第 3 章 SQL 语言基础及数据定义功能

用户使用数据库时需要对数据库进行各种各样的操作，如查询数据，添加、删除和修改数据，定义、修改数据模式等。DBMS 必须为用户提供相应的命令或语言，这就构成了用户和数据库的接口。接口的好坏会直接影响用户对数据库的接受程度。

数据库所提供的语言一般局限于对数据库的操作，它不是完备的程序设计语言，也不能独立地用来编写应用程序。

SQL（Structured Query Language，结构化查询语言）是用户操作关系数据库的通用语言。SQL 虽然叫结构化查询语言，而且查询操作确实是数据库中的主要操作，但并不是说 SQL 只支持查询操作，它实际上包含数据定义、数据操纵和数据控制等与数据库有关的全部功能。

SQL 已经成为关系数据库的标准语言，所以现在所有的关系数据库管理系统，包括小型数据库管理系统（如 Access）都支持 SQL 语言，只是不同的系统支持的 SQL 语言功能有所区别。本章首先介绍一般 SQL 语言支持的数据类型，然后介绍 SQL 语言中的数据定义功能。

3.1 基本概念

SQL 语言是操作关系数据库的标准语言，本节介绍 SQL 语言的发展过程、特点以及主要功能。

3.1.1 SQL 语言的发展

最早的 SQL 原型是 IBM 的研究人员在 20 世纪 70 年代开发的，该原型被命名为 SEQUEL（由 Structured English QUEry Language 的首字母缩写组成）。现在许多人仍将在这个原型之后推出的 SQL 语言发音为“sequel”，但根据 ANSI SQL 委员会的规定，其正式发音应该是“ess cue ell”。随着 SQL 语言的颁布，各数据库厂商纷纷在他们的产品中引入并支持 SQL 语言，但尽管绝大多数产品对 SQL 语言的支持大部分是相似的，但它们之间也存在着一定的差异，这些差异不利于初学者的学习。因此，本章主要介绍标准的 SQL 语言，

即基本 SQL。

从 20 世纪 80 年代以来，SQL 就一直是关系数据库管理系统（RDBMS）的标准语言。最早的 SQL 标准是 1986 年 10 月由美国 ANSI（American National Standards Institute）颁布的。随后，ISO（International Standards Organization）于 1987 年 6 月也正式采纳它为国际标准，并在此基础上进行了补充，到 1989 年 4 月，ISO 提出了具有完整性特征的 SQL，并称之为 SQL-89，SQL-89 标准的颁布对数据库技术的发展和数据库的应用都起了很大的推动作用。尽管如此，SQL-89 仍有许多不足或不能满足应用需求的地方。为此，在 SQL-89 的基础上，经过 3 年多的研究和修改，ISO 和 ANSI 共同于 1992 年 8 月又颁布了 SQL 的新标准，即 SQL-92（或称为 SQL2）。SQL-92 标准也不是非常完备的，1999 年又颁布了新的 SQL 标准，称为 SQL-99 或 SQL3。

3.1.2 SQL 语言特点

SQL 之所以能够被用户和业界所接受并成为国际标准，是因为它是一个综合的、功能强大的且比较简洁易学的语言。SQL 语言集数据查询、数据操纵、数据定义和数据控制功能于一身，其主要特点包括：

1. 一体化

SQL 语言风格统一，可以满足数据库应用的全部需要，包括创建数据库、定义模式、更改和查询数据以及安全控制和维护数据库等。这为数据库应用系统的开发提供了良好的环境。用户在数据库系统投入使用之后，还可以根据需要随时修改模式结构，并且可以不影响数据库的运行，从而使系统具有良好的可扩展性。

2. 高度自动化

在使用 SQL 语言访问数据库时，用户没有必要告诉计算机“如何”一步步地实现操作，而只需要描述清楚要“做什么”，SQL 语言就可以将要求提交给数据库管理系统，然后由数据库管理系统自动完成全部工作。

3. 简洁

虽然 SQL 语言功能很强，但它只有为数不多的几条命令，另外，SQL 的语法也比较简单，比较接近自然语言（英语），因此容易学习、掌握。

4. 能以多种方式使用

SQL 语言可以直接以命令方式交互使用，也可以嵌入程序设计语言中使用。现在很多数据库应用开发工具（比如 Visual Basic、PowerBuilder、C# 等）都将 SQL 语言直接融入自身的语言当中，使用起来非常方便。这些使用方式为用户提供了灵活的选择余地。而且不管是哪种使用方式，SQL 语言的语法基本都是一样的。在本书的第 III 篇，Visual Basic 数据库应用编程中可以看到这点。

