第5章 Transact-SQL 基础

Transcat-SQL 即 T-SQL，是 SQL Server 数据管理系统的操作语言。很多管理 SQL Server 的操作，无论是通过图形界面的 SQL Server Management Studio，还是其他应用程序界面执行的操作，其实质基本都是调用了 T-SQL 语句。

T-SQL 是众多 SQL 中的一种。SQL 在数据库管理系统中如此重要，以至于微软将自己的产品命名为 SQL Server，另一款使用面也非常广泛的数据库管理系统 MySQL 也采用 SQL 来命名，可以说 SQL 是关系数据库系统应用的基础。

前面的章节中，在数据库、数据表的创建和管理中已经介绍了部分 T-SQL 的语句，如 Create Database、Create Table、Alter Database、Alter Table 等，本章将对 T-SQL 语句的相关内容进行更深入的介绍。

本章要点：

- T-SQL 概述
- T-SQL 数据操纵语句
- T-SQL 数据查询语句
- T-SQL 附加语言和通用表表达式(CTE)

5.1 T-SQL 概述

5.1.1 T-SQL 的发展

说起 T-SQL，不能不先说 SQL。而讨论 SQL，又不得不说 E. F. Codd。SQL 是关系数据库的标准语言，而 E. F. Codd 则是关系数据库理论的奠基者，E. F. Codd 对关系数据库的发展，以及对 SQL 的发展都做出了重大的贡献。

在 20 世纪 70 年代初，IBM 公司的 E. F. Codd 发表了 *A Relation Modal of Data for Large Shared Data Banks*，即《大型共享数据库的数据关系模型》，确立了关系数据库的概念。20 世纪 70 年代中期，IBM 公司首先使用该模型开发出了结构化英语查询语言(SEQUEL-Structed English Query Language)，作为其关系数据库原型 system R 系统的操作语言，实现了对关系数据库的信息检索；Oracle 公司则在 1979 年率先推出商用的 SQL。由于 SQL 简单易用，近似于自然语言，因此，自推出之后，受到了数据库软件开发厂商和用户的广泛欢迎。无论是 DB2、Oracle，还是 Sybase、Informix、SQL Server，以及 Visual Foxpro、Access 等都支持 SQL 作为查询语言。

1986 年 10 月，SQL 成为美国国家标准组织 ANSI 的关系数据管理系统操作语言的国家标准。由于 SQL 使用普遍性且发展迅速，ANSI 分别于 1989 年、1992 年、1999 年、2003 年

和 2006 年推出了 SQL 的 ANSI 标准的新版本。国际标准化组织(ISO)也将之确定为关系数据库语言的国际标准。

T-SQL 是微软公司在遵循 SQL 标准的基础上,经过进一步发展,应用于 SQL Server 数据库系统的操作语言。由于市场竞争和客户需求多样化等原因,不同软件厂商的数据库管理系统产品都在 SQL 国际标准的基础上,添加了各自的特色语句,SQL Server 的 T-SQL 如此,Oracle 的 PL/SQL 也是如此。这些带有各自特色的语言,虽然在一定程度上给标准的一致性带来了一些问题,但是不同的特色也进一步丰富了 SQL 标准的内涵,推动了 SQL 标准的进一步发展。

对于普通的开发人员来说,由于不同产品的 SQL 的基本语句是大致相同的,并不会带来太多平台切换的困难。因此,掌握 T-SQL,同样有利于更深入地去使用其他数据库产品。

5.1.2 T-SQL 的语言分类

T-SQL 中语句众多,非常丰富,按其功能不同可以大致分为以下 4 类。

- 数据控制语言(Data Control Language,DCL):用于安全性管理,可以确定哪些用户能查看或者修改数据,包含 GRANT、DENY、REVOKE、COMMIT、ROLLBACK 等。
- 数据定义语言(Data Definition Language,DDL):用于执行数据库管理任务,创建和管理数据库以及数据库中的各种对象,包含 CREATE、ALERT、DROP 等。
- 数据操纵语言(Data Manipulation Language,DML):用于在数据库中操纵各种对象、检索和修改数据,包含 SELECT、INSERT、UPDATE、DELETE 等。本部分语言中,根据对数据的影响情况又可以细分为数据操纵语言和数据查询语言。数据操纵语言是对数据库数据产生变更影响的语言,包含 INSERT、UPDATE、DELETE。数据查询语言指从数据库中获取与指定条件相符合的数据,而对原始数据不会产生变更影响的语言,主要是各种 SELECT 语句。
- 附加语言:包含变量、运算符、函数、流程控制语言和注释等。

数据控制语言将在第 10 章中进行介绍,数据定义语言已在数据库管理、数据表管理中做了部分介绍,在后续的索引管理、存储过程、视图、触发器等章节中还将介绍相关内容。本章着重介绍数据操纵语言和附加语言。

5.2 T-SQL 数据操纵语言

往数据表中输入数据、编辑数据和删除数据是数据库系统管理数据的三项最基本的操作。在 SQL Server 2022 中实现上述三项操作的基本语句是 INSERT、UPDATE 和 DELETE。

5.2.1 INSERT 插入数据

INSERT 是 SQL Server 插入数据的语句,其基本语法如下:

```
INSERT
    [ INTO ]
    {table_name or View_name}
{
```

```
    [ ( column_list ) ]
    [ < OUTPUT Clause > ]
    { VALUES ( ( { DEFAULT | NULL | expression } [ ,…n ] ) [ ,…n ] ) }
}
```


各主要参数的含义如下：

- table_name or View_name，接收插入数据的数据表或者视图名称。
- column_list，插入数据表或视图中的列的列表，是可选项。如果不添加列名的列表，则插入的数据值的个数和顺序要求与列在表或视图中一致。
- OUTPUT Clause，执行 INSERT 语句后，系统返回值的子句。
- VALUES，插入的数据值的列表，需要使用英文半角逗号分隔。

1. INSERT 插入单行数据

使用 INSERT 插入单行数据的代码较为简单，例如，以下代码为向“产品数据表”中输入一行数据：

```
INSERT INTO 产品数据表(ProductName,UnitPrice,Unit,Description) VALUES('联想(Lenovo)小新 Pro13.3 英寸',5990.00,'台','联想(Lenovo)小新 Pro13.3 英寸英特尔')
```

执行完毕后，“产品数据表”中的数据如图 5-1 所示，末行数据即为上述代码执行后新添到数据表中的数据。可以看出，数据值与列之间是根据代码中列的顺序和值的顺序一一对应的，即 ProductName 对应'联想(Lenovo)小新 Pro13.3 英寸'，UnitPrice 对应 5990.00，Unit 对应'台'，Description 对应'联想(Lenovo)小新 Pro13.3 英寸英特尔'。

图 5-1　INSERT 语句执行后数据表的变化

如果表中列的顺序比较明确，INSERT 语句中列列表也可以不显式列出，如上述代码，可以改写成以下代码：

```
INSERT INTO 产品数据表 VALUES('联想 Y450A - TSI',4899.00,'台','T6600 酷睿 2GBDDR320G 独立显存')
```

注意：在使用 INSERT 输入数据时，必须遵守表结构设计时设置的约束要求。例如主键由系统自动生成，不能手工输入；Not Null 约束要求列必须输入值。如果将上述语句改写为以下语句，就会出错，因为 ProductID 是标识列，不允许赋值。

```
INSERT INTO 产品数据表(ProductID,ProductName,UnitPrice,Unit,Description) VALUES(10,'联想 Y450A - TSI',4899.00,'台','T6600 酷睿 2GBDDR320G 独立显存')
```

但是如果确实需要强行向标识列中输入数据时，可以采用 SET IDENTITY_INSERT 语句，如：

```
SET IDENTITY_INSERT 产品数据表 on
```

```
INSERT INTO 产品数据表(ProductID,ProductName,UnitPrice,Unit,Description) VALUES(10,'联想
Y450A-TSI',4899.00,'台','T6600 酷睿 2GBDDR320G 独立显存')
```

如果对某列设置了默认值约束或允许为 NULL,则该列可以不输入,系统会自动取默认值或者将其设置为 NULL 值,但不输入数据的列不应该出现在列列表中。如下述代码是错误的,因为与原式相比,不需要输入 UNIT,因为 UNIT 设有默认值"台",但在列列表中出现了 UNIT,从而造成列列表与值列表的不匹配("列列表"有 4 项,而"值列表"只有 3 项)。

```
INSERT INTO 产品数据表(ProductName,UnitPrice,Unit,Description) VALUES('联想 Y450A-TSI',
4899.00,'T6600 酷睿 2GBDDR320G 独立显存')
```

2. INSERT 插入多行数据

INSERT 语句可以一次插入多行数据,插入的数据行可以以数据列表的形式列在 VALUES 列表中,如以下代码向"产品数据表"中输入了三条数据。

```
INSERT INTO 产品数据表(ProductName,UnitPrice,[Description])
VALUES ('联想',4899.00,'T66001'),
       ('联想',4999.00,'T66002'),
       ('联想',5099.00,'T66003')
```

INSERT 语句还可以插入来自另一张表中的数据,这时需使用查询子语句 SELECT。如以下代码表示从"产品"数据表中取出数据添加到"产品数据表"中,从代码中可以看到,两个数据表的列名不需要相同,但要求列数相同且第二张表中取出的数据符合第一张表中列出的约束要求。

```
INSERT INTO 产品数据表(ProductName,UnitPrice,Unit,Description) SELECT NAME,UPRICE,UNIT,P_
DESC from 产品
```

上述代码是将"产品"表中的所有数据添加到"产品数据表"中,如果只需要将"产品"表中的部分数据添加到"产品数据表"中,则可以使用 TOP 和 PERCENT 关键词。如以下代码表示将"产品"表中的 10 行数据添加到"产品数据表"中;而 PERCENT 表示百分比范围内的数据,如 TOP (10) PERCENT 表示取符合查询条件的数据集中 10%的数据。

```
INSERT TOP(10) INTO 产品数据表(ProductName,UnitPrice,Unit,Description) SELECT NAME,UPRICE,
UNIT,P_DESC from 产品
```

但是 TOP 和 PERCENT 指定的数据是无顺序随机的,如 TOP(10)取的并不一定是前 10 行,TOP (10) PERCENT 也并不一定是前 10%的数据,如果需要指定顺序,则需要添加 ORDER BY 子句。例如,以下代码从"产品"表中取出数据并按"P_ID"列的值从小到大排序后,再将前 10 行数据添加到"产品数据表"中。

```
INSERT TOP(10)INTO 产品数据表(ProductName,UnitPrice,Unit, Description) SELECT NAME,UPRICE,
UNIT,P_DESC from 产品 ORDER BY P_ID
```

5.2.2 UPDATE 更新数据

T-SQL 中更新数据的语句是 UPDATE,UPDATE 可以一次更新一行数据,也可以一次更新多行数据,还可以一次只更新一列的值,或一次更新多列的值。在 UPDATE 中可以通过更新列列表指定更新的列及数据值,通过 WHERE 条件子句可以指定更新的数据行。

UPDATE 语句的基本语法如下：

```
UPDATE table_or_View_name
SET < SET caluse expression > [{,< SET caluse expression >}, … ]
[WHERE < search condition >]
```

各主要参数的含义如下：

- table_or_View_name，被更改的数据表或视图名称。
- SET caluse expression，被更改的列的表达式，如“unit＝'PCS'”等。
- search condition，用于行数据筛选的条件表达式。

例如，需要将“产品数据表”中所有行的 UNIT 的值改为 PCS，则代码如下：

```
UPDATE 产品数据表 SET UNIT = 'PCS'
```

如果只需要将 ProductID 列值为 3 的数据行的 UNIT 列的值更新为 PCS，ProductName 列的值更改为“LENOVO 计算机”，则代码如下：

```
UPDATE 产品数据表 SET UNIT = 'PCS',ProductName = 'LENOVO 计算机'
WHERE ProductID = 3
```

其中，UNIT＝'PCS'，ProductName＝'LENOVO 计算机'是被更新的列列表及数据，使用英文半角逗号分隔；ProductID＝3 是被更新的行数据筛选条件。

如果一次更新的数据行较多，且只需要更新其中的部分数据行，那么可以配合使用 TOP 和 PERCENT 来完成。例如，下例更新产品数据表中满足条件 ProductID＝3 的三行数据。

```
UPDATE TOP 3 产品数据表 SET UNIT = 'PCS',ProductName = 'LENOVO 计算机' WHERE ProductID = 3
```

UPDATE 语句中被更改列的值可以使用常量，如前述例子；也可以使用表达式，如需要将“产品数据表”中的单价打 9 折销售，则可以使用如下代码，此句中 UnitPrice ＝ UnitPrice * .9 表示将 UnitPrice 列的值更改为原值的 90%。

```
UPDATE 产品数据表 SET UnitPrice = UnitPrice * .9
```

注意：在 UPDATE 中如果不指定 WHERE 条件子句，则修改的是数据表中的全部数据。由于更改过程往往是不可逆的，因此为了避免出现数据被误改，除非有确实的需要，否则指定 WHERE 条件是必需的。同样，修改后的数据必须要符合表所设置的各种约束的要求。

UPDATE 还可以借助另外一张表的信息来修改数据，如下例中实现了一个相对较为复杂的更新操作。从 Sales 表中取 qty 值来更新 titles 表的 ytd_sales 列的值，条件是满足 titles. title_id ＝ sales. title_id and sales. ord_date ＝ (select max(sales. ord_date) from sales))。

```
UPDATE titles SET ytd_sales = titles. ytd_sales + sales. qty From titles, sales where
titles. title_id = sales. title_id and sales. ord_date = (select max(sales. ord_date) from sales))
```

5.2.3 DELETE 删除数据

T-SQL 中删除数据的语句是 DELETE。DELETE 可以删除指定表的一行或者多行数

据。DELETE 的基本语法如下：

```
DELETE FROM table_or_View_Name [WHERE < search condition >]
```

各主要参数的含义如下：

- table_or_View_Name，指定要删除数据的表或者视图名称。
- search condition，指定要删除的行数据的条件。

例如，以下代码从“产品数据表”中删除 ProductID 列值为 3 的数据行。

```
DELETE FROM 产品数据表 WHERE ProductID = 3
```

如果不指定 WHERE 条件子句，则指定表中的所有数据都会被删除，如以下代码删除了数据表“产品数据表”中的所有数据。

```
DELETE FROM 产品数据表
```

注意：如果没有明确要求从表中删除所有数据，就应该使用 WHERE 子句指定要删除的数据行的条件；否则可能会出现数据被误删的严重后果。

DELETE 语句同样支持 TOP 语句，TOP 子句的作用与 INSERT、UPDATE 语句中的 TOP 相同，删除时会随机删除 TOP 指定的数据，由于未对数据排序，删除的数据可能是随机的。

如果需要删除表中的所有数据，除了使用 DELETE 语句外，还可以使用 TRUNCATE Table 语句，如以下代码与“DELETE FROM 产品数据表”一样都能把“产品数据表”中的数据删除：

```
TRUNCATE Table 产品数据表
```

相对来说，使用 TRUNCATE Table 语句删除表中的所有数据效率更高，且占用系统资源量要少。原因是 TRUNCATE Table 删除表中的数据时，并不会把删除操作记录在日志中，而且会立即释放表中数据和索引所占的空间。因此，在需要删除表中所有数据时，建议使用 TRUNCATE Table 语句。

注意：当删除表中数据时，需要符合表所设置的约束的要求。如果涉及外键约束，还可能会影响其他表中的数据或受到其他表中数据的影响，如删除的是主键表，外键关系是“级联”，则外键表中相关联的数据也会被删除。

删除表中的全部数据时，并不会删除表结构，对数据表结构的定义还将保留在数据库。要彻底删除数据表需要使用 DROP TABLE 语句。

5.3 T-SQL 数据查询语言

数据查询同样是数据库管理系统中最常用的操作之一。在 T-SQL 中，提供了众多功能强大的数据查询方式，可以供用户实现多角度、多条件的灵活的数据查询。

5.3.1 单表数据查询

在 T-SQL 中数据查询的基本语句是 SELECT。SELECT 语句最基本的应用如以下代码所示，表示从“产品数据表”中查询所有数据，包括所有列，“*”代表将所有列都显示出来。

因此,该段代码执行的结果如图 5-2 所示。

```
SELECT * FROM 产品数据表
```

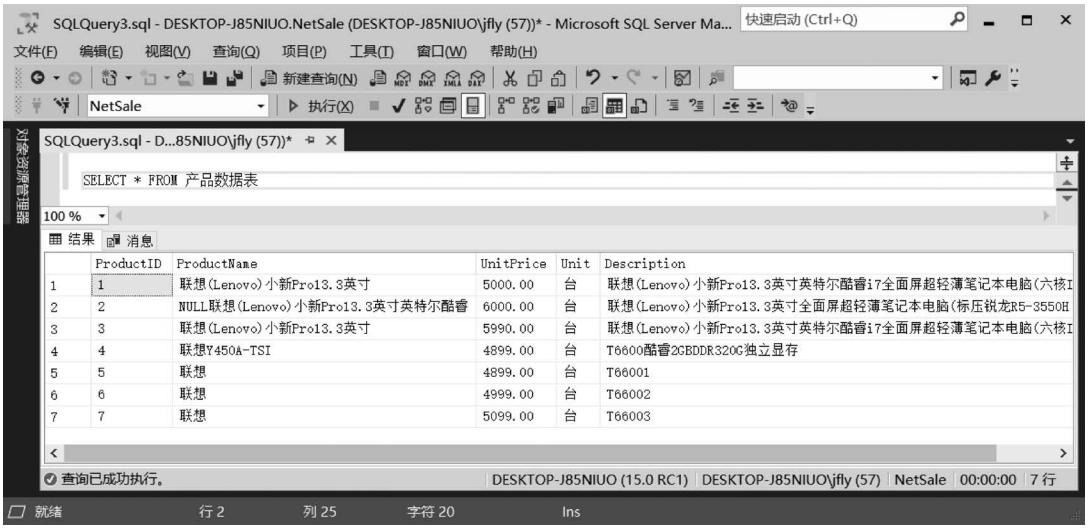

图 5-2　查询结果

1. 查询指定列的数据

在某些列数较多的数据表或者某些只需要显示部分列的应用中,如新闻网站的首页只需显示新闻标题等场合时,可以使用 SELECT 语句显示部分列查询需要的数据。此时,可以采用列列表来代替“*”,并且列列表中列的顺序可以与表结构中列的顺序不一致。如以下代码表示,从“产品数据表”中查看 ProductName、UnitPrice 两列的数据,查询结果如图 5-3 所示。

```
SELECT ProductName,UnitPrice FROM 产品数据表
```

2. 更改列标题的名称

在图 5-3 所示的查询结果中,列标题显示的是数据表中列的名称。在有些场合,需要将标题更改为更易于理解的名称,如将 ProductName 显示为“产品名称”更易于理解,且不会产生歧义。此时,可以通过更改列标题来实现,如以下代码的查询结果如图 5-4 所示。

```
SELECT ProductName AS '产品名称',UnitPrice AS '单价' FROM 产品数据表
```

如上代码中使用的 AS 关键词在 SELECT 语句中是一个可选项,也可以将其去掉,即以下代码可以完成同样的更改列标题显示的作用。

```
SELECT ProductName '产品名称', UnitPrice '单价' FROM 产品数据表
```

3. 数据运算

在 SELECT 语句中还可以添加各种常量、函数、表达式,对查询数据执行各种运算。如对数值型列可以在查询中执行算术运算,对字符型列可以执行字符串的合并、比较等各种运算。

例如以下代码可在 ProductName 前添加字符常量,并对 UnitPrice 列打 9 折后显示为“9 折单价”,其中“+”可以用于连接两个字符型数据列,或者连接字符型数据列和字符型

