第3章

结构查询语言(SQL)

结构化查询语言 SQL(Structured Query Language)是一个通用的、功能极强的关系数据库语言,也是关系数据库的标准语言。目前,几乎所有的关系数据库管理系统软件都支持 SQL 语言,而且许多数据库软件厂商都对 SQL 语言中的语句命令进行了不同程度的扩充和修改。掌握 SQL 语言的使用是学习数据库技术最基本的要求。

本书在介绍 SQL 语言的特性时,将一并介绍 SQL Server 2014 中 SQL 语言的相关特性。SQL Server 2014 中的 SQL 语言称为 Transact-SQL 语言,简称 T-SQL 语言。T-SQL 语言是 Sybase 公司和 Microsoft 公司联合开发,后来被 Microsoft 公司移植到 SQL Server 的一种 SQL 语言,它不仅包含了 SQL92 标准的大多数功能,而且还对 SQL 进行了一系列的扩展,增加了许多新特性,增强了可编程性和灵活性。本书不是 T-SQL 语言的用户手册,关于 T-SQL 语言的更多细节请参考 SQL Server 2014 提供的联机丛书。

本章介绍 SQL 语言中的数据定义、数据查询、数据更新以及完整性约束的实现等内容。关于 SQL 语言的其他方面(如安全性管理、事务管理等)将在后续章节中陆续介绍。

3.1 SQL 概述

3.1.1 SQL 的产生与发展

结构化查询语言最早是 1974 年由 Boyce 等人提出的,1975 年在 IBM 公司的 RDBMS 原型 System R 中实现。由于它功能丰富、使用方便、简洁易学而备受用户和计算机工业界欢迎,众多的关系数据库系统产品都实现了 SQL 语言。经过不断的修改、扩充和完善,SQL 语言最终成为关系数据库的标准语言。

1986 年 10 月美国国家标准局(American National Standard Institute,ANSI)的数据库委员会批准 SQL 作为关系数据库语言的美国标准,同年公布了 SQL 标准文本。1987 年 6 月国际标准化组织(International Organization for Standardization,ISO)把该标准文本采纳为国际标准,现在这两个标准称为 SQL86。ISO 在 1989 年 4 月公布了增强完整性特征的 SQL89 标准,后来对 SQL89 标准进行了修改和扩充后,在 1992 年又公布了 SQL92 标准(也称 SQL2)。

SQL92 是一次非常重要的 SQL 语言标准的升级,其文档达到了 622 页,与 SQL89 的 120 页文档相比,标准的内容增加了许多。SQL92 标准有 4 个层次,即入门级、过渡级、中间级、完备级。入门级只对前一个标准 SQL89 稍作修改;过渡级则指定了内连接语法和外连接语法;中间级增加了许多新特性,如 DATE 和 TIME 数据类型、DOMAIN、CASE 表达式、数据类型之间的 CAST 函数、级联 DELETE 以保证参照完整性等;完备级增加了 BIT 数据类型、FROM 子句中的导出表、CHECK 子句中的子查询等新特性。

在 SQL92 的基础上，经过大量的研究工作，1999 年 ISO 正式公布了包括面向对象和许多新的数据库概念的 SQL 语言标准 SQL99（也称 SQL3）。目前，SQL 的标准化工作还在继续，已经发布的标准有 SQL2003、SQL2008、SQL2011 和 SQL2016。

自 SQL 语言成为国际标准后，各数据库厂家均采用 SQL 作为自己 RDBMS 的数据语言和标准接口，这使得不同数据库系统之间的互操作有了共同的基础。这个意义十分重大，因此，有人将确立 SQL 为关系数据库语言标准及其后的发展称为是一场革命。

最后必须指出的是，目前的 RDBMS 产品基本上都支持 SQL92 的入门级，如果使用了 SQL92 过渡级、中间级或完备级里的特性，或者使用了 SQL 新标准中的特性，就可能存在无法“移植”应用的风险，因为各数据库厂家在自己 RDBMS 中实现的 SQL 语言与标准 SQL 语言有一些差别，有的支持标准以外的 SQL 特性，有的不支持标准中的某些特性。关于各 RDBMS 软件的更多信息可在各产品的 SQL 用户手册中找到。

3.1.2　SQL 的功能与特点

SQL 语言具备了对数据库进行所有操作的功能，主要分成三个部分：①数据定义语言（Data Definition Language，DDL）主要定义数据库的逻辑结构，包括定义基本表、索引和视图三个部分；②数据操纵语言（Data Manipulation Language，DML）包括数据查询和数据更新两大类操作，其中数据更新又包括插入、删除和修改三种操作；③数据控制语言（Data Control Language，DCL）主要有对基本表和视图的授权，事务控制语句等。SQL 语言主要有以下五个特点。

1）综合统一

数据库系统的主要功能是通过系统所支持的语言来实现的。非关系模型的数据语言一般分 DDL、DML 和 DCL，分别用于定义模式（包括完整性约束条件）、插入数据建立数据库、更新数据、查询数据、数据库重构、数据库安全性控制等一系列操作要求。在 SQL 语言之前，这些语言互相独立，使用很不方便。而 SQL 语言集 DDL、DML 和 DCL 功能于一体，可以独立完成上面所提到的一系列操作要求，数据结构的单一而带来的操作符统一（不像 DBTG 中用 STORE 插入数据，用 CONNECT 插入联系）使得其使用更加方便。另外，用户在数据库系统投入运行后，可根据需要随时修改模式，并不影响数据库的运行，从而使系统具有良好的可扩展性。

2）面向集合的操作方式

非关系模型采用的是面向记录的操作方式，操作对象是一条记录。例如，查询所有选修了“数据库原理”课程的女生姓名，用户必须一条一条把满足条件的学生记录找出来（通常还要说明具体的处理过程）。而关系模型的基本数据结构是关系，即元组的集合，SQL 语言操作的对象和操作的结果都是关系，因此一次插入、修改、删除、查询操作的对象都可以是元组的集合。

3）高度非过程化

由于非关系模型通常是用存储路径来表达实体之间的联系，因此非关系系统中的数据语言是面向过程的语言。用它来完成某项查询要求时，用户必须指出存取路径及详细的处理过程。而 SQL 语言是非过程化的语言，在完成某项查询要求时，用户无须了解存取路径，只要提出“做什么”，不必指出“怎么做”。存储路径的选择及 SQL 语言的处理过程由 DBMS

自动完成。这不但大大减轻了用户的负担，而且也有利于数据独立性的提高。

4）以同一种语法结构提供两种使用方式

SQL 既是自含式语言，又是嵌入式语言。作为自含式语言，它能独立地用于联机交互的使用方式，用户在键盘上输入一条 SQL 语句，系统执行一条，非常有利于用户学习 SQL 语言。作为嵌入式语言，SQL 语句能嵌入到高级语言（如 C 语言或 Java 语言）中，便于用户开发应用程序。而在这两种不同的使用方式下，SQL 语句的语法是基本上一致的。

5）支持三级模式结构

SQL 语言完全支持关系数据库的三级模式结构，如图 3.1 所示。模式对应于基本表（Base Table），外模式对应于视图（View）和部分基本表，而内模式对应于存储文件。由于在关系数据库中存储文件的物理结构对用户是透明的，存储方法的选择及 SQL 语言的处理过程由 DBMS 自动完成，因此用户一般不必定义关系数据库的内模式，但创建索引可以认为是定义内模式。有关索引和视图的内容在第 4 章中介绍。

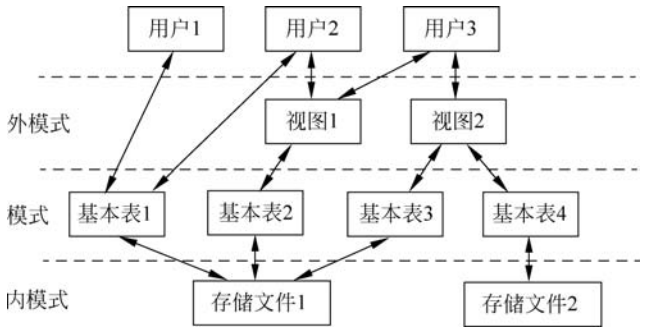

图 3.1　SQL 对关系数据库三级模式的支持

3.2　数 据 定 义

SQL 语言完全支持关系数据库的三级模式结构，其模式、外模式和内模式中的基本对象有数据库、表、视图和索引。因此，SQL 的数据定义功能包括定义数据库、定义表、定义索引和定义视图。本节只介绍数据库和表的定义，索引和视图的定义在第 4 章中介绍。

需要说明的是，在 SQL Server 中许多操作都可以通过多种方法来实现，本书主要介绍如何通过 T-SQL 语言来实现，其他实现方法请参考相关文献。

3.2.1　数据库的创建、修改与删除

在定义基本表之前，首先要创建数据库，但 SQL 语言没有创建数据库的语句。在介绍 T-SQL 中的 CREATE DATABASE 语句之前，先介绍 SQL Server 数据库的结构。

1. SQL Server 数据库结构

在 SQL Server 中，数据库分为两个层次，即物理数据库和逻辑数据库。物理数据库是面向操作系统的，由数据文件、日志文件、文件组、盘区与页等组成；逻辑数据库是面向用户的，由表、约束、默认值、规则、用户自定义数据类型、索引、视图、用户、角色、存储过程、触发器等一系列数据库对象组成。

1) 数据文件

数据文件是存放数据库数据和数据库对象的文件。一个数据库可以有一个或多个数据文件,一个数据文件只属于一个数据库。当有多个数据文件时,其中一个被定义为主数据文件,其余的被称为辅助数据文件。一个数据库必须有且只能有一个主数据文件,而辅助数据文件可以有多个,也可以没有。主数据文件用于存储数据库的启动信息以及部分或全部数据和数据库对象,其扩展名为.mdf。辅助数据文件用于存储主数据文件中未存储的剩余数据和数据库对象,其扩展名为.ndf。

2) 日志文件

日志文件用于存储用户对数据库进行任何操作(如插入、删除或修改等)的日志信息,以便于数据库的恢复,也有利于保护数据库的安全。每个数据库必须至少有一个日志文件,也可以有多个日志文件。日志文件的扩展名为.ldf。

3) 文件组

文件组是数据库中数据文件的逻辑组合,利用文件组可以将数据分布在不同磁盘上,当查询数据时系统会创建多个线程来并行读取分布在不同磁盘上的文件,从而加快查询速度。一个数据文件只能属于一个文件组,一个文件组也只能属于一个数据库。事务日志文件是独立存在的,不属于任何文件组。另外,文件组也分为主文件组和辅助文件组。

主文件组(Primary)是数据库系统提供的,每个数据库有且仅有一个主文件组。主文件组中包含了所有系统表、主数据文件和没有明确指派给其他文件组的其他数据文件。

辅助文件组是用户在使用 CREATE DATABASE 语句时自行定义的文件组。每个数据库既可以有若干个辅助文件组,也可以没有辅助文件组。辅助文件组可以包含用户指定的辅助数据文件。

默认文件组是没有分配文件组的用户自定义对象的首选文件组,每个数据库只能有一个默认文件组。注意,默认文件组和主文件组不是同一个概念,数据库初始建立时,主文件组是默认文件组,db_owner 固定数据库角色成员可以通过命令将用户定义的文件组指定为后续数据文件的默认文件组。

4) 盘区与页

SQL Server 利用盘区和页数据结构给数据库对象分配存储空间。每个盘区是由 8 个连续页组成的数据结构,大小为 8KB×8=64KB。当创建一个数据库对象(如一个表、一个索引)时,SQL Server 自动以盘区为单位给它们分配存储空间。每个盘区只能包含一个数据库对象,每个数据库对象可以占用多个盘区。

SQL Server 页的大小为 8KB,每页开始部分的 96 字节是页头信息,用于存放页的类型、该页的可用空间数量、占用该页的数据库对象的对象标识等系统信息;其余的 8096 字节用于存放该页数据库对象的数据信息。SQL Server 2014 中的页分为数据页、索引页、文本页、图像页等 8 种。

2. CREATE DATABASE 语句

在 SQL Server 中,数据库有系统数据库和用户数据库两类。系统数据库在安装 SQL Server 系统时由系统自动创建,用来存放 SQL Server 专用的,用于管理自身和用户数据库的数据。用户数据库是用户创建的数据库,用来存放用户数据。

创建用户数据库的过程实际上是确定数据库名称、数据库相关文件存放位置、存储空间

大小及其相关属性的设置。T-SQL 语言中 CREATE DATABASE 语句格式如下：

```
CREATE DATABASE <数据库名>
    [ ON [ primary ] <文件说明> [,…n ]
          [ , <文件组说明> [,…n ] ]
       [ LOG ON <文件说明> [,…n ] ]
    ]
```

语句格式中的方括号表示其中的内容可有可无，是可选项。下面通过举例来说明该语句的用法。

例 3.1 创建一个数据库 test1，其他所有参数都取默认值。

```
CREATE DATABASE test1
```

说明：T-SQL 语言是大小写不敏感的，所以语句也可以写成：create database test1。SQL Server 中，每个数据库至少有两个文件（一个主数据文件和一个日志文件）和一个主文件组，所以语句执行后会自动创建一个数据文件、一个日志文件、一个文件组。