3.1.3 SQL 语言功能概述

SQL 按其功能可分为四大部分：数据定义功能、数据控制功能、数据查询功能和数据操纵功能。表 3-1 列出了实现这四部分功能的动词。

表 3-1 SQL 包含的主要动词

SQL 功能	动　词
数据定义	CREATE、DROP、ALTER
数据查询	SELECT
数据操纵	INSERT、UPDATE、DELETE
数据控制	GRANT、REVOKE、DENY

数据定义用于定义、删除和修改数据库中的对象，包括关系表、视图等；数据查询用于查询数据，是数据库中使用最多的操作；数据操纵用于增加、删除和修改数据库数据；数据控制用于控制用户对数据库的操作权限。

本章介绍数据定义功能中定义关系表的 SQL 语句，同时介绍在定义表时如何实现数据的完整性约束。在第 4 章介绍实现数据查询和数据操纵功能的 SQL 语句，在第 11 章介绍实现数据控制功能的语句，创建数据库的语句在第 10 章介绍。在介绍这些功能之前，首先介绍 SQL 语言所支持的数据类型。因为本书是以 SQL Server 2012 作为实践平台，因此这里主要介绍的是 SQL Server 2012 提供的数据类型。

3.2　SQL Server 提供的主要数据类型

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

每个数据库厂商提供的数据库管理系统所支持的数据类型并不完全相同，而且与标准的 SQL 也有差异，本书主要介绍 Microsoft SQL Server 支持的常用数据类型。

SQL Server 的数据类型类别如表 3-2 所示。

表 3-2　SQL Server 提供的数据类型类别

数 据 类 型	类　别　名
精确数字类型	Unicode 字符串类型
近似数字类型	二进制字符串类型
日期和时间类型	其他数据类型
字符串类型	

下面介绍常用的数据类型。

3.2.1　数字类型

1. 精确数字类型

精确数字类型是指在计算机中能够精确存储的数据，比如整型、定点小数等都是精确数字类型。表 3-3 列出了 SQL Server 支持的整型类型。

表 3-3　整型类型

数 据 类 型	范　　围	存　储
bigint	存储从 -2^{63}（−9 223 372 036 854 775 808）到 $2^{63}-1$（9 223 372 036 854 775 807）范围的整数	8 字节
int	存储从 -2^{31}（−2 147 483 648）到 $2^{31}-1$（2 147 483 647）范围的整数	4 字节
smallint	存储从 -2^{15}（−32 768）到 $2^{15}-1$（32 767）范围的整数	2 字节
tinyint	存储从 0 到 255 之间的整数	1 字节
bit	存储 1、0 或 NULL。SQL Server 数据库引擎可优化 bit 列的存储。如果表中有不多于 8 个列是 bit 类型，则这些列公用 1 字节存储。如果 bit 类型的列为 9～16 个，则这些列作为 2 字节存储，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	1 字节

SQL Server 支持的带固定精度和小数位数的数值数据类型有两个，分别是 decimal 和 numeric，这两个类型的语法格式分别为：decimal[(p[,s])] 和 numeric[(p[,s])]。

使用最大精度时，有效值的范围为 $-10^{38}+1$ 到 $+10^{38}-1$。decimal 的 ISO 同义词为 dec 和 dec(p,s)。numeric 在功能上等价于 decimal。

- p（精度）：最多可以存储的十进制数字的总位数，包括小数点左边和右边的位数。该精度必须是从 1 到最大精度 38 之间的值。默认精度为 18。

- s（小数位数）：小数点右边可以存储的十进制数字的位数。从 p 中减去此数字可确定小数点左边的最大位数。小数位数必须是从 0 到 p 之间的值。仅在指定精度后才可以指定小数位数。默认的小数位数为 0；因此，0≤s≤p。最大存储大小基于精度而变化，如表 3-4 所示。

表 3-4 定点小数类型

精 度	存储字节数
1～9	5
10～19	9
20～28	13
29～38	17

2. 近似数字类型

用于表示浮点数值数据的大致数值数据类型。浮点数据为近似值，因此，并非数据类型范围内的所有值都能精确地表示。近似数字类型是用于表示浮点型数据的近似数据类型。

表 3-5 列出了 SQL Server 支持的近似数字类型。

表 3-5 近似数字类型

数据类型	说 明	存 储
Float[(*n*)]	−1.79E+308～−2.23E−308、0，以及 2.23E−308 ～ 1.79E+308	取决于 *n* 的值
real	−3.40E+38 ～−1.18E−38、0，以及 1.18E−38 ～ 3.40E+38	4 字节

float[(*n*)] 中的 *n* 为用于存储 float 数值尾数的位数（以科学计数法表示），因此可以确定精度和存储大小。*n* 的值介于 1～53，默认值为 53。表 3-6 列出了 *n* 的值对应的精度和存储空间。