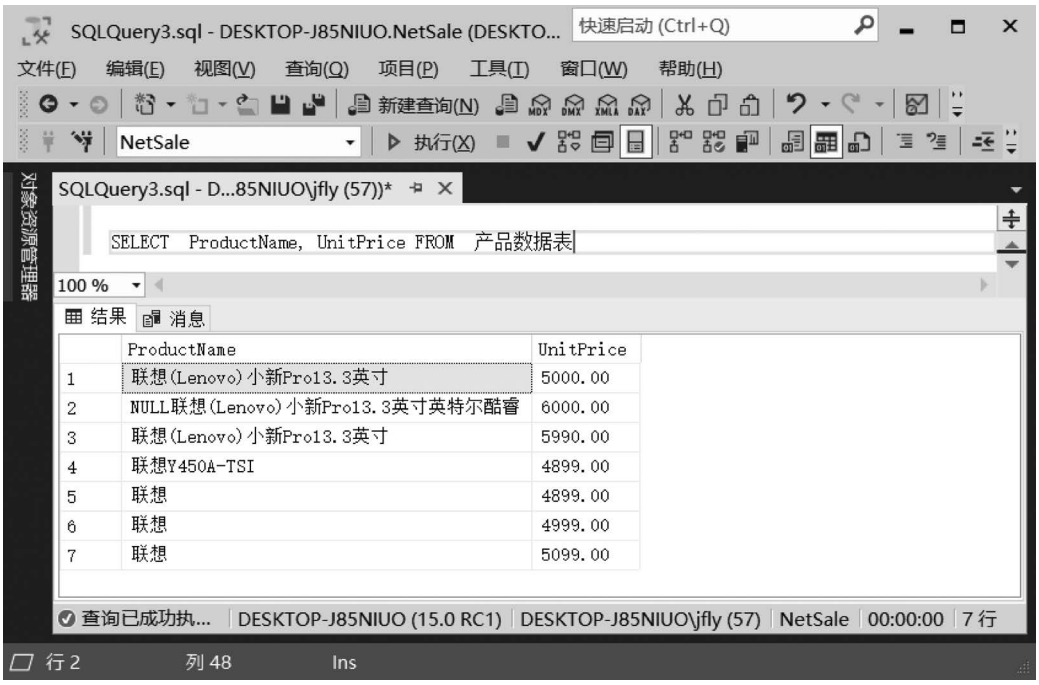

图 5-3　查询部分列的数据

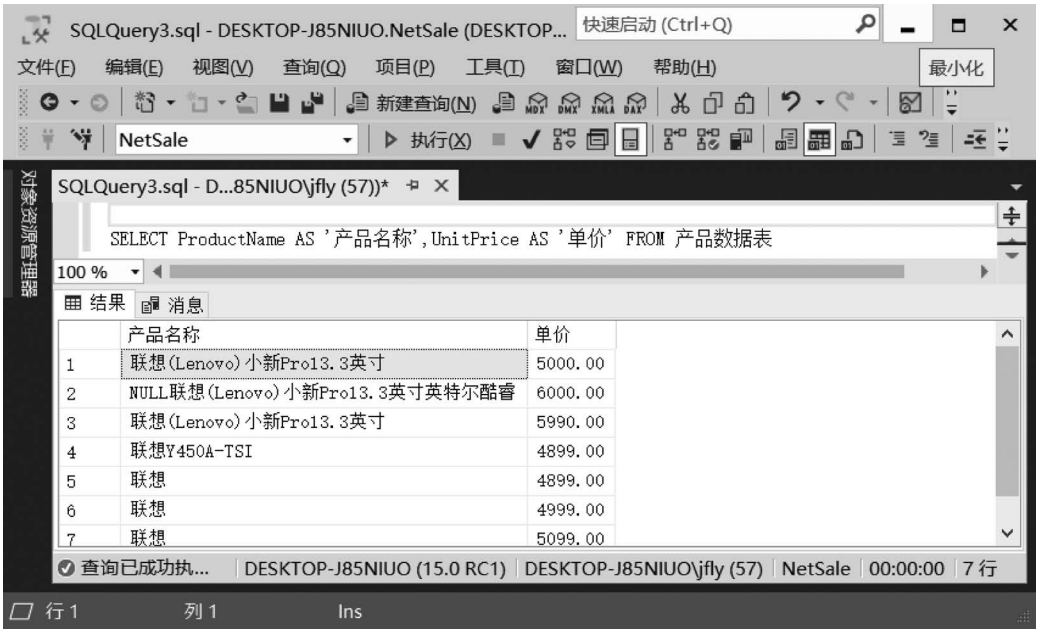

图 5-4　更改查询结果的标题

常量。查询执行结果如图 5-5 所示。SELECT 语句对列的运算不会改变表中的实际数据值。

```
SELECT '产品名称:' + ProductName,UnitPrice * 0.9 '9 折单价:' FROM 产品数据表
```

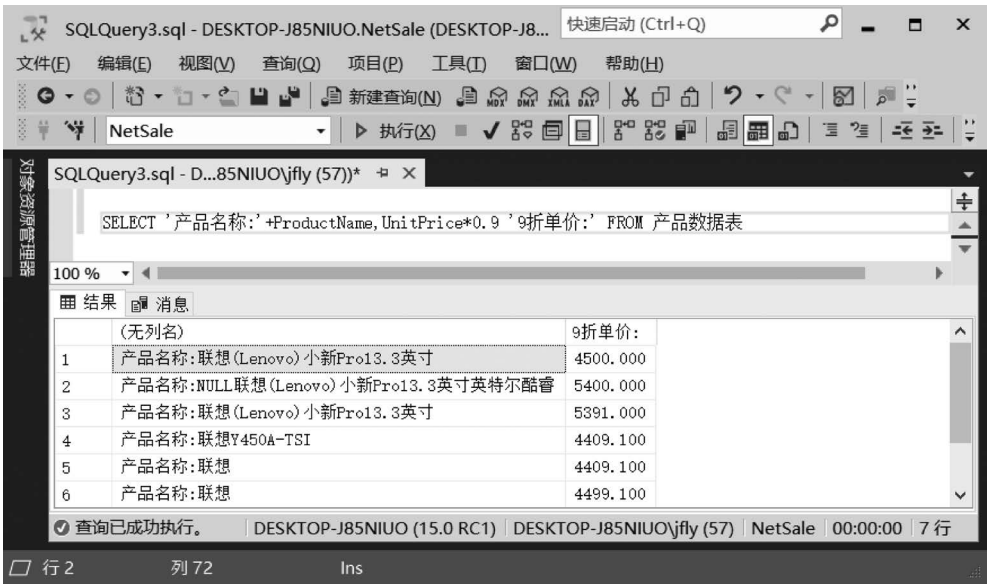

图 5-5　对列进行运算后的查询结果

如果需要对不同数据类型的列或数据执行运算,则需要将数据类型转换成相同的数据类型,在 T-SQL 中可以使用的转换函数有 CAST 和 CONVERT。这两个函数的语法如下:

```
CAST ( expression AS data_type [ (length ) ])
CONVERT ( data_type [ ( length ) ] , expression [ , style ] )
```

主要参数含义如下:

- Expression,需要转换的列或表达式。
- data_type,转换的目标数据类型。
- Length,目标数据类型的长度。
- Style,样式参数。

例如,需要将 ProductID 列从 int 数据类型转换成 varchar 数据类型,可以采用以下代码:

```
CAST(productid as varchar)
```

或

```
convert(varchar,productid)
```

经转换后,可以实现不同数据类型之间的运算,如以下代码将 ProductID 列转换成为 varchar 后与 ProductName 列和常量连接显示为“产品编号与名称”。查询执行结果如图 5-6 所示。

```
SELECT '产品编号' + convert(varchar,productid) + ',' + '产品名称:' + ProductName as '产品编号与名称',UnitPrice * 0.9 '9 折单价:'FROM 产品数据表
```

4. 简单的查询条件

如果不指定 WHERE 条件子句,SELECT 查询显示的是数据表中所有行的数据。但在很多场合,用户所需的往往不是全部数据,而是符合条件的部分数据,如在搜索引擎中用户输入的查询关键词,希望得到的是符合条件的数据;银行 ATM 刷卡后,希望得到的是与卡

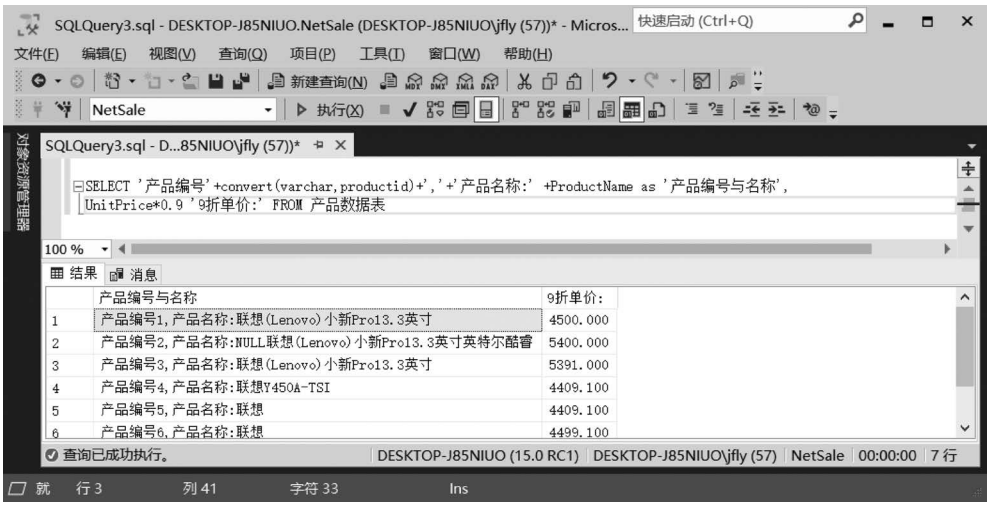

图 5-6　使用数据转换后的查询结果

号相关的数据。因此，设置 WHERE 条件，查询满足要求的数据在现实中的应用比查询全部数据更广泛。

如果只需要设定一个条件，如在“产品数据库”查询 ProductID 列值为 4 的数据，可以称为简单的查询条件，SELECT 语句可表示为：

```
SELECT * FROM 产品数据表 WHERE ProductID = 4
```

上述构建的条件 ProductID＝4，采用“＝”构建，属于精确匹配。在有些场合，如查询新闻时，可能不知道新闻标题的完整内容，但知道其中的部分关键词，则可以采用模糊匹配条件。模糊匹配采用的关键词是 LIKE，一般用于字符型数据的匹配条件中。如想从“产品数据表”中查询 ProductName 列中包含“联想”的产品时，条件表达式可以写为 ProductName like '%联想%'，则查询语句如下：

```
SELECT * FROM 产品数据表 WHERE ProductName like '%联想%'
```

查询结果如图 5-7 所示。其中需要说明的是，ProductName like '%联想%'中，两端的“%”是通配符，可以匹配任意字符。因此，该表达式的含义是，只要 ProductName 列中含有“联想”，不论“联想”两字出现在哪个位置，都是符合条件的。在 T-SQL 中共有 4 种通配符，除了“%”之外，还有三种通配符，分别为_、[]和[^]，这 4 个通配符代表的含义请参见表 5-1。

表 5-1　T-SQL 的通配符

通　配　符	含　　义
%	代表任意零个或多个字符构成的字符串
_	代表一个任意字符
[]	代表其间指定的范围或集合中的任意单个字符
[^]	代表不在指定范围或集合内的任意单个字符

例如：

- ProductName like '联想%'，表示检索“联想”两字符开头的 ProductName 列，如联想计算机、联想笔记本。
- ProductName like '%联想'，表示检索“联想”两字符结尾的 ProductName 列，如世

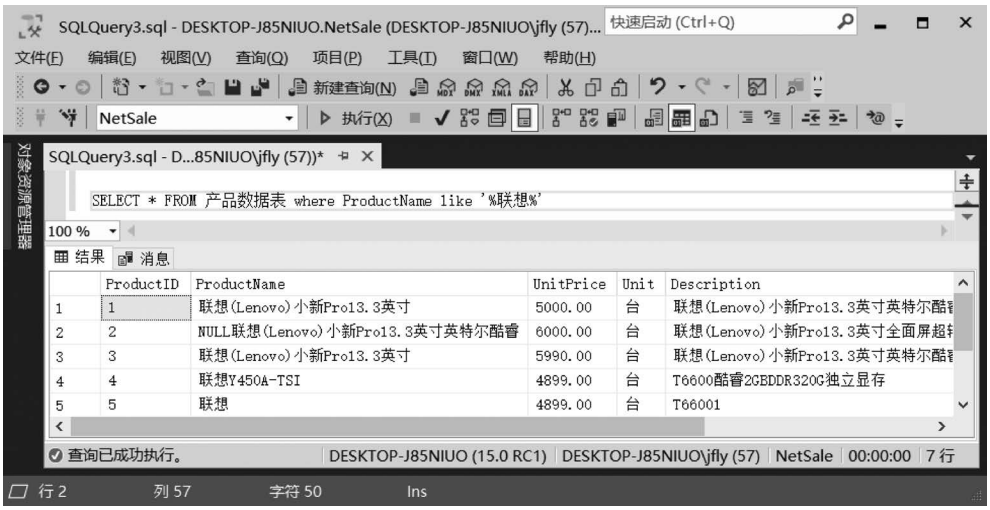

图 5-7　模糊查询的结果

纪联想、LENOVO 联想。

- ProductName like '_ab',表示检索以 ab 结尾的三个字符的 ProductName,如 Mab、cab 等。
- ProductName like 'a_b',表示检索以 a 开头,b 结束的三个字符的 ProductName,如 agb、aab 等。
- ProductName like 'ab_',表示检索以 ab 开头的三个字符的 ProductName,如 abc、abm 等。
- ProductName like '[abc]',表示检索含有 a 或 b 或 c 字符的 ProductName,如 adfghj、bdefhj、ddckjjd 等。
- ProductName like '[^abc]',表示检索不含有 a 或 b 或 c 中一个的 ProductName,如 ghk、ttmj 等。

例如,以下代码可以从 AdventureWorks2017 数据库的 Person.Address 表中查找 4 个字符的邮政编码,且第一个字符限制在 a～e,第二个字符限制在 a～z,最后两位为数字。查询执行的结果如图 5-8 所示。

```
SELECT city,postalcode,AddressLine1 FROM Person.Address
where postalcode like '[a-e][a-z][0-9][0-9]
```

5. 复合查询条件

如果查询条件中包含的条件不止一个,如要求同时满足两个以上的条件,或者满足几个条件中的一个,这样的复杂条件称为复合查询条件。复合查询条件根据逻辑关系的不同,可以分为“与”条件和“或”条件两种。

“与”条件表示要求同时满足两个以上的条件,使用 AND 关键词,可以构造“与”条件。“或”条件表示在几个条件中只要满足其中的一个,使用 OR 关键词,可以构造“或”条件。

例如,以下代码构造了一个“与”条件,即要求从“产品数据表”中查询既满足 ProductName 列中含有“联想”字符,且 UnitPrice 列低于 5000 的产品数据,查询结果如图 5-9 所示。

```
SELECT * FROM 产品数据表 where ProductName like '%联想%' AND UnitPrice<5000
```

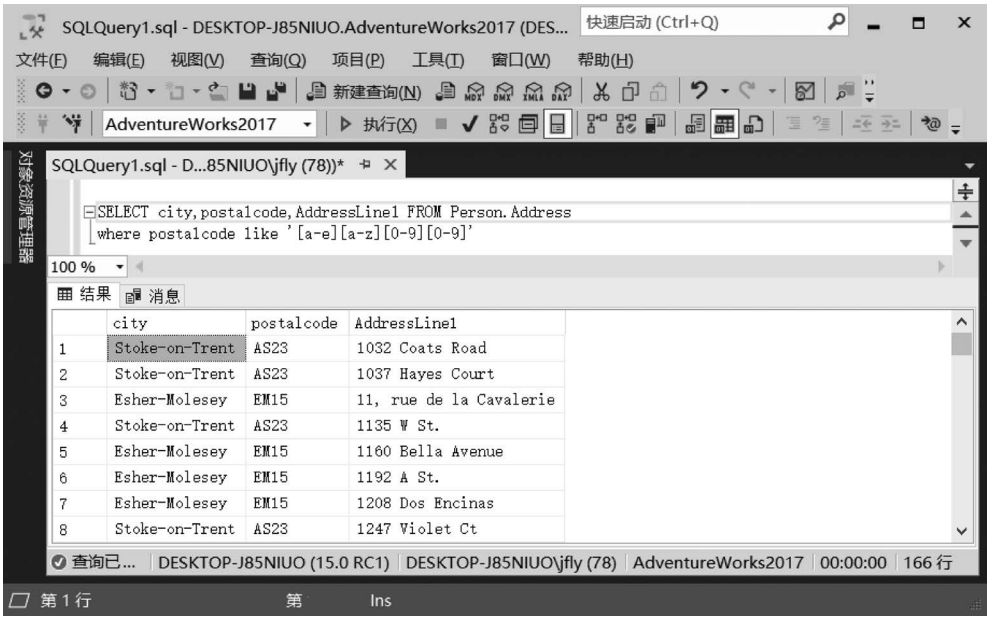

图 5-8　使用通配符的查询结果

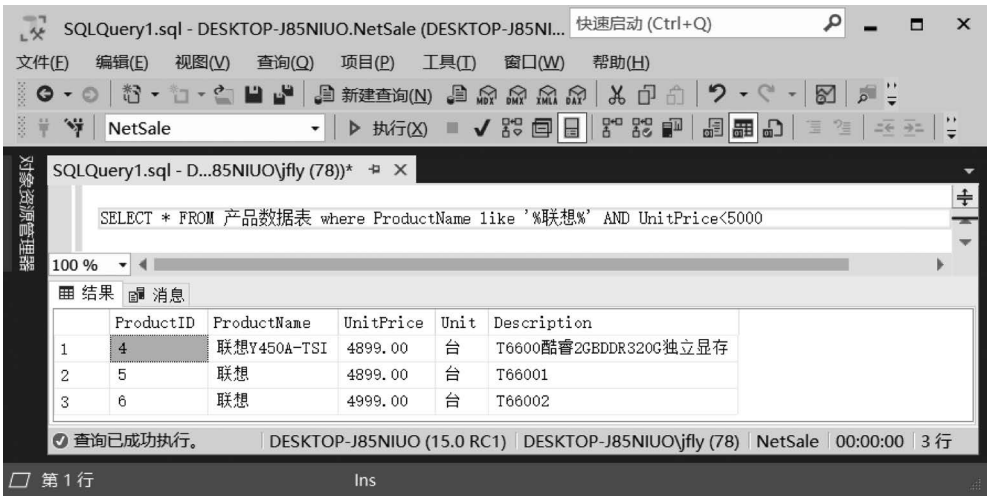

图 5-9　复合“与”条件的产品数据

再如以下代码构造了一个“或”条件，即要求查询 ProductName 列中含有“联想”字符，或者 UnitPrice 列低于 5000 的产品数据，查询结果如图 5-10 所示。

```
SELECT * FROM 产品数据表 where ProductName like '%联想%' OR UnitPrice<5000
```

“与”和“或”条件还可以组合在一起构成更加复杂的复合查询语句，如以下代码表示在“产品数据表”中查询同时满足 ProductName like '%联想%' 和 UnitPrice＜4000 两个条件，或者满足 ProductID＜8 条件的产品数据，查询结果如图 5-11 所示。由于 AND 运算的优先级高于 OR，前一个 AND 条件中的括号可以不写，但为了使代码容易阅读，还是应该添

