第5章 表的管理

表是 SQL Server 实例中全部数据的容器，关系数据库中的所有数据都存储在表中，是数据库中最重要的部分。表也是数据库所有其他对象的基础，管理数据库首先必须合理、有效地管理表。本章首先介绍 SQL Server 中表的相关概念，然后介绍表的设计、创建、修改和删除以及如何对表中的数据进行添加、修改和删除。

5.1 SQL Server 表概述

在 SQL Server 2012 中，一个数据库最多可以存储 20 亿个表。表是数据库的一种对象，由行和列组成，在数据库中，表的每一列表示数据库表的一个属性，每个表最多可以有 1024 列，表的每一行表示一条记录，是对实体完整性的描述，每行最多可以存储 8060 字节内容，表的行数及总大小仅受可用存储空间的限制。在设计数据库时，要根据数据库逻辑结构设计的要求，确定需要什么样的表、各表中都有哪些数据、表的各列的数据类型及列宽、哪些列允许空值、哪里需要索引、哪些列是主键、哪些是外键以及是否要使用和何时使用约束、默认设置或规则等。

5.1.1 数据类型简介

在 SQL Server 中，每个列、局部变量、表达式和参数都具有一个相关的数据类型。数据类型是一种属性，用于指定对象可保存的数据的类型：整数数据、字符数据、货币数据、日期和时间数据、二进制字符串等。

SQL Server 提供系统数据类型集，该类型集定义了可与 SQL Server 一起使用的所有数据类型。用户还可以使用 Transact-SQL 或 Microsoft .NET Framework 定义自己的数据类型。别名数据类型基于系统提供的数据类型。

SQL Server 2012 支持的数据类型可以归纳为字符数据、二进制数据、日期和时间数据、逻辑数据、数字数据和其他数据类型。

1. 字符数据类型

1) 字符型数据类型

字符型数据类型用于存储汉字、英文字母、数字符号和其他各种符号。输入字符型数据时要用单引号(')或双引号(")将字符括起来。字符型数据有定长字符型(char)、变长字符型(varchar)和文本型(text)3 种。

char 数据类型的定义形式为 char[(n)]，n 的取值为 1～8000，即最多可存储 8000 个字符。指定的字符取决于安装 SQL Server 时所指定的字符集，通常采用 ANSI 字符集。在用

char(n)数据类型对列进行说明时，指示列长度为 n。在数据定义或变量声明语句中如果没有指定 n，则系统默认长度为 1。对于超过列长度的输入将被截断，若输入字符的长度短于指定字符长度时则用空格填满。

varchar 数据类型的定义形式为 varchar[(n|max)]，n 的取值为 1～8000。max 指示最大存储大小是 $2^{31}-1$ 字节(2GB)。varchar 数据类型的结构与 char 数据类型一致，它们的主要区别是当输入 varchar 字符长度小于 n 时不用空格来填满，按输入字符的实际长度存储。若输入的数据超过 n 个字符，则截断后存储。varchar 类型所需存储空间要比 char 数据小一些，但 varchar 列的存取速度比 char 列要慢一些。

text 数据类型用于存储数据量庞大而变长的字符文本数据。text 列的长度可变，最多可包含 $2^{31}-1$ 个字符。用户要求表中的某列能存储 255 个字符以上的数据，可使用 text 数据类型。text 数据类型不能用作变量或存储过程的参数。

2) Unicode 字符数据类型

SQL Server 允许使用多国语言，采用 Unicode 标准字符集。为此 SQL Server 提供多字节的字符数据类型：nchar(n)、nvarchar(n|max)和 ntext。

Unicode 字符串的格式与普通字符串相似，但 Unicode 数据中的每个字符都使用两字节进行存储。Unicode 字符串常量的前面有一个大写 N(N 代表 SQL-92 标准中的国际语言——National Language)。例如，'Michael'是字符串常量，而 N'Michael'则是 Unicode 常量。类似地，Unicode 字符串的几种类型也是在普通字符串的类型名前增加了一个字母 n 来标识的。

nchar 可存放 Unicode 字符的固定长度字符类型，n 最大长度为 4000 个字符。

nvarchar 可存放 Unicode 字符的可变长度字符类型，n 最大长度为 4000 个字符。max 指示最大存储大小是 $2^{31}-1$ 字节。

ntext 可存放 Unicode 字符的文本类型，其最大长度为 $2^{30}-1$ 个字符。存储大小是所输入字符串长度的两倍(以字节为单位)。

nchar、nvarchar 和 ntext 的用法分别与 char、varchar 和 text 相同，只是 Unicode 支持的字符范围更大，存储 Unicode 字符需要一些额外开销空间。由于这些额外开销和增加的空间，应该避免使用 Unicode 列，除非确实有需要使用它们的业务或语言需求。

text 数据类型用于在数据页内外存储大型字符数据。与 text 数据类型相比，更好的选择是使用 varchar(max)类型，因为将获得更好的性能。另外，text 和 ntext 数据类型在 SQL Server 的一些未来版本中将不可用，因此最好使用 varchar(max)和 nvarchar(max)，而不是 text 和 ntext 数据类型。

SQL Server 2012 的字符数据类型如表 5-1 所示。

表 5-1　SQL Server 2012 的字符数据类型

数 据 类 型	数据范围描述	存 储 空 间
char(n)	n 为 1～8000 字符	n 字节
nchar(n)	n 为 1～4000 Unicode 字符	n 字节的两倍
varchar(n)	n 为 1～8000 字符	2×字符数＋2 字节额外开销
nvarchar(n)	n 为 1～4000 字符	2×字符数＋2 字节额外开销

续表

数据类型	数据范围描述	存储空间
varchar(max)	最多为 $2^{31}-1$ 字节	2×字符数+2 字节额外开销
nvarchar(max)	最多为 $2^{31}-1$ 个 Unicode 字符	2×字符数+2 字节额外开销
text	最多为 $2^{31}-1$ 个字符	每字符 1 字节
ntext	最多为 $2^{30}-1$ 个 Unicode 字符	每字符 2 字节

2. 二进制数据类型

SQL Server 二进制数据类型用于存储二进制数或字符串。与字符数据类型相似,在列中插入二进制数据时,用引号标识,或用 0x 开头的两个十六进制数构成一字节。SQL Server 有 3 种有效二进制数据类型,即定长二进制类型 binary、变长二进制类型 varbinary 和大块二进制类型 image。

binary 数据类型的定义形式为 binary[(n)],n 的取值为 1～8000,若不指定 n,则 n 默认为 1。binary 数据用于存储二进制字符,例如程序代码和图像数据。

varbinary[(n|max)]数据类型与 binary 数据类型基本相同,但通过存储输入数据的实际长度而节省存储空间,但存取速度比 binary 类型要慢。n 的取值范围为 1～8000。除非数据长度超过 8KB,一般宜用 varbinary 类型来存储二进制数据,建议列宽的定义不超过所存储的二进制数据可能的最大长度。max 指示最大存储大小是 $2^{31}-1$ 字节。

image 数据类型与 text 数据类型类似,可存储 1～$2^{31}-1$ 字节的二进制数据。image 数据类型存储的是二进制数据而不是文本字符,不能用作变量或存储过程的参数。image 数据列可以用来存储超过 8KB 的可变长度的二进制数据,如 Word 文档、Excel 电子表格、图像或 MP3 等其他文件。image 数据类型可在数据页外部存储最多 2GB 的文件。image 数据类型的首选替代数据类型是 varbinary(max),其性能通常比 image 数据类型好。

表 5-2 列出了二进制数据类型,对其做了简单描述,并说明了要求的存储空间。

表 5-2　二进制数据类型

数据类型	数据范围描述	存储空间
binary(n)	n 为 1～8000 十六进制数字	n 字节
image	最多为 $2^{31}-1$ 字节	每字符 1 字节
varbinary(n)	n 为 1～8000 十六进制数字	每字符 1 字节+2 字节额外开销
varbinary(max)	最多为 $2^{31}-1$ 字节	每字符 1 字节+2 字节额外开销

3. 日期和时间数据类型

日期和时间数据类型用于存储日期和时间数据。SQL Server 2012 支持多种日期和时间数据类型:datetime、datetime2、dateoffset、smalldatetime、date 和 time。

datetime 数据类型存储两个长度为 4 字节的整数:日期和时间。datetime 数据类型有许多格式,可被 SQL Server 的内置日期函数操作。

datetime2 数据类型是 datetime 数据类型的扩展,有着更广的日期范围。时间总是用时、分钟、秒形式来存储的。可以定义末尾带有可变参数的 datetime2 数据类型,如 datetime2(3)。这个表达式中的 3 表示存储时秒的小数精度为 3 位,或 0.999。有效值为 0～9,默认值为 3。

datetimeoffset 数据类型和 datetime2 数据类型一样，带有时区偏移量。该时区偏移量最大为+/−14 小时，包含了 UTC（Universal Time Coordinated，协调世界时）偏移量，因此可以合理化不同时区捕捉的时间。

smalldatetime 数据类型只需 4 字节的存储空间，时间值是按小时和分钟来存储的。插入数据时，日期和时间值以字符串形式传给服务器。

date 数据类型只存储日期，而 time 数据类型只存储时间。它也支持 time(n)声明，因此可以控制小数秒的粒度。与 datetime2 和 datetimeoffset 一样，n 可为 0～7。