表 3-6 float[(*n*)] 中 *n* 的值对应的精度和存储空间

n 值	精 度	存 储
1～24	7 位数	4 字节
25～53	15 位数	8 字节

说明：SQL Server 将 *n* 视为下列两个可能值之一。如果 1≤*n*≤24，则将 *n* 视为 24。如果 25≤*n*≤53，则将 *n* 视为 53。

3.2.2 字符串类型

字符串型数据由汉字、英文字母、数字和各种符号组成。目前字符的编码方式有两种：一种是非 Unicode 编码，另一种是 Unicode（统一字符编码）。非 Unicode 编码指的是不同国家或地区的编码长度不一样，比如英文字母的编码是 1 字节（8 位），中文汉字的编码是 2 字节（16 位）。统一字符编码是指不管对哪个地区、哪种语言均采用双字节（16 位）编码，即将世界上所有的字符统一进行编码。

1. 非 Unicode 字符串类型

表 3-7 列出了 SQL Server 支持的普通编码的字符串类型。

表 3-7 非 Unicode 字符串类型

数据类型	说 明	存 储
Char[(*n*)]	固定长度，非 Unicode 字符串数据。*n* 用于定义字符串长度，取值范围为 1～8000。char 的 ISO 同义词为 character	*n* 字节
Varchar[(*n*|max)]	可变长度，非 Unicode 字符串数据。*n* 用于定义字符串长度，取值范围为 1～8000。max 指示最大存储大小是 $2^{31}-1$ 字节（2GB）。varchar 的 ISO 同义词为 char varying 或 character varying	*n*+2 字节
text	可存储 $2^{31}-1$（2 147 483 647）个长度可变的非 Unicode 字符数据	

说明：如果没有在数据定义或变量声明语句中指定 *n*，则默认长度为 1。

2. Unicode 字符串类型

SQL Server 的 Unicode 字符串数据使用 Unicode UCS-2 字符集。表 3-8 列出了 SQL Server 支持的 Unicode 字符串类型。

表 3-8　Unicode 字符串类型

数据类型	说　明	存　储
nchar[(*n*)]	固定长度的 Unicode 字符串数据。*n* 用于定义字符串长度，取值范围为 1～4000。nchar 的 ISO 同义词为 national char 和 national character	2*n* 字节
nvarchar[(n|max)]	可变长度的 Unicode 字符串数据。*n* 用于定义字符串长度，取值范围为 1～4000。max 指示最大存储大小是 $2^{31}-1$ 字节 (2GB)。nvarchar 的 ISO 同义词为 national char varying 和 national character varying	2*n*＋2 字节
ntext	长度可变的 Unicode 字符串数据，字符串最大长度为 $2^{30}-1$（1 073 741 823）字节。ntext 的 ISO 同义词为 national text	所输入字符串长度的两倍（以字节为单位）

说明：如果没有在数据定义或变量声明语句中指定 *n*，则默认长度为 1。

对 char、varchar、nchar 和 nvarchar 的使用，可参考如下建议：

- 如果列数据项的大小一致，则建议使用 char 或 nchar。
- 如果列数据项的大小差异相当大，则建议使用 varchar 或 nvarchar。
- 如果列数据项大小相差很大，而且大小可能超过 8000 字节，则使用 varchar(max) 或 nvarchar(max)。
- 如果希望支持多语言，则建议使用 nchar 或 nvarchar 类型，以最大限度地消除字符转换问题。

注意

- SQL Server 中的字符串常量要用单引号括起来，比如 '计算机系 '。
- 对固定长度的字符串数据，系统分配固定的字节数。如果空间未被占满，则系统自动用空格填充。比如对 char(6) 类型数据，若存储 'abc'，则系统分配 6 字节空间，后边补 3 个空格。

3. 二进制字符串类型

二进制字符串数据类型用于存储图形图像数据，表 3-9 列出了 SQL Server 支持的二进制字符串类型。

表 3-9　二进制字符串类型

数据类型	说　明	存　储
binary[(*n*)]	固定长度为 *n* 字节的二进制数据，*n* 的取值从 1 到 8000	*n* 字节
varbinary[(*n*|max)]	可变长度二进制数据。*n* 的取值从 1 到 8000，max 指示最大存储大小为 $2^{31}-1$ 字节。varbinary 的 ANSI SQL 同义词为 binary varying	所输入数据的实际长度＋2 字节
image	长度可变的二进制数据，二进制数据最大长度为 $2^{31}-1$（2 147 483 647）字节	0 ～ $2^{31}-1$ 字节

