM   BORROW;
```

例 3-11 中的两个查询语句是等价的，查询结果相同。因此，如果要查询某个表的所有属

性列，其查询语句有两种写法，一种是在 SELECT 子句中列出所有的属性列，一种是在 SELECT 子句中直接使用“*”代替所有的属性列。但是使用第二种用法前提是，用户所需要属性列的顺序与数据库中的存储顺序相同。

（3）查询经过计算的值。使用 SELECT 进行查询时，不仅可以直接以列的原始值作为结果，而且还可以将对列值进行计算后所得的值作为查询结果，即在 SELECT 子句中可以使用表达式作为属性列。

【例 3-12】查询学生的姓名和年龄（目前的年份为 2009 年）。

由于表 3-3 中没有年龄属性，所以不能直接列出年龄，但是 SELECT 子句中可以出现算术表达式，从而可以查询经过计算的值。

```
SELECT  姓名,2009-Year(出生时间)
FROM   STUDENT;
```

查询结果为：

姓名	Expr
吕亭亭	21
张玉玲	20
汪东升	21
陈　艺	22
张　彦	21
孙森茂	23
⋮	⋮

该例中，通过运算得到的属性列系统都会自动地给它赋一个列名（Expr），用户看了之后不易理解，可以添加列的别名，以替换在结果中列出的默认列标题。则上述查询可以表示为：

```
SELECT  姓名,2009-Year(出生时间)  AS  '年龄'
FROM    STUDENT;
```

查询结果为：

姓名	年龄
吕亭亭	21
张玉玲	20
汪东升	21
陈　艺	22
张　彦	21
孙森茂	23
⋮	⋮

SELECT 子句中除了使用算术表达式外，还可以使用字符串常量、函数以及列别名，从而大大加强 SQL 查询的功能。

【例 3-13】查询所有读者的姓名、还可借书籍的数量，要求两个字段之间用字符串“还可借书数：”进行连接，并为所计算的列指定列名“可借书数”（设每人最多可借书 6 本）。

```
SELECT 姓名, '还可借书数：', 6-借书数 as '可借书数'
FROM   STUDENT;
```

查询结果为：

姓名	'还可借书数：'	借书数
吕亭亭	还可借书数：	3
张玉玲	还可借书数：	5
汪东升	还可借书数：	4
陈　艺	还可借书数：	4
张　彦	还可借书数：	6
孙森茂	还可借书数：	5
⋮	⋮	⋮

2. 选择表中的若干元组

选择表中的若干元组对应于关系代数中的选择运算。

（1）消除结果集中的重复行。在关系数据库中，不允许出现完全相同的两个元组，但是当我们只选择表中的某些列时，就可能会出现重复的行。

【例 3-14】从 BORROW 表中找出所有借了书的读者的借书证号。

```
SELECT  借书证号
FROM   BORROW;
```

本例中，可能某个读者借了 6 本书，则上述查询中就出现了“重号”，即一个借书证号出现了 6 次，在 SELECT 子句中用 DISTINCT 关键字可以消除结果集中的重复行，下述语句就去掉了查询结果中重复的元组。

```
SELECT  DISTINCT  借书证号
FROM   BORROW;
```

（2）查询满足条件的元组。在 SQL 中，查询满足条件的元组，利用 WHERE 子句实现。WHERE 子句常用的查询条件如表 3-5 所示。

表 3-5　常用的查询条件

查询条件	谓词
比较	=，<>，>，<，>=，<=，!=，!>，!<；NOT+上述比较运算符
算术运算	+，-，*，/
确定范围	BETWEEN AND，NOT BETWEEN AND
确定集合	IN，NOT IN
字符匹配	LIKE，NOT LIKE
空值	IS NULL，IS NOT NULL
多重条件	AND，OR，NOT

下面分别针对以上列出的查询条件，给出查询的实例：

（1）比较运算。在表 3-5 中列出了一般的比较运算符，那么比较常用的运算符有：=（等于）、<>或!=（不等于）、>（大于）、<（小于）、>=（大于等于）、<=（小于等于）。

【例 3-15】查询 STUDENT 表中借书数在 3 本以上的学生情况。

```
SELECT *
FROM STUDENT
WHERE 借书数 >= 3;
```

（2）指定范围。用于确定范围的关键字有 BETWEEN…AND…和 NOT BETWEEN…AND…，用来查找属性值在（或不在）指定范围内的元组，其中 BETWEEN 后是范围的下限（即低值），AND 后是范围的上限（即高值）。

【例 3-16】查询 BOOK 表中价格介于 30 元到 50 元之间（包括 30 和 50）的书籍的 ISBN、书名、作者和出版社信息。

```
SELECT  ISBN,书名,作者,出版社
FROM    BOOK
WHERE   价格 BETWEEN 30 AND 50;
```

相反的，如果要查询价格不在 30 元到 50 元之间的书籍信息，则可用 NOT BETWEEN…AND…来表达：

```
SELECT  ISBN,书名,作者,出版社
FROM    BOOK
WHERE   价格 NOT BETWEEN 30 AND 50;
```

（3）确定集合。谓词 IN 可以用来查找属性值属于指定值表集合的元组，值表中列出所有可能的值，当 IN 前面的表达式与值表中的任何一个值匹配时，则返回 True，否则返回 False。

【例 3-17】查询 STUDENT 表中专业名为“计算机”、“网络工程”、“软件工程”的学生信息。

```
SELECT  *
FROM   STUDENT
WHERE 专业名 IN ('计算机', '网络工程', '软件工程');
```

相反的，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。如查询除“计算机”、“网络工程”、“软件工程”之外的所有专业的学生信息：

```
SELECT  *
FROM   STUDENT
WHERE 专业名 NOT IN ('计算机', '网络工程', '软件工程');
```

（4）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：

[NOT]　LIKE '<匹配串>'　[ESCAPE '<换码字符>']

其含义是查找指定的属性列值与<匹配串>相匹配的行。<匹配串>可以是一个完整的字符串，也可以含有通配符“%”和“_”。%（百分号）代表任意长度（包括 0）的字符串，_（下横线）代表任意单个字符，ESCAPE 表示转义符。

LIKE 匹配中使用通配符的查询又称模糊查询。

【例 3-18】查询 STUDENT 表中电子专业的学生的借书证号、姓名和借书数。

```
SELECT 借书证号, 姓名, 借书数
FROM  STUDENT
WHERE 专业名 LIKE  '电子';
```

如果 LIKE 后面的匹配串中不含有通配符，则可以用等号（=）代替 LIKE 谓词，可以用不等于（<>或!=）运算符代替 NOT LIKE 谓词。

因此例 3-18 中的查询语句还可以表示为：

```
SELECT 借书证号, 姓名, 借书数
FROM  STUDENT
WHERE 专业名= '电子'
```

【例 3-19】查询 STUDENT 表中所有姓王的学生的借书证号、姓名、专业名和性别。

```
SELECT 借书证号, 姓名, 专业名, 性别
```

```
FROM  STUDENT
WHERE 姓名 LIKE  '王%';
```

【例 3-20】查询 STUDENT 表中名字中第二字为“晓”字的学生的详细情况。

```
SELECT  *
FROM  STUDENT
WHERE 姓名 LIKE  '_晓%';
```

如果用户要查询的匹配字符串本身就含有“%”或“_”，则要使用转义符（ESCAPE）'\'，对通配符进行转义。转义符 ESCAPE 指出其后的每个字符均作为实际的字符对待，而不再作为通配符。

【例 3-21】查询 BOOK 表中书名以 100%结束的书籍的 ISBN 和出版社。

```
SELECT  ISBN, 出版社
FROM   BOOK
WHERE 书名 LIKE '%100'  ESCAPE '% ';
```

（5）空值。当需要判定一个表达式的值是否为空值时，使用 IS NULL 关键字。

【例 3-22】查询 STUDENT 表中专业名尚不确定的学生的详细信息。

```
SELECT  *
FROM  STUDENT
WHERE 专业名 IS  NULL;
```

注意：此处 IS 不能用等号（“=”）代替。

（6）多重条件查询。逻辑运算符 AND 与 OR 可以用来连接多个查询条件，且前者的优先级高于后者，必要时可以通过括号改变优先级。

【例 3-23】查询 BOOK 表中价格在 40 元以上电子工业出版社的图书和所有清华大学出版社的图书的 ISBN、书名、出版社和价格信息。

