软件开发视频大讲堂
RUANJIANKAIFASHIPINDAJIANGTANG

八年锤炼
打造经典

SQL Server

从入门到精通

13小时语音视频讲解

明日科技 编著

☑ 实例资源库 ☑ 模块资源库 ☑ 项目资源库
☑ 测试题库系统 ☑ PPT电子课件

（第2版）

循序渐进，实战讲述

基础知识➪ 核心技术➪ 高级应用➪项目实战
270个学习实例，45个练习实践，3个项目案例

海量资源，可查可练

除本书配套的13小时视频讲解外，根据学习顺序，光盘还额外配备如下海量开发资源库：

实例资源库（126个实例）➪模块资源库（15个经典模块）➪项目资源库（15个项目案例）➪ 测试题库系统（596道测试题）

在线解答，高效学习

QQ: 400 675 1066（可容纳10万人在线）

清华大学出版社

内 容 简 介

《SQL Server 从入门到精通（第 2 版）》从初学者角度出发，通过通俗易懂的语言、丰富多彩的实例，详细介绍了 SQL Server 2012 开发应该掌握的各方面技术。全书共分为 4 篇 20 章，包括数据库基础、初识 SQL Server 2012、SQL Server 2012 服务的启动与注册、创建与管理数据库、操作数据表、SQL 基础、SQL 函数的使用、SQL 数据查询基础、SQL 数据高级查询、视图的使用、存储过程、触发器、游标的使用、索引与数据完整性、SQL 中的事务、维护 SQL Server 2012、数据库的安全机制、Visual C++ + SQL Server 实现图书管理系统、C# + SQL Server 实现企业人事管理系统、Java + SQL Server 实现企业进销存管理系统等。所有知识都结合具体实例进行介绍，涉及的程序代码给出了详细的注释，读者可以轻松领会 SQL Server 2012 的精髓，快速提高开发技能。另外，本书除了纸质内容之外，配书光盘中还给出了海量开发资源库，主要内容如下：

- ☑ 语音视频讲解：总时长 13 小时，共 103 段
- ☑ 实例资源库：126 个实例及源码分析
- ☑ 模块资源库：15 个经典模块开发过程完整展现
- ☑ 项目资源库：15 个企业开发项目
- ☑ 测试资源库：596 道能力测试题目
- ☑ PPT 电子教案

本书内容详尽，实例丰富，非常适合作为编程初学者的学习用书，也适合作为开发人员的查阅、参考资料。

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。
版权所有，侵权必究。侵权举报电话：010-62782989　13701121933

图书在版编目（CIP）数据

SQL Server 从入门到精通/明日科技编著. —2 版. —北京：清华大学出版社，2017
（软件开发视频大讲堂）
ISBN 978-7-302-45821-0

I. ①S…　II. ①明…　III. ①关系数据库系统　IV. ①TP311.138

责任编辑：赵洛育
封面设计：刘洪利
版式设计：魏　远
责任校对：王　云
责任印制：王静怡

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编**：100084
社 总 机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：北京嘉实印刷有限公司
经　　销：全国新华书店
开　　本：203mm×260mm　　**印　　张**：32　　**字　　数**：873 千字
（附海量开发资源库 DVD 1 张）
版　　次：2012 年 9 月第 1 版　2017 年 9 月第 2 版　　**印　　次**：2017 年 9 月第1 次印刷
印　　数：1～5000
定　　价：79.80 元

产品编号：058856-01

光盘“开发资源库”简明目录

☑ 实例资源库

- SQL语言范例库
  - SQL语言基础
  - 常规数据查询
  - 高级数据过滤
  - 字符串查询
  - 日期查询
  - 数据排序
  - 聚合函数与分组统计
  - 使用子查询
  - 多表查询
  - 高级查询
  - 插入数据
  - 更新和删除数据
  - ……

☑项目资源库

- 供求信息网
- 明日播客网
- 电子商务平台
- 都市网络新闻中心
- 基于XML技术的在线论坛
- 物流信息发布平台
- 电子商务网站
- 图书馆管理系统
- 新闻发布系统
- OA企业办公自动化
- ……

☑ 模块资源库

- 论坛模块
- 博客模块
- 播客
- 网络硬盘
- 在线考试模块
- ……

☑ 测试题库系统

- 数学及逻辑思维能力测试
  - 基本测试
  - 进阶测试
  - 高级测试
- 面试能力测试
  - 常规面试测试
- 编程英语能力测试
- ……

在线解答，高效学习
QQ: 400 675 1066(可容纳10万人在线)
官方网站: www.mingribook.com

丛书历史与荣誉

“软件开发视频大讲堂”丛书自2008年出版，已经过两次改版，因其编写细腻、易学实用、配套资源完善（包括全程实例视频讲解、实例源程序、教学课件PPT等）而备受读者欢迎，多个品种被近百所高校计算机相关专业、软件学院选为教学参考书，并连续八年在全国软件开发类零售图书排行榜中名列前茅。

推荐阅读

实例类图书

《Java开发实例大全》的基础卷、提高卷分别精选了约600个实例和约600个秘笈心法，涵盖了Java编程多个方面的各种应用，是目前市场上最全面的软件开发实例类丛书（丛书名：软件工程师开发大系）中的品种，该丛书堪称软件开发实例的“四库全书”，开发中所需的技术、技巧在本丛书中几乎都可以找到。

如何使用本书开发资源库

在学习《SQL Server 从入门到精通(第 2 版)》一书时，配合随书光盘提供了“ASP.NET + SQL Server 自主学习系统.exe”，可以帮助读者快速提升编程水平和解决实际问题的能力。《SQL Server 从入门到精通（第 2 版）》和“ASP.NET + SQL Server 自主学习系统.exe”配合学习流程如图 1 所示。

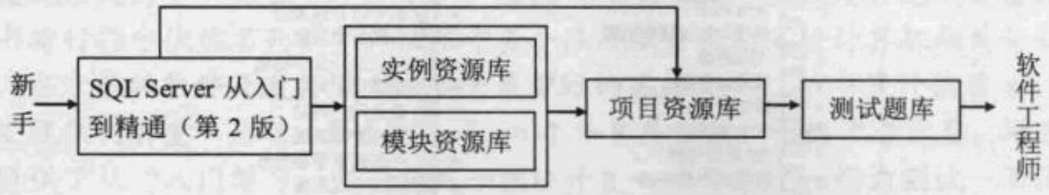


图 1　从入门到精通与自主学习系统配合学习流程图

打开光盘的“开发资源库”文件夹，运行 ASP.NET + SQL Server 自主学习系统.exe 程序，即可进入自主学习系统，主界面如图 2 所示。

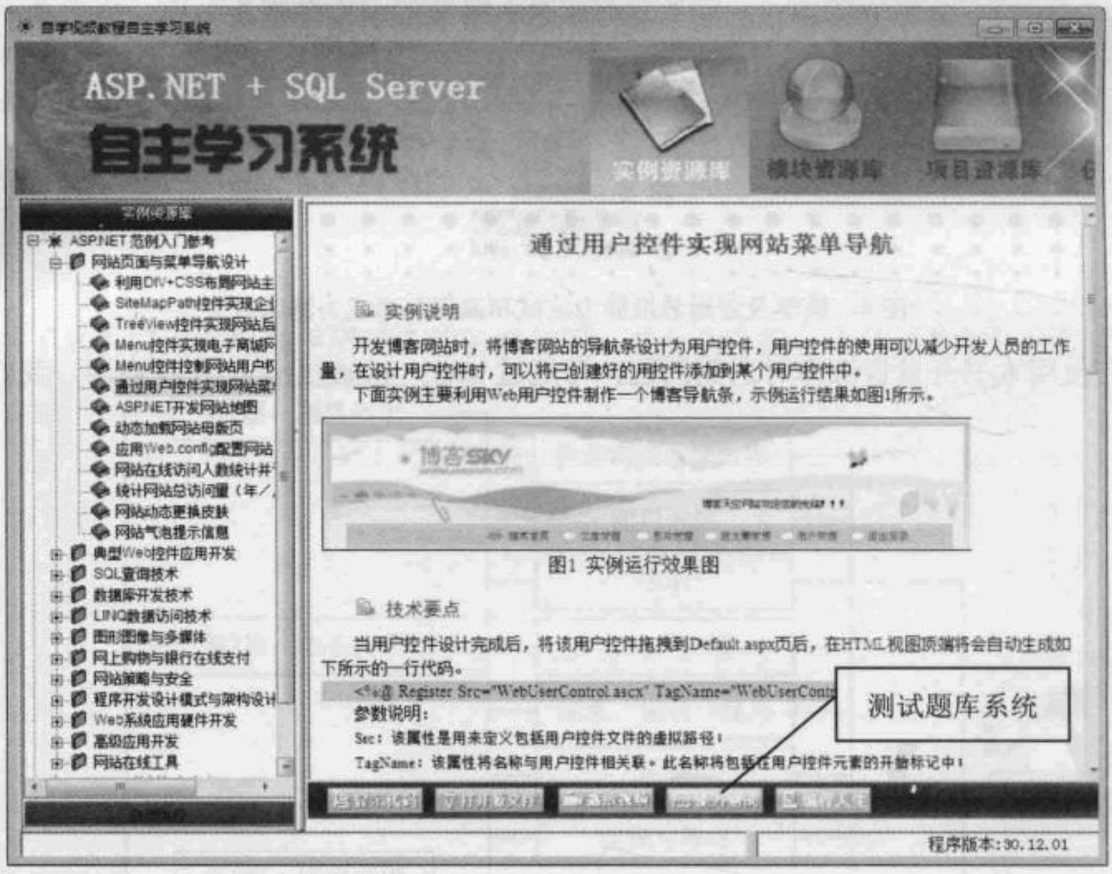


图 2　ASP.NET + SQL Server 自主学习系统

在学习《SQL Server 从入门到精通（第 2 版）》的过程中，可以选择实例资源库和项目资源库的相应内容，全面提升个人综合编程技能和解决实际开发问题的能力，为成为软件开发工程师打下坚实

基础。具体实例资源库和项目资源库目录如图 3 所示。

图 3　实例资源库和项目资源库目录

对于数学逻辑能力和英语基础较为薄弱的读者，或者想了解个人数学逻辑思维能力和编程英语基础的用户，本书提供了数学及逻辑思维能力测试和编程英语能力测试供练习和测试，如图 4 所示。

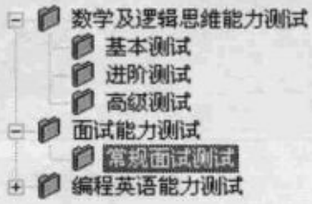


图 4　数学及逻辑思维能力测试和编程英语能力测试目录

如果您在使用本书开发资源库时遇到问题，可通过 QQ：4006751066 在线咨询，我们将竭诚为您服务。

前 言

Preface

丛书说明：“软件开发视频大讲堂”丛书（第1版）于2008年8月出版，因其编写细腻，易学实用，配备全程视频等，在软件开发类图书市场上产生了很大反响，绝大部分品种在全国软件开发零售图书排行榜中名列前茅，2009年多个品种被评为“全国优秀畅销书”。

“软件开发视频大讲堂”丛书（第2版）于2010年8月出版，出版后，绝大部分品种在全国软件开发类零售图书排行榜中依然名列前茅。丛书中多个品种被百余所高校计算机相关专业、软件学院选为教学参考书，在众多的软件开发类图书中成为最耀眼的品牌之一。丛书累计销售40多万册。

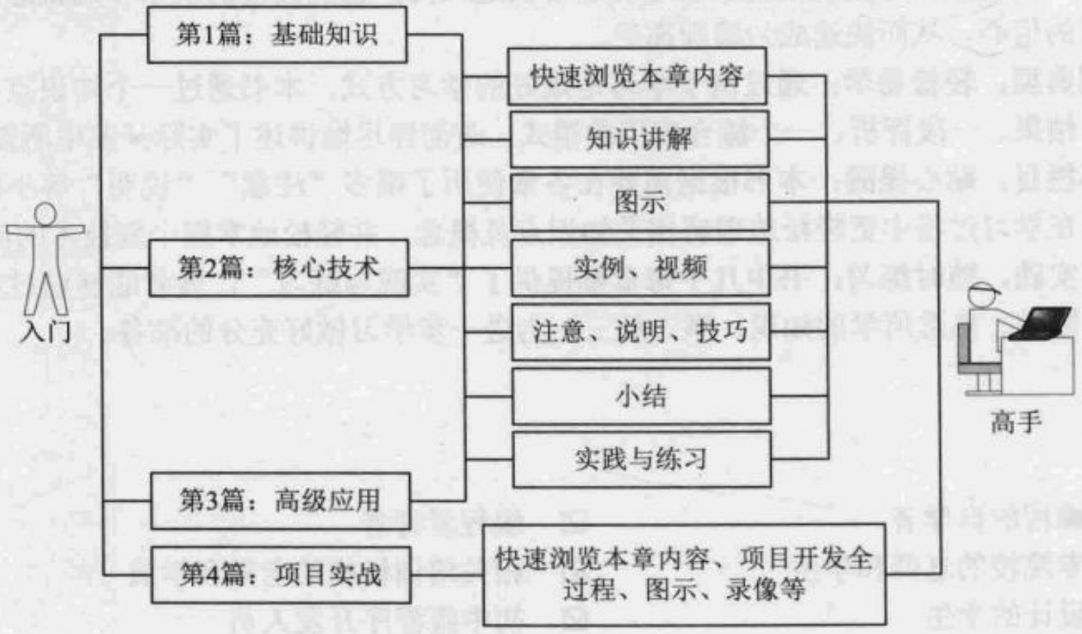
“软件开发视频大讲堂”丛书（第3版）于2012年8月出版，根据读者需要，增删了品种，重新录制了视频，提供了从“入门学习→实例应用→模块开发→项目开发→能力测试→面试”等各个阶段的海量开发资源库。因丛书编写结构合理、实例选择经典实用，丛书迄今累计销售90多万册。

“软件开发视频大讲堂”丛书（第4版）在继承前3版所有优点的基础上，修正了前3版图书中发现的疏漏之处，并结合目前市场需要，进一步对丛书品种进行了完善，对相关内容进行了更新优化，使之更适合读者学习，为了方便教学，还提供了教学课件PPT。

SQL Server 是由美国微软公司制作并发布的一种性能优越的关系型数据库管理系统（Relational Database Management System，RDBMS），因其具有良好的数据库设计、管理与网络功能，又与 Windows、Windows 2000 以及 Windows XP 系统紧密集成，因此成为数据库产品的首选。

本书内容

本书提供了从入门到编程高手所必备的各类知识，共分为4篇，大体结构如下所示。



第 1 篇：基础知识。本篇通过数据库基础、初识 SQL Server 2012、SQL Server 2012 服务的启动与注册、创建与管理数据库、操作数据表的内容介绍，并结合大量的图示、举例、录像等帮助读者快速掌握 SQL Server 2012，并为以后的知识奠定坚实的基础。

第 2 篇：核心技术。本篇介绍 SQL 基础、SQL 函数的使用、SQL 数据查询基础、SQL 数据高级查询、视图的使用等。学习完这一部分，能够了解和熟悉 T-SQL 及常用的函数，使用 T-SQL 操作 SQL Server 2012 数据库中的视图，掌握 SQL 查询、子查询、嵌套查询、联接查询的用法等。

第 3 篇：高级应用。本篇介绍存储过程、触发器、游标的使用、索引与数据完整性、SQL 中的事务、维护 SQL Server 2012、数据库的安全机制等。学习完这一部分，能够使用索引优化数据库查询；使用存储过程、触发器、游标、事务等编写 SQL 语句，不仅可以优化查询，还可以提高数据访问速度；更好地维护 SQL Server 2012 及其安全。

第 4 篇：项目实战。本篇分别使用 Visual C++、C#、Java 3 种语言，结合 SQL Server 2012 实现了 3 个大型、完整的管理系统，通过这 3 个项目，运用软件工程的设计思想，帮助读者学习如何进行软件项目的实践开发。书中按照编写系统分析→系统设计→数据库与数据表设计→公共类设计→创建项目→实现项目→项目总结的过程进行介绍，带领读者一步一步亲身体验开发项目的全过程。

本书特点

- ☑ **由浅入深，循序渐进：**本书以初、中级程序员为对象，先从 SQL Server 基础学起，再学习 SQL Server 的核心技术，然后学习 SQL Server 的高级应用，最后学习分别使用 Visual C++、C#、Java 等语言结合 SQL Server 2012 开发完整项目。讲解过程中步骤详尽，版式新颖，在操作的内容图片上以“❶❷❸…”编号+内容的方式进行标注，让读者在阅读中一目了然，从而快速把握书中内容。
- ☑ **语音视频，讲解详尽：**书中每一章节均提供声图并茂的教学视频，读者可以在光盘中找到相应章节的视频。这些视频能够引导初学者快速入门，感受编程的快乐和成就感，增强进一步学习的信心，从而快速成为编程高手。
- ☑ **实例典型，轻松易学：**通过例子学习是最好的学习方式，本书通过一个知识点、一个例子、一个结果、一段评析、一个综合应用的模式，透彻详尽地讲述了实际开发中所需的各类知识。
- ☑ **精彩栏目，贴心提醒：**本书根据需要在各章使用了很多“注意”“说明”等小栏目，让读者可以在学习过程中更轻松地理解相关知识点及概念，并轻松地掌握个别技术的应用技巧。
- ☑ **应用实践，随时练习：**书中几乎每章都提供了“实践与练习”，读者能够通过对问题的解答重新回顾、熟悉所学的知识，举一反三，为进一步学习做好充分的准备。

读者对象

- ☑ 初学编程的自学者
- ☑ 大中专院校的老师和学生
- ☑ 毕业设计的学生
- ☑ 程序测试及维护人员
- ☑ 编程爱好者
- ☑ 相关培训机构的老师和学员
- ☑ 初中级程序开发人员
- ☑ 参加实习的“菜鸟”程序员

读者服务

为了方便读者，本书提供了学习答疑网站：www.mingribook.com。有关本书的内容读者均可在网站上留言，我们力求在 24 小时内回复，节假日除外。

致读者

本书由明日科技策划并组织编写，主要编写人员有申小琦、王小科、王国辉、董刚、赛奎春、房德山、杨丽、高春艳、辛洪郁、周佳星、张鑫、张宝华、葛忠月、刘杰、白宏健、张雳霆、马新新、冯春龙、宋万勇、李文欣、王东东、柳琳、王盛鑫、徐明明、杨柳、赵宁、王佳雪、于国良、李磊、李彦骏、王泽奇、贾景波、谭慧、李丹、吕玉翠、孙巧辰、赵颖、江玉贞、周艳梅、房雪坤、裴莹、郭铁、张金辉、王敬杰、高茹、李贺、陈威、高飞、刘志铭、高润岭、于国槐、郭锐、郭鑫、邹淑芳、李根福、杨贵发、王喜平等，在编写本书的过程中，作者以科学、严谨的态度，力求精益求精，但疏漏之处在所难免，敬请广大读者批评指正。我们的服务邮箱是 tmoonbook@sina.com，th_press@263.net。读者在阅读本书时，如果发现或遇到问题，可以发送电子邮件及时与我们联系，我们会尽快给予答复。

感谢您购买本书，希望本书能成为您编程路上的领航者。

“零门槛”编程，一切皆有可能。祝读书快乐！

编　者

目 录

Contents

第1篇 基 础 知 识

第 2 篇 核心技术

第 3 篇 高级应用

第 4 篇 项目实战

光盘“开发资源库”目录

第 1 大部分　实例资源库

（126 个实例，光盘路径：开发资源库/实例资源库）

第 2 大部分　模块资源库

（15 个经典模块，光盘路径：开发资源库/模块资源库）

第 3 大部分　项目资源库

（15 个企业开发项目，光盘路径：开发资源库/项目资源库）

第 4 大部分　测试资源库

（596 道能力测试题目，光盘路径：开发资源库/能力测试）

基础知识

本篇通过数据库基础、初识 SQL Server 2012、SQL Server 2012 服务的启动与注册、创建与管理数据库、操作数据表的内容介绍，并结合大量的图示、举例、录像等帮助读者快速掌握 SQL Server 2012，并为以后的学习奠定坚实的基础。

第 1 章

数据库基础

（视频讲解：26 分钟）

本章主要介绍数据库的相关概念，主要包括数据库系统的简介、数据库的体系结构、数据模型、常见关系数据库。通过本章的学习，读者应该掌握数据库系统、数据模型、数据库三级模式结构以及数据库规范化等概念，对比常见的关系数据库。

通过阅读本章，您可以：

- 了解数据库技术的发展
- 掌握数据库系统的组成
- 掌握数据库的体系结构
- 熟悉数据模型
- 掌握常见的关系数据库

1.1　数据库系统简介

视频讲解：光盘\TM\lx\1\数据库系统简介.mp4

数据库系统（DataBase System）是由数据库及其管理软件组成的系统，人们常把与数据库有关的硬件和软件系统称为数据库系统。

1.1.1　数据库技术的发展

数据库技术是应数据管理任务的需求而产生的，随着计算机技术的发展，对数据管理技术也不断地提出更高的要求，其先后经历了人工管理、文件系统、数据库系统等 3 个阶段。

1．人工管理阶段

20 世纪 50 年代中期以前，计算机主要用于科学计算。当时硬件和软件设备都很落后。数据基本依赖于人工管理。人工管理数据具有如下特点：

- ☑ 数据不保存。
- ☑ 使用应用程序管理数据。
- ☑ 数据不共享。
- ☑ 数据不具有独立性。

2．文件系统阶段

20 世纪 50 年代后期到 20 世纪 60 年代中期，硬件和软件技术都有了进一步发展，有了磁盘等存储设备和专门的数据管理软件即文件系统，其具有如下特点：

- ☑ 数据可以长期保存。
- ☑ 由文件系统管理数据。
- ☑ 共享性差，数据冗余大。
- ☑ 数据独立性差。

3．数据库系统阶段

20 世纪 60 年代后期以来，计算机应用于管理系统，而且规模越来越大，应用越来越广泛，数据量急剧增长，对共享功能的要求越来越强烈。这样使用文件系统管理数据已经不能满足要求，于是为了解决一系列问题，出现了数据库系统来统一管理数据。其满足了多用户、多应用共享数据的需求，比文件系统具有明显的优点，标志着管理技术的飞跃。

1.1.2　数据库系统的组成

数据库系统（DataBase System，DBS）是采用数据库技术的计算机系统，是由数据库（数据）、数

据库管理系统、数据库管理员（人员）、支持数据库系统的硬件和软件（应用开发工具、应用系统等）、用户 5 部分构成的运行实体，如图 1.1 所示。其中，数据库管理员（DataBase Administrator，DBA）是对数据库进行规划、设计、维护和监视等的专业管理人员，在数据库系统中起着非常重要的作用。

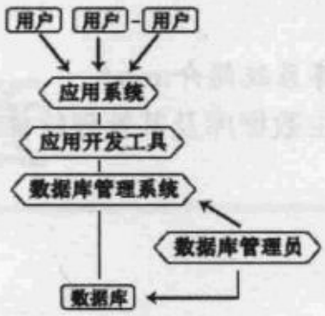


图 1.1　数据库系统的组成

1.2　数据库的体系结构

视频讲解：光盘\TM\lx\1\数据库的体系结构.mp4

数据库具有一个严谨的体系结构，这样可以有效地组织、管理数据，提高数据库的逻辑独立性和物理独立性。数据库领域公认的标准结构是三级模式结构，如图 1.2 所示。

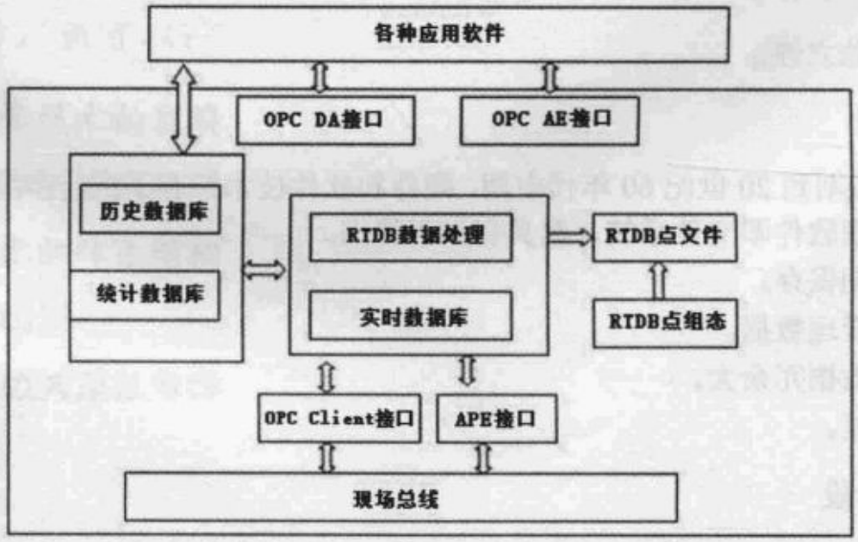


图 1.2　数据库体系结构

1.2.1　数据库三级模式结构

数据库系统的三级模式结构是指模式、外模式和内模式。下面分别进行介绍。

1．模式

模式也称逻辑模式或概念模式，是数据库中全体数据的逻辑结构和特征的描述，是所有用户的公

共数据视图。一个数据库只有一个模式。模式处于三级结构的中间层。

注意

定义模式时不仅要定义数据的逻辑结构，而且要定义数据之间的联系，定义与数据有关的安全性、完整性要求。

2. 外模式

外模式也称用户模式，它是数据库用户（包括应用程序员和最终用户）能够看见和使用的局部数据的逻辑结构和特征的描述，是数据库用户的数据视图，是与某一应用有关的数据的逻辑表示。外模式是模式的子集。一个数据库可以有多个外模式。

说明

外模式是保证数据安全性的一个有力措施。

3. 内模式

内模式也称存储模式，一个数据库只有一个内模式。它是数据物理结构和存储方式的描述，是数据在数据库内部的表示方式。

1.2.2　三级模式之间的映射

为了能够在内部实现数据库的 3 个抽象层次的联系和转换，数据库管理系统在三级模式之间提供了两层映射。

1. 外模式/模式映射

对应于同一个模式可以有任意多个外模式。对于每一个外模式，数据库系统都有一个外模式/模式映射。当模式改变时，由数据库管理员对各个外模式/模式映射做相应的改变，可以使外模式保持不变。这样，依据数据外模式编写的应用程序就不用修改，保证了数据与程序的逻辑独立性。

2. 模式/内模式映射

数据库中只有一个模式和一个内模式，所以模式/内模式映射是唯一的，它定义了数据库的全局逻辑结构与存储结构之间的对应关系。当数据库的存储结构改变时，由数据库管理员对模式/内模式映射做相应改变，可以使模式保持不变，应用程序相应地也不做变动。这样，保证了数据与程序的物理独立性。

1.3　数 据 模 型

视频讲解：光盘\TM\lx\1\数据模型.mp4

数据模型就是一种对客观事物抽象化的表现形式。它对客观事物加以抽象，通过计算机来处理现

实世界中的具体事物。它客观地反映了现实世界，易于理解，与人们对外部事物描述的认识相一致。

1.3.1 数据模型的概念

数据模型是数据库系统的核心与基础，是关于描述数据与数据之间的联系、数据的语义、数据一致性约束的概念性工具的集合。

数据模型通常是由数据结构、数据操作和完整性约束 3 部分组成的。

- ☑ 数据结构：是对系统静态特征的描述。描述对象包括数据的类型、内容、性质和数据之间的相互关系。
- ☑ 数据操作：是对系统动态特征的描述。是对数据库中各种对象实例的操作。
- ☑ 完整性约束：是完整性规则的集合。它定义了给定数据模型中数据及其联系所具有的制约和依存规则。

1.3.2 常见的数据模型

常用的数据库数据模型主要有层次模型、网状模型和关系模型，下面分别进行介绍。

（1）层次模型：用树状结构表示实体类型及实体间联系的数据模型称为层次模型，它具有以下特点。

- ☑ 每棵树有且仅有一个无双亲结点，称为根。
- ☑ 树中除根外所有结点有且仅有一个双亲。

层次模型示例图如图 1.3 所示。

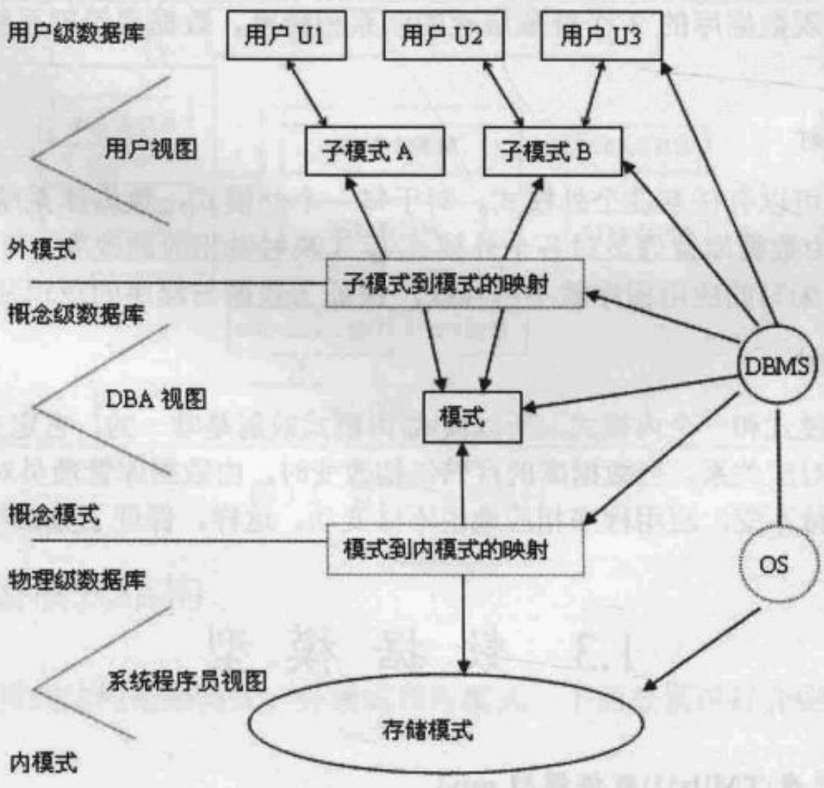


图 1.3 层次模型

（2）网状模型：用有向图结构表示实体类型及实体间联系的数据模型称为网状模型。用网状模型编写应用程序极其复杂，数据的独立性较差。网状模型示例图如图 1.4 所示。

（3）关系模型：以二维表来描述数据。关系模型中，每个表有多个字段列和记录行，每个字段列有固定的属性（数字、字符、日期等）。关系模型数据结构简单、清晰、具有很高的数据独立性，因此是目前主流的数据库数据模型。

关系模型的基本术语如下。

☑ 关系：一个二维表就是一个关系。

☑ 元组：就是二维表中的一行，即表中的记录。

☑ 属性：就是二维表中的一列，用类型和值表示。

☑ 域：每个属性取值的变化范围，如性别的域为{男，女}。

关系中的数据约束如下。

☑ 实体完整性约束：约束关系的主键中属性值不能为空值。

☑ 参照完整性约束：关系之间的基本约束。

☑ 用户定义的完整性约束：它反映了具体应用中数据的语义要求。

关系模型示例图如图 1.5 所示。

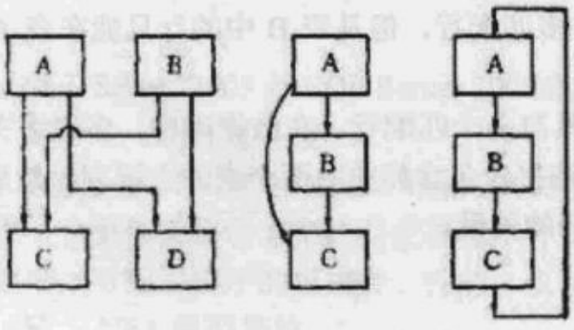


图 1.4　网状模型

学生信息表

学生姓名	年级	家庭住址
张三	2000	成都
李四	2000	北京
王五	2000	上海

成绩表

学生姓名	课程	成绩
张三	数学	100
张三	物理	95
张三	社会	90
李四	数学	85
李四	社会	90
王五	数学	80
王五	物理	75

图 1.5　关系模型

1.3.3　关系数据库的规范化

关系数据库的规范化理论认为：关系数据库中的每一个关系都要满足一定的规范。根据满足规范的条件不同，可以分为 5 个等级：第一范式（1NF）、第二范式（2NF）、……、第五范式（5NF）。其

中，NF 是 Normal Form 的缩写。一般情况下，只要把数据规范到第三个范式标准就可以满足需要了。

- ☑ 第一范式（1NF）：在一个关系中，消除重复字段，且各字段都是最小的逻辑存储单位。
- ☑ 第二范式（2NF）：若关系模型属于第一范式，则关系中每一个非主关键字段都完全依赖于主关键字段，不能只部分依赖于主关键字的一部分。
- ☑ 第三范式（3NF）：若关系属于第一个范式，且关系中所有非主关键字段都只依赖于主关键字段。第三范式要求去除传递依赖。

1.3.4 关系数据库的设计原则

数据库设计是指对于一个给定的应用环境，根据用户的需求，利用数据模型和应用程序模拟现实世界中该应用环境的数据结构和处理活动的过程。

数据库设计原则如下。

（1）数据库内数据文件的数据组织应获得最大限度的共享、最小的冗余度，消除数据及数据依赖关系中的冗余部分，使依赖于同一个数据模型的数据达到有效的分离。

（2）保证输入、修改数据时数据的一致性与正确性。

（3）保证数据与使用数据的应用程序之间的高度独立性。

1.3.5 实体与关系

实体是指客观存在并可相互区别的事物。实体既可以是实际的事物，也可以是抽象的概念或关系。实体之间有以下 3 种关系。

- ☑ 一对一关系：是指表 A 中的一条记录确实在表 B 中有且只有一条相匹配的记录。在一对一关系中，大部分相关信息都在一个表中。
- ☑ 一对多关系：是指表 A 中的行可以在表 B 中有许多匹配行，但是表 B 中的行只能在表 A 中有一个匹配行。
- ☑ 多对多关系：是指关系中每个表的行在相关表中具有多个匹配行。在数据库中，多对多关系的建立是依靠第 3 个表（称作连接表）实现的，连接表包含相关的两个表的主键列，然后从两个相关表的主键列分别创建与连接表中的匹配列的关系。

1.4 常见关系数据库

视频讲解：光盘\TM\lx\1\常见关系数据库.mp4

关系数据库，是建立在关系数据库模型基础上的数据库，通过集合代数等概念和方法来处理数据库中的数据库。在这里主要介绍 Access、SQL Server 和 Oracle 数据库。

1.4.1　Access 数据库

Microsoft Access 是当前流行的关系型数据库管理系统之一，其核心是 Microsoft Jet 数据库引擎。通常情况下，安装 Microsoft Office 时选择“默认安装”，Access 数据库即被安装到计算机上。

Microsoft Access 是一个非常容易掌握的数据库管理系统。利用它可以创建、修改和维护数据库和数据库中的数据，并且可以利用向导来完成对数据库的一系列操作。Access 能够满足小型企业客户/服务器解决方案的要求，是一种功能较完备的系统，它几乎包含数据库领域的所有技术和内容，对于初学者学习数据库知识非常有帮助。

1.4.2　SQL Server 2000 数据库

SQL Server 是由微软公司开发的一个大型的关系数据库系统，它为用户提供了一个安全、可靠、易管理和高端的客户/服务器数据库平台。

SQL Server 是一种高性能的关系型数据库管理系统，以 Client/Server 为设计结构、支持多个不同的开发平台、支持企业级的应用程序、支持 XML 等。能够满足不同类型的数据库解决方案。SQL Server 数据库大大扩展了系统的性能、可靠性和易用性。

SQL Server 2000 的主要特点包括简便的操作方式、以 Client/Server 为设计结构、支持多个不同的开发平台、支持企业级的应用程序、支持 XML、数据仓库支持、支持虚拟根、用户自定义函数、增加了 BIGINT、SQL_VARIANT 和 TABLE 3 种数据类型、提供了语言排序规则、增加了全文搜索和文档管理功能、增加了索引视图功能、增强了分布式查询功能等。

1.4.3　SQL Server 2005 数据库

SQL Server 2005 是 SQL Server 2000 的升级版本，其优势主要表现在以下两个方面。

☑　统一的开发环境

SQL Server 2005 和 Visual Studio 2005 拥有一个统一的开发环境，使得集成于其中的编程模型能够提供一个整体的解决方案，并使得程序开发语言、产品配置环境和数据操作这 3 种专业技能紧密地结合起来，对应用程序的可用性、性能、安全性和可伸缩性带来全面提升。

☑　.NET 框架集成

SQL Server 2005 的核心由.NET Framework 2.0 构成，数据库的工具整合了开发环境，有利于数据库的程序开发。

1.4.4　SQL Server 2008 数据库

SQL Server 2008 是 SQL Server 2005 的升级版本，其主要分为企业版、标准版和工作组版这 3 个版

本，下面分别进行简单介绍。

- ☑ SQL Server 2008 企业版是一个全面的数据管理和业务智能平台，为关键业务应用提供了企业级的可扩展性、数据仓库、安全、高级分析和报表支持，这一版本将为用户提供更加坚固的服务器和执行大规模在线事务处理。
- ☑ SQL Server 2008 标准版是一个完整的数据管理和业务智能平台，为部门级应用提供了最佳的易用性和可管理特性。
- ☑ SQL Server 2008 工作组版是一个值得信赖的数据管理和报表平台，用以实现安全的发布、远程同步和对运行分支应用的管理能力，这一版本拥有核心的数据库特性，可以很容易地升级到标准版或企业版。

1.4.5 SQL Server 2012 数据库

SQL Server 数据库系列历经多次升级，于 2012 年 4 月 1 日，微软正式发布了 SQL Server 2012 版本。SQL Server 2012 是 SQL Server 的最新版本，是迄今为止最强大和最全面的 SQL Server 版本。SQL Server 2012 作为新一代的数据平台产品，它不仅延续了现有数据平台的强大能力，全面支持云技术与平台，而且能够快速构建相应的解决方案，实现私有云与公有云之间数据的扩展与应用的迁移。

SQL Server 2012 主要包括企业版和标准版两个版本，另外，还有一个新增的商业智能版，对于个人用户或者中小型企业来说，一般选择标准版即可，而如果要求比较高，可以使用企业版。SQL Server 2012 的优点如下：

- ☑ 提高服务器正常运行时间，并加强数据保护，无须浪费时间和金钱即可实现服务器到云端的扩展。
- ☑ 内置的安全性功能及 IT 管理功能，能够在极大程度上帮助企业提供安全性能级别，并实现有效管理。
- ☑ 得益于卓越的服务和技术支持、大量值得信赖的合作伙伴以及丰富的免费工具，用户可以放心使用。
- ☑ 通过快速的数据探索和数据可视化对成堆的数据进行细致深入的研究，从而能够引导企业提出更为深刻的商业观点。
- ☑ 针对所有业务数据提供一个全方位的视图。
- ☑ 凭借全方位数据仓库解决方案，以低成本向用户提供大规模的数据容量，能够实现较强的灵活性和可伸缩性。
- ☑ 个性私有云。

说明

（1）SQL Server 2012 只能安装在 Windows 7 及以上版本的操作系统中，而不能安装在 Windows XP 或者 Windows 2003 等低版本的操作系统中。

（2）本书的所有内容都是用 SQL Server 2012 进行讲解。

1.4.6　Oracle 数据库

Oracle 是美国 Oracle 公司（甲骨文）提供的以分布式数据库为核心的一组软件产品。Oracle 是目前世界上使用最为广泛的关系型数据库。它具有完整的数据管理功能，包括数据的大量性、数据保存的持久性、数据的共享性、数据的可靠性。

Oracle 在并行处理、实时性、数据处理速度方面都有较好的性能。一般情况下，大型企业选择 Oracle 作为后台数据库来处理海量数据。

1.5　小　　结

本章介绍了数据库的基本概念、数据库系统的组成、数据库三级模式结构及映射、关系数据库的规范化及设计原则等。通过本章的学习，读者可以对数据库有一个系统的了解，在此基础上了解 SQL Server 2012 数据库，为进一步的学习奠定基础。

1.6　实践与练习

1．数据库技术的发展经历了哪 3 个阶段？（**答案位置：光盘\TM\sl\1\1**）

2．数据模型通常是由哪 3 部分组成的？（**答案位置：光盘\TM\sl\1\2**）

3．下面哪些是关系数据库？（**答案位置：光盘\TM\sl\1\3**）

（1）Access　　（2）SQL Server

（3）Oracle　　（4）XML

第 2 章

初识 SQL Server 2012

（视频讲解：33 分钟）

本章主要介绍 SQL Server 2012 的特点、安装、卸载及 SQL Server 2012 的帮助等基础知识。通过本章内容的学习，读者将对 SQL Server 2012 有一个全面的认识。

通过阅读本章，您可以：

- 了解 SQL Server 2012 的版本
- 了解 SQL Server 2012 的特点
- 熟悉 SQL Server 2012 新增的功能及技术
- 掌握 SQL Server 2012 的安装过程
- 掌握 SQL Server 2012 帮助文档的使用

2.1　SQL Server 2012 简介

SQL Server 2012 是一个重大的产品版本，它推出了许多新的特性和关键的改进，使得它成为目前最新和最强大的 SQL Server 版本。SQL Server 是使用客户/服务器体系结构的关系型数据库管理系统（RDBMS）。1996 年，微软公司推出了 SQL Server 6.5 版本，1998 年推出了 SQL Server 7.0 版本，2000 年推出了 SQL Server 2000，2005 年推出了 SQL Server 2005，2008 年推出了 SQL Server 2008。2012 年微软正式发布了 SQL Server 2012。目前，SQL Server 已经是世界上应用最广泛的大型数据库之一。

SQL Server 2012 提供了更高级别的企业稳定性，更加商业智能化，同时提供多种功能满足共有云以及私有云环境下的应用，它主要包括企业版和标准版两个版本，另外，还有一个新增的商业智能版，对于个人用户或者中小型企业来说，一般选择标准版即可，而如果要求比较高，可以使用企业版。下面对 SQL Server 2012 的主要版本进行介绍。

- ☑ 企业版：全功能版本，包含商业智能版的全部功能。
- ☑ 标准版：提供基本的数据库功能，主要面向工作组或者个人用户。
- ☑ 商业智能版：主要面向中小企业，包含标准版的全部功能，另外，还具有统一商业智能模型、管理自服务分析、使用 Power View 快速发现信息等功能。

2.2　SQL Server 2012 的特点

SQL Server 2012 实现了一个为云做准备的信息平台，使得企业和用户可以在它的帮助下快速生成解决方案和扩展数据。SQL Server 2012 提供了一个可信的高效率智能数据平台，可以满足所有的数据需求。

1. SQL Server 2012 的主要特点

SQL Server 2012 的主要特点如下：

- ☑ 提高服务器正常运行时间，并加强数据保护，无须浪费时间和金钱即可实现服务器到云端的扩展。
- ☑ 内置的安全性功能及 IT 管理功能，能够在极大程度上帮助企业提供安全性能级别，并实现有效管理。
- ☑ 得益于卓越的服务和技术支持、大量值得信赖的合作伙伴以及丰富的免费工具，用户可以放心使用。
- ☑ 通过快速的数据探索和数据可视化对成堆的数据进行细致深入的研究，从而能够引导企业提出更为深刻的商业观点。
- ☑ 针对所有业务数据提供一个全方位的视图。
- ☑ 凭借全方位数据仓库解决方案，以低成本向用户提供大规模的数据容量，能够实现较强的灵

活性和可伸缩性。

☑ 个性私有云，可以根据用户需要进行扩展、优化提高工作效率，并且通过易于扩展的开发技术，能够在服务器或云端对数据进行任意扩展。

2. SQL Server 2012 的新增功能

☑ AlwaysOn。通过该功能提供所需运行时间和数据保护。

☑ Windows Server Core 支持。Windows Server Core 是命令行界面的 Windows，使用 DOS 和 PowerShell 来做用户交互。它的资源占用更少，更安全，支持 SQL Server 2012。

☑ 列存储索引。通过列存储索引获得可突破性和可预测性能，实现快速数据恢复，以便更深入地了解组织。

☑ 自定义服务器权限。DBA 可以创建数据库的权限，但不能创建服务器的权限。例如，DBA 想要一个开发组拥有某台服务器上所有数据库的读写权限，他必须手动地完成这个操作。但是 SQL Server 2012 支持针对服务器的权限设置。

☑ 新用户定义角色和默认架构。通过该特性可以实现安全性和遵从性。

☑ BI 语义模型。这个功能是用来替代“Analysis Services Unified Dimentional Model”的。这是一种支持 SQL Server 所有 BI 体验的混合数据模型。

☑ 改进的 SSIS。通过改进的 SSIS、用于 Excel 的 Master Data Services 外接程序和新的 Data Quality Services，确保更加可靠、一致性的数据。

☑ PowerView。这是一个强大的自主 BI 工具，可以让用户创建 BI 报告。它可以为用户对数据的转换和勘探提供强大的交互操作能力，并协助做出正确的决策。

☑ 增强的 PowerShell 支持。所有的 Windows 和 SQL Server 管理员都应该认真地学习 PowderShell 的技能。微软正在大力开发服务器端产品对 PowerShell 的支持。

☑ 分布式回放（Distributed Replay）。这个功能类似 Oracle 的 Real Application Testing 功能。不同的是 SQL Server 企业版自带了这个功能，而用 Oracle 的话，还得额外购买这个功能。这个功能可以记录生产环境的工作状况，然后在另外一个环境重现这些工作状况。

☑ 大数据支持。这是最重要的一点。PASS 会议上，微软宣布了与 Hadoop 的合作，开发 Hadoop 的连接器，标志着 SQL Server 也跨入了 NoSQL 领域。

2.3 SQL Server 2012 的安装与卸载

视频讲解：光盘\TM\lx\2\SQL Server 2012 的安装.mp4

在对 SQL Server 2012 有了初步了解之后，就可以安装 SQL Server 2012 了。由于 SQL Server 2012 的安装程序提供了浅显、易懂的图形化操作界面，所以安装过程相对简单、快捷。但是，因为 SQL Server 2012 是由一系列相互协作的组件构成，又是网络数据库产品，所以安装时就必须要了解其中的选项含义及其参数配置，否则将直接影响安装过程。本节将向读者详细介绍 SQL Server 2012 的安装要求以及安装的全过程。

2.3.1　SQL Server 2012 安装必备

安装 SQL Server 2012 之前，首先要了解安装 SQL Server 2012 所需的必备条件，检查计算机的软硬件配置是否满足 SQL Server 2012 开发环境的安装要求，具体要求如表 2.1 所示。

表 2.1　安装 SQL Server 2012 所需的必备条件

软　硬　件	描　　述
操作系统	Windows 7、Windows 8、Windows 8.1、Windows Server 2008、Windows Server 2012
软件	SQL Server 安装程序需要使用 Microsoft Windows Installer 4.5 或更高版本以及 Microsoft 数据访问组件（MDAC）2.8 SP1 或更高版本
处理器	1.4GHz 处理器，建议使用 2.0GHz 或速度更快的处理器
RAM	最小 1GB，建议使用 2GB 或更大的内存
可用硬盘空间	至少 2.2GB 的可用磁盘空间
CD-ROM 驱动器或 DVD-ROM	从磁盘进行安装时需要相应的 DVD 驱动器
显示器	SQL Server 2012 要求有 Super-VGA（800×600）或更高分辨率的显示器

2.3.2　SQL Server 2012 的安装

安装 SQL Server 2012 数据库的步骤如下：

（1）将安装盘放入光驱，光盘会自动运行，运行界面如图 2.1 所示。

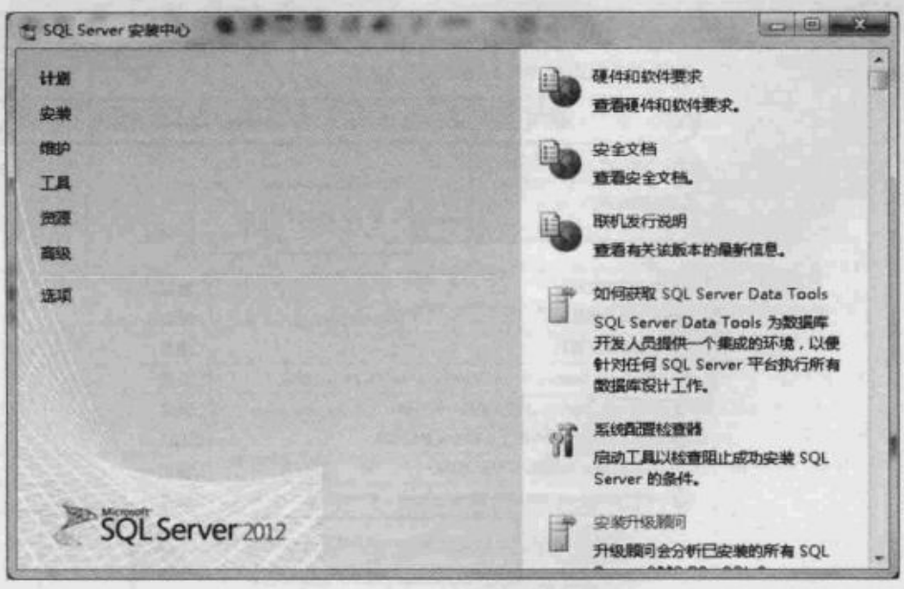


图 2.1　SQL Server 安装中心

（2）在 SQL Server 安装中心窗体中单击左侧的“安装”选项，再单击“全新 SQL Server 独立安装或向现有安装添加功能”超链接，如图 2.2 所示。

