第 5 章　Transact-SQL

SQL 的全称为 Structured Query Language(结构化查询语言)，它利用一些简单的句子构成基本的语法，来存取数据库的内容。由于 SQL 简单易学，目前它已经成为关系数据库系统中使用最广泛的语言。Transact-SQL 是 Microsoft SQL Server 提供的一种结构化查询语言，本章主要介绍 Transact-SQL 程序设计基础。

5.1　SQL 语言

SQL 是在 20 世纪 70 年代末由 IBM 公司开发出来的一套程序语言，并被用在 DB2 关系数据库系统中。但是，直到 1981 年，IBM 推出商用的 SQL/DS 关系型数据库系统，Oracle 及其他大型关系型数据库系统相继出现，SQL 才得以广泛应用。

5.1.1 概述

由于在产业界有多种关系型数据库系统，因此各家公司都可能有自己的 SQL 语法或者可以定义不同的数据类型。例如，Sybase 与 Microsoft 公司使用 Transact-SQL，而 Oracle 公司使用 PL/SQL(Procedural Language extension to SQL)，将原来非过程性的 SQL 语法改变为过程性语法。

基于上面的原因，使得 SQL 有 ANSI(American National Standards Institute，美国国家标准局)SQL-92 标准与产业界的标准之分。ANSI SQL-92 标准定义了 SQL 关键字与语法的标准，而各公司在其基础上又增加了各自的扩充。因此，虽然各公司的数据库系统使用的 SQL 不尽相同，但是基本语法以及关键字等还是相互兼容的。

> 提示：Transact-SQL 是 SQL Server 使用的 SQL 语言，它也在 ANSI SQL-92 标准的基础上进行了扩充，使得其功能更为强大，使用更为方便。

SQL 语言是应用于数据库的语言，本身是不能独立存在的。它是一种非过程性语言，与一般的高级语言，例如 C、Pascal，是大不相同的。一般的高级语言在存取数据库时，需要依照每一行程序的顺序处理许多的动作。但是使用 SQL 时，只需告诉数据库需要什么数据，怎么显示就可以了。具体的内部操作则由数据库系统来完成。

例如，要从 school 数据库中的 student 表中查找姓名为“李军”的学生记录，使用简单的几行命令即可(实际上，该语句经常写为一行，这里只是为了说明，将其分成成了 3 行)，对应的命令如下：

```
SELECT *
```

火龙果•整理
uml.org.cn


```
FROM student
WHERE sname="李军"
```

其中第 1 行表示选择表中的字段，“*”号表示选择所有字段，第 2 行指定表名称，这里表示从表 student 中获取数据，第 3 行表示指定查询的条件，这里是指定姓名条件。

5.1.2 SQL 语言的分类

SQL 语言按照用途可以分为如下 3 类：

- DQL (Data Query Language) 数据查询语言
- DDL(Data Definition Language) 数据定义语言
- DML(Data Manipulation Language) 数据操纵语言
- DCL(Data Control Language) 数据控制语言

下面分别介绍。

1．数据查询语言

查询是数据库的基本功能，查询操作通过 SQL 数据查询语言来实现的，例如，用 SELECT 查询表中的内容。

2．数据定义语言

在数据库系统中，每一个数据库、数据库中的表、视图和索引等都是对象。要建立一个对象，都可以通过 SQL 语言来完成。类似于这一类定义数据库对象的 SQL 叙述即为 DDL 语言。例如，数据库和表的创建。

3．数据操纵语言

SQL 语法中处理数据的叙述称为 DML。例如，使用 INSERT(插入)、DELETE(删除)和 UPDATE(更新)一笔记录等。这些属于 DML。

> 注意：具体的 DDL、DML 语言的 SQL 语言操作，例如表的建立，查询表中的数据等，均可参考下面的介绍。

4．数据控制语言

对单个的 SQL 语句来说，不管执行成功或者失败，都不会影响到其他的 SQL 语句。但是在某些情况下，可能需要一次处理好几个 SQL 语句，而且希望它们必须全部执行成功，如果其中一个执行失败，则这一批 SQL 语句都不要执行。已经执行的应该恢复到开始的状态。

举个简单的银行转帐的例子。假设要从 A 帐户中转 10000 元到 B 帐户中，首先从 A 帐户中扣除 10000 元，然后在 B 帐户中加入 10000 元。但是，如果从 A 帐户中扣除 10000 元后，出现错误，导致下一步在 B 帐户中加入 10000 元的操作不能完成，则 A 帐户白白被扣除了 10000 万。因此，应保证这些操作要么一起完成，要么都不要执行。这种方式在 SQL 中称做事务(Transaction)。

火龙果•整理
uml.org.cn

在 SQL 中，可以使用 DCL 将数个 SQL 语句组合起来，然后交给数据库系统一并处理。详细内容参见下一章。

5.2 Transact-SQL 基础

SQL Server 提供了多种图形和命令行工具，用户可以使用不同的方法来访问数据库。但是这些工具的核心却是 Transact-SQL 语言。SQL 查询分析器是一个图形用户界面，用以交互地设计和测试 Transact-SQL 语句、批处理和脚本。

本节通过数据库和表的操作语句以及 SELECT 查询语句来介绍 Transact-SQL 的基本使用。因此，本节首先介绍一下查询分析器的使用，然后再介绍如何使用查询分析器来执行相关的 SQL 语句。

5.2.1　查询分析器

在 SQL 查询分析器中，用户可在全文窗口中输入 Transact-SQL 语句，执行语句并在结果窗口中查看结果。用户也可以打开包含 Transact-SQL 语句的文本文件，执行语句并在结果窗口中查看结果。

SQL 查询分析器提供：

- 用于键入 Transact-SQL 语句的自由格式文本编辑器。
- 在 Transact-SQL 语句中使用不同的颜色，以提高复杂语句的易读性。
- 对象浏览器和对象搜索工具，可以轻松查找数据库中的对象和对象结构。
- 模板，可用于加快创建 SQL Server 对象的 Transact-SQL 语句的开发速度。模板是包含创建数据库对象所需的 Transact-SQL 语句基本结构的文件。
- 用于分析存储过程的交互式调试工具。
- 以网格或自由格式文本窗口的形式显示结果。
- 显示计划信息的图形关系图，用以说明内置在 Transact-SQL 语句执行计划中的逻辑步骤。这使程序员得以确定在性能差的查询中，具体是哪一部分使用了大量资源。之后，程序员可以试着采用不同的方法更改查询，使查询使用的资源减到最小的同时仍返回正确的数据。
- 使用索引优化向导分析 Transact-SQL 语句以及它所引用的表，以了解通过添加其他索引是否可以提高查询的性能。

可以直接运行“开始|程序|Microsoft SQL Server|查询分析器”命令，或从 SQL Server 企业管理器内运行它。还可以通过执行 isqlw 实用工具从命令提示符运行 SQL 查询分析器。

> 提示：使用 isqlw 实用工具(SQL 查询分析器)可以输入 Transact-SQL 语句、系统存储过程和脚本文件。通过设置快捷方式或创建批处理文件，可以启动预配置的 SQL 查询分析器。具体使用方法可参考联机帮助。

启动 SQL 查询分析器的操作步骤如下：

火龙果•整理
uml.org.cn

(1) 在“开始”菜单的“程序”级联菜单中，选择 Microsoft SQL Server 程序组中的“查询分析器”选项，即可启动 SQL 在询分析器。

(2) 在启动 SQL 查询分析器时，系统首先打开“连接到 SQL Server”对话框，如图 5.1 所示。使用该对话框可以指定连接到哪个 SQL Server 服务器。

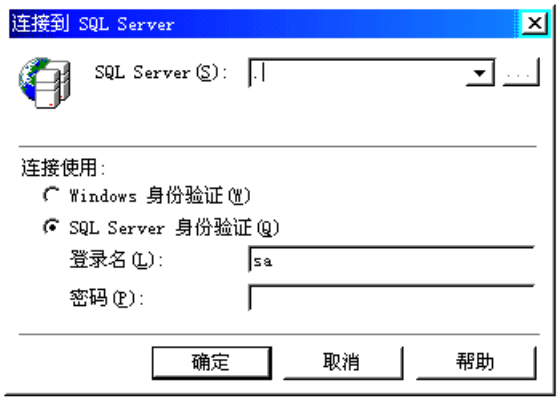
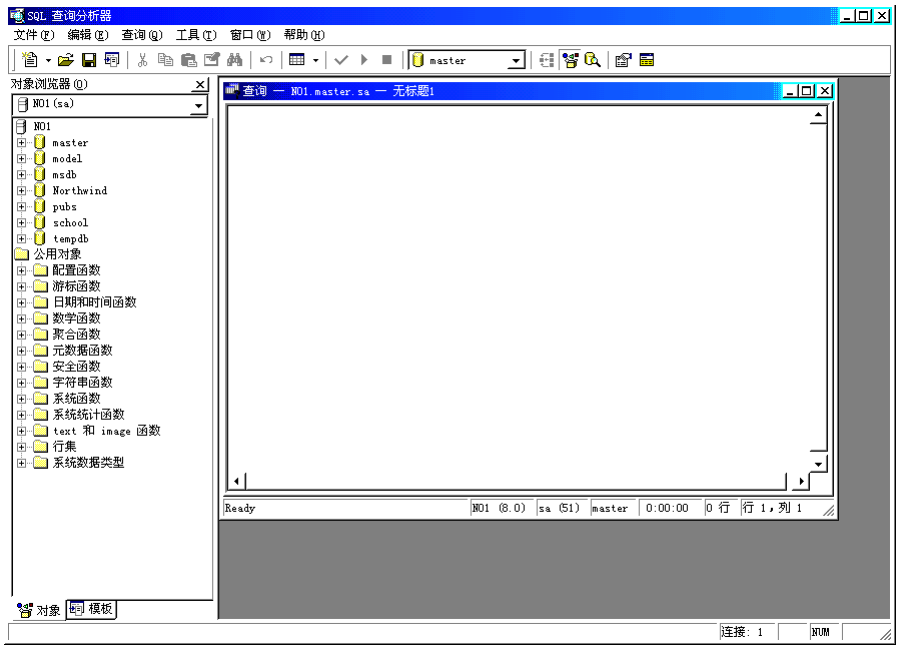


图 5.1“连接到 SQL Server”对话框

提示：如果是远程 SQL Server 服务器，可以直接在 SQL Server 文本框中输入远程计算机的 IP 地址。

(3) 选择 SQL Server 服务器后，单击“确定”按钮，然后在“连接到 SQL Server”对话框中设置正确的验证模式，即可连接到 SQL Server 服务器，并打开 SQL 查询分析器，如图 5.2 所示。



火龙果•整理
uml.org.cn

图 5.2　SQL 查询分析器

SQL 查询分析器分为两个窗口，左侧窗口为“对象浏览器”窗口，“对象”选项卡显示了连接到的 SQL Server 服务器中所包含的数据库以及数据库对象，而“模板”选项卡则显示了查询分析器中所包含的各种 Transact-SQL 语句模板，可以使用这些模板方便地创建自己的 SQL 语句。

右侧窗口为 SQL 语句的输入窗口，输入 SQL 语句或者打开一个 SQL 脚本文件(后缀名为.sql)后，可单击工具栏上的“执行查询”按钮▶来执行输入的 SQL 语句。执行完成后，将在右侧窗口的下部显示查询结果或者出错信息。

> 提示：SQL Server 2000 的 SQL 查询分析器对 SQL Server 7.0 版本的做了很大的改进，增加了“对象浏览器”窗口和对象搜索的功能。在企业管理器中，执行“工具”菜单中的“SQL 查询分析器”命令即可启动 SQL 查询分析器。

5.2.2　数据库的操作语句

在上一章介绍了使用企业管理器创建数据库的方法，使用 SQL 语句同样也可创建数据库，并对数据库进行修改和删除。下面简单介绍有关数据库操作的 SQL 语句。

1．创建数据库

创建数据库可以使用 CREATE DATABASE 语句，该语句简化的语法格式如下：

```
CREATE DATABASE database_name
[
   [ON  [filespec] ]
   [ LOG ON [filespec] ]
]
filespec 定义为
    ( [ NAME = logical_file_name , ]
      FILENAME = 'os_file_name'
      [ , SIZE = size ]
      [ , MAXSIZE = { max_size | UNLIMITED } ]
      [ , FILEGROWTH = growth_increment ] )
```

其中，database_name 指出新数据库的名称。数据库名称在服务器中必须惟一，并且符合标识符的规则。database_name 最多可以包含 128 个字符，除非没有为日志指定逻辑名。ON 子句指定显式定义用来存储数据库数据部分的磁盘文件(数据文件)。该关键字后跟以逗号分隔的“filespec”项列表，“filespec”项用以定义主文件组的数据文件。LOG ON 子句指定显式定义用来存储数据库日志的磁盘文件 (日志文件)。该关键字后跟以逗号分隔的“filespec”项列表，“filespec”项用以定义日志文件。如果没有指定 LOG ON，将自动创建一个日志文件，该文件使用系统生成的名称，大小为数据库中所有数据文件总大小的 25%。

火龙果•整理
uml.org.cn

FILENAME 为“filespec”定义的文件指定操作系统文件名。“os_file_name”指出操作系统创建“filespec”定义的物理文件时使用的路径名和文件名。“os_file_name”中的路径必须指定 SQL Server 实例上的目录。“os_file_name”不能指定压缩文件系统中的目录。如果文件在原始分区上创建，则“os_file_name”必须只指定现有原始分区的驱动器字母。每个原始分区上只能创建一个文件。原始分区上的文件不会自动增长；因此，“os_file_name”指定原始分区时，不需要指定 MAXSIZE 和 FILEGROWTH 参数。

SIZE 子句指定“filespec”中定义的文件的大小。如果主文件的“filespec”中没有提供 SIZE 参数，那么 SQL Server 将使用 model 数据库中的主文件大小。如果次要文件或日志文件的“filespec”中没有指定 SIZE 参数，则 SQL Server 将使文件大小为 1MB。“size”为“filespec”中定义的文件的初始大小，可以使用千字节(KB)、兆字节(MB)、千兆字节(GB)或兆兆字节(TB)后缀。默认值为 MB。指定一个整数，不要包含小数位，“size”的最小值为 512 KB。如果没有指定“size”，则默认值为 1MB。为主文件指定的大小至少应与 model 数据库的主文件大小相同。

MAXSIZE 子句指定“filespec”中定义的文件可以增长到的最大大小。“max_size”指出“filespec”中定义的文件可以增长到的最大大小。可以使用千字节(KB)、兆字节(MB)、千兆字节(GB)或兆兆字节(TB)后缀，默认值为 MB，指定一个整数，不要包含小数位。如果没有指定“max_size”，那么文件将增长到磁盘变满为止。

FILEGROWTH 子句指定“filespec”中定义的文件的增长增量。文件的 FILEGROWTH 设置不能超过 MAXSIZE 设置。“growth_increment”指出每次需要新的空间时为文件添加的空间大小，指定一个整数，不要包含小数位，0 值表示不增长，该值可以 MB、KB、GB、TB 或百分比(%)为单位指定。如果未在数量后面指定 MB、KB 或%，则默认值为 MB。如果指定%，则增量大小为发生增长时文件大小的指定百分比。如果没有指定 FILEGROWTH，则默认值为 10%，最小值为 64KB。指定的大小舍入为最接近的 64KB 的倍数。

使用一条 CREATE DATABASE 语句即可创建数据库以及存储该数据库的文件。SQL Server 分两步实现 CREATE DATABASE 语句：

(1) SQL Server 使用 model 数据库的复本初始化数据库及其元数据。

(2) SQL Server 使用空页填充数据库的剩余部分，除了包含记录数据库中空间使用情况以外的内部数据页。

> 注意：创建用户数据库后，应备份 master 数据库。SIZE、MAXSIZE 和 FILEGROWTH 参数中不能指定分数。若要在 SIZE 参数中指定以兆字节为单位的分数，请将该数字乘以 1,024 转换成千字节。例如，指定 1,536 KB 而不要指定 1.5 MB(1.5 乘以 1,024 等于 1,536)。

如果仅指定 CREATE DATABASE database_name 语句而不带其他参数，那么数据库的大小将与 model 数据库的大小相等。

例如，建立一个名称为 test 的数据库，可以使用如下的 SQL 语句：

```
CREATE DATABASE test
```

火龙果•整理
uml.org.cn

按 F5 键或单击▶按钮，系统提示如下消息，如图 5.3 所示。

```
CREATE DATABASE 进程正在磁盘'test'上分配 0.75MB 的空间。
CREATE DATABASE 进程正在磁盘'test_log'上分配 0.49MB 的空间。
```