说明：如果没有在数据定义或变量声明语句中指定 *n*，则默认长度为 1。

注意

在 Microsoft SQL Server 的未来版本中将删除 ntext、text 和 image 数据类型。请避免在新开发工作中使用这些数据类型，并考虑修改当前已使用这些数据类型的应用程序。请改用 nvarchar(max)、varchar(max) 和 varbinary(max)。

3.2.3 日期和时间类型

SQL Server 2012 支持的日期和时间类型有：datatime、smalldatetime、date、time、datetime2 和 datetimeoffset，这些类型的介绍如表 3-10 所示。

表 3-10 日期时间类型

数据类型	说明	范围	存储
datetime	用于定义一个与采用 24 小时制并带有秒小数部分的一日内时间相组合的日期 格式：YYYY-MM-DD hh:mm:ss.n* n* 为一个 0 到 3 位的数字，范围为 0 到 999，表示秒的小数部分	日期范围： 1753-1-1 到 9999-12-31 时间范围： 00:00:00 到 23:59:59.997	8 字节
smalldatetime	定义结合了一天中时间日期。此时间为 24 小时制，秒始终为零 (:00)，并且不带秒小数部分 格式：YYYY-MM-DD hh:mm:00 精确到 1 分钟	1900-01-01 到 2079-06-06	4 字节
date	定义一个日期。格式：YYYY-MM-DD	0001-01-01 到 9999-12-31	3 字节
time	定义一天中的某个时间 格式：hh:mm:ss[.n*] n* 是 0 到 7 位的数字，范围为 0 到 9 999 999，表示秒的小数部分	00:00:00.0000000 到 23:59:59.9999999	5 字节
datetime2	定义结合了 24 小时制时间的日期。可将 datetime2 视作现有 datetime 类型的扩展，其数据范围更大，默认的小数精度更高，并具有可选的用户定义的精度 格式：YYYY-MM-DD hh:mm:ss[.n*] n* 代表 0 到 7 位的数字，范围从 0 到 9 999 999，表示秒小数部分。准确度为 100ns，默认精度为 7 位数	0001-01-01 到 9999-12-31	精度小于 3 时为 6 字节；精度为 3 和 4 时为 7 字节。所有其他精度则需要 8 字节
datetimeoffset	用于定义一个与采用 24 小时制并可识别时区的一日内时间相组合的日期 YYYY-MM-DD hh:mm:ss[.nnnnnnn] [{+|-}hh:mm] n* 是 0 到 7 位的数字，范围为 0 到 9999999，表示秒的小数部分 hh 是两位数，范围为－14 到＋14 mm 是两位数，范围为 00 到 59	0001-01-01 到 9999-12-31	10 字节

说明：在今后的编程中，推荐使用 time、date、datetime2 和 datetimeoffset 数据类型，这些类型符合 SQL 标准，更易于移植。time、datetime2 和 datetimeoffset 提供更高精度的秒数。

对于 SQL Server 来说，日期和时间类型的数据常量要用单引号括起来，比如 '2015-8-20'、'2015-8-22 10:23:50'。

3.3　数据定义功能

表是数据库中非常重要的对象，它用于存储用户的数据。在有了数据类型的基础知识后，就可以开始创建数据表了。关系数据库的表是二维表，包含行和列，创建表就是定义表所包含的各列的结构，其中包括列的名称、数据类型、约束等。列的名称是人们为列取的名字，一般为了便于记忆，最好取有意义的名字，比如“学号”或“Sno”，而不要取无意义的名字，比如“a1”；列的数据类型说明了列的可取值范围；列的约束更进一步限制了列的取值范围，这些约束包括：列取值是否允许为空、主码约束、外码约束、列取值范围约束等。

3.3.1　基本表的定义与删除

1. 定义基本表

定义基本表使用 SQL 语言数据定义功能中的 CREATE TABLE 语句实现，其一般格式为：

```
CREATE  TABLE  <表名> (
<列名>  <数据类型>  [列级完整性约束定义]
{,  <列名>  <数据类型> [列级完整性约束定义]… }
[, 表级完整性约束定义 ] )
```

注意

默认 SQL 语言不区分大小写。

其中：

- <表名>是所要定义的基本表的名字，同样，这个名字最好能表达表的应用语义，比如，“学生表”或“Student”。
- <列名>是表中所包含的属性列的名字，<数据类型>指明列的数据类型，一个表可以包含多个列，因此也就可以包含多个列定义。
- 在定义表的同时还可以定义与表有关的完整性约束条件，这些完整性约束条件都会存储在系统的数据字典中。大部分完整性约束都既可以在“列级完整性约束定义”处定义，也可以在“表级完整性约束定义”处定义；但有些涉及多个列的完整性约束则必须在“表级完整性约束定义”处定义。