```
SELECT  ISBN, 书名, 出版社, 价格
FROM   BOOK
WHERE   出版社='电子工业出版社'  AND 价格>40 OR 出版社='清华大学出版社';
```

3. 使用聚集函数

在检索数据时，经常需要对结果进行计算和统计。为了进一步方便用户，增强检索功能，SQL 提供了许多聚集函数，经常使用的主要包括以下三类：

（1）COUNT。COUNT 用于统计组中满足条件的值或元组的个数：

COUNT([DISTINCT|ALL] *) 统计元组个数

COUNT([DISTINCT|ALL] <列名>) 统计一列中值的个数

（2）SUM 和 AVG。SUM 和 AVG 分别用于求表达式中所有值项的总和与表达式：

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）

（3）MAX 和 MIN。MAX 和 MIN 分别用于求表达式中所有值项的最大值和最小值：

MAX([DISTINCT|ALL] <列名>) 求一列值中的最大值

MIN([DISTINCT|ALL] <列名>) 求一列值中的最小值

在这些聚集函数中，如果指定 DISTINCT 短语，则表示在计算时要取消指定列中的重复值；反之，如果不指定 DISTINCT 短语或指定 ALL 短语（默认为 ALL），则表示不取消重复值。

【例 3-24】查询读者总人数。

```
SELECT COUNT(*)  AS  '总人数'
```

```
FROM STUDENT;
```

在该查询中，除了计算出 STUDENT 表中的总人数，还使用 AS 关键字为查询结果指定了新列名“总人数”。

【例 3-25】查询借阅了图书的学生数。

```
SELECT COUNT(DISTINCT 借书证号)  AS '借阅了图书的学生数'
FROM  BORROW;
```

【例 3-26】查询图书总册数和库存图书总册数。

```
SELECT  SUM(复本量) AS '图书总册数', SUM(库存量) AS '库存图书总册数'
FROM    BOOK;
```

【例 3-27】查询计算机或网络工程专业借书最多和最少的册数。

```
SELECT  MAX(借书数) AS '借书最多册数', MIN(借书数) AS '借书最少册数'
FROM    STUDENT
WHERE   专业名='计算机' OR 专业名='网络工程';
```

4. GROUP BY 子句

GROUP BY 子句可用于将查询结果的各行按某一列或多列值进行分组，值相等的为一组。对查询结果分组的主要目的是为了细化聚集函数的作用对象。如果未对查询结果分组，聚集函数将作用于整个查询结果，即整个查询结果只有一个函数值，而使用 GROUP BY 子句进行分组后，聚集函数将作用于每一个组，即每一组都有一个函数值。

需要说明的是，使用 GROUP BY 子句后，SELECT 子句中的列表只能是 GROUP BY 子句中指定的列或在聚集函数中指定的列，否则系统会报错处理。

【例 3-28】查询 STUDENT 表中各个专业的学生数。

```
SELECT  专业名, COUNT(借书证号) AS '学生数'
FROM    STUDENT
GROUP BY 专业名;
```

该语句对 STUDENT 表按专业名的取值进行分组，所有具有相同专业名的元组为一组，然后对每一组用聚集函数 COUNT 求得该组的学生人数，本查询中 COUNT(借书证号)也可以换成 COUNT(*)。

【例 3-29】查询 BOOK 表中各个出版社各有多少本图书。

```
SELECT  出版社, COUNT(*) AS '图书数'
FROM    BOOK
GROUP BY 出版社;
```

如果分组后还需要按一定的条件对这些组进行筛选，最终只输出满足指定条件的组，则可以使用 HAVING 短语来指定筛选条件。

【例 3-30】查询 STUDENT 表中，女生人数不超过 5 人的专业名。

```
SELECT 专业名,count(*)AS '女生人数'
FROM STUDENT
WHERE 性别 = '女'
GROUP BY 专业名 HAVING COUNT(*)<=5;
```

该例中的查询首先从 STUDENT 表中查出所有性别为“女”的元组，然后再按照专业名进行分组，用聚集函数 COUNT（在本例中，COUNT(*)也可以表述为 COUNT(借书证号)）对每一组计数并使用 HAVING 短语对结果进行限制，HAVING 短语指定选择组的条件，只有满足条件（即元组个数<=5）的组才会被选出来。

在 SELECT 查询语句中，WHERE 子句与 HAVING 短语的根本区别在于其作用对象不同。

如果 WHERE 子句与 GROUP BY…HAVING 子句同时出现，则其作用及执行顺序为：WHERE 子句作用于基本表或视图，用于筛选 FROM 指定的数据对象；GROUP BY 用于对 WHERE 限制的结果进行分组；HAVING 短语则是对 GROUP BY 以后的分组数据进行过滤，从中选择满足条件的组。

5. ORDER BY 子句

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。但在实际应用中，用户经常要对查询的结果排序输出，ORDER BY 子句可用于对查询结果按照一个或多个属性列进行升序（ASC）或降序（DESC）排列，默认值为升序排列。

【例 3-31】查询 BOOK 表中水利水电出版社的图书，并按照价格进行由高向低排列。

```
SELECT  *
FROM   BOOK
WHERE  出版社='水利水电出版社'
ORDER BY 价格 DESC;
```

【例 3-32】查询 STUDENT 表中所有学生的信息，查询结果按专业名升序排列，同一个专业的学生按照年龄升序排列（即按照出生年份降序排列）。

```
SELECT  *
FROM  STUDENT
ORDER BY 专业名, 出生年份 DESC;
```

说明：使用 ORDER BY 子句对查询结果进行排序，当排序列含空值时，如果按升序排列，排序列为空值的元组最后显示；如果按照降序排列，则排序列为空值的元组最先显示。

3.3.2 连接查询

3.3.1 节中都是针对单个表的查询，若一个查询同时涉及两个以上的表，则称为连接查询。连接运算可能发生在两个表之间，也可能发生在多个表之间。连接查询根据连接的对象和方法不同，可以包含以下几个方面的内容：

（1）等值连接和非等值连接查询。

（2）自身连接查询。

（3）外连接查询。

（4）复合条件连接查询。

（5）集合运算查询。

1. 等值与非等值连接查询

WHERE 子句中用来连接两个表的条件称为连接条件或连接谓词，其一般格式如下：

[<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!=）等，当比较运算符为=时，称为等值连接，若在 SELECT 子句的目标列中去除相同的字段名，则为自然连接。使用其他的比较运算符称为非等值连接。

此外，连接谓词还可以使用下面形式：

[<表名 1>.]<列名 1> BETWEEN [<表名 2>.]<列名 2> AND [<表名 2>.]<列名 3>

说明：连接谓词中的列名称为连接字段。连接条件中的各连接字段类型必须是可比的，但名字不必相同。例如，可以都是字符型或都是日期型；也可以一个是整型，另一个是实型，整型和实型都是数值型，因此是可比的。但若一个是字符型，另一个是整数型就不允许了，因为

它们是不可比的类型。

【例 3-33】查询每个读者的基本信息及借书情况。

在图书借阅管理系统中，读者的基本信息存放在 STUDENT 表中，读者的借书情况存放在 BORROW 表中，因此该查询实际上同时涉及到了 STUDENT 和 BORROW 两个表中的数据。而这两个表之间的联系是通过两个表的共有属性“借书证号”实现的。完成本查询的 SQL 语句为：

```
SELECT STUDENT.*, BORROW.*
FROM  STUDENT, BORROW
WHERE STUDENT.借书证号 = BORROW.借书证号;
```

查询结果为：

STUDENT. 借书证号	姓名	…	办证日期	BORROW. 借书证号	ISBN	借书时间	应还时间
080101	吕亭亭	…	2008-06	080101	7040100959	2008-09-01	2008-12-01
080101	吕亭亭	…	2008-06	080101	7040195835	2008-09-01	2008-12-01
080101	吕亭亭	…	2008-06	080101	7115101620	2008-09-01	2008-12-01
080102	张玉玲	…	2008-07	080102	730200899X	2008-08-25	2008-11-25
080105	汪东升	…	2008-06	080105	781067224X	2008-09-10	2008-12-10
080208	陈　艺	…	2008-09	080208	781067224X	2008-10-09	2009-01-09
080208	陈　艺	…	2008-09	080208	7115101620	2008-09-28	2008-11-28
⋮	⋮	⋮	⋮				

该查询是等值连接查询，相同的属性列出现了两次。其执行过程是：首先在 STUDENT 表中找到第一个元组，然后从头开始顺序扫描或按索引扫描表 BORROW，查找 BORROW 表中借书证号与 STUDENT 表中第一个元组的借书证号相等的元组，每找到一个元组，就将 STUDENT 表中的第一个元组与该元组拼接起来，形成结果表中一个元组。BORROW 表全部扫描完毕后，再到表 STUDENT 中找第二个元组，然后再从头开始顺序扫描或按索引扫描 BORROW 表，查找满足连接条件的元组，每找到一个元组，就将 STUDENT 表中的第二个元组与该元组拼接起来，形成结果表中一个元组。重复上述操作，直到 STUDENT 表全部元组都处理完毕为止。

需要说明的是，如果一个属性列属于两个或多个表，那么在使用时一定要在列名前加前缀“表名”，否则系统无法判断。比如在例 3-33 中，两个表中都有属性列“借书证号”，那么当查询语句中出现“借书证号”字段时，一定要注明“表名”前缀。

【例 3-34】对例 3-33 用自然连接来完成。