又例如，要创建一个 test1 数据库，并设定数据文件为“E:\SQL Server\测试数据 1.MDF”，大小为 10MB，最大为 50MB，每次增长 5MB。事务日志文件为“E:\SQL Server\测试数据 1 日志.MDF”，大小为 10MB，最大为 20MB，每次增长为 5MB。按 F5 键或单击▶按钮，系统提示相应的消息，如图 5.4 所示。

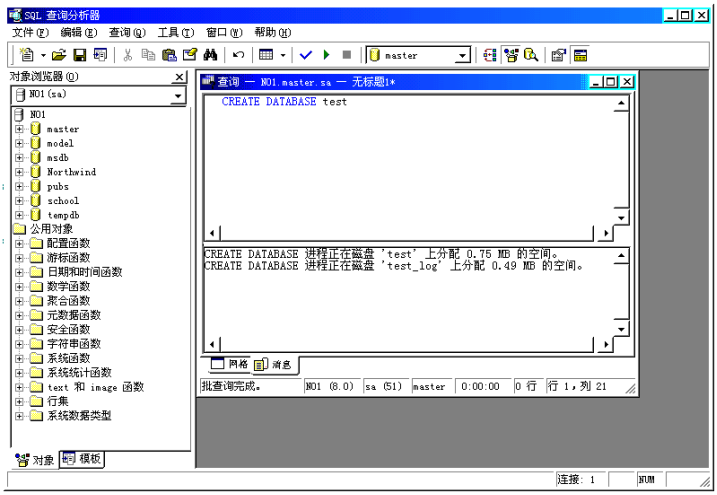


图 5.3　创建 test 数据库

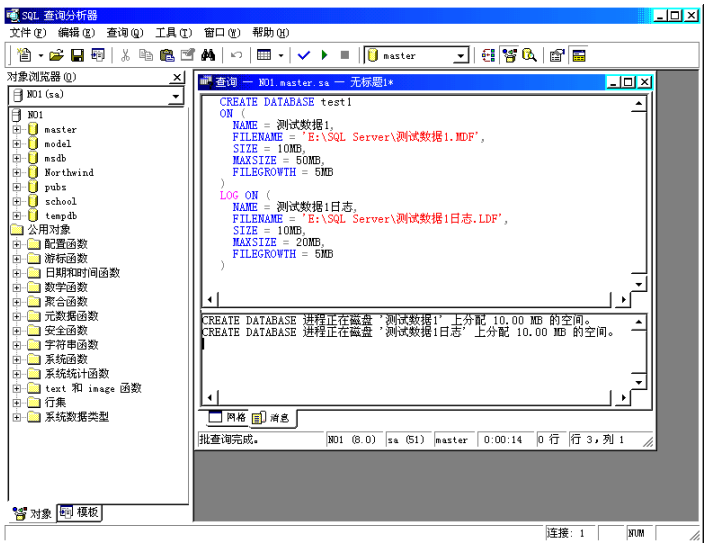


图 5.4　创建 test1 数据库

打开企业管理器，可看到新建立的数据库 test 和 test1。如果企业管理器处于打开状态，可执行“操作”菜单中的“刷新”命令，来查看新建立的数据库。

火龙果•整理
uml.org.cn

2．修改数据库

在建立数据库后，可根据需要修改数据库的设置。修改数据库可以使用 ALTER DATABASE 语句，该语句简化的语法格式如下：

```
ALTER DATABASE database
{ ADD FILE filespec
| ADD LOG FILE filespec
| REMOVE FILE logical_file_name
| MODIFY FILE filespec
| MODIFY NAME = new_dbname
}
filespec 定义为
( [ NAME = logical_file_name , ]
  FILENAME = 'os_file_name'
  [ , SIZE = size ]
  [ , MAXSIZE = { max_size | UNLIMITED } ]
  [ , FILEGROWTH = growth_increment ] )
```

其中，各参数和子句的说明如下：

- database 指出要更改的数据库的名称。
- ADD FILE 子句指定要添加文件。
- ADD LOG FILE 子句指定要将日志文件添加到指定的数据库。
- REMOVE FILE 指出从数据库系统表中删除文件描述并删除物理文件。只有在文件为空时才能删除。
- MODIFY FILE 指定要更改给定的文件，更改选项包括 FILENAME、SIZE、FILEGROWTH 和 MAXSIZE。一次只能更改这些属性中的一种。必须在“filespec”中指定 NAME，以标识要更改的文件。如果指定了 SIZE，那么新大小必须比文件当前大小要大。若要更改数据文件或日志文件的逻辑名称，应在 NAME 选项中指定要改名的逻辑文件名称，并在 NEWNAME 选项中指定文件的新逻辑名称。例如，MODIFY FILE (NAME = logical_file_name, NEWNAME = new_logical_name…)。可同时运行几个 ALTER DATABASE database MODIFY FILE 语句以实现多个修改文件操作时性能最优。
- MODIFY NAME = new_dbname，用于重命名数据库。

例如，为 test1 数据库新增一个逻辑名为“测试数据 2”的数据文件，其大小及其最大值分别为 10MB 和 50MB。输入的 SQL 语句和执行结果如图 5.5 所示。

3．使用和删除数据库

使用和删除数据库，可以使用 USE 和 DROP 语句。其语法如下：

```
USE DATABASE database_name
```

火龙果•整理
uml.org.cn


```
DROP DATABASE database_name
```

其中，“database_name”表示要使用或删除的数据库。

例如，可以使用如下 SQL 语句来删除 test1 数据库：

```
DROP DATABASE test1
```

执行结果如图 5.6 所示。

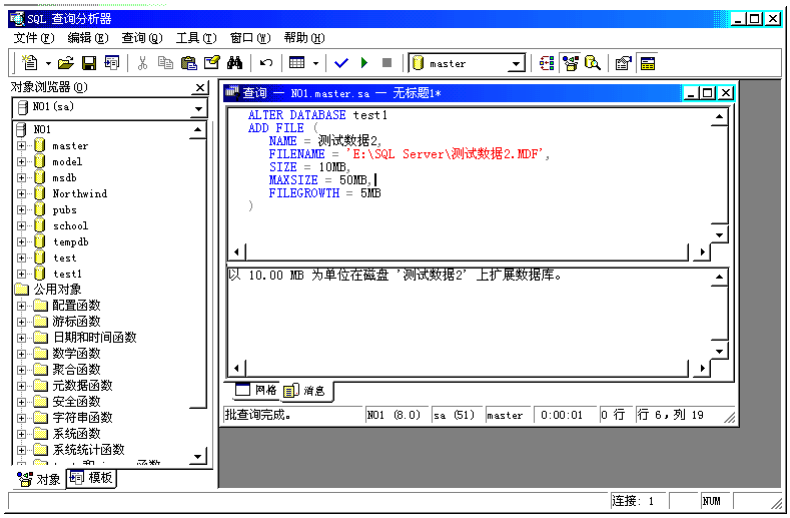


图 5.5 修改销售数据库

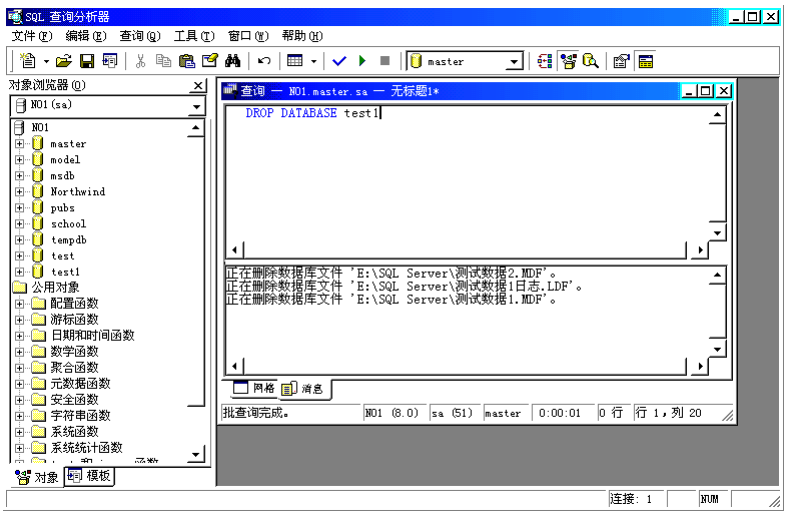


图 5.6 删除 test1 数据库

火龙果•整理
uml.org.cn

提示：如果不知道目前的 SQL Server 服务器中包含哪些数据库，可以执行 sp_helpdb 存储过程，使用方式为：EXEC sp_helpdb。如果后面再加上数据库名，则表示查询特定的数据库。

5.2.3 表的操作语句

同样，可以使用 SQL 语言创建表，也可以修改和删除表。

1．表的创建

使用 CREATE TABLE 语句来建立表，其语法如下：

```
CREATE TABLE table_name
(
   column_namel data_type [NULL | NOT NULL] [PRIMARY | UNIQUE]
                          [FOREIGN KEY [(column_name)]]
                          REFERENCES ref_table [(ref_column)]
   [column_name2 data_type …]
   ……
)
```

其中，各参数含义如下：

- table_name 要创建的表的名称。
- column_namel 第一个字段名称。
- data_type 指定字段的数据类型。

其余为字段属性设置参数，将在下面介绍。

(1) 基本用法

例如，在 test 数据库中创建一个 clients 表，SQL 语句如下：

```
USE test
CREATE TABLE clients (
   cid int,
   cname char(8),
   address char(50)
)
```

其中第 1 行表示使用 test 数据库，创建的表 clients 中包含 3 个字段：cid、cname 和 address。数据类型分别为整型、字符型(长度为 8)和字符型(长度为 50)。

提示：USE 语句只要在第一次使用即可，后续的 SQL 语句都是作用在该数据库中。若要使用其他的数据库，才需要再次执行 USE 语句。

火龙果•整理
uml.org.cn

(2) 段属性参数

除了可以设置字段的数据类型外，还可以利用一些属性参数来对字段做出限定。例如，将字段设置为主键，限制字段不能为空等。

常用的属性参数如下：

- NULL 和 NOT NULL　限制字段可以为 NULL(空)，或者不能为 NULL(空)。
- PRIMARY KEY　设置字段为主键。
- UNIQUE　指定字段具有惟一性。

例如，下面的 SQL 语句是在 test 数据库中建立一个 book 表，并指定 bid 为主键，而 bname 为非空：

```
CREATE TABLE book (
    bid int NOT NULL PRIMARY KEY,
    bname char(8) NOT NULL,
    authorid char(10)
)
```

(3) 与其他表建立关联

表的字段可能参考到其他表的字段，这就需要将两个表建立关联。此时，就可以使用如下的语法：

```
FOREIGN KEY REFERENCE ref_table(ref_column)
```

其中，“ref_table”指出要关联的表，“ref_column”指出要关联的字段名称。

例如，可以将 book 表中的 authorid 字段关联到 authors 表的 authorid 字段。在企业管理器中将前面创建的 book 表删除，然后执行下面的语句：

```
CREATE TABLE authors (
   authorid int NOT NULL PRIMARY KEY,
   authorname char(20),
   address char(30)
)

CREATE TABLE book (
    bid int NOT NULL PRIMARY KEY,
    bname char(8) NOT NULL,
    authorid int FOREIGN KEY REFERENCES authors(authorid)
)
```

上面的语句首先创建一个 authors 表，然后创建 book 表，并将 authorid 字段关联到 authors 表的 authorid 字段。

火龙果•整理
uml.org.cn

提示：在创建 book 表时，由于将 authorid 字段关联到了 authors 表，因此 authors 表必须存在。这也是上面首先创建 authors 表的原因。

另外，可以使用 SELECT INTO 语句创建一个新表，并用 SELECT 的结果集填充该表。新表的结构由选择列表中表达式的特性定义，例如，以下命令创建 student1 表，它包含 student 表的 sno、sname 和 class 等三个字段和对应的记录：

```
SELECT sno,sname,class INTO student1 FROM student
```

2．修改表

SQL 语言提供了 ALTER TABLE 语句来修改表的结构。基本语法如下：

```
ALTER TABLE table_name
  ADD [column_name data_type]
      [PRIMARY KEY | CONSTRAIN]
      [FOREIGN KEY (column_name)
  REFERENCES ref_table(ref_column)]
  DROP [CONSTRAINT] constraint_name | COLUMN column_name
```

其中，各参数含义如下：

- ADD 子句　增加字段，后面为属性参数设置。
- DROP 子句　删除限制或者字段。CONSTRAINT 表示删除限制；COLUMN 表示删除字段。

例如，在 test 数据库中给 book 表增加一个“price”字段：

```
ALTER TABLE book ADDprice int
```

3．删除关联和表

使用 SQL 语言要比使用企业管理器删除表容易得多。删除表的语法如下：

```
DROP TABLE table_name
```

其中，“table_name”指出要删除表的名称。
例如，要删除 book 表，可执行下述 SQL 语句：

```
DROP TABLE book
```

5.2.4　数据查询

数据库存在的意义在于将数据组织在一起，以方便查询。“查询”的含义就是用来描述从数据库中获取数据和操纵数据的过程。

SQL 语言中最主要、最核心的部分是它的查询功能。查询语言用来对已经存在于数据库中的数据按照特定的组合、条件表达式或者一定次序进行检索。其基本格式是由 SELECT 子句、FROM 子句和 WHERE 子句组成的 SQL 查询语句：

```
SELECT 列名表
```

火龙果•整理
uml.org.cn


```
FROM 表或视图名
WHERE 查询限定条件
```

也就是说，SELECT 指定了要查看的列(字段)，FROM 指定这些数据来自哪里(表或者视图)，WHERE 则指定了要查询哪些行(记录)。

> 提示：在 SQL 语言中，SELECT 子句除了进行查询外，其他的很多功能也都离不开 SELECT 子句，例如，创建视图是利用查询语句来完成的；插入数据时，在很多情况下是从另外一个表或者多个表中选择符合条件的数据。所以查询语句是掌握 SQL 语言的关健。

完整的 SELECT 语句的用法如下所示：

```
SELECT  列名表
[INTO 新表名]
FROM  表或视图名
[WHERE 查询限定条件]
[GROUP BY 分组表达式]
[HAVING 分组条件]
[ORDER BY 次序表达式[ASC|DESC]]
```

其中，带有方括号的子句均是可选子句，大写的单词表示 SQL 的关键字，而小写的单词或者单词组合表示表(视图)名称或者给定条件。

下面以 school 数据库(其中各表的数据见第 4 章的图 4.14～图 4.17)为例，来介绍各个子句的使用。为先在图 5.3 中的数据库下拉列表中选择“school”数据库名称，再在 SQL 语句的输入窗口中输入下面的各个 SELECT 语句。

1．投影查询

使用 SELECT 语句可以选择查询表中的任意列，其中，“列表名”指出要检索的列的名称，可以为一个或多个列。当为多个列时，中间要用“，”分隔。FROM 子句指出从什么表中提取数据，如果从多个表中取数据，每个表的表名都要写出，表名之间用“，”分隔开。

【例 5.1】查询 student 表中所有记录的 name、sex 和 class 列。

```
SELECT sname,ssex,class
FROM student
```

本例执行结果如下：

```
sname    ssex class
-------- ---- -----
李军      男   95033
陆君      男   95031
```

火龙果•整理
uml.org.cn


```
匡明        男    95031
王丽        女    95033
曾华        男    95033
王芳        女    95031
```

说明：在 SQL Server 中，SQL 语句的执行结果默认以网格方式显示，为了像上述文本方式显示结果，应选择工具栏的按钮，在出现的菜单中选择“文本显示结果”。以后的 SQL 语句的执行结果均以文本方式显示。

如果要去掉重复的显示行，可以在字段名前加上“DISTINCT”关键字来说明。

【例 5.2】查询教师所有的单位即不重复的 depart 列。

```
SELECT DISTINCT depart
FROM teacher
```

本例执行结果如下：

```
depart
----------
电子工程系
计算机系
```

当显示查询结果时，选择列通常是以原表中的列名作为标题显示。这些列名在建表时，出于节省空间的考虑通常较短，含义也模糊。为了改变查询结果中的显示的标题，可在列名后使用“AS 标题名”，在显示时便以该标题名来显示。

【例 5.3】查询 student 表的所有记录。

```
SELECT sno AS '学号',sname AS '姓名',ssex AS '性别',
       sbirthday AS '出生日期',class AS '班号'
FROM student
```