表 5-3 列出了日期和时间数据类型，对其进行简单描述，并说明了要求的存储空间。

表 5-3　日期和时间数据类型

数 据 类 型	数据范围描述	存 储 空 间
Date	0001-01-01～ 9999-12-31	3 字节
Datetime	1753 年 1 月 1 日～9999 年 12 月 31 日，时间范围为 00：00：00～23：59：59.997	8 字节
Datetime2(n)	0001-01-01～ 9999-12-31，n 为 0～7，指定小数秒	6～8 字节
Datetimeoffset(n)	0001-01-01～ 9999-12-31 日，n 为 0～7，指定小数秒 +/−偏移量	默认 10 字节
SmalldateTime	1900 年 1 月 1 日～2079 年 6 月 6 日，精确到 1 分钟	4 字节
Time(n)	小时：分钟：秒.9999999，n 为 0～7，指定小数秒	5 字节

4. 逻辑数据类型

SQL Server 的逻辑数据类型也称为位(bit)数据类型，适用于判断真/假的场合，长度为一字节。位数据类型取值为 1、0 或 NULL。非 0 的数据被当成 1 处理，位列不允许建立索引，多个位列可以占用同一字节。如果一个表有不多于 8 个的位列，SQL Server 将这些列合在一起用一字节存储。如果表中有 9～16 个位列，这些列将作为两字节存储。更多列的情况以此类推。字符串值 TRUE 和 FALSE 可转换为 bit 值，TRUE 将转换为 1，FALSE 将转换为 0。

5. 数字数据类型

SQL Server 提供了多种方法存储数值，SQL Server 的数字数据大致可分为 4 种基本类型。

1）整数数据类型

有 4 种整数数据类型：tinyint、smallint、int 和 bigint，用于存储不同范围的值。整数可以用较少的字节存储较大的精确数字，考虑到其高效的存储机制，只要有可能，对数值列应尽量使用整数。SQL Server 2012 的整数类型如表 5-4 所示。

表 5-4　SQL Server 2012 的整数数据类型

数 据 类 型	数据范围描述	存储空间/B
tinyint	0～255 的整数	1
smallint	-2^{15}～$2^{15}-1$ 的整数	2
int	-2^{31}～$2^{31}-1$ 的整数	4
bigint	-2^{63}～$2^{63}-1$ 的整数	8

2) 近似数值数据类型

近似数值数据类型包括 float 和 real 类型。它们用于表示浮点数据。但是,由于它们是近似的,因此不能精确地表示所有值。

float(n)中的 n 是用于存储该数尾数的位数(以科学记数法表示)。SQL Server 对此只使用两个值。如果指定位于 1～24,SQL Server 就使用 24。如果指定位于 25～53,SQL Server 就使用 53。当指定 float()时(括号中为空),默认为 53。real 的同义词为 float(24)。

表 5-5 列出了近似数值数据类型,对其进行简单描述,并说明了要求的存储空间。

表 5-5 近似数值数据类型

数 据 类 型	数据范围描述	存 储 空 间
float[(n)]	-1.79×10^{308}～-2.23×10^{-308}, 0,2.23×10^{-308}～1.79×10^{308}	n 为 1～24 时,4 字节 n 为 25～53 时,8 字节
real()	-3.40×10^{38}～-1.18×10^{-38}, 0,1.18×10^{-38}～3.40×10^{38}	4 字节

3) 精确数值数据类型

精确数值数据类型用于存储有小数点且小数点后位数确定的实数。SQL Server 支持两种精确的数值数据类型:decimal 和 numeric。这两种数据类型在功能上等价,定义格式如下:

```
decimal[(p[, s])]
numeric[(p[, s])]
```

其中,p 指定精度,即小数点左边和右边可以存储的十进制数字的最大个数。s 指定小数位数,即小数点右边可以存储的十进制数字的最大个数。精确数值数据类型如表 5-6 所示。

表 5-6 精确数值数据类型

数 据 类 型	数据范围描述	存 储 空 间
numeric(p,s)或 decimal(p,s)	$-10^{38}+1$～$10^{38}-1$ 的数值	最多 17 字节

4) 货币数据类型

除了 decimal 和 numeric 类型适用于货币数据的处理外,SQL Server 还专门提供了两种货币数据类型:money 和 smallmoney,如表 5-7 所示。

输入货币数据时必须在货币数据前加 $ 符号,如果未提供该符号,其值被当成浮点数,可能会损失值的精度,甚至被拒绝。在显示货币值时,数值的小数部分仅保留 2 位有效位。

表 5-7 货币数据类型

数 据 类 型	数据范围描述	存储空间/B
money	−922 337 203 685 477.580 8～ 922 337 203 685 477.580 7	8
smallmoney	−214 748.3648～2 14 748.3647	4

6. 其他数据类型

除了以上基本数据类型外,SQL Server 2012 还支持其他一些数据类型,如表 5-8 所示。

表 5-8　SQL Server 2012 还支持的其他数据类型

数据类型	数据范围描述	存储空间
cursor	包含一个对游标的引用和可以用作变量或存储过程 OUTPUT 参数	不适用
hierarchyid	包含一个对层次结构中位置的引用	1～892 字节＋2 字节的额外开销
sql_variant	可能包含任何系统数据类型的值，除了 text、ntext、image、timestamp、xml、varchar(max)、nvarchar(max)、varbinary (max)、sql _ variant、geography、geometry、hierarchyid、datetimeoffset 以及用户定义的数据类型。最大为 8000 字节数据＋16 字节	8016 字节
table	用于存储结果集以进行后续处理。主要用于返回表值函数的结果集，也可用于函数、存储过程和批处理中	取决于表定义和存储的行数
timestamp or rowversion	rowversion 对于每个表来说是唯一的、自动存储的值。通常用于版本戳，该值在插入和每次更新时自动改变。timestamp 数据类型为 rowversion 数据类型的同义词。在 DDL 语句中，尽量使用 rowversion	8 字节
uniqueidentifier	可以包含全局唯一标识符(Globally Unique Identifier，GUID)。guid 值可以从 Newid()函数获得。这个函数返回的值对所有计算机来说是唯一的	16 字节
xml	存储 XML 数据的数据类型。可以在列中或者 xml 类型的变量中存储 XML 实例	最多 2GB

5.1.2　空值和默认值

当用户往表中插入一行而未对其中的某列指定值时，该列将出现空值(NULL)。空值不同于空白(空字符串)或数值零，通常表示未填写、未知(Unknown)、不可用或将在以后添加的数据。例如，某公司的某份销售订单在初下单时，是无法确定货物的发货日期(send_date)和到货日期(arrival_date)的，故该订单信息在进入数据库时，send_date 和 arrival_date 不能填写，系统将用空值标识该订单记录的这两列。

因为每个空值均为未知，所以没有两个空值是相等的，比较两个空值或将空值与其他任何数值相比均返回未知。空值会对查询命令或统计函数产生影响。实际应用中，应尽量少使用空值，或对查询和数据修改语句进行规划，使空值的影响降到最小。可以使用查询或数据修改语句消除空值或将空值转换成其他值，也可以使用列的默认值约束来避免一些空值。

可通过以下方法在列中插入空值：在 INSERT 或 UPDATE 语句中显式声明 NULL，或不使此列进入 INSERT 语句，或使用 ALTER TABLE 语句在现有表中新添一列。若要判断某列中的值是否为空值，可以使用关键字 IS NULL 或 IS NOT NULL。

默认值是指表中数据的默认取值，默认值对象是数据库的对象不依附于具体的表对象，即默认值对象的作用范围是整个数据库。

5.1.3　约束

约束定义了关于列中允许值的规则，SQL Server 通过限制列中数据、行中数据和表之

间数据来保证数据的完整性。SQL Server 2012 支持非空值约束、默认约束、唯一性约束、主键约束、外键约束等多种约束。

(1) 非空值约束(Not Null)限制数据列不接受 NULL 值,即当对表进行插入(INSERT)操作时,非空值约束的列必须给出确定的值。例如,如果 employee 表的 employee_name(员工姓名)列定义为非空约束,则当录入员工信息时,必须提供员工的姓名。

(2) 默认约束(Default)为数据列定义一个默认值,输入数据时若没有为该列提供值,则将所定义的默认值提供给该列。默认值可以是常量,也可以是表达式。例如,为 employee 表的 hire_date(雇用日期)列定义默认约束表达式"GetDate()"(获取当前日期),将使数据库服务器在用户没有输入时为该列填上默认值,即当天的日期。