```
SELECT STUDENT.借书证号,姓名,专业名,性别,借书数,出生年份,办证日期,ISBN,借书时间,应还
时间
FROM  STUDENT, BORROW
WHERE STUDENT.借书证号 = BORROW.借书证号;
```

查询结果如下：

STUDENT. 借书证号	姓名	…	办证日期	ISBN	借书时间	应还时间
080101	吕亭亭	…	2008-06	7040100959	2008-09-01	2008-12-01
080101	吕亭亭	…	2008-06	7040195835	2008-09-01	2008-12-01
080101	吕亭亭	…	2008-06	7115101620	2008-09-01	2008-12-01
080102	张玉玲	…	2008-07	730200899X	2008-08-25	2008-11-25
080105	汪东升	…	2008-06	781067224X	2008-09-10	2008-12-10
080208	陈　艺	…	2008-09	781067224X	2008-10-09	2009-01-09
080208	陈　艺	…	2008-09	7115101620	2008-09-28	2008-11-28
⋮	⋮	⋮	⋮	⋮	⋮	⋮

在本查询中，由于姓名、专业名、性别、ISBN 等属性列在两表中是唯一的，因此引用时可以去掉表名前缀，而借书证号在两个表都出现了，因此引用时必须加上表名前缀。该查询的执行结果不再出现借阅表的借书证号属性列 BORROW.借书证号。

2. 自身连接

自身连接是连接查询的一个特例，也就是将一个表与它自身进行连接，又称为自连接。一般来说，如果要在一个表中查找具有相同列值的行可使用自身连接，使用自身连接时，需要为表指定两个别名，且对所有的引用均要用到别名限定。

【例 3-35】查找在同一天借阅了不同图书的学生的借书证号、ISBN 和借书时间。

```
SELECT  B1.借书证号, B1.ISBN, B1.ISBN, B1.借书时间
FROM    BORROW B1, BORROW B2
WHERE   B1.借书证号= B2.借书证号 AND B1.借书时间= B2.借书时间 AND
        B1.ISBN!= B2.ISBN
```

3. 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，如在例 3-33 和例 3-34 的结果表中没有关于 080210 和 080511 两位读者的信息，原因在于他们没有借阅图书，在借阅表 BORROW 中没有相应的元组，从而造成这两位读者的信息被舍弃了。但是，有时可能需要以学生表 STUDENT 为主体列出每个学生的基本情况及其借阅图书情况，若某个学生没有借阅图书，则只输出其基本信息，其借阅信息为空值即可，这时就需要使用外连接（Outer Join），外连接分为左外连接和右外连接。左外连接列出连接语句左边关系中所有的元组，如果连接语句右边关系中没有与之相匹配的元组，则在相应的属性上填空值（NULL），而右外连接是列出右边关系中所有的元组，连接语句左边关系中没有与之相匹配的元组，则在相应的属性上填空值（NULL）。

【例 3-36】查询所有学生的信息及其借阅图书的 ISBN 和应还时间，如果没有借阅图书则直接列出其基本信息。

```
SELECT  STUDENT.借书证号,姓名,专业名,性别,借书数,出生年份,办证日期,ISBN,应还时间
FROM   STUDENT LEFT OUT JOIN BORROW ON (STUDENT.借书证号=BORROW.借书证号);
```

查询结果为：

STUDENT.借书证号	姓名	专业名	性别	借书数	出生年份	办证日期	ISBN	应还时间
080101	吕亭亭	计算机	女	3	1988-01	2008-06	7040100959	2008-12-01
080101	吕亭亭	计算机	女	3	1988-01	2008-06	7040195835	2008-12-01
080101	吕亭亭	计算机	女	3	1988-01	2008-06	7115101620	2008-12-01
080102	张玉玲	计算机	女	1	1989-05	2008-07	730200899X	2008-11-25
080105	汪东升	网络工程	男	2	1988-06	2008-06	781067224X	2008-12-10
080208	陈艺	电子	女	2	1987-10	2008-09	781067224X	2009-01-09
080208	陈艺	电子	女	2	1987-10	2008-09	7115101620	2008-11-28
080210	张彦	电子	男	0	1988-09	2008-07	NULL	NULL
080511	孙森茂	外语	男	1	1986-05	2008-10	NULL	NULL
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

在该例中，属于左外连接，列出了连接语句左边关系 STUDENT 中的所有元组，若该学生没有借阅图书，则 BORROW 表中相应字段赋值为 NULL；同样可以用 RIGHT OUT JOIN 表示右外连接。

【例 3-37】查询被借阅了的图书的借阅情况和图书的书名。

```
SELECT 借书证号, BORROW.ISBN,借书时间,应还时间,书名
FROM  BORROW RIGHT JOIN BOOK ON (BORROW.ISBN= BOOK.ISBN);
```

查询结果为：

借书证号	ISBN	借书时间	应还时间	书名
080101	7040100959	2008-09-01	2008-12-01	C++程序设计语言（特别版）
080101	7040195835	2008-09-01	2008-12-01	数据库系统概论
080101	7115101620	2008-09-01	2008-12-01	计算机网络
080102	730200899X	2008-08-25	2008-11-25	Delphi 高级编程技巧
080105	781067224X	2008-09-10	2008-12-10	大学英语词汇记忆点津与考点要览
080208	781067224X	2008-10-09	2009-01-09	大学英语词汇记忆点津与考点要览
080208	7115101620	2008-09-28	2008-11-28	计算机网络
080511	7800048381	2008-10-23	2009-01-23	英语网上文摘
NULL	NULL	NULL	NULL	数据库系统导论
NULL	NULL	NULL	NULL	亲密接触 ASP.NET
⋮	⋮	⋮	⋮	⋮

在该例中，属于右外连接，列出了连接语句右边关系 BOOK 中的所有元组，若该图书没有被借阅，则 BORROW 表中相应字段赋值为 NULL。

4. 复合条件连接

上面各个连接查询中，WHERE 子句中只有一个条件，WHERE 子句中也可以有多个条件，有多个连接条件的操作，称为复合条件连接。

【例 3-38】查询借阅了书名中含有“数据库”的图书的学生的借书证号、姓名、专业名、所借图书的 ISBN、书名和应还书时间。

```
SELECT STUDENT.借书证号, 姓名, 专业名, BORROW.ISBN, 书名, 应还书时间
FROM  STUDENT, BORROW, BOOK
WHERE STUDENT.借书证号 = BORROW.借书证号 AND BOOK.ISBN = BORROW.ISBN
     AND  书名 LIKE '%数据库%';
```

【例 3-39】查询计算机专业学生借阅图书的信息，包括学生的姓名、借书数、所借书的书名、应还时间。

```
SELECT 姓名, 借书数, 书名, 应还时间
FROM  STUDENT, BORROW, BOOK
WHERE STUDENT.借书证号 = BORROW.借书证号 AND BOOK.ISBN = BORROW.ISBN
     AND  专业名='计算机';
```

5. 集合运算

在连接查询中，还有一种比较特殊的查询，即集合查询。众所周知，简单查询的结果是元组的集合，那么多个简单查询的结果就可以进行集合的操作，在关系代数中，对于集合的基本操作主要包括并、交、差等运算，在 SQL 语言中，也提供了相应的运算符，分别为 UNION（∪）、INTERSECT（∩）、EXCEPT（−）。

注意：

（1）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必须相同。

（2）不同的 DBMS 所支持的集合操作不尽相同，如在 Access、SQL Server 2000 数据库中仅支持并运算，目前的 SQL Server 2005 就可以支持这三种集合运算。

【例 3-40】查询借阅了 ISBN 为 7040100959 或 781067224X 的图书的读者的借书证号、图书的 ISBN 及借书时间和应还时间。

```
(SELECT 借书证号, ISBN, 借书时间, 应还时间
FROM   BORROW
WHERE   ISBN='7040100959')
UNION
(SELECT 借书证号, ISBN, 借书时间, 应还时间
FROM    BORROW
WHERE   ISBN='781067224X');
```

查询结果为：

借书证号	ISBN	借书时间	应还时间
080101	7040100959	2008-09-01	2008-12-01
080105	781067224X	2008-09-10	2008-12-10
080208	781067224X	2008-10-09	2009-01-09
⋮	⋮	⋮	⋮

【例 3-41】查询同时借阅了 ISBN 为 7040100959 和 7115101620 的图书的读者的借书证号。