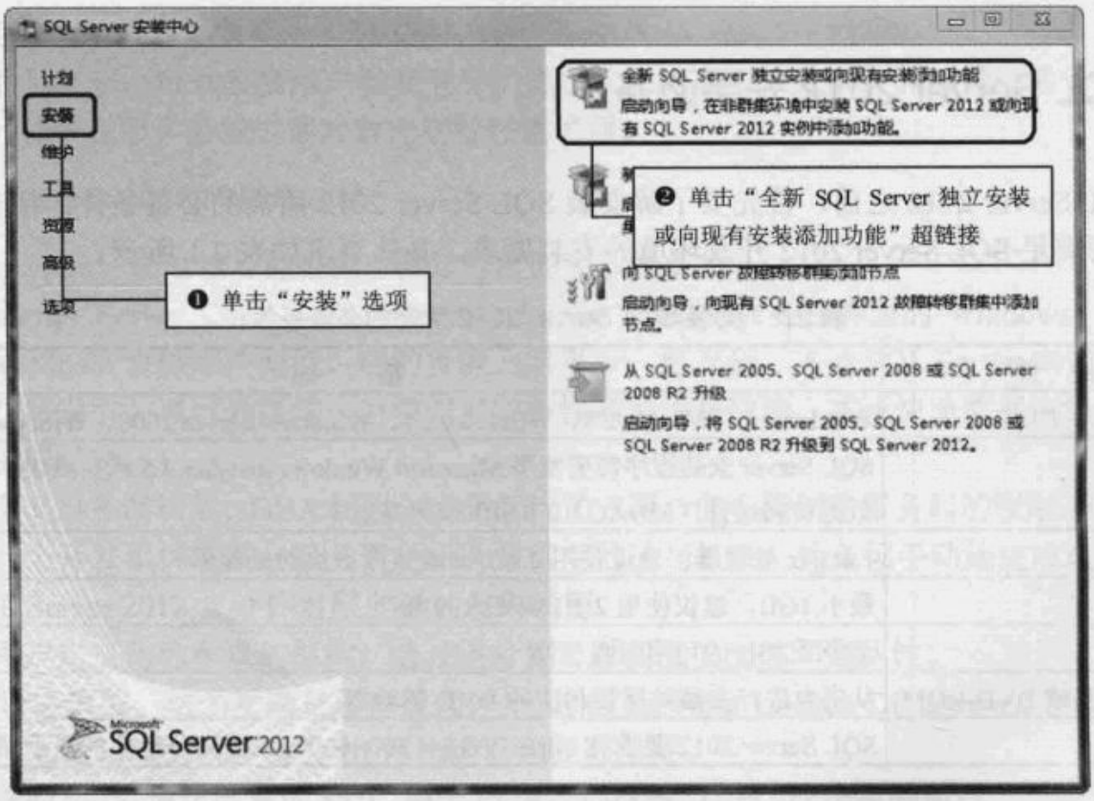


图 2.2　单击左侧的“安装”选项

（3）进入“安装程序支持规则”界面，“确定”按钮可用，如图 2.3 所示。

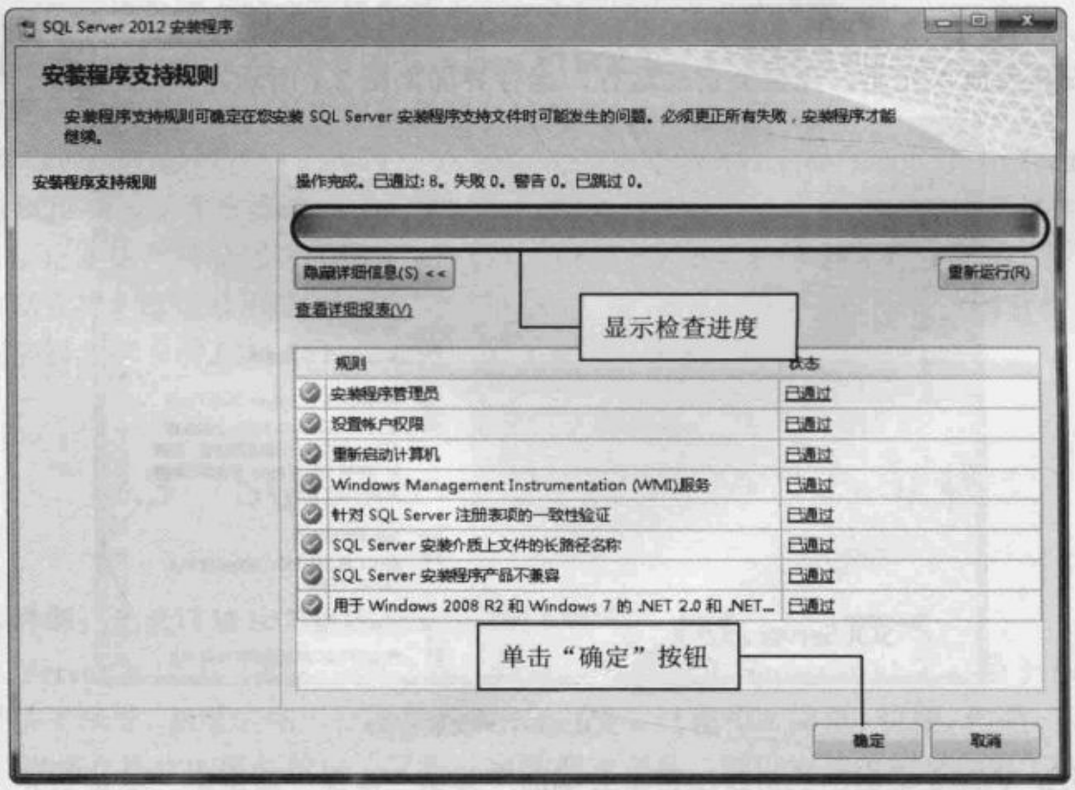


图 2.3　“安装程序支持规则”界面

（4）单击“确定”按钮，打开“产品密钥”界面，如图 2.4 所示，在该界面中输入产品密钥。

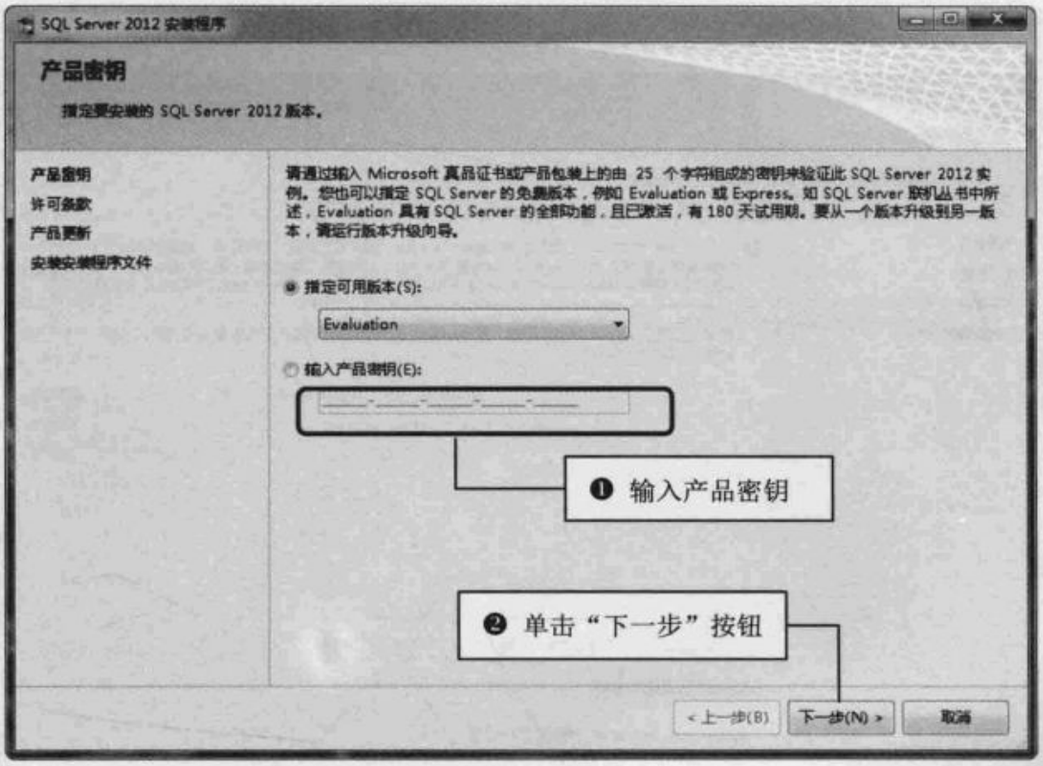


图 2.4　“产品密钥”界面

（5）单击“下一步”按钮，进入“许可条款”界面，如图 2.5 所示，选中“我接受许可条款”复选框，单击“下一步”按钮。

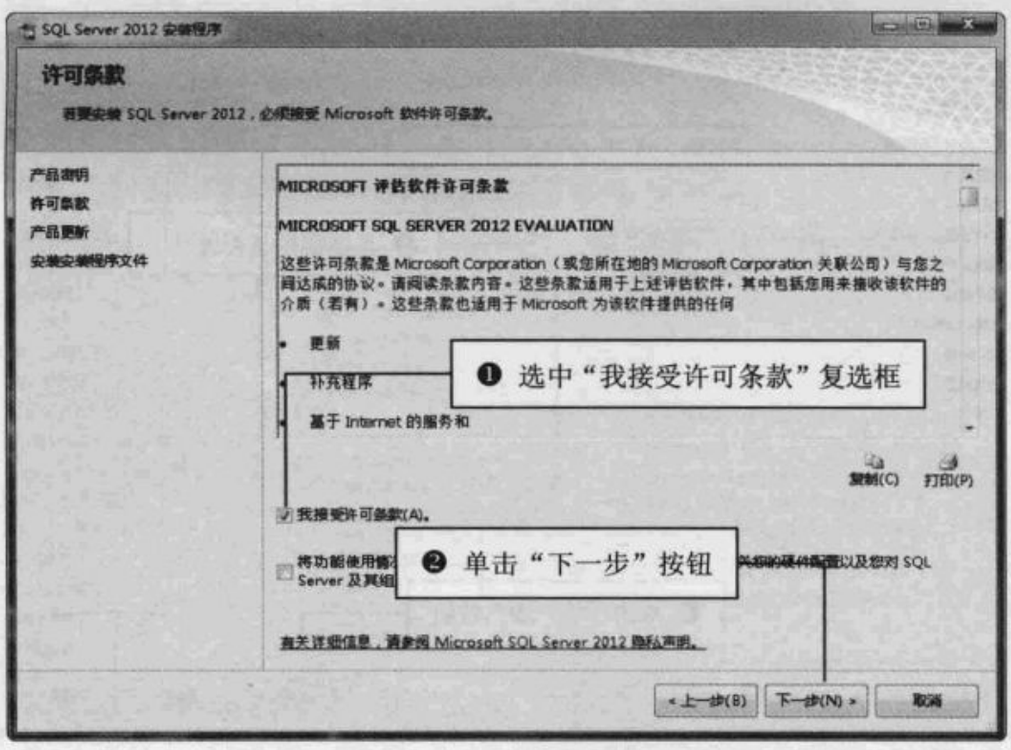


图 2.5　“许可条款”界面

（6）进入“产品更新”窗口，如图 2.6 所示，单击“下一步”按钮。

（7）进入“安装程序支持规则”界面，如图 2.7 所示，该界面中，如果所有规则都通过，则“下

一步”按钮可用。

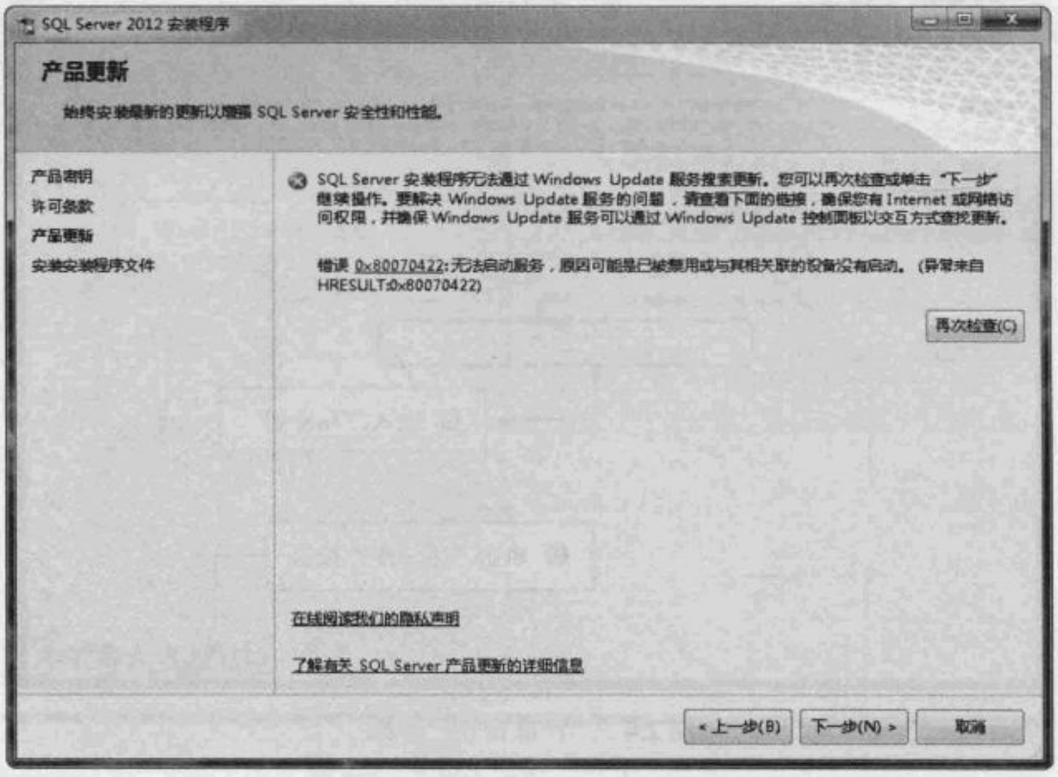


图 2.6 “产品更新”界面

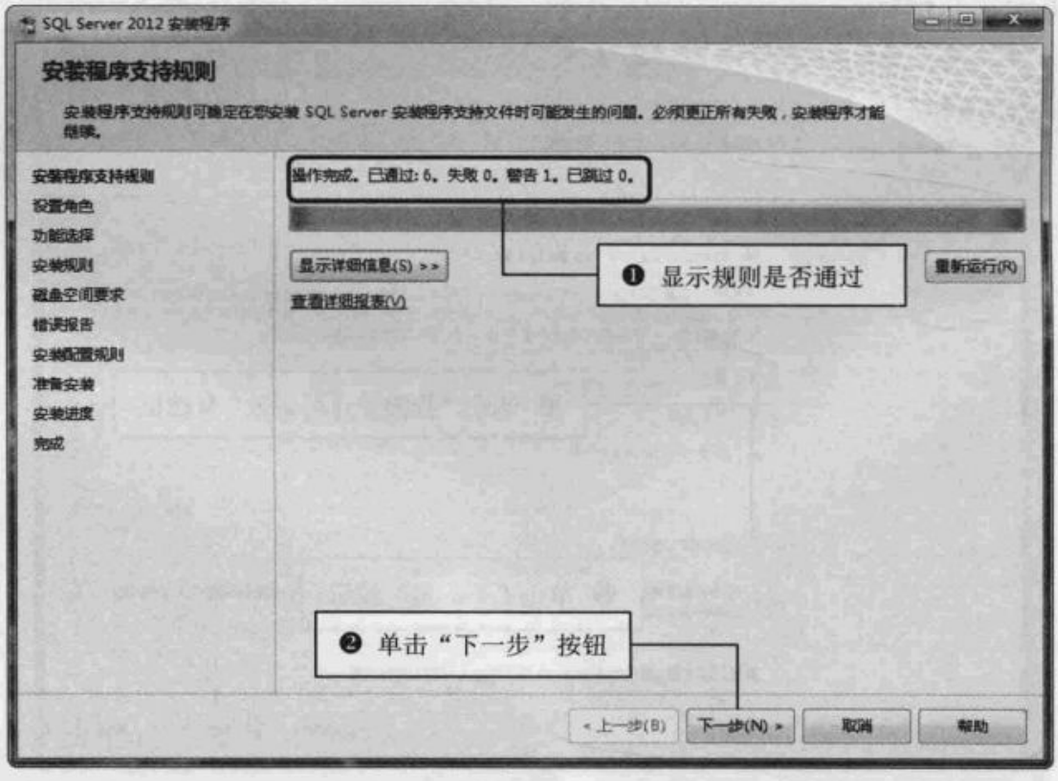


图 2.7 “安装程序支持规则”界面

（8）单击“下一步”按钮，进入“设置角色”界面，如图 2.8 所示。选中“SQL Server 功能安装”单选按钮，单击“下一步”按钮。

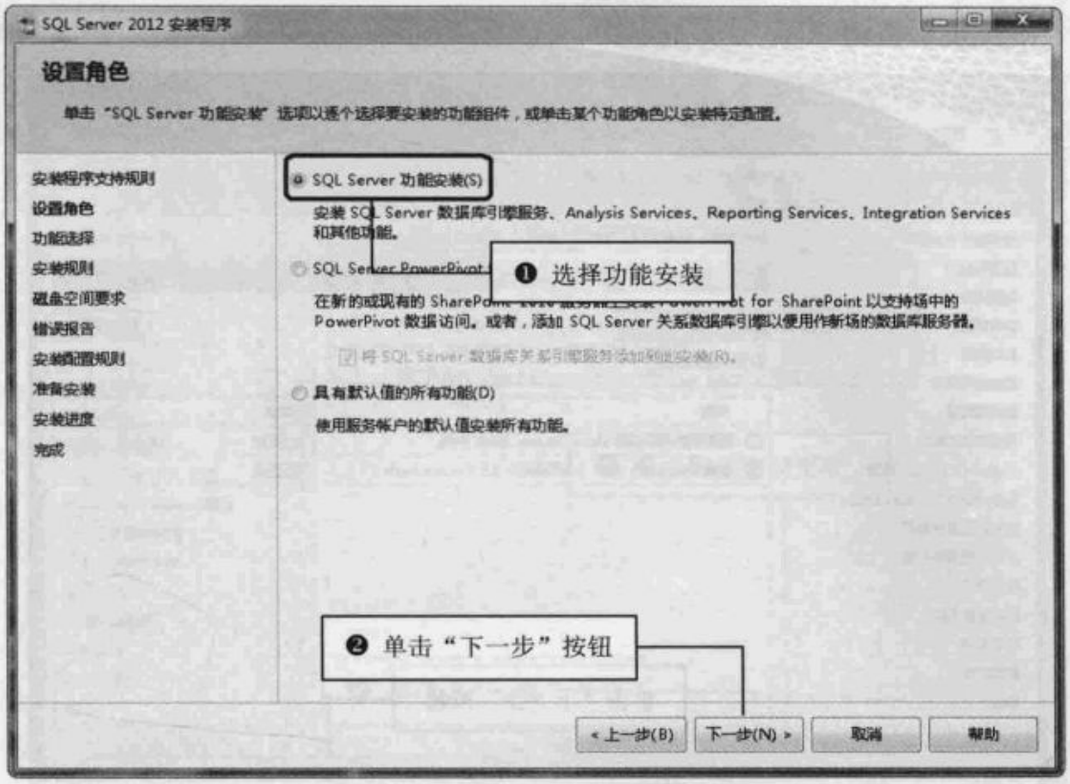


图 2.8　“设置角色”界面

（9）进入“功能选择”界面，这里可以选择要安装的功能，如果全部安装，则可以单击“全选”按钮进行选择，如图 2.9 所示。

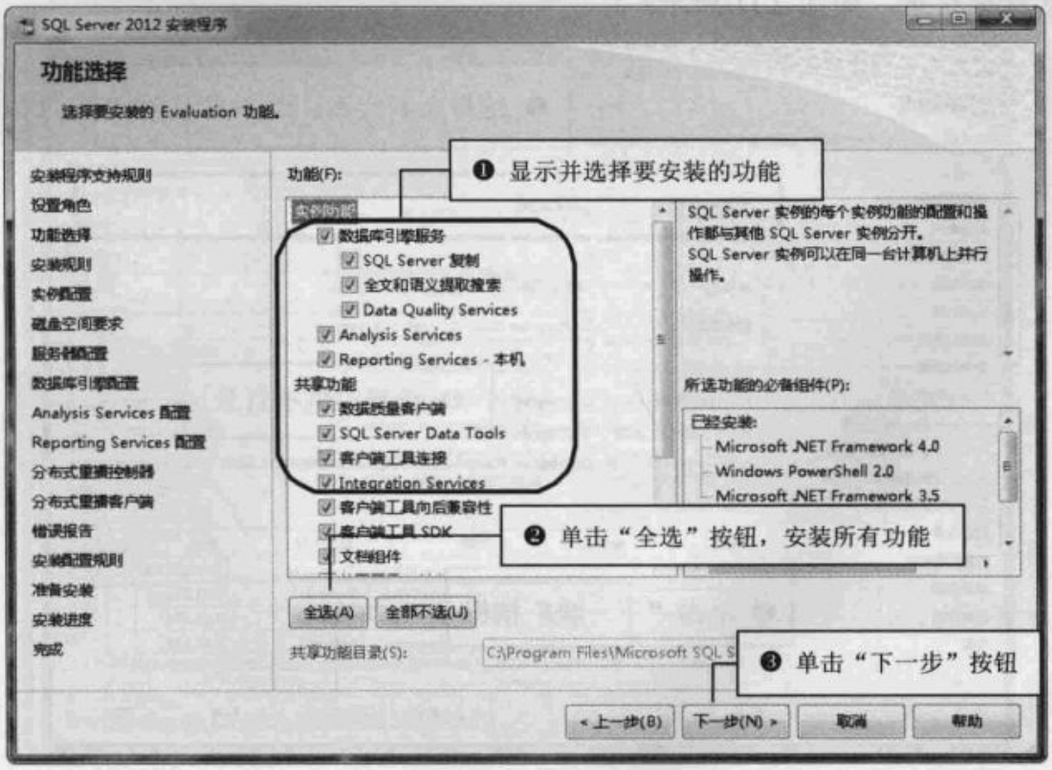


图 2.9　“功能选择”界面

（10）单击“下一步”按钮，进入“安装规则”界面，如图 2.10 所示。这里运行安装规则，操作

完成后，单击“下一步”按钮。

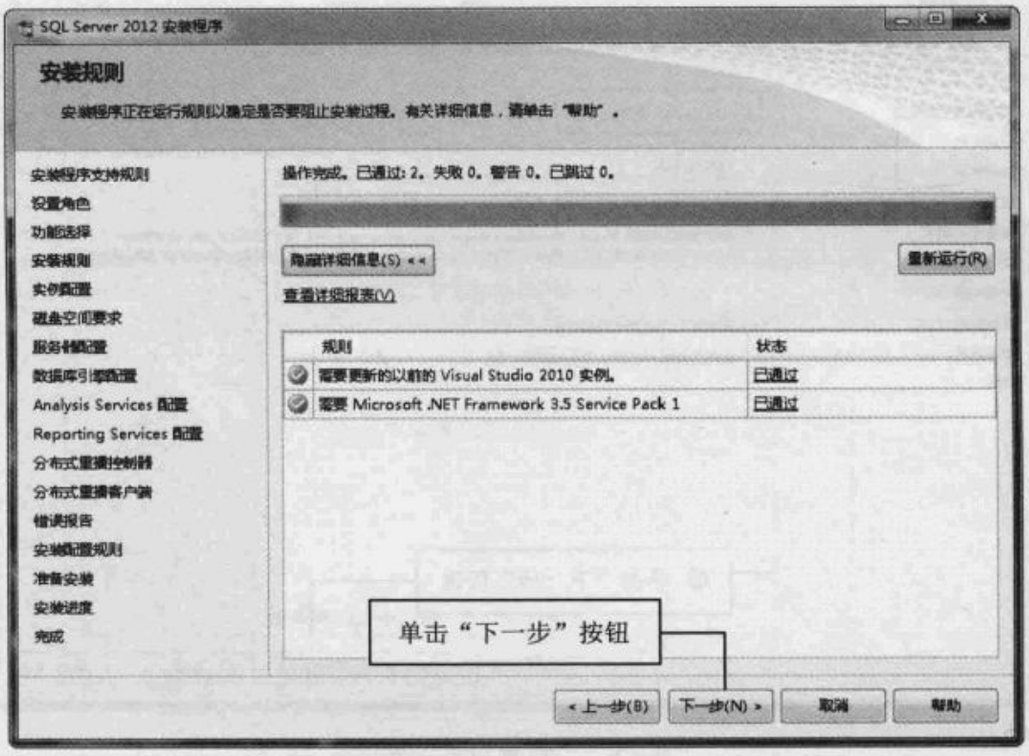


图 2.10 “安装规则”界面

（11）单击“下一步”按钮，进入“实例配置”界面，在该界面中选择实例的命名方式并命名实例，然后选择实例根目录，如图 2.11 所示。

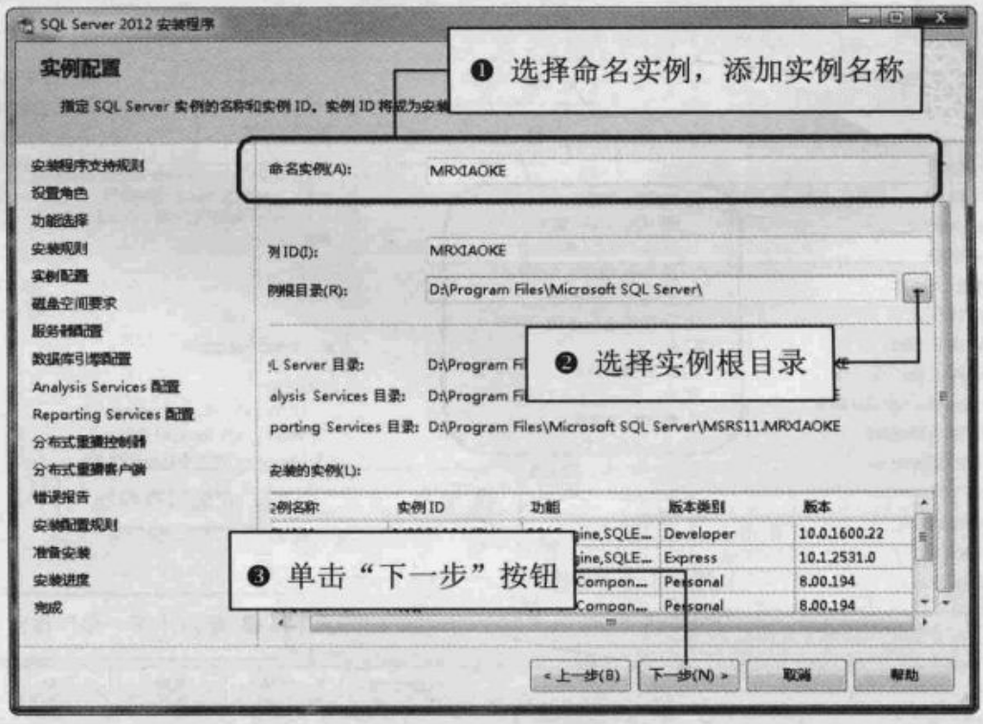


图 2.11 “实例配置”界面

（12）单击“下一步”按钮，进入“磁盘空间要求”界面，该界面中显示安装 SQL Server 2012 所

需的磁盘空间，如图 2.12 所示。

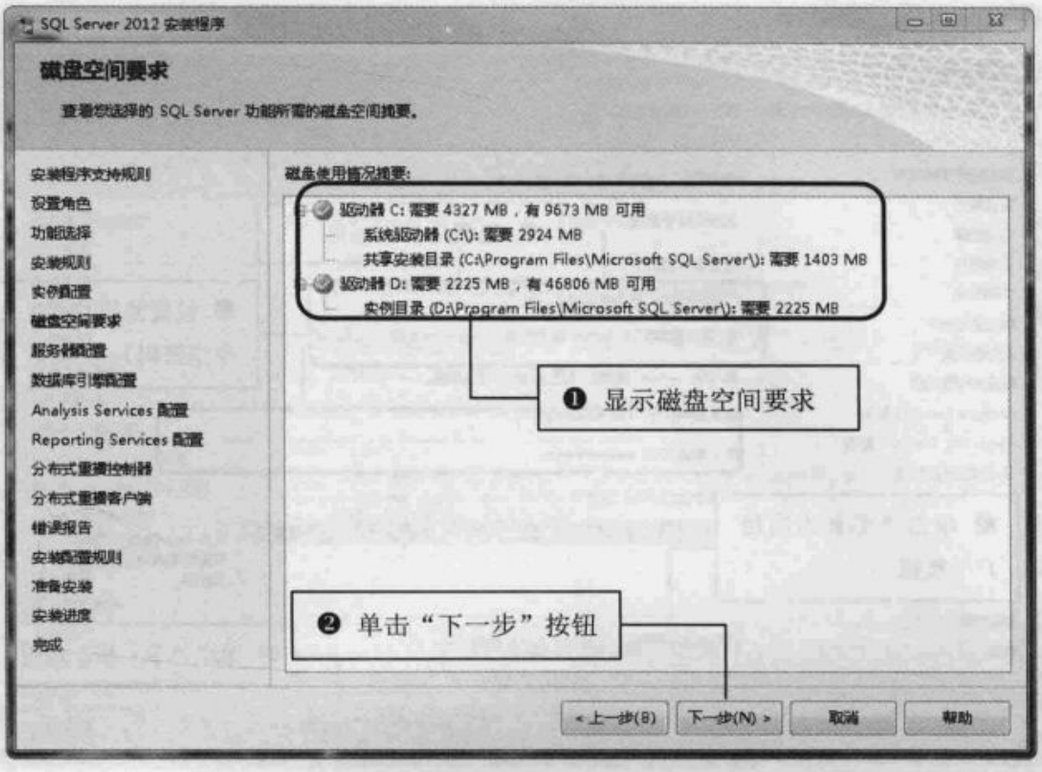


图 2.12　“磁盘空间要求”界面

（13）单击“下一步”按钮，进入“服务器配置”界面，如图 2.13 所示。

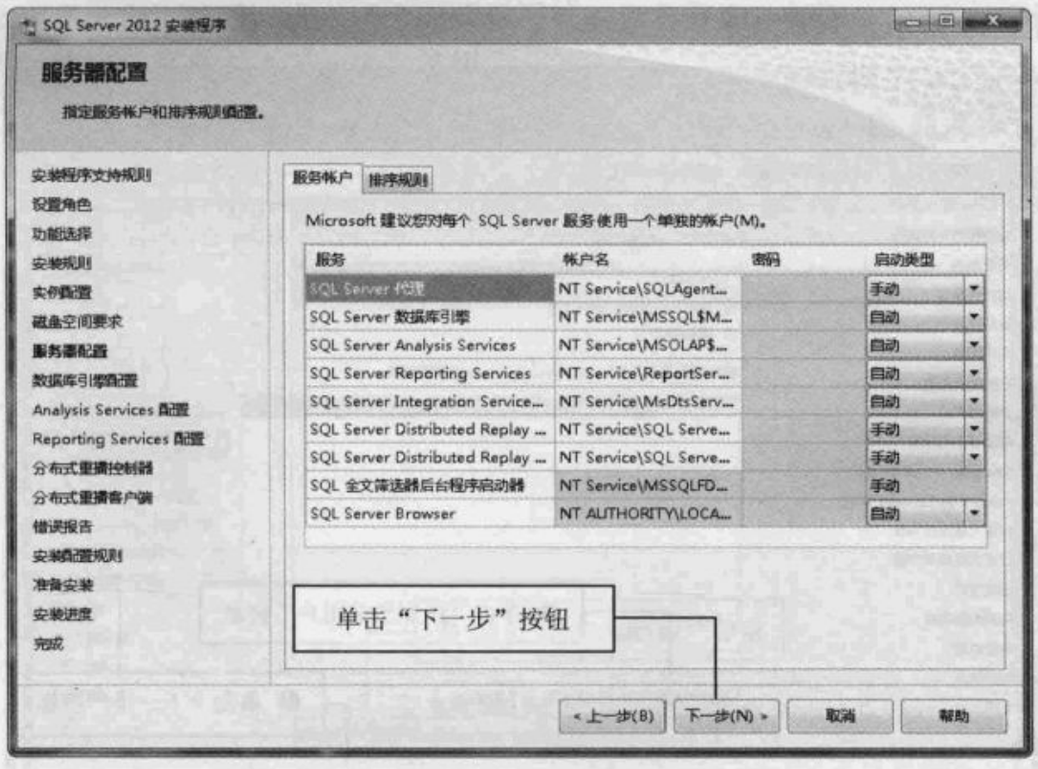


图 2.13　“服务器配置”界面

（14）单击“下一步”按钮，进入“数据库引擎配置”界面，在该界面中选择身份验证模式，并

输入密码；然后单击“添加当前用户”按钮，如图 2.14 所示。

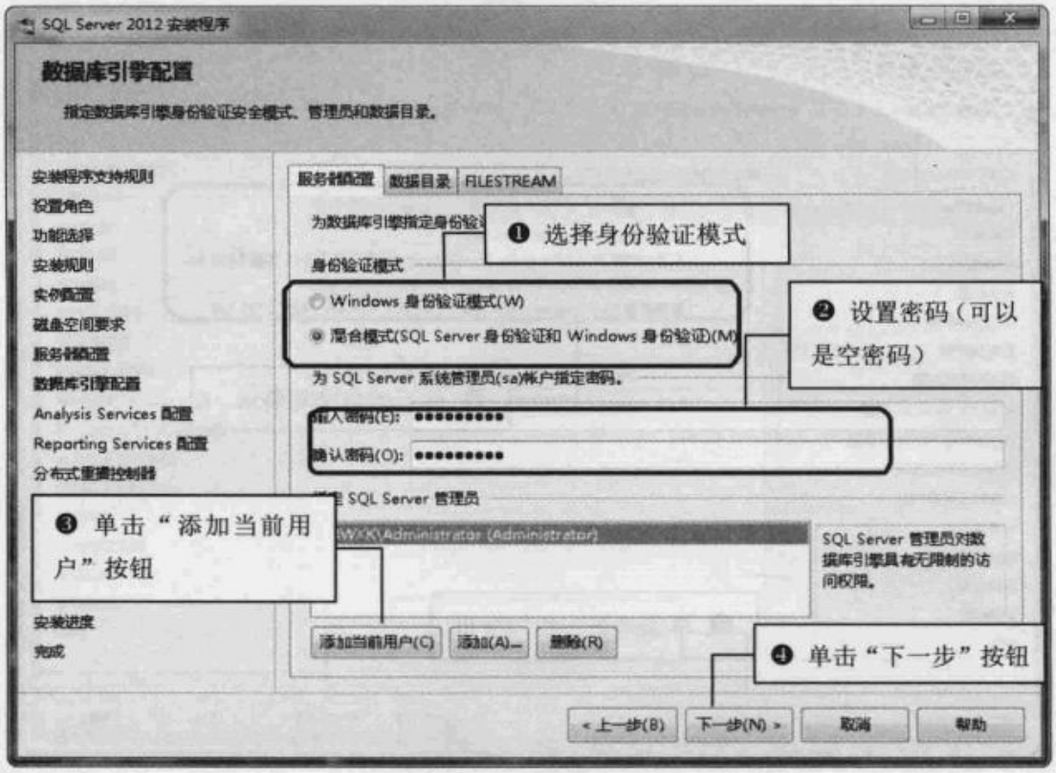


图 2.14 “数据库引擎配置”界面

（15）单击“下一步”按钮，进入“Analysis Services 配置”界面，在该界面中单击“添加当前用户”按钮，如图 2.15 所示。

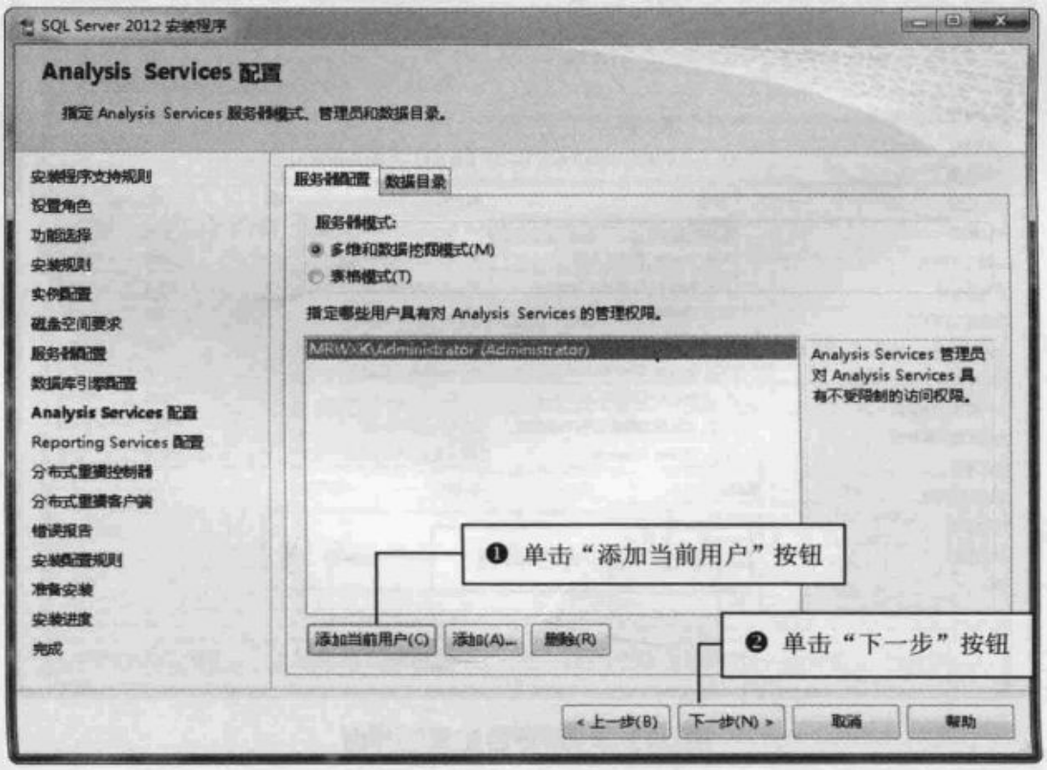


图 2.15 “Analysis Services 配置”界面

（16）单击“下一步”按钮，进入“Reporting Services 配置”界面，在该界面中选中“安装和配置”单选按钮，如图 2.16 所示。

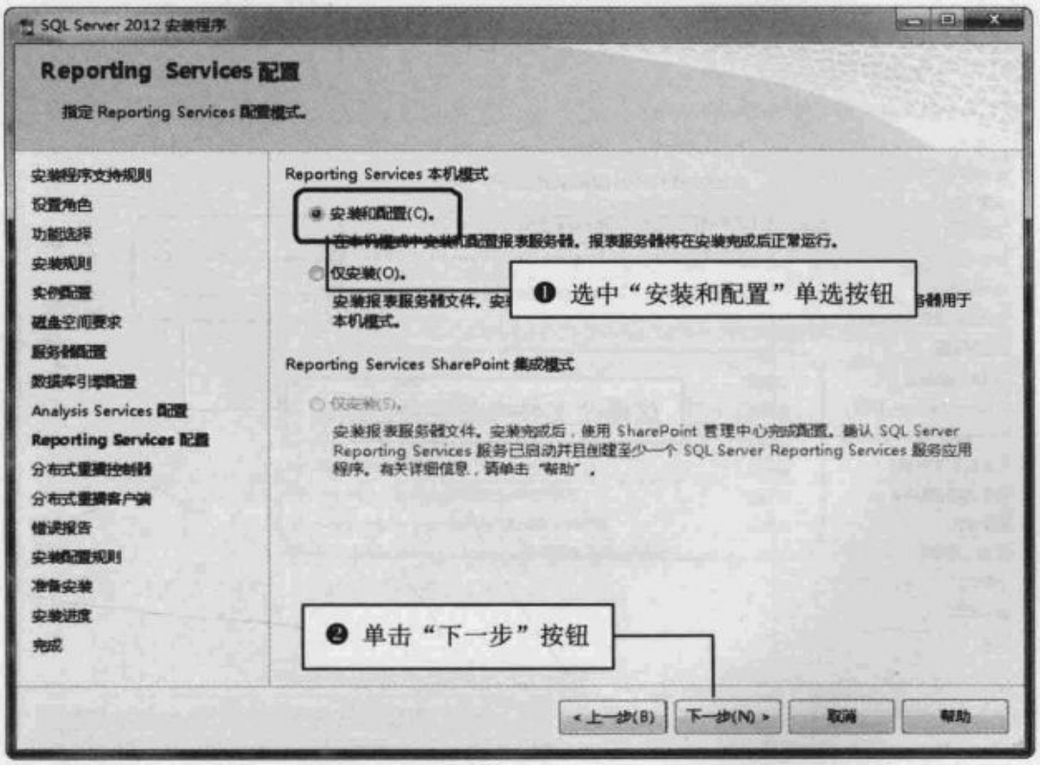


图 2.16　“Reporting Services 配置”界面

（17）单击“下一步”按钮，进入“分布式重播控制器”界面，如图 2.17 所示，在该界面中单击“添加当前用户”按钮，再单击“下一步”按钮。

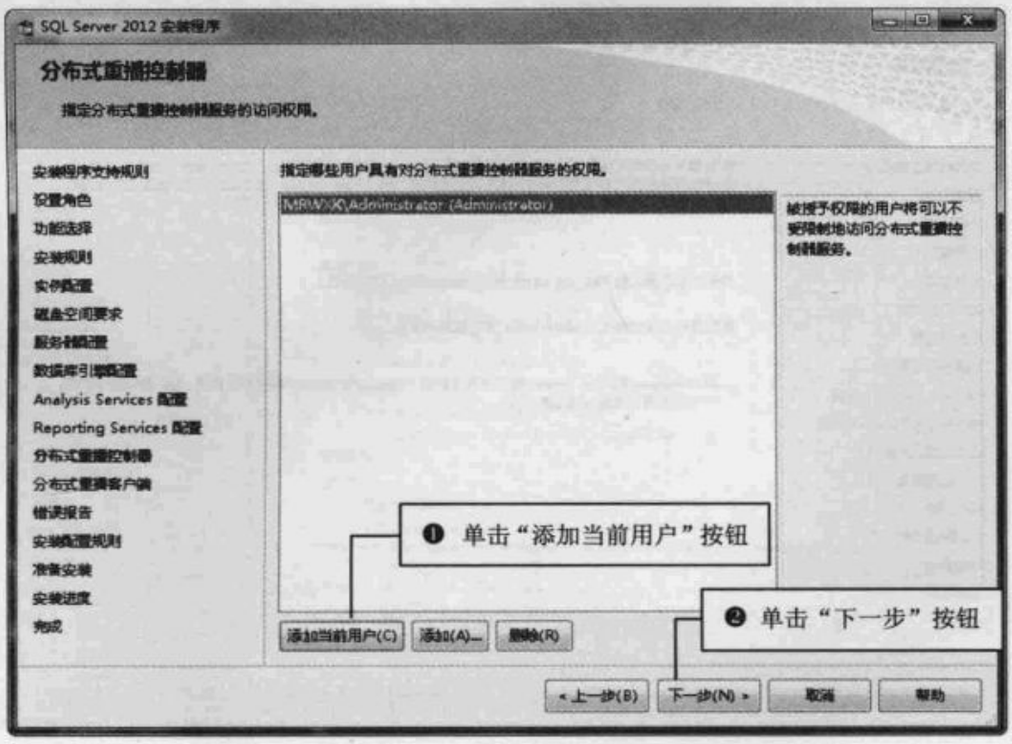


图 2.17　“分布式重播控制器”界面

（18）单击“下一步”按钮，进入“分布式重播客户端”界面，如图 2.18 所示（该界面中，控制器名称后面的文件夹需要自己创建）。

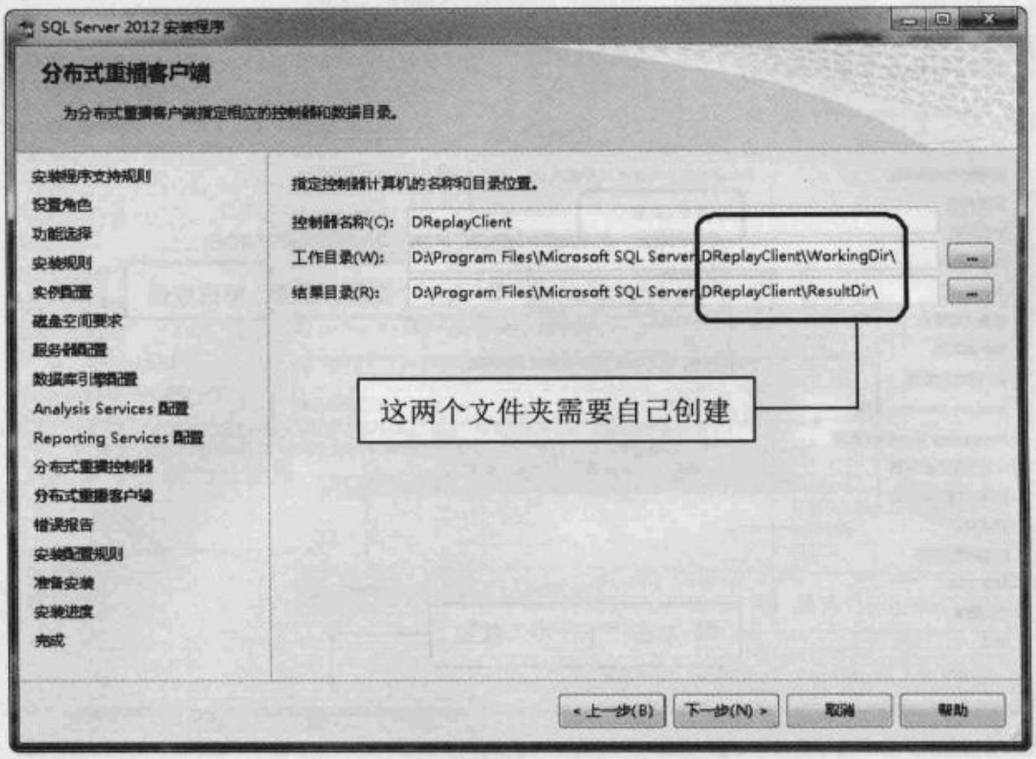


图 2.18 “分布式重播客户端”界面

（19）单击“下一步”按钮，进入“错误报告”界面，如图 2.19 所示，该界面中若没有错误，单击“下一步”按钮。

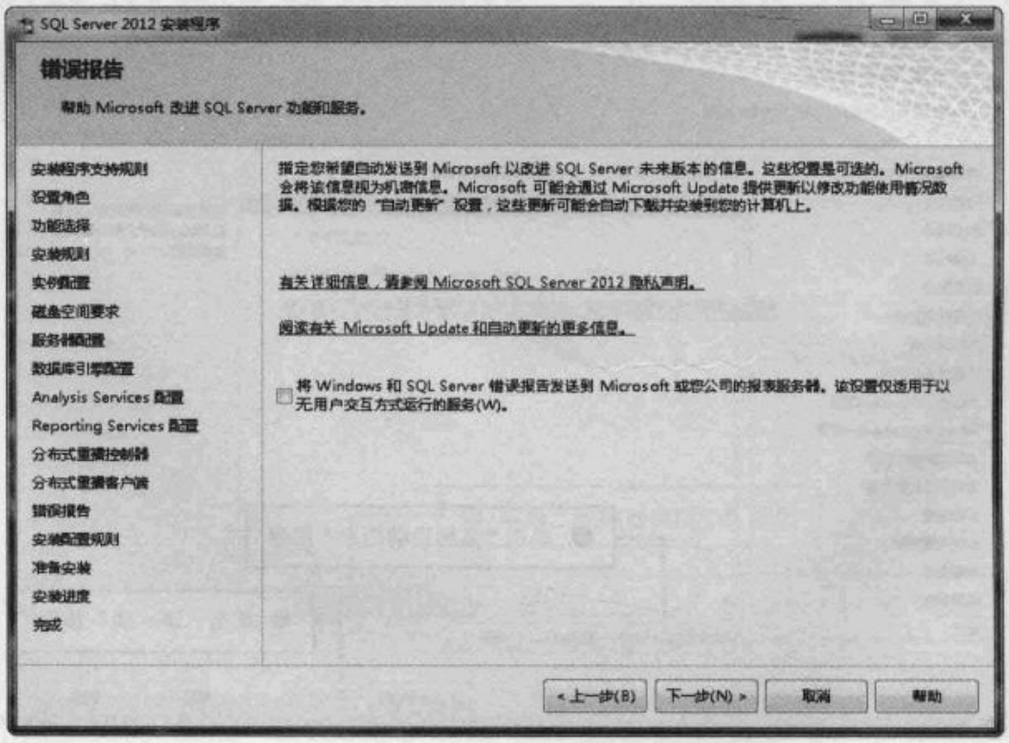


图 2.19 “错误报告”界面

（20）单击“下一步”按钮，进入“安装配置规则”界面，如图 2.20 所示，该界面中显示配置规则的安装情况，若全部通过，可单击“下一步”按钮。

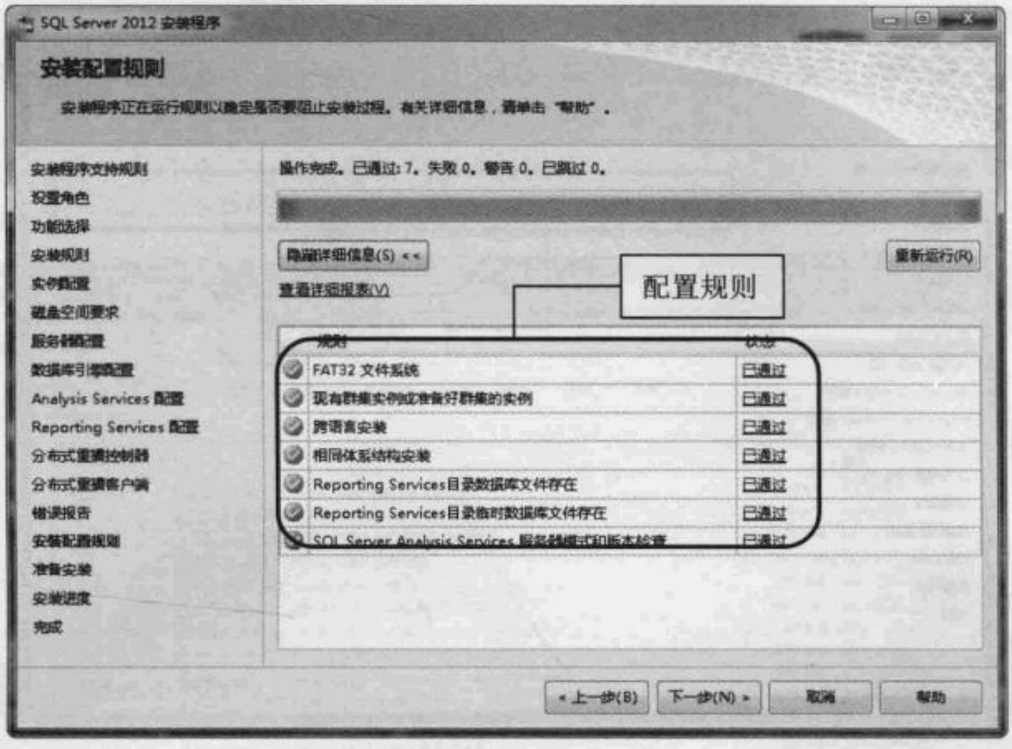


图 2.20　“安装配置规则”界面

（21）单击“下一步”按钮，进入“准备安装”界面，如图 2.21 所示，该界面中显示了准备安装的 SQL Server 2012 功能。

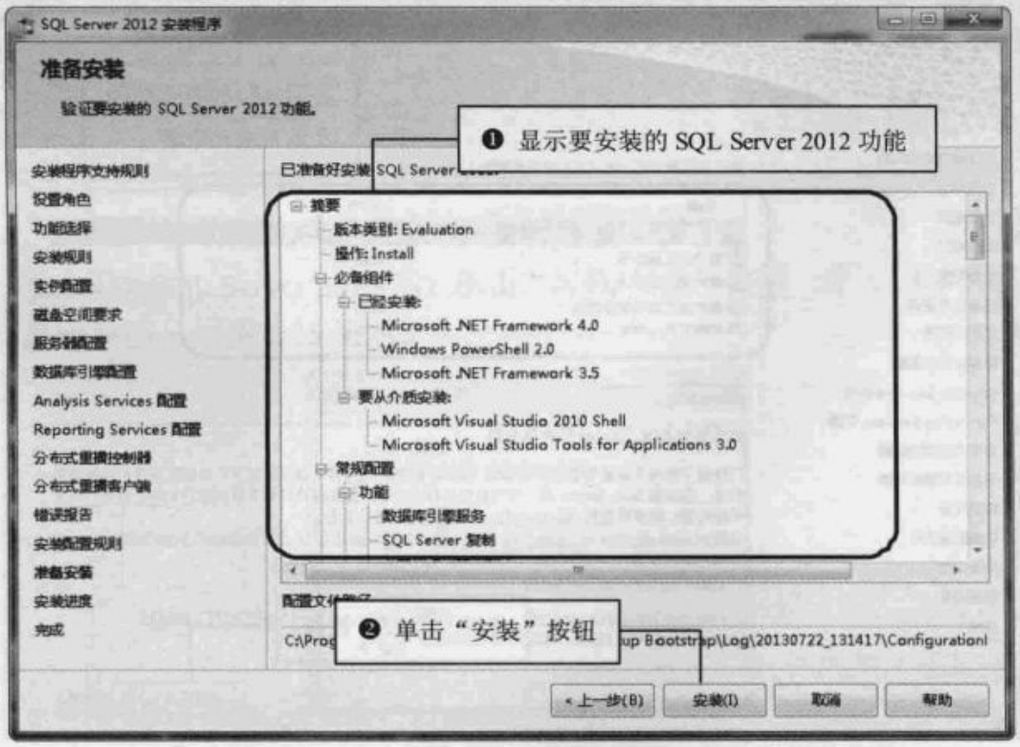


图 2.21　“准备安装”界面

（22）单击“安装”按钮，进入“安装进度”界面，如图 2.22 所示，该界面中显示 SQL Server 2012 的安装进度。

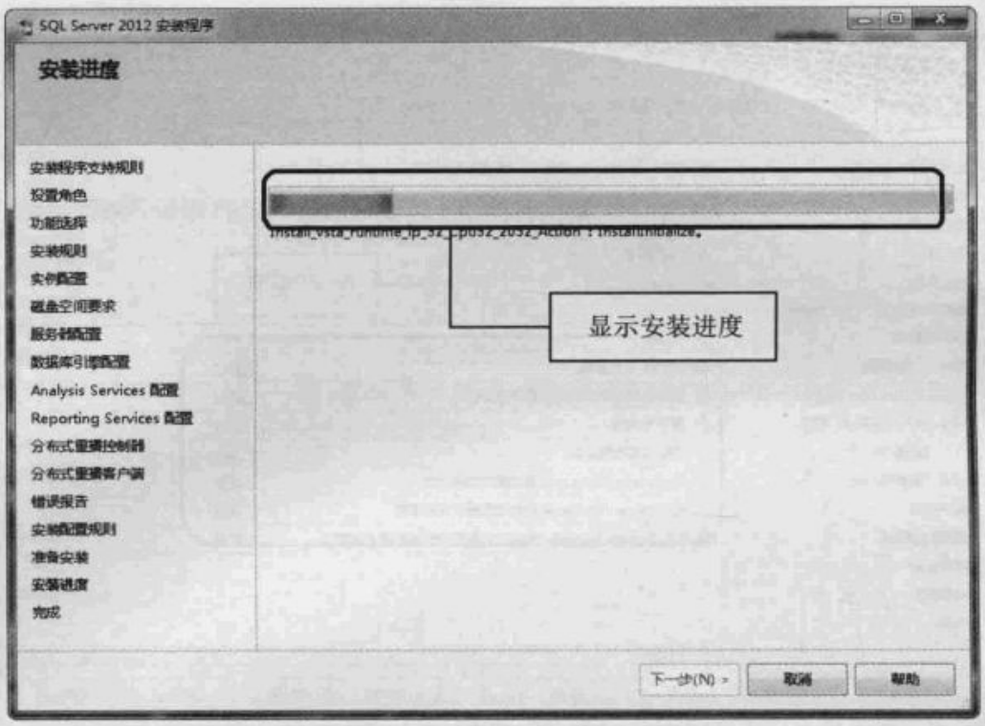


图 2.22 “安装进度”窗口

（23）单击“下一步”按钮，进入“完成”窗口，如图 2.23 所示，单击“关闭”按钮，即可完成 SQL Server 2012 的安装。

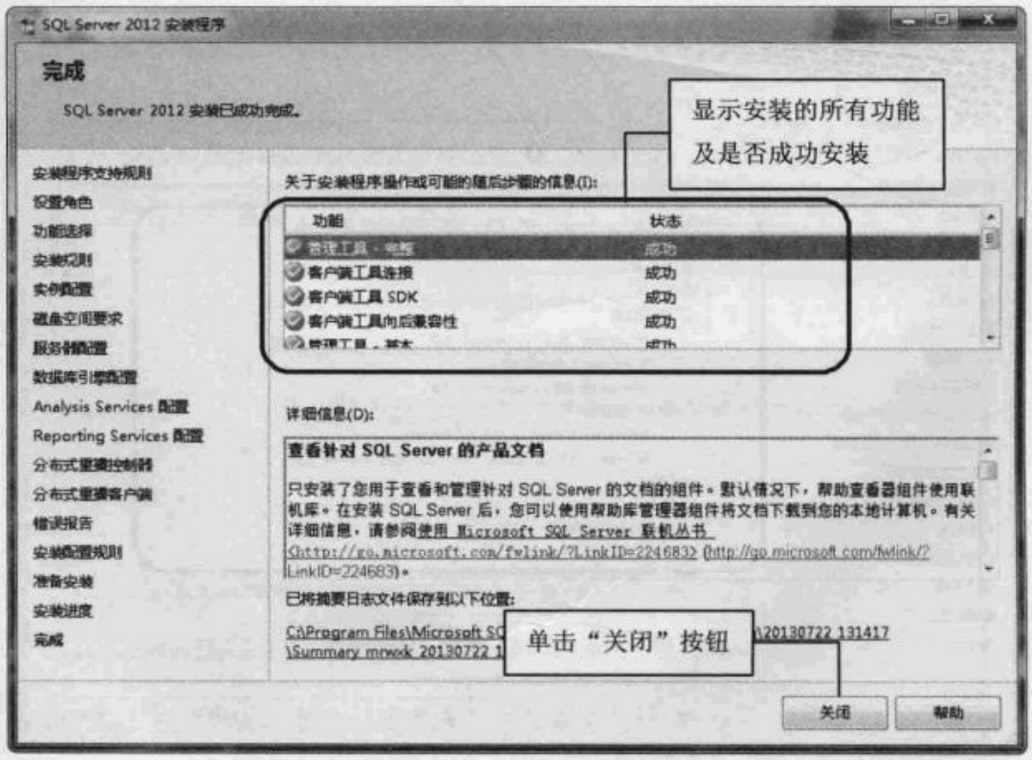


图 2.23 “完成”界面

2.3.3　SQL Server 2012 的卸载

如果 SQL Server 2012 被损坏而导致无法使用时，读者可以将其卸载。卸载 SQL Server 2012 的步骤如下：

（1）在 Windows 7 操作系统中，打开“控制面板”/“程序”/“程序和功能”，在打开的窗口中选中 Microsoft SQL Server 2012，如图 2.24 所示。

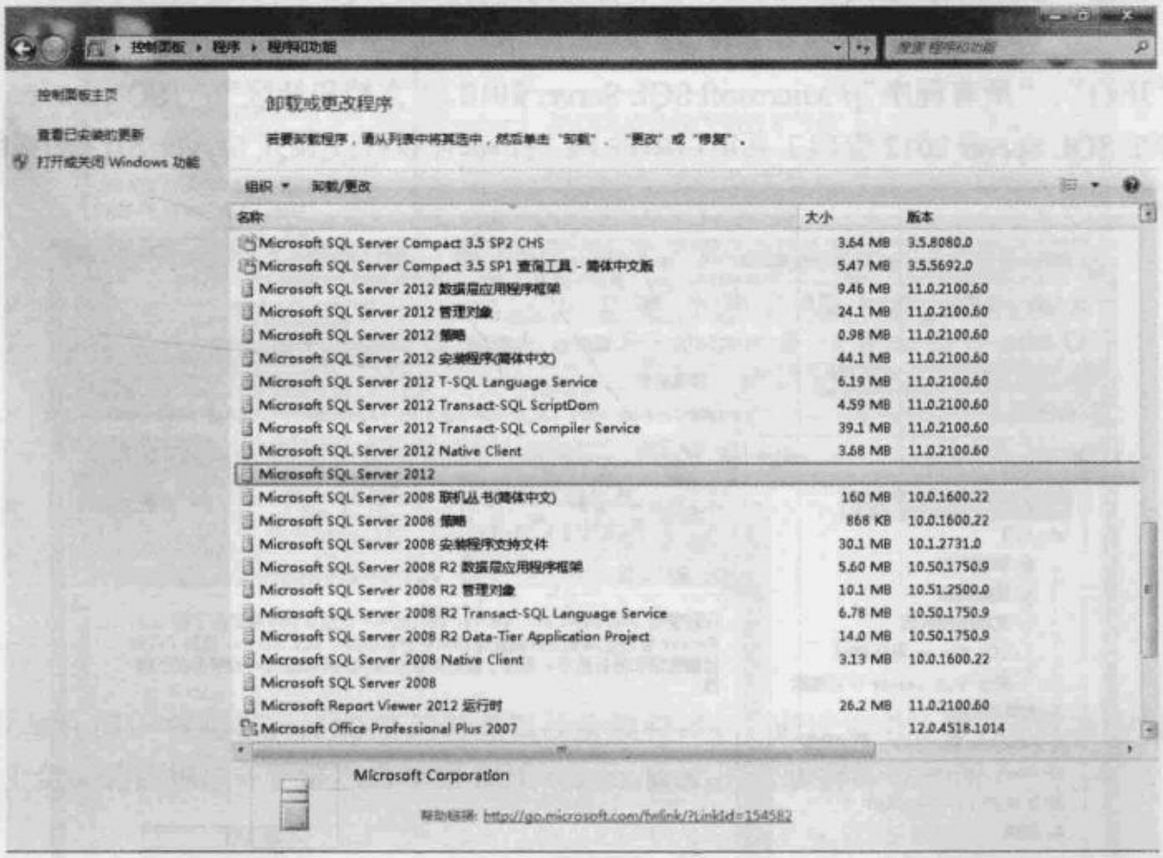


图 2.24　添加或删除程序

（2）选中 Microsoft SQL Server 2012 后，单击“卸载/更改”按钮，进入 Microsoft SQL Server 2012 的添加、修复和删除页面，如图 2.25 所示。

图 2.25　Microsoft SQL Server 2012 添加、修复和删除页面

（3）单击“删除”按钮，即可根据向导卸载 SQL Server 2012 数据库。

2.4 使用 SQL Server 2012 的帮助

与微软的其他产品一样，SQL Server 2012 在安装时也提供了安装帮助文档系统。通过帮助文档可以帮助用户使用、维护及管理 SQL Server。本节将介绍如何使用 SQL Server 2012 的帮助。具体操作步骤如下：

选择“开始”/“所有程序”/ Microsoft SQL Server 2012 /“文档和社区”/“SQL Server 联机丛书”命令，或者在 SQL Server 2012 管理工具窗口中按 F1 键，都可以打开 SQL Server 2012 的帮助文档，主界面如图 2.26 所示。

图 2.26 SQL Server 的帮助文档

SQL Server 2012 的帮助文档还为使用者提供了一种强大的搜索功能，在工具栏中单击“索引”按钮，在“查找”文本框中输入要查找的内容提要，搜索的结果将自动以概要的方式呈现在主界面的左侧，开发人员可以根据自己的需要选择不同的文档进行阅读，例如查找 CREATE，其使用示意图如图 2.27 所示。

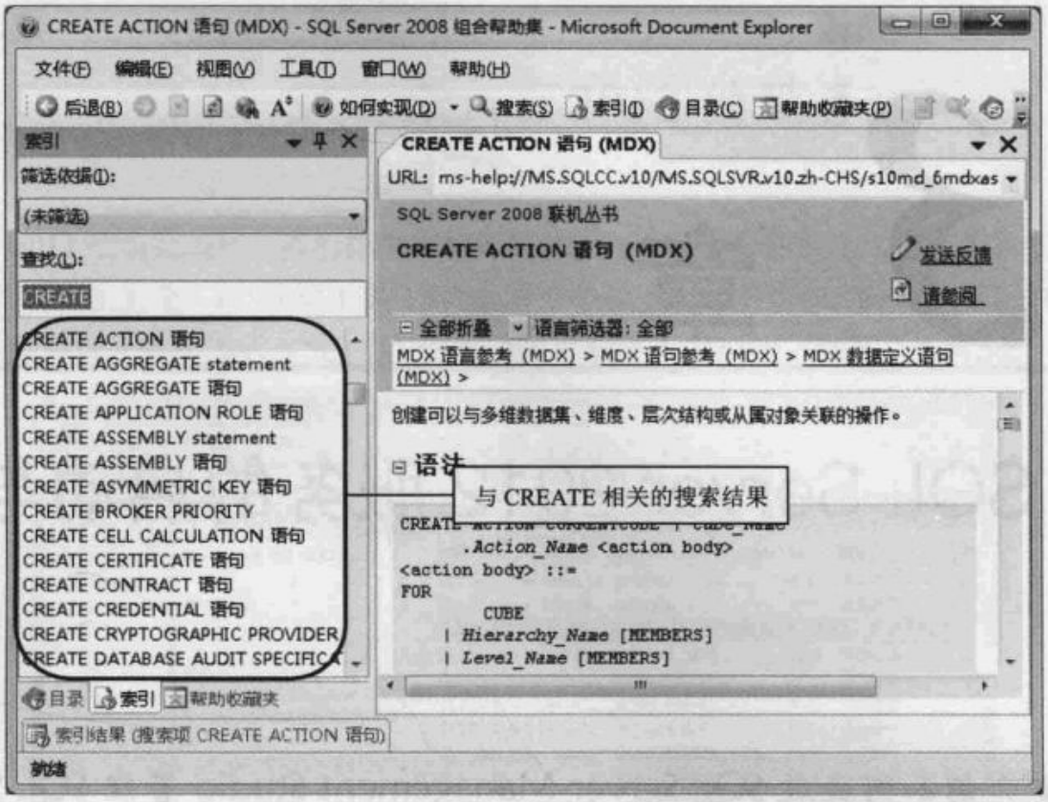


图 2.27　SQL Server 2012 的搜索功能

2.5　小　　结

本章主要介绍了 SQL Server 2012 的特点以及安装和卸载。通过本章的学习，读者将对 SQL Server 2012 有一个全面的认识，并能够熟练掌握 SQL Server 2012 的安装及卸载过程。

2.6　实践与练习

1. SQL Server 2012 都有哪些版本？（**答案位置：光盘\TM\sl\2\1**）
2. 简单描述 SQL Server 2012 的主要特点。（**答案位置：光盘\TM\sl\2\2**）

第 3 章

SQL Server 2012 服务的启动与注册

本章主要介绍如何通过 SQL Server Management Studio 管理 SQL Server 2012 数据库，其中主要包括启动 SQL Server 2012 的服务、注册 SQL Server 2012 数据库等内容，这些内容相对于后面章节中所讲解的内容而言是比较简单的，但是为了开发人员更加熟练地使用 SQL Server 2012 的数据库，学习这些内容又是必不可少的。

通过阅读本章，您可以：

- 熟悉如何启动 SQL Server 2012 的服务
- 掌握服务器组的创建与删除
- 掌握服务器的注册与删除

3.1　SQL Server 2012 的服务

SQL Server 2012 安装完成后，其所提供的服务都体现在系统服务的后台，这些服务可以在“开始”/“控制面板”/“系统和安全”/“管理工具”/“服务”中找到，而且 SQL Server 2012 的每个后台服务都代表一个或一组进程。SQL Server 2012 在后台中的“服务”窗口中，如图 3.1 所示。

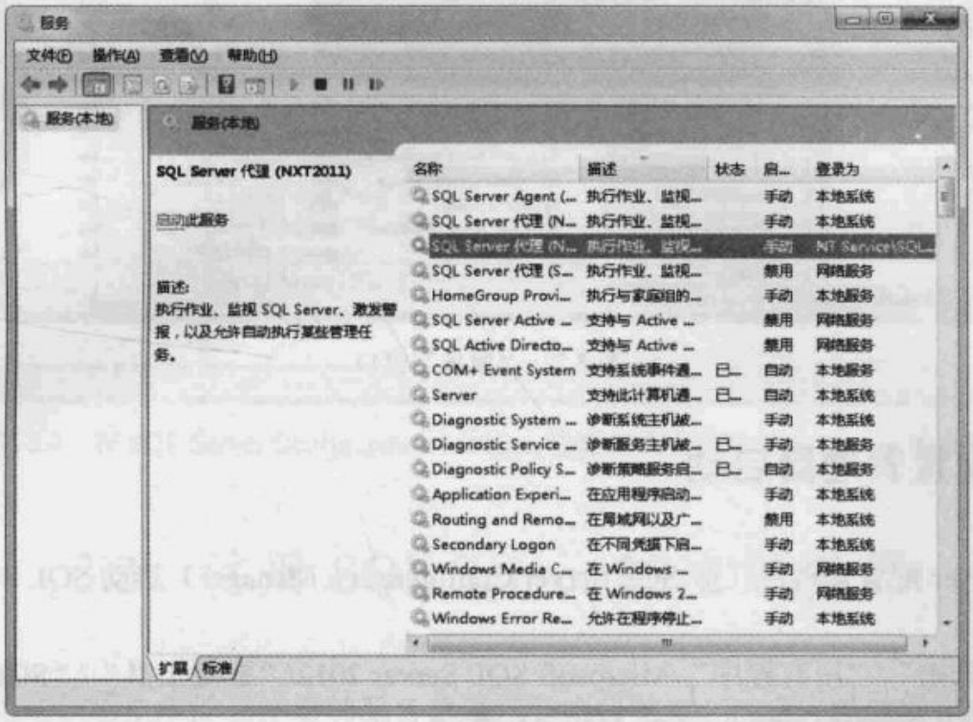


图 3.1　SQL Server 2012 在后台中的“服务”窗口

3.2　启动 SQL Server 2012 服务

3.2.1　后台启动服务

后台启动 SQL Server 2012 服务的操作步骤如下：

（1）选择“开始”/“控制面板”/“系统和安全”/“管理工具”/“服务”命令，打开“服务”窗口。

（2）在“服务”窗口中找到需要启动的 SQL Server 2012 服务，单击鼠标右键，弹出的快捷菜单如图 3.2 所示。

（3）在弹出的快捷菜单中选择“启动”命令，等待 Windows 启动 SQL Server 2012 的服务。

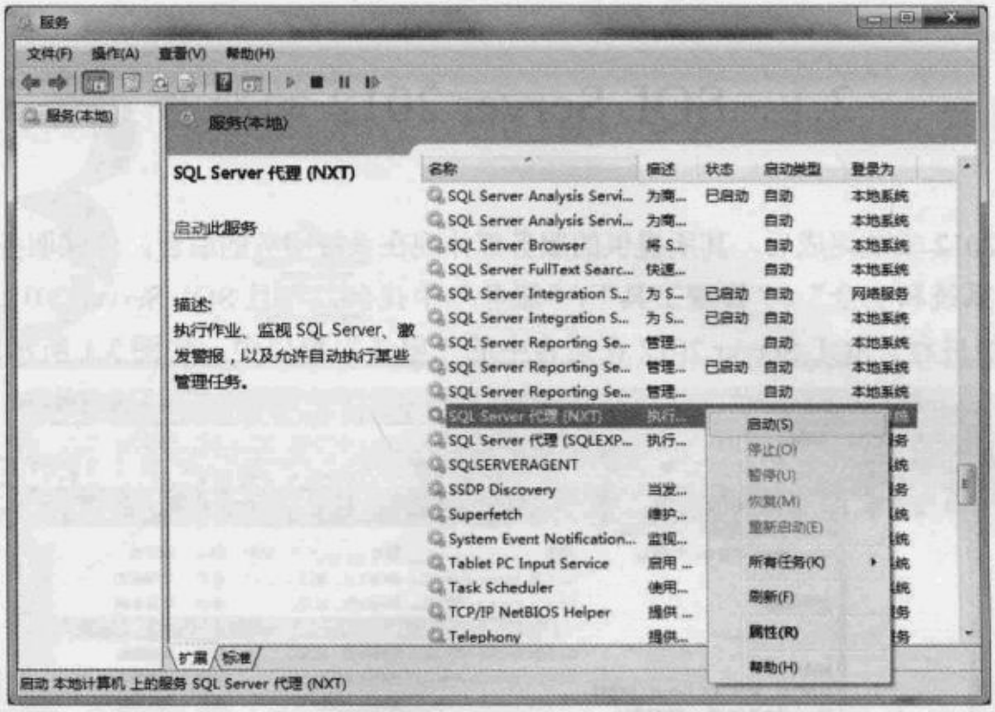


图 3.2 “服务”窗口

3.2.2 通过配置管理器启动

通过 SQL Server 配置管理器（即 SQL Server Configuration Manager）启动 SQL Server 2012 服务的步骤如下：

（1）选择“开始”/“所有程序”/Microsoft SQL Server 2012/“配置工具”/“SQL Server 配置管理器”命令，打开 SQL Server Configuration Manager 管理工具。

（2）选择 SQL Server Configuration Manager 管理工具中左边树状结构下的“SQL Server 服务”，这时右边将显示 SQL Server 中的服务，如图 3.3 所示。

图 3.3 SQL Server Configuration Manager 管理工具

（3）在 SQL Server Configuration Manager 管理工具右边列出的 SQL Server 服务中选择需要启动的服务，单击鼠标右键，在弹出的快捷菜单中选择“启动”命令，启动所选中的服务，如图 3.4 所示。

图 3.4　在 SQL Server Configuration Manager 管理工具中启动 SQL Server 的服务

3.3　注册 SQL Server 2012 服务器

创建服务器组可以将众多的已注册的服务器进行分组化管理。而通过注册服务器，可以储存服务器连接的信息，以供在连接该服务器时使用。

3.3.1　创建与删除服务器组

1．创建服务器组

使用 SQL Server 2012 创建服务器组的步骤如下：

（1）选择“开始”/“所有程序”/ Microsoft SQL Server 2012/SQL Server Management Studio 菜单，打开 SQL Server Management Studio 工具的“连接服务器”对话框。

（2）单击“连接服务器”对话框中的“取消”按钮，如图 3.5 所示。

（3）执行 SQL Server Management Studio 中的“视图”/“已注册的服务器”菜单命令，将“已注册的服务器”面板添加到 SQL Server Management Studio 中。添加“已注册的服务器”面板后的 SQL Server Management Studio 如图 3.6 所示。

（4）在“已注册的服务器”面板中选择服务器组要创建在哪种服务器类型当中。服务器类型如表 3.1 所示。

图 3.5 “连接到服务器”对话框

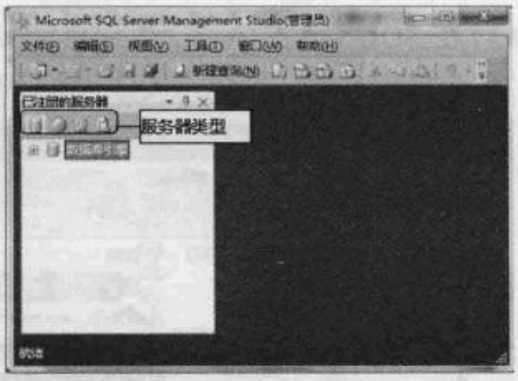


图 3.6 Microsoft SQL Server Management Studio

表 3.1 “已注册的服务器”面板中服务器的类型

图　标	服务器类型	图　标	服务器类型
	数据库引擎		Reporting Services
	Analysis Services		Integration Services

（5）选择完服务器后，在“已注册的服务器”面板的显示服务器区域中选择“本地服务器组”，单击鼠标右键，在弹出的快捷菜单中选择“新建服务器组”命令，如图 3.7 所示。

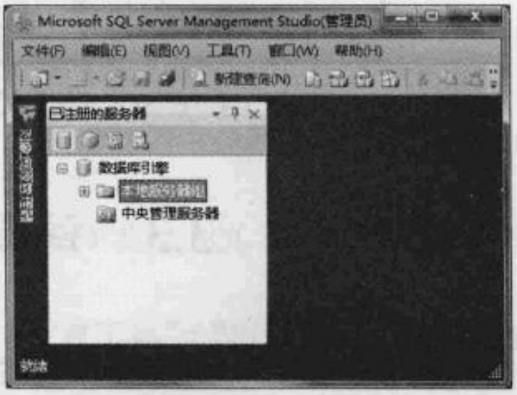


图 3.7 新建服务器组菜单

（6）在弹出的“新建服务器组属性”对话框中的“组名”文本框中输入要创建服务器组的名称，在“组说明”文本框中写入关于创建的这个服务器组的简要说明，如图 3.8 所示。信息输入完毕后，单击“确定”按钮即可完成服务器组的创建。

2. 删除服务器组

使用 SQL Server 2012 删除服务器组的步骤如下：

（1）按照“创建服务器组”一节中打开“已注册的服务器”的步骤，打开“已注册的服务器”页面。

（2）选择需要删除的服务器组，单击鼠标右键，在弹出的快捷菜单中选择“删除”命令，如图 3.9 所示。

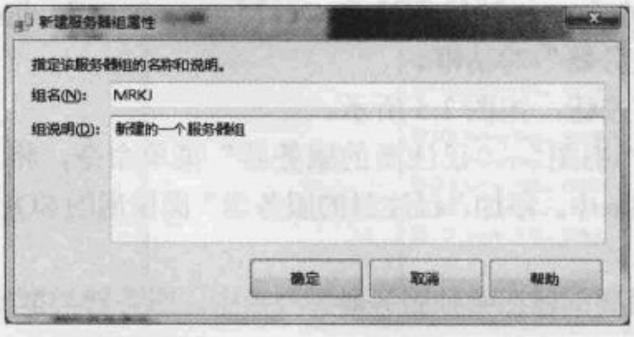


图 3.8 “新建服务器组属性”对话框

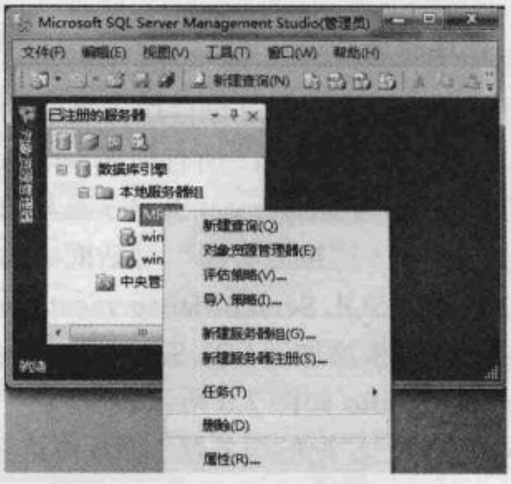


图 3.9 删除服务器组

（3）在弹出的“确认删除”对话框中单击“是”按钮，即可完成服务器组的删除，如图 3.10 所示。

注意

在删除服务器组的同时，也会将该组内所注册的服务器一同删除。

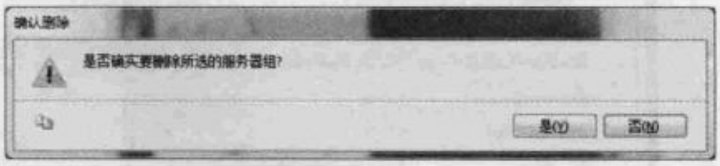


图 3.10　“确认删除”对话框

3.3.2　注册与删除服务器

服务器是计算机的一种，它是网络上一种为客户端计算机提供各种服务的高性能的计算机，它在网络操作系统的控制下，也能为网络用户提供集中计算、信息发表及数据管理等服务。本节将讲解如何注册服务器及删除服务器。

1．注册服务器

使用 SQL Server 2012 注册服务器的步骤如下：

（1）按照 3.3.1 节中打开“已注册的服务器”的步骤，打开“已注册的服务器”页面。

（2）选择完服务器后，在“已注册的服务器”页面的显示服务器区域中选择“SQL Server 组”，单击鼠标右键，在弹出的快捷菜单中选择“新建服务器注册”命令，如图 3.11 所示。

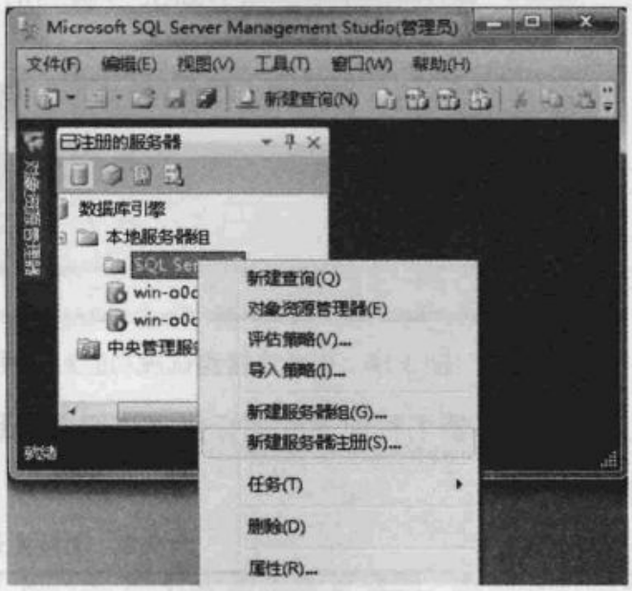


图 3.11　“新建服务器注册”菜单命令

（3）弹出“新建服务器注册”对话框，在该对话框中有“常规”与“连接属性”两个选项卡。

- ☑ “常规”选项卡：包括服务器类型、服务器名称、登录时身份验证的方式、登录所用的用户名、密码、已注册的服务器名称、已注册的服务器说明等设置信息。“新建服务器注册”对话框的“常规”选项卡如图 3.12 所示。
- ☑ “连接属性”选项卡：包括所要连接服务器中的数据库、连接服务器时使用的网络协议、发送的网络数据包的大小、连接时等待建立连接的秒数、连接后等待任务执行的秒数等设置信息。“连接属性”选项卡如图 3.13 所示。

设置完成这些信息后，单击“测试”按钮，测试与所注册服务器的连接，如果成功连接，则弹出如图 3.14 所示的对话框。

单击“确定”按钮后，在弹出的“新建服务器注册”对话框中单击“保存”按钮，即可完成服务器的注册。注册了服务器的“已注册的服务器”页面如图 3.15 所示。

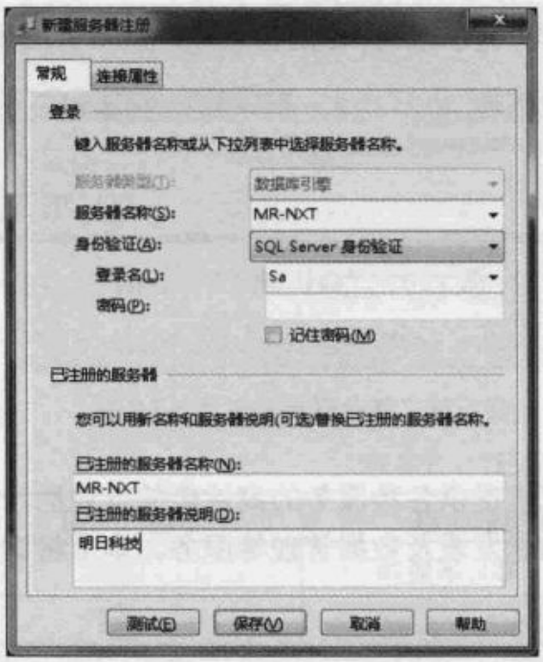


图 3.12 “新建服务器注册”对话框的“常规”选项卡

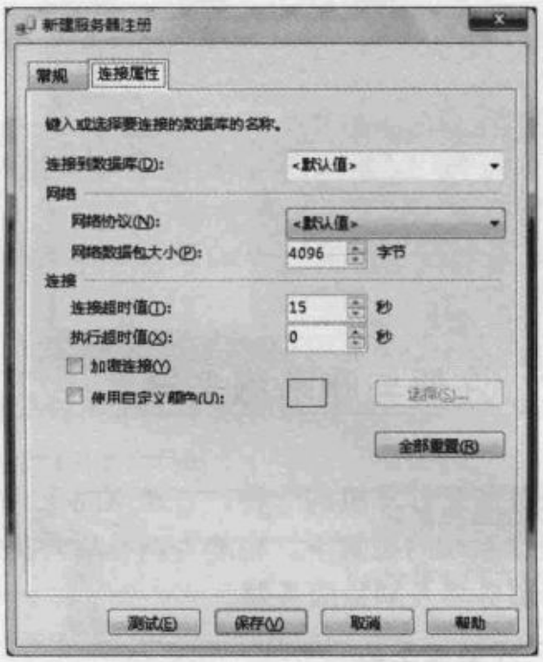


图 3.13 “新建服务器注册”对话框的“连接属性”选项卡

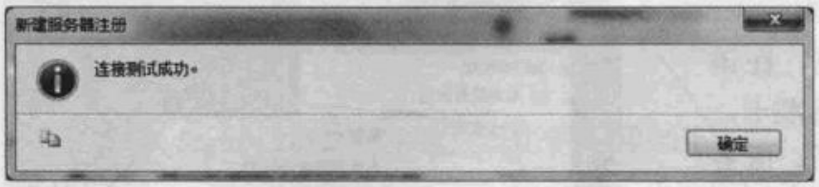


图 3.14 提示连接测试成功的对话框

图 3.15 已注册的服务器页面

每个服务器名称前面的图标代表该服务器目前的运行状态。各图标所代表的服务器运行状态如表 3.2 所示。

表 3.2 图标所代表服务器运行状态的说明

图　标	含　义
	服务器正常运行
	服务器暂停运行
	服务器停止运行
	服务器无法联系

2. 删除服务器

使用 SQL Server 2012 删除服务器的步骤如下：

（1）按照 3.3.1 节中打开“已注册的服务器”的步骤，打开“已注册的服务器”页面。

（2）选择需要删除的服务器，单击鼠标右键，在弹出的快捷菜单中选择“删除”命令，如图 3.16 所示。

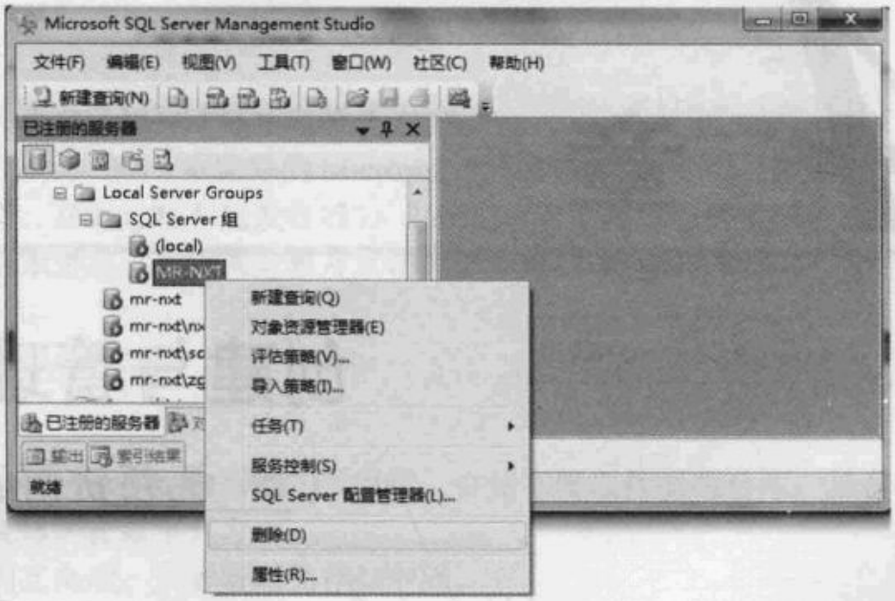


图 3.16　删除服务器菜单

（3）在弹出的“确认删除”对话框中单击“是”按钮，即可完成注册服务器的删除，如图 3.17 所示。

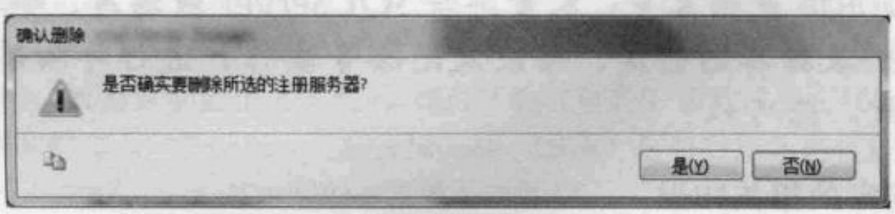


图 3.17　“确认删除”对话框

3.4　小　　结

SQL Server 2012 是微软开发的数据库产品，它将 SQL Server 2000 中的企业管理器、查询分析器和分析管理器等管理工具组合到了一个统一界面中，本章主要对如何启动 SQL Server 2012 的服务以及如何注册 SQL Server 2012 的服务进行了详细讲解。

3.5　实践与练习

1. 启动 SQL Server 2012 服务的两种方式分别是什么？如何操作？（**答案位置：光盘\TM\sl\3\1**）
2. 描述使用 SQL Server Management Studio 创建服务器组的基本步骤。（**答案位置：光盘\TM\sl\3\2**）

第 4 章

创建与管理数据库

（视频讲解：35 分钟）

数据库，顾名思义就是存储数据的仓库，它是程序开发中非常重要的一部分内容，在开发程序时，数据一般都需要通过数据库进行存储，例如大家所熟知的 SQL Server 数据库、Oracle 数据库等。本章将对 SQL Server 数据库、命名规则及其管理（包括 SQL Server 数据库的创建、修改及删除等操作）进行详细讲解。

通过阅读本章，您可以：

- 掌握数据库的相关知识
- 了解 SQL Server 的命名规则
- 掌握创建数据库的两种方法
- 掌握修改数据库的两种方法
- 掌握删除数据库的两种方法

4.1　认识数据库

视频讲解：光盘\TM\lx\4\认识数据库.mp4

Microsoft SQL Server 2012 数据库同 Microsoft 的其他数据库类似，主要用于存储数据及其相同的对象（如视图、索引、存储过程和触发器等），以便随时对数据库中的数据及其对象进行访问和管理。本节将对数据库的基本概念、数据库对象及其相关知识进行详细的介绍。

4.1.1　数据库基本概念

数据库（DataBase）是按照数据结构来组织、存储和管理数据的仓库，是存储在一起的相关数据的集合。其优点主要体现在以下几方面：

- ☑ 减少数据的冗余度，节省数据的存储空间。
- ☑ 具有较高的数据独立性和易扩充性。
- ☑ 实现数据资源的充分共享。

下面介绍与数据库相关的几个概念。

（1）数据库系统。

数据库系统（DataBase System，DBS）是采用数据库技术的计算机系统，是由数据库（数据）、数据库管理系统（软件）、数据库管理员（人员）、硬件平台（硬件）和软件平台（软件）5 部分构成的运行实体。其中，数据库管理员（DataBase Administrator，DBA）是对数据库进行规划、设计、维护和监视等的专业管理人员，在数据库系统中起着非常重要的作用。

（2）数据库管理系统。

数据库管理系统（DataBase Management System，DBMS）是数据库系统的一个重要组成部分，是位于用户与操作之间的一层数据管理软件，负责数据库中的数据组织、数据操纵、数据维护和数据服务等。主要具有如下功能。

- ☑ 数据存取的物理构建：为数据模式的物理存取与构建提供有效的存取方法与手段。
- ☑ 数据操纵功能：为用户使用数据库的数据提供方便，如查询、插入、修改、删除等以及简单的算术运算和统计。
- ☑ 数据定义功能：用户可以通过数据库管理系统提供的数据定义语言（Data Definition Language，DDL）方便地对数据库中的对象进行定义。
- ☑ 数据库的运行管理：数据库管理系统统一管理数据库的运行和维护，以保障数据的安全性、完整性、并发性和故障的系统恢复性。
- ☑ 数据库的建立和维护功能：数据库管理系统能够完成初始数据的输入和转换、数据库的转储和恢复、数据库的性能监视和分析等任务。

（3）关系数据库。

关系数据库是支持关系模型的数据库。关系模型由关系数据结构、关系操作集合和完整性约束等 3

部分组成。

- ☑ 关系数据结构：在关系模型中数据结构单一，现实世界的实体以及实体间的联系均用关系来表示，实际上关系模型中数据结构就是一张二维表。
- ☑ 关系操作集合：关系操作分为关系代数、关系演算、具有关系代数和关系演算双重特点的语言（SQL）。
- ☑ 完整性约束：完整性约束包括实体完整性、参照完整性和用户定义的完整性。

4.1.2 数据库常用对象

在 SQL Server 2012 的数据库中，表、视图、存储过程和索引等具体存储数据或对数据进行操作的实体都被称为数据库对象。下面介绍几种常用的数据库对象。

（1）表。

表是包含数据库中所有数据的数据库对象，由行和列组成，用于组织和存储数据。

（2）字段。

表中每列称为一个字段，字段具有自己的属性，如字段类型、字段大小等，其中字段类型是字段最重要的属性，它决定了字段能够存储哪种数据。

SQL 规范支持 5 种基本字段类型：字符型、文本型、数值型、逻辑型和日期时间型。

（3）索引。

索引是一个单独的、物理的数据库结构。它是依赖于表建立的，在数据库中索引使数据库程序无须对整个表进行扫描，就可以在其中找到所需的数据。

（4）视图。

视图是从一张或多张表中导出的表（也称虚拟表），是用户查看数据表中数据的一种方式。表中包括几个被定义的数据列与数据行，其结构和数据建立在对表的查询基础之上。

（5）存储过程。

存储过程（Stored Procedure）是一组为了完成特定功能的 SQL 语句集合（包含查询、插入、删除和更新等操作），经编译后以名称的形式存储在 SQL Server 服务器端的数据库中，由用户通过指定存储过程的名字来执行。当这个存储过程被调用执行时，这些操作也会同时执行。

4.1.3 数据库组成

SQL Server 2012 数据库主要由文件和文件组组成。数据库中的所有数据和对象（如表、存储过程和触发器）都被存储在文件中。

（1）文件。

文件主要分为以下 3 种类型。

- ☑ 主要数据文件：存放数据和数据库的初始化信息。每个数据库有且只有一个主要数据文件，默认扩展名是.mdf。
- ☑ 次要数据文件：存放除主要数据文件以外的所有数据文件。有些数据库可能没有次要数据文

件，也可能有多个次要数据文件，默认扩展名是.ndf。

☑ 事务日志文件：存放用于恢复数据库的所有日志信息。每个数据库至少有一个事务日志文件，也可以有多个事务日志文件，默认扩展名是.ldf。

（2）文件组。

文件组是 SQL Server 2012 数据文件的一种逻辑管理单位，它将数据库文件分成不同的文件组，方便于对文件的分配和管理。

文件组主要分为以下两种类型。

☑ 主文件组：包含主要数据文件和任何没有明确指派给其他文件组的文件。系统表的所有页都分配在主文件组中。

☑ 用户定义文件组：主要是在 CREATE DATABASE 或 ALTER DATABASE 语句中，使用 FILEGROUP 关键字指定的文件组。

说明

每个数据库中都有一个文件组作为默认文件组运行，默认文件组包含在创建时没有指定文件组的所有表和索引的页。在没有指定的情况下，主文件组作为默认文件组。

对文件进行分组时，一定要遵循文件和文件组的设计规则：

☑ 文件只能是一个文件组的成员。

☑ 文件或文件组不能由一个以上的数据库使用。

☑ 数据和事务日志信息不能属于同一文件或文件组。

☑ 日志文件不能作为文件组的一部分。日志空间与数据空间分开管理。

注意

系统管理员在进行备份操作时，可以备份或恢复个别的文件或文件组，而不用备份或恢复整个数据库。

4.1.4　系统数据库

SQL Server 2012 的安装程序在安装时默认将建立 4 个系统数据库（master、tempdb、model 和 msdb）。下面分别进行介绍。

（1）master 数据库。

master 数据库是 SQL Server 2012 中最重要的数据库，记录 SQL Server 实例的所有系统级信息，包括实例范围的元数据、端点、链接服务器和系统配置设置。

（2）tempdb 数据库。

tempdb 是一个临时数据库，用于保存临时对象或中间结果集。

（3）model 数据库。

用作 SQL Server 实例上创建的所有数据库的模板。对 model 数据库进行的修改（如数据库大小、排序规则、恢复模式和其他数据库选项）将应用于以后创建的所有数据库。

（4）msdb 数据库。

用于 SQL Server 代理计划警报和作业。

4.2 SQL Server 的命名规则

视频讲解：光盘\TM\lx\4\SQL Server 的命名规则.mp4

SQL Server 为了完善数据库的管理机制，设计了严格的命名规则。用户在创建数据库及数据库对象时必须严格遵守 SQL Server 的命名规则。本节将对标识符、对象和实例的命名进行详细的介绍。

4.2.1 标识符

在 SQL Server 中，服务器、数据库和数据库对象（如表、视图、列、索引、触发器、过程、约束和规则等）都有标识符，数据库对象的名称被看成是该对象的标识符。大多数对象要求带有标识符，但有些对象（如约束）中标识符是可选项。

对象标识符是在定义对象时创建的，标识符随后用于引用该对象，下面分别对标识符的格式及分类进行介绍。

1. 标识符格式

在定义标识符时必须遵守以下规定：

（1）标识符的首字符必须是下列字符之一。

☑ 统一码（Unicode）2.0 标准中所定义的字母，包括拉丁字母 a～z 和 A～Z，以及来自其他语言的字符。

☑ 下划线“_”、符号“@”或者数字符号“#”。

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“@”符号开始的标识符表示局部变量或参数；以一个数字符号“#”开始的标识符表示临时表或过程，如表“#gzb”就是一张临时表；以双数字符号“##”开始的标识符表示全局临时对象，如表“##gzb”则是全局临时表。

注意

某些 Transact-SQL 函数的名称以双 at 符号（@@）开始，为避免混淆这些函数，建议不要使用以@@开始的名称。

（2）标识符的后续字符可以是以下 3 种。

☑ 统一码（Unicode）2.0 标准中所定义的字母。

☑ 来自拉丁字母或其他国家/地区脚本的十进制数字。

☑ “@”符号、美元符号“$”、数字符号“#”或下划线“_”。

（3）标识符不允许是 Transact-SQL 的保留字。

（4）不允许嵌入空格或其他特殊字符。

例如，要为明日科技公司创建一个工资管理系统，则可以将其数据库命名为“MR_NXT”。名字除了要遵守命名规则以外，最好还能准确表达数据库的内容，本例中的数据库名称是以每个字的大写字母命名的，其中还使用了下划线“_”。

2．标识符分类

SQL Server 将标识符分为以下两种类型。

☑ 常规标识符：符合标识符的格式规则。

☑ 分隔标识符：包含在双引号（“”）或者方括号（[ ]）内的标识符。该标识符可以不符合标识符的格式规则，如[MR GZGLXT]、MR 和 GZGLXT 之间含有空格，但因为使用了方括号，所以视为分隔标识符。

注意

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。

4.2.2　对象命名规则

SQL Server 2012 的数据库对象的名字由 1～128 个字符组成，不区分大小写。使用标识符也可以作为对象的名称。

在一个数据库中创建了一个数据库对象后，数据库对象的完整名称应该由服务器名、数据库名、拥有者名和对象名 4 部分组成，其语法格式如下：

```
[[[ server. ][ database ] .][ owner_name ] .] object_name
```

服务器、数据库和所有者的名称即所谓的对象名称限定符。当引用一个对象时，不需要指定服务器、数据库和所有者，可以利用句号标出它们的位置，从而省略限定符。

对象名的有效格式如下：

```
server.database.owner_name.object_name
server.database..object_name
server..owner_name.object_name
server...object_name
database.owner_name.object_name
database..object_name
owner_name.object_name
object_name
```

指定了所有 4 个部分的对象名称被称为完全合法名称。

注意

不允许存在 4 部分名称完全相同的数据库对象。在同一个数据库中可以存在两个名为 EXAMPLE 的表格，但前提必须是这两个表的拥有者不同。

4.2.3 实例命名规则

SQL Server 2012 提供了两种类型的实例：默认实例和命名实例。

☑ 默认实例

此实例由运行它的计算机的网络名称标识。使用以前版本 SQL Server 客户端软件的应用程序可以连接到默认实例。SQL Server 6.5 版或 SQL Server 7.0 版服务器可作为默认实例操作。但是，一台计算机上每次只能有一个版本作为默认实例运行。

☑ 命名实例

计算机可以同时运行任意个 SQL Server 命名实例。实例通过计算机的网络名称加上实例名称以<计算机名称>\<实例名称>格式进行标识，即 computer_name\instance_name，但该实例名不能超过 16 个字符。

4.3 数据库的创建与管理

视频讲解：光盘\TM\lx\4\数据库的创建与管理.mp4

在 Microsoft SQL Server 中，数据库主要用来存储数据及数据库对象（如表、索引等）。在本节中主要介绍如何创建、修改和删除数据库。

4.3.1 创建数据库

在 SQL Server 创建用户数据库之前，用户必须设计好数据库的名称以及它的所有者、空间大小和存储信息的文件和文件组。

1. 以界面方式创建数据库

下面在 SQL Server Management Studio 中创建数据库 db_database，具体操作步骤如下：

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库。

（2）鼠标右键单击“数据库”选项，在弹出的快捷菜单中选择“新建数据库”命令，如图 4.1 所示。

（3）进入“添加库”对话框，如图 4.2 所示。在列表框中填写数据库名“db_database”，单击“确定”按钮，添加数据库成功。

☑ “常规”选项卡：用于设置新建数据库的名称。

☑ “选项”和“文件组”选项卡：定义数据库的一些选项，显示文件和文件组的统计信息。这里均采用默认设置。

图 4.1 新建数据库

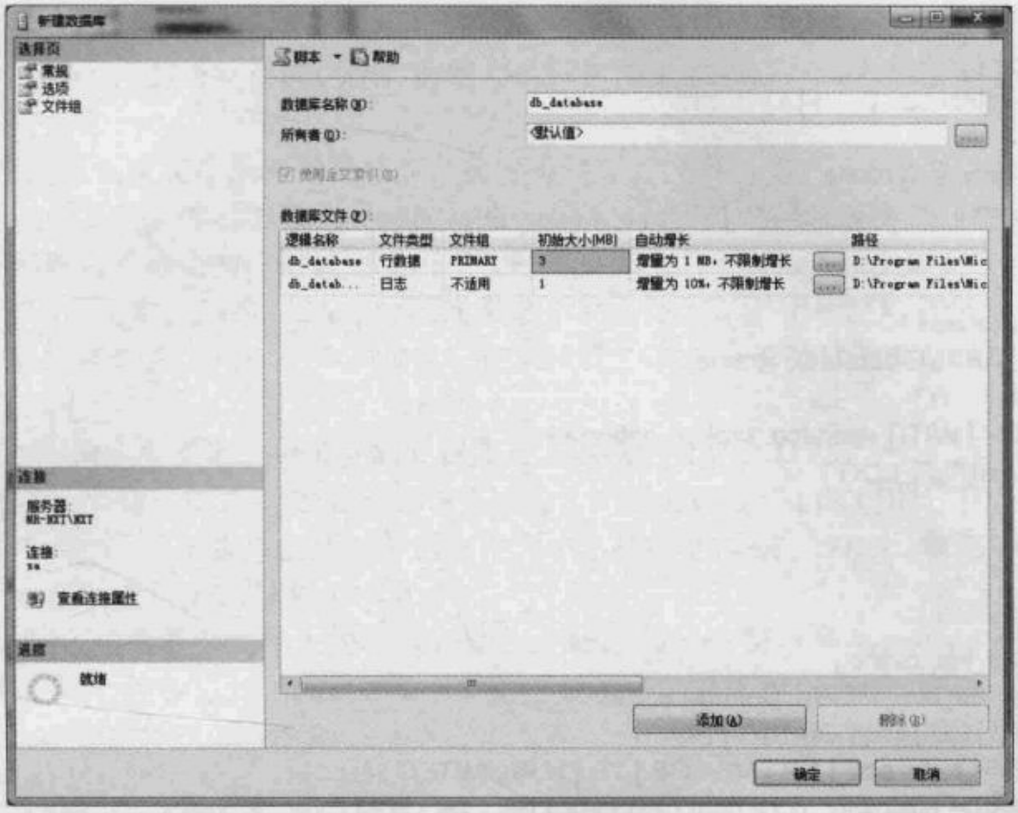


图 4.2　创建数据库名称

说明

SQL Server 2012 默认创建了一个 PRIMARY 文件组，用于存放若干个数据文件。但日志文件没有文件组。

（4）单击“所有者”的“浏览”按钮…，在弹出的列表框中选择数据库的所有者。数据库所有者是对数据库具有完全操作权限的用户，这里选择“默认值”选项，表示数据库所有者为用户登录 Windows 操作系统使用的管理员账户，如 Administrator。

注意

SQL Server 2012 数据库的数据文件分为逻辑名称和物理名称。逻辑名称是在 SQL 语句中引用文件时所使用的名称；物理名称用于操作系统管理。

（5）在“数据库名称”文本框中输入新建数据库的名称“db_database”，数据库名称设置完成后，系统自动在“数据库文件”列表中产生一个主要数据文件（初始大小为 5MB）和一个日志文件（初始大小为 1MB），同时显示文件组、自动增长和路径等默认设置，用户可以根据需要自行修改这些默认的设置，也可以单击右下角的“添加”按钮添加数据文件。这里主要数据文件和日志文件均采用默认设置。

2. 使用 CREATE DATABASE 语句创建数据库

语法如下：

```
CREATE DATABASE database_name
[ ON
```

```
 [ PRIMARY ] [ <filespec> [ ,...n ]
 [ , <filegroup> [ ,...n ] ]
 [ LOG ON { <filespec> [ ,...n ] } ]
]
 [ COLLATE collation_name ]
 [ WITH <external_access_option> ]
]
[;]
To attach a database
CREATE DATABASE database_name
ON <filespec> [ ,...n ]
FOR { ATTACH [ WITH <service_broker_option> ]
| ATTACH_REBUILD_LOG }
[;]
<filespec> ::=
{
(
NAME = logical_file_name ,
FILENAME = { 'os_file_name' | 'filestream_path' }
 [ , SIZE = size [ KB | MB | GB | TB ] ]
 [ , MAXSIZE = { max_size [ KB | MB | GB | TB ] | UNLIMITED } ]
 [ , FILEGROWTH = growth_increment [ KB | MB | GB | TB | % ] ]
) [ ,...n ]
}
<filegroup> ::=
{
FILEGROUP filegroup_name [ CONTAINS FILESTREAM ] [ DEFAULT ]
<filespec> [ ,...n ]
}
<external_access_option> ::=
{
 [ DB_CHAINING { ON | OFF } ]
 [ , TRUSTWORTHY { ON | OFF } ]
}
<service_broker_option> ::=
{
ENABLE_BROKER
|NEW_BROKER
|ERROR_BROKER_CONVERSATIONS
}
Create a database snapshot
CREATE DATABASE database_snapshot_name
ON
 (
NAME = logical_file_name,
FILENAME = 'os_file_name'
) [ ,...n ]
AS SNAPSHOT OF source_database_name
[;]
```

参数说明如下。

- ☑ database_name：新数据库的名称。数据库名称在 SQL Server 的实例中必须唯一，并且必须符合标识符规则。
- ☑ ON：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）。当后面是以逗号分隔的、用以定义主文件组的数据文件的<filespec>项列表时，需要使用 ON。主文件组的文件列表可后跟以逗号分隔的、用以定义用户文件组及其文件的<filegroup>项列表（可选）。
- ☑ PRIMARY：指定关联的 <filespec> 列表定义主文件。在主文件组的<filespec>项中指定的第一个文件将成为主文件。一个数据库只能有一个主文件。有关详细信息，请参阅文件和文件组体系结构。
- ☑ LOG ON：指定显式定义用来存储数据库日志的磁盘文件（日志文件）。LOG ON 后跟以逗号分隔的用以定义日志文件的<filespec>项列表。如果没有指定 LOG ON，将自动创建一个日志文件，其大小为该数据库的所有数据文件大小总和的 25%或 512KB，取两者之中的较大者。不能对数据库快照指定 LOG ON。
- ☑ COLLATE：指明数据库使用的校验方式。collation_name 可以是 Windows 的校验方式名称，也可以是 SQL 校验方式名称。如果省略此子句，则数据库使用当前的 SQL Server 校验方式。
- ☑ NAME：指定文件在 SQL Server 中的逻辑名称。当使用 FOR ATTACH 选项时，就不需要使用 NAME 选项了。
- ☑ FILENAME：指定文件在操作系统中存储的路径和文件名称。
- ☑ SIZE：指定数据库的初始容量大小。如果没有指定主文件的大小，则 SQL Server 默认其与模板数据库中的主文件大小一致，其他数据库文件和事务日志文件则默认为 1MB。指定大小的数字 SIZE 可以使用 KB、MB、GB 和 TB 作为后缀，默认的后缀是 MB。SIZE 中不能使用小数，其最小值为 512KB，默认值是 1MB。主文件的 SIZE 不能小于模板数据库中的主文件。
- ☑ MAXSIZE：指定文件的最大容量。如果没有指定 MAXSIZE，则文件可以不断增长直到充满磁盘。
- ☑ UNLIMITED：指明文件无容量限制。
- ☑ FILEGROWTH：指定文件每次增容时增加的容量大小。增加量可以用以 KB、MB 作后缀的字节数或以%作后缀的被增容文件的百分比来表示。默认后缀为 MB。如果没有指定 FILEGROWTH，则默认值为 10%，每次扩容的最小值为 64KB。

【例 4.1】 用 CREATE DATABASE 命令创建一个名称为“MingRi”的数据库，如图 4.3 所示。（**实例位置：光盘\TM\sl\4\1**）

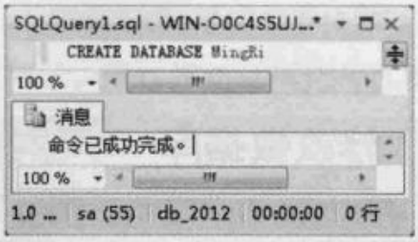


图 4.3　创建一个名称为 MingRi 的数据库

代码如下：

```
CREATE DATABASE MingRi                    --使用 create database 命令创建一个名称是 MingRi 的数据库
```

注意

在创建数据库时，所要创建的数据库名称必须是系统中不存在的，如果存在相同名称的数据库，在创建数据库时系统将会报错。另外，数据库的名称也可以是中文名称。

【例 4.2】 在数据库中，使用 CREATE DATABASE 命令创建名为“mrkj”的数据库。其中，主数据文件名称是“mrkj.mdf”，初始大小是 10MB，最大存储空间是 100MB，增长大小是 5MB。而日志文件名称是“mrkj.ldf”，初始大小是 8MB，最大的存储空间是 50MB，增长大小是 8MB，如图 4.4 所示。（实例位置：光盘\TM\sl\4\2）

代码如下：

```
create database mrkj                    --创建数据库 mrkj
on                                      --主数据文件
(name=mrdat,                            --name 文件名，filename 文件路径
filename='G:\sql\mrkj.mdf',             --指定主数据文件的路径
size=10,                                --文件大小
maxsize=100,                            --最大值
filegrowth=5)                           --标识增量
log on                                  --事务日志文件
(name='mingrilog',                      --name 文件名，filename 文件路径
filename='G:\sql\mrkj.ldf',             --指定事务日志文件的路径
size=8mb,                               --文件大小
maxsize=50mb,                           --最大值
filegrowth=8mb )                        --增长率
```

图 4.4 自定义选项创建数据库

4.3.2 修改数据库

数据库创建完成以后，用户在使用过程中可以根据需要对其原始定义进行修改。修改的内容主要包括以下几项：

- ☑ 更改数据库文件。
- ☑ 添加和删除文件组。
- ☑ 更改选项。
- ☑ 更改跟踪。
- ☑ 更改权限。
- ☑ 更改扩展属性。

☑ 更改镜像。

☑ 更改事务日志传送。

1. 以界面方式修改数据库

下面介绍如何更改数据库 db_2012 的所有者。具体操作步骤如下：

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库，在“对象资源管理器”中展开“数据库”节点。

（2）右击需要更改的数据库 db_2012 选项，在弹出的快捷菜单中选择“属性”命令，如图 4.5 所示。

（3）进入“数据库属性”窗口，如图 4.6 所示。通过该窗口可以修改数据库的相关选项。

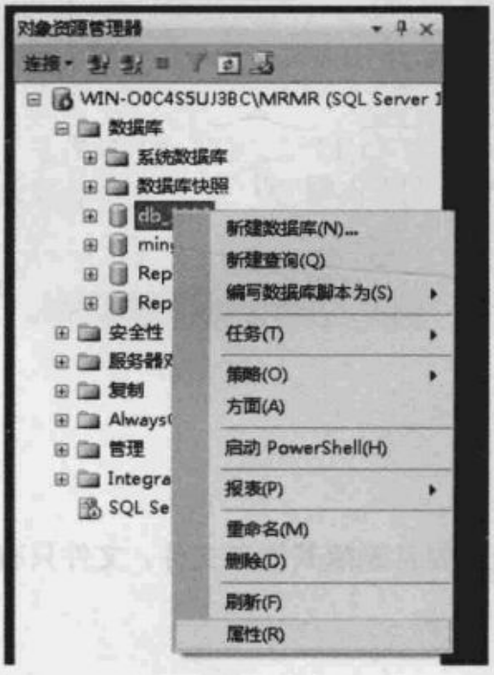


图 4.5　选择数据库属性

图 4.6　“文件”选项卡

（4）打开“数据库属性”窗口中的“文件”选项卡，然后单击“所有者”后的“浏览”按钮 ...，弹出“选择数据库所有者”对话框，如图 4.7 所示。

（5）单击“浏览”按钮，弹出“查找对象”对话框，如图 4.8 所示。通过该对话框选择匹配对象。

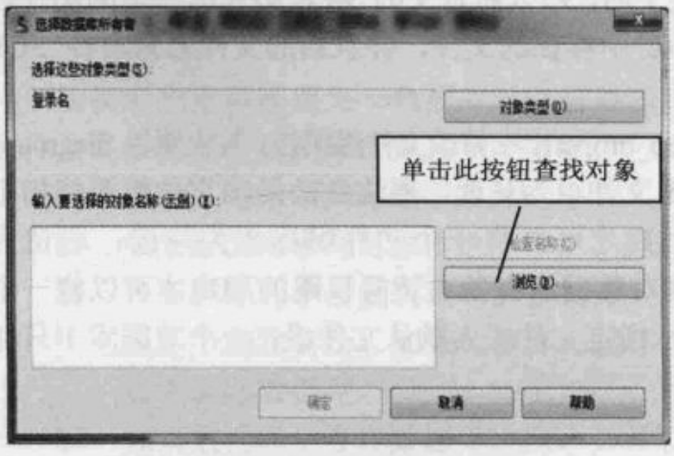


图 4.7　选择数据库所有者

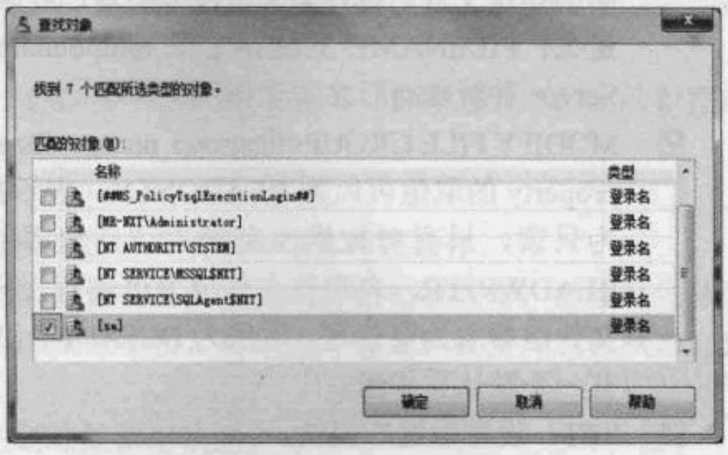


图 4.8　“查找对象”对话框

（6）在“匹配的对象”列表框中选择数据库的所有者 sa 选项，单击“确定”按钮，完成数据库所有者的更改操作。

2. 使用 ALTER DATABASE 语句修改数据库

T-SQL 中修改数据库的命令为 ALTER DATABASE。

语法格式如下：

```
ALTER DATABASE database_name
{ADD FILE<filespec>[,…n][TO FILEGROUP filegroup_name]
|ADD LOG FILE<filespec>[,…n]
|REMOVE FILE logical_file_name
|ADD FILEGROUP filegroup_name
|REMOVE FILEGROUP filegroup_name
|MODIFY FILE<filespec>
|MODIFY NAME=new_dbname
|MODIFY FILEGROUP filegroup_name{filegroup_property|NAME=new_filegroup_name}
|SET<optionspec>[,…n][WITH<termination>]
|COLLATE<collation_name>
}
```

参数说明如下。

- ☑ ADD FILE：指定要增加的数据库文件。
- ☑ TO FILEGROUP：指定要增加文件到哪个文件组。
- ☑ ADD LOG FILE：指定要增加的事务日志文件。
- ☑ REMOVE FILE：从数据库系统表中删除指定文件的定义，并且删除其物理文件。文件只有为空时才能被删除。
- ☑ ADD FILEGROUP：指定要增加的文件组。
- ☑ REMOVE FILEGROUP：从数据库中删除指定文件组的定义，并且删除其包含的所有数据库文件。文件组只有为空时才能被删除。
- ☑ MODIFY FILE：修改指定文件的文件名、容量大小、最大容量、文件增容方式等属性，但一次只能修改一个文件的一个属性。使用此选项时应注意，在文件格式 filespec 中必须用 NAME 明确指定文件名称，如果文件大小是已经确定了的，那么新定义的 SIZE 必须比当前的文件容量大；FILENAME 只能指定在 tempdbdatabase 中存在的文件，并且新的文件名只有在 SQL Server 重新启动后才发生作用。
- ☑ MODIFY FILE GROUP<filegroup_name><filegroup_property>：修改文件组属性，其中属性 filegroup_property 的取值可以为 READONLY，表示指定文件组为只读，要注意的是主文件组不能指定为只读，只有对数据库有独占访问权限的用户才可以将一个文件组标志为只读；取值为 READWRITE，表示使文件组为可读写，只有对数据库有独占访问权限的用户才可以将一个文件组标志为可读写；取值为 DEFAULT，表示指定文件组为默认文件组，一个数据库中只能有一个默认文件组。
- ☑ SET：设置数据库属性。
- ☑ ALTER DATABASE 命令可以修改数据库大小、缩小数据库、更改数据库名称等。

【例 4.3】 将一个大小为 10MB 的数据文件 mrkj 添加到 MingRi 数据库中，该数据文件的大小为 10MB，最大的文件大小为 100MB，增长速度为 2MB，MingRi 数据库的物理地址为 G 盘文件夹下。（**实例位置：光盘\TM\sl\4\3**）

SQL 语句如下：

```
ALTER DATABASE Mingri                    --更改数据库
ADD FILE                                 --添加文件
(
NAME=mrkj,                               --文件名
Filename='G:\mrkj.ndf',                  --路径
size=10MB,                               --大小
Maxsize=100MB,                           --最大值
Filegrowth=2MB                           --标识增量
)
```

【例 4.4】 将数据库名称“mr”更名为“mrsoft”，在 mr 数据库中，使用系统存储过程 sp_renamedb 将数据库名称“mr”更名为“mrsoft”，如图 4.9 所示。（**实例位置：光盘\TM\sl\4\4**）

代码如下：

```
exec sp_renamedb 'mr', 'mrsoft'          --数据库重命名
```

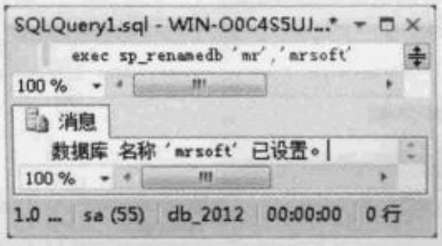


图 4.9　将数据库名称“mr”更名为“mrsoft”

注意

只有属于 sysadmin 固定服务器角色的成员可以执行 sp_renamedb 系统存储过程。

4.3.3　删除数据库

如果用户不再需要某一数据库时，只要满足一定的条件即可将其删除，删除之后，相应的数据库文件及其数据都会被删除，并且不可恢复。

删除数据库时必须满足以下条件：

- ☑ 如果数据库涉及日志传送操作，在删除数据库之前必须取消日志传送操作。
- ☑ 若要删除为事务复制发布的数据库，或删除为合并复制发布或订阅的数据库，必须首先从数据库中删除备份。如果数据库已损坏，不能删除备份，可以先将数据库设置为脱机状态，然后再删除数据库。
- ☑ 如果数据库上存在数据库快照，必须首先删除数据库快照。

1. 以界面方式删除数据库

下面介绍如何删除数据库 MingRi。具体操作步骤如下：

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库。在“对象资源管理器”中展开“数据库”节点。

（2）右击要删除的数据库 MingRi 选项，在弹出的快捷菜单中选择“删除”命令，如图 4.10 所示。

（3）在弹出的“删除对象”窗口中单击“确定”按钮即可删除数据库，如图 4.11 所示。

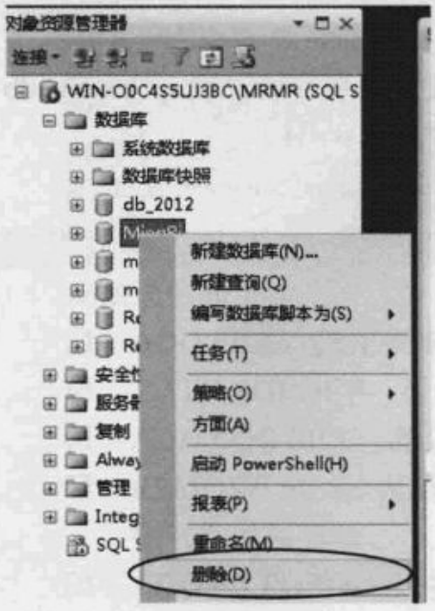


图 4.10　删除数据库

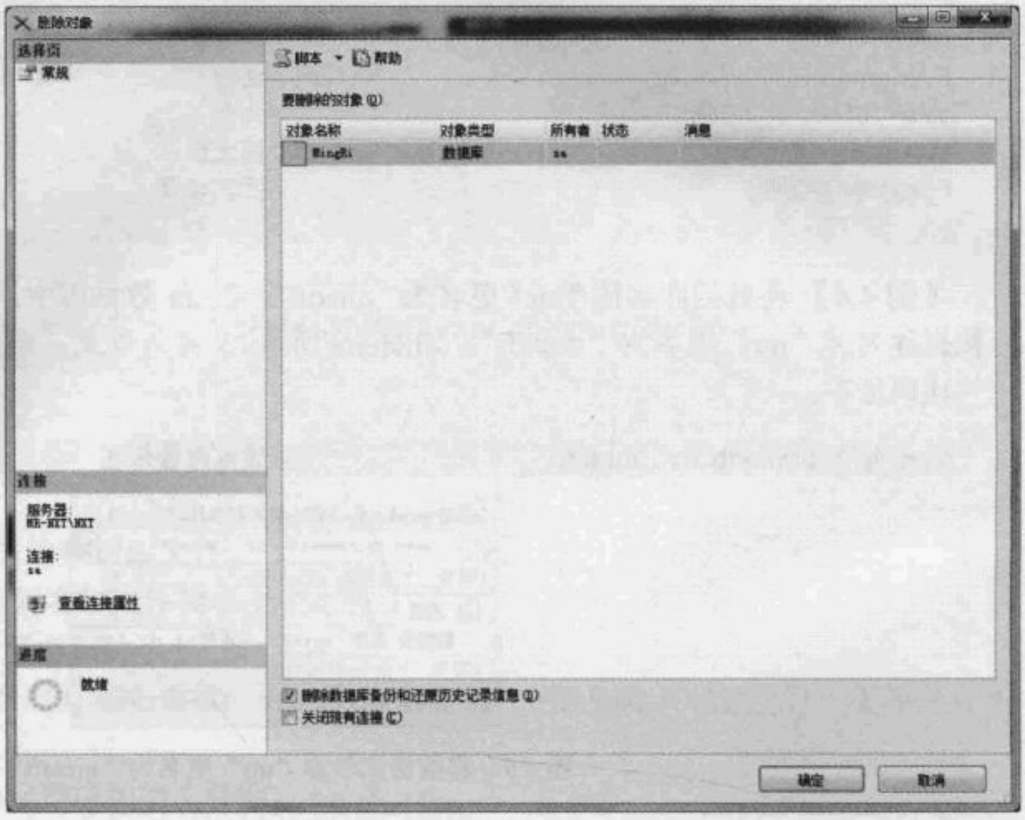


图 4.11　除去对象

注意

系统数据库（msdb、model、master、tempdb）无法删除。删除数据库后应立即备份 master 数据库，因为删除数据库将更新 master 数据库中的信息。

2. 使用 DROP DATABASE 语句删除数据库

语法格式如下：

```
DROP DATABASE database_name [ ,...n ]                    --如果有多个要删除的数据库，用逗号隔开
```

其中，database_name 是要删除的数据库名称，中括号内为多个数据库的情况。

注意

使用 DROP DATABASE 命令删除数据库时，系统中必须存在所要删除的数据库，否则系统将会出现错误。

另外，如果删除正在使用的数据库，系统将会出现错误。

例如，不能在“学生档案管理”数据库中删除“学生档案管理”数据库，SQL 代码如下：

```
Use 学生档案管理                          --使用学生档案管理数据库
Drop database 学生档案管理                --删除正在使用的数据库
```

删除学生档案管理数据库的操作没有成功，系统会报错，运行结果如图 4.12 所示。

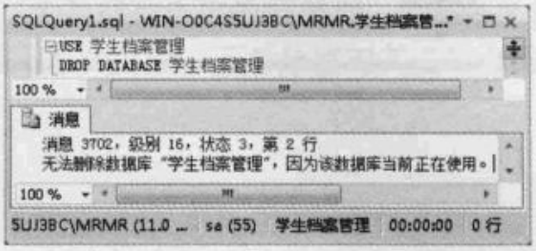


图 4.12　删除正在使用的数据库，系统会报错的效果图

在“学生档案管理”数据库没有被使用的情况下，将其删除（可以先使用其他数据库，使要删除的数据库处于未使用状态）。运行结果如图 4.13 所示。

使用 DROP DATABASE 命令还可以批量删除数据库。例如，将“m1”、“m2”和“m3”这 3 个数据库批量删除。要删除的数据库间用逗号隔开，运行结果如图 4.14 所示。

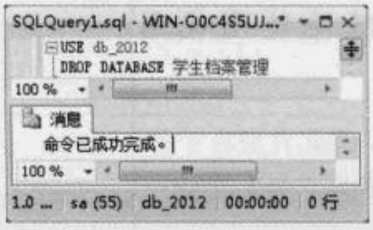


图 4.13　删除“学生档案管理”数据库

图 4.14　批量删除数据库

4.4　小　　结

本章主要讲解了 SQL Server 2012 数据库的组成，及 SQL Server 2012 数据库的创建、修改和删除操作。其中，讲解 SQL Server 2012 数据库的创建、修改和删除时，分别用了使用向导和使用 SQL 语句两种方法，这部分内容是本章学习的重点，一定要熟练掌握。

4.5　实践与练习

1. 使用 CREATE DATABASE 语句创建一个名称为 mrsoft 的数据库。（**答案位置：光盘\TM\sl\4\5**）
2. 使用系统存储过程 sp_renamedb 将 mrsoft 数据库修改为“明日科技”。（**答案位置：光盘\TM\sl\4\6**）
3. 使用 DROP DATABASE 语句删除“明日科技”数据库。（**答案位置：光盘\TM\sl\4\7**）

第 5 章

操作数据表

（视频讲解：78 分钟）

本章主要介绍使用管理器创建数据表、修改数据表、删除数据表和对视图进行操作的过程，通过本章的学习，读者不仅可以熟悉 SQL Server 2012 数据表的组成，掌握创建和管理数据表的方法，另外，还能够熟练地使用视图。

通过阅读本章，您可以：

- 熟悉 SQL Server 2012 中的数据类型
- 掌握如何使用管理器管理数据表
- 熟悉管理数据表的 SQL 语句
- 熟练掌握如何对数据表执行添加、修改和删除数据的操作
- 掌握创建、修改及删除约束的方法
- 熟悉数据表关系的建立与维护

5.1　数据表基础

视频讲解：光盘\TM\lx\5\数据表基础.mp4

在创建数据表的过程中，需要为数据列选定数据类型，用于定义各列所允许的数据值。SQL Server 2012 提供了基本数据类型和自定义数据类型，下面分别对其进行介绍。

5.1.1　基本数据类型

基本数据类型按数据的表现方式及存储方式的不同可以分为整数数据类型、货币数据类型、浮点数据类型、日期/时间数据类型、字符数据类型、二进制数据类型、图像和文本数据类型以及 SQL Server 2012 引用的 3 种新数据类型。具体介绍如表 5.1 所示。

表 5.1　基本数据类型

分　类	数 据 特 性	数 据 类 型
整数数据类型	常用的一种数据类型，可以存储整数或者小数	BIT
		INT
		SMALLINT
		TINYINT
货币数据类型	用于存储货币值，使用时在数据前加上货币符号，不加货币符号的情况下默认为“￥”	MONEY
		SMALLMONEY
浮点数据类型	用于存储十进制小数	REAL
		FLOAT
		DECIMAL
		NUMERIC
日期/时间数据类型	用于存储日期类型和时间类型的组合数据	DATETIME
		SMALLDATETIME
		DATA
		DATETIME(2)
		DATETIMESTAMPOFFSET
字符数据类型	用于存储各种字母、数字符号和特殊符号	CHAR
		NCHAR（n）
		VARCHAR
		NVARCHAR(n)
二进制数据类型	用于存储二进制数据	BINARY
		VARBINARY
图像和文本数据类型	用于存储大量的字符及二进制数据（Binary Data）	TEXT
		NTEXT(n)
		IMAGE

5.1.2 用户自定义数据类型

用户自定义数据类型并不是真正的数据类型，它只是提供了一种加强数据库内部元素和基本数据类型之间一致性的机制。通过使用用户自定义数据类型能够简化对常用规则和默认值的管理。

在 SQL Server 2012 中，创建用户自定义数据类型有两种方法：一是使用管理器，二是使用 Transact-SQL 语句，下面分别进行介绍。

1. 使用管理器创建用户定义数据类型

在 db_CSharp 数据库中，创建用来存储邮政编码信息的“postcode”用户定义数据类型，数据类型为 char，长度为 8000。

操作步骤如下：

（1）在操作系统的任务栏中单击“开始”菜单，选择“所有程序”/ Microsoft SQL Server 2012 / SQL Server Management Studio 命令，打开 SQL Server 2012。

（2）在 SQL Server 2012 的对象资源管理器中，依次展开“数据库”/“选择指定数据库”/“可编程性”/“类型”节点。

（3）在下拉列表中选中“用户定义的数据类型”，单击鼠标右键，在弹出的快捷菜单中选择“新建用户定义的数据类型”命令。在打开的窗口中设置用户定义数据类型的名称、依据的系统数据类型以及是否允许 NULL 值等，如图 5.1 所示，还可以将已创建的规则和默认值绑定到该用户定义的数据类型上。

