

前言

SQL 是使用最为广泛的数据库语言之一。不管你是应用开发人员、数据库管理人员、Web 应用设计人员，还是 Microsoft Office 用户，掌握良好的 SQL 知识对于与数据库打交道是很重要的。

读者对象

本书适合以下读者：

- q SQL 新手；
- q 希望快速学会使用 SQL；
- q 希望知道如何在应用程序开发中使用 SQL；
- q 希望在无人帮助的情况下有效快速地使用 SQL。

本书涵盖的 DBMS

一般来说，本书中所讲授的 SQL 可以应用到任何数据库管理系统（DBMS）。但是，因为所有 SQL 实现并不都是相同的，所以本书介绍的 SQL 主要适用于以下系统（需要时会给出特定的说明和注释）：

- q IBM DB2；
- q Microsoft Access；
- q Microsoft SQL Server；
- q MySQL；
- q Oracle；
- q PostgreSQL；
- q Sybase Adaptive Server。

本书中所有数据库和 SQL 脚本例子对于这些 DBMS 都是适用的。

本书约定

本书采用等宽字体表示代码，读者输入的文本与应该出现在屏幕上的文本也以等宽字型给出。如：

```
It will look like this to mimic the way text looks on your screen.
```

代码行前的箭头（>）表示行中代码太长，该行容纳不下。在符号后输入的所有字符都应该是前一行的内容。



说明：给出上下文讨论中比较重要的信息。



提示：为某任务给出建议或一种更简单的方法。



注意：提醒可能出现的问题，避免出现事故。

新术语，提供新的基本词汇的清晰定义。

表示读者可以自己输入的代码。

强调某个程序执行时的输出。

告诉读者将对程序代码进行逐行分析。

数据库基础

本章将介绍 SQL 究竟是什么，它能做什么事情。

1.1 数据库基础

你正在阅读一本 SQL 图书这个事实表明，你需要以某种方式与数据库打交道。SQL 正是用来实现这一任务的一种语言，因此在学习 SQL 本身以前，应该对数据库及数据库技术的某些基本概念有所了解。

你可能还没有意识到，其实你自己一直在使用数据库。每当你从自己的电子邮件地址簿里查找名字时，你就在使用数据库。如果你在某个因特网搜索站点上进行搜索，也是在使用数据库。如果你在在工作中登录网络，也需要依靠数据库验证自己的名字和密码。即使是在自动取款机上使用 ATM 卡，也要利用数据库进行 PIN 码验证和余额检查。

虽然我们一直都在使用数据库，但对究竟什么是数据库并不十分清楚。特别是不同的人可能会使用相同的数据库术语表示不同的事物，这更是加剧了这种混乱。因此，我们学习的良好切入点就是给出一张最重要的数据库术语清单，并加以说明。



基本概念回顾 下面是某些基本数据库概念的简要介绍。如果你已经具有一定的数据库经验，这可以用于复习巩固；如果你是一个数据库新手，这将给你提供一些必需的基本知识。理解数据库是掌握 SQL 的一个重要部分，如果有必要的话，你应该参阅一些有关数据库基础知识的书籍。

1.1.1 什么是数据库



数据库这个术语的用法很多，但就本书而言（以及从 SQL 的角度来看），数据库是一个以某种有组织的方式存储的数据集合。理解数据库的一种最简单的办法是将其想象为一个文件柜。此文件柜是一个存放数据的物理位置，不管数据是什么以及如何组织。

数据库（**database**）保存有组织的数据的容器（通常是一个文件或一组文件）。



误用导致混淆 人们通常用数据库这个术语来代表他们使用的数据库软件。这是不正确的，它是产生许多混淆的根源。确切地说，数据库软件应称为数据库管理系统（或 **DBMS**）。数据库是通过 **DBMS** 创建和操纵的。数据库可以是保存在硬设备上的文件，但也可以不是。在很大程度上说，数据库究竟是文件还是别的什么东西并不重要，因为你并不直接访问数据库；你使用的是 **DBMS**，它为你访问数据库。

1.1.2 表

在你将资料放入自己的文件柜时，并不是随便将它们扔进某个抽屉就完事了，而是在文件柜中创建文件，然后将相关的资料放入特定的文件中。

在数据库领域中，这种文件称为表。表是一种结构化的文件，可用于存储某种特定类型的数据。表可以保存顾客清单、产品目录，或者其他信息清单。

表（**table**） 某种特定类型数据的结构化清单。

这里关键的一点在于，存储在表中的数据是一种类型的数据或一个清单。决不应该将顾客的清单与订单的清单存储在同一个数据库表中。这样做将使以后的检索和访问很困难。应该创建两个表，每个清单一个表。

数据库中的每个表都有一个用来标识自己的名字。此名字是唯一的，这表示数据库中没有其他表具有相同的名字。



表名 使表名成为唯一的，实际上是数据库名和表名等因素的组合。有的数据库还使用数据库拥有者的名字作为唯一名的组成部分。这表示，虽然在相同数据库中不能两次使用相同的表名，但在不同的数据库中却可以使用相同的表名。

表具有一些特性，这些特性定义了数据在表中如何存储，如可以存储什么样的数据，数据如何分解，各部分信息如何命名，等等信息。描述表的这组信息就是所谓的模式，模式可以用来描述数据库中特定的表以及整个数据库（和其中表的关系）。