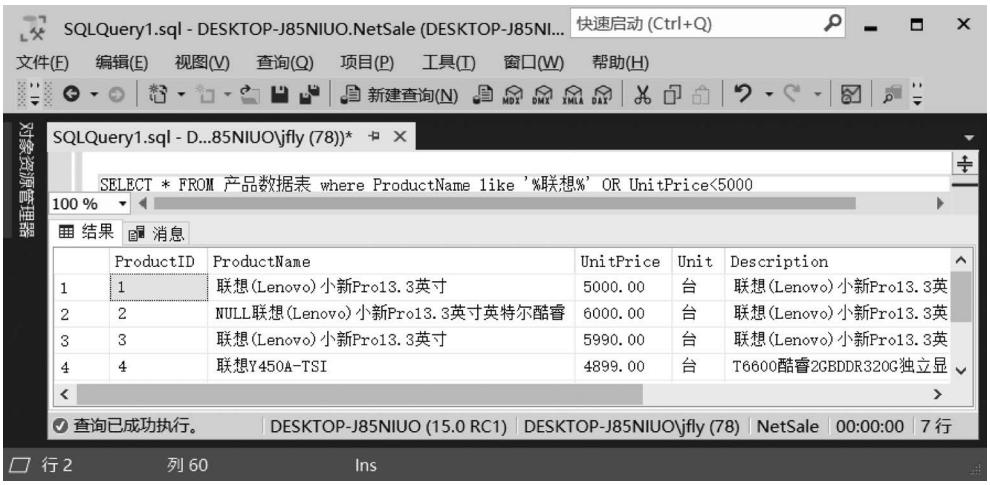

图 5-10　复合“或”条件的产品数据查询

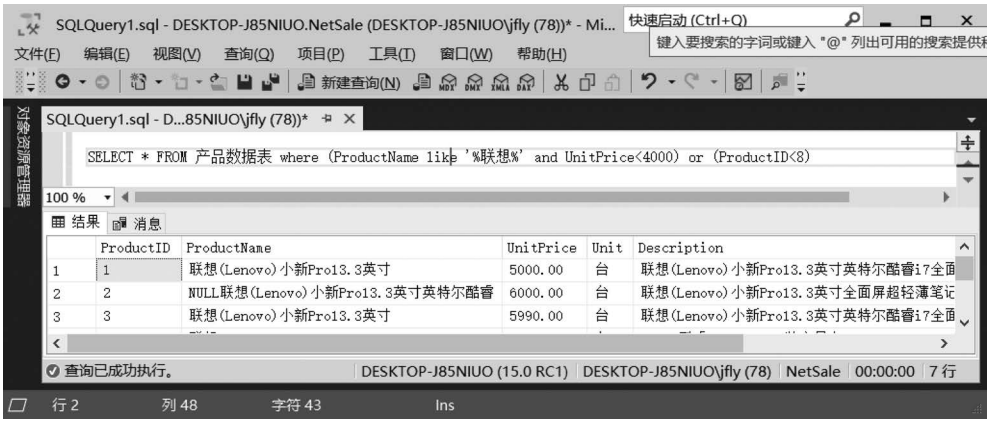

图 5-11　复杂的复合查询语句

加合适的括号。

```
SELECT * FROM 产品数据表 where (ProductName like '%联想%' and UnitPrice < 4000) or
(ProductID < 8)
```

注意：如果是字符串、日期时间作为条件值，请使用英文半角单引号，如 ProductName='联想'；如果数值作为条件值，可以直接写数字，如 ProductID<8。

AND 和 OR 也被称为逻辑运算符，构造的条件表达式也被称为逻辑表达式，T-SQL 还有一个逻辑运算符是 NOT，即对逻辑运算结果取反。

6. 使用比较运算符

T-SQL 支持采用=、>、<、>=、<=、<>、!=、!>、!<等比较运算符来构造查询条件，其中“<>”表示“不等于”，与“!=”含义相同。“!>”表示“不大于”，与“<=”含义相同，“!<”表示“不小于”，与“>=”含义相同。

在图 5-11 的例子中已介绍了 UnitPrice<4000 的使用形式，其他比较运算符的使用形式基本相同，不另作介绍。

7. 使用范围和列表

在某些情况下，要求查询的条件是某一个范围，如要求查询在 2010-01-01 到 2010-01-31 范围内的订单，或者销售量在 100～200 范围内的产品等，这时可以使用范围关键词 BETWEEN 和 AND 来构建条件。

例如，以下代码要求从 Orders 表中查询 orderdate 列在 2010-01-01 到 2010-01-15 范围内的订单数据。

```
Select orderid,customerid,employeeid,orderdate from orders where orderdate between '2010 - 01 - 01' and '2010 - 01 - 15'
```

如上代码也可以使用比较运算符来构造，如：

```
Select orderid,customerid,employeeid,orderdate from orders where orderdate >= '2010 - 01 - 01' and orderdate <= '2010 - 01 - 15'
```

虽然都能获得相同的结果，但相比较而言，BETWEEN 和 AND 范围语句相对简洁一些。

在另外一些场合，范围不是连续的范围，而是一些离散的值，这时可以采用列表。在 T-SQL 中列表可以使用 IN 关键词来构造，如以下代码表示要求从 authors 表查询满足 state 在"'CA','IN','MD'"三项列表中的数据。

```
Select * from authors where state in('CA','IN','MD')
```

如果列表范围是数值，可以使用数字值来构造列表，如以下代码表示从"产品数据表"中查询 ProductID 列在"1,3,4,8"列表范围中的产品数据。

```
SELECT * FROM 产品数据表 where ProductID IN (1,3,4,8)
```

8. 使用聚合函数

聚合函数包括 AVG、MIN、MAX、SUM、COUNT、STDEV、STDEVP、VAR、VARP，这些聚合函数可以在 SELECT 语句中实现对数据的统计。例如，以下代码分别使用上述聚合函数计算"产品数据表"中 UnitPrice 列的平均值、最小值、最大值、合计等。查询的执行结果如图 5-12 所示。

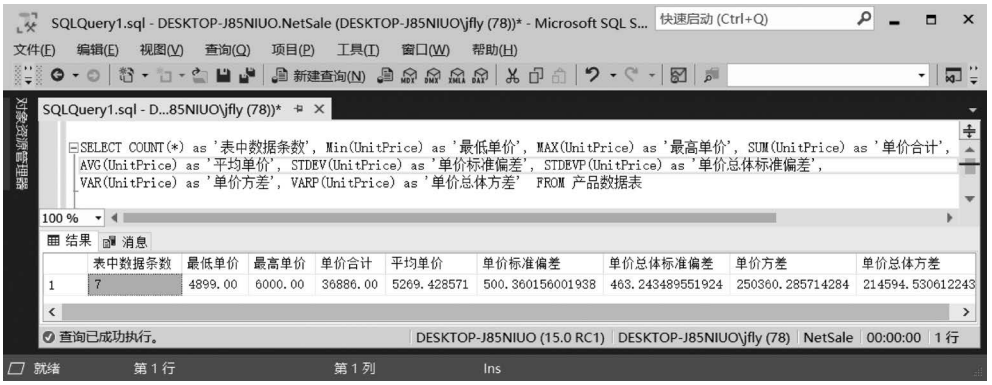

图 5-12　使用聚合函数查询的结果

```
SELECT COUNT( * ) as '表中数据条数', Min(UnitPrice) as '最低单价', MAX(UnitPrice) as '最高单价',
SUM(UnitPrice) as '单价合计',AVG(UnitPrice) as '平均单价',
STDEV(UnitPrice) as '单价标准偏差', STDEVP(UnitPrice) as '单价总体标准偏差',
```

```
VAR(UnitPrice) as '单价方差', VARP(UnitPrice) as '单价总体方差' FROM 产品数据表
```

9. 数据排序

在有些时候,用户要求查询获取的结果数据具有某种顺序。如新闻网站的新闻列表,往往会按时间排序,把最新的新闻排在最前面;在网络论坛中,很多人气比较高的帖子会被置顶。这些应用都需要对查询结果进行排序,在 T-SQL 中用来排序的语句是 ORDER BY。

ORDER BY 可以对数据按升序或者降序排列,升序使用的关键词是 ASC,降序使用的关键词是 DESC,系统默认设置是升序,即 ASC 可以省略。

例如,以下代码对"产品数据表"中的数据按 UnitPrice 列从低到高的顺序排列,查询执行的结果如图 5-13 所示。

```
SELECT * FROM 产品数据表 ORDER BY UnitPrice
```

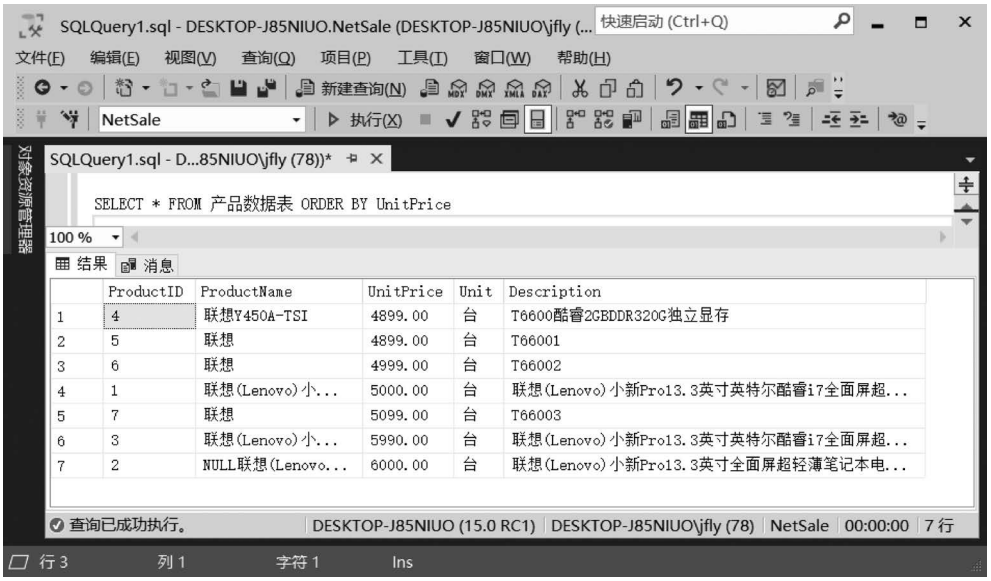

图 5-13 使用排序后的查询结果

ORDER BY 可以按多个列进行排序,如 ORDER BY UnitPrice,ProductID,排序时会先以第一个指定列的顺序进行排列,如果第一列的值相同,再按第二列的顺序进行排序,以此类推。在对多个列进行排序时,可以对每个列分别指定排列的顺序,如 ORDER BY UnitPrice DESC,ProductID,对第一列 UnitPrice 按从高到低的降序排列,第二列按默认设定的升序排列。

10. 去除重复数据

虽然在数据表中应用约束,如主键约束可以实现对数据完整性的检验,但并不能完全避免数据重复,如在某些列上出现相同内容。事实上在某些列出现数据相同,有时也是业务所需要的,如"订单细节表"往往会出现 orderid 列相同的情况,因为有多行订单细节数据是属于同一订单的。但是在有些情况应该避免在查询结果中显示重复的数据,如新闻网站中新闻列表的标题不应该出现重复等。

在 T-SQL 中可以使用 DISTINCT 关键词来避免在查询结果中显示重复数据,如在不使用 DISTINCT 时,"产品数据表"的查询结果中出现了重复(第 2、3、4 行数据),如图 5-14 所示。

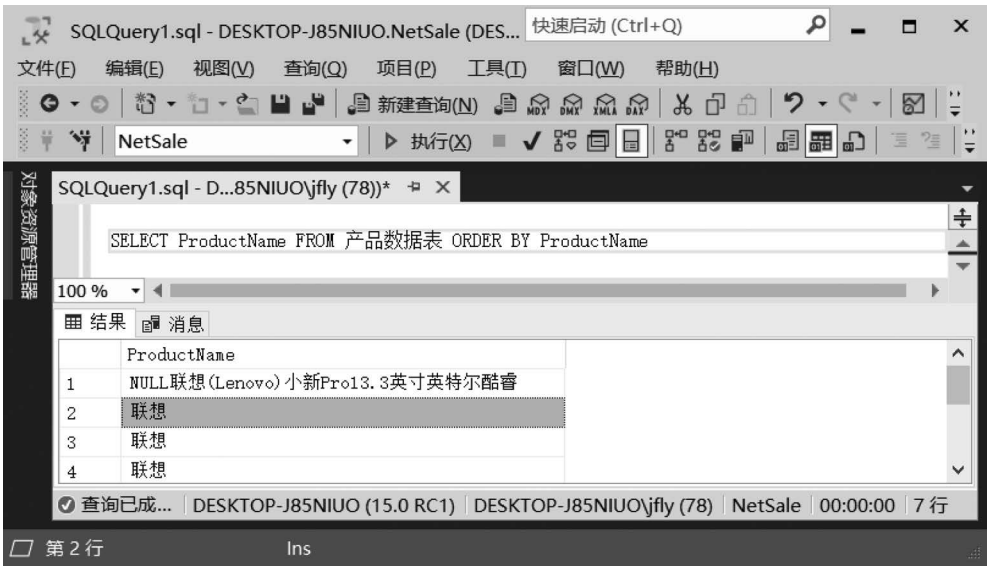

图 5-14　出现重复的查询结果集

以下代码使用了 DISTINCT，则查询结果如图 5-15 所示，重复行已经被去除。

```
SELECT DISTINCT ProductName FROM 产品数据表 ORDER BY ProductName
```

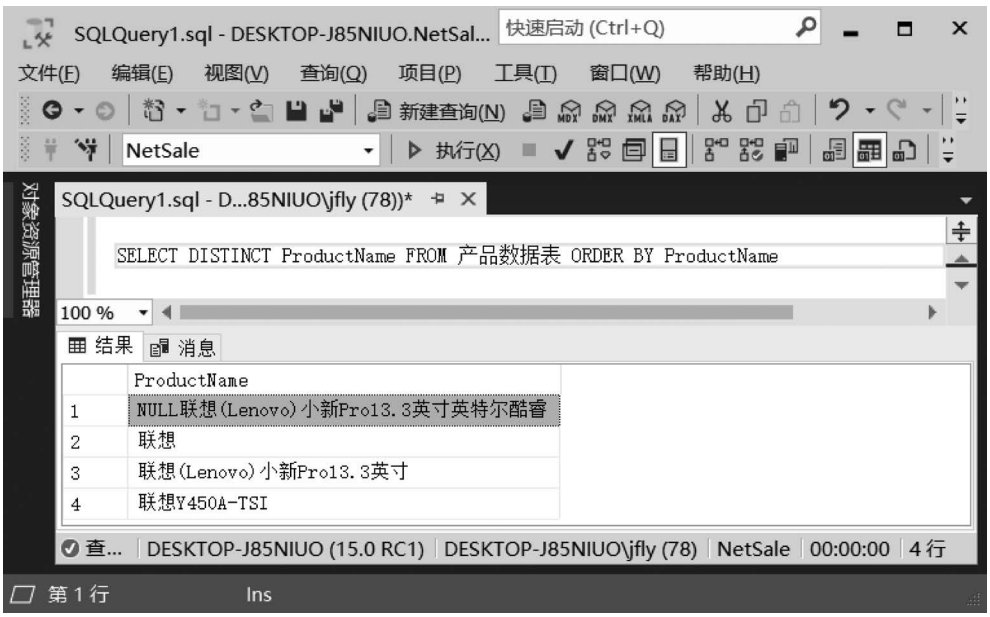

图 5-15　使用 DISTINCT 去除重复行数据

DISTINCT 是以列列表作为重复判断的依据。因此，如果将上述查询语句更改为 SELECT DISTINCT ProductName, ProductID FROM 产品数据表 ORDER BY ProductName，由于 ProductName，ProductID 两列数据组合不重复，因此不会去除任何数据行。

11. 关于 NULL

NULL 是一个特殊值,如果需要查看数据表中某一列值为 NULL 的数据,有两种方式可以构造 NULL 条件:“IS NULL”和“= NULL”,如 ProductName IS NULL 或者 ProductName=NULL。究竟取哪种表达方式,取决于系统的设置:SET ANSI_NULLS {ON|OFF}。

在 SQL Server 早期的版本中,允许使用 where ProductName=null 查询 ProductName 列中是否含有 NULL 值。但是,这不符合 ANSI 标准,因为 ANSI 标准将 NULL 看成一个完全未知的值,不能等于任何其他值。设置 SET ANSI_NULLS ON,将无法使用 where ProductName=NULL,此时可以使用 where ProductName is null。

例如下列代码中,第一行代码没有查询到数据,而第二行代码有数据产生,如图 5-16 所示。

```
SELECT * FROM 产品数据表 where ProductName = null
SELECT * FROM 产品数据表 where ProductName is null
```

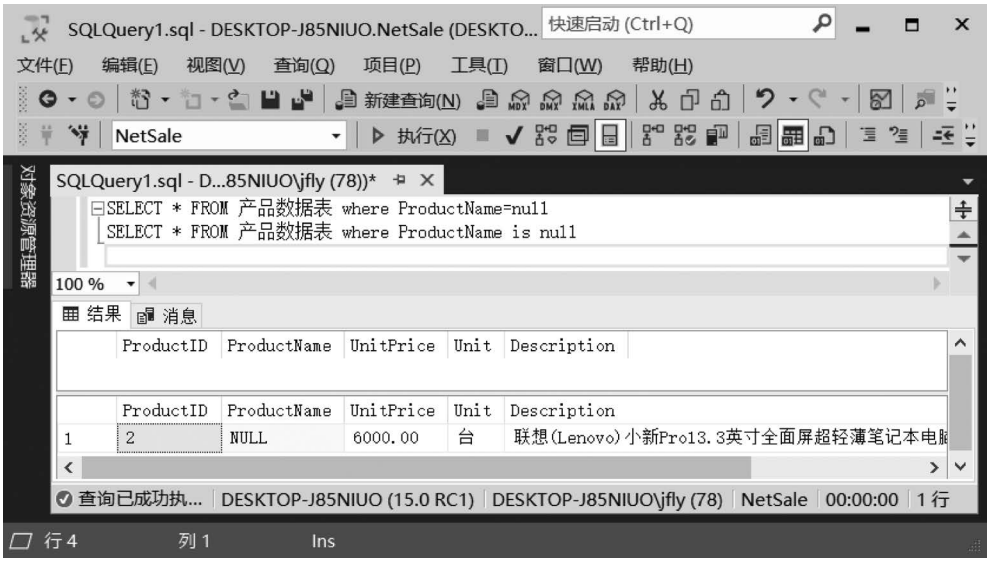

图 5-16 NULL 与 IS NULL 的区别

如果在查询代码前添加 SET ANSI_NULLS OFF,即关闭 ANSI_NULLS 选项的设置后,则两条语句执行的结果是相同的,如图 5-17 所示。

```
SET ANSI_NULLS OFF
SELECT * FROM 产品数据表 where ProductName is null
SELECT * FROM 产品数据表 where ProductName = null
```

注意:对于空值,可以使用 where ProductName=''。

12. 使用 GROUP BY 分组

在实际业务运行过程中,企业会经常需要对数据进行分组。如需要查看不同产品的销售量、不同业务员的业绩时,需要对查询数据进行分组,然后执行统计。在 T-SQL 中可以利用 GROUP BY 来实现对数据的分组。

例如,需要对“订单细节表”中的数据按照产品进行分组统计,计算不同产品的销售数量

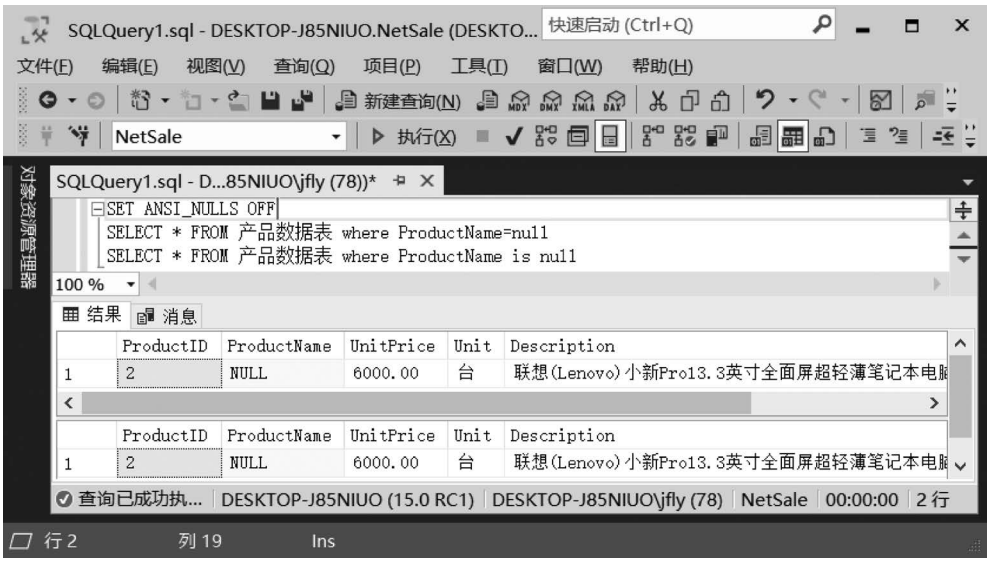

图 5-17　设置 SET ANSI_NULLS OFF 后 NULL 与 IS NULL 相同

和销售金额，可以用以下代码实现。查询执行的结果如图 5-18 所示。

```
SELECT productid, sum(sales) as '销售量',sum(subtotal) as '销售额'
from dbo.订单细节表 GROUP BY productid
```