上述语法中用到了一些特殊的符号，比如“[ ]”，这些符号是文法描述的常用符号，而不是 SQL 语句的部分。下面简单介绍一下这些符号的含义（在后面的语法介绍中也要用到这些符号），有些符号在上述这个语法中可能没有用到。

方括号（[ ]）中的内容表示是可选的（即可出现 0 次或 1 次），比如 [列级完整性约束定义] 代表可以有也可以没有列级完整性约束定义。花括号（{ }）与省略号（…）一起，表示其中的内容可以出现 0 次或多次。竖杠（|）表示在多个短语中选择一个，比如 term1 | term2 | term3，表示在三个选项中任选一项。竖杠也能用在方括号中，表示可以选择由竖杠分隔的子句中的一个，但整个句子又是可选的（也就是可以没有子句出现）。

在定义基本表时可以同时定义表的完整性约束。定义完整性约束时可以在定义列的同时定义，也可以将完整性约束作为独立的项定义。在定义列的同时定义的约束称为列级完整性约束，作为表中独立的一项定义的完整性约束称为表级完整性约束。在列级完整性约束定义处可以定义如下约束：

- NOT NULL：限制列取值非空。
- DEFAULT：指定列的默认值。
- UNIQUE：限制列取值不能重复。
- CHECK：限制列的取值范围。
- PRIMARY KEY：定义主码。
- FOREIGN KEY：定义外码。

在上述约束中，除了 NOT NULL 和 DEFAULT 只能在“列级完整性约束定义”处定义之外，其他约束均可在“表级完整性约束定义”处定义。但有几点需要注意，第一，如果 CHECK 约束是定义多列之间的取值约束，则只能在“表级完整性约束定义”处定义；第二，如果在“表级完整性约束定义”处定义主码，则主码列要用括号括起来，即 PRIMARY KEY（列 1｛ [，列 2] …｝）；第三，如果在“表级完整性约束定义”处定义外码，则 FOREIGN KEY 和 < 外码列名 > 均不能省略。

本节只简单介绍如何实现非空约束、主码约束和外码约束，在 3.4 节将详细介绍数据完整性的概念和实现方法。

（1）定义主码约束

如果是在列级完整性约束处定义主码，则语法格式为：

```
< 列名 > 数据类型 [CONSTRAINT 约束名 ] PRIMARY KEY [(< 列名 > [, … n] )]
```

如果是在表级完整性约束处定义主码，则语法格式为：

```
[CONSTRAINT 约束名 ] PRIMARY KEY (< 列名 > [, … n] )
```

（2）定义外码约束

一般情况下外码都是单列的，它可以定义在列级完整性约束处，也可以定义在表级完整性约束处。定义外码的语法格式为：

```
[CONSTRAINT 约束名 ][FOREIGN KEY (< 列名 >)] REFERENCES < 外表名 >(< 外表列名 >)
```

如果是在列级完整性约束处定义外码，则可以省略“FOREIGN KEY(< 列名 >)”部分。

【例 1】 用 SQL 语句创建如下三张表：学生（Student）表、课程（Course）表和学生修课（SC）表，这三张表的结构如表 3-11 到表 3-13 所示。

表 3-11 Student 表结构

列 名	含 义	数 据 类 型	约 束
Sno	学号	普通编码定长字符串，长度为 7	主码
Sname	姓名	普通编码定长字符串，长度为 10	非空
Ssex	性别	普通编码定长字符串，长度为 2	
Sage	年龄	微整型	
Sdept	所在系	普通编码定长字符串，长度为 20	

表 3-12　Course 表结构

列　名	含　义	数据类型	约　束
Cno	课程号	普通编码定长字符串，长度为 6	主码
Cname	课程名	普通编码定长字符串，长度为 20	非空
Credit	学分	微整型	
Semster	学期	微整型	

表 3-13　SC 表结构

列　名	含　义	数据类型	约　束
Sno	学号	普通编码定长字符串，长度为 7	主码，引用 Student 的外码
Cno	课程名	普通编码定长字符串，长度为 6	主码，引用 Course 的外码
Grade	成绩	小整型	