本例执行结果如下：

```
学号  姓名  性别  出生日期                    班号
----- ----- ---- ------------------------   -------
101   李军   男   1976-02-20 00:00:00.000   95033
103   陆君   男   1974-06-03 00:00:00.000   95031
105   匡明   男   1975-10-02 00:00:00.000   95031
107   王丽   女   1976-01-23 00:00:00.000   95033
108   曾华   男   1976-09-01 00:00:00.000   95033
109   王芳   女   1975-02-10 00:00:00.000   95031
```

火龙果•整理
uml.org.cn

2. 选择查询

选择查询就是指定查询条件，只从表提取或显示满足该查询条件的记录。为了选择表中满足查询条件的某些行，可以使用 SQL 命令中的 WHERE 子句。WHERE 子句的查询条件是一个逻辑表达式，它是由多个关系表达式通过逻辑运算符(AND、OR、NOT)连接而成的。

【例 5.4】查询 score 表中成绩在 60 到 80 之间的所有记录。

```
SELECT *
FROM score
WHERE degree BETWEEN 60 and 80
```

说明：BETWEEN m AND n 表示在指定的范围 m ~ n 内，在本章后面详细介绍。

本例执行结果如下：

```
sno   cno       degree
----- ---------- ---------
105   3-245      75
109   3-245      68
109   3-105      76
101   3-105      64
108   3-105      78
107   6-166      79
```

【例 5.5】查询 score 表中成绩为 85、86 或 88 的记录。

```
SELECT *
FROM score
WHERE degree IN(85,86,88)
```

说明：IN …表示在指定的列表中，在本章后面详细介绍。

本例执行结果如下：

```
sno   cno       degree
----- ---------- ---------
103   3-245      86
105   3-105      88
101   6-166      85
```

【例 5.6】查询 student 表中“95031”班或性别为“女”的学生记录。

```
SELECT *
FROM student
```

火龙果•整理
uml.org.cn


```
WHERE class='95031' or ssex='女'
```

本例执行结果如下：

```
sno   sname  ssex sbirthday                class
----- ----- ---- ------------------------ -----
103   陆君    男   1974-06-03 00:00:00.000   95031
105   匡明    男   1975-10-02 00:00:00.000   95031
107   王丽    女   1976-01-23 00:00:00.000   95033
109   王芳    女   1975-02-10 00:00:00.000   95031
```

3．排序查询

通过在 SELECT 命令中加入 ORDER BY 子句来控制选择行的显示顺序。ORDER BY 子句可以按升序(缺省或 ASC)、降序(DESC)排列各行，也可以按多个列来排序。

> 注意：ORDER BY 子句必须是 SQL 命令中的最后一个子句(除指定目的地子句外)。

【例 5.7】以下命令以 class 降序显示 student 表的所有记录。

```
SELECT *
FROM student
ORDER BY class DESC
```

本例执行结果如下：

```
sno    sname    ssex   sbirthday                class
----- -------- ---- ------------------------ -----
101     李军     男    1976-02-20 00:00:00.000   95033
107     王丽     女    1976-01-23 00:00:00.000   95033
108     曾华     男    1976-09-01 00:00:00.000   95033
109     王芳     女    1975-02-10 00:00:00.000   95031
105     匡明     男    1975-10-02 00:00:00.000   95031
103     陆君     男    1974-06-03 00:00:00.000   95031
```

【例 5.8】以下命令以 cno 升序、degree 降序显示 score 表的所有记录。

```
SELECT *
FROM score
ORDER BY cno,degree DESC
```

本例执行结果如下：

```
sno   cno       degree
----- ---------- ----------
103   3-105      92
107   3-105      91
105   3-105      88
108   3-105      78
```

火龙果•整理
uml.org.cn


```
109   3-105     76
101   3-105     64
103   3-245     86
105   3-245     75
109   3-245     68
101   6-166     85
108   6-166     81
107   6-166     79
```

4．使用聚合函数

聚合函数用于实现数据统计等功能，常月的聚合函数如表 5.1 所列。

表 5.1 聚合函数

函数名	功能
AVG	计算一个数值型列的平均值
COUNT	计算指定列中选择的项数，COUNT(*)统计查询输出的行数
MIN	计算指定列中的最小值
MAX	计算指定列中的最大值
SUM	计算指定列中的数值总和
STDEV	计算统计标准偏差
VAR	统计方差

【例 5.9】查询“95031”班的学生人数。

```
SELECT COUNT(*)AS '95031 班人数'
FROM student
WHERE class='95031'
```

本例执行结果如下：

```
95031班人数
-----------
3
```

【例 5.10】查询 score 表中的最高分的学生学号和课程号。

```
SELECT no AS '学号',cno AS '课程号', MAX(degree)AS '最高分' ;
FROM score
```

本例执行结果如下：

```
  学号     课程号        最高分
-----------------------------
  108     6-166           92
-----------------------------
```

【例 5.11】查询编号为“3-105”的课程的平均分。

火龙果•整理
uml.org.cn


```
SELECT AVG(degree)as '课程平均分' ;
FROM score WHERE cno='3-105'
```

本例执行结果如下：

```
 课程平均分
------------
     81.5
------------
```

上述例子中使用了聚合函数，我们还可以加上 GROUP BY 子句，通常一个聚合函数的范围是满足 WHERE 子句指定的条件的所有记录。在加上 GROUP BY 子句后，SQL 命令把查询结果按指定列分成集合组。当一个聚合函数和一个 GROUP BY 子句一起使用时，聚合函数的范围变成为每组的所有记录。换句话说，一个结果是由组成一组的每个记录集合产生的。

使用 HAVING 子句可以对这些组进一步加以控制。用这一子句定义这些组所必须满足的条件，以便将其包含在结果中。

当 WHERE 子句，GROUP BY 子句和 HAVING 子句同时出现在一个查询中时，SQL 的执行顺序如下：

(1) 执行 WHERE 子句，从表中选取行。
(2) 由 GROUP BY 对选取的行进行分组。
(3) 执行聚合函数。
(4) 执行 HAVING 子句选取满足条件的分组。

【例 5.12】查询 score 表中至少有 5 名学生选修的并以 3 开头的课程号的平均分数。

```
SELECT cno,AVG(degree)AS '平均分'
FROM score
WHERE cno LIKE '3%'
GROUP BY cno
HAVING COUNT(*)>5
```

本例执行结果如下：

```
cno       平均分
-------- ---------------
3-105     81.500000
```

【例 5.13】查询最低分大于 70，最高分小于 90 的 NO 列。

```
SELECT sno
FROM score
GROUP BY sno
HAVING MIN(degree)>70 and MAX(degree)<90
```

本例执行结果如下：

```
sno
-----
```

火龙果•整理
uml.org.cn


```
105
108
```

5．表的连接查询

在数据查询中，经常涉及到提取两个或多个表的数据，这就需要使用表的连接来实现若干个表数据的联合查询。

在一个查询中，当需要对两个或多个表连接时，可以指定连接列，在 WHERE 子句中给出连接条件，在 FROM 子句中指定要连接的表，其格式如下：

```
SELECT 列名 1，列名 2，…
FROM 表 1，表 2，…
WHERE 连接条件
```

对于连接的多个表通常存在公共列，为了区别是哪个表中的列，在连接条件中通过表名前缀指定连接列，例如，teacher.tno 表示 teacher 表的 tno 列，student.sno 表示 student 表的 sno 列，由此来区别连接列所在的表。

【例 5.14】查询所有学生的 sname、cno 和 degree 列。

```
SELECT student.sname,score.cno,score.degree
FROM student,score
WHERE student.sno=score.sno
```

本例执行结果如下：

```
sname    cno       degree
-------- ---------- -----------
陆君      3-245     86
匡明      3-245     75
王芳      3-245     68
陆君      3-105     92
匡明      3-105     88
王芳      3-105     76
李军      3-105     64
王丽      3-105     91
曾华      3-105     78
李军      6-166     85
王丽      6-166     79
曾华      6-166     81
```

SQL 为了简化输入，允许在查询中使用表的别名，以缩写表名，我们可以在 FROM 子句中为表定义一个临时别名，然后在查询中引用。

根据连接条件分为等值连接和非等值连接。所谓等值连接，是指表之间通过“等于”关系连接起来，产生一个连接临时表，然后对该临时表进行处理后生成最终结果。

【例 5.15】查询所有学生的 sno、cname 和 degree 列。

火龙果•整理
uml.org.cn


```
SELECT x.sno,y.cname,x.degree
FROM score x,course y
WHERE x.cno=y.cno
```

本例执行结果如下：

```
sno   cname              degree
----- ---------------- --------
103   操作系统           86
105   操作系统           75
109   操作系统           68
103   计算机导论         92
105   计算机导论         88
109   计算机导论         76
101   计算机导论         64
107   计算机导论         91
108   计算机导论         78
101   数字电路           85
107   数字电路           79
108   数字电路           81
```

【例 5.16】查询“95033”班所选课程的平均分。

```
SELECT y.cno,avg(y.degree)AS '平均分'
FROM student x,score y
WHERE x.sno=y.sno and x.class='95033'
GROUP BY y.cno
```

本例执行结果如下：

```
cno        平均分
---------- --------------
3-105      77.666666
6-166      81.666666
```

所谓非等值连接，是指表之间的连接关系不是“等于”，而是其他关系。通过指定的非等值关系将两个表连接起来的，产生一个连接临时表，然后对该临时表进行处理后生成最终结果。

【例 5.17】假设使用如下命令建立了一个 grade 的表：

```
CREATE TABLE grade(low int,upp int,rank char(1))
INSERT INTO grade VALUES(90,100,'A')
INSERT INTO grade VALUES(80,89,'B')
INSERT INTO grade VALUES(70,79,'C')
INSERT INTO grade VALUES(60,69,'D')
INSERT INTO grade VALUES(0,59,'E')
```

查询所有学生的 no，cno 和 rank 列：

火龙果•整理
uml.org.cn


```
SELECT sno,cno,rank
FROM score,grade
WHERE degree between low and upp
ORDER BY rank
```

本例执行结果如下：

```
sno   cno       rank
----- ---------- ----
103   3-105      A
107   3-105      A
103   3-245      B
105   3-105      B
101   6-166      B
108   6-166      B
105   3-245      C
109   3-105      C
108   3-105      C
107   6-166      C
109   3-245      D
101   3-105      D
```

在数据查询中有时需要将同一个表进行连接，这种连接称之为自连接，进行自连接就如同两个分开的表一样，可以把一个表的某行与同一表中的另一行连接起来。

【例 5.18】查询选学“3-105”课程的成绩高于“109”号学生成绩的所有学生记录，并按成绩从高到低排列。

```
SELECT x.cno,x.sno,x.degree
FROM score x,score y
WHERE x.cno='3-105' and x.degree>y.degree and y.sno='109' and y.cno='3-105'
ORDER BY x.degree DESC
```

本例执行结果如下：

```
cno       sno   degree
---------- ----- -----------
3-105     103   92
3-105     107   91
3-105     105   88
3-105     108   78
```

【例 5.19】查询成绩高于学号“109”的课程号“3-105”的成绩的所有记录，并按成绩从低到高排列。

```
SELECT x.cno,x.sno,x.degree
FROM score x,score y
WHERE x.degree>y.degree and y.sno='109' and y.cno='3-105'
ORDER BY x.degree
```

火龙果•整理
uml.org.cn

本例执行结果如下：

```
cno       sno   degree
---------- ----- -----------
3-105     108   78
6-166     107   79
6-166     108   81
6-166     101   85
3-245     103   86
3-105     105   88
3-105     107   91
3-105     103   92
```

6．子查询

当一个查询是另一个查询的条件时，称之为子查询。子查询可以使用几个简单命令构造功能强大的复合命令。子查询最常用于 SQL 命令的 WHERE 子句中，我们按照子查询返回单个值还是一组值(此时子查询前应接关键字 IN、ANY 或 ALL 等)、查询一个表还是多个表进行分类，分别举例说明子查询的形式。

【例 5.20】查询和学号为 105 的学生同年出生的所有学生的 no、name 和 birthday 列。

```
SELECT sno,sname,sbirthday
FROM student
WHERE year(sbirthday)=
   (SELECT year(sbirthday)
     FROM student
     WHERE sno='105')
```

本例执行结果如下：

```
sno   sname    sbirthday
----- -------- -----------------------
105   匡明      1975-10-02 00:00:00.000
109   王芳      1975-02-10 00:00:00.000
```

本例的执行过程是：先执行以下子查询：

```
SELECT year(sbirthday) FROM student WHERE sno='105'
```

其返回结果为 1975，再执行主查询：

```
SELECT sno,sname,sbirthday
FROM student
WHERE year(sbirthday)=1975
```

这样得到本例的结果。

我们知道，一个查询可以涉及多个表的联合数据，一个子查询也可以使用表的连接方法从多个表中提取数据。但在这里，该子查询尽管从多个表提取数据，但也只能有一个返回值。

火龙果•整理
uml.org.cn

【例 5.21】查询“张旭”教师任课的学生成绩，并按成绩递增排列。

```
SELECT cno,sno,degree
FROM score
WHERE cno=
   ( SELECT x.cno
     FROM course x,teacher y
     WHERE x.tno=y.tno and y.tname='张旭')
ORDER BY degree DESC
```

本例执行结果如下：

```
cno       sno   degree
---------- ----- ----------
6-166     101   85
6-166     108   81
6-166     107   79
```

上面的查询，返回值不论来自一个表还是多个表，都只返回单个值，即返回表中的某一行，如果返回多值，则要使用本节介绍的 ANY、ALL、IN 和 NOT IN 等，它们与查询条件一起构造返回一组值的子查询。

【例 5.22】查询选修某课程的学生人数多于 5 人的教师姓名。

```
SELECT tname
FROM teacher
WHERE tno IN
     (SELECT x.tno
      FROM course x,score y
      WHERE x.cno=y.cno
      GROUP BY x.tno
      HAVING COUNT(x.tno)>5)
```

本例执行结果如下：

```
tname
--------
王萍
```

【例 5.23】查询存在有 85 分以上成绩的课程 cno。

```
SELECT DISTINCT cno
FROM score
WHERE degree IN
    (SELECT degree
     FROM score
     WHERE degree>85)
```

本例执行结果如下：

```
cno
----------
```

火龙果•整理
uml.org.cn


```
3-105
3-245
```

【例 5.24】查询“计算机系”中与“电子工程系”不同职称的教师 name 和 prof。

```
SELECT tname,prof
FROM teacher
WHERE depart='计算机系' and prof NOT IN
   (SELECT prof
    FROM teacher
    WHERE depart='电子工程系')
```

本例执行结果如下：

```
tname    prof
-------- ----------
李诚       副教授
```

【例 5.25】查询选修编号为“3-105”的课程，且成绩至少高于编号为“3-245”的学生的 cno、no 和 degree，并按 degree 从高到低次序排列。

```
SELECT cno,sno,degree
FROM score
WHERE cno='3-105' and degree> ANY
    (SELECT degree
     FROM score
     WHERE cno='3-245')
ORDER BY degree DESC
```

本例执行结果如下：

```
cno       sno   degree
---------- ----- --------------------
3-105     103   92
3-105     107   91
3-105     105   88
3-105     108   78
3-105     109   76
```

【例 5.26】查询选修编号为“3-105”的课程，且成绩高于所有编号为“3-245”成绩的学生的 cno、no 和 degree，并按 degree 从高到低次序排列。

```
SELECT cno, sno,degree
FROM score
WHERE cno='3-105' and degree>ALL
    (SELECT degree
      FROM score
      WHERE cno='3-245')
ORDER BY degree DESC
```

本例执行结果如下：

火龙果•整理
uml.org.cn


```
cno        sno   degree
---------- ----- ----------
3-105      103   92
3-105      107   91
3-105      105   88
```

SQL 命令还提供了 UNION 子句，它可以将多个 SQL 命令连接起来生成单个 SQL 无法做到的结果集合。VFP 将第二次查询的结果附加到第一个结果上。

注意：在 VFP 中使用 UNION 子句遵守下列规则：

- 使用 UNION 连接的 SELECT 必须有相同的输出表达式格式，即对应栏目具有相同的数据类型的宽度。
- 仅最后一个 SELECT 可以带 ORDER BY 及 INTO 子句，如果带 ORDER BY，它影响整个显示结果。
- UNION 不能连接嵌套的 SELECT 语句。

【例 5.27】查询所有教师和学生的 name、sex 和 birthday。

```
SELECT tname AS '姓名',tsex AS '性别',tbirthday AS '出生日期'
FROM teacher
UNION
SELECT sname,ssex,sbirthday
FROM student
```