例 3.2 创建一个数据库 test2，要求主数据文件逻辑名为 test2_data，物理文件名为 d:\database\test2_data.mdf，其他所有参数都取默认值。

```
CREATE DATABASE test2
    ON
       ( name = test2_data, filename = 'd:\database\test2_data.mdf' )
```

说明：逻辑文件名是指在 T-SQL 语句中引用文件时使用的名称。创建数据库时，用户可以只指定数据文件，而不指定日志文件；但不可以不指定数据文件，而只指定日志文件。

例 3.3 创建一个数据库 test3，要求：①主数据文件逻辑名为 test3_data，物理文件名为 d:\database\test3_data.mdf，文件初始大小为 10MB，最大容量不受限制，文件增长量为 2MB；②日志文件逻辑名为 test3_log，物理文件名为 d:\database\test3_log.ldf，文件初始大小为 5MB，最大容量为 10MB，文件增长量为 5%。

```
CREATE DATABASE test3
    ON
       ( name = test3_data, filename = 'd:\database\test3_data.mdf',
         size = 10, maxsize = unlimited, filegrowth = 2 )
    LOG ON
       ( name = test3_log, filename = 'd:\database\test3_log.ldf',
         size = 5, maxsize = 10, filegrowth = 5 % )
```

说明：①关键字 ON 引导的是数据文件，而关键字 LOG ON 引导的是日志文件；②每一个文件的属性信息单独写在一对圆括号内，各属性之间用逗号隔开；③同类型的文件之间也用逗号隔开。

例 3.4 创建一个数据库 test4，要求：①数据文件有 4 个，其逻辑名分别为 test4a_data、test4b_data、test4c_data、test4d_data，物理文件都放在 d:\database 文件夹中，其文件名分别为 test4a.mdf、test4b.ndf、test4c.ndf、test4d.ndf，文件其他所有参数都取默认值；②文件 test4a_data 组成主文件组 primary，文件 test4b_data 和 test4c_data 组成辅助文件组 group1，文件 test4d_data 组成辅助文件组 group2；③日志文件逻辑名为 test4_log，物理文

件名为 d:\database\test4.ldf,文件其他所有参数都取默认值。

```
CREATE DATABASE test4
    ON primary
       ( name = test4a_data, filename = 'd:\database\test4a.mdf' ),
       filegroup group1
       ( name = test4b_data, filename = 'd:\database\test4b.ndf' ),
       ( name = test4c_data, filename = 'd:\database\test4c.ndf' ),
       filegroup group2
       ( name = test4d_data, filename = 'd:\database\test4d.ndf' )
    LOG ON
       ( name = test4_log, filename = 'd:\database\test4.ldf' )
```

本语句执行后数据库 test4 共有 3 个文件组,其中的主文件组(primary)是默认文件组。

3. CREATE SCHEMA 语句

在 SQL 语言中,定义模式(Schema)实际上是定义了一个命名空间,在这个空间中可以定义该模式包含的数据库对象,如基本表、视图等。目前,在 CREATE SCHEMA 中可以接受 CREATE TABLE、CREATE VIEW 和 GRANT 子句,也就是说用户在创建模式的同时可以在这个模式定义中进一步创建基本表、视图,定义授权。T-SQL 语言支持 CREATE SCHEMA 语句,创建的模式(T-SQL 语言称为架构)都属于当前数据库,该语句格式如下:

```
CREATE SCHEMA <模式名> [ AUTHORIZATION <所有者名> ]
    [ { <表定义子句> | <视图定义子句> | <授权定义子句> } [,…n ] ]
```

说明:语句格式中的"|"表示或,花括号表示必须在其中选择一项。模式的所有者可以是用户或角色,在模式内创建的对象由模式所有者拥有。如果在创建模式时缺省所有者,则所有者为当前用户。

例 3.5　为 test2 数据库中的用户 dbo 创建一个模式 Study。

```
CREATE SCHEMA Study AUTHORIZATION dbo
```

例 3.6　为 test2 数据库中的用户 dbo 创建一个模式 Exam,并在其中定义一张表 Table1。

```
CREATE SCHEMA Exam AUTHORIZATION dbo
    CREATE TABLE Table1( Tno smallint, Cname varchar(20), Result char(2) )
```

4. ALTER DATABASE 语句

T-SQL 语言提供了修改数据库语句 ALTER DATABASE,通过该语句可以增加或删除数据文件(或日志文件),增加或删除文件组,修改文件或文件组的属性,也可以重命名文件组名或数据库名。ALTER DATABASE 语句格式如下:

```
ALTER DATABASE <数据库名>{
    ADD FILE <文件说明>[,…n] [TO FILEGROUP <文件组名>]
    | ADD LOG FILE [<文件说明>[,…n]]
    | REMOVE FILE <逻辑文件名>
    | ADD FILEGROUP <文件组名>
```

```
| REMOVE FILEGROUP <文件组名>
| MODIFY FILE <文件说明>
| MODIFY FILEGROUP <文件组名> {<文件组属性> | < NAME = 文件组新名>}
| MODIFY NAME = <数据库新名>
}
```

文件组属性有只读 Readonly、可读写 Readwrite 以及默认文件组 Default 三种，主文件组不能设置为只读。

例 3.7 将例 3.2 中数据库 test2 的主数据文件的最大容量改为 6MB，文件增长量改为 10%。

```
ALTER DATABASE test2
    MODIFY FILE ( name = test2_data, maxsize = 6, filegrowth = 10 % )
```

例 3.8 给例 3.3 中数据库 test3 添加一个包含两个辅助数据文件的辅助文件组 usergroup。

```
ALTER DATABASE test3
    ADD FILEGROUP usergroup
GO
ALTER DATABASE test3
    ADD FILE ( name = test3a_data, filename = 'd:\database\test3a_data.ndf'),
             ( name = test3b_data, filename = 'd:\database\test3b_data.ndf')
             TO FILEGROUP usergroup
```

说明：必须先添加文件组，再添加数据文件，这两个语句之间需加 GO 命令。GO 命令不是 SQL 语言中的语句，也不是 T-SQL 语言中的语句，它用于向 SQL Server 服务器发出一个信号，表示一批 T-SQL 语句结束了。

例 3.9 删除例 3.4 中数据库 test4 中的数据文件 test4d_data 和文件组 group2。

```
ALTER DATABASE test4
   REMOVE FILE test4d_data
GO
ALTER DATABASE test4
   REMOVE FILEGROUP group2
```

说明：必须先删除数据文件，再删除文件组。删除数据文件时，用的是文件的逻辑名，而不是物理名。

5. ALTER SCHEMA 语句

T-SQL 语言提供了修改模式语句 ALTER SCHEMA，通过该语句可以在同一数据库中的模式之间移动对象。ALTER SCHEMA 语句格式如下：

```
ALTER SCHEMA <模式名> TRANSFER <对象名>
```

例 3.10 将数据库 test2 中的模式 Exam 中的表 Table1 移到模式 Study 中。

```
ALTER SCHEMA Study TRANSFER Exam.Table1
```

6. DROP DATABASE 语句

当一个数据库中的数据失去使用价值后，可以删除该数据库以释放被占用的磁盘空间。

数据库被删除后，数据库中的所有数据库对象和数据库所使用的所有磁盘文件会一起被删除。T-SQL 语言中删除数据库的语句格式如下：

```
DROP DATABASE <数据库名>
```

例 3.11　删除数据库 test4。

```
DROP DATABASE test4
```

7. DROP SCHEMA 语句

当一个数据库中的模式失去使用价值后，可以删除该模式。SQL 语言中删除模式语句的格式如下：

```
DROP SCHEMA <模式名> { CASCADE | RESTRICT }
```

说明：语句中的 CASCADE 和 RESTRICT 两者必选其一。CASCADE(级联)表示在删除模式的同时把该模式中的所有数据库对象全部删除；RESTRICT(限制)表示如果该模式中定义了下属的数据库对象(如表、视图等)，则拒绝执行该删除语句，只有当该模式中没有任何下属的对象时，才能执行 DROP SCHEMA 语句。

T-SQL 语言支持 DROP SCHEMA 语句，但被删除的模式中不能包含任何对象，所以该语句不能有 CASCADE 或 RESTRICT 选项。

例 3.12　删除数据库 test2 中的模式 Exam 和 Study。

```
DROP SCHEMA Exam
GO
DROP TABLE Study.Table1
GO
DROP SCHEMA Study
```

在结束本节之前，对数据库和模式再做一些说明。早期数据库的用户不得不相互协调以保证他们没有对不同的关系(表)使用相同的名字，当代数据库系统提供了三层结构的关系(表)命名机制。最顶层的由目录(Catalog)构成(在一些 RDBMS 的实现中用术语“数据库”代替术语“目录”)，每个目录都可以包含模式，诸如表和视图等 SQL 对象都包含在模式中。每个连接到 RDBMS 的用户都有一个默认的目录和模式，如果用户想访问其他目录和模式中的表，那么必须指定目录名和模式名。

3.2.2　SQL 中的数据类型

关系模型中一个很重要的概念是域。关系中的每一个属性来自一个域，它的取值必须是域中的值。关系模型中域的概念，在 SQL 语言中用数据类型来实现。另一方面，与高级语言中的数据类型一样，SQL 语言中的数据类型就是指数据在计算机内的表现形式，包括数据的存储格式、分配的字节数、取值范围、数据的精度和小数位数以及可参加的运算。

1. 基本数据类型

SQL 标准定义了多种主要的数据类型，如表 3.1 所示。

表 3.1 SQL 的主要数据类型

数 据 类 型	含 义
char(n)	长度为 *n* 的定长字符串
varchar(n)	最大长度为 *n* 的变长字符串
int	整数类型(等价于 integer)
smallint	小整数类型
numeric(p,d)	定点数,由 p 位数字(不包括符号、小数点)组成,小数点后面有 d 位数字
real	单精度浮点数
double precision	双精度浮点数
float(n)	精度至少为 n 位数字的浮点数
date	日期,包含年、月、日,如'2018-07-30'
time	时间,包含时、分、秒,如'21:30:10'
timestamp	date 和 time 的组合,如'2018-07-30 21:30:10'

2. SQL Server 2014 中的数据类型

在 SQL Server 2014 中,数据类型包括字符类型、数值类型、日期时间类型、货币类型和二进制数据类型等,而数值类型又分为整数类型、精确数值类型和近似(浮点)数值类型。

1) 字符类型

字符类型用于存储非 Unicode 字符,包括汉字、英文字母、数字、标点符号及其他字符,有以下三种类型。

char(n)用于存储定长字符串,1≤n≤8 000,占用 n 字节存储空间。如字符串实际存储字节数小于 n 字节,则补足存储。如字符串实际存储字节数大于 n 字节,则截断存储。注意,char(4)能存储 4 个英文字母,但只能存储 2 个汉字,因为每个汉字需要 2 字节存储空间。

varchar(n)用于存储可变长度字符串,1≤n≤8 000。如字符串实际存储字节数小于 n+2 字节,则按实际字节数存储。如字符串实际存储字节数大于 n+2 字节,则截断存储。

text 可用于存储最多 $2^{31}-1$ 字节的非 Unicode 字符数据,如一个报告或一篇小说。

2) Unicode 字符类型

Unicode 字符类型中的每个字符都采用 2 字节编码,它也有三种类型,其用法与字符类型中相应类型的用法相同。

nchar(n)用于存储定长 Unicode 字符串,1≤n≤4 000,占用 2n 字节存储空间。注意,nchar(4)能存储 4 个英文字母,也能存储 4 个汉字。

nvarchar(n)用于存储可变长度 Unicode 字符串,1≤n≤4 000。

ntext 可用于存储最多 $2^{30}-1$ 个 Unicode 字符数据。

3) bit 类型

bit 类型只能用于存储 1、0 或 null,占用 1bit 的存储空间。如果一个表中只有一个 bit 类型的列,则也要占用 1 字节;如果一个表中有小于等于 8bit 类型的列,则这些列作为 1 字节存储;如果有 9～16 个 bit 类型的列,则这些列作为 2 字节存储,以此类推。

4) 整数类型

整数类型用于存储整数,有四种类型。

tinyint 用于存储范围在 0～255 的整数，占用 1 字节存储空间。

smallint 用于存储范围在 -2^{15}～$2^{15}-1$ 之间的整数，占用 2 字节存储空间。

int 用于存储范围在 -2^{31}～$2^{31}-1$ 之间的整数，占用 4 字节存储空间。

bigint 用于存储范围在 -2^{63}～$2^{63}-1$ 之间的整数，占用 8 字节存储空间。

5) 精确数值类型

精确数值类型为 decimal[(p[,s])](等价于 numeric[(p[,s])])用于存储 $-10^{38}+1$～$10^{38}-1$ 之间的数据。它在使用时可以指定精度 p(1≤p≤38)和小数位数 s(0≤s≤p)，默认精度 p 为 18，默认小数位数 s 为 0。精确数值类型占用的存储空间与 p 的取值有关。当 p 取值 1～9 时，占用 5 字节；当 p 取值 10～19 时，占用 9 字节；当 p 取值 20～28 时，占用 13 字节；当 p 取值 29～38 时，占用 17 字节。