模式（**schema**） 关于数据库和表的布局及特性的信息。

1.1.3 列和数据类型

表由列组成。列中存储着表中某部分的信息。

列（**column**） 表中的一个字段。所有表都是由一个或多个列组成的。



理解列的最好办法是将数据库表想象为一个网格。网格中每一列存储着一条特定的信息。例如，在顾客表中，一个列存储着顾客编号，另一个列存储着顾客名，而地址、城市、州以及邮政编码全都存储在各自的列中。



分解数据 正确地将数据分解为多个列极为重要。例如，城市、州、邮政编码应该总是独立的列。通过把它分解开，才有可能利用特定的列对数据进行分类和过滤（如，找出特定州或特定城市的所有顾客）。如果城市和州组合在一个列中，则按州进行分类或过滤会很困难。

数据库中每个列都有相应的数据类型。数据类型定义列可以存储的数据种类。例如，如果列中存储的为数字（或许是订单中的物品数），则相应的数据类型应该为数值类型。如果列中存储的是日期、文本、注释、金额等，则应该用恰当的数据类型规定出来。

数据类型（datatype） 所容许的数据的类型。每个表列都有相应的数据类型，它限制（或容许）该列中存储的数据。

数据类型限制可存储在列中的数据种类（例如，防止在数值字段中录入字符值）。数据类型还帮助正确地分类数据，并在优化磁盘使用方面起重要的作用。因此，在创建表时必须对数据类型给予特别的关注。



数据类型兼容 数据类型及其名称是 SQL 不兼容的一个主要原因。虽然大多数数据类型得到一致的支持，但许多更为高级的数据类型却不是这样。更糟的是，我们偶然会发现相同的数据类型在不同的 DBMS 中具有不同的名称。对此用户毫无办法，重要的是在创建表结构时要记住这些差异。

1.1.4 行

表中的数据是按行存储的；所保存的每个记录存储在自己的行内。如果将表想象为网格，网格中垂直的列为表列，水平行为表行。

例如，顾客表可以每行存储一个顾客。表中的行编号为记录的编号。

行（row） 表中的一个记录。



是记录还是行？ 你可能听到用户在提到行（row）时称其为数据库记录（record）。在很大程度上，这两个术语是可以互相交换使用的，但从技术上说，行才是正确的术语。

1.1.5 主键

表中每一行都应该有可以唯一标识自己的一列（或一组列）。一个顾客表可以将顾客编号用于此目的，而包含订单的表可以使用订单 ID。雇员表可以使用雇员 ID 或雇员社会保险号。





主键（**primary key**）一列（或一组列），其值能够唯一标识表中每个行。

唯一标识表中每行的这个列（或这组列）称为主键。主键用来表示一个特定的行。没有主键，更新或删除表中特定行很困难，因为没有安全的方法保证只涉及相关的行。



应该总是定义主键 虽然并不总是都需要主键，但大多数数据库设计人员都保证他们创建的每个表具有一个主键，以便于以后的数据操纵和管理。

表中的任何列都可以作为主键，只要它满足以下条件：

- q 任意两行都不具有相同的主键值；
- q 每个行都必须具有一个主键值（主键列不允许 **NULL** 值）；
- q 主键列中的值不允许修改或更新；
- q 主键值不能重用（如果某行从表中删除，它的主键不能赋给以后的新行）。

主键通常定义在表的一列上，但这并不是必需的，也可以一起使用多个列作为主键。在使用多列作为主键时，上述条件必须应用到构成主键的所有列，所有列值的组合必须是唯一的（但单个列的值可以不唯一）。

还有一种非常重要的键，称为外键，我们将在第 12 章中介绍。

什么是 SQL

1.2 什么是 SQL

SQL（发音为字母 S-Q-L 或 sequel）是结构化查询语言（Structured Query Language）的缩写。SQL 是一种专门用来与数据库通信的语言。

与其他语言（如英语或 Java 或 Visual Basic 这样的程序设计语言）不一样，SQL 由很少的词构成，这是有意而为的。设计 SQL 的目的是很好地完成一项任务——提供一种从数据库中读写数据的简单有效的方法。

SQL 有如下的优点：

- q SQL 不是某个特定数据库供应商专有的语言。几乎所有重要的 **DBMS** 都支持 **SQL**，所以，学习此语言使你几乎能与所有数据库打交道。
- q SQL 简单易学。它的语句全都是由有很强描述性的英语单词组成，而且这些单词的数目不多。
- q SQL 尽管看上去很简单，但它实际上是一种强有力的语言，灵活使用其语言元素，可以进行非常复杂和高级的数据库操作。

下面我们将开始真正学习 **SQL**。



SQL 的扩展 许多 DBMS 供应商通过增加语句或指令,对 SQL 进行了扩展。这种扩展的目的是提供执行特定操作的额外功能或简化方法。虽然这种扩展很有用,但一般都是针对个别 DBMS 的,很少有两个以上的供应商支持这种扩展。

标准 SQL 由 ANSI 标准委员会管理,从而称为 ANSI SQL。所有主要的 DBMS,即使有自己的扩展,但都支持 ANSI SQL。各个实现有自己的名称,如 PL / SQL、Transact-SQL 等。