本例执行结果如下：

```
姓名       性别   出生日期
-------- ---- ---------------------------
匡明       男    1975-10-02 00:00:00.000
李诚       男    1958-12-02 00:00:00.000
李军       男    1976-02-20 00:00:00.000
刘冰       女    1977-08-14 00:00:00.000
陆君       男    1974-06-03 00:00:00.000
王芳       女    1975-02-10 00:00:00.000
王丽       女    1976-01-23 00:00:00.000
王萍       女    1972-05-05 00:00:00.000
曾华       男    1976-09-01 00:00:00.000
张旭       男    1969-03-12 00:00:00.000
```

本例的结果将两个 SELECT 命令的结果已按照姓名进行了排序。

【例 5.28】查询所有“女”教师和“女”学生的 name、sex 和 birthday。

```
SELECT tname AS '姓名',tsex AS '性别',tbirthday AS '出生日期'
FROM teacher
WHERE tsex='女'
UNION
```

火龙果•整理
uml.org.cn


```
SELECT sname,ssex,sbirthday
FROM student
WHERE ssex='女'
```

本例执行结果如下：

```
姓名       性别   出生日期
-------- ---- ------------------------------------
刘冰       女    1977-08-14 00:00:00.000
王芳       女    1975-02-10 00:00:00.000
王丽       女    1976-01-23 00:00:00.000
王萍       女    1972-05-05 00:00:00.000
```

本例的结果已将两个 SELECT 命令的结果按姓名进行了排序。

7. 相关子查询

在前面的例子中，每个子查询仅执行一次，返回的值为主查询的 WHERE 子句所用。在有的查询中，子查询不只执行一次，例如，要显示其成绩比该课程平均成绩高的成绩表，其主查询为：

```
SELECT sno,cno,degree
FROM score
WHERE degree >(待选学生所修课程的平均分)
```

该子查询为：

```
SELECT AVG(degree)
FROM score
WHERE cno=(主查询待选行的课程号 cno)
```

这样，主查询在判断每个待选行时，必须“唤醒”子查询，告诉它该学生选修的课程号，并由子查询计算课程的平均成绩，然后将该学生的 degree 与平均成绩进行比较，找出相应的符合条件的行，我们把这种子查询称为相关子查询。

【例 5.29】查询成绩比该课程平均成绩低的学生成绩表。

```
SELECT sno,cno,degree
FROM score a
WHERE degree<
   (SELECT AVG(degree)
    FROM score b
    WHERE a.cno=b.cno)
```

本例执行结果如下：

```
sno   cno       degree
----- ---------- ----------
101   3-105      64
```

火龙果•整理
uml.org.cn


```
109   3-105     76
108   3-105     78
109   3-245     68
105   3-245     75
107   6-166     79
108   6-166     81
```

理解上述相关子查询的关键是别名，它出现在主查询“FROM score a”和子查询“FROM score b”中。这样同一个表相当于两个表，当在子查询中使用 a.cno 时，它访问待选行的 cno，这时是一个常量，从而在 b 别名中找出该常量课程的平均分。由于这个过程很费时，因此不要对大于使用相关子查询。

8．带 EXISTS 测试的子查询

在子查询中，还可以使用 EXISTS，它一般用在 WHERE 子句中，其后紧跟一个 SQL 子查询，从而构成一个条件，当该子查询至少存在一个返回值时，这个条件为真(.T.)，否则为假(.F.)。

注意：使用 EXISTS 引入的子查询在以下几方面与其它子查询略有不同：

- EXISTS 关键字前面没有列名、常量或其它表达式。
- 由 EXISTS 引入的子查询的选择列表通常几乎都是由星号(*)组成。由于只是测试是否存在符合子查询中指定条件的行，所以不必列出列名。

【例 5.30】查询所有任课教师的 tname 和 depart。

```
SELECT tname,depart
FROM teacher a
WHERE EXISTS
   (SELECT *
    FROM course b
    WHERE a.tno=b.tno)
```

本例执行结果如下：

```
tname    depart
-------- ----------
李诚       计算机系
王萍       计算机系
张旭       电子工程系
```

本例也有一个相关子查询，其执行过程是：从头到尾扫描 teacher 表的行，对于每个行，执行子查询，此时，a.tno 是一个常量，子查询便是在 course 表中查找 tno 等于该常量的行，如果存在这样的行，EXISTS 子句便返回真，主查询屏幕显示 teacher 表中的当前行；如果子查询未找到这样的行，EXISTS 子句返回假，主查询不显示 teacher 表中的当前行。主查询继续查找 teacher 表的下一行，其过程与上述相同。

【例 5.31】查询所有未讲课的教师的 tname 和 depart。

火龙果•整理
uml.org.cn


```
SELECT tname,depart
FROM teacher a
WHERE NOT EXISTS
(SELECT *
   FROM course b
   WHERE a.tno=b.tno)
```

本例执行结果如下：

```
tname    depart
-------- ----------
刘冰       电子工程系
```

本例的执行过程与上例基本相同，只是将 EXISTS 子句的结果取反，当子查询找到了这样的行，WHERE 条件为假；当子查询未找到这样的行，则 WHERE 条件为真，所示查询结果与上例正好相反。

5.3 Transact-SQL 程序设计基础

SQL 虽然和高级语言不同，但是它本身也具有运算、流控制等功能，也可以利用 SQL 语言进行编程。因此，就需要了解 SQL 语言的基础知识。本节主要介绍 Transact-SQL 语言程序设计的基础概念。

5.3.1 标识符

在 SQL Server 中，标识符就是指用来定义服务器、数据库、数据库对象和变量等的名称。可以分为常规标识符和分隔标识符。

1．常规标识符

常规标识符就是不需要使用分隔标识符进行分隔的标识符 。常规标识符符合标识符的格式规则。在 Transact-SQL 语句中使用常规标识符时不用将其分隔。

例如，以下语句中 book 和 bname 就是两个常规标识符：

```
SELECT * FROM book WHERE bname='C程序设计'
```

在 SQL Server 2000 中，常规标识符的格式规则取决于数据库的兼容级别，兼容级别可以用存储过程 sp_dbcmptlevel 来设置。当兼容级别为 80 时，规则如下：

- 第一个字符必须是 Unicode 2.0 标准所定义的字母、下划线(_)、at 符号(@)和数字符号(#)。
- 后续字符可以是 Unicode 2.0 所定义的字母、来自基本拉丁字母或其他国家／地区脚本的十进制数字、at 符号(@)、美元符号($)、数字符号(#)或下划线(_)。
- 标识符不能是 Transact-SQL 的保留字。SQL Server 保留其保留字的大写和小写形式。

火龙果•整理
uml.org.cn

- 不允许嵌入空格或其他特殊字符。
- 常规标识符和分隔标识符包含的字符数必须在 1～128 之间。对于本地临时表，标识符最多可以有 116 个字符。

注意：在 SQL Server 中，某些处于标识符开始位置的符号具有特殊意义。以 at 符号(@)开始的标识符表示局部变量或参数; 以双 at 符号(@@)开始的标识符表示全局变量。以一个数字符号(#)开始的标识符表示临时表或过程。以双数字符号(##)开始的标识符表示全局临时对象。

2．分隔标识符

在 Transact-SQL 语句中，对不符合所有标识符规则的标识符必须进行分隔。符合标识符格式规则的标识符可以分隔，也可以不分隔。

例如，下面语句中的 My Table 和 order 均不符合标识符规则，其中 My Table 中间出现了空格，而 order 为 Transact-SQL 的保留字，因此必须使用分隔符中括号([])进行分隔：

```
SELECT * FROM  [My Table] WHERE [order]=10
```

分隔标识符在下列情况下使用：

- 当在对象名称或对象名称的组成部分中使用保留字时　推荐不要使用保留关键字作为对象名称。从 SQL Server 早期版本升级的数据库可能含有标识符，这些标识符包括早期版本中未保留而在 SQL Server 2000 中保留的字。可用分隔标识符引用对象直到可改变其名称。
- 当使用未被列为合法标识符的字符时　SQL Server 允许在分隔标识符中使用当前代码页中的任何字符。但是，不加选择地在对象名称中使用特殊字符将使 SQL 语句和脚本难以阅读和维护。

在 SQL Server 中，Transact-SQL 所使用的分隔标识符类型有下面两种：

- 被引用的标识符用双引号(")分隔开 例如：SELECT * FROM "My Table"。
- 括在括号中的标识符用方括号([ ])分隔例如：SELECT * FROM [My Table]。

分隔标识符的格式规则是：

- 分隔标识符可以包含与常规标识符相同的字符数(1～128 个，不包括分隔符字符)。本地临时表标识符最多可以包含 116 个字符。
- 标识符的主体可以包含当前代码页内字母(分隔符本身除外)的任意组合。例如，分隔符标识符可以包含空格、对常规标识符有效的任何字符以及下列任何字符：代字号(~)、连字符(-)、惊叹号(！)、左括号({)、百分号(%)、右括号(})、插入号(^)、撇号(')、and 号(&)、句号(.)、左圆括号(()、反斜杠(\)、右圆括号())、重音符号(`)。

使用引号分隔标识符时，仅当 QUOTED_IDENTIFIER 选项设置为 ON 时才有效。QUOTED_IDENTIFIER 称作连接选项。在默认情况下，当用于 SQL Server 的 Microsoft OLE DB 提供程序和 SQL Server ODBC 驱动程序连接时，将 QUOTED_IDENTIFIER 设置为 ON。

火龙果•整理
uml.org.cn

在默认情况下，DB-Library 不将 QUOTED_IDENTIFIER 设置为 ON。不管使用何种接口，个别应用程序或用户可随时更改设置。

SQL Server 提供了多种方法来指定该选项。例如，在 SQL Server 企业管理器和 SQL 查询分析器中，该选项可在对话框中设置。在 Transact-SQL 中，可以使用 SET QUOTED_IDENTIFIER、sp_dboption 的 quoted identifier 选项或 sp_configure 的 user options 选项将此选项设为多种级别。例如，打开和关闭该选项的 SET 语句如下：

```
SET QUOTED_IDENTIFIER ON

SET QUOTED_IDENTIFIER OFF
```

注意：如果打开了 QUOTED_IDENTIFIER 选项，双引号只能用于分隔标识符，不能用于分隔字符串；单引号必须用来包含字符串，不能用于分隔标识符。如果关闭了该选项，引号不能用于分隔标识符，而是用括号作为分隔符。单引号或双引号可用于包含字符串。

3．使用标识符

数据库对象的名称被看成是该对象的标识符。SQL Server 中的每一内容都可带有标识符。服务器、数据库和数据库对象(例如表、视图、列、索引、触发器、过程、约束、规则等)都有标识符。大多数对象要求带有标识符，但对有些对象(如约束)，标识符是可选项。

在 SQL Server 2000 中，一个对象的全称语法格式为：

```
server.database.owner.object
```

其中，server 为服务器名，database 为数据库名，owner 为所有者，object 为对象名。例如，在服务器 MyServer 中，test 数据库中的 sysusers 表的全称就是：

```
MyServer.test.dbo.sysusers
```

在实际使用时，使用全称比较繁琐，因此经常使用简写格式。可用的简写格式包含下面几种：

```
server.database..object
server..owner.object
server…object
database.ownerobject
database..object
owner.object
object
```

在上面的简写格式中，没有指明的部分使用如下的默认设置值：

- 服务器　本地服务器。
- 数据库　当前数据库。
- 所有者　在指定的数据库中与当前连接会话的登录标识相对应的数据库用户或者

火龙果•整理
uml.org.cn

数据库所有者。

例如，一个用户名为 bookadm 的用户登录到 MyServer 服务器上，并使用 book 数据库。使用下述语句创建了一个 MyTable 表：

```
CREATE TABLE MyTable
(
    columnl int,
    column2 char(20)
)
```

则表 MyTable 的全称就是 MyServer.book.bookadm.MyTable。

5.3.2 数据类型

数据类型是指列、存储过程参数、表达式和局部变量的数据特征，它决定了数据的存储格式，代表了不同的信息类型。包含数据的对象都具有一个相关的数据类型，此数据类型定义对象所能包含的数据种类(字符、整数、二进制数等)。

SQL Server 提供了各种系统数据类型，在 2000 版本中，又新增了几种系统数据类型。除了系统数据类型外，在 SQL Server 中，还可以自定义数据类型。用户定义的数据类型是在系统数据类型的基础上，使用存储过程 sp_addtype 所建立的数据类型。

> 提示：在 SQL Server 2000 中，所有系统数据类型名称都是不区分大小写的。另外，用户定义数据类型是在已有的系统数据类型基础上生成的，而不是定义一个存储结构的新类型。

在 SQL Server 2000 中，以下对象可以具有数据类型：

- 表和视图中的列
- 存储过程中的参数
- 变量
- 返回一个或多个特定数据类型数据值的 Transact-SQL 函数
- 具有一个返回代码的存储过程(返回代码总是具有 integer 数据类型)

指定对象的数据类型定义了该对象的 4 个特性：

- 对象所含的数据类型，如字符、整数或二进制数。
- 所存储值的长度或它的大小。image、binary 和 varbinary 数据类型的长度以字节定义。任何数字数据类型的长度是指保存此数据类型所允许的数字个数所需要的字节数。string 和 Unicode 数据类型的长度以字符数定义。
- 数字精度(仅用于数字数据类型)。精度是数字可以包含的数字个数。例如，smallint 对象最多能拥有 5 个数字，所以其精度为 5。
- 数值小数位数(仅用于数字数据类型)。小数位数是能够存储在小数点右边的数字个数。例如，int 对象不能含有小数点，小数位数为 0。money 对象的小数点右边最

火龙果•整理
uml.org.cn

多可以有 4 个数字，小数位数为 4。

1．系统数据类型

可以按照存放在数据库中的数据的类型对 SQL Server 提供的系统数据类型进行分类，如表 5.2 所示。

表 5.2 SQL Server 2000 提供的系统数据类型

分类	数据类型定义符
整数型	bigint、int、smallint、tinyint
逻辑数值型	bit
小数数据类型	decimal、numeric
货币型	money、smallmoney
近似数值型	float、real
字符型	char、varchar、text
Unicode 字符型	Nchar、nvarchar、ntext
二进制数据类型	binary、varbinary、image
日期时间类型	datetime、smalldatetime
其他数据类型	cursor、sal_variant、table、timestamp、uniqueidentifier

(1) 整数型

整数类型包括 bigint、int、smallint 和 tinyint 4 种，其中 bigint 是 SQL Server 2000 中新增的一种类型。

整型数据由负整数或正整数组成，如-15、0、5 和 2509。在 SQL Server 2000 中，整型数据使用 bigint、int、smallint 和 tinyint 数据类型存储。bigint 数据类型可存储的数字范围比 int 数据类型广。int 数据类型比 smallint 数据类型的存储范围大，而 smallint 的数值范围又比 tinyint 类型大。

各种类型能存储的数值的范围如下：

- bigint 数据类型　大整数型，长度为 8 个字节，可以存储从-2^{63}(-9 223 372 036 854 775 808)到$2^{63}-1$((9 223 372 036 854 775 807)范围内的数字。
- int 数据类型　整数型，长度为 4 个字节，可存储范围是-2^{31} (-2 147 483 648)到$2^{31}-1$(2 147 483 647)。
- smallint 数据类型　短整数型，长度为 2 个字节，可存储范围只有-2^{31} (-32 768)到$2^{31}-1$(32 767)。
- tinyint 数据类型　微短整数型，长度为 1 个字节，只能存储 0～255 范围内的数字。

例如，下面的语句创建了一个表 Int_table，其中的 4 个字段分别使用这 4 种整数类型：

```
USE test
CREATE TABLE Int_table
(
```

火龙果•整理
uml.org.cn


```
    cl tinyint,
    c2 smallint,
    c3 int,
    c4 bigint
)
INSERT Int_table VALUES (50,5000,50000,500000)
SELECT * FROM Int_table
```

执行结果是显示出表 Int_table 中的一笔记录，它包含了 4 种数据类型。

注意：上面的 SQL 语句可以在自己另外创建的数据库中进行，对于下面的例子，本书都在 test 数据库中进行。

(2) 小数数据类型

也称为精确数据类型，它们由两部分组成，其数据精度保留到最低有效位，所以它们能以完整的精度存储十进制数。

在声明小数数据类型时，可以定义数据的精度和小数位。声明格式如下：

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

和

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

各参数含义如下：

- p(精度)　指定小数点左边和右边可以存储的十进制数字的最大个数。精度必须是从 1 到最大精度之间的值。最大精度为 38。使用最大精度时，有效值从$-10^{38}+1$ 到 $10^{38}-1$。
- s(小数位数)　指定小数点右边可以存储的十进制数字的最大个数。小数位数必须是从 0～p 之间的值。默认小数位数是 0，因而 0≤s≤p。最大存储大小基于精度而变化。