（4）单击“确定”按钮，完成创建工作。

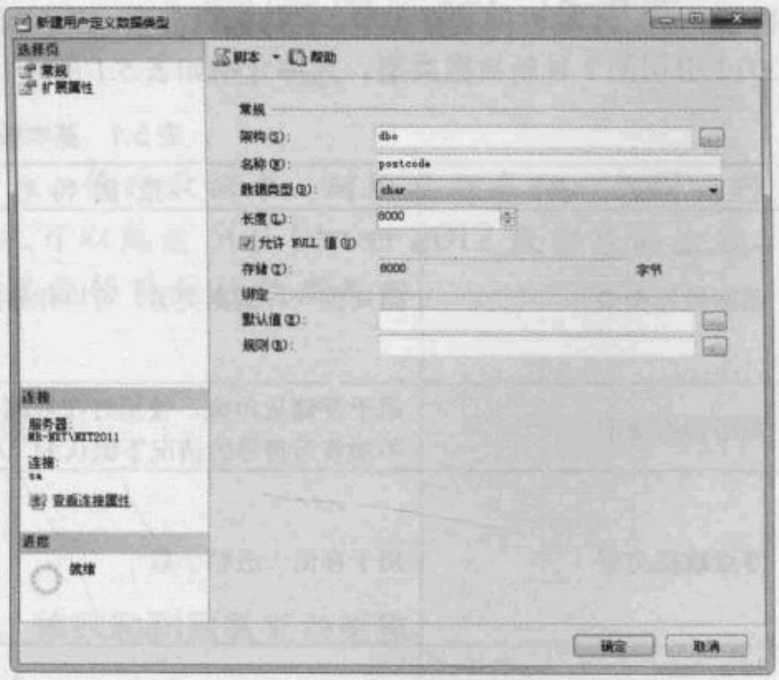


图 5.1 创建用户自定义数据类型

2. 使用 Transact-SQL 语句创建用户自定义数据类型

在 SQL Server 2012 中，使用系统数据类型 sp_addtype 创建用户自定义数据类型。

语法如下：

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

参数说明如下。

☑ [@typename=]type：指定待创建的用户自定义数据类型的名称。用户定义数据类型名称必须遵循标识符的命名规则，而且在数据库中唯一。

☑ [@phystype=]system_data_type'：指定用户定义数据类型所依赖的系统数据类型。

☑ [@nulltype=]'null_type'：指定用户定义数据类型的可空属性，即用户定义数据类型处理空值的方式。取值为 NULL、NOT NULL 或 NONULL。

在 db_CSharp 数据库中，创建用来存储邮政编码信息的“postcode”用户自定义数据类型。在查询分析器中运行的结果如图 5.2 所示。

SQL 语句如下：

```
USE    db_CSharp
EXEC sp_addtype postcode,'char(8) ','not null'
```

创建用户定义数据类型后，就可以像系统数据类型一样使用用户自定义数据类型。例如，在 db_CSharp 数据库的 tb_Student 表中创建新的字段，为字段“邮政编码”指定数据类型时，就可以在下拉列表框中选择刚刚创建的用户数据类型 postalcode 了，如图 5.3 所示。

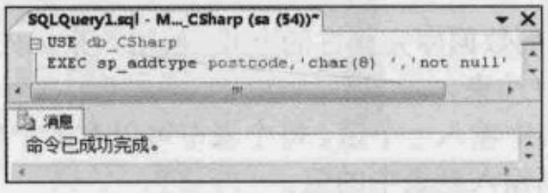


图 5.2　用户自定义“postalcode”类型

图 5.3　创建字段时使用了 postalcode 数据类型

根据需要，还可以修改、删除用户数据类型。SQL Server 2012 提供系统存储过程 sp_droptype，该存储过程从 systypes 删除别名数据类型。

5.1.3　数据表的数据完整性

表列中除了具有数据类型和大小属性之外，还有其他属性。其他属性是保证数据库中数据完整性和表的引用完整性的重要部分。

数据完整性是指列中每个事件都有正确的数据值。数据值的数据类型必须正确，并且数据值必须位于正确的域中。

引用完整性指示表之间的关系得到正确维护。一个表中的数据只应指向另一个表中的现有行，不应指向不存在的行。

SQL Server 2012 提供多种强制数据完整性的机制，下面分别对其进行介绍。

（1）空值与非空值（NULL 或 NOT NULL）。

表的每一列都有一组属性，如名称、数据类型、数据长度和为空性等，列的所有属性即构成列的定义。列可以定义为允许或不允许空值。

☑ 允许空值（NULL）：默认情况下，列允许空值，即允许用户在添加数据时省略该列的值。

☑ 不允许空值（NOT NULL）：不允许在没有指定列默认值的情况下省略该列的值。

（2）默认值。

如果在插入行时没有指定列的值，那么默认值将指定列中所使用的值。默认值可以是任何取值为

常量的对象，如内置函数和数学表达式等。下面介绍两种使用默认值的方法。

☑ 在 CREATE TABLE 中使用 DEFAULT 关键字创建默认定义，将常量表达式指派为列的默认值，这是标准方法。

☑ 使用 CREATE DEFAULT 语句创建默认对象，然后使用 sp_bindefault 系统存储过程将它绑定到列上，这是一个向前兼容的功能。

（3）特定标识属性（IDENTITY）。

数据表中如果某列被指派特定标识属性（IDENTITY），系统将自动为表中插入的新行生成连续递增的编号。因为标识值通常唯一，所以标识列常定义为主键。

IDENTITY 属性适用于 INT、SMALLINT、TINYINT、DECIMAL（P,0）、UMERIC（P,0）数据类型的列。

注意

一个列不能同时具有 NULL 属性和 IDENTITY 属性，二者只能选其一。

（4）约束。

约束是用来定义 Microsoft SQL Server 2012 自动强制数据库完整性的方式。使用约束优先于使用触发器、规则和默认值。SQL Server 2012 中共有以下 5 种约束。

☑ 非空（NOT NULL）：使用户必须在表的指定列中输入一个值。每个表中可以有多个非空约束。

☑ 检查（Check）：用来指定一个布尔操作，限制输入到表中的值。

☑ 唯一性（Unique）：使用户的应用程序必须向列中输入一个唯一的值，值不能重复，但可以为空。

☑ 主键（Primary key）：建立一列或多列的组合以唯一标识表中的每一行。主键可以保证实体完整性，一个表只能有一个主键，同时主键中的列不能接受空值。

☑ 外键（Foreign key）：外键是用于建立和加强两个表数据之间的链接的一列或多列。当一个表中作为主键的一列被添加到另一个表中时，链接就建立了，主要目的是控制存储在外键表中的数据。

5.2 数据表的创建与管理

视频讲解：光盘\TM\lx\5\数据表的创建与管理.mp4

在前面的章节中，读者已经掌握如何在 SQL Server 的企业管理器中创建和维护数据表。这些操作方式虽然非常简单、直观，也便于学习和掌握，但是，它不能将工作的过程保存下来，每次操作都需要重复进行，操作量大的时候不易使用。所以在很多情况下，需要用 Transact-SQL 语句创建数据表、修改数据表、删除数据表、添加数据、修改数据和删除数据等。

5.2.1 以界面方式操作数据表

1. 创建数据表

下面在 SQL Server Management Studio 中创建数据表“mrkj”，具体操作步骤如下：

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库。

（2）右击“表”选项，在弹出的快捷菜单中选择“新建表”命令，如图 5.4 所示。

（3）进入“添加表”对话框，如图 5.5 所示。在列表框中填写所需要的字段名，单击“保存”按钮，添加表成功。

图 5.4　新建表

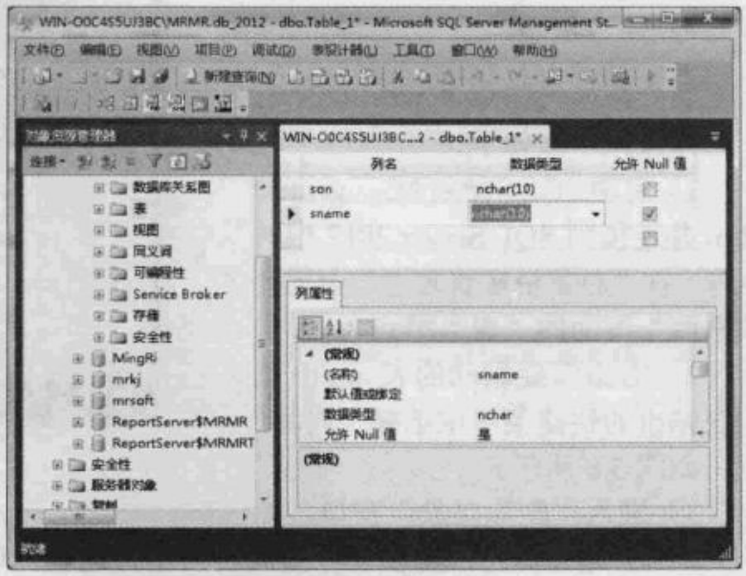


图 5.5　创建数据表名称

2. 修改数据表

下面介绍如何更改表 mrkj 的字段。具体操作步骤如下：

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库，在“对象资源管理器”中展开“数据库”下面的“表”节点。

（2）右击需要更改的表 mrkj 选项，在弹出的快捷菜单中选择“设计”命令，如图 5.6 所示。

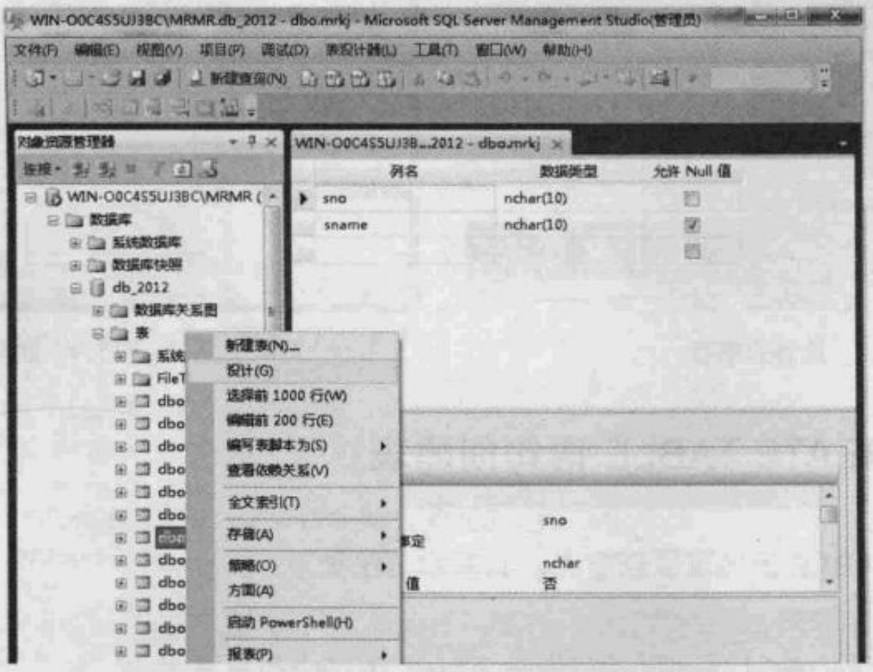


图 5.6　选择表设计

（3）进入“表设计”对话框，如图 5.7 所示。通过该对话框可以修改数据表的相关选项。修改完成后，单击“保存”按钮，修改成功。

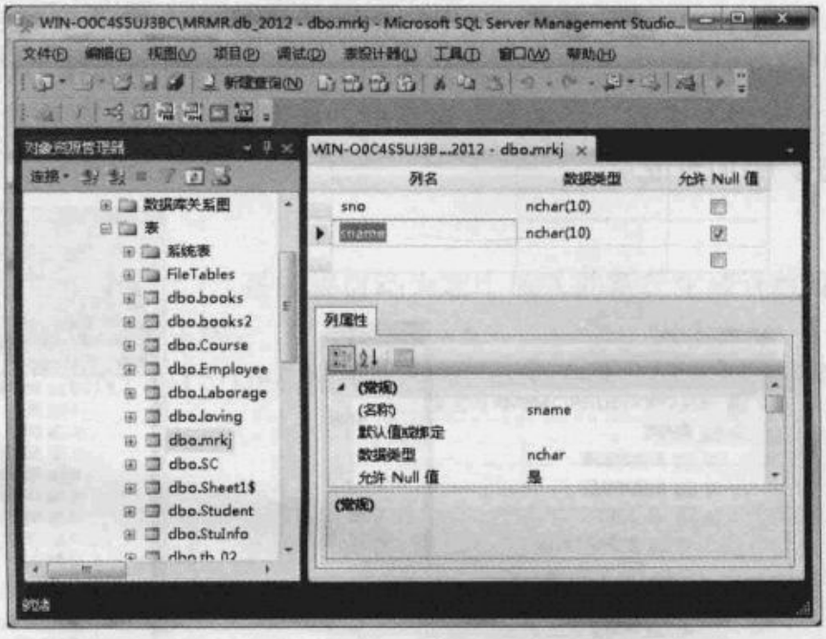


图 5.7　修改表字段

3. 删除数据表

下面介绍如何删除表 mrkj 的所有者。具体操作步骤如下：

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库，在“对象资源管理器”中展开“数据库”下面的“表”节点。

（2）右击需要删除的表 mrkj 选项，在弹出的快捷菜单中选择“删除”命令，如图 5.8 所示。

（3）进入“删除对象”窗口，如图 5.9 所示。通过该窗口可以删除数据表的相关选项。单击“确定”按钮，删除成功。

图 5.8　选择表删除

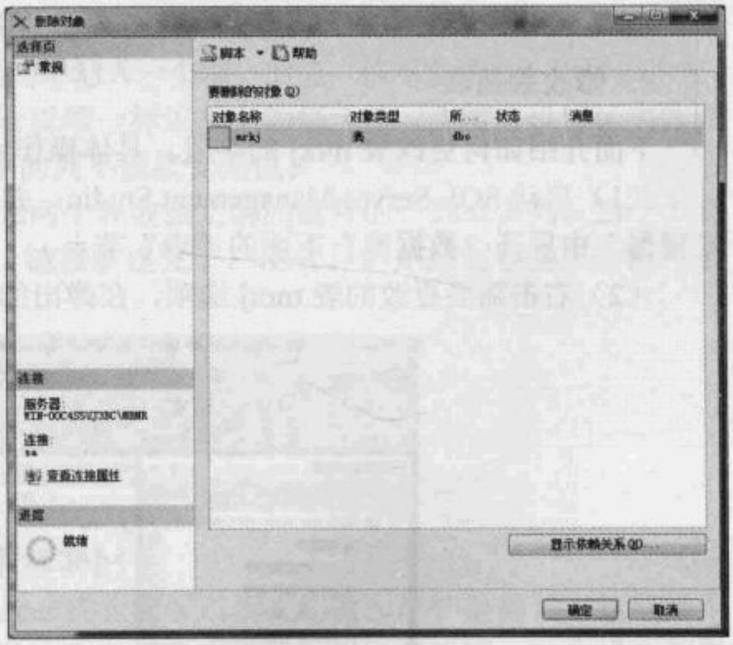


图 5.9　删除表

5.2.2　使用 CREATE TABLE 语句创建表

使用 CREATE TABLE 语句可以创建表，其基本语法如下：

```
CREATE TABLE
[ database_name . [ schema_name ] . | schema_name . ] table_name
        ( { <column_definition> | <computed_column_definition>
```

```
                    | <column_set_definition> }
[ <table_constraint> ] [ ,...n ] )
<column_definition> ::=
column_name <data_type>
    [ FILESTREAM ]
    [ COLLATE collation_name ]
    [ NULL | NOT NULL ]
    [
        [ CONSTRAINT constraint_name ] DEFAULT constant_expression ]
      | [ IDENTITY [ ( seed ,increment ) ] [ NOT FOR REPLICATION ]
    ]
    [ ROWGUIDCOL ] [ <column_constraint> [ ...n ] ]
    [ SPARSE ]
<computed_column_definition> ::=
column_name AS computed_column_expression
[ PERSISTED [ NOT NULL ] ]
[
    [ CONSTRAINT constraint_name ]
    { PRIMARY KEY | UNIQUE }
        [ CLUSTERED | NONCLUSTERED ]
        [
            WITH FILLFACTOR = fillfactor
          | WITH ( <index_option> [ , ...n ] )
        ]
    | [ FOREIGN KEY ]
        REFERENCES referenced_table_name [ ( ref_column ) ]
        [ ON DELETE { NO ACTION | CASCADE } ]
        [ ON UPDATE { NO ACTION } ]
        [ NOT FOR REPLICATION ]
    | CHECK [ NOT FOR REPLICATION ] ( logical_expression )
    [ ON { partition_scheme_name ( partition_column_name )
        | filegroup | "default" } ]
]
<column_set_definition> ::=
column_set_name XML COLUMN_SET FOR ALL_SPARSE_COLUMNS
< table_constraint > ::=
[ CONSTRAINT constraint_name ]
{
    { PRIMARY KEY | UNIQUE }
        [ CLUSTERED | NONCLUSTERED ]
                (column [ ASC | DESC ] [ ,...n ] )
        [
            WITH FILLFACTOR = fillfactor
          |WITH ( <index_option> [ , ...n ] )
        ]
        [ ON { partition_scheme_name (partition_column_name)
            | filegroup | "default" } ]
    | FOREIGN KEY
                ( column [ ,...n ] )
        REFERENCES referenced_table_name [ ( ref_column [ ,...n ] ) ]
        [ ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
```

```
        [ ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
        [ NOT FOR REPLICATION ]
    | CHECK [ NOT FOR REPLICATION ] ( logical_expression )
}
```

CREATE TABLE 语句的参数及说明如表 5.2 所示。

表 5.2　CREATE TABLE 语句的参数及说明

参　数	描　述
database_name	在其中创建表的数据库的名称。database_name 必须指定现有数据库的名称。如果未指定，则 database_name 默认为当前数据库
schema_name	新表所属架构的名称
table_name	新表的名称。表名必须遵循标识符规则。除了本地临时表名（以单个数字符号(#)为前缀的名称）不能超过 116 个字符外，table_name 最多可包含 128 个字符
<column_definition>	列定义
column_name	表中列的名称。列名必须遵循标识符规则并且在表中是唯一的
computed_column_expression	定义计算列的值的表达式
PERSISTED	指定 SQL Server 数据库引擎将在表中物理存储计算值，而且，当计算列依赖的任何其他列发生更新时对这些计算值进行更新
ON{<partion_scheme>|filegroup|"default"}	指定存储表的分区架构或文件组
<table_constraint>	表约束
CONSTRAINT	可选关键字，表示 PRIMARY KEY、NOT NULL、UNIQUE、FOREIGN KEY 或 CHECK 约束定义的开始
constraint_name	约束的名称。约束名称必须在表所属的架构中唯一
NULL | NOT NULL	确定列中是否允许使用空值
PRIMARY KEY	是通过唯一索引对给定的一列或多列强制实体完整性的约束。每个表只能创建一个 PRIMARY KEY 约束
UNIQUE	一个约束，该约束通过唯一索引为一个或多个指定列提供实体完整性。一个表可以有多个 UNIQUE 约束
CLUSTERED | NONCLUSTERED	指示为 PRIMARY KEY 或 UNIQUE 约束创建聚集索引还是非聚集索引。PRIMARY KEY 约束默认为 CLUSTERED，UNIQUE 约束默认为 NONCLUSTERED
column	用括号括起来的一列或多列，在表约束中表示这些列用在约束定义中
[ASC | DESC]	指定加入到表约束中的一列或多列的排序顺序。默认值为 ASC
WITH FILLFACTOR = fillfactor	指定数据库引擎存储索引数据时每个索引页的填充程度。用户指定的 fillfactor 值可以为介于 1～100 之间的任意值。如果未指定值，则默认值为 0
partition_scheme_name	分区架构的名称，该分区架构定义要将已分区表的分区映射到的文件组。数据库中必须存在该分区架构
[partition_column_name.]	指定对已分区表进行分区所依据的列
FOREIGN KEY REFERENCES	为列中的数据提供引用完整性的约束。FOREIGN KEY 约束要求列中的每个值在所引用的表中对应的被引用列中都存在
(ref_column [,... n])	是 FOREIGN KEY 约束所引用的表中的一列或多列
ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT }	指定如果已创建表中的行具有引用关系，并且被引用行已从父表中删除，则对这些行采取的操作。默认值为 NO ACTION

续表

参　数	描　述
NO ACTION	数据库引擎将引发错误，并回滚对父表中相应行的删除操作
CASCADE	如果从父表中删除一行，则将从引用表中删除相应行
SET NULL	如果父表中对应的行被删除，则组成外键的所有值都将设置为 NULL。若要执行此约束，外键列必须可为空值
SET DEFAULT	如果父表中对应的行被删除，则组成外键的所有值都将设置为默认值。若要执行此约束，所有外键列都必须有默认定义
ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT }	指定在发生更改的表中，如果行有引用关系且引用的行在父表中被更新，则对这些行采取什么操作。默认值为 NO ACTION
CHECK	一个约束，该约束通过限制可输入一列或多列中的可能值来强制实现域完整性。计算列上的 CHECK 约束也必须标记为 PERSISTED
logical_expression	返回 TRUE 或 FALSE 的逻辑表达式。别名数据类型不能作为表达式的一部分
NOT FOR REPLICATION	在 CREATE TABLE 语句中，可为 IDENTITY 属性、FOREIGN KEY 约束和 CHECK 约束指定 NOT FOR REPLICATION 子句

【例 5.1】 使用 CREATE TABLE 语句创建数据表 mingri，ID 字段为 int 类型并且不允许为空；Name 字段长度为 50 的 varchar 类型；Age 字段为 int 类型。（**实例位置：光盘\TM\sl\5\1**）

SQL 语句如下：

```
USE db_2012                          --打开数据库
CREATE TABLE [dbo].[mingri](         --创建表
    [ID] [int] NOT NULL,             --字段 ID，int 型，不能为空
    [Name] [varchar](50) ,           --Name 字段，varchar 类型
    [Age] [int]                      --Age 字段 int 类型
)
```

5.2.3　使用 ALTER TABLE 语句修改表结构

使用 ALTER TABLE 语句可以修改表的结构，语法如下：

```
ALTER TABLE [ database_name . [ schema_name ] . | schema_name . ] table_name
{
    ALTER COLUMN column_name
    {
        [ type_schema_name. ] type_name [ ( { precision [ , scale ]
            | max | xml_schema_collection } ) ]
[ COLLATE collation_name ]
        [ NULL | NOT NULL ]
| {ADD | DROP }
 { ROWGUIDCOL | PERSISTED| NOT FOR REPLICATION | SPARSE   }
    }
| [ WITH { CHECK | NOCHECK } ]
| ADD
```

```
    {
        <column_definition>
      | <computed_column_definition>
      | <table_constraint>
| <column_set_definition>
    } [ ,...n ]
    | DROP
    {
        [ CONSTRAINT ] constraint_name
        [ WITH ( <drop_clustered_constraint_option> [ ,...n ] ) ]
        | COLUMN column_name
    } [ ,...n ]
```

ALTER TABLE 语句的参数及说明如表 5.3 所示。

表 5.3 ALTER TABLE 语句的参数及说明

参　数	描　述
database_name	创建表时所在的数据库的名称
schema_name	表所属架构的名称
table_name	要更改的表的名称
ALTER COLUMN	指定要更改命名列
column_name	要更改、添加或删除的列的名称
[type_schema_name.] type_name	更改后的列的新数据类型或添加的列的数据类型
Precision	指定的数据类型的精度
scale	指定的数据类型的小数位数
max	仅应用于 varchar、nvarchar 和 varbinary 数据类型
xml_schema_collection	仅应用于 xml 数据类型
COLLATE < collation_name >	指定更改后的列的新排序规则
NULL | NOT NULL	指定列是否可接受空值
[{ADD | DROP} ROWGUIDCOL]	指定在指定列中添加或删除 ROWGUIDCOL 属性
[{ADD | DROP} PERSISTED]	指定在指定列中添加或删除 PERSISTED 属性
DROP NOT FOR REPLICATION	指定当复制代理执行插入操作时，标识列中的值将增加
SPARSE	指示列为稀疏列。稀疏列已针对 NULL 值进行了存储优化。不能将稀疏列指定为 NOT NULL
WITH CHECK | WITH NOCHECK	指定表中的数据是否用新添加的或重新启用的 FOREIGN KEY 或 CHECK 约束进行验证
ADD	指定添加一个或多个列定义、计算列定义或者表约束
DROP { [CONSTRAINT] constraint_name | COLUMN column_name }	指定从表中删除 constraint_name 或 column_name。可以列出多个列或约束
WITH <drop_clustered_constraint_option>	指定设置一个或多个删除聚集约束选项

【例 5.2】 向 db_2012 数据库中的 mingri 表中添加 Sex 字段。（**实例位置：光盘\TM\sl\5\2**）

SQL 语句如下：

```
USE db_2012
ALTER TABLE mingri
ADD   Sex char(2)
```

【例 5.3】 删除 db_2012 数据库中 mingri 中的 Sex 字段。（实例位置：光盘\TM\sl\5\3）

SQL 代码如下：

```
USE db_2012
ALTER TABLE mingri
DROP COLUMN Sex
```

5.2.4　使用 DROP TABLE 语句删除表

使用 DROP TABLE 语句可以删除数据表，其语法如下：

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

参数说明如下。

☑ database_name：要在其中删除表的数据库的名称。

☑ schema_name：表所属架构的名称。

☑ table_name：要删除的表的名称。

【例 5.4】 删除 db_2012 数据库中的数据表 mingri。（实例位置：光盘\TM\sl\5\4）

SQL 语句如下：

```
USE db_2012
DROP TABLE mingri
```

5.3　管 理 数 据

视频讲解：光盘\TM\lx\5\管理数据.mp4

对于数据库使用来说，设计好数据表只是一个框架而已，只有添加完数据的数据表才可以称为一个完整的数据表。

5.3.1　使用 INSERT 语句添加数据

INSERT 语句可以实现向表中添加新记录的操作。该语句可以向表中插入一条新记录或者插入一个结果集。其语法如下：

```
INSERT [ INTO]
table_or_view_name
VALUES
(expression) [,. . . n]
```

参数说明如下。

☑ table_or_view_name：要接收数据的表或视图的名称。

☑ VALUES：引入要插入的数据值的列表。

☑ expression：一个常量、变量或表达式。表达式不能包含 SELECT 或 EXECUTE 语句。

【例 5.5】 利用 INSERT 语句向数据表 Employee 添加数据记录。（**实例位置：光盘\TM\sl\5\5**）

SQL 语句如下：

```
USE db_2012
INSERT INTO Employee
(ID,Name,Sex,Age)VALUES(12,'雨涵','女',24,null)
```

【例 5.6】 如果要向表中添加所有的字段，可以省略要插入的数据的列名。（**实例位置：光盘\TM\sl\5\6**）

SQL 语句如下：

```
USE db_2012
INSERT INTO Employee
VALUES(13,'雨欣','女',24,NULL)
```

运行结果如图 5.10 所示。

结果 消息

	ID	Name	Sex	Age	备注
7	12	雨涵	女	24	NULL
8	13	雨欣	女	24	NULL
9	17	李立	男	26	NULL

图 5.10 INSERT 语句添加数据

5.3.2 使用 UPDATE 语句修改数据

修改数据表中不符合要求的数据或是错误的字段时，可以使用 UPDATE 语句进行修改。

UPDATE 语句修改数据的语法如下：

```
UPDATE table_or_view_name
[ FROM{ <table_source> } [ ,...n ] ]
SET
{ column_name = { expression | DEFAULT | NULL }
[ WHERE   <search_condition>]
```

UPDATE 语句的参数及说明如表 5.4 所示。

表 5.4 UPDATE 语句的参数及说明

参　数	描　述
table_or_view_name	要更新行的表或视图的名称
FROM <table_source>	指定将表、视图或派生表源用于为更新操作提供条件
expression	返回单个值的变量、文字值、表达式或嵌套 select 语句（加括号）
DEFAULT	指定用为列定义的默认值替换列中的现有值
WHERE	指定条件来限定所更新的行。根据所使用的 WHERE 子句的形式，有两种更新形式，分别为：（1）搜索更新指定搜索条件来限定要删除的行。（2）定位更新使用 CURRENT OF 子句指定游标。更新操作发生在游标的当前位置
<search_condition>	为要更新的行指定需满足的条件。搜索条件也可以是联接所基于的条件。对搜索条件中可以包含的谓词数量没有限制

【例 5.7】 将 Employee 表中所有员工的年龄加两岁。（**实例位置：光盘\TM\sl\5\7**）

SQL 语句如下：

```
USE db_2012
UPDATE Employee
SET Age=Age+2
```

【例 5.8】 将 Employee 表中“肖一子”的性别修改为女。（**实例位置：光盘\TM\sl\5\8**）

SQL 语句如下：

```
USE db_2012
UPDATE Employee
SET Sex='女'
WHERE Name='肖一子'
```

运行结果如图 5.11 所示。

结果　消息

	ID	Name	Sex	Age	备注
4	4	小李	女	27	明日科技
5	5	肖一子	女	27	NULL
6	6	赵一	男	26	NULL
7	12	雨涵	女	24	NULL

图 5.11　UPDATE 语句修改数据

5.3.3　使用 DELETE 语句删除数据

DELETE 语句用于从表或视图中删除行。语法如下：

```
DELETE
[ FROM <table_source> [ ,...n ] ]
[ WHERE { <search_condition> } ]
```

DELETE 语句的参数及说明如表 5.5 所示。

表 5.5　DELETE 语句的参数及说明

参　数	描　述
FROM <table_source>	指定将表、视图或派生表源用于为删除操作提供条件
WHERE	指定用于限制删除行数的条件。如果没有提供 WHERE 子句，则 DELETE 删除表中的所有行。基于 WHERE 子句中所指定的条件，有两种形式的删除操作，分别为：（1）搜索删除指定搜索条件以限定要删除的行；（2）定位删除使用 CURRENT OF 子句指定游标。删除操作在游标的当前位置执行。这比使用 WHERE search_condition 子句限定要删除的行的搜索 DELETE 语句更为精确。如果搜索条件不唯一标识单行，则搜索 DELETE 语句删除多行
<search_condition>	指定删除行的限定条件。对搜索条件中可以包含的谓词数量没有限制

【例 5.9】 删除 Employee 表中 ID 为 17 的员工的信息。（**实例位置：光盘\TM\sl\5\9**）

SQL 语句如下：

```
USE db_2012
DELETE FROM Employee WHERE ID=17
```

说明

在 DELETE 语句中如果不指定 WHERE 子句时，则删除表中的所有记录。

5.4 创建、删除和修改约束

视频讲解：光盘\TM\lx\5\创建、删除和修改约束.mp4

约束是 SQL Server 提供的自动强制数据完整性的一种方式，它是通过定义列的取值规则来维护数据的完整性，是强制完整性的标准机制。使用约束优先于使用触发器、规则和默认值。查询分析器也使用约束定义生成高性能的查询执行计划。

5.4.1 非空约束

列的为空性决定表中的行是否可为该列包含空值。空值（或 NULL）不同于零（0）、空白或长度为零的字符串（如""）。NULL 的意思是没有输入。出现 NULL 通常表示值未知或未定义。

（1）创建非空约束。

可以在用 CREATE TABLE 创建表时，使用 NOT NULL 关键字指定非空约束，其语法格式如下：

```
[CONSTRAINT    <约束名>]  NOT NULL
```

（2）修改非空约束。

修改非空约束的语法如下：

```
ALTER TABLE table_name
alter column column_name column_type   null | not null
```

参数说明如下。

- ☑ table_name：要修改非空约束的表名称。
- ☑ column_name：要修改非空约束的列名称。
- ☑ column_type：要修改非空约束的类型。
- ☑ null | not null：修改为空或者非空。

图 5.12 非空约束

【例 5.10】 修改 mingri 表中的非空约束，SQL 语句及运行结果如图 5.12 所示。（**实例位置：光盘\TM\sl\5\10**）

```
USE db_2012                          --打开数据库
ALTER TABLE mingri                   --更改表
alter column ID int null             --更改 ID 字段属性
```

5.4.2 主键约束

可以通过定义 PRIMARY KEY 约束来创建主键，用于强制表的实体完整性。一个表只能有一个

PRIMARY KEY 约束，并且 PRIMARY KEY 约束中的列不能接受空值。由于 PRIMARY KEY 约束可保证数据的唯一性，因此经常对标识列定义这种约束。

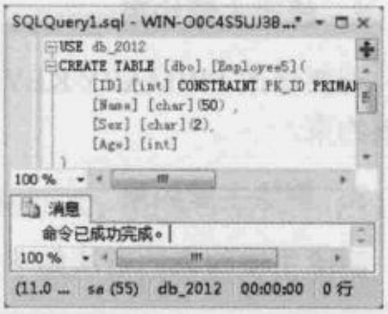


图 5.13　主键约束

1. 创建主键约束

（1）在创建表时创建主键约束。

【例 5.11】 创建数据表 Employee5，并将字段 ID 设置主键约束，SQL 语句及运行结果如图 5.13 所示。（**实例位置：光盘\TM\sl\5\11**）

```
USE db_2012                                          --打开数据库
CREATE TABLE [dbo].[Employee5](                      --创建表
    [ID] [int] CONSTRAINT PK_ID PRIMARY KEY,         --ID 字段，设为主键约束
    [Name] [char](50) ,
    [Sex] [char](2),
    [Age] [int]
)
```

说明

上面的代码中，CONSTRAINT PK_ID PRIMARY KEY 为创建一个主键约束，PK_ID 为用户自定义的主键约束名称，主键约束名称必须是合法的标识符。

（2）在现有表中创建主键约束。

在现有表中创建主键约束的语法如下：

```
ALTER TABLE table_name
ADD
CONSTRAINT constraint_name
PRIMARY KEY [CLUSTERED | NONCLUSTERED]
{(Column[,...n])}
```

参数说明如下。

- ☑ CONSTRAINT：创建约束的关键字。
- ☑ constraint_name：创建约束的名称。
- ☑ PRIMARY KEY：表示所创建约束的类型为主键约束。
- ☑ CLUSTERED | NONCLUSTERED：是表示为 PRIMARY KEY 或 UNIQUE 约束创建聚集或非聚集索引的关键字。PRIMARY KEY 约束默认为 CLUSTERED，UNIQUE 约束默认为 NONCLUSTERED。

【例 5.12】 将 mingri 表中的 ID 字段指定设置主键约束。（**实例位置：光盘\TM\sl\5\12**）

SQL 语句如下：

```
USE db_2012
ALTER TABLE mingri                                   --更改表
ADD CONSTRAINT PRM_son   PRIMARY KEY (ID)            --对 ID 字段添加主键约束
```

2. 修改主键约束

若要修改 PRIMARY KEY 约束，必须先删除现有的 PRIMARY KEY 约束，然后再用新定义重新创建该约束。

3. 删除主键约束

删除主键约束的语法如下：

```
ALTER TABLE table_name
DROP CONSTRAINT constraint_name[,...n]
```

【例 5.13】 删除 mignri 表中的主键约束。（**实例位置：光盘\TM\sl\5\13**）

SQL 语句如下：

```
USE db_2012
ALTER TABLE mingri
DROP CONSTRAINT PRM_son                    --删除主键约束
```

5.4.3 唯一约束

唯一约束 UNIQUE 用于强制实施列集中值的唯一性。根据 UNIQUE 约束，表中的任何两行都不能有相同的列值。另外，主键也强制实施唯一性，但主键不允许 NULL 作为一个唯一值。

图 5.14 唯一约束

1. 创建唯一约束

（1）在创建表时创建唯一约束。

【例 5.14】 在 db_2012 数据库中创建数据表 Employee6，并将字段 ID 设置唯一约束，如图 5.14 所示。（**实例位置：光盘\TM\sl\5\14**）

SQL 语句如下：

```
USE db_2012
CREATE TABLE [dbo].[Employee6](                --创建表
    [ID] [int] CONSTRAINT UQE_ID UNIQUE,       --设置唯一约束
    [Name] [char](50) ,
    [Sex] [char](2),
    [Age] [int]
)
```

（2）在现有表中创建唯一约束。

在现有表中创建唯一约束的语法如下：

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name
UNIQUE [CLUSTERED | NONCLUSTERED]
{(column [,...n])}
```

参数说明如下。

☑ table_name：要创建唯一约束的表名称。

☑ constraint_name：唯一约束名称。

☑ column：要创建唯一约束的列名称。

【例 5.15】 将 Employee6 表中的 ID 字段指定设置唯一约束。（**实例位置：光盘\TM\ sl\5\15**）

SQL 语句如下：

```
USE db_2012
ALTER TABLE Employee6
ADD CONSTRAINT Unique1_ID                    --设置唯一约束
UNIQUE(ID)
```

运行结果如图 5.15 所示。

图 5.15　修改唯一约束

2. 修改唯一约束

若要修改 UNIQUE 约束，必须首先删除现有的 UNIQUE 约束，然后用新定义重新创建。

3. 删除唯一约束

删除唯一约束的语法如下：

```
ALTER TABLE table_name
DROP CONSTRAINT constraint_name[,...n]
```

【例 5.16】 删除 Employee6 表中的唯一约束。（**实例位置：光盘\TM\sl\5\16**）

SQL 语句如下：

```
USE db_2012
ALTER TABLE Employee6
DROP CONSTRAINT Unique1_ID
```

5.4.4　检查约束

检查约束 CHECK 可以强制域的完整性。CHECK 约束类似于 FOREIGN KEY 约束，可以控制放入列中的值。但是，它们在确定有效值的方式上有所不同：FOREIGN KEY 约束从其他表获得有效值列表，而 CHECK 约束通过不基于其他列中的数据的逻辑表达式确定有效值。

1. 创建检查约束

（1）在创建表时创建检查约束。

【例 5.17】 创建数据表 Employee7，并将字段 Sex 设置检查约束，在输入性别字段时，只能接受“男”或者“女”，而不能接受其他数据。（**实例位置：光盘\TM\sl\5\17**）

SQL 语句如下：

```
USE db_2012
CREATE TABLE [dbo].[Employee7](
```

```
    [ID] [int],
    [Name] [char](50) ,
    [Sex] [char](2) CONSTRAINT CK_Sex Check(sex in('男','女')),
    [Age] [int]
)
```

运行结果如图 5.16 所示。

（2）在现有表中创建检查约束。

在现有表中创建检查约束的语法如下：

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name
CHECK (logical_expression)
```

参数说明如下。

☑ table_name：要创建检查约束的表名称。

☑ constraint_name：检查约束名称。

☑ logical_expression：要检查约束的条件表达式。

【例 5.18】 为 Employee5 表中的 Sex 字段设置检查约束，在输入性别的时候只能接受“女”，不能接受其他字段。（**实例位置：光盘\TM\sl\5\18**）

SQL 语句如下：

```
USE db_2012
ALTER TABLE [Employee5]
ADD CONSTRAINT Check_Sex Check(sex='女')
```

运行结果如图 5.17 所示。

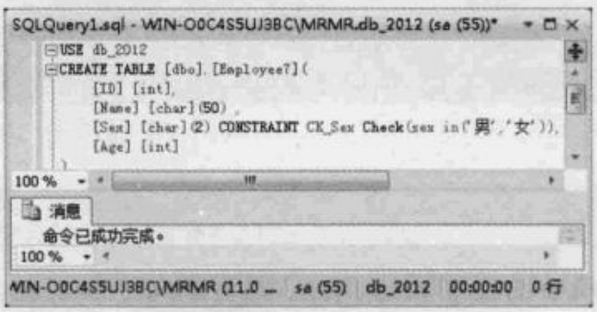


图 5.16 检查约束

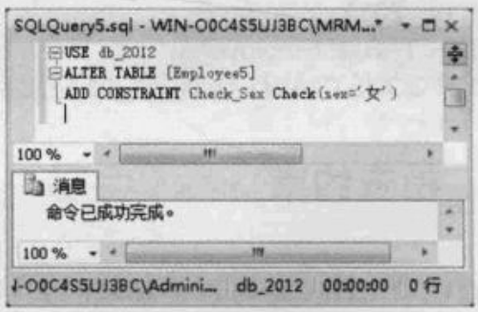


图 5.17 在现有表中创建检查约束

2. 修改检查约束

修改表中某列的 CHECK 约束使用的表达式，必须首先删除现有的 CHECK 约束，然后使用新定义重新创建，才能修改 CHECK 约束。

3. 删除检查约束

删除检查约束的语法如下：

```
ALTER TABLE table_name
DROP CONSTRAINT constraint_name[,...n]
```

删除 Employee5 表中的检查约束，SQL 语句如下：

```
USE db_2012
ALTER TABLE Employee5
DROP CONSTRAINT Check_Sex                    --删除约束
```

5.4.5　默认约束

在创建或修改表时可通过定义默认约束 DEFAULT 来创建默认值。默认值可以是计算结果为常量的任何值，例如常量、内置函数或数学表达式。这将为每一列分配一个常量表达式作为默认值。

1．创建默认约束

（1）在创建表时创建默认约束。

【例 5.19】 创建数据表 Employee8，并为字段 Sex 设置默认约束“女”。（**实例位置：光盘\TM\sl\5\19**）

SQL 语句如下：

```
USE db_2012
CREATE TABLE [dbo].[Employee8](                              --创建表
    [ID] [int],
    [Name] [char](50) ,
    [Sex] [char](2) CONSTRAINT Default_Sex Default '女',     --设置默认约束
    [Age] [int]
)
```

运行结果如图 5.18 所示。

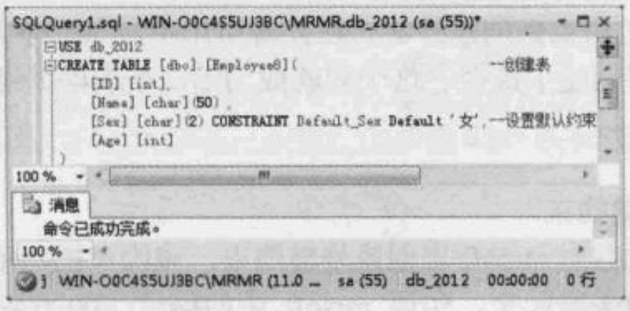


图 5.18　默认约束

（2）在现有表中创建默认约束。

在现有表中创建默认约束的语法如下：

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name
DEFAULT constant_expression [FOR column_name]
```

参数说明如下。

☑ table_name：要创建默认约束的表名称。

☑ constraint_name：默认约束名称。

☑ constant_expression：默认值。

【例 5.20】 为 Employee6 表中的 Sex 字段设置默认约束“男”。（实例位置：光盘\TM\sl\5\20）

SQL 语句如下：

```
ALTER TABLE [Employee6]
ADD CONSTRAINT Default_Sex_Man                --设置默认约束
DEFAULT '男' FOR Sex                          --默认值'男'
```

2. 修改默认约束

修改表中某列的 Default 约束使用的表达式，必须首先删除现有的 Default 约束，然后使用新定义重新创建，才能修改 Default 约束。

3. 删除默认约束

删除检查约束的语法如下：

```
ALTER TABLE table_name
DROP CONSTRAINT constraint_name[,...n]
```

【例 5.21】 删除 Employee6 表中的默认约束。（实例位置：光盘\TM\sl\5\21）

SQL 语句如下：

```
USE db_2012
ALTER TABLE Employee6
DROP CONSTRAINT Default_Sex_Man
```

5.4.6 外键约束

通过定义 FOREIGN KEY 约束来创建外键。在外键引用中，当一个表的列被引用作为另一个表的主键值的列时，就在两表之间创建了链接。这个列就成为第二个表的外键。

1. 创建外键约束

（1）在创建表时创建外键约束。

例 5.22 创建表 mrsoft，并为 mrsoft 表创建外键约束，该约束把 mrsoft 中的编号（ID）字段和表 Employe 中的编号（ID）字段关联起来，实现 mrsoft 中的编号（ID）字段的取值要参照表 Employee 中编号（ID）字段的数据值。（实例位置：光盘\TM\sl\5\22）

SQL 语句如下：

```
use db_2012
CREATE TABLE mrsoft                           --创建表
(
 ID INT ,
 Wage MONEY,
 CONSTRAINT FKEY_ID
 FOREIGN KEY (ID)                             --外键约束
 REFERENCES Employee(ID)
)
```

说明

FOREIGN KEY (ID)中的 ID 字段为 Employee 表中的编号（ID）字段。

（2）在现有表中创建默认约束。

在现有表中创建外键约束的语法如下：

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name
[FOREIGN KEY]{(column_name[,…n])}
  REFERENCES ref_table[(ref_column_name[,…n])]
```

创建外键约束语句的参数及说明如表 5.6 所示。

表 5.6　创建外键约束语句的参数及说明

参　数	描　述
table_name	要创建外键的表名称
constraint_name	外键约束名称
FOREIGN KEY... REFERENCES	为列中的数据提供引用完整性的约束。FOREIGN KEY 约束要求列中的每个值在被引用表中对应的被引用列中都存在。FOREIGN KEY 约束只能引用被引用表中为 PRIMARY KEY 或 UNIQUE 约束的列或被引用表中在 UNIQUE INDEX 内引用的列
ref_table	FOREIGN KEY 约束所引用的表名
(ref_column[,...n])	FOREIGN KEY 约束所引用的表中的一列或多列

【例 5.23】 将 Employee 表中的 ID 字段设置为 mrsoft 表中的外键。（实例位置：光盘\TM\sl\5\23）

SQL 语句如下：

```
use db_2012
ALTER TABLE mrsoft
ADD CONSTRAINT Fkey_ID_2                    --添加外键
FOREIGN KEY (ID)                            --所选字段为 ID
REFERENCES Employee(ID)                     --参照 Employee 表的 ID 字段
```

2. 修改外键约束

修改表中某列的 FOREIGN KEY 约束。必须首先删除现有的 FOREIGN KEY 约束，然后使用新定义重新创建，才能修改 FOREIGN KEY 约束。

3. 删除默认约束

删除外键约束的语法如下：

```
ALTER TABLE table_name
DROP CONSTRAINT constraint_name[,…n]
```

【例 5.24】 删除 Employee 表中的外键约束。（实例位置：光盘\TM\sl\5\24）

SQL 语句如下：

```
use db_2012
Alter Table mrsoft
Drop CONSTRAINT FKEY_ID
```

5.5 关系的创建与维护

视频讲解：光盘\TM\lx\5\关系的创建与维护.mp4

SQL Server 2012 是一个关系数据库管理系统（Relational Database Management System，RDMS），当数据库中包含多个表时，需要通过主关键字来建立表间的关系，使各表之间能够协调工作。

关系是通过匹配键列中的数据而工作的，而键列通常是两个表中具有相同名称的列，在数据表间创建关系可以显示某个表中的列连接到另一个表中的列。表与表之间存在 3 种类型的关系，所创建的关系类型取决于相关联的列是如何定义的。表与表之间存在的 3 种关系如下。

- ☑ 一对一关系。
- ☑ 一对多关系。
- ☑ 多对多关系。

5.5.1 一对一关系

一对一关系是指表 A 中的一条记录确实在表 B 中有且只有一条相匹配的记录。在一对一关系中，大部分相关信息都在一个表中。此关系的特点主要体现在以下几点：

- ☑ 分割一个含有许多列的表。
- ☑ 出于安全考虑而隔离表的某一部分。
- ☑ 存储可以很容易删除的临时数据，只需删除表即可删除这些数据。
- ☑ 存储只应用于主表子集的信息。
- ☑ 如果两个相关列都是主键或具有唯一约束，创建的就是一对一关系。

在学生管理系统中，Course 表用于存放课程的基础信息，这里定义为主表；teacher 表用于存放教师信息，这里定义为从表，且一个教师只能教一门课程。下面介绍如何通过这两张表创建一对一关系。

说明

“一个教师只能教一门课程”，在这里不考虑一名教师教多门课程的情况。例如，英语专业的英语老师，只能教英语。

操作步骤如下：

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库。

（2）在“对象资源管理器”中展开“数据库”节点，展开指定的数据库 db_2012。

（3）右击 Course 表，在弹出的快捷菜单中选择“设计”命令。

（4）在表设计器界面中，右键单击 Cno 字段，在弹出的快捷菜单中选择“关系”命令，打开“外键关系”对话框，在该窗体中单击“添加”按钮，如图 5.19 所示。

（5）在“外键关系”对话框中，单击“常规”下面的“表和列规范”文本框中的…按钮，添加表和列规范属性，弹出“表和列”对话框，在该对话框中设置关系名及主外键的表，如图 5.20 所示。

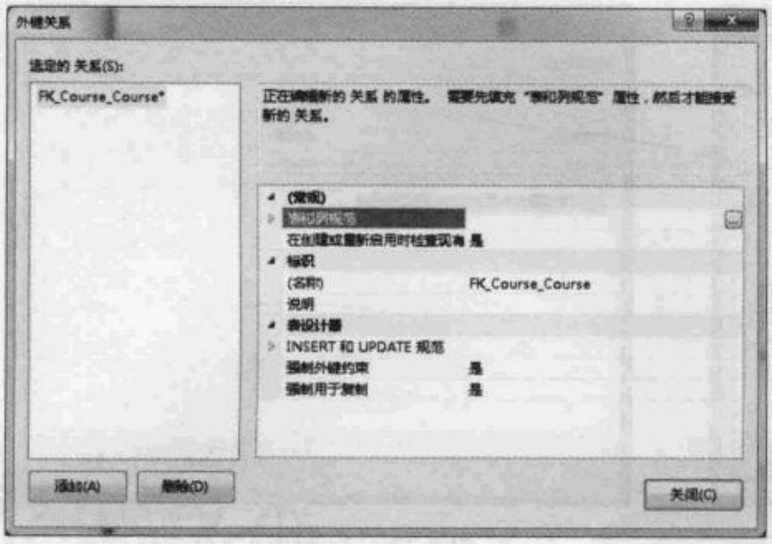


图 5.19　“外键关系”对话框

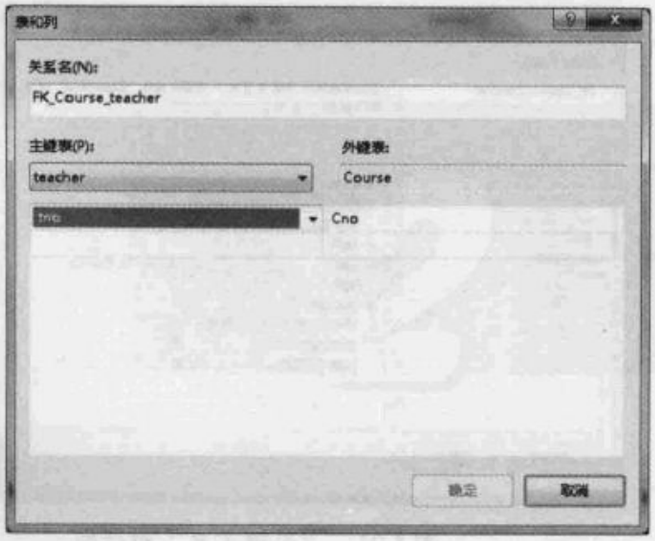


图 5.20　“表和列”对话框

（6）在“表和列”对话框中，单击“确定”按钮，返回到“外键关系”对话框，在“外键关系”对话框中单击“关闭”按钮，完成一对一关系的创建。

注意

创建一对一关系之前，tno、Cno 都应该设置为这两个表的主键，且关联字段类型必须相同。

5.5.2　一对多关系

一对多关系是最常见的关系类型，是指表 A 中的行可以在表 B 中有许多匹配行，但是表 B 中的行只能在表 A 中有一个匹配行。

如果在相关列中只有一列是主键或具有唯一约束，则创建的是一对多关系。例如，Student 表用于存储学生的基础信息，这里定义为主表；Course 表用于存储课程的基础信息，一个学生可以学多门课程，这里定义为从表。下面介绍如何通过这两张表创建一对多关系。

操作步骤如下：

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库。

（2）在“对象资源管理器”中展开“数据库”节点，展开指定的数据库 db_2012。

（3）右击 Course 表，在弹出的快捷菜单中选择“设计”命令。

（4）在表设计器界面中，右键单击 Cno 字段，在弹出的快捷菜单中选择“关系”命令，打开“外键关系”对话框，在该对话框中单击“添加”按钮，如图 5.21 所示。

（5）在“外键关系”对话框中，单击“常规”下面的“表和列规范”文本框中的…按钮，选择要创建一对多关系的数据表和列。弹出“表和列”对话框，在该对话框中设置关系名及主外键的表，如图 5.22 所示。

（6）在“表和列”对话框中，单击“确定”按钮，返回到“外键关系”对话框，在“外键关系”对话框中单击“关闭”按钮，完成一对多关系的创建。

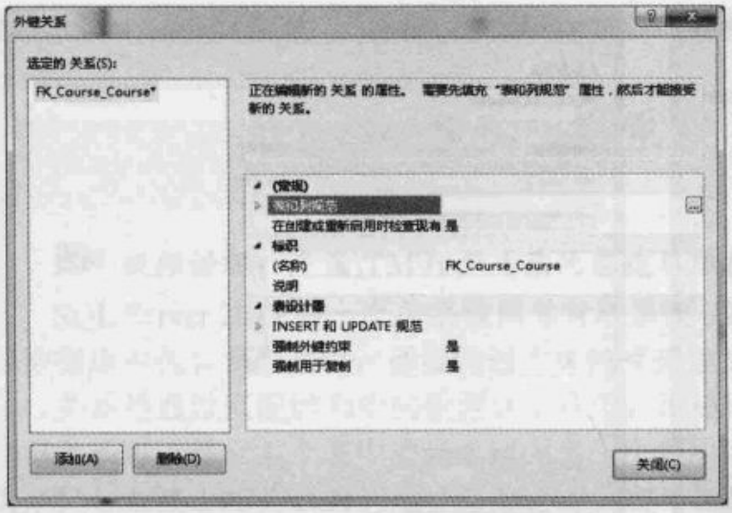


图 5.21 “外键关系”对话框

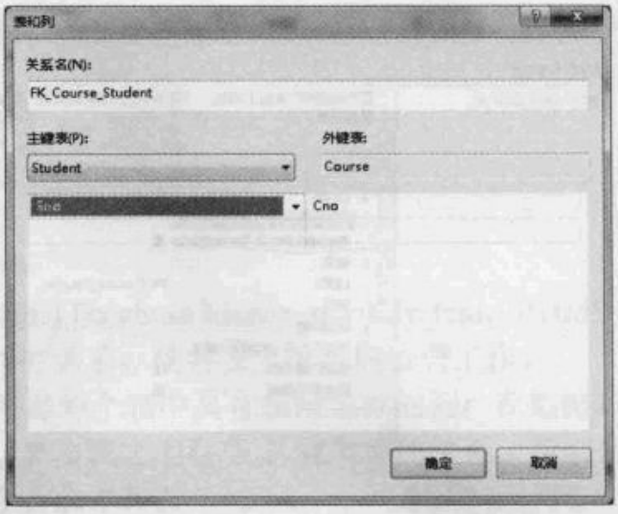


图 5.22 “表和列”对话框

5.5.3 多对多关系

多对多关系是指关系中每个表的行在相关表中具有多个匹配行。在数据库中，多对多关系的建立是依靠第 3 个表即连接表实现的，连接表包含相关的两个表的主键列，然后从两个相关表的主键列分别创建与连接表中匹配列的关系。

例如，通过“商品信息表”与“商品订单表”创建多对多关系。首先就需要建立一个连接表（如“商品订单信息表”），该表中应该包含上述两个表的主键列，然后“商品信息表”和“商品订单表”分别与连接表建立一对多关系，以此来实现“商品信息表”和“商品订单表”的多对多关系。

5.6 小　　结

本章介绍了 SQL Server 2012 数据类型、使用企业管理器创建数据表、修改数据表和删除数据表的方法以及创建视图、修改视图和删除视图的方法。读者应在掌握 SQL Server 2012 数据类型的基础上，熟练运用企业管理器创建和设计表，针对表创建视图并能够通过视图实现对表的操作。

5.7 实践与练习

1. 使用 create table、select * from 和 insert into 实现批量插入数据。首先创建图书信息表 books2，然后在 INSERT INTO 语句中查询数据表 books 中出版社是“人邮”的图书信息，将查询结果插入到数据表 books2 中。（**答案位置：光盘\TM\sl\5\25**）

2. 使用 create 语句创建一个 loving 数据表，该表中只有一个 int 类型的 ID 字段，然后使用 sp_help 存储过程查看该表的相关信息。（**答案位置：光盘\TM\sl\5\26**）

3. 通过 select 语句查询表 Employee 中备注字段为空和不为空的数据。（**答案位置：光盘\TM\sl\5\27**）

第2篇

核心技术

- 第6章 SQL基础
- 第7章 SQL函数的使用
- 第8章 SQL数据查询基础
- 第9章 SQL数据高级查询
- 第10章 视图的使用

本篇介绍SQL基础、SQL函数的使用、SQL数据查询基础、SQL数据高级查询、视图的使用等。学习完这一部分，能够了解和熟悉T-SQL及常用的函数，使用T-SQL操作SQL Server 2012数据库中的视图，掌握SQL查询、子查询、嵌套查询、联接查询的用法等。

第 6 章

SQL 基础

（视频讲解：51 分钟）

本章将介绍 T-SQL 的基础知识，包括数据类型、常量、变量、运算符、流程控制语句以及一些常用的命令等，学习这些内容后读者可以熟悉 T-SQL，掌握 SQL Server 2012 的基础，有助于进一步的学习和程序开发。

通过阅读本章，您可以：

- 了解 T-SQL 的基本概念
- 掌握常量的使用
- 掌握变量的使用
- 熟悉注释符、运算符与通配符
- 掌握流程控制语句
- 熟悉一些常用的命令

6.1　T-SQL 概述

视频讲解：光盘\TM\lx\6\T-SQL 概述.mp4

T-SQL（Transact Structured Query Language）是标准的 Microsoft SQL Server 的扩展，是标准的 SQL 程序设计语言的增强版，是用来让程序与 SQL Server 沟通的主要语言。

SQL 是关系数据库系统的标准语言，标准的 SQL 语句几乎可以在所有的关系型数据库上不加修改地使用。Access、Visual FoxPro、Oracle 这样的数据库同样支持标准的 SQL，但这些关系数据库不支持 T-SQL，T-SQL 是 SQL Server 系统产品独有的。

6.1.1　T-SQL 的组成

T-SQL 主要由以下几部分组成。

- ☑ 数据定义语言（Data Definition Language，DDL）：用于在数据库系统中对数据库、表、视图、索引等数据库对象进行创建和管理。
- ☑ 数据控制语言（Data Control Language，DCL）：用于实现对数据库中数据的完整性、安全性等的控制。
- ☑ 数据操纵语言（Data Manipulation Language，DML）：用于插入、修改、删除和查询数据库中的数据。

6.1.2　T-SQL 语句结构

每条 SQL 语句均由一个谓词（Verb）开始，该谓词描述这条语句要产生的动作，例如 SELECT 或 UPDATE 关键字。谓词后紧接着一个或多个子句（Clause），子句中给出了被谓词作用的数据或提供谓词动作的详细信息。每一条子句都由一个关键字开始。下面介绍 SELECT 语句的主要结构。

语法如下：

```
SELECT  子句
[INTO 子句]
FROM 子句
[WHERE 子句]
[GROUP  BY 子句]
[HAVING  子句]
[ORDER BY 子句]
```

【例 6.1】 在 db_2012 数据库中查询 Student 表中女同学的信息。在查询分析器中运行的结果如图 6.1 所示。（**实例位置：光盘\TM\sl\6\1**）

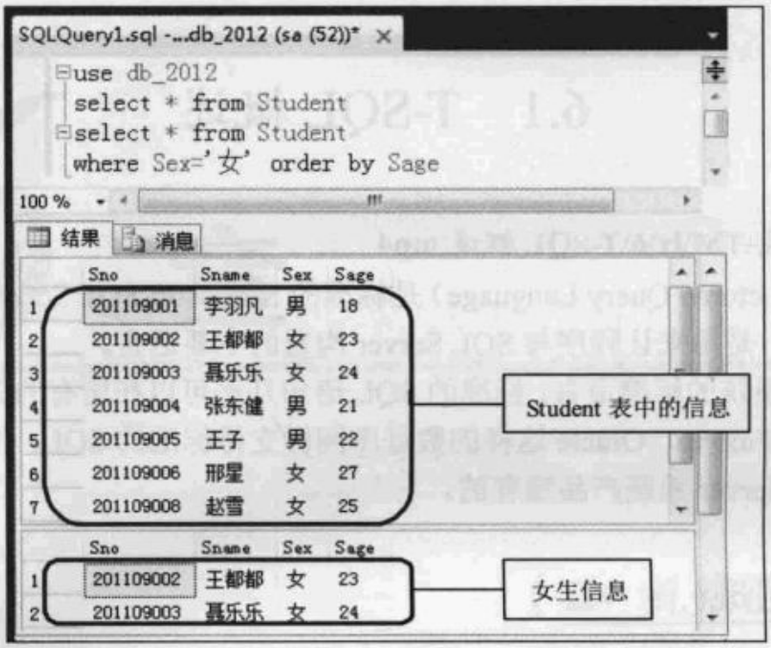


图 6.1　查询 Student 表中女生的信息

SQL 语句如下：

```
use db_2012
select * from Student
where Sex='女' order by Sage
```

6.1.3　T-SQL 语句分类

Transact-SQL 语句的分类如下。

（1）变量说明语句：用来说明变量的命令。

（2）数据定义语句：用来建立数据库、数据库对象和定义列，大部分是以 CREATE 开头的命令，如 CREATE TABLE、CREATE VIEW 和 DROP TABLE 等。

（3）数据操纵语句：用来操纵数据库中数据的命令，如 SELECT、INSERT、UPDATE、DELETE 和 CURSOR 等。

（4）数据控制语句：用来控制数据库组件的存取许可、存取权限等命令，如 GRANT、REVOKE 等。

（5）流程控制语句：用于设计应用程序流程的语句，如 IF WHILE 和 CASE 等。

（6）内嵌函数：说明变量的命令。

（7）其他命令：嵌于命令中使用的标准函数。

6.2　常　　量

视频讲解：光盘\TM\lx\6\常量.mp4

数据在内存中存储始终不变化的量叫常量。常量，也称为文字值或标量值，是表示一个特定数据

值的符号。常量的格式取决于它所表示的值的数据类型。

6.2.1　数字常量

数字常量包括整数常量、小数常量以及浮点常量。

整数和小数常量在 SQL 中被写成普通的小数数字，前面可加正负号，例如：

```
12，-37，200.45
```

在数字常量的各个位之间不要加逗号，例如，123123 这个数字不能表示为：123,123。

浮点常量使用符号 e 来指定，例如：

```
1.5e3, -3.14e1,2.5e-7
```

e 读作“乘 10 的几次幂”。

6.2.2　字符串常量

字符串常量括在单引号内并包含字母数字字符（a～z、A～Z 和 0～9）以及特殊字符，如感叹号（!）、at 符（@）和数字号（#）。

如果单引号中的字符串包含一个嵌入的引号，可以使用两个单引号表示嵌入的单引号。

以下是字符串的示例：

```
'MingRi'
'O'  'Brien'
'Process X is 50% complete.'
```

6.2.3　日期和时间常量

SQL 规定日期、时间和时间间隔的常量值被指定为日期和时间常量。

例如：'1984-03-10 ' ,'03/03/1976'。

日期和时间根据国家不同，书写方式也不同。例如，美国表示为 mm/dd/yyyy，欧洲表示为 dd.mm.yyyy，日本表示为 yyyy-mm-dd 等。

6.2.4　符号常量

除了用户提供的常量外，SQL 包含几个特有的符号常量，这些常量代表不同的常用数据值。

例如，CURRENT_DATE 表示当前的日期，类似的如 CURRENT_TIME、CURRENT_TIMESTAMP 等。

这些符号常量也可以通过 SQL Server 的内嵌函数访问。

6.3 变　　量

视频讲解：光盘\TM\lx\6\变量.mp4

数据在内存中存储可以变化的量叫变量。为了在内存中存储信息，用户必须指定存储信息的单元，并为该存储单元命名，以方便获取信息，这就是变量的功能。Transact-SQL 可以使用两种变量，一种是局部变量，另外一种是全局变量。局部变量和全局变量的主要区别在于存储的数据作用范围不一样。

6.3.1 局部变量

局部变量是用户可自定义的变量，它的作用范围仅在程序内部。局部变量的名称是用户自定义的，命名的局部变量名要符合 SQL Server 2012 标识符命名规则，局部变量名必须以@开头。

1. 声明局部变量

局部变量的声明需要使用 DECLARE 语句。

语法如下：

```
DECLARE
{
@varaible_name datatype    [ ,... n  ]
}
```

参数说明如下。

- ☑ @varaible_name：局部变量的变量名，必须以@开头，另外变量名的形式必须符合 SQL Server 标识符的命名方式。
- ☑ datatype：局部变量使用的数据类型，可以是除 text、ntext 或者 image 类型外所有的系统数据类型和用户自定义数据类型。一般来说，如果没有特殊的用途，建议在应用时尽量使用系统提供的数据类型。这样做可以减少维护应用程序的工作量。

例如，声明局部变量@ songname。

SQL 语句如下：

```
declare  @songname    char(10)
```

2. 为局部变量赋值

为变量赋值的方式一般有两种，一种是使用 SELECT 语句，一种是使用 SET 语句。使用 SELECT 语句为变量赋值的语法如下：

```
SELECT    @varible_name    =  expression
[FROM     table_name [ ,... n ]
WHERE     clause  ]
```

上面的 SELECT 语句的作用是为了给变量赋值，而不是为了从表中查询出数据。而且在使用 SELECT 语句进行赋值的过程中，并不一定非要使用 FROM 关键字和 WHERE 子句。

【例 6.2】在 db_2012 数据库的 tb_Student 表中，把“所学专业”是“会计学”的信息赋值给局部变量@songname，并把它的值用 print 关键字显示出来。在查询分析器中运行的结果如图 6.2 所示。（**实例位置：光盘\TM\ sl\6\2**）

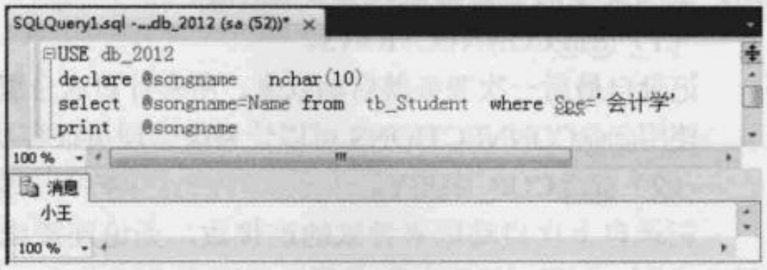


图 6.2　把查询内容赋值给局部变量

SQL 语句如下：

```
USE db_2012
declare @songname    nchar(10)
select   @songname=Name from   tb_Student   where Spe='会计学'
print    @songname
```

SELECT 语句赋值和查询不能混淆，例如，声明一个局部变量名是@b 并给它赋值的 SQL 语句如下：

```
declare   @b   int
select   @b=1
```

另一种为局部变量赋值的方式是使用 SET 语句。使用 SET 语句对变量进行赋值的常用语法如下：

```
{ SET   @varible_name   =   ecpression } [ ,... n ]
```

下面是一个简单的赋值语句：

```
DECLARE   @song   char(20)
SET   @song   =   'I   love   flower'
```

还可以为多个变量一起赋值，相应的 SQL 语句如下：

```
declare   @b   int, @c   char(10),@a   int
select @b=1, @c='love',@a=2
```

注意

数据库语言和编程语言有一些关键字，关键字是在某一环境下能够促使某一操作发生的字符组。为避免冲突和产生错误，在命名表、列、变量以及其他对象时应避免使用关键字。

6.3.2　全局变量

全局变量是 SQL Server 系统内部事先定义好的变量，不用用户参与定义，对用户而言，其作用范围并不局限于某一程序，而是任何程序均可随时调用。全局变量通常用于存储一些 SQL Server 的配置设定值和效能统计数据。

SQL Server 一共提供了 30 多个全局变量，本节只对一些常用的全局变量的功能和使用方法进行介绍。全局变量的名称都是以@@开头的。

（1）@@CONNECTIONS。

记录自最后一次服务器启动以来，所有针对这台服务器进行的连接数目，包括没有连接成功的尝试。使用@@CONNECTIONS 可以让系统管理员很容易地得到今天所有试图连接本服务器的连接数目。

（2）@@CUP_BUSY。

记录自上次启动以来尝试的连接数，无论连接成功还是失败，都以 ms 为单位的 CPU 工作时间。

（3）@@CURSOR_ROWS。

返回在本次服务器连接中，打开游标取出数据行的数目。

（4）@@DBTS。

返回当前数据库中 timestamp 数据类型的当前值。

（5）@@ERROR。

返回执行上一条 Transact-SQL 语句所返回的错误代码。

在 SQL Server 服务器执行完一条语句后，如果该语句执行成功，则将返回@@ERROR 的值为 0，如果该语句执行过程中发生错误，则将返回错误的信息，而@@ERROR 将返回相应的错误编号，该标号将一直保持下去，直到下一条语句得到执行为止。

由于@@ERROR 在每一条语句执行后被清除并且重置，应在语句验证后立即检查它，或将其保存到一个局部变量中以备事后查看。

（6）@@FETCH_STATUS。

返回上一次使用游标 FETCH 操作所返回的状态值，且返回值为整型。

@@FETCH_STATUS 全局变量的返回值描述如表 6.1 所示。

表 6.1　@@FETCH_STATUS 返回值的描述

返　回　值	描　　述
0	FETCH 语句成功
−1	FETCH 语句失败或此行不在结果集中
−2	被提取的行不存在

例如，到了最后一行数据后，还要接着取下一行数据等，返回的值为−2，表示返回的值已经丢失。

（7）@@IDENTITY。

返回最近一次插入的 identity 列的数值，返回值是 numeric。

（8）@@IDLE。

返回以 ms 为单位计算 SQL Server 服务器自最近一次启动以来处于停顿状态的时间。

（9）@@IO_BUSY。

返回以 ms 为单位计算的 SQL Server 服务器自最近一次启动以来花在输入和输出上的时间。

（10）@@LOCK_TIMEOUT。

返回当前对数据锁定的超时设置。

（11）@@PACK_RECEIVED。

返回 SQL Server 服务器自最近一次启动以来一共从网络上接收数据分组的数目。

（12）@@PACK_SENT。

返回 SQL Server 服务器自最近一次启动以来一共向网络上发送数据分组的数目。

（13）@@PROCID。

返回当前存储过程的 ID 标识。

（14）@@REMSERVER。

返回在登录记录中记载远程 SQL Server 服务器的名字。

（15）@@ROWCOUNT。

返回上一条 SQL 语句所影响到数据行的数目。对所有不影响数据库数据的 SQL 语句，这个全局变量返回的结果是 0。在进行数据库编程时，经常要检测@@ROWCOUNT 的返回值，以便明确所执行的操作是否达到了目标。

（16）@@SPID。

返回当前服务器进程的 ID 标识。

（17）@@TOTAL_ERRORS。

返回自 SQL Server 服务器启动以来，所遇到读写错误的总数。

（18）@@TOTAL_READ。

返回自 SQL Server 服务器启动以来，读磁盘的次数。

（19）@@TOTAL_WRITE。

返回自 SQL Server 服务器启动以来，写磁盘的次数。

（20）@@TRANCOUNT。

返回当前连接中，处于活动状态事务的数目。

（21）@@VERSION。

返回当前 SQL Server 服务器安装日期、版本，以及处理器的类型。

6.4　注释符、运算符与通配符

视频讲解：光盘\TM\lx\6\注释符、运算符与通配符.mp4

注释符是对代码给出解释或说明。运算符是 Transact-SQL 重要的部分，常见的运算符有：算术运算符、赋值运算符、比较运算符、逻辑运算符等。常用的通配符有%、_（下划线）、[ ]、[^]。

6.4.1　注释符

注释语句不是可执行语句，不参与程序的编译，通常是一些说明性的文字，对代码的功能或者代码的实现方式给出简要的解释和提示。

在 Transact-SQL 中，可使用以下两类注释符。

☑　ANSI 标准的注释符（--），用于单行注释；例如下面 SQL 语句所加的注释。

```
use  pubs     --打开数据表
```

☑ 与 C 语言相同的程序注释符号，即“/*”、“*/”。“/*”用于注释文字的开头，“*/”用于注释文字的结尾，可在程序中标识多行文字为注释。

例如，有多行注释的 SQL 语句如下：

```
USE  db_2012
declare @songname    char(10)
select  @songname=Stu_col  from  tb_Student  where Stu_spe='会计学'
print    @songname
/*打开 db_2012 数据库，定义一个变量
把查询到的结果赋值给所定义的变量*/
```

说明

把所选的行一次都注释的快捷键是 Shift+Ctrl+C；一次取消多行注释的快捷键是 Shift+Ctrl+R。

6.4.2 运算符

运算符是一种符号，用来进行常量、变量或者列之间的数学运算和比较操作，它是 Transact-SQL 很重要的部分。运算符有几种类型，分别为：算术运算符、赋值运算符、比较运算符、逻辑运算符、位运算符、连接运算符。

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。

算术运算符包括：+（加）、-（减）、×（乘）、/（除）、%（取余）。

【例 6.3】 求 2 对 5 取余。在查询分析器中运行的结果如图 6.3 所示。（**实例位置：光盘\TM\sl\6\3**）

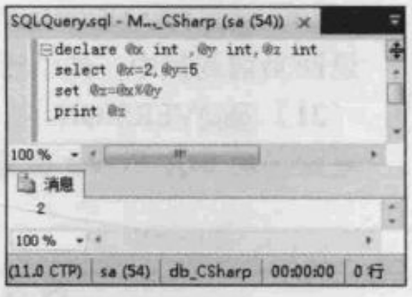


图 6.3 求 2%5 的结果

SQL 语句如下：

```
declare @x int ,@y int,@z int
select @x=2,@y=5
set @z=@x%@y
print @z
```

注意

取余运算两边的表达式必须是整型数据。

2. 赋值运算符

T-SQL 有一个赋值运算符，即等号（=）。在下面的示例中，创建了@songname 变量。然后利用赋值运算符将@songname 设置成一个由表达式返回的值。代码如下：

```
DECLARE @songname   char(20)
SET @songname='loving'
```

还可以使用 SELECT 语句进行赋值，并输出该值。

```
DECLARE @songname    char(20)
SELECT @songname ='loving'
print @songname
```

3．比较运算符

比较运算符测试两个表达式是否相同。除了 text、ntext 或 image 数据类型的表达式外，比较运算符可以用于所有的表达式。比较运算符包括：>（大于）、<（小于）、=（等于）、>=（大于等于）、<=（小于等于）、<>（不等于）、!=（不等于）、!>（不大于）、!<（不小于），其中，!=、!<、!>不是 ANSI 标准的运算符。

比较运算符的结果为布尔数据类型，它有 3 个值：TRUE、FALSE 及 UNKNOWN。那些返回布尔数据类型的表达式被称为布尔表达式。

和其他 SQL Server 数据类型不同，不能将布尔数据类型指定为表列或变量的数据类型，也不能在结果集中返回布尔数据类型。

例如：3>5=FALSE,6<>9=TRUE。

4．逻辑运算符

逻辑运算符对某个条件进行测试，以获得其真实情况。逻辑运算符和比较运算符一样，返回带有 TRUE 或 FALSE 值的布尔数据类型。SQL 支持的逻辑运算符如表 6.2 所示。

表 6.2　SQL 支持的逻辑运算符

运　算　符	行　　为
ALL	如果一个比较集中全部都是 TRUE，则值为 TRUE
AND	如果两个布尔表达式均为 TRUE，则值为 TRUE
ANY	如果一个比较集中任何一个为 TRUE，则值为 TRUE
BETWEEN	如果操作数是在某个范围内，则值为 TRUE
EXISTS	如果子查询包含任何行，则值为 TRUE
IN	如果操作数与一个表达式列表中的某个相等的话，则值为 TRUE
LIKE	如果操作数匹配某个模式的话，则值为 TRUE
NOT	对任何其他布尔运算符的值取反
OR	如果任何一个布尔表达式是 TRUE，则值为 TRUE
SOME	如果一个比较集中的某些为 TRUE 的话，则值为 TRUE

【例 6.4】 在 Student 表中，查询女生中年龄大于 24 岁的学生信息。在查询分析器中运行的结果如图 6.4 所示。（**实例位置：光盘\TM\sl\6\4**）

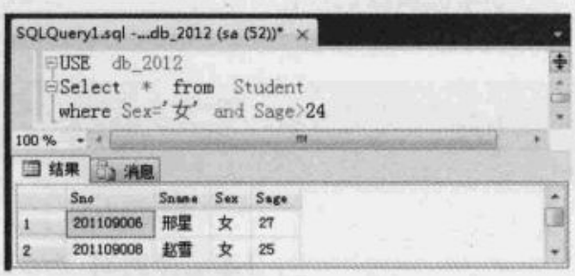


图 6.4　查询年龄大于 24 的女生信息

SQL 语句如下：

```
USE  db_2012
Select  *  from  Student
where Sex='女' and Sage>24
```

当 NOT、AND 和 OR 出现在同一表达式中时，优先级是：NOT、AND、OR。

例如：3>5 or 6>3 and not 6>4=FALSE。

先计算 not 6>4=FALSE；然后再计算 6>3 AND FALSE =FALSE，最后计算 3>5 or FALSE= FALSE。

5. 位运算符

位运算符的操作数可以是整数数据类型或二进制串数据类型（image 数据类型除外）范畴的。SQL 支持的位运算符如表 6.3 所示。

表 6.3　位运算符

运　算　符	说　　明	运　算　符	说　　明
&	按位 AND	^	按位互斥 OR
|	按位 OR	~	按位 NOT

6. 字符串连接运算符

连接运算符“+”用于连接两个或两个以上的字符或二进制串、列名或者串和列的混合体，将一个串加入到另一个串的末尾。

语法如下：

```
<expression1>+<expression2>
```

【例 6.5】 用“+”连接两个字符串。在查询分析器中运行的结果如图 6.5 所示。（**实例位置：光盘\TM\sl\6\5**）

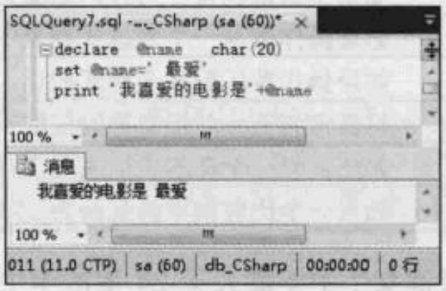


图 6.5　用“+”连接两个字符串

SQL 语句如下：

```
declare  @name   char(20)
set @name=' 最爱'
print '我喜爱的电影是'+@name
```

7. 运算符优先级

当一个复杂表达式中包含多个运算符时，运算符的优先级决定了表达式计算和比较操作的先后顺

序。运算符的优先级由高到低的顺序如下。

（1）+（正）、-（负）、～（位反）

（2）*（乘）、/（除）、%（取余）

（3）+（加）、+（字符串串联运算符）、-（减）

（4）=、>、<、>、=、<、=、<、>、!、=、!、>、!、<（比较运算符）

（5）^（按位异或）、&（按位与）、|（按位或）

（6）NOT

（7）AND

（8）ALL ANY BETWEEN IN LIKE OR SOME（逻辑运算符）

（9）=（赋值）

若表达式中含有相同优先级的运算符，则从左向右依次处理。还可以使用括号来提高运算的优先级，在括号中的表达式优先级最高。如果表达式有嵌套的括号，那么首先对嵌套最内层的表达式求值。例如：

```
DECLARE  @num  int
SET @num = 2 * (4 + (5 - 3) )
```

上面的代码中，先计算（5-3），然后再加 4，最后再和 2 相乘。

6.4.3　通配符

匹配指定范围内或者属于方括号所指定的集合中的任意单个字符。可以在涉及模式匹配的字符串比较（例如，LIKE 和 PATINDEX）中使用这些通配符。

在 SQL 中通常用 LIKE 关键字与通配符结合起来实现模糊查询。其中，SQL 支持的通配符如表 6.4 所示。

表 6.4　SQL 支持的通配符的描述和示例

通　配　符	描　　述	示　　例
%	包含零个或更多字符的任意字符	"loving%"可以表示："loving"、"loving you"、"loving?"
_（下划线）	任何单个字符	"loving_"可以表示："lovingc"，后面只能再接一个字符
[]	指定范围([a～f])或集合([abcdef])中的任何单个字符	[0～9]123 表示以 0～9 之间任意一个字符开头，以'123'结尾的字符
[^]	不属于指定范围([a～f])或集合([abcdef])的任何单个字符	[^0～5]123 表示不以 0～5 之间任意一个字符开头，却以'123'结尾的字符

6.5　流 程 控 制

视频讲解：光盘\TM\lx\6\流程控制.mp4

流程控制语句是用来控制程序执行流程的语句。使用流程控制语句可以提高编程语言的处理能力。

与程序设计语言（如 C 语言）一样，Transact-SQL 提供的流程控制语句如下：

```
BEGIN...END      WAITFOR      GOTO
WHILE            IF...ELSE    BREAK
RETURN           CONTINUE
```

6.5.1 BEGIN...END

BEGIN...END 语句用于将多个 Transact-SQL 语句组合为一个逻辑块。当流程控制语句必须执行一个包含两条或两条以上的 T-SQL 语句的语句块时，使用 BEGIN...END 语句。

语法如下：

```
BEGIN
{sql_statement...}
END
```

其中，sql_statement 是指包含的 Transact-SQL 语句。

BEGIN 和 END 语句必须成对使用，任何一条语句均不能单独使用。BEGIN 语句后为 Transact-SQL 语句块，END 语句行指示语句块结束。

【例 6.6】 在 BEGIN...END 语句块中完成把两个变量的值交换。在查询分析器中运行的结果如图 6.6 所示。（**实例位置：光盘\TM\sl\6\6**）

图 6.6 交换两个变量的值

SQL 语句如下：

```
declare   @x int,   @y int,@t int
set @x=1
set @y=2
begin
set @t=@x
set @x=@y
set @y=@t
end
print @x
print @y
```

此例子不用 BEGIN...END 语句结果也完全一样，但 BEGIN...END 和一些流程控制语句结合起来就有作用了。在 BEGIN...END 中可嵌套另外的 BEGIN...END 来定义另一程序块。

6.5.2　IF

在 SQL Server 中为了控制程序的执行方向，也会像其他语言（如 C 语言）一样有顺序、选择和循环 3 种控制语句，其中，IF 就属于选择判断结构。IF 结构的语法如下：

```
IF<条件表达式>
    {命令行|程序块}
```

其中，<条件表达式>可以是各种表达式的组合，但表达式的值必须是逻辑值“真”或“假”。其中命令行和程序块可以是合法的 Transact-SQL 任意语句，但含两条或两条以上语句的程序块必须加 BEGIN...END 子句。

执行顺序是：遇到选择结构 IF 子句，先判断 IF 子句后的条件表达式，如果条件表达式的逻辑值是“真”，就执行后面的命令行或程序块，然后再执行 IF 结构下一条语句；如果条件式的逻辑值是“假”，就不执行后面的命令行或程序块，直接执行 IF 结构的下一条语句。

【例 6.7】 判断数字“3”是否是正数。在查询分析器中运行的结果如图 6.7 所示。（**实例位置：光盘\TM\sl\6\7**）

SQL 语句如下：

```
declare   @x int
set @x=3
if @x>0
print '@x 是正数'
print'end'
```

【例 6.8】 判断数字“8”的奇偶性，在查询分析器中运行的结果如图 6.8 所示。（**实例位置：光盘\TM\sl\6\8**）

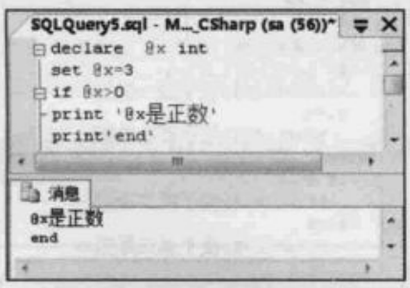


图 6.7　判断“3”的正负

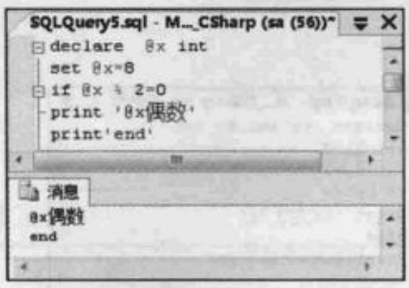


图 6.8　判断一个数的奇偶性

SQL 语句如下：

```
declare   @x int
set @x=8
if @x % 2=0
print '@x 偶数'
print'end'
```

6.5.3 IF...ELSE

IF 选择结构可以带 ELSE 子句。IF...ELSE 的语法如下：

```
IF<条件表达式>
    {命令行 1|程序块 1}
[ELSE
    {命令行 2|程序块 2}
```

如果逻辑判断表达式返回的结果是“真”，那么程序接下来会执行命令行 1 或程序块 1；如果逻辑判断表达式返回的结果是“假”，那么程序接下来会执行命令行 2 或程序块 2。无论哪种情况，最后都要执行 IF...ELSE 语句的下一条语句。

【例 6.9】 判断两个数“8”和“3”的大小。在查询分析器中运行的结果如图 6.9 所示。（**实例位置：光盘\TM\sl\6\9**）

SQL 语句如下：

```
declare   @x int,@y int
set @x=8
set @y=3
if @x>@y
print '@x 大于@y'
else
print'@x 小于等于@y'
```

IF...ELSE 结构还可以嵌套解决一些复杂的判断。

【例 6.10】 输入一个坐标值，然后判断它在哪一个象限。在查询分析器中的运行结果如图 6.10 所示。（**实例位置：光盘\TM\sl\6\10**）

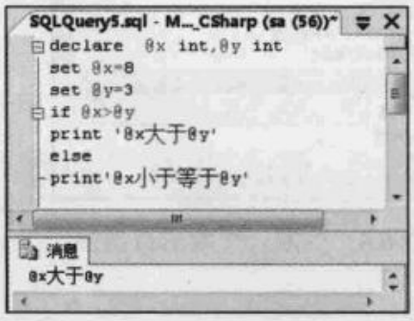


图 6.9 判断两个数的大小

图 6.10 判断坐标位于的象限

SQL 语句如下：

```
declare   @x int,@y int
set @x=8
```

```
set @y=-3
if @x>0
  if @y>0
    print'@x@y 位于第一象限'
  else
    print'@x@y 位于第四象限'
else
  if @y>0
    print'@x@y 位于第二象限'
  else
    print'@x@y 位于第三象限'
```

6.5.4　CASE

使用 CASE 语句可以很方便地实现多重选择的情况，比 IF...THEN 结构有更多的选择和判断的机会，从而可以避免编写多重的 IF...THEN 嵌套循环。

Transact-SQL 支持 CASE 有以下两种语句格式。

简单 CASE 函数：

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

CASE 搜索函数：

```
CASE
    WHEN Boolean_expression THEN result_expression
         [ ...n ]
    [
         ELSE else_result_expression
    END
```

CASE 函数的参数及说明如表 6.5 所示。

表 6.5　CASE 函数的参数及说明

参　数	描　述
input_expression	使用简单 CASE 格式时所计算的表达式。input_expression 是任何有效的 Microsoft® SQL Server™ 表达式
WHEN when_expression	使用简单 CASE 格式时 input_expression 所比较的简单表达式。when_expression 是任意有效的 SQL Server 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
n	占位符，表明可以使用多个 WHEN when_expression THEN result_expression 子句或 WHEN Boolean_expression THEN result_expression 子句

续表

参　数	描　述
THEN result_expression	当 input_expression = when_expression 取值为 TRUE，或者 Boolean_expression 取值为 TRUE 时返回的表达式。result_expression 是任意有效的 SQL Server 表达式
ELSE else_result_expression	当比较运算取值不为 TRUE 时返回的表达式。如果省略此参数并且比较运算取值不为 TRUE，CASE 将返回 NULL 值 else_result_expression 是任意有效的 SQL Server 表达式。else_result_expression 和所有 result_expression 的数据类型必须相同，或者必须是隐性转换
WHEN Boolean_expression	使用 CASE 搜索格式时所计算的布尔表达式。Boolean_expression 是任意有效的布尔表达式

下面介绍简单 CASE 函数和 CASE 搜索函数两种格式的执行顺序。

☑ 简单 CASE 函数

（1）计算 input_expression，然后按指定顺序对每个 WHEN 子句的 input_expression=when_expression 进行计算。

（2）如果 input_expression = when_expression 为 TRUE，则返回 result_expression。

（3）如果没有取值为 TRUE 的 input_expression = when_expression，则当指定 ELSE 子句时，SQL Server 将返回 else_result_expression；若没有指定 ELSE 子句，则返回 NULL 值。

☑ CASE 搜索函数

（1）按指定顺序为每个 WHEN 子句的 Boolean_expression 求值。

（2）返回第一个取值为 TRUE 的 Boolean_expression 的 result_expression。

（3）如果没有取值为 TRUE 的 Boolean_expression，则当指定 ELSE 子句时，SQL Server 将返回 else_result_expression；若没有指定 ELSE 子句，则返回 NULL 值。

【例 6.11】 在 tb_Grade 表中，查询每个同学的成绩。如果成绩大于等于 90，显示成绩优秀，如果成绩小于 90 大于等于 80，显示成绩良好。如果成绩小于 80 大于等于 70，显示成绩及格。否则将显示不及格。在查询分析器中的运行结果如图 6.11 所示。查询前的 tb_Grade 表如图 6.12 所示。（**实例位置：光盘\TM\sl\6\11**）

```
select *,
备注=case
when Grade>=90 then '成绩优秀'
when Grade<90 and Grade>=80  then '成绩良好'
when Grade<80 and Grade>=70  then '成绩及格'
else '不及格'
end
from tb_Grade
```

100 %

结果　消息

	ID	Name	Subjet	Grade	备注
1	1	婷子	语文	90	成绩优秀
2	2	小明	语文	85	成绩良好
3	3	小心	语文	70	成绩及格
4	1	婷子	数学	65	成绩良好
5	2	小明	数学	95	成绩优秀
6	3	小心	数学	65	不及格

图 6.11　用 case 查询 tb_Grade 表

ID	Name	Subjet	Grade
1	婷子	语文	90
2	小明	语文	85
3	小心	语文	70
1	婷子	数学	85
2	小明	数学	95
3	小心	数学	65

图 6.12　查询前的 tb_Grade 表

SQL 语句如下：

```
use db_2012
go
```

```
select *,
备注=case
when Grade>=90 then '成绩优秀'
when Grade<90 and Grade>=80   then '成绩良好'
when Grade<80 and Grade>=70   then '成绩及格'
else '不及格'
end
from tb_Grade
```

6.5.5　WHILE

WHILE 子句是 T-SQL 语句支持的循环结构。在条件为真的情况下，WHILE 子句可以循环地执行其后的一条 T-SQL 命令。如果想循环执行一组命令，则需要配合 BEGIN...END 子句使用。WHILE 的语法如下：

```
WHILE<条件表达式>
BEGIN
    <命令行|程序块>
END
```

遇到 WHILE 子句，先判断条件表达式的值，当条件表达式的值为“真”时，执行循环体中的命令行或程序块，遇到 END 子句会自动地再次判断条件表达式的值的真假，决定是否执行循环体中的语句。只能当条件表达式的值为“假”时，才结束执行循环体的语句。

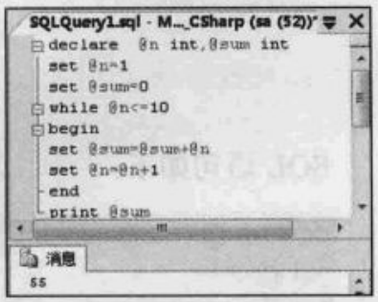


图 6.13　求 1～10 之间整数的和

【例 6.12】 求 1～10 之间整数的和。在查询分析器中运行的结果如图 6.13 所示。（**实例位置：光盘\TM\sl\6\12**）

SQL 语句如下：

```
declare   @n int,@sum int
set @n=1
set @sum=0
while @n<=10
begin
set @sum=@sum+@n
set @n=@n+1
end
print @sum
```