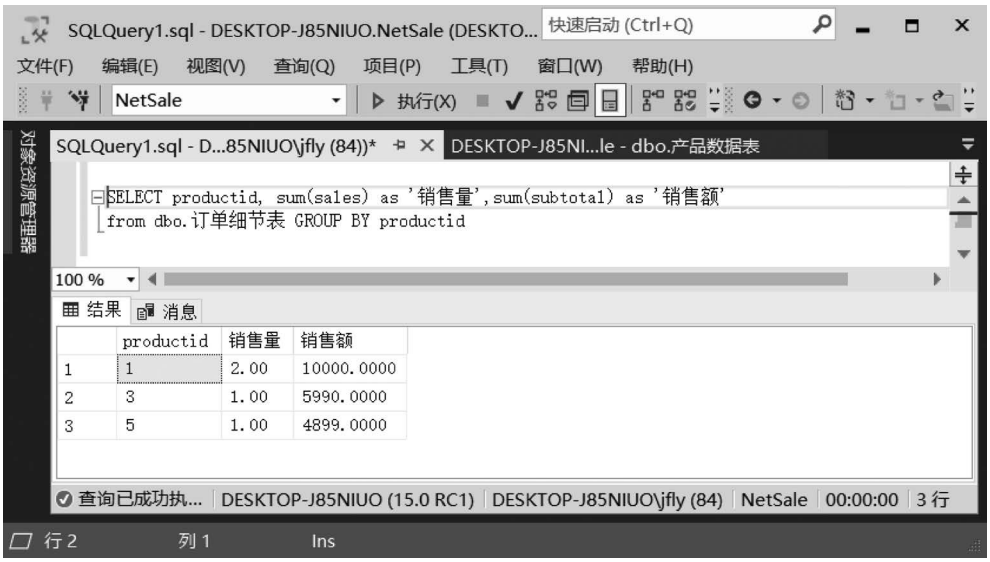

图 5-18　使用 GROUP BY 分组统计

如果需要从 GROUP BY 分组的数据中筛选符合特定条件的数据，需要使用 HAVING，而不能使用 WHERE。如要从上述分组汇总的数据集中筛选出 ProductID 列值为 3 的数据，可以使用以下代码：

```
SELECT productid, sum(sales) as '销售量',sum(subtotal) as '销售额' from dbo.订单细节表 GROUP
BY productid HAVING ProductID = 3
```

查询执行后的结果如图 5-19 所示。

图 5-19　使用 HAVING 筛选分组数据

但 WHERE 可以置于 GROUP BY 子句之前,即先按 WHERE 条件过滤要用于分组的数据,然后使用 HAVING 从分组后的数据集中筛选需要的数据,如以下代码可以实现上述要求。如果直接将以下代码中的 WHERE 替换为 HAVING,就会产生语法错误。

```
SELECT productid,sum(sales) as '销售量',sum(subtotal) as '销售额'
from dbo.订单细节表 Where ProductID > 1 GROUP BY productid HAVING ProductID = 3
```

另外,GROUP BY 还可以组合 TOP(n) PERCENT、ROLLUP、CUBE 等,实现更加复杂的查询分组统计的结果。例如以下代码,添加了 WITH ROLLUP,执行结果如图 5-20 所示,与图 5-18 相比,多一行汇总数据,汇总行汇总了总销售量和销售额。

```
SELECT productid, sum(sales) as '销售量',sum(subtotal) as '销售额'
from dbo.订单细节表 Where ProductID > 1 GROUP BY productid WITH ROLLUP
```

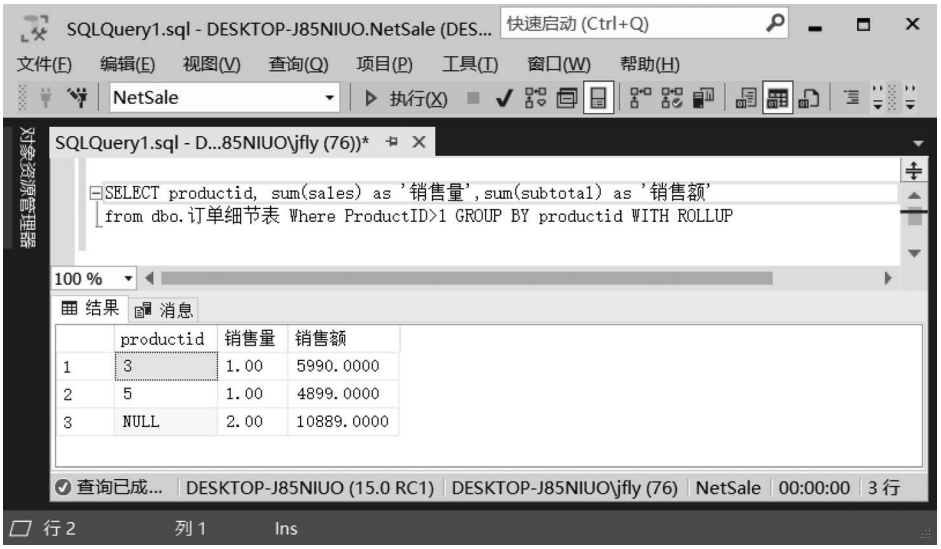

图 5-20　使用 ROLLUP 执行分组统计的结果

13. 使用子查询

上述实例在构造 where 条件表达式时，所采用的表达式值都是某一常量或者一个常量列表。事实上，在实际使用过程中，表达式值可以来自其他数据表中的值。如有两张数据表：客户信息表(Customer)和订单表(Order)。如果需要从订单表中查阅下单客户的信息，而客户信息保存在客户信息表中。要实现这一类应用，可以通过子查询实现。

子查询指嵌套在查询语句中的查询，子查询可以出现在查询语句的条件表达式中，即在 Where 子句中。

例如，以下代码可以查看下订单的客户信息。

```
select * from [dbo].[客户数据表] where CustomerID in (select [customerID] from [Orders].[订单表])
```

代码中，where 子句中“select[customerID]from[Orders].[订单表]”表示从订单表中获取 customerID，返回的是已下过单的客户编号列表，以此列表为范围构造查询的条件，可以进一步获取客户数据表中这些客户编号列表对应的客户的详细信息。代码执行的结果如图 5-21 所示。

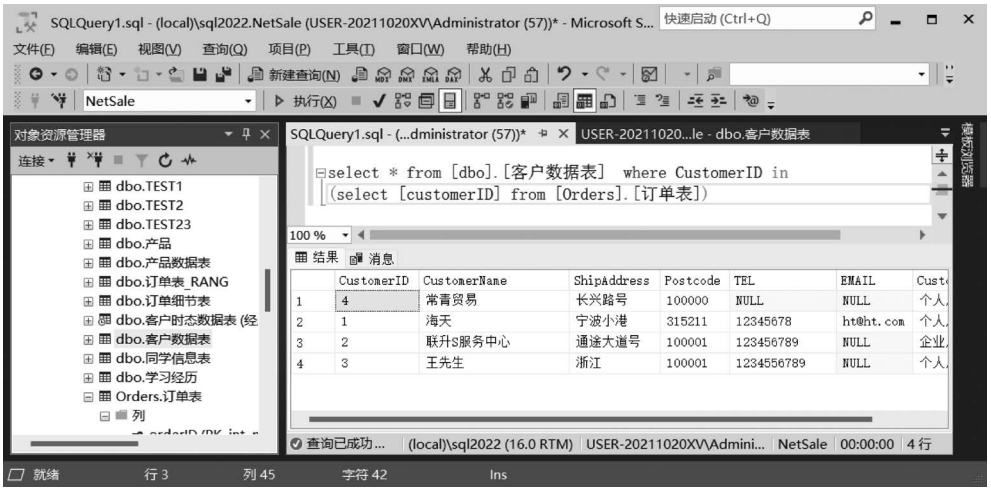

图 5-21　子查询返回值列表

子查询在查询语句中，可以嵌套使用。即在一个子查询中可以包含其他子查询，如下代码使用到了两个子查询，其中，最后一个子查询获取到了“客户名称”中包含“海天”的客户的客户编号值，并将这个返回的结果值作为前一个子查询的条件值。代码执行的结果如图 5-22 所示。

```
select * from [dbo].[订单细节表] where [orderid] = (select [orderid] from [Orders].[订单表]
where [customerID] = (select CustomerID from [dbo].[客户数据表] where [CustomerName] like '%
海天%'))
```

子查询也可以出现在查询的列列表中，从而构造复杂的查询列表。如以下代码，在图 5-22 所示代码的基础上，应用子查询“(select [ProductName] from [dbo].[产品数据表] where productid= d.productid) as 产品名称”从产品数据表中获取到了对应产品编号的产品名称，其中“d”是订单细节表的别名。代码执行的结果如图 5-23 所示。

```
select *,(select [ProductName] from [dbo].[产品数据表] where productid = d.productid) as 产
```

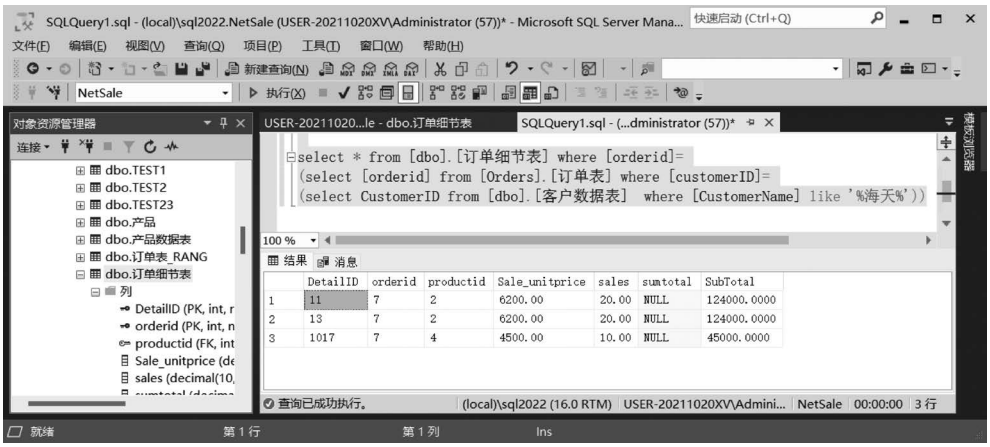

图 5-22　子查询嵌套

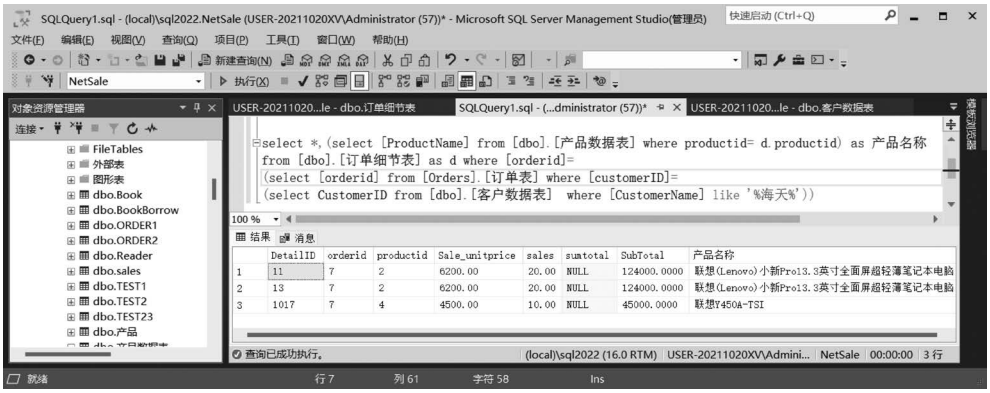

图 5-23　子查询获取查询列

```
品名称 from [dbo].[订单细节表] as d where [orderid] =
(select [orderid] from [Orders].[订单表] where [customerID] =
(select CustomerID from [dbo].[客户数据表] where [CustomerName] like '%海天%'))
```

子查询在使用时一般需要包含在括号内。

14. ALL、SOME 和 ANY

ALL、SOME 和 ANY 用于与指定的结果集进行比较。其中 ALL 表示结果集中的所有值都满足条件时,返回结果为 True。而 SOME 和 ANY 只要结果集中有满足条件的值,其返回的结果就是 True,并不需要所有的值都满足条件;如果所有的值都不满足条件,则返回的结果为 False。

ALL、SOME 和 ANY 使用的基本语法如下:

```
scalar_expression { = | <> | != | > | >= | !> | < | <= | !< } { SOME | ANY } ( subquery )
```

其中,subquery 为子查询,子查询返回一个结果集,scalar_expression 为表达式,表达式通过各种运算符与子查询结果中的值进行逐一比较。

例如,以下代码为从订单表中查看那些未被处理(Status=0),并且送货时间(shiptime)早于系统当前时间的订单。由于采用 ANY(或 SOME),只要子查询"select shiptime from [Orders].[订单表]"返回的送货时间列表中有一项以上的值早于系统时间,则返回结果值

为 True。代码执行的结果如图 5-24 所示。

```
select * from [Orders].[订单表] where GETDATE()>=ANY(select shiptime from [Orders].[订单表]) and Status=0
```

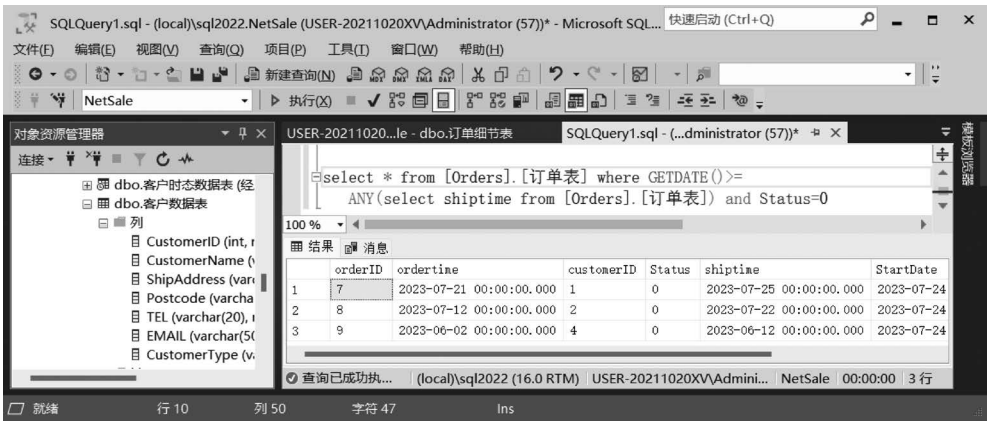

图 5-24　使用 ANY 或 SOME

而当采用 ALL 构造查询条件时，由于订单数据表中的并不是所有的订单的送货时间都早于系统当前时间，因此查询条件的返回结果为 False。代码执行返回结果为空，如图 5-25 所示。

```
select * from [Orders].[订单表] where GETDATE()>=
  ALL(select shiptime from [Orders].[订单表]) and Status=0
```

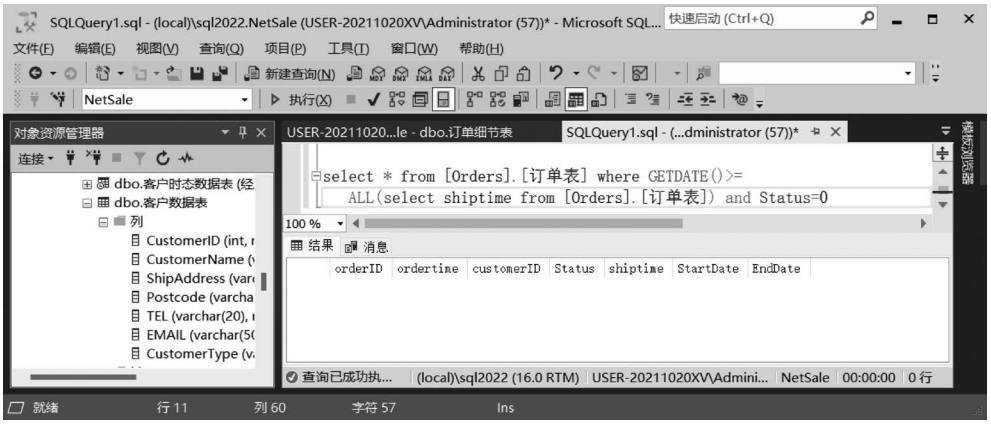

图 5-25　使用 ALL

5.3.2　多表联接数据查询

在 SQL Server 中根据关系数据库规范化的要求，有时为避免数据的重复冗余，需要将数据表进行水平分隔，以满足规范范式的要求。这样，在进行数据查询时，为了解查询数据的全貌，经常需要将多张表的数据联接到一起来显示。前述介绍的子查询可以实现上述要求，但很多时候子查询并非是最佳选择。

请先来看一下本书的案例数据库 NetSale 中数据表的情况，如图 5-26 所示。

图 5-26　NetSale 数据库中的数据表

从图 5-26 中可见,如果要从“订单表”中查看下单客户的详细信息,必须联接“客户数据表”,要从“订单细节表”中查看订购的产品的详细信息,必须联接“产品数据表”。如果还想知道下单的时间与客户的详细信息,还必须联接“订单表”且需要从“订单表”再去联接“客户数据表”。虽然在数据表的设计过程中,表间已建立了多种外键关系,但这种外键关系只对数据管理起约束作用,对查询不起作用;用户如果要从多张表中获取数据,必须使用查询联接语句在表间建立联接。

T-SQL 中,用于联接多表的联接查询语句可以划分为内联接(INNER JOIN)、外联接(OUTER JOIN)和交叉联接(CROSS JOIN)。

1. 内联接

内联接采用 INNER JOIN 联接两张数据表,通过 ON 关键词构造两张表之间的关系。内联接可以把两表中符合联接条件的数据抽取出来生成第三张表。因此,内联接生成的表的数据通常会比两源表中任一表的数据量少,或者等于数据量较小的那张表的数据量。

例如,要从表“订单细节表”中查询“购买的产品名称、数量以及金额”,则需要将“订单细节表”与“产品数据表”联接,以下代码可以实现上述要求。

```
SELECT 产品数据表.ProductName, 订单细节表.sales as '销售量',订单细节表.subtotal as '销售额'
from dbo.订单细节表 inner join 产品数据表 on 订单细节表.ProductID = 产品数据表.ProductID
```

如上代码中,“产品数据表.ProductName, 订单细节表.sales as '销售量',订单细节表.subtotal as '销售额'”为查询的列列表,包含从“产品数据表”中提取的 ProductName 列和“订单细节表”表中的 sales 和 subtotal 列;“INNER JOIN 产品数据表”表示从“订单细节表”联接“产品数据表”;“ON 订单细节表.ProductID=产品数据表.ProductID”是联接的条件,即要求“订单细节表”中 ProductID 列的值等于“产品数据表”中 ProductID 列的值。查询结果如图 5-27 所示。

为简化代码编写,在联接语句中可以使用别名代替表名,如以下代码使用 P 代表“产品数据表”,使用 D 代表“订单细节表”,查询执行的结果同图 5-27。

```
SELECT P.ProductName, D.sales as '销售量',D.subtotal as '销售额' from dbo.订单细节表 AS D inner
join 产品数据表 AS P on D.ProductID = P.ProductID
```

如果需要联接两张以上数据表,则可以使用 INNER JOIN 继续联接,例如以下代码联接了“订单细节表”“订单表”“客户数据表”“产品数据表”,查询结果如图 5-28 所示。

```
SELECT P.ProductName,D.sales as '销售量',D.subtotal as '销售额', C.CustomerName as '客户名称',
C.ShipAddress as '送货地址',O.ordertime as '订购时间' from dbo.订单细节表 AS D
```