本书讲授的 SQL 主要是 ANSI SQL。在使用某种 DBMS 特定的 SQL 时,将会进行说明。



1.3 动手实践

与其他任何语言一样,学习 SQL 的最好方法是自己动手实践。为此,需要一个数据库和用来测试 SQL 语句的应用系统。

本书中所有章节采用的都是真实的 SQL 语句和数据表。附录 A 给出了具体的例子表,并提供了如何获得(或创建)它们的细节,以便读者能理解每一章讲授的内容。附录 B 介绍在各种应用系统中执行 SQL 所需的步骤。在进入下一章之前,强烈建议读者先熟悉这两个附录的内容,为以后的学习做好准备。

1.4 小结

这一章介绍了什么是 SQL 以及它为什么很有用。因为 SQL 是用来与数据库打交道的,所以,我们也复习了一些基本的数据库术语。

2.1 SELECT 语句

本章将介绍如何使用 SELECT 语句从表中检索一个或多个数据列。

2.1 SELECT 语句

正如第 1 章所述,SQL 语句是由简单的英语单词构成的。这些单词称为关键字,每个 SQL 语句都是由一个或多个关键字构成的。大概,最经常使用的 SQL 语句就是 SELECT 语句了。它的用途是从一个或多个表中检索信息。

关键字(keyword) 作为 SQL 组成部分的保留字。关键字不能用作表或列的名字。附录 E 列出了某些经常使用的保留字。

为了使用 SELECT 检索表数据,必须至少给出两条信息——想选择什么,以及从什么地方选择。



理解例子 本书各章中的样例 SQL 语句(和样例输出)使用了附录 A 中描述的一组数据文件。如果想要理解和试验这些样例(我强烈建议这样做),请参阅附录 A,它包含有如何下载或创建这些数据文件的说明。

重要的是理解 **SQL** 是一种语言而不是一个应用程序。给出 **SQL** 语句并显示语句输出的方法随不同的应用程序而变化。为帮助读者根据自己的环境使用相应的例子，附录 **B** 介绍了如何针对许多流行的应用程序及开发环境发布本书中给出的语句。如果读者需要了解某个应用程序，附录 **B** 中也给出了相应的建议。

2.2 检索单个列

我们将从简单的 **SQL SELECT** 语句开始介绍，此语句如下所示：

```
SELECT prod_name
FROM Products;
```

上述语句利用 **SELECT** 语句从 **Products** 表中检索一个名为 **prod_name** 的列。所需的列名在 **SELECT** 关键字之后给出，**FROM** 关键字指出从其中检索数据的表名。此语句的输出如下所示：

```
prod_name
-----
Fish bean bag toy
Bird bean bag toy
Rabbit bean bag toy
8 inch teddy bear
      12 inch teddy bear
      18 inch teddy bear
      Raggedy Ann
      King doll
      Queen doll
```



未排序数据 如果读者自己试验这个查询，可能会发现显示输出的数据顺序与这里的不同。出现这种情况很正常。如果没有明确排序查询结果（下一章介绍），则返回的数据的顺序没有特殊意义。返回数据的顺序可能是数据被添加到表中的顺序，也可能不是。只要返回相同数目的行，就是正常的。

如上的一条简单 **SELECT** 语句将返回表中所有行。数据没有过滤（过滤将得出结果集的一个子集），也没有排序。以后几章将讨论这些内容。



使用空格 在处理 **SQL** 语句时，其中所有空格都被忽略。**SQL** 语句可以在一行上给出，也可以分成许多行。多数 **SQL** 开发人员认为将 **SQL** 语句分成多行更容易阅读和调试。




结束 SQL 语句 多条 **SQL** 语句必须以分号（**;**）分隔。多数 **DBMS** 不需要在单条 **SQL** 语句后加分号。但特定的 **DBMS** 可能必须在单条 **SQL** 语句后加上分号。当然，如果愿意可以总是加上分号。事实上，即使不一定需要，但加上分号肯定没有坏处。这条规则的例外就是 **Sybase Adaptive Server**，它不允许 **SQL** 语句以分号结束。

SQL 语句和大小写 请注意，**SQL** 语句不区分大小写，因此 **SELECT** 与 **select** 是相同的。同样，写成 **Select** 也没有关系。

许多 SQL 开发人员喜欢对所有 SQL 关键字使用大写，而对所有列和表名使用小写，这样做使代码更易于阅读和调试。不过，一定要认识到虽然 SQL 是不区分大小写的，但表名、列名以及值可能不同（这有赖于具体的 DBMS 及其如何配置）。

2.3 检索多个列

要想从一个表中检索多个列，使用相同的 SELECT 语句。唯一的不同是必须在 SELECT 关键字后给出多个列名，列名之间必须以逗号分隔。


 当心逗号 在选择多个列时，一定要在列名之间加上逗号，但最后一个列名后不加。如果在最后一个列名后加了逗号，将出现错误。

下面的 SELECT 语句从 Products 表中选择 3 列：

```
SELECT prod_id, prod_name, prod_price
FROM Products;
```

与前一个例子一样，这条语句使用 SELECT 语句从表 Products 中选择数据。在这个例子中，指定了 3 个列名，列名之间用逗号分隔。此语句的输出如下：

prod_id	prod_name	prod_price
BNBG01	Fish bean bag toy	3.4900
BNBG02	Bird bean bag toy	3.4900
BNBG03	Rabbit bean bag toy	3.4900
BR01	8 inch teddy bear	5.9900
BR02	12 inch teddy bear	8.9900
BR03	18 inch teddy bear	11.9900
RGAN01	Raggedy Ann	4.9900
RYL01	King doll	9.4900
RYL02	Queen dool	9.4900