6.5.6　WHILE...CONTINUE...BREAK

循环结构 WHILE 子句还可以用 CONTINUE 和 BREAK 命令控制 WHILE 循环中语句的执行。语法如下：

```
WHILE<条件表达式>
BEGIN
```

```
    <命令行|程序块>
    [BREAK]
    [CONTINUE]
    [命令行|程序块]
END
```

其中，CONTINUTE 命令可以让程序跳过 CONTINUE 命令之后的语句，回到 WHILE 循环的第一行命令。BREAK 命令则让程序完全跳出循环，结束 WHILE 命令的执行。

【例 6.13】 求 1～10 之间偶数的和，并用 CONTINUE 控制语句的输出。在查询分析器中运行的结果如图 6.14 所示。（**实例位置：光盘\TM\sl\6\13**）

图 6.14 求 1～10 之间偶数的和

SQL 语句如下：

```
declare @x int,@sum int
set @x=1
set @sum=0
while @x<10
begin
set @x=@x+1
if @x%2=0
set @sum=@sum+@x
else
continue
end
print @sum
```

6.5.7 RETURN

RETURN 语句用于从查询或过程中无条件退出。RETURN 语句可在任何时候用于从过程、批处理或语句块中退出。位于 RETURN 之后的语句不会被执行。

语法如下：

```
RETURN[整数值]
```

在括号内可指定一个返回值。如果没有指定返回值，SQL Server 系统会根据程序执行的结果返回一个内定值，内定值如表 6.6 所示。

表 6.6　RETURN 命令返回的内定值

返　回　值	含　　义	返　回　值	含　　义
F	程序执行成功	−7	资源错误，如磁盘空间不足
−1	找不到对象	−8	非致命的内部错误
−2	数据类型错误	−9	已达到系统的极限
−3	死锁	−10 或−11	致命的内部不一致性错误
−4	违反权限原则	−12	表或指针破坏
−5	语法错误	−13	数据库破坏
−6	用户造成的一般错误	−14	硬件错误

【例 6.14】RETURN 语句实现退出功能。在查询分析器中运行的结果如图 6.15 所示。（**实例位置：光盘\TM\sl\6\14**）

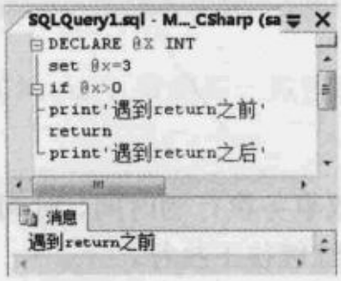


图 6.15　RETURN 实现退出功能

SQL 语句如下：

```
DECLARE @X INT
set @x=3
if @x>0
print'遇到 return 之前'
return
print'遇到 return 之后'
```

6.5.8　GOTO

GOTO 命令用来改变程序执行的流程，使程序跳到标识符指定的程序行再继续往下执行。

语法如下：

```
GOTO　标识符
```

标识符需要在其名称后加上一个冒号“:”。

例如：“33:”，“loving:”。

【例 6.15】 用 GOTO 语句实现跳转输出小于等于 3 的值。在查询分析器中执行的结果如图 6.16 所示。（**实例位置：光盘\TM\sl\6\15**）

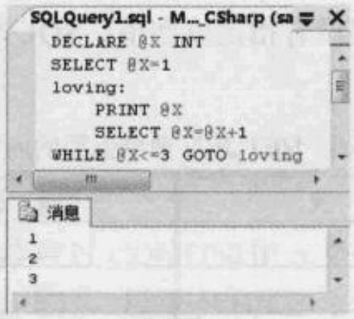


图 6.16　GOTO 实现跳转功能

SQL 语句如下：

```
DECLARE    @X INT
SELECT    @X=1
loving:
    PRINT @X
    SELECT @X=@X+1
WHILE   @X<=3    GOTO    loving
```

6.5.9　WAITFOR

WAITFOR 指定触发器、存储过程或事务执行的时间、时间间隔或事件；还可以用来暂时停止程序的执行，直到所设定的等待时间已过才继续往下执行。

语法如下：

```
WAITFOR DELAY<'时间'>|TIME<'时间'>
```

其中，“时间”必须为 DATETIME 类型的数据，如“11:15:27”，但不能包括日期。各关键字含义如下。

☑ DELAY：用来设定等待的时间，最多可达 24 小时。

☑ TIME：用来设定等待结束的时间点。

【例 6.16】 等待 3s 后显示“祝你节日快乐!”，运行结果如图 6.17 所示。（**实例位置：光盘\TM\sl\6\16**）

图 6.17　等待 3s 后输出信息

SQL 语句如下：

```
WAITFOR DELAY'00:00:03'
PRINT'祝你节日快乐！'
```

【例 6.17】 15 点显示“《新三国演义》开始了”。（**实例位置：光盘\TM\sl\6\17**）

SQL 语句如下：

```
WAITFOR TIME'15:00:00'
PRINT'《新三国演义》开始了！"
```

6.6 常 用 命 令

视频讲解：光盘\TM\lx\6\常用命令.mp4

本节介绍 SQL Server 中常用的命令，如常见的输出命令、数据备份命令、数据还原命令等，应用这些命令可以提高数据库的完整性和安全性。

6.6.1 DBCC

DBCC（Database Base Consistency Checker，数据库一致性检查程序）命令用于验证数据库完整性、查找错误和分析系统使用情况等。

DBCC 命令后必须加上子命令系统才知道要做什么。

1. DBCC CHECKALLOC

检查指定数据库的磁盘空间分配结构的一致性。

【例 6.18】 执行 DBCC CHECKALLOC 命令检测 db_2012 数据库磁盘空间分配结构。在查询分析器中运行的结果如图 6.18 所示。（**实例位置：光盘\TM\sl\6\18**）

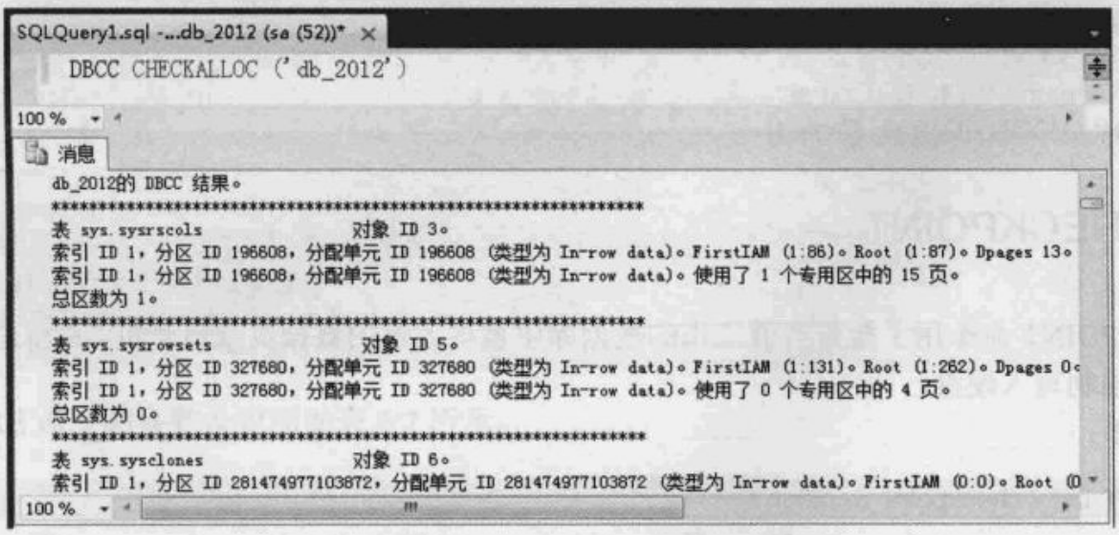


图 6.18　检测 db_2012 数据库磁盘空间分配结构

SQL 语句如下：

```
DBCC CHECKALLOC (' db_2012')
```

2. DBCC SHOWCONTIG

显示指定表的数据和索引的碎片信息。

【例 6.19】 使用 OBJECT_ID 获得表 ID，使用 sys.indexes 获得索引 ID，使用 DBCC SHOWCONTIG 显示指定表的数据和索引的碎片信息。在查询分析器中运行的结果如图 6.19 所示。（**实例位置：光盘\TM\sl\6\19**）

图 6.19 获得表 ID 和索引 ID

SQL 语句如下：

```
DECLARE @id int, @indid int
SET @id = OBJECT_ID('tb_Course')
SELECT @indid = index_id
FROM sys.indexes
WHERE object_id   = @id
      AND name = 'PK_tb_Course'
DBCC SHOWCONTIG(@id, @indid)
```

6.6.2 CHECKPOINT

CHECKPOINT 命令用于检查当前工作的数据库中被更改过的数据页或日志页，并将这些数据从数据缓冲器中强制写入硬盘。

语法如下：

```
CHECKPOINT [ checkpoint_duration ]
```

参数 checkpoint_duration 表示以秒为单位指定检查点完成所需的时间。如果指定 checkpoint_duration，则 SQL Server 数据库引擎会在请求的持续时间内尝试执行检查点。checkpoint_duration 必须是一个数据类型为 int 的表达式，并且必须大于零。如果省略该参数，SQL Server 数据库引擎将自动调整检查点持续时间，以便最大程度地降低对数据库应用程序性能的影响。

CHECKPOINT 权限默认授予 sysadmin 固定服务器角色、db_owner 和 db_backupoperator 固定数据库角色的成员且不可转让。

【例 6.20】 使用 CHECKPOINT 命令检查 db_2012 数据库中被更改过的数据页或日志页。在查询分析器中运行的结果如图 6.20 所示。（**实例位置：光盘\TM\sl\6\20**）

图 6.20　使用 CHECKPOINT 命令检查 db_2012 数据库

SQL 语句如下：

```
Use db_2012
CHECKPOINT
```

6.6.3　DECLARE

DECLARE 命令用于声明一个或多个局部变量、游标变量或表变量。

语法如下：

```
DECLARE
      {
{{ @local_variable [AS] data_type } | [ = value ] }
    | { @cursor_variable_name CURSOR }
} [,...n]
    | { @table_variable_name [AS] <table_type_definition> | <user-defined table type> }
<table_type_definition> ::=
      TABLE ( { <column_definition> | <table_constraint> } [ ,... ]
       )
<column_definition> ::=
            column_name { scalar_data_type | AS computed_column_expression }
      [ COLLATE collation_name ]
      [ [ DEFAULT constant_expression ] | IDENTITY [ ( seed ,increment ) ] ]
      [ ROWGUIDCOL ]
      [ <column_constraint> ]
<column_constraint> ::=
      { [ NULL | NOT NULL ]
      | [ PRIMARY KEY | UNIQUE ]
      | CHECK ( logical_expression )
      }
```

DECLARE 命令的参数及说明如表 6.7 所示。

表 6.7　DECLARE 命令的参数及说明

参　数	描　述
@ local_variable	变量的名称。变量名必须以“@”符开头。局部变量名称必须符合标识符规则
data_type	用户定义表类型或别名数据类型。变量的数据类型不能是 text、ntext 或 image
= value	以内联方式为变量赋值。值可以是常量或表达式，但它必须与变量声明类型匹配，或者可隐式转换为该类型

续表

参　数	描　述
@cursor_variable_name	游标变量的名称
CURSOR	指定变量是局部游标变量
@table_variable_name	table 类型的变量的名称
<table_type_definition>	定义 table 数据类型。表声明包括列定义、名称、数据类型和约束
n	指示可以指定多个变量并对变量赋值的占位符
column_name	表中的列的名称
scalar_data_type	指定列是标量数据类型
computed_column_expression	定义计算列值的表达式
[COLLATE collation_name]	指定列的排序规则
DEFAULT	如果在插入过程中未显式提供值，则指定为列提供的值
constant_expression	用作列的默认值的常量、NULL 或系统函数
IDENTITY	指示新列是标识列
Seed	是装入表的第一行所使用的值
Increment	添加到以前装载的列标识值的增量值
ROWGUIDCOL	指示新列是行的全局唯一标识符列
NULL | NOT NULL	决定在列中是否允许 Null 值的关键字
PRIMARY KEY	通过唯一索引对给定的一列或多列强制实现实体完整性的约束
UNIQUE	通过唯一索引为给定的一列或多列提供实体完整性的约束
CHECK	一个约束，该约束通过限制可输入一列或多列中的可能值来强制实现域完整性
logical_expression	返回 TRUE 或 FALSE 的逻辑表达式

【例 6.21】 定义一个变量，SQL 语句如下：

```
declare  @x  int
```

如果定义的变量是字符型，应该指定 data_type 表达式中其最大长度，否则系统认为其长度为 1。

【例 6.22】 定义一字符变量，SQL 语句如下：

```
declare  @c  char(8)
```

【例 6.23】 使用 DECLARE 命令定义多个变量，中间用逗号相隔，SQL 语句如下：

```
declare  @x  int ,@y  char(8),@z   datetime
```

6.6.4 PRINT

PRINT 命令向客户端返回一个用户自定义的信息，即显示一个字符串（最长为 255 个字符）、局部变量或全局变量的内容。

语法如下：

```
PRINT msg_str | @local_variable | string_expr
```

参数说明如下。

☑ msg_str：字符串或 Unicode 字符串常量。

☑ @local_variable：任何有效的字符数据类型变量。@local_variable 的数据类型必须为 char 或 varchar，或者必须能够隐式转换为这些数据类型。

☑ string_expr：返回字符串的表达式。可包括串联的文字值、函数和变量。

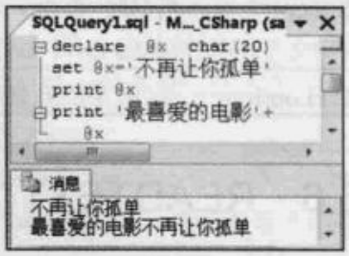


图 6.21　使用 print 命令输出信息

【例 6.24】 定义一个变量，为其赋值。用 print 命令显示变量，并生成字符串。在查询分析器中运行的结果如图 6.21 所示。（**实例位置：光盘\TM\sl\6\21**）

SQL 语句如下：

```
declare  @x  char(20)
set @x='不再让你孤单'
print @x
print '最喜爱的电影'+
   @x
```

6.6.5　RAISERROR

RAISERROR 命令用于在 SQL Server 系统中返回错误信息时同时返回用户指定的信息。

语法如下：

```
RAISERROR ( { msg_id | msg_str | @local_variable }
   { , severity , state }
   [ , argument [ ,...n ] ] )
[ WITH option [ ,...n ] ]
```

RAISERROR 命令的参数及说明如表 6.8 所示。

表 6.8　RAISERROR 命令的参数及说明

参　数	描　述
msg_id	存储于 sysmessages 表中的用户定义的错误信息。用户定义错误信息的错误号应大于 50000。由特殊消息产生的错误是第 50000 号
msg_str	是一条特殊消息，其格式与 C 语言中使用的 PRINTF 格式样式相似。此错误信息最多可包含 400 个字符。如果该信息包含的字符超过 400 个，则只能显示前 397 个并将添加一个省略号以表示该信息已被截断。所有特定消息的标准消息 ID 是 14000。msg_str 支持的格式有% [[flag] [width] [precision] [{h | l}]] type
@local_variable	是一个可以为任何有效字符数据类型的变量，其中包含的字符串的格式化方式与 msg_str 相同。@local_variable 必须为 char 或 varchar，或者能够隐式转换为这些数据类型
severity	用户定义的与消息关联的严重级别。用户可以使用 0～18 之间的严重级别。19～25 之间的严重级别只能由 sysadmin 固定服务器角色成员使用。若要使用 19～25 之间的严重级别，必须将 WITH option 设置为 WTHLOG
state	1～127 的任意整数，表示有关错误调用状态的信息。state 的值默认为 1

续表

参　数	描　述
argument	用于取代在 msg_str 中定义的变量或取代对应于 msg_id 的消息的参数。可以有 0 或更多的替代参数；然而，替代参数的总数不能超过 20 个。每个替代参数可以是局部变量或这些任意数据类型，如 int1、int2、int4、char、varchar、binary 或 varbinary。不支持其他数据类型
WITH option	错误的自定义选项

6.6.6 READTEXT

READTEXT 命令用于读取 text、ntext 或 image 列中的值，从指定的位置开始读取指定的字符数。语法如下：

```
READTEXT { table.column  text_ptr offset  size } [ HOLDLOCK ]
```

READTEXT 命令的参数及说明如表 6.9 所示。

表 6.9　READTEXT 命令的参数及说明

参　数	描　述
table.column	从中读取的表和列的名称。表名和列名必须符合标识符的规则。必须指定表名和列名，不过可以选择是否指定数据库名称和所有者名称
text_ptr	有效文本指针。text_ptr 必须是 binary(16)
offset	开始读取 text、image 或 ntext 数据之前跳过的字节数（使用 text 或 image 数据类型时）或字符数（使用 ntext 数据类型时）
size	要读取数据的字节数（使用 text 或 image 数据类型时）或字符数（使用 ntext 数据类型时）。如果 size 是 0，则表示读取了 4KB 的数据
HOLDLOCK	使文本值一直锁定到事务结束。其他用户可以读取该值，但是不能对其进行修改

6.6.7 BACKUP

计算机在操作过程中难免出现意外，为了保证用户数据的安全性，防止数据库中的数据意外丢失，应对数据库及时进行备份。BACKUP 命令用于将数据库内容或其事务处理日志备份到存储介质上（硬盘或磁带等）。

BACKUP 命令的语法如下：

```
Backup Database { database_name | @database_name_var }
To < backup_device > [ ,...n ]
[ <MIRROR TO clause>][ next-mirror-to ]
[ WITH { DIFFERENTIAL | <general_WITH_options> [ ,...n ] }]
[ ; ]
```

BACKUP 命令的参数及说明如表 6.10 所示。

表 6.10　READTEXT 命令的参数及说明

参　数	描　述
Backup Database	关键字
{ database_name | @database_name_var }	备份事务日志、部分数据库或完整的数据库时所用的源数据库。如果作为变量 (@database_name_var) 提供，则可以将该名称指定为字符串常量 (@database_name_var = database name) 或指定为字符串数据类型（ntext 或 text 数据类型除外）的变量
To	关键字，用于指定备份设备
<backup_device>	是一个备份设备，用于存储备份数据，其中 Disk 表示在磁盘上存储备份数据，Tape 表示在磁带设备上存储备份数据。physical_backup_device_name 表示磁盘或磁带上的物理路径，通常用于指定一个备份文件
n	一个占位符，表示可以在逗号分隔的列表中指定多个文件和文件组。数目没有限制
[next-mirror-to]	一个占位符，表示一个 BACKUP 语句除了包含一个 TO 子句外，最多还可包含 3 个 MIRROR TO 子句
DIFFERENTIAL	只能与 BACKUP DATABASE 一起使用，指定数据库备份或文件备份应该只包含上次完整备份后更改的数据库或文件部分。差异备份一般会比完整备份占用更少的空间。对于上一次完整备份后执行的所有单个日志备份，使用该选项可以不必再进行备份

【例 6.25】 把 db_2012 数据库备份到名称是“backup.bak”的备份文件中。在查询分析器中运行的结果如图 6.22 所示。（**实例位置：光盘\TM\sl\6\22**）

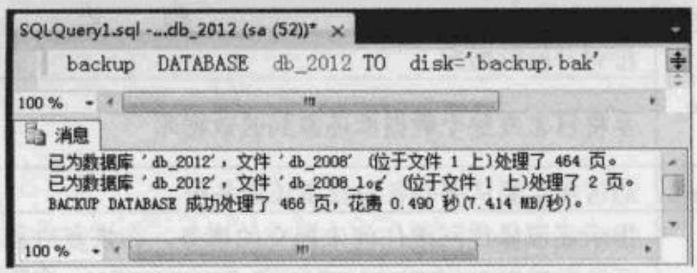


图 6.22　备份 db_2012 数据库

SQL 语句如下：

```
backup DATABASE db_2012 TO disk='backup.bak'
```

6.6.8 RESTORE

如果数据库中的数据发生丢失或被破坏，操作员应该及时还原数据库，尽可能地减小损失。RESTORE 命令用来将数据库或其事务处理日志备份文件由存储介质还原到 SQL Server 系统中。

通过 RESTORE 命令还原数据库，可以执行下列还原方案：

- ☑ 基于完整数据库备份还原整个数据库（完整还原）。
- ☑ 还原数据库的一部分（部分还原）。
- ☑ 将特定文件或文件组还原到数据库（文件还原）。
- ☑ 将特定页面还原到数据库（页面还原）。
- ☑ 将事务日志还原到数据库（事务日志还原）。

☑ 将数据库恢复到数据库快照捕获的时间点。

完整数据库还原的语法如下：

```
RESTORE DATABASE { database_name | @database_name_var }
 [ FROM <backup_device> [ ,...n ] ]
 [ WITH
   {
        [ RECOVERY | NORECOVERY | STANDBY =
        {standby_file_name | @standby_file_name_var }
       ]
    | ,  <general_WITH_options> [ ,...n ]
      | , <replication_WITH_option>
   | , <change_data_capture_WITH_option>
      | , <service_broker_WITH options>
      | , <point_in_time_WITH_options—RESTORE_DATABASE>
      } [ ,...n ]
 ]
[;]
```

RESTORE 命令的参数及说明如表 6.11 所示。

表 6.11 RESTORE 命令的参数及说明

参　数	描　述
Restore Database	指定目标数据库
{ database_name | @database_name_var}	是将日志或整个数据库还原到的数据库
FROM { <backup_device> [,...n]	通常指定要从哪些备份设备还原备份
RECOVERY	指示还原操作回滚任何未提交的事务。在恢复进程后即可随时使用数据库。如果既没有指定 NORECOVERY 和 RECOVERY，也没有指定 STANDBY，则默认为 RECOVERY
NORECOVER	指示还原操作不回滚任何未提交的事务。如果稍后必须应用另一个事务日志，则应指定 NORECOVERY 或 STANDBY 选项。如果既没有指定 NORECOVERY 和 RECOVERY，也没有指定 STANDBY，则默认为 RECOVERY
STANDBY = standby_file_name	指定一个允许撤销恢复效果的备用文件。STANDBY 选项可以用于脱机还原（包括部分还原），但不能用于联机还原。尝试为联机还原操作指定 STANDBY 选项将会导致还原操作失败。如果必须升级数据库，也不允许使用 STANDBY 选项
<general_WITH_options> [,...n]	RESTORE DATABASE 和 RESTORE LOG 语句均支持常规 WITH 选项。一个或多个辅助语句也支持其中某些选项
replication_WITH_option>	此选项只适用于在创建备份时对数据库进行了复制的情况

【例 6.26】 还原例 6.25 中备份了 db_2012 数据库的备份文件 backup.bak。在查询分析器中运行的结果如图 6.23 所示。（**实例位置：光盘\TM\sl\6\23**）

SQL 语句如下：

```
RESTORE DATABASE   db_2012   from disk='backup.bak'   WITH   REPLACE
```

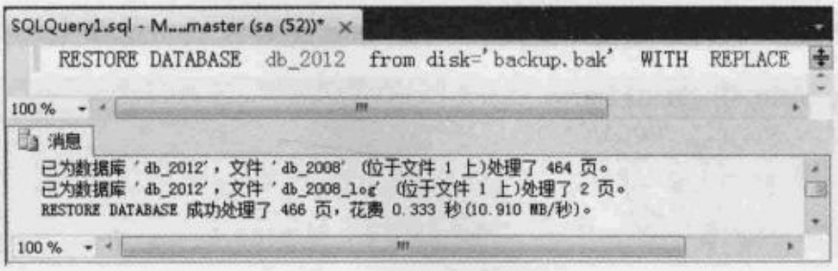


图 6.23　还原备份文件 backup.bak

6.6.9　SELECT

SELECT 语句除了有强大的查询功能外，还可用于给变量赋值。

语法如下：

```
SELECT { @local_variable { = | += | -= | *= | /= | %= | &= | ^= | |= } expression } [ ,...n ] [ ; ]
```

参数说明如下。

- ☑ @local_variable：要为其赋值的声明变量。
- ☑ =：将右边的值赋给左边的变量。
- ☑ { = | += | -= | *= | /= | %= | &= | ^= | |= }：复合赋值运算符。
- ☑ +=：相加并赋值。
- ☑ -=：相减并赋值。
- ☑ *=：相乘并赋值。
- ☑ /=：相除并赋值。
- ☑ %=：取模并赋值。
- ☑ &=：“位与”并赋值。
- ☑ ^=：“位异或”并赋值。
- ☑ |=：“位或”并赋值。
- ☑ expression：任何有效的表达式。此参数包含一个标量子查询。

说明

SELECT @local_variable 通常用于将单个值返回到变量中。但是，如果 expression 是列的名称，则可返回多个值。如果 SELECT 语句返回多个值，则将返回的最后一个值赋给变量。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。

【例 6.27】 给一个变量赋值，SQL 语句如下：

```
declare  @x int
select @x=1
print @x
```

一个 SELECT 语句可以初始化多个局部变量。

【例 6.28】 一次给多个变量赋值，SQL 语句如下：

```
declare   @x int,@y char(20),@z datetime
select @x=1,@y='LOVING',@z='2001/01/01'
print @x
print @y
print @z
```

【例 6.29】 对 tb_Grade 表中的 Subjet 进行查询，并赋值给变量 courses。在查询分析器中运行的结果如图 6.24 所示。（**实例位置：光盘\TM\sl\6\24**）

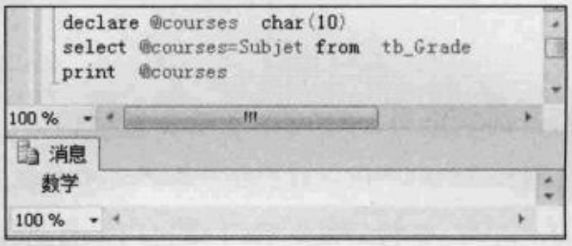


图 6.24 查询的课程内容赋给变量

SQL 语句如下：

```
use   db_2012
declare @courses   char(10)
select @courses= Subjet
from   tb_Grade
print   @courses
```

6.6.10 SET

SET 命令有两种用法，具体如下。

1. 用于给局部变量赋值

在用 DECLARE 命令声明之后，所有的变量都被赋予初值 NULL。这时需要用 SET 命令来给变量赋值。

语法如下：

```
SET{{ @ local_variable=expression}
      |{ @ cursor_variable={@ cursor_variable|cursor_name
            |{CURSOR[FORWARD_ONLY|SCROLL]
               [STATIC|KEYSET|DYNAMIC|FAST_FORWARD]
               [READ_ONLY|SCROLL_LOCKS|OPTIMISTIC]
               [TYPE_VARNING]
            FOR select_statement
               [FOR {READ ONLY|UPDATE[OF column_name[,…n]]}
               ]
            }
        }}
    }
```

【例 6.30】 定义一个变量，并为该变量赋值，SQL 语句如下：

```
declare  @x  int
set  @x=1
print @x
```

SET 命令与 SELECT 命令赋值不同的是，SET 命令一次只能给一个变量赋值。不过由于 SET 命令功能更强且更严密，因此，推荐使用 SET 命令来给变量赋值。

2. 用于执行 SQL 命令时 SQL Server 的处理选项设定

如果要对计算列或索引视图创建和操作索引，则必须将 SET 选项 ARITHABORT、CONCAT_NULL_YIELDS_NULL、QUOTED_IDENTIFIER、ANSI_NULLS、ANSI_PADDING 和 ANSI_WARNINGS 设置为 ON，并将选项 NUMERIC_ROUNDABORT 设置为 OFF。

当从批处理或其他存储过程执行某个存储过程时，执行该存储过程时使用的选项值，就是当前包含该存储过程的数据库中设置的选项值。

6.6.11 SHUTDOWN

SHUTDOWN 命令用于立即停止 SQL Server 的执行。

语法如下：

```
SHUTDOWN[WITH  NOWAIT]
```

参数说明如下。

WITH NOWAIT：当使用 NOWAIT 参数时，SHUTDOWN 命令立即终止所有的用户过程，并在对每一现行的事务发生一个回滚后退出 SQL Server。当没有用 NOWAIT 参数时，SHUTDOWN 命令将按以下步骤执行。

（1）终止任何用户登录 SQL Server。

（2）等待尚未完成的 Transact-SQL 命令或存储过程执行完毕。

（3）在每个数据库中执行 CHECKPOINT 命令。

（4）停止 SQL Server 的执行。

【例 6.31】 使用 SHUTDOWN 命令停止 SQL Server 服务器。在查询分析器中运行的结果如图 6.25 所示。**（实例位置：光盘\TM\sl\6\25）**

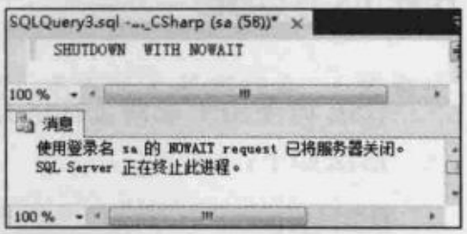


图 6.25　停止 SQL Server 服务器

SQL 语句如下：

```
SHUTDOWN  WITH NOWAIT
```

6.6.12 WRITETEXT

WRITETEXT 命令允许对数据类型为 text、ntext 或 image 的列进行交互式更新。WRITETEXT 语

句不能用在视图中的 text、ntext 和 image 列上。

语法如下：

```
WRITETEXT { table.column text_ptr }
[ WITH LOG ] { data }
```

参数说明如下。

- ☑ table.column：要更新的表和 text、ntext 或 image 列的名称。表名和列名必须符合标识符的规则。指定数据库名和所有者名是可选的。
- ☑ text_ptr：指向 text、ntext 或 image 数据的指针的值。text_ptr 的数据类型必须为 binary(16)。若要创建文本指针，可对 text、ntext 或 image 列用非 NULL 数据执行 INSERT 或 UPDATE 语句。
- ☑ WITH LOG：在 SQL Server 2012 中忽略。日志记录由数据库的实际恢复模型决定。
- ☑ data：要存储的实际 text、ntext 或 image 数据。data 可以是字面值，也可以是变量。对于 text、ntext 和 image 数据，可以用 WRITETEXT 交互插入，文本的最大长度大约是 120KB。

【例 6.32】 将文本指针放到局部变量@val 中，使用 WRITETEXT 命令将新的文本字符串放到@val 所指向的行中，SQL 语句如下：

```
USE pubs
EXEC sp_dboption 'pubs', 'select into/bulkcopy', 'true'
DECLARE @val binary(16)
SELECT @val = TEXTPTR(pr_info)
FROM pub_info p1, publishers p2
WHERE p2.pub_id = p1.pub_id
    AND p2.pub_name = 'New Moon Books'
WRITETEXT pub_info.pr_info @val 'New Moon Books (NMB) has just released another
 top ten publication. With the latest publication this makes NMB the hottest new
publisher of the year!'
EXEC sp_dboption 'pubs', 'select into/bulkcopy', 'false'
```

6.6.13 USE

USE 命令用于在前工作区打开或关闭数据库。

语法如下：

```
USE {数据库}
```

数据库：用户上下文要切换到的数据库的名称。数据库名称必须符合标识符的规则。

要使用 db_2012 数据库时，必须选中该数据库，一种方法是在工具栏上“数据库名称”列表框中选择数据库；另一种方法是使用 USE 命令打开。

【例 6.33】 查询 db_2012 数据库中 Student 的全部信息。SQL 语句如下：

```
USE db_2012
SELECT * FROM Student
```

如果没有在工具栏“数据库名称”列表中选择数据库，也没有使用 USE 命令打开数据库，就会提

示如图 6.26 所示的错误。

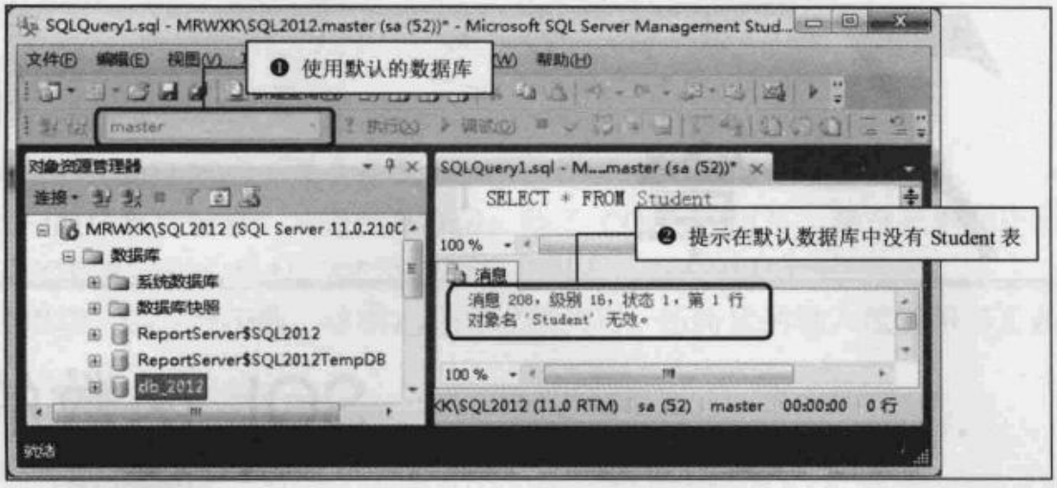


图 6.26　提示对象名无效

6.7　小　　结

本章介绍了 T-SQL 的基础知识，包括 SQL 的数据类型、常量、变量、流程控制语句和常用的命令等。学习本章时，读者需要了解与 T-SQL 相关的概念，区分常量和变量，熟悉这些常用的命令，熟练地掌握流程控制语句的用法、流程控制语句可以提高 SQL 语句的处理能力。

6.8　实践与练习

1．通过使用分组查询中的 GROUPING SETS 运算符，在 books 表中，查询分别按照书名、出版社分组的销售价格。（**答案位置：光盘\TM\sl\6\26**）

2．使用 IF EXISTS 语句检测 Employee 表中性别为女、姓名为“赵小小”的人是否存在。（**答案位置：光盘\TM\sl\6\27**）

第 7 章

SQL 函数的使用

（视频讲解：43 分钟）

在 SQL Server 中提供了许多内置函数，按函数种类可以分为聚合函数、数学函数、字符串函数、日期时间函数、转换函数和元数据函数等 6 种。在进行查询操作时，经常能够用到 SQL 函数，使用 SQL 函数会给查询带来很多的方便，在本章中将会对不同类型的 SQL 函数进行讲解，从而使读者能够快速地掌握好 SQL 函数的使用方法。

通过阅读本章，您可以：

- 熟悉 SQL 函数的几种主要分类
- 掌握常用聚合函数的使用
- 熟悉常用数学函数的使用
- 掌握常用字符串函数的使用
- 掌握常用日期时间函数的使用
- 掌握转换函数的使用
- 熟悉常用元数据函数的使用

7.1 聚合函数

视频讲解：光盘\TM\lx\7\聚合函数.mp4

聚合函数对一组值执行计算，并返回单个值。除了 COUNT 以外，聚合函数都会忽略空值。聚合函数经常与 SELECT 语句的 GROUP BY 子句一起使用。

所有聚合函数均为确定性函数。这表示任何时候使用一组特定的输入值调用聚合函数，所返回的值都是相同的。

7.1.1 聚合函数概述

聚合函数对一组值进行计算并返回单一的值，通常聚合函数会与 SELECT 语句的 GROUP BY 子句一同使用，在与 GROUP BY 子句使用时，聚合函数会为每一个组产生一个单一值，而不会为整个表产生一个单一值。常用的聚合函数及说明如表 7.1 所示。

表 7.1　常用的聚合函数及说明

函数名称	说　明	函数名称	说　明
SUM	返回表达式中所有值的和	MAX	返回表达式的最大值
AVG	计算平均值	COUNT	返回组中项目的数量
MIN	返回表达式的最小值	DISTINCT	返回一个集合，并从指定集合中删除重复的元组

7.1.2 SUM（求和）函数

SUM 函数返回表达式中所有值的和或仅非重复值的和。SUM 只能用于数字列。空值将被忽略。语法如下：

```
SUM ( [ ALL | DISTINCT ] expression )
```

参数说明如下。

☑ ALL：对所有的值应用此聚合函数。ALL 是默认值。

☑ DISTINCT：指定 SUM 返回唯一值的和。

☑ expression：常量、列或函数与算术、位和字符串运算符的任意组合。expression 是精确数字或近似数字数据类型类别（bit 数据类型除外）的表达式。

☑ 返回类型：以最精确的 expression 数据类型返回所有 expression 值的和。

有关 SUM 函数使用的几点说明如下。

☑ 含有索引的字段能够加快聚合函数的运行。

☑ 字段数据类型为 int、smallint、tinyint、decimal、numeric、float、real、money 以及 smallmoney

的字段才可以使用 SUM 函数。

☑ 在使用 SUM 函数时，SQL Server 把结果集中的 smallint 或 tinyint 这些数据类型当作 int 处理。

☑ 在使用 SUM 函数时，SQL Server 将忽略空值（NULL），即计算时不计算这些空值。

【例 7.1】 使用 SUM 函数，求 SC 表中 001（数据结构）课程的总成绩，SQL 语句及运行结果如图 7.1 所示。（**实例位置：光盘\TM\sl\7\1**）

SQL 语句如下：

```
USE db_2012
SELECT SUM(Grade) AS 数据结构总成绩
FROM SC WHERE Cno=001
```

在 SC 表中 001 的成绩如图 7.2 所示。

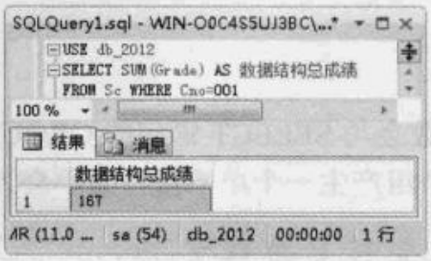


图 7.1 使用 SUM 函数获得数据结构的总成绩

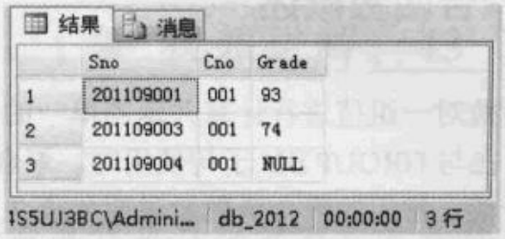


图 7.2 SC 表中 001 的成绩

7.1.3 AVG（平均值）函数

AVG 函数返回组中各值的平均值。将忽略空值。

语法如下：

```
AVG ( [ ALL | DISTINCT ] expression )
```

参数说明如下。

☑ ALL：对所有的值进行聚合函数运算。ALL 是默认值。

☑ DISTINCT：指定 AVG 只在每个值的唯一实例上执行，而不管该值出现了多少次。

☑ expression：是精确数值或近似数值数据类别（bit 数据类型除外）的表达式。不允许使用聚合函数和子查询。

☑ 返回类型：返回类型由 expression 的计算结果类型确定。

有关 AVG 函数使用的几点说明如下。

☑ AVG 函数不一定返回与传递到函数的列完全相同的数据类型。

☑ AVG 函数只能用于数据类型是 int、smallint、tinyint、decimal、float、real、money 和 smallmoney 的字段。

☑ 在使用 AVG 函数时，SQL Server 把结果集中的 smallint 或 tinyint 这些数据类型当作 int 处理。

AVG 函数的返回值类型由表达式的运算结果类型决定，如表 7.2 所示。

表 7.2　AVG 函数返回值类型

表达式结果	返 回 类 型	表达式结果	返 回 类 型
整数分类	int	money 和 smallmoney 分类	money
decimal 分类(p，s)decimal(38，s)	除以 decimal(10，0)	float 和 read 分类	float

【例 7.2】 使用 AVG 函数，求 SC 表中 001（数据结构）课程的平均成绩，SQL 语句及运行结构如图 7.3 所示。（**实例位置：光盘\TM\sl\7\2**）

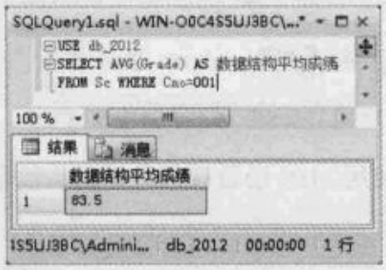


图 7.3　使用 AVG 函数获得数据结构的平均成绩

SQL 语句如下：

```
USE db_2012
SELECT AVG(Grade) AS 数据结构平均成绩
FROM SC WHERE Cno=001
```

7.1.4　MIN（最小值）函数

MIN 函数返回表达式中的最小值。

语法如下：

```
MIN ( [ ALL | DISTINCT ] expression )
```

参数说明如下。

- ☑ ALL：对所有的值进行聚合函数运算。ALL 是默认值。
- ☑ DISTINCT：指定每个唯一值都被考虑。DISTINCT 对于 MIN 无意义，使用它仅仅是为了符合 ISO 标准。
- ☑ expression：常量、列名、函数以及算术运算符、位运算符和字符串运算符的任意组合。MIN 可用于 numeric、char、varchar 或 datetime 列，但不能用于 bit 列。不允许使用聚合函数和子查询。
- ☑ 返回类型：返回与 expression 相同的值。

有关 MIN 函数使用的几点说明如下。

- ☑ MIN 函数不能用于数据类型是 bit 的字段。
- ☑ 在确定列中的最小值时，MIN 函数忽略 NULL 值，但是如果在该列中的所有行都有 NULL 值，将返回 NULL 值。
- ☑ 不允许使用聚合函数和子查询。

【例 7.3】 使用 MIN 函数，查询 Student 表中男同学的最小年龄，SQL 语句及运行结构如图 7.4 所示。（**实例位置：光盘\TM\sl\7\3**）

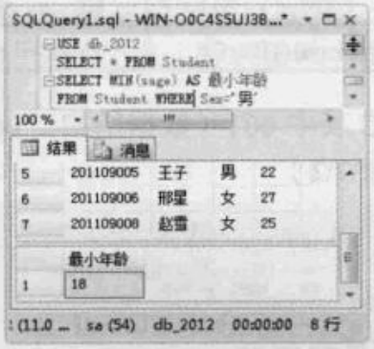


图 7.4 使用 MIN 函数获得数据结构的最小成绩

SQL 语句如下：

```
USE db_2012
SELECT * FROM Student
SELECT MIN(Sage) AS 最小年龄
FROM Student WHERE Sex ='男'
```

7.1.5 MAX（最大值）函数

MAX 函数返回表达式的最大值。

语法如下：

```
MAX ( [ ALL | DISTINCT ] expression )
```

参数说明如下。

- ☑ ALL：对所有的值应用此聚合函数。ALL 是默认值。
- ☑ DISTINCT：指定考虑每个唯一值。DISTINCT 对于 MAX 无意义，使用它仅仅是为了与 ISO 实现兼容。
- ☑ expression：常量、列名、函数以及算术运算符、位运算符和字符串运算符的任意组合。MAX 可用于 numeric、character 和 datetime 列，但不能用于 bit 列。不允许使用聚合函数和子查询。
- ☑ 返回类型：返回与 expression 相同的值。

有关 MAX 函数使用的几点说明如下。

- ☑ MAX 函数将忽略选取对象中的空值。
- ☑ 不能通过 MAX 函数从 bit、text 和 image 数据类型的字段中选取最大值。
- ☑ 在 SQL Server 中，MAX 函数可以用于数据类型为数字、字符、datetime 的列，但是不能用于数据类型为 bit 的列。不能使用聚合函数和子查询。
- ☑ 对于字符列，MAX 查找排序序列的最大值。

【例 7.4】 在本示例中使用了一个子查询，并在子查询中使用了 MAX 函数将查询条件指定为 Student 表中年龄最大的同学信息，SQL 语句及运行结构如图 7.5 所示。（**实例位置：光盘\TM\sl\7\4**）

图 7.5　使用 MAX 函数获取 Student 表中年龄最大的同学信息

SQL 语句如下：

```
USE db_2012
SELECT * FROM Student
SELECT Sname,Sex,Sage FROM Student
WHERE Sage=(SELECT MAX(Sage) FROM Student)
```

首先在 Student 表中选择指定列的数据并显示，然后在 WHERE 条件中使用子查询，并在子查询中使用 MAX 函数选择 Student 中年龄最大的操作员。

如果用户不想获取其他列的信息，可以直接在 SELECT 语句中使用 MAX 函数加上要查询的列即可。

【例 7.5】 直接查询学生中年龄最大的同学，SQL 语句如下。（**实例位置：光盘\TM\sl\7\5**）

```
USE  db_2012
SELECT MAX(Sage) AS 最大年龄 FROM Student
```

7.1.6　COUNT（统计）函数

COUNT 函数返回组中的项数。COUNT 返回 int 数据类型值。

语法如下：

```
COUNT( { [ [ ALL | DISTINCT ] expression ] | * } )
```

参数说明如下。

- ☑ ALL：对所有的值进行聚合函数运算。ALL 是默认值。
- ☑ DISTINCT：指定 COUNT 返回唯一非空值的数量。
- ☑ expression：除 text、image 或 ntext 以外任何类型的表达式。不允许使用聚合函数和子查询。
- ☑ *：指定应该计算所有行以返回表中行的总数。COUNT(*)不需要任何参数，而且不能与 DISTINCT 一起使用。COUNT(*)不需要 expression 参数，因为根据定义，该函数不使用有关任何特定列的信息。COUNT(*)返回指定表中行数而不删除副本。它对各行分别计数。包括包含空值的行。
- ☑ 返回类型：int 类型。

【例 7.6】 使用 SELECT 语句显示商品名称，并使用 COUNT 函数查询所有销售的商品名称，然后使用 AS 语句，将“Sex”重命名为“人数”，最后显示查询结果，SQL 语句及运行结果如图 7.6 所示。（**实例位置：光盘\TM\sl\7\6**）

SQL 语句如下：

```
USE   db_2012
SELECT * FROM Student
SELECT Sex,COUNT (Sex) AS  人数  FROM Student
GROUP BY Sex
```

【例 7.7】 查询 Student 表中的总人数，SQL 语句及运行结果如图 7.7 所示。（**实例位置：光盘\TM\sl\7\7**）

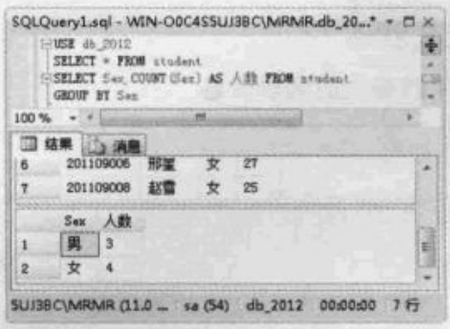


图 7.6　使用 COUNT 函数计算男女同学的人数

图 7.7　使用 COUNT 函数计算学生的总人数

SQL 语句如下：

```
USE   db_2012
SELECT * FROM Student
SELECT COUNT (*) AS  总人数  FROM Student
```

7.1.7　DISTINCT（取不重复记录）函数

DISTINCT 函数，对指定的集求值，删除该集中的重复元组，然后返回结果集。

语法如下：

```
Distinct(Set_Expression)
```

参数 Set_Expression 表示返回集的有效多维表达式。

说明

如果 Distinct 函数在指定的集中找到了重复的元组，则此函数只保留重复元组的第一个实例，同时保留该集原来的顺序。

【例 7.8】 使用 DISTINCT 函数查询 Course 表中不重复的课程信息，SQL 语句及运行结果如图 7.8 所示。（**实例位置：光盘\TM\sl\7\8**）

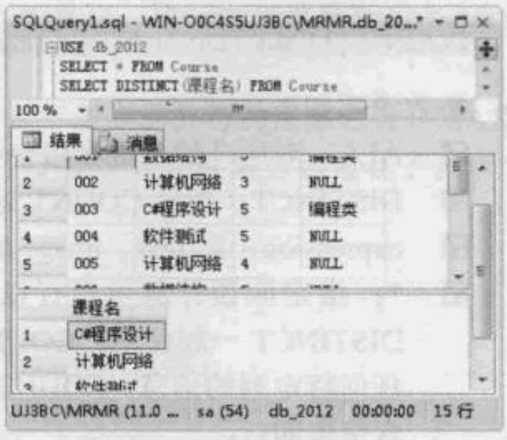


图 7.8　查询 Course 表中不重复的课程信息

SQL 语句如下：

```
SELECT * FROM Course
SELECT DISTINCT(Cname)
FROM Course ORDER BY Cname
```

7.1.8　查询重复记录

查询数据表中的重复记录，可以借助 HAVING 子句实现，该子句用来指定组或聚合的搜索条件。HAVING 子句只能与 SELECT 语句一起使用，而且，它通常在 GROUP BY 子句中使用。

HAVING 子句语法如下：

```
[ HAVING <search condition> ]
```

参数<search condition>用来指定组或聚合应满足的搜索条件。

【例 7.9】 使用 HAVING 子句为组指定条件，当同种课程的记录大于等于一条时，显示此课程的名称及重复数量，SQL 语句及运行结果如图 7.9 所示。（**实例位置：光盘\TM\sl\7\9**）

图 7.9　查询重复的课程及重复数量

SQL 语句如下：

```
USE db_2012
SELECT * FROM Course
SELECT Cname,COUNT(Cname) AS 重复数量 FROM Course
GROUP BY Cname
HAVING COUNT(Cname)>1
ORDER BY Cname
```

7.2　数 学 函 数

视频讲解：光盘\TM\lx\7\数学函数.mp4

数学函数能够对数字表达式进行数学运算，并能够将结果返回给用户。默认情况下，传递给数学函数的数字将被解释为双精度浮点数。

7.2.1 数学函数概述

数学函数可以对数据类型为整型（integer）、实型（real）、浮点型（float）、货币型（money）和 smallmoney 的列进行操作。它的返回值是 6 位小数，如果使用出错，则返回 NULL 值并显示提示信息，通常该函数可以用在 SQL 语句的表达式中。常用的数学函数及说明如表 7.3 所示。

表 7.3 常用的数学函数及说明

函数名称		说明	
ABS	返回指定数字表达式的绝对值	ROUND	将数字表达式四舍五入为指定的长度或精度
COS	返回指定的表达式中指定弧度的三角余弦值	SIGN	返回指定表达式的零（0）、正号（+1）或负号（-1）
COT	返回指定的表达式中指定弧度的三角余切值	SIN	返回指定的表达式中指定弧度的三角正弦值
PI	返回值为圆周率	SQUARE	返回指定表达式的平方
POWER	将指定的表达式乘指定次方	SQRT	返回指定表达式的平方根
RAND	返回 0～1 之间的随机 float 数	TAN	返回指定的表达式中指定弧度的三角正切值

注意

算术函数（例如 ABS、CEILING、DEGREES、FLOOR、POWER、RADIANS 和 SIGN）返回与输入值具有相同数据类型的值。三角函数和其他函数（包括 EXP、LOG、LOG10、SQUARE 和 SQRT）将输入值转换为 float 并返回 float 值。

7.2.2 ABS（绝对值）函数

ABS 函数返回数值表达式的绝对值。

语法如下：

```
ABS(numeric_expression)
```

参数说明如下。

☑ numeric_expression：是有符号或无符号的数值表达式。

☑ 结果类型：提交给函数的数值表达式的数据类型。

说明

如果该参数为空，则 ABS 返回的结果为空。

【例 7.10】 使用 ABS 函数求指定表达式的绝对值，SQL 语句及运行结果如图 7.10 所示。（**实例位置：光盘\TM\sl\7\10**）

图 7.10 指定表达式的绝对值

SQL 语句如下：

```
SELECT ABS(1.0) AS "1.0 的绝对值",
ABS(0.0) AS "0.0 的绝对值",
ABS(-1.0) AS "-1.0 的绝对值"
```

7.2.3　PI（圆周率）函数

PI 函数返回 PI 的常量值。

语法如下：

```
PI( )
```

返回类型：float 型。

【例 7.11】 使用 PI 函数返回指定 PI 的值，SQL 语句及运行结果如图 7.11 所示。（**实例位置：光盘\TM\sl\7\11**）

图 7.11　返回 PI 的值

SQL 语句如下：

```
SELECT PI() AS 圆周率
```

7.2.4　POWER（乘方）函数

POWER 函数返回对数值表达式进行幂运算的结果。Power 参数的计算结果必须为整数。

语法如下：

```
POWER(numeric_expression,power)
```

参数说明如下。

☑ numeric_expression：有效的数值表达式。

☑ power：有效的数值表达式。

【例 7.12】 使用 POWER 函数分别求 2、3、4 的乘方的结果，SQL 语句及运行结果如图 7.12 所示。（**实例位置：光盘\TM\sl\7\12**）

SQL 语句如下：

```
SELECT POWER(2,2)AS "2 的平方结果",
POWER(3,3)AS "3 的 3 次幂结果",
POWER(4,4) AS "4 的 4 次幂结果"
```

图 7.12　计算指定数的乘方

7.2.5　RAND（随机浮点数）函数

RAND 函数返回从 0 到 1 之间的随机 float 值。

语法如下：

```
RAND( [ seed ] )
```

参数说明如下。

☑ seed：提供种子值的整数表达式（tinyint、smallint 或 int）。如果未指定 seed，则 Microsoft SQL Server 数据库引擎随机分配种子值。对于指定的种子值，返回的结果始终相同。

☑ 返回类型：float 类型。

注意

使用同一个种子值重复调用 RAND()会返回相同的结果。

【例 7.13】 使用同一种子值调用 RAND 函数，返回相同的数字序列，SQL 语句及运行结果如图 7.13 所示。（**实例位置：光盘\TM\sl\7\13**）

SQL 语句如下：

```
SELECT RAND(100), RAND(), RAND()
```

【例 7.14】 使用 RAND 函数生成 3 个不同的随机数，SQL 语句及运行结果如图 7.14 所示。（**实例位置：光盘\TM\sl\7\14**）

图 7.13　使用同一种子值调用 RAND 函数

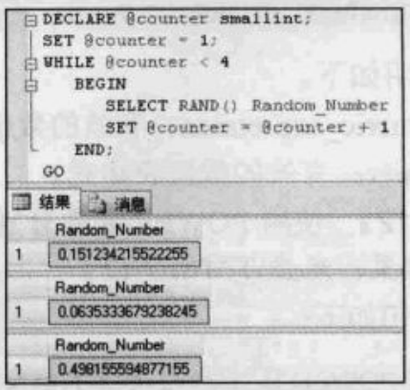


图 7.14　生成 3 个不同的随机数

SQL 语句如下：

```
DECLARE @counter smallint;
SET @counter = 1;
WHILE @counter < 4
   BEGIN
      SELECT RAND() Random_Number
      SET @counter = @counter + 1
   END;
GO
```

7.2.6　ROUND（四舍五入）函数

ROUND 函数返回一个数值，舍入到指定的长度或精度。

语法如下：

```
ROUND( numeric_expression , length [ ,function ] )
```

参数说明如下。

- ☑ numeric_expression：精确数值或近似数值数据类别（bit 数据类型除外）的表达式。
- ☑ length：numeric_expression 的舍入精度。length 必须是 tinyint、smallint 或 int 类型的表达式。如果 length 为正数，则将 numeric_expression 舍入到 length 指定的小数位数。如果 length 为负数，则将 numeric_expression 小数点左边部分舍入到 length 指定的长度。
- ☑ function：要执行的操作的类型。function 必须为 tinyint、smallint 或 int。如果省略 function 或其值为 0（默认值），则将舍入 numeric_expression。如果指定了 0 以外的值，则将截断 numeric_expression。
- ☑ 返回类型：返回与 numeric_expression 相同的类型。

【例 7.15】 使用 ROUND 函数计算指定表达式的值，SQL 语句及运行结果如图 7.15 所示。（**实例位置：光盘\TM\sl\7\15**）

图 7.15　使用 ROUND 函数计算表达式

SQL 语句如下：

```
SELECT ROUND(123.9994, 3), ROUND(123.9995, 3)
```

7.2.7　SQUARE（平方）函数和 SQRT（平方根）函数

1. SQUARE（平方）函数

SQUARE 函数返回数值表达式的平方。

语法如下：

```
SQUARE(numeric_expression)
```

参数 numeric_expression 表示任意数值数据类型的数值表达式。

【例 7.16】 使用 SQUARE 函数计算指定表达式的值，SQL 语句及运行结果如图 7.16 所示。（实例位置：光盘\TM\sl\7\16）

SQL 语句如下：

```
SELECT SQUARE (4) AS "4 的平方"
```

2. SQRT（平方根）函数

SQRT 函数返回数值表达式的平方根。

语法如下：

```
SQRT(numeric_expression)
```

参数 numeric_expression 表示任意数值数据类型的数值表达式。

【例 7.17】 使用 SQRT 函数计算指定表达式的值，SQL 语句及运行结果如图 7.17 所示。（实例位置：光盘\TM\sl\7\17）

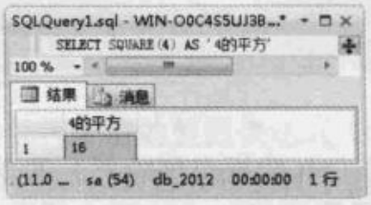


图 7.16 使用 SQUARE 函数计算表达式

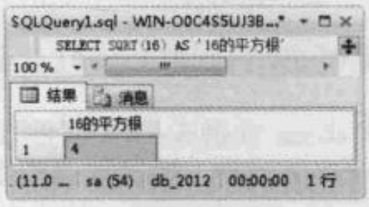


图 7.17 使用 SQRT 函数计算表达式

SQL 语句如下：

```
SELECT SQRT(16) AS '16 的平方根'
```

【例 7.18】 使用 SQRT 函数返回 1.00～10.00 之间的数字平方根，SQL 语句如下。（实例位置：光盘\ TM\sl\7\18）

```
DECLARE @mysqrt float
SET @mysqrt = 1.00
WHILE @mysqrt < 10.00
BEGIN
  SELECT SQRT(@mysqrt)
  SELECT @mysqrt = @mysqrt + 1
END
```

程序运行结果如表 7.4 所示。

表 7.4 结果集

数 字	平 方 根	数 字	平 方 根
1.00	1.0	6.00	2.44948974278318
2.00	1.4142135623731	7.00	2.64575131106459
3.00	1.73205080756888	8.00	2.82842712474619
4.00	2.0	9.00	3.0
5.00	2.23606797749979		

7.2.8　三角函数

三角函数包括 COS、COT、SIN 以及 TAN 函数，分别表示三角余弦值、三角余切值、三角正弦值和三角正切值，下面分别对这几种三角函数进行详细讲解。

1. COS 函数

返回指定表达式中以弧度表示的指定角的三角余弦。

语法如下：

```
COS( float_expression )
```

参数说明如下。

☑　float_expression：float 类型的表达式。

☑　返回类型：float 类型。

【例 7.19】　使用 COS 函数返回指定表达式的余弦值，SQL 语句及运行结果如图 7.18 所示。（实例位置：光盘\TM\sl\7\19）

SQL 语句如下：

```
DECLARE @angle float
SET @angle =10
SELECT CONVERT(varchar,COS(@angle)) AS COS_
GO
```

2. COT 函数

COT 函数返回指定的 float 表达式中所指定角度（以弧度为单位）的三角余切值。

语法如下：

```
COT( float_expression )
```

参数说明如下。

☑　float_expression：属于 float 类型或能够隐式转换为 float 类型的表达式。

☑　返回类型：float 类型。

【例 7.20】　使用 COT 函数返回指定表达式的余切值，SQL 语句及运行结果如图 7.19 所示。（实例位置：光盘\TM\sl\7\20）

图 7.18　返回指定表达式的余弦值

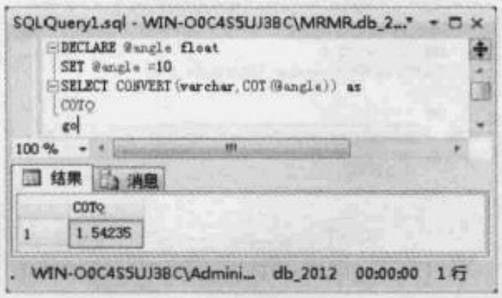


图 7.19　返回指定表达式的余切值

SQL 语句如下：

```
DECLARE @angle float
SET @angle =10
SELECT CONVERT(varchar,COT(@angle)) AS COT_
GO
```

3. SIN 函数

SIN 函数以近似数字（float）表达式返回指定角度（以弧度为单位）的三角正弦值。

语法如下：

```
SIN( float_expression )
```

参数说明如下。

☑ float_expression：属于 float 类型或能够隐式转换为 float 类型的表达式。

☑ 返回类型：float 类型。

【例 7.21】 使用 SIN 函数返回指定表达式的正弦值，SQL 语句及运行结果如图 7.20 所示。（实例位置：光盘\TM\sl\7\21）

SQL 语句如下：

```
DECLARE @angle float
SET @angle =12.5
SELECT CONVERT(varchar,SIN(@angle)) AS SIN_
GO
```

4. TAN 函数

TAN 函数返回输入表达式的正切值。

语法如下：

```
TAN( float_expression )
```

参数说明如下。

☑ float_expression：是 float 类型或可隐式转换为 float 类型的表达式，解释为弧度数。

☑ 返回类型：float 类型。

【例 7.22】 使用 TAN 函数返回指定表达式的正切值，SQL 语句及运行结果如图 7.21 所示。（实例位置：光盘\TM\sl\7\22）

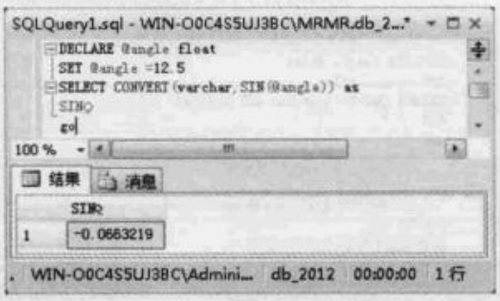


图 7.20 返回指定表达式的正弦值

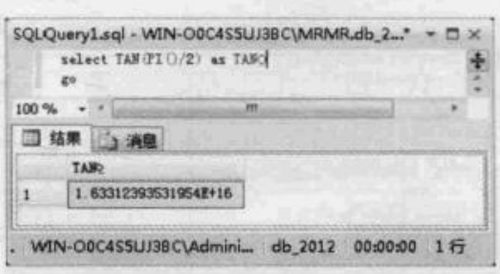


图 7.21 返回指定表达式的正切值

SQL 语句如下：

```
SELECT TAN(PI()/2) AS TAN
```

7.3　字符串函数

视频讲解：光盘\TM\lx\7\字符串函数.mp4

字符串函数对 N 进制数据、字符串和表达式执行不同的运算，如返回字符串的起始位置、返回字符串的个数等。本节介绍 SQL Server 中常用的字符串函数。

7.3.1　字符串函数概述

字符串函数作用于 char、varchar、binary 和 varbinary 数据类型以及可以隐式转换为 char 或 varchar 的数据类型。通常字符串函数可以用在 SQL 语句的表达式中。常用的字符串函数及说明如表 7.5 所示。

表 7.5　常用的字符串函数及说明

函数名称	说　明	函数名称	说　明
ASCII	返回字符表达式最左端字符的 ASCII 代码值	REVERSE	返回字符表达式的反转
CHARINDEX	返回字符串中指定表达式的起始位置	RIGHT	从右边开始，取得字符串右边指定个数的字符
LEFT	从左边开始，取得字符串左边指定个数的字符	STR	返回由数字数据转换来的字符数据
LEN	返回指定字符串的字符（而不是字节）个数	SUBSTRING	返回指定个数的字符
REPLACE	将指定的字符串替换为另一指定的字符串		

7.3.2　ASCII（获取 ASCII 码）函数

ASCII 函数返回字符表达式中最左侧的字符的 ASCII 代码值。
语法如下：

```
ASCII( character_expression )
```

参数说明如下。

☑ character_expression：char 或 varchar 类型的表达式。
☑ 返回类型：int 类型。

说明

ASCII 码共有 127 个，其中 Microsoft Windows 不支持 1～7、11～12 和 14～31 之间的字符。值 8、9、10 和 13 分别转换为退格、制表、换行和回车字符。它们并没有特定的图形显示，但会依不同的应用程序而对文本显示有不同的影响。

ASCII 码值对照表如表 7.6 所示。

表 7.6　ASCII 码值对照表

ASCII 码	按　键	ASCII 码	按　键	ASCII 码	按　键	ASCII 码	按　键
0	?/FONT>	32	[space]	64	@	96	
1	不支持	33	!	65	A	97	A
2	不支持	34	"	66	B	98	B
3	不支持	35	#	67	C	99	C
4	不支持	36	$	68	D	100	D
5	不支持	37	%	69	E	101	E
6	不支持	38	&	70	F	102	F
7	不支持	39	'	71	G	103	G
8	* *	40	(	72	H	104	H
9	* *	41	)	73	I	105	I
10	* *	42	*	74	J	106	j
11	不支持	43	+	75	K	107	k
12	不支持	44	,	76	L	108	l
13	* *	45	-	77	M	109	m
14	不支持	46	.	78	N	110	n
15	不支持	47	/	79	O	111	o
16	不支持	48	0	80	P	112	p
17	不支持	49	1	81	Q	113	q
18	不支持	50	2	82	R	114	r

【例 7.23】 使用 ASCII 函数返回 NXT 的 ASCII 代码值，SQL 语句及运行结果如图 7.22 所示。（**实例位置：光盘\TM\sl\7\23**）

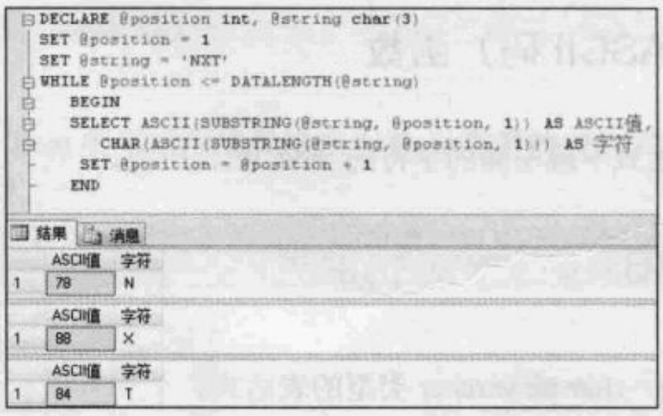


图 7.22　返回指定表达式的 ASCII 值

SQL 语句如下：

```
DECLARE @position int, @string char(3)
SET @position = 1
SET @string = 'NXT'
```

```
WHILE @position <= DATALENGTH(@string)
   BEGIN
   SELECT ASCII(SUBSTRING(@string, @position, 1)) AS ASCII 值,
      CHAR(ASCII(SUBSTRING(@string, @position, 1))) AS  字符
    SET @position = @position + 1
   END
```

7.3.3　CHARINDEX（返回字符串的起始位置）函数

CHARINDEX 函数返回字符串中指定表达式的起始位置（如果找到）。搜索的起始位置为 start_location。

语法如下：

```
CHARINDEX( expression1 ,expression2 [ , start_location ] )
```

参数说明如下。

- ☑ expression1：包含要查找的序列的字符表达式。expression1 最大长度限制为 8000 个字符。
- ☑ expression2：要搜索的字符表达式。
- ☑ start_location：表示搜索起始位置的整数或 bigint 表达式。如果未指定 start_location，或者 start_location 为负数或 0，则将从 expression2 的开头开始搜索。
- ☑ 返回类型：如果 expression2 的数据类型为 varchar(max)、nvarchar(max)或 varbinary(max)，则为 bigint，否则为 int。

【例 7.24】 使用 CHARINDEX 函数返回指定字符串的起始位置，SQL 语句及运行结果如图 7.23 所示。（**实例位置：光盘\TM\sl\7\24**）

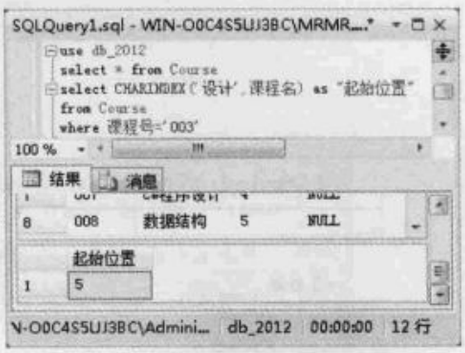


图 7.23　返回指定字符串的起始位置

SQL 语句如下：

```
USE db_2012
SELECT * FROM Course
SELECT CHARINDEX('设计',Cname) AS "起始位置" FROM Course
WHERE Cno = '003'
```

7.3.4　LEFT（取左边指定个数的字符）函数

LEFT 函数返回字符串中从左边开始指定个数的字符。

语法如下：

```
LEFT( character_expression , integer_expression )
```

参数说明如下。

- ☑ character_expression：字符或二进制数据表达式。character_expression 可以是常量、变量或列。character_expression 可以是任何能够隐式转换为 varchar 或 nvarchar 的数据类型，但 text 或 ntext

除外。否则，请使用 CAST 函数对 character_expression 进行显式转换。

☑ integer_expression：正整数，指定 character_expression 将返回的字符数。如果 integer_expression 为负，则将返回错误。如果 integer_expression 的数据类型为 bigint 且包含一个较大值，character_expression 必须是大型数据类型，如 varchar(max)。

返回类型如下：

☑ 当 character_expression 为非 Unicode 字符数据类型时，返回 varchar。

☑ 当 character_expression 为 Unicode 字符数据类型时，返回 nvarchar。

【例 7.25】 使用 LEFT 函数返回指定字符串的最左边 4 个字符，SQL 语句及运行结果如图 7.24 所示。（**实例位置：光盘\TM\sl\7\25**）

SQL 语句如下：

```
SELECT LEFT('明日科技有限公司',4)
```

【例 7.26】 使用 LEFT 函数查询 Student 表中的姓氏（姓氏是姓名的第一位）并计算出每个姓氏的数量，SQL 语句及运行结果如图 7.25 所示。（**实例位置：光盘\TM\sl\7\26**）

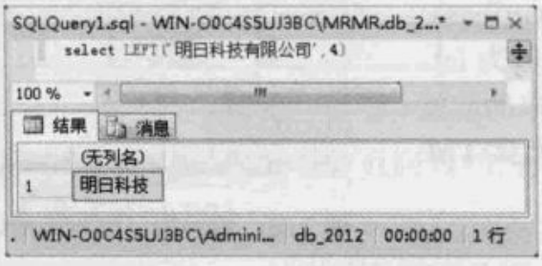


图 7.24 返回指定字符串中的字符

图 7.25 查询 Student 表中的姓氏

SQL 语句如下：

```
USE db_2012
SELECT Sno ,Sname FROM Student
SELECT LEFT(Sname,1) AS '姓氏', COUNT(LEFT(Sname,1)) AS '数量'
FROM Student Group BY LEFT(Sname,1)
```

7.3.5 RIGHT（取右边指定个数的字符）函数

RIGHT 函数返回字符表达式中从起始位置（从右端开始）到指定字符位置（从右端开始计数）的部分。

语法如下：

```
RIGHT(character_expression,integer_expression)
```

参数说明如下。

☑ character_expression：是从中提取字符的字符表达式。

☑ integer_expression：是指示返回字符数的整数表达式。

【例 7.27】 使用 RIGHT 函数查询 Student 表中编号的后 3 位，SQL 语句及运行结果如图 7.26 所示。（**实例位置：光盘\TM\sl\7\27**）

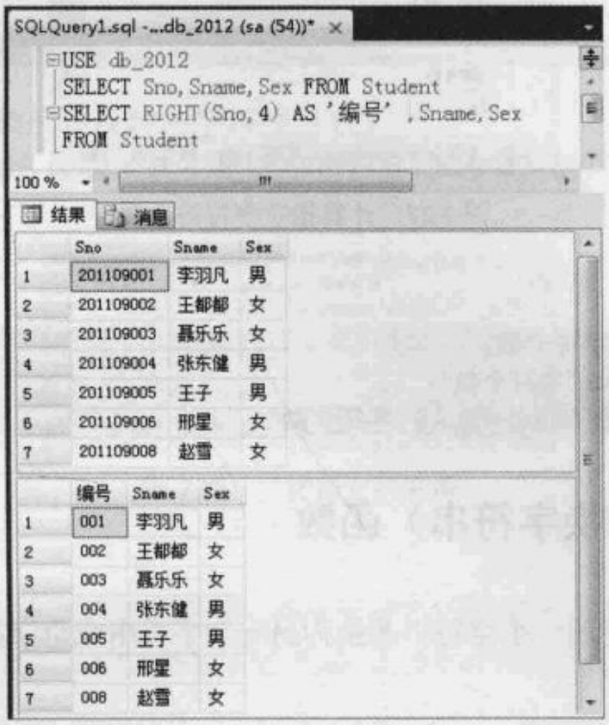


图 7.26　查询 Student 表中的编号后 3 位

SQL 语句如下：

```
USE db_2012
SELECT Sno,Sname,Sex FROM Student
SELECT RIGHT(Sno,4) AS '编号',Sname,Sex
FROM Student
```

7.3.6　LEN（返回字符个数）函数

LEN 函数返回字符表达式中的字符数。如果字符串中包含前导空格和尾随空格，则函数会将它们包含在计数内。LEN 对相同的单字节和双字节字符串返回相同的值。

语法如下：

```
LEN(character_expression)
```

参数 character_expression 表示要处理的表达式。

【例 7.28】 使用 LEN 函数计算指定字符的个数，SQL 语句及运行结果如图 7.27 所示。（**实例位置：光盘\TM\sl\7\28**）

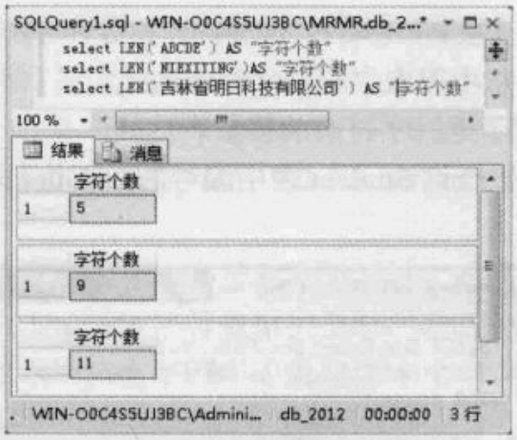


图 7.27　计算指定字符的个数

SQL 语句如下：

```
SELECT LEN('ABCDE') AS "字符个数"
SELECT LEN('NIEXITING') AS "字符个数"
SELECT LEN('吉林省明日科技有限公司') AS "字符个数"
```

7.3.7　REPLACE（替换字符串）函数

REPLACE 函数将表达式中的一个字符串替换为另一个字符串或空字符串后，返回一个字符表达式。语法如下：

```
REPLACE(character_expression,searchstring,replacementstring)
```

参数说明如下。

☑ character_expression：是函数要搜索的有效字符表达式。

☑ searchstring：是函数尝试定位的有效字符表达式。

☑ replacementstring：是用作替换表达式的有效字符表达式。

【例 7.29】 使用 REPLACE 函数替换指定的字符，SQL 语句及运行结果如图 7.28 所示。（**实例位置：光盘\TM\sl\7\29**）

图 7.28　替换指定的字符

SQL 语句如下：

```
SELECT REPLACE('MingRMRM','RMRM','Ri')
AS '替换结果'
```

7.3.8　REVERSE（返回字符表达式的反转）函数

REVERSE 函数按相反顺序返回字符表达式。

语法如下：

```
REVERSE(character_expression)
```

参数 character_expression 表示要反转的字符表达式。

【例 7.30】 使用 REVERSE 函数反转指定的字符，SQL 语句及运行结果如图 7.29 所示。（**实例位置：光盘\TM\sl\7\30**）

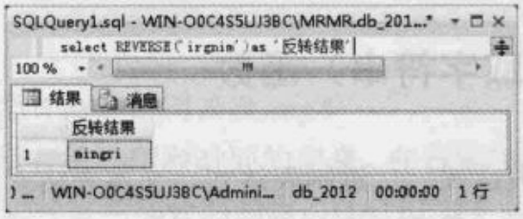


图 7.29　反转指定的字符

SQL 语句如下：

```
SELECT REVERSE ('irgnim')
AS '反转结果'
```

7.3.9　STR 函数

STR 函数返回由数字数据转换来的字符数据。

语法如下：

```
STR( float_expression [ , length [ , decimal ] ] )
```

参数说明如下。

☑ float_expression：带小数点的近似数字（float）数据类型的表达式。

☑ length：总长度。它包括小数点、符号、数字以及空格。默认值为 10。

☑ decimal：小数点后的位数。decimal 必须小于或等于 16。如果 decimal 大于 16，则会截断结果，使其保持为小数点后具有 16 位。

【例 7.31】 使用 STR 函数返回以下字符数据，SQL 语句及运行结果如图 7.30 所示。（**实例位置：光盘\TM\sl\7\31**）

SQL 语句如下：

```
SELECT STR(123.45) AS 'STR',
STR(123.45,5,1) AS 'STR',
STR(123.45,8,1) AS 'STR',
STR(123.45,2,2) AS 'STR'
```

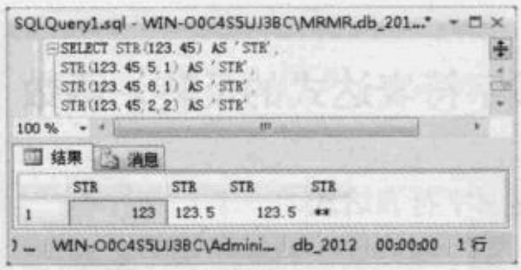


图 7.30　使用 STR 函数转换字符串

注意

当表达式超出指定长度时，字符串为指定长度返回 **。

7.3.10　SUBSTRING（取字符串）函数

SUBSTRING 函数返回字符表达式、二进制表达式、文本表达式或图像表达式的一部分。语法如下：

```
SUBSTRING( value_expression,start_expression, length_expression )
```

参数说明如下。

- ☑ value_expression：是 character、binary、text、ntext 或 image 表达式。
- ☑ start_expression：指定返回字符的起始位置的整数或 bigint 表达式。如果 start_expression 小于 0，会生成错误并终止语句。如果 start_expression 大于值表达式中的字符数，将返回一个零长度的表达式。
- ☑ length_expression：是正整数或指定要返回的 value_expression 的字符数的 bigint 表达式。如果 length_expression 是负数，会生成错误并终止语句。如果 start_expression 与 length_expression 的总和大于 value_expression 中的字符数，则返回整个值表达式。
- ☑ 返回类型：如果 expression 是受支持的字符数据类型，则返回字符数据。如果 expression 是支持的 binary 数据类型中的一种数据类型，则返回二进制数据。返回的字符串类型与指定表达式的类型相同，表 7.7 中显示的除外。

表 7.7　返回的字符串类型与指定表达式的类型不相同

指定的表达式	返 回 类 型
char/varchar/text	varchar
nchar/nvarchar/ntext	nvarchar
binary/varbinary/image	varbinary

【例 7.32】 使用 SUBSTRING 函数，在 Sno 字段中从第 5 位开始取字符串，共 5 位，SQL 语句及运行结果如图 7.31 所示。（**实例位置：光盘\TM\sl\7\32**）

SQL 语句如下：

```
SELECT Sno, SUBSTRING(Sno,5,5) AS '编号'
FROM Student
```

```
SELECT Sno, SUBSTRING(Sno,5,5) AS '编号'
FROM Student
```

结果　消息

	Sno	编号
1	201109001	09001
2	201109002	09002
3	201109003	09003
4	201109004	09004
5	201109005	09005
6	201109006	09006
7	201109007	09007

图 7.31　使用 SUBSTRING 函数取字符串

7.4　日期和时间函数

视频讲解：光盘\TM\lx\7\日期和时间函数.mp4

日期和时间函数主要用来显示有关日期和时间的信息。在日期和时间函数中，DAY 函数、MONTH 函数、YEAR 函数是用来获取时间和日期部分的函数。DATEDIFF 函数是用来获取日期和时间差的函数。DATEADD 函数是用来修改日期和时间值的函数。本节详细介绍这些函数。

7.4.1　日期和时间函数概述

日期和时间函数主要用来操作 datetime、smalldatetime 类型的数据，日期和时间函数执行算术运算与其他函数一样，也可以在 SQL 语句的 SELECT、WHERE 子句以及表达式中使用。常用的日期时间函数及说明如表 7.8 所示。

表 7.8　常用的日期时间函数及说明

函 数 名 称	说　　明
DATEADD	在向指定日期加上一段时间的基础上，返回新的 datetime 值
DATEDIFF	返回跨两个指定日期的日期和时间边界数
GETDATE	返回当前系统日期和时间
DAY	返回指定日期中的天的整数
MONTH	返回指定日期中的月份的整数
YEAR	返回指定日期中的年份的整数

7.4.2　GETDATE（返回当前系统日期和时间）函数

GETDATE 函数返回系统的当前日期。GETDATE 函数不使用参数。

注意

GETDATE 函数的返回结果的长度为 29 个字符。

语法如下：

```
GETDATE()
```

【例 7.33】 使用 GETDATE 函数，返回当前系统日期和时间，SQL 语句及运行结果如图 7.32 所示。（**实例位置：光盘\TM\sl\7\33**）

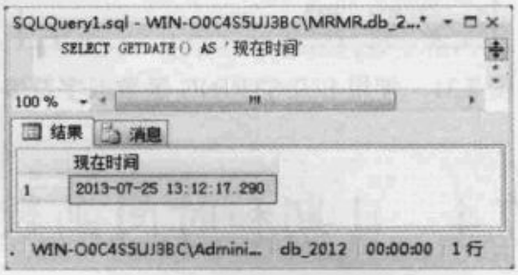


图 7.32 获取当前系统日期和时间

SQL 语句如下：

```
SELECT GETDATE() AS '现在时间'
```

7.4.3 DAY（返回指定日期的天）函数

DAY 函数返回一个整数，表示日期的“日”部分。

语法如下：

```
DAY(date)
```

参数 date 表示以日期格式返回有效的日期或字符串的表达式。

【例 7.34】 使用 DAY 函数，返回现有日期的日部分，SQL 语句及运行结果如图 7.33 所示。（**实例位置：光盘\TM\sl\7\34**）

SQL 语句如下：

```
SELECT DAY('2011-10-14') AS 'DAY'
```

【例 7.35】 使用 DAY 函数，返回当前日期的“日”部分的整数，SQL 语句及运行结果如图 7.34 所示。（**实例位置：光盘\TM\sl\7\35**）

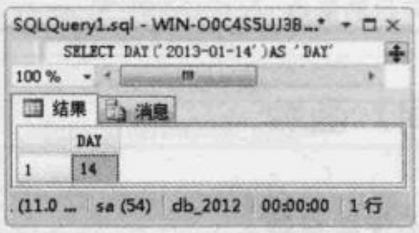


图 7.33 返回现有日期的日部分

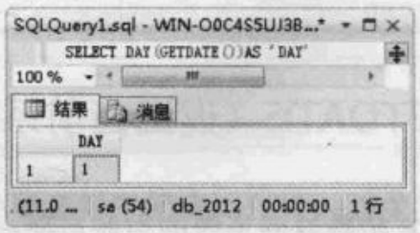


图 7.34 返回当前日期的日部分

SQL 语句如下：

```
SELECT DAY(GETDATE()) AS 'DAY'
```

7.4.4　MONTH（返回指定日期的月）函数

MONTH 函数返回一个表示日期中的“月份”日期部分的整数。

语法如下：

```
MONTH(date)
```

参数 date 表示任意日期格式的日期。

【例 7.36】 使用 MONTH 函数，返回指定日期时间的月份，SQL 语句及运行结果如图 7.35 所示。（**实例位置：光盘\TM\sl\7\36**）

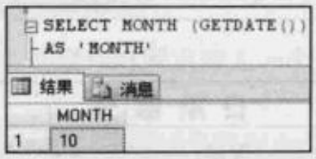


图 7.35　返回当前日期的月份

SQL 语句如下：

```
SELECT MONTH(GETDATE()) AS 'MONTH'
```

7.4.5　YEAR（返回指定日期的年）函数

YEAR 函数用于返回指定日期的年份。

语法如下：

```
YEAR(date)
```

参数 date 表示返回类型为 datetime 或 smalldatetime 的日期表达式。

有关 YEAR 函数使用的几点说明如下。

☑ 该函数等价于 DATEPART(yy,date)。

☑ SQL Server 数据库将 0 解释为 1900 年 1 月 1 日。

☑ 在使用日期函数时，其日期只应在 1753—9999 年之间，这是 SQL Server 系统所能识别的日期范围，否则会出现错误。

【例 7.37】 使用 YEAR 函数，返回指定日期时间的年份，SQL 语句及运行结果如图 7.36 所示。（**实例位置：光盘\TM\sl\7\37**）

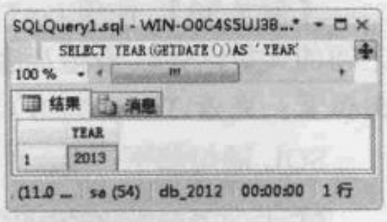


图 7.36　返回当前日期的年份

SQL 语句如下：

```
SELECT YEAR(GETDATE()) AS 'YEAR'
```

7.4.6　DATEDIFF（返回日期和时间的边界数）函数

DATEDIFF 函数用于返回日期和时间的边界数。

语法如下：

```
DATEDIFF(datepart,startdate,enddate)
```

参数说明如下。

☑ datepart 规定了应在日期的哪一部分计算差额的参数。

☑ startdate 表示计算的开始日期，startdate 是返回 datetime 值、smalldatetime 值或日期格式字符串的表达式。

☑ enddate 表示计算的终止日期。enddate 是返回 datetime 值、smalldatetime 值或日期格式字符串的表达式。

SQL Server 识别的日期部分和缩写如表 7.9 所示。

表 7.9 日期部分和缩写对照表

日 期 部 分	缩　写	日 期 部 分	缩　写
Year	yy,yyyy	Week	wk, ww
quarter	qq, q	Hour	hh
month	mm, m	minute	mi, n
dayofyear	dy, y	second	ss, s
day	dd, d	millisecond	ms

有关 DATEDIFF 函数使用的几点说明如下。

☑ startdate 是从 enddate 中减去。如果 startdate 比 enddate 晚，则返回负值。

☑ 当结果超出整数值范围，DATEDIFF 产生错误。对于毫秒，最大数是 24 天 20 小时 31 分钟零 23.647 秒。对于秒，最大数是 68 年。

☑ 计算跨分钟、秒和毫秒这些边界的方法，使得 DATEDIFF 给出的结果在全部数据类型中是一致的。结果是带正负号的整数值，其等于跨第一个和第二个日期间的 datepart 边界数。例如，在 1 月 4 日（星期日）和 1 月 11 日（星期日）之间的星期数是 1。

【例 7.38】 使用 DATEDIFF 函数，返回两个日期之间的天数，SQL 语句及运行结果如图 7.37 所示。（**实例位置：光盘\TM\sl\7\38**）

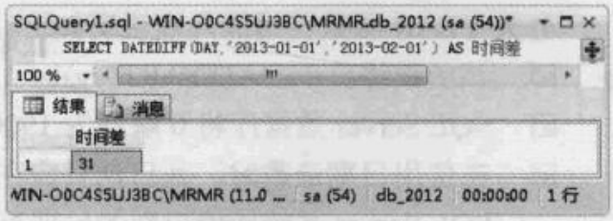


图 7.37 返回两个日期之间的天数

SQL 语句如下：

```
SELECT DATEDIFF(DAY,'2011-10-14','2011-11-14') AS 时间差距
```

7.4.7 DATEADD（添加日期时间）函数

DATEADD 函数将表示日期或时间间隔的数值与日期中指定的日期部分相加后，返回一个新的 DT_DBTIMESTAMP 值。number 参数的值必须为整数，而 date 参数的取值必须为有效日期。

语法如下：

```
DATEADD(datepart, number, date)
```

参数说明如下。

☑ datepart：指定要与数值相加的日期部分的参数。

☑ number：用于与 datepart 相加的值。该值必须是分析表达式时已知的整数值。

☑ date：返回有效日期或日期格式的字符串的表达式。

SQL Server 识别的日期部分和缩写如表 7.9 所示。

注意

如果指定一个不是整数的值，则将废弃此值的小数部分。

【例 7.39】使用 DATEADD 函数，在现在时间上加上一个月，SQL 语句及运行结果如图 7.38 所示。（**实例位置：光盘\TM\sl\7\39**）

图 7.38　在现在时间上加上一个月

SQL 语句如下：

```
SELECT GETDATE() AS '现在时间'
SELECT DATEADD("Month", 1,GETDATE())
AS '加一个月的时间'
```

【例 7.40】使用 DATEADD 函数，在现在时间上加上两天，SQL 语句及运行结果如图 7.39 所示。（**实例位置：光盘\TM\sl\7\40**）

SQL 语句如下：

```
SELECT GETDATE() AS '现在时间'
SELECT DATEADD("DAY", 2,GETDATE())
AS '加两天的时间'
```

【例 7.41】使用 DATEADD 函数，在现在时间上加上一年，SQL 语句及运行结果如图 7.40 所示。（**实例位置：光盘\TM\sl\7\41**）

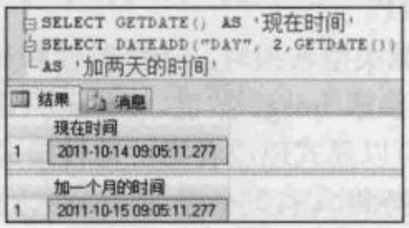


图 7.39　在现在时间上加两天

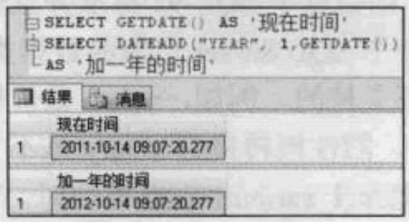


图 7.40　在现在时间上加上一年

SQL 语句如下：

```
SELECT GETDATE() AS '现在时间'
SELECT DATEADD("YEAR", 1,GETDATE())
AS '加一年的时间'
```

7.5　转 换 函 数

视频讲解：光盘\TM\lx\7\转换函数.mp4

如果 SQL Server 没有自动执行数据类型的转换，可以使用 CAST 和 CONVERT 转换函数将一种数据类型的表达式转换为另一种数据类型的表达式。例如，如果比较 char 和 datetime 表达式、smallint

和 int 表达式或不同长度的 char 表达式，则 SQL Server 自动对这些表达式进行转换。

7.5.1 转换函数概述

当遇到类型转换的问题时，可以使用 SQL Server 所提供的 CAST 和 CONVERT 函数。这两种函数不但可以将指定的数据类型转换为另一种数据类型，还可用来获得各种特殊的数据格式。CAST 和 CONVERT 函数都可用于选择列表、WHERE 子句和允许使用表达式的任何地方。

在 SQL Server 中数据类型转换分为两种，分别如下。

- ☑ 隐性转换：SQL Server 自动处理某些数据类型的转换。例如，如果比较 char 和 datetime 表达式、smallint 和 int 表达式或不同长度的 char 表达式，SQL Server 可将它们自动转换，这种转换称为隐性转换，对这些转换不必使用 CAST 函数。
- ☑ 显式转换：显式转换是指 CAST 和 CONVERT 函数，CAST 和 CONVERT 函数将数值从一种数据类型（局部变量、列或其他表达式）转换到另一种数据类型。

说明

隐性转换对用户是不可见的，SQL Server 自动将数据从一种数据类型转换成另一种数据类型。例如，如果一个 smallint 变量和一个 int 变量相比较，这个 smallint 变量在比较前即被隐性转换成 int 变量。