```
(SELECT 借书证号
FROM   BORROW
WHERE   ISBN='7040100959')
INTERSECT
(SELECT 借书证号
FROM    BORROW
WHERE   ISBN='7115101620');
```

【例 3-42】查询借阅了 ISBN 为 7040100959 而没有借阅 ISBN 为 7115101620 图书的读者

的借书证号。

```
(SELECT 借书证号
FROM   BORROW
WHERE   ISBN='7040100959')
EXCEPT
(SELECT 借书证号
FROM    BORROW
WHERE   ISBN='7115101620');
```

当然，对于以上集合查询也可以用其他 SELECT 语句实现，如例 3-40 可以表述为：

```
SELECT 借书证号, ISBN, 借书时间, 应还时间
FROM   BORROW
WHERE   ISBN='7040100959' OR ISBN='781067224X'
```

同理，例 3-41 可以用 AND 来实现，例 3-42 也可以表述为：

```
SELECT 借书证号
FROM   BORROW
WHERE   ISBN='7040100959' AND 借书证号 NOT IN
    (SELECT   借书证号
     FROM    BORROW
     WHERE   ISBN='7115101620');
```

3.3.3　嵌套查询

在 SQL 语言中，一个 SELECT-FROM-WHERE 语句称为一个查询块，在 WHERE 子句或 HAVING 短语所表示的条件中，可以使用一个查询块作为条件的一部分，这种将一个查询块嵌套在另一个查询块的 WHERE 子句或 HAVING 短语条件中的查询称为嵌套查询。例如：

```
SELECT 书名
FROM   BOOK
WHERE  ISBN  IN
    (SELECT ISBN
     FROM BORROW
     WHERE 借书时间='2008-09-01');
```

说明：在上例中，下层查询块“SELECT ISBN FROM BORROW WHERE 借书时间='2008-09-01'”是嵌套在上层查询块“SELECT 书名 FROM BOOK WHERE ISBN IN”的 WHERE 条件中的。上层的查询块又称为外层查询或父查询或主查询，下层查询块又称为内层查询或子查询。SQL 语言允许多层嵌套查询，即一个子查询中还可以嵌套其他子查询，用来表示复杂的查询。嵌套查询可以用一系列简单查询构成复杂的查询，明显地增强了 SQL 的查询能力。以层层嵌套的方式构造程序是 SQL 中“结构化”的含义所在。

需要特别指出的是，子查询的 SELECT 语句中不能使用 ORDER BY 子句，ORDER BY 子句永远只能对最终查询结果排序。

嵌套查询一般的求解方法是由里向外处理。即每个子查询在其上一级查询处理之前求解，子查询的结果用于建立其父查询的查找条件。嵌套查询主要包括以下几类：

（1）带有 IN 谓词的子查询。

（2）带有比较运算符的子查询（子查询一定要跟在比较符之后）。

（3）带有 ANY 或 ALL 谓词的子查询（使用 ANY 和 ALL 时必须同时使用比较运算符）。

（4）带有 EXISTS 谓词的子查询（查询结果不返回任何数据，只产生逻辑“真”或“假”）。

1. 带有 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，用于判断某个属性列值是否在子查询的结果中。在嵌套查询中，由于子查询的结果往往是一个集合，所以谓词 IN 是嵌套查询中最常使用的谓词。

【例 3-43】查询与“张彦”同一个专业学生的借书证号、姓名、性别和借书数。

查询与“张彦”同专业学习的学生，可以首先确定“张彦”的专业名，然后再查找所有该专业的学生。所以可以先分步完成此查询，然后再构造嵌套查询。

（1）确定“张彦”所在院系名。

```
SELECT 专业名
FROM   STUDENT
WHERE  姓名='张彦';
```

查询结果为：电子

（2）查找所有电子专业的学生基本信息。

```
SELECT 借书证号, 姓名, 性别, 借书数
FROM   STUDENT
WHERE  专业名 IN ('电子');
```

将（1）嵌入到（2）的条件中，构造嵌套查询的形式，表示为：

```
SELECT 借书证号, 姓名, 性别, 借书数
FROM   STUDENT
WHERE  专业名 IN
   ( SELECT 专业名
     FROM   STUDENT
     WHERE  姓名='张彦');
```

该查询还可以用自连接查询来实现：

```
SELECT S1.借书证号, S1.姓名, S1.性别, S1.借书数
FROM   STUDENT S1, STUDENT S2
WHERE  S1.专业名 = S2.专业名 AND S2.姓名='张彦';
```

IN 谓词用于判断某个属性列值是否在子查询的结果中，同样道理，NOT IN 则可以用来判断某个属性列是否不在子查询的结果中。

【例 3-44】查询没有借阅书名中含有“数据库”字样图书的计算机专业学生的借书证号、姓名、性别、专业名和借书数。

```
SELECT  借书证号, 姓名, 性别, 专业名, 借书数
FROM    STUDENT
WHERE   专业名= '计算机'  AND 借书证号 NOT  IN
       (SELECT   借书证号
        FROM    BORROW
        WHERE   ISBN  IN
              ( SELECT   ISBN
                FROM    BOOK
                WHERE  书名 like '%数据库%'));
```

本查询也可以用连接查询的形式表示：

```
SELECT  STUDENT.借书证号, 姓名, 性别, 专业名, 借书数
FROM    STUDENT, BORROW , BOOK
WHERE   STUDENT.借书证号 = BORROW.借书证号 AND
          BOOK.ISBN = BORROW.ISBN  AND
          专业名= '计算机'  AND 书名 like '%数据库%';
```

在例 3-43 和例 3-44 中，嵌套查询都可以用连接运算来代替，但并非所有的嵌套查询均可以用连接运算来表示，那么对于可以用连接运算代替的嵌套查询，读者可以根据具体的情况及其相应优化策略决定选择使用哪种查询方式。

此外，读者通过观察可以发现，在例 3-43 和例 3-44 中，子查询的查询条件不依赖于父查询，此类子查询被称为不相关子查询，此类子查询是较简单的一类子查询。其求解法则符合一般规律，从里向外，逐层进行。那么如果子查询的查询条件依赖于父查询，即为相关子查询，相应的嵌套查询也被称为相关嵌套查询。对于相关嵌套查询也有自己的求解算法：

（1）取外层查询表的第一个元组，根据它与内层查询相关的属性值处理内层查询，若 WHERE 子句返回值为真，则取此元组放入结果表。

（2）再取外层表的下一个元组。

（3）重复（1）、（2）这一过程，直至外层表全部检查完为止。

对于不相关子查询的实例，将在其他形式的子查询中进行讲解。

2. 带有比较运算符的子查询

使用带有比较运算符的子查询时，子查询一定要跟在比较运算符之后，常用的比较运算符有=，>，<，>=，<=，<>或!=等。有如下两种情况可以使用比较运算符：

（1）当用户可以确定内层查询返回的是单值时，则可以使用带有比较运算符的子查询。在例 3-43 中，张彦的专业名是唯一的，子查询返回值为单值，故父查询中的 IN 谓词也可以用比较运算符“=”代替，即表示为：

```
SELECT 借书证号, 姓名, 性别, 借书数
FROM   STUDENT
WHERE  专业名 =
   ( SELECT 专业名
     FROM   STUDENT
     WHERE  姓名='张彦');
```

【例 3-45】查询图书表中每个出版社的图书的库存量低于该出版社的所有图书的平均库存量的图书的 ISBN 和书名。

```
SELECT  ISBN, 书名
FROM    BOOK  B1
WHERE   库存量<= (SELECT  AVG(库存量)
                  FROM   BOOK  B2
                  WHERE   B2.出版社=B1.出版社);
```

本例中，使用了比较运算符“<=”，通过观察，读者可以发现本查询来自同一个表 BOOK，但是为其指定了两个别名 B1 和 B2，原因是子查询的查询条件需要依赖于父查询，因此此查询属于相关子查询，那么该查询的可能执行过程为：

1）从外层查询中取出 BOOK 表的第一个元组 x，将元组 x 的出版社（机械工业出版社）传送给内层查询，则为：

```
SELECT  AVG(库存量)
FROM   BOOK  B2
WHERE   B2.出版社='机械工业出版社';
```

2）执行内层查询，得到值 y，用该值代替内层查询，从而得到外层查询：