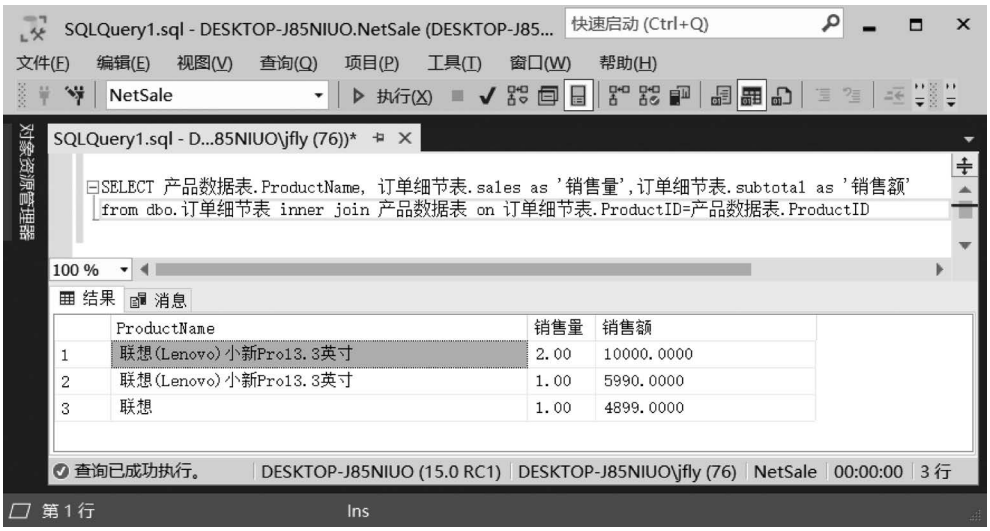

图 5-27　联接查询

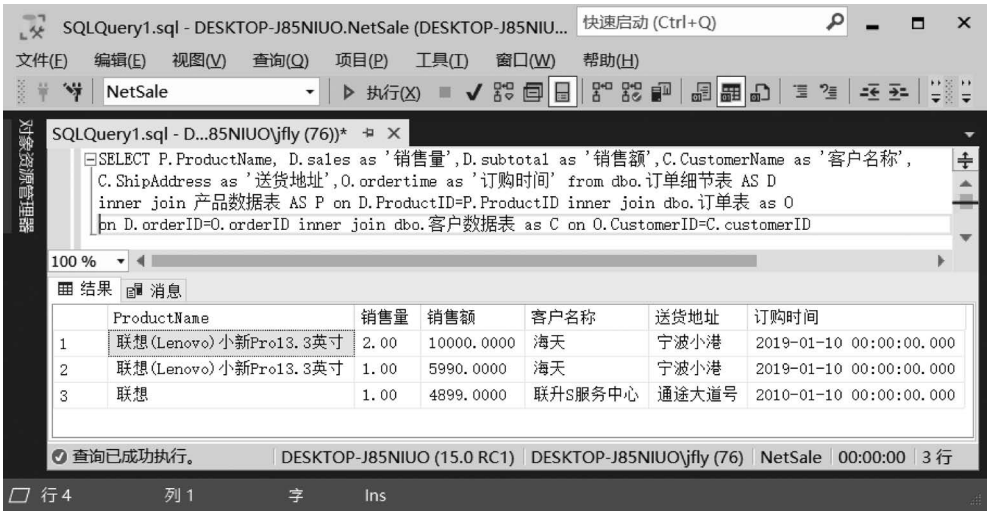

图 5-28　联接多张表的查询

```
inner join 产品数据表 AS P on D.ProductID = P.ProductID inner join dbo.订单表 as O on D.orderID
= O.orderID inner join dbo.客户数据表 as C on O.CustomerID = C.customerID
```

联接数据表并不要求用作联接条件的两列的名称必须相同，如 D.orderID＝O.orderID 中两端的 orderID 并不要求名称相同，但是数据类型最好相同或者能够转换为相同的类型。系统默认联接是内联接，因此 JOIN 等同于 INNER JOIN，即当 INNER 省略时，执行的联接方式是内联接。

2. 外联接

外联接采用 OUTER JOIN 联接两张数据表，同样通过关键词 ON 构造联接条件。外联接根据联接方向不同，可以划分为左联接 LEFT OUTER JOIN、右联接 RIGHT OUTER JOIN 和完全外联接 FULL OUTER JOIN。与内联接只显示满足连接条件的数据不同，外联接可以分为主表和从表，主表中的数据会被全部显示出来，而从表中只显示满足联接条件

的数据,这样外联接结果一般会比两表中数据量小的表中的数据多。

如果是左联接 LEFT JOIN,则位于 FROM 子句左端的表为主表,另一端的表为从表;右联接则反之。

例如,以下代码构造了一个左连接,“订单细节表”为主表,“产品数据表”为从表,联接条件是 D. ProductID=P. ProductID。由于“订单细节表”与“产品数据表”建立了外键关联,即在输入“订单细节表”的 ProductID 列值时,要求必须已经在“产品数据表”中存在。因此,查询结果与内联接相同,如图 5-29 所示。

```
SELECT P.ProductName, D.sales as '销售量',D.subtotal as '销售额' from dbo.订单细节表 AS D LEFT
join 产品数据表 AS P on D.ProductID = P.ProductID
```

图 5-29　左联接查询结果

同样,可以构造上述两张表的右联接,查询结果如图 5-30 所示。与图 5-29 相比较可见,在第二个联接中,主表“产品数据表”原有 7 行数据全部显示出来了,另外,产品“联想(Lenovo)小新 Pro13.3 英寸”在“订单细节表”中有 2 条对应数据,所以出现了 2 次。因此,查询结果共计产生 8 行数据。与从表“订单细节表”有对应关系的 3 行数据,在“销售量”和“销售额”中填的是“订单细节表”中获取的数据,而另 5 行在“订单细节表”未找到对应数据,则填充为 NULL。

```
SELECT P.ProductName, D.sales as '销售量',D.subtotal as '销售额' from dbo.订单细节表 AS D RIGHT
join 产品数据表 AS P on D.ProductID = P.ProductID
```

FULL OUTER JOIN 与上述两种外联接不同,FULL OUTER JOIN 会从两联接表中获取数据,如果有对应关系的会填上对应数据,没有对应关系的以 NULL 值填充。因此,FULL OUTER JOIN 联接生成的数据量会比两联接表的数据量大,一般会等于左联接与右联接两种方式产生的数据量大的数量。

例如,以下代码构造了“订单细节表”与“产品数据表”之间的 FULL OUTER JOIN,查询的结果如图 5-31 所示。数据量与数据量大的右联接查询的数量相同。

```
SELECT P.ProductName, D.sales as '销售量',D.subtotal as '销售额' from dbo.订单细节表 AS D FULL
join 产品数据表 AS P on D.ProductID = P.ProductID
```

图 5-30　右联接查询结果

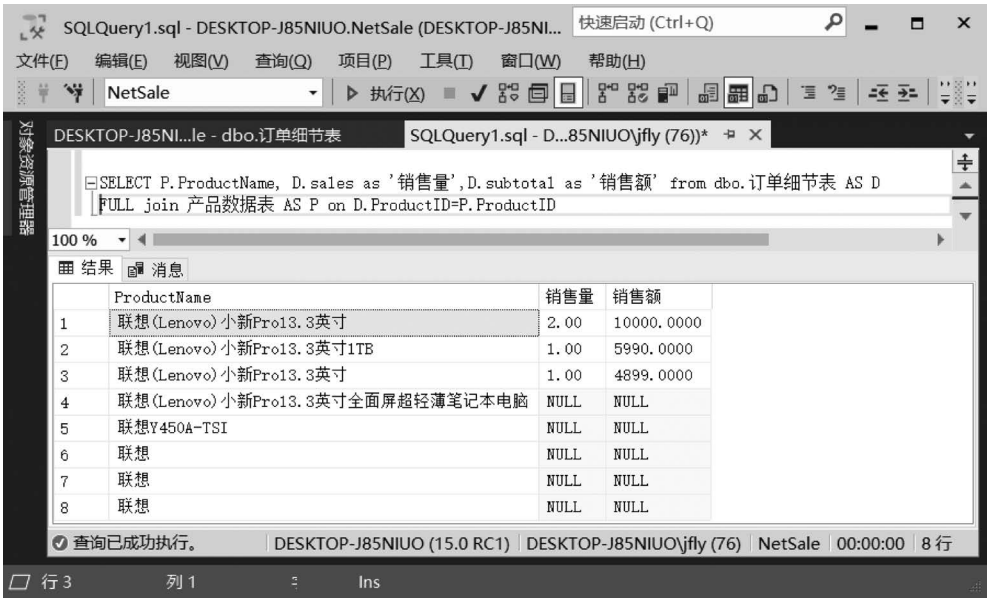

图 5-31　FULL OUTER JOIN 联接查询结果

OUTER JOIN 中的 OUTER 可以省略不写，不会影响查询结果。

3. CROSS JOIN 联接

CROSS JOIN 联接也被称为交叉联接。与内联接、外联接都不同，交叉联接会罗列所有可能的数据，最终形成的结果是两联接表的笛卡儿乘积，即第一张表的数据行数与第二张表的数据行数的乘积。如果联接的两表分别有 10 行和 20 行数据，则 CROSS JOIN 查询结果的数据行数为 200 行。

例如，以下代码构造了一个 CROSS JOIN 联接查询的实例，查询结果如图 5-32 所示，

生成的数量为“产品数据表”7 行和“订单细节表”3 行的乘积,共计 21 行数据。

```
SELECT P.ProductName, D.sales as '销售量',D.subtotal as '销售额' from dbo.订单细节表 AS D CROSS
join 产品数据表 AS P
```

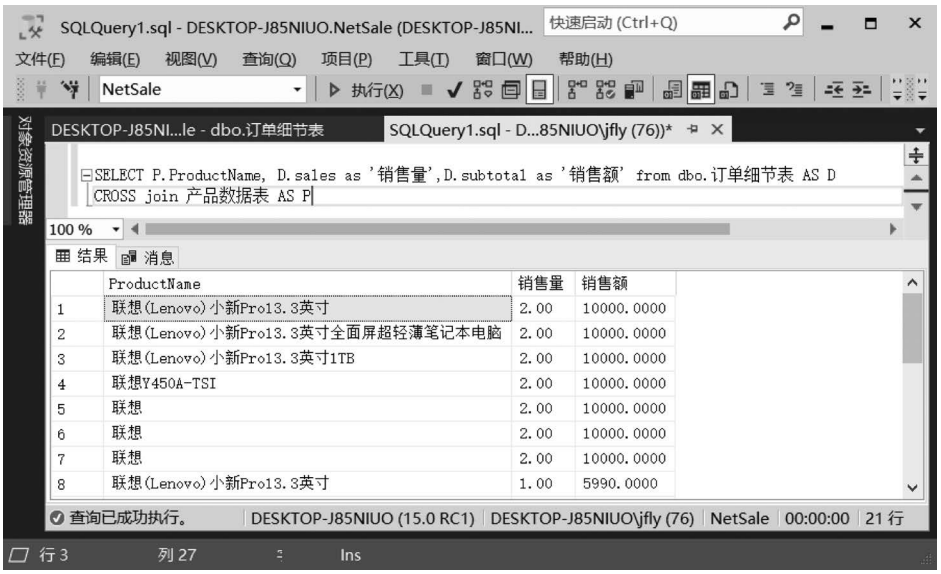

图 5-32　CROSS JOIN 查询结果

4. 多表联接查询中使用 WHERE 子句

与单表查询一样,在多表联接查询中,同样可以使用 WHERE 来构造数据查询的条件,如以下代码在内联接 4 张表的基础上添加了数据筛选的条件,查询结果如图 5-33 所示。由于参与联接的多张表中可能存在列名相同的情况,因此在构造条件时,应在“列名”前添加表名或者表的别名,如以下代码中 C.CustomerID=3 表示取 Customer 表的 CustomerID 列的值作为筛选条件。

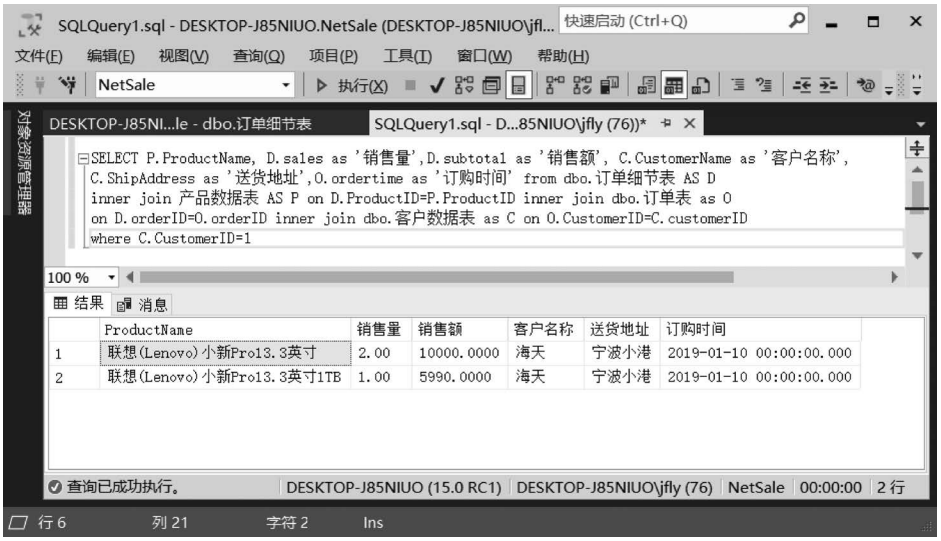

图 5-33　多表联接查询中使用 WHERE 条件

```
SELECT P.ProductName, D.sales as '销售量',D.subtotal as '销售额', C.CustomerName as '客户名称',C.
ShipAddress as '送货地址',O.ordertime as '订购时间' from dbo.订单细节表 AS D inner join 产品数
据表 AS P on D.ProductID = P.ProductID inner join dbo.订单表 as O on D.orderID = O.orderID inner
join dbo.客户数据表 as C on O.CustomerID = C.customerID where C.CustomerID = 3
```

提示：在多表联接中，同样可以使用单表联接中的各种关键词，如 TOP(n) PERCENT、DISTINCT、ORDER BY、GROUP BY 等。

5.3.3 使用 SELECT INTO 语句

SELECT INTO 可以把由 SELECT 语句中选定的数据保存到一张新数据表中。事实上 SELECT INTO 语句包含三个子过程：第一个子过程为 SELECT INTO 语句根据 SELECT 选择的列列表及各列的数据类型生成一段创建数据表的代码；第二个子过程为 SELECT INTO 语句使用第一过程生成的代码，新建这张数据表；第三个子过程则是将从源表中选定的数据导入新建的数据表中。

例如，以下代码是图 5-32 执行的 CROSS JOIN 语句，会生成 21 行数据，以下代码稍作修改，添加了 INTO Sales，改造成为一条 SELECT INTO 语句，执行之后会在当前数据库 NetSale 中创建数据表 Sales，并且会在 Sales 中添加 21 行数据。

```
SELECT P.ProductName, D.sales as '销售量',D.subtotal as '销售额' INTO sales from dbo.订单细节
表 AS D CROSS join 产品数据表 AS P
```

新生成的 Sales 表中的数据如图 5-34 所示。

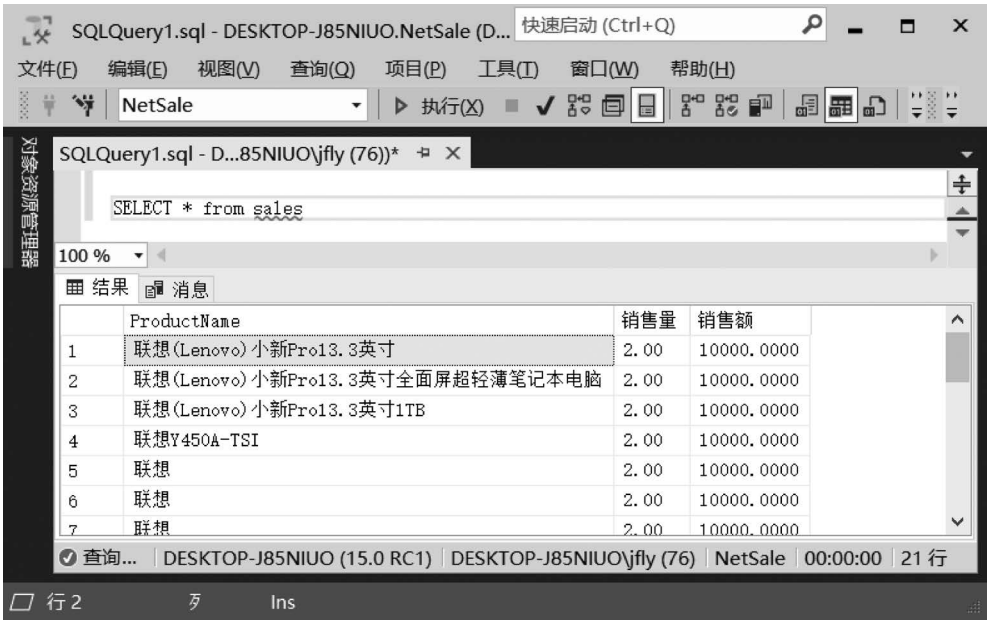

图 5-34 SELECT INTO 生成的新数据表

通过 SELECT INTO 生成的数据表可以是普通用户表，也可以是临时表。如果在 SELECT INTO 语句中添加 WHERE 条件，则生成的数据是满足特定条件的数据。这使 SELECT INTO 非常适合在一些复杂的数据库应用程序中，通过 SELECT INTO 生成临时数据以备下一步使用的场合。

以下代码将 SELECT INTO 生成的数据保存到本地临时数据表 # sales 中。本地临时数据表在使用完毕后,会自动删除,不会给系统造成负担。

```
SELECT P.ProductName, D.sales as '销售量',D.subtotal as '销售额' into #sales from dbo.订单细节表 AS D CROSS join 产品数据表 AS P
```


5.3.4 组合查询

T-SQL 提供了 UNION、INTERSECT 和 EXCEPT 运算符,可以实现组合查询。组合查询指在某些场合,查询结果需要通过多条 SELECT 语句从一张或多张表中来获取,查询最终结果是多条 SELECT 语句查询结果的汇总数据集。组合查询可以看作数据的垂直联接,而前面所介绍的多表联接查询可以看作数据的水平联接。

T-SQL 提供的三个查询组合运算符可以分别满足三种不同的需要,但是组合查询要求每条 SELECT 语句生成的数据集中列的个数、列的数据类型和顺序必须相同。

1. UNION

UNION 是一种并集运算,可以将两个以上的查询结果合并成一个结果,并在后续的结果集中去除前面结果集中已有的数据行。例如以下代码采用 UNION 组合了两条 SELECT 语句,这两条 SELECT 语句分别从“产品数据表”和“产品”表中查询 ProductName 数据。

```
SELECT ProductName from 产品数据表
UNION
SELECT ProductName from 产品
```

该段代码执行的查询结果如图 5-35 所示,共计生成 11 条数据。图 5-36 是“产品数据表”和“产品”中的原始数据情况,“产品数据表”原有 7 条数据,但 3 条重复,被去除 2 条,因此第一个查询从“产品数据表”中获取了 5 条数据;而第二个查询原本应该出来 6 条数据,由于有 2 条与第一个查询重复,被去除 2 条,因此共计生成了 9 条数据。

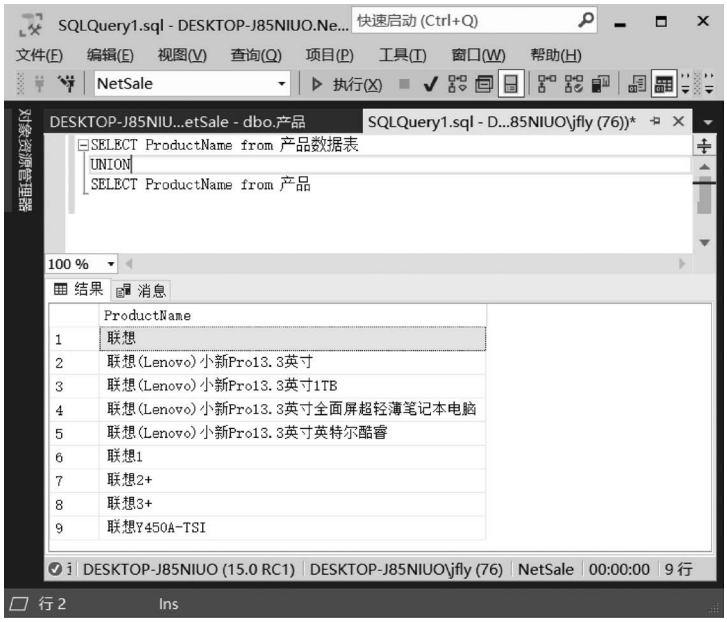

图 5-35 UNION 给合查询结果

	来自产品数据表
1	联想(Lenovo)小新Pro13.3英寸
2	联想(Lenovo)小新Pro13.3英寸全...
3	联想(Lenovo)小新Pro13.3英寸1TB
4	联想Y450A-TSI
5	联想
6	联想
7	联想

	来自产品
1	联想(Lenovo)小新Pro13.3英寸
2	联想(Lenovo)小新Pro13.3英寸英...
3	联想(Lenovo)小新Pro13.3英寸
4	联想1
5	联想2+
6	联想3+

图 5-36 “产品数据表”和“产品”表的原始数据

如果需要保留所有重复数据，可以使用 UNION ALL。如以下代码执行后，生成的数据集是两条查询语句产生的数据集的总和，共计 13 条，执行结果如图 5-37 所示。

```
SELECT ProductName from 产品数据表
UNION ALL
SELECT ProductName from 产品
```