创建满足约束条件的上述三张表的 SQL 语句如下（注意，为了说明约束定义的灵活性，我们将 Student 表的主码约束定义在列级完整性约束处，将 Course 表的主码约束定义在表级完整性约束处）：

```
CREATE TABLE Student (
  Sno     char(7)    PRIMARY KEY,    /* 在列级完整性约束处定义主码约束 */
  Sname   char(10) NOT NULL,         /* 非空约束 */
  Ssex    char(2),
  Sage    tinyint,
  Sdept   char(20)
)

CREATE TABLE Course (
  Cno       char(6)   NOT NULL,
  Cname     char(20)  NOT NULL,
  Credit    tinyint,
  Semester tinyint,
  PRIMARY  KEY(Cno)                 /* 在表级完整性约束处定义主码约束 */
)

CREATE TABLE SC (
  Sno     char(7)  NOT NULL,
  Cno     char(6)  NOT NULL,
  Grade   smallint,
  PRIMARY KEY(Sno, Cno),
  FOREIGN KEY(Sno) REFERENCES Student(Sno),
  FOREIGN KEY(Cno) REFERENCES Course(Cno)
)
```

2. 删除表

当确信不再需要某个表时，可以将其删除，删除表时会将表中的数据一起删掉。

删除表的 SQL 语句为 DROP TABLE，其语法格式为：

```
DROP  TABLE  <表名>  { [, <表名> ] … }
```

【例 2】 删除 test 表，语句为：

```
DROP TABLE test
```

3.3.2 修改表结构

在定义完表之后，如果需求有变化，比如需要添加列、删除列或修改列定义，则可以使用 ALTER TABLE 语句实现。ALTER TABLE 语句可以实现添加列、删除列和修改列定义的功能，也可以实现添加和删除约束的功能。

不同数据库厂商的数据库产品的 ALTER TABLE 语句的格式略有不同，这里给出 SQL Server 支持的 ALTER TABLE 语句格式，对于其他数据库管理系统，可以参考它们的语言参考手册。

ALTER TABLE 语句的部分语法格式如下：

```
ALTER TABLE <表名>
{ ALTER COLUMN <列名> <新数据类型>                    -- 修改列定义
| ADD  <列名> <数据类型> <约束>                        -- 添加新列
| DROP COLUMN <列名>                                  -- 删除列
| ADD  [constraint <约束名>] 约束定义                 -- 添加约束
| DROP [constraint] <约束名>}                         -- 删除约束
```

注：“--”为 T-SQL 语言的单行注释符，“/* … */”为 T-SQL 语言的块注释符。

本节介绍添加、删除和修改列定义的例子，下一节介绍添加和删除约束的例子。

【例 3】 为 Student 表添加“专业”列，此列的定义为 Spec char(10)，允许空。语句为：

```
ALTER TABLE Student
  ADD Spec char(10) NULL
```

【例 4】 将新添加的“专业”列的类型改为 char(20)。语句为：

```
ALTER TABLE Student
  ALTER COLUMN Spec char(20)
```

【例 5】 删除新添加的“专业”列。语句为：

```
ALTER TABLE Student
  DROP COLUMN Spec
```

3.4 数据完整性

数据完整性是指数据的正确性和相容性。例如，每个人的身份证号必须是唯一的，人的性别只能是“男”或“女”，人的年龄应该在 0 到 150 岁之间（假设人现在最多能活到 150 岁）的数字，学生所在的系必须是学校已有的系等。

数据完整性约束是为了防止数据库中存在不符合语义的数据，为了维护数据的完整性，数据库管理系统必须提供一种机制来检查数据库中的数据，看其是否满足语义规定的条件。这些加在数据库数据之上的语义约束条件就是数据完整性约束，这些约束条件作为表定义的一部分存储在数据库中。而 DBMS 检查数据是否满足完整性约束条件的机制就称为完整性检查。

3.4.1 完整性约束条件的作用对象

完整性检查是围绕完整性约束条件进行的，因此，完整性约束条件是完整性控制机制的核心。完整性约束条件的作用对象可以是表、元组和列。

1. 列级约束

列级约束主要指对列的类型、取值范围、精度等的约束，具体包括如下内容：

- 对数据类型的约束：包括数据类型、长度、精度等。例如，学生学号的数据类型为

普通编码定长字符型，长度为 7。

- 对数据格式的约束：如规定学号的前两位表示学生的入学年份，第 3 位表示系的编号，第 4 位表示专业编号，第 5 位表示班的编号等。
- 对取值范围或取值集合的约束：如学生的成绩取值范围为 0 ～ 100，最低工资要大于 1600 等。
- 对空值的约束：有些列允许有空值（比如成绩），有些列则不允许有空值（比如姓名），在定义列时应指明其是否允许取空值。

2. 元组约束

元组约束指元组中各个字段之间的相互约束，如某个活动的开始日期小于结束日期。

3. 关系约束

关系约束指若干元组之间、关系之间的联系的约束。比如，学号的取值不能重复也不能取空值，学生修课表中的学号的取值受学生表中的学号取值的约束等。

3.4.2　实现数据完整性

实现数据完整性一般是在服务器端完成的。在服务器端实现数据完整性的方法主要有两种，一种是在定义表时声明数据完整性，另一种是在服务器端编写触发器来实现数据完整性（本书只介绍在定义表时声明完整性的方法）。不管采用哪种方法，只要用户定义好了数据完整性，后续执行对数据的增加、删除、修改操作时，数据库管理系统都会自动检查用户定义的完整性约束，只有符合约束条件的操作才会被执行。