(3) 唯一性约束(Unique)限制约束的列,在表的范围内,不允许有两行包含相同的非空值(可以出现多个空值)。

(4) 主键约束(Primary Key)标识列(或列集),这些列(或列集)的值唯一标识表中的行。在一个表中,不能有两行包含相同的主键值,主键的值不能为 NULL。例如,employee 表的 employee_id 列可以选作主键。

每个表都应有一个主键,建议使用一个整数列作为主键。实际应用中,有些表中的数据不便于提供主键列。在 SQL Server 中,通常可以另外建一列并使之成为一个易于使用的主键列。

(5) 外键约束(Foreign Key)也称为外部关键字约束,根据从另一个表中某列(通常是主键列)获得的数据集合来进行有效值判定。这时,被约束列所在的表称为外键表,提供数据的表称为主键表或引用表,提供数据的列称为引用列,所提供的数据称为键值。外键常用来标识表与表之间的关系。

例如,Sales 数据库中的 employee 表、sell_order 表之间存在一种逻辑联系,即 sell_order(销售订单)表的 employee_id(员工编号)列的值必须是 employee 表 employee_id 列中多个值当中的一个,因为签订销售订单的人必须是当前公司员工,因此,在 sell_order 表上应建立外键约束 FK_sell_order_employee 来限制 sell_order 表的 employee_id 列的值必须来自 employee 表的 employee_id 列。

关于约束的详细情况将在第 8 章中进行介绍。

5.2 表的创建与维护

创建表就是定义一个新表的结构以及它与其他表之间的关系。表的维护是指在数据库中创建表以后,对表进行修改、删除等操作。修改表是指更改表结构或表间关系,而删除表是指从数据库中去除该表的表结构、表间关系和表中所有数据。所谓表结构指的是构成表的列、各列的定义(列名、数据类型、数据精度、列上的约束等)和表上的约束。

5.2.1 使用 SQL Server 管理平台对表进行操作

在 SQL Server 管理平台中,表的操作可以可视化完成。管理平台中可以对单个表进行设计,也可以对同一数据库的多个表进行设计,并生成一个或多个关系图,以显示数据库中的部分或全部表、列、键和表间关系。

1. 使用 SQL Server 管理平台创建和修改表

在 SQL Server 管理平台中创建数据表的最常用方法是直接输入字段法。其创建数据表的一般步骤如下：

视频讲解

(1) 打开“对象资源管理器”窗格，打开需要创建表的数据库 Sales，在“表”上右击，在弹出的快捷菜单中选择“新建表”命令，打开表设计器对话框，如图 5-1 所示。

图 5-1　表的创建

(2) 如果要创建 employee 表，在该对话框中，输入雇员表的列名，选择每列的数据类型，设置各列是否允许为空，如图 5-1 所示。列名在一个表中的唯一性是由 SQL Server 强制实现的。每一列都有一个唯一的数据类型，数据类型确定列的精度和长度，可以根据实际的需要进行选择。列允许为空值时将显示“√”，表示该列可以不包含任何数据，空值既不是 0，也不是空字符，而是表示未知，如果不允许列包含空值，则必须为该列提供具体的数据。

(3) 输入完成后，单击工具栏中的“保存”按钮，打开“选择名称”对话框，如图 5-2 所示。输入新建表的名称后，单击“确定”按钮，则创建了一个表。

(4) 若要修改该表，展开“数据库”结点，在需要修改的表上右击，在弹出的快捷菜单中选择“修改”命令，可重新在打开表设计器中进行上述操作。

2. 使用 SQL Server 管理平台设计数据库关系

在 SQL Server 管理平台设计器以图形方式显示部分或全部数据库结构，这种图形被称为数据库关系图。关系图可用来创建和修改表、列、关系、键、索引和约束。可创建一个或更多的关系图，以显示数据库中的部分或全部表、列、键和关系。

图 5-2　“选择名称”对话框

在管理平台中,展开要操作的数据库,选择“数据库关系图”选项,然后右击,在弹出的快捷菜单中选择“新建数据库关系图”命令,如图 5-3 所示。在弹出的对话框中选择要建立关系的表后,则会弹出数据库关系图设计器窗口。

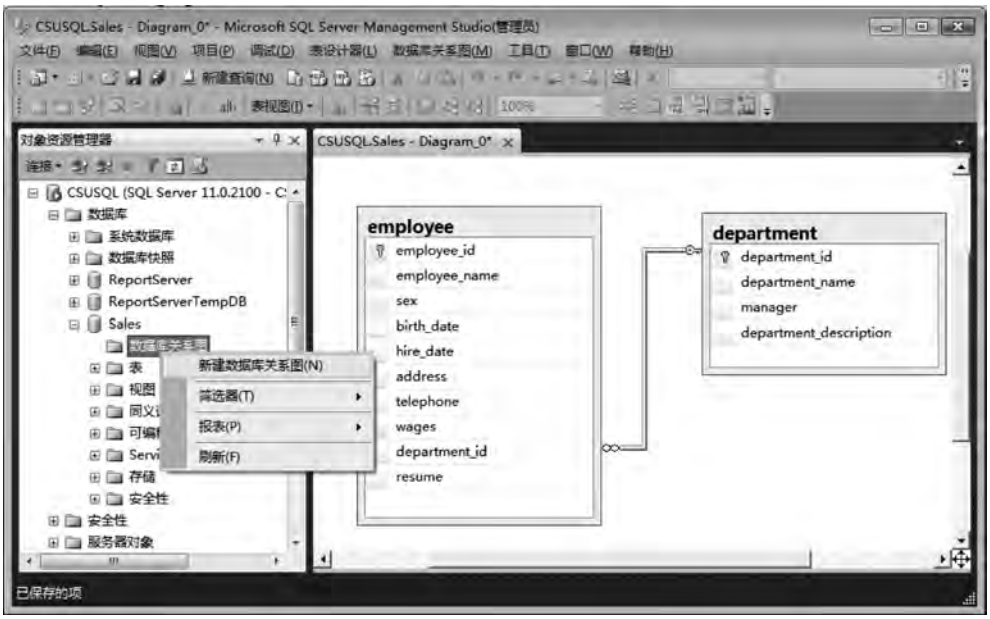

图 5-3　关系图(仅显示表中列名)

图 5-3 是 Sales 数据库中一个简单而典型的关系图。在该关系图中,可以看到表 department 与表 employee (仅显示了列名)由一条连接线联系起来了,这就是这两个表之间的关系,当鼠标移到该连线上时,出现提示框显示该关系的名称信息 FK_employee_department。

在关系图的空白处右击,在弹出的快捷菜单中可以选择新建表或添加数据库中已定义(但未出现在关系图中)的表。关系图中将出现与表设计器上半窗格同样的网格,用以定义新表中各列的基本属性。在该表的级联快捷菜单中选择“属性”选项,可创建或定义该表的关系、键、索引和约束或修改当前列的附加特性。

可以切换表视图以显示表的完整列定义,这样,可以直接在关系图中对表结构进行修改。

在关系图的某个表上右击,在弹出的快捷菜单中,可以选择从关系图或从数据库中删除该表。

3. 在 SQL Server 管理平台中删除表

当某个表不再使用时,就可以将其删除以释放数据库空间。表被删除后,它的结构定义、数据、全文索引、约束和索引都永久地从数据库中删除。表上的规则或默认值将解除绑定,任何与表关联的约束或触发器将自动删除。

在管理平台中可以很方便地删除数据库中已有的表。其操作方法如下:

(1) 在管理平台中右击要删除的表,在弹出的快捷菜单中选择“删除”命令,则会弹出如图 5-4 所示的“删除对象”对话框,单击“确定”按钮即可删除表。

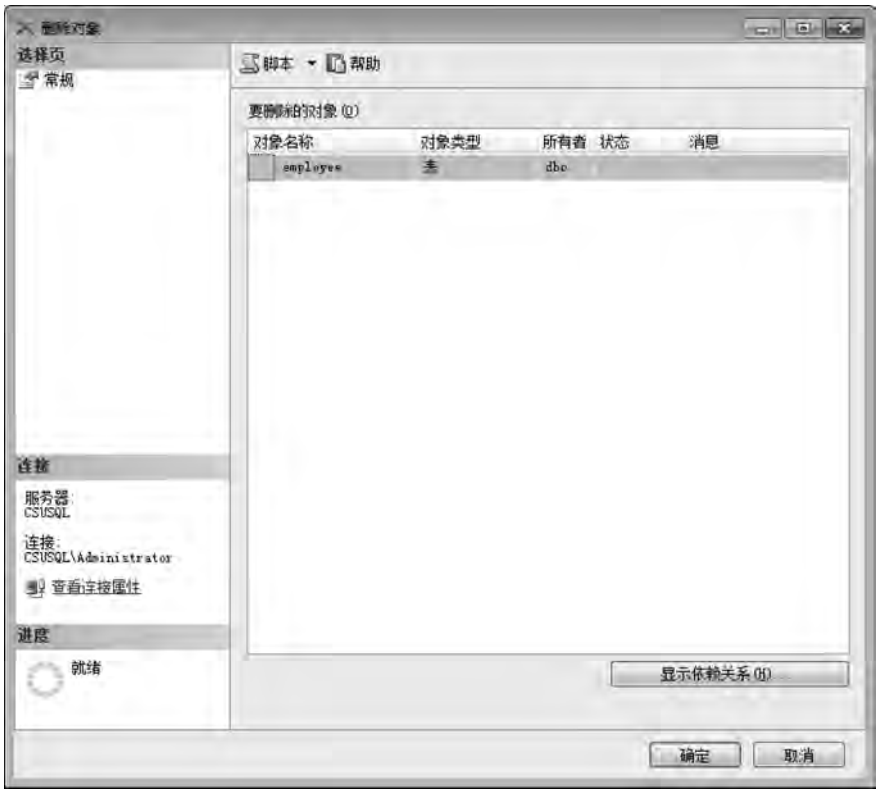

图 5-4 “删除对象”对话框

（2）单击“显示依赖关系”按钮即会出现如图 5-5 所示的对话框，它可以分别列出表所依靠的对象和依赖于表的对象，当有对象依赖关系时就不能删除表。

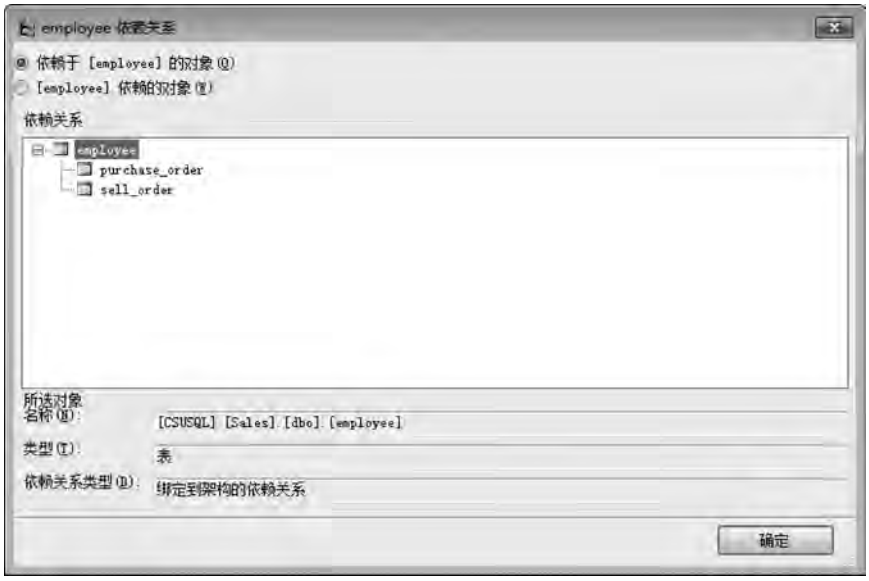

图 5-5 表的依赖关系对话框

5.2.2 使用 Transact-SQL 语句创建表

设计完数据库后就可创建数据库中将存储数据的表。在 Transact-SQL 中,创建表可使用 CREATE TABLE 语句,其语法格式如下:

```
CREATE TABLE
  [database_name. [schema_name].|schema_name.] table_name
  [AS FileTable]
  ( { <column_definition>|<computed_column_definition>}
    [<table_constraint>] [,...,n] )
    [ON { partition_scheme_name ( partition_column_name )|filegroup|"default" }]
    [{ TEXTIMAGE_ON { filegroup|"default" }]
```

各选项的含义如下:

(1) database_name:要在其中创建表的数据库名称,必须是现有数据库的名称,默认为当前数据库。

(2) schema_name:新表所属架构的名称。

(3) table_name:新表的名称。表名必须遵循有关标识符的规则。table_name 最多可包含 128 个字符,本地临时表名(以单个数字符号(#)为前缀的名称)不能超过 116 个字符。

(4) AS FileTable:将新表创建为 FileTable。FileTable 具有固定架构,无须指定列。在 SQL Server 中 FileTable 是一种专用的用户表,可以将文件和文档存储在 FileTable 的表中。

(5) column_definition:表示数据列的语法结构,其语法格式如下。

```
<column_definition> ::=
column_name [type_schema_name.] type_name
    [COLLATE collation_name] [NULL|NOT NULL]
    [[CONSTRAINT constraint_name] DEFAULT constant_expression]
      |[IDENTITY [( seed,increment )] [NOT FOR REPLICATION]]
    [ROWGUIDCOL] [<column_constraint> [,...,n]]
    [SPARSE]
```

其中选项的含义如下。

① column_name [type_schema_name.] type_name:指定列名和存储在该列的数据类型。

- column_name:表中列的名称。列名称必须遵循标识符的规则,且在表中必须唯一。column_name 最多可以有 128 个字符。对于使用 timestamp 数据类型创建的列,可以省略 column_name。如果未指定 column_name,则 timestamp 列的名称默认为 timestamp。
- [type_schema_name.] type_name:指定列的数据类型以及该列所属的架构。

② COLLATE collation_name:指定该列的排序规则。

③ NULL |NOT NULL:确定列中是否允许使用空值。

④ [CONSTRAINT constraint_name] DEFAULT constant_expression:定义约束。各选项含义如下。

- CONSTRAINT:可选关键字,表示 PRIMARY KEY、NOT NULL、UNIQUE、

FOREIGN KEY 或 CHECK 约束定义的开始。

- constraint_name：约束的名称。约束名称必须在表所属的架构中唯一。
- DEFAULT constant_expression：用一个常量表达式设置该列的默认约束。

⑤ IDENTITY [(seed,increment)]：设置该列为标识列，并由 seed 和 increment 分别指定种子和增量(默认都为 1)。

⑥ NOT FOR REPLICATION：指定列的 IDENTITY 列的属性，在把从其他表中复制的数据插入到表中时不发生作用。

⑦ ROWGUIDCOL：指定该列为全局唯一标识符列。

⑧ < column_constraint >：定义在该列上的列约束。取 NULL 或 NOT NULL 时，指定是否在该列上设置非空约束。

⑨ [SPARSE]：指示列为稀疏列。稀疏列已针对 NULL 值进行了存储优化。不能将稀疏列指定为 NOT NULL。

(6) computed_column_definition：某计算列的列定义，定义计算列的值的表达式。计算列并不是物理地存储在表中的虚拟列，除非此列标记为 PERSISTED。该列由同一表中的其他列通过表达式计算得到。表达式可以是非计算列的列名、常量、函数、变量，也可以是用一个或多个运算符连接的上述元素的任意组合。表达式不能是子查询，也不能包含别名数据类型。

在使用计算列时，应注意如下几点：

① 计算列不能作为 INSERT 或 UPDATE 语句的目标。

② 计算列不能用作 DEFAULT 或 FOREIGN KEY 约束定义，也不能与 NOT NULL 约束定义一起使用。

③ 如果计算列由具有确定性的表达式定义，并且索引列中允许计算结果的数据类型，则可将该列用作索引中的键列，或用作 PRIMARY KEY 或 UNIQUE 约束的一部分。

(7) table_constraint：表示对数据表的约束进行设置。

(8) partition_scheme_name (partition_column_name) |filegroup |"default"：指定存储表的分区架构或文件组。各选项含义如下。

- partition_scheme_name：分区架构的名称，该分区架构定义要将已分区表的分区映射到的文件组。
- partition_column_name：表示分区策略依据的列。
- filegroup：表将存储在指定的文件组中。
- "default"：表存储在默认文件组中。

(9) TEXTIMAGE_ON { filegroup |"default"}：指示 text、ntext、image、xml、varchar(max)、nvarchar(max)、varbinary(max)和 CLR(Common Language Runtime)用户定义类型的列存储在指定文件组的关键字。如果表中没有较大值的列，则不允许使用 TEXTIMAGE_ON。如果指定了< partition_scheme >，则不能指定 TEXTIMAGE_ON。如果指定了"default"，或者未指定 TEXTIMAGE_ON，则较大值的列存储在默认文件组中。

【例 5-1】 简单的表定义。

```
USE Sales
CREATE TABLE employee
```

```
( employee_id char(4)NOT NULL,
  employee_name varchar(20)NOT NULL,
  sex char(2)NOT NULL,
  birth_date date NOT NULL,
  hire_date date NOT NULL,
  address varchar(50),
  telephone varchar(12),
  wages money,
  department_id char(4),
  resume text
)
```

本例使用 USE 语句打开 Sales 数据库,使之成为当前数据库,然后在当前数据库创建了 employee 表,所有者为当前用户。

employee 表共有 10 列,使用 char(n)、varchar(n)、date、money 和 text 共 5 种数据类型,并设置其中 5 列采用了非空约束,address、telephone 和 wages、department_id、resume 列默认指定允许为空。