 数据表示 从上述输出可以看到，SQL 语句一般返回原始的、无格式的数据。数据的格式化是一个表示问题，而不是一个检索问题。因此，表示（如把上面的价格值显示为正确的十进制数值货币金额）一般在显示该数据的应用程序中规定。一般很少使用实际检索出的数据（没有应用程序提供的格式）。

2.4 检索所有列

除了指定所需的列外（如上所述，一个或多个列），SELECT 语句还可以检索所有的列而不必逐个列出它们。这可以通过在实际列名的位置使用星号（*）通配符来达到，如下所示：

```
SELECT *
FROM Products;
```

如果给定一个通配符（*），则返回表中所有列。列的顺序一般（但并不总是）是列在表定义中出现的物理顺序。但 SQL 数据很少这样（通常，数据返回给一个根据需要进行格式化或表示的应用程序）。严格说来，这不应该造成什么问题。



使用通配符 一般，除非你确实需要表中的每个列，否则最好别使用*通配符。虽然褂猛ㄅ浞 瞻芭屺鼓阙约荷∈拢 挥妹魅妨谐鏊 枇校 炯鞑恍枰 牧型ǔ; 峤档图炯骺陀 τ 贸绦虻男阅棚?/p>



检索未知列 使用通配符有一个大优点。由于不明确指定列名（因为星号检索每个列），所以能检索出名字未知的列。

2.5 小结

本章学习了如何使用 SQL 的 SELECT 语句来检索单个表列、多个表列以及所有表列。下一章将讲授如何排序检索出来的数据。

3.1 排序数据

本章将讲授如何使用 SELECT 语句的 ORDER BY 子句，根据需要排序检索出的数据。

3.1 排序数据

正如前一章所述，下面的 SQL 语句返回某个数据库表的单个列。但请看其输出，并没有特定的顺序。

```
SELECT prod_name
FROM Products;
```

```
prod_name
-----
Fish bean bag toy
Bird bean bag toy
Rabbit bean bag toy
8 inch teddy bear
12 inch teddy bear
18 inch teddy bear
Raggedy Ann
King doll
Queen doll
```

其实，检索出的数据并不是以纯粹的随机方式显示的。如果不排序，数据一般将以它在底层表中出现的顺序显示。这可以是数据最初添加到表中的顺序。但是，如果数据后来进行过更新或删除，则此顺序将会受到 DBMS 重用回收存储空间的影响。因此，如果不明确控制的话，不能（也不应该）依赖该排序顺序。关系数据库设计理论认为，如果不明确规定排序顺序，则不应该假定检索出的数据的顺序有意义。

子句（**clause**） SQL 语句由子句构成，有些子句是必需的，而有的可选。一个子句通常由一个关键字加上所提供的数据组成。子句的例子有 SELECT 语句的 FROM 子句，我们在前一章看到过这个子句。

为了明确地排序用 SELECT 语句检索出的数据，可使用 ORDER BY 子句。ORDER BY 子句取一个或多个列的 郑 菱硕允涑鼋 信判颀G脞聪旅嫫睦 樱?/p>

```
SELECT prod_name
FROM Products
ORDER BY prod_name;
```

这条语句除了指示 DBMS 软件对 prod_name 列以字母顺序排序数据的 ORDER BY 子句外，与前面的语句相同。

结果如下：

```
prod_name
-----
12 inch teddy bear
18 inch teddy bear
8 inch teddy bear
Bird bean bag toy
Fish bean bag toy
King doll
Queen doll
Rabbit bean bag toy
Raggedy Ann
```



ORDER BY 子句的位置 在指定一条 ORDER BY 子句时，应保证它是 SELECT 语句中最后一条子句。该子句的次序不对将会出现错误消息。



通过非选择列进行排序 通常，ORDER BY 子句中使用的列将是为显示所选择的列。但是，实际上并不一定要这样，用非检索的列排序数据是完全合法的。

3.2 按多个列排序

经常需要按不止一个列进行数据排序。例如，如果要显示雇员清单，可能希望按姓和名排序（首先按姓排序，然后在每个姓中再按名排序）。如果多个雇员具有相同的姓，这样做很有用。

为了按多个列排序，简单指定列名，列名之间用逗号分开即可（就像选择多个列时所做的那样）。

下面的代码检索 3 个列，并按其中两个列对结果进行排序——首先按价格，然后再按名称排序。

```
SELECT prod_id, prod_price, prod_name
FROM Products
ORDER BY prod_price, prod_name;
```

prod_id	prod_price	prod_name
BNBG02	3.4900	Bird bean bag toy
BNBG01	3.4900	Fish bean bag toy
BNBG03	3.4900	Rabbit bean bag toy
RGAN01	4.9900	Raggedy Ann
BR01	5.9900	8 inch teddy bear
BR02	8.9900	12 inch teddy bear
RYL01	9.4900	King doll
RYL02	9.4900	Queen doll
BR03	11.9900	18 inch teddy bear

重要的是理解在按多个列排序时，排序的顺序完全按所规定的进行。换句话说，对于上述例子中的输出，仅在多个行具有相同的 prod_price 值时才对产品按 prod_name 进行排序。如果 prod_price 列中所有的值都是唯一的，则不会按 prod_name 排序。