例如，在下面的表 Decimal_table 中，字段 cl 就是一个 decimal 数据类型：

```
CREATE TABLE Decimal_table
(
    cl decimal(3,2)
)
INSERT Decimal_table VALUES (4.5678)
SELECT * FROM Decimal_table
```

执行结果如下：

```
cl
------
```

火龙果•整理
uml.org.cn


```
4.57
```

在为小数数值型数据赋值时，应保证所赋数据整数部分的位小于或者等于定义的长度，否则会出现溢出错误。

例如，给 Decimal_table 插入一笔记录，在 SQL 查询分析器中执行下面语句：

```
INSERT Decimal_table VALUES (45.678)
```

执行出现错误，并出现如下消息：

```
服务器：消息 8115，级别 16，状态 8，行 1
将 numeric 转换为数据类型 numeric 时发生算术溢出错误。
语句已终止。
```

这是由于 45.678 的整数部分超出了定义的长度造成的。

在 SQL Server 中，小数数据使用 decimal 或 numeric 数据类型存储。存储 decimal 或 numeric 数值所需的字节数取决于该数据的数字总数和小数点右边的小数位数。例如，存储数值 19283.29383 比存储 1.1 需要更多的字节。具体存储长度随其精度的变化而改变，如表 5.3 所示。

表 5.3 存储字节长度和数据精度的关系

精度	存储字节长度
1～9	5
10～19	9
20～28	13
29～38	17

提示：在 SQL Server 中，numenc 数据类型等价于 decimal 数据类型。但是只有 numenc 可以用于带有 identity 关健字的列(字段)。identity 指示为标识列，这种类型的列包含由系统自动生成的能够标识表中每一行数据的惟一序列值。这种机制在某些情况下很有用，比如学生基本情况表中，需要有一列存放学生编号，最简单的方法就是把它作为标识列，这样每次向表中插入一条学生记录时，SQLServer 都会自动生成惟一的值作为学生编号，就可以避免人工添加序号带来的序号冲突问题。

(3) 近似数值型

并非数据类型范围内的所有数据都能精确地表示，因此 SQL Server 提供了用于表示浮点数字数据的近似数值数据类型。

近似数值数据类型不能精确记录数据的精度，它们所保留的精度由二进制数字系统的精度决定。SQL Server 提供了两种近似数值数据类型：

- float [(n)]　-1.79^{308}～1.79308 之间的浮点数字数据。n 为用于存储科学记数法 float 数尾数的位数，同时指示其精度和存储大小。n 必须为 1～53 之间的值，它同精度

火龙果•整理
uml.org.cn

和存储字节的关系如表 5.4 所示。

- real 数据类型　-3.40^{38}～3.40^{38} 之间的浮点数字数据。存储大小为 4 字节。

表 5.4　n 与精度和存储字节之间的关系

n	精度	存储字节长度
1～24	7 位数	4 字节
25～53	15 位数	8 字节

提示：float 和 real 通常按照科学记数法来表示，即以 1.79E+38 的方式表示。

(4) 字符型

字符串存储时采用字符型数据类型。字符数据由字母、符号和数字组成。例如，"928 "、"Johnson”和“(0*&(%B99nh jkJ”都是有效的字符数据。

提示：字符常量必须包括在单引号(')或双引号(")中。建议用单引号括住字符常量。因为当 QUOTED IDENTIFIER 选项设为 ON 时，有时不允许用双引号括住字符常量。当使用单引号分隔一个包括嵌入单引号的字符常量时，用两个单引号表示嵌入单引号。

在 SQL Server 中，字符数据使用 char、varchar 和 text 数据类型存储。当列中各项的字符长度可变时可用 varchar 类型，但任何项的长度都不能超过 8 KB。当列中各项为同一固定长度时使用 char 类型(最多 8KB)。text 数据类型的列可用于存储大于 8KB 的 ASCII 字符。例如，由于 HTML 文档均由 ASCII 字符组成且一般长于 8KB，所以用浏览器查看之前应在 SQL Server 中存储在 text 列中。

建议字符列宽度的定义不超过所存储的字符数据可能的最大长度，以节省存储空间。

另外，若要在 SQL Server 中存储国际化字符数据，请使用 nchar、nvarchar 和 ntext 数据类型。支持多语言的网络站点应考虑使用 Unicode 的 nchar 或 nvarchar 数据类型，以尽量减少字符转换问题。如果希望列中的数据值大小接近一致，或者列中的数据值大小显著不同，则可以考虑使用 char 或 varchar。

char、varchar 和 text 3 种类型的定义方式如下：

- char[(n)]　长度为 n 个字节的固定长度且非 Unicode 的字符数据。n 必须是一个介于 1～8000 之间的数值。存储大小为 n 个字节。
- varchar[(n)]　长度为 n 个字节的可变长度且非 Unicode 的字符数据。n 必须是一个介于 1～8 000 之间的数值。存储大小为输入数据的字节的实际长度，而不是 n 个字节。所输入的数据字符长度可以为零。
- text 数据类型　也是用来声明变长的字符数据。在定义过程中，不需要指定字符的长度。最大长度为 2^{31}-1(2 147 483 647)个字符。当服务器代码页使用双字节字符时，存储量仍是 2 147 483 647 个字节。存储大小可能小于 2 147 483 647 字节(取决于字符串)。SQL Server 会根据数据的长度自动分配空间。

例如，下面的 SQL 语句将局部变量 MyCharVar 声明为 char 类型，长度为 25：

火龙果•整理
uml.org.cn


```
DECLARE  @MyCharVar CHAR(25)
SET @MyCharVar='This is a string'
```

下面则是使用两个单引号来表示嵌入单引号：

```
SET @MyCharVar='This is a "string"'
```

如果要存储的数据比允许的字符数多，则数据就会被截断。例如，如果某列被定义为char(10)并且值"This is a really long character string"被存储到该列中，则 SQL Server 将该字符串截断为"This is a"。

CHAR 函数可以把一个整数转换为 ASCII 字符。当确定控制字符时(比如回车或换行)，这是很有用的。在字符串中用 CHAR(13)和 CHAR(10)产生一个回车并生成一个新行。例如：

```
PRINT 'First line.'+CHAR(13)+CHAR(10)+'Second line.'
```

执行结果为：

```
First line.
Second line.
```

提示：PRINT 函数用于打印字符串，将其显示在屏幕上。

(5) 逻辑数值型

SQL Server 支持逻辑数据类型 bit，它可以存储整型数据 1、0 或 NULL。如果输入 0 以外的其他值时，SQL Server 均将它们当作 1 看待。

SQL Server 优化用于 bit 列的存储。如果一个表中有不多于 8 个的 bit 列，这些列将作为一个字节存储。如果表中有 9～16 个 bit 列，这些列将作为两个字节存储。更多列的情况依此类推。

注意：不能对 bit 类型的列(字段)使用索引。

例如，下面是使用 bit 数据类型的例子：

```
CREATE TABLE Bit_table
(
   cl bit,
   c2 bit,
   c3 bit
)
INSERT Bit_table VALUES(12,1,0)
SELECT * FROM Bit_table
```

执行结果为：

```
cl   c2  c3
--- --- ---
```

火龙果•整理
uml.org.cn


```
1    1   0
```

(6) 货币型

货币数据表示正的或负的货币值。在 SQL Server 中使用 money 和 smallmoney 数据类型存储货币数据。货币数据存储的精确度为 4 位小数。

money 和 smallmoney 数据类型存储范围和占用字节如下：

- money 数据类型　可存储的货币数据值介于-2^{63} (-922337203685477.5808)与 $2^{63}-1$ (+922337203685477.5807)之间，精确到货币单位的千分之十。存储大小为 8 个字节。
- smallmoney 数据类型　可存储的货币数据值介于-214 748.3648 与+214.748 3647 之间，精确到货币单位的千分之十。存储大小为 4 个字节。

货币数据不需要用单引号(')括起来。但是，货币数值之前必须带有适当的货币符号。例如，若要指定 100 英镑，则使用￡100。若要指定 100 美元，则使用$100。不同货币的符号可参考 SQL Server 2000 的帮助文档。

下面是使用货币数据类型的例子：

```
CREATE TABLE Money_table
(
    c1 money,
    c2 smallmoney
)
INSERT Money_table VALUES ($12345678,$1234)
SELECT * FROM Money_table
```

执行结果如下：

```
c1               c2
---------------- -------------
12345678.0000    1234.0000
```

(7) 二进制数据类型

二进制数据由十六进制数表示。例如，十进制数 245 等于十六进制数 F5。在 SQL Server 2000 中，二进制数据使用 binary、varbinary 和 image 数据类型存储：

- binary 数据类型的列　在每行中都是固定的长度(最多为 8KB)。
- varbinary 数据类型的列　在每行中所包含的十六进制数字的个数可以不同(最多为 8 KB)。
- image 数据类型列　可以用来存储超过 8 KB 的可变长度的二进制数据，如 Word 文档、Excel 电子表格、位图图像、图形交换格式(GIF)文件和联合图像专家组(JPEG)文件。

火龙果•整理
uml.org.cn

声明格式如下：

- binary[(n)]　固定长度的 n 个字节二进制数据。n 必须从 1～8000。存储空间大小为 n+4 个字节。
- varbinary[(n)]　n 个字节变长二进制数据。n 必须从 1～8000。存储空间大小为实际输入数据长度+4 个字节，而不是 n 个字节。输入的数据长度可能为 0 字节。
- image　可变长度二进制数据在 0～2^{31}-1 (2 147 483 647)字节之间。

二进制常量以 0x(一个零和小写字母 x)开始，后面跟着位模式的十六进制表示。例如，0x2A 表示十六进制的值 2A，它等于十进制的数 42 或单字节位模式 00101010。

下面是使用二进制数据类型的例子：

```
CREATE TABLE Binary_table
(
  c1 binary(10),
  c2 varbinary(20),
  c3 image
)
INSERT Binary_table VALUES (0x123,0xffff,0x14ffff)
SELECT * FROM Binary_table
```

执行结果如下：

```
c1                        c2                  c3
----------------------- -----------------  --------------
0x01230000000000000000   0xFFFF              0x14FFFF
```

(8) 日期时间类型

SQL Server 提供了专门的日期时间类型。日期和时间数据由有效的日期或时间组成。例如，“4/01/2005 12:15:00:00:00 PM”和“1:28:29:15:01 AM 8/17/2005”都是有效的日期和时间数据。

在 SQL Server 中，日期和时间数据使用 datetime 和 smalldatetime 数据类型存储：

- datetime　从 1753 年 1 月 1 日到 9999 年 12 月 31 日的日期和时间数据，精确度为 3‰秒(等于 3 毫秒或 0.003 秒)。表 5.5 为用户输入的值和 SQL Server 自动把值调整到.000、.003 或.007 秒的增量后的值。
- smalldatetime　从 1900 年 1 月 1 日到 2079 年 6 月 6 日的日期和时伺数据精确到分钟。29.998 秒或更低的 smalldatetime 值向下舍入为最接近的分钟，29.999 秒或更高的 smalldatetime 值向上舍入为最接近的分钟。

表 5.5 用户输入的值和 SQL Server 自动调整后的值

输入的值	SQL Server 自动调整后的值

火龙果•整理
uml.org.cn

01/01/05 23:59:59.999	2005-01-02 00:00:00.000
01/01/05 23:59:59.995, 01/01/05 23:59:59.996, 01/01/05 23:59:59.997, 01/01/05 23:59:59.998	2005-01-01 23:59:59.997
01/01/05 23:59:59.992, 01/01/05 23:59:59.993, 01/01/05 23:59:59.994	2005-01-01 23:59:59.993
01/01/05 23:59:59.990 01/01/05 23:59:59.991	2005-01-0123:59:59.990

下面是一个使用 smalldatetime 数据类型的例子：

```
SELECT CAST('2005-05-08 12:35:29.998' AS smalldatetime) AS '日期'
```

执行结果为：

```
日期
------------------------------
2005-05-08 12:35:00
```

由于最后是 29.998 秒，而 smalldatetime 数据类型精确到分钟，故执行结果将秒舍弃，最终结果为 12:35。

> 提示：CAST 函数的功能是将某种数据类型的表达式显式转换为另一种数据类型。AS 后面的数据类型就是要转换成为的数据类型。

SQL Server 可以识别的日期格式有字母格式、数字格式和无分隔字符串格式 3 种。字符格式允许使用以当前语言给出的月的全名(如 April)或月的缩写(如 Apr)来指定日期数据。字符格式的日期需要放在单引号内。可用的字符型日期格式如下：

```
Apr[il] [15][,] 2005
Apr[il] 15[,] [20]05
Apr[il] 2005 [15]
[15] Apr[il][,] 2005
15 Apr[il][,][20]05
15 [20]05 apr[il]
[15] 2005 Apr[il]
2005 APR[IL] [15]
2005 [15] APR[IL]
```

数字格式允许用指定的数字月份指定日期数据。例如，5/20/05 表示 2005 年 5 月的第 20 天，当使用数字日期格式时，在字符串中以斜杠(/)、连字符(-)或句号(.)作为分隔符来指

火龙果•整理
uml.org.cn

定月、日、年。例如，下面均表示 2005 年 4 月 15 日(--后面指明的是日期格式)：

```
[0]4/15/[20]05--(mdy)
[0]4-15-[20]05--(mdy)
[0]4.15.[20]05--(mdy)
[04]/[20]05/15--(myd)
15/[0]4/[20]05--(dmy)
15/[20]05/[0]4--(dym)
[20]05/15/[0]4--(ydm)
[20]05/[04]/15--(ymd)
```

> 提示：当语言被设置为 us_english 时，默认的日期顺序是 mdy。可以使用 SET DATEFORMAT 语句改变日期的顺序，根据所用的语言，它也会影响日期顺序。对 SET DATEFORMAT 的设置决定了如何解释日期数据．如果顺序和设置不匹配，则该值不会被解释为日期(因为它们超出了范围)，或者被错误地解释．例如，根据不同的 DATEFORMAT 设置，12/10/08 能被解释为 6 种日期的一种。

无分隔字符串格式是指数字间不需要分隔符号，可以使用 4、6、8 位数字来表达日期。如果使用 4 位表示，则只表示年份；如果使用 6 或 8 位，月和日必须用两位。例如“20051207”、“December 12, 2005”。

SQL Server 2000 可识别以下时间数据格式。用单引号(')把每一种格式括起来。下面都是有效的时间格式：

```
14:30
14:30[:20:999]
14:30[:20.9]
4am
4 PM
[0]4[:30:20:500]AM
```

可以用一个 AM 或 PM 后缀来表明时间值是在中午 12 点之前还是之后。AM 或 PM 的大小写可忽略。小时可以用 12 小时或 24 小时的时钟来指定。小时值解释如下：

- 小时值 0 表示午夜(AM)后的小时，不论是否指定 AM。当小时值等于 0 时不能指定 PM。
- 如果未指定 AM 和 PM，小时值 1 到 11 表示中午以前的小时。当指定 AM 时，也表示中午以前的小时。当指定 PM 时，则表示中午以后的小时。
- 如果未指定 AM 和 PM，小时值 12 表示始于中午的小时。如果指定为 AM，则表示始于午夜的小时。如果指定为 PM，则表示始于中午的小时。例如：12:01 是指中午过后 1 分钟，即 12:01 PM，而 12:01AM 是指午夜过后 1 分钟。指定为 12:01AM 与指定为 00:01 或 00:01 AM 相同。

火龙果•整理
uml.org.cn

- 如果指定 AM 或 PM，小时值 13 到 23 表示中午以后的小时。当指定 PM 时，也表示中午以后的小时。当小时值为 13 到 23 时，不能指定为 AM。
- 小时值 24 无效，用 12:00 AM 或 00:00 表示午夜。

可以在毫秒之前加上冒号(:)或者句号(.)。如果前面加冒号，这个数字表示千分之一秒。如果前面加句号，单个数字表示十分之一秒，两个数字表示百分之一秒，三个数字表示千分之一秒。例如，12:30:20:1 表示 12:30 过了 20 又千分之一秒；12:30:20.1 表示 12:30 过了 20 又十分之一秒。