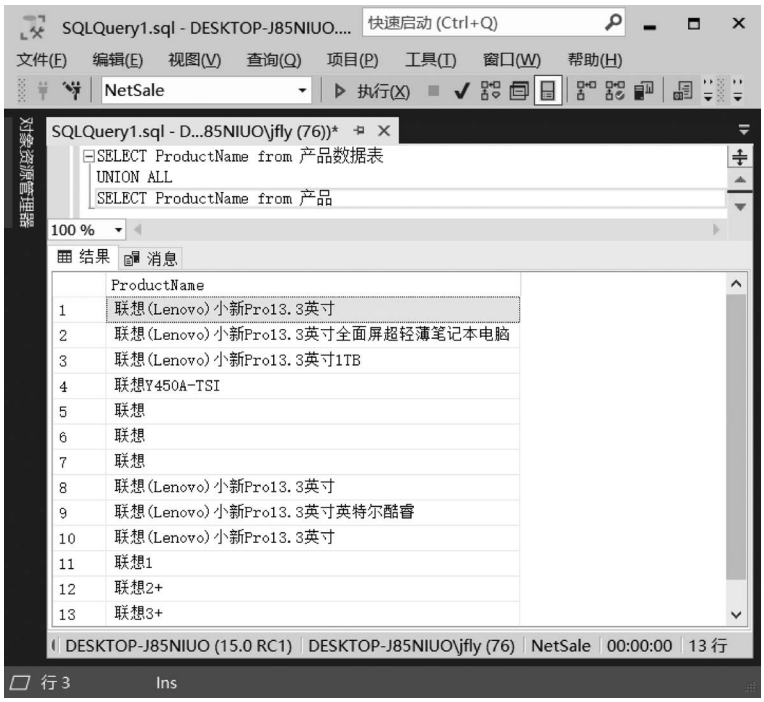

图 5-37 UNION ALL 查询结果

2. INTERSECT

INTERSECT 可以返回多条查询语句中都包含的非重复数据。例如以下代码，从“产品数据表”和“产品”表中执行 INTERSECT 组合查询，结果如图 5-38 所示。由于两张表都有且不重复的数据只有 1 条，因此最终结果只有 1 条。

```
SELECT ProductName from 产品数据表
INTERSECT
SELECT ProductName from 产品
```

注意：INTERSECT 不支持 ALL 操作。

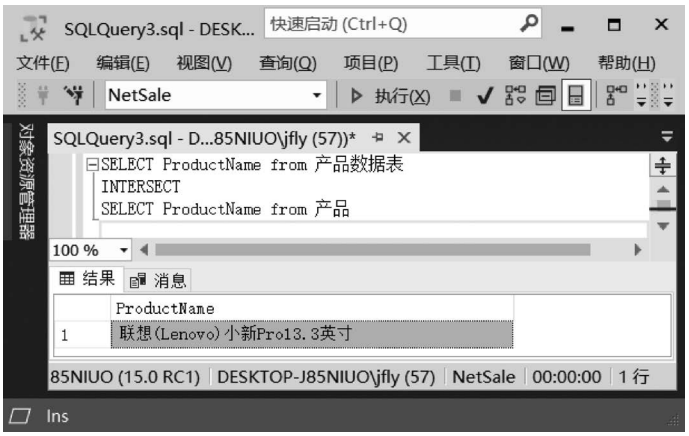

图 5-38 INTERSECT 组合查询结果

3. EXCEPT

EXCEPT 可以比较左右两个查询结果集的差异,并从左侧的查询结果集中返回在右侧找不到的数据,即从左侧的结果集中减去与右侧结果集相同的数据后得到的结果。

例如,以下代码使用 EXCEPT 从“产品数据表”和“产品”表中获取组合查询结果。执行结果如图 5-39 所示。因为“产品数据表”中原有 5 条非重复的数据,经去除“产品”表中的 2 条与之相同的数据后,最后得到 4 条非重复的数据。

```
SELECT ProductName from 产品数据表
EXCEPT
SELECT ProductName from 产品
```

图 5-39 EXCEPT 组合查询结果

5.3.5 使用 FETCH 与 OFFSET 分页

当查询结果数据量较大时,采用分页方式显示数据是一种必然的选择。在 T-SQL 未提供直接的分页语句之前,实现分页功能是一个相对较为复杂的过程。自 SQL Server 2012

起，T-SQL 提供了 FETCH 和 OFFSET 参数，可以非常简捷和高效地实现大数据集的分页显示。

FETCH 和 OFFSET 可以作为 Order By 子句的参数，其语法如下：

```
OFFSET { integer_constant | offset_row_count_expression } { ROW | ROWS }
    [
      FETCH { FIRST | NEXT } {integer_constant | fetch_row_count_expression } { ROW | ROWS
} ONLY
    ]
```

其中，OFFSET 参数用于指定从结果集中获取记录的起始行，其后的参数可以是整数，也可以是表达式，integer_constant 代表整数值，offset_row_count_expression 代表表达式。FETCH 用于指定在处理 OFFSET 子句后返回的行数，该值可以是大于或等于 1 的整数常量或表达式，同样 integer_constant 代表整数值，offset_row_count_expression 代表表达式。参数中 ROW 和 ROWS 具有相同含义，FIRST 和 NEXT 也具有相同含义。

以下代码演示了 FETCH 和 OFFSET 参数的使用方法。

在 SQL Server Management Studio 中，单击工具栏中的“新建查询”按钮，在查询编辑器窗口执行以下代码，执行结果如图 5-40 所示。第一条 SELECT 语句显示了产品数据表中的所有数据，第二条 SELECT 从产品数据表返回从第一行开始的三条记录。

```
Use NetSale
SELECT ProductID,ProductName,UnitPrice,Unit,Instocks FROM 产品数据表 order by ProductID
GO
SELECT ProductID, ProductName, UnitPrice, Unit, Instocks FROM 产品数据表 order by ProductID
OFFSET 0 ROW FETCH NEXT 3 ROW ONLY
```

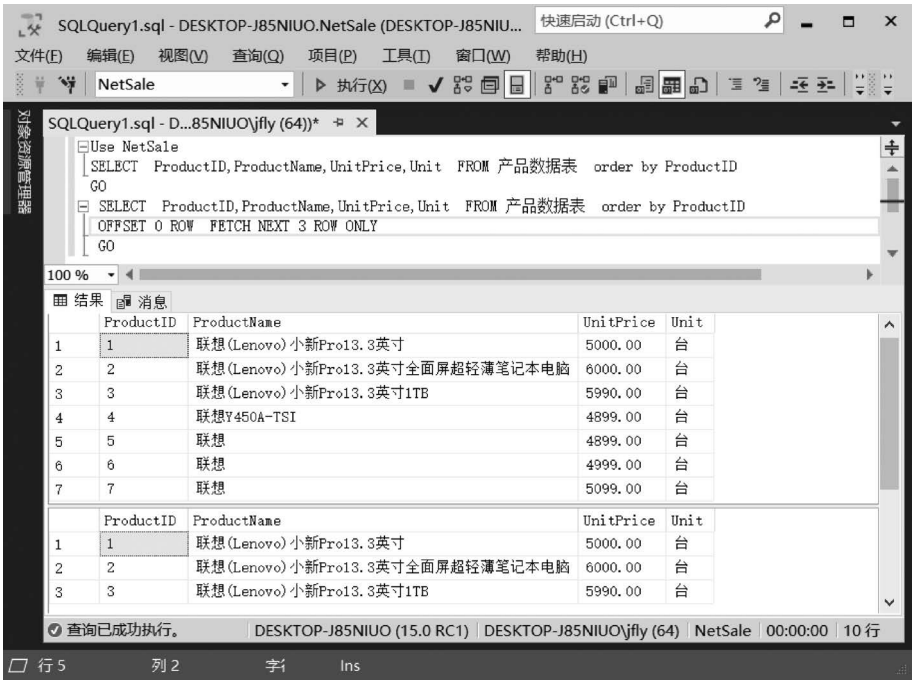

图 5-40　使用确定整数值的 FETCH 和 OFFSET 分页

在上述代码中,FETCH 和 OFFSET 参数都使用了确定的整数值。在实际使用过程中,FETCH 和 OFFSET 参数可以是变量值。以下代码演示了变量的使用方法,执行结果如图 5-41 所示。

```
Use NetSale
DECLARE @OffsetRows tinyint = 1,@FetchRows tinyint  = 5
SELECT ProductID, ProductName, UnitPrice, Unit, Instocks FROM 产品数据表 Order by ProductID
OFFSET @OffsetRows ROWS FETCH NEXT @FetchRows ROWS ONLY
```

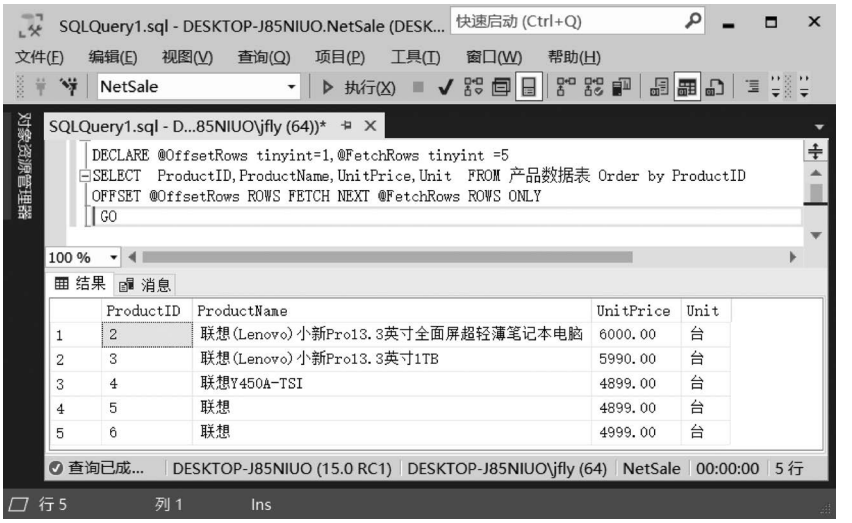

图 5-41　使用变量实现分页

上述代码中定义了两 tinyint 型变量@OffsetRows 和@FetchRows,分别用于保存分页起始的记录行和每页返回的行数。根据这两个变量的取值变化可以实现灵活的分页,FETCH 和 OFFSET 参数的分页可以结合存储过程,接收客户端程序传递的参数实现大记录集的分页。

5.4　T-SQL 附加语言元素

T-SQL 作为数据库操作语言,除了上述语句之外,还包含很多附加的语言元素,如标识符、保留关键字、常量、变量、运算符、流程控制语句、函数、注释等。本节介绍这些内容。

5.4.1　标识符

在 T-SQL 中,标识符用于命名各种对象,如数据库名称、数据表名称、存储过程等,以及变量、函数等名称。与其他语言类似,T-SQL 中的标识符也必须符合标识符命名的规则,这些规则包括以下几项。

(1) 首字符可以是 Unicode 字符集中的一个字母,包含英文字母 A～Z、a～z,以及其他 Unicode 字符,如汉字等。

(2) 首字符还可以是下画线(_)、位置符号(@)、数字符号(#)。但以这些符号作为首字符时会有不同的含义,如位置符号(@)开头表示定义的标识为局部变量,以两个@开头表

示系统内置的某些函数；以一个＃开头表示局部临时表或过程，以两个＃开头表示全局临时对象。

(3) 标识符长度限制在 128 个字符以内，除了局部临时表的名称之外，其他标识长度限制在 116 个字符以内。

(4) 后续字符可以是 Unicode 字母、数字、@、$、_、＃等符号。

(5) 标识符不能是 SQL Server 的保留关键字。

(6) 标识符不能嵌入空格或除上述字符以外的其他特殊字符。

例如，Products129、_129、@1G 等都是合法的标识符，而 Customer Name、1_name 等则不是合法标识符。

在有些场合中可以使用双引号("")和中括号([])来引用标识符，被称为分隔标识符，如以下语句，因为 USER 是 SQL Server 保留关键字，会出现语法错误。

```
SELECT * from USER
```

但是将语句改写为以下代码，则不会出现语法错误。

```
SELECT * from [user] 或者 SELECT * from "user"
```

但是双引号引用的标识符只有在 QUOTED_IDENTIFIER 选项设为 ON 时才会有效。默认可以使用中括号([])作为分隔标识符。

5.4.2 保留关键字

保留关键字是 SQL Server 预留的用于定义、操作和访问数据库的关键词，是 T-SQL 的组成部分。这些关键词不能直接用于命名标识符，虽然允许通过分隔标识符，如[]或""来引用这些关键词来作为标识符，但为了避免引起不必要的误解，建议不要使用关键字作为标识符。

T-SQL 中的保留关键字包括 ADD、ALL、ALTER、AND、ANY、AS、ASC、AUTHORIZATION、BACKUP、BEGIN、BETWEEN、BREAK、BROWSE、BULK、BY 等现在用的 180 多个关键字，还包括 ABSOLUTE、ACTION、ADMIN、AFTER、AGGREGATE、BEFORE、FREE 等将来可能使用的关键字 190 多个。

5.4.3 常量与变量

常量是表示特定数据值的符号，如'SQL Server 2022'表示一个字符串常量，1 表示一个整型常量，1.0 表示浮点数常量。在 SQL Server 2022 中，要求字符串常量需要使用一对单引号(')，数值型常量直接使用数值，日期时间型常量需要使用一对单引号(')。

变量指在 T-SQL 代码执行过程中，其值可变，需要赋值的对象。在 SQL Server 2022 中，变量可用于批处理和脚本中，例如用来计算循环的次数，也可以保存数据值以供控制流语句测试，还可以用于保存存储过程或函数返回的数据值等。

在 T-SQL 中，定义变量的语句为 DECLARE，所定义变量的首字符必须是@，且必须指定数据类型和长度。例如：

```
DECLARE @SalesCount int
```

DECLARE 语句可以一次指定多个变量，如：

```
DECLARE @SalesCount int,@saler_name varchar(20)
```

默认定义的变量其值为 NULL,如果需要对变量赋值,可以使用 SET 语句,如:

```
DECLARE @saler_name varchar(20)
SET @saler_name = '王强'
```

变量定义后,可以在存储过程、函数或者其他过程中使用。如以下代码定义了变量@id,并将之用于 SELECT 语句,执行结果如图 5-42 所示。

```
DECLARE @ProductID int
Set @ProductID = 1
select * from 产品数据表 where ProductID>@ProductID
```

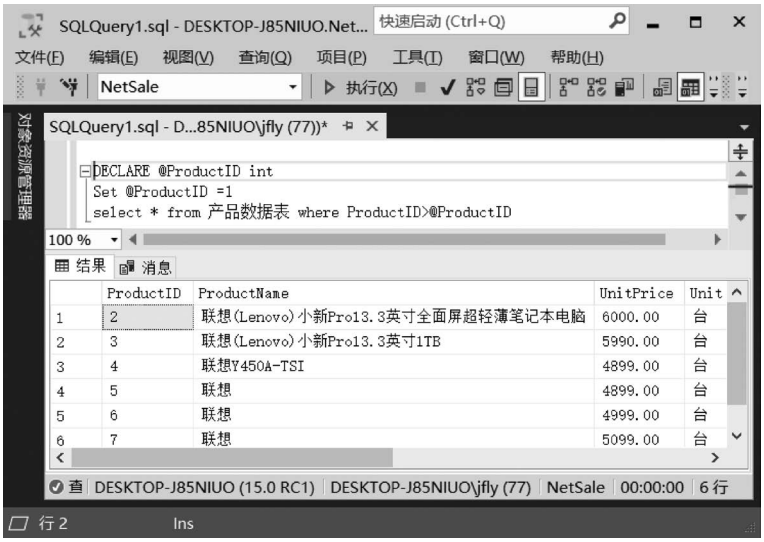

图 5-42　使用变量

5.4.4　运算符

运算符是程序设计语言中最重要的元素之一。T-SQL 提供了算术运算符、比较运算符、逻辑运算符、赋值运算符、字符串连接运算符、位运算符和一元运算符等多种运算符,使 T-SQL 具备完成各种运算的能力。

T-SQL 运算符的含义见表 5-2。

表 5-2　T-SQL 的运算符

类　别	运　算　符	含　义
算术运算符	+(加)	加
	-(减)	减
	*(乘)	乘
	/(除)	除
	%(取模)	返回一个除法运算的整数余数。例如,13 % 5=3
赋值运算符	=	给变量赋值,或其他相关的赋值运算

续表

类　别	运　算　符	含　义
逻辑运算符	ALL	如果一组的比较都为 True,那么就为 True
	AND	如果两个布尔表达式都为 True,那么就为 True
	ANY	如果一组的比较中任何一个为 True,那么就为 True
	BETWEEN	如果操作数在某个范围之内,那么就为 True
	EXISTS	如果子查询包含一些行,那么就为 True
	IN	如果操作数等于表达式列表中的一个,那么就为 True
	LIKE	如果操作数与一种模式相匹配,那么就为 True
	NOT	对任何其他布尔运算符的值取反
	OR	如果两个布尔表达式中的一个为 True,那么就为 True
	SOME	如果在一组比较中,有些为 True,那么就为 True
字符串连接运算符	+	连接两个字符串组成一个新的字符串,如'abc'+'123',值为'abc123'
位运算符	&	位与
	|	位或
	^	位异或
	+	数值为正
	−	数值为负
	~	返回数字的非
比较运算符	=	等于
	>	大于
	<	小于
	>=	大于或等于
	<=	小于或等于
比较运算符	<>	不等于
	!=	不等于
	!<	不小于
	!>	不大于

当一个表达式中包含上述运算符中的多个时,T-SQL 会根据不同运算符的优先级来执行运算。这些运算符的优先级顺序如下。

- 正、负、非(+、−、~)
- *、/、%
- +(加法或字符串连接)、−(减法)、&
- =(比较)、>、<、>=、<=、<>、!=、!>、!<
- ^、|
- NOT
- AND
- ALL、ANY、BETWEEN、IN、LIKE、OR、SOME
- =(赋值)

例如,以下代码中使用了 SOME 运算符,由于“产品数据表”中的 ProductID 列有小于 3 的值,所以下式中的 IF 条件成立,会继续执行“SELECT * from 订单细节表”查询语句。

```
if (3 > SOME(select Productid from dbo.产品数据表))
SELECT * from 订单细节表
```

而如果改为以下表达式,因为“产品数据表”中的 ProductID 列并不全小于 3,因此,下式中的 IF 条件不成立,不会执行“SELECT * from 订单细节表”。


```
if (3 > ALL(select Productid from dbo.产品数据表))
SELECT * from 订单细节表
```

再如:

```
30&12 = 12,30|12 = 30,30^12 = 18,～50 = -51,3 % 4 = 3
```

5.4.5 控制流语句

T-SQL 中提供了 9 种控制流语句,可以实现对程序流程的控制。这些语句包括 BEGIN…END、BREAK、CONTINUE、GOTO、IF…ELSE、RETURN、TRY…CATCH、WAITFOR、WHILE 等,这些语句的含义如表 5-3 所示。

表 5-3 T-SQL 控制流语句的含义

控制流语句	含 义
BEGIN…END	用于定义一组要求连续执行的语句块,语句块可以嵌套定义
BREAK	跳出循环语句的循环过程,继续执行循环语句后面的语句
CONTINUE	重新开始新的 WHILE 循环
GOTO	跳转到由 GOTO 后指定的语句,并执行
IF…ELSE	条件分支语句,如果条件成立,执行 IF 后的语句;条件不成立则执行 ELSE 后的语句
RETURN	从过程、函数中返回,不再执行 RETURN 后的语句,如果 RETURN 语句指定有返回值,则将值返回,否则返回值为 0
TRY…CATCH	错误捕捉语句,程序先执行 TRY 后的语句,如果出现错误,则执行 CATCH 后的语句。因此,可以在 CATCH 中添加错误处理语句,实现对错误的响应
WAITFOR	挂起后续语句,直到以下情况发生:已超过指定的时间间隔、到达一天中指定的时间、指定的 RECVEIVE 语句至少修改一行数据;再继续执行挂起的及后续的语句
WHILE	为循环语句。当条件成立时,循环执行循环体内的语句,条件不成立时,执行循环体后续的语句