3.3 按列位置排序

除了能用列名指出排序顺序外，ORDER BY 还支持按相对列位置进行排序。理解这个内容的最好办法是看一个例子：

```
SELECT prod_id, prod_price, prod_name
FROM Products
ORDER BY 2, 3;
```

prod_id	prod_price	prod_name
-----	-----	-----
BNBG02	3.4900	Bird bean bag toy
BNBG01	3.4900	Fish bean bag toy
BNBG03	3.4900	Rabbit bean bag toy
RGAN01	4.9900	Raggedy Ann
BR01	5.9900	8 inch teddy bear
BR02	8.9900	12 inch teddy bear
RYL01	9.4900	King doll
RYL02	9.4900	Queen doll
BR03	11.9900	18 inch teddy bear

正如所见，这里的输出与上面的查询相同。不同之处在于 ORDER BY 子句。SELECT 清单中指定的是选择列的相对位置而不是列名。ORDER BY 2 表示按 SELECT 清单中第二个列，prod_name 列进行排序。ORDER BY 2, 3 表示先按 prod_price，再按 prod_name 进行排序。

此技术的主要好处在于不用重新输入列名。但它也有缺点。首先，不明确给出列名增加了错用列名排序的可能性。其次，在对 SELECT 清单进行更改时容易错误地对数据进行排序（忘记对 ORDER BY 子句做相应的改动）。最后，如果进行排序的列不在 SELECT 清单中，显然不能使用这项技术。



按非选择列排序 显然，当根据不出现在 SELECT 清单中的列进行排序时，这项技术不能采用。但是，如果有必要，可以混合匹配使用实际列名和相对列位置。

3.4 指定排序方向

3.4 指定排序方向

数据排序不限于升序排序（从 A 到 Z）。这只是默认的排序顺序，还可以使用 ORDER BY 子句以降序（从 Z 到 A）顺序排序。为了进行降序排序，必须指定 DESC 关键字。

下面的例子按价格以降序排序产品（最贵的排在最前面）：

```
SELECT prod_id, prod_price, prod_name
FROM Products
ORDER BY prod_price DESC;
```

prod_id	prod_price	prod_name
-----	-----	-----
BR03	11.9900	18 inch teddy bear
RYL01	9.4900	King doll
RYL02	9.4900	Queen doll
BR02	8.9900	12 inch teddy bear
BR01	5.9900	8 inch teddy bear
RGAN01	4.9900	Raggedy Ann
BNBG01	3.4900	Fish bean bag toy
BNBG02	3.4900	Bird bean bag toy
BNBG03	3.4900	Rabbit bean bag toy

但是，如果打算用多个列排序怎么办？下面的例子以降序排序产品（最贵的在最前面），然后再对产品名排序：

```
SELECT prod_id, prod_price, prod_name
FROM Products
ORDER BY prod_price DESC, prod_name;
```

prod_id	prod_price	prod_name
BR03	11.9900	18 inch teddy bear
RYL01	9.4900	King doll
RYL02	9.4900	Queen doll
BR02	8.9900	12 inch teddy bear
BR01	5.9900	8 inch teddy bear
RGAN01	4.9900	Raggedy Ann
BNBG02	3.4900	Bird bean bag toy
BNBG01	3.4900	Fish bean bag toy
BNBG03	3.4900	Rabbit bean bag toy

DESC 关键字只应用到直接位于其前面的列名。在上例中，只对 prod_price 列指定 DESC，对 prod_name 列不指定。因此，prod_price 列以降序排序，而 prod_name 列（在每个价格内）仍然按标准的升序排序。



在多个列上降序排序 如果想在多个列上进行降序排序，必须对每个列指定 DESC 关键字。

请注意，DESC 为 DESCENDING 的缩写，这两个关键字都可以使用。DESC 的反面是 ASC（或 ASCENDING），在升序排序时可以指定它。但实际上，ASC 没有多大用处，因为升序是默认的（如果既不指定 ASC 也不指定 DESC，则假定为 ASC）。



区分大小写和排序顺序 在对文本性的数据进行排序时，A 与 a 相同吗？a 位于 B 之前还是位于 Z 之后？这些问题不是理论问题，其答案依赖于数据库如何设置。

在字典（dictionary）排序顺序中，A 被视为与 a 相同，这是大多数数据库管理系统的默认行为。但是，许多 DBMS 允许数据库管理员在需要时改变这种行为（如果你的数据库包含大量外语字符，可能必须这样做）。

这里，关键的问题是，如果确实需要改变这种排序顺序，用简单的 ORDER BY 子句做不到。你必须请求数据库管理员的帮助。

3.5 小结

本章学习了如何用 SELECT 语句的 ORDER BY 子句对检索出的数据进行排序。这个子句必须是 SELECT 语句中的最后一条子句。可根据需要，利用它在一个或多个列上对数据进行排序。

4.1 使用 WHERE 子句

本章将讲授如何使用 SELECT 语句的 WHERE 子句指定搜索条件。

4.1 使用 WHERE 子句

数据库表一般包含大量的数据，很少需要检索表中所有行。通常只会根据特定操作或报告的需要提取表数据的子集。只检索所需数据需要指定搜索条件（search criteria），搜索条件也称为过滤条件（filter condition）。


在 SELECT 语句中，数据根据 WHERE 子句中指定的搜索条件进行过滤。WHERE 子句在表名（FROM 子句）之后给出，如下所示：

```
SELECT prod_name, prod_price
FROM Products
WHERE prod_price = 3.49;
```