这里介绍如何使用 SQL 语句实现实体完整性（PRIMARY KEY）、引用完整性（FOREIGN KEY）和用户定义的完整性。在用户定义的完整性中介绍默认值（DEFAULT）约束、列值取值范围（CHECK）约束和唯一值（UNIQUE）约束的实现方法。

下面以雇员表和工作表为例，逐步在这两张表上添加约束。这两张表的结构如下：

雇员表：

雇员编号　普通编码定长字符型　长度为 7，非空
雇员名　　普通编码定长字符型，长度为 10
工作编号　普通编码定长字符型，长度为 8
工资　　整型
电话号码　普通编码定长字符型，长度为 8，非空

工作表：

工作编号　普通编码定长字符型，长度为 8，非空
最低工资　整型
最高工资　整型

其中，“雇员编号”是雇员表的主码，“工作编号”是工作表的主码，而且雇员表中的“工作编号”是引用工作表中的“工作编号”的外码。

假设我们已经按上述要求创建好了这两张表，现在要为这两张表添加需要的约束。

1. 主码约束

添加主码约束要注意如下问题：

- 每个表只能有一个 PRIMARY KEY 约束。

- 用 PRIMARY KEY 约束的列取值不能有重复，而且不允许有空值。

添加主码约束的 ALTER TABLE 语句语法格式如下：

```
ALTER TABLE 表名
  ADD [ CONSTRAINT <约束名> ] PRIMARY KEY (<列名> [, … n])
```

【例 6】 对雇员表和工作表分别添加主码约束。语句如下：

```
ALTER TABLE 雇员表
  ADD CONSTRAINT  PK_EMP PRIMARY KEY (雇员编号)
ALTER TABLE 工作表
  ADD CONSTRAINT PK_JOB PRIMARY KEY (工作编号)
```

2. UNIQUE 约束

UNIQUE 约束用于限制在一个列中不能有重复的值。这个约束用在事实上具有唯一性的属性列上，比如每个人的身份证号码、驾驶证号码等均不能有重复值。定义 UNIQUE 约束时需要注意如下事项：

- UNIQUE 约束的列允许有一个空值。
- 在一个表中可以定义多个 UNIQUE 约束。
- 可以在多个列上定义一个 UNIQUE 约束，表示这些列组合起来不能有重复值。

当一个表中存在多个不允许有重复值的属性时，UNIQUE 约束就非常有用，比如雇员表中，雇员编号和电话号码都不允许取值重复（假设雇员的电话号码不能重复），已经在雇员编号上定义了主码约束，因此可以保证雇员编号取值不重复。这时电话号码取值不重复就必须使用 UNIQUE 约束来保证了。

添加 UNIQUE 约束的语法格式如下：

```
ALTER TABLE 表名
  ADD [ CONSTRAINT <约束名> ] UNIQUE (<列名> [, … n])
```

【例 7】 为雇员表的“电话号码”列添加 UNIQUE 约束。语句如下：

```
ALTER TABLE 雇员表
  ADD CONSTRAINT UK_SID UNIQUE (电话号码)
```

3. 外码约束

外码约束实现了引用完整性，添加外码约束时要注意：外码所引用的列必须是有 PRIMARY KEY 约束或 UNIQUE 约束定义的列。

添加 FOREIGN KEY 约束的语法格式如下：

```
ALTER TABLE 表名
  ADD [ CONSTRAINT <约束名> ]
  FOREIGN KEY (<列名>) REFERENCES 引用表名 (<列名>)
```

【例 8】 为雇员表的工作编号添加外码引用约束，此列引用工作表的工作编号列。语句如下：

```
ALTER TABLE 雇员表
  ADD CONSTRAINT FK_job_id
  FOREIGN KEY (工作编号) REFERENCES 工作表 (工作编号)
```

4. DEFAULT 约束

DEFAULT 约束用于提供列的默认值。只有在向表中插入数据时系统才检查 DEFAULT 约束。添加 DEFAULT 约束的语法格式如下：

```
ALTER TABLE 表名
  ADD [ CONSTRAINT < 约束名 > ] DEFAULT 默认值 FOR 列名
```

【例 9】 定义雇员表的工资的默认值为 1600。语句如下：

```
ALTER TABLE 雇员表
   ADD CONSTRAINT DF_SALARY DEFAULT 1600 FOR 工资
```