```
SELECT  ISBN, 书名
FROM    BOOK  B1
WHERE   库存量<=y;
```

3）执行该查询，假设得到如下两个元组：

（7871112133，数据库系统导论）

（7871112226，ASP.NET 程序设计教程（C#））

4）再从外层查询取出下一个元组，重复上述 1）～3）步骤，直到外层的 BOOK 表的元组全部处理完毕。

（2）当内层查询返回值的个数为多值时，则比较运算符要与 ANY 或 ALL 谓词配合使用。此类情况将在第三种嵌套查询带有 ANY 或 ALL 谓词的子查询中讲解。

3. 带有 ANY 或 ALL 谓词的子查询

ANY 或 ALL 谓词适用于返回多值的子查询，必须同时使用比较运算符，其语义如表 3-6 所示。

表 3-6 比较运算符

运算符	ANY	ALL
>	大于子查询结果中的某个值	大于子查询结果中的所有值
<	小于子查询结果中的某个值	小于子查询结果中的所有值
>=	大于等于子查询结果中的某个值	大于等于子查询结果中的所有值
<=	小于等于子查询结果中的某个值	小于等于子查询结果中的所有值
=	等于子查询结果中的某个值	通常没有实际意义
!=或<>	不等于子查询结果中的某个值	不等于子查询结果中的任何一个值

【例 3-46】查询其他专业比所有计算机专业的学生年龄都小的学生的信息。

```
SELECT  *
FROM   STUDENT
WHERE  专业名<> '计算机'  AND 出生年份>ALL
     (SELECT 出生年份
       FROM STUDENT
       WHERE 专业名='计算机' );
```

该查询的含义是：其他专业的学生的出生年份要比计算机专业中的所有学生的出生年份都大，就可以满足条件，也就是说，如果其他专业的某个学生比计算机专业的学生出生年份最大的还要大就满足选择条件，因此该查询也可以用集函数 MAX 实现。

```
SELECT  *
FROM   STUDENT
WHERE  专业名<> '计算机'  AND 出生年份>
      (SELECT MAX(出生年份)
       FROM STUDENT
       WHERE 专业名='计算机' );
```

事实上，用集函数实现子查询通常比直接用 ANY 或 ALL 查询效率要高，因为前者通常能够减少比较次数，ANY、ALL 与集函数的对应关系如表 3-7 所示。

表 3-7 ANY、ALL 谓词与集函数及 IN 谓词的等价转换关系

	=	<>或!=	<	<=	>	>=
ANY	IN	--	<MAX	<=MAX	>MIN	>= MIN
ALL	--	NOT IN	<MIN	<MIN	>MAX	>MAX

在表 3-7 中，可以看出<ANY 等价于<MAX，>ALL 等价于>MAX，>ANY 等价于>MIN，=ANY 等价于 IN 谓词等。

【例 3-47】查询其他专业比计算机专业的某个学生年龄小的学生的信息。

```
SELECT  *
FROM   STUDENT
WHERE  专业名<> '计算机'  AND 出生年份>ANY
     (SELECT 出生年份
       FROM STUDENT
       WHERE 专业名='计算机' );
```

该查询等价于：

```
SELECT  *
FROM   STUDENT
WHERE  专业名<> '计算机'  AND 出生年份>
     (SELECT MIN(出生年份)
       FROM STUDENT
       WHERE 专业名='计算机' );
```

4. 带有 EXISTS 谓词的子查询

EXISTS 谓词代表存在量词∃。带有 EXISTS 谓词的子查询不返回任何数据，只产生逻辑“真”或逻辑“假”。使用 EXISTS 谓词后，若内层查询结果非空，则外层的 WHERE 子句返回 TRUE，相反的，若内层查询结果为空，则外层的 WHERE 子句返回 FALSE。

【例 3-48】查询借阅 ISBN 为 730200899X 的图书的学生姓名、性别、专业名。

思路分析：本查询涉及 STUDENT 和 BORROW 关系。在 STUDENT 中依次取每个元组的“借书证号”值，用此值去检查 BORROW 关系，若 BORROW 中存在这样的元组，其“借书证号”值等于此“STUDENT.借书证号”值，并且其 ISBN ='730200899X'，则取此 STUDENT 的关系姓名、性别、专业名送入结果关系。

```
SELECT  姓名, 性别, 专业名
FROM    STUDENT
WHERE  EXISTS
     (SELECT  *
      FROM  BORROW
      WHERE   借书证号=STUDENT.借书证号 AND ISBN ='730200899X');
```

通过以上思路分析以及所给定的查询语句可知，本例中子查询的查询条件需要依赖于父查询，因此该查询属于相关子查询，故其执行过程遵守相关子查询的执行算法，但是读者通过分析可以发现，对于外层查询而言，每次取一个元组的“借书证号”值后，子查询只要执行到了有结果值满足查询条件，则可停止执行子查询，返回逻辑真值给父查询，然后继续取外层查询的第二个元组……。

此外，读者可以发现，在本例中，子查询没有指定具体的属性列，而是用“*”代替，这是带有 EXISTS 谓词的子查询的一个特点，因为父查询只关心子查询是否有值返回，而不关心返回的是什么值。因此子查询给出的列名无实际意义。

本例中的查询也可以用连接运算来实现，表示如下：

```
SELECT  姓名, 性别, 专业名
FROM    STUDENT, BORROW
WHERE  STUDENT.借书证号= BORROW.借书证号 AND ISBN ='730200899X';
```

与 EXISTS 谓词相对应的是 NOT EXISTS 谓词，使用 NOT EXISTS 谓词后，若内层查询

结果为空，则外层的 WHERE 子句返回 TRUE，相反，若内层查询结果非空，则外层的 WHERE 子句返回 FALSE。

【例 3-49】查询没有借阅 ISBN 为 730200899X 的图书的学生姓名、性别、专业名。

```
SELECT  姓名, 性别, 专业名
FROM    STUDENT
WHERE  NOT EXISTS
     (SELECT  *
      FROM  BORROW
      WHERE 借书证号=STUDENT.借书证号 AND ISBN ='730200899X');
```

此例用连接运算很难实现。对于不同的查询语句之间的替换，一般遵循如下原则：

（1）一些带 EXISTS 或 NOT EXISTS 谓词的子查询不能被其他形式的子查询等价替换，但有时可以用连接运算替换。

（2）所有带 IN 谓词、比较运算符、ANY 和 ALL 谓词的子查询都能用带 EXISTS 谓词的子查询等价替换。

因此，对于前面所讲例 3-43，查询与“张彦”同一个专业的学生的借书证号、姓名、性别和借书数，除了可以用 IN 谓词、比较运算符“=”和连接运算描述外，还有第四种使用 EXISTS 谓词的解法：

```
SELECT 借书证号, 姓名, 性别, 借书数
FROM   STUDENT S1
WHERE  EXISTS
     (SELECT *
       FROM   STUDENT S2
       WHERE  借书证号=S1.借书证号 AND 姓名='张彦');
```

由于带 EXISTS 谓词的相关子查询只关心内层查询是否有返回值，并不需要查具体的值，因此在子查询执行过程中，只要有值返回即跳出本次循环，故其效率并不一定低于其他形式的查询。

EXISTS/NOT EXISTS 谓词除了代表存在量词∃外，还可以实现全称量词∀（For All）和逻辑蕴函。在 SQL 语言中，没有全称量词∀和逻辑蕴函，但可以将其转换为带有存在量词的谓词。

（1）用 EXISTS/NOT EXISTS 实现全称量词。存在量词与全称量词的等价转换为：

$$(\forall x)P \equiv \neg(\exists x(\neg P))$$

【例 3-50】查询借阅了全部图书的学生的姓名、专业名。

思路分析：本查询的含义是，查询这样的学生，没有一本书他没有借的。其 SQL 语句如下：

```
SELECT 姓名, 专业名
FROM   STUDENT
WHERE  NOT EXISTS
     (SELECT  *
       FROM   BOOK
       WHERE  NOT EXISTS
            (SELECT  *
             FORM   BORROW
             WHERE 借书证号= STUDENT.借书证号 AND ISBN=BOOK.ISBN));
```

（2）用 EXISTS/NOT EXISTS 实现逻辑蕴函。利用谓词演算将逻辑蕴函谓词等价转换如下：

$$p \rightarrow q \equiv \neg p \vee q$$

【例 3-51】查询至少借阅了借书证号为 080208 的学生所借阅的全部图书的学生的借书证号。

思路分析：本查询的含义是，查询借书证号为 x 的学生，对于所有的图书，只要 080208 学生借阅了 y，则 x 也借阅了 y。

对于图书 y， 用 p 表示谓词“学生 080208 借阅了图书 y”，用 q 表示谓词“学生 x 选修了课程 y”，则上述查询可以表述为：

$$(\forall y)\ p \rightarrow q$$

根据存在量词和全称量词之间的等价变换及谓词演算可得：

$$\begin{aligned}(\forall y)p \rightarrow q &\equiv \neg (\exists y\ (\neg(p \rightarrow q\ )) \\ &\equiv \neg (\exists y\ (\neg(\neg\ p \vee q))) \\ &\equiv \neg\ \exists y(p \wedge \neg q)\end{aligned}$$

变换后语义：不存在这样的图书 y，学生 080208 借阅了，而学生 x 没有借。用 NOT EXISTS 谓词表示为：