这条语句从 products 表中检索两个列，但不返回所有行，只返回 prod_price 值为 3.49 的行，如下所示：

prod_name	prod_price
.....
Fish bean bag toy	3.4900
Bird bean bag toy	3.4900
Rabbit bean bag toy	3.4900

这个例子采用了简单的相等测试：它检查一个列是否具有指定的值，据此进行过滤。但是 SQL 允许做的事情不仅仅是相等测试。




PostgreSQL 例外 PostgreSQL 对传递给 SQL 语句的值具有严格的管理条件，特别是对于十进制数的列所用的数更是如此。因此，上面的例子对于 PostgreSQL 可能不起作用。为使这个例子在 PostgreSQL 中正常工作，可能需要在 WHERE 子句中包含类型，明确告诉 PostgreSQL，3.49 是一个合法的数。为此目的，应该将=3.49 替换为= decimal '3.49'。



SQL 过滤与应用过滤 数据也可以在应用层过滤。为此目的，SQL 的 SELECT 语句为客户机应用检索出超过实际所需的数据，然后客户机代码对返回数据进行循环，以提取出需要的行。

通常，这种实现并不令人满意。因此，对数据库进行了优化，以便快速有效地对数据进行过滤。让客户机应用（或开发语言）处理数据库的工作将会极大地影响应用的性能，并且使所创建的应用完全不具备可伸缩性。此外，如果在客户机上过滤数据，服务器不得不通过网络发送多余的数据，这将导致网络带宽的浪费。



WHERE 子句的位置 在同时使用 ORDER BY 和 WHERE 子句时，应该让 ORDER BY 位于 WHERE 之后，否则将会产生错误（关于 ORDER BY 的使用，请参阅第 3 章）。

4.2 WHERE 子句操作符

我们在关于相等的测试时看到了第一个 WHERE 子句，它确定一个列是否包含特定的值。SQL 支持表 4-1 列出的所有条件操作符。

表 4-1 WHERE 子句操作符

操 作 符	说 明
=	等于
< >	不等于
!=	不等于
<	小于
<=	小于等于
!<	不小于
>	大于
>=	大于等于
!>	不大于
BETWEEN	在指定的两个值之 间
IS NULL	为 NULL 值



操作符兼容 表 4-1 中列出的某些操作符是冗余的（如 < > 与 != 相同，!<（不小于）相当于 >=（大于等于））。并非所有 DBMS 都支持这些操作符。为了确定你的 DBMS 支持哪些操作符，请参阅相应的文档。

4.2.1 检查单个值

我们已经看到了测试相等的例子。现在来看看几个使用其他操作符的例子。

第一个例子是列出价格小于 10 美元的所有产品：

```
SELECT prod_name, prod_price
FROM Products
WHERE prod_price < 10;
```

prod_name	prod_price
.....
Fish bean bag toy	3.4900
Bird bean bag toy	3.4900
Rabbit bean bag toy	3.4900
8 inch teddy bear	5.9900
12 inch teddy bear	8.9900
Raggedy Ann	4.9900

King doll	9.4900
Queen doll	9.4900

下一条语句检索价格小于等于 10 美元的所有产品（因为没有价格恰好是 10 美元的产品，所以结果与前一个例子相同）：

```
SELECT prod_name, prod_price
FROM Products
WHERE prod_price <= 10;
```

4.2.2 不匹配检查

以下例子列出不是由供应商 DLL01 制造的所有产品：


```
SELECT vend_id, prod_name
FROM Products
WHERE vend_id <> 'DLL01';
```

vend_id	prod_name
-----	-----
BRS01	8 inch teddy bear
BRS01	12 inch teddy bear
BRS01	18 inch teddy bear
FNG01	King doll
FNG01	Queen doll



何时使用引号 如果仔细观察上述 **WHERE** 子句中使用的条件，会看到有的值括在单引号内，而有的值未括起来。单引号用来限定字符串。如果将值与串类型的列进行比较，则需要限定引号。用来与数值列进行比较的值不用引号。

下面是相同的例子，其中使用 **!=** 而不是 **<>** 操作符：

```
SELECT vend_id, prod_name
FROM Products
WHERE vend_id != 'DLL01';
```



是 **!=** 还是 **<>**？ **!=** 和 **<>** 通常可以互换使用。但是，并非所有 **DBMS** 都支持这两种不等于操作符。例如，**Microsoft Access** 支持 **<>** 而不支持 **!=**。如果有疑问，请参阅相应的 **DBMS** 文档。

4.2.3 范围值检查

为了检查某个范围的值，可使用 **BETWEEN** 操作符。其语法与其他 **WHERE** 子句的操作符稍有不同，因为它需要两个值，即范围的开始值和结束值。例如，**BETWEEN** 操作符可用来检索价格在 5 美元和 10 美元之间或日期在指定的开始日期和结束日期之间的所有产品。

下面的例子说明如何使用 **BETWEEN** 操作符，它检索价格在 5 美元和 10 美元之间的所有产品：

```
SELECT prod_name, prod_price
FROM Products
WHERE prod_price BETWEEN 5 AND 10;
```

prod_name	prod_price
-----	-----
8 inch teddy bear	5.9900
12 inch teddy bear	8.9900
King doll	9.4900
Queen doll	9.4900