有关转换函数使用的几点说明如下。

- ☑ CAST 函数基于 SQL-92 标准并且优先于 CONVERT。
- ☑ 当从一个 SQL Server 对象的数据类型向另一个数据类型转换时，一些隐性和显式数据类型转换是不支持的。例如，nchar 数值根本就不能被转换成 image 数值。nchar 只能显式地转换成 binary，隐性地转换到 binary 是不支持的。nchar 可以显式地或者隐性地转换成 nvarchar。
- ☑ 当处理 sql_variant 数据类型时，SQL Server 支持将具有其他数据类型的对象隐性转换成 sql_variant 类型。然而，SQL Server 并不支持从 sql_variant 数据类型隐性地转换到其他数据类型的对象。

7.5.2 CAST 函数

CAST 函数用于将某种数据类型的表达式显式转换为另一种数据类型。

语法如下：

```
CAST(expression AS data_type)
```

参数说明如下。

- ☑ expression：表示任何有效的 SQL Server 表达式
- ☑ AS：用于分隔两个参数，在 AS 之前的是要处理的数据，在 AS 之后的是要转换的数据类型。

☑ data_type：表示目标系统所提供的数据类型，包括 bigint 和 sql_variant，不能使用用户定义的数据类型。

使用 CAST 函数进行数据类型转换时，在下列情况下能够被接受。

☑ 两个表达式的数据类型完全相同。

☑ 两个表达式可隐性转换。

☑ 必须显式转换数据类型。

如果试图进行不可能的转换（例如，将含有字母的 char 表达式转换为 int 类型），SQL Server 将显示一条错误信息。

如果转换时没有指定数据类型的长度，则 SQL Server 自动提供长度为 30。

【例 7.42】 使用 CAST 函数将字符串“MINGRIKEJI”转换为 NVARCHAR(6)类型，SQL 语句及运行结果如图 7.41 所示。（**实例位置：光盘\TM\sl\7\42**）

图 7.41　使用 CAST 函数转换字符串

SQL 语句如下：

```
SELECT CAST('MINGRIKEJI' AS NVARCHAR(6)) AS 结果
```

7.5.3　CONVERT 函数

CONVERT 函数与 CAST 函数的功能相似。该函数不是一个 ANSI 标准 SQL 函数，它可以按照指定的格式将数据转换为另一种数据类型。

语法如下：

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

参数说明如下。

☑ data_type：表示目标系统所提供的数据类型，包括 bigint 和 sql_variant。不能使用用户定义的数据类型。

☑ length：为 nchar、nvarchar、char、varchar、binary 和 varbinary 数据类型的可选参数。参数 expression 表示任何有效的 SQL Server 表达式。

☑ style：为日期样式，指定当将 datetime 数据转换为某种字符数据时或将某种字符数据转换为 datetime 数据时会使用 style 中的样式。

style 日期样式如表 7.10 所示。

表 7.10　style 日期样式

样　式	说　明	输入/输出格式
0 或 100(*)	默认值	mon dd yyyy hh:mi AM（或者 PM）
1/101	美国	mm/dd/yyyy
2/102	ANSI	yy.mm.dd
3/103	英国/法国	dd/mm/yy
4/104	德国	dd.mm.yy
5/105	意大利	dd-mm-yy
6/106	-	dd mon yy
7/107	-	mon dd,yy
8/108	-	hh:mm:ss
9 或 109(*)	默认值+毫秒	mon dd yyyy hh:mi:ss:mmmAM（或者 PM）
10 或 110	美国	mm-dd-yy
11 或 111	日本	yy/mm/dd
12 或 112	ISO	yymmdd
13 或 113(*)	欧洲默认值+毫秒	dd mon yyyy hh:mm:ss:mmm（24h）
14 或 114	-	hh:mi:ss:mmm（24h）
20 或 120(*)	ODBC 规范	yyyy-mm-dd hh:mm:ss（24h）
21 或 121(*)	ODBC 规范（带毫秒）	yyyy-mm-dd hh:mm:ss.mmm（24h）
126	ISO8601	yyyy-mm-dd Thh:mm:ss:mmm（不含空格）
130	科威特	dd mon yyyy hh:mi:ss:mmmAM（或者 PM）
131	科威特	dd/mm/yy hh:mi:ss.mmmAM（或者 PM）

【例 7.43】 显示当前日期和时间，并使用 CAST 将当前日期和时间改为字符数据类型，然后使用 CONVERT 以 ISO 8901 格式显示日期和时间，SQL 语句及运行结果如图 7.42 所示。（**实例位置：光盘\TM\sl\7\43**）

```
SELECT
    GETDATE() AS UnconvertedDateTime,
    CAST(GETDATE() AS nvarchar(30)) AS UsingCast,
    CONVERT(nvarchar(30), GETDATE(), 126) AS UsingConvertTo_ISO8601 ;
GO
```

结果　消息

	UnconvertedDateTime	UsingCast	UsingConvertTo_ISO8601
1	2011-10-14 10:57:24.557	10 14 2011 10:57AM	2011-10-14T10:57:24.557

图 7.42　转换数据类型（1）

SQL 语句如下：

```
SELECT
    GETDATE() AS UnconvertedText,
    CAST(GETDATE() AS datetime) AS UsingCast,
    CONVERT(datetime, GETDATE(), 126) AS UsingConvertFrom_ISO8601 ;
GO
```

【例 7.44】 将当前日期和时间显示为字符数据，并使用 CAST 将字符数据改为 datetime 数据类型，

然后使用 CONVERT 将字符数据改为 datetime 数据类型，SQL 语句及运行结果如图 7.43 所示。（**实例位置：光盘\TM\sl\7\44**）

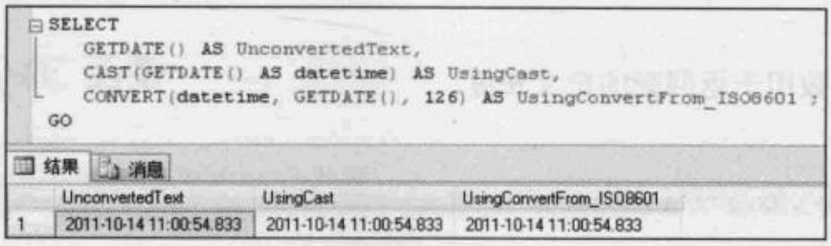


图 7.43　转换数据类型（2）

SQL 语句如下：

```
SELECT
   GETDATE() AS UnconvertedText,
   CAST(GETDATE() AS datetime) AS UsingCast,
   CONVERT(datetime, GETDATE(), 126) AS UsingConvertFrom_ISO8601 ;
GO
```

7.6　元数据函数

视频讲解：光盘\TM\lx\7\元数据函数.mp4

元数据函数主要是返回与数据库相关的信息，本节介绍常用的元数据函数 COL_LENGTH 函数、COL_NAME 函数和 DB_NAME 函数。

7.6.1　元数据函数概述

元数据函数描述了数据的结构和意义，主要用于返回数据库中的相应信息，其中包括：

☑　返回数据库中数据表或视图的个数和名称。

☑　返回数据表中数据字段的名称、数据类型、长度等描述信息。

☑　返回数据表中定义的约束、索引、主键或外键等信息。

常用的元数据函数及说明如表 7.11 所示。

表 7.11　常用的元数据函数及说明

函 数 名 称	说　明
COL_LENGTH	返回列的定义长度（以字节为单位）
COL_NAME	返回数据库列的名称，该列具有相应的表标识号和列标识号
DB_NAME	返回数据库名
OBJECT_ID	返回数据库对象标识号

7.6.2 COL_LENGTH 函数

COL_LENGTH 函数用于返回列的定义长度。

语法如下：

```
COL_LENGTH( 'table' , 'column' )
```

说明

参数 table 表示数据表名称，参数 column 表示数据表的列名称。

【例 7.45】 首先创建一个数据表，然后使用 COL_LENGTH 函数返回指定列定义的长度，SQL 语句及运行结果如图 7.44 所示。（**实例位置：光盘\TM\sl\7\45**）

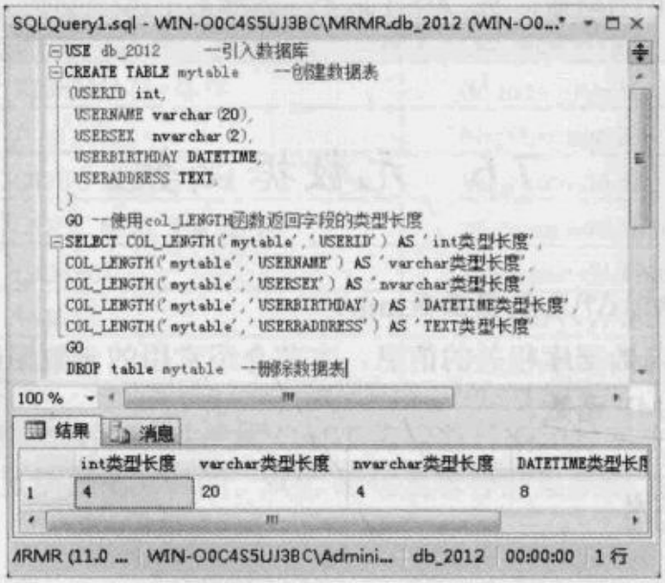


图 7.44 返回字段类型的长度

SQL 语句如下：

```
USE db_2012                                    --引入数据库
CREATE TABLE mytable                           --创建数据表
 (USERID int,
 USERNAME varchar(20),
 USERSEX nvarchar (2),
 USERBIRTHDAY DATETIME,
 USERADDRESS TEXT,
 )
GO                                             --使用 COL_LENGTH 函数返回字段的类型长度
SELECT COL_LENGTH ( 'mytable' , 'USERID' ) AS 'int 类型长度',
       COL_LENGTH ( 'mytable' , 'USERNAME' ) AS 'varchar 类型长度',
            COL_LENGTH ( 'mytable' , 'USERSEX' ) AS 'nvarchar 类型长度',
            COL_LENGTH ( 'mytable' , 'USERBIRTHDAY' ) AS 'DATETIME 类型长度',
```

```
        COL_LENGTH ( 'mytable' , 'USERADDRESS' )AS 'TEXT 类型长度'
GO
DROP table mytable                    --删除数据表
```

7.6.3　COL_NAME 函数

COL_NAME 函数用于返回数据库列的名称。

语法如下：

```
COL_NAME( table_id , column_id )
```

参数说明如下。

☑　table_id：包含数据库列的表的标识号，table_id 属于 int 类型。

☑　column_id：表示列的标识号，column_id 属于 int 类型。

【例 7.46】 使用 COL_NAME 函数，返回 db_2012 数据库的 Employee 表中首列的名称，SQL 语句及运行结果如图 7.45 所示。（**实例位置：光盘\TM\sl\7\46**）

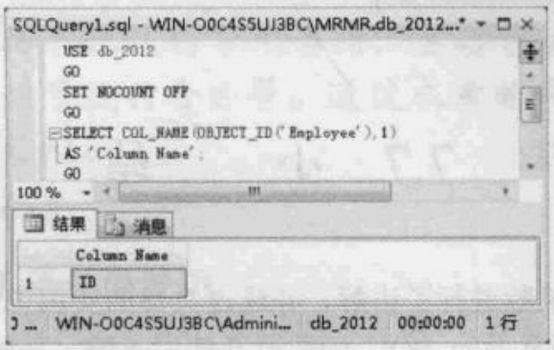


图 7.45　返回 Employee 表中首列的名称

SQL 语句如下：

```
USE db_2012
GO
SET NOCOUNT OFF
GO
SELECT COL_NAME(OBJECT_ID('Employee'), 1)
AS 'Column Name';
GO
```

7.6.4　DB_NAME 函数

DB_NAME 函数返回数据库名称。

语法如下：

```
DB_NAME( [ database_id ] )
```

参数说明如下。

☑ database_id：要返回的数据库的标识号（ID）。database_id 的数据类型为 int，无默认值。如果未指定 ID，则返回当前数据库名称。

☑ 返回类型：nvarchar(128)类型。

【例 7.47】 使用 DB_NAME 函数，返回当前数据库的名称，SQL 语句及运行结果如图 7.46 所示。（**实例位置：光盘\TM\sl\7\47**）

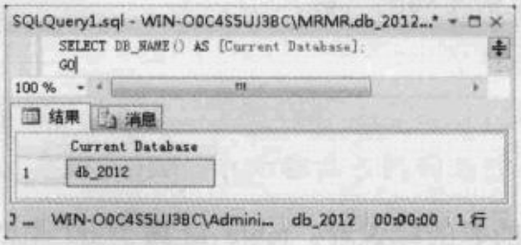


图 7.46 返回当前数据库的名称

SQL 语句如下：

```
SELECT DB_NAME() AS [Current Database];
GO
```

7.7 小 结

本章主要对 SQL 中常用的函数进行了讲解，并通过具体的实例说明了各个函数的使用方法。通过本章的学习，读者应该能够掌握常用的 SQL 函数及其使用方法，并能够在实际应用中使用这些 SQL 函数提高工作的效率。

7.8 实践与练习

1．在图书信息（TB_ASPNETBOOK）中查询出版日期在 10 月份的图书名称以及其出版日期，并按出版日期排序。（**答案位置：光盘\TM\sl\7\48**）

2．将学生报名数据表 tb_stu05 中的“学号”字段的第二个和第 3 个字符删除，然后在此位置插入新的字符串“200900”，生成新的字符串。（**答案位置：光盘\TM\sl\7\49**）

3．利用 MAX 函数实现在销售员表（tb_Seller）中查询月销售额最多的员工信息。（**答案位置：光盘\TM\sl\7\50**）

第 8 章

SQL 数据查询基础

（视频讲解：48 分钟）

本章主要介绍针对数据表记录的常用查询，主要包括使用 SELECT 检索数据、使用 UNION 将多个查询结果进行合并等。通过本章的学习，读者可以应用各种查询对数据表中的记录进行访问。

通过阅读本章，您可以：

- 熟练掌握 SELECT 语句的使用
- 掌握如何对 SQL 查询进行分组、排序
- 掌握如何查询不重复记录及前几条记录
- 熟悉 UNION 合并语句的使用
- 掌握常见的几种合并查询

8.1 SELECT 检索数据

视频讲解：光盘\TM\lx\8\Select 检索数据.mp4

查询是 SQL 的中心内容，而用于表示 SQL 查询的 SELECT 语句，是 SQL 语句中功能最强大也是最复杂的语句。

SELECT 语句的作用是让数据库服务器根据客户的要求搜索出所需要的信息资料，并按规定的格式进行整理，再返回给客户端。

8.1.1 SELECT 语句的基本结构

SELECT 语句主要是从数据库中检索行，并允许从一个或多个表中选择一个或多个行或列。SELECT 语句的语法如下：

```
<SELECT statement> ::=
    [WITH <common_table_expression> [,...n]]
    <query_expression>
    [ ORDER BY { order_by_expression | column_position [ ASC | DESC ] }
  [ ,...n ] ]
    [ COMPUTE
  { { AVG | COUNT | MAX | MIN | SUM } ( expression ) } [ ,...n ]
  [ BY expression [ ,...n ] ]
    ]
    [ <FOR Clause>]
    [ OPTION ( <query_hint> [ ,...n ] ) ]
<query_expression> ::=
    { <query_specification> | ( <query_expression> ) }
    [   { UNION [ ALL ] | EXCEPT | INTERSECT }
          <query_specification> | ( <query_expression> ) [...n ] ]
<query_specification> ::=
SELECT [ ALL | DISTINCT ]
    [TOP expression [PERCENT] [ WITH TIES ] ]
    < select_list >
    [ INTO new_table ]
    [ FROM { <table_source> } [ ,...n ] ]
    [ WHERE <search_condition> ]
    [ <GROUP BY> ]
    [ HAVING < search_condition > ]
```

虽然 SELECT 语句的完整语法较复杂，但其主要子句可归纳如下：

```
[ WITH <common_table_expression>]
SELECT select_list [ INTO new_table ]
[ FROM table_source ] [ WHERE search_condition ]
```

```
[ GROUP BY group_by_expression]
[ HAVING search_condition]
[ ORDER BY order_expression [ ASC | DESC ] ]
```

Select 语句的参数及说明如表 8.1 所示。

表 8.1　SELECT 语句的参数及说明

参　数	描　述
WITH <common_table_expression>	指定临时命名的结果集，这些结果集称为公用表表达式
select_list	指定由查询返回的列。它是一个逗号分隔的表达式列表。每个表达式同时定义格式（数据类型和大小）和结果集列的数据来源。每个选择列表表达式通常是对从中获取数据的表源或视图的列的引用，但也可能是其他表达式，例如常量或 Transact-SQL 函数。在选择列表中使用*表达式指定返回源表中的所有列
INTO new_table_name	创建新表并将查询行从查询插入新表中。new_table_name 指定新表的名称
FROM table_source	指定从其中检索行的表。这些来源可能包括基表、视图和链接表。FROM 子句还可包含联接说明，该说明定义了 SQL Server 用来在表之间进行导航的特定路径。FROM 子句还用在 DELETE 和 UPDATE 语句中以定义要修改的表
WHERE　search_conditions	WHERE 子句指定用于限制返回的行的搜索条件。WHERE 子句还用在 DELETE 和 UPDATE 语句中以定义目标表中要修改的行
group_by_expression	GROUP BY 子句根据 group_by_list 列中的值将结果集分成组。例如，student 表在“性别”中有两个值。GROUP BY 性别子句将结果集分成两组，每组对应于性别的一个值
HAVING search_condition	HAVING 子句是指定组或聚合的搜索条件。从逻辑上讲，HAVING 子句从中间结果集中对行进行筛选，这些中间结果集是用 SELECT 语句中的 FROM、WHERE 或 GROUP BY 子句创建的。HAVING 子句通常与 GROUP BY 子句一起使用，尽管 HAVING 子句前面不必有 GROUP BY 子句
ORDER BY order_expression [ASC | DESC]	ORDER BY 子句定义结果集中的行排列的顺序。order_list 指定组成排序列表的结果集的列。ASC 和 DESC 关键字用于指定行是按升序还是按降序排序。ORDER BY 之所以重要，是因为关系理论规定，除非已经指定 ORDER BY，否则不能假设结果集中的行带有任何序列。如果结果集行的顺序对于 SELECT 语句来说很重要，那么在该语句中就必须使用 ORDER BY 子句

8.1.2　WITH 子句

WITH 子句用于指定临时命名的结果集，这些结果集称为公用表表达式。该表达式源自简单查询，并且在单条 SELECT、INSERT、UPDATE 或 DELETE 语句的执行范围内定义。

语法如下：

```
 [ WITH <common_table_expression> [ ,...n ] ]
<common_table_expression>::=
        expression_name [ ( column_name [ ,...n ] ) ]
    AS
        ( CTE_query_definition )
```

参数说明如下。

☑ expression_name：公用表表达式的有效标识符。

☑ column_name：在公用表表达式中指定列名。

☑ CTE_query_definition：指定一个其结果集填充公用表表达式的 SELECT 语句。

【例 8.1】 创建公用表表达式，计算 Employee 数据表中 Age 字段中每一年龄员工的数量。（**实例位置：光盘\TM\sl\8\1**）

SQL 语句如下：

```
USE db_2012
WITH AgeReps(Age, AgeCount) AS
(
    SELECT Age, COUNT(*)
    FROM Employee AS AgeReports
    WHERE Age IS NOT NULL
    GROUP BY Age
)
SELECT Age, AgeCount
FROM AgeReps
```

运行结果如图 8.1 所示，Employee 表中的数据信息如图 8.2 所示。

【例 8.2】 创建公用表表达式，计算 Employee 数据表中员工 Age 的平均值，SQL 语句及运行结果如图 8.3 所示。（**实例位置：光盘\TM\sl\8\2**）

	Age	AgeCount
1	22	1
2	23	1
3	24	3
4	25	3
5	26	2
6	27	2

图 8.1 公用表表达式运行结果

ID	Name	Sex	Age
001	张子婷	女	24
002	王子行	男	26
003	李开	女	25
004	赵小小	女	27
005	田飞飞	女	23
006	肖一子	男	24
007	王婷	女	22
008	王一	女	26
010	赵行	男	27
011	张子行	女	24
012	雨涵	女	25
013	雨欣	女	25

图 8.2 Employee 表中的数据信息

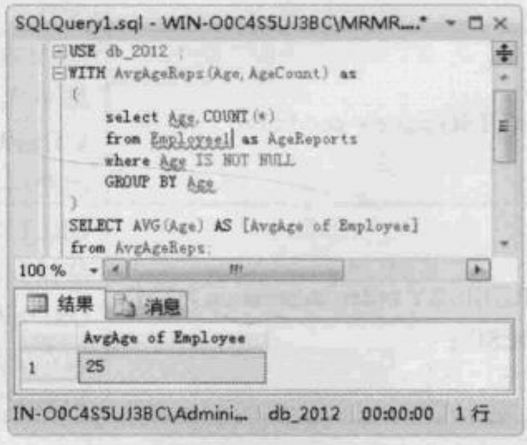


图 8.3 创建公用表表达式计算员工年龄平均值

SQL 语句如下：

```
USE db_2012
WITH AvgAgeReps(Age, AgeCount) AS
(
    SELECT Age, COUNT(*)
    FROM Employee AS AgeReports
    WHERE Age IS NOT NULL
    GROUP BY Age
)
SELECT AVG(Age) AS [AvgAge of Employee]
FROM AvgAgeReps
```

8.1.3　SELECT…FROM 子句

SELECT 表明要读取的信息，FROM 指定要从中获取数据的一个或多个表的名称。SELECT…FROM 就构成了一个基本的查询语句。

语法如下：

```
SELECT [ ALL | DISTINCT ]
[ TOP expression [ PERCENT ] [ WITH TIES ] ]
<select_list> [ FROM { <table_source> } [ ,...n ] ]
<select_list> ::=
    {
                *
      | { table_name | view_name | table_alias }.*
      | {
          [ { table_name | view_name | table_alias }. ]
              { column_name | $IDENTITY | $ROWGUID }
          | udt_column_name [ { . | :: } { { property_name | field_name }
            | method_name ( argument [ ,...n] ) } ]
          | expression
          [ [ AS ] column_alias ]
        }
      | column_alias = expression
    } [ ,...n ]
<table_source> ::=
{
        table_or_view_name [ [ AS ] table_alias ] [ <tablesample_clause> ]
        [ WITH ( < table_hint > [ [ , ]...n ] ) ]
    | rowset_function [ [ AS ] table_alias ]
        [ ( bulk_column_alias [ ,...n ] ) ]
        | user_defined_function [ [ AS ] table_alias ] [ (column_alias [ ,...n ] ) ]
    | OPENXML <openxml_clause>
    | derived_table [ AS ] table_alias [ ( column_alias [ ,...n ] ) ]
    | <joined_table>
    | <pivoted_table>
    | <unpivoted_table>
      | @variable [ [ AS ] table_alias ]
        | @variable.function_call ( expression [ ,...n ] ) [ [ AS ] table_alias ] [ (column_alias [ ,...n ] ) ]
}
<tablesample_clause> ::=
    TABLESAMPLE [SYSTEM] ( sample_number [ PERCENT | ROWS ] )
        [ REPEATABLE ( repeat_seed ) ]
<joined_table> ::=
{
    <table_source> <join_type> <table_source> ON <search_condition>
    | <table_source> CROSS JOIN <table_source>
    | left_table_source { CROSS | OUTER } APPLY right_table_source
```

```
    | [ ( ] <joined_table> [ ) ]
}
<join_type> ::=
    [ { INNER | { { LEFT | RIGHT | FULL } [ OUTER ] } } [ <join_hint> ] ]
    JOIN
```

SELECT…FROM 子句的参数及说明如表 8.2 所示。

表 8.2　SELECT…FROM 子句的参数及说明

参　数	描　述
ALL	指定在结果集中可以包含重复行。ALL 是默认值
DISTINCT	指定在结果集中只能包含唯一行。对于 DISTINCT 关键字来说，Null 值是相等的
TOP expression [PERCENT] [WITH TIES]	指示只能从查询结果集返回指定的第一组行或指定的百分比数目的行。expression 可以是指定数目或百分比数目的行
< select_list>	要为结果集选择的列表。选择列表是以逗号分隔的一系列表达式。可在选择列表中指定的表达式的最大数目是 4096
<table_source>	要从中获取数据的表的名称
*	指定返回 FROM 子句中的所有表和视图中的所有列，这些列按 FROM 子句中指定的表或视图顺序返回，并对应于它们在表或视图中的顺序
table_ name | view_ name | table_ alias.*	将*的作用域限制为指定的表或视图
column_ name	要返回的列名
expression	常量、函数以及由一个或多个运算符联接的列名、常量和函数的任意组合，或者是子查询。例如，在表达式中可以使用行聚合函数（又称为统计函数），SQL Server 中常用的行聚合函数如表 8.3 所示
<table_source>	指定要在 Transact-SQL 语句中使用的表、视图或派生表源（有无别名均可）
table_or_view_name	表或视图的名称
WITH (<table_hint>)	指定查询优化器对此表和此语句使用优化或锁定策略
rowset_function	指定其中一个行集函数（如 OPENROWSET），该函数返回可用于替代表引用的对象
[AS] table_alias	table_source 的别名，别名可带来使用上的方便，也可用于区分自联接或子查询中的表或视图。别名往往是一个缩短了的表名，用于在联接中引用表的特定列
<tablesample_clause>	指定返回来自表的数据样本
bulk_column_alias	代替结果集内列名的可选别名。只允许在使用 OPENROWSET 函数和 BULK 选项的 SELECT 语句中使用列别名
user_defined_function	指定表值函数
OPENXML <openxml_clause>	通过 XML 文档提供行集视图
derived_table	从数据库中检索行的子查询。derived_table 用作外部查询的输入
column_alias	代替派生表的结果集内列名的可选别名。在选择列表中的每个列包括一个列别名，并将整个列别名列表用圆括号括起来
SYSTEM	ISO 标准指定的依赖于实现的抽样方法
sample_number	表示行的百分比或行数的精确或近似的常量数值表达式
PERCENT	指定应该从表中检索表行的 sample_number 百分比
ROWS	指定将检索的行的近似 sample_number

续表

参　数	描　述
REPEATABLE	指示可以再次返回选定的样本
repeat_seed	SQL Server 用于生成随机数的常量整数表达式
<joined_table>	由两个或更多表的积构成的结果集
<join_type>	指定联接操作的类型
INNER	指定返回所有匹配的行对
FULL [OUTER]	指定在结果集中包括左表或右表中不满足联接条件的行，并将对应于另一个表的输出列设为 NULL。这是对通常由 INNER JOIN 返回的所有行的补充
LEFT [OUTER]	指定在结果集中包括左表中所有不满足联接条件的行，并在由内部联接返回所有的行之外，将另外一个表的输出列设为 NULL
RIGHT [OUTER]	指定在结果集中包括右表中所有不满足联接条件的行，且在由内部联接返回的所有行之外，将与另外一个表对应的输出列设为 NULL
<join_hint>	指定 SQL Server 查询优化器为在查询的 FROM 子句中指定的每个联接使用一个联接提示或执行算法
JOIN	指示指定的联接操作应在指定的表源或视图之间执行

表 8.3　常用的行聚合函数和功能

行聚合函数	描　述
COUNT(*)	返回组中的项数
COUNT ({ [[ALL | DISTINCT] 列名] })	返回某列的个数
AVG ({ [[ALL | DISTINCT] 列名] })	返回某列的平均值
MAX ({ [[ALL | DISTINCT] 列名] })	返回某列的最大值
MIN ({ [[ALL | DISTINCT] 列名] })	返回某列的最小值
SUM ({ [[ALL | DISTINCT] 列名] })	返回某列值的和

【例 8.3】 查询 Employee 表中的所有列的信息，SQL 语句及运行结果如图 8.4 所示。（**实例位置：光盘\TM\sl\8\3**）

SQL 语句如下：

```
SELECT ID,Name,Sex,Age
FROM Employee
```

上面的查询语句还等价于：

```
SELECT * FROM Employee
```

【例 8.4】 查询 Employee 表中员工的 ID、Name、Sex，SQL 语句及运行结果如图 8.5 所示。（**实例位置：光盘\TM\sl\8\4**）

SQL 语句如下：

```
SELECT ID,Name,Sex
From Employee
```

在例 8.4 的查询语句中，还可以以表的名称作为前缀，代码如下：

```
use db_2012
SELECT Employee.ID,Employee.Name,Employee.Sex
FROM Employee
```

【例 8.5】 查询 Employee 表中所有信息，并分别为列起别名为 ID（员工编号），Name（姓名），Sex（性别），Age（年龄），SQL 语句及运行结果如图 8.6 所示。（**实例位置：光盘\TM\sl\8\5**）

```
SELECT ID,Name,Sex,Age
FROM Employee
```

结果 消息

	ID	Name	Sex	Age
1	001	张子婷	女	24
2	002	王子行	男	26
3	003	李开	女	25
4	004	赵小小	女	27
5	005	田飞飞	女	23
6	006	肖一子	男	24
7	007	王婷	女	22
8	008	王一	女	26
9	010	赵行	男	27
10	011	张子行	女	24
11	012	雨涵	女	25
12	013	雨欣	女	25

图 8.4 查询 Employee 表中的所有信息

```
SELECT ID,Name,Sex
From Employee
```

结果 消息

	ID	Name	Sex
1	001	张子婷	女
2	002	王子行	男
3	003	李开	女
4	004	赵小小	女
5	005	田飞飞	女
6	006	肖一子	男
7	007	王婷	女
8	008	王一	女
9	010	赵行	男
10	011	张子行	女
11	012	雨涵	女
12	013	雨欣	女

图 8.5 查询 Employee 表中某列的信息

```
SELECT Distinct 员工编号=ID,Name AS 姓名,
Sex 性别,Age 年龄
From Employee
```

结果 消息

	员工编号	姓名	性别	年龄
1	001	张子婷	女	24
2	002	王子行	男	26
3	003	李开	女	25
4	004	赵小小	女	27
5	005	田飞飞	女	23
6	006	肖一子	男	24
7	007	王婷	女	22
8	008	王一	女	26
9	010	赵行	男	27
10	011	张子行	女	24
11	012	雨涵	女	25
12	013	雨欣	女	25

图 8.6 为 Employee 表中的列起别名

SQL 语句如下：

```
SELECT Distinct 员工编号=ID,Name AS 姓名,
Sex 性别,Age 年龄
From Employee
```

例 8.5 中使用了别名的 3 种定义方法，分别如下：

☑ 别名=列名

☑ 列名 AS 别名

☑ 列名 别名

【例 8.6】 查询 Employee 表中的最大 Age，SQL 语句及运行结果如图 8.7 所示。（**实例位置：光盘\TM\sl\8\6**）

```
SELECT MAX(Age) 最大年龄
From Employee
```

结果 消息

	最大年龄
1	27

图 8.7 查询 Employee 表中的最大 Age

SQL 语句如下：

```
SELECT MAX(Age) 最大年龄
From Employee
```

8.1.4 INTO 子句

创建新表并将来自查询的结果行插入新表中。

语法如下：

```
[ INTO new_table ]
```

参数new_table用来根据选择列表中的列和WHERE子句选择的行，指定要创建的新表名。new_table的格式通过对选择列表中的表达式进行取值来确定。new_table 中的列按选择列表指定的顺序创建。new_table 中的每列与选择列表中的相应表达式具有相同的名称、数据类型和值。

【例 8.7】使用 INTO 子句创建一个新表 tb_Employe，tb_Employe 表中包含 Employe 表中的 Name 和 Age 字段。（**实例位置：光盘\TM\sl\8\7**）

SQL 语句如下：

```
use db_2012
select   Name,Age   into tb_Employee From Employee
```

tb_Employe 表中的记录如图 8.8 所示。

MR-NXT\NXT.db_20... dbo.tb_Employee

	Name	Age
▶	张子婷	24
	王子行	26
	李开	25
	赵小小	27
	田飞飞	23
	肖一子	24
	王婷	22
	王一	26
	赵行	27
	张子行	24
	雨涵	25
	雨欣	25
*	NULL	NULL

图 8.8　tb_Employe 表中的记录

8.1.5　WHERE 子句

指定查询返回的行的搜索条件。语法如下：

```
WHERE <search_condition>
< search_condition > ::=
    { [ NOT ] <predicate> | ( <search_condition> ) }
    [ { AND | OR } [ NOT ] { <predicate> | ( <search_condition> ) } ]
[ ,...n ]
<predicate> ::=
    { expression { = | < > | ! = | > | > = | ! > | < | < = | ! < } expression
    | string_expression [ NOT ] LIKE string_expression
  [ ESCAPE 'escape_character' ]
    | expression [ NOT ] BETWEEN expression AND expression
    | expression IS [ NOT ] NULL
    | CONTAINS
    ( { column | * } , '< contains_search_condition >' )
    | FREETEXT ( { column | * } , 'freetext_string' )
    | expression [ NOT ] IN ( subquery | expression [ ,...n ] )
    | expression { = | < > | ! = | > | > = | ! > | < | < = | ! < }
  { ALL | SOME | ANY} ( subquery )
    | EXISTS ( subquery )       }
```

WHERE 子句的参数及说明如表 8.4 所示。

表 8.4 WHERE 子句的参数及说明

参　数	描　述
<search_condition>	指定要在 SELECT 语句、查询表达式或子查询的结果集中返回的行的条件
NOT	对谓词指定的布尔表达式求反
AND	组合两个条件，并在两个条件都为 TRUE 时取值为 TRUE
OR	组合两个条件，并在任何一个条件为 TRUE 时取值为 TRUE
< predicate>	返回 TRUE、FALSE 或 UNKNOWN 的表达式
expression	列名、常量、函数、变量、标量子查询，或者是通过运算符或子查询联接的列名、常量和函数的任意组合。表达式还可以包含 CASE 函数
string_expression	字符串和通配符
[NOT] LIKE	指示后续字符串使用时要进行模式匹配
ESCAPE 'escape_character'	允许在字符串中搜索通配符，而不是将其作为通配符使用。escape_character 是放在通配符前表示此特殊用法的字符
[NOT] BETWEEN	指定值的包含范围。使用 AND 分隔开始值和结束值
IS [NOT] NULL	根据使用的关键字，指定是否搜索空值或非空值。如果有任何一个操作数为 NULL，则包含位运算符或算术运算符的表达式的计算结果为 NULL
CONTAINS	在包含字符数据的列中，搜索单个词和短语的精确或不精确（“模糊”）的匹配项、在一定范围内相同的近似词以及加权匹配项
FREETEXT	在包含字符数据的列中，搜索与谓词中的词的含义相符而非精确匹配的值，提供一种形式简单的自然语言查询。此选项只能与 SELECT 语句一起使用
[NOT] IN	根据是在列表中包含还是排除某表达式，指定对该表达式的搜索
Subquery	可以看成是受限的 SELECT 语句，与 SELECT 语句中的<query_expresssion>相似。不允许使用 ORDER BY 子句、COMPUTE 子句和 INTO 关键字
ALL	与比较运算符和子查询一起使用。如果子查询检索的所有值都满足比较运算，则为<predicate>，返回 TRUE；如果并非所有值都满足比较运算或子查询未向外部语句返回行，则返回 FALSE
{ SOME | ANY }	与比较运算符和子查询一起使用。如果子查询检索的任何值都满足比较运算，则为<谓词>，返回 TRUE；如果子查询内没有值满足比较运算或子查询未向外部语句返回行，则返回 FALSE。其他情况下，表达式为 UNKNOWN
EXISTS	与子查询一起使用，用于测试是否存在子查询返回的行

由于 WHERE 子句的复杂性，下面按参数的先后顺序进行详细的介绍。

1. 逻辑运算符（NOT、AND、OR）

如果想把几个单一条件组合成一个复合条件，这就需要使用逻辑运算符 NOT、AND 和 OR，才能完成复合条件查询。

NOT：对布尔型输入取反，使用 NOT 返回不满足表达式的行。

语法如下：

```
[ NOT ] boolean_expression
```

参数说明如下。

☑ boolean_expression：任何有效的布尔表达式。

☑ 结果类型：Boolean 类型。

AND：组合两个布尔表达式，当两个表达式均为 TRUE 时返回 TRUE。当语句中使用多个逻辑运算符时，将首先计算 AND 运算符。可以通过使用括号改变求值顺序。使用 AND 返回满足所有条件的行。

语法如下：

```
boolean_expression AND boolean_expression
```

参数说明如下。

☑ boolean_expression：返回布尔值的任何有效表达式：TRUE、FALSE 或 UNKNOWN。

☑ 结果类型：Boolean 类型。

OR：将两个条件组合起来。在一个语句中使用多个逻辑运算符时，在 AND 运算符之后对 OR 运算符求值。不过，使用括号可以更改求值的顺序。使用 OR 返回满足任一条件的行。

语法如下：

```
boolean_expression OR boolean_expression
```

参数说明如下。

☑ boolean_expression：返回 TRUE、FALSE 或 UNKNOWN 的任何有效表达式。

☑ 结果类型：Boolean 类型。

逻辑运算符的优先顺序是 NOT（最高），然后是 AND，最后是 OR。

【例 8.8】 使用 AND 查询 Employee 表中 Age 等于 23 的女员工信息，SQL 语句及运行结果如图 8.9 所示。（**实例位置：光盘\TM\sl\8\8**）

SQL 语句如下：

```
SELECT * FROM Employee
WHERE Sex='Ů' AND Age = 23
```

【例 8.9】 使用 OR 查询 Employee 表中编号是 001 或者是 002 的员工信息，SQL 语句及运行结果如图 8.10 所示。（**实例位置：光盘\TM\sl\8\9**）

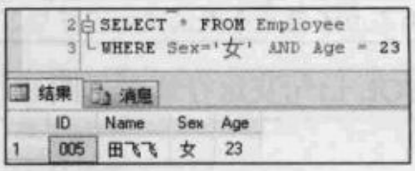


图 8.9　使用 AND 查询

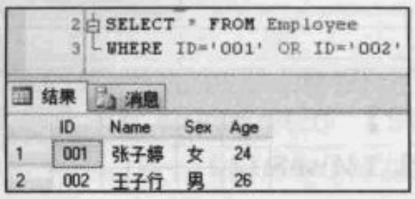


图 8.10　使用 OR 查询

SQL 语句如下：

```
SELECT * FROM Employee
WHERE ID='001' OR ID='002'
```

【例 8.10】 使用 NOT 查询 Employee 表中 Age 不大于 24 的男员工信息，SQL 语句及运行结果如图 8.11 所示。（**实例位置：光盘\TM\sl\8\10**）

SQL 语句如下：

```
SELECT * FROM Employee
WHERE Sex='男' AND NOT Age >24
```

【例 8.11】 使用 NOT、AND、OR 复合查询 Employee 表中 Age 不等于 24 的男员工信息或者 Age 等于 23 的女员工信息，SQL 语句及运行结果如图 8.12 所示。（**实例位置：光盘\TM\sl\8\11**）

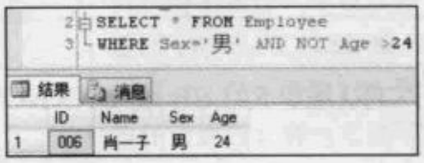


图 8.11 使用 NOT 查询

图 8.12 使用 NOT、AND、OR 复合查询

SQL 语句如下：

```
SELECT * FROM Employee
WHERE Sex='男' AND NOT Age=24
OR Sex='女' AND Age=23
```

2. 比较运算符

在 WHERE 子句中，允许出现的比较运算符如表 8.5 所示。

表 8.5 比较运算符

运 算 符	说 明
=	用于测试两个表达式是否相等的运算符
<>	用于测试两个表达式彼此不相等的条件的运算符
!=	用于测试两个表达式彼此不相等的条件的运算符
>	用于测试一个表达式是否大于另一个表达式的运算符
>=	用于测试一个表达式是否大于或等于另一个表达式的运算符
!>	用于测试一个表达式是否不大于另一个表达式的运算符
<	用于测试一个表达式是否小于另一个表达式的运算符
<=	用于测试一个表达式是否小于或等于另一个表达式的运算符
!<	用于测试一个表达式是否不小于另一个表达式的运算符

【例 8.12】 在 Employee 表中查询张子婷的详细信息，SQL 语句及运行结果如图 8.13 所示。（**实例位置：光盘\TM\sl\8\12**）

SQL 语句如下：

```
SELECT * FROM Employee
WHERE Name='张子婷'
```

说明

在 Employee 表中，“张子婷”属于 Name 列中的字段，所以在 WHERE 中的查询条件是 Name='张子婷'。

【例 8.13】 在 Employee 表中查询 Age 大于 24 岁的员工信息，SQL 语句及运行结果如图 8.14 所示。（**实例位置：光盘\TM\sl\8\13**）

图 8.13　查询张子婷的详细信息

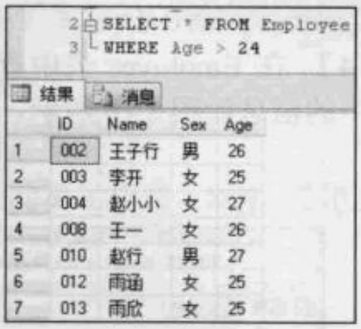


图 8.14　查询 Age 大于 24 岁的员工信息

SQL 语句如下：

```
SELECT * FROM Employee
WHERE Age > 24
```

3. Like 关键字

使用 Like 关键字可以确定特定字符串是否与指定模式相匹配。模式可以包含常规字符和通配符。模式匹配过程中，常规字符必须与字符串中指定的字符完全匹配。但是，通配符可以与字符串的任意部分相匹配。

语法如下：

```
match_expression [ NOT ] LIKE pattern [ ESCAPE escape_character ]
```

参数说明如下。

☑ match_expression：任何有效的字符数据类型的表达式。

☑ pattern：要在 match_expression 中搜索并且可以包括下列有效通配符的特定字符串。pattern 的最大长度可达 8000 字节。

在 WHERE 子句中，允许出现的通配符如表 8.6 所示。

表 8.6　通配符

通 配 符	说　明	示　例
%	包含零个或多个字符的任意字符串	WHERE title LIKE '%computer%'将查找在书名中任意位置包含单词"computer"的所有书名
_（下划线）	任何单个字符	WHERE au_fname LIKE '_ean'将查找以 ean 结尾的所有 4 个字母的名字（如 Dean、Sean 等）
[ ]	指定范围([a～f])或集合([abcdef])中的任何单个字符	WHERE au_lname LIKE '[C-P]arsen' 将查找以 arsen 结尾并且以介于 C 与 P 之间的任何单个字符开始的作者姓氏，例如 Carsen、Larsen、Karsen 等
[^]	不属于指定范围([a～f])或集合([abcdef])的任何单个字符	WHERE au_lname LIKE 'de[^l]%'将查找以 de 开始并且其后的字母不为 l 的所有作者的姓氏

☑ escape_character：放在通配符之前用于指示通配符应当解释为常规字符而不是通配符的字符。

escape_character 是字符表达式，无默认值，并且计算结果必须仅为一个字符。

（1）%通配符。

包含零个或多个字符的任意字符串。

【例 8.14】 在 Employee 表中查询姓“王”的员工信息，SQL 语句及运行结果如图 8.15 所示。Employee 表中的信息如图 8.16 所示。（**实例位置：光盘\TM\sl\8\14**）

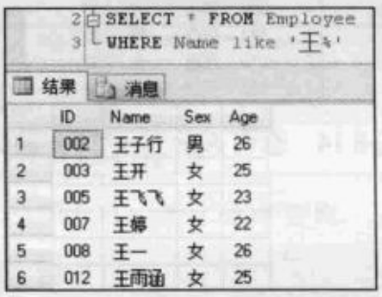


图 8.15 查询姓“王”的员工信息

ID	Name	Sex	Age
001	张子婷	女	24
002	王子行	男	26
003	王开	女	25
004	赵小小	女	27
005	王飞飞	女	23
006	肖一子	男	24
007	王婷	女	22
008	王一	女	26
010	赵行	男	27
011	张子行	女	24
012	王雨涵	女	25

图 8.16 Employee 表中的信息

SQL 语句如下：

```
SELECT * FROM Employee
WHERE Name like '王%'
```

说明

在 SQL Server 语句中，可以在查询条件的任意位置放置一个“%”符号来代表任意长度的字符串。在设置查询条件时，也可以放置两个“%”，但最好不要连续出现两个“%”符号。

（2）_（下划线）。

匹配任意单个字符。

【例 8.15】 在 Employee 表中查询姓“王”并且名字只是两个字的员工信息，SQL 语句及运行结果如图 8.17 所示。（**实例位置：光盘\TM\sl\8\15**）

SQL 语句如下：

```
SELECT * FROM Employee
WHERE Name LIKE '王_'
```

【例 8.16】 在 Employee 表中查询姓“王”并且最后一个字是“行”的员工信息，SQL 语句及运行结果如图 8.18 所示。（**实例位置：光盘\TM\sl\8\16**）

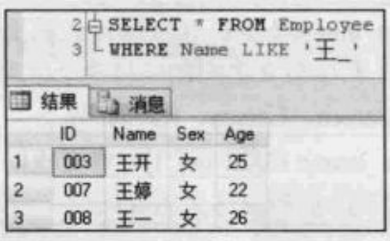


图 8.17 查询指定姓名个数的信息

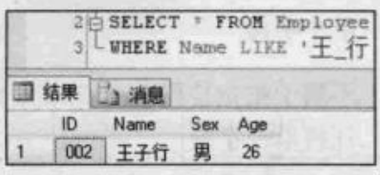


图 8.18 查询以指定字段开始和结尾的信息

SQL 语句如下：

```
SELECT * FROM Employee
WHERE Name LIKE '王_行'
```

（3）[ ]通配符。

[ ]通配符表示查询一定范围内的任意单个字符，它包含两端数据。

【例 8.17】 在 Employee 表中查询 Age 在 22～24 岁之间的员工信息，SQL 语句及运行结果如图 8.19 所示。（**实例位置：光盘\TM\sl\8\17**）

SQL 语句如下：

```
SELECT * FROM Employee
WHERE Age LIKE '2[2-4]'
```

说明

[2-4]表示从 2 到 4 的数，包括 2、3、4。'2[2-4]'就表示 22,23,24。

（4）[^]通配符。

[^]通配符表示查询不在一定范围内的任意单个字符，它包含两端数据。

【例 8.18】 在 Employee 表中查询 Age 不在 22～24 岁之间的员工信息，SQL 语句及运行结果如图 8.20 所示。（**实例位置：光盘\TM\sl\8\18**）

```
SELECT * FROM Employee
WHERE Age LIKE '2[2-4]'
```

结果　消息

	ID	Name	Sex	Age
1	001	张子婷	女	24
2	005	王飞飞	女	23
3	006	肖一子	男	24
4	007	王婷	女	22
5	011	张子行	女	24

图 8.19　查询 Age 在 22～24 岁之间的员工信息

```
SELECT * FROM Employee
WHERE Age LIKE '2[^2-4]'
```

结果　消息

	ID	Name	Sex	Age
1	002	王子行	男	26
2	003	王开	女	25
3	004	赵小小	女	27
4	008	王一	女	26
5	010	赵行	男	27
6	012	王雨涵	女	25

图 8.20　查询 Age 不在 22～24 岁之间的员工信息

SQL 语句如下：

```
SELECT * FROM Employee
WHERE Age LIKE '2[^2-4]'
```

4．BETWEEN 关键字

BETWEEN…AND 和 NOT…BETWEEN…AND 用来指定范围条件。使用 BETWEEN AND 查询条件时，指定的第一个值必须小于第二个值。因为 BETWEEN…AND 实质是查询条件“大于等于第一个值，并且小于等于第二个值”的简写形式。

语法如下：

```
test_expression [ NOT ] BETWEEN begin_expression AND end_expression
```

BETWEEN 关键字的参数及说明如表 8.7 所示。

表 8.7 BETWEEN 关键字的参数及说明

参　数	描　述
test_expression	要在由 begin_expression 和 end_expression 定义的范围内测试的表达式。test_expression 必须与 begin_expression 和 end_expression 具有相同的数据类型
NOT	指定谓词的结果被取反
begin_expression	任何有效的表达式。begin_expression 必须与 test_expression 和 end_expression 具有相同的数据类型
end_expression	任何有效的表达式。end_expression 必须与 test_expression 和 begin_expression 具有相同的数据类型
AND	用作一个占位符，指示 test_expression 应该处于由 begin_expression 和 end_expression 指定的范围内

【例 8.19】 在 Employee 表中查询 Age 在 22～24 岁之间的员工信息，SQL 语句及运行结果如图 8.21 所示。（**实例位置：光盘\TM\sl\8\19**）

SQL 语句如下：

```
SELECT * FROM Employee
WHERE Age BETWEEN 22 AND 24
```

NOT…BETWEEN…AND 语句返回某个数据值在两个指定值的范围以外的，但并不包括两个指定的值。

【例 8.20】 在 Employee 表中查询 Age 不在 22～24 岁之间的员工信息，SQL 语句及运行结果如图 8.22 所示。（**实例位置：光盘\TM\sl\8\20**）

```
SELECT * FROM Employee
WHERE Age BETWEEN 22 and 24
```

结果　消息

	ID	Name	Sex	Age
1	001	张子婷	女	24
2	005	王飞飞	女	23
3	006	肖一子	男	24
4	007	王婷	女	22
5	011	张子行	女	24

图 8.21　使用 BETWEEN…AND 查询员工信息

```
SELECT * FROM Employee
WHERE Age NOT BETWEEN 22 and 24
```

结果　消息

	ID	Name	Sex	Age
1	002	王子行	男	26
2	003	王开	女	25
3	004	赵小小	女	27
4	008	王一	女	26
5	010	赵行	男	27
6	012	王雨涵	女	25

图 8.22　使用 NOT…BETWEEN…AND 查询员工信息

SQL 语句如下：

```
SELECT * FROM Employee
WHERE Age NOT BETWEEN 22 AND 24
```

5. IS （NOT）NULL 关键字

在 WHERE 子句中不能使用比较运算符（=）对空值进行判断，只能使用 IS（NOT）NULL 对空值进行查询。

【例 8.21】 在 Employee 表中将张子婷和王子行的 Sex 列设置为空（NULL），然后在表中查询 Sex 为空的员工信息，SQL 语句及运行结果如图 8.23 所示。（**实例位置：光盘\TM\sl\8\21**）

```
SELECT * FROM Employee
WHERE Sex IS NULL
```

结果　消息

	ID	Name	Sex	Age
1	001	张子婷	NULL	24
2	002	王子行	NULL	26

图 8.23　使用 IS NULL 查询员工信息

SQL 语句如下：

```
SELECT * FROM Employee
WHERE Sex IS NULL
```

【例 8.22】 在 Employee 表中查询 Sex 不为空的员工信息，SQL 语句及运行结果如图 8.24 所示。（实例位置：光盘\TM\sl\8\22）

SQL 语句如下：

```
SELECT * FROM Employee
WHERE Sex IS NOT NULL
```

6. IN 关键字

使用 IN 关键字来指定列表搜索的条件，确定指定的值是否与子查询或列表中的值相匹配。

语法如下：

```
test_expression [ NOT ] IN
        ( subquery | expression [ ,...n ]
        )
```

参数说明如下。

- ☑ test_expression：任何有效的表达式。
- ☑ subquery：包含某列结果集的子查询。该列必须与 test_expression 具有相同的数据类型。
- ☑ expression[,... n]：一个表达式列表，用来测试是否匹配。所有的表达式必须与 test_expression 具有相同的类型。
- ☑ 结果类型：Boolean 类型。

【例 8.23】 在 Employee 表中查询 ID 是 001、002 和 003 的员工信息，SQL 语句及运行结果如图 8.25 所示。（实例位置：光盘\TM\sl\8\23）

```
SELECT * FROM Employee
WHERE Sex IS NOT NULL
```

结果　消息

	ID	Name	Sex	Age
1	003	王开	女	25
2	004	赵小小	女	27
3	005	王飞飞	女	23
4	006	肖一子	男	24
5	007	王婷	女	22
6	008	王一	女	26
7	010	赵行	男	27
8	011	张子行	女	24
9	012	王雨涵	女	25

图 8.24　使用 IS NOT NULL 查询员工信息

```
SELECT *
FROM Employee
WHERE ID IN('001','002','003')
```

结果　消息

	ID	Name	Sex	Age
1	001	张子婷	女	24
2	002	王子行	男	26
3	003	王开	女	25

图 8.25　使用 IN 查询员工信息

SQL 语句如下：

```
SELECT *  FROM Employee
WHERE ID IN('001','002','003')
```

【例 8.24】 在 Employee 表中查询 ID 不是 001、002 和 003 的员工信息，SQL 语句及运行结果如图 8.26 所示。（实例位置：光盘\TM\sl\8\24）

SQL 语句如下：

```
SELECT * FROM Employee
WHERE ID NOT IN('001','002','003')
```

7. ALL、SOME、ANY 关键字

ALL：比较标量值和单列集中的值。与比较运算符和子查询一起使用。>ALL 表示大于条件的每一个值，换句话说，就是大于最大值。

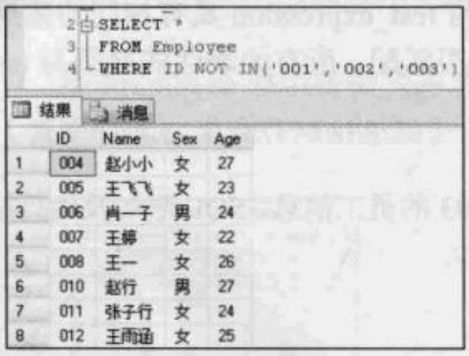
语法如下：

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

参数说明如下。

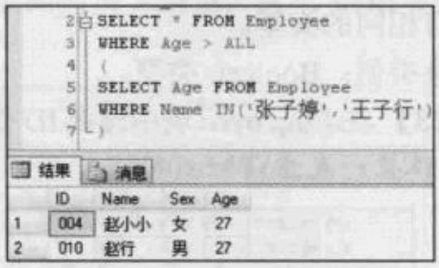
- ☑ scalar_expression：任何有效的表达式。
- ☑ {=|<>|!=|>|>=|!>|<|<=|!<}：比较运算符。
- ☑ subquery：返回单列结果集的子查询。返回列的数据类型必须与 scalar_expression 的数据类型相同。
- ☑ 结果类型：Boolean。

【例 8.25】 在 Employee 表中查询 Age 大于张子婷和王子行的员工信息，SQL 语句及运行结果如图 8.27 所示。（**实例位置：光盘\TM\sl\8\25**）



	ID	Name	Sex	Age
1	004	赵小小	女	27
2	005	王飞飞	女	23
3	006	肖一子	男	24
4	007	王婷	女	22
5	008	王一	女	26
6	010	赵行	男	27
7	011	张子行	女	24
8	012	王雨涵	女	25

图 8.26 使用 NOT IN 查询员工信息



	ID	Name	Sex	Age
1	004	赵小小	女	27
2	010	赵行	男	27

图 8.27 使用 ALL 查询员工信息

SQL 语句如下：

```
SELECT * FROM Employee
WHERE Age > ALL
(
SELECT Age FROM Employee
WHERE Name IN('张子婷','王子行')
)
```

说明

在本例中的 SELECT 语句中又包含一个 SELECT 语句，这种查询属于嵌套查询，在语句 SELECT Age FROM Employee WHERE Name IN('张子婷','王子行')中查询的是张子婷和王子行的 Age。语句 Age > ALL 就是大于张子婷和王子行 Age 的最大值。

SOME | ANY：比较标量值和单列集中的值。SOME 和 ANY 是等效的。与比较运算符和子查询一起使用。>ANY 表示至少大于条件的一个值，换句话说，就是大于最小值。

语法如下：

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

参数说明如下。

☑ scalar_expression：任何有效的表达式。

☑ { = | <> | != | > | >= | !> | < | <= | !< }：任何有效的比较运算符。

☑ SOME | ANY：指定应进行比较。

☑ subquery：包含某列结果集的子查询。所返回列的数据类型必须是与 scalar_expression 相同的数据类型。

☑ 结果类型：Boolean 类型。

【例 8.26】 在 Employee 表中查询 Age 大于张子婷和王子行的任意员工信息，SQL 语句及运行结果如图 8.28 所示。（**实例位置：光盘\TM\sl\8\26**）

SQL 语句如下：

```
SELECT * FROM Employee
WHERE Age > ALL
(
SELECT Age FROM Employee
WHERE Name IN('张子婷','王子行')
)
```

8. EXISTS 关键字

EXISTS：指定一个子查询，测试行是否存在。

语法如下：

```
EXISTS subquery
```

参数说明如下。

☑ subquery：受限制的 SELECT 语句。不允许使用 COMPUTE 子句和 INTO 关键字。

☑ 结果类型：Boolean 类型。

【例 8.27】 在子查询中指定了结果集为 NULL，并且使用 EXISTS 求值，此时值仍然为 TRUE，SQL 语句及运行结果如图 8.29 所示。（**实例位置：光盘\TM\sl\8\27**）

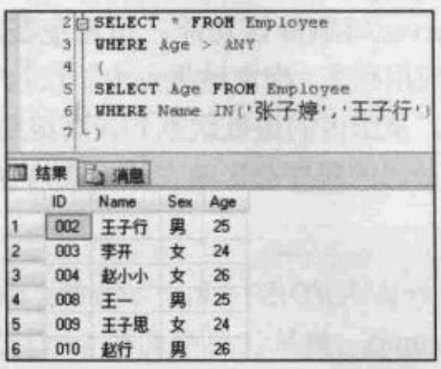


图 8.28　使用 ANY 查询员工信息

图 8.29　使用 EXISTS 关键字查询

SQL 语句如下：

```
SELECT ID, Name   FROM Employee
WHERE EXISTS (SELECT NULL)
```

8.1.6 GROUP BY 子句

GROUP BY 表示按一个或多个列或表达式的值将一组选定行组合成一个摘要行集。针对每一组返回一行。

语法如下：

```
[ GROUP BY [ ALL ] group_by_expression[ ,...n ]
                [ WITH { CUBE | ROLLUP } ] ]
```

参数说明如下。

- ☑ ALL：后续版本的 Microsoft SQL Server 将删除该功能。请避免在新的开发工作中使用该功能，并着手修改当前还在使用该功能的应用程序。包含所有组和结果集，甚至包含那些其中任何行都不满足 WHERE 子句指定的搜索条件的组和结果集。如果指定了 ALL，将对组中不满足搜索条件的汇总列返回空值。不能用 CUBE 或 ROLLUP 运算符指定 ALL。
- ☑ group_by_expression：针对其执行分组操作的表达式。group_by_expression 也称为分组列。group_by_expression 可以是列，也可以是引用由 FROM 子句返回的列的非聚合表达式。不能使用在 SELECT 列表中定义的列别名来指定组合列。

注意

不能在 group_by_expression 中使用类型为 text、ntext 和 image 的列。

- ☑ WITH CUBE：后续版本的 Microsoft SQL Server 将删除该功能。请避免在新的开发工作中使用该功能，并着手修改当前还在使用该功能的应用程序。指定结果集内不仅包含由 GROUP BY 提供的行，同时还包含汇总行。GROUP BY 汇总行针对每个可能的组和子组组合在结果集内返回。使用 GROUPING 函数可确定结果集内的空值是否为 GROUP BY 汇总值。结果集内的汇总行数取决于 GROUP BY 子句内包含的列数。由于 CUBE 返回每个可能的组和子组组合，因此不论在列分组时指定使用什么顺序，行数都相同。
- ☑ WITH ROLLUP：后续版本的 Microsoft SQL Server 将删除该功能。请避免在新的开发工作中使用该功能，并着手修改当前还在使用该功能的应用程序。指定结果集内不仅包含由 GROUP BY 提供的行，同时还包含汇总行。按层次结构顺序，从组内的最低级别到最高级别汇总组。组的层次结构取决于列分组时指定使用的顺序。更改列分组的顺序会影响在结果集内生成的行数。

注意

使用 CUBE 或 ROLLUP 时，不支持非重复聚合，如 AVG(DISTINCT column_name)、COUNT (DISTINCT column_name)和 SUM(DISTINCT column_name)。如果使用此类聚合，则 SQL Server 数据库引擎将返回错误消息并取消查询。

【例 8.28】 将 Employee 表中的员工信息按性别进行分组，SQL 语句及运行结果如图 8.30 所示。（**实例位置：光盘\TM\sl\8\28**）

SQL 语句如下：

```
SELECT Sex 性别 FROM Employee
GROUP BY Sex
```

注意

SELECT 子句必须包括在聚合函数或 GROUP BY 子句中。

例如，由于下列查询中 Name 列既不包含在 GROUP BY 子句中，也不包含在聚合函数中，所以是错误的。错误的 SQL 语句及错误提示如图 8.31 所示。

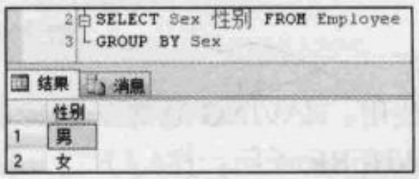


图 8.30　将 Employee 表按性别分组

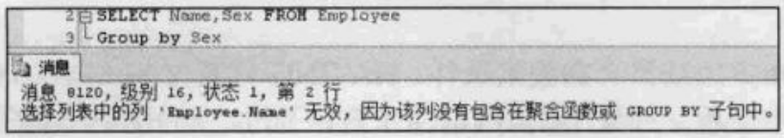


图 8.31　错误的 SQL 语句及提示

【例 8.29】 将 Employee 表中的员工信息按年龄进行分组，并统计每个年龄段的人数，SQL 语句及运行结果如图 8.32 所示。（**实例位置：光盘\TM\sl\8\29**）

SQL 语句如下：

```
SELECT Age 年龄,Count(Age)人数 FROM Employee
GROUP BY Age
```

【例 8.30】 将 Employee 表中的员工信息按性别和年龄进行分组，SQL 语句及运行结果如图 8.33 所示。（**实例位置：光盘\TM\sl\8\30**）

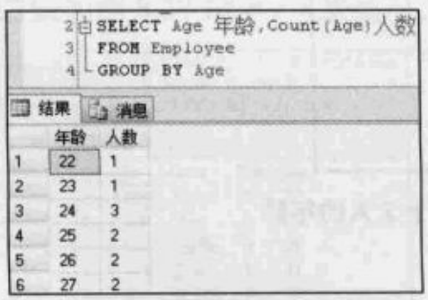


图 8.32　将 Employee 表按年龄分组

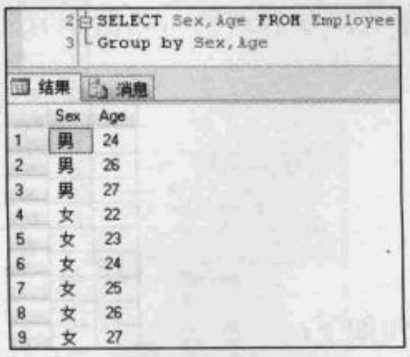


图 8.33　将 Employee 表按性别、年龄分组

SQL 语句如下：

```
SELECT Sex,Age FROM Employee
Group by Sex,Age
```

【例 8.31】 在 Employee 表中求女员工的平均年龄，SQL 语句及运行结果如图 8.34 所示。（**实例位置：光盘\TM\sl\8\31**）

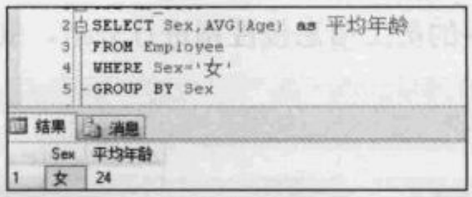


图 8.34　求 Employee 表中女员工的平均年龄

SQL 语句如下：

```
SELECT Sex,AVG(Age) as 平均年龄 FROM Employee
WHERE Sex='女'
GROUP BY Sex
```

8.1.7　HAVING 子句

指定组或聚合的搜索条件。HAVING 只能与 SELECT 语句一起使用。HAVING 通常在 GROUP BY 子句中使用。如果不使用 GROUP BY 子句，则 HAVING 的行为与 WHERE 子句一样。

语法如下：

```
[ HAVING <search condition> ]
```

参数<search_condition>用来指定组或聚合应满足的搜索条件。

注意

在 HAVING 子句中不能使用 text、image 和 ntext 数据类型。

【例 8.32】 在 Employee 表中查询每个年龄段的人数大于等于 2 人的年龄，SQL 语句及运行结果如图 8.35 所示。（**实例位置：光盘\TM\sl\8\32**）

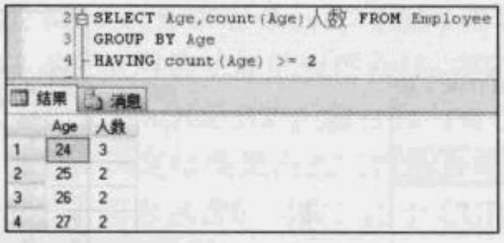


图 8.35　每个年龄段的人数大于等于 2 人的年龄

SQL 语句如下：

```
SELECT Age,count(Age)人数 FROM Employee
GROUP BY Age
HAVING count(Age) >= 2
```

8.1.8　ORDER BY 子句

指定在 SELECT 语句返回的列中所使用的排序顺序。除非同时指定了 TOP，否则 ORDER BY 子

句在视图、内联函数、派生表和子查询中无效。

```
[ ORDER BY
    {
    order_by_expression
  [ COLLATE collation_name ]
  [ ASC | DESC ]
    } [ ,...n ]
]
```

参数说明如下。

☑ order_by_expression：指定要排序的列。可以将排序列指定为一个名称或列别名。可指定多个排序列。

注意

ntext、text、image 或 xml 列不能用于 ORDER BY 子句。

☑ COLLATE {collation_name}：指定根据 collation_name 中指定的排序规则，而不是表或视图中所定义的列的排序规则，应执行的 ORDER BY 操作。

☑ ASC：指定按升序，从最低值到最高值对指定列中的值进行排序。

☑ DESC：指定按降序，从最高值到最低值对指定列中的值进行排序。

【例 8.33】 在 Employee 表中查询女员工的详细信息，并按年龄的降序排列，SQL 语句及运行结果如图 8.36 所示。（**实例位置：光盘\TM\sl\8\33**）

SQL 语句如下：

```
SELECT * FROM Employee
WHERE Sex='Ů'
ORDER BY Age DESC
```

【例 8.34】 在 Employee 表中查询 Age 大于 24 的员工的详细信息，并按姓名的升序排列，SQL 语句及运行结果如图 8.37 所示。（**实例位置：光盘\TM\sl\8\34**）

```
SELECT * FROM Employee
WHERE Sex='女'
ORDER BY Age DESC
```

结果　消息

	ID	Name	Sex	Age
1	004	赵小小	女	27
2	008	王一	女	26
3	012	王雨涵	女	25
4	003	王开	女	25
5	001	张子婷	女	24
6	011	张子行	女	24
7	005	王飞飞	女	23
8	007	王婷	女	22

图 8.36　按年龄的降序排列

```
SELECT * FROM Employee
WHERE Age > 24
ORDER BY Name ASC
```

结果　消息

	ID	Name	Sex	Age
1	003	王开	女	25
2	008	王一	女	26
3	012	王雨涵	女	25
4	002	王子行	男	26
5	004	赵小小	女	27
6	010	赵行	男	27

图 8.37　按姓名的升序排列

SQL 语句如下：

```
SELECT * FROM Employee
WHERE Age > 24
ORDER BY Name ASC
```

8.1.9 COMPUTE 子句

生成合计作为附加的汇总列出现在结果集的最后。当与 BY 一起使用时，COMPUTE 子句在结果集内生成控制中断和小计。可在同一查询内指定 COMPUTE BY 和 COMPUTE。

语法如下：

```
[ COMPUTE
    { { AVG | COUNT | MAX | MIN | STDEV | STDEVP | VAR | VARP | SUM }
    ( expression ) } [ ,...n ]
    [ BY expression [ ,...n ] ]
]
```

参数说明如下。

☑ AVG | COUNT | MAX | MIN | STDEV | STDEVP | VAR | VARP | SUM：指定要执行的聚合。在 COMPUTE 子句中可以使用的行聚合函数如表 8.8 所示。

表 8.8 COMPUTE 子句可以使用的行聚合函数

行聚合函数	描　述	行聚合函数	描　述
AVG	数值表达式中所有值的平均值	STDEVP	表达式中所有值的总体标准偏差
COUNT	选定的行数	SUM	数值表达式中所有值的和
MAX	表达式中的最高值	VAR	表达式中所有值的方差
MIN	表达式中的最低值	VARP	表达式中所有值的总体方差
STDEV	表达式中所有值的标准偏差		

☑ expression：表达式（Transact-SQL），如对其执行计算的列名。expression 必须出现在选择列表中，并且必须被指定为与选择列表中的某个表达式相同。不能在 expression 中使用选择列表中所指定的列别名。

☑ BY expression：在结果集中生成控制中断和小计。expression 是关联 ORDER BY 子句中 order_by_expression 的相同副本。通常，这是列名或列别名。可以指定多个表达式。在 BY 之后列出多个表达式将把组划分为子组，并在每个组级别应用聚合函数。

说明

没有等价于 COUNT(*)的函数。若要查找由 GROUP BY 和 COUNT(*)生成的汇总信息，请使用不带 BY 的 COMPUTE 子句。这些函数忽略空值。

注意

如果是用 COMPUTE 子句指定的行聚合函数，则不允许它们使用 DISTINCT 关键字。

【例 8.35】 在 Employee 表中求 Age 字段的平均值，SQL 语句及运行结果如图 8.38 所示。（**实例位置：光盘\TM\sl\8\35**）

SQL 语句如下：

```
select *  from Employee
order by  Sex
compute  avg(Age)
```

说明

在 COMPUTE 或 COMPUTE BY 子句中，不能指定 ntext、text 和 image 数据类型。

下面为 COMPUTE 和 COMPUTE BY 两个子句的区别。

（1）没有 BY 时，查询结果将包含两个结果集。第一个结果集将是包含选择列表中所有字段的详细记录。第二个结果集只有一条记录，这条记录只包含 COMPUTE 子句中所指定的汇总函数的合计。

（2）有 BY 时，查询结果将根据 BY 后的字段名称进行分组，并且为每个符合 SELECT 语句查询条件的组返回两个结果集。第一个结果集是详细记录集，包含结果集中将包含选择列表中所有的字段信息。第二个结果集是只包含一条记录，这条记录的内容只有该组的 COMPUTE 子句中所指定的汇总函数的小计。

【例 8.36】 在 Employee 表中分别求男员工和女员工 Age 字段的平均值，SQL 语句及运行结果如图 8.39 所示。（**实例位置：光盘\TM\sl\8\36**）

```
2  select *
3  from Employee
4  order by  Sex
5  compute  avg(Age)
```

结果　消息

	ID	Name	Sex	Age
1	002	王子行	男	26
2	006	肖一子	男	24
3	010	赵行	男	27
4	011	张子行	女	24
5	012	王雨诵	女	25
6	007	王婷	女	22
7	008	王一	女	26
8	003	王开	女	25
9	004	赵小小	女	27
10	005	王飞飞	女	23
11	001	张子婷	女	24

	avg
1	24

图 8.38　求 Employee 表中年龄的平均值

```
2  select *
3  from Employee
4  order by  Sex
5  compute  avg(Age)  by Sex
```

结果　消息

	ID	Name	Sex	Age
1	002	王子行	男	26
2	006	肖一子	男	24
3	010	赵行	男	27

	avg
1	25

	ID	Name	Sex	Age
1	011	张子行	女	24
2	012	王雨诵	女	25
3	007	王婷	女	22
4	008	王一	女	26
5	003	王开	女	25
6	004	赵小小	女	27
7	005	王飞飞	女	23
8	001	张子婷	女	24

	avg
1	24

图 8.39　分别求 Employee 表中男员工和女员工年龄的平均值

SQL 语句如下：

```
select *  from Employee
order by  Sex
compute  avg(Age)  by Sex
```

8.1.10　DISTINCT 关键字

DISTINCT 关键字主要用来从 SELECT 语句的结果集中去掉重复的记录。如果用户没有指定

DISTINCT 关键字，那么系统将返回所有符合条件的记录组成结果集，其中包括重复的记录。

【例 8.37】 查询 Employee 表中 Age 列的信息，并去掉重复值，SQL 语句及运行结果如图 8.40 所示。（**实例位置：光盘\TM\sl\8\37**）

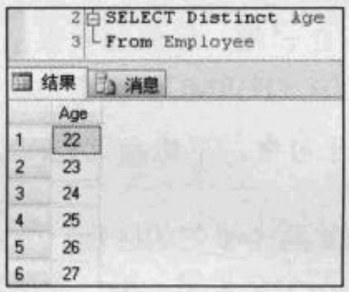


图 8.40 去掉 Age 列的重复值

SQL 语句如下：

```
SELECT Distinct Age From Employee
```

8.1.11 TOP 关键字

TOP 关键字可以限制查询结果显示的行数，不仅可以列出结果集中的前几行，还可以列出结果集中的后几行。

TOP 关键字的语法如下：

```
SELECT TOP n [PERCENT]
FROM table
WHERE
ORDER BY...
```

参数说明如下。

☑ [PERCENT]：返回行的百分之 n，而不是 n 行。

☑ n：如果 SELECT 语句中没有 ORDER BY 子句，TOP n 返回满足 WHERE 子句的前 n 条记录。如果子句中满足条件的记录少于 n，那么仅返回这些记录。

【例 8.38】 查询 Employee 表中前 5 名员工的所有信息，SQL 语句及运行结果如图 8.41 所示。（**实例位置：光盘\TM\sl\8\38**）

SQL 语句如下：

```
SELECT Top 5 * From Employee
```

【例 8.39】 查询 Employee 表中 Name、Sex、Age 列的前 3 条信息，SQL 语句及运行结果如图 8.42 所示。（**实例位置：光盘\TM\sl\8\39**）

SQL 语句如下：

```
SELECT Top 3 Name,Sex,Age From Employee
```

```
SELECT Top 5 *
From Employee
```

结果　消息

	ID	Name	Sex	Age
1	001	张子婷	女	24
2	002	王子行	男	26
3	003	王开	女	25
4	004	赵小小	女	27
5	005	王飞飞	女	23

图 8.41　查询 Employee 表中前 5 条记录

```
SELECT Top 3 Name,Sex,Age
From Employee
```

结果　消息

	Name	Sex	Age
1	张子婷	女	24
2	王子行	男	26
3	王开	女	25

图 8.42　查询 Employee 表中前 3 条记录

8.2　UNION 合并多个查询结果

视频讲解：光盘\TM\lx\8\UNION 合并多个查询结果.mp4

表的合并操作将两个表的行合并到了一个表中，且不需要对这些行做任何更改。

在构造合并查询时必须遵循以下几条规则：

（1）两个 SELECT 语句选择列表中的列数目必须一样多，而且对应位置上的列的数据类型必须相同或者兼容。

（2）列的名字或者别名是由第一个 SELECT 语句的选择列表决定的。

（3）可以为每个 SELECT 语句都增加一个表示行的数据来源的表达式。

（4）可以将合并操作作为 SELECT INTO 命令的一部分使用，但是 INTO 关键字必须放在第一个 SELECT 语句中。

（5）虽然 SELECT 命令在默认情况下不会去掉重复行，除非明确地为它指定 DISTINCT 关键字，但是，合并操作却与之相反。在默认情况下，合并操作将会去掉重复的行；如果希望返回重复的行，就必须明确地指定 All 关键字。

（6）用对所有 SELECT 语句的合并操作结果进行排序的 ORDER BY 子句，必须放到最后一个 SELECT 后面，但它所使用的排序列名必须是第一个 SELECT 选择列表中的列名。

8.2.1　UNION 与联接之间的区别

合并操作与联接相似，因为它们都是将两个表合并起来形成另一个表的方法。然而，它们的合并方法有本质上的不同，结果表的形状如图 8.43 所示。

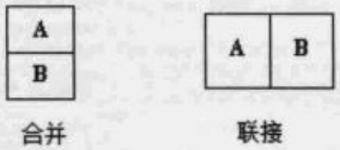


图 8.43　合并和联接具有不同的结构

注：A 和 B 分别代表两个数据源表。

它们具体的不同如下：

（1）在合并中，两个表源列的数量与数据类型必须相同；在联接中，一个表的行可能与另一个表的行有很大区别，结果表的列可能来自第一个表、第二个表或两个表的都有。

（2）在合并中，行的最大数量是两个表行的“和”；在联接中，行的最大数量是它们的“乘积”。

【例 8.40】 把“select Cno,Cname from Course”和“select Sname,Sex from Student”的查询结果合并。SQL 语句及运行结果如图 8.44 所示。（**实例位置：光盘\TM\sl\8\40**）

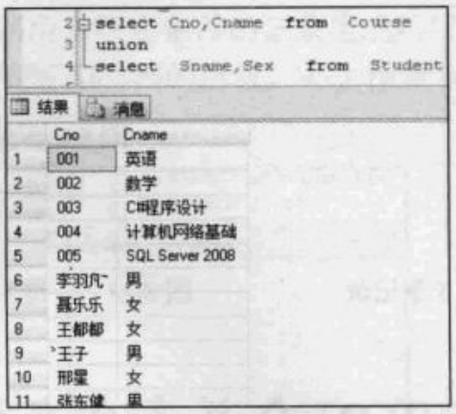


图 8.44　简单的合并查询

SQL 语句如下：

```
select Cno,Cname   from   Course
union
select   Sname,Sex    from   Student
```

8.2.2　使用 UNION ALL 合并表

UNION 加上关键字 ALL，功能是不删除重复行也不对行进行自动排序。加上 ALL 关键字需要的计算资源少，所以应尽可能使用它，尤其是处理大型表的时候。下列情况是应该使用 UNION ALL 的情况。

（1）知道有重复行并想保留这些行。

（2）知道不可能有任何重复的行。

（3）不在乎是否有任何重复的行。

【例 8.41】 用 UNION ALL 把“SELECT * FROM Student WHERE Sage > 20”和“SELECT * FROM Student WHERE Sex ='男'”的查询结果合并。SQL 语句及运行结果如图 8.45 所示。（**实例位置：光盘\TM\sl\8\41**）

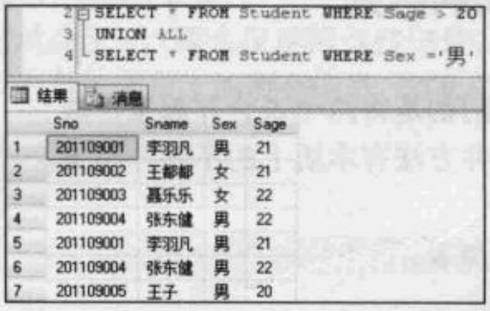


图 8.45　用 UNION ALL 合并查询

SQL 语句如下：

```
SELECT * FROM Student WHERE Sage > 20
UNION ALL
SELECT * FROM Student WHERE Sex ='男'
```

8.2.3　UNION 中的 ORDER BY 子句

合并表时有且只能有一个 ORDER BY 子句，并且必须将它放置在语句的末尾。它在两个 SELECT 语句中都提供了用于合并所有行的排序。下面列出 ORDER BY 子句可以使用的排序依据。

（1）来自第一个 SELECT 子句的别名。

（2）来自第一个 SELECT 子句的列别名。

（3）UNION 中列的位置的编号。

说明

前两种排序依据更常用、更容易理解。

【例 8.42】 把“SELECT Sname,Sage FROM Student WHERE Sex = '男'”和“SELECT Cname,Credit FROM Course ORDER BY Sage ASC”的查询结果合并。SQL 语句及运行结果如图 8.46 所示。（**实例位置：光盘\TM\sl\8\42**）

```
SELECT Sname,Sage FROM Student
WHERE Sex = '男'
UNION ALL
SELECT Cname,Credit FROM Course
ORDER BY Sage ASC
```

结果

	Sname	Sage
1	数学	3
2	计算机网络基础	4
3	SQL Server 2008	4
4	C#程序设计	5
5	英语	5
6	王子	20
7	李羽凡	21
8	张东健	22

图 8.46　合并查询结果

SQL 语句如下：

```
SELECT Sname,Sage FROM Student
WHERE Sex = '男'
UNION ALL
SELECT Cname,Credit FROM Course
ORDER BY Sage ASC
```

8.2.4　UNION 中的自动数据类型转换

合并表时，两个表源中对应的每个列数据类型必须相同吗？不，只要是数据类型兼容就可以。

首先说文本数据类型。假设合并的两个表源中第一列数据类型虽然都是文本类型，但长度不一致。当合并表时，字符长度短的列等于字符长度长的列的长度，这样长度长的列不会丢失任何数据。

其次说数值类型。当合并的两个表源中第一列数据类型虽都是数值类型，但长度不同，合并表时，所有数字保持所有数字都允许的长度来消除它们数据类型的差别。

因为这种都是自动数据类型转换，所以说任何两个文本列都是兼容的，任何两个数字列也都是兼容的。

【例 8.43】 把“SELECT Sno,Sage FROM Student”和“SELECT Cno,Grade FROM Sc”的查询结果合并。其中，Sage 列的数据类型是整型，Grade 列的数据类型是单精度浮点型。SQL 语句及运行结果如图 8.47 所示。（**实例位置：光盘\TM\sl\8\43**）

```
SELECT Sno,Sage FROM Student
UNION ALL
SELECT Cno,Grade FROM Sc
```

结果

	Sno	Sage
1	201109001	21
2	201109002	21
3	201109003	22
4	201109004	22
5	201109005	20
6	201109006	20
7	001	85
8	002	83
9	003	52
10	004	64
11	005	75

图 8.47　合并不同数据类型的表

SQL 语句如下：

```
SELECT Sno,Sage FROM Student
UNION ALL
SELECT Cno,Grade FROM Sc
```

8.2.5 使用 UNION 合并不同类型的数据

当合并表时，两个表源中相对应的列即使数据类型不一致也能合并，这时需要借助数据类型转换函数。

当合并的两个表源相对应的列数据类型不一致，例如，一个是数值型，另一个是字符型，如果数值型的被转换成文本类型，完全可以合并两个表。

【例 8.44】 把“SELECT Sname,Sex FROM Student”和“SELECT Cname,str(Credit) FROM Course”的查询结果合并，并把整型的 Grade 转换成字符类型。SQL 语句及运行结果如图 8.48 所示。（**实例位置：光盘\TM\sl\8\44**）

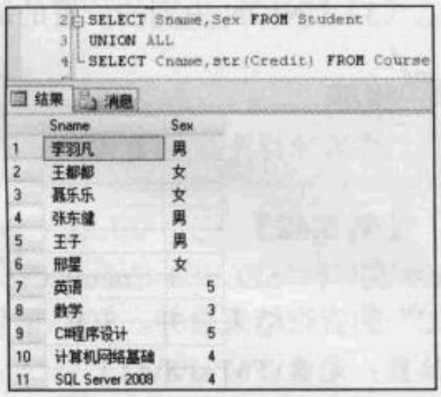


图 8.48 合并不同数据类型

SQL 语句如下：

```
SELECT Sname,Sex FROM Student
UNION ALL
SELECT Cname,str(Credit) FROM Course
```

上面的代码中，STR 函数是返回由数字数据转换来的字符数据。

语法如下：

```
STR( float_expression [ , length [ , decimal ] ] )
```

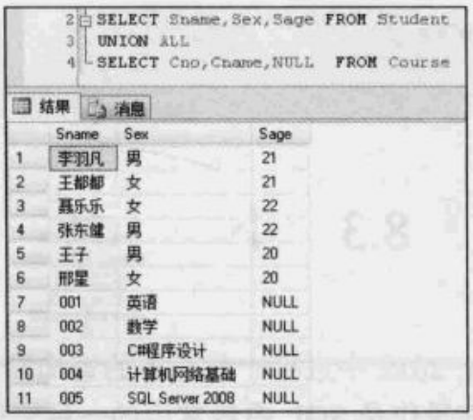
参数说明如下。

- ☑ float_expression：带小数点的近似数字（float）数据类型的表达式。
- ☑ length：总长度。它包括小数点、符号、数字以及空格。默认值为 10。
- ☑ decimal：小数点后的位数。decimal 必须小于或等于 16。如果 decimal 大于 16，则会截断结果，使其保持为小数点后具有 16 位。

8.2.6 使用 UNION 合并有不同列数的两个表

当合并两个表源时列数不同，只要向其中一个表源中添加列，就可以使两表源的列数相同，这时即可合并列了。

【例 8.45】 把“SELECT Sname,Sex,Sage FROM Student”和“SELECT Cno,Cname,NULL FROM Course”的查询结果合并，并用 NULL 值添加到 Course 表，SQL 语句及运行结果如图 8.49 所示。（**实例位置：光盘\TM\sl\8\45**）



```
SELECT Sname,Sex,Sage FROM Student
 UNION ALL
SELECT Cno,Cname,NULL  FROM Course
```

结果　消息

	Sname	Sex	Sage
1	李羽凡	男	21
2	王都都	女	21
3	聂乐乐	女	22
4	张东健	男	22
5	王子	男	20
6	邢星	女	20
7	001	英语	NULL
8	002	数学	NULL
9	003	C#程序设计	NULL
10	004	计算机网络基础	NULL
11	005	SQL Server 2008	NULL

图 8.49　合并不同列数的两个表

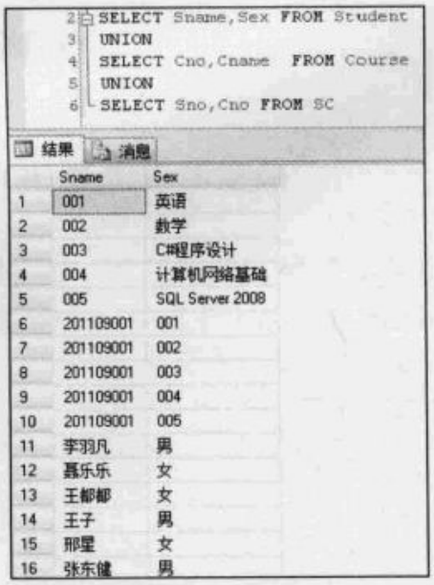
SQL 语句如下：

```
SELECT Sname,Sex,Sage FROM Student
UNION ALL
SELECT Cno,Cname,NULL   FROM Course
```

8.2.7　使用 UNION 进行多表合并

可以把很多数量的表进行合并，表的数量可达十多个。但仍要遵循合并表时的规则。

【例 8.46】 合并 Student、Course 和 SC 这 3 张表，从表 Student 中查询 Sname、Sex，从表 Course 中查询 Cno、Cname，从 SC 表中查询 Sno、Cno。并把这 3 张表的查询结果合并，SQL 语句及运行结果如图 8.50 所示。（**实例位置：光盘\TM\sl\8\46**）



```
SELECT Sname,Sex FROM Student
 UNION
SELECT Cno,Cname  FROM Course
 UNION
SELECT Sno,Cno FROM SC
```

结果　消息

	Sname	Sex
1	001	英语
2	002	数学
3	003	C#程序设计
4	004	计算机网络基础
5	005	SQL Server 2008
6	201109001	001
7	201109001	002
8	201109001	003
9	201109001	004
10	201109001	005
11	李羽凡	男
12	聂乐乐	女
13	王都都	女
14	王子	男
15	邢星	女
16	张东健	男

图 8.50　多表合并

SQL 语句如下：

```
SELECT Sname,Sex FROM Student
UNION
```

```
SELECT Cno,Cname   FROM Course
UNION
SELECT Sno,Cno FROM SC
```

8.3 小　　结

本章介绍了如何在 SQL Server 2012 中进行一些基本的查询操作，包括 SELECT 基本查询语句、UNION 合并查询结果等内容。查询操作是 SQL 中最常用的一种操作，因此，读者在学习本章内容时，一定要熟练掌握。

8.4 实践与练习

1．在商品销售信息表（tb_xsb）中查询商品利润大于 300 元的商品名称信息。（**答案位置：光盘\TM\sl\8\47**）

2．在明日科技图书信息表（mrbooks）中查询图书价格在 68～88 之间的图书信息。（**答案位置：光盘\TM\sl\8\48**）

3．在学生成绩表（tb_StuScore）中查询数学成绩大于等于 95 分或者音乐成绩大于 90 分同时还得满足英语成绩大于等于 90 分的学生信息。（**答案位置：光盘\TM\sl\8\49**）

第 9 章

SQL 数据高级查询

（视频讲解：26 分钟）

数据查询是 SQL 中最常用的一种操作，本章将对 SQL 数据查询中比较高级的查询进行讲解，主要包括子查询、多种形式的嵌套查询、各种联接查询及如何使用 CASE 函数进行查询。

通过阅读本章，您可以：

- 熟悉子查询及嵌套查询的基本概念
- 掌握如何进行嵌套查询
- 掌握联接查询的使用
- 熟悉 CASE 函数在 SQL 查询中的使用

9.1 子查询与嵌套查询

视频讲解：光盘\TM\lx\9\子查询与嵌套查询.mp4

在使用 SELECT 语句检索数据时，可以使用 WHERE 子句指定用于限制返回的行的搜索条件，GROUP BY 子句将结果集分成组，ORDER BY 子句定义结果集中的行排列的顺序。使用这些子句可以方便地查询表中的数据。但是，当由 WHERE 子句指定的搜索条件指向另一张表时，就需要使用子查询或嵌套查询。在本节中将详细介绍什么是子查询和嵌套查询。

9.1.1 子查询概述

子查询是一个嵌套在 SELECT、INSERT、UPDATE 或 DELETE 语句或其他子查询中的查询。任何允许使用表达式的地方都可以使用子查询。

☑ 子查询语法

```
SELECT [ALL | DISTINCT]<select item list>
FROM <table list>
[WHERE<search condition>]
[GROUP BY <group item list>
[HAVING <group by search conditoon>]])
```

☑ 语法规则

（1）子查询的 SELECT 查询总使用圆括号括起来。

（2）不能包括 COMPUTE 或 FOR BROWSE 子句。

（3）如果同时指定 TOP 子句，则可能只包括 ORDER BY 子句。

（4）子查询最多可以嵌套 32 层，个别查询可能会不支持 32 层嵌套。

（5）任何可以使用表达式的地方都可以使用子查询，只要它返回的是单个值。

（6）如果某个表只出现在子查询中而不出现在外部查询中，那么该表中的列就无法包含在输出中。

☑ 语法格式

```
WHERE  查询表达式  [NOT]  IN(子查询)
WHERE  查询表达式  比较运算符  [ ANY | ALL ](子查询)
WHERE  [NOT]  EXISTS(子查询)
```

9.1.2 嵌套查询概述

嵌套查询是指将一个查询块嵌套在另一个查询块的 WHERE 子句或 HAVING 短语的条件中的查询。

嵌套查询中上层的查询块称为外侧查询或父查询，下层查询块称为内层查询或子查询。SQL 允许多层嵌套，但是在子查询中不允许出现 ORDER BY 子句，ORDER BY 子句只能用在最外层的查询块中。

嵌套查询的处理方法是：先处理最内侧的子查询，然后一层一层向上处理，直到最外层的查询块。

9.1.3　简单的嵌套查询

嵌套查询中的内层子查询通常作为搜索条件的一部分呈现在 WHERE 或 HAVING 子句中。例如，把一个表达式的值和一个由子查询生成的一个值相比较，这个测试类似于简单比较测试。