因为未指定 employee 表的所在文件组,该表将被放置在默认文件组中。

【例 5-2】 为表指定文件组。

```
CREATE TABLE Sales.dbo.supplier
(supplier_id char(6)  NOT NULL,
 supplier_name  varchar(50)NOT NULL,
 linkman_name  varchar(20),
 address  varchar(50),
 telephone varchar(12) NOT NULL
)ON [PRIMARY]
```

本例在 Sales 数据库创建表 supplier,所有者为 dbo。该表被显式地放置在 PRIMARY 文件组中。

【例 5-3】 对计算列使用表达式。

```
CREATE TABLE salary
( 姓名 varchar(10),
  基本工资 money,
  奖金 money,
  总计 AS 基本工资 + 奖金)
```

本例创建了 salary 表,定义了 3 个数值列,其中“总计”为计算列,其值由表中另外两列的数据从表达式“基本工资+奖金”计算而来。

【例 5-4】 定义表 autouser 自动获取用户名称。

```
CREATE TABLE autouser
( 编号 int identity(1,1)NOT NULL,
  用户代码 varchar(18),
  登录时间 AS Getdate(),
  用户名 AS User_name()
)
```

本例创建了表 autouser，该表“登录时间”和“用户名”列的信息可以分别通过函数 Getdate()和 User_name()自动获取。当表插入数据时，只需添加“用户代码”数据，其他列的值都自动产生。图 5-6 为表 autouser 插入数据行时表中数据情况。

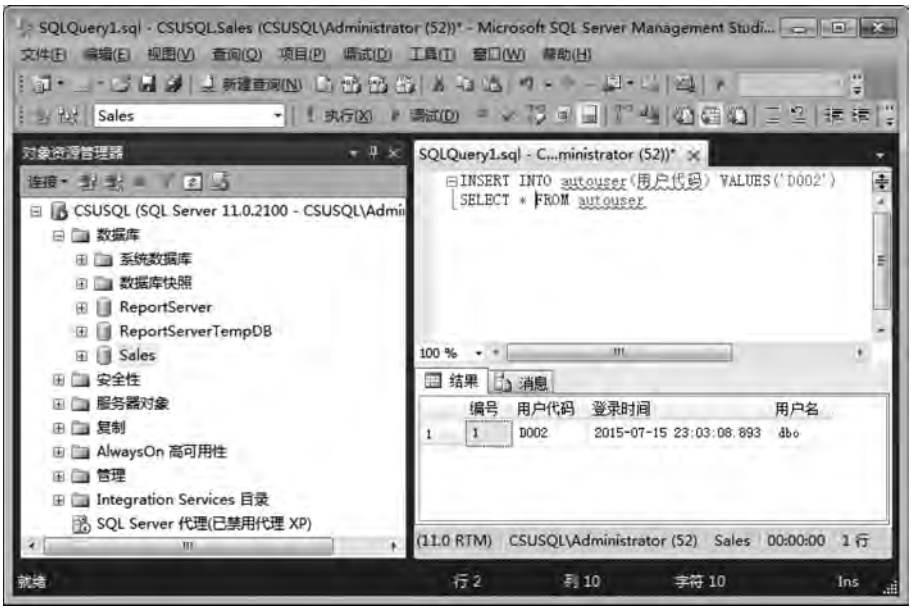

图 5-6　表 autouser 插入数据行时表中数据情况

【例 5-5】 创建临时表。

```
CREATE TABLE #students
( 学号 varchar(8),
  姓名 varchar(10),
  性别 varchar(2),
  班级 varchar(10)
)
```

在实际应用中，经常会用到临时表来暂存储数据。SQL Server 中使用代码创建临时表，需要在表名前加“#”或“##”符号。其中“#”表示本地临时表，在当前数据库内使用，“##”表示全局临时表，可在所有数据库内使用。临时表存储在 tempdb 中。

当用户与 SQL Server 实例断开连接后，将自动删除本地临时表。全局临时表在创建后对任何用户和任何连接都是可见的，当引用该表的所有用户都与 SQL Server 实例断开连接后，全局临时表将自动被删除。

5.2.3　使用 Transact-SQL 语句修改表

在创建数据表之后，经常需要对原先的某些定义进行一定的修改，例如添加、修改、删除列以及添加、删除各种约束。但列的某些数据类型、NULL 值或 IDENTITY 属性不能直接进行修改。SQL Server 使用 ALTER TABLE 语句对数据表的结构进行重新定义。

SQL Server2012 的 ALTER TABLE 语句提供了丰富的参数项，这里列出常用语法结构。其语法格式如下：

```
ALTER TABLE [database_name. [schema_name].|schema_name.] table_name
{ ALTER COLUMN column_name
  { [type_schema_name.]type_name[({precision [,scale]|max| xml_schema_collection } )]
    [COLLATE collation_name] [NULL|NOT NULL] [SPARSE]
      | {ADD|DROP }{ ROWGUIDCOL|PERSISTED|NOT FOR REPLICATION|SPARSE }
  }|[WITH { CHECK|NOCHECK }]
  | ADD
  { <column_definition>|<computed_column_definition>
    |<table_constraint>|<column_set_definition>
  } [,...,n]
  | DROP
  { [CONSTRAINT] constraint_name
    [WITH ( <drop_clustered_constraint_option> [,...,n] )]|COLUMN column_name
} [,...,n]
   |[WITH { CHECK|NOCHECK }] { CHECK|NOCHECK }
    CONSTRAINT { ALL|constraint_name [,...,n] }
   |{ ENABLE|DISABLE } TRIGGER { ALL|trigger_name [,...,n] } }
```

各选项的含义如下：

(1) schema_name：更改表所属架构的名称。

(2) table_name：指定要修改的表名。

(3) ALTER COLUMN column_name：指定表中要更改的列名为 column_name。对列的更改不能与列或表的其他定义相冲突。如某列的默认值为字符串，则数据类型不能更改为非字符串类型，但可先删除默认约束再更改数据类型。以下类型的列不能直接更改。

① 数据类型为 text、image、ntext 或 timestamp 的列。

② 表的 ROWGUIDCOL 列。

③ 计算列或用于计算列中的列。

④ 用于索引中的列。除非该列数据类型是 varchar、nvarchar 或 varbinary，数据类型没有更改，而且新列大小大于或等于旧列大小。

⑤ 用于主键/外键/检查/唯一约束中的列。

⑥ 有默认约束的列的数据类型不能更改，但可更改列的长度、精度或小数位数。

有些数据类型的更改可能导致数据的更改。例如，将数据类型为 nchar 或 nvarchar 的列更改为 char 或 varchar 类型，将导致扩展字符的转换。降低列的精度或小数位数可能导致数据截断。

(4) [type_schema_name.] type_name：更改后的列的新数据类型或添加的列的数据类型。原来的数据类型必须可以隐式转换为新数据类型。如果要更改的列是标识列，新数据类型必须是支持标识属性的数据类型(整型)。新数据类型不能为 timestamp。

(5) precision：指定的数据类型的精度。

(6) scale：指定的数据类型的小数位数。

(7) xml_schema_collection：仅应用于 xml 数据类型，以便将 xml 架构与类型相关联。

(8) COLLATE collation_name：指定更改后的列的新排序规则。

(9) NULL|NOT NULL：指定该列是否可接受空值。

(10) [SPARSE]：指示列为稀疏列。稀疏列已针对 NULL 值进行了存储优化。不能

将稀疏列指定为 NOT NULL。

(11) WITH{CHECK|NOCHECK}：指定表中的数据是否用新添加的或重新启用的 FOREIGN KEY 或 CHECK 约束进行验证。

(12) ADD：指定添加一个或多个列定义、计算列定义或者表约束。

(13) DROP { [CONSTRAINT] constraint_name|COLUMN column_name }：指定从表中删除名为 constraint_name 的约束或者名为 column_name 的列。必须删除所有基于列的索引和约束后，才能删除列。WITH(< drop_clustered_constraint_option >指定设置一个或多个删除聚集约束选项。

(14) {CHECK |NOCHECK } CONSTRAINT：指定启用或禁用 constraint_name。

(15) ALL：指定使用 NOCHECK 选项禁用所有约束，或者使用 CHECK 选项启用所有约束。

(16) { ENABLE |DISABLE } TRIGGER：指定启用或禁用 trigger_name。

(17) trigger_name：指定要启用或禁用的触发器的名称。

【例 5-6】 更改表以添加新列，然后再删除该列。

```
USE Sales
ALTER TABLE employee ADD email varchar(20)NULL
GO
sp_help employee
ALTER TABLE employee DROP COLUMN email
GO
sp_help employee
```

本例先为 employee 表添加了 email 列，通过系统存储过程 sp_help 可以查看修改后的 employee 表的各列，再使用 DROP 子句删除添加的列。