上述控制流语句经组合后,可以实现顺序结构、条件分支结构和循环结构等程序结构,以下对常用的程序结构进行介绍。

1. 使用 IF…ELSE 实现条件分支结构

IF…ELSE 语句是实现条件分支最常用的语句,基本语法如下。当 Boolean_expression 为真时,执行 IF 后的语句块;为假时,执行 ELSE 后的语句块。

```
IF Boolean_expression
    { sql_statement | statement_block }
[ ELSE
    { sql_statement | statement_block } ]
```

例如,以下代码配合 EXISTS 构建了一个条件分支语句。在 IF 语句中先判断是否存在“产品”表,如果存在,就执行 BEGIN…END 之间的语句块,不存在则提示“数据表产品不存在。”该段代码执行的结果如图 5-43 所示。

```
USE NetSale
IF EXISTS(SELECT * from INFORMATION_SCHEMA.TABLES where TABLE_NAME = '产品')
BEGIN
SELECT '数据表产品存在.'
SELECT * from 产品
END
else
SELECT '数据表产品不存在.'
```

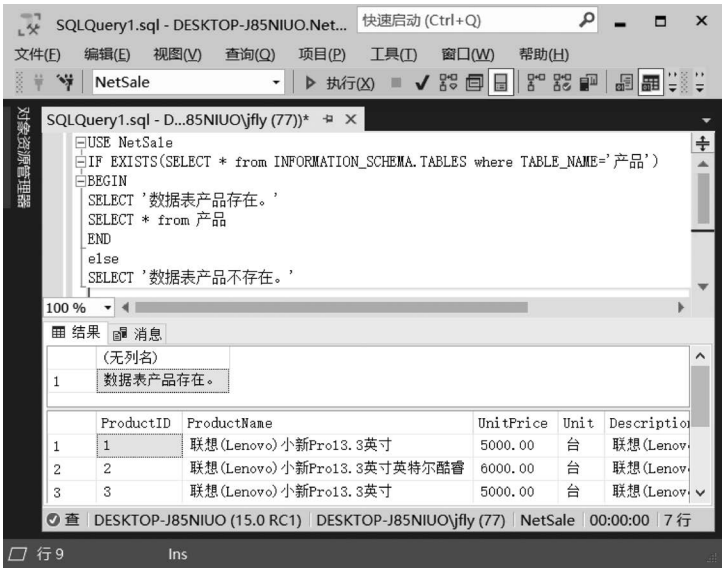

图 5-43　执行 IF…ELSE 条件分支语句的结果

2. 使用 WHILE 构造循环结构

WHILE 语句可以用来构造循环结构程序。WHILE 语句的语法如下，即当 Boolean_ex pression 表达式为真时，执行循环体内的语句，执行到 BREAK 时跳出循环，执行循环体后续的语句。

```
WHILE Boolean_expression
    { sql_statement | statement_block }
    [ BREAK ]
    { sql_statement | statement_block }
    [ CONTINUE ]
    { sql_statement | statement_block }
```

例如以下使用 WHILE 语句构造了一个循环结构的程序，WHILE 循环体执行了三次，当变量@i＝4 时，退出循环。执行结果如图 5-44 所示。

```
declare @i int
set @i = 1
while 0 <(SELECT COUNT( * ) from 产品)
BEGIN
  SELECT * from 产品
  set @i = @i + 1
  if @i > 3
  break
END
```

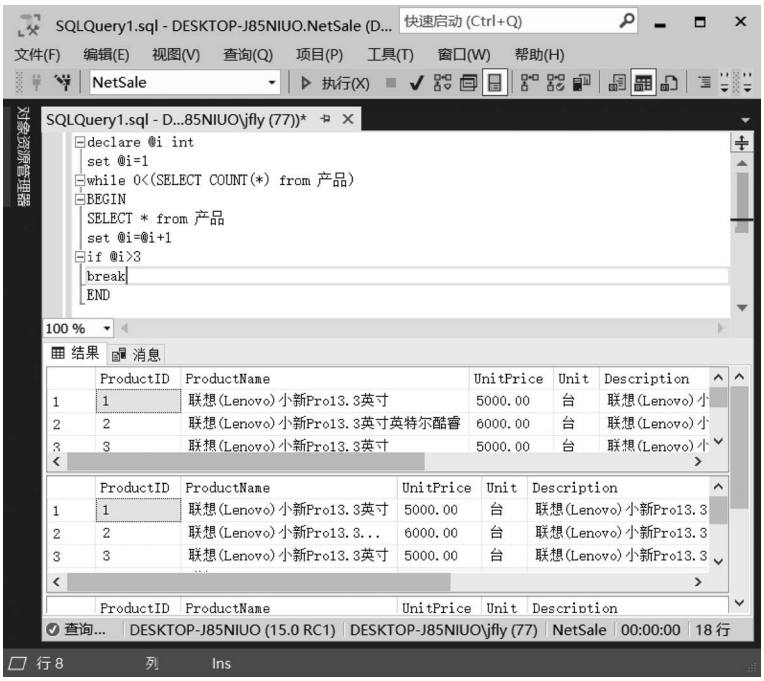

图 5-44　WHILE 循环执行结果

3. 使用 CASE

在 T-SQL 中还可使用 CASE 语句来构造条件分支结构。CASE 的基本语法如下，input_expression 为输入的用于判断的表达式，当 when_expression 表达式为真时，执行 THEN 后的语句，WHEN 语句可以有多条，当所有的 WHEN 语句条件都不满足时，执行 ELSE 后的语句。

```
CASE input_expression
     WHEN when_expression THEN result_expression
   [ …n ]
   [ ELSE else_result_expression ]
END
```

例如以下代码，构造了一个应用 CASE 语句实现的条件分支结构，根据 CategoryID 列取值不同，将数据分成三类，即“台式计算机”、“笔记本电脑”和“其他”，执行结果如图 5-45 所示。

```
SELECT ProductID, ProductName, UnitPrice, 产品类别 =
CASE
WHEN CategoryID = 1 THEN '台式计算机'
WHEN CategoryID = 2 THEN '笔记本电脑'
ELSE '其他'
END
FROM 产品数据表
```

T-SQL 的控制流语句同样可以实现其他程序设计语言的设计功能，如以下代码实现了 1＋2＋…＋100 的计算。

```
Declare @i int,@sum int
```

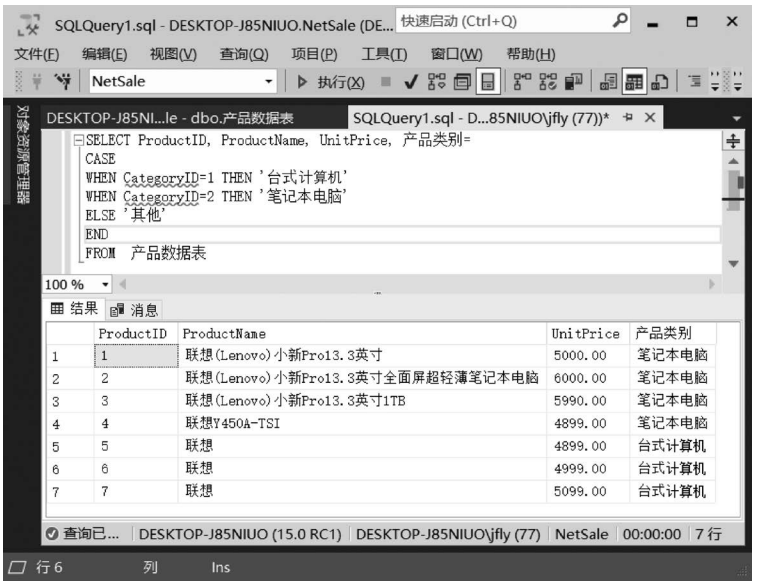

图 5-45　CASE 构造的条件分支结构执行结果

```
SET @i = 0
SET @sum = 0
while @i < = 100
BEGIN
SET @sum = @sum + @i
SET @i = @i + 1
END
SELECT @sum
```

执行结果为 5050。

5.4.6　函数

T-SQL 提供了大量的系统函数，可用于实现查询语句列运算、查询条件构造、触发器、视图以及各种表达式等。这些系统函数提高了 T-SQL 程序设计的效率，也进一步提升了 SQL Server 的易用性。

在 T-SQL 中，函数根据执行功能的不同，可以分为聚合函数、配置函数、加密函数、游标函数、日期和时间函数、数学函数、元数据函数、排名函数、行集函数、安全函数、字符串函数、系统函数、系统统计函数、文本函数 14 类，还可以根据返回值是否能够确定的不同，分为确定和非确定两类。确定指对于一组确定的输入值，函数始终返回相同的结果，如 SQUARE(3)的值始终为 9，则 SQUARE()就是严格确定函数。非确定指针对一组特定的输入值，返回的结果可能会不同，如 GETDATE()，会返回系统当前的时间，由于其返回值会根据执行时间的不同产生不同的值，因此是非确定的。

以下介绍几种常用的函数。

1. 聚合函数

聚合函数经常用于数据统计，如统计不同产品的销售量、统计最高成绩等。这类函数包括 AVG()、MIN()、MAX()、SUM()、COUNT()、STDEV()、STDEVP()、VAR()、VARP()等，这

些函数的使用方法在 5.3.1 节中已经介绍,此处不再重复介绍。

2. 日期和时间函数

日期和时间函数用于日期和时间的计算,在 T-SQL 查询中经常需要使用到对当月销售数量、本周出勤率等与时间有关的统计应用。使用日期和时间函数可以处理日期和时间数据,生成需要的结果值。

在 T-SQL 中,日期和时间函数主要包括 DATEADD()、DATEDIFF()、DATENAME()、DATEPART()、DAY()、GETDATE()、GETUTCDATE()、MONTH()、YEAR()等,这些函数的含义如表 5-4 所示。

表 5-4 日期时间函数

函 数 名 称	语法/含义
DATEADD()	DATEADD(datepart,number,date)
	在指定的日期时间值上加上一个时间间隔值,产生一个新值,如 Dateadd(dd,1,'2019-10-12')表示在“2019-10-12”时间值上加上 1 天(dd 代表天数),形成的新日期时间值为“2019-10-13”
DATEDIFF	DATEDIFF(datepart,startdate,enddate)
	用于返回两个日期时间值的边界数。如 DATEDIFF(year, '2009-12-31', '2019-01-02'),表示比较两日期的年份(year)的差值
DATENAME	DATENAME(datepart,datetoinspect)
	用于返回指定日期时间值中指定部分的值,如 DATENAME(month, '2018-01-25'),要求返回月份(MONTH)的值
DATEPART()	DATEPART(datepart,datetoinspect)
	用于返回指定日期值指定部分的值。如 DATEPART(DAY , '2018-01- 25'),要求返回天(DAY)的部分
DAY()	DAY(date)
	用于返回指定日期中的“天”的值,如 DAY('2010-02-07')
GETDATE()	GETDATE()
	返回系统当前的日期和时间值
GETUTCDATE()	GETUTCDATE()
	与 GETDATE()一样都能返回系统当前的日期时间,但值是 UTC 的日期时间值
MONTH()	MONTH(date)
	返回指定日期时间值的“月份”的值
YEAR()	YEAR(date)
	返回指定日期时间值的“年份”的值
EOMONTH()	EOMONTH(start_date,month_to_add)
	返回值为 datetime2(7),是针对指定开始日期 start_date 的月份的最后一天。month_to_add 是可选参数,如果提供这个参数值,函数会将此值添加到开始日期 start_date 的月数上,然后再返回结果月份的最后一天
DateFromParts()	DateFromParts(year,month,day)
	基于给定的 year,month,day 返回 Date 值
DateTime2FromParts()	DateTime2FromParts (year, month, day, hour, minute, seconds, fractions, precision)
	基于给定的 year,month,day,hour,minute,seconds,fractions,precision 返回 DateTime2 值

续表

函 数 名 称	语法/含义
DateTimeFromParts()	DateTimeFromParts（year，month，day，hour，minute，seconds，fractions，precision）
	基于给定的 year，month，day，hour，minute，seconds，fractions，precision 返回 DateTime 值
DateTimeOffSetFrom Parts()	DateTimeOffSetFromParts（year，month，day，hour，minute，seconds，fractions，hour_offset，minute_offset，seconds_offset，fractions_offset，precision）
	基于给定的 year，month，day，hour，minute，seconds，fractions，hour_offset，minute_offset，seconds_offset，fractions_offset，precision 返回 DateTime Offset 值
SmallDateTimeFrom Parts()	SmallDateTimeFromParts（year，month，day，hour，minute）
	基于给定的 year，month，day，hour，minute 返回 DateTime 值
TimeFromParts()	TimeFromParts（year，month，day，hour，minute，seconds，fractions，precision）
	基于给定的 year，month，day，hour，minute，seconds，fractions，precision 返回 Time 值

表 5-4 中，参数 datepart 的取值与含义如表 5-5 所示。

表 5-5　datepart 的取值及含义

日期部分	缩写	说明	日期部分	缩写	说明
year	yy，yyyy	年	week	wk，ww	第几星期
quarter	qq，q	季度	weekday	dw	周几
month	mm，m	月	hour	hh	小时
dayofyear	dy，y	一年中的第几天	minute	Mi，n	分
day	dd，d	日期	second	ss，s	秒
millisecond	Ms	毫秒			

例如，以下代码可以从“订单表”中返回一个月内的订单，如果今天是“2019-12-07”，则订单的时间范围为“2019-12-07”到“2020-01-08”，执行结果如图 5-46 所示。

```
SELECT * FROM dbo.订单表
WHERE ordertime < DATEADD(m,1,ATEADD(d,1,getdate())) and ordertime > = GETDATE()
```

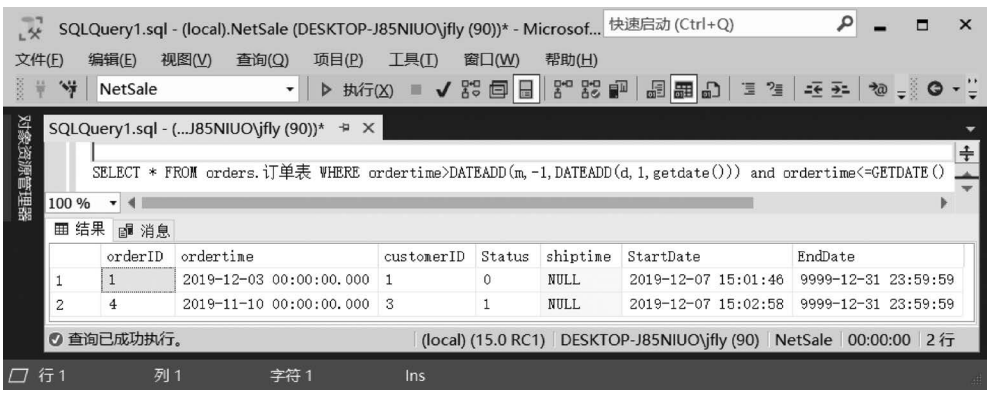

图 5-46　使用日期函数查询 1 月的内订单

如果需要查询当月订单，可以使用以下代码，即通过判断年份与月份是否同时相等来获取当月的订单。

```
SELECT * FROM dbo.订单表
WHERE datePART(yy,ordertime) = datePART(yy,getdate()) and datePART(m,ordertime) = datePART
(m,getdate())
```


3. 数学函数

T-SQL 中提供了 23 种数学函数,这些数学函数可以对 SQL Server 2022 中的各种数值型数据进行运算。这些数学函数包括 ABS()、ACOS()、ASIN()、ATAN()、ATN2()、CEILING()、COS()、COT()、DEGREES()、EXP()、FLOOR()、LOG()、LOG10()、PI()、POWER()、RADIANS()、RAND()、ROUND()、SIGN()、SIN()、SQRT()、SQUARE()、TAN()等,这些数学函数的功能与 C#、C 等编程语言的函数功能相同,请参见相关资料。

4. 字符串函数

字符串函数是对字符串进行各种操作的函数。T-SQL 提供的字符串函数包括 ASCII()、CHAR()、CHARINDEX()、CONCAT ()、DIFFERENCE()、FORMAT ()、LEFT()、LEN()、LOWER()、LTRIM()、NCHAR()、PATINDEX()、QUOTENAME()、REPLACE()、REPLCATE()、REVERSE()、RIGHT()、RTRIM()、SOUNDEX()、SPACE()、STR()、STUFF()、SUBSTRING()、UNICODE()、UPPER()等。

这些函数的具体含义见表 5-6。

表 5-6 字符串函数的含义

函 数	含 义
ASCII()	将单个字符转换成对应的 ASCII 码
CHAR()	将一个数字值转换成字符
CHARINDEX()	返回字符串在另一个字符串中的起始位置
CONCAT()	将两个或多个字符串组合为单个字符串
DIFFERENCE()	返回一个表示两个字符表达式的 SOUNDEX 值差异的整数
FORMAT()	返回指定格式的值
LEFT()	返回字符串中从左边开始到指定长度的字符
LEN()	返回字符串的长度
LOWER()	返回字符串的小写形式
LTRIM()	返回去除字符串左边空格之后的字符串值
NCHAR()	返回指定整数代码的 UNICODE 字符
PATINDEX()	返回指定表达式中某模式第一次出现的起始位置;如果在全部有效的文本和字符数据类型中没有找到该模式,则返回零
QUOTENAME()	返回带有分隔符的 UNICODE 字符
REPLACE()	将表达式中的字符串转换成其他字符串或空格
REPLCATE()	返回多次复制后的字符串表达式
REVERSE()	返回反转后的字符串
RIGHT()	返回字符串中从右边开始到指定长度的字符
RTRIM()	返回去除字符串右边空格之后的字符串值
SOUNDEX()	返回一个由 4 个字符组成的代码,用于评估两个字符串的相似性

续表

函　　数	含　　义
SPACE()	返回由指定数量空格组成的字符串
STR()	返回由数字数据转换来的字符数据
STUFF()	将字符串插入另一个字符串
SUBSTRING()	返回整个字符串中指定的部分字符串
UNICODE()	返回输入表达式的第一个字符的整数值
UPPER()	返回字符串的大写形式

例如以下代码使用了部分字符串函数，执行的结果如图 5-47 所示。

```
SELECT ASCII('SQL Server') as 'ASCII',CHAR(56) AS 'CHAR(56)',
LEFT('SQL Server',5) as 'LEFT5',LOWER('SQL Server') as 'LOWER',
UPPER('SQL Server') AS 'UPPER',RIGHT('SQL Server',5) as 'RIGHT5',
SUBSTRING('SQL Server',3,LEN('SQL Server')) as 'SUMSTRING'
```

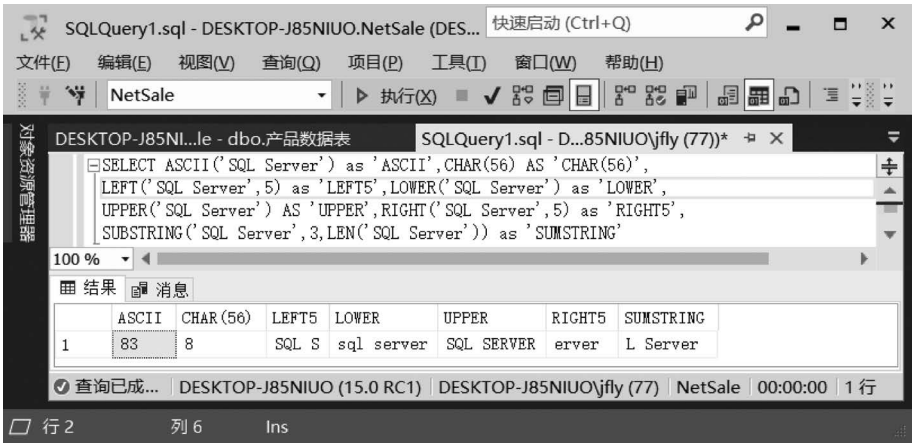

图 5-47　字符串函数执行的结果

5. 窗口函数

窗口函数也被称为 OLAP(OnLine Analytical Processing)函数，即数据在线分析处理函数。SQL Server 提供的窗口函数由两部分组成：一部分是聚合函数(如 SUM、AVG、COUNT、MAX、MIN)；另一部分是 RANK、DENSE_RANK、ROW_NUMBER 等专用窗口函数。应用窗口函数结合 OVER、PARTITION BY 和 ORDER BY 子句，可以实现对数据进行聚合统计、分组排名等多种实时在线分析。