6) 近似(浮点)数值类型

近似数值类型用于存储浮点数，有两种类型。

real 用于存储 -3.40E+38～3.40E+38 之间的浮点数，占用 4 字节存储空间。

float[(n)]用于存储 -1.79E+308～1.79E+308 之间的浮点数，1≤n≤53，n 的默认值为 53。SQL Server 2014 对 n 做这样处理：当 n 为 1～24 时，则将 n 视为 24，有 7 位精度，占用 4 字节存储空间；当 n 为 25～53 时，则将 n 视为 53，有 15 位精度，占用 8 字节存储空间。

注意，real 的 SQL92 同义词为 float(24)，double precision 的同义词为 float(53)，但 SQL Server 2014 本身没有 double precision 类型。

7) 日期时间类型

日期时间类型用于存储日期和时间，它等价于 SQL2003 标准中的 timestamp 数据类型。如果没有提供日期，则将 1900 年 1 月 1 日作为默认值；如果没有提供时间，则将 00:00:00.000 作为默认值。年月日之间的分隔符可以是连字符(-)、斜杠(/)或小数点(.)。它有两种类型。

smalldatetime 用于存储 1900 年 1 月 1 日—2079 年 6 月 6 日的日期时间，精度为 1 分钟，占用 4 字节存储空间。

datetime 用于存储 1753 年 1 月 1 日—9999 年 12 月 31 日的日期时间，精度为 3.33 毫秒，占用 8 字节存储空间。

8) 货币类型

货币类型精确到它们所代表的货币单位的万分之一，有两种类型。

smallmoney 用于存储 -214 748.364 8～214 748.364 7 之间的数据，占用 4 字节存储空间。

money 用于存储 -922 337 203 685 477.580 8～922 337 203 685 477.580 7 之间的数据，占用 8 字节存储空间。

9) 二进制数据类型

二进制数据类型用于存储图像、声音等数据，有三种类型。

binary(n)用于存储定长二进制数据，1≤n≤8 000，占用 n 字节存储空间。如数据实际存储字节数小于 n 字节，则补足存储。如数据实际存储字节数大于 n 字节，则截断存储。

varbinary(n)用于存储可变长二进制数据，1≤n≤8 000。如数据实际存储字节数小于 n+2 字节，则按实际字节数存储。如数据实际存储字节数大于 n+2 字节，则截断存储。

image 可用于存储最多 $2^{31}-1$ 字节的二进制数据，如一首歌曲或一部电影。

最后要说明的是：

(1) SQL Server 的未来版本中将删除 text、ntext 和 image 数据类型，分别改为 varchar(max)、nvarchar(max) 和 varbinary(max)数据类型。因此，请尽量避免在新开发工作中使用 text、ntext 和 image 数据类型，并考虑修改当前还在使用这些数据类型的应用程序。顺便说明一下，SQL 语言标准中提供了与此相关的数据类型，称为“大对象类型”。大对象类型有 clob 和 blob 两种，前者用于存放字符数据，而后者用于存放二进制数据。显然，T-SQL 语言在支持大对象类型时采用了非标准格式。

(2) SQL Server 2014 中也有 timestamp 数据类型，但它不同于 SQL 2003 标准中定义的 timestamp 数据类型。timestamp 通常用于给表中的行加版本戳的机制，存储大小为 8 字节。一个表只能有一个 timestamp 列，每次插入(或修改)包含 timestamp 列的行时，就会在 timestamp 列中插入(或修改)时间戳值(它是一个相对时间，而不是与时钟相关联的实际时间)，以便确定该行中的任何值自上次读取以后是否发生了修改。

3. 用户自定义的数据类型

SQL 语言提供了两种形式的用户自定义数据类型。一种称为独特类型(Distinct Type)，另一种称为结构化数据类型(Structured Data Type)，如记录结构、数组、多重集合等复杂的数据类型。在 SQL99 标准中提供了 CREATE TYPE 语句来创建用户自定义数据类型。下面只介绍独特类型，不介绍结构化数据类型。

SQL Server 将独特类型的用户自定义数据类型称为别名数据类型。在 SQL Server 早期的版本中只能通过系统存储过程 sp_addtype 来创建别名数据类型，SQL Server 2014 支持 CREATE TYPE 语句，所以也可以用该语句来创建别名数据类型。下面通过举例说明如何用 sp_addtype 和 CREATE TYPE 创建别名数据类型。

例 3.13 创建别名数据类型 ssn，它基于 char(8)，并且不允许取空值。

```
EXEC sp_addtype ssn, 'char(8)', 'NOT NULL'
```

或

```
CREATE TYPE ssn FROM char(8) NOT NULL
```

需要说明的是：

(1) 用户创建的别名数据类型如果不再使用，可以删除。如例 3.13 中创建的 ssn 可以用 EXEC sp_droptype ssn 或 DROP TYPE ssn 删除。

(2) 虽然可以用 sp_addtype 和 CREATE TYPE 两种方法创建别名数据类型，但考虑到在 SQL Server 的未来版本中将删除 sp_addtype，所以请尽量避免在新的开发工作中使用 sp_addtype，改为使用 CREATE TYPE 语句。另外，SQL Server 2014 中系统将自动授予 PUBLIC 数据库角色对通过使用 sp_addtype 创建的别名数据类型具有 REFERENCES 权限。当使用 CREATE TYPE 语句而不是 sp_addtype 创建别名数据类型时，系统不会进行自动授权。

(3) 就创建独特类型的用户自定义数据类型而言，SQL Server 2014 中的 CREATE TYPE 语句与 SQL99 标准中的 CREATE TYPE 语句用法相似，其他情况下用法差异较大。

(4) SQL92 标准中通过引入 CREATE DOMAIN 语句来创建域,域与类型相似但又有差异,关于域与类型的差别请参考相关文献。SQL Server 2014 不支持 CREATE DOMAIN 语句。

3.2.3　基本表的创建、修改与删除

基本表是实际存在的表,在数据库中既要存放它的定义(即基本表的结构),又要存放它的数据,基本表的定义存放在数据库的数据字典中。在 SQL Server 中,基本表分为系统表和用户表两种。

系统表中的数据构成了 SQL Server 系统的数据字典,系统表记录了服务器所有活动的信息,包括所有存储在其中的对象(如数据类型、表、列、约束、索引、视图等)、系统配置信息、本地登录信息、错误或警告信息、数据封锁信息等。一些系统表只存在于 master 数据库中,它们包含系统级信息;另一些系统表则存在于每一个数据库(包括 master 数据库)中,它们包含属于这个特定数据库的对象和资源的相关信息。用户不能直接修改系统表,也不能直接检索系统表中的信息,如确要检索存储在系统表中的信息,应通过系统视图(sys 开头或 INFORMATION_SCHEMA 开头的视图)来进行。

用户表是由用户创建的表,用来存储用户的数据,它又分为永久表和临时表两种。永久表存储在用户数据库中,用户数据通常存储在永久表中,如果用户没有删除永久表,永久表及其存储的数据将永久存在。临时表存储在 tempdb 数据库中,临时表又分为本地临时表和全局临时表。本地临时表的表名以 # 开头,仅对连接数据库的当前用户有效,用户断开连接,自动删除。全局临时表的表名以 # # 开头,对连接数据库的所有用户有效,所有用户断开连接,才自动删除。

1. CREATE TABLE 语句

关系模型中域的概念在 SQL 语言中用数据类型来实现,所以在定义表的列(即属性)时,需要指明其数据类型及长度。创建表时还可以定义各类完整性约束条件,3.3 节中将详细介绍如何在创建表时定义各类完整性约束条件。T-SQL 语言中 CREATE TABLE 语句格式如下:

```
CREATE TABLE [<数据库名> . [ <模式名> ] . | <模式名> . ] <表名>
    ( <列名> <数据类型> [ <列级完整性约束条件> ] [,…n]
     [ , <表级完整性约束条件> [,…n] ]
     [ ON { 文件组名 | default } ]
    )
```

说明:在创建表时,如果缺省数据库名,则创建在当前数据库中;如果缺省模式名,则创建的表属于当前用户的默认模式。另外,通过 ON 子句,用户可以指定存储表的文件组。

例 3.14　创建一张名为 S 的学生表。

```
CREATE TABLE S
(  Sno    char(8) ,
   Sname  char(10) ,
   Ssex   char(2) ,
   Sage   tinyint ,
   Major  char(8)
)
```

实际应用常常要求创建与现有的某个表的模式(结构)相同的表,SQL 语言提供了 CREATE TABLE LIKE 语句来支持这项任务。如 CREATE TABLE S_Copy LIKE S 创建的表 S_Copy 与例 3.14 中创建的表 S 的结构完全相同。MySQL 5.5 支持 CREATE TABLE LIKE 语句,但 SQL Server 2014 不支持该语句。

2. ALTER TABLE 语句

随着应用环境和应用需求的变化,有时需要修改已经创建好的基本表。修改表不仅能修改表的结构(如增加列、删除列和修改现有列的数据类型),还能增加约束、删除约束等。但是修改表时,不能破坏表原有的数据完整性,如不能为有空值的列设置为主码等。T-SQL 语言中 ALTER TABLE 语句格式如下:

```
ALTER TABLE [<数据库名> . [ <模式名> ] . | <模式名> . ] <表名> {
   ADD { <列名> <数据类型> [ <列级约束条件> ] | <表级约束条件> } [ , … n ]
   | DROP { COLUMN <列名> | [ CONSTRAINT ] <约束名> } [ , … n ]
   | ALTER COLUMN <列名> <数据类型>
   }
```

例 3.15 修改例 3.14 中创建的 S 表,给它增加 Address 和 Mphone 列,同时将 Major 列的数据类型改为 char(12)。

```
ALTER TABLE S
   ADD Address varchar(50) , Mphone char(11)
GO
ALTER TABLE S
   ALTER COLUMN Major char(12)
```

3. DROP TABLE 语句

当表确实不需要时,可以删除表。表的所有者可以删除所属数据库中的任何表,但不能删除系统表和外码约束所参照的表,如果确实需要删除,必须先删除外码约束或参照表。删除表的同时会删除表中的所有数据以及表相关的索引、约束、触发器和指定的权限。任何引用已删除表的视图或存储过程都必须使用 DROP VIEW 或 DROP PROCEDURE 显式删除。T-SQL 语言中 DROP TABLE 语句格式如下:

```
DROP TABLE [<数据库名> . [ <模式名> ] . | <模式名> . ] <表名> [ , … n ]
```

例 3.16 删除例 3.14 中创建的 S 表。

```
DROP TABLE S
```

3.3 完整性约束的实现

3.3.1 数据库完整性的概念

数据库完整性是指数据库的任何状态变化都能反映真实存在的客观世界的合理状态,数据库中的数据应始终保持正确且合理的状态。也就是说,数据库的完整性是指数据的正确性和相容性。所谓正确性是指数据是有效的,有意义的,而不是荒谬的或不符合实际的。例如,学生的学号必须唯一;性别只能是男或女;学生所选修的课程必须是本校开设的课

程；成绩必须是 0～100 的数。所谓相容性是指数据之间不能相互矛盾。例如，从一个数据本身来说，工人的年龄是 30 岁和工人的工龄 20 岁两者都是正确的，但如果同一个工人的年龄是 30 岁而工龄是 20 岁却是矛盾的，不相容的。

关系数据库中数据的正确性和相容性是通过关系模型中的完整性约束条件来保证的。从理论上来说，这一组完整性约束条件的实现既可以在应用程序中实现，也可以由 DBMS 系统自动实现。由应用程序实现的缺点是对每个更新操作都要编写程序进行完整性检查，显得比较烦琐且效率低下。而由 DBMS 实现就有效减轻了程序员的负担，既提高了完整性检测的效率又可以防止漏检。

为了维护数据库的完整性，DBMS 必须具有如下功能。

(1) 提供定义完整性约束条件的机制。现在，这一机制是由 SQL 语言中的 DDL 语句(如 CREATE TABLE 语句)提供的。

(2) 提供完整性检查的方法。DBMS 检查数据是否满足完整性约束条件的机制称为完整性检查。现在，一般在 INSERT、DELETE、UPDATE 语句执行后开始检查，也可以在事务提交时检查。检查这些操作执行后，数据库中的数据是否违反了完整性约束条件。

(3) 违约处理。DBMS 若发现操作违反了完整性约束条件，就应采取一定的动作(如拒绝执行该操作)进行违约处理来保证数据库的完整性。

目前商用的 DBMS 产品普遍都支持完整性控制，即 DBMS 都有完整性定义、完整性检查和违约处理这三方面的机制。

3.3.2　各类完整性约束的实现

1. 实体完整性约束

关系模型中的实体完整性约束可用 CREATE TABLE 语句中的 PRIMARY KEY 短语实现。用 PRIMARY KEY 定义一个表的主码，也就实现了实体完整性约束。当主码由单个属性构成时，PRIMARY KEY 可以定义为列级约束条件，也可以定义为表级约束条件。当主码由多个属性构成时，PRIMARY KEY 必须定义为表级约束条件。