【例 5-7】 将表 employee 的列 address 改为 varchar(150)数据类型，并且不允许为空。

```
ALTER TABLE employee ALTER COLUMN address varchar(150) NOT NULL
GO
```

注意：一定要确认已有的数据中列 address 均不为空后，才能进行此操作。

关于修改表的各种约束操作参见 8.3 节的内容。

5.2.4 使用 Transact-SQL 语句删除表

表所有者可以使用 Transact-SQL 语句删除任何其所有的表。如果不想等待临时表自动删除，则可明确删除临时表。

删除表的 Transact-SQL 语句格式如下：

```
DROP TABLE [database_name. [schema_name]. |schema_name. ]
  table_name [,...,n] [;]
```

各选项的含义如下。

(1) database_name：要在其中创建表的数据库的名称。

(2) schema_name：表所属架构的名称。

(3) table_name：要删除的表的名称。

注意：

(1) 删除表时，表上的规则或默认值将解除绑定，任何与表关联的约束或触发器将自动除去。如果重新创建表，则必须重新绑定适当的规则和默认值，重新创建任何触发器并添加必要的约束。

(2) 不能使用 DROP TABLE 删除被 FOREIGN KEY 约束引用的表。必须先删除引用 FOREIGN KEY 约束或引用表。如果要在同一个 DROP TABLE 语句中删除引用表以及包含主键的表，则必须先列出引用表。

(3) 系统表不能使用 DROP TABLE 语句删除。

【例 5-8】 删除当前数据库内的表。

```
USE Sales
GO
DROP TABLE employee
```

本例从当前数据库 Sales 中删除 employee 表及其数据和索引。

【例 5-9】 删除另外一个数据库内的表。

```
DROP TABLE Sales.dbo.employee
```

本例删除 Sales 数据库内的 employee 表。可以在服务器实例上的任何数据库内执行此操作。

5.3 表中数据的维护

视频讲解

数据库的主要用途是存储数据并使授权的应用程序和用户能够使用这些数据。在数据库中的表对象建立后，用户对表的访问可以归纳为 4 个基本操作：添加或插入新数据、检索现有数据、更改或更新现有数据和删除现有数据。

这 4 种操作通常称为 CRUD(Create，Retrieve/Read，Update，Delete)操作。其中的 R 操作，即对数据表的检索，通常也被称为查询(Query)，将在第 6 章详细介绍，本节只讨论其余 3 种操作。

对表中数据进行维护也有两种方法：一是使用 SQL Server 管理平台；二是使用 Transact-SQL 语句。同前面介绍管理平台的操作类似，在管理平台中，右击需要操作的表，在弹出的快捷菜单中选择“打开表”命令，再选择有关命令，即可完成查询、修改和删除表中数据的操作。下面重点介绍表中数据维护的 Transact-SQL 语句。

5.3.1 插入数据

INSERT 语句可向表中添加一个或多个新行，其语法格式如下：

```
[WITH <common_table_expression> [,...,n]]
INSERT [TOP ( expression ) [PERCENT]]
        [INTO]
    { [server_name. database_name. schema_name. |database_name.[schema_name].
      |schema_name. ] table_or_view_name
```

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

各选项的含义如下：

(1) WITH < common_table_expression >：指定在 INSERT 语句作用域内定义的临时命名结果集(也称为公用表表达式)。结果集源自 SELECT 语句。

(2) TOP(expression)[PERCENT]：指定将插入的随机行的数目或百分比。expression 可以是行数或行的百分比。在和 INSERT、UPDATE 或 DELETE 语句结合使用的 TOP 表达式中引用的行不按任何顺序排列。

(3) INTO：一个可选的关键字，可以将它用在 INSERT 和目标表之间。

(4) server_name：表或视图所在服务器的名称。如果指定了 server_name，则需要 database_name 和 schema_name。

(5) database_name：数据库的名称。

(6) schema_name：表或视图所属架构的名称。

(7) table_or view_name：要接收数据的表或视图的名称。

(8) (column_list)：要在其中插入数据的一列或多列的列表。必须用括号将 column_list 括起来，并且用逗号进行分隔。

(9) OUTPUT 子句：将插入行作为插入操作的一部分返回。引用本地分区视图、分布式分区视图或远程表的 DML 语句，或包含 execute_statement 的 INSERT 语句，都不支持 OUTPUT 子句。在包含< dml_table_source >子句的 INSERT 语句中不支持 OUTPUT INTO 子句。

(10) VALUES：引入要插入的数据值的列表。

如果 VALUES 列表中的各值与表中各列的顺序不相同，或者未包含表中各列的值，则必须使用 column_list 显式指定存储每个传入值的列。

若要插入多行值，VALUES 列表的顺序必须与表中各列的顺序相同，且此列表必须包含与表中各列或 column_list 对应的值以便显式指定存储每个传入值的列。

(11) DEFAULT：强制数据库引擎加载为列定义的默认值。若某列不存在默认值，并且该列允许 NULL 值，则插入 NULL。DEFAULT 对标识列无效。

(12) expression：一个常量、变量或表达式。表达式不能包含 EXECUTE 语句。

(13) derived_table：任何有效的 SELECT 语句，它返回将加载到表中的数据行。

(14) execute_statement：任何有效的 EXECUTE 语句，它使用 SELECT 或 READTEXT 语句返回数据。

(15) DEFAULT VALUES：强制新行包含为每个列定义的默认值。

【例 5-10】 使用简单的 INSERT 语句。

```
USE Sales
GO
INSERT Supplier
  VALUES ('R001','华科电子有限公司','施宾彬  ','朝阳路 56 号','2636565')
```

本例在例 5-2 创建的表 Supplier 中插入一行。由于省略了列表，默认对表中所有列按

顺序填入。

另外,如果将值装载到可变长度数据类型的列时,尾随空格(对 varchar/nvarchar 是空格,对 varbinary 是零)将被删除。例如本例中向 linkman_name 列(varchar(20))填入值"施宾彬 ",SQL Server 将"施宾彬"填入该列并删除其尾随的空格。

而如果向 char/nchar 类型的列填入的值长度小于列的定义长度时,SQL Server 将对填入值向右填充至定义长度。例如本例中向 supplier_id 列(char(6))填入 R001 后,系统将增加两个空格在该字符串右边以达到定义长度 6。

【例 5-11】 显式指定列列表。

```
INSERT Sales.dbo.supplier
  (supplier_id,supplier_name,linkman_name,address,telephone)
  VALUES ('R00015','华科电子有限公司','施宾彬','朝阳路 56 号','2636565')
```

本例与例 5-10 的 INSERT 命令功能完全等同。但在本例中表中各列被显式地列出来。显式指定列列表还可用来插入值少于列个数的数据或插入与列顺序不同的数据。例如:

```
-- 插入值少于列个数的数据
INSERT Sales.dbo.supplier (supplier_id,supplier_name,telephone)
  VALUES ('R00016','晨光电子实业公司','4561681')
-- 插入与列顺序不同的数据
INSERT Sales.dbo.Supplier(Supplier_name,telephone,Supplier_id)
  VALUES ('晨光电子实业公司','4561681','R00016')
```

【例 5-12】 将数据装载到带有标识符的表。

```
-- 创建 customer2 表,该表的 customer_id 为标识列
CREATE TABLE customer2
(customer_id bigint NOT NULL
 IDENTITY(0,1),
 customer_name varchar(50)NOT NULL,
 linkman_name varchar(20),
 address varchar(50),
 telephone char(12)NOT NULL)
GO
-- 以下语句为 customer2 表插入数据
INSERT customer2
  VALUES ('东方体育用品公司','刘平','东方市中山路 25 号','75368025')
INSERT customer2(customer_name,linkman_name,address,telephone)
VALUES ('北京泛亚实业公司','张卫民','长岭市五一路号','68510231')
SET IDENTITY_INSERT Sales.dbo.customer2 ON
INSERT customer2
(customer_id,customer_name,linkman_name,address,telephone)
VALUES ('2','洞庭强华电器公司','马东','滨海市洞庭大道号','76053331')
```

本例创建了表 customer2,其 customer_id 被定义为标识列。第 1、2 个 INSERT 语句允许系统为新行生成标识值。执行第 3 个 INSERT 语句前用 SET IDENTITY_INSERT Sales.dbo.customer2 ON 语句允许标识列的手动插入,并且将一个显式的值(2)插入到标识列。撤销标识列的手动插入使用语句 SET IDENTITY_INSERT。Sales.dbo.customer2 OFF。