窗口函数的基本语法如下：

```
<窗口函数> OVER ([partition_by_clause] order_by_clause )
```

例如以下代码，使专用窗口函数 RANK、DENSE_RANK、ROW_NUMBER 统计了各产品销售量的排名，执行结果如图 5-48 所示。

```
SELECT p.ProductID, ProductName, o.sumtotal as 销售金额,o.[sales],产品类别 =
CASE
WHEN CategoryID = 1 THEN '台式计算机'
WHEN CategoryID = 2 THEN '笔记本电脑'
```

```
ELSE '其他'
END,
rank() over (order by o.sales desc) as rank,
dense_rank() over (order by o.sales desc) as dense_rank,
row_number() over (order by o.sales desc) as row_num
FROM 产品数据表 p inner join [dbo].[订单细节表] o on p.ProductID = o.productid order by rank
```

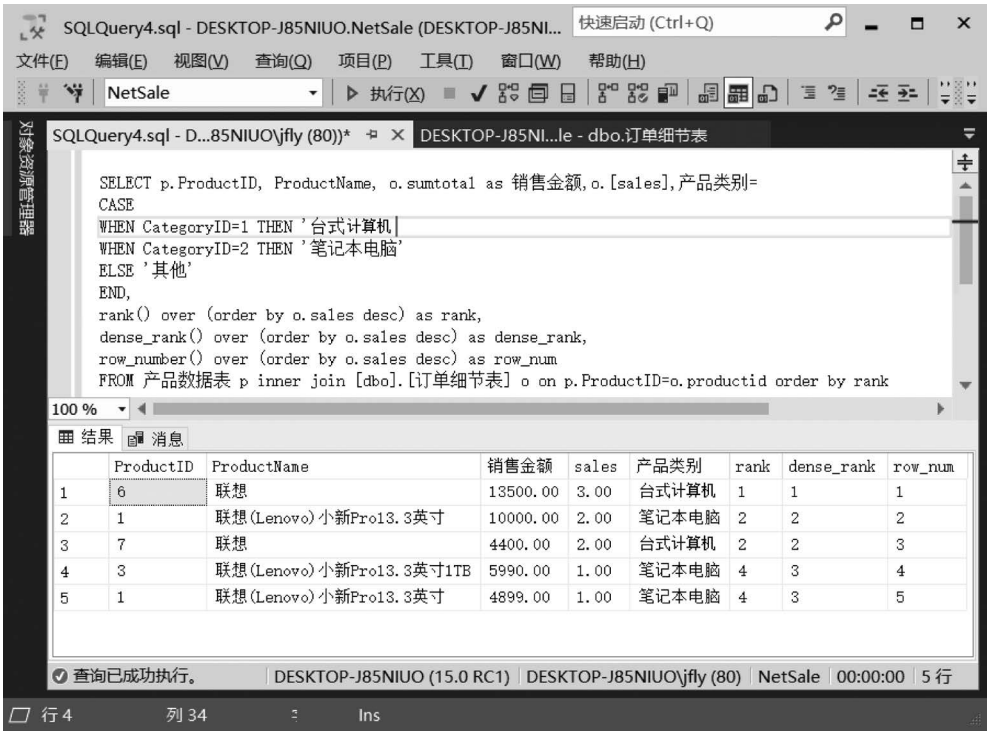

图 5-48　使用窗口函数统计销量排名

从图 5-48 中可以看出,RANK、DENSE_RANK 和 ROW_NUMBER 三者在生成排序值方面的区别。

- RANK 函数在排序时,如果有多行排序值相同,则这些行的排名值相同,但后续行排名值会跳过这些行数再排名,如图 5-48 中 rank 排名值有两行值为 2,则后续会从 4 开始,而不是从 3 开始。也即 RANK 函数排名时,排名值并不总是连续的。
- DENSE_RANK 函数在排序时,也会出现多行排名值相同的情况,但不会跳过。即如图 5-48 中所示,dense_rank 列有两行值为 2,后续依旧会从 3 开始。
- ROW_NUMBER 函数的作用是给每一行生成一个连续的排序值。

窗口函数可以结合 PARTITION BY 生成分区的排名,如上述实例中,可以根据需要实现按产品类别分区,生成每一区的销量排名。如以下代码实现上述要求,执行结果如图 5-49 所示。

```
SELECT p.ProductID, ProductName, o.sumtotal as 销售金额,o.[sales],产品类别 =
CASE
WHEN CategoryID = 1 THEN '台式计算机'
WHEN CategoryID = 2 THEN '笔记本电脑'
ELSE '其他'
```

```
END,
rank () over (partition by CategoryID order by o.sales desc) as rank,
dense_rank () over (partition by CategoryID order by o.sales desc) as dense_ranking,row_number ()
over (partition by CategoryID order by o.sales desc) as row_num FROM 产品数据表 p inner join [dbo].
[订单细节表] o on p.ProductID = o.productid order by CategoryID,rank
```

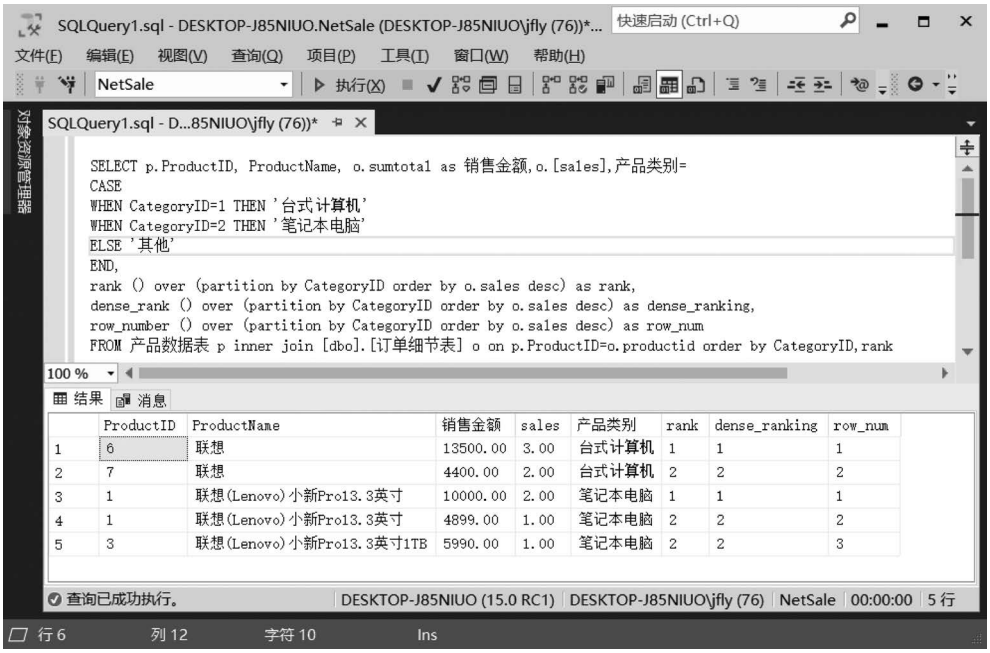

图 5-49　使用 PARTITION BY 分区的窗口函数

从执行结果中可见,使用 PARTITION BY 的窗口函数,先对数据按 CategoryID 列进行分组,然后再按 ORDER By 指定的列排序,再生成各组对应的排序值。

6. 其他常用函数

1) ISDATE()

ISDATE()函数是一个对列、变量或常量进行判断,判别该列、变量或常量是否是一个日期时间值的函数。如果是日期时间,则返回 1,否则返回 0。这个函数能否正确运算与操作系统的区域和日期格式设置有关。

例如:

```
ISDATE('20010 - 01 - 01')
```

2) ISNULL()

ISNULL()函数用于判断列或变量的值是否为 NULL,如果为 NULL,可以用指定的值替换 NULL 值。该函数的语法如下:

```
ISNULL(value_to_test,new_value)
```

例如以下代码使用 ISNULL 函数对变量进行判断,并使用 ISNULL 字符串去替换 NULL 值。

```
DECLARE @p varchar(20)
select ISNULL(@p,'ISNULL')
```

由于变量刚定义,未赋值前其值为 NULL,所以该代码的输出结果为 ISNULL。

3) ISNUMERIC()

ISNUMERIC()函数用于对变量、列或常量进行判断,确定是否是数字值,如果是数字值则返回 1,否则返回 0。货币符号也会被判定为数字值。

例如以下代码中,@p 被赋值为 1.2 是数字值,因此判定结果为 1。

```
DECLARE @p varchar(20)
SET @p = 1.2
select ISNUMERIC(@p)
```

7. 用户自定义函数

在 SQL Server 2022 中,除了可以使用由系统提供的上述大量的系统函数之外,用户根据需要还可以构建用户自定义函数(User-Defined Functions,UDF)。用户自定义函数是一组有序的 T-SQL 语句,这些语句被预先优化和编译,可以作为一个整体进行调用。使用用户自定义函数可以提高代码的可重用性,也有助于简化业务系统的复杂程度。

在 SQL Server 2022 中,根据用户定义函数返回值的不同,可以分成为两大类:标量值函数和表值函数。用户可以将自己定义的函数分别归类到上述类别中。

1) 标量值函数

标量值函数指执行后返回结果是某种具体数据类型的用户自定义函数,这些数据可以是除 BLOB(二进制类型的大对象)、游标(Cursor)和时间戳(Timestamp)之外的任何有效的 SQL Server 数据类型。

以下代码创建了用于判断产品是否可销售的用户自定义函数,函数的名称为 IsCanSale,返回的数据类型为 BIT。函数输入两个整型参数,@ProductID 用于代表产品编号,@Qty 代表产品销售数量,通过判断"产品数据表"中指定的产品是否有足够库存量来确定返回值,返回值为 1,表示产品可销售;为 0,表示产品数量不足或产品不存在,不能销售。

```
USE [NetSale]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE FUNCTION [dbo].[IsCanSale](@ProductID int,@Qty int)
RETURNS BIT
AS
BEGIN
DECLARE @iReturn BIT
SET @iReturn = 0
IF(EXISTS(SELECT * FROM [dbo].[产品数据表] WHERE [ProductID] = @ProductID AND [Instock]> =
@Qty))
SET @iReturn = 1
ELSE
SET @iReturn = 0
Return @iReturn
END
```

此函数创建后,可以在当前数据库中作为一个对象,在查询、视图、存储过程等多处调

用，甚至可以被其他函数调用，如以下代码演示了函数在查询中调用的方法。

```
SELECT * FROM [dbo].[产品数据表] WHERE [dbo].[IsCanSale](ProductID,20) = 1
```

执行结果如图 5-50 所示。

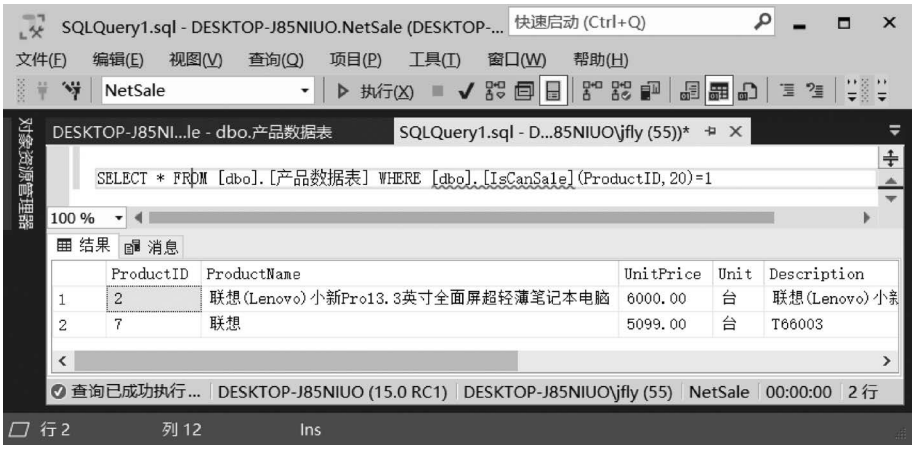

图 5-50　调用标量值用户自定义函数

2）表值函数

与标量值函数相对应，表值函数指返回值的数据类型为表的用户自定义函数。由于返回结果为表，这使表值函数在需要提取中间数据的场合有很重要的适用性。如在某些场合，需要先了解“产品数据表”中库存不足的产品数据，然后再由这些数据生成对应的采购数据，在这个业务应用中，库存不足的产品数据是中间表，可以由表值函数来实现。

以下代码创建了一个用于获取“产品数据表”库存小于某一指定值的产品数据的用户自定义函数。函数的名称为 Get_LowStockProduct，返回值的数据类型为 TABLE，使用的参数@Qty 用于指定库存的最低数量。

```
USE [NetSale]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE FUNCTION [dbo].[Get_LowStockProduct](@Qty int)
RETURNS TABLE
AS
RETURN (SELECT * FROM [dbo].[产品数据表] WHERE [Instock]<=@Qty)
```

同样，表值函数也可以在查询、视图、存储过程和函数等多处重复调用。以下代码演示了在查询中调用的方法。

```
DECLARE @Qty int
set @Qty = 20
SELECT * FROM [dbo].[Get_LowStockProduct](@Qty)
```

执行结果如图 5-51 所示。

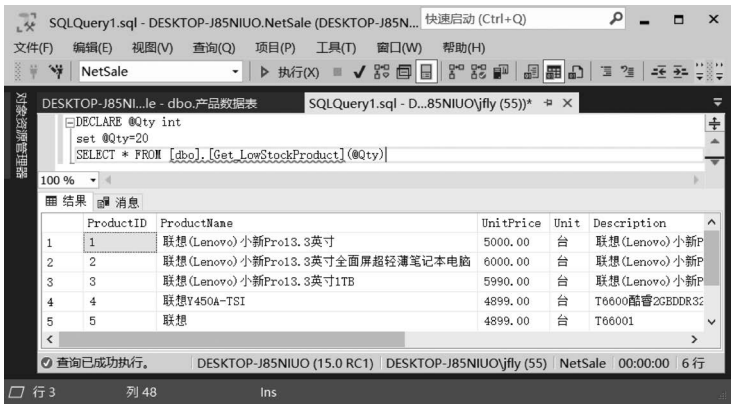

图 5-51 调用表值函数

5.5 使用通用表表达式

通常,使用 SELECT 可以从表中获取需要的数据,形成一个数据集。在有些场合,需要先生成这个数据集然后在后续的代码中使用,SQL Server 可以使用临时表等去生成这个数据集并临时保存,但会降低效率。采用通用表表达式(Common Table Expression,CTE)在需要的时候生成,用完丢弃,也是非常好的一种解决方案。

通用表表达式,一方面可以让代码更加简洁,减少重复代码;另一方面也可以实现递归调用,实现层次化的数据查询。

5.5.1 定义通用表表达式

CTE 的定义需要使用 WITH 关键词,其基本定义语法如下:

```
WITH expression_name [(column_name [,…n] )]
AS(
   cte_query_definition
  )
```

各参数含义如下:

- expression_name,通用表表达式的名称。
- column_name[,…n],通用表表达式可以返回的列列表,如果返回的是 cte_query_definition 中定义的所有列,那么此项参数可以省略。
- cte_query_definition,通用表表达式定义的查询语句。

如以下代码定义了一个名称为 myCTE 的通用表表达式,定义的查询语句为从产品数据表和订单细节表中获取某一产品的销量和销售金额。此处忽略了通用表表达式返回的列列表,即表示返回的是查询语句中的所有列。

```
WITH myCTE
AS
(
SELECT p.ProductID,ProductName,o.sumtotal as 销售金额,o.sales FROM 产品数据表 p inner join
[dbo].[订单细节表] o on p.ProductID = o.productid
)
```

当通用表表达式返回的列与定义的不一致时，如以下代码，在列表中 Amount 和 sale_Count 列与定义的“销售金额”和 sales 列是不相同的，需要明确定义，但会顺序对应。

```
WITH myCTE(ProductID,ProductName,Amount,sale_Count)
AS
(SELECT p.ProductID, ProductName,o.sumtotal as 销售金额,o.sales
FROM 产品数据表 p inner join [dbo].[订单细节表] o on p.ProductID = o.productid)
```

如果有多个通用表表达式需要定义，可以在同一个 WITH 关键词下定义，如以下代码定义了两个通用表表达 myCTE 和 myCTE1，其中在 myCTE1 的查询语句中使用了 myCTE。

```
WITH myCTE(ProductID, ProductName, 销售金额,sales,orderID)
AS
(SELECT p.ProductID, ProductName, od.sumtotal as 销售金额,od.sales,od.orderid
FROM 产品数据表 p inner join [dbo].[订单细节表] od on p.ProductID = od.productid
),myCTE1
AS(Select m.ProductID,m.ProductName,m.sales,m.销售金额,c.CustomerName from 订单表 o inner
join myCTE m on o.orderID = m.orderID inner join 客户数据表 c on o.customerID = c.CustomerID )
```

在定义通用表表达式时，需要注意的是，如果定义语句在一个代码体中并非处在最前面，即在 WITH 关键词的前面还有其他语句，则需要在 WITH 之前添加“;”，表示之前的语句在此 WITH 前已结束。因为 WITH 关键词并不仅仅只用于定义 CTE，还可用于定义其他选项参数，因此必须做好区分。

5.5.2 使用通用表表达式

通用表表达式定义完成后，就可以在其他语句中进行使用，但是需要注意的是，只能在通用表表达式的后续第一条语句中使用。如以下代码，在定义通用表表达式之后，直接在后续的 SELECT 语句中得到了使用。

```
WITH myCTE
AS(
SELECT p.ProductID, ProductName, o.sumtotal as 销售金额,o.sales FROM 产品数据表 p inner join
[dbo].[订单细节表] o on p.ProductID = o.productid)
select * from myCTE
```

执行结果如图 5-52 所示。

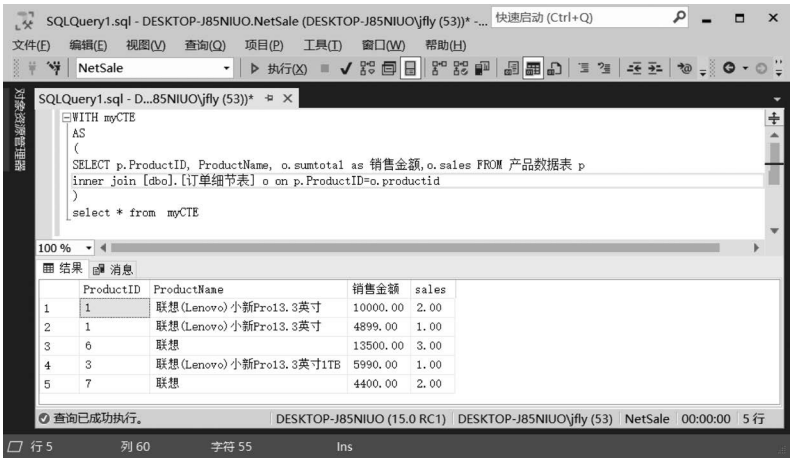

图 5-52 使用通用表表达式

5.6 本章小结

T-SQL 是 SQL Server 数据管理的基础,几乎所有 SQL Server 的管理操作都可以通过 T-SQL 完成。T-SQL 是基于 SQL 国际标准,应用于 SQL Server 中的数据操纵语言;与 SQL 国际标准相比,增加了很多独具特色的语言,使 SQL Server 具备更强的功能和特色。

本章介绍了 T-SQL,包括 T-SQL 的数据操纵语言、数据查询语言,并对 T-SQL 的附加元素,如变量、运算符、控制流语句、函数和通用表达式等进行了深入介绍。掌握 T-SQL 是进一步深入 SQL Server 2022 应用和管理的重要前提,也有益于深入了解 SQL Server 2022 的管理机制。

习题与思考

1. 请简述 T-SQL 的发展过程及与 ANSI SQL 的区别。

2. T-SQL 语言的分类有哪几种?分别包含哪些语句?

3. 如何使用 INSERT 语句在数据表中添加数据?如何使用 UPDATE 语句修改指定条件数据行的列的值?如何使用 DELETE 语句删除表中指定条件的数据?

4. 如何使用 SELECT 语句检索表中指定条件的数据?如何构造模糊匹配条件?如何使用通配符构建检索条件?

5. 如何联接多表检索数据?SQL Server 中多表联接检索的种类有哪些?

6. 如何对检索结果数据进行排序和分组?

7. 如何去除检索结果中的重复数据?

8. 如何使用 UNION、INTERSECT 和 EXCEPT 运算符实现组合查询?

9. SQL Server 中标识符命名有何要求?

10. 如何使用 Select into 批量加载数据到数据表中?

11. 如何使用 IF…ELSE、WHILE、CASE 等语句构造流程控制程序?

12. 如何使用 OFFSET 和 FETCH 参数实现分页?

13. 通用表表达式如何定义?请说明使用的方法。