从这个例子中可以看到，在使用 **BETWEEN** 时，必须指定两个值——所需范围的低端和高端值。这两个值必须用 **AND** 关键字分隔。**BETWEEN** 匹配范围中所有的值，包括指定的开始和结束值。

4.2.4 空值检查

在创建表时，表设计人员可以指定其中的列是否可以不包含值。在一个列不包含值时，称其为包含空值 **NULL**。

NULL 无值（no value），它与字段包含 **0**、空字符串或仅仅包含空格不同。



SELECT 语句有一个特殊的 WHERE 子句，可用来检查具有 NULL 值的列。这个 WHERE 子句就是 IS NULL 子句。其语法如下：

```
SELECT prod_name
FROM Products
WHERE prod_price IS NULL;
```

这条语句返回没有价格（空 prod_price 字段，不是价格为 0）的所有产品，由于表中没有这样的行，所以没有返回数据。但是，Vendors 表确实包含有具有空值的列，如果没有州数据，则 vend_state 列将包含 NULL 值（在没有 U.S. 地址时类似）：

```
SELECT vend_id
FROM Vendors
WHERE vend_state IS NULL;
```

```
vend_id
-----
FNG01
JTS01
```



DBMS 的特定操作符 许多 DBMS 扩展了标准的操作符集，提供了更高级的过滤选择。更多信息请参阅相应的 DBMS 文档。

4.3 小结

本章介绍了如何用 SELECT 语句的 WHERE 子句过滤返回的数据。我们学习了如何对相等、不相等、大于、小于、值的范围以及 NULL 值等进行测试。

5.1 组合 WHERE 子句

本章讲授如何组合 WHERE 子句以建立功能更强的更高级的搜索条件。我们还将学习如何使用 NOT 和 IN 操作符。

5.1 组合 WHERE 子句

第 4 章中介绍的所有 WHERE 子句在过滤数据时使用的都是单一的条件。为了进行更强的过滤控制，SQL 允许给出多个 WHERE 子句。这些子句可以两种方式使用，即：以 AND 子句的方式或 OR 子句的方式使用。

操作符（operator） 用来联结或改变 WHERE 子句中的子句的关键字。也称为逻辑操作符（logical operator）。

5.1.1 AND 操作符

为了通过不止一个列进行过滤，可使用 AND 操作符给 WHERE 子句附加条件。下面的代码给出了一个例子：

```
SELECT prod_id, prod_price, prod_name
FROM Products
WHERE vend_id = 'DLL01' AND prod_price <= 4;
```

此 SQL 语句检索由供应商 DLL01 制造且价格小于等于 4 美元的所有产品的名称和价格。这条 SELECT 语句中的 WHERE 子句包含两个条件，并且用 AND 关键字联结它们。AND 指示数据库管理系统软件只返回满足所有给定条件的行。如果某个产品由供应商 DLL01 制造，但它的价格高于 4 美元，则不检索它。类似，如果产品价格小于 4 美元，但不是由指定供应商制造的也不被检索。这条 SQL 语句产生的输出如下：

prod_id	prod_price	prod_name
BNBG02	3.4900	Bird bean bag toy
BNBG01	3.4900	Fish bean bag toy
BNBG03	3.4900	Rabbit bean bag toy

AND 用在 WHERE 子句中的关键字，用来指示检索满足所有给定条件的行。

5.1.2 OR 操作符

OR 操作符与 AND 操作符不同，它指示数据库管理系统软件检索匹配任一条件的行。事实上，许多 DBMS 在 OR WHERE 子句的第一个条件满足的情况下，不再计算第二个条件（在第一个条件满足时，不管第二个条件是否满足，相应的行都将被检索出来）。

请看如下的 SELECT 语句：

```
SELECT prod_name, prod_price
FROM Products
WHERE vend_id = 'DLL01' OR vend_id = 'BRS01';
```

此 SQL 语句检索由任一个指定供应商制造的所有产品的产品名和价格。OR 操作符告诉 DBMS 匹配任一条件而不是同时匹配两个条件。如果这里使用的是 AND 操作符，则没有数据返回。这条 SQL 语句产生的输出如下：

prod_name	prod_price
Fish bean bag toy	3.4900
Bird bean bag toy	3.4900
Rabbit bean bag toy	3.4900
8 inch teddy bear	5.9900
12 inch teddy bear	8.9900
18 inch teddy bear	11.9900
Raggedy Ann	4.9900

OR WHERE 子句中使用的关键字，用来表示检索匹配任一给定条件的行。

5.1.3 计算次序

WHERE 可包含任意数目的 AND 和 OR 操作符。允许两者结合以进行复杂和高级的过滤。

但是，组合 AND 和 OR 带来了一个有趣的问题。为了说明这个问题，来看一个例子。假如需要列出价格为 10 美元（含）以上且由 DLL01 或 BRS01 制造的所有产品。下面的 SELECT 语句使用 AND 和 OR 操作符的组合建立了一个 WHERE 子句：

```
SELECT prod_name, prod_price
FROM Products
WHERE vend_id = 'DLL01' OR vend_id = 'BRS01'
AND prod_price >= 10;
```

prod_name	prod_price
-----	-----
Fish bean bag toy	3.4900
Bird bean bag toy	3.4900
Rabbit bean bag toy	3.4900
18 inch teddy bear	11.9900
Raggedy Ann	4.9900