子查询比较测试用到的运算符是：=、<>、<、>、<=、>=。子查询比较测试把一个表达式的值和由子查询产生的一个值进行比较，返回比较结果为 TRUE 的记录。

【例 9.1】 Student 表中存储的是学生的基本信息，SC 表中存储的是学生的成绩（Grade）信息，使用嵌套查询，查询在 Student 表中 Grade>90 分的学生信息，SQL 语句及运行结果如图 9.1 所示。（**实例位置：光盘\TM\sl\9\1**）

SQL 语句如下：

```
SELECT * FROM Student
WHERE Sno = (SELECT Sno FROM SC WHERE Grade > 90)
```

这里给出本节中用到的所有表中的信息，SC 表、Student 表、Course 表，如图 9.2 所示。

```
SELECT * FROM Student
WHERE Sno = (SELECT Sno FROM SC WHERE Grade > 90)
```

结果　消息

	Sno	Sname	Sex	Sage
1	201109001	李羽凡	男	21

图 9.1　查询成绩大于 90 分的学生信息

SC表

Sno	Cno	Grade
201109001	001	93
201109001	002	83
201109001	003	52
201109002	001	NULL
201109002	002	70
201109002	003	NULL
201109003	001	74
201109003	002	69
201109004	001	NULL
201109004	002	89
201109004	003	85
201109005	001	79
201109005	002	90
201109005	003	45

Student表

Sno	Sname	Sex	Sage
201109001	李羽凡	男	20
201109002	王都都	女	21
201109003	聂乐乐	女	22
201109004	张东健	男	23
201109005	王子	男	24
201109006	邢星	女	25

Course表

Cno	Cname	Credit
001	数据结构	5
002	计算机网络	3
003	C#程序设计	5
004	数学	4
005	SQL Server 2008	4

图 9.2　表 SC、表 Student 和 Course 表中的信息

9.1.4　带 IN 的嵌套查询

带 IN 的嵌套查询语法格式如下：

```
WHERE 查询表达式 IN(子查询)
```

一些嵌套内层的子查询会产生一个值，也有一些子查询会返回一列值，即子查询不能返回带几行和几列数据的表。原因在于子查询的结果必须适合外层查询的语句。当子查询产生一系列值时，适合用带 IN 的嵌套查询。

把查询表达式单个数据和由子查询产生的一系列的数值相比较，如果数值匹配一系列值中的一个，则返回 TRUE。

【例 9.2】 在 Student 表和 SC 表中，查询参加考试的学生信息，SQL 语句及运行结果如图 9.3 所

示。（实例位置：光盘\TM\sl\9\2）

```
SELECT * FROM Student
WHERE Sno IN (SELECT Sno FROM SC )
```

结果 消息

	Sno	Sname	Sex	Sage
1	201109001	李羽凡	男	21
2	201109002	王都都	女	21
3	201109003	聂乐乐	女	22
4	201109004	张东健	男	22
5	201109005	王子	男	20

图 9.3　参加考试的学生的信息

SQL 语句如下：

```
SELECT * FROM Student
WHERE Sno IN (SELECT Sno FROM SC )
```

9.1.5　带 NOT IN 的嵌套查询

NOT IN 的嵌套查询语法格式如下：

```
WHERE 查询表达式 NOT IN(子查询)
```

【例 9.3】 在 Course 表和 SC 表中，查询没有考试的课程信息，SQL 语句及运行结果如图 9.4 所示。（实例位置：光盘\TM\sl\9\3）

SQL 语句如下：

```
SELECT * FROM Course
WHERE Cno NOT IN
(SELECT CNO FROM SC WHERE Cno IS NOT NULL )
```

查询过程是用主查询中 Cno 的值与子查询结果中的值比较，不匹配返回真值。由于主查询中的“004”和“005”的课程代号值与子查询的结果的数据不匹配，返回真值。所以查询结果显示 Cno 为“004”和“005”的课程信息。

【例 9.4】 在 Student 表和 SC 表中，查询没有考试的学生信息，SQL 语句及运行结果如图 9.5 所示。（实例位置：光盘\TM\sl\9\4）

```
SELECT * FROM Course
WHERE Cno NOT IN
(SELECT CNO FROM SC WHERE Cno IS NOT NULL )
```

结果 消息

	Cno	Cname	Credit
1	004	数学	4
2	005	SQL Server 2008	4

图 9.4　没考试的课程信息

```
SELECT * FROM Student
WHERE Sno NOT IN
(SELECT Sno FROM SC WHERE Sno IS NOT NULL)
```

结果 消息

	Sno	Sname	Sex	Sage
1	201109006	邢星	女	20

图 9.5　没有参加考试的学生

SQL 语句如下：

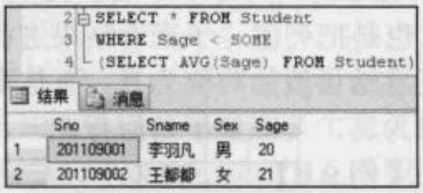
```
SELECT * FROM Student
WHERE Sno NOT IN
(SELECT Sno FROM SC WHERE Sno IS NOT NULL)
```

9.1.6　带 SOME 的嵌套查询

SQL 支持 3 种定量比较谓词：SOME、ANY 和 ALL。它们都是判断是否任何或全部返回值都满足搜索要求的。其中，SOME 和 ANY 谓词是存在量的，只注重是否有返回值满足搜索要求。这两种谓词含义相同，可以替换使用。

【例 9.5】 在 Student 表中，查询 Sage 小于平均年龄的所有学生的信息，SQL 语句及运行结果如图 9.6 所示。（**实例位置：光盘\TM\sl\9\5**）

```
SELECT * FROM Student
WHERE Sage < SOME
(SELECT AVG(Sage) FROM Student)
```



	Sno	Sname	Sex	Sage
1	201109001	李羽凡	男	20
2	201109002	王都都	女	21

图 9.6　查询年龄小于平均年龄的学生信息

SQL 语句如下：

9.1.7　带 ANY 的嵌套查询

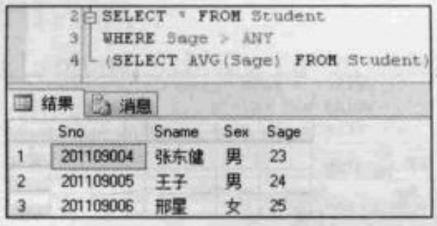
ANY 属于 SQL 支持的 3 种定量谓词之一，且和 SOME 完全等价，即能用 SOME 的地方完全可以使用 ANY。

【例 9.6】 在 Student 表中，查询 Sage 大于平均年龄的所有学生的信息，SQL 语句及运行结果如图 9.7 所示。（**实例位置：光盘\TM\sl\9\6**）

SQL 语句如下：

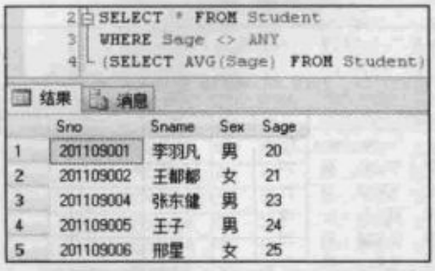
```
SELECT * FROM Student
WHERE Sage > ANY
(SELECT AVG(Sage) FROM Student)
```

【例 9.7】 在 Student 表中，查询 Sage 不等于平均年龄的所有学生的信息，SQL 语句及运行结果如图 9.8 所示（**实例位置：光盘\TM\sl\9\7**）



	Sno	Sname	Sex	Sage
1	201109004	张东健	男	23
2	201109005	王子	男	24
3	201109006	邢星	女	25

图 9.7　查询年龄大于平均年龄的学生信息



	Sno	Sname	Sex	Sage
1	201109001	李羽凡	男	20
2	201109002	王都都	女	21
3	201109004	张东健	男	23
4	201109005	王子	男	24
5	201109006	邢星	女	25

图 9.8　查询年龄不等于平均年龄的学生信息

SQL 语句如下：

```
SELECT * FROM Student
WHERE Sage <> ANY
(SELECT AVG(Sage) FROM Student)
```

9.1.8 带 ALL 的嵌套查询

ALL 谓词的使用方法和 ANY 或者 SOME 谓词一样，也是把列值与子查询结果进行比较，但是它不要求任意结果值的列值为真，而是要求所有列的查询结果都为真，否则就不返回行。

【例 9.8】 在 SC 表中，查询 Grade 没有大于 90 分的 Cno 的详细信息，SQL 语句及运行结果如图 9.9 所示。（**实例位置：光盘\TM\sl\9\8**）

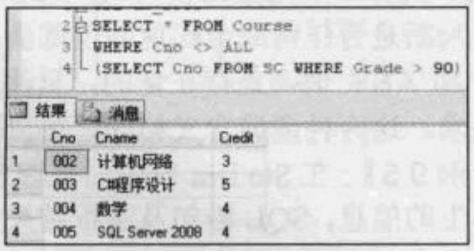


图 9.9 查询某课程成绩没有大于 90 分的课程信息

SQL 语句如下：

```
SELECT * FROM Course
WHERE Cno <> ALL
(SELECT Cno FROM SC WHERE Grade > 90)
```

9.1.9 带 EXISTS 的嵌套查询

EXISTS 谓词只注重子查询是否返回行。如果子查询返回一个或多个行，谓词返回为真值，否则为假。EXISTS 搜索条件并不真正地使用子查询的结果。它仅测试子查询是否产生任何结果。

用带 IN 的嵌套查询也可以用带 EXISTS 的嵌套查询改写。

【例 9.9】 在 Student 表中，查询参加考试的学生信息，SQL 语句及运行结果如图 9.10 所示。（**实例位置：光盘\TM\sl\9\9**）

SQL 语句如下：

```
SELECT * FROM Student
WHERE EXISTS
(SELECT Sno FROM SC WHERE Student.Sno = SC.Sno)
```

【例 9.10】 在 Student 表中，查询没有参加考试的学生信息，SQL 语句及运行结果如图 9.11 所示。（**实例位置：光盘\TM\sl\9\10**）

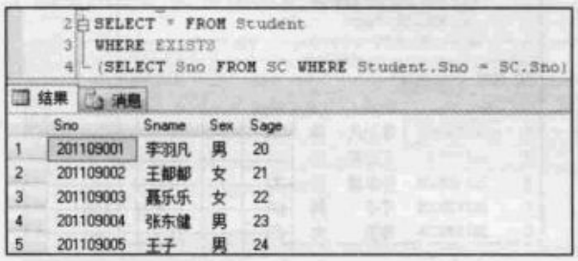


图 9.10 参加考试的学生信息

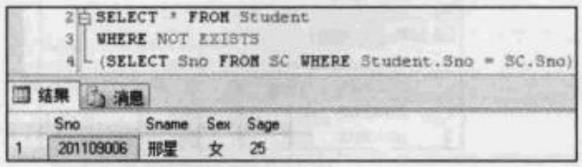


图 9.11 没有参加考试的学生信息

SQL 语句如下：

```
SELECT * FROM Student
WHERE NOT EXISTS
(SELECT Sno FROM SC WHERE Student.Sno = SC.Sno)
```

9.2　联 接 查 询

视频讲解：光盘\TM\lx\9\\联接查询.mp4

前面已经讲解过使用两个或两个以上的表进行查询。本节讲解的联接查询也是使用多个表进行查询。只不过联接查询是由一个笛卡儿乘积运算再加一个选取运算构成的查询。首先用笛卡儿乘积完成对两个数据集合的乘运算，然后对生成的结果集合进行选取运算，确保只把分别来自两个数据集合并且具有重叠部分的行合并在一起。联接的全部意义在于水平方向上合并两个数据集合，并产生一个新的结果集合。

联接条件可在 FROM 或 WHERE 子句中指定，建议在 FROM 子句中指定联接条件。WHERE 和 HAVING 子句还可以包含搜索条件，以进一步筛选根据联接条件选择的行。

联接可分为以下几类：内部联接、外部联接、交叉联接。

9.2.1　内部联接

内部联接是使用比较运算符比较要联接列中的值的联接。内联接也叫联接，是最早的一种联接，最早被称为普通联接或自然联接。内联接是从结果中删除其他被联接表中没有匹配行的所有行，所以内联接可能会丢失信息。

内部联接使用 JOIN 进行联接，具体语法如下：

```
SELECT fieldlist
FROM table1 [INNER] JOIN table2
ON table1.column=table2.column
```

参数说明如下。

- ☑ fieldlist：搜索条件。
- ☑ table1 [INNER] JOIN table2：将 table1 表与 table2 表进行内部联接。
- ☑ table1.column=table2.column：table1 表中与 table2 表中相同的列。

【例 9.11】 在 Student 表中，Sno 具有唯一值，而 SC 成绩表中的 Sno 有重复值。现在实现这两个表的内联接，SQL 语句及运行结果如图 9.12 所示。（**实例位置：光盘\TM\sl\9\11**）

SQL 语句如下：

```
SELECT * FROM SC
JOIN Student
ON Student.Sno = SC.Sno
```

```
SELECT * FROM SC
  JOIN Student
  ON Student.Sno = SC.Sno
```

结果　消息

	Sno	Cno	Grade	Sno	Sname	Sex	Sage
1	201109001	001	93	201109001	李羽凡	男	20
2	201109001	002	83	201109001	李羽凡	男	20
3	201109001	003	52	201109001	李羽凡	男	20
4	201109002	001	NULL	201109002	王都都	女	21
5	201109002	002	70	201109002	王都都	女	21
6	201109002	003	NULL	201109002	王都都	女	21
7	201109003	001	74	201109003	聂乐乐	女	22
8	201109003	002	69	201109003	聂乐乐	女	22
9	201109004	001	NULL	201109004	张东健	男	23
10	201109004	002	89	201109004	张东健	男	23
11	201109004	003	85	201109004	张东健	男	23
12	201109005	001	79	201109005	王子	男	24
13	201109005	002	90	201109005	王子	男	24
14	201109005	003	45	201109005	王子	男	24

图 9.12　内部联接

9.2.2 外部联接

外部联接则扩充了内联接的功能，会把内联接中删除表源中的一些保留下来，由于保留下来的行不同，可将外部联接分为左向外部联接、右向外部联接或完整外部联接。

1. 左向外联接

左向外部联接使用 LEFT JOIN 进行联接，左向外部联接的结果集包括 LEFT JOIN 子句中指定的左表的所有行，而不仅仅是联接列所匹配的行。如果左表的某一行在右表中没有匹配行，则在关联的结果集行中，来自右表的所有选择列表列均为空值。

左向外联接的语法如下：

```
SELECT fieldlist
FROM table1 left JOIN table2
ON table1.column=table2.column
```

参数说明如下。

- ☑ fieldlist：搜索条件。
- ☑ table1 [INNER] JOIN table2：将 table1 表与 table2 表进行外部联接。
- ☑ table1.column=table2.column：table1 表中与 table2 表中相同的列。

【例 9.12】 把 Student 表和 SC 表左外联接，第二个表 SC 有不满足联接条件的行，则用 NULL 表示，SQL 语句及运行结果如图 9.13 所示。（**实例位置：光盘\TM\sl\9\12**）

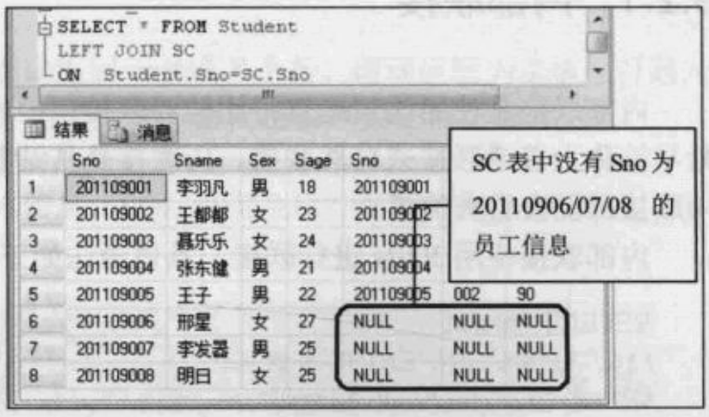


图 9.13 左向外联接

SQL 语句如下：

```
SELECT * FROM Student
LEFT JOIN SC
ON   Student.Sno=SC.Sno
```

2. 右向外联接

右向外部联接使用 RIGHT JOIN 进行联接，是左向外部联接的反向联接。将返回右表的所有行。如果右表的某一行在左表中没有匹配行，则将为左表返回空值。

右外联接的语法如下：

```
SELECT fieldlist
FROM table1 right JOIN table2
ON table1.column=table2.column
```

【例 9.13】 把 SC 表和 Course 表右外联接，第一个表 SC 有不满足联接条件的行，则用 NULL 表

示，SQL 语句及运行结果如图 9.14 所示。（**实例位置：光盘\TM\sl\9\13**）

SQL 语句如下：

```
SELECT * FROM SC
RIGHT JOIN Course
ON Course.Cno=SC.Cno
```

3. 完整外连接

完整外部联接使用 FULL JOIN 进行联接，将返回左表和右表中的所有行。当某一行在另一个表中没有匹配行时，另一个表的选择列表列将包含空值。如果表之间有匹配行，则整个结果集行包含基表的数据值。

完整外连接的语法如下：

```
SELECT fieldlist
FROM table1 full JOIN table2
ON table1.column=table2.column
```

【例 9.14】 把 SC 表和 Course 表完整外部联接，显示两个表中所有的行，SQL 语句及运行结果如图 9.15 所示。（**实例位置：光盘\TM\sl\9\14**）

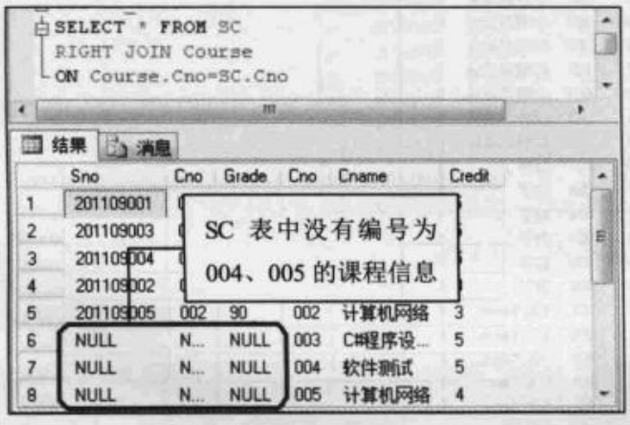


图 9.14　右向外联接

	Sno	Cno	Grade	Cno	Cname	Credit
1	201109001	001	93	001	数据结构	5
2	201109001	002	83	002	计算机网络	3
3	201109001	003	52	003	C#程序设计	5
4	201109002	001	NULL	001	数据结构	5
5	201109002	002	70	002	计算机网络	3
6	201109002	003	NULL	003	C#程序设计	5
7	201109003	001	74	001	数据结构	5
8	201109003	002	69	002	计算机网络	3
9	201109004	001	NULL	001	数据结构	5
10	201109004	002	89	002	计算机网络	3
11	201109004	003	85	003	C#程序设计	5
12	201109005	001	79	001	数据结构	5
13	201109005	002	90	002	计算机网络	3
14	201109005	003	45	003	C#程序设计	5
15	NULL	NULL	NULL	004	数学	4
16	NULL	NULL	NULL	005	SQL Server 2008	4

图 9.15　完整外部联接

SQL 语句如下：

```
SELECT * FROM SC
FULL JOIN Course
ON Course.Cno=SC.Cno
```

9.2.3　交叉联接

交叉联接使用 CROSS JOIN 进行联接，没有 WHERE 子句的交叉联接将产生联接所涉及的表的笛卡儿积。第一个表的行数乘以第二个表的行数等于笛卡儿积结果集的大小。

交叉连接中列和行的数量是这样计算的：

☑ 交叉连接中的列=原表中列的数量的总和（相加）。

☑ 交叉连接中的行=原表中的行数的积（相乘）。

交叉连接的语法如下：

```
SELECT fieldlist
FROM table1
cross JOIN table2
```

其中，忽略 on 方法来创建交叉连接。

【例 9.15】 把 Student 表和 Course 表进行交叉联接，SQL 语句及运行结果如图 9.16 所示。（**实例位置：光盘\TM\sl\9\15**）

```
SELECT * FROM Student
CROSS JOIN Course
```

结果 消息

	Sno	Sname	Sex	Sage	Cno	Cname	Credit
1	201109001	李羽凡	男	20	001	数据结构	5
2	201109002	王郁郁	女	21	001	数据结构	5
3	201109003	聂乐乐	女	22	001	数据结构	5
4	201109004	张东健	男	23	001	数据结构	5
5	201109005	王子	男	24	001	数据结构	5
6	201109006	邢星	女	25	001	数据结构	5
7	201109001	李羽凡	男	20	002	计算机网络	3
8	201109002	王郁郁	女	21	002	计算机网络	3
9	201109003	聂乐乐	女	22	002	计算机网络	3
10	201109004	张东健	男	23	002	计算机网络	3
11	201109005	王子	男	24	002	计算机网络	3
12	201109006	邢星	女	25	002	计算机网络	3
13	201109001	李羽凡	男	20	003	C#程序设计	5
14	201109002	王郁郁	女	21	003	C#程序设计	5
15	201109003	聂乐乐	女	22	003	C#程序设计	5
16	201109004	张东健	男	23	003	C#程序设计	5
17	201109005	王子	男	24	003	C#程序设计	5
18	201109006	邢星	女	25	003	C#程序设计	5
19	201109001	李羽凡	男	20	004	数学	4
20	201109002	王郁郁	女	21	004	数学	4
21	201109003	聂乐乐	女	22	004	数学	4
22	201109004	张东健	男	23	004	数学	4
23	201109005	王子	男	24	004	数学	4
24	201109006	邢星	女	25	004	数学	4
25	201109001	李羽凡	男	20	005	SQL Serve...	4
26	201109002	王郁郁	女	21	005	SQL Serve...	4
27	201109003	聂乐乐	女	22	005	SQL Serve...	4
28	201109004	张东健	男	23	005	SQL Serve...	4
29	201109005	王子	男	24	005	SQL Serve...	4
30	201109006	邢星	女	25	005	SQL Serve...	4

图 9.16 交叉联接

SQL 语句如下：

```
SELECT * FROM Student
CROSS JOIN Course
```

因为 Student 表中有 6 行数据，Course 表中有 5 行数据，所以最后结果表中的行数是 5×6=30 行。

注意

由于交叉联接的结果集中行数是两个表所有行数的乘积，所以应避免对大型表使用交叉联接，否则会导致大型计算机的瘫痪。

9.2.4　联接多表

1. 在 WHERE 子句中联接多表

在 FROM 子句中写联接多个表的名称，然后将任意两个表的联接条件分别写在 WHERE 子句后。在 WHERE 子句中联接多表的语法如下：

```
SELECT fieldlist
FROM table1 , table2 , table3 ...
where table1.column=table2.column
and table2.column=table3.column and ...
```

【例 9.16】 把 Student 表、Course 表和 SC 表这 3 个表在 WHERE 子句中联接，SQL 语句及运行结果如图 9.17 所示。（**实例位置：光盘\TM\sl\9\16**）

```
SELECT * FROM Student,Course,SC
WHERE Student.Sno=SC.Sno
AND SC.Cno=Course.Cno
```

结果　消息

	Sno	Sname	Sex	Sage	Cno	Cname	Credit	Sno	Cno	Grade
1	201109001	李羽凡	男	20	001	数据结构	5	201109001	001	93
2	201109001	李羽凡	男	20	002	计算机网络	3	201109001	002	83
3	201109001	李羽凡	男	20	003	C#程序设计	5	201109001	003	52
4	201109002	王都都	女	21	001	数据结构	5	201109002	001	NULL
5	201109002	王都都	女	21	002	计算机网络	3	201109002	002	70
6	201109002	王都都	女	21	003	C#程序设计	5	201109002	003	NULL
7	201109003	聂乐乐	女	22	001	数据结构	5	201109003	001	74
8	201109003	聂乐乐	女	22	002	计算机网络	3	201109003	002	69
9	201109004	张东健	男	23	001	数据结构	5	201109004	001	NULL
10	201109004	张东健	男	23	002	计算机网络	3	201109004	002	89
11	201109004	张东健	男	23	003	C#程序设计	5	201109004	003	85
12	201109005	王子	男	24	001	数据结构	5	201109005	001	79
13	201109005	王子	男	24	002	计算机网络	3	201109005	002	90
14	201109005	王子	男	24	003	C#程序设计	5	201109005	003	45

图 9.17　用 WHERE 子句实现多表联接

SQL 语句如下：

```
SELECT * FROM Student,Course,SC
WHERE Student.Sno=SC.Sno
AND SC.Cno=Course.Cno
```

2. 在 FROM 子句中联接多表

在 FROM 子句中联接多个表是内部联接的扩展。在 FROM 子句中联接多表的语法如下：

```
SELECT fieldlist
FROM table1
join table2
join table3 ...
on table1.column=table2.column
and table2.column=table3.column
```

【例 9.17】 把 Student 表、Course 表和 SC 表这 3 个表在 FROM 子句中联接，SQL 语句及运行结

果如图 9.18 所示。（**实例位置：光盘\TM\sl\9\17**）

```
SELECT * FROM Student
 join SC
 join Course
 on SC.Cno=Course.Cno
 on Student.Sno=SC.Sno
```

结果 | 消息

	Sno	Sname	Sex	Sage	Sno	Cno	Grade	Cno	Cname	Credit
1	201109001	李羽凡	男	20	201109001	001	93	001	数据结构	5
2	201109001	李羽凡	男	20	201109001	002	83	002	计算机网络	3
3	201109001	李羽凡	男	20	201109001	003	52	003	C#程序设计	5
4	201109002	王都都	女	21	201109002	001	NULL	001	数据结构	5
5	201109002	王都都	女	21	201109002	002	70	002	计算机网络	3
6	201109002	王都都	女	21	201109002	003	NULL	003	C#程序设计	5
7	201109003	聂乐乐	女	22	201109003	001	74	001	数据结构	5
8	201109003	聂乐乐	女	22	201109003	002	69	002	计算机网络	3
9	201109004	张东健	男	23	201109004	001	NULL	001	数据结构	5
10	201109004	张东健	男	23	201109004	002	89	002	计算机网络	3
11	201109004	张东健	男	23	201109004	003	85	003	C#程序设计	5
12	201109005	王子	男	24	201109005	001	79	001	数据结构	5
13	201109005	王子	男	24	201109005	002	90	002	计算机网络	3
14	201109005	王子	男	24	201109005	003	45	003	C#程序设计	5

图 9.18　用 FROM 子句实现多表联接

SQL 语句如下：

```
SELECT * FROM Student
join SC
join Course
on SC.Cno=Course.Cno
on Student.Sno=SC.Sno
```

注意

当在 FROM 子句中联接多表时，要书写多个用来定义其中两个表的公共部分的 ON 语句，ON 语句必须遵循 FROM 后面所列表的顺序，即 FROM 后面先写的表相应的 ON 语句要先写。

在例 9.17 中，如果把两个 ON 的顺序反写，就会造成如图 9.19 所示的错误。

```
SELECT * FROM Student
 join SC
 join Course
 on Student.Sno=SC.Sno
 on SC.Cno=Course.Cno
```

消息

```
消息 4104，级别 16，状态 1，第 5 行
无法绑定由多个部分组成的标识符 "Student.Sno"。
```

图 9.19　反写 ON 语句造成的错误

9.3　使用 CASE 函数进行查询

视频讲解：光盘\TM\lx\9\\使用 CASE 函数进行查询.mp4

CASE 函数用于计算条件列表并返回多个可能结果表达式之一。

CASE 函数具有以下两种格式。

☑　简单 CASE 函数将某个表达式与一组简单表达式进行比较以确定结果。

☑　CASE 搜索函数计算一组布尔表达式以确定结果。

两种格式都支持可选的 ELSE 参数。

简单 CASE 函数的语法如下：

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

CASE 搜索函数的语法如下：

```
CASE
     WHEN Boolean_expression THEN result_expression
    [ ...n ]
     [
    ELSE else_result_expression
     ]
END
```

CASE 函数的参数及说明如表 9.1 所示。

表 9.1　CASE 函数的参数及说明

参　数	描　述
input_expression	使用简单 CASE 格式时所计算的表达式。input_expression 是任何有效的 Microsoft® SQL Server™ 表达式
WHEN when_expression	使用简单 CASE 格式时 input_expression 所比较的简单表达式。when_expression 是任意有效的 SQL Server 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
n	占位符，表明可以使用多个 WHEN when_expression THEN result_expression 子句或 WHEN Boolean_expression THEN result_expression 子句
THEN result_expression	当 input_expression = when_expression 取值为 TRUE，或者 Boolean_expression 取值为 TRUE 时返回的表达式。result_expression 是任意有效的 SQL Server 表达式
ELSE else_result_expression	当比较运算取值不为 TRUE 时返回的表达式。如果省略此参数并且比较运算取值不为 TRUE，CASE 将返回 NULL 值。else_result_expression 是任意有效的 SQL Server 表达式。else_result_expression 和所有 result_expression 的数据类型必须相同，或者必须是隐性转换
WHEN Boolean_expression	使用 CASE 搜索格式时所计算的布尔表达式。Boolean_expression 是任意有效的布尔表达式

在 SELECT 语句中，简单 CASE 函数仅检查是否相等，而不进行其他比较。本示例使用 CASE 函

数更改产品系列类别的显示，以使这些类别更易理解。SQL 语句如下：

```
SELECT    ProductNumber, Category =
      CASE ProductLine
         WHEN 'R' THEN 'Road'
         WHEN 'M' THEN 'Mountain'
         WHEN 'T' THEN 'Touring'
         WHEN 'S' THEN 'Other sale items'
         ELSE 'Not for sale'
      END,
   Name
FROM Production.Product
ORDER BY ProductNumber;
```

【例 9.18】 在 SC 表中，查询每个学生的 Sno 和 Cno，如果 Grade 大于等于 90 分显示优秀，在 80 分和 90 分之间显示“良好”，在 70 分和 80 分之间显示“中等”，在 60 分和 70 分之间显示“及格”，否则显示“不及格”，SQL 语句及运行结果如图 9.20 所示。（**实例位置：光盘\TM\sl\9\18**）

```
SELECT Sno,Cno,
等级=CASE
WHEN Grade >= 90 then '优秀'
WHEN Grade >= 80 and Grade < 90 then '良好'
WHEN Grade >= 70 and Grade < 80 then '中等'
WHEN Grade >= 60 and Grade < 70 then '及格'
ELSE '不及格'
END
FROM SC
```

结果 消息

	Sno	Cno	等级
1	201109001	001	优秀
2	201109001	002	良好
3	201109001	003	不及格
4	201109002	001	不及格
5	201109002	002	中等
6	201109002	003	不及格
7	201109003	001	中等
8	201109003	002	及格
9	201109004	001	不及格
10	201109004	002	良好
11	201109004	003	良好
12	201109005	001	中等
13	201109005	002	优秀
14	201109005	003	不及格

图 9.20　使用 CASE 查询 SC 表

SQL 语句如下：

```
SELECT Sno,Cno,
等级=CASE
WHEN Grade >= 90 then '优秀'
WHEN Grade >= 80 and Grade < 90 then '良好'
WHEN Grade >= 70 and Grade < 80 then '中等'
WHEN Grade >= 60 and Grade < 70 then '及格'
ELSE '不及格'
END
FROM SC
```

【例 9.19】 使用 CASE 语句更新 Student 表中的学生信息，使所有男生的年龄减 1，所有女生年

龄加 1。（**实例位置：光盘\TM\sl\9\19**）

SQL 语句如下：

```
USE db_2012
UPDATE Student
SET Sage=
CASE WHEN Sex= '男' THEN Sage - 1
WHEN Sex = '女' THEN Sage + 1
END
```

9.4　小　　结

本章主要对 SQL 中的高级数据查询进行了详细讲解，具体讲解过程中，由于嵌套查询中必然用到子查询，因此首先介绍了子查询和嵌套查询的概念，然后对各种嵌套查询进行了详细讲解；另外，对内外联接查询、交叉联接查询和多表联接查询进行了讲解；最后，讲解了 CASE 函数在 SQL 查询中的使用。学习本章内容时，应该重点掌握各种嵌套查询及联接查询的使用。

9.5　实践与练习

1．尝试使用 collate 函数将 Student 表中的学生信息，按姓名字段的笔画重新排序，排序规则为 chinese_prc_stroke_cs_as_ks_ws。（**答案位置：光盘\TM\sl\9\20**）

2．通过联接学生信息表 Student 与学生成绩表 Sc，实现按升序排列学习成绩名列前三的学生信息。（**答案位置：光盘\TM\sl\9\21**）

3．查询 Student 表和 Sc 表中编号相同的男同学的编号、性别、年龄和分数，并按照年龄降序排列。（**答案位置：光盘\TM\sl\9\22**）

第10章

视图的使用

（视频讲解：20分钟）

视图是一种常用的数据库对象，它将查询的结果以虚拟表的形式存储在数据中。视图并不在数据库中以存储数据集的形式存在。视图的结构和内容是建立在对表的查询基础之上的，和表一样包括行和列，这些行列数据都来源于其所引用的表，并且是在引用视图过程中动态生成的。

通过阅读本章，您可以：

- 了解视图的基本概念
- 掌握创建和删除视图
- 通过存储过程或 SQL 语句修改视图
- 通过视图查询、修改和删除数据

10.1　视 图 概 述

视图中的内容是由查询定义来的，并且视图和查询都是通过 SQL 语句定义的，它们有着许多相同和不同之处。具体如下。

- ☑ 存储：视图存储为数据库设计的一部分，而查询则不是。视图可以禁止所有用户访问数据库中的基表，而要求用户只能通过视图操作数据。这种方法可以保护用户和应用程序不受某些数据库修改的影响，同样也可以保护数据表的安全性。
- ☑ 排序：可以排序任何查询结果，但是只有当视图包括 TOP 子句时才能排序视图。
- ☑ 加密：可以加密视图，但不能加密查询。

10.2　视图的分类与操作

视图为数据呈现提供了多样的表现形式，用户可以通过它浏览表中感兴趣的数据。在 SQL Server 2012 中视图分为以下 3 类。

- ☑ 标准视图：保存在数据库中的 SELECT 查询语句。即通常意义上理解的视图。
- ☑ 索引视图：创建有索引的视图称为索引视图。它经过计算并存储有自己的数据，可以提高某些类型查询的性能，尤其适用于聚合许多行的查询，但不太适用于经常更新的基本数据集。
- ☑ 分区视图：是在一台或多台服务器间水平连接一组表中的分区数据，以使数据看上去来源于一个表。

10.2.1　以界面方式操作视图

1．视图的创建

下面在 SQL Server Management Studio 中创建视图“View_Student”，具体操作步骤如下：

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库。

（2）在“对象资源浏览器”中展开“数据库”节点，展开指定的数据库 db_2012。

（3）右击“视图”选项，在弹出的快捷菜单中选择“新建视图”命令，如图 10.1 所示。

（4）进入“添加表”对话框，如图 10.2 所示。在列表框中选择学生信息表 Student，单击“添加”按钮，然后单击“关闭”按钮关闭该对话框。

（5）进入“视图设计器”界面，如图 10.3 所示。在“表选择区”中选择“所有列”选项，单击“执行”按钮，视图结果区中自动显示视图结果。

（6）单击工具栏中的“保存”按钮，弹出“选择名称”对话框，如图 10.4 所示。在“输入视图名称”文本框中输入视图名称“View_Student”，单击“确定”按钮即可保存该视图。

图 10.1　新建视图

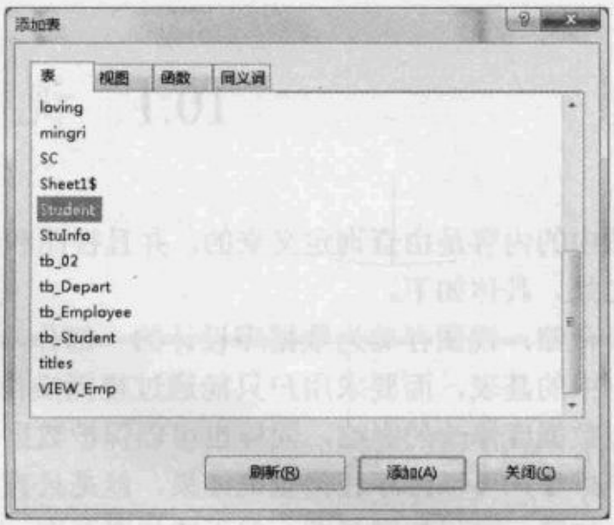


图 10.2　“添加表”对话框

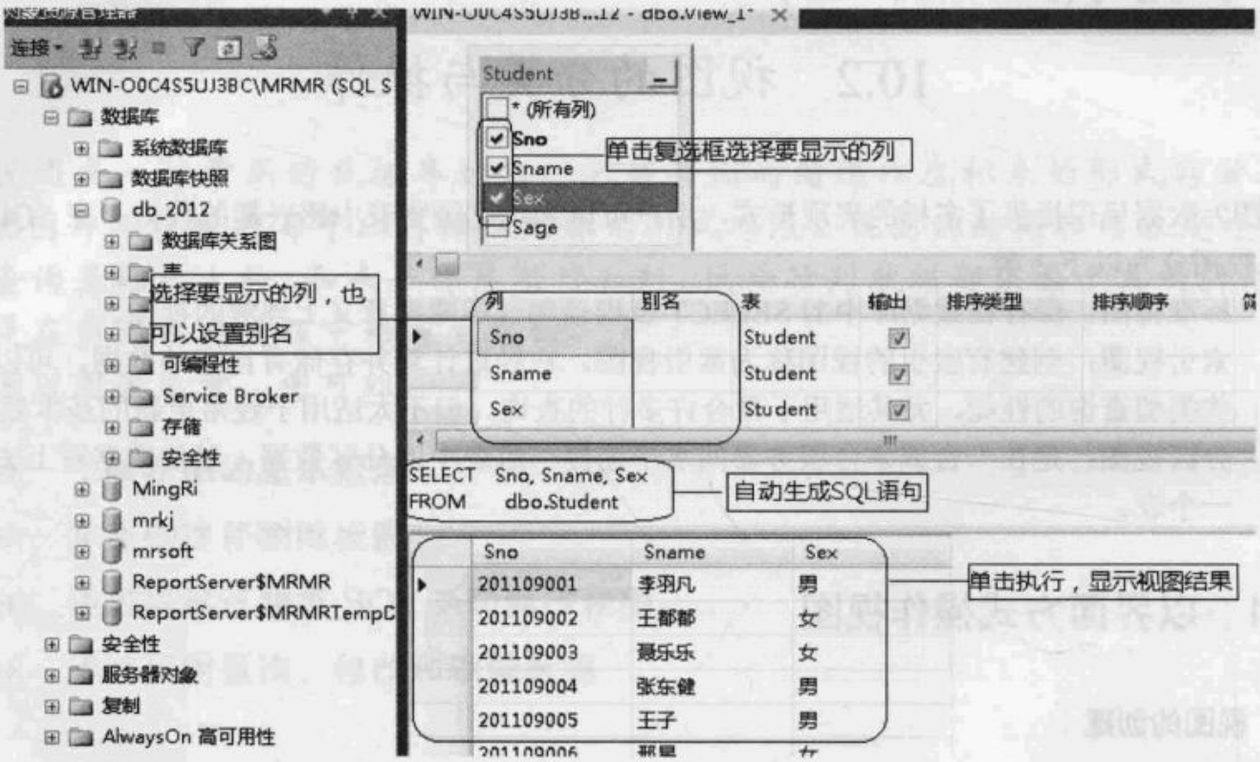


图 10.3　视图设计器

2. 视图的删除

用户可以删除视图。删除视图时，底层数据表不受影响，但会造成与该视图关联的权限丢失。下面介绍如何在 SQL Server Management Studio 管理器中删除视图，具体操作步骤如下：

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库。

（2）在“对象资源浏览器”中展开“数据库”节点，展开指定的数据库 db_2012。

（3）展开“视图”节点，鼠标右键单击要删除的视图 View_Student，在弹出的快捷菜单中选择“删除”命令，如图 10.5 所示。

（4）在弹出的“删除对象”对话框中单击“确定”按钮即可删除该视图。

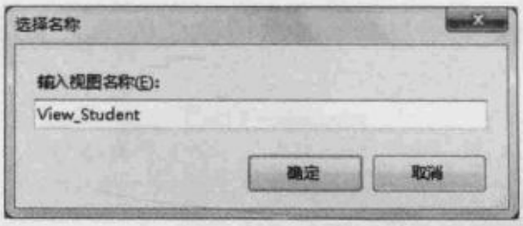


图 10.4　“选择名称”对话框

图 10.5　删除视图

10.2.2　使用 CREATE VIEW 语句创建视图

使用 CREATE VIEW 语句可以创建视图，语法如下：

```
CREATE VIEW [ schema_name . ] view_name [ (column [ ,...n ] ) ]
[ WITH <view_attribute> [ ,...n ] ]
AS select_statement [ ; ]
[ WITH CHECK OPTION ]
<view_attribute> ::=
{
  [ ENCRYPTION ]  [ SCHEMABINDING ]  [ VIEW_METADATA ]
}
```

参数如表 10.1 所示。

表 10.1　CREATE VIEW 语句参数说明

参　数	说　明
schema_name	视图所属架构的名称
view_name	视图的名称。视图名称必须符合有关标识符的规则。可以选择是否指定视图所有者名称
column	视图中的列使用的名称
AS	指定视图要执行的操作
select_statement	定义视图的 SELECT 语句
CHECK OPTION	强制针对视图执行的所有数据修改语句都必须符合在 select_statement 中设置的条件
ENCRYPTION	对视图进行加密

续表

参　数	说　明
SCHEMABINDING	将视图绑定到基础表的架构
VIEW_METADATA	指定为引用视图的查询请求浏览模式的元数据时，SQL Server 实例将向 DB-Library、ODBC 和 OLE DB API 返回有关视图的元数据信息，而不返回基表的元数据信息

【例 10.1】 创建查询 Student 数据表中的所有记录的视图 VIEW1。（**实例位置：光盘\TM\ sl\10\1**）

代码如下：

```
CREATE VIEW VIEW1                    --创建视图
AS                                   --指定视图要执行的操作
SELECT * FROM Student                --结果集
```

【例 10.2】 在新视图中只显示 ID、Name、Sex、Age 的信息，同时获得视图的相关信息。（**实例位置：光盘\TM\sl\10\2**）

代码如下：

```
use db_2012
go
--创建视图
create view v1
as
--定义 select 语句
select ID,Name,Sex,Age from Employee
go
--查询所创建的视图中数据
select * from v1
```

运行结果如图 10.6 所示。

图 10.6　创建视图获取相关信息

【例 10.3】 创建视图，使用 INSERT 语句向信息表中添加数据信息。（**实例位置：光盘\TM\sl\10\3**）

代码如下：

```
USE db_2012
GO
```

```
CREATE VIEW view3
AS
SELECT * FROM Employee1
GO
INSERT INTO view3(ID,Name)
VALUES(7,'刘莉')
GO
INSERT INTO view3(ID,Name,Sex)
VALUES(8,'张一','男')
```

运行结果如图 10.7 所示。

【例 10.4】 创建带检查约束的视图，名称是 view5。（**实例位置：光盘\TM\sl\10\4**）

代码如下：

```
create view view5
as
select ID,Name,Age from Employee where Age>10
WITH CHECK OPTION                    --带有检查约束
go
insert into view5 (ID,Age,Name)values(11,8,'Peter')
```

当在视图 VIEW5 的年龄字段中输入的值小于等于 10 时，弹出错误提示，如图 10.8 所示。

图 10.7　向视图中添加数据

图 10.8　约束提示

10.2.3　使用 ALTER VIEW 语句修改视图

使用 ALTER VIEW 语句可以修改视图，语法如下：

```
ALTER VIEW   view_name [( column [,...n])]
[WITH ENCRYPTION]
AS
select_statement
[WITH CHECK OPTION]
```

参数如表 10.2 所示。

表 10.2 ALTER VIEW 语句参数说明

参　数	说　明
view_name	要更改的视图
column	一列或多列的名称，用逗号分开，将成为给定视图的一部分
n	表示 column 可重复 n 次的占位符
WITH ENCRYPTION	加密 syscomments 表中包含 ALTER VIEW 语句文本的条目。使用 WITH ENCRYPTION 可防止将视图作为 SQL Server 复制的一部分发布
AS	视图要执行的操作
select_statement	定义视图的 SELECT 语句
WITH CHECK OPTION	强制视图上执行的所有数据的修改语句都必须符合由定义视图的 select_statement 设置的准则

说明

如果原来的视图定义是用 WITH ENCRYPTION 或 CHECK OPTION 创建的，那么只有在 ALTER VIEW 中也包含这些选项时，这些选项才有效。

【例 10.5】 通过 Transact-SQL 语句中的 ALTER VIEW 对已存在的视图进行修改操作。（**实例位置：光盘\TM\sl\10\5**）

代码如下：

```
ALTER VIEW View_Student(Sname,Sage)          --修改已存在的视图
AS
SELECT Sname,Sage FROM Student WHERE sno='201109002'
go
EXEC sp_helptext 'View_student'              --查看视图定义
```

使用 ALTER VIEW 是对视图的修改，使用 UPDATE 是通过视图对数据表做修改。例如，使用 UPDATE 语句通过视图对数据表中的数据进行更新，修改信息表中的数据。

```
use db_2012
go
--通过视图修改数据
update v1 set Name='张一' where ID=2
--查询视图中修改后的数据
select * from v1
```

运行结果如图 10.9 所示。

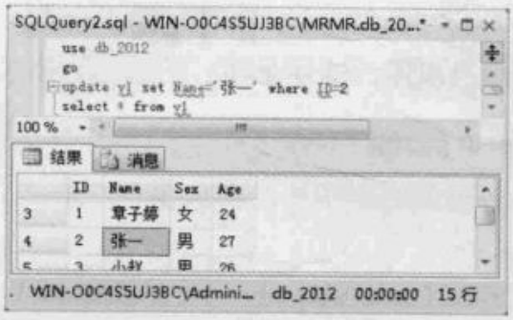


图 10.9 通过视图修改数据

10.2.4　使用 DROP VIEW 语句删除视图

使用 DROP VIEW 语句可以删除视图，语法如下：

```
DROP VIEW view_name [,...n]
```

参数如表 10.3 所示。

表 10.3　DROP VIEW 语句参数说明

参　数	说　明
view_name	要删除的视图名称。视图名称必须符合标识符规则。可以选择是否指定视图所有者名称。若要查看当前创建的视图列表，请使用 sp_help
n	表示可以指定多个视图的占位符

注意

在单击“全部除去”按钮删除视图以前，可以在“除去对象”对话框中单击“显示相关性”按钮，即可查看该视图依附的对象，以确认该视图是否为想要删除的视图。

【例 10.6】 使用 Transact-SQL 删除视图的实现过程如下。（**实例位置：光盘\TM\sl\10\6**）

（1）首先单击“新建查询”按钮。

（2）在代码编辑窗口中输入以下代码，单击工具栏上的“执行”按钮。此时执行查询结果将在下面的窗口中显示出来。相关代码如下：

```
USE db_2012
GO
DROP VIEW View_Student                    --删除视图
GO
```

【例 10.7】 使用 DELETE 语句通过视图将数据表中的数据删除。（**实例位置：光盘\TM\sl\ 10\7**）

代码如下：

```
use db_2012
go
DELETE v1
WHERE Name='张一'
--查看创建的视图中的数据
SELECT * FROM v1
go
```

运行结果如图 10.10 所示。

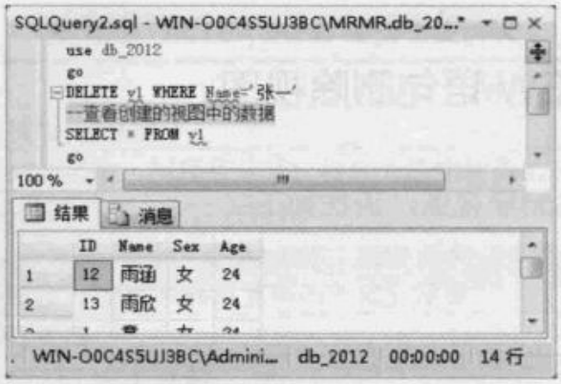


图 10.10　删除视图中的数据

10.2.5　使用存储过程 sp_rename 修改视图

使用 sp_rename 视图，语法如下：

```
exec sp_rename old_view, new_view
```

将视图 view1 重命名为视图 view2，代码如下：

```
exex sp_rename view1,view2
```

10.3　通过视图操作数据

10.3.1　从视图中浏览数据

下面在 SQL Server Management Studio 中查看视图 View_Student 的信息（如果该视图已删除，请按 10.2.1 节的方法创建此视图），具体操作步骤如下：

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库。

（2）在“对象资源浏览器”中展开“数据库”节点，展开指定的数据库 db_2012。

（3）再依次展开“视图”节点，就会显示出当前数据库中的所有视图，鼠标右键单击要查看信息的视图。

（4）在弹出的快捷菜单中，如果想要查看视图的属性，单击“属性”选项，如图 10.11 所示，弹出“视图属性”对话框，如图 10.12 所示。

（5）如果想要查看视图中的内容，可在如图 10.11 所示的快捷菜单中选择“编辑前 200 行”选项，在右侧即可显示视图中的内容，如图 10.13 所示。

（6）如果想要重新设置视图，可在如图 10.11 所示的快捷菜单中单击“设计”选项，弹出视图的设计窗体，如图 10.14 所示。在此窗体中可对视图进行重新设置。

图 10.11　查看视图属性

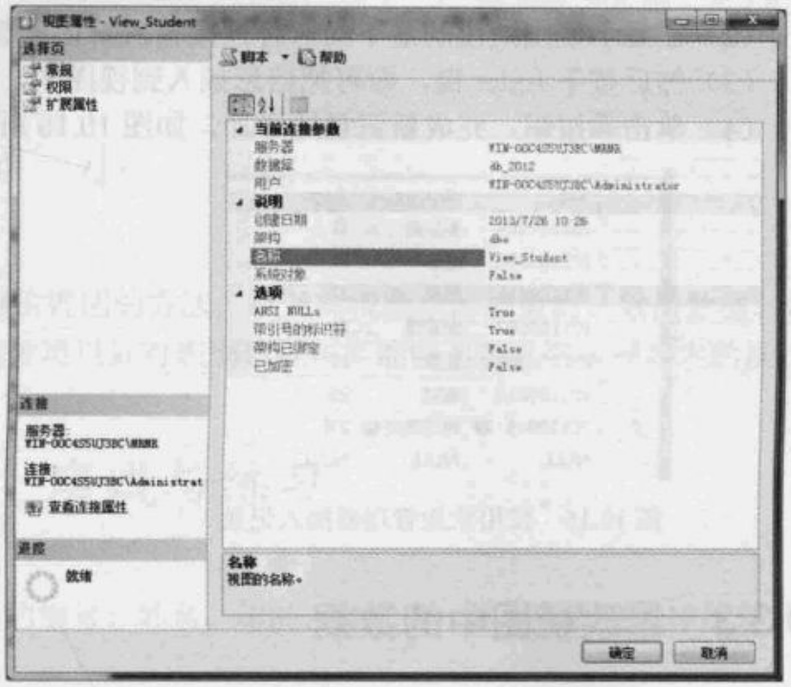


图 10.12　“视图属性”对话框

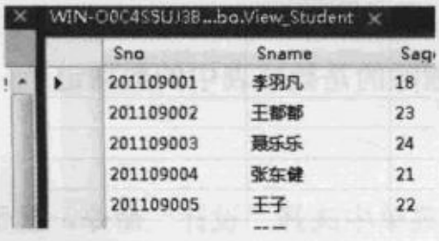


图 10.13　显示视图中的内容

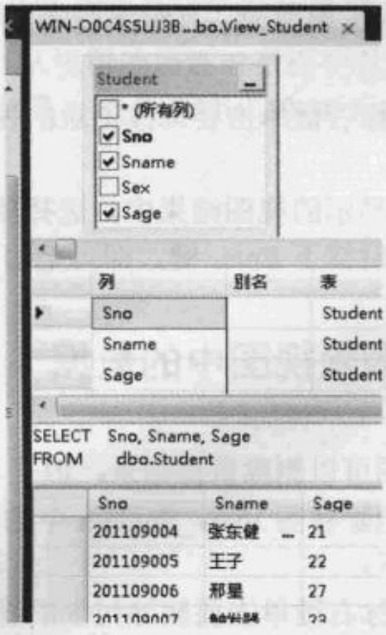


图 10.14　视图设计界面

10.3.2　向视图中添加数据

使用视图可以添加新的记录，但应该注意的是，新添加的数据实际上是存储在与视图相关的表中。例如，向视图 View_Student 中插入信息“20110901，明日科技，27”。

步骤如下：

（1）鼠标右键单击要插入记录的视图，在弹出的快捷菜单中选择“设计”命令，显示视图的设计界面。

（2）在显示视图结果的最下面一行直接输入新记录即可，如图 10.15 所示。

（3）然后按下 Enter 键，即可把信息插入到视图中。

（4）单击 按钮，完成新记录的添加，如图 10.16 所示。

	Sno	Sname	Sage
	201109004	张东健 ...	21
	201109005	王子	22
	201109006	邢星	27
	201109007	触发器 ...	23
	201109008	赵雪	25
	201109019	章立	26
	20110901	明日科技	27
*	NULL	NULL	NULL

图 10.15　使用企业管理器插入记录

Sno	Sname	Sage
201109004	张东健 ...	21
201109005	王子	22
201109006	邢星	27
201109007	触发器 ...	23
201109008	赵雪	25
201109019	章立	26
20110901	明日科技	27

图 10.16　插入记录后的视图

10.3.3　修改视图中的数据

使用视图可以修改数据记录，但是与插入记录相同，修改的是数据表中的数据记录。

例如，修改视图 View_Student 中的记录，将“明日科技”修改为“明日”。

步骤如下：

（1）鼠标右键单击要修改记录的视图，在弹出的快捷菜单中选择“设计”命令，显示视图的设计界面。

（2）在显示的视图结果中，选择要修改的内容，直接修改即可。

（3）最后按下 Enter 键，即可把信息保存到视图中。

10.3.4　删除视图中的数据

使用视图可以删除数据记录，但是与插入记录相同，删除的是数据表中的数据记录。

例如，删除视图 View_Student 中的记录“明日科技”。

步骤如下：

（1）鼠标右键单击要删除记录的视图，在弹出的快捷菜单中选择“设计”命令，显示视图的设计界面。

（2）在显示视图的结果中，选择要删除的行“明日科技”，在弹出的快捷菜单中选择“删除”命令，弹出删除对话框，如图 10.17 所示。

图 10.17　删除视图对话框

（3）单击“是”按钮，便将该记录删除。

10.4　小　　结

本章介绍了视图的创建、修改视图和删除视图的方法。以及向视图中添加数据、修改数据、删除数据等，并且在视图中对数据进行操作。读者可以针对表创建视图并能够通过视图实现对表的操作。

10.5　实践与练习

1．在学生信息表 Student 中存储了学生的编号、姓名、年龄、性别等，使用视图过滤掉性别。（**答案位置：光盘\TM\sl\10\8**）

2．使用系统存储过程 sp_helptext 获取视图 vv1 定义时的相关信息。（**答案位置：光盘\TM\sl\10\9**）

3．创建一个视图，使用 WITH ENCRYPTION 将原文本转换为模糊格式，实现视图定义文本加密。加密后的视图无法使用系统存储过程 sp_helptext 查看其信息。从视图的属性对话框中看加密后的视图信息，视图定义文本将被一段不可使用的信息所替代。（**答案位置：光盘\TM\sl\10\10**）

第3篇

高级应用

- 第 11 章　存储过程
- 第 12 章　触发器
- 第 13 章　游标的使用
- 第 14 章　索引与数据完整性
- 第 15 章　SQL 中的事务
- 第 16 章　维护 SQL Server 2012
- 第 17 章　数据库的安全机制

本篇介绍存储过程、触发器、游标的使用、索引与数据完整性、SQL 中的事务、维护 SQL Server 2012、数据库的安全机制等。学习完这一部分，能够使用索引优化数据库查询；使用存储过程、触发器、游标、事务等编写 SQL 语句，不仅可以优化查询，还可以提高数据访问速度；更好地维护 SQL Server 2012 及其安全。

第11章

存储过程

（视频讲解：18分钟）

存储过程代替了传统的逐条执行SQL语句的方式。存储过程是预编译SQL语句的集合，这些语句存储在一个名称下并作为一个单元来处理。一个存储过程中可包含查询、插入、删除、更新等操作的一系列SQL语句，当这个存储过程被调用执行时，这些操作也会同时执行。

通过阅读本章，您可以：

- 了解存储过程的基本概念
- 掌握创建存储过程的两种方法
- 执行存储过程
- 使用sys.sql_modules查看存储过程的定义
- 通过ALTER PROCEDURE语句修改存储过程
- 存储过程重命名
- 删除存储过程

11.1　存储过程概述

视频讲解：光盘\TM\lx\11\存储过程概述.mp4

11.1.1　存储过程的概念

存储过程（Stored Procedure）是预编译 SQL 语句的集合，这些语句存储在一个名称下并作为一个单元来处理。存储过程代替了传统的逐条执行 SQL 语句的方式。一个存储过程中可包含查询、插入、删除、更新等操作的一系列 SQL 语句，当这个存储过程被调用执行时，这些操作也会同时执行。

存储过程与其他编程语言中的过程类似，它可以接受输入参数并以输出参数的格式向调用过程或批处理返回多个值；包含用于在数据库中执行操作（包括调用其他过程）的编程语句；向调用过程或批处理返回状态值，以指明成功或失败（以及失败的原因）。

SQL Server 提供了 3 种类型的存储过程。各类型存储过程如下。

- ☑ 系统存储过程：用来管理 SQL Server 和显示有关数据库和用户的信息的存储过程。
- ☑ 自定义存储过程：用户在 SQL Server 中通过采用 SQL 语句创建存储过程。
- ☑ 扩展存储过程：通过编程语言（例如 C）创建外部例程，并将这个例程在 SQL Server 中作为存储过程使用。

11.1.2　存储过程的优点

存储过程的优点表现在以下几个方面：

（1）存储过程可以嵌套使用，支持代码重用。

（2）存储过程可以接受与使用参数动态执行其中的 SQL 语句。

（3）存储过程比一般的 SQL 语句执行速度快。存储过程在创建时已经被编译，每次执行时不需要重新编译。而 SQL 语句每次执行都需要编译。

（4）存储过程具有安全特性（例如权限）和所有权链接，以及可以附加到它们的证书。用户可以被授予权限来执行存储过程而不必直接对存储过程中引用的对象具有权限。

（5）存储过程允许模块化程序设计。存储过程一旦创建，以后即可在程序中调用任意多次。这可以改进应用程序的可维护性，并允许应用程序统一访问数据库。

（6）存储过程可以减少网络通信流量。一个需要数百行 SQL 语句代码的操作可以通过一条执行过程代码的语句来执行，而不需要在网络中发送数百行代码。

（7）存储过程可以强制应用程序的安全性。参数化存储过程有助于保护应用程序不受 SQL Injection 攻击。

说明

SQL Injection 是一种攻击方法，它可以将恶意代码插入到以后将传递给 SQL Server 供分析和执行的字符串中。任何构成 SQL 语句的过程都应进行注入漏洞检查，因为 SQL Server 将执行其接收到的所有语法有效的查询。

11.2 创建存储过程

视频讲解：光盘\TM\lx\11\创建存储过程.mp4

存储过程是在数据库服务器端执行的一组 T-SQL 语句的集合，经编译后存放在数据库服务器中。本节主要介绍如何通过企业管理器和 Transact-SQL 语句创建存储过程。

11.2.1 使用向导创建存储过程

在 SQL Server 2012 中，使用向导创建存储过程的步骤如下：

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库。

（2）在“对象资源管理器”中选择指定的服务器和数据库，展开数据库的“可编辑性”节点，鼠标右键单击“存储过程”，在弹出的快捷菜单中选择“新建存储过程”命令，如图 11.1 所示。

（3）在弹出的“连接到数据库引擎”窗口中，单击“连接”按钮，便出现创建存储过程的窗口，如图 11.2 所示。

图 11.1 选择“新建存储过程”命令

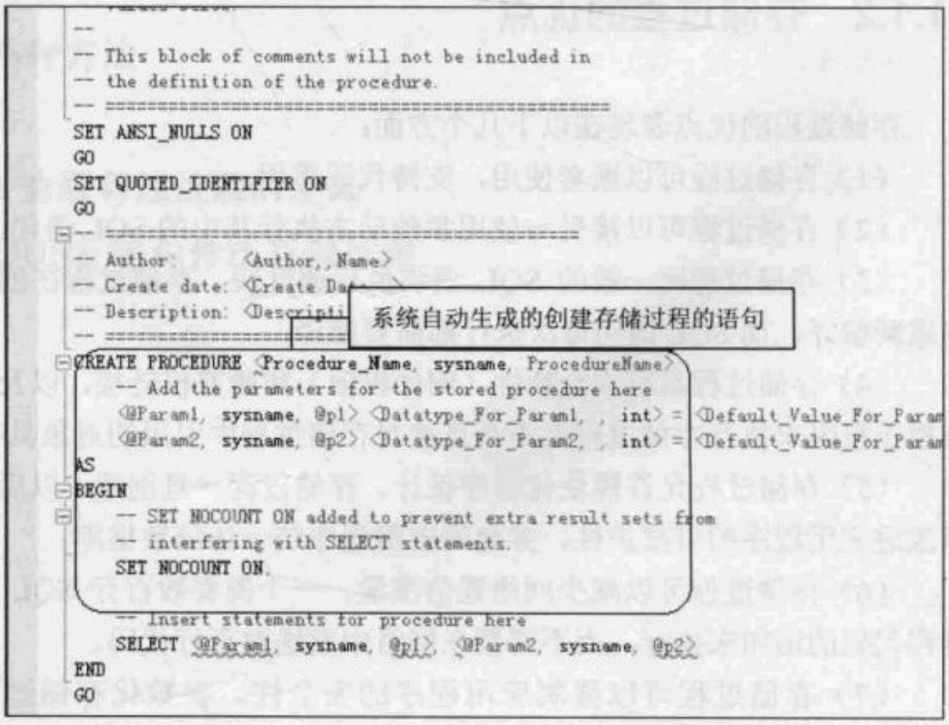


图 11.2 创建存储过程窗口

在存储过程窗口的文本框中，可以看到系统自动给出了创建存储过程的格式模板语句，可以根据模板格式进行修改来创建新的存储过程。

【例 11.1】 创建一个名称为 Proc_Stu 的存储过程，要求完成以下功能：在 Student 表中查询男生的 Sno，Sex，Sage 这几个字段的内容。（**实例位置：光盘\TM\sl\11\1**）

具体的操作步骤如下：

（1）在创建存储过程的窗口中单击“查询”菜单，选择“指定模板参数的值”，弹出“指定模板参数的值”对话框，如图 11.3 所示。

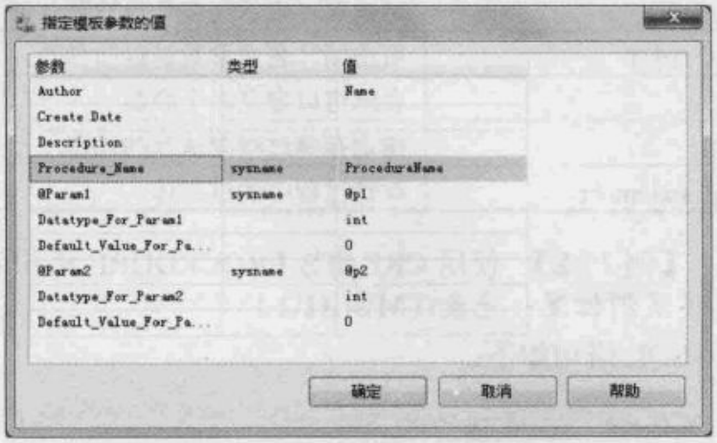


图 11.3　指定模板参数的值

（2）在“指定模板参数的值”对话框中将“Procedure_Name”参数对应的名称修改为“Proc_Stu”，单击“确定”按钮，关闭此对话框。

（3）在创建存储过程的窗口中，将对应的 SELECT 语句修改为以下的语句：

```
SELECT Sno,Sname,Sex,Sage
FROM Student
WHERE Sex='男'
```

11.2.2　使用 CREATE PROC 语句创建存储过程

在 SQL 中，可以使用 CREATE PROCEDURE 语句创建存储过程，其语法格式如下：

```
CREATE PROC [ EDURE ] procedure_name [ ; number ]
    [ { @parameter data_type }
        [ VARYING ] [ = default ] [ OUTPUT ]
    ] [ ,...n ]
AS sql_statement
```

CREATE PROC 语句的参数及说明如表 11.1 所示。

表 11.1　CREATE PROC 语句的参数及说明

参　数	描　述
CREATE PROCEDURE	关键字，也可以写成 CREATE PROC
procedure_name	创建的存储过程名称
number	对存储过程进行分组
@parameter	存储过程参数，存储过程可以声明一个或多个参数
data_type	参数的数据类型，所有数据类型（包括 text、ntext 和 image）均可以用作存储过程的参数，但是，cursor 数据类型只能用于 OUTPUT 参数
VARYING	可选项，指定作为输出参数支持的结果集（由存储过程动态构造，内容可以变化），该关键字仅适用于游标参数

续表

参　数	描　述
default	可选项，表示为参数设置默认值
OUTPUT	可选项，表明参数是返回参数，可以将参数值返回给调用的过程
n	表示可以定义多个参数
AS	指定存储过程要执行的操作
sql_statement	存储过程中的过程体

【例 11.2】 使用 CREATE PROCEDURE 语句创建一个存储过程，用来根据学生编号查询学生信息。（**实例位置：光盘\TM\sl\11\2**）

SQL 语句如下：

```
Create Procedure Proc_Student
@Proc_Sno int
as
select * from Student where Sno = @Proc_Sno
```

运行结果如图 11.4 所示。

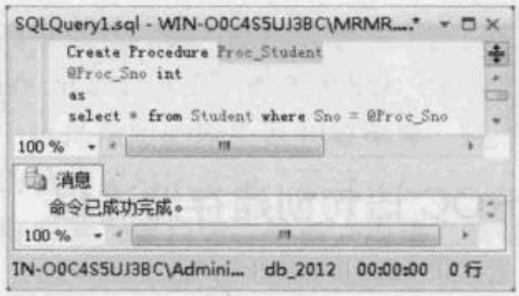


图 11.4　创建存储过程

11.3　管理存储过程

视频讲解：光盘\TM\lx\11\管理存储过程.mp4

存储过程创建完成后，用户可以通过 SQL Server Management Studio 工具对其进行管理。数据库中的存储过程都被保存在“数据库”/“数据库名称”/“可编程性”/“存储过程”路径下。本节介绍使用 SQL Server Management Studio 工具对存储过程进行查看代码、修改代码及名称、删除、执行等管理。

11.3.1　执行存储过程

存储过程创建完成后，可以通过 EXECUTE 命令执行，可简写为 EXEC。

1. EXECUTE

EXECUTE 用来执行 Transact-SQL 中的命令字符串、字符串或执行下列模块之一：系统存储过程、用户定义存储过程、标量值用户定义函数或扩展存储过程。

EXECUTE 的语法如下：

```
[{ EXEC | EXECUTE }]
    {
      [ @return_status = ]
      { module_name [ ;number ] | @module_name_var }
        [ [ @parameter = ] { value
                           | @variable [ OUTPUT ]
                           | [ DEFAULT ]
                           }
        ]
      [ ,...n ]
      [ WITH RECOMPILE ]
    }
[;]
```

EXECUTE 语句的参数及说明如表 11.2 所示。

表 11.2　EXECUTE 语句的参数及说明

参　数	描　述
@return_status	可选的整型变量，存储模块的返回状态。这个变量在用于 EXECUTE 语句前，必须在批处理、存储过程或函数中声明过
module_name	是要调用的存储过程或标量值用户定义函数的完全限定或者不完全限定名称。模块名称必须符合标识符规则。无论服务器的排序规则如何，扩展存储过程的名称总是区分大小写
number	是可选整数，用于对同名的过程分组。该参数不能用于扩展存储过程
@module_name_var	是局部定义的变量名，代表模块名称
@parameter	module_name 的参数，与在模块中定义的相同。参数名称前必须加上符号（@）
value	传递给模块或传递命令的参数值。如果参数名称没有指定，参数值必须以在模块中定义的顺序提供
@variable	是用来存储参数或返回参数的变量
OUTPUT	指定模块或命令字符串返回一个参数。该模块或命令字符串中的匹配参数也必须已使用关键字 OUTPUT 创建。使用游标变量作为参数时使用该关键字
DEFAULT	根据模块的定义，提供参数的默认值。当模块需要的参数值没有定义默认值并且缺少参数或指定了 DEFAULT 关键字，会出现错误
WITH RECOMPILE	执行模块后，强制编译、使用和放弃新计划。如果该模块存在现有查询计划，则该计划将保留在缓存中

注意

后续版本的 Microsoft SQL Server 将删除该功能。请避免在新的开发工作中使用该功能，并着手修改当前还在使用该功能的应用程序。

2. 使用 EXECUTE 执行存储过程

【例 11.3】 使用 EXECUTE 执行存储过程 Proc_Stu。（**实例位置：光盘\TM\sl\11\3**）

SQL 语句如下：

```
exec Proc_Stu
```

使用 EXECUTE 执行存储过程的步骤如下：

（1）打开 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库。

（2）单击工具栏中的 新建查询(N) 按钮，新建查询编辑器。并输入如下 SQL 语句代码。

```
exec Proc_Stu
```

（3）单击 执行(X) 按钮，就可以执行上述 SQL 语句代码，即可完成执行 Proc_Stu 存储过程。执行结果如图 11.5 所示。

图 11.5　执行存储过程的结果

11.3.2　查看存储过程

许多系统存储过程、系统函数和目录视图都提供有关存储过程的信息。可以使用这些系统存储过程来查看存储过程的定义，即用于创建存储过程的 Transact-SQL 语句。

可以通过下面 3 种系统存储过程和目录视图查看存储过程。

1. 使用 sys.sql_modules 查看存储过程的定义

sys.sql_modules 为系统视图，通过该视图可以查看数据库中的存储过程。查看存储过程的操作方法如下：

（1）单击工具栏中的 新建查询(N) 按钮，新建查询编辑器。

（2）在新建查询编辑器中输入如下代码：

```
select * from sys.sql_modules
```

（3）单击 执行(X) 按钮，执行该查询命令。查询结果如图 11.6 所示。

2. 使用 OBJECT_DEFINITION 查看存储过程的定义

返回指定对象定义的 Transact-SQL 源文本。语法如下：

```
OBJECT_DEFINITION ( object_id )
```

参数说明如下。

object_id：要使用的对象的 ID。object_id 的数据类型为 int，并假定表示当前数据库上下文中的对象。

【例 11.4】 使用 OBJECT_DEFINITION 查看 ID 为“309576141”的存储过程的代码。（**实例位置：光盘\TM\sl\11\4**）

SQL 语句如下：

```
select OBJECT_DEFINITION(309576141)
```

3. 使用 sp_helptext 查看存储过程的定义

显示用户定义规则的定义、默认值、未加密的 Transact-SQL 存储过程、用户定义 Transact-SQL 函数、触发器、计算列、CHECK 约束、视图或系统对象（如系统存储过程）。语法如下：

```
sp_helptext [ @objname = ] 'name' [ , [ @columnname = ] computed_column_name ]
```

参数说明如下。

☑ [@objname =] 'name'：架构范围内的用户定义对象的限定名称和非限定名称。仅当指定限定对象时才需要引号。如果提供的是完全限定名称（包括数据库名称），则数据库名称必须是当前数据库的名称。对象必须在当前数据库中。name 的数据类型为 nvarchar(776)，无默认值。

☑ [@columnname =] 'computed_column_name'：要显示其定义信息的计算列的名称。必须将包含列的表指定为 name。column_name 的数据类型为 sysname，无默认值。

【例 11.5】 通过 sp_helptext 系统存储过程查看名为“Proc_Stu”存储过程的代码。（**实例位置：光盘\TM\sl\11\5**）

SQL 语句如下：

```
sp_helptext 'Proc_Stu'
```

操作步骤如下：

（1）打开 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库。

（2）选择存储过程所在的数据库，例如 db_2012 数据库。

（3）单击工具栏中的 新建查询(N) 按钮，新建查询编辑器。并输入如下 SQL 语句代码。

```
sp_helptext 'Proc_Stu'
```

（4）单击 执行(X) 按钮，就可以执行上述 SQL 语句代码。执行结果如图 11.7 所示。

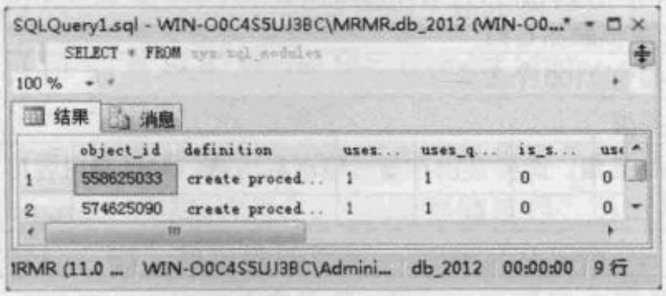


图 11.6　使用 sys.sql_modules 视图查询的存储过程

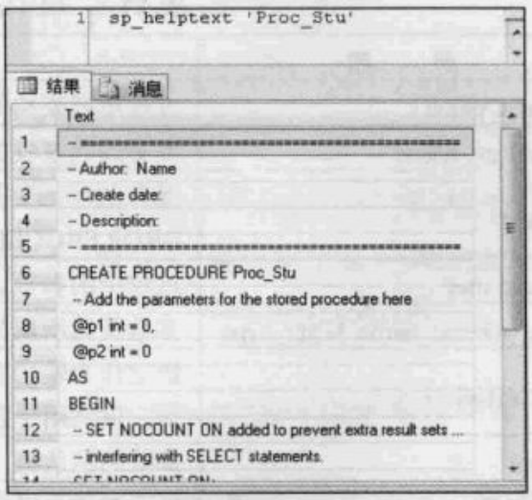


图 11.7　查看 Proc_Stu 存储过程的结果

11.3.3　修改存储过程

修改存储过程可以改变存储过程当中的参数或者语句，可以通过 SQL 语句中的 ALTER PROCEDURE 语句实现。虽然删除并重新创建该存储过程，也可以达到修改存储过程的目的，但是将丢失与该存储过程关联的所有权限。

1. ALTER PROCEDURE 语句

ALTER PROCEDURE 语句用来修改通过执行 CREATE PROCEDURE 语句创建的过程。该语句修

改存储过程时不会更改权限，也不影响相关的存储过程或触发器。

ALTER PROCEDURE 语句的语法如下：

```
ALTER { PROC | PROCEDURE } [schema_name.] procedure_name [ ; number ]
    [ { @parameter [ type_schema_name. ] data_type }
    [ VARYING ] [ = default ] [ [ OUT [ PUT ]
    ] [ ,...n ]
[ WITH <procedure_option> [ ,...n ] ]
[ FOR REPLICATION ]
AS
     { <sql_statement> [ ...n ] | <method_specifier> }
<procedure_option> ::=
    [ ENCRYPTION ]
    [ RECOMPILE ]
    [ EXECUTE_AS_Clause ]
<sql_statement> ::=
{ [ BEGIN ] statements [ END ] }
<method_specifier> ::=
EXTERNAL NAME
assembly_name.class_name.method_name
```

ALTER PROCEDURE 语句的参数及说明如表 11.3 所示。

表 11.3　ALTER PROCEDURE 语句的参数及说明

参　数	描　述
schema_name	过程所属架构的名称
procedure_name	要更改的过程的名称。过程名称必须符合标识符规则
number	现有的可选整数，该整数用来对具有同一名称的过程进行分组，以便可以用一个 DROP PROCEDURE 语句全部删除它们
@parameter	过程中的参数。最多可以指定 2100 个参数
[type_schema_name.] data_type	参数及其所属架构的数据类型
VARYING	指定作为输出参数支持的结果集。此参数由存储过程动态构造，并且其内容可以不同。仅适用于游标参数
default	参数的默认值
OUTPUT	指示参数是返回参数
FOR REPLICATION	指定不能在订阅服务器上执行为复制创建的存储过程
AS	过程将要执行的操作
ENCRYPTION	指示数据库引擎会将 ALTER PROCEDURE 语句的原始文本转换为模糊格式
RECOMPILE	指示 SQL Server 2012 数据库引擎不会缓存该过程的计划，该过程在运行时重新编译
EXECUTE AS	指定访问存储过程后执行该存储过程所用的安全上下文
<sql_statement>	过程中要包含的任意数目和类型的 Transact-SQL 语句。但有一些限制
EXTERNAL NAME assembly_name.class_name.method_name	指定 Microsoft .NET Framework 程序集的方法，以便 CLR 存储过程引用。class_name 必须为有效的 SQL Server 标识符，并且必须作为类存在于程序集中。如果类具有使用句点(.)分隔命名空间部分的命名空间限定名称，则必须使用方括号([ ])或引号(" ")来分隔类名。指定的方法必须为该类的静态方法

注意

默认情况下，SQL Server 不能执行 CLR 代码。可以创建、修改和删除引用公共语言运行时模块的数据库对象；不过，只有在启用 clr enabled 选项之后，才能在 SQL Server 中执行这些引用。若要启用该选项，请使用 sp_configure。

2. 使用 ALTER PROCEDURE 语句修改存储过程

【例 11.6】 通过 ALTER PROCEDURE 语句修改名为“Proc_Stu”的存储过程。（**实例位置：光盘\TM\sl\11\6**）

具体操作步骤如下：

（1）打开 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库。

（2）选择存储过程所在的数据库，例如 db_2012 数据库。

（3）单击工具栏中的 新建查询(N) 按钮，新建查询编辑器。并输入如下 SQL 语句代码。

```
ALTER PROCEDURE [dbo].[Proc_Stu]
@Sno varchar(10)
as
select * from b '
```

（4）单击 执行(X) 按钮，就可以执行上述 SQL 语句代码。执行结果如图 11.8 所示。

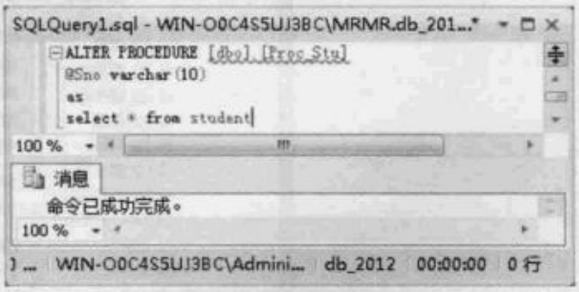


图 11.8　使用 ALTER PROCEDURE 语句修改存储过程

除了上述方法修改存储过程外，也可以通过 SQL Server 2012 自动生成的 ALTER PROCEDURE 语句修改存储过程。以修改系统数据库 master 中系统存储过程 sp_MScleanupmergepublisher 为例，操作步骤如下：

（1）打开 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库。

（2）展开对象资源管理器中“数据库”/“系统数据库”/master/“可编程性”/“系统存储过程”节点后，在 sp_MScleanupmergepublisher 系统存储过程上单击鼠标右键，弹出快捷菜单，如图 11.9 所示。

（3）选择“修改”菜单项，在查询编辑器中自动生成修改该存储过程的语句。生成的语句如图 11.10 所示。

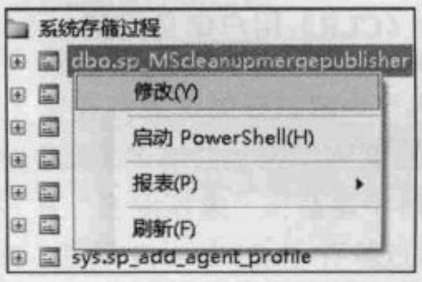


图 11.9　修改存储过程

```
USE [master]
GO
/****** Object:  StoredProcedure [dbo].
[sp_MScleanupmergepublisher]    Script Date: 10/07
/2011 13:24:41 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER OFF
GO
ALTER procedure [dbo].[sp_MScleanupmergepublisher]
as
    exec sys.sp_MScleanupmergepublisher_internal
```

图 11.10　自动生成的 SQL 语句

（4）修改该段 SQL 语句并执行，即可完成修改该存储过程。

11.3.4 重命名存储过程

重新命名存储过程可以通过手动操作或执行 sp_rename 系统存储过程实现。

1. 手动操作重新命名存储过程

（1）打开 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库。

（2）展开对象资源管理器中“数据库”/“数据库名称”/“可编程性”/“存储过程”节点，鼠标右键单击需要重新命名的存储过程，在弹出的快捷菜单中选择“重命名”命令。例如，修改 db_2012 数据库中的 Proc_stu 存储过程名称，如图 11.11 所示。

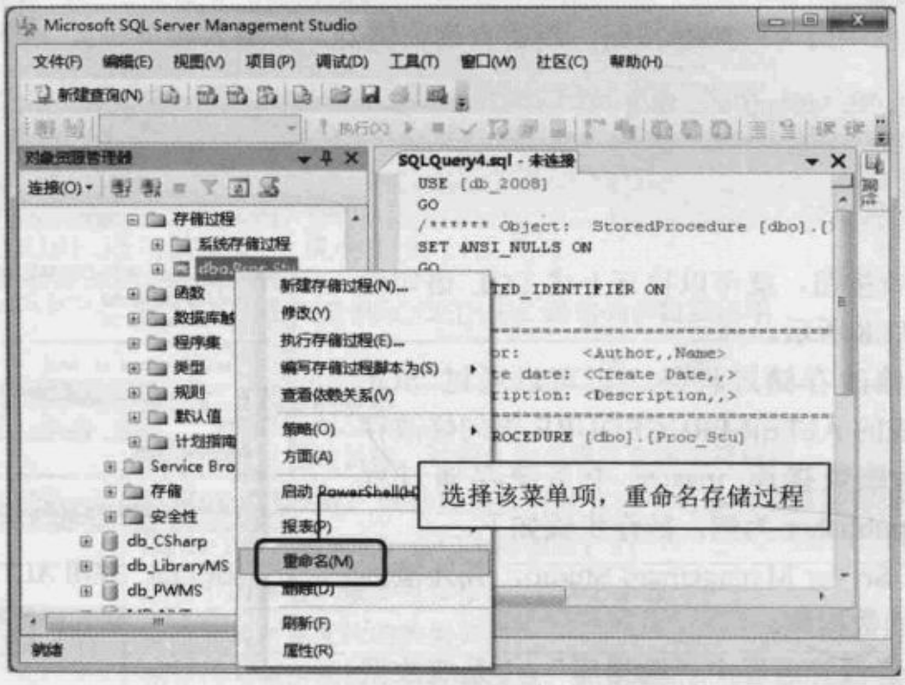


图 11.11 重命名存储过程

（3）此时，在存储过程名称的文本框中输入要修改的名称，即可重命名存储过程。

2. 执行 sp_rename 系统存储过程重新命名存储过程

sp_rename 系统存储过程可以在当前数据库中更改用户创建对象的名称。此对象可以是表、索引、列、别名数据类型或 Microsoft .NET Framework 公共语言运行时（CLR）用户定义类型。

语法如下：

```
sp_rename [ @objname = ] 'object_name' , [ @newname = ] 'new_name'
    [ , [ @objtype = ] 'object_type' ]
```

参数说明如下。

☑ [@objname =] 'object_name'：用户对象或数据类型的当前限定或非限定名称。如果要重命名

的对象是表中的列，则 object_name 的格式必须是 table.column。如果要重命名的对象是索引，则 object_name 的格式必须是 table.index。

☑ [@newname =] 'new_name'：指定对象的新名称。new_name 必须是名称的一部分，并且必须遵循标识符的规则。newname 的数据类型为 sysname，无默认值。

☑ [@objtype =] 'object_type'：要重命名的对象的类型。

使用 sp_rename 系统存储过程重新命名存储过程的步骤如下：

（1）打开 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库。

（2）选择需要重新命名的存储过程所在的数据库，单击工具栏中的 新建查询(N) 按钮，新建查询编辑器，输入执行 sp_rename 系统存储过程重新命名的 SQL 语句。

【例 11.7】 将 Proc_Stu 存储过程重新命名为“Proc_StuInfo”。（**实例位置：光盘\TM\sl\11\7**）

SQL 语句代码如下：

```
sp_rename 'Proc_Stu','Proc_StuInfo'
```

（3）单击 执行(X) 按钮，就可以执行上述 SQL 语句代码。结果如图 11.12 所示。

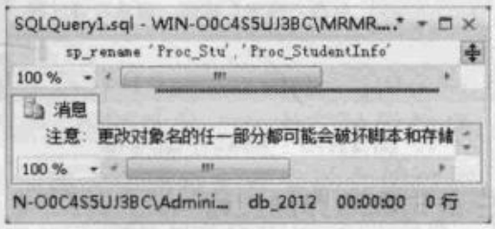


图 11.12　重新命名的存储过程

注意

更改对象名的任一部分都可能破坏脚本和存储过程。建议不要使用此语句来重命名存储过程、触发器、用户定义函数或视图；而是删除该对象，然后使用新名称重新创建该对象。

11.3.5　删除存储过程

数据库中某些不再应用的存储过程可以将其删除，这样可释放该存储过程所占的数据库空间。删除存储过程可以通过手动删除或执行 DROP PROCEDURE 语句实现。

1．手动删除存储过程

（1）打开 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库。

（2）展开对象资源管理器中“数据库”/“数据库名称”/“可编程性”/“存储过程”节点，鼠标右键单击要删除的存储过程，在弹出的快捷菜单中选择“删除”命令。

（3）在“删除对象”对话框中确认所删除的存储过程，单击“确定”按钮即可将该存储过程删除。

例如，删除 Proc_StuInfo 存储过程，如图 11.13 所示。

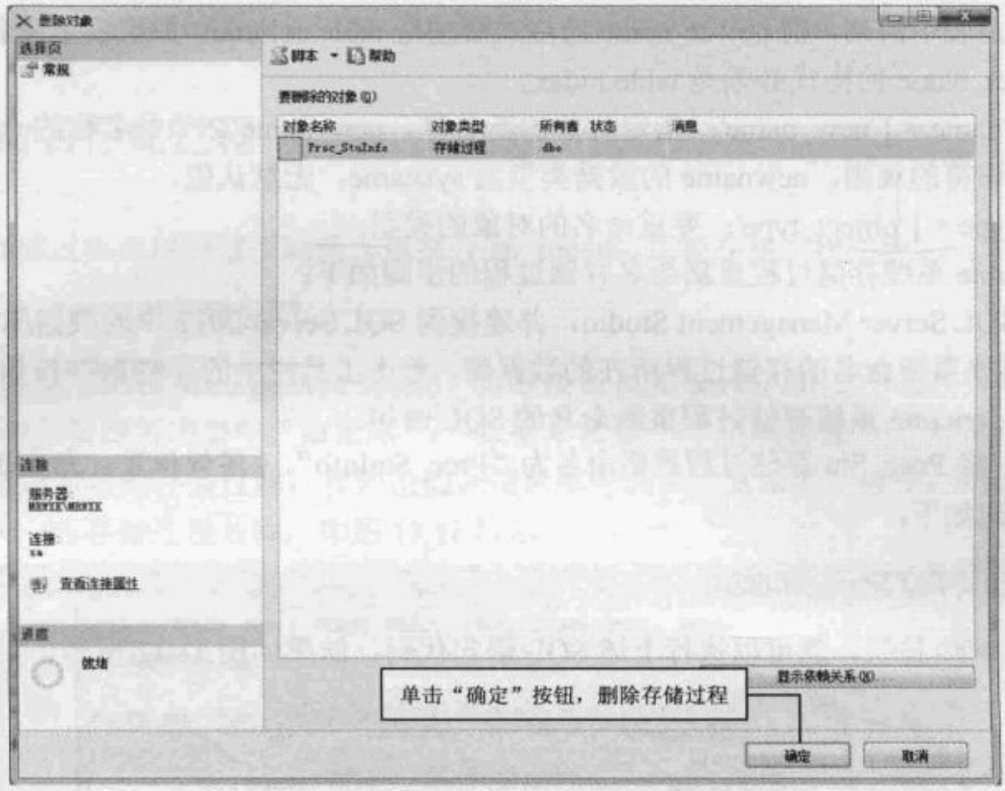


图 11.13　删除存储过程

2. 执行 DROP PROCEDURE 语句删除存储过程

DROP PROCEDURE 语句用来从当前数据库中删除一个或多个存储过程。语法如下：

```
DROP { PROC | PROCEDURE } { [ schema_name. ] procedure } [ ,...n ]
```

参数说明如下。

☑ schema_name：过程所属架构的名称。不能指定服务器名称或数据库名称。

☑ procedure：要删除的存储过程或存储过程组的名称。

执行 DROP PROCEDURE 语句删除存储过程的步骤如下：

（1）打开 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库。

（2）选择需要删除的存储过程所在的数据库，单击工具栏中的 新建查询(N) 按钮，在新建查询编辑器中输入执行 DROP PROCEDURE 语句删除存储过程的 SQL 语句。

【例 11.8】 删除名为 Proc_Student 的存储过程。（**实例位置：光盘\TM\sl\11\8**）

SQL 语句如下：

```
DROP PROCEDURE Proc_Student
```

（3）单击 执行(X) 按钮，就可以执行上述 SQL 语句代码，将 Proc_Student 存储过程删除。

注意

不可以删除正在使用的存储过程，否则 Microsoft SQL Server 2012 将在执行调用进程时显示一条错误消息。

11.4 小 结

本章介绍了存储过程的概念，以及创建和管理存储过程的方法。读者使用存储过程可以增强代码的重用性，创建存储过程后可以调用 Execute 语句执行存储过程或者设置其自动执行，另外，还可以查看、修改或者删除存储过程，使读者能够更容易地理解存储过程。

11.5 实践与练习

1．在存储过程中声明一个整型的变量@truc，并且通过 if 条件判断语句变量的值，如果变量等于“2”，则回滚事务，并且返回一个值 25，如果变量等于“0”则提交事务，并且返回一个值为“0”。（**答案位置：光盘\TM\sl\11\9**）

2．创建一个加密存储过程，使用 WITH ENCRYPION 子句对用户隐藏存储过程的文本，使文本内容更安全。使用 sp_helptext 系统存储过程获取关于加密存储的信息。（**答案位置：光盘\TM\sl\11\10**）

3．基于学生成绩信息表 score 创建一个带返回参数的存储过程，用来查询分数。在存储过程中声明两个变量，一个作为分数查询条件，一个作为返回值的接收参数。利用 exec 关键字执行该存储过程。查询结果会自动保存在变量@stuscore 中，利用此返回值判断分数等级。（**答案位置：光盘\TM\sl\11\11**）

第12章

触发器

（视频讲解：16分钟）

本章主要介绍如何使用触发器，主要包括触发器概述、创建触发器、修改触发器和删除触发器等内容。通过本章的学习，读者可以掌握使用管理器和Transact-SQL创建触发器，并应用触发器编写SQL语句，从而优化查询和提高数据访问速度。

通过阅读本章，您可以：

- 了解触发器的基本概念
- 掌握触发器的分类及创建
- 熟悉使用 sp_helptext 存储过程查看触发器
- 掌握如何使用 DML、DDL 操作触发器
- 掌握使用 sp_rename 重命名触发器
- 熟悉禁用和启用触发器
- 掌握如何删除触发器