定义了实体完整性约束后，当插入或修改操作使得表中属性的取值违反实体完整性约束(即主码值不唯一或主码的各个属性中有一个属性取值为空)时，系统一般采用拒绝执行方式处理。

2. 参照完整性约束

关系模型中的参照完整性约束可用 CREATE TABLE 语句中的 FOREIGN KEY… REFERENCES 短语实现。其中用 FOREIGN KEY 定义一个表的外码，用 REFERENCES 指明外码参照哪张表的主码，这样就实现了参照完整性约束。当外码由单个属性构成时，FOREIGN KEY 可以定义为列级约束条件，也可以定义为表级约束条件。当外码由多个属性构成时，FOREIGN KEY 必须定义为表级约束条件。另外，在定义参照完整性约束时，参照表的外码的列数与被参照表主码的列数必须相同，并且对应列的数据类型也必须相同，但是外码的列名与被参照表主码的列名不必相同。

一个参照完整性约束将两张表中的相关元组联系起来了。以后，当对被参照表或参照表进行插入、删除或修改时都有可能破坏参照完整性约束，必须进行检查。表 3.2 给出了可能破坏参照完整性的情况及违约处理方式。

表 3.2 可能破坏参照完整性的情况及违约处理方式

被参照表(例如 S)		参照表(例如 SC)	违约处理方式
可能破坏	←	插入元组	拒绝
可能破坏	←	修改外码值	拒绝
删除元组	→	可能破坏	拒绝/级联删除/设置为空值
修改主码值	→	可能破坏	拒绝/级联修改/设置为空值

下面介绍违约处理方式。

(1) 拒绝(NO ACTION)执行,即不允许执行该操作,一般为默认违约处理方式。

(2) 级联(CASCADE)操作,表示当删除或修改被参照表的一个元组造成了参照表中某些元组的外码违反了参照完整性约束,则系统会自动删除或修改参照表中所有违反参照完整性约束的元组。例如,删除了被参照表 S 中学号为 16001 的学生,则系统会自动删除参照表 SC 中所有学号为 16001 的元组。

(3) 设置为空值(SET NULL),表示当删除或修改被参照表的一个元组造成了参照表中某些元组的外码违反了参照完整性约束,则系统会自动将参照表中所有违反参照完整性约束的元组的外码设置为空值。例如,设有下列关系模式(见 2.4.2 节)。

专业(专业号,专业名,创办日期,所属学院)

学生(学号,姓名,性别,年龄,专业号,班长)

课程(课程号,课程名,课程类型,学时,学分)

选修(学号,课程号,选修日期,成绩)

当专业表中专业号为 14 的元组被删除后,学生表中专业号为 14 的所有元组的专业号(外码)设置为空值。这样处理的语义是:某个专业被删除了(撤销了),该专业的所有学生专业未定,等待重新分配专业。

需要说明的是,有时不能选择“设置为空值”这种处理方式。例如,当学生表中学号为 16001 的学生被删除后,选修表中的学号(外码)不能设置为空值。因为选修表中的学号既是外码又是本表主码中的属性,如果主码中的属性取空值,就违反了实体完整性约束。

3. 用户定义的完整性约束

关系模型中的用户定义的完整性约束就是针对某一具体应用的数据必须满足的语义要求,这种约束具体反映在三个方面:①单个属性上的完整性约束;②同一个表中各属性之间(即元组上)的完整性约束;③多个表中各属性之间的完整性约束。通过这三个方面的约束来对表中属性的取值进行限制。当插入或修改操作使得表中属性的取值违反用户定义的完整性约束时,系统一般采用拒绝执行的方式处理。

一个关系的关系模式中的属性(列)和域分别对应 SQL 语言中的一个基本表的属性和数据类型。虽然在用 CREATE TABLE 语句创建基本表时已经定义了各属性所对应的数据类型,但为了实现属性上和元组上的完整性约束,还应分别进一步定义列级完整性约束条件和表级完整性约束条件。在 SQL 语言中,列级和表级完整性约束都有 NOT NULL(属性取非空值)、UNIQUE(属性取值唯一)和 CHECK(检查属性值是否满足一个逻辑表达式)三种形式。另外,T-SQL 语言中的 DEFAULT(属性取默认值)也可以理解为列级完整性约束。

多个表中各属性之间的完整性约束(除了参照完整性约束)的实现一般可采用触发器机制。关于触发器的详细介绍见第 6 章。

4. 各类完整性约束实现举例

下面采用 T-SQL 语言来举例说明各类完整性约束的实现,例中所涉及的关系模式(即为 2.5.4 节中的关系模式)如下。

学生关系模式:S(Sno,Sname,Ssex,Sage,Major,Address,Mphone)

课程关系模式:C(Cno,Cname,Ctype,Ctime,Credit)

教师关系模式:T(Tno,Tname,Tsex,Tage,Title,Salary)

选修关系模式:SC(Sno,Cno,Sdate,Score)

授课关系模式:TC(Tno,Cno,Tdate,Remark)

例 3.17　创建学生表 S、课程表 C 和教师表 T,同时定义各表的主码并给相关属性加上必要的约束条件,以实现实体完整性约束和用户定义的完整性约束。

```
CREATE TABLE S
(  Sno      char(8)       PRIMARY KEY ,                    /* 实体完整性 */
   Sname    char(10)      NOT NULL ,                       /* 用户定义的完整性 */
   Ssex     char(2)       CHECK ( Ssex IN ('男','女') ) ,  /* 用户定义的完整性 */
   Sage     tinyint ,
   Major    char(8) ,
   Address  varchar(50) ,                                  /* 下面也是用户定义的完整性 */
   Mphone   char(11)      CONSTRAINT  uq_S  UNIQUE
)
CREATE TABLE C
(  Cno      char(6)       PRIMARY KEY ,
   Cname    char(20)      NOT NULL ,
   Ctype    char(4)       CHECK ( Ctype IN ('公共','基础','必修','选修','实践') ) ,
   Ctime    tinyint       CHECK ( Ctime BETWEEN 18 AND 108 ) ,
   Credit   tinyint       CHECK ( Credit BETWEEN 1 AND 6 )
)
CREATE TABLE T
(  Tno      char(8)       PRIMARY KEY ,
   Tname    char(10)      NOT NULL ,
   Tsex     char(2)       CHECK ( Tsex IN ('男','女') ) ,
   Tage     tinyint       CHECK ( Tage BETWEEN 20 AND 80 ) ,
   Title    char(6)       CHECK ( Title IN ('助教','讲师','副教授','教授') ) ,
   Salary   decimal(8,2)  CHECK ( Salary BETWEEN 2000 AND 100000 )
)
```

说明:在 T-SQL 语言中,用"/*"表示注释的开始,用"*/"表示注释的结束。CHECK 条件中出现的 IN 和 BETWEEN…AND 运算符的含义在 3.4 节中介绍。

例 3.18　创建选修表 SC 和授课表 TC,同时定义各表的主码和外码以实现实体完整性约束和参照完整性约束。

```
CREATE TABLE TC
(  Tno   char(8) FOREIGN KEY REFERENCES T(Tno) ON UPDATE CASCADE ,
   Cno   char(6) FOREIGN KEY REFERENCES C(Cno) ON UPDATE CASCADE ,
   Tdate  char(6) ,
```

```
    Remark decimal(3,1) ,
    CONSTRAINT   pk_TC   PRIMARY KEY ( Cno,Tdate )
)
CREATE TABLE SC
(   Sno   char(8) FOREIGN KEY REFERENCES S(Sno) ON UPDATE CASCADE ,
    Cno   char(6) ,
    Sdate   char(6) ,
    Score   tinyint ,
    CONSTRAINT   pk_SC   PRIMARY KEY ( Sno,Cno,Sdate ) ,
    FOREIGN KEY ( Cno,Sdate ) REFERENCES TC ( Cno,Tdate )
                          ON DELETE CASCADE ON UPDATE CASCADE
)
```

说明：本例中假定同一门课程同一时间段(即同一个学期)只有一位任课老师，但同一位老师同一时间段可以有一门以上课程的教学任务，所以 TC 表的主码为(Cno,Tdate)。因为两张表的主码都由一个以上的属性组成，所以 PRIMARY KEY 必须作为表级约束条件，而不能作为列级约束条件。注意，SC 表中的外码(Cno,Sdate)与被参照表 TC 中的主码(Cno,Tdate)的属性名可以不同，由于该外码由两个属性组成，所以它也必须作为表级约束条件。另外，外码 Tno、Cno 和 Sno 对于修改都采用级联修改处理方式，对于删除则采用默认的拒绝执行处理方式；而外码(Cno,Sdate)对于删除或修改都采用级联删除或修改处理方式。

3.3.3 表中完整性约束的增加与删除

前面在用 CREATE TABLE 语句实现各类完整性约束时，对表级或列级的约束条件，有的用关键字 CONSTRAINT 给约束条件命名，有的没有用关键字 CONSTRAINT 给约束条件命名。事实上，在 SQL Server 2014 中无论对表级还是列级约束条件，只要用户没有给约束条件命名，系统都会自动给约束条件命名，如图 3.2 所示。由于在例 3.17 中用 CREATE TABLE 语句创建表 S 时，只有 UNIQUE 约束是命名的，其他三个约束都没有命名，但从图中可以看出其中两个约束(NOT NULL 除外)系统都自动给命名了。

图 3.2 例 3.17 中创建的表 S

如果在创建基本表时，遗漏了各类完整性约束条件(NOT NULL 除外)，都可以用 ALTER TABLE 语句增加。

例 3.19 如果规定：①职称不是教授的教师必须在 60 岁退休；②对教师授课的评价在 0 到 10 分之间；③学生选修某门课程的成绩在 0 到 100 分之间。请给已经创建的基本表 T、TC 和 SC 增加这些完整性约束条件。

```
ALTER TABLE T
    ADD CONSTRAINT   ck_T
                CHECK ( Title!= '教授' AND Tage < 60 OR Title = '教授' )
ALTER TABLE TC
    ADD CONSTRAINT   ck_TC CHECK ( Remark BETWEEN 0 AND 10 )
```

```
ALTER TABLE SC
   ADD  CONSTRAINT  ck_SC  CHECK ( Score BETWEEN 0 AND 100 )
```

对于实际具体应用取消了的某些约束条件(NOT NULL 除外),可以用 ALTER TABLE 语句删除。如果要修改已经存在的某些约束条件,可以通过删除后再增加的方法来实现。给完整性约束条件命名的作用就在于方便删除。

例 3.20　如果规定对教师授课的评价在 0 到 5 分之间,请修改例 3.19 中的完整性约束条件 ck_TC。

```
ALTER TABLE TC
   DROP  CONSTRAINT  ck_TC
ALTER TABLE TC
   ADD  CONSTRAINT  ck_TC  CHECK ( Remark BETWEEN 0 AND 5 )
```

3.4　数 据 查 询

前面已经介绍了数据库和基本表的创建以及各类完整性约束的实现,用 3.5 节中介绍的 INSERT 语句向表插入数据后,就可以根据用户的需要对表中的数据进行查询了。查询操作是数据库的核心操作,SQL 语言提供了 SELECT 语句实现对数据库的查询,该语句的一般格式如下:

```
SELECT [ALL | DISTINCT] <目标列表达式> [, … ]
FROM <表名或视图名> [, … ]
[ WHERE <元组筛选条件> ]
[ GROUP BY <列名> [, … ] [ HAVING <小组筛选条件> ] ]
[ ORDER BY <列名> [ ASC | DESC ] [, … ] ]
```

整个 SELECT 语句的语义是:根据 WHERE 子句中的元组筛选条件,从 FROM 子句指定的表或视图中找出满足条件的元组,再按 SELECT 子句中的目标列表达式,选出元组中的属性值形成结果表。如果有 GROUP BY 子句,则将结果按指定的列分组。分组后,通常会在每组中作用聚集函数,一个小组中的数据经聚集函数运算后在结果表中形成一个元组。如果 GROUP BY 子句后有 HAVING 子句,则只有满足小组筛选条件的小组才会在结果表输出。如果有 ORDER BY 子句,则结果表中的元组还要按指定列的值升序或降序排序。

由于 SELECT 语句使用灵活、功能强大,下面以 2.5.4 节中的表 2.5～表 2.9 为背景来详细介绍该语句的使用。希望读者把这些举例用于上机实践,加深对语句语义的理解。

3.4.1　单表查询

从 SELECT 语句的一般格式可以看出,许多子句(如 WHERE 子句)是可选项,但 FROM 子句是必选的,它表示查询的对象。如果查询的对象是一张表,就是单表查询。

1. SELECT 子句

SELECT 子句是必选项,用来列出查询结果表中所需要的列,实际上,它就是关系代数中的广义投影运算。

例 3.21 找出每个学生的学号、姓名和出生年份。

```
SELECT Sno, Sname, 2018 - Sage AS BirthYear
FROM S
```