```
SELECT DISTINCT 借书证号
FROM  BORROW B1
WHERE NOT EXISTS
     (SELECT  *
      FROM  BORROW B2
      WHERE 借书证号 = '080208'  AND  NOT EXISTS
            (SELECT *
            FROM BORROW B3
            WHERE B3.借书证号=B1.借书证号 AND B3.ISBN=B2.ISBN));
```

3.4　数据更新

关系数据库的数据更新包括插入、删除和修改三个方面的功能，这些功能均可以用 SQL 语言来实现。

3.4.1　插入数据

SQL 语言的数据插入通过 INSERT 语句实现，包括两类：插入单个元组和插入子查询结果。

1. 插入单个元组

使用 INSERT 语句实现插入单个元组的基本格式如下：

INSERT
INTO　<表名>　[(<属性列 1>[,<属性列 2>…])
VALUES　(<常量 1>[,<常量 2>]…);

说明：

（1）该语句的功能是将新元组插入指定表中。

（2）若 INTO 子句中表名后有各属性列选项，则插入的新元组的属性列 1 的值为常量 1，属性列 2 的值为常量 2……。如果某些属性列在 INTO 子句中没有出现，则新记录在这些列上将取空值。但必须注意的是，在表定义时说明了 NOT NULL 的属性列不能取空值，否则会出错。

（3）若属性列表和常量值表的顺序与表结构中的顺序相同，且给所有的属性列都指定值，

则可以省略属性列表。

（4）VALUES 子句提供的值，不管是值的个数还是值的类型必须与 INTO 子句匹配，否则系统会报错处理。

【例 3-52】将一个新学生记录（借书证号：080801；姓名：夏雨；性别：男；专业名：计算机；出生年份：1989-09；办证日期：2008-12-27）插入 STUDENT 表中。

```
INSERT
INTO  STUDENT(借书证号,姓名,性别,专业名,出生年份,办证日期)
VALUES ('080801','夏雨','男','计算机','1989-09','2008-12-27');
```

本例中，属性列表的顺序与表结构中的顺序不一致，因此不能省略 INTO 子句中的属性列表，但是如果本例表示为：

```
INSERT
INTO  STUDENT(借书证号,姓名,专业名,性别,出生年份,办证日期)
VALUES ('080801', '夏雨', '计算机', '男','1989-09', '2008-12-27');
```

此时，属性列表的顺序与表结构中顺序相同，且为每个属性列都指定了值，此时可省略属性列表，表示为：

```
INSERT
INTO  STUDENT
VALUES ('080801', '夏雨', '计算机', '男','1989-09', '2008-12-27');
```

【例 3-53】插入一条图书记录（'7800737985', '水煮三国', '成君忆', '中信出版社', '26.0'）。

```
INSERT
INTO  BOOK(ISBN,书名,作者,出版社,价格)
VALUES ('7800737985', '水煮三国', '成君忆', '中信出版社', '26.0');
```

则新插入的记录将在本数属性列上赋默认值 0，在库存量列上自动赋空值，本例也可以表示为：

```
INSERT
INTO  BOOK
VALUES ('7800737985', '水煮三国', '成君忆', '中信出版社', 26.0, 0,NULL);
```

2. 插入子查询结果

插入数据时，除了插入单个元组外，还可以将子查询嵌套在 INSERT 语句中，从而插入子查询的结果。插入子查询结果的 INSERT 语句格式如下：

INSERT
INTO <表名> [(<属性列 1>[,<属性列 2>…)]
子查询;

该语句中，子查询用以生成要插入的数据，整个语句的功能是批量插入，一次将子查询的结果全部插入指定表中。子查询中 SELECT 子句目标列不管是值的个数还是值的类型必须与 INTO 子句匹配。

【例 3-54】在数据库新建一个表，存放 STUDENT 表中各专业的学生人数。

首先在数据库中新建一张表，存放各专业的名称及学生人数。

```
CREATE TABLE Major_num
(专业名 char(12),
人数 int);
```

然后求得各专业的人数并插入新建的表中。

```
INSERT
INTO Major_num
SELECT 专业名,count(借书证号)
```

```
FROM STUDENT
GROUP BY 专业名;
```

注意：DBMS 在执行插入语句时会检查所插元组是否破坏表上已定义的完整性规则，包括实体完整性、参照完整性和用户定义的完整性。

3.4.2　修改数据

修改操作又称为更新操作，语句的一般格式如下：

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

该语句的功能是修改指定表中满足 WHERE 子句条件的元组。其中 SET 子句用于指定修改方式、要修改的列和修改后的取值，即用<表达式>的值取代相应的属性列值。WHERE 子句指定要修改的元组，如果省略 WHERE 子句，则表示要修改表中的所有元组。

注意：DBMS 在执行修改语句时会检查所修改元组是否破坏表上已定义的完整性规则，包括实体完整性（一些 DBMS 规定主码不允许修改）、参照完整性和用户定义的完整性。

更新包括如下几种操作：

1. 更新表中全部的数据

【例 3-55】将所有学生的借书数清为 0。

```
UPDATE  STUDENT
SET  借书数=0;
```

2. 更新表中某些元组的数据

【例 3-56】将清华大学出版社的图书的复本数和库存量加 5。

```
UPDATE  BOOK
SET  复本数=复本数+5,库存量=库存量+5
WHERE 出版社='清华大学出版社';
```

3. 带子查询的修改

【例 3-57】将计算机专业的所有学生的应还书日期改为 2009-01-01。

```
UPDATE  BORROW
SET  应还时间='2009-01-01'
WHERE  借书证号 IN
     (SELECT 借书证号
       FROM    STUDENT
       WHERE 专业名='计算机');
```

或表示为：

```
UPDATE  BORROW
SET  应还时间='2009-01-01'
WHERE  '计算机'=
     (SELECT 专业名
       FROM    STUDENT
       WHERE 借书证号= BORROW.借书证号);
```

3.4.3　删除数据

删除语句的一般格式如下：

DELETE

FROM <表名>

[WHERE <条件>];

该语句的功能是删除指定表中满足 WHERE 子句条件的元组，如果 WHERE 子句缺省表示要删除表中的全部元组，但表的定义仍在数据字典中，即 DELETE 语句删除的是数据库表中的数据，而不是表的定义，注意与 DROP 语句的用法进行区分。

删除包括如下几种操作：

1. 删除某个（某些）元组的值

【例 3-58】2005 级的学生毕业了，删除 STUDENT 表中 2005 级学生的记录（2005 级学生的借书证号是以 05 开头的）。

```
DELETE
FROM  STUDENT
WHERE 借书证号 like '05%';
```

2. 删除全部元组的值

【例 3-59】清空借阅记录表。

```
DELETE
FROM  BORROW;
```

3. 带子查询的删除语句

【例 3-60】删除计算机专业的所有学生的借阅记录。

```
DELETE
FROM  BORROW
WHERE 借书证号 in
     (SELECT  借书证号
       FROM  STUDENT
       WHERE 专业名='计算机');
```

或表示为：

```
DELETE
FROM  BORROW
WHERE '计算机'=
     (SELECT  专业名
       FROM  STUDENT
       WHERE 借书证号=BORROW.借书证号);
```

注意：DBMS 在执行删除语句时，会检查所删元组是否破坏表上已定义的参照完整性规则，检查是否不允许删除或是需要级联删除。

3.5 视图

视图是一种虚表，是从一个或几个基本表（或视图）导出的表，数据库中只存放视图的定义而不存放视图的数据，这些数据仍存放在导出视图的基本表中。因此如果基本表中的数据发生变化，那么从视图查询的数据也随之发生改变，因此有这样一个说法，视图就像一个窗口，透过它可以看到数据库中自己感兴趣的数据及其变化。

一个用户可以定义若干个视图，因此，用户的外模式就由若干基本表和若干视图组成。视图一旦被定义，就可以像基本表一样，可以对其查询和删除，在某些情况下还可以对其修改。

3.5.1　定义视图

在 SQL 语言中，定义视图的基本语句如下：

CREATE　VIEW　<视图名>[(<列名>[,<列名>]…)]
AS　<子查询>
[WITH　CHECK　OPTION];

该语句中，子查询可以是任意复杂的 SELECT 语句，可以来自一个表，也可以来自多个表，还可以来自一个或多个视图。但一般来说，SELECT 语句中不允许含有 ORDER BY 子句和 DISTINCT 短语。

WITH CHECK OPTION 选项指出在视图上进行 UPDATE、INSERT、DELETE 操作时要符合子查询中条件表达式所指定的限制条件。

【例 3-61】建立计算机专业学生借阅图书的视图 CS_VIEW，包括学生的借书证号、姓名、性别、所借书的 ISBN 和借书时间，且要保证对该视图进行修改和插入操作时都是计算机专业的学生。