以下是使用日期时间数据类型的例子：

```
CREATE TABLE Datetime_table
(
   c1 datetime,
   c2 smalldatetime
)
INSERT   Datetime_table   VALUES('2005-05-15   00:04:39.257','04/15/2005
14:30:20 PM')
SELECT * FROM Datetime_table
```

执行结果如下：

```
c1                          c2
--------------------------- -------------------------
2005-05-15 00:04:39.257     2005-04-15 14:30:00
```

(9) Unicode 字符型

在 SQL Server 2000 中，传统上非 Unicode 数据类型允许使用由特定字符集定义的字符。字符集是在安装 SQL Server 时选择的，不能更改。使用 Unicode(统一字符编码标准)数据类型，列(字段)可存储由 Unicode 标准定义的任何字符，包含由不同字符集定义的所有字符。Unicode 数据类型需要相当于非 Unicode 数据类型两倍的存储空间。

Unicode 数据使用 SQL Server 中的 nchar、varchar 和 ntext 数据类型进行存储。对于存储来源于多种字符集的字符的列，可采用这些数据类型。当列中各项所包含的 Unicode 字符数不同时(至多为 4000)，使用 nvarchar 类型。当列中各项为同一固定长度时(至多为 4000 个 Unicode 字符)，使用 nchar 类型。当列中任意项超过 4000 个 Unicode 字符时，使用 ntext 类型，其最大长度为 2^{30}-1。它们分别和字符型的 char[(n)]、varchar[(n)]和 text 类型相对应。

使用 Unicode 字符时，应该在前面加一个标识符 N，但是存储时并不存储该标识符。例如：

```
DECLARE @MyUnicodeVar NCHAR(25)
SET @MyUnicodeVar = N'This is a Unicode string.'
PRINT @MyUnicodeVar
```

执行结果为：

火龙果•整理
uml.org.cn


```
This is a Unicode string.
```

(10) 其他数据类型

在 SQL Server 2000 中，还提供了其他几种数据类型，包括下面几种：

- cursor　游标数据类型，用于创建游标变量或者定义存储过程的输出参数。它是惟一种不能赋值给表的列(字段)的基本数据类型。
- sql_variant　这是 SQL Server 2000 的新增数据类型，该数据类型可以存储除了 text、ntext、timestamp 和自己本身以外的其他所有类型的变量。
- table　该数据类型也是 SQL Server 2000 的新增数据类型，它可以暂时存储应用程序的结果，以便在以后用到。
- timestamp　时间戳数据类型，它可以反映数据库中数据修改的相对顺序。
- uniqueidentifier　全局惟一标识符(Globally Unique Identification Numbers, GUID)。它是一个 16 字节长的二进制数据类型，是 SQL Server 根据计算机网络适配器地址和主机 CPU 时钟产生的惟一号码而生成的全局惟一标识符代码。惟一标识符代码可以通过调用 NEWID 函数或者其他 SQL Server 应用程序编程接口来获得。

下面的示例使用 NEWID 对声明为 uniqueidentifier 数据类型的变量赋值，并将其打印出来：

```
DECLARE @MyID uniqueidentifier
SET @MyID = NEWID()
PRINT 'Value of @MyID is:'+ CONVERT(varchar(255),@MyID)
```

执行结果为：

```
Value of @MyID is:190C5BE0-234B-11D9-9947-00112F0997FA
```

> 注意：对于每台计算机，由 NEWID 返回的位不同。上面所显示的数字仅起解释的作用。另外，CONVERT 函数的作用是将一个数位转换为字符串。

2．用户定义数据类型

用户定义的数据类型总是根据基本数据类型进行定义的。它们提供了一种机制，可以将一个名称用于一个数据类型，这个名称更能清楚地说明该对象中保存的值的类型。这样程序员和数据库管理员就能更容易地理解以该数据类型定义的对象的意图。

用户定义的数据类型使表结构对程序员更有意义，并有助于确保包含相似数据类的列具有相同的基本数据类型。

> 提示：用户定义数据类型基于 SQL Server 2000 中的系统数据类型。当多个表的列中要存储同样类型的数据，且想确保这些列具有完全相同的数据类型、长度和为空性时，可使用用户定义数据类型．例如，可以基于 char 数据类型创建名为 postal_code 的用户定义数据类型。

创建用户定义的数据类型时必须提供以下 3 个参数：

火龙果•整理
uml.org.cn

- 名称
- 新数据类型所依据的系统数据类型
- 为空性(数据类型是否允许空值)　如果为空性未明确定义，系统将依据数据库或连接的 ANSI Null 默认设置进行指派。

如果用户定义数据类型是在 model 数据库中创建的，它将作用于所有用户定义的新数据库中。如果数据类型在用户定义的数据库中创建，则该数据类型只作用于此用户定义的数据库。

(1) 通过企业管理器来创建用户定义的数据类型

使用企业管理器创建用户定义的数据类型的操作步骤如下：

1) 展开服务器组，然后展开服务器。

2) 展开“数据库”文件夹，再展开要在其中创建用户定义的数据类型的数据库，例如 test。

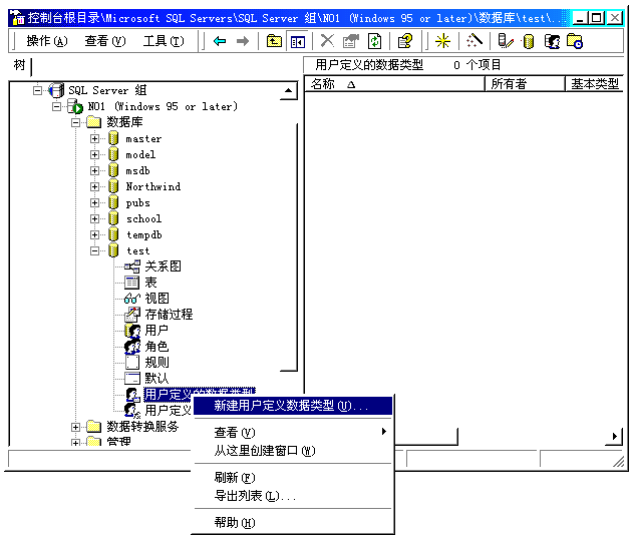
3) 右击“用户定义数据类型”选项，然后选择“新建用户定义数据类型”命令，如图 5.19 所示。

4) 此时，会打开“用户定义的数据类型属性”对话框，如图 5.19 所示。

在“名称”文本框中输入新建数据类型的名称；在“数据类型”下拉列表中，选择基数据类型。如“长度”处于活动状态，若要更改此数据类型可存储的最大数据长度，请键入另外的值。长度可变的数据类型有 binary、char、nchar、nvarchar、varbinary 和 varchar。若要允许此数据类型接受空值，请选中“允许 NULL 值”复选框。在“规则”和“默认值”下拉列表中选择一个规则或默认值(若有)，以将其绑定到用户定义数据类型上。

5) 设置完成后，单击“确定”按钮，即可创建一个用户定义数据类型。

若要删除用户定义数据类型，可在该用户定义数据类型上右击鼠标，然后选择“删除”命令，在打开的“除去对象”对话框中，单击“全部除去”按钮，即可删除用户定义数据类型。



火龙果•整理
uml.org.cn

图 5.19 企业管理器窗口

图 5.19 创建用户定义的数据类型

(2) 使用存储过程

可以使用存储过程 sp_addtype 来创建用户定义数据类型，其语法格式如下：

```
sp_addtype [@typename = ] type,
    [@phystype = ] system_data_type
    [,  [@nulltype =] 'null_type' ]
    [,  [@owner = ] 'owner_name' ]
```

各参数含义如下：

- [@typename =] type　用户定义的数据类型的名称。数据类型名称必须遵照标识符的规则，而且在每个数据库中必须是惟一的。“type”的数据类型为 sysname，没有默认值。
- [@phystype =] 'system_data_type'　是用户定义的数据类型所基于的物理数据类型或系统数据类型。
- [@nulltype =] 'null_type'　指明用户定义的数据类型处理空值的方式。null_type 的数据类型为 varchar(8)，默认值为 NULL，并且必须用单引号引起来('NULL', 'NOT NULL'或'NONULL')。如果没有用 sp_addtype 显式定义 nulltype，则将其设置为当前默认的(为空)。
- [@owner =] 'owner_name'　指定新数据类型的创建者或所有者。“owner_name”的数据类型为 sysname。当没有指定时，“owner_name”为当前用户。

该存储过程返回 0 表示成功，返回 1 表示失败。

提示：sysname 是一个系统支持的用户定义数据类型，等同于 nvarchar(128)，用来表示数据库对象名．另外，用户定义的数据类型名称在数据库中必须是惟一的，但是名称不同的用户定义的数据类型可以有相同的定义。

火龙果•整理
uml.org.cn

下面的示例为国内及国际电话和传真号码创建两个用户定义的数据类型 telephone 和 fax：

```
EXEC sp_addtype telephone, 'varchar(24)', 'NOT NULL'
EXEC sp_addtype fax, 'varchar(24)','NULL'
```

执行后，提示相应的信息如图 5.20 所示。表示已经成功执行，并生成两个用户定义数据类型。

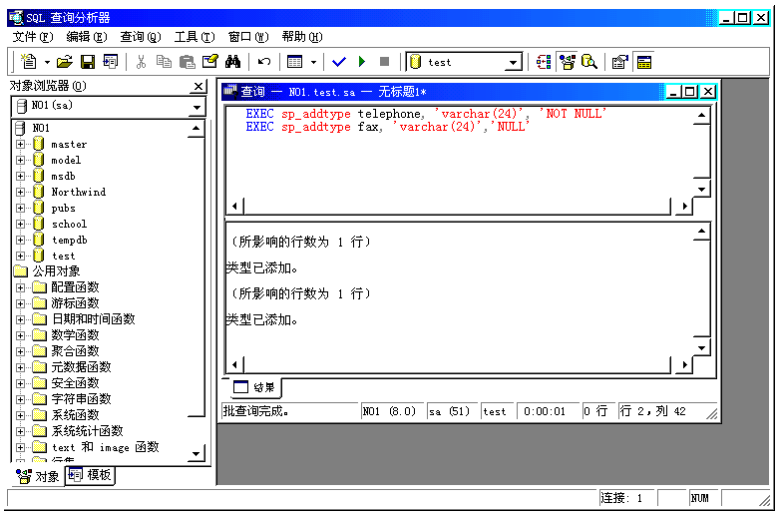


图 5.20 使用存储过程创建两个用户定义的数据类型

删除用户定义数据类型可以使用 sp_droptype 存储过程，其语法如下：

```
sp_droptype [@typename = ] 'type'
```

其中，“type”表示要删除的用户定义数据类型的名称。

5.3.3 运算符

运算符是一种符号，用来指定要在一个或多个表达式中执行的操作。SQL Server 提供的运算符有算术运算符、赋值运算符、按位运算符、比较运算符、逻辑运算符、字符串连接运算符和一元运算符。

1．算术运算符

算术运算符在两个表达式上执行数学运算，这两个表达式可以是数字数据类型分类的任何数据类型。在 SQL Server 中，算术运算符包括+(加)、-(减)、*(乘)、/(除)和%(取模)。

取模运算返回一个除法的整数余数。例如，16%3=1，这是因为 16 除以 3，余数为 1。

另外，加(+)和减(-)运算符也可用于对 datetime 及 smalldatetime 值执行算术运算，其使用格式如下：

日期±整数

火龙果•整理
uml.org.cn

该表达式返回“日期”加上或减去“整数”天数的日期。例如：

```
SELECT CAST('2005-1-1' AS smalldatetime) + 100 AS '2005.1.1加上100天的日期'
```

执行的结果如下：

```
2005.1.1 加上 100 天的日期
----------------------------------
2005-04-11 00:00:00
```

又例如：

```
SELECT CAST('2005-1-1' AS smalldatetime) - 100 AS '2005.1.14 减去 100 天的日期'
```

执行的结果如下：

```
2005.1.14 减去 100 天的日期
----------------------------------
2004-09-23 00:00:00
```

2．赋值运算符

Transact-SQL 有一个赋值运算符，即等号(=)。它将表达式的值赋予另外一个变量。例如，下面的 SQL 语句先声明一个变量，然后将一个取模运算的结果赋予该变量，最后是打印该变量的值：

```
DECLARE @MyCounter INT
SET @MyCounter=23%4
PRINT CONVERT(varchar(255), @MyCounter)
```

执行结果为：

```
3
```

也可以使用赋值运算符在列标题和为列定义值的表达式之间建立关系。例如，下面的 SQL 语句是将 school 数据库中的 student 表的 sno 均以“学生”显示：

```
USE school
SELECT sno='学生',sname,class FROM student
```

执行结果如下：

```
sno  sname    class
---- -------- -----
学生   李军      95033
学生   陆君      95031
学生   匡明      95031
```

火龙果•整理
uml.org.cn


```
学生    王丽      95033
学生    曾华      95033
学生    王芳      95031
```

3．按位运算符

按位运算符可以对两个表达式进行位操作，这两个表达式可以是整型数据或者二进制数据。按位运算符包括&(按位与)、|(按位或)和^(按位异或)。

Transact-SQL 首先把整数数据转换为二进制数据，然后再对二进制数据进行按位运算。例如，下面的 SQL 语句对两个变量进行按位运算：

```
DECLARE @a INT,@b INT
SET @a=3
SET @b=8
SELECT @a&@b AS 'a&b',@a|@b AS 'a|b',@a^@b AS 'a^b'
```

执行结果为：

```
a&b          a|b          a^b
----------- ----------- -----------
0            11           11
```

按位运算的两个操作数的数据类型有相应的规定，SQL Server 所支持的操作数数据类型如表 5.6 所示。

表 5.6 对按位运算的两个操作数的要求

左边操作数	右边操作数
Binary	int、smallint 或 tinyint
Bit	int、smallint、tinyint 或 bit
Int	int、smallint、tinyint、binary 或 varbinary
Smallint	int、smallint、tinyint、binary 或 varbinary
Tinyint	int、smallint、tinyint、binary 或 varbinary
Varbinary	int、smallint 或 tinvint

注意：按位运算符的两个操作数不能为 image 数据类型。

4．比较运算符

比较运算符用来比较两个表达式，表达式可以是字符、数字或日期数据，并可用在查询的 WHERE 或 HAVING 子句中。比较运算符的计算结果为布尔数据类型，它们根据测试条件的输出结果返回 TRUE 或 FALSE。

SQL Server 提供的比较运算符有下面几种：

- >(大于)

火龙果•整理
uml.org.cn

- <(小于)
- =(等于)
- <=(小于或等于)
- >=(大于或等于)
- !=(不等于)
- <>(不等于)
- !<(不小于)
- !>(不大于)

下面的 SQL 语句查询 score 表中成绩高于 88 分的成绩记录：

```
USE school
SELECT * FROM score WHERE degree>88
```

执行结果如下：

```
sno   cno       degree
----- ---------- -----------
103   3-105      92
107   3-105      91
```

5．逻辑运算符

逻辑运算符用来判断条件是为 TRUE 或者 FALSE，SQL Server 总共提供了 10 个逻辑运算符，如表 5.7 所示。

表 5.7 逻辑运算符

逻辑运算符	含　　义
ALL	当一组比较关系的值都为 TRUE 时，才返回 TRUE
AND	当要比较的两个布尔表达式的值都为 TRUE，才返回 TRUE
ANY	只要一组比较关系中有一个值为 TRUE，就返回 TRUE
BETWEEN	只有操作数在定义的范围内，才返回 TRUE
EXISTS	如果在子查询中存在，就返回 TRUE
IN	如果操作数在所给的列表表达式中，则返回 TRUE
LIKE	如果操作数与模式相匹配，则返回 TRUE
NOT	对所有其他的布尔运算取反
OR	只要比较的两个表达式有一个为 TRUE，就返回 TRUE
SOME	如果一组比较关系中有一些为 TRUE，则返回 TRUE

由于 LIKE 使用部分字符串来寻找记录，因此，在部分字符串中可以使用通配符。SQL Server 中可以使用的通配符及其含义如表 5.8 所示。

火龙果•整理
uml.org.cn

表 5.8 通配符及其含义

通配符	含　义	示　　例
%	包含零个或更多字符的任意字符串	WHERE sname LIKE '%华%'，将查找姓名中含有“华”所有学生
_(下划线)	任何单个字符	WHERE sname LIKE '王__'(有两个下划线)，将查找姓王的，名字包含 3 个字的学生
[]	指定范围　([a-f]) 或集合([abcdef])中的任何单个字符	WHERE sanme LIKE '[刘,王]__'，将查找姓刘的和姓王的，名字包含 3 个字的学生
[^]	不属于指定范围 ([a-f]) 或集合([abcdef])的任何单个字符	WHERE sname LIKE '[^刘,王]__'，将查找除姓刘的和姓王的，名字包含 3 个字的学生以外的其他学生