12.1　触发器概述

视频讲解：光盘\TM\lx\12\触发器概述.mp4

12.1.1　触发器的概念

Microsoft SQL Server 提供两种主要机制来强制使用业务规则和数据完整性：约束和触发器。

触发器是一种特殊类型的存储过程，当指定表中的数据发生变化时触发器自动生效。它与表紧密相连，可以看作是表定义的一部分。触发器不能通过名称被直接调用，更不允许设置参数。

在 SQL Server 中一张表可以有多个触发器。用户可以使用 INSERT、UPDATE 或 DELETE 语句对触发器进行设置，也可以对一张表上的特定操作设置多个触发器。触发器可以包含复杂的 Transact-SQL 语句。不论触发器所进行的操作有多复杂，触发器都只作为一个独立的单元被执行，被看作是一个事务。如果在执行触发器的过程中发生了错误，则整个事务将会自动回滚。

12.1.2　触发器的优点

触发器的优点表现在以下几个方面：

（1）触发器自动执行，对表中的数据进行修改后，触发器立即被激活。

（2）为了实现复杂的数据库更新操作，触发器可以调用一个或多个存储过程，甚至可以通过调用外部过程（不是数据库管理系统本身）完成相应的操作。

（3）触发器能够实现比 CHECK 约束更为复杂的数据完整性约束。在数据库中，为了实现数据完整性约束，可以使用 CHECK 约束或触发器。CHECK 约束不允许引用其他表中的列来完成检查工作，而触发器可以引用其他表中的列。它更适合在大型数据库管理系统中用来约束数据的完整性。

（4）触发器可以检测数据库内的操作，从而取消了数据库未经许可的更新操作，使数据库修改、更新操作更安全，数据库的运行也更稳定。

（5）触发器能够对数据库中的相关表实现级联更改。触发器是基于一个表创建的，但是可以针对多个表进行操作，实现数据库中相关表的级联更改。

（6）一个表中可以同时存在 3 个不同操作的触发器（INSERT、UPDATE 和 DELETE）。

12.1.3　触发器的种类

SQL Server 包括 3 种常规类型的触发器：DML 触发器、DDL 触发器和登录触发器。

当数据库中发生数据操作语言（DML）事件时将调用 DML 触发器。DML 事件包括在指定表或视图中修改数据的 INSERT 语句、UPDATE 语句或 DELETE 语句。DML 触发器可以查询其他表，还可以

包含复杂的 Transact-SQL 语句。

可以设计以下类型的 DML 触发器。

- ☑ AFTER 触发器：在执行了 INSERT、UPDATE 或 DELETE 语句操作之后执行 AFTER 触发器。
- ☑ INSTEAD OF 触发器：执行 INSTEAD OF 触发器代替通常的触发动作。还可为带有一个或多个基表的视图定义 INSTEAD OF 触发器，而这些触发器能够扩展视图可支持的更新类型。
- ☑ CLR 触发器：CLR 触发器可以是 AFTER 触发器或 INSTEAD OF 触发器。CLR 触发器还可以是 DDL 触发器。CLR 触发器将执行在托管代码（在.NET Framework 中创建并在 SQL Server 中上载的程序集的成员）中编写的方法，而不用执行 Transact-SQL 存储过程。

DDL 触发器是一种特殊的触发器，它在响应数据定义语言（DDL）语句时触发，可以用于在数据库中执行管理任务，例如，审核以及规范数据库操作。

登录触发器将为响应 LOGON 事件而激发存储过程。与 SQL Server 实例建立用户会话时将引发此事件。登录触发器将在登录的身份验证阶段完成之后且用户会话实际建立之前激发。可以使用登录触发器来审核和控制服务器会话，例如，通过跟踪登录活动、限制 SQL Server 的登录名或限制特定登录名的会话数。

12.2 创建触发器

视频讲解：光盘\TM\lx\12\创建触发器.mp4

创建 DML、DDL 触发器和登录触发器可以通过执行 CREATE TRIGGER 语句实现。但在使用该语句创建 DML、DDL 触发器和登录触发器时，其语法存在差异。本节讲解 CREATE TRIGGER 语句与使用该语句创建 DML、DDL 触发器和登录触发器。

12.2.1 创建 DML 触发器

如果用户要通过数据操作语言（DML）事件编辑数据，则执行 DML 触发器。DML 事件是针对表或视图的 INSERT、UPDATE 或 DELETE 语句。

创建 DML 触发器的语法如下：

```
CREATE TRIGGER [ schema_name . ]trigger_name
ON { table | view }
[ WITH <dml_trigger_option> [ ,...n ] ]
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
[ WITH APPEND ]
[ NOT FOR REPLICATION ]
AS { sql_statement   [ ; ] [ ,...n ] | EXTERNAL NAME <method specifier [ ; ] > }
<dml_trigger_option> ::=
    [ ENCRYPTION ]
    [ EXECUTE AS Clause ]
```

```
<method_specifier> ::=
    assembly_name.class_name.method_name
```

创建 DML 触发器的参数及说明如表 12.1 所示。

表 12.1　创建 DML 触发器的参数及说明

参　数	描　述
schema_name	DML 触发器所属架构的名称。DML 触发器的作用域是为其创建该触发器的表或视图的架构
trigger_name	触发器的名称。trigger_name 必须遵循标识符规则，但 trigger_name 不能以#或##开头
table | view	对其执行 DML 触发器的表或视图，有时称为触发器表或触发器视图。可以根据需要指定表或视图的完全限定名称。视图只能被 INSTEAD OF 触发器引用。不能对局部或全局临时表定义 DML 触发器
FOR | AFTER	AFTER 指定 DML 触发器仅在触发 SQL 语句中指定的所有操作都已成功执行时才被触发
INSTEAD OF	指定执行 DML 触发器而不是触发 SQL 语句，因此，其优先级高于触发语句的操作
{ [INSERT] [,] [UPDATE] [,] [DELETE] }	指定数据修改语句，这些语句可在 DML 触发器对此表或视图进行尝试时激活该触发器。必须至少指定一个选项
WITH APPEND	指定应该再添加一个现有类型的触发器
NOT FOR REPLICATION	指示当复制代理修改涉及触发器的表时，不应执行触发器
sql_statement	触发条件和操作。触发器条件指定其他标准，用于确定尝试的 DML、DDL 或 LOGON 事件是否导致执行触发器操作
EXECUTE AS	指定用于执行该触发器的安全上下文
<method_specifier>	对于 CLR 触发器，指定程序集与触发器绑定的方法。该方法不能带有任何参数，并且必须返回空值

【例 12.1】 为员工表 employee3 创建 DML 触发器，当向该表中插入数据时给出提示信息。（**实例位置：光盘\TM\sl\12\1**）

设计步骤如下：

（1）打开 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库。

（2）单击工具栏中的 新建查询(N) 按钮，新建查询编辑器，输入如下 SQL 语句代码。

```
CREATE TRIGGER T_DML_Emp3                    --创建触发器 T_DML_Emp3
ON employee3                                 --依赖于表 employee3
AFTER INSERT                                 --执行插入语句之后
AS
RAISERROR ('正在向表中插入数据', 16, 10);      --提示信息
```

（3）单击 执行(X) 按钮，就可以执行上述 SQL 语句代码。创建名称为 T_DML_Emp3 的 DML 触发器。

每次对 employee3 表的数据进行添加时，都会显示如图 12.1 所示的消息内容。

图 12.1　向表中插入数据时给出的信息

12.2.2　创建 DDL 触发器

DDL 触发器用于响应各种数据定义语言（DDL）事件。这些事件主要对应于 Transact-SQL CREATE、ALTER 和 DROP 语句，以及执行类似 DDL 操作的某些系统存储过程。

创建 DDL 触发器的语法如下：

```
CREATE TRIGGER trigger_name
ON { ALL SERVER | DATABASE }
[ WITH <ddl_trigger_option> [ ,...n ] ]
{ FOR | AFTER } { event_type | event_group } [ ,...n ]
AS { sql_statement   [ ; ] [ ,...n ] | EXTERNAL NAME < method specifier >   [ ; ] }
<ddl_trigger_option> ::=
    [ ENCRYPTION ]
    [ EXECUTE AS Clause ]
<method_specifier> ::=
    assembly_name.class_name.method_name
```

创建 DDL 触发器的参数及说明如表 12.2 所示。

表 12.2　创建 DDL 触发器的参数及说明

参　数	描　述
trigger_name	触发器的名称。trigger_name 必须遵循标识符规则，但 trigger_name 不能以#或##开头
ALL SERVER	将 DDL 或登录触发器的作用域应用于当前服务器
DATABASE	将 DDL 触发器的作用域应用于当前数据库
FOR | AFTER	AFTER 指定 DML 触发器仅在触发 SQL 语句中指定的所有操作都已成功执行时才被触发
event_type	执行之后将导致激发 DDL 触发器的 Transact-SQL 事件的名称。DDL 事件中列出了 DDL 触发器的有效事件
event_group	预定义的 Transact-SQL 事件分组的名称
sql_statement	触发条件和操作。触发器条件指定其他标准，用于确定尝试的 DML、DDL 或 LOGON 事件是否导致执行触发器操作
<method_specifier>	对于 CLR 触发器，指定程序集与触发器绑定的方法

【例 12.2】 为数据库 db_2012 创建 DDL 触发器，防止用户对表进行删除或修改等操作。（**实例位置：光盘\TM\sl\12\2**）

设计步骤如下：

（1）打开 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库。

（2）单击工具栏中的 新建查询(N) 按钮，新建查询编辑器，输入如下 SQL 语句代码。

```
CREATE TRIGGER T_DDL_DATABASE                    --创建 DDL 触发器
ON DATABASE                                      --将该触发器应用于当前数据库
FOR DROP_TABLE, ALTER_TABLE                      --对表修改时，提示信息
AS
PRINT '只有“T_DDL_DATABASE”触发器无效时，才可以删除或修改表。'
ROLLBACK                                         --回滚操作
```

（3）单击 执行(X) 按钮，就可以执行上述 SQL 语句代码。创建名称为 T_DDL_DATABASE 的 DDL 触发器。

创建完该触发器后，当对数据库中的表进行修改与删除等操作时，都会提示“只有 T_DDL_DATABASE 触发器无效时，才可以删除或修改表。”信息，并将删除后修改操作进行回滚。显示信息如图 12.2 所示。

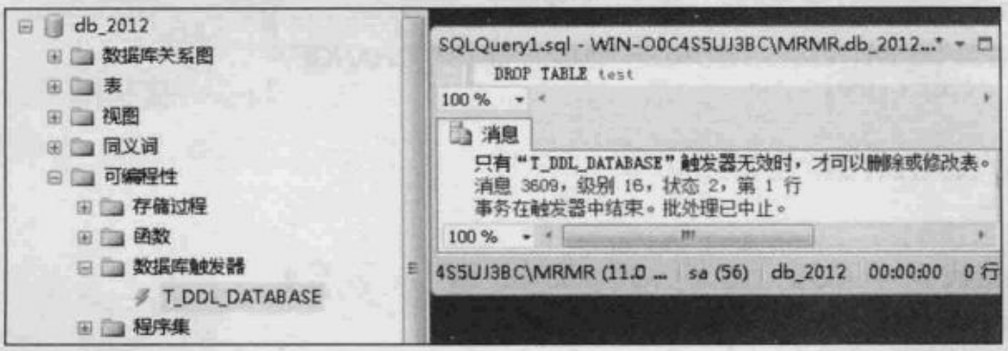


图 12.2　对数据库中表进行修改与删除等操作时显示的消息

12.2.3　创建登录触发器

登录触发器在遇到 LOGON 事件时触发。LOGON 事件是在建立用户会话时引发的。触发器可以由 Transact-SQL 语句直接创建，也可以由程序集方法创建，这些方法是在 Microsoft .NET Framework 公共语言运行时（CLR）中创建并上载到 SQL Server 实例的。SQL Server 允许为任何特定语句创建多个触发器。

创建登录触发器的语法如下：

```
CREATE TRIGGER trigger_name
ON ALL SERVER
[ WITH <logon_trigger_option> [ ,...n ] ]
{ FOR | AFTER } LOGON
AS { sql_statement  [ ; ] [ ,...n ] | EXTERNAL NAME < method specifier >  [ ; ] }
<logon_trigger_option> ::=
    [ ENCRYPTION ]
    [ EXECUTE AS Clause ]
```

```
<method_specifier> ::=
    assembly_name.class_name.method_name
```

创建登录触发器的参数及说明如表 12.3 所示。

表 12.3 创建登录触发器的参数及说明

参　数	描　述
trigger_name	触发器的名称。trigger_name 必须遵循标识符规则，但 trigger_name 不能以#或##开头
ALL SERVER	将 DDL 或登录触发器的作用域应用于当前服务器
FOR | AFTER	AFTER 指定 DML 触发器仅在触发 SQL 语句中指定的所有操作都已成功执行时才被触发
sql_statement	触发条件和操作。触发器条件指定其他标准，用于确定尝试的 DML、DDL 或 LOGON 事件是否导致执行触发器操作
<method_specifier>	对于 CLR 触发器，指定程序集与触发器绑定的方法

【例 12.3】 创建一个登录触发器，该触发器拒绝 mr 登录名的成员登录 SQL Server。（**实例位置：光盘\TM\sl\12\3**）

SQL 语句如下：

```
USE master;
GO
CREATE LOGIN TM WITH PASSWORD = 'TMsoft' MUST_CHANGE,
    CHECK_EXPIRATION = ON;
GO
GRANT VIEW SERVER STATE TO TM;
GO
CREATE TRIGGER connection_limit_trigger
ON ALL SERVER WITH EXECUTE AS 'mr'
FOR LOGON
AS
BEGIN
IF ORIGINAL_LOGIN()= 'mr' AND
    (SELECT COUNT(*) FROM sys.dm_exec_sessions
            WHERE is_user_process = 1 AND
                original_login_name = 'mr') > 1
    ROLLBACK;
END;
```

设计步骤如下：

（1）打开 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库。

（2）单击工具栏中的 新建查询(N) 按钮，新建查询编辑器，输入例 12.3 中的 SQL 语句。

（3）单击 执行(X) 按钮，就可以执行上述 SQL 语句代码。创建名称为 connection_limit_trigger 的登录触发器。

登录触发器与 DML 触发器、DDL 触发器所存储的位置不同，其存储位置为对象资源管理器中“服务器对象”/“触发器”。登录触发器 connection_limit_trigger 中的 mr 为登录到 SQL Server 中的登录名。触发器及 mr 所在的位置如图 12.3 所示。

创建完该触发器后，当以 mr 的登录名登录 SQL Server 时，就会显示如图 12.4 所示的提示信息。

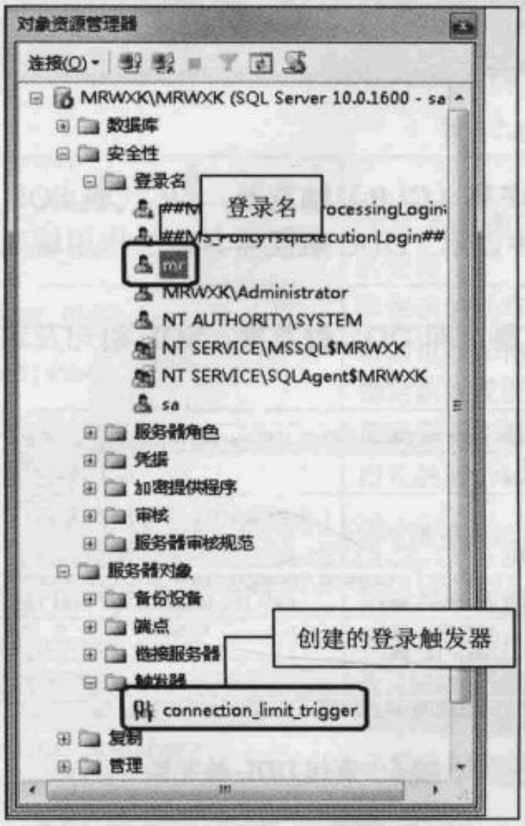


图 12.3　触发器及 mr 所在的位置

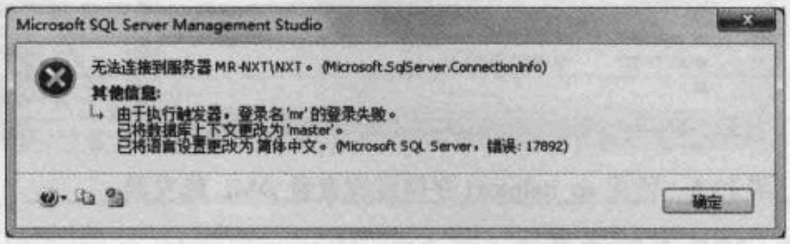


图 12.4　TM 的登录名登录 SQL Server 时提示的信息

12.3　管理触发器

视频讲解：光盘\TM\lx\12\管理触发器.mp4

触发器的查看、修改、重命名与删除等操作都可以使用 SQL Server Management Studio 管理工具实现。本节讲解通过 SQL Server Management Studio 管理工具查看、修改、重命名与删除触发器。

12.3.1　查看触发器

查看触发器与查看存储过程相同，同样可以使用 sp_helptext 存储过程与 sys.sql_modules 视图查看触发器。

1. 使用 sp_helptext 存储过程查看触发器

sp_helptext 存储过程可以查看架构范围内的触发器，非架构范围内的触发器是不能用此存储过程查看的，如 DDL 触发器、登录触发器。

【例 12.4】 sp_helptext 存储过程查看 DML 触发器，SQL 语句及运行结果如图 12.5 所示。（**实例位置：光盘\TM\sl\12\4**）

SQL 语句如下：

```
USE db_2012
EXEC sp_helptext 'T_DML_Emp3'
```

2. 获取数据库中触发器的信息

每个类型为 TR 或 TA 的触发器对象对应一行，TA 代表程序集（CLR）触发器，TR 代表 SQL 触发器。DML 触发器名称在架构范围内，因此，可在 sys.objects 中显示。DDL 触发器名称的作用域取决于父实体，只能在对象目录视图中显示。

【例 12.5】 在 db_2012 数据库中，查找类型为 TR 的触发器，即 DDL 触发器，SQL 语句及运行结果如图 12.6 所示。（**实例位置：光盘\TM\sl\12\5**）

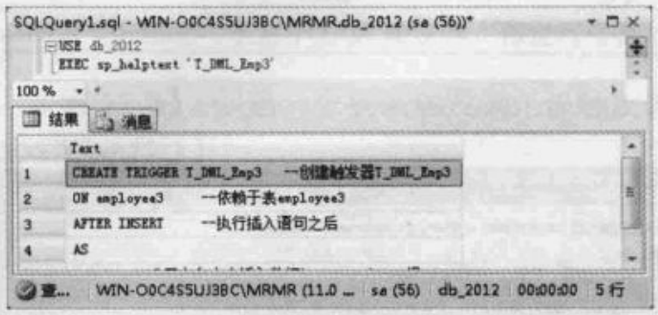


图 12.5 使用 sp_helptext 存储过程查看 DML 触发器

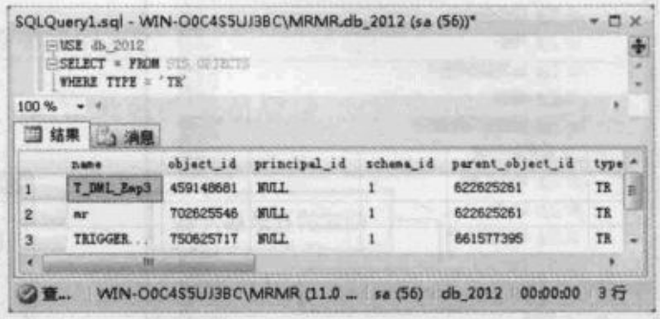


图 12.6 查找 DDL 触发器

SQL 语句如下：

```
USE db_2012
SELECT * FROM sys.objects
WHERE TYPE='TR'
```

12.3.2 修改触发器

修改触发器可以通过 ALTER TRIGGER 语句实现，下面分别对修改 DML 触发器、修改 DDL 触发器、修改登录触发器进行介绍。

1. 修改 DML 触发器

修改 DML 触发器的语法如下：

```
ALTER TRIGGER schema_name.trigger_name
ON ( table | view )
[ WITH <dml_trigger_option> [ ,...n ] ]
( FOR | AFTER | INSTEAD OF )
{ [ DELETE ] [ , ] [ INSERT ] [ , ] [ UPDATE ] }
[ NOT FOR REPLICATION ]
AS { sql_statement [ ; ] [ ...n ] | EXTERNAL NAME <method specifier> [ ; ] }
<dml_trigger_option> ::=
    [ ENCRYPTION ]
    [ <EXECUTE AS Clause> ]
```

```
<method_specifier> ::=
        assembly_name.class_name.method_name
```

修改 DML 触发器的参数及说明如表 12.4 所示。

表 12.4　修改 DML 触发器的参数及说明

参　数	描　述
schema_name	DML 触发器所属架构的名称。DML 触发器的作用域是为其创建该触发器的表或视图的架构
trigger_name	要修改的现有触发器
table | view	对其执行 DML 触发器的表或视图，有时称为触发器表或触发器视图。可以根据需要指定表或视图的完全限定名称
AFTER	指定只有在触发 SQL 语句成功执行后，才会激发触发器
INSTEAD OF	指定执行 DML 触发器而不是触发 SQL 语句，因此，其优先级高于触发语句的操作
{ [DELETE] [,] [INSERT] [,] [UPDATE] }	指定数据修改语句在试图修改表或视图时，激活 DML 触发器。必须至少指定一个选项
NOT FOR REPLICATION	指示当复制代理修改涉及触发器的表时，不应执行触发器
sql_statement	触发条件和操作
EXECUTE AS	指定用于执行该触发器的安全上下文
<method_specifier>	对于 CLR 触发器，指定程序集与触发器绑定的方法。该方法不能带有任何参数，并且必须返回空值

【例 12.6】 使用 ALTER TRIGGER 语句修改 DML 触发器 T_DML_Emp3，当向该表中插入、修改或删除数据时给出提示信息。（**实例位置：光盘\TM\sl\12\6**）

SQL 语句如下：

```
ALTER TRIGGER T_DML_Emp3
ON employee3
AFTER INSERT,UPDATE,DELETE
AS
RAISERROR ('正在向表中插入、修改或删除数据', 16, 10);
```

运行结果如图 12.7 所示。

图 12.7　使用 ALTER TRIGGER 修改 DML 触发器

2. 修改 DDL 触发器

修改 DDL 触发器的语法如下：

```
ALTER TRIGGER trigger_name
ON { DATABASE | ALL SERVER }
```

```
[ WITH <ddl_trigger_option> [ ,...n ] ]
{ FOR | AFTER } { event_type [ ,...n ] | event_group }
AS { sql_statement [ ; ] | EXTERNAL NAME <method specifier>
[ ; ] }
 }
<ddl_trigger_option> ::=
    [ ENCRYPTION ]
    [ <EXECUTE AS Clause> ]
<method_specifier> ::=
        assembly_name.class_name.method_name
```

修改 DDL 触发器的参数及说明如表 12.5 所示。

表 12.5　修改 DDL 触发器的参数及说明

参　数	描　述
trigger_name	要修改的现有触发器
DATABASE	将 DDL 触发器的作用域应用于当前数据库
ALL SERVER	将 DDL 或登录触发器的作用域应用于当前服务器
AFTER	指定只有在触发 SQL 语句成功执行后，才会激发触发器
event_type	执行之后将导致激发 DDL 触发器的 Transact-SQL 事件的名称
event_group	预定义的 Transact-SQL 事件分组的名称
sql_statement	触发条件和操作
EXECUTE AS	指定用于执行该触发器的安全上下文
<method_specifier>	对于 CLR 触发器，指定程序集与触发器绑定的方法。该方法不能带有任何参数，并且必须返回空值

【例 12.7】 使用 ALTER TRIGGER 语句修改 DDL 触发器 T_DDL_DATABASE，防止用户修改数据。（**实例位置：光盘\TM\sl\12\7**）

SQL 语句如下：

```
ALTER TRIGGER T_DDL_DATABASE                    --修改触发器
ON DATABASE                                     --应用于当前数据库
FOR ALTER_TABLE
AS
RAISERROR ('只有“T_DDL_DATABASE”触发器无效时，才可以修改表。', 16, 10)
ROLLBACK                                        --回滚事务
```

3．修改登录触发器

修改登录触发器的语法如下：

```
ALTER TRIGGER trigger_name
ON ALL SERVER
[ WITH <logon_trigger_option> [ ,...n ] ]
{ FOR | AFTER } LOGON
AS { sql_statement  [ ; ] [ ,...n ] | EXTERNAL NAME < method specifier >  [ ; ] }
<logon_trigger_option> ::=
    [ ENCRYPTION ]
    [ EXECUTE AS Clause ]
```

```
<method_specifier> ::=
    assembly_name.class_name.method_name
```

修改登录触发器的参数及说明如表 12.6 所示。

表 12.6　修改登录触发器的参数及说明

参　数	描　述
trigger_name	要修改的现有触发器
ALL SERVER	将 DDL 或登录触发器的作用域应用于当前服务器
AFTER	指定只有在触发 SQL 语句成功执行后，才会激发触发器
sql_statement	触发条件和操作
EXECUTE AS	指定用于执行该触发器的安全上下文
<method_specifier>	指定要与触发器绑定的程序集的方法

【例 12.8】 使用 ALTER TRIGGER 语句修改登录触发器 connection_limit_trigger，将用户名修改为 nxt，如果在此登录名下已运行 3 个用户会话，拒绝 nxt 登录到 SQL Server。（**实例位置：光盘\TM\sl\12\8**）

SQL 语句如下：

```
ALTER TRIGGER connection_limit_trigger
ON ALL SERVER WITH EXECUTE AS 'nxt'
FOR LOGON
AS
BEGIN
IF ORIGINAL_LOGIN()= 'nxt' AND
    (SELECT COUNT(*) FROM sys.dm_exec_sessions
            WHERE is_user_process = 1 AND
                original_login_name = 'nxt') > 3
    ROLLBACK;
END;
```

12.3.3　重命名触发器

重命名触发器可以使用 sp_rename 系统存储过程实现。使用 sp_rename 系统存储过程重命名触发器与重命名存储过程相同。但是使用该系统存储过程重命名触发器，不会更改 sys.sql_modules 类别视图的 definition（用于定义此模块的 SQL 文本）列中相应对象名的名称，所以建议用户不要使用该系统存储过程重命名触发器，而是删除该触发器，然后使用新名称重新创建该触发器。

【例 12.9】 使用 sp_rename 将触发器 T_DML_Emp3 重命名为 T_DML_3。（**实例位置：光盘\TM\sl\12\9**）

SQL 语句如下：

```
sp_rename  'T_DML_Emp3','T_DML_3'
```

12.3.4　禁用和启用触发器

当不再需要某个触发器时，可将其禁用或删除。禁用触发器不会删除该触发器，该触发器仍然作

为对象存在于当前数据库中。但是，当执行任意 INSERT、UPDATE 或 DELETE 语句（在其上对触发器进行了编程）时，触发器将不会激发。已禁用的触发器可以被重新启用。启用触发器会以最初创建它时的方式将其激发。默认情况下，创建触发器后会启用触发器。

1. 禁用触发器

使用 DISABLE TRIGGER 语句禁用触发器，其语法如下：

```
DISABLE TRIGGER { [ schema_name . ] trigger_name [ ,...n ] | ALL }
ON { object_name | DATABASE | ALL SERVER } [ ; ]
```

参数说明如下。

- ☑ schema_name：触发器所属架构的名称。
- ☑ trigger_name：要禁用的触发器的名称。
- ☑ ALL：指示禁用在 ON 子句作用域中定义的所有触发器。

注意

SQL Server 在为合并复制发布的数据库中创建触发器。在已发布数据库中指定 ALL 可禁用这些触发器，这样会中断复制。在指定 ALL 之前，请验证没有为合并复制发布当前数据库。

- ☑ object_name：要对其创建要执行的 DML 触发器 trigger_name 的表或视图的名称。
- ☑ DATABASE：对于 DDL 触发器，指示所创建或修改的 trigger_name 将在数据库范围内执行。
- ☑ ALL SERVER：对于 DDL 触发器，指示所创建或修改的 trigger_name 将在服务器范围内执行。ALL SERVER 也适用于登录触发器。

【例 12.10】 使用 DISABLE TRIGGER 语句禁用 DML 触发器 T_DML_3。（**实例位置：光盘\TM\sl\12\10**）

SQL 语句如下：

```
DISABLE TRIGGER T_DML_3 ON employee3
```

禁用后触发器的状态如图 12.8 所示。

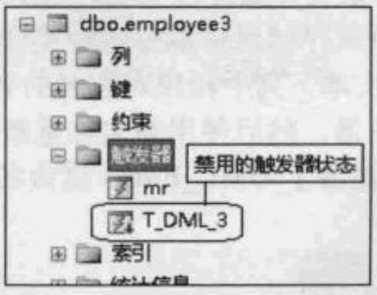


图 12.8　禁用触发器的状态

【例 12.11】 使用 DISABLE TRIGGER 语句禁用 DDL 触发器 T_DDL_DATABASE。（**实例位置：光盘\TM\sl\12\11**）

SQL 语句如下：

```
DISABLE TRIGGER T_DDL_DATABASE ON DATABASE
```

【例 12.12】 使用 DISABLE TRIGGER 语句禁用登录触发器 connection_limit_trigger。（**实例位置：光盘\TM\sl\12\12**）

SQL 语句如下：

```
DISABLE TRIGGER  connection_limit_trigger ON ALL SERVER
```

2. 启用触发器

启用触发器并不是重新创建它。已禁用的 DDL、DML 或登录触发器可以通过执行 ENABLE TRIGGER 语句重新起用。语法如下：

```
ENABLE TRIGGER { [ schema_name . ] trigger_name [ ,...n ] | ALL }
ON { object_name | DATABASE | ALL SERVER } [ ; ]
```

启用触发器的参数及说明如表 12.7 所示。

表 12.7　启用触发器的参数及说明

参　数	描　述
schema_name	触发器所属架构的名称。不能为 DDL 或登录触发器指定 schema_name
trigger_name	要启用的触发器的名称
ALL	指示启用在 ON 子句作用域中定义的所有触发器
object_name	要对其创建要执行的 DML 触发器 trigger_name 的表或视图的名称
DATABASE	对于 DDL 触发器，指示所创建或修改的 trigger_name 将在数据库范围内执行
ALL SERVER	对于 DDL 触发器，指示所创建或修改的 trigger_name 将在服务器范围内执行。ALL SERVER 也适用于登录触发器

【例 12.13】 使用 ENABLE TRIGGER 语句启用 DML 触发器 T_DML_3。（**实例位置：光盘\TM\sl\12\13**）

SQL 语句如下：

```
ENABLE  TRIGGER T_DML_3 on employee3
```

【例 12.14】 使用 ENABLE TRIGGER 语句启用 DDL 触发器 T_DDL_DATABASE。（**实例位置：光盘\TM\sl\12\14**）

SQL 语句如下：

```
ENABLE TRIGGER T_DDL_DATABASE ON DATABASE
```

【例 12.15】 使用 ENABLE TRIGGER 语句启用登录触发器 connection_limit_trigger。（**实例位置：光盘\TM\sl\12\15**）

SQL 语句如下：

```
ENABLE TRIGGER  connection_limit_trigger ON ALL SERVER
```

启用后触发器的状态如图 12.9 所示。

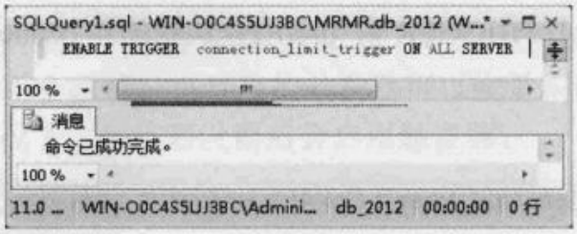


图 12.9　启用后触发器的状态

12.3.5　删除触发器

删除触发器是将触发器对象从当前数据库中永久地删除。通过执行 DROP TRIGGER 语句可以将 DML、DDL 或登录触发器删除。也可以通过操作 SQL Server Management Studio 手动删除 DML、DDL 或登录触发器。

1. DROP TRIGGER 语句删除触发器

DROP TRIGGER 语句可以从当前数据库中删除一个或多个 DML、DDL 或登录触发器。

（1）删除 DML 触发器。

删除 DML 触发器的语法如下：

```
DROP TRIGGER schema_name.trigger_name [ ,...n ] [ ; ]
```

参数说明如下。

☑ schema_name：DML 触发器所属架构的名称。

☑ trigger_name：要删除的触发器的名称

【例 12.16】 使用 DROP TRIGGER 语句删除 DML 触发器 T_DML_3。（**实例位置：光盘\TM\sl\12\16**）

SQL 语句如下：

```
DROP TRIGGER  T_DML_3
```

（2）删除 DDL 触发器。

删除 DDL 触发器的语法如下：

```
DROP TRIGGER trigger_name [ ,...n ]
ON { DATABASE | ALL SERVER }
 [ ; ]
```

参数说明如下。

☑ trigger_name：要删除的触发器的名称。

☑ DATABASE：指示 DDL 触发器的作用域应用于当前数据库。如果在创建或修改触发器时也指定了 DATABASE，则必须指定 DATABASE。

☑ ALL SERVER：指示 DDL 触发器的作用域应用于当前服务器。如果在创建或修改触发器时也指定了 ALL SERVER，则必须指定 ALL SERVER。ALL SERVER 也适用于登录触发器。

【例 12.17】 使用 DROP TRIGGER 语句删除 DDL 触发器 T_DDL_DATABASE。（**实例位置：光盘\TM\sl\12\17**）

SQL 语句如下：

```
DROP TRIGGER  T_DDL_DATABASE  ON DATABASE
```

（3）删除登录触发器。

删除登录触发器的语法如下：

```
DROP TRIGGER trigger_name [ ,...n ]
ON ALL SERVER
```

参数说明如下。

☑ trigger_name：要删除的触发器的名称。

☑ ALL SERVER：也适用于登录触发器。

【例 12.18】 使用 DROP TRIGGER 语句删除登录触发器 connection_limit_trigger。（**实例位置：光盘\TM\sl\12\18**）

SQL 语句如下：

```
DROP TRIGGER  connection_limit_trigger ON ALL SERVER
```

2. SQL Server Management Studio 手动删除触发器

手动删除触发器的步骤如下：

（1）打开 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库。

（2）展开“对象资源管理器”中触发器所在位置。例如，要删除创建在 db_2012 数据库中的 T_DDL_DATABASE 触发器，则展开如图 12.10 所示的树状结构。

（3）鼠标右键单击要删除的触发器，在弹出的快捷菜单中选择“删除”命令，打开“删除对象”窗口，如图 12.11 所示。

图 12.10　展开触发器所在的位置

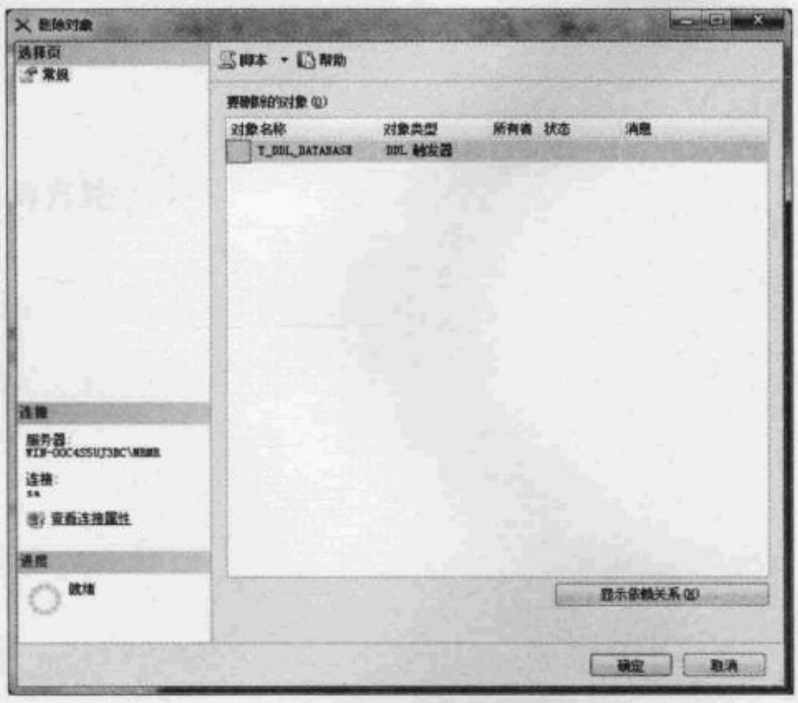


图 12.11　“删除对象”窗口

（4）在“删除对象”窗口中确认所删除的触发器，单击“确定”按钮即可将该触发器删除。

12.4 小　　结

本章介绍了触发器的概念，以及创建和管理触发器的方法。读者使用触发器可以在操作数据的同时触发指定的事件从而维护数据完整性。触发器可分为 DML 触发器、DDL 触发器和登录触发器，可以使用企业管理器或者 Transact-SQL 语句对触发器进行管理。通过本章的学习，希望读者能更深入地熟悉触发器的使用。

12.5 实践与练习

1．创建一个递归触发器，只允许一次删除一条记录，以实现对删除记录条数的控制。该触发器中设有变量@rowcount 用以保存删除记录的条数，通过检测该变量的值来判断是否执行删除操作。（**答案位置：光盘\TM\sl\12\19**）

2．创建名称为 mr 的触发器，每次对 employee3 表的数据进行添加时，都会显示“正在向表中插入数据”。（**答案位置：光盘\TM\sl\12\20**）

3．为 Student 表创建一个触发器，实现对插入操作的约束。当插入的学生年龄不在 10～50 之间时，不执行插入，打印错误提示，并回滚事务。（**答案位置：光盘\TM\sl\12\21**）

第13章

游标的使用

（视频讲解：13分钟）

游标是取用一组数据并能够一次与一个单独的数据进行交互的方法，然而，不能通过在整个行集中修改或者选取数据来获得所需要的结果。本章将对游标的使用进行详细讲解。

通过阅读本章，您可以：

- 掌握游标的概念
- 了解游标的类型
- 掌握游标的基本操作
- 了解游标系统存储过程
- 掌握使用系统过程查看游标的方法

13.1 游标的概述

视频讲解：光盘\TM\lx\13\游标的概述.mp4

游标是取用一组数据并能够一次与一个单独的数据进行交互的方法。关系数据库中的操作会对整个行集起作用。由 SELECT 语句返回的行集包括满足该语句的 WHERE 子句中条件的所有行。这种由语句返回的完整行集称为结果集。应用程序，特别是交互式联机应用程序，并不总能将整个结果集作为一个单元来有效地处理。这些应用程序需要一种机制以便每次处理一行或一部分行。游标就是提供这种机制并对结果集的一种扩展。

游标通过以下方式来扩展结果处理：

- ☑ 允许定位在结果集的特定行。
- ☑ 从结果集的当前位置检索一行或一部分行。
- ☑ 支持对结果集中当前位置的行进行数据修改。
- ☑ 为由其他用户对显示在结果集中的数据库数据所做的更改提供不同级别的可见性支持。
- ☑ 提供脚本、存储过程和触发器中用于访问结果集中的数据的 Transact-SQL 语句。

游标可以定在该单元中的特定行，从结果集的当前行检索一行或多行。可以对结果集当前行做修改。一般不使用游标，但是需要逐条处理数据时，游标显得十分重要。

13.1.1 游标的实现

游标提供了一种从表中检索数据并进行操作的灵活手段，游标主要用在服务器上，处理由客户端发送给服务器端的 SQL 语句，或是批处理、存储过程、触发器中的数据处理请求。游标的优点在于它可以定位到结果集中的某一行，并可以对该行数据执行特定操作，为用户在处理数据的过程中提供了很大方便。一个完整的游标由 5 部分组成，并且这 5 个部分应符合下面的顺序。

（1）声明游标。

（2）打开游标。

（3）从一个游标中查找信息。

（4）关闭游标。

（5）释放游标。

13.1.2 游标的类型

SQL Server 提供了 4 种类型的游标：静态游标、动态游标、只进游标和键集驱动的游标。这些游标的检测结果集变化的能力和内存占用的情况都有所不同，数据源没有办法通知游标当前提取行的更

改。游标检测这些变化的能力也受事务隔离级别的影响。

1．静态游标

静态游标的完整结果集在游标打开时建立在 tempdb 中。静态游标总是按照游标打开时的原样显示结果集。静态游标在滚动期间很少或根本检测不到变化，虽然它在 tempdb 中存储了整个游标，但消耗的资源很少。尽管动态游标使用 tempdb 的程度最低，在滚动期间它能够检测到所有变化，但消耗的资源也更多。键集驱动游标介于二者之间，它能检测到大部分的变化，但比动态游标消耗更少的资源。

2．动态游标

动态游标与静态游标相对。当滚动游标时，动态游标反映结果集中所做的所有更改。结果集中的行数据值、顺序和成员在每次提取时都会改变。所有用户做的全部 UPDATE、INSERT 和 DELETE 语句均通过游标可见。

3．只进游标

只进游标不支持滚动，它只支持游标从头到尾顺序提取。只有从数据库中提取出来后才能进行检索。对所有由当前用户发出或由其他用户提交、并影响结果集中的行的 INSERT、UPDATE 和 DELETE 语句，其效果在这些行从游标中提取时是可见的。

4．键集驱动游标

打开游标时，键集驱动游标中的成员和行顺序是固定的。键集驱动游标由一套被称为键集的唯一标识符（键）控制。键由以唯一方式在结果集中标识行的列构成。键集是游标打开时来自所有适合 SELECT 语句的行中的一系列键值。键集驱动游标的键集在游标打开时建立在 tempdb 中。对非键集列中的数据值所做的更改（由游标所有者更改或其他用户提交）在用户滚动游标时是可见的。在游标外对数据库所做的插入在游标内是不可见的，除非关闭并重新打开游标。

13.2　游标的基本操作

视频讲解：光盘\TM\lx\13\游标的基本操作.mp4

游标的基本操作包括声明游标、打开游标、读取游标中的数据、关闭游标和释放游标。本节就详细介绍如何操作游标。

13.2.1　声明游标

声明游标可以使用 DECLARE CURSOR 语句。此语句有两种语法声明格式，分别为 ISO 标准语法和 Transact-SQL 扩展的语法，下面将分别介绍声明游标的两种语法格式。

1. ISO 标准语法

语法如下：

```
DECLARE cursor_name [ INSENSITIVE ] [ SCROLL ] CURSOR
FOR select_statement
FOR { READ ONLY | UPDATE [ OF column_name [ ,...n ] ] } ]
```

参数说明如下。

- ☑ DECLARE cursor_name：指定一个游标名称，其游标名称必须符合标识符规则。
- ☑ INSENSITIVE：定义一个游标，以创建将由该游标使用的数据的临时复本。对游标的所有请求都从 tempdb 中的临时表中得到应答；因此，在对该游标进行提取操作时返回的数据中不反映对基表所做的修改，并且该游标不允许修改。使用 SQL-92 语法时，如果省略 INSENSITIVE，（任何用户）对基表提交的删除和更新都反映在后面的提取中。
- ☑ SCROLL：指定所有的提取选项（FIRST、LAST、PRIOR、NEXT、RELATIVE、ABSOLUTE）均可用。
  - ➢ FIRST：取第一行数据。
  - ➢ LAST：取最后一行数据。
  - ➢ PRIOR：取前一行数据。
  - ➢ NEXT：取后一行数据。
  - ➢ RELATIVE：按相对位置取数据。
  - ➢ ABSOLUTE：按绝对位置取数据。

如果未指定 SCROLL，则 NEXT 是唯一支持的提取选项。

- ☑ select_statement：定义游标结果集的标准 SELECT 语句。在游标声明的 select_statement 内不允许使用关键字 COMPUTE、COMPUTE BY、FOR BROWSE 和 INTO。
- ☑ READ ONLY：表明不允许游标内的数据被更新，尽管在默认状态下游标是允许更新的。在 UPDATE 或 DELETE 语句的 WHERE CURRENT OF 子句中不允许引用游标。
- ☑ UPDATE [OF column_name [,...n]]：定义游标内可更新的列。如果指定 OF column_name [,...n] 参数，则只允许修改所列出的列。如果在 UPDATE 中未指定列的列表，则可以更新所有列。

2. Transact-SQL 扩展的语法

语法如下：

```
DECLARE cursor_name CURSOR
[ LOCAL | GLOBAL ]
[ FORWARD_ONLY | SCROLL ]
[ STATIC | KEYSET | DYNAMIC | FAST_FORWARD ]
[ READ_ONLY | SCROLL_LOCKS | OPTIMISTIC ]
[ TYPE_WARNING ]
FOR select_statement
[ FOR UPDATE [ OF column_name [ ,...n ] ] ]
```

DECLARE CURSOR 语句的参数及说明如表 13.1 所示。

表 13.1　DECLARE CURSOR 语句的参数及说明

参　数	描　述
DECLARE cursor_name	指定一个游标名称，其游标名称必须符合标识符规则
LOCAL	定义游标的作用域仅限在其所在的批处理、存储过程或触发器中。当建立游标在存储过程执行结束后，游标会被自动释放
GLOBAL	指定该游标的作用域对连接是全局的。在由连接执行的任何存储过程或批处理中，都可以引用该游标名称。该游标仅在脱接时隐性释放
FORWARD_ONLY	指定游标只能从第一行滚动到最后一行。FETCH NEXT 是唯一受支持的提取选项非指定 STATIC、KEYSET 或 DYNAMIC 关键字，否则默认为 FORWARD_ONLY。STATIC、KEYSET 和 DYNAMIC 游标默认为 SCROLL。与 ODBC 和 ADO 这类数据库 API 不同，STATIC、KEYSET 和 DYNAMIC Transact-SQL 游标支持 FORWARD_ONLY。FAST_FORWARD 和 FORWARD_ONLY 是互斥的；如果指定一个，则不能指定另一个
STATIC	定义一个游标，以创建将由该游标使用的数据的临时复本。对游标的所有请求都从 tempdb 中的该临时表中得到应答；因此，在对该游标进行提取操作时返回的数据中不反映对基表所做的修改，并且该游标不允许修改
KEYSET	指定当游标打开时，游标中行的成员资格和顺序已经固定。对行进行唯一标识的键集内置在 tempdb 内一个称为 keyset 的表中。对基表中的非键值所做的更改（由游标所有者更改或由其他用户提交）在用户滚动游标时是可视的。其他用户进行的插入是不可视的（不能通过 Transact-SQL 服务器游标进行插入）。如果某行已删除，则对该行的提取操作将返回@@FETCH_STATUS 值-2。从游标外更新键值类似于删除旧行后接着插入新行的操作。含有新值的行不可视，对含有旧值的行的提取操作将返回@@FETCH_STATUS 值-2。如果通过指定 WHERE CURRENT OF 子句用游标完成更新，则新值可视
DYNAMIC	定义一个游标，以反映在滚动游标时对结果集内的行所做的所有数据的更改。行的数据值、顺序和成员在每次提取时都会更改。动态游标不支持 ABSOLUTE 提取选项
FAST_FORWARD	指明一个 FORWARD_ONLY、READ_ONLY 型游标
SCROLL_LOCKS	指定确保通过游标完成的定位更新或定位删除可以成功。将行读入游标以确保它们可用于以后的修改时，SQL Server 会锁定这些行。如果还指定了 FAST_FORWARD，则不能指定 SCROLL_LOCKS
OPTIMISTIC	指明在数据被读入游标后，如果游标中某行数据已发生变化，那么对游标数据进行更新或删除可能会导致失败
TYPE_WARNING	指定如果游标从所请求的类型隐性转换为另一种类型，则给客户端发送警告消息

【例 13.1】 创建一个名为 Cur_Emp 的标准游标。（**实例位置：光盘\TM\sl\13\1**）

SQL 语句如下：

```
USE db_2012
DECLARE Cur_Emp CURSOR FOR
SELECT * FROM Employee
GO
```

运行结果如图 13.1 所示。

【例 13.2】 创建一个名为 Cur_Emp_01 的只读游标。（**实例位置：光盘\TM\sl\13\2**）

SQL 语句如下：

```
USE db_2012
DECLARE Cur_Emp_01 CURSOR FOR
```

```
SELECT * FROM Employee
FOR READ ONLY      --只读游标
GO
```

运行结果如图 13.2 所示。

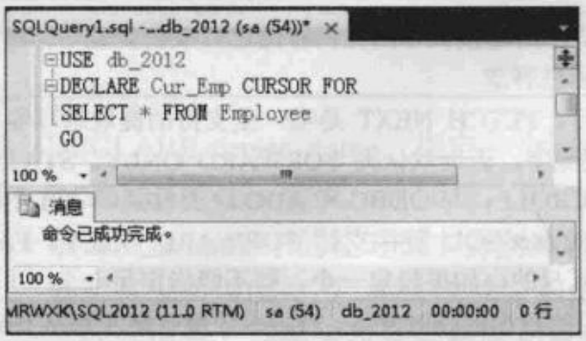


图 13.1　创建标准游标

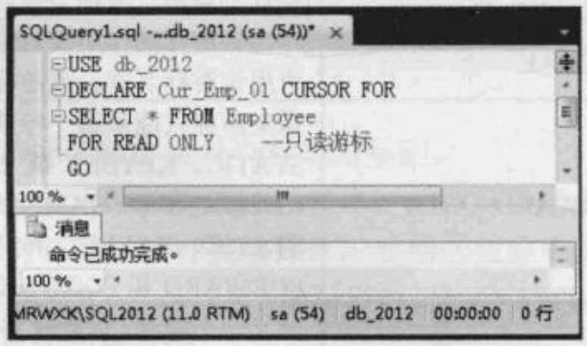


图 13.2　创建只读游标

【例 13.3】 创建一个名为 Cur_Emp_02 的更新游标。（**实例位置：光盘\TM\sl\13\3**）

SQL 语句如下：

```
USE db_2012
DECLARE Cur_Emp_02 CURSOR FOR
SELECT Name,Sex,Age FROM Employee
FOR UPDATE      --更新游标
GO
```

运行结果如图 13.3 所示。

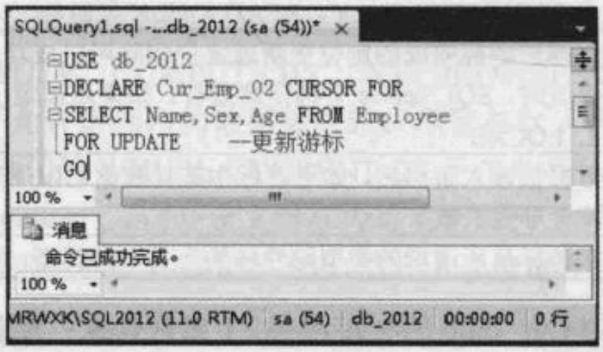


图 13.3　创建更新游标

13.2.2　打开游标

打开一个声明的游标可以使用 OPEN 命令。语法如下：

```
OPEN { { [ GLOBAL ] cursor_name } | cursor_variable_name }
```

参数说明如下。

☑ GLOBAL：指定 cursor_name 为全局游标。

☑ cursor_name：已声明的游标名称，如果全局游标和局部游标都使用 cursor_name 作为其名称，

那么如果指定了 GLOBAL，cursor_name 指的是全局游标，否则，cursor_name 指的是局部游标。

☑ cursor_variable_name：游标变量的名称，该名称引用一个游标。

说明

如果使用 INSENSITIV 或 STATIC 选项声明了游标，那么 OPEN 将创建一个临时表以保留结果集。如果结果集中任意行的大小超过 SQL Server 表的最大行大小，OPEN 将失败。如果使用 KEYSET 选项声明了游标，那么 OPEN 将创建一个临时表以保留键集。临时表存储在 tempdb 中。

【例 13.4】 首先声明一个名为 Emp_01 的游标，然后使用 OPEN 命令打开该游标。（**实例位置：光盘\TM\sl\13\4**）

SQL 语句如下：

```
USE db_2012
DECLARE Emp_01 CURSOR FOR        --声明游标
SELECT * FROM Employee
WHERE ID = '1'
OPEN Emp_01                      --打开游标
GO
```

运行结果如图 13.4 所示。

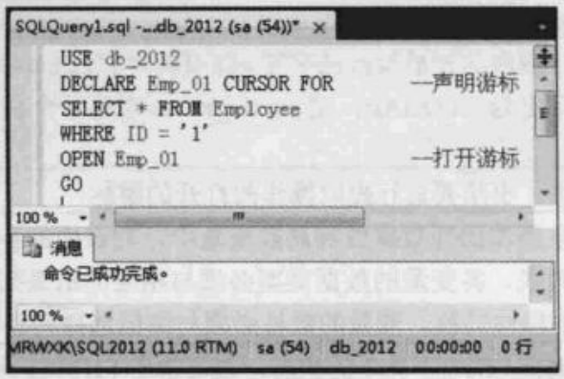


图 13.4　打开游标

13.2.3　读取游标中的数据

当打开一个游标之后，就可以读取游标中的数据了。可以使用 FETCH 命令读取游标中的某一行数据。语法如下：

```
FETCH
        [[ NEXT | PRIOR | FIRST | LAST
                | ABSOLUTE { n | @nvar }
                | RELATIVE { n | @nvar }
          ]
          FROM
        ]
```

```
{{ [ GLOBAL ] cursor_name } | @cursor_variable_name }
[ INTO @variable_name [ ,...n ] ]
```

FETCH 命令的参数及说明如表 13.2 所示。

表 13.2 FETCH 命令的参数及说明

参　数	描　述
NEXT	返回紧跟当前行之后的结果行，并且当前行递增为结果行。如果 FETCH NEXT 为对游标的第一次提取操作，则返回结果集中的第一行。NEXT 为默认的游标提取选项
PRIOR	返回紧临当前行前面的结果行，并且当前行递减为结果行。如果 FETCH PRIOR 为对游标的第一次提取操作，则没有行返回并且游标置于第一行之前
FIRST	返回游标中的第一行并将其作为当前行
LAST	返回游标中的最后一行并将其作为当前行
ABSOLUTE {n | @nvar}	如果 n 或@nvar 为正数，返回从游标头开始的第 n 行，并将返回的行变成新的当前行。如果 n 或@nvar 为负数，返回游标尾之前的第 n 行，并将返回的行变成新的当前行。如果 n 或@nvar 为 0，则没有行返回
RELATIVE {n | @nvar}	如果 n 或@nvar 为正数，返回当前行之后的第 n 行，并将返回的行变成新的当前行。如果 n 或@nvar 为负数，返回当前行之前的第 n 行，并将返回的行变成新的当前行。如果 n 或@nvar 为 0，返回当前行。如果对游标的第一次提取操作时将 FETCHRELATIVE 的 n 或@nvar 指定为负数或 0，则没有行返回。n 必须为整型常量且@nvar 必须为 smallint、tinyint 或 int
GLOBAL	指定 cursor_name 为全局游标
cursor_name	要从中进行提取的开放游标的名称。如果同时有以 cursor_name 作为名称的全局和局部游标存在，若指定为 GLOBAL，则 cursor_name 对应于全局游标，未指定 GLOBAL，则对应于局部游标
@cursor_variable_name	游标变量名，引用要进行提取操作的打开的游标
INTO @variable_name[,...n]	允许将提取操作的列数据放到局部变量中。列表中的各个变量从左到右与游标结果集中的相应列相关联。各变量的数据类型必须与相应的结果列的数据类型匹配或是结果列数据类型所支持的隐性转换。变量的数目必须与游标选择列表中的列的数目一致
@@FETCH_STATUS	返回上次执行 FETCH 命令的状态。在每次用 FETCH 从游标中读取数据时，都应检查该变量，以确定上次 FETCH 操作是否成功，决定如何进行下一步处理。@@FETCH_STATUS 变量有 3 个不同的返回值，说明如下：（1）返回值为 0，FETCH 语句成功；（2）返回值为-1，FETCH 语句失败或此行不在结果集中；（3）返回值为-2，被提取的行不存在

说明

（1）在前两个参数中，包含 n 和@nvar 其表示游标相对于作为基准的数据行所偏离的位置。

（2）当使用 SQL-92 语法来声明一个游标时，没有选择 SCROLL 选项，则只能使用 FETCH NEXT 命令来从游标中读取数据，即只能从结果集第一行按顺序地每次读取一行。由于不能使用 FIRST、LAST、PRIOR，所以无法回滚读取以前的数据。如果选择了 SCROLL 选项，则可以使用所有的 FETCH 操作。

【例 13.5】 用@@FETCH_STATUS 控制一个 WHILE 循环中的游标活动，SQL 语句及运行结果如图 13.5 所示。（**实例位置：光盘\TM\sl\13\5**）

图 13.5　从游标中读取数据

SQL 语句如下：

```
USE db_2012                                --引入数据库
DECLARE ReadCursor CURSOR FOR              --声明一个游标
SELECT * FROM Student
OPEN ReadCursor                            --打开游标
FETCH NEXT FROM ReadCursor                 --执行取数操作
WHILE @@FETCH_STATUS=0                     --检查@@FETCH_STATUS，以确定是否还可以继续取数
BEGIN
  FETCH NEXT FROM ReadCursor
END
```

13.2.4　关闭游标

当游标使用完毕之后，使用 CLOSE 语句可以关闭游标，但不释放游标占用的系统资源。

语法如下：

```
CLOSE { { [ GLOBAL ] cursor_name } | cursor_variable_name }
```

参数说明如下。

☑ GLOBAL：指定 cursor_name 为全局游标。

☑ cursor_name：开放游标的名称。如果全局游标和局部游标都使用 cursor_name 作为它们的名称，那么当指定 GLOBAL 时，cursor_name 引用全局游标；否则，cursor_name 引用局部游标。

☑ cursor_variable_name：与开放游标关联的游标变量名称。

【例 13.6】 声明一个名为 CloseCursor 的游标，并使用 Close 语句关闭游标。（**实例位置：光盘\TM\sl\13\6**）

SQL 语句如下：

```
USE db_2012
DECLARE CloseCursor Cursor FOR
```

```
SELECT * FROM   Student
FOR READ ONLY
OPEN CloseCursor
CLOSE CloseCursor
```

运行结果如图 13.6 所示。

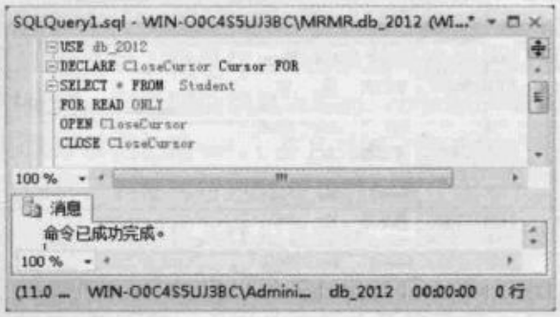


图 13.6　关闭游标

13.2.5　释放游标

当游标关闭之后，并没有在内存中释放所占用的系统资源，所以可以使用 DEALLOCATE 命令删除游标引用。当释放最后的游标引用时，组成该游标的数据结构由 SQL Server 释放。

语法如下：

```
DEALLOCATE { { [ GLOBAL ] cursor_name } | @cursor_variable_name }
```

参数说明如下。

☑ cursor_name：已声明游标的名称。当全局和局部游标都以 cursor_name 作为它们的名称存在时，如果指定 GLOBAL，则 cursor_name 引用全局游标，如果未指定 GLOBAL，则 cursor_name 引用局部游标。

☑ @cursor_variable_name：cursor 变量的名称。@cursor_variable_name 必须为 cursor 类型。

当使用 DEALLOCATE @cursor_variable_name 来删除游标时，游标变量并不会被释放，除非超过使用该游标的存储过程和触发器的范围。

【例 13.7】 使用 DEALLOCATE 命令释放名为 FreeCursor 的游标。（**实例位置：光盘\TM\sl\13\7**）

SQL 语句如下：

```
USE db_2012
DECLARE FreeCursor Cursor FOR
SELECT * FROM Student
OPEN FreeCursor
Close FreeCursor
DEALLOCATE FreeCursor
```

运行结果如图 13.7 所示。

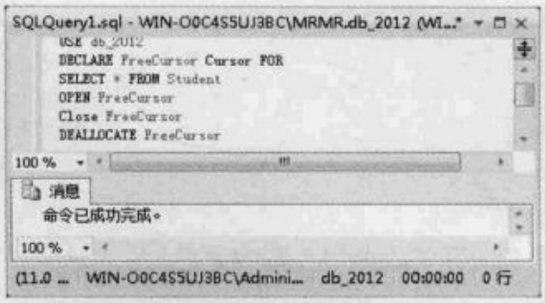


图 13.7　释放游标

13.3　使用系统过程查看游标

视频讲解：光盘\TM\lx\13\使用系统过程查看游标.mp4

创建游标后，通常使用 sp_cursor_list 和 sp_describe_cursor 查看游标的属性。sp_cursor_list 用来报告当前为连接打开的服务器游标的属性，sp_describe_cursor 用于报告服务器游标的属性。本节就详细介绍这两个系统过程。

13.3.1　sp_cursor_list

sp_cursor_list 报告当前为连接打开的服务器游标的属性。语法如下：

```
sp_cursor_list [ @cursor_return = ] cursor_variable_name OUTPUT
        , [ @cursor_scope = ] cursor_scope
```

参数说明如下。

☑ [@cursor_return =] cursor_variable_name OUTPUT：已声明的游标变量的名称。cursor_variable_name 的数据类型为 cursor，无默认值。游标是只读的可滚动动态游标。

☑ [@cursor_scope =] cursor_scope：指定要报告的游标级别。cursor_scope 的数据类型为 int，无默认值，可取值如表 13.3 所示。

表 13.3　cursor_scope 可取的值

值	说　明
1	报告所有本地游标
2	报告所有全局游标
3	报告本地游标和全局游标

【例 13.8】 声明一个游标 Cur_Employee，并使用 sp_cursor_list 报告该游标的属性。（**实例位置：光盘\TM\sl\13\8**）

SQL 语句如下：

```
USE db_2012
GO
```

```
DECLARE Cur_Employee CURSOR FOR
SELECT Name
FROM Employee
WHERE Name LIKE '王%'
OPEN Cur_Employee
DECLARE @Report CURSOR
EXEC master.dbo.sp_cursor_list @cursor_return = @Report OUTPUT,
        @cursor_scope = 2
FETCH NEXT from @Report
WHILE (@@FETCH_STATUS <> -1)
BEGIN
    FETCH NEXT from @Report
END
CLOSE @Report
DEALLOCATE @Report
GO
CLOSE Cur_Employee
DEALLOCATE Cur_Employee
GO
```

运行结果如图 13.8 所示。

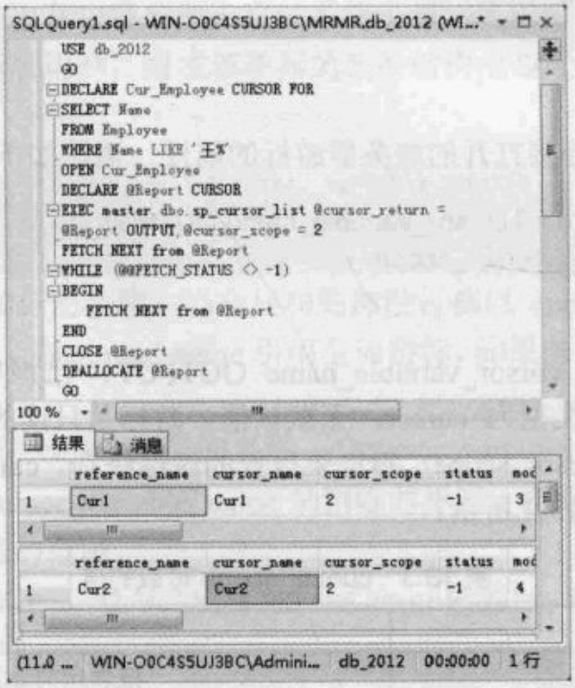


图 13.8　sp_cursor_list 属性

13.3.2　sp_describe_cursor

sp_describe_cursor 用于报告服务器游标的属性。语法如下：

```
sp_describe_cursor [ @cursor_return = ] output_cursor_variable OUTPUT
    { [ , [ @cursor_source = ] N'local'
```

```
    , [ @cursor_identity = ] N'local_cursor_name' ]
  | [ , [ @cursor_source = ] N'global'
    , [ @cursor_identity = ] N'global_cursor_name' ]
  | [ , [ @cursor_source = ] N'variable'
    , [ @cursor_identity = ] N'input_cursor_variable' ]
    }
```

sp_describe_cursor 语句的参数及说明如表 13.4 所示。

表 13.4　sp_describe_cursor 语句的参数及说明

参　数	描　述
[@cursor_return =] output_cursor_variable OUTPUT	用于接收游标输出的声明游标变量的名称。output_cursor_variable 的数据类型为 cursor，无默认值。调用 sp_describe_cursor 时，该参数不得与任何游标关联。返回的游标是可滚动的动态只读游标
[@cursor_source =] { N'local'| N'global' | N'variable' }	指定是使用局部游标的名称、全局游标的名称还是游标变量的名称来指定要报告的游标。该参数的类型为 nvarchar(30)
[@cursor_identity =] N'local_cursor_name']	由具有 LOCAL 关键字或默认设置为 LOCAL 的 DECLARE CURSOR 语句创建的游标名称。local_cursor_name 的数据类型为 nvarchar(128)
[@cursor_identity =] N'global_cursor_name']	由具有 GLOBAL 关键字或默认设置为 GLOBAL 的 DECLARE CURSOR 语句创建的游标名称。global_cursor_name 的数据类型为 nvarchar(128)
[@cursor_identity =] N'input_cursor_variable']	与所打开游标相关联的游标变量的名称。input_cursor_variable 的数据类型为 nvarchar(128)

【例 13.9】 声明一个游标，并使用 sp_describe_cursor 报告该游标的属性。（**实例位置：光盘\TM\sl\13\9**）

SQL 语句如下：

```
USE db_2012
GO
DECLARE Cur_Employee CURSOR STATIC FOR
SELECT Name
FROM Employee
OPEN Cur_Employee
DECLARE @Report CURSOR
EXEC master.dbo.sp_describe_cursor @cursor_return = @Report OUTPUT,
        @cursor_source = N'global', @cursor_identity = N'Cur_Employee'
FETCH NEXT from @Report
WHILE (@@FETCH_STATUS <> -1)
BEGIN
    FETCH NEXT from @Report
END
CLOSE @Report
DEALLOCATE @Report
GO
CLOSE Cur_Employee
DEALLOCATE Cur_Employee
GO
```

运行结果如图 13.9 所示。

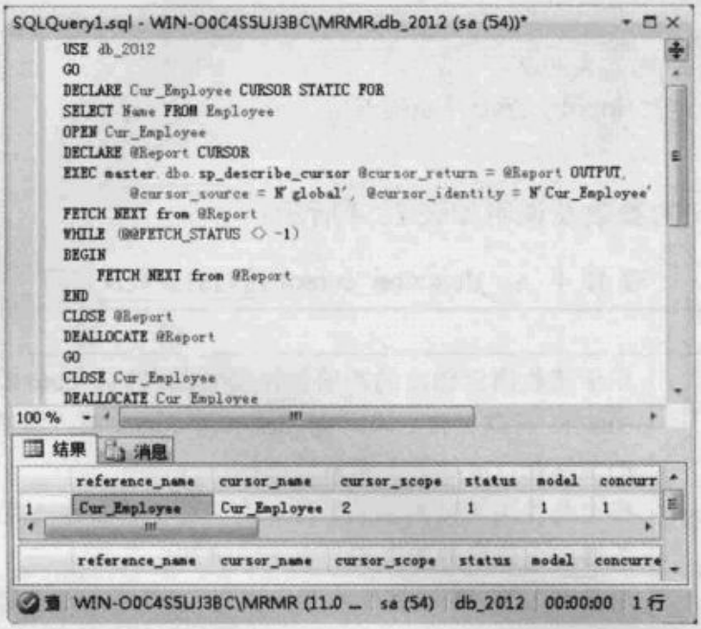


图 13.9　sp_describe_cursor 属性

13.4　小　　结

本章主要介绍了游标的概念、类型及游标的基本操作。游标为应用程序提供了每次对结果集处理一行或一部分行的机制。虽然游标可以解决结果集无法完成的所有操作，但要避免使用游标，游标非常消耗资源，而且会对性能产生很大的影响。游标只能在别无选择的时候使用。

13.5　实践与练习

1．使用游标查询数据，在商品表中返回指定的商品行信息数据。（**答案位置：光盘\TM\sl\13\10**）

2．为表 employee4 创建一个游标，包括编号、姓名、性别、年龄、电话号码字段。使用该游标限制用户只能更新游标的姓名和性别字段中的值，在 DECLARE CURSOR 语句的 UPDATE 子句中列出要更新的字段。（**答案位置：光盘\TM\sl\13\11**）

第14章

索引与数据完整性

（视频讲解：56分钟）

本章主要介绍索引与数据完整性，主要包括索引的概念、索引的建立、索引的删除、索引的分析与维护、域完整性、实体完整性和引用完整性。通过本章的学习，读者应掌握建立和删除索引的方法，能够使用索引优化数据库查询，了解数据完整性。

通过阅读本章，您可以：

- 熟悉索引的基本概念及分类
- 了解索引的优缺点
- 掌握如何创建、修改和删除索引
- 掌握索引的分析与维护
- 掌握如何使用数据库进行全文索引
- 熟悉全文目录的创建与删除
- 熟悉数据完整性

14.1 索引的概念

视频讲解：光盘\TM\lx\14\索引的概念.mp4

与书中的索引一样，数据库中的索引使用户可以快速找到表或索引视图中的特定信息。索引包含从表或视图中一个或多个列生成的键，以及映射到指定数据的存储位置的指针。通过创建设计良好的索引以支持查询，可以显著提高数据库查询和应用程序的性能。索引可以减少为返回查询结果集而必须读取的数据量。索引还可以强制表中的行具有唯一性，从而确保表数据的数据完整性。

索引是一个单独的、物理的数据库结构，在 SQL Server 中，索引是为了加速对表中数据行的检索而创建的一种分散存储结构。它是针对一个表而建立的，每个索引页面中的行都含有逻辑指针，指向数据表中的物理位置，以便加速检索物理数据。因此，对表中的列是否创建索引，将对查询速度有很大的影响。一个表的存储是由两部分组成的，一部分用来存放表的数据页，另一部分存放索引页。通常索引页面对于数据页来说小得多。在进行数据检索时，系统首先搜索索引页面，从中找到所需数据的指针，然后直接通过该指针从数据页面中读取数据，从而提高查询速度。

14.2 索引的优缺点

视频讲解：光盘\TM\lx\14\索引的优缺点.mp4

索引是与表或视图关联的磁盘上的结构，可以加快从表或视图中检索行的速度。本节将介绍索引的优缺点。

14.2.1 索引的优点

索引具有以下优点：

- ☑ 创建唯一性索引，保证数据库表中每一行数据的唯一性。
- ☑ 大大加快数据的检索速度，这也是创建索引的最主要原因。
- ☑ 加速表与表之间的连接，特别是在实现数据的参考完整性方面特别有意义。
- ☑ 在使用分组和排序子句进行数据检索时，同样可以减少查询中分组和排序的时间。
- ☑ 通过使用索引，可以在查询的过程中使用优化隐藏器，提高系统的性能。

14.2.2 索引的缺点

索引具有以下缺点：

- ☑ 创建索引和维护索引要耗费时间，这种时间随着数据量的增加而增加。

☑ 索引需要占物理空间，除了数据表占数据空间之外，每一个索引还要占一定的物理空间，如果要建立聚集索引，那么需要的空间就会更大。

☑ 当对表中的数据进行增加、删除和修改时，索引也要动态地维护，降低了数据的维护速度。

14.3　索引的分类

视频讲解：光盘\TM\lx\14\索引的分类.mp4

在 SQL Server 2012 中提供的索引类型主要有以下几类：聚集索引、非聚集索引、唯一索引、包含性列索引、索引视图、全文索引、空间索引、筛选索引和 XML 索引。

按照存储结构的不同，可以将索引分为两类：聚集索引和非聚集索引。

14.3.1　聚集索引

聚集索引根据数据行的键值在表或视图中排序和存储这些数据行。索引定义中包含聚集索引列。每个表只能有一个聚集索引，因为数据行本身只能按一个顺序排序。

只有当表包含聚集索引时，表中的数据行才按排序顺序存储。如果表具有聚集索引，则该表称为聚集表。如果表没有聚集索引，则其数据行存储在一个称为堆的无序结构中。

除了个别表之外，每个表都应该有聚集索引。聚集索引除了可以提高查询性能之外，还可以按需重新生成或重新组织来控制表碎片。

聚集索引按下列方式实现：

☑ PRIMARY KEY 和 UNIQUE 约束

- 在创建 PRIMARY KEY 约束时，如果不存在该表的聚集索引且未指定唯一非聚集索引，则将自动对一列或多列创建唯一聚集索引。主键列不允许空值。
- 在创建 UNIQUE 约束时，默认情况下将创建唯一非聚集索引，以便强制 UNIQUE 约束。如果不存在该表的聚集索引，则可以指定唯一聚集索引。

☑ 独立于约束的索引

指定非聚集主键约束后，可以对非主键列的列创建聚集索引。

☑ 索引视图

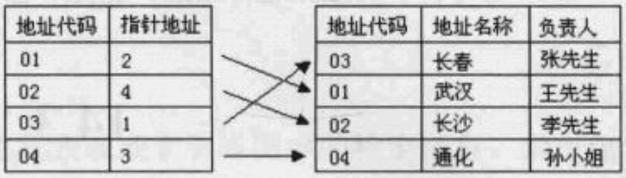
若要创建索引视图，请对一个或多个视图列定义唯一聚集索引。视图将具体化，并且结果集存储在该索引的页级别中，其存储方式与表数据存储在聚集索引中的方式相同。

14.3.2　非聚集索引

非聚集索引具有独立于数据行的结构。非聚集索引包含非聚集索引键值，并且每个键值项都有指向包含该键值的数据行的指针。

从非聚集索引中的索引行指向数据行的指针称为行定位器。行定位器的结构取决于数据页是存储在堆中还是聚集表中。对于堆，行定位器是指向行的指针。对于聚集表，行定位器是聚集索引键。

下面以图 14.1 为例对非聚集索引的结构进行详细说明。图 14.1（a）中的数据是按图 14.1（b）中的数据进行顺序存储的，在图 14.1（a）中为“地址代码”列建立索引，“指针地址”列是每条记录在表中的存储位置（通常称为指针），当查询地址代码为 01 的信息时，先在索引表中查找地址代码 01，然后根据索引表中的指针地址（在这里指针地址为 2）找到第 2 条记录，这样就很大地提高了查询速度。



地址代码	指针地址
01	2
02	4
03	1
04	3

（a）

地址代码	地址名称	负责人
03	长春	张先生
01	武汉	王先生
02	长沙	李先生
04	通化	孙小姐

（b）

图 14.1　非聚集索引结构

14.4　索引的操作

视频讲解：光盘\TM\lx\14\索引的操作.mp4

索引就是加快检索表中数据的方法。它对数据表中一个或多个列的值进行结构排序，是数据库中一个非常有用的对象。本节主要介绍如何通过企业管理器和 Transact-SQL 语句创建索引。

14.4.1　索引的创建

1. 使用企业管理器创建索引

操作步骤如下：

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2012 数据库。

（2）选择指定的数据库 db_2012，然后展开要创建索引的表，在表的下级菜单中，鼠标右键单击“索引”，在弹出的快捷菜单中选择“新建索引”命令，如图 14.2 所示。弹出“新建索引”窗口，如图 14.3 所示。

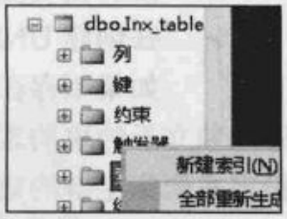


图 14.2　选择“新建索引”命令

（3）在“新建索引”窗口中单击“添加”按钮，弹出“从表中选择列”窗口，在该窗口中选择要添加到索引键的表列，如图 14.4 所示。

（4）单击“确定”按钮，返回到“新建索引”窗口，在“新建索引”窗口中，单击“确定”按钮，便完成了索引的创建。

2. 使用 Transact-SQL 语句创建索引

CREATE INDEX 语句为给定表或视图创建一个改变物理顺序的聚集索引，也可以创建一个具有查询功能的非聚集索引。

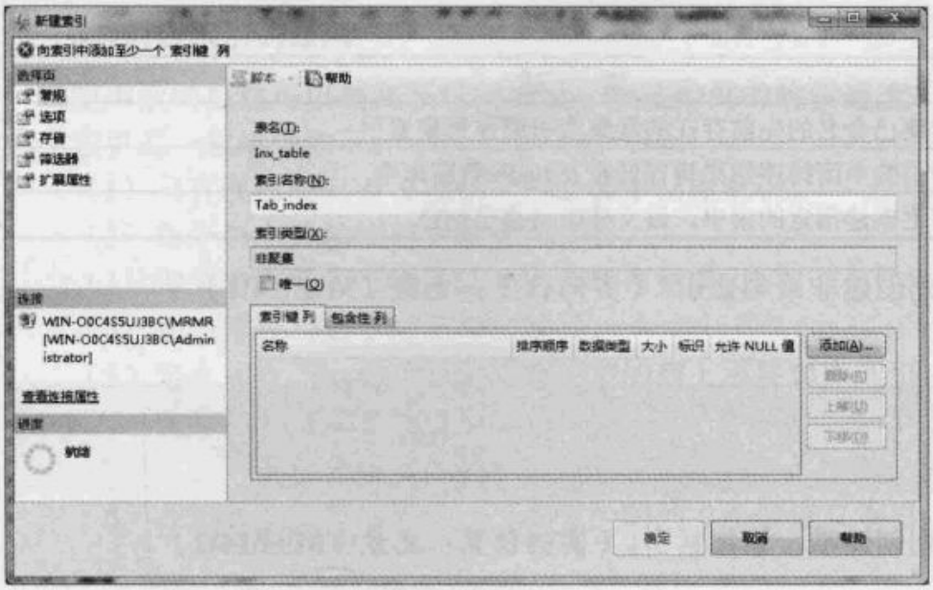


图 14.3　新建索引

图 14.4　选择列

语法如下：

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX index_name
    ON { table | view } ( column [ ASC | DESC ] [ ,...n ] )
[ WITH < index_option > [ ,...n] ]
[ ON filegroup ]
< index_option > ::=
  { PAD_INDEX |
      FILLFACTOR = fillfactor |
      IGNORE_DUP_KEY |
      DROP_EXISTING |
  STATISTICS_NORECOMPUTE |
  SORT_IN_TEMPDB
}
```

CREATE INDEX 语句的参数及说明如表 14.1 所示。

表 14.1　CREATE INDEX 语句的参数及说明

参　数	描　述
[UNIQUE][CLUSTERED|NONCLUSTERED]	指定创建索引的类型，参数依次为唯一索引、聚集索引和非聚集索引。当省略 UNIQUE 选项时，建立非唯一索引，省略 CLUSTERED|NONCLUSTERED 选项时，建立聚集索引，省略 NONCLUSTERED 选项时，建立唯一聚集索引
index_name	索引名。索引名在表或视图中必须唯一，但在数据库中不必唯一。索引名必须遵循标识符规则
table	包含要创建索引的列的表。可以选择指定数据库和表所有者
column	应用索引的列。指定两个或多个列名，可为指定列的组合值创建组合索引
[ASC | DESC]	确定具体某个索引列的升序或降序排序方向。默认设置为 ASC
PAD_INDEX	指定索引中间级中每个页（节点）上保持开放的空间
FILLFACTOR	指定在 SQL Server 创建索引的过程中，各索引页的填满程度
IGNORE_DUP_KEY	控制向唯一聚集索引的列插入重复的键值时所发生的情况。如果为索引指定了 IGNORE_DUP_KEY，并且执行了创建重复键的 INSERT 语句，SQL Server 将发出警告消息并忽略重复的行

续表

参　数	描　述
DROP_EXISTING	指定应删除并重建已命名的先前存在的聚集索引或非聚集索引
SORT_IN_TEMPDB	指定用于生成索引的中间排序结果将存储在 tempdb 数据库中
ON filegroup	在给定的文件组上创建指定的索引。该文件组必须已创建

【例 14.1】 为 Student 表的 Sno 列创建非聚集索引。（**实例位置：光盘\TM\sl\14\1**）

SQL 语句如下：

```
USE db_2012
CREATE   INDEX IX_Stu_Sno
ON Student (Sno)
```

【例 14.2】 为 Student 表的 Sno 列创建唯一聚集索引。（**实例位置：光盘\TM\sl\14\2**）

SQL 语句如下：

```
USE db_2012
CREATE UNIQUE CLUSTERED INDEX   IX_Stu_Sno1
ON Student (Sno)
```

注意

无法对表创建多个聚集索引。

【例 14.3】 为 Student 表的 Sno 列创建组合索引。（**实例位置：光盘\TM\sl\14\3**）

SQL 语句如下：

```
USE db_2012
CREATE INDEX IX_Stu_Sno2
ON Student (Sno,Sname DESC)
```

【例 14.4】 用 FILLFACTOR 参数为 Student 表的 Sno 列创建一个填充因子为 100 的非聚集索引。（**实例位置：光盘\TM\sl\14\4**）

SQL 语句如下：

```
USE db_2012
CREATE NONCLUSTERED INDEX   IX_Stu_Sno3
ON Student (Sno)
WITH FILLFACTOR = 100
```

【例 14.5】 用 IGNORE_DUP_KEY 参数为 Student 表的 Sno 列创建唯一聚集索引，并且不能输入重复值。（**实例位置：光盘\TM\sl\14\5**）

SQL 语句如下：

```
USE db_2012
CREATE UNIQUE CLUSTERED INDEX IX_Stu_Sno4
ON Student (Sno)
WITH IGNORE_DUP_KEY
```

3. 创建索引的原则

使用索引虽然可以提高系统的性能，增强数据的检索速度，但它需要占用大量的物理存储空间，建立索引的一般原则如下。

（1）只有表的所有者可以在同一个表中创建索引。
（2）每个表中只能创建一个聚集索引。
（3）每个表中最多可以创建 249 个非聚集索引。
（4）在经常查询的字段上建立索引。
（5）定义 text、image 和 bit 数据类型的列上不要建立索引。
（6）在外键列上可以建立索引。
（7）主键列上一定要建立索引。
（8）在那些重复值比较多、查询较少的列上不要建立索引。

14.4.2　查看索引信息

1. 使用企业管理器查看索引

使用企业管理器查看索引的步骤如下：

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2012 数据库。
（2）选择指定的数据库 db_2012，然后展开要查看索引的表。
（3）鼠标右键单击该表，在弹出的快捷菜单中选择“设计”命令。
（4）弹出“表结构设计”窗体，鼠标右键单击该窗体，在弹出的快捷菜单中选择“索引/键”命令。
（5）打开“索引/键”对话框，如图 14.5 所示。在对话框的左侧选中某个索引，在对话框的右侧就可以查看此索引的信息，并可以修改相关的信息。

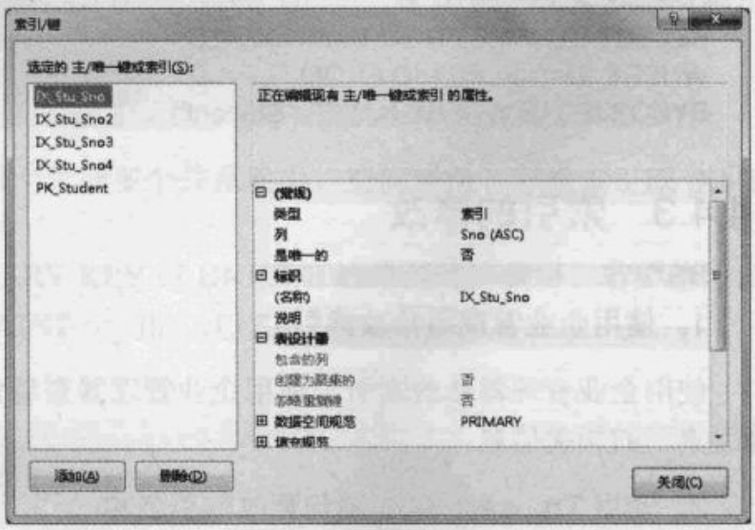


图 14.5　“索引/键”对话框

2. 使用系统存储过程查看索引

系统存储过程 sp_helpindex 可以报告有关表或视图上索引的信息。语法如下：

```
sp_helpindex [ @objname = ] 'name'
```

参数[@objname =] 'name'表示用户定义的表或视图的限定或非限定名称。

【例 14.6】 用系统存储过程 sp_helpindex，查看 db_2012 数据库中 Student 表的索引信息。**（实例位置：光盘\TM\sl\14\6）**

SQL 语句如下：

```
use db_2012
EXEC Sp_helpindex Student
```

运行结果如图 14.6 所示。

3. 利用系统表查看索引信息

查看数据库中指定表的索引信息，可以利用该数据库中的系统表 sysobjects（记录当前数据库中所有对象的相关信息）和 sysindexes（记录有关索引和建立索引表的相关信息）进行查询，系统表 sysobjects 可以根据表名查找到索引表的 ID 号，再利用系统表 sysindexes 根据 ID 号查找到索引文件的相关信息。

【例 14.7】 利用系统表查看 db_2012 数据库中 Student 表中的索引信息，SQL 语句及运行结果如图 14.7 所示。（**实例位置：光盘\TM\sl\14\7**）

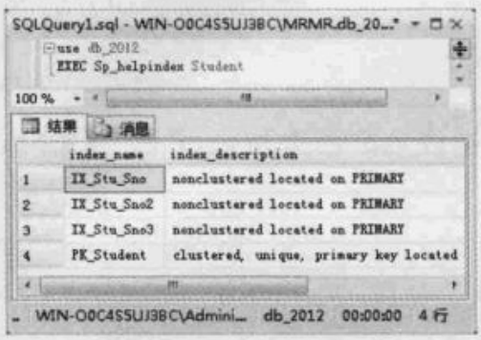


图 14.6 使用系统存储过程查看索引

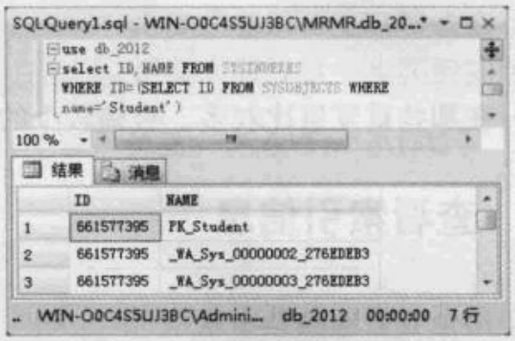


图 14.7 查看 Student 表中的索引

SQL 语句如下：

```
USE db_2012
SELECT ID,NAME FROM SYSINDEXES
WHERE ID=(SELECT ID FROM
SYSOBJECTS WHERE NAME ='Student')
```

14.4.3 索引的修改

1. 使用企业管理器修改索引

使用企业管理器修改索引与使用企业管理器查看索引的步骤相同，在“索引/键”对话框中就可以修改索引的相关信息。

2. 使用 Transact-SQL 语句更改索引名称

在当前数据库中更改用户创建对象的名称。此对象可以是表、索引、列、别名数据类型或 Microsoft .NET Framework 公共语言运行时（CLR）用户定义类型。

语法如下：

```
sp_rename [ @objname = ] 'object_name' ,
[ @newname = ] 'new_name'
[ , [ @objtype = ] 'object_type' ]
```

参数说明如下。

☑ [@objname =] 'object_name'：用户对象或数据类型的当前限定或非限定名称。

☑ [@newname =] 'new_name'：指定对象的新名称。

☑ [@objtype =] 'object_type'：要重命名的对象的类型。

【例 14.8】 利用系统存储过程 sp_rename，将 IX_Stu_Sno 索引重命名为 IX_Stu_Sno1。（实例位置：光盘\TM\sl\14\8）

SQL 语句如下：

```
USE db_2012
EXEC sp_rename 'Student.IX_Stu_Sno','IX_Stu_Sno1'
```

运行结果如图 14.8 所示。

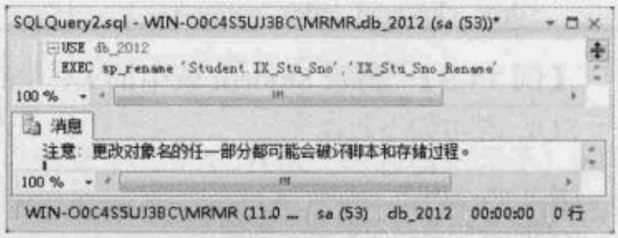


图 14.8　更改索引名称

注意

要对索引进行重命名时，需要修改的索引名格式必须为“表名.索引名”。

14.4.4　索引的删除

1. 使用企业管理器删除索引

使用企业管理器删除索引与使用企业管理器查看索引的步骤相同，在“索引/键”对话框中，单击“删除”按钮，就可以把当前选中的索引删除。

2. 使用 Transact-SQL 语句删除索引

DROP INDEX 语句表示从当前数据库中删除一个或多个关系索引、空间索引、筛选索引或 XML 索引。

DROP INDEX 语句不适用于通过定义 PRIMARY KEY 或 UNIQUE 约束创建的索引。若要删除该约束和相应的索引，请使用带有 DROP CONSTRAINT 子句的 ALTER TABLE。

DROP INDEX 语句的语法如下：

```
DROP INDEX
{ <drop_relational_or_xml_or_spatial_index> [ ,...n ]
| <drop_backward_compatible_index> [ ,...n ]
}
<drop_relational_or_xml_or_spatial_index> ::=
        index_name ON <object>
    [ WITH ( <drop_clustered_index_option> [ ,...n ] ) ]
<drop_backward_compatible_index> ::=
    [ owner_name. ] table_or_view_name.index_name
<object> ::=
{
    [ database_name. [ schema_name ] . | schema_name. ]
        table_or_view_name
}t
```

DROP INDEX 语句的参数及说明如表 14.2 所示。

表 14.2 DROP INDEX 语句的参数及说明

参　数	描　述
index_name	要删除的索引名称
database_name	数据库的名称
schema_name	该表或视图所属架构的名称
table_or_view_name	与该索引关联的表或视图的名称
<drop_clustered_index_option>	控制聚集索引选项。这些选项不能与其他索引类型一起使用

【例 14.9】 删除 Student 表中的 IX_Stu_Sno1 索引。（**实例位置：光盘\TM\sl\14\9**）

SQL 语句如下：

```
USE db_2012
--判断表中是否有要删除的索引
If EXISTS(Select * from sysindexes where name='IX _Stu_Sno1')
 Drop Index Student.IX_Stu_Sno1
```

运行结果如图 14.9 所示。

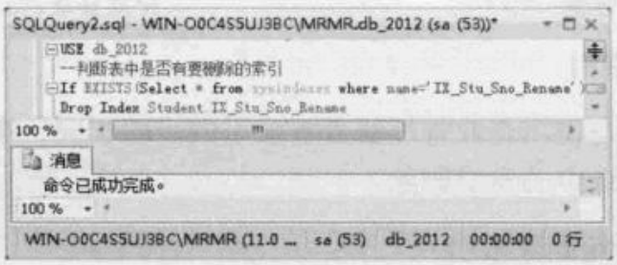


图 14.9 索引的删除

【例 14.10】 删除 Student 表中的 IX_Stu_Sno3 索引和 SC 表中的 IX_SC_Sno 索引。（**实例位置：光盘\TM\sl\14\10**）

SQL 语句如下：