```
CREATE VIEW CS_VIEW
AS SELECT  STUDENT.借书证号,姓名,性别,ISBN,借书时间
FROM  STUDENT,BORROW
WHERE STUDENT.借书证号= BORROW.借书证号 AND 专业名='计算机'
WITH CHECK OPTION ;
```

【例 3-62】创建一个视图 CS_VIEW_081010，该视图中定义的是计算机专业学生 2008 年 10 月 10 日前的借阅图书情况。

分析：该例可以直接对表进行查询，建立视图，也可以对视图进行查询建立视图。

```
CREATE VIEW CS_VIEW_081010
AS SELECT  *
FROM  CS_VIEW
WHERE 借书时间<='2008-10-10' ;
```

定义基本表时，为了减少数据库中的冗余数据，表中只存放基本数据，由基本数据经过各种计算派生出的数据一般是不存储的。由于视图中的数据并不实际存储，所以定义视图时可以根据应用的需要设置一些派生属性列。这些派生属性由于在基本表中并不实际存在，所以有时也称他们为虚拟列，带虚拟列的视图我们称为带表达式的视图。

【例 3-63】定义学生所借图书总价值的视图 TOTAL_PRICE，包括该学生的借书证号、姓名和总价值。

```
CREATE VIEW TOTAL_PRICE (借书证号, 姓名, 总价值)
AS SELECT BORROW.借书证号, 姓名, SUM(价格)
FROM STUDENT,BOOK,BORROW
WHERE STUDENT.借书证号= BORROW.借书证号 AND BOOROW.ISBN=BOOK.ISBN
GROUP BY BORROW.借书证号;
```

3.5.2　查询视图

视图定义后，用户就可以像查询基本表一样查询视图了。DBMS 在执行对视图的查询时，首先进行有效性检查，检查查询涉及的表、视图等是否在数据库中存在，如果存在，则从数据字典中取出查询涉及的视图的定义，把定义中的子查询和用户对视图的查询结合起来，转换成等价的对基本表的查询，然后再执行转换以后的查询。将对视图的查询转换为对基本表的查询

过程称为视图的消解（View Resolution）。

【例 3-64】查询计算机学院 2008 年 12 月 30 日借书的学生的借书证号、姓名和 ISBN。

```
SELECT 借书证号,姓名, ISBN
FROM  CS_VIEW
WHERE  借书时间= '2008-12-30';
```

DBMS 在执行此查询时，首先进行有效性检查，然后从数据字典中取出 CS_VIEW 视图的定义：

```
CREATE VIEW CS_VIEW
AS SELECT  STUDENT.借书证号,姓名,性别,ISBN,借书时间
FROM  STUDENT,BORROW
WHERE STUDENT.借书证号= BORROW.借书证号 AND 专业名='计算机'
WITH CHECK OPTION ;
```

再将二者进行合并消解，转换为对基本表的查询：

```
SELECT  STUDENT.借书证号,姓名, ISBN,
FROM  STUDENT,BORROW
WHERE STUDENT.借书证号= BORROW.借书证号 AND 专业名='计算机' AND 借书时间=
'2008-12-30';
```

一般来说，DBMS 都可以将对视图的查询正确转换为对基本表的视图，但是，当对有些视图进行查询时，可能会出现语法错误。

【例 3-65】查询学生所借图书的总价值超过 800 元的学生的借书证号、姓名和总价值。

```
SELECT 借书证号, 姓名, 总价值
FROM  TOTAL_PRICE
WHERE 总价值>800;
```

将该查询与对视图 TOTAL_PRICE 的定义结合起来，消解得到的查询语句为：

```
SELECT  BORROW.借书证号, 姓名, SUM(价格)
FROM   STUDENT,BOOK,BORROW
WHERE  STUDENT.借书证号= BORROW.借书证号
        AND BOOROW.ISBN=BOOK.ISBN AND SUM(价格)>800
GROUP BY BORROW.借书证号;
```

显然，转换为对基本表的查询语句是错误的，因为 WHERE 子句中不能用聚集函数作为条件表达式，正确的查询语句应该为：

```
SELECT  BORROW.借书证号, 姓名, SUM(价格)
FROM   STUDENT,BOOK,BORROW
WHERE  STUDENT.借书证号= BORROW.借书证号 AND BOOROW.ISBN=BOOK.ISBN
GROUP BY BORROW.借书证号 HAVING  SUM(价格)>800;
```

因此，当视图的定义中出现了聚集函数所生成的属性列时，如果要对该视图进行有条件限制的查询，应该直接对基本表进行查询。

3.5.3 更新视图

更新视图包括插入（INSERT）、删除（DELETE）和修改（UPDATE）三类操作，对视图的更新最终要转换为对基本表的更新。但并非所有的视图都是可更新的。那么到底什么样的视图是可更新的？若一个视图是从单个基本表导出的，并且只是去掉了某些行和列（不包括关键字），如视图 CS_VIEW，称这类视图为行列子集视图，一般来说，行列子集视图是允许更新的，且为了防止不合法的更新，在定义视图时要加上 WITH CHECK OPTION。此外，还有其他的一些视图也是允许更新的，但是确切的哪些视图可以更新是有待进一步研究的课题。在

DB2 中对视图的更新有如下限制：

（1）若视图的属性来自属性表达式或常数，则不允许对视图执行 INSERT 和 UPDATE 操作，但允许执行 DELETE 操作。

（2）若视图的属性来自库函数，则不允许对此视图更新。

（3）若视图定义中有 GROUP BY 子句，则不允许对此视图更新。

（4）若视图定义中有 DISTINCT 任选项，则不允许对此视图更新。

（5）若视图定义中有嵌套查询，并且嵌套查询的 FROM 子句涉及导出该视图的基本表，则不允许对此视图更新。

（6）若视图由两个以上的基本表导出，则不允许对此视图更新。

（7）如果在一个不允许更新的视图上再定义一个视图，这种二次视图是不允许更新的。

【例 3-66】将视图 CS_VIEW 中借书证号为“080102”的学生的借书时间改为 2008-12-25。

```
UPDATE  CS_VIEW
SET 借书时间='2008-12-25'
WHERE 借书证号='080102';
```

本例中该更新语句是可以执行的，且执行时也是转换为对基本表的更新。但是对于视图 CS_VIEW_081010 和 TOTAL_PRICE 的更新是不允许的。

3.5.4　撤消视图

撤消视图的语句格式如下：

DROP　VIEW　<视图名>

一个视图被删除后，由此视图导出的其他视图也将失效，用户应该使用 DROP VIEW 语句将它们一一删除。

3.5.5　视图的作用

通过前面的讲解，可能有的读者会问，对视图的操作最终要转换为对基本表的操作，而且对其进行更新还有很多的限制，既然如此，为什么还要使用视图呢？

合理的使用视图有如下几个优点：

（1）视图能够简化用户观点。在使用数据库的过程中，可能有部分数据是用户集中关心的数据，而且此数据要经过多次投影和连接操作才可获得，视图机制正好适应了用户的需要，用户所做的只是对一个虚表的简单查询，这个虚表是如何得来的、数据库是如何实现该查询的，用户不必关心，从而更加清晰地表达查询。

（2）视图在一定程度上保证了数据的逻辑独立性。第 1 章曾经介绍过，数据的逻辑独立性是指用户的应用程序与数据的逻辑结构是相互独立的，即数据的逻辑结构改变了，用户程序也可以不变。因为视图来自于基本表，因此如果基本表的结构发生了改变，则只需要修改定义视图的子查询，一般不需要修改基于视图的操作或应用程序，从而在一定程度上保证了数据的逻辑独立性。

（3）视图在一定程度上提高数据的安全性。有了视图机制就可以为不同的用户定义不同的视图，把数据对象限制在一定的范围内，也就是说，不同的用户只能看到与自己有关的数据，自动地对数据提供一定的安全保护。例如，在 STUDENT 表的基础上建立几个视图，分别包括各个专业的学生数据，这样就可以把学生的数据按照专业限制在不同的范围内，只有计算机

专业的老师才可以查看本专业的学生，而无权去查看其他学院专业的学生信息。视图机制是为数据提供安全保护的一种方法，此部分将在第 6 章数据库的安全性部分详细讲解。

3.6 数据控制

由 DBMS 提供统一的数据控制功能是数据库系统的特点之一，其控制主要包括 4 个方面：数据库的恢复、并发控制、完整性控制和安全性控制。这里所说的是 SQL 语言的安全性控制。SQL 语言所提供的数据的安全性控制主要包括两个方面，一是对用户或者角色授予操作权限，二是收回对某用户或者角色的权限，所使用的语句是 GRANT 和 REVOKE 语句。对于 SQL 语句数据库的安全性控制将在第 6 章详细介绍，这里只简单介绍其一般格式。

3.6.1 授予权限

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