提示：在使用通配符时，对于汉字，一个汉字也算一个字符。另外，当使用 LIKE 进行字符串比较时，模式字符串中的所有字符都有意义，包括起始或尾随空格。如果查询中的比较要返回包含“abc”(abc 后有一个空格)的所有行，则将不会返回包含“abc”(abc 后没有空格)的所有行。因此，对于 datetime 数据类型的值，应当使用 LIKE 进行查询，因为 datetime 项可能包含各种日期部分。

例如，下面的 SQL 语句查询姓“王”的考试分数在 70～80 之间的学生学号、姓名、课程号和分数：

```
SELECT  student.sno,student.sname,score.cno,score.degree
FROM student,score
WHERE student.sno=score.sno AND student.sname LIKE '王%' AND score.degree
BETWEEN 70 AND 80
```

执行结果如下：

```
sno   sname       cno       degree
----- -------- ---------- --------------------
107   王丽        6-166     79
109   王芳        3-105     76
```

6．字符串连接运算符

字符串连接运算符为加号(+)。可以将两个或多个字符串合并或连接成一个字符串。还可以连接二进制字符串。

例如，下面的 SQL 语句将两个字符串连接在一起：

```
SELECT ('abc'+'def') AS '串连接'
```

执行结果如下：

```
串连接
```

火龙果•整理
uml.org.cn


```
------
abcdef
```

注意：其他数据类型，如 datetime 和 smalldatetime，在与字符串连接之前必须使用 CAST 转换函数将其转换成字符串。

7．一元运算符

一元运算符是指只有一个操作数的运算符。SQL Server 提供的一元操作符包含+(正)、-(负)和~(位反)。

正和负运算符表示数据的正和负，可以对所有的数据类型进行操作。位反运算符返回一个数的补数，只能对整数数据进行操作。

例如，下面的 SQL 语句，首先声明一个变量，并对变量赋值，然后对变量取负：

```
DECLARE @Num1 INT
SET @Num1 = 5
SELECT ~@Num1 AS '位反运算'
```

执行结果如下：

```
位反运算
-----------
-6
```

8．运算符优先级

当一个复杂的表达式有多个运算符时，运算符优先性决定执行运算的先后次序。执行的顺序可能严重地影响所得到的值。

在 SQL Server 中，运算符的优先级如下：

- +(正)、-(负)、~(按位 NOT)
- *(乘)、/(除)、%(模)
- +(加)、+(连接)、-(减)
- =、>、<、>=、<=、<>、!=、!>和!<比较运算符
- ^(位异或)、&(位与)、|(位或)
- NOT
- AND
- ALL、ANY、BETWEEN、IN、LIKE、OR、SOME
- =(赋值)

当一个表达式中的两个运算符有相同的运算符优先级时，基于它们在表达式中的位置来对其从左到右进行求值。例如：

```
DECLARE @MyNumber INT
SET @MyNumber = 6-1+2*2
```

火龙果•整理
uml.org.cn


```
SELECT @MyNumber AS '计算结果'
```

执行结果为：

```
计算结果
-----------
9
```

使用括号可以提高运算符的优先级，首先对括号中的内容进行求值，从而产生一个值，然后括号外的运算符才可以使用这个值。如果有嵌套的括号，则处于最里面的括号最先计算。例如：

```
DECLARE @MyNumber INT
SET @MyNumber = 3 * (5+(7-3))
SELECT @MyNumber AS '计算结果'
```

执行结果为：

```
计算结果
-----------
27
```

5.3.4 变量

在 SQL Server 中，变量分为局部变量和全局变量。全局变量名称前面有两个@字符，由系统定义和维护。局部变量前面有一个@字符，由用户定义和使用。

1．局部变最

局部变量由用户定义，仅在声明它的批处理(在下一小节中介绍)、存储过程或者触发器中有效。批处理结束后，局部变量将变成无效。

局部的定义可以使用 DECLARE 语句，其语法格式如下：

```
DECLARE  {@local_variable data_type } [,…n]
```

各参数含义如下：

- @local_variable　是变量的名称。变量名必须以 at 符(@)开头。局部变量名必须符合标识符规则。
- data_type　是任何由系统提供的或用户定义的数据类型 。变量不能是 text、ntext 或 image 数据类型。

在 SQL Server 中，一次可以定义多个变量。例如：

```
DECLARE @f float,@cn char(8)
```

如果要给变量赋值，可以使用 SET 和 SELECT 语句。其语法格式如下：

```
SET @local_variable = expression
```

火龙果•整理
uml.org.cn


```
SELECT {@local_variable = expression} [,…n]
```

其中，@local_variable 为定义的局部变量名称，expression 为一表达式。

例如，下面首先定义了两个变量，并分别使用 SET 和 SELECT 为其赋值，然后使用这两个变量查询 score 表中选修课程号为“3-105”且成绩高于 85 的记录。

```
DECLARE @f float,@cn char(8)
SET @f=85
SELECT @cn='3-105'
SELECT * FROM score WHERE cno=@cn AND degree>=@f
```

执行结果如下：

```
sno   cno          degree
----- ---------- -------------
103   3-105        92
105   3-105        88
107   3-105        91
```

提示：SELECT @local_variable 通常用于将单个值返回到变量中。例如，如果“expression”为列名，则返回多个值。如果 SELECT 语句返回多个值，则将返回的最后一个值赋予变量．如果 SELECT 语句没有返回行，变量将保留当前值。如果“expression”是不返回值的标量子查询，则将变量设为 NULL。一般来说，应该使用 SET，而不是 SELECT 给变量赋值。

2．全局变量

全局变量记录了 SQL Server 的各种状态信息，它们不能被显式地赋值或声明，而且不能由用户定义。在 SQL Server 2000 中，定义了全局变量，如表 5.9 所示。

表 5.9 SQL Server 2000 中的全局变量

变量名称	说　　明
@@CONNECTIONS	返回自 SQL Server 本次启动以来，所接受的连接或试图连接的次数
@@CPU_BUSY	返回自 SQL Server 本次启动以来，CPU 工作的时间，单位为毫秒
@@CURSOR_ROWS	返回游标打开后，游标中的行数
@@DATEFIRST	返回 SET DATAFIRST 参数的当前值
@@DBTS	返回当前数据库的当前 timestamp 数据类型的值
@@ERROR	返回上次执行的 SQL Transact 语句产生的错误编号
@@FETCH_STATUS	返回 FETCH 语句游标的状态
@@identity	返回最新插入的 identity 列值
@@IDLE	返回自 SQL Server 本次启动以来，CPU 空闲的时间，单位为毫秒
@@IO_BUSY	返回自 SQL Server 本次启动以来，CPU 处理输入和输出操作的时间，单位为毫秒

火龙果•整理
uml.org.cn

@@LANGID	返回本地当前使用的语言标识符
@@LANGUAGE	返回当前使用的语言名称
@@LOCK_TIMEOUT	返回当前的锁定超时设置，单位为毫秒
@@MAX_CONNECTIONS	返回 SQL Server 允许同时连接的最大用户数目
@@MAX PRECISION	返回当前服务器设置的 decimal 和 numeric 数据类型使用的精度
@@NESTLEVEL	返回当前存储过程的嵌套层数
@@OPTIONS	返回当前 SET 选项信息
@@PACK_RECEIVED	返回自 SQL Server 本次启动以来，通过网络读取的输入数据包数目
@@PACK_SENT	返回自 SQL Server 本次启动以来，通过网络发送的输出数据包数目
@@PACKET_ERRORS	返回自 SQL Server 本次启动以来，SQL Server 中出现的网络数据包的错误数目
@@PROCID	返回当前的存储过程标识符
@@REMSERVER	返回注册记录中显示的远程数据服务器的名称
@@ROWCOUNT	返回上一个语句所处理的行数
@@SERVERNAME	返回运行 SQL Server 的本地服务器名称
@@SERVICENAME	返回 SQL Server 运行时的注册键名称
@@SPID	返回服务器处理标识符
@@TEXTSIZE	返回当前 TESTSIZE 选项的设置值
@@TIMETICKS	返回一个计时单位的微秒数，操作系统的一个计时单位是 31.25 毫秒
@@TOTAL_ERRORS	返回自 SQL Server 本次启动以来，磁盘的读写错误次数
@@TOTAL_READ	返回自 SQL Server 本次启动以来，读磁盘的次数
@@TOTAL_WRITE	返回自 SQL Server 本次启动以来，写磁盘的次数
@@TRANCOUNT	返回当前连接的有效事务数
@@ VERSION	返回当前 SOL Server 服务器的日期，版本和处理器类型

5.3.5 批处理

批处理是包含一个或多个 Transact-SQL 语句的组，从应用程序一次性地发送到 SQL Server 执行。SQL Server 将批处理语句编译成一个可执行单元，此单元称为执行计划。执行计划中的语句每次执行一条。一个批处理语句以 GO 结束。

编译错误(如语法错误)使执行计划无法编译，从而导致批处理中的任何语句均无法执行。

运行时错误(如算术溢出或违反约束)会产生以下两种影响之一：

- 大多数运行时错误将停止执行批处理中当前语句和它之后的语句。
- 少数运行时错误(如违反约束)仅停止执行当前语句。而继续执行批处理中其他所有语句。

在遇到运行时错误之前执行的语句不受影响。惟一的例外是，如果批处理在事务中，而且错误导致事务回滚。在这种情况下，回滚运行时错误之前所进行的未提交的数据修改。

火龙果•整理
uml.org.cn

假定在批处理中有 10 条语句。如果第 5 条语句有一个语法错误，则不执行批处理中的任何语句。如果编译了批处理，而第 2 条语句在执行时失败，则第 1 条语句的结果不受影响，因为它已经执行。

在建立一个批处理的时候，应该注意下面的规则：

- CREATE DEFAULT、CREATE PROCEDURE、CREATE RULE、CREATE TRIGGER 和 CREATE VIEW 语句不能在批处理中与其他语句组合使用。批处理必须以 CREATE 语句开始。所有跟在该批处理后的其他语句将被解释为第一个 CREATE 语句定义的一部分。
- 不能在同一个批处理中更改表，然后引用新列。
- 如果 EXECUTE 语句是批处理中的第一句，则不需要 EXECUTE 关键字。如果 EXECUTE 语句不是批处理中的第一条语句，则需要 EXECUTE 关键字。

下面的 SQL 语句创建一个视图。因为 CREATE VIEW 必须是批处理中的惟一语句，所以需要 GO 命令将 CREATE VIEW 语句与其周围的 USE 和 SELECT 语句隔离：

```
USE school
GO
CREATE VIEW view95031 AS SELECT * FROM student WHERE class='95031'
GO
SELECT * FROM view95031
GO
```

执行结果如下：

```
sno   sname     ssex sbirthday                    class
----- -------- ---- ---------------------------- -----
103   陆君       男    1974-06-03 00:00:00.000     95031
105   匡明       男    1975-10-02 00:00:00.000     95031
109   王芳       女    1975-02-10 00:00:00.000     95031
```

5.3.6 注释

注释是指程序代码中不执行的文本字符串，也称为注解。使用注释对代码进行说明，可使程序代码更易于维护。注释通常用于记录程序名称、作者姓名和主要代码更改的日期。注释可用于描述复杂计算或解释编程方法。

SQL Server 支持两种类型的注释字符：

- --(双连字符)　这些注释字符可与要执行的代码处在同一行，也可另起一行。从双连字符开始到行尾均为注释。对于多行注释，必须在每个注释行的开始使用双连字符。
- /*…*/(正斜杠-星号对)　这些注释字符可与要执行的代码处在同一行，也可另起一行，甚至在可执行代码内。从开始注释对(/*)到结束注释对(*/)之间的全部内容均

火龙果•整理
uml.org.cn

视为注释部分。对于多行注释，必须使用开始注释字符对(/*)开始注释，使用结束注释字符对(*/)结束注释。注释行上不应出现其他注释字符。

下面就是使用注释的例子：

```
USE school  /*打开 school 数据库*/
GO
--执行一个 SELECT 语句
SELECT tname,depart FROM teacher   --显示所有的教师的姓名和单位记录
GO
```

执行结果如下：

```
tname    depart
-------- ----------
李诚       计算机系
王萍       计算机系
刘冰       电子工程系
张旭       电子工程系
```

注意：多行“/*…*/”注释不能跨越批处理。整个注释必须包含在一个批处理内。

5.3.7 控制流语句

Transact-SQL 提供称为控制流的特殊关键字，用于控制 Transact-SQL 语句、语句块和存储过程的执行流。这些关键字可用于 Transact-SQL 语句、批处理和存储过程中。

控制流语句就是用来控制程序执行流程的语句，使用控制流语句可以在程序中组织语句的执行流程，提高编程语言的处理能力。SQL Server 提供的控制流语句如表 5.10 所示。

表 5.10 控制流语句

控制流语句	说明
BEGIN…END	定义语句块
GOTO	无条件跳转语句
CASE	分支语句
IF … ELSE	条件处理语句，如果条件成立，执行 IF 语句；否则执行 ELSE 语句
RETURN	无条件退出语句
WAITFOR	延迟语句
WHILE	循环语句
BREAK	跳出循环语句
CONTINUE	重新开始循环语句

火龙果•整理
uml.org.cn

1．BEGIN…END 语句

BEGIN…END 语句用于将多个 Transact-SQL 语句组合为一个逻辑块。在执行时，该逻辑块作为一个整体被执行。

语法格式为：

```
BEGIN
{
    sql 语句|语句块
}
END
```

其中，“sql 语句|语句块”是任何有效的 Transact-SQL 语句或以语句块定义的语句分组。

任何时候当控制流语句必须执行一个包含两条或两条以上 Transact-SQL 语句的语句块时，都可以使用 BEGIN 和 END 语句。它们必须成对使用，任何一条语句均不能单独使用。

BEGIN 语句行后为 Transact-SQL 语句块。最后，END 语句行指示语句块结束。

BEGIN … END 语句可以嵌套使用。例如：

```
BEGIN
  DECLARE @MyVar float
  SET @MyVar = 456.256
  BEGIN
    PRINT '变量@MyVar 的值为：'
    PRINT CAST(@MyVar AS varchar(12))
  END
END
```

执行结果为：

```
变量@MyVar 的值为：
456.256
```

下面几种情况经常要用到 BEGIN 和 END 语句：

- WHILE 循环需要包含语句块。
- CASE 函数的元素需要包含语句块。
- IF 或 ELSE 子句需要包含语句块。

注意：上述情况下，如果只有一条语句，则不需要使用 BEGIN…END 语句。

2．IF…ELSE 语句

使用 IF…ELSE 语句，可以有条件地执行语句。其语法格式如下：

```
IF Boolean_expression
```

火龙果•整理
uml.org.cn


```
  {sql 语句|语句块}
[ELSE
  {sql 语句|语句块}]
```

各参数含义如下：

- Boolean expression　布尔表达式，可以返回 TRUE 或 FALSE。如果布尔表达式中含有 SELECT 语句，必须用圆括号将 SELECT 语句括起来。
- {sql 语句|语句块}　Transact-SQL 语句或用语句块定义的语句分组。除非使用语句块，否则 IF 或 ELSE 条件只能影响一个 Transact-SQL 语句的性能。若要定义语句块，可以使用控制流关键字 BEGIN…END。

IF…ELSE 语句的执行方式是：如果布尔表达式的值为 TRUE，则执行 IF 后面的语句块；否则执行 ELSE 后面的语句块。例如：

```
USE school
IF (SELECT AVG(degree) FROM score WHERE cno='3-105')>80
BEGIN
  PRINT '课程:3-105'
  PRINT '考试成绩还不错'
END
ELSE
BEGIN
  PRINT '课程:3-105'
  PRINT '考试成绩一般'
END
```

执行结果为：

```
课程:3-105
考试成绩还不错
```

在 IF…ELSE 语句中，IF 和 ELSE 后面的子句都允许嵌套，嵌套层数不受限制。

> 注意：如果在 IF…ELSE 语句的 IF 部分和 ELSE 部分都使用了 CREATE TABL 语句或 SELECT INTO 语句，那么 CREATE TABLE 语句或 SELECT INTO 语句必须指向相同的表名。