说明：与广义投影一样，SELECT 子句后面列出的可以是属性名，也可以是表达式。如果是表达式可以用“AS 属性名”的形式给表达式对应的列命名，其中关键字 AS 可以不写。

例 3.22 找出全体学生所学专业的专业名。

```
SELECT DISTINCT Major
FROM S
```

说明：投影运算会自动删除结果关系中重复的行，但 SELECT 子句不会这样做。从 SELECT 子句的语法上看，在关键字 SELECT 和“目标列表达式”之间可以加关键字 ALL 或 DISTINCT。如果缺省关键字，默认就是 ALL，也就是保留重复行的意思。显然，加关键字 DISTINCT 就是自动删除查询结果表中重复的行。

例 3.23 找出全体学生的详细信息。

```
SELECT *
FROM S
```

说明：* 表示 S 表中的所有列都要输出显示，输出显示的顺序与创建 S 表时的属性列的顺序一致。如果顺序不一致，还是要把属性一个一个列出来，例如：

```
SELECT Sno, Sname, Ssex, Sage, Mphone, Major, Address
FROM S
```

2. WHERE 子句

WHERE 子句是可选项，只有满足“元组筛选条件”的元组才能出现在查询结果表中，实际上，它就是关系代数中的选择运算。元组筛选条件是一个逻辑表达式，其中常用的运算符见表 3.3，下面通过举例来说明各类运算符的使用。

表 3.3 元组筛选条件中常用的运算符

运算符种类	运 算 符
比较	=,>,>=,<,<=,<>,!=,!>,!<
逻辑	AND,OR,NOT
确定范围	BETWEEN AND,NOT BETWEEN AND
集合属于	IN,NOT IN
字符匹配	LIKE,NOT LIKE
空值比较	IS NULL,IS NOT NULL

例 3.24 找出计算机专业中年龄为 21 岁的学生的学号、姓名和联系电话。

```
SELECT Sno, Sname, Mphone
FROM S
WHERE Major = '计算机' AND Sage = 21
```

说明：SQL 语言使用一对单引号来表示字符串，例如'Computer'。如果单引号是字符串的

组成部分，那就用两个单引号字符来表示，例如字符串“It's right”可表示为“It''s right”。在SQL 标准中，字符串上的比较运算是大小写敏感的，例如'Computer'＝'COMputer'的结果是假。但 SQL Server 2014 在默认情况下做比较运算时是不区分大小写的，所以'Computer'＝'COMputer'的结果是真。另外，如果任何一个参数不属于字符串数据类型，则 SQL Server 2014 会将其转换为字符串数据类型（如果可能）后再比较，例如本例中的条件 Sage＝21 写成 Sage＝'21'也可以（当然不建议这样做）。

例 3.25　找出年龄在 30～50 岁（包括 30 岁和 50 岁）的教师的工号、姓名和职称。

```
SELECT Tno, Tname, Title
FROM T
WHERE Tage BETWEEN 30 AND 50
```

说明：元组筛选条件中的 Tage BETWEEN 30 AND 50 等价于 30 <＝ Tage AND Tage <＝50。而条件不在 30～50 岁可表示为 Tage NOT BETWEEN 30 AND 50，等价于 Tage < 30 OR Tage > 50。

例 3.26　找出职称是教授或副教授的教师的工号、姓名和年龄。

```
SELECT Tno, Tname, Tage
FROM T
WHERE Title IN ('教授', '副教授')
```

说明：元组筛选条件中的 Title IN ('教授','副教授')等价于 Title＝'教授' OR Title＝'副教授'。而条件职称不是教授或副教授可表示为 Title NOT IN ('教授','副教授')，等价于 Title!＝'教授' AND Title!＝'副教授'（注意，这里不能用 OR 运算符）。

字符匹配运算符 LIKE 用来确定某字符串是否与指定的模式串相匹配，其一般格式如下：

```
<待匹配的字符串表达式> [NOT] LIKE <模式串> [ESCAPE <转义字符>]
```

其中模式串可以包含普通字符和通配符。在模式匹配过程中，普通字符必须与待匹配的字符串表达式中指定的字符完全匹配（即是大小写敏感的）。通配符主要有％和_，其中％（百分号）表示任意长度（长度可以为 0）的字符串；_（下画线）表示任意单个字符。

特殊情况下，如果模式串中不包含通配符，则 LIKE 等价于＝运算符，NOT LIKE 等价于!＝运算符。由于通配符可以与字符串中的任意部分字符串相匹配，所以，与使用＝和!＝比较运算符相比，使用通配符可使 LIKE 运算符更加灵活。

需要说明的是：SQL Server 2014 中文版在默认情况下，模式串中的普通字符与待匹配的字符串表达式中指定的字符匹配是不区分大小写的，_既可以表示一个 ASCII 字符，也可以表示一个汉字。

例 3.27　找出家在上海市的学生的学号、姓名和家庭地址。

```
SELECT Sno, Sname, Address
FROM S
WHERE Address LIKE '上海市％'
```

例 3.28　找出职称不是教授或副教授的教师的工号、姓名和年龄。

```
SELECT Tno, Tname, Tage
```

```
FROM T
WHERE Title NOT LIKE '%教授'
```

说明：本例中%可以表示长度为0的字符串，所以职称是教授的教师不满足查询条件。

例 3.29 找出姓名中第二个字是“文”的学生的学号、姓名和家庭地址。

```
SELECT Sno, Sname, Address
FROM S
WHERE Sname LIKE '_文%'
```

说明：本例中_表示一个汉字，%可以表示一个汉字或多个汉字，也可以表示0个汉字，所以姓名是“张文杰”或“李文”的学生均满足查询条件。但姓名是“欧阳文杰”的学生不满足查询条件，因为_只能表示一个汉字。

如果待匹配的字符串表达式中本身就含有通配符%或_，这时就要使用 ESCAPE <转义字符>，对通配符进行转义。

例 3.30 找出课程名以“C_”开头的课程的课程号、课程名和学分。

```
SELECT Cno, Cname, Credit
FROM C
WHERE Cname LIKE 'C\_%' ESCAPE '\'
```

说明：由于在通配符_前有转义字符\，所以该_被转义为普通的_字符，而它后面的%仍然作为通配符。

例 3.31 找出专业还没有确定的学生的详细信息。

```
SELECT *
FROM S
WHERE Major IS NULL
```

说明：专业还没有确定就是 Major 的值为空值，与空值比较不能写成 Major=NULL，而应用专用的 IS NULL 运算符。条件“专业已经确定”应写成 Major IS NOT NULL。

SQL92 规定对空值 NULL 进行运算时使用下列规则：

(1) NULL 值和其他任何值(包括另一个 NULL 值)进行算术运算(+、-、×、÷)时，其结果为 NULL；

(2) NULL 值和其他任何值(包括另一个 NULL 值)进行比较运算(>、>=、<、<=)时，其结果为 UNKNOWN。

在 SQL Server 2014 中，如果一个元组对应它的元组筛选条件计算出的结果值为 UNKNOWN，那么该元组不会出现在结果集中。

3. ORDER BY 子句

ORDER BY 子句是可选项，它表示对查询结果按照一个或多个属性值的升序(ASC)或降序(DESC)排列，如果缺省，默认就是 ASC。

例 3.32 找出所有课程的课程号、课程名和学分，查询结果按学分降序排列。

```
SELECT Cno, Cname, Credit
FROM C
ORDER BY Credit DESC
```

说明：对于 Credit 值为空值的元组，排序时的次序由具体系统实现来决定，含有空值的元组可以排在最前，也可以排在最后。各个系统的实现可以不同，但只要保持一致就行。

例 3.33　找出每个学生的姓名、专业和出生年份，查询结果按专业升序排列，专业相同按出生年份降序排列。

```
SELECT Sname, Major, 2018 - Sage
FROM S
ORDER BY Major, 3 DESC
```

说明：由于出生年份是通过计算表达式 2018-Sage 得到，又没有给出生年份列命名，这时可以用该列的序号“3”(即第 3 列)来代表出生年份。

4. 聚集函数和 GROUP BY 子句

在 2.5.3 节中已经介绍过聚集运算，这里介绍的是聚集运算的 SQL 实现。如果没有 GROUP BY 子句相当于不分组，查询结果表肯定只有一个元组(即一行)。如果有 GROUP BY 子句，那么每个小组在查询结果表中都会产生一个元组。SQL 语言中常用的聚集函数如表 3.4 所示，显然 SUM 和 AVG 函数中的列必须是数值型。如果指定 DISTINCT 关键字，则表示在计算时要取消指定列中的重复值。如果指定 ALL 关键字(如果缺省，默认就是 ALL)，则表示不取消指定列中的重复值。

表 3.4　SQL 语言中常用的聚集函数

聚 集 函 数	功　　能
COUNT([DISTINCT|ALL] *)	统计元组的个数
COUNT([DISTINCT|ALL]列名)	统计一列中值的个数
SUM([DISTINCT|ALL]列名)	计算一列中值的总和
AVG([DISTINCT|ALL]列名)	计算一列中值的平均值
MAX([DISTINCT|ALL]列名)	求一列中值的最大值
MIN([DISTINCT|ALL]列名)	求一列中值的最小值

例 3.34　找出教师的总人数。

```
SELECT COUNT( * )
FROM T
```

例 3.35　找出选修了 1002 号课程的学生人数。

```
SELECT COUNT(DISTINCT Sno)
FROM SC
WHERE Cno = '1002'
```

说明：学生每选修一次 1002 号课程，在 SC 表中就对应一个元组。一个学生(如 16001 号学生)有可能因为不及格而多次选修 1002 号课程，为避免重复计算学生人数，必须在 COUNT 函数中使用 DISTINCT 关键字。

例 3.36　找出 2017 年秋 1001 号课程成绩的最高分、最低分、平均分以及选修人数。

```
SELECT MAX(Score), MIN(Score), AVG(Score), COUNT(Sno)
FROM SC
WHERE Sdate = '2017 秋' AND Cno = '1001'
```

需要说明的是：①在聚集函数遇到空值时，除 COUNT(*)外，都会忽略空值而只处理非空值；②WHERE 子句的元组筛选条件中不能出现聚集函数。

例 3.37 找出 2017 年秋各门课程成绩的最高分、最低分、平均分以及选修人数。

```
SELECT Cno, MAX(Score), MIN(Score), AVG(Score), COUNT(Sno)
FROM SC
WHERE Sdate = '2017 秋'
GROUP BY Cno
```

说明：在求解时先把不是 2017 秋的选课记录丢掉，再根据课程号分组，课程号的值相同的为一组，然后对每一组分别求最高分、最低分、平均分以及选修人数。

例 3.38 找出各个学期各门课程成绩的最高分、最低分、平均分以及选修人数，查询结果按选修日期升序排列，选修日期相同则按课程号升序排列。

```
SELECT Sdate, Cno, MAX(Score), MIN(Score), AVG(Score), COUNT(Sno)
FROM SC
GROUP BY Sdate, Cno
ORDER BY Sdate, Cno
```

说明：可以根据多个属性分组，本例中按选修日期和课程号分组，选修日期的值和课程号的值均相等的为一组。也可以理解为：先按 Sdate 属性分组，对每一小组再按 Cno 进一步分组。

需要说明的是：当 SQL 查询使用分组时，很重要的一点是要保证出现在 SELECT 子句中但没有被聚集的属性必须出现在 GROUP BY 子句中。也就是说，任何没有出现在 GROUP BY 子句中的属性如果出现在 SELECT 子句中的话，它只能出现在聚集函数内部，否则这样的查询是错误的。例如，下面的查询就是错误的，因为 Tno 没有出现在 GROUP BY 子句中，但它出现在 SELECT 子句中，而且没有被聚集。

```
SELECT Title, Tno, AVG(Salary)
FROM T
GROUP BY Title
```

5. HAVING 子句

有时，查询者可能并不对所有小组的统计结果感兴趣。例如，需要找出从未有过考试成绩不及格的学生。该条件并不针对选修表 SC 中的单个元组，而是针对 GROUP BY 分组后的小组。为表达这样的查询，需要 HAVING 子句来给出小组筛选条件。由于小组筛选条件是作用于一个组的，因此小组筛选条件中常会使用聚集函数。

例 3.39 找出从未有过考试成绩不及格的学生的学号。

```
SELECT Sno
FROM SC
GROUP BY Sno
HAVING MIN(Score)>=60
```

说明：①与 SELECT 子句类似，出现在 HAVING 子句中但没有被聚集的属性只能是出现在 GROUP BY 子句中的那些属性；②HAVING 子句只能出现在 GROUP BY 子句的后面，在没有 GROUP BY 子句的情况下，出现 HAVING 子句是没有意义的。因为

HAVING 子句中的条件是小组筛选条件，只有用 GROUP BY 子句分组后，才能用小组筛选条件来选择小组，满足小组筛选条件的组才能被选出来。

如果在同一个查询语句中同时出现 WHERE 子句与 HAVING 子句时，它们的区别在于：WHERE 子句作用于元组，从中选择满足条件的元组；而 HAVING 子句作用于小组，从中选择满足条件的小组。