```
GRANT <权限>[,<权限>]…
[ON <对象类型> <对象名>]
[TO <用户>[,<用户>]…
[WITH GRANT OPTION];
```

其语义为：将对指定操作对象的指定操作权限授予指定的用户或角色。

接受权限的用户可以是一个或多个具体用户，也可以是 PUBLIC 即全体用户。如果指定了 WITH GRANT OPTION 子句，则获得某种权限的用户还可以把这种权限再授予别的用户。如果没有指定 WITH GRANT OPTION 子句，则获得某种权限的用户只能使用该权限，但不能传播该权限。

3.6.2 回收权限

授予的权限可以由 DBA 或其他授权者用 REVOKE 语句收回，REVOKE 语句的一般格式如下：

```
REVOKE <权限>[,<权限>]…
[ON <对象类型> <对象名>]
[FROM<用户>[,<用户>]…
```

3.7 不完善的 SQL

SQL 是一个庞大的语言，是关系数据库的标准语言，但是，它远远不是完美的关系语言，它被承载的期望和责任以及自身的烦琐冗长这些缺陷所拖累。SQL 最大的不足之处是：它在很多方面不能合适地支持关系模型，因此，现在很多 SQL 产品被称为“关系的”完全是名不副实的。人们提出关系理论的目的是为了给实际的系统提供理论基础，但各开发商都没有将理论作为整体来实施。就某些方面来说，现在的“关系的”产品都没有能够真正贯彻关系理论，这是 SQL 的不完善之处。尽管如此，SQL 语言是关系数据库标准语言，现在市场上差不多每一种产品都支持它，数据库的专业人员也需要了解它。这也正是本章学习的目的。

本章小结

关系数据库 SQL 语言包括数据定义、数据查询、数据更新和数据控制四个部分。本章就是以 SQL 语言的这四个部分为主线进行介绍的。总体来说本章的知识点可以分为五个部分。

1. 数据定义

SQL 语言的数据定义功能包括定义表、定义视图和定义索引，基本表和视图都是表，但基本表是实际存储在数据库中的表，视图是虚表，它是从基本表或其他视图中导出的表。本部分主要讲的是对基本表的建立、修改和删除，以及对索引的建立和删除。建立基本表主要使用的语句是 CREATE TABLE，修改基本表的语句是 ALTER TABLE，删除基本表的语句是 DROP TABLE。

2. 数据查询

数据查询是本章的重点，也是本课程的重点，包括单表查询、连接查询和嵌套查询。连接查询和嵌套查询既是重点也是学习的难点，二者皆可来自多张表，可表达出复杂的 SQL 语句，这些需要读者在具体的使用中慢慢体会。SELECT 语句是 SQL 的核心语句，其语句的各个部分丰富多彩，不管是复杂的嵌套查询还是来自多张表连接查询都是对该语句的扩展，其一般格式如下：

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 1> [HAVING <条件表达式>]]
[ORDER BY <列名 2> [ASC|DESC]];

（1）目标列表达式格式：

1）[<表名>.] *

2）[<表名>.]<属性列名表达式>[，[<表名>.]<属性列名表达式>] ...

这里<属性列名表达式>可以由属性列、作用于属性列的集函数和常量的任意算术运算（+，–，*，/）组成的运算公式。

（2）聚集函数的格式：

{ COUNT / SUM / AVG / MAX / MIN } ([DISTINCT|ALL] <列名>)

（3）WHERE 子句中的条件表达式也可以是如下几种：

1）

<属性列名> θ { <属性列名> / <常量> / [ANY|ALL] (SELECT 语句) }

其中θ可以是=，>，<，>=，<=，<>或!=等比较运算符。

2）

<属性列名> [NOT] BETWEEN { <属性列名> / <常量> / (SELECT 语句) } AND { <属性列名> / <常量> / (SELECT 语句) }

3）

<属性列名> [NOT] IN { (<值 1>[,<值 2>] …) / (SELECT 语句) }

4）<属性列名> [NOT] LIKE <匹配串>

5）<属性列名> IS [NOT] NULL

6）[NOT] EXISTS (SELECT 语句)

7）

<条件表达式> { AND / OR } <条件表达式> [{ AND / OR } <条件表达式>] …

所有的查询语句都是对以上种种情况的扩展，读者在学习和使用中会发现 SQL 语句的博大精深。

3. 数据更新

关系数据库的数据更新包括插入、删除和修改三方面的功能，这些功能均可以用 SQL 语言中 INSERT 语句、DELETE 语句和 UPDATE 语句来实现。

4. 视图

视图是一种虚表，是从一个或几个基本表（或视图）导出的表，数据库中只存放视图的定义而不存放视图的数据，这些数据仍存放在导出视图的基本表中。对视图的操作同样也包括定义、查询、更新和删除。对视图定义的子查询可以是任意复杂的 SELECT 语句，可以来自一个表，也可以来自多个表，还可以来自一个或多个视图；对视图的查询可以像对基本表的操作一样；对视图的更新包括插入（INSERT）、删除（DELETE）和修改（UPDATE）三类操作，对视图的更新最终要转换为对基本表的更新，需要注意的是，并非所有的视图都是可更新的；删除视图所使用的语句仍为 DROP 语句。

5. 数据控制

由 DBMS 提供统一的数据控制功能是数据库系统的特点之一，包括数据的安全性控制、完整性控制、并发控制和恢复。本章只介绍了 SQL 的数据控制功能。SQL 语言用 GRANT 语句向用户授予操作权限，授予的权限也可以由 DBA 或其他授权者用 REVOKE 语句收回。

习题三

3.1 试述 SQL 语言的特点。

3.2 试述 SQL 体系结构和关系数据库模式之间的关系。

3.3 SQL 是如何实现实体完整性、参照完整性和用户定义完整性的？

3.4 设有两个基本表 R(A,B,C)和 S(A,B,C)，试用 SQL 查询语句表达下列关系代数表达式。

（1）R∩S　（2）R－S　（3）R∪S　（4）R×S

3.5　用 SQL 语句建立 2.10 中的四个表：

S(SNO,SNAME,STATUS,CITY)

P(PNO,PNAME,COLOR,WEIGHT)

J(JNO,JNANE,CITY)

SPJ(SNO,PNO,JNO,QTY)

其中，在 SPJ 表中，SNO、PNO 和 JNO 是外键，分别参照 S、P、J 中的相应字段。

3.6　针对上题中四个表，用 SQL 语句完成下述操作。

（l）找出天津市供应商的姓名和状态。

（2）找出所有零件的名称、颜色和重量。

（3）找出使用供应商 S1 所供零件的工程号码。

（4）找出工程项目 J2 使用的各种零件名称及其数量。

（5）找出上海厂商供应的所有零件号码。

（6）找出使用上海产的零件的工程名称。

（7）找出没有使用天津产零件的工程号码。

（8）把全部红色零件的颜色改成蓝色。

（9）将由供应商 S5 供给工程代码为 J4 的零件 P6 改为由 S3 供应，并作其他必要的修改。

（10）从供应商关系中删除 S2 的记录，并从供应零件关系中删除相应的记录。

（11）请将（S2，J6，P4，500）插入供应情况表。

3.7　对于教学数据库的三个基本表：

S(S# ,SNAME,AGE ,SEX)

SC(S# ,C# ,GRADE)

C(C# ,CNAME,TEACHAR)

试用 SQL 语句表达下列查询：

（1）查询“刘某”老师所授课程的课程号和课程名。

（2）查询年龄大于 23 岁的男同学的学号和姓名。

（3）查询学号为 S3 学生所学课程的课程号、课程名和任课教师名。

（4）查询“张小飞”没有选修的课程号和课程名。

（5）查询至少选修了 3 门课程的学生的学号和姓名。

（6）查询全部学生都选修了的课程编号和课程名称。

（7）在 SC 中删除尚无成绩的选课元组。

（8）把“高等数学”课的所有不及格成绩都改为 60。

（9）把低于总平均成绩的女同学的成绩提高 5%。

（10）向 C 中插入元组（'C8'，'VC++'，'王昆'）。

3.8　什么是基本表？什么是视图？两者的区别和联系是什么？

3.9　试述视图的优点。

3.10　所有的视图都是可更新的吗？为什么？

3.11　哪类视图是可更新的？哪类视图是不可更新的？并举例说明。

3.12 请为三建工程项目建立一个供应情况的视图，包括供应商代码（SNO）、零件代码（PNO）、供应数量（QTY）。针对该视图完成下列查询：

（1）找出三建工程项目使用的各种零件代码及其数量。

（2）找出供应商 S1 的供应情况。

3.13 针对 3.5 建立的表，用 SQL 语言完成下列操作：

（1）把对所有表的 INSERT 权限授予“张丽”，并允许她将此权限授予其他用户。

（2）把查询和修改 SPJ 表的权限授给用户“王伟”。