3．CASE 语句

使用 CASE 语句可以进行多个分支的选择。CASE 具有两种格式：

- 简单 CASE 格式　将某个表达式与一组简单表达式进行比较以确定结果。
- 搜索 CASE 格式　计算一组布尔表达式以确定结果。

火龙果•整理
uml.org.cn

(1) 简单 CASE 格式

其语法格式如下：

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

其中各参数的含义如下：

- input_expression　使用简单 CASE 格式时所计算的表达式，可以是任何有效的表达式。
- when_expression　用来和“input_expression”做比较的表达式。“input_expression”和每个“when_expression”的数据类型必须相同，或者是隐性转换。
- result_expression　当 input_expression=when_expression 的取值为 TRUE 时，需要返回的表达式。
- else_result_expression　当 input_expression=when_expression 的取值为 FALSE 时，需要返回的表达式。

简单 CASE 格式的执行方式为：当 input_expression=when_expression 的取值为 TRUE，则返回 result_expression；否则返回 else_result_expression。如果没有 ELSE 子句，则返回 NULL。例如：

```
USE school
GO
SELECT tname AS '姓名',depart AS '单位',
   CASE prof
      WHEN '教授' THEN '高级职称'
      WHEN '副教授' THEN '高级职称'
      WHEN '讲师' THEN '中级职称'
      WHEN '助教' THEN '初级职称'
   END AS '职称类型'
FROM teacher
GO
```

执行结果如下：

```
姓名       单位          职称类型
-------- ---------- --------
李诚       计算机系      高级职称
王萍       计算机系      初级职称
刘冰       电子工程系    初级职称
```

火龙果•整理
uml.org.cn


```
张旭        电子工程系     中级职称
```

(2) 搜索 CASE 格式

其语法格式如下：

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

其中各参数的含义与简单 CASE 格式的参数含义类似。

搜索 CASE 格式的执行方式为：当“Boolean_expression”表达式的值为 TRUE，则返回 THEN 后面的表达式“result_expression”，然后跳出 CASE 语句；否则继续测试下一个 WHEN 后面的布尔表达式。如果所有的 WHEN 后面的布尔表达式均为 FALSE，则返回 ELSE 后面的表达式。如果没有 ELSE 子句，则返回 NULL。例如：

```
USE school
GO
SELECT sno AS '学号',cno AS '课程号',
   CASE
       WHEN degree>=90 THEN 'A'
       WHEN degree>=80 THEN 'B'
       WHEN degree>=70 THEN 'C'
       WHEN degree>=60 THEN 'D'
       WHEN degree<60 THEN  'E'
   END AS '成绩'
FROM score ORDER BY sno
GO
```

执行结果如下：

```
学号    课程号        成绩
----- ---------- ----
101   3-105     D
101   6-166     B
103   3-245     B
103   3-105     A
105   3-245     C
105   3-105     B
107   3-105     A
107   6-166     C
```

火龙果•整理
uml.org.cn


```
108   6-166     B
108   3-105     C
109   3-105     C
109   3-245     D
```

4．WHILE 语句

WHILE 语句可以设置重复执行 SQL 语句或语句块的条件。只要指定的条件为真，就重复执行语句。可以使用 BREAK 和 CONTINUE 关键字在循环内部控制 WHILE 循环中语句的执行。

其语法格式如下：

```
WHILE Boolean_ression
{sql 语句|语句块}
[BREAK]
{sql 语句|语句块}
[CONTINUE]
```

各参数含义如下：

- Boolean expression　布尔表达式，可以返回 TRUE 或 FALSE。如果布尔表达式中含有 SELECT 语句，必须用圆括号将 SELECT 语句括起来。
- {sql 语句|语句块}　Transact-SQL 语句或用语句块定义的语句分组。若要定义语句块，可以使用控制流关键字 BEGIN…END。
- BREAK　导致从最内层的 WHILE 循环中退出。将执行出现在 END 关键字后面的任何语句，END 关键字为循环结束标记。
- CONTINUE　使 WHILE 循环重新开始执行，忽略 CONTINUE 关键字后的任何语句

WHILE 语句的执行方式是：如果布尔表达式的值为 TRUE，则反复执行 WHILE 语句后面的语句块；否则将跳过后面的语句块。例如，下面的 SQL 语句是计算从 1 加到 100 的值：

```
DECLARE @s int,@i int
SET @i = 0
SET @s = 0
WHILE @i<=100
  BEGIN
    SET @s = @s+@i
    SET @i = @i+1
  END
PRINT '1+2+...+100='+CAST(@s AS char(25))
```

火龙果•整理
uml.org.cn

执行结果为：

```
1+2+...+100=5050
```

5．GOTO 语句

GOTO 语句可以实现无条件的跳转。其语法格式为：

```
GOTO lable
```

其中，“lable”为要跳转到的语句标号。其名称要符合标识符的规定。

GOTO 语句的执行方式为：遇到 GOTO 语句后，直接跳转到 lable 标号处继续执行，而 GOTO 后面的语句将不被执行。例如，使用下面的语句重新计算从 1 加到 100 的值：

```
DECLARE @s int,@i int
SET @i = 0
SET @s = 0
my_loop:  --定义标号
SET @s = @s+@i
SET @i = @i+1
IF @i<=100 GOTO my_loop  --如果小于 100,跳转到 my_loop 标号处
PRINT '1+2+...+100='+CAST(@s AS char(25))
```

执行结果与上面相同。

又例如，以下语句输出 108 学号学生的平均成绩，若没有该学生成绩时显示相应的提示信息：

```
DECLARE @avg float
USE school
IF (SELECT COUNT(*) FROM score WHERE sno='108')=0
    GOTO label1
BEGIN
   PRINT '108 学号学生的平均成绩:'
   SELECT @avg=AVG(degree) FROM score WHERE sno='108'
   PRINT @avg
   RETURN
END
label1:
   PRINT '108 学号的学生无成绩'
```

执行结果如下：

```
108 学号学生的平均成绩:
79.5
```

火龙果•整理
uml.org.cn

6．RETURN 语句

使用 RETURN 语句，可以从查询或过程中无条件退出。可在任何时候用于从过程、批处理或语句块中退出，而不执行位于 RETURN 之后的语句。

语法格式为：

```
RETURN  [integer_expression]
```

其中，“integer_expression”为一个整数数值，是 RETURN 语句要返回的值。

注意：当用于存储过程时，RETURN 不能返回空值。如果试图返回空值，将生成警告信息并返回 0 值。

例如，首先执行下面的 SQL 语句创建一个存储过程：

```
USE school
GO
CREATE PROC MyPro @no char(5)   --创建存储过程 MyPro,fs 为形参
AS RETURN(SELECT AVG(degree) FROM score WHERE sno=@no)
```

然后执行下面的 SQL 语句：

```
DECLARE @no char(5),@avg float
SET @no='108'
EXEC @avg=MyPro @no  --以 no 实参调用存储过程 MyPro
SELECT sname,@avg AS '平均分' FROM student WHERE sno=@no
GO
```

执行结果如下：

```
sname    平均分
-------- ---------------
曾华      79.0
```

提示：关于存储过程的详细信息，可参考第 9 章。

7．WAITFOR 语句

使用 WAITFOR 语句，可以在指定的时间或者过了一定时间后，执行语句块、存储过程或者事务。

其语法格式为：

```
WAITFOR {DELAY 'time' | TIME 'time'}
```

其中，各参数的说明如下：

- DELAY　指示 SQL Server 一直等到指定的时间过去，最长可达 24 小时。
- 'time'　要等待的时间。可以按 datetime 数据可接受的格式指定 time，也可以用局

火龙果•整理
uml.org.cn

部变量指定此参数。不能指定日期，因此在 datetime 值中不允许有日期部分。

- TIME　指示 SQL Server 等待到指定时间。

例如，下面的 SQL 语句指定在 1 时 10 分时执行一个语句：

```
BEGIN
   WAITFOR TIME '1:10:00'
   PRINT '现在是 1:10:00'
END
```

执行后，等计算机上的时间到了 1 时 10 分时，出现下面结果：

```
现在是 1:10:00
```

5.3.8 函数

编程语言中的函数是用于封装经常执行的逻辑的子例程。任何代码若必须执行函数所包含的逻辑，都可以调用该函数，而不必重复所有的函数逻辑。SQL Server 2000 支持两种函数类型：内置函数和用户定义函数。

1．内置函数

SQL Server 2000 提供了丰富的具有执行某些运算功能的内置函数，可分为 12 类，如表 5.11 所示。

表 5.11　SQL Server 2000 提供的内置函数

函数分类	说明
聚合函数	执行的操作是将多个值合并为一个值，例如 COUNT、SUM、MIN 和 MAX
配置函数	是一种标量函数，可返回有关配置设置的信息
游标函数	返回有关游标状态的信息
日期和时间函数	操作 datetime 和 smalldatetime 值
数学函数	执行三角、几何和其他数字运算
元数据函数	返回数据库和数据库对象的特性信息
行集函数	返回行集，这些行集可用在 Transact-SQL 语句中表引用所在的位置
安全性函数	返回有关用户和角色的信息
字符串函数	操作 char、varchar、nchar、nvarchar、binary 和 varbinary 值
系统函数	对系统级别的各种选项和对象进行操作或报告
系统统计函数	返回有关 SQL Server 性能的信息
文本和图像函数	操作 text 和 image 值

使用函数时总是带有圆括号，即使没有参数也是如此。例外情况是与 DEFAULT 关键字一起使用的 niladic 函数(不带参数的函数)。而且函数可以嵌套(一个函数用于另一个函数内部)。例如，下面是一个使用内置函数的例子：

```
PRINT '现在日期和时间是：' + CAST(GETDATE() AS char(50))
```

火龙果•整理
uml.org.cn

执行结果如下：

```
现在日期和时间是：10 24 2004  7:15PM
```

有时，用来指定数据库、计算机、登录或数据库用户的参数是可选的。如果未指定这些参数，就默认地将这些参数赋值为当前的数据库、主机、登录或数据库用户。

函数可以是确定性的，也可以是非确定性的。如果任何时候使用特定的输入值集调用函数时总是返回相同的结果，则称该函数为确定性函数。如果每次使用特定的输入值集调用函数时返回不同的结果，则该函数是非确定性函数。

2．用户定义函数

函数是由一个或多个 Transact-SOL 语句组成的子程序，可用于封装代码以便重新使用。SQL Server 2000 并不将用户限制在定义为 Transact-SQL 语言一部分的内置函数上，而是允许用户创建自己的用户定义函数。

可使用 CREATE FUNCTION 语句创建，使用 ALTER FUNCTION 语句修改，以及使用 DROP FUNCTION 语句除去用户定义函数．每个完全合法的用户定义函数名必须惟一。

SQL Server 2000 支持 3 种用户定义函数：

- 标量函数
- 内嵌表值函数
- 多语句表值函数

> 注意：在用户定义函数中，BEGIN…END 块中的语句不能有任何副作用。函数副作用是指对具有函数外作用城 (例如数据库表的修改)的资源状态的任何永久性更改。函数中的语句惟一能做的更改是对函数上的局部对象(如局部游标成局部变量)的更改。不能在函数中执行的操作包括：对数据库表的修改、对不在函数上的局部游标进行操作、发送电子邮件、尝试修改目录，以及生成返回至用户的结果集。

用户定义函数接受零个或更多的输入参数，并返回单值。函数最多可以有 1024 个输入参数。

例如：下面的 SQL 语句在 test 数据库中定义了一个 CubicVolume 用户定义函数，然后使用该函数计算一个长方体的体积：

```
USE test
GO
CREATE FUNCTION CubicVolume
--输入参数
(@CubeLength decimal(4,1), @CubeWidth decimal(4,1),
       @CubeHeight decimal(4,1))
RETURNS decimal(12,3) --返回立方体的体积
AS
```

火龙果•整理
uml.org.cn


```
  BEGIN
     RETURN(@CubeLength*@CubeWidth*@CubeHeight)
END
GO
PRINT '长、宽、高分别为 6、4、3 的立方体的体积'+
       CAST (dbo.CubicVolume(6,4,3) AS char(10))
GO
```

执行结果为：

```
长、宽、高分别为 6、4、3 的立方体的体积 72.000
```

SQL Server 2000 还支持返回 table 数据类型的用户定义函数：

- 该函数可声明内部 table 变量，将行插入该变量，然后将该变量作为返回值返回。
- 一类称为内嵌函数的用户定义函数，将 SELECT 语句的结果集作为变量类型 table 返回。内嵌用户定义函数遵从以下规则：(1)RETURNS 子句仅包含关键字 table。不必定义返回变量的格式，因为它由 RETURN 子句中的 SELECT 语句的结果集的格式设置。(2)函数体不由 BEGIN 和 END 分隔。(3)RETURN 子句在括号中包含单个 SELECT 语句。SELECT 语句的结果集构成函数所返回的表。内嵌函数中使用的 SELECT 语句受到与视图中使用的 SELECT 语句相同的限制。

这些函数可用在能指定表达式的地方。返回 table 的用户定义函数可以是替代视图的强大方式。返回 table 的用户定义函数可用在 Transact-SQL 查询中允许表或视图表达式的地方。视图受限于单个 SELECT 语句，而用户定义函数可包含附加的语句，使函数所包含的逻辑比视图可能具有的逻辑更强大。

返回 table 的用户定义函数还可替换返回单个结果集的存储过程。由用户定义函数返回的 table 可在 Transact-SQL 语句的 FROM 子句中引用，而返回结果集的存储过程则不能。例如：

```
USE school
GO
CREATE FUNCTION student1(@classno char(5))  --建立函数 student1
  RETURNS @st TABLE     --返回表@st,下面定义其表结构
  (
     sno char(5),
     sname char(8),
     cno char(5),
     degree int
  )
AS
BEGIN
```

火龙果•整理
uml.org.cn


```
  INSERT @st     --向@st 中插入满足条件的记录
      SELECT student.sno,student.sname,score.cno,score.degree
      FROM student,score
      WHERE student.sno=score.sno AND student.class=@classno
  RETURN
END
GO
```

在这个函数中，返回的本地变量名是@st。函数中的语句在@st 变量中插入行，以生成由该函数返回的 table 结果。外部语句唤醒调用该函数以引用由该函数返回的 table。下面的 SQL 语句即使用该函数查询 95031 班所有学生的考试成绩记录：

```
SELECT * FROM student1(95031)
```

执行结果为：

```
sno   sname     cno    degree
----- -------- ----- -----------
103   陆君      3-245  86
105   匡明      3-245  75
109   王芳      3-245  68
103   陆君      3-105  92
105   匡明      3-105  88
109   王芳      3-105  76
```

练习题 5

1．SQL 语句分为哪 4 类？

2．NULL 代表什么含义？将其与其他值进行比较会产生什么结果？如果数值型列中存在 NULL，会产生什么结果？

3．使用 identity 列时应注意什么？

4．使用 Transact-SQL 语句向表中插入数据应注意什么？

5．LIKE 匹配字符有哪几种？如果要检索的字符中包含匹配字符，那么该如何处理？

6．要使查询的结果有序显示，应使用什么语句？

7．什么是局部变量？什么是全局变量？如何标识它们？

8．什么是批处理？使用批处理有何限制？

9．在默认情况下，SQL 脚本文件的后缀是什么？SQL 脚本执行的结果有哪几种形式？

火龙果•整理
uml.org.cn

上机实验题 4

在前一个实验中建立的 factory 数据库上，完成如下各题：

(1) 显示所有职工的年龄。

(2) 求出各部门的党员人数。

(3) 显示所有职工的姓名和 2004 年 1 月份工资数。

(4) 显示所有职工的职工号、姓名和平均工资。

(5) 显示所有职工的职工号、姓名、部门名和 2004 年 2 月份工资，并按部门名顺序排列。

(6) 显示各部门名和该部门的所有职工平均工资。

(7) 显示所有平均工资高于 1200 的部门名和对应的平均工资。

(8) 显示所有职工的职工号、姓名和部门类型，其中财务处和人事处属管理部门，市场部属市场部门。

(9) 若存在职工号为 10 的职工，则显示其工作部门名称，否则显示相应提示信息。

(10) 求出男女职工的平均工资，若男职工平均工资高出女职工平均工资 50%，则显示“男职工比女职工的工资高多了 ”的信息，若男职工平均工资与女职工平均工资比率在 1.5～0.8 之间，则显示“男职工跟女职工的工资差不多”的信息，否则，显示“女职工比男职工的工资高多了”的信息。