例 3.40　找出至少有过两次考试成绩不及格的学生的学号。

```
SELECT Sno
FROM SC
WHERE Score < 60
GROUP BY Sno
HAVING COUNT( * )> = 2
```

说明：查询时先执行 WHERE 子句找出考试成绩不及格的元组，再按学号分组，然后把满足 HAVING 子句中小组筛选条件的学生的学号输出。

3.4.2　连接查询

前面的查询都是针对一个表进行的。关系数据库中各个表之间是有联系的，凡查询条件或结果涉及多个表时，就需要将多个表连接起来，形成一个包含条件和结果中涉及的全部数据的临时表，再对该临时表用上面单表查询的方法进行查询。

连接两个表一定要有连接条件，由于外码起到联系两个表的作用，因此，连接条件中一般都有外码与被参照表的主码。当外码与被参照表的主码同名时，必须加上表名前缀，方法是：表名.列名。事实上，当参与连接的两个表存在同名列时，任何子句引用同名列，都必须加表名前缀，否则会引起“列名不明确”错误。

连接查询就是关系代数中连接运算的 SQL 实现，也是关系数据库中最主要的查询，包括内连接、自连接和外连接等。

1. 内连接

为了与 2.5.3 节中介绍的外连接相区别，常把 2.5.2 节中介绍的普通的连接称为内连接。内连接运算中最常用的是等值连接和自然连接。在 SQL 语言的早期标准中没有连接运算符，也不区分等值连接和自然连接，连接条件写在 WHERE 子句中。

例 3.41　找出学生信息以及他(她)选修课程的信息。

```
SELECT *
FROM S, SC
WHERE S.Sno = SC.Sno
```

说明：本例实现的是 S 和 SC 表之间的等值连接，连接条件是 S.Sno = SC.Sno，SELECT 后面的 * 代表两张表中的所有列。如果把 SELECT 子句改写为 SELECT S.*，Cno，Sdate，Score，那么实现的就是 S 和 SC 表之间的自然连接。执行该连接操作的一种可能过程是：首先在表 S 中找到第一个元组，然后从头开始扫描表 SC，逐一查找满足连接条件的表 SC 中的元组，找到后就将表 S 中的第一个元组与该元组拼接起来，形成结果表中一个元组。表 SC 全部查找完后，再找表 S 中第二个元组，然后再从头开始扫描表 SC，逐一查找满足连接条件的元组，找到后就将表 S 中的第二个元组与该元组拼接起来，形成结果表中

一个元组。重复上述操作,直到表 S 中的全部元组都处理完毕。

但 SQL 的实际实现中一般不会按照这种形式执行连接操作,RDBMS 往往会对 SQL 查询的执行进行优化,如利用索引加快连接操作,详细内容可参考相关文献。

在 SQL92 标准中,引进了 JOIN 运算符,例 3.41 中的查询可改写为:

```
SELECT *
FROM S INNER JOIN SC ON S.Sno = SC.Sno
```

其中关键字 INNER 可以省略不写。表面上看 ON 子句完全可以被 WHERE 子句所代替,似乎是一个冗余的 SQL 特征。但引入 ON 子句有两个优点:①对于马上要介绍的外连接来说,ON 子句的表现与 WHERE 子句是不同的;②用 ON 子句指定连接条件,用 WHERE 子句指定其余的查询条件,这样的 SQL 查询更容易让人理解。

例 3.42 找出选修了"数据库原理"课程的学生的学号、姓名、选修日期和成绩。

```
SELECT S.Sno, Sname, Sdate, Score
FROM S JOIN SC ON S.Sno = SC.Sno JOIN C ON SC.Cno = C.Cno
WHERE Cname = '数据库原理'
```

本例说明了在 FROM 子句中如何实现三张表的连接操作。

2. 自连接

连接操作可以在两张不同的表之间进行,也可以是一张表与自己进行连接,后一种连接称为自连接。

例 3.43 找出至少选修过 1001 号课程和 1002 号课程的学生的学号。

```
SELECT DISTINCT A.Sno
FROM SC AS A JOIN SC AS B ON A.Sno = B.Sno
WHERE A.Cno = '1001' AND B.Cno = '1002'
```

说明:由于是一张表与自己进行连接,为了区分可以用"AS 别名"的形式给表起一个别名,其中关键字 AS 可以省略不写。

3. 外连接

内连接操作中,只有满足连接条件的元组才能作为结果输出。如例 3.41 中由于 16005 号学生没有选修课程,在 SC 表中没有相应的元组,造成该生的信息在连接操作时被丢弃了。为了在查询结果关系中保留该生的信息,就需要使用外连接。

例 3.44 用左外连接改写例 3.41 中的查询要求,保留没有选修课程的学生信息。

```
SELECT *
FROM S LEFT OUTER JOIN SC ON S.Sno = SC.Sno
```

说明:关键字 OUTER 可以省略不写。右外连接和全外连接的运算符分别为 RIGHT OUTER JOIN 和 FULL OUTER JOIN。本例也充分说明了 ON 子句不能用 WHERE 子句来代替,因为 ON 子句是表达外连接运算的一部分,而 WHERE 子句不是。

下面介绍连接的类型与条件

自连接本质上不是一种新的连接类型,只不过连接的对象是自身,所以连接分为内连接和外连接两种。SQL 语言的早期标准中把连接条件写在 WHERE 子句中,而 SQL 新标准中连接条件可以有三种表达方法,除了前面介绍的 ON 子句外,还可以通过关键字 NATURAL

或者 USING 子句来表达。

ON 子句表达的能力强且灵活(因为 ON 后面的是任意一个合法的逻辑表达式)；而 NATURAL 或者 USING 子句只能对被连接的两表中的同名列做“=”比较，但书写比较简洁、方便。注意，SQL Server 2014 不支持用 NATURAL 或者 USING 子句表达连接条件，而 MySQL 5.5 则支持。

NATURAL 与 USING 子句的区别在于：NATURAL 是对被连接的两表中所有的同名列做“=”比较；而 USING 子句只对被连接的两表中指定的同名列做“=”比较。例如，假定有表 R(A,B,C)和表 S(B,C,D)，则 R NATURAL JOIN S 等价于 R JOIN S ON R.B=S.B AND R.C=S.C；而 R JOIN S USING(B)等价于 R JOIN S ON R.B=S.B。当然，语法上 USING 后的圆括号中可以有两个或两个以上的列。

例 3.45　用 NATURAL 或者 USING 子句重做例 3.41 中的查询要求。

```
SELECT *
FROM S NATURAL JOIN SC
```

或者

```
SELECT *
FROM S JOIN SC USING(Sno)
```

说明：因为做的是自然连接，所以本例的结果表中没有重复的列 Sno，而例 3.41 做的是等值连接，所以结果表中有重复的列 Sno。

最后要说明的是，如果 SELECT 语句中缺省连接条件，那么本质上做的就是关系代数中的广义笛卡儿积。例如，SELECT * FROM R,S 实现的就是关系 R 和 S 的广义笛卡儿积。

3.4.3　嵌套查询

在 SQL 语言中，一个 SELECT—FROM—WHERE 语句称为一个查询块，将一个查询块嵌套在另一个查询块的 WHERE 子句或者其他子句中的查询称为嵌套查询，这也正是“结构化”的含义所在。嵌套查询使用户可以用多个简单查询构造复杂的查询，从而增强 SQL 语言的查询能力。

1. 带有 IN 谓词的子查询

在嵌套查询中，子查询的结果往往是元组的集合，所以谓词 IN 是嵌套查询中最常用的谓词。

例 3.46　找出选修了 1001 号课程的学生的姓名和性别。

```
SELECT Sname, Ssex
FROM S
WHERE Sno IN
          (SELECT Sno
           FROM SC
           WHERE Cno = '1001')
```

说明：本例中，查询块 SELECT Sname,Ssex FROM S WHERE Sno 称为外层查询或

父查询，查询块 SELECT Sno FROM SC WHERE Cno＝'1001'称为内层查询或子查询。SQL 语言允许多层嵌套查询，即一个子查询中还可以嵌套其他子查询。需要指出的是：子查询中不能使用 ORDER BY 子句，ORDER BY 子句只能对最终查询结果排序。

嵌套查询的执行过程是由里向外进行，即每个子查询在上一级查询处理之前求解，子查询的结果用于建立父查询的查找条件。本例中子查询的结果有三个元组，即 16001、16004 和 16007。将子查询的结果代入外层查询的查询条件中，父查询变为：

```
SELECT Sname, Ssex
FROM S
WHERE Sno IN ( '16001', '16004', '16007' )
```

上述查询也可用下列连接查询来实现，当然不同方法的执行效率可能会有差别，甚至相差很大，这取决于实际 RDBMS 的优化算法。

```
SELECT Sname, Ssex
FROM S JOIN SC ON S. Sno = SC. Sno
WHERE Cno = '1001'
```

例 3.47 找出选修了“数据库原理”课程的学生的姓名和性别。

```
SELECT Sname, Ssex
FROM S
WHERE Sno IN
            ( SELECT Sno
              FROM SC
              WHERE Cno IN
                           ( SELECT Cno
                             FROM C
                             WHERE Cname = '数据库原理' ) )
```

说明：本例的子查询中还嵌套了一层子查询，执行时由里向外进行，先对 C 表查询，再对 SC 表查询，最后对 S 表查询。另外，该查询也可用连接查询来实现。

```
SELECT Sname, Ssex
FROM S JOIN SC ON S. Sno = SC. Sno JOIN C ON SC. Cno = C. Cno
WHERE Cname = '数据库原理'
```

到此，也许读者会认为，嵌套查询与连接查询是可以相互转换的，但事实并不是这样。首先，当查询结果表中的列来自两张或两张以上的表时，就必须要用连接查询来实现，而不能用嵌套查询来实现，例 3.42 就是这样。其次，有时嵌套查询也不一定能用连接查询来实现，例 3.58 就是这样。再次，有时嵌套查询比连接查询更容易实现，例 3.48 和例 3.49 就是这样。

例 3.48 用嵌套查询重做例 3.43 中的查询要求。

```
SELECT DISTINCT Sno
FROM SC
WHERE Cno = '1001' AND Sno IN
                           ( SELECT Sno
                             FROM SC
                             WHERE Cno = '1002' )
```

说明：在选修了 1002 号课程的学生中再选择又选修了 1001 号课程的学生，也就是至少选修过 1001 号课程和 1002 号课程的学生。

例 3.49　找出没有选修过 1001 号课程的学生的姓名和性别。

```
SELECT Sname, Ssex
FROM S
WHERE Sno NOT IN
                ( SELECT Sno
                  FROM SC
                  WHERE Cno = '1001' )
```

2. 带有比较运算符的子查询

当能肯定子查询返回的是一个标量（单值）时，就可用比较运算符（>、>=、<、<=、=、!=或<>）把父查询与子查询连接起来，这就是带有比较运算符的子查询。

例 3.50　找出与“张文杰”在同一个专业学习的学生的姓名、性别和年龄。

```
SELECT Sname, Ssex, Sage
FROM S
WHERE Major =
             ( SELECT Major
               FROM S
               WHERE Sname = '张文杰' )
```

说明：这里假定姓名为“张文杰”的学生只有一个，如果不能肯定，那么保险起见，还是用 IN 代替=。另外，本例也可以用自连接实现，作为一个练习留给读者自己完成。

例 3.51　找出每个教师的工资低于全体教师的平均工资的姓名、年龄和职称。

```
SELECT Tname, Tage, Title
FROM T
WHERE Salary <
             ( SELECT AVG(Salary)
               FROM T )
```

例 3.52　找出每个学生选修某课程的平均成绩低于全体学生选修该课程的平均成绩的学号和课程号。

```
SELECT Sno, Cno
FROM SC A
GROUP BY Sno, Cno
HAVING AVG(Score) <
                  ( SELECT AVG(Score)
                    FROM SC B
                    WHERE B.Cno = A.Cno )
```

说明：前面的举例中，子查询中的查询条件不依赖于父查询，这类子查询称为不相关子查询。反之，如果子查询中的查询条件依赖于父查询，这类子查询称为相关子查询（Correlated Subquery）。本例中子查询是求某门课程的平均成绩，至于是哪一门课程的平均成绩取决于 A.Cno 的值，而该值是与父查询相关的，因此本例中的查询为相关子查询。

相关子查询的执行过程与不相关子查询大不相同，其类似于 C 语言中 for 语句嵌套的

执行过程。本例中父查询的 A.Cno 取一个值(一个小组),做一次子查询,求出该门课程的平均成绩返回父查询,决定 Sno 和 Cno 是否放入结果表;然后父查询的 A.Cno 取下一个值(下一个小组),再做一次子查询;直到父查询中的小组全部处理完毕。