请看上面的结果。返回的行中有 4 行价格小于 10 美元，显然，返回的行未按预期的进行过滤。为什么会这样呢？原因在于计算的次序。SQL（像多数语言一样）在处理 OR 操作符前，优先处理 AND 操作符。当 SQL 看到上述 WHERE 子句时，它理解为由供应商 BRS01 制造的任何价格为 10 美元以上的产品，或者由供应商 D LL01 制造的任何产品，而不管其价格如何。换句话说，由于 AND 在计算次序中优先级更高，操作符被错误地组合了。

此问题的解决方法是使用圆括号明确地分组相应的操作符。请看下面的 SELECT 语句及输出：

```
SELECT prod_name, prod_price
FROM Products
WHERE (vend_id = 'DLL01' OR vend_id = 'BRS01')
    AND prod_price >= 10;
```

prod_name	prod_price
-----	-----
18 inch teddy bear	11.9900

这条 SELECT 语句与前一条的唯一差别是，这条语句中，前两个条件用圆括号括了起来。因为圆括号具有较 AND 或 OR 操作符高的计算次序，DBMS 首先过滤圆括号内的 OR 条件。这时，SQL 语句变成了选择由供应商 D LL01 或 BRS01 制造的且价格都在 10 美元（含）以上的任何产品，这正是我们所希望的。



在 WHERE 子句中使用圆括号 任何时候使用具有 AND 和 OR 操作符的 WHERE 子句，都应该使用圆括号明确地分组操作符。不要过分依赖默认计算次序，即使它确实是你想要的东西也是如此。使用圆括号没有什么坏处，它能消除歧义。

5.2 IN 操作符

IN 操作符用来指定条件范围，范围中的每个条件都可以进行匹配。IN 取合法值的由逗号分隔的清单，全都括在圆括号中。下面的例子说明了这个操作符：

```
SELECT prod_name, prod_price
FROM Products
WHERE vend_id IN ('DLL01','BRS01')
ORDER BY prod_name;
```

prod_name	prod_price
-----	-----
12 inch teddy bear	8.9900
18 inch teddy bear	11.9900

8 inch teddy bear	5.9900
Bird bean bag toy	3.4900
Fish bean bag toy	3.4900
Rabbit bean bag toy	3.4900
Raggedy Ann	4.9900



此 SELECT 语句检索供应商 DLL01 和 BRS01 制造的所有产品。IN 操作符后跟由逗号分隔的合法值清单，整个清单必须括在圆括号中。

如果你认为 IN 操作符完成与 OR 相同的功能，那么你的这种猜测是对的。下面的 SQL 语句完成与上面的例子相同的工作：

```
SELECT prod_name, prod_price
FROM Products
WHERE vend_id = 'DLL01' OR vend_id = 'BRS01'
ORDER BY prod_name;
```

prod_name	prod_price
12 inch teddy bear	8.9900
18 inch teddy bear	11.9900
8 inch teddy bear	5.9900
Bird bean bag toy	3.4900
Fish bean bag toy	3.4900
Rabbit bean bag toy	3.4900
Raggedy Ann	4.9900

为什么要使用 IN 操作符？其优点为：

- q 在使用长的合法选项清单时，IN 操作符的语法更清楚且更直观。
- q 在使用 IN 时，计算的次序更容易管理（因为使用的操作符更少）。
- q IN 操作符一般比 OR 操作符清单执行更快。
- q IN 的最大优点是可以包含其他 SELECT 语句，使得能够更动态地建立 WHERE 子句。第 11 章将对此进行详细介绍。

IN WHERE 子句中用来指定要匹配值的清单的关键字，功能与 OR 相当。

5.3 NOT 操作符

WHERE 子句中的 NOT 操作符有且只有一个功能，那就是否定它之后所跟的任何条件。因为 NOT 从不自己使用（它总是与其他操作符一起使用），它的语法与其他操作符有所不同。与其他操作符不一样，NOT 可以用在要过滤的列前，而不仅是在其后。

NOT WHERE 子句中用来否定后跟条件的关键字。

下面的例子说明 NOT 的使用。为了列出除 DLL01 之外的所有供应商制造的产品，可编写如下的代码：

```
SELECT prod_name
FROM Products
WHERE NOT vend_id = 'DLL01'
ORDER BY prod_name;
```

prod_name

```
12 inch teddy bear
18 inch teddy bear
8 inch teddy bear
King doll
Queen doll
```

这里的 NOT 否定跟在它之后的条件；因此，DBMS 不是匹配 vend_id 为 DLL01，而是匹配非 DLL01 之外的其他所有东西。

前面的例子也可以使用 <> 操作符来完成，如下所示：

```
SELECT prod_name
FROM Products
WHERE vend_id <> 'DLL01'
ORDER BY prod_name;
```

```
prod_name
-----
12 inch teddy bear
18 inch teddy bear
8 inch teddy bear
King doll
Queen doll
```

为什么使用 NOT？对于这里的这种简单的 WHERE 子句，使用 NOT 确实没有什么优势。但在更复杂的子句中，NOT 是非常有用的。例如，在与 IN 操作符联合使用时，NOT 使找出与条件列表不匹配的行非常简单。



MySQL 中的 NOT MySQL 不支持这里描述的 NOT 的格式。
在 MySQL 中，NOT 只用来否定 EXISTS（如 NOT EXISTS）。

5.4 小结

本章讲授如何用 AND 和 OR 操作符组合成 WHERE 子句，而且还讲授了如何明确地管理计算的次序，如何使用 IN 和 NOT 操作符。