【例 5-13】 使用 SELECT 和 EXECUTE 选项装载数据。

```
-- 创建一个新表 newcustomer
CREATE TABLE Sales.dbo.newcustomer
(customerName varchar(50)NOT NULL,
linkmanName varchar(20)
)
-- 用 INSERT…SELECT 从 customer2 表查询数据填入 NewCustomer 表
INSERT newcustomer
  SELECT customer_name,linkman_name FROM customer2
-- 先创建一个存储过程(详细内容参见第 10 章),再使用 INSERT…EXECUTE 语句
-- 从 customer2 表用存储过程查询数据填入 newcustomer 表
CREATE PROCEDURE MySp_Customer
AS
SELECT customer_name,linkman_name FROM customer2
GO
INSERT newcustomer
  EXECUTE MySp_Customer
-- 用 INSERT…EXECUTE('string')从 customer2 表查询数据填入 newcustomer 表
INSERT newcustomer
EXECUTE('SELECT customer_name,linkman_name FROM customer2')
```

本例演示了 3 种不同的方法,用来从 customer2 表获取数据,并将数据填入 newcustomer 表。每种方法都基于一个 SELECT 语句,该语句从 customer2 表查询其中两列的值。

第 1 个 INSERT 语句使用 SELECT 语句直接从源表(customer2)检索获取数据;第 2 个 INSERT 语句执行一个包含 SELECT 语句的存储过程;第 3 个 INSERT 语句执行以字符串形式表示的 SELECT 语句。

5.3.2 修改数据

在创建表并添加数据之后,更改或更新表中的数据就成为维护数据库的一个日常过程。UPDATE 语句可以更改表或视图中单行、行组或所有行的数据值。其语法格式如下:

```
[WITH <common_table_expression> [,...,n]]
UPDATE [TOP ( expression ) [PERCENT]]
  {{ table_alias|{[server_name. database_name. schema_name.|database_name.[schema_name].
  | schema_name.] table_or_view_name }}| @table_variable}
  SET { column_name = { expression|DEFAULT|NULL }
       | { udt_column_name.{ { property_name = expression
       | field_name = expression }|method_name ( argument [,...,n] ) } }
       | column_name {.WRITE ( expression, @Offset, @Length ) }
       | @variable = expression
       | @variable = column = expression } [,...,n]
  [<OUTPUT Clause>] [FROM{ <table_source> } [,...n,]]
  [WHERE { <search_condition> }]
```

各选项的含义如下:

(1) WITH <common_table_expression>:指定在 UPDATE 语句作用域内定义的临时命名结果集或视图,也称为公用表表达式(CTE)。CTE 结果集派生自简单查询并由 UPDATE 语句引用。

(2) TOP (expression) [PERCENT]：指定将要更新的行数或行百分比。expression可以为行数或行百分比。

(3) table_alias：在表示要从中更新行的表或视图的 FROM 子句中指定的别名。

(4) server_name：表或视图所在服务器的名称(使用链接服务器名称或 OPENDATASOURCE 函数作为服务器名称)。如果指定了 server_name，则需要 database_name 和 schema_name。

(5) database_name：数据库的名称。

(6) schema_name：表或视图所属架构的名称。

(7) table_or_view_name：要更新行的表或视图的名称。

(8) @ table_variable：将表变量指定为表源。

(9) SET：指定要更新的列或变量名称的列表。

(10) column_name：包含要更改的数据的列。column_name 必须已存在于 table_or_view_name 中。不能更新标识列。

(11) expression：返回单个值的变量、文字值、表达式或嵌套 SELECT 语句(加括号)。expression 返回的值替换 column_name 或@variable 中的现有值。

(12) DEFAULT：指定用为列定义的默认值替换列中的现有值。如果该列没有默认值并且定义为允许 NULL 值，则该参数也可用于将列更改为 NULL。

(13) udt_column_name：用户定义类型列。

(14) property_name | field_name：用户定义类型的公共属性或公共数据成员。

(15) method_name (argument [,...,n])：带一个或多个参数的 udt_column_name 的非静态公共赋值函数方法。

(16) WRITE (expression, @Offset, @Length)：指定修改 column_name 值的一部分。expression 替换@Length 单位(从 column_name 的@Offset 开始)。只有 varchar(max)、nvarchar(max)或 varbinary(max)列才能使用此子句来指定。column_name 不能为 NULL，也不能由表名或表别名限定。

(17) @ variable：已声明的变量，该变量将设置为 expression 所返回的值。

(18) < OUTPUT Clause >：在 UPDATE 操作中，返回更新后的数据或基于更新后的数据的表达式。针对远程表或视图的任何 DML 语句都不支持 OUTPUT 子句。

(19) FROM{< table_source >}：指定将表、视图或派生表源用于为更新操作提供条件。

(20) WHERE：指定条件来限定所更新的行。

(21) < search_condition >：为要更新的行指定需满足的条件。

【例 5-14】 使用简单的 UPDATE 语句。

```
UPDATE customer2 SET linkman_name = '佚名', address = NULL, telephone = ''
```

本例将所有客户单位的联系人设置为“佚名”，地址设置为空值，电话号码设置为空字符串(非空值)。

也可以在更新中使用计算值。下面的 SQL 语句将表 salary 中的“奖金”加倍。

```
UPDATEsalary SET 奖金 = 奖金 * 2
```

【例 5-15】 在 UPDATE 语句中使用 WHERE 子句。

```
UPDATE customer2
  SET telephone = '0731 - ' + telephone WHERE LEN(telephone) = 8
```

本例将 customer2 表中的所有本地电话号码前加上区号。判断本地电话号码的逻辑条件是：列 telephone 的字符串长度为 8(LEN()函数返回字符串长度)。

【例 5-16】 在 UPDATE 语句中使用来自另一个表的信息。

首先创建 sell_order 表和 goods 表(参考表 2-12 和表 2-13)并输入适量数据，这一步请读者自己完成。然后执行下列命令。

```
UPDATE sell_order
SET cost =
sell_order.order_num * goods.unit_price * (1 - sell_order.discount)
FROM sell_order,goods
WHERE sell_order.goods_id = goods.goods_id
```

本例修改表 sell_order 中的 cost 列，以计算每个销售订单的收费。也可以使用表的别名改写为一个较简单的形式。表 sell_order 的别名为 so，goods 的别名为 g。

```
UPDATE sell_order
  SET cost = so.order_num * g.unit_price * (1 - so.discount)
  FROM sell_order so,goods g
  WHERE so.goods_id = g.goods_id
```

【例 5-17】 在 UPDATE 语句中使用 SELECT…TOP 语句。

```
UPDATE goods
SET unit_price = unit_price * 0.9
FROM goods,
(SELECT TOP 10 goods_ID, sum(order_Num)AS total_num FROM sell_order
GROUP BY goods_ID
ORDER BY total_num ASC
) AS total_sum
WHERE goods.goods_id = total_sum.goods_id
```

本例使用 SELECT…TOP 语句的结果集作为依据对 goods 表进行更新。SELECT 查询返回销售量最少的 10 件货物的编号。WHERE 子句使用查询结果作为搜索条件对 goods 表进行过滤。SET 子句对满足条件的货物的单价进行更改。整个 UPDATE 语句的实际意义是：对销售状况最差的 10 件商品进行降价。

5.3.3 删除数据

Transact-SQL 支持两种删除现有表中数据的语句，分别是 DELETE 和 TRUNCATE TABLE 语句。

1. DELETE 语句

DELETE 语句可删除表或视图中的一行或多行，每一行的删除都将被记入日志。DELETE 语句的语法格式如下：

```
[WITH < common_table_expression > [,...,n]]
DELETE [TOP ( expression ) [PERCENT]]
```

```
[FROM]
{{ table_alias|{ [server_name.database_name.schema_name.
 | database_name. [schema_name].|schema_name.]
table_or_view_name}}| @table_variable}
[< OUTPUT Clause >]
[FROM < table_source > [,...,n]]
[WHERE { < search_condition >}]
```

各选项的含义如下：

(1) WITH < common_table_expression >：指定在 DELETE 语句作用域内定义的临时命名结果集，也称为公用表表达式。结果集源自 SELECT 语句。

(2) TOP(expression)[PERCENT]：指定将要删除的任意行数或任意行的百分比。expression 可以为行数或行的百分比。与 INSERT、UPDATE 或 DELETE 一起使用的 TOP 表达式中被引用行将不按任何顺序排列。

(3) FROM：可选的关键字，可用在 DELETE 关键字与目标 table_or_view_name 之间。

(4) server_name：表或视图所在服务器的名称。如果指定了 server_name，则需要 database_name 和 schema_name。

(5) database_name：数据库的名称。

(6) schema_name：该表或视图所属架构的名称。

(7) table_or view_name：要删除行的表或视图的名称。

(8) < OUTPUT Clause >：将已删除行或基于这些行的表达式作为 DELETE 操作的一部分返回。

(9) FROM < table_source >：指定附加的 FROM 子句。这个对 DELETE 的 Transact-SQL 扩展允许从< table_source >指定数据，并从第一个 FROM 子句内的表中删除相应的行。这个扩展指定联接，可在 WHERE 子句中取代子查询来标识要删除的行。