3. 带有 SOME 或 ALL 谓词的子查询

当子查询返回的是一个集合(多值)时,就不能直接用比较运算符,而必须用 SOME(早期为 ANY,因为容易误解而被改为 SOME)或 ALL 谓词修饰。使用 SOME 或 ALL 谓词时必须同时使用比较运算符,SOME 表示一组值中的某些值,ALL 表示一组值中的全部值。例如,"> SOME"表示大于子查询结果中的某些值,而"> ALL"表示大于子查询结果中的所有值,其余的比较运算符加 SOME 或 ALL 谓词的理解类似。显然"=ALL"和"!=SOME"没有意义。

例 3.53 找出其他专业中比通信专业某些学生年龄小的学生姓名和年龄。

```
SELECT Sname, Sage
FROM S
WHERE Major!= '通信' AND Sage < SOME
                                ( SELECT Sage
                                  FROM S
                                  WHERE Major = '通信' )
```

说明:执行此查询时,首先处理子查询,找出通信专业中所有学生的年龄,构成一个集合(20,21),然后处理父查询,找所有不是通信专业且年龄小于 20 岁或 21 岁的学生。

本例也可以用聚集函数来实现。首先用子查询找出通信专业中年龄的最大值(21),然后在父查询中找所有不是通信专业且年龄小于 21 岁的学生。SQL 语句如下:

```
SELECT Sname, Sage
FROM S
WHERE Major!= '通信' AND Sage <
                             ( SELECT MAX(Sage)
                               FROM S
                               WHERE Major = '通信' )
```

例 3.54 找出其他专业中比通信专业所有学生年龄都小的学生姓名和年龄。

```
SELECT Sname, Sage
FROM S
WHERE Major!= '通信' AND Sage < ALL
                                ( SELECT Sage
                                  FROM S
                                  WHERE Major = '通信' )
```

显然,本例也可以用聚集函数来实现。SQL 语句如下:

```
SELECT Sname, Sage
FROM S
WHERE Major!= '通信' AND Sage <
                             ( SELECT MIN(Sage)
                               FROM S
                               WHERE Major = '通信' )
```

从上述两例可以看出，SOME 和 ALL 谓词可以改用聚集函数来实现，表 3.5 给出了 SOME 和 ALL 谓词与聚集函数和 IN 运算的等价转换关系。

表 3.5　SOME 和 ALL 谓词与聚集函数和 IN 运算的等价转换关系

	=	!=或<>	<	<=	>	>=
SOME	IN	—	<MAX	<=MAX	>MIN	>=MIN
ALL	—	NOT IN	<MIN	<=MIN	>MAX	>=MAX

表 3.5 中，=SOME 等价于 IN，!=ALL 等价于 NOT IN，<SOME 等价于<MAX，<ALL 等价于<MIN 等。

有了上述对照表，可能有人会认为没有必要使用带 SOME 或 ALL 谓词的子查询，这种想法是不对的。

例 3.55　找出所有考试的平均成绩最高的学生的学号。

```
SELECT Sno
FROM SC
GROUP BY Sno
HAVING AVG(Score) >= ALL
                        ( SELECT AVG(Score)
                          FROM SC
                          GROUP BY Sno )
```

说明：由于 SQL 语言不允许聚集函数嵌套使用，所以子查询不能写成 SELECT MAX (AVG(Score)) FROM SC GROUP BY Sno。

4. 带有 EXISTS 谓词的子查询

EXISTS 谓词代表存在量词∃。带有 EXISTS 谓词的子查询不返回任何数据，只产生逻辑真值 true 或逻辑假值 false。若子查询结果非空，则外层的 WHERE 子句返回真值；若子查询结果为空，则外层的 WHERE 子句返回假值。

由 EXISTS 引出的子查询，其目标列表达式通常都用 *，因为带 EXISTS 的子查询只返回真值或假值，给出列名无实际意义。

例 3.56　用 EXISTS 谓词重做例 3.46 中的查询要求。

```
SELECT Sname, Ssex
FROM S
WHERE EXISTS
             ( SELECT *
               FROM SC
               WHERE Sno = S.Sno AND Cno = '1001' )
```

说明：带 EXISTS 谓词的查询一般都是相关子查询。本例中由于子查询的查询条件中用到了父查询的 S.Sno，所以为相关子查询，其执行过程是：首先父查询取 S 表的第一个元组，根据它的 Sno 执行一次子查询，如果该生选修了 1001 号课程，那么子查询的结果非空，父查询的查询条件为真，该生的 Sname 和 Ssex 放入结果表；然后父查询再取 S 表的第二个元组，再执行一次子查询；直到 S 表中的元组全部处理完毕。

带有 NOT EXISTS 谓词的子查询也不返回任何数据，只产生逻辑真值 true 或逻辑假

值 false。若子查询结果为空,则外层的 WHERE 子句返回真值,否则返回假值。

例 3.57 用 NOT EXISTS 谓词重做例 3.49 中的查询要求。

```
SELECT Sname, Ssex
FROM S
WHERE NOT EXISTS
              ( SELECT *
               FROM SC
               WHERE Sno = S.Sno AND Cno = '1001' )
```

带有 NOT EXISTS 谓词的子查询常用于模拟集合“包含”操作,即用来实现关系代数中的除法运算。

例 3.58 找出至少选修了 16001 号学生选修的全部课程的学生的学号。

```
SELECT Sno
FROM S
WHERE Sno != '16001' AND
      NOT EXISTS
             ( SELECT *
              FROM SC A
              WHERE A.Sno = '16001' AND
                    NOT EXISTS
                           ( SELECT *
                            FROM SC B
                            WHERE B.Sno = S.Sno AND
                                  B.Cno = A.Cno ) )
```

需要特别说明的是:一些带 EXISTS 或 NOT EXISTS 谓词的子查询不能被其他形式的子查询等价替换,但所有带 IN 谓词、比较运算符、SOME 和 ALL 谓词的子查询都能用带 EXISTS 谓词的子查询等价替换。

5. FROM 子句中的子查询

前面的子查询都出现在父查询的 WHERE 子句或 HAVING 子句中作为父查询的查询条件中的一部分。SQL 语言允许在 FROM 子句中使用子查询,因为任何子查询的查询结果都是一个表,它可以出现在 FROM 后表可以出现的位置上。

例 3.59 找出所有考试的平均成绩低于 75 分的学生的学号。

```
SELECT Sno
FROM ( SELECT Sno, AVG(Score) AS AvgScore
      FROM SC
      GROUP BY Sno ) AS Temp
WHERE AvgScore < 75
```

说明:SQL 标准并没有要求给子查询的结果表命名,但 SQL Server 2014 要求给子查询的结果表命名,即使该名字从未被使用。

例 3.60 找出至少还有两门以上课程考试成绩不及格的学生的姓名、性别和专业。

```
SELECT Sname, Ssex, Major
FROM S, ( SELECT Sno, Cno
         FROM SC
```

```
        GROUP BY Sno, Cno
        HAVING MAX(Score)< 60 ) AS Fail
WHERE S.Sno = Fail.Sno
GROUP BY S.Sno, Sname, Ssex, Major
HAVING COUNT( * )> 2
```

说明：对于每个学生 Sno 是唯一的，之所以 GROUP BY 中出现 Sname、Ssex、Major，是为了属性 Sname、Ssex、Major 能够出现在 SELECT 子句中。

6. WITH 子句

WITH 子句是 SQL99 中引入的，SQL Server 2014 支持该子句。WITH 子句提供了定义临时表的方法，这个定义只对随后紧跟的 SELECT 语句有效。

例 3.61　用 WITH 子句重做例 3.60 中的查询要求。

```
WITH Fail AS
     (SELECT Sno, Cno
      FROM SC
      GROUP BY Sno, Cno
      HAVING MAX(Score)< 60 )
SELECT Sname, Ssex, Major
FROM S JOIN Fail ON S.Sno = Fail.Sno
GROUP BY S.Sno, Sname, Ssex, Major
HAVING COUNT( * )> 2
```

说明：尽管用 FROM 子句中的子查询也能完成查询，但 WITH 子句使得查询在逻辑上更加清晰。

3.4.4　集合查询

SELECT 语句的查询结果是元组的集合，所以多个 SELECT 语句的查询结果可以进行集合运算。集合运算主要包括并运算 UNION、交运算 INTERSECT 和差运算 EXCEPT。注意：参加集合运算的各查询结果的列数必须相同，对应列的数据类型也必须相同。

例 3.62　找出选修了 1001 号课程或者 1002 号课程的学生的学号。

```
SELECT Sno
FROM SC
WHERE Cno = '1001'
UNION
SELECT Sno
FROM SC
WHERE Cno = '1002'
```

说明：UNION 将多个查询结果合并起来时，系统会自动去掉重复元组。如果要保留重复元组，则应用 UNION ALL 运算符。

例 3.63　用 INTERSECT 运算符重做例 3.43 中的查询要求。

```
SELECT Sno
FROM SC
WHERE Cno = '1001'
INTERSECT
```

```
SELECT Sno
FROM SC
WHERE Cno = '1002'
```

说明：有的 RDBMS 不支持 INTERSECT 运算符，这时就像例 3.48 那样可以用 IN 谓词的子查询来实现集合的交运算。

例 3.64 找出选修了 1002 号课程，但没有选修 1001 号课程的学生的学号。

```
SELECT Sno
FROM SC
WHERE Cno = '1002'
EXCEPT
SELECT Sno
FROM SC
WHERE Cno = '1001'
```

说明：有的 RDBMS 不支持 EXCEPT 运算符，这时可以用 NOT IN 谓词的子查询来实现集合的差运算。

例 3.65 用 EXCEPT 运算符重做例 3.58 中的查询要求。

```
SELECT Sno
FROM S
WHERE Sno != '16001' AND
      NOT EXISTS
            ( ( SELECT DISTINCT Cno
                FROM SC
                WHERE Sno = '16001' )
              EXCEPT
                ( SELECT DISTINCT Cno
                FROM SC
                WHERE SC.Sno = S.Sno ) )
```

说明："集合 A 包含集合 B"可以写成"NOT EXISTS (B EXCEPT A)"。显然，本例中的实现方法比例 3.58 中的更清晰、更易理解。

3.5 数 据 更 新

数据更新是指数据的插入、删除和修改操作，SQL 中的数据更新语句有 INSERT 语句、DELETE 语句和 UPDATE 语句。

3.5.1 插入数据

SQL 中的数据插入语句是 INSERT 语句。它有两种基本方式：一种是一次插入一个元组，另一种是一次插入一个查询结果表。

插入一个元组的 INSERT 语句的一般格式如下：

```
INSERT
INTO <表名> [ (<属性列 1>[, <属性列 2> [, … ] ] ) ]
VALUES (<常量 1>[, <常量 2> [, … ] ] )
```

例 3.66　将一门课程（3002，ASP.NET 程序设计，选修，54，3）插入到 C 表中。

```
INSERT
INTO C
VALUES ('3002', 'ASP.NET 程序设计', '选修', 54, 3)
```

说明：当一个表中的所有属性列都有值插入时，表后面的属性列名可省略不写。这时，VALUES 子句中的值必须与表定义时属性列的个数相等，顺序和类型一致。

例 3.67　将一个学生（16008，陈亮，男，20）插入到 S 表中。

```
INSERT
INTO S(Sno, Sname, Ssex, Sage)
VALUES ('16008', '陈亮', '男', 20)
```

说明：当一个表中的部分属性列有值插入时，必须在表名后明确指出有值属性的列名，在 INTO 子句中没有出现的属性列上的取值视其默认值和空值约束等情况而定。对于列有默认值约束的，列值为默认值；对于列没有默认值约束但允许空值的，列值为空值；对于列既没有默认值约束也不允许空值的，则导致插入操作失败。

插入子查询结果的 INSERT 语句的一般格式如下：

```
INSERT
INTO <表名> [ (<属性列 1>[, <属性列 2> [, …] ] ) ]
    子查询
```

使用该格式的 INSERT 语句，可以将一个查询结果表中的数据成批插入指定的表中。

例 3.68　找出每个学生所有已考课程的平均成绩，并将结果存入数据库。

首先创建一个存放结果的基本表：

```
CREATE TABLE SAvgScore
(   Sno       char(8),
    AvgScore decimal(5,2)
)
```

再将查询结果插入到新表中：

```
INSERT
INTO SAvgScore
    SELECT Sno, AVG(Score)
    FROM SC
    WHERE Score IS NOT NULL
    GROUP BY Sno
```

说明：RDBMS 在执行插入语句时会检查所插元组是否破坏表上已定义的完整性规则。

3.5.2　修改数据

对存放在表中的数据进行修改也是数据库日常维护的一项重要工作。SQL 中的 UPDATE 语句可以修改表中的一个或多个元组，一列或多列数据。UPDATE 语句修改表中现有的数据有两种方式：一种是通过直接赋值进行修改，另一种是使用子查询将要取代列中原有值的数据先查询出来，再修改原有列值。

UPDATE 语句的一般格式如下：

```
UPDATE <表名>
SET <列名 1> = <表达式 1>[, <列名 2> = <表达式 2> [, … ] ]
[WHERE <条件>]
```