5. CHECK 约束

CHECK 约束用于限制列的取值在指定范围内，使数据库中存放的值都是有意义的。例如，人的性别只能是“男”或“女”，工资必须大于或等于 1000 元（假设最低工资为 1000 元）。使用 CHECK 约束可以限制一个列的取值范围，也可以约束同一个表中多个列之间的相互取值约束，比如最低工资小于最高工资。

添加 CHECK 约束的语法格式如下：

```
ALTER TABLE 表名
  ADD [CONSTRAINT < 约束名 >] CHECK （逻辑表达式）
```

【例 10】 在雇员表中，添加限制雇员的工资必须大于等于 1000 的约束。语句如下：

```
ALTER TABLE 雇员表 （
  ADD CONSTRAINT  CHK_Salary CHECK （工资 >= 1000)
```

【例 11】 添加限制工作表的最低工资小于等于最高工资的约束。语句如下：

```
ALTER TABLE 工作表
  ADD CONSTRAINT  CHK_Job_Salary CHECK （最低工资 <= 最高工资）
```

上述这些约束都可以在定义表的同时定义，综合起来如下：

```
CREATE TABLE 工作表 （
  工作编号  char(8) PRIMARY KEY,
  最低工资 int,
  最高工资 int,
  CHECK （最低工资 <= 最高工资）
)
CREATE TABLE 雇员表 （
  雇员编号 char(7) PRIMARY KEY,
  雇员名   char(10),
  工作编号 char(8) REFERENCES 工作表（工作编号）,
  工资  int DEFAULT 1600 CHECK （工资 >= 1000),
  电话号码 char(8) NOT NULL UNIQUE
)
```

3.5　小结

本章首先介绍了 SQL 语言的发展、特点以及所支持的数据类型。SQL 支持的数据类型有数值型、字符串型和日期时间类型。

本章还介绍了基本表的创建、删除和修改语句，并详细介绍了实现数据完整性的方法。数据完整性可以在定义表的同时定义，也可以在定义完表之后再通过修改表结构的方法添

加。当创建完表而约束又有变化时，就可以使用修改表结构的语句来添加数据约束。定义了数据完整性约束之后，每当执行修改数据的操作时，数据库管理系统会首先检查所做的操作是否满足数据完整性约束要求，若不满足，则不执行这个修改数据的操作。

对数据完整性约束的检查是由数据库管理系统自动实现的，而且是在数据更改操作执行之前作完整性约束判断。

习题

1. Tinyint 数据类型定义的数据的取值范围是多少？
2. 日期时间类型中的日期和时间的输入格式是什么？
3. SmallDatatime 类型精确到哪个时间单位？
4. 定点小数类型 numeric 中的 p 和 q 的含义分别是什么？
5. Char(10)、nchar(10) 的区别是什么？它们各能存放多少个字符？占用多少空间？
6. Char(*n*) 和 varchar(*n*) 的区别是什么？其中 *n* 的含义是什么？各占用多少空间？
7. 数据完整性的含义是什么？
8. 在对数据进行什么操作时，系统检查 DEFAULT 约束？在进行什么操作时，检查 CHECK 约束？
9. UNIQUE 约束的作用是什么？
10. 写出创建如下三张表的 SQL 语句，要求在定义表的同时定义数据的完整性约束。

（1）“图书表”结构如下：

书号：统一字符编码定长类型，长度为 6，主码

书名：统一字符编码可变长类型，长度为 30，非空

第一作者：普通编码定长字符类型，长度为 10，非空

出版日期：小日期时间型

价格：定点小数，小数部分 1 位，整数部分 3 位

（2）“书店表”结构如下：

书店编号：统一字符编码定长类型，长度为 6，主码

店名：统一字符编码可变长类型，长度为 30，非空

电话：普通编码定长字符类型，8 位长，每一位的取值均是 0 ～ 9 的数字

地址：普通编码可变长字符类型，40 位长

邮政编码：普通编码定长字符类型，6 位长

（3）“图书销售表”结构如下：

书号：统一字符编码定长类型，长度为 6，非空

书店编号：统一字符编码定长类型，长度为 6，非空

销售日期：小日期时间型，非空

销售数量：微整型，大于等于 1

主码为（书号，书店编号，销售日期）；其中“书号”为引用“图书表”的“书号”的外码；“书店编号”为引用“书店表”的“书店编号”的外码。

11. 为图书表添加“印刷数量”列，类型为整数，同时添加约束，要求此列的取值要大于或等于 1000。
12. 删除书店表中的“邮政编码”列。
13. 将图书销售表中的“销售数量”列的数据类型改为整型。