(10) WHERE：指定用于限制删除行数的条件。如果没有提供 WHERE 子句，则 DELETE 删除表中的所有行。

(11) < search_condition >：指定删除行的限定条件。

如果 DELETE 删除了多行，而在删除的行中有任何一行违反约束(主要是外键约束)或触发器，则将取消该语句，返回错误且不删除任何行。如果 DELETE 语句执行中出现了表达式算术错误(溢出、被零除或域错误)时，SQL Server 将取消批处理中的其余部分并返回错误信息。

【例 5-18】 不带参数使用 DELETE 命令删除所有行。

```
USE Sales
GO
DELETE customer2
```

本例从 customer2 表中删除所有行。

注意：将 DELETE 语句与 DROP TABLE 语句的功能区分开。

【例 5-19】 带 WHERE 子句的 DELETE 语句，有条件地删除行。

```
DELETE FROM sell_order WHERE custom_id = 'C00006'
```

本例删除 sell_order 表中 custom_id 是 C00006 的所有行。

【例 5-20】 在 DELETE 中使用连接或子查询。

```
-- 使用 INNER JOIN 进行表连接,然后在表 sell_order 中删除以"东方市"开头的客户的销售订单
DELETE sell_order
  FROM sell_order so INNER JOIN customer2 c   ON so.customer_id = c.customer_id
  WHERE C.address LIKE'东方市 % '
-- 等同于下列命令
DELETE sell_order
  FROM sell_order so,customer2 c
  WHERE so.customer_id = c.customer_id   AND c.address LIKE'东方市 % '
-- 使用嵌套子查询查找以"东方市"开头的客户的 customer_id,然后在表 sell_order 中删除这些
-- 客户的销售订单
DELETE FROM sell_order
  WHERE customer_id IN
    (SELECT customer_id FROM customer2 WHERE address LIKE'东方市 % ')
```

本例采用三个语句分别演示了基于联接或子查询从基表中删除记录。三个语句实现同样的功能,即将所有地址(address)以“东方市”开头的客户的销售订单从 sell_order 表中删除。

2. TRUNCATE TABLE 语句

TRUNCATE TABLE 语句可一次删除表中的所有行,而不会把每一行的删除操作都记入日志。所以 TRUNCATE TABLE 语句是一种快速清空表的方法。其语法格式如下:

```
TRUNCATE TABLE
  [{ database_name.[schema_name].|schema_name. }] table_name
```

各选项的含义如下:

(1) database_name:数据库的名称。

(2) schema_name:表所属架构的名称。

(3) table_name:要删除其全部行的表的名称。

注意:

(1) TRUNCATE TABLE 语句在功能上与不带 WHERE 子句的 DELETE 语句相同,两者均删除表中的全部行。

(2) DELETE 语句每次删除一行,并在事务日志中进行一次记录,而 TRUNCATE TABLE 通过释放存储表数据所用的数据页来删除数据,并且在事务日志中只记录页的释放。所以 TRUNCATE TABLE 比 DELETE 速度快,且使用的系统和事务日志资源少。

(3) TRUNCATE TABLE 删除表中的所有行,但表结构及其列、约束、索引等保持不变。

(4) TRUNCATE TABLE 使对新行标识符列(IDENTITY)所用的计数值重置为该列的种子。如果想保留标识计数值,可改用 DELETE。

(5) 对于被外键约束所引用的表,不能使用 TRUNCATE TABLE,而应使用不带 WHERE 子句的 DELETE 语句。由于 TRUNCATE TABLE 不记录行删除日志,所以它

不能激活触发器。

【例 5-21】 使用 TRUNCATE TABLE 语句清空表。

```
TRUNCATE TABLE customer2
```

本例清空 customer2 表中的所有数据,并使 customer_id 列(bigint,IDENTITY(0,1))的新行标识重置为种子(整数值 0)。

本章小结

本章首先介绍了 SQL Server 中与表相关的一些知识,包括数据库中的表、数据类型和约束。然后,从表结构的维护出发,介绍了在 SQL Server 2012 管理平台中创建表、修改表和删除表的操作方式,以及使用 Transact-SQL 语句来完成这些工作的方法。最后,从表中数据的维护出发,介绍了使用 Transact-SQL 语句来对表中数据进行新增、修改和删除的方法。

(1) 表的相关概念:表是数据库中数据的实际存储位置,一个表代表一个实体。表由行和列组成,每行标识实体的一个个体,每列代表实体的一个属性。

(2) 数据类型:数据类型描述并约束了列中所能包含的数据的种类、所存储值的长度或大小、数字精度和小数位数(对数值数据类型)。

(3) 空值:未对列指定值时,该列将出现空值。空值不同于空字符串或数值零,通常表示未知。空值会对查询命令或统计函数产生影响,应尽量少使用空值。

(4) 约束:约束是数据库自动保持数据完整性的机制,它是通过限制列中数据、行中数据和表之间数据来保持数据完整性。SQL Server 2012 支持 Not Null、Default、Check、Primary Key、Foreign Key、Unique 6 种约束。关于约束的操作将在第 8 章中详细介绍。

(5) 可以使用 SQL Server 2012 管理平台和 Transact-SQL 语句创建表并对表进行维护,包括修改和删除等操作。

(6) 可以使用 SQL Server 2012 管理平台和 Transact-SQL 语句对表中数据进行编辑,包括插入、更新和删除等操作。

习 题 5

一、选择题

1. 在定义数据表的字段时,“允许 NULL 值”用于设置该字段是否可输入空值,实际上就是创建该字段的(　　)约束。

A. 主键　　B. 外键　　C. 非空　　D. CHECK

2. 下列关于主关键字的叙述正确的是(　　)。

A. 一个表可以没有主关键字

B. 只能将一个字段定义为主关键字

C. 如果一个表只有一个记录,则主关键字字段可以为空值

D. 以上选项都正确

3. 使用 CREATE TABLE 语句创建数据表时(　　)。

A. 必须在数据表名称中指定表所属的数据库

B. 必须指明数据表的所有者

C. 指定的所有者和表名称组合起来在数据库中必须唯一

D. 省略数据表名称时,则自动创建一个本地临时表

4. 下列关于 ALTER TABLE 语句的叙述错误的是(　　)。

A. ALTER TABLE 语句可以添加字段

B. ALTER TABLE 语句可以删除字段

C. ALTER TABLE 语句可以修改字段名称

D. ALTER TABLE 语句可以修改字段数据类型

5. 若要删除数据库中已经存在的表 A,则可用(　　)。

A. DELETE TABLE A　　B. DELETE A

C. DROP TABLE A　　D. DROP A

二、填空题

1. 整数型的 int 型数的范围为_____,整数型的 tinyint 型数的范围为_____。

2. 表中某列为变长字符数据类型 varchar(100),其中 100 表示_____。假如输入的字符串为 gtym13e5,存储的字符长度为_____字节。

3. SQL Server 支持的基本数据类型有字符和二进制数据类型、_____数据类型、逻辑数据类型、_____数据类型,用于各类数据值的存储、检索和解释。

4. ALTER TABLE 语句不能修改数据表的_____和_____。

5. Transact-SQL 中添加记录使用语句_____,修改记录使用语句_____,删除记录可使用_____或_____语句。

三、问答题

1. 简述下列数据类型中每一组之间的区别。

```
char/varchar  char/nchar  decimal/float/money
```

2. 在 Sales 数据库的 Sell_Order 表中 cost(费用)列允许为空。当一个 NULL 值和一个货币类型数据被填入到某两行的 cost 列后,试比较这两个 cost 值的大小。

3. 简述以下 3 条 SQL 语句的异同。

```
DROP TABLE Orders
DELETE Orders
TRUNCATE TABLE Orders
```

四、应用题

分别使用一条 Transact-SQL 语句完成下列操作。

(1) 在 Sales 数据库中创建销售订单表 Sell_Order(包含所有列,只含非空约束),其中销售订单编号(order_id1)为标识符列(1,2)。

(2) 在该表中删除列 send_date,增加列“发货日期”。

(3) 往表中插入一行,以记录这个销售事件:2019 年 2 月 26 日,编号为 99 的客户从本公司订购了 30 件编号为 135 的货物;编号为 16 的员工洽谈了该业务并给予该客户 95 折优惠。

(4) 往表中插入一行,以记录这个销售事件:2018 年 10 月 10 日,编号为 6 的客户从本公司订购了 200 件编号为 26 的货物;编号为 02 的员工洽谈了该业务并给予该客户 8 折优惠。该批货物已于 2018 年 12 月 1 日,由编号为 10 的运输商承运,客户已于 2018 年 12 月 12 日验收,并付清了费用人民币 200 000 元整。

(5) 因编号为 29 的员工辞职,将他所有未结账(cost 未计算)的销售订单转交编号为 15 的员工处理。

(6) 因编号为 100 的客户升级为本公司 VIP 客户,其所有未结账订单的折扣在原折扣上再进行 9 折。

(7) 将所有发生于 2019 年 1 月 1 日的销售订单删除。