如果省略 WHERE 子句，表示表中的所有元组都按 SET 子句中的要求进行修改，否则仅修改满足条件的部分元组。<列名 1>=<表达式 1>的含义是用<表达式 1>的值取代<列名 1>中原有的值，表达式可以是常量、变量、NULL 或返回单个值的子查询。

例 3.69 将 16008 号学生的专业改为通信，联系电话改为 13267812345。

```
UPDATE S
SET Major = '通信', Mphone = '13267812345'
WHERE Sno = '16008'
```

例 3.70 将工资超过 5000 元的教师加 6%工资，其余教师加 8%工资。

```
UPDATE T
SET Salary = Salary * 1.06
WHERE Salary > 5000
UPDATE T
SET Salary = Salary * 1.08
WHERE Salary < = 5000
```

说明：这两条 UPDATE 语句的顺序十分重要。如果颠倒这两条语句的顺序，那么工资略低于 5000 元的教师将先加了 8%的工资，再加 6%的工资。

SQL 语言提供了 CASE 表达式，利用它可以用一条 UPDATE 语句实现例 3.70 中的加薪要求，避免语句次序引发的问题。

```
UPDATE T
SET Salary = CASE
                WHEN Salary > 5000 THEN Salary * 1.06
                ELSE Salary * 1.08
             END
```

例 3.71 假定教师加薪的幅度与他授课的评价有关，每个教师工资增加的百分比就是他所有授课评价的平均值。

```
UPDATE T
SET Salary = Salary * (1 + 0.01 * ( SELECT AVG(Remark)
                                    FROM TC
                                    WHERE TC.Tno = T.Tno ) )
```

说明：表达式中不但用到了子查询，而且子查询用到了被修改表 T 中的有关属性。

例 3.72 将 2016 秋 C 程序设计课程的考试成绩每人加 5 分。

```
UPDATE SC
SET Score = Score + 5
WHERE Sdate = '2016 秋' AND Cno =
                                  ( SELECT Cno
                                    FROM C
                                    WHERE Cname = 'C 程序设计' )
```

说明：这里 UPDATE 语句的 WHERE 子句中用到了子查询。事实上，UPDATE 语句的 WHERE 子句可以使用 SELECT 语句的 WHERE 子句中的任何合法结构。

3.5.3　删除数据

SQL 中的 DELETE 语句可以删除表中的一个或多个元组，甚至可以删除表中的所有元组，但表的定义仍然在系统的数据字典中。DELETE 语句的一般格式如下：

```
DELETE
FROM <表名>
[WHERE <条件>]
```

如果省略 WHERE 子句，表示删除表中的所有元组，否则仅删除满足条件的部分元组。这里的 WHERE 子句可以使用 SELECT 语句的 WHERE 子句中的任何合法结构。

例 3.73　删除 S 表中的 16008 号学生。

```
DELETE
FROM S
WHERE Sno = '16008'
```

例 3.74　删除所有 2016 秋 C 程序设计课程的选课记录。

```
DELETE
FROM SC
WHERE Sdate = '2016 秋' AND Cno  =
                                    ( SELECT Cno
                                     FROM C
                                     WHERE Cname = 'C 程序设计' )
```

最后要说明的是：由于更新数据时一次只对一个表进行操作，这会带来一些问题。例如，在将 S 表中学号为 16001 的学生删除后，SC 表中原来 16001 号学生的选课记录也必须同时删除，否则就会出现不一致性问题，即数据的参照完整性受到破坏。只有将相应的选课记录也同时删除，数据库才重新处于一致状态。再如，在将 S 表中学号为 16001 学生的学号修改为 17001 之后，如果未对 SC 表中原来 16001 学生的选课记录进行修改，则也将违反参照完整性。插入数据时也会产生数据的不一致性问题。

由于关系模型中的实体完整性与参照完整性是必须满足的完整性约束条件，关系数据库系统一般都自动支持。对于违反实体完整性的操作，系统一般都采用拒绝执行该操作的策略。而对于违反参照完整性的操作，各种 RDBMS 产品都提供了不同的实现策略。例如，对于上面提到的两个例子，现在大部分的产品都可进行级联删除和级联修改，来保证数据库中数据的一致性。

习　题　3

一、单项选择题

1. 在 SQL 语言中，属于“模式 DDL”语言的是(　　)语句。

A. SELECT　　B. UPDATE

C. INSERT　　D. CREATE TABLE

2. 在 SQL Server 系统中，数据库中的(　　)不属于任何文件组。

A. 日志文件　　B. 主数据文件　　C. 辅助数据文件　　D. 索引数据文件

3. 在 SQL Server 系统中，一个数据库可以有(　　)个辅助文件组。

A. 0　　B. 1　　C. 0 到多　　D. 1 到多

4. 在 SQL Server 系统中，下列关于删除文件组的叙述中，错误的是(　　)。

A. 文件组可以用 T-SQL 语句删除

B. 删除文件组前必须先删除其中的数据文件

C. 只能删除辅助文件组，不能删除主文件组

D. 文件组类似于文件夹，因而可有可无，删除文件组不影响数据库的使用

5. 在 T-SQL 语言中，用于存储定长非 Unicode 字符串且最大长度不超过 8000 个字符的是(　　)。

A. nchar　　B. char　　C. nvarchar　　D. varchar

6. 在 T-SQL 语言中，用于存储实数且是精确数值而不是近似数值的是(　　)。

A. decimal　　B. real　　C. float　　D. float(n)

7. 在 T-SQL 语言中，用于存储图像、声音等数据且最大长度不超过 8000 字节的是(　　)。

A. char　　B. varchar　　C. text　　D. varbinary

8. 在 T-SQL 语言中，用 ALTER TABLE 语句修改表结构时不可以(　　)。

A. 修改表名　　B. 增加列　　C. 删除列　　D. 增加列级约束

9. 在 T-SQL 语言中，ALTER TABLE 语句的 ALTER COLUMN 子句能够实现(　　)功能。

A. 修改列名　　B. 增加列

C. 删除列　　D. 改变列的数据类型

10. 在 SQL 语言中，创建基本表时是通过(　　)短语实现实体完整性规则的。

A. CHECK　　B. PRIMARY KEY

C. FOREIGN KEY　　D. NOT NULL

11. 为了防止向“学生”表中插入数据时姓名出现空值，建表时应使用(　　)来进行约束。

A. UNIQUE　　B. NOT NULL

C. PRIMARY KEY　　D. CHECK

12. 在 SQL 语言中，可用语句(　　)删除一个基本表。

A. DELETE　　B. CLEAR　　C. DROP　　D. REMOVE

13. 在 SQL 语言中，下列选项中关于 UPDATE 语句正确的是(　　)。

A. 只能修改表中的一条记录　　B. 可以修改表中的多条记录

C. 不能修改表中的全部记录　　D. 可以修改表的结构

14. 在 SQL 语言的 SELECT 语句中，对应关系代数中“选择”运算的是(　　)子句。

A. WHERE　　B. FROM　　C. SELECT　　D. GROUP BY

15. 当 WHERE 子句中使用条件“A　LIKE　'_a%'”时，属性 A 取(　　)值可以满足查询条件。

A. also　　B. bats　　C. that　　D. clear

16. 下列选项中对输出结果的行数没有影响的是(　　)。

A. GROUP BY　　B. WHERE　　C. HAVING　　D. ORDER BY

17. 下列聚集函数中不忽略空值(NULL)的是(　　)。

A. SUM(列名)　　B. MAX(列名)

C. COUNT(列名)　　D. AVG(列名)

18. 设有关系 R(A,B)和 S(B,C),则下列 SQL 语句中含有语法错误的是(　　)。

Ⅰ SELECT A,B　FROM R　GROUP BY A

Ⅱ　SELECT A,B　FROM R,S　WHERE R.A=S.C

Ⅲ　SELECT COUNT(B)　FROM R

Ⅳ　SELECT A　FROM R　WHERE B>=MAX(B)

A. Ⅲ、Ⅳ　　B. Ⅰ、Ⅲ　　C. Ⅱ、Ⅲ　　D. Ⅰ、Ⅱ、Ⅳ

19. 使用带有 IN 谓词的子查询时,子查询的 SELECT 子句中最多可以指定(　　)个列。

A. 1　　B. 2　　C. 3　　D. 任意多

20. 使用嵌套查询时,当子查询的 SELECT 子句中出现多个列时,可以使用(　　)运算符。

A. =　　B. >=　　C. EXISTS　　D. IN

二、填空题

1. 在 SQL Server 系统中,每个数据库至少含有两个文件,其中一个为_____文件,其扩展名为.mdf,另一个为_____文件,其扩展名为.ldf。

2. 在 SQL Server 系统中,文件组是数据库中数据文件的逻辑组合。利用文件组可以加快_____,一个数据文件只能属于_____个文件组。

3. 在 SQL Server 系统中,每个数据库有且仅有_____个主文件组,主文件组中一定包含_____数据文件。

4. SQL 语言标准中提供的"大对象类型"有 clob 和 blob 两种,前者用于存放_____,而后者用于存放_____。

5. 在 SQL Server 系统中,数据库有_____数据库和用户数据库两类。基本表分为_____表和用户表两种。基本表的定义存放在数据库的_____中,而_____表中的数据构成了 SQL Server 系统的数据字典。

6. 在 SQL Server 系统中,用户表是由用户创建的表,它又分为永久表和临时表两种。永久表存储在_____数据库中用于存储用户数据,临时表存储在_____数据库中。

7. 在关系数据库系统中,对违反实体完整性和用户定义完整性约束条件的操作一般都采用_____执行方式处理。

8. 给完整性约束条件命名的作用就在于方便_____约束条件。

三、简答题

1. 什么是 DDL 语言、DML 语言、DCL 语言?

2. 试述 SQL 语言的特点。

3. 在 RDBMS 中,完整性控制机制应具有哪些功能?

4. 试述列级完整性约束条件和表级完整性约束条件的区别。

5. SQL 语言中是如何实现实体完整性和参照完整性的?

6. 在 SQL Server 系统中有哪几种形式实现用户定义完整性?

7. 在 RDBMS 中,在被参照关系中删除元组或修改主码值时,若违反参照完整性,一般有哪几种处理方式?

8. 在 RDBMS 中,在参照关系中插入元组或修改外码值时,若违反参照完整性,一般有哪几种处理方式?

四、SQL 语言

设某电子商务公司的数据库中四张基本表的结构如下(表中各属性的含义及各表的主码见第 2 章练习题四):

Customers 表

Cno	Cname	Csex	Cage	Caddress	Mphone	Email
char(8)	char(12)	char(2)	tinyint	varchar(50)	char(11)	varchar(30)

Goods 表

Gno	Gname	Gtype	Price	Manufac
char(9)	char(20)	char(8)	decimal(9,2)	char(12)

Sells 表

Sno	Sdate	Saddress	Cno	IsPay
char(14)	datetime	varchar(50)	char(8)	char(1)

Detail 表

Sno	Gno	Quantity
char(14)	char(9)	smallint

请用 T-SQL 语言实现下列要求:

1. 创建上述四张表,并定义主码、外码和必要的用户定义完整性约束条件(除了 Caddress 和 Email 外,其余所有属性均不可为空,Csex 取值只能为男或女,Price 取值为 1～10^5,IsPay 取值只能为 N 或 Y,Quantity 取值为 1～100)。

2. 添加约束条件:Cage 取值不小于 10,Mphone 取值唯一。

3. 将 Quantity 的取值范围改为 1～1000。

4. 找出“海尔”公司生产的所有商品的名称和价格,并按价格降序排列。

5. 找出“华为”公司生产的商品名称中有“手机”两字的所有商品的名称和价格。

6. 找出在售商品的品种总数。

7. 找出各大类商品中各种商品的品种数和平均价格。

8. 找出各大类商品中各种商品的平均价格大于 1000 元的商品类别。

9. 找出 2018 年 5 月份的所有未付款的销售单的编号、客户编号、客户姓名和手机号。

10. 找出每一个客户的编号、姓名、手机号以及他每次购物的日期和是否已付款的信息，即使该客户没有购买过商品，也要找出他的编号、姓名和手机号。

11. 找出每一张销售单的销售单号、销售日期、客户姓名和销售单总金额。

12. 找出“TP-LINK”公司生产的商品名为“WR700N 无线路由器”的销售总数量。

13. 找出仅仅注册但至今还没有购买过商品的客户编号。

14. 找出 2018 年 1 月 1 日以后没有购买过商品的客户编号、客户姓名和手机号。

15. 找出同类商品中价格最低的商品编号、商品名称和生产商。

16. 找出销售数量最多的商品的名称、价格和生产商。

17. 找出同一张销售单中既有 140010123 号商品又有 150020234 号商品的所有销售单号。

18. 找出购买过“奶粉”类商品中所有品种奶粉的客户编号。

19. 在 FROM 子句中使用子查询，重做第 15 题中的查询要求。

20. 将商品表中“手机”类商品中所有品种手机的价格降价 5%。

21. 删除商品表中商品编号为 110050111 的商品。

22. 在商品表中添加一种新商品('150050111','P8 手机','手机',2499,'华为')。