```
USE db_2012
Drop Index Student.IX_Stu_Sno3,SC.IX_SC_Sno
```

14.4.5 设置索引的选项

1. 设置 PAD_INDEX 选项

PAD_INDEX 选项是设置创建索引期间中间级别页中可用空间的百分比。

对于非叶级索引页需要使用 PAD_INDEX 选项设置其预留空间的大小。PAD_INDEX 选项只有在指定了 FILLFACTOR 选项时才有用，因为 PAD_INDEX 是由 FILLFACTOR 所指定的百分比决定。默认情况下，给定中间级页上的键集，SQL Server 将确保每个索引页上的可用空间至少可以容纳一个索引允许的最大行。如果 FILLFACTOR 指定的百分比不够大，无法容纳一行，SQL Server 将在内部使用允许的最小值替代该百分比。

【例 14.11】 为 Student 表的 Sno 列创建一个簇索引 IX_Stu_Sno，并将预留空间设置为 10。（**实例位置：光盘\TM\sl\14\11**）

SQL 语句如下：

```
USE db_2012
CREATE UNIQUE CLUSTERED INDEX IX_Stu_Sno
on Student(Sno)
with pad_index,fillfactor = 10
```

2. 设置 FILLFACTOR 选项

FILLFACTOR 选项是设置创建索引期间每个索引页的页级别中可用空间的百分比。

数据库系统在存储数据库文件时，有时会将用到的数据页隔断，在使用数据索引的同时会产生一定程度的碎片。为了尽量减少页拆分，在创建索引时，可以选择 FILLFACTOR（称为填充因子）选项，此选项用来指定各索引页的填满程度，即指定索引页上所留出的额外的间隙和保留一定的百分比空间，从而扩充数据的存储容量和减少页拆分。FILLFACTOR 选项的取值范围是 1～100，表示用户创建索引时数据容量所占页容量的百分比。

【例 14.12】在 db_2012 数据库中的 Student 表上创建基于 Sname 列的非聚集索引 IX_Stu_Snanme，并且为升序，填充因子为 80。（**实例位置：光盘\TM\sl\14\12**）

SQL 语句如下：

```
USE db_2012
GO
CREATE   INDEX IX_Stu_Sname ON Student(Sname)
WITH   FILLFACTOR=80
GO
```

3. 设置 ASC/DESC 选项

排序查询是指将查询结果按指定属性的升序（ASC）或降序（DESC）排列，由 ORDER BY 子句指明。ASC/DESC 选项可以在创建索引时设置索引方式。

【例 14.13】 在 Student 表中创建一个聚集索引 MR_Stu_Sage，将 Sage 列按从小到大排序。（**实例位置：光盘\TM\sl\14\13**）

SQL 语句如下：

```
USE db_2012
CREATE CLUSTERED INDEX MR_Stu_Sage
ON Student (Sage DESC)
```

创建索引后，数据表如图 14.10 所示。

【例 14.14】 在 Student 表中创建一个聚集索引 MR_Stu，将 Sage 列按从小到大排序，Sno 列从大到小排序。（**实例位置：光盘\TM\sl\14\14**）

SQL 语句如下：

```
USE db_2012
CREATE CLUSTERED INDEX MR_Stu
ON Student (Sage DESC,Sno ASC)
```

创建索引后，数据表如图 14.11 所示。

4. 设置 SORT_IN_TEMPDB 选项

SORT_IN_TEMPDB 选项是确定对创建索引期间生成的中间排序结果进行排序的位置。如果为 ON，则排序结果存储在 tempdb 中。如果为 OFF，则排序结果存储在存储结果索引的文件组或分区方案中。

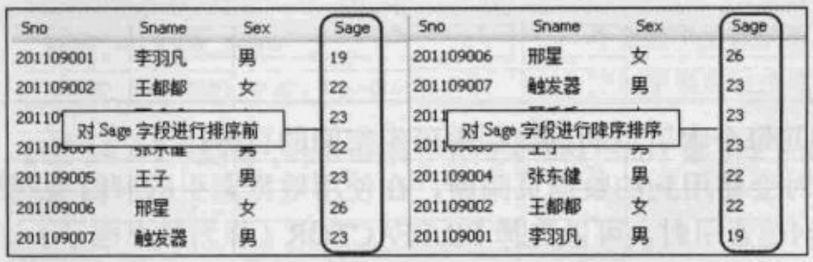


图 14.10　对 Sage 字段进行排序

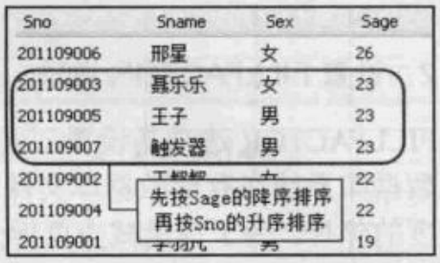


图 14.11　按多字段进行排序

【例 14.15】 用 SORT_IN_TEMPDB 选项创建 MR_Stu 索引，当 tempdb 与用户数据库位于不同的磁盘集上时，可以减少创建索引所需的时间。（**实例位置：光盘\TM\sl\14\15**）

SQL 语句如下：

```
CREATE UNIQUE CLUSTERED INDEX MR_Stu_Sno
ON Student (Sno ASC)
with SORT_IN_TEMPDB
```

5. 设置 STATISTICS_NORECOMPUTE 选项

STATISTICS_NORECOMPUTE 选项指定是否应自动重新计算过期的索引统计信息。

【例 14.16】 在 Student 表上创建索引 MR_Stu，其功能是不自动重新计算过期的索引统计信息。（**实例位置：光盘\TM\sl\14\16**）

SQL 语句如下：

```
USE db_2012
CREATE UNIQUE CLUSTERED INDEX MR_Stu
ON Student (Sno ASC)
with STATISTICS_NORECOMPUTE
```

6. 设置 UNIQUE 选项

UNIQUE 选项是确定是否允许并发用户在索引操作期间访问基础表或聚集索引数据以及任何关联非聚集索引。

为表或视图创建唯一索引（不允许存在索引值相同的两行）。视图上的聚集索引必须是 UNIQUE 索引。如果存在唯一索引，当使用 UPDATE 或 INSERT 语句产生重复值时将回滚，并显示错误信息。即使 UPDATE 或 INSERT 语句更改了许多行但只产生了一个重复值，也会出现这种情况。如果在有唯一索引并且指定了 IGNORE_DUP_KEY 子句情况下输入数据，则只有违反 UNIQUE 索引的行才会失败。在处理 UPDATE 语句时，IGNORE_DUP_KEY 不起作用。

【例 14.17】 用 IGNORE_DUP_KEY 参数创建唯一聚集索引，并且不能输入重复值，改变行的物理排序。（**实例位置：光盘\TM\sl\14\17**）

SQL 语句如下：

```
USE db_2012
CREATE UNIQUE CLUSTERED INDEX MR_Stu_Sno ON Student (Sno)
WITH IGNORE_DUP_KEY
```

7. 设置 DROP_EXISTING 选项

DROP_EXISTING 选项指示应删除和重新创建现有索引。

删除 SQL Server 2012 中已存在的索引，并根据修改重新创建一个索引，如果创建的是一个聚集索引，并且被索引的表上还存在其他非聚集索引，通过创建可以提高表的查询性能，因为重建聚集索引将强制重建所有的非聚集索引。

【例 14.18】 对已有的索引 MR_Stu，进行重新创建。（**实例位置：光盘\TM\sl\14\18**）

SQL 语句如下：

```
CREATE UNIQUE CLUSTERED INDEX MR_Stu
ON Student (Sno ASC)
with DROP_EXISTING
```

14.5　索引的分析与维护

视频讲解：光盘\TM\lx\14\索引的分析与维护.mp4

索引建立后，还需对它们进行分析和维护。本节主要讲解索引的分析及维护的方法。

14.5.1　索引的分析

1. 使用 SHOWPLAN 语句

显示查询语句的执行信息，包含查询过程中连接表时所采取的每个步骤以及选择哪个索引。语法如下：

```
SET SHOWPLAN_ALL { ON | OFF }
SET SHOWPLAN_TEXT { ON | OFF }
```

参数说明如下。

☑ ON：显示查询执行信息。

☑ OFF：不显示查询执行信息（系统默认）。

SET SHOWPLAN_ALL 的设置是在执行或运行时设置，而不是在分析时设置。如果 SET SHOWPLAN_ALL 为 ON，则 SQL Server 将返回每个语句的执行信息但不执行语句。Transact-SQL 语句不会被执行。在将此选项设置为 ON 后，将始终返回有关所有后续 Transact-SQL 语句的信息，直到将该选项设置为 OFF 为止。

SET SHOWPLAN_TEXT 的设置是在执行或运行时设置的，而不是在分析时设置的。当 SET

SHOWPLAN_TEXT 为 ON 时，SQL Server 将返回每个 Transact-SQL 语句的执行信息，但不执行语句。将该选项设置为 ON 以后，将返回有关所有后续 SQL Server 语句的执行计划信息，直到将该选项设置为 OFF 为止。

【例 14.19】 在 db_2012 数据库中的 Student 表中查询所有性别为男且年龄大于 23 岁的学生信息。**（实例位置：光盘\TM\sl\14\19）**

SQL 语句如下：

```
USE db_2012
GO
SET SHOWPLAN_ALL ON
GO
SELECT Sname,Sex,Sage FROM Student WHERE Sex='男' AND Sage >23
GO
SET SHOWPLAN_ALLOFF
GO
```

运行结果如图 14.12 所示。

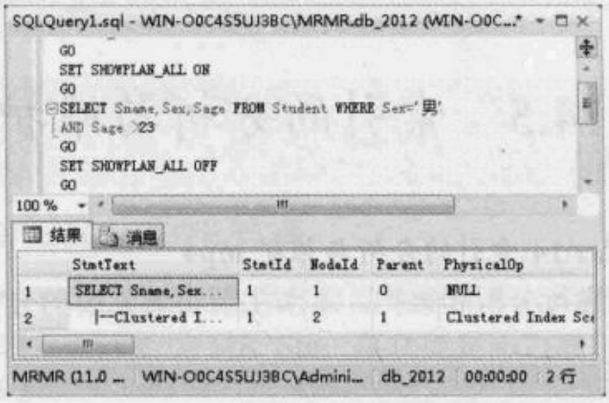


图 14.12　SHOWPLAN 语句的使用

2. 使用 STATISTICS IO 语句

STATISTICS IO 语句表示使 SQL Server 显示有关由 Transact-SQL 语句生成的磁盘活动量的信息。语法如下：

```
SET STATISTICS IO { ON | OFF }
```

如果 STATISTICS IO 为 ON，则显示统计信息。如果为 OFF，则不显示统计信息。如果将此选项设置为 ON，则所有后续的 Transact-SQL 语句将返回统计信息，直到将该选项设置为 OFF 为止。

【例 14.20】 在 db_2012 数据库中的 Student 表中查询所有性别为男且年龄大于 20 岁的学生信息，并显示查询处理过程在磁盘活动的统计信息。**（实例位置：光盘\TM\sl\14\20）**

SQL 语句如下：

```
USE db_2012
GO
SET STATISTICS IO ON
GO
```

```
SELECT Sname,Sex,Sage FROM Student WHERE Sex='男' AND Sage >20
GO
SET STATISTICS IO OFF;
GO
```

14.5.2　索引的维护

1. 使用 DBCC SHOWCONTIG 语句

显示指定表的数据和索引的碎片信息。当对表进行大量的修改或添加数据后，应该执行此语句来查看有无碎片。

显示指定的表或视图的数据和索引的碎片信息。

语法如下：

```
DBCC SHOWCONTIG
[(
    { table_name | table_id | view_name | view_id }
    [ , index_name | index_id ]
)]
    [ WITH
        {
         [ , [ ALL_INDEXES ] ]
         [ , [ TABLERESULTS ] ]
         [ , [ FAST ] ]
         [ , [ ALL_LEVELS ] ]
         [ NO_INFOMSGS ]
         }
    ]
```

DBCC SHOWCONTIG 语句的参数及说明如表 14.3 所示。

表 14.3　DBCC SHOWCONTIG 语句的参数及说明

参　数	描　述
table_name | table_id | view_name | view_id	要检查碎片信息的表或视图。如果未指定，则检查当前数据库中的所有表和索引视图
index_name | index_id	要检查碎片信息的索引。如果未指定，则该语句将处理指定表或视图的基本索引
WITH	指定有关 DBCC 语句返回的信息类型的选项
FAST	指定是否要对索引执行快速扫描和输出最少信息。快速扫描不读取索引的叶级或数据级页
ALL_INDEXES	显示指定表和视图的所有索引的结果，即使指定了特定索引也是如此
TABLERESULTS	将结果显示为含附加信息的行集
ALL_LEVELS	仅为保持向后兼容性而保留
NO_INFOMSGS	取消严重级别从 0 到 10 的所有信息性消息

【例 14.21】显示 db_2012 数据库中 Student 表的碎片信息，SQL 语句及运行结果如图 14.13 所示。（实例位置：光盘\TM\sl\14\21）

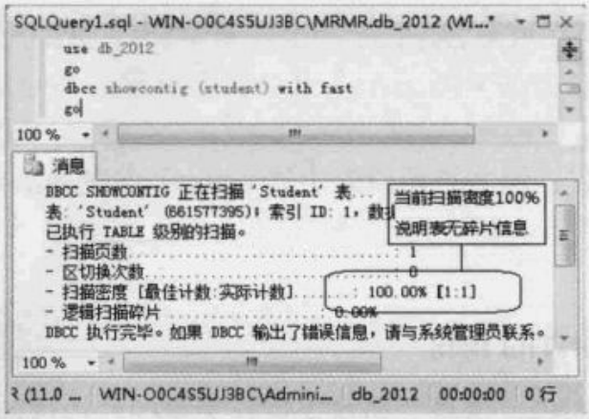


图 14.13　Student 表的碎片信息

SQL 语句如下：

```
USE db_2012
GO
DBCC SHOWCONTIG (Student) WITH FAST
GO
```

说明

当扫描密度为 100%时，说明表无碎片信息。

2. 使用 DBCC DBREINDEX 语句

DBCC DBREINDEX 表示对指定数据库中的表重新生成一个或多个索引。语法如下：

```
DBCC DBREINDEX
(
    table_name
    [ , index_name [ , fillfactor ] ]
)
    [ WITH NO_INFOMSGS ]
```

参数说明如下。

☑ table_name：包含要重新生成的指定索引的表的名称。表名称必须遵循有关标识符的规则。

☑ index_name：要重新生成的索引名。索引名称必须符合标识符规则。

☑ fillfactor：在创建或重新生成索引时，每个索引页上用于存储数据的空间百分比。

☑ WITH NO_INFOMSGS：取消显示严重级别从 0 到 10 的所有信息性消息。

【例 14.22】 使用填充因子 100 重建 db_2012 数据库中 Student 表上的 MR_Stu_Sno 聚集索引。（实例位置：光盘\TM\sl\14\22）

SQL 语句如下：

```
USE db_2012
GO
DBCC DBREINDEX('db_2012.dbo.Student',MR_Stu_Sno, 100)
GO
```

【例 14.23】 使用填充因子 100 重建 db_2012 数据库中 Student 表上的所有索引。（**实例位置：光盘\TM\sl\14\23**）

SQL 语句如下：

```
USE db_2012
GO
DBCC DBREINDEX('db_2012.dbo.Student','',100)
GO
```

3. 使用 DBCC INDEXDEFRAG 语句

DBCC INDEXDEFRAG 语句指定表或视图的索引碎片整理。语法如下：

```
DBCC INDEXDEFRAG
(
    { database_name | database_id | 0 }
        , { table_name | table_id | view_name | view_id }
    [ , { index_name | index_id } [ , { partition_number | 0 } ] ]
)
    [ WITH NO_INFOMSGS ]
```

DBCC INDEXDEFRAG 语句的参数及说明如表 14.4 所示。

表 14.4　DBCC INDEXDEFRAG 语句的参数及说明

参　数	描　述
database_name | database_id | 0	包含要进行碎片整理的索引的数据库。如果指定 0，则使用当前数据库
table_name | table_id | view_ name | view_id	包含要进行碎片整理的索引的表或视图
index_name | index_id	要进行碎片整理的索引的名称或 ID。如果未指定，该语句将针对指定表或视图的所有索引进行碎片整理
partition_number | 0	要进行碎片整理的索引的分区号。如果未指定或指定 0，该语句将对指定索引的所有分区进行碎片整理
WITH NO_INFOMSGS	取消严重级别从 0 到 10 的所有信息性消息

【例 14.24】 清除 db_2012 数据库中 Student 表的 MR_Stu_Sno 索引上的碎片。（**实例位置：光盘\TM\sl\14\24**）

SQL 语句如下：

```
USE db_2012
GO
DBCC INDEXDEFRAG (db_2012,Student,MR_Stu_Sno)
GO
```

14.6　全文索引

视频讲解：光盘\TM\lx\14\全文索引.mp4

全文索引是一种特殊类型的基于标记的功能性索引，它是由 Microsoft SQL Server 全文引擎生成

和维护的。生成全文索引的过程不同于生成其他类型的索引。全文引擎并非基于特定行中存储的值来构造 B 树结构，而是基于要编制索引的文本中的各个标记来生成倒排、堆积且压缩的索引结构。

14.6.1 使用企业管理器启用全文索引

操作步骤如下：

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2012 数据库。

（2）选择指定的数据库 db_2012，然后鼠标右键单击要创建索引的表，在弹出的快捷菜单中选择“全文索引”/“定义全文索引”命令，弹出“新建索引”窗体，如图 14.14 所示。

图 14.14 选择创建全文索引

（3）打开“全文索引向导”窗口，如图 14.15 所示。

（4）单击“下一步”按钮，选择“唯一索引”，如图 14.16 所示。

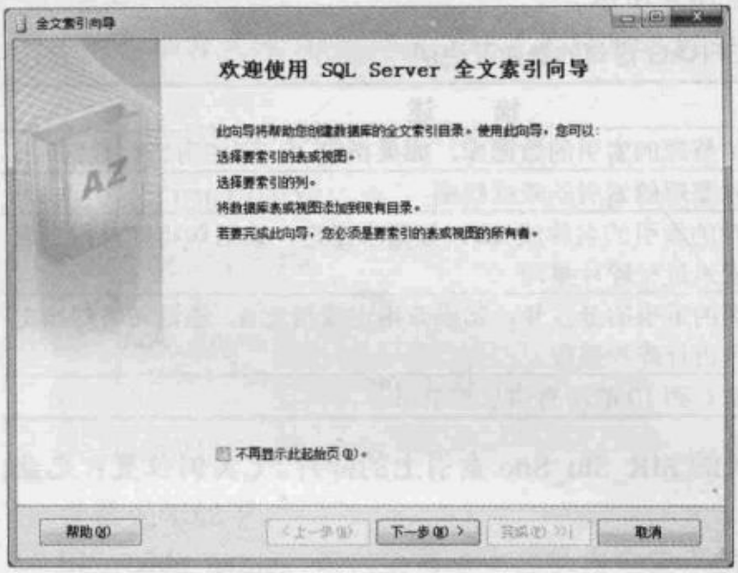


图 14.15 全文索引向导

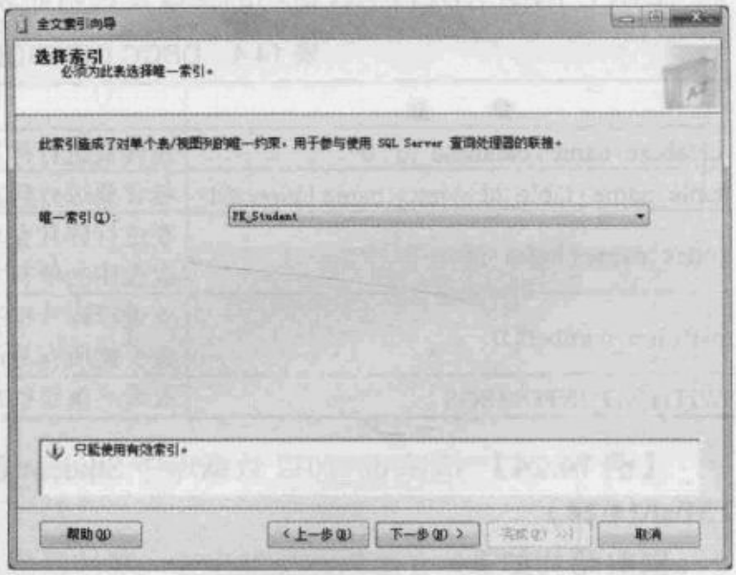


图 14.16 选择索引

（5）单击“下一步”按钮，选择表列，如图 14.17 所示。

（6）单击“下一步”按钮，选择跟踪表和视图更新的方式，如图 14.18 所示。

（7）单击“下一步”按钮，在弹出的窗口中选中“创建新目录”复选框，在“名称”文本框中输入全文目录的名称，如图 14.19 所示。

（8）单击“下一步”按钮，弹出“定义填充计划”界面，如图 14.20 所示，此窗口用来创建或修改此全文目录的填充计划（此计划是可选的）。在该窗口中单击“新建表计划”或“新建目录计划”按钮，弹出新建计划的窗体，在新建窗体中输入计划的名称，设置执行的日期和时间，单击“确定”按钮即可。

（9）单击“下一步”按钮，弹出“全文索引向导说明”界面，如图 14.21 所示。

（10）单击“完成”按钮，弹出“全文索引向导进度”界面，如图 14.22 所示。

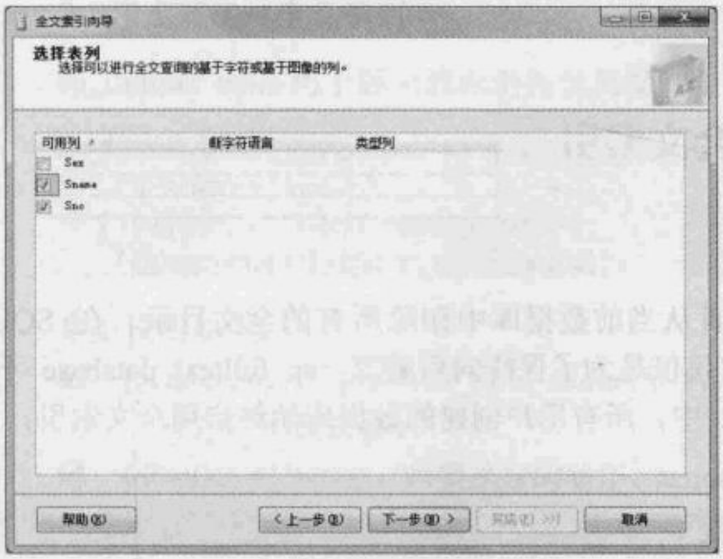


图 14.17　选择表列

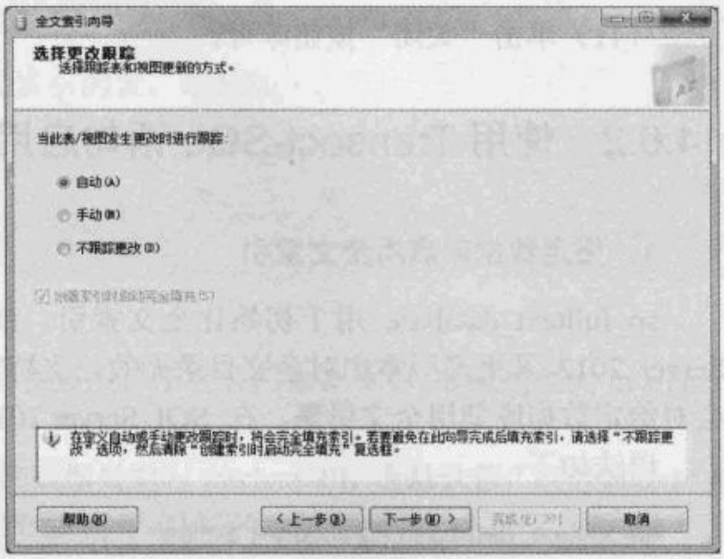


图 14.18　选择更改跟踪的方式

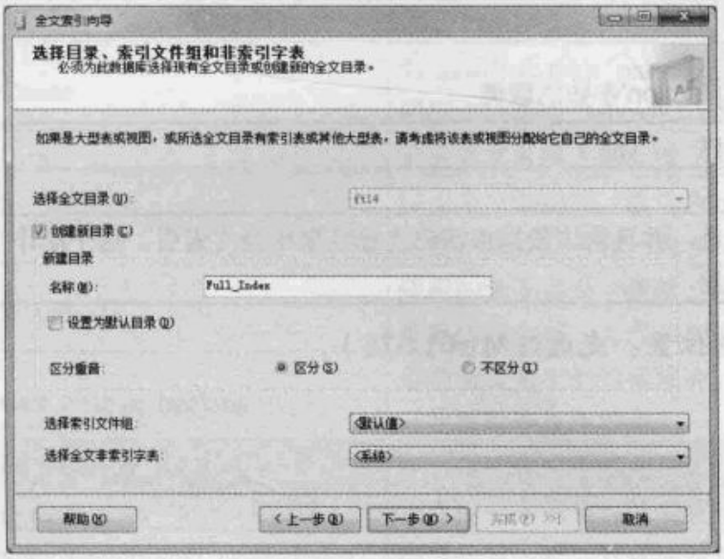


图 14.19　设置全文目录

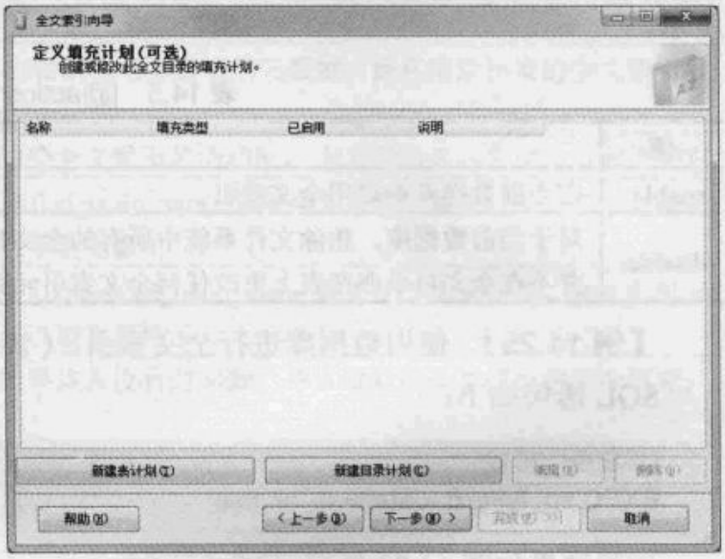


图 14.20　定义填充计划

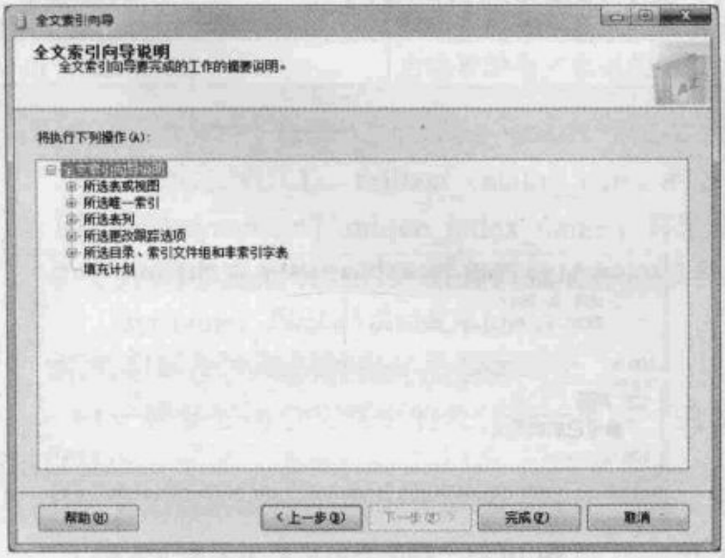


图 14.21　全文索引向导说明

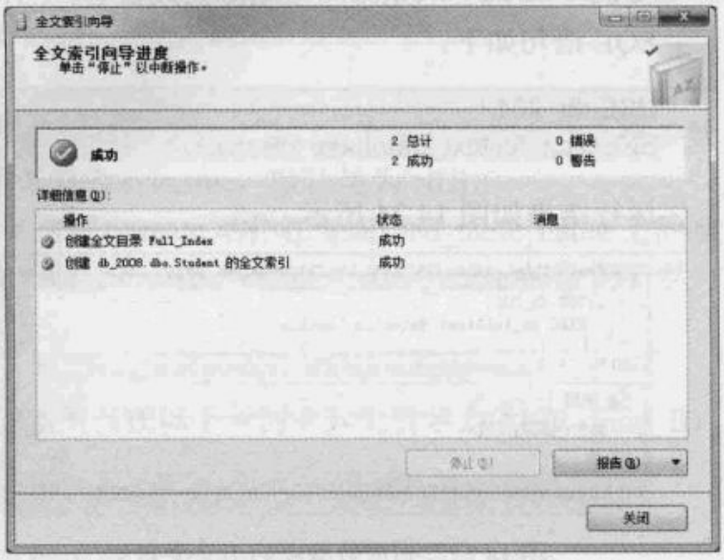


图 14.22　全文索引向导进度

（11）单击“关闭”按钮即可。

14.6.2 使用 Transact-SQL 语句启用全文索引

1. 指定数据库启用全文索引

sp_fulltext_database 用于初始化全文索引，或者从当前数据库中删除所有的全文目录。在 SQL Server 2012 及更高版本中对全文目录无效，支持它仅仅是为了保持向后兼容。sp_fulltext_database 不会对给定数据库禁用全文引擎。在 SQL Server 2012 中，所有用户创建的数据库始终启用全文索引。

语法如下：

```
sp_fulltext_database [@action=] 'action'
```

参数[@action=] 'action'表示要执行的操作。action 的数据类型为 varchar(20)，参数取值如表 14.5 所示。

表 14.5 [@action =] 'action'参数的取值

值	描　述
enable	在当前数据库中启用全文索引
disable	对于当前数据库，删除文件系统中所有的全文目录，并且将该数据库标记为已经禁用全文索引。这个动作并不在全文目录或在表上更改任何全文索引元数据

【例 14.25】 使用数据库进行全文索引。（**实例位置：光盘\TM\sl\14\25**）

SQL 语句如下：

```
USE db_2012
EXEC sp_fulltext_database 'enable'
```

运行结果如图 14.23 所示。

【例 14.26】 从数据库中删除全文索引。（**实例位置：光盘\TM\sl\14\26**）

SQL 语句如下：

```
USE db_2012
EXEC sp_fulltext_database 'disable'
```

运行结果如图 14.24 所示。

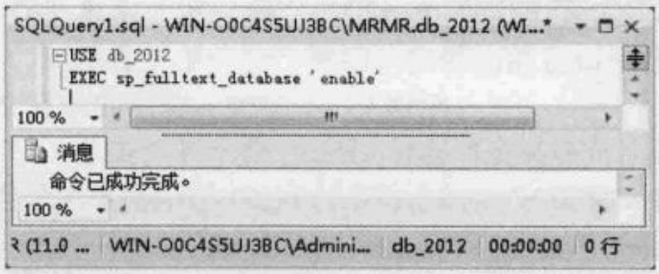


图 14.23 当前数据库启用全文索引

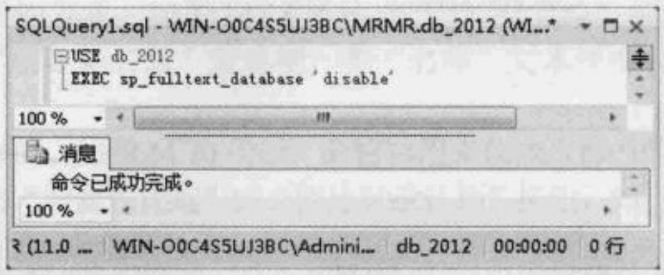


图 14.24 删除当前数据库的全文索引

2. 指定表启用全文索引

sp_fulltext_table 用于标记或取消标记要编制全文索引的表。语法如下：

```
sp_fulltext_table [ @tabname = ] 'qualified_table_name'
    , [ @action = ] 'action'
    [ , [ @ftcat = ] 'fulltext_catalog_name'
    , [ @keyname = ] 'unique_index_name' ]
```

参数说明如下。

☑ [@tabname =] 'qualified_table_name'：表名。该表必须存在当前的数据库中。数据类型为 nvarchar（517），无默认值。

☑ [@action =] 'action'：将要执行的动作。action 的数据类型为 varchar(20)，无默认值，取值如表 14.6 所示。

表 14.6　[@action =] 'action'参数的取值

值	描　述
Create	为 qualified_table_name 引用的表创建全文索引的元数据，并且指定该表的全文索引数据应该驻留在 fulltext_catalog_name 中
Drop	除去全文索引上的元数据。如果全文索引是活动的，那么在除去它之前会自动停用它
Activate	停用全文索引后，激活为 qualified_table_name 聚集全文索引的数据。在激活全文索引之前，应该至少有一列参与这个全文索引
Deactivate	停用的全文索引，使得无法再为 qualified_table_name 聚集全文索引数据。全文索引元数据依然保留，并且该表还可以被重新激活
start_change_tracking	启动全文索引的增量填充。如果该表没有时间戳，那么就启动全文索引的完全填充，开始跟踪表发生的变化
stop_change_tracking	停止跟踪表发生的变化
update_index	将当前一系列跟踪的变化传播到全文索引
Start_background_updateindex	在变化发生时，开始将跟踪的变化传播到全文索引
Stop_background_updateindex	在变化发生时，停止将跟踪的变化传播到全文索引
start_full	启动表的全文索引的完全填充
start_incremental	启动表的全文索引的增量填充

☑ [@ftcat =] 'fulltext_catalog_name'：create 动作有效的全文目录名。对于所有其他动作，该参数必须为 NULL。fulltext_catalog_name 的数据类型为 sysname，默认值为 NULL。

☑ [@keyname =] 'unique_index_name'：有效的单键列，create 动作在 qualified_table_name 上的唯一的非空索引。对于所有其他动作，该参数必须为 NULL。unique_index_name 的数据类型为 sysname，默认值为 NULL。

用表启用全文索引的操作步骤如下：

（1）将要启用全文索引的表创建一个唯一的非空索引（在以下示例中其索引名为“MR_Emp_ID_FIND”）。

（2）用表所在的数据库启用全文索引。

（3）在该数据库中创建全文索引目录（在以下示例中全文索引目录为 ML_Employ）。

（4）用表启用全文索引标记。
（5）向表中添加索引字段。
（6）激活全文索引。
（7）启动完全填充。

【例 14.27】 创建一个全文索引标记，并在全文索引中添加字段。（实例位置：光盘\TM\sl\14\27）

SQL 语句如下：

```
--将 Employee 表设为唯一索引
CREATE UNIQUE CLUSTERED INDEX MR_Emp_ID_FIND ON Employee (ID)
WITH IGNORE_DUP_KEY
--判断 db_2012 数据库是否可以创建全文索引
if (select DatabaseProperty('db_2012','IsFulltextEnabled'))=0
EXEC sp_fulltext_database 'enable'                                    --数据库启用全文索引
EXEC sp_fulltext_catalog 'ML_Employ','create'                         --创建全文索引目录为 ML_Employ
EXEC sp_fulltext_table 'Employee','create','ML_Employ','MR_Emp_ID_FIND'  --表启用全文索引标记
EXEC sp_fulltext_column 'Employee','Name','add'                       --添加全文索引字段
EXEC sp_fulltext_table 'Employee','activate'                          --激活全文索引
EXEC sp_fulltext_catalog 'ML_Employ','start_full'                     --启动表的全文索引的完全填充
```

14.6.3 使用 Transact-SQL 语句删除全文索引

DROP FULLTEXT INDEX 从指定的表或索引视图中删除全文索引。语法如下：

```
DROP FULLTEXT INDEX ON table_name
```

参数 table_name 表示包含要删除的全文索引的表或索引视图的名称。

【例 14.28】 删除 Employee 数据表的全文索引 MR_Emp_ID_FIND。（实例位置：光盘\TM\sl\14\28）

SQL 语句如下：

```
USE db_2012
DROP FULLTEXT INDEX ON Employee
```

14.6.4 全文目录

对于 SQL Server 2012 数据库，全文目录为虚拟对象，并不属于任何文件组；它是一个表示一组全文索引的逻辑概念。

1. 全文目录的创建、删除和重创建

sp_fulltext_catalog 用于创建和删除全文目录，并启动和停止目录的索引操作。可为每个数据库创建多个全文目录。

注意

在以后的 SQL Server 版本中，将删除 sp_fulltext_catalog 存储过程。所以应避免在新的开发工作中使用此功能，并计划修改当前使用该存储过程的应用程序。

语法如下：

```
sp_fulltext_catalog [ @ftcat = ] 'fulltext_catalog_name' ,
    [ @action = ] 'action'
    [ , [ @path = ] 'root_directory' ]
```

参数说明如下。

☑ [@ftcat =] 'fulltext_catalog_name'：全文目录的名称。对于每个数据库，目录名必须是唯一的。其数据类型为 sysname。

☑ [@action =] 'action'：将要执行的动作。action 的数据类型为 varchar(20)，取值如表 14.7 所示。

表 14.7　[@action =] 'action'参数的取值

值	描　述
Create	在文件系统中创建一个空的新全文目录，并向 sysfulltextcatalogs 添加一行
Drop	将全文目录从文件系统中删除，并且删除 sysfulltextcatalogs 中相关的行
start_incremental	启动全文目录的增量填充。如果目录不存在，就会显示错误
start_full	启动全文目录的完全填充。即使与此全文目录相关联的每一个表的每一行都进行过索引，也会对其检索全文索引
Stop	停止全文目录的索引填充。如果目录不存在，就会显示错误。如果已经停止了填充，那么并不会显示警告
Rebuild	重建全文目录，方法是从文件系统中删除现有的全文目录，然后重建全文目录，并使该全文目录与所有带有全文索引引用的表重新建立关联

【例 14.29】 创建一个空的全文目录 QWML。（**实例位置：光盘\TM\sl\14\29**）

SQL 语句如下：

```
USE db_2012
GO
EXEC sp_fulltext_database 'enable'      --数据库启用全文索引
EXEC sp_fulltext_catalog 'QWML','create'
```

【例 14.30】 重新创建一个已有的全文目录 QWML。（**实例位置：光盘\TM\sl\14\30**）

SQL 语句如下：

```
USE db_2012
GO
EXEC sp_fulltext_database 'enable'      --数据库启用全文索引
EXEC sp_fulltext_catalog 'QWML','rebuild'
```

运行结果如图 14.25 所示。

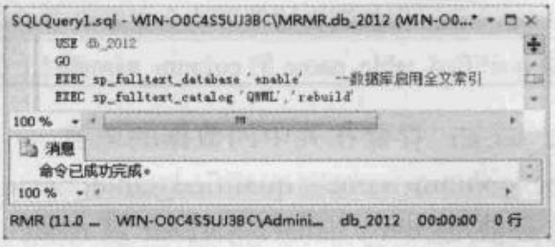


图 14.25　重建一个全文目录

【例 14.31】 删除全文目录 QWML。（实例位置：光盘\TM\sl\14\31）

SQL 语句如下：

```
USE db_2012
GO
EXEC sp_fulltext_catalog 'QWML','Drop'
```

2. 向全文目录中增加、删除列

sp_fulltext_column 指定表的某个特定列是否参与全文索引。

注意

后续版本的 Microsoft SQL Server 将删除该功能。请避免在新的开发工作中使用该功能，并着手修改当前还在使用该功能的应用程序。

语法如下：

```
sp_fulltext_column [ @tabname= ] 'qualified_table_name' ,
    [ @colname= ] 'column_name' ,
    [ @action= ] 'action'
    [ , [ @language= ] 'language_term' ]
    [ , [ @type_colname= ] 'type_column_name' ]
```

参数说明如下。

☑ [@tabname=] 'qualified_table_name'：由一部分或两部分组成的表的名称。表必须在当前数据库中。表必须有全文索引。qualified_table_name 的数据类型为 nvarchar(517)，无默认值。

☑ [@colname=] 'column_name'：qualified_table_name 中列的名称。列必须为字符列、varbinary(max)列或 image 列，不能是计算列。column_name 的数据类型为 sysname，无默认值。

注意

SQL Server 可以为存储在数据类型为 varbinary(max)或 image 的列中的文本数据创建全文索引。不对图像和图片进行索引。

☑ [@action=] 'action'：要执行的操作。action 的数据类型为 varchar(20)，无默认值，可以是表 14.8 中的列值之一。

表 14.8 [@action =] 'action'参数的取值

值	描　述
add	将 qualified_table_name 的 column_name 添加到表的非活动全文索引中。该动作启用全文索引的列
drop	从表的非活动全文索引中删除 qualified_table_name 的 column_name

☑ [@language=] 'language_term'：存储在列中的数据的语言。

☑ [@type_colname =] 'type_column_name'：qualified_table_name 中列的名称，用于保存 column_name 的文档类型。此列必须是 char、nchar、varchar 或 nvarchar。仅当 column_name 数据类型为 varbinary(max)或 image 时才使用该列。type_column_name 的数据类型为 sysname，无默认值。

【例 14.32】 将 Student 表的 Sex 列添加到表的全文索引。（**实例位置：光盘\TM\sl\ 14\32**）

SQL 语句如下：

```
USE db_2012
EXEC sp_fulltext_column Student, Sex, 'add'
```

运行结果如图 14.26 所示。

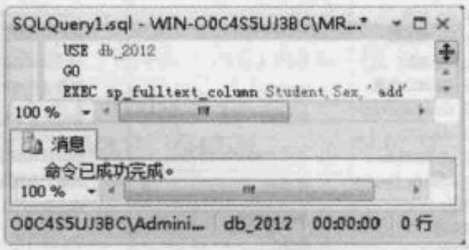


图 14.26　在列中添加表的全文索引

【例 14.33】 将 Student 表的 Sex 列从全文索引中删除。（**实例位置：光盘\TM\sl\14\33**）

SQL 语句如下：

```
USE db_2012
EXEC sp_fulltext_column Student, Sex, 'drop'
```

3. 激活全文目录

要激活表 Student 的全文目录，首先要在表中创建全文索引。

【例 14.34】 激活 Employee 表中的全文目录。（**实例位置：光盘\TM\sl\14\34**）

SQL 语句如下：

```
USE db_2012
EXEC sp_fulltext_table 'Employee','activate'
```

这样就完成了对全文目录的定义，如果要对创建的全文目录进行初始化填充，可以使用如下 SQL 语句：

```
USE db_2012
EXEC sp_fulltext_table 'Employee','start_full'
```

填充（也称为爬网）：创建和维护全文索引的过程。

14.6.5　全文目录的维护

1. 用企业管理器来维护全文目录

操作步骤如下：

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2012 数据库。

（2）选择指定数据库中的数据表（这里以 db_2012 数据库中的 Employee 表为例，该表已经创建全文索引）。

（3）在 Employee 表上单击鼠标右键，在弹出的快捷菜单中选择“全文索引”命令，如图 14.27 所示。

图 14.27　维护全文目录

（4）在“全文索引”的级联菜单中就可以对全文目录进行修改，具体功能如表 14.9 所示。

表 14.9　维护全文目录

选　项	描　述
删除全文索引	将选定的表从它的全文目录中删除
启动完全填充	使用选定表中的全部行对全文目录进行初始的数据填充
启动增量填充	识别选定的表从最后一次填充所发生的数据变化，并利用最后一次添加、删除或修改的行对全文索引进行填充
停止填充	终止当前正在运行的全文索引填充任务
手动跟踪更改	手动的方式使应用程序可以仅获取对用户表所做的更改以及与这些更改有关的信息
自动跟踪更改	自动使应用程序可以仅获取对用户表所做的更改以及与这些更改有关的信息
禁止跟踪更改	不让应用程序获取对用户表所做的更改以及与这些更改有关的信息
应用跟踪的更改	使应用程序获取对用户表所做的更改及与这些更改有关的信息

2．使用 T-SQL 语句维护全文目录

下面以 Employee 表为例介绍如何使用 T-SQL 语句维护全文目录，Employee 为已经创建全文索引的数据表。

（1）完全填充。

```
EXEC sp_fulltext_table 'Employee','start_full'
```

（2）增量填充。

```
EXEC sp_fulltext_table 'Employee','start_incremental'
```

（3）更改跟踪。

```
EXEC sp_fulltext_table ' Employee ','start_change_tracking'
```

（4）后台更新。

```
EXEC sp_fulltext_table ' Employee ','start_background_updateindex'
```

（5）清除无用的全文目录。

```
EXEC sp_fulltext_service 'clean_up'
```

（6）sp_help_fulltext_catalogs。

返回指定的全文目录的 ID（ftcatid）、名称（NAME）、根目录（PATH）、状态（STATUS）以及全文索引表的数量（NUMBER_FULLTEXT_TABLES）。

【例 14.35】 返回有关全文目录 QWML 的信息。（**实例位置：光盘\TM\sl\14\35**）

SQL 语句如下：

```
USE db_2012
GO
EXEC sp_help_fulltext_catalogs 'QWML' ;
GOSTATUS
```

运行结果如图 14.28 所示。

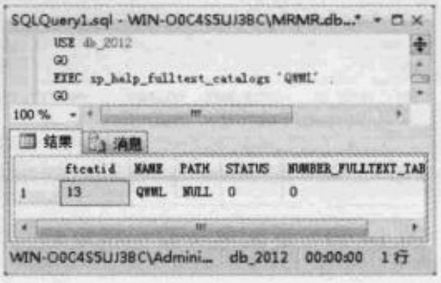


图 14.28　返回全文目录 MR 的信息

列将返回指定全文目录的当前状态，如表 14.10 所示。

表 14.10　STATUS 列的返回状态

返　回　值	描　　述	返　回　值	描　　述
0	空闲	5	关闭
1	正在进行完全填充	6	正在进行增量填充
2	暂停	7	生成索引
3	已中止	8	磁盘已满，已暂停
4	正在恢复	9	更改跟踪

（7）sp_help_fulltext_tables。

该存储过程返回为全文索引注册的表的列表。

【例 14.36】 返回包含在指定全文目录 QWML 中的表的信息。（**实例位置：光盘\TM\sl\14\36**）

SQL 语句如下：

```
USE db_2012
EXEC sp_help_fulltext_tables 'QWML"
```

运行结果如图 14.29 所示。

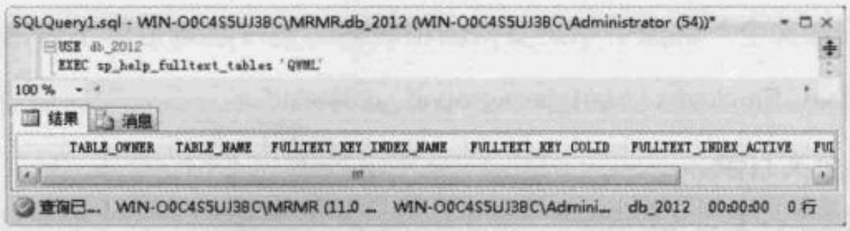


图 14.29　返回包含在指定全文目录

（8）sp_help_fulltext_columns。

该存储过程返回为全文索引指定的列。

【例 14.37】 返回 Inx_table 表中全文索引，Inx_table 表为已创建全文索引的数据表。（**实例位置：光盘\TM\sl\14\37**）

SQL 语句如下：

```
USE db_2012
EXEC sp_help_fulltext_columns 'Inx_table'
```

运行结果如图 14.30 所示。

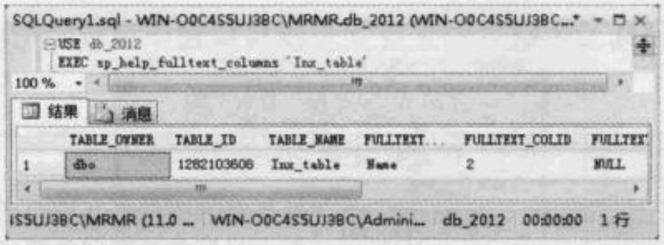


图 14.30　返回全文索引指定的列

14.7　数据完整性

视频讲解：光盘\TM\lx\14\数据完整性.mp4

数据完整性是 SQL Server 用于保证数据库中数据一致性的一种机制，防止非法数据存入数据库。具体的数据完整性主要体现在以下几点。

☑　数据类型准确无误。

☑　数据取值符合规定的范围。

☑　多个数据表之间的数据不存在冲突。

下面介绍 SQL Server 2012 提供的 4 种数据完整性机制：域完整性、实体完整性、引用完整性和用户定义完整性。

14.7.1　域完整性

域是指数据表中的列（字段），域完整性就是指列的完整性。实现域完整性的方法有：限制类型（通过数据类型）、格式（通过 CHECK 约束和规则）或可能的取值范围（通过 CHECK 约束、DEFAULT

定义、NOT NULL 定义和规则）等，它要求数据表中指定列的数据具有正确的数据类型、格式和有效的数据范围。

域完整性常见的实现机制包括以下几种。

☑ 默认值（Default）。

☑ 检查（Check）。

☑ 外键（Foreign Key）。

☑ 数据类型（Data Type）。

☑ 规则（Rule）。

【例 14.38】 创建表 student2，有学号、最好成绩和平均成绩 3 列，且最好成绩必须大于平均成绩。（**实例位置：光盘\TM\sl\14\38**）

SQL 语句如下：

```
CREATE TABLE student2
(
学号   char(6) not null,
最好成绩   int   not null,
平均成绩   int   not null,
CHECK(最好成绩>平均成绩)
)
```

运行结果如图 14.31 所示。

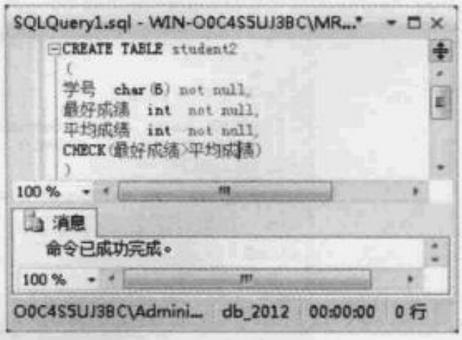


图 14.31　域完整性

14.7.2　实体完整性

现实世界中，任何一个实体都有区别于其他实体的特征，即实体完整性。在 SQL Server 数据库中，实体完整性是指所有的记录都应该有一个唯一的标识，以确保数据表中数据的唯一性。

如果将数据库中数据表的第一行看作一个实体，可以通过以下几项实施实体完整性。

☑ 唯一索引（Unique Index）。

☑ 主键（Primary Key）。

☑ 唯一码（Unique Key）。

☑ 标识列（Identity Column）。

【例 14.39】 创建表 student3，并对借书证号字段创建 PRIMARY KEY 约束，对姓名字段定义 UNIQUE 约束。（**实例位置：光盘\TM\sl\14\39**）

SQL 语句如下：

```
USE db_2012
Go
CREATE TABLE student3
(
借书证号 char(8)   not null   CONSTRAINT   py   PRIMARY KEY,
```

```
姓名 char(8)  not null  CONSTRAINT  uk  UNIQUE,
专业  char(12) not null,
性别  bit  not null,
借书量 int  CHECK(借书量>=0 AND 借书量<=20) null
)
go
```

运行结果如图 14.32 所示。

【例 14.40】 创建表 student4，由借书证号、索书名、借书时间作为联合主键。（**实例位置：光盘\TM\sl\14\40**）

SQL 语句如下：

```
Use db_2012
CREATE TABLE student4
(
 借书证号 char(8)  not null,
 索书名   char(10) not null,
 借书时间 date  not null,
 还书时间 date  not null,
PRIMARY KEY(索书名,借书证号,借书时间)
)
```

运行结果如图 14.33 所示。

图 14.32 实体完整性设置

图 14.33 联合主键

14.7.3 引用完整性

引用完整性又称参照完整性，引用完整性保证主表中的数据与从表中数据的一致性。SQL Server 2012 中，参照完整性的实现是通过定义外键与主键之间或外键与唯一键之间的对应关系实现的。参照完整性确保键值在所有表中一致。参照完整性的实现方法如下：

（1）外键（Foreign Key）。

（2）检查（Check）。

（3）触发器（Trigger）。

（4）存储过程（Stored Procedure）

【例 14.41】 创建表 student5，要求表中所用的索书名、借书证号和借书时间组合都必须出现在 student4 表中。（**实例位置：光盘\TM\sl\14\41**）

SQL 语句如下：

```
Use db_2012
CREATE TABLE student5
(
借书证号 char(8)  NOT NULL,
ISBN char(16) NOT NULL,
索书名 char(10) NOT NULL,
借书时间 date NOT NULL,
还书时间 date NOT NULL,
CONSTRAINT FK_point FOREIGN KEY (索书名,借书证号,借书时间)
REFERENCES student4 (索书名,借书证号,借书时间)
ON DELETE NO ACTION
)
```

运行结果如图 14.34 所示。

图 14.34　引用完整性

14.7.4　用户定义完整性

用户定义完整性使用户可以定义不属于其他任何完整性类别的特定业务规则。所有完整性类别都支持用户定义完整性，这包括 CREATE TABLE 中所有列级约束和表级约束、存储过程以及触发器。

14.8　小　　结

本章介绍索引的建立、删除、分析与维护，以及 4 种数据完整性。读者在了解索引概念的前提下，可以使用 SQL Server Management Studio 或者 SQL 语句来建立和删除索引，进而对索引进行分析和维护，以优化对数据的访问。为了保证存储数据的合理性，读者应了解域完整性、实体完整性和引用完整性。

14.9　实践与练习

1．基于表 student 和 student1 创建一个视图，并在该视图上创建一个索引并查询数据。（**答案位置：光盘\TM\sl\14\42**）

2．创建一个多字段非聚集索引检索数据，具体实现时，为员工表（employee1）的 Name 列和 Age 列创建索引，在创建的索引中，Name 字段的优先级要高于 Age 字段。在创建多字段的索引时，各字段的排列顺序决定了其优先级，排列的字段越靠前，则具有越高的优先级。（**答案位置：光盘\TM\sl\14\43**）

3．在 Student 表的 Sno 字段上创建索引，并用 DBCC CHECKTABLE 命令检查 Student 表中的 I_Stu 索引的完整性。（**答案位置：光盘\TM\sl\14\44**）

第15章

SQL 中的事务

（视频讲解：28 分钟）

在数据提交过程中，事务非常重要，它是一个独立的工作单元，如果某一事务成功，则在该事务中进行的所有数据修改均会提交，成为数据库中的永久组成部分，如果事务遇到错误且必须取消或回滚，则所有数据修改均被清除，本章将从事务概念、隐式与显式事务、使用事务、事务工作机制、事务并发、锁和分布式事务处理等多个方面对 SQL 事务进行详细讲解。

通过阅读本章，您可以：

- 理解事务的概念
- 掌握显式事务与隐式事务
- 掌握如何使用事务
- 理解事务的工作机制
- 理解自动提交事务
- 理解事务的并发问题
- 理解事务的隔离级别
- 掌握锁的机制
- 了解死锁的产生原理
- 了解分布式事务处理

15.1　事务的概念

视频讲解：光盘\TM\lx\15\事务的概念.mp4

事务是由一系列语句构成的逻辑工作单元。事务和存储过程等批处理有一定程度上的相似之处，通常都是为了完成一定业务逻辑而将一条或者多条语句“封装”起来，使它们与其他语句之间出现一个逻辑上的边界，并形成相对独立的一个工作单元。

当使用事务修改多个数据表时，如果在处理的过程中出现了某种错误，例如系统死机或突然断电等情况，则返回结果是数据全部没有被保存。因为事务处理的结果只有两种：一种是在事务处理的过程中，如果发生了某种错误则整个事务全部回滚，使所有对数据的修改全部撤销，事务对数据库的操作是单步执行的，当遇到错误时可以随时回滚；另一种是如果没有发生任何错误且每一步的执行都成功，则整个事务全部被提交。从而可以看出，有效地使用事务不但可以提高数据的安全性，还可以增强数据的处理效率。

事务包含 4 种重要的属性，被统称为 ACID（原子性、一致性、隔离性和持久性），一个事务必须通过 ACID。

（1）原子性（Atomic）：事务是一个整体的工作单元，事务对数据库所做的操作要么全部执行，要么全部取消。如果某条语句执行失败，则所有语句全部回滚。

（2）一致性（Consistency）：事务在完成时，必须使所有的数据都保持一致状态。在相关数据库中，所有规则都必须应用于事务的修改，以保持所有数据的完整性。如果事务成功，则所有数据将变为一个新的状态；如果事务失败，则所有数据将处于开始之前的状态。

（3）隔离性（Isolated）：由事务所做的修改必须与其他事务所做的修改隔离。事务查看数据时数据所处的状态，要么是另一并发事务修改它之前的状态，要么是另一事务修改它之后的状态，事务不会查看中间状态的数据。

（4）持久性（Durability）：当事务提交后，对数据库所做的修改就会永久保存下来。

15.2　显式事务与隐式事务

视频讲解：光盘\TM\lx\15\显式事务与隐式事务.mp4

事务是单个的工作单元。如果某一事务成功，则在该事务中进行的所有数据修改均会提交，成为数据库中的永久组成部分。如果事务遇到错误且必须取消或回滚，则所有数据修改均被清除。

SQL Server 以下列事务模式运行。

- ☑ 自动提交事务：每条单独的语句都是一个事务。
- ☑ 显式事务：每个事务均以 BEGIN TRANSACTION 语句显式开始，以 COMMIT 或 ROLLBACK 语句显式结束。
- ☑ 隐式事务：在前一个事务完成时新事务隐式启动，但每个事务仍以 COMMIT 或 ROLLBACK

语句显式完成。

☑ 批处理级事务：只能应用于多个活动结果集（MARS），在 MARS 会话中启动的 Transact-SQL 显式或隐式事务变为批处理级事务。当批处理完成时没有提交或回滚的批处理级事务自动由 SQL Server 进行回滚。

本节主要介绍显式事务和隐式事务。

15.2.1 显式事务

显式事务是用户自定义或用户指定的事务。可以通过 BEGIN TRANSACTION、COMMIT TRANSACTION、COMMIT WORK、ROLLBACK TRANSACTION 或 ROLLBACK WORK 事务处理语句定义显式事务。下面将简单介绍以上几种事务处理语句的语法和参数。

（1）BEGIN TRANSACTION 语句。

用于启动一个事务，它标志着事务的开始。

语法如下：

```
BEGIN TRAN [ SACTION ] [ transaction_name | @tran_name_variable[ WITH MARK [ 'description' ] ] ]
```

参数说明如下。

☑ transaction_name：表示设定事务的名称，字符个数最多为 32 个字符。

☑ @tran_name_variable：表示用户定义的、含有有效事务名称的变量名称，必须用 char、varchar、nchar 或 nvarchar 数据类型声明该变量。

☑ WITH MARK ['description']：表示指定在日志中标记事务，description 是描述该标记的字符串。

（2）COMMIT TRANSACTION 语句。

用于标志一个成功的隐式事务或用户定义事务的结束。

语法如下：

```
COMMIT [ TRAN [ SACTION ] [ transaction_name | @tran_name_variable ] ]
```

参数说明如下。

☑ transaction_name：表示此参数指定由前面的 BEGIN TRANSACTION 指派的事务名称，此处的事务名称仅用来帮助程序员阅读，以及指明 COMMIT TRANSACTION 与哪些嵌套的 BEGIN TRANSACTION 相关联。

☑ @tran_name_variable：表示用户定义的、含有有效事务名称的变量名称，必须用 char、varchar、nchar 或 nvarchar 数据类型声明该变量。

说明

如果@@TRANCOUNT 为 1，COMMIT TRANSACTION 使得自从事务开始以来所执行的所有数据修改成为数据库的永久部分，释放连接占用的资源，并将@@TRANCOUNT 减少到 0。如果@@TRANCOUNT 大于 1，则 COMMIT TRANSACTION 使@@TRANCOUNT 按 1 递减。

（3）COMMIT WORK 语句。

用于标志事务的结束。

语法如下：

```
COMMIT [WORK]
```

此语句的功能与 COMMIT TRANSACTION 相同，但 COMMIT TRANSACTION 接受用户定义的事务名称。

（4）ROLLBACK TRANSACTION 语句。

用于将显式事务或隐式事务回滚到事务的起点或事务内的某个保存点。当执行事务的过程中发生某种错误，可以使用 ROLLBACK TRANSACTION 语句或 ROLLBACK WORK 语句，使数据库撤销在事务中所做的更改，并使数据恢复到事务开始之前的状态。

语法如下：

```
ROLLBACK [ TRAN [ SACTION ] [ transaction_name | @tran_name_variable| savepoint_name | @savepoint_variable ] ]
```

参数说明如下。

- ☑ transaction_name：表示 BEGIN TRAN 对事务名称的指派。
- ☑ @tran_name_variable：表示用户定义的、含有有效事务名称的变量名称，必须用 char、varchar、nchar 或 nvarchar 数据类型声明该变量。
- ☑ savepoint_name：是来自 SAVE TRANSACTION 语句对保存点的定义，当条件回滚只影响事务的一部分时使用 savepoint_name。
- ☑ @savepoint_variable：表示用户定义的、含有有效保存点名称的变量名称。

（5）ROLLBACK WORK 语句。

用于将用户定义的事务回滚到事务的起点。

语法如下：

```
ROLLBACK [WORK]
```

此语句的功能与 ROLLBACK TRANSACTION 相同，除非 ROLLBACK TRANSACTION 接受用户定义的事务名称。

15.2.2　隐式事务

隐式事务需要使用 SET IMPLICIT_TRANSACTIONS ON 语句将隐式事务模式设置为打开。在打开了隐式事务的设置开关时，执行下一条语句时自动启动一个新事务，并且每关闭一个事务时，执行下一条语句又会启动一个新事务，直到关闭了隐式事务的设置开关。

SQL Server 的任何数据修改语句都是隐式事务，例如，ALTER TABLE、CREATE、DELETE、DROP、FETCH、GRANT、INSERT、OPEN、REVOKE、SELECT、TRUNCATE TABLE、UPDATE。这些语句都可以作为一个隐式事务的开始。如果要结束隐式事务，需要使用 COMMIT TRANSACTION 或

ROLLBACK TRANSACTION 语句来结束事务。

15.2.3 API 中控制隐式事务

用来设置隐式事务的 API 机制是 ODBC 和 OLE DB。

（1）ODBC。

- ☑ 调用 SQLSetConnectAttr 函数启动隐式事务模式，其中，Attribute 设置为 SQL_ATTR_AUTOCOMMIT，ValuePtr 设置为 SQL_AUTOCOMMIT_OFF。
- ☑ 在调用 SQLSetConnectAttr 之前，连接将一直保持为隐式事务模式，其中，Attribute 设置为 SQL_ATTR_AUTOCOMMIT，ValuePtr 设置为 SQL_AUTOCOMMIT_ON。
- ☑ 调用 SQLEndTran 函数提交或回滚每个事务，其中，CompletionType 设置为 SQL_COMMIT 或 SQL_ROLLBACK。

（2）OLE DB。

OLE DB 没有专门用来设置隐式事务模式的方法。

- ☑ 调用 ITransactionLocal::StartTransaction 方法启动显式模式。
- ☑ 当调用 ITransaction::Commit 或 ITransaction::Abort 方法（其中，fRetaining 设置为 TRUE）时，OLE DB 将完成当前的事务并进入隐式事务模式。只要 ITransaction::Commit 或 ITransaction::Abort 中的 fRetaining 设置为 TRUE，那么连接就将保持隐式事务模式。
- ☑ 调用 ITransaction::Commit 或 ITransaction::Abort（其中 fRetaining 设置为 FALSE）停止隐式事务模式。

15.2.4 事务的 COMMIT 和 ROLLBACK

结束事务包括“成功时提交事务”和“失败时回滚事务”两种情况，在 Transact-SQL 中可以使用 COMMIT 和 ROLLBACK 结束事务。

（1）COMMIT。

提交事务，用在事务执行成功的情况下。COMMIT 语句保证事务的所有修改都被保存，同时 COMMIT 语句也释放事务中使用的资源，例如事务使用的锁。

（2）ROLLBACK。

回滚事务，用于事务在执行失败的情况下，将显式事务或隐式事务回滚到事务的起点或事务内的某个保存点。

15.3 使 用 事 务

视频讲解：光盘\TM\lx\15\使用事务.mp4

在掌握事务的概念与运行模式之后，本节继续介绍如何使用事务。

15.3.1　开始事务

当一个数据库连接启动事务时，在该连接上执行的所有 Transact-SQL 语句都是事务的一部分，直到事务结束。开始事务使用 BEGIN TRANSACTION 语句。下面将以示例的形式演示如何在 SQL 中使用开始事务。

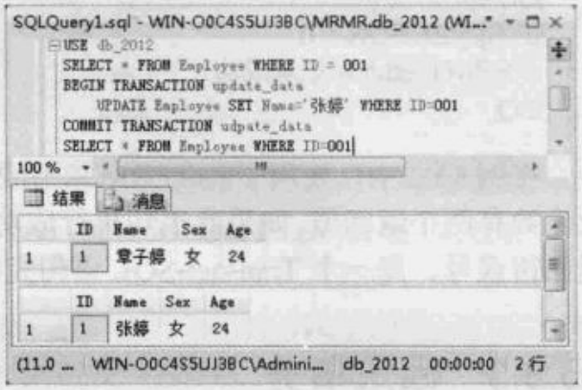


图 15.1　使用事务修改“操作员信息表”中的数据

【例 15.1】 使用事务修改 Employee 表中的数据，首先使用 BEGIN TRANSACTION 语句启动事务 update_data，然后修改指定条件的数据，最后使用 COMMIT TRANSACTION 提交事务，SQL 语句及运行结果如图 15.1 所示。（**实例位置：光盘\TM\sl\15\1**）

SQL 语句如下：

```
USE db_2012                                    --引入数据库
SELECT * FROM Employee WHERE ID = 001
BEGIN TRANSACTION update_data                  --开始事务
  UPDATE Employee SET Name = '张婷'             --修改数据
    Where ID = 1                               --条件
     COMMIT TRANSACTION update_data
     SELECT * FROM Employee   WHERE ID =001
```

在例 15.1 中，BEGIN TRANSACTION 语句指定一个事务的开始，update_data 语句为事务名称，它可由用户自定义，但必须是有效的标识符。COMMIT TRANSACTION 语句指定事务的结束。

说明

BEGIN TRANSACTION 与 COMMIT TRANSACTION 之间的语句，可以是任何对数据库进行修改的语句。

15.3.2　结束事务

当一个事务执行完成之后，要将其结束，以便释放所占用的内存资源，结束事务使用 COMMIT 语句。

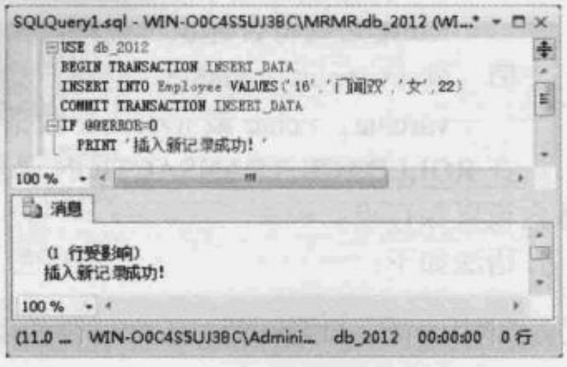


图 15.2　使用 COMMIT 结束事务

【例 15.2】 使用事务在 Employee 表中添加一条记录，并使用 COMMIT 语句结束事务，SQL 语句及运行结果如图 15.2 所示。（**实例位置：光盘\TM\sl\15\2**）

SQL 语句如下：

```
USE db_2012                                    --打开数据库
SELECT * FROM Employee
```

```
BEGIN TRANSACTION INSERT_DATA          --开始事务
  INSERT INTO Employee
  VALUES('16','门闻双','女','22')
COMMIT TRANSACTION INSERT_DATA         --结束事务
GO
IF @@ERROR = 0
  PRINT '插入新记录成功！'              --输出插入成功的信息
GO
```

在例 15.2 中，使用了@@ERROR 函数，此函数用于判断最后的 Transact-SQL 语句是否执行成功。此函数有两个返回值，如果此语句执行成功，则@@ERROR 返回 0；如果此语句产生错误，则@@ERROR 返回错误号。每一个 Transact-SQL 语句完成时，@@ERROR 的值都会改变。

15.3.3 回滚事务

使用 ROLLBACK TRANSACTION 语句可以将显式事务或隐式事务回滚到事务的起点或事务内的某个保存点。

语法如下：

```
ROLLBACK { TRAN | TRANSACTION }
      [ transaction_name | @tran_name_variable
      | savepoint_name | @savepoint_variable ]
[ ; ]
```

参数说明如下。

- ☑ transaction_name：是为 BEGIN TRANSACTION 上的事务分配的名称（即事务名称），它必须符合标识符规则，但只使用事务名称的前 32 个字符，当嵌套事务时，transaction_name 必须是最外面的 BEGIN TRANSACTION 语句中的名称。
- ☑ @tran_name_variable：是用户定义的、包含有效事务名称的变量的名称，它必须用 char、varchar、nchar 或 nvarchar 数据类型声明变量。
- ☑ savepoint_name：是 SAVE TRANSACTION 语句中的 savepoint_name（即保存点的名称），savepoint_name 必须符合标识符规则，当条件回滚应只影响事务的一部分时，可使用 savepoint_name。
- ☑ @savepoint_variable：是用户定义的、包含有效保存点名称的变量的名称，它必须用 char、varchar、nchar 或 nvarchar 数据类型声明变量。

在 ROLLBACK TRANSACTION 语句中用到了保存点，通常使用 SAVE TRANSACTION 语句在事务内设置保存点。

语法如下：

```
SAVE { TRAN | TRANSACTION } { savepoint_name | @savepoint_variable }[ ; ]
```

参数说明如下。

- ☑ savepoint_name：是保存点的名称，它必须符合标识符规则。当条件回滚应只影响事务的一部分时，可使用 savepoint_name。
- ☑ @savepoint_variable：是用户定义的、包含有效保存点名称的变量的名称，它必须用 char、

varchar、nchar 或 nvarchar 数据类型声明变量。

15.3.4　事务的工作机制

下面将通过一个示例讲解事务的工作机制。

【例 15.3】 使用事务修改 Employee 表中的数据，并将指定的员工记录删除，SQL 语句及运行结果如图 15.3 所示。（**实例位置：光盘\TM\sl\15\3**）

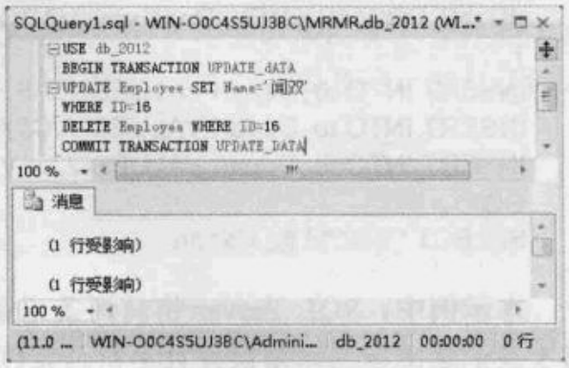


图 15.3　修改 Employee 表中的数据

SQL 语句如下：

```
USE db_2012                                    --打开数据库
SELECT * FROM Employee                         --显示 Employee 表数据
BEGIN TRANSACTION UPDATE_DAT                   --开始事务
  UPDATE Employee SET Name = '闻双'             --修改员工信息
  WHERE ID = 16
  DELETE Employee WHERE ID = 16                --删除指定的员工记录
COMMIT TRANSACTION UPDATE_DATA                 --提交事务
```

例 15.3 中的事务的工作机制可以分为以下几点：

（1）当在代码中出现 BEGIN TRANSACTION 语句时，SQL Server 将会显示事务，并会给新事务分配一个事务 ID。

（2）当事务开始后，SQL Server 将会运行事务体语句，并将事务体语句记录到事务日志中。

（3）在内存中执行事务日志中所记录的事务体语句。

（4）当执行到 COMMIT 语句时会结束事务，同时事务日志也会被写到数据库的日志设备上，从而保证日志可以被恢复。

15.3.5　自动提交事务

自动提交事务是 SQL Server 默认的事务处理方式，当任何一条有效的 SQL 语句被执行后，它对数据库所做的修改都将会被自动提交，如果发生错误，则将会自动回滚并返回错误信息。

【例 15.4】 使用 INSERT 语句向数据库中添加 3 条记录，但由于添加了重复的主键，导致最后一条 INSERT 语句在编译时产生错误，从而使这条语句没有被执行，SQL 语句及运行结果如图 15.4 所示。（**实例位置：光盘\TM\sl\15\4**）

图 15.4　自动提交事务出现错误

SQL 语句如下：

```
USE db_2012                                    --打开数据库
CREATE TABLE tb_Depart                         --创建数据表
```

```
(ID INT PRIMARY KEY, DepName VARCHAR(10)
)
INSERT INTO tb_Depart VALUES(1,'ASP.NET 部门')          --插入记录
INSERT INTO tb_Depart VALUES(2,'C#部门')                --插入记录
INSERT INTO tb_Depart VALUES(2,'JAVA 部门')             --插入记录
GO
SELECT * FROM tb_Depart                                 --检索记录
```

本示例中，SQL Server 将前两条记录添加到了指定的数据表中，而将第 3 条记录回滚，这是因为第 3 条记录出现编译错误并且不符合条件（主键不允许重复），所以被事务回滚。

15.3.6 事务的并发问题

事务的并发问题主要体现在丢失或覆盖更新、未确认的相关性（脏读）、不一致的分析（不可重复读）和幻象读 4 个方面，这些是影响事务完整性的主要因素。如果没有锁定且多个用户同时访问一个数据库，则当他们的事务同时使用相同的数据时可能会发生以上几种问题。下面分别进行说明。

（1）丢失更新。

当两个或多个事务选择同一行，然后基于最初选定的值更新该行时，会发生丢失更新问题。每个事务都不知道其他事务的存在。最后的更新将重写由其他事务所做的更新，这样就会导致数据丢失。

例如，最初有一份原始的电子文档，文档人员 A 和 B 同时修改此文档，当修改完成之后保存时，最后修改完成的文档必将替换第一个修改完成的文档，那么就造成了数据丢失更新的后果。如果文档人员 A 修改并保存之后，文档人员 B 再进行修改则可以避免该问题。

（2）未确认的相关性（脏读）。

如果一个事务读取了另外一个事务尚未提交的更新，则称为脏读。

例如，文档人员 B 复制了文档人员 A 正在修改的文档，并将文档人员 A 的文档发布，此后，文档人员 A 认为文档中存在着一些问题需要重新修改，此时文档人员 B 所发布的文档就将与重新修改的文档内容不一致。如果文档人员 A 将文档修改完成并确认无误的情况下，文档人员 B 再复制则可以避免该问题。

（3）不一致的分析（不可重复读）。

当事务多次访问同一行数据，并且每次读取的数据不同时，将会发生不一致分析问题。不一致的分析与未确认的相关性类似，因为其他事务也正在更改该数据。然而，在不一致的分析中，事务所读取的数据是由进行了更改的事务提交的。而且，不一致的分析涉及多次读取同一行，并且每次信息都由其他事务更改，因而该行被不可重复读取。

例如，文档人员 B 两次读取文档人员 A 的文档，但在文档人员 B 读取时，文档人员 A 又重新修改了该文档中的内容，在文档人员 B 第二次读取文档人员 A 的文档时，文档中的内容已被修改，此时则发生了不可重复读的情况。如果文档人员 B 在文档人员 A 全部修改后读取文档，则可以避免该问题。

（4）幻象读。

幻象读和不一致的分析有些相似，当一个事务的更新结果影响到另一个事务时，将会发生幻象读问题。事务第一次读的行范围显示出其中一行已不复存在于第二次读或后续读中，因为该行已被其他事务删除。同样，由于其他事务的插入操作，事务的第二次或后续读显示有一行已不存在于原始读中。

例如，文档人员 B 更改了文档人员 A 所提交的文档，但当文档人员 B 将更改后的文档合并到主副本时，却发现文档人员 A 已将新数据添加到该文档中。如果文档人员 B 在修改文档之前，没有任何人将新数据添加到该文档中，则可以避免该问题。

15.3.7　事务的隔离级别

当事务接受不一致的数据级别时被称为事务的隔离级别。如果事务的隔离级别比较低，会增加事务的并发问题，有效地设置事务的隔离级别可以降低并发问题的发生。

设置隔离数据可以使一个进程使用，同时还可以防止其他进程的干扰。设置隔离级别定义了 SQL Server 会话中所有 SELECT 语句的默认锁定行为，当锁定用作并发控制机制时，它可以解决并发问题。这使所有事务得以在彼此完全隔离的环境中运行，但是任何时候都可以有多个正在运行的事务。

在 SQL Server 中，可以使用 SET TRANSACTION ISOLATION LEVEL 语句来设置事务的隔离级别。

SET TRANSACTION ISOLATION LEVEL：控制由连接发出的所有 SELECT 语句的默认事务锁定行为。语法如下：

```
SET TRANSACTION ISOLATION LEVEL{ READ COMMITTED | READ UNCOMMITTED | REPEATABLE READ | SERIALIZABLE}
```

参数说明如下。

- ☑ READ COMMITTED：指定在读取数据时控制共享锁以避免脏读，但数据可在事务结束前更改，从而产生不可重复读取或幻象读取数据，该选项是 SQL Server 的默认值。
- ☑ READ UNCOMMITTED：执行脏读或 0 级隔离锁定，这表示不发出共享锁，也不接受排他锁，该选项的作用与在事务内所有语句中的所有表上设置 NOLOCK 相同，这是 4 个隔离级别中限制最小的级别。
- ☑ REPEATABLE READ：锁定查询中使用的所有数据以防止其他用户更新数据，但是其他用户可以将新的幻象读插入数据集，且幻象读包括在当前事务的后续读取中，因为并发低于默认隔离级别，所以应只在必要时才使用该选项。
- ☑ SERIALIZABLE：表示在数据集上放置一个范围锁，以防止其他用户在事务完成之前更新数据集或将行插入数据集内。

SQL Server 提供了 4 种事务的隔离级别，如表 15.1 所示。

表 15.1　事务的隔离级别

隔 离 级 别	脏　读	不可重复读	幻　象　读
Read Uncommitted（未提交读）	是	是	是
Read Committed（提交读）	否	是	是
Repeatable Read（可重复读）	否	否	是
Serializable（可串行读）	否	否	否

SQL Server 的默认隔离级别为 Read Committed，可以使用锁来实现隔离级别。

（1）Read Uncommitted（未提交读）。

此隔离级别为隔离级别中最低的级别，如果将 SQL Server 的隔离级别设置为 Read Uncommitted，

则可以对数据执行未提交读或脏读，并且等同于将锁设置为 NOLOCK。

【例 15.5】 设置未提交读隔离级别。（实例位置：光盘\TM\sl\15\5）

SQL 语句如下：

```
BEGIN TRANSACTION
UPDATE Employee SET Name = '章子婷'
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED        --设置未提交读隔离级别
COMMIT TRANSACTION
SELECT * FROM Employee
```

运行结果如图 15.5 所示。

（2）Read Committed（提交读）。

此项隔离级别为 SQL 中默认的隔离级别，将事务设置为此级别，可以在读取数据时控制共享锁以避免脏读，从而产生不可重复读取或幻象读取数据。

【例 15.6】 设置提交读隔离级别。（实例位置：光盘\TM\sl\15\6）

SQL 语句如下：

```
SET TRANSACTION ISOLATION LEVEL Read Committed
BEGIN TRANSACTION
SELECT * FROM Employee
ROLLBACK TRANSACTION
SET TRANSACTION ISOLATION LEVEL Read Committed      --设置提交读隔离级别
UPDATE Employee SET Name = '高丽'
```

运行结果如图 15.6 所示。

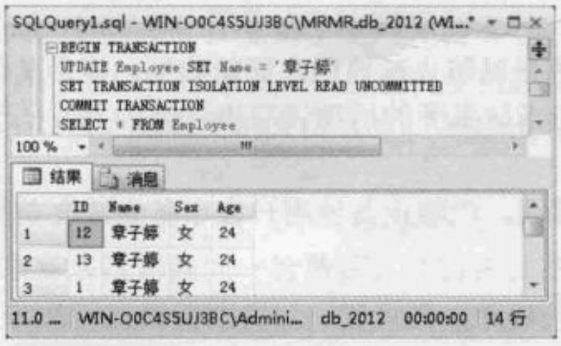


图 15.5 设置未提交读隔离级别

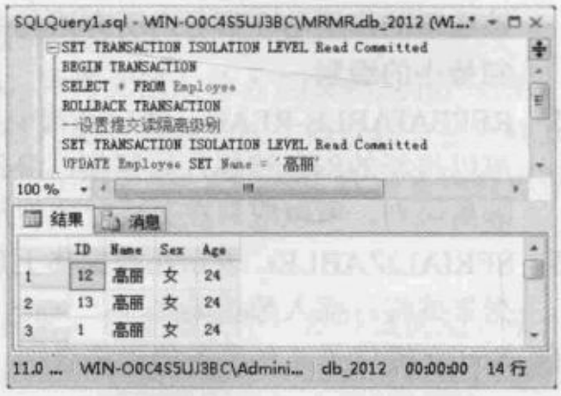


图 15.6 设置提交读隔离级别

（3）Repeatable Read（可重复读）。

此项隔离级别增加了事务的隔离级别，将事务设置为此级别可以防止脏读、不可重复读和幻象读。

【例 15.7】 设置可重复读隔离级别。（实例位置：光盘\TM\sl\15\7）

SQL 语句如下：

```
SET TRANSACTION ISOLATION LEVEL Repeatable Read
BEGIN TRANSACTION
SELECT * FROM Employee
ROLLBACK TRANSACTION
```

```
SET TRANSACTION   ISOLATION LEVEL Repeatable Read      --设置可重复读隔离级别
INSERT INTO Employee values ('18','张雨','男','22,'明日科技")
```

运行结果如图 15.7 所示。

（4）Serializable（可串行读）。

此项隔离级别是所有隔离级别中限制最大的级别，它防止了所有的事务并发问题，此级别可以适用于绝对的事务完整性的要求。

【例 15.8】 设置可串行读隔离级别。（**实例位置：光盘\TM\sl\15\8**）

SQL 语句如下：

```
SET TRANSACTION ISOLATION LEVEL Serializable
BEGIN TRANSACTION
SELECT * FROM Employee
ROLLBACK TRANSACTION
SET TRANSACTION ISOLATION LEVEL Serializable      --设置可串行读
DELETE FROM   Employee   WHERE ID = '1'
```

运行结果如图 15.8 所示。

图 15.7　设置可重复读隔离级别

图 15.8　设置可串行读

15.4　锁

视频讲解：光盘\TM\lx\15\锁.mp4

锁是一种机制，用于防止一个过程在对象上进行操作时，同某些已经在该对象上完成的事情发生冲突。锁可以防止事务的并发问题，如丢失更新、脏读、不可重复读和幻影等问题。本节主要介绍锁的机制、模式等。

15.4.1　SQL Server 锁机制

锁在数据库中是一个非常重要的概念，锁可以防止事务的并发问题，在多个事务访问下能够保证数据库完整性和一致性。例如，当多个用户同时修改或查询同一个数据库中的数据时，可能会导致数

据不一致的情况，为了控制此类问题的发生，SQL Server 引入了锁机制。

在各类数据库中所使用的锁机制基本是一致的，但也有个别上的区别。当使用数据库时，SQL Server 采用系统来管理锁，例如，当用户向 SQL Server 发送某些命令时，SQL Server 将通过满足锁的条件为数据库加上适当的锁，这也就是动态加锁。

在用户对数据库没有特定要求的情况下，通过系统自动管理锁即可满足基本的使用要求，相反，如果用户在数据库的完整性和一致性方面有特殊的要求，则需要使用锁来实现用户的要求。

15.4.2 锁模式

锁具有模式属性，它用于确定锁的用途，如表 15.2 所示。

表 15.2 锁模式

锁 模 式	描 述
共享（S）	用于不更改或不更新数据的操作（只读操作），如 SELECT 语句
更新（U）	用于可更新的资源中。防止当多个会话在读取、锁定以及随后可能进行的资源更新时发生常见形式的死锁
排他（X）	用于数据修改操作，例如 INSERT、UPDATE 或 DELETE。确保不会同时出现同一资源进行多重更新
意向	用于建立锁的层次结构。意向锁的类型为：意向共享（IS）、意向排他（IX）以及与意向排他共享（SIX）
架构	在执行依赖于表架构的操作时使用。架构锁的类型为：架构修改（Sch-M）和架构稳定性（Sch-S）
大容量更新（BU）	向表中大容量复制数据并指定了 TABLOCK 提示时使用

（1）共享锁。

共享锁用于保护读取的操作，它允许多个并发事务读取其锁定的资源。在默认情况下，数据被读取后，SQL Server 立即释放共享锁并可以对释放的数据进行修改。例如，执行查询“SELECT * FROM table1”时，首先锁定第一页，直到在读取后的第一页被释放锁时才锁定下一页。但是，事务隔离级别连接的选项设置和 SELECT 语句中的锁定设置都可以改变 SQL Server 的这种默认设置。例如，“SELECT * FROM table1 HOLDLOCK”在表的查询过程中一直保存锁定，直到查询完成才释放锁定。

（2）更新锁。

更新锁在修改操作的初始化阶段用来锁定要被修改的资源。它避免使用共享锁造成的死机现象，因为使用共享锁修改数据时，如果有两个或多个事务同时对一个事务申请了共享锁，而这些事务都将共享锁升级为排他锁，这时，这些事务都不会释放共享锁而是一直等待对方释放，这样很容易造成死锁。如果一个数据在修改前直接申请更新锁并在修改数据时升级为排他锁，就可以避免死机现象。

（3）排他锁。

排他锁是为修改数据而保留的，它锁定的资源既不能读取也不能修改。

（4）意向锁。

意向锁表示 SQL Server 在资源的底层获得共享锁或排他锁的意向。例如，表级的共享意向锁表示事务意图将排他锁释放到表的页或行中。意向锁又可以分为共享意向锁、独占意向锁和共享式独占意

向锁。共享意向锁表明事务意图锁定底层资源上放置共享锁来读取数据；独占意向锁表明事务意图锁定底层资源上放置排他锁来修改数据。共享式排他锁表明事务允许其他事务使用共享锁来读取顶层资源，并意图在该资源底层上放置排他锁。

（5）架构锁。

架构锁用于执行依赖于表架构的操作。构架锁又分为架构修改（sch-m）锁和架构稳定性 （sch-s）锁。架构修改（sch-m）锁表示执行表的数据定义语言（DDL）操作；架构稳定性（sch-s）锁表示不阻塞任何事务锁并包括排他锁。在编译查询时，其他事务（包括在表上有排他锁的事务）都能继续运行，但不能在表上执行 DDL 操作。

（6）大容量更新锁。

向表中大容量复制数据并且指定 tablock 提示，或者在 sp_tableoption 设置 table lock on bulk 表选项时而使用大容量更新锁。大容量更新锁允许进程将数据并发地大容量复制到同一表中，同时防止其他不进行大容量复制数据的进程访问该表。

15.4.3　锁的粒度

为了优化数据的并发性，可以使用 SQL Server 中锁的粒度，它可以锁定不同类型的资源。为了使锁定的成本减至最少，SQL Server 自动将资源锁定在适合任务的级别。如果锁的粒度大，则并发性高且开销大，如果锁的粒度小，则并发性低且开销小。

SQL Server 支持的锁粒度如表 15.3 所示。

表 15.3　锁的粒度

锁 大 小	描　述
行锁（RID）	行标识符。用于单独锁定表中的一行，这是最小的锁
键锁	锁定索引中的节点。用于保护可串行事务中的键范围
页锁	锁定 8KB 的数据页或索引页
扩展盘区锁	锁定相邻的 8 个数据页或索引页
表锁	锁定整个表
数据库锁	锁定整个数据库

（1）行锁（RID）。

行锁为锁的粒度当中最小的资源。行锁就是指事务在操作数据的过程中，锁定一行或多行的数据，其他事务不能同时处理这些行的数据。行级锁占用的数据资源最小，所以在事务的处理过程中，允许其他事务操作同一个表中的其他数据。

（2）页锁。

页锁是指事务在操作数据的过程中，一次锁定一页。在 SQL Server 中 25 个行锁可以升级为一个页锁，当此页被锁定后，其他事务就不能够操作此页数据，即使只锁定一条数据，那么其他事务也不能够对此页数据进行操作。从而可以看出页锁与其行锁相比，页锁占用的数据资源要多。

（3）表锁。

表锁是指事务在操作数据的过程中，锁定了整个数据表。当整个数据表被锁定后，其他事务不能

够使用此表中的其他数据。表锁的特点是使用事务处理的数据量大，并且使用较少的系统资源。但是当使用表锁时，如果所占用的数据量大，那么将会延迟其他事务的等待时间，从而降低了系统的并发性能。

（4）数据库锁。

数据库锁可锁定整个数据库，可防止任何事务或用户对此数据库进行访问，数据库锁是一种比较特殊的锁，它可以控制整个数据库的操作。

数据库锁可用于在进行数据恢复操作时，防止其他用户对此数据库进行各种操作。

15.4.4 查看锁

在 SQL Server 2012 中，查看锁的相关信息，通常使用 sys.dm_tran_locks 动态管理视图，下面来看一个示例。

【例 15.9】 使用 sys.dm_tran_locks 动态管理视图查看活动锁的信息，SQL 语句及运行结果如图 15.9 所示。（**实例位置：光盘\TM\sl\15\9**）

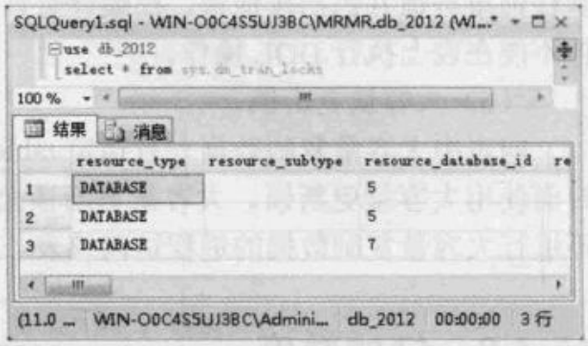


图 15.9 显示锁信息

SQL 语句如下：

```
select * from sys.dm_tran_locks
```

另外，在早期的版本中，通常使用 sp_lock 存储过程来查看，在 SQL Server 2012 数据库中，该存储过程同样适用。

语法如下：

```
sp_lock [[@spid1 =] 'spid1'] [,[@spid2 =] 'spid2']
```

参数说明如下。

- ☑ [@spid1 =] 'spid1'：表示来自 master.dbo.sysprocesses 的 SQL Server 进程 ID 号，spid1 的数据类型为 int，默认值为 NULL，执行 sp_who 可获取有关该锁的进程信息，如果没有指定 spid1，则显示所有锁的信息。
- ☑ [@spid2 =] 'spid2'：用于检查锁信息的另一个 SQL Server 进程 ID 号，spid2 的数据类型为 int，默认设置为 NULL，spid2 为可以与 spid1 同时拥有锁的另一个 spid，用户可以获取有关它的信息。

15.4.5 死锁

当两个或多个线程之间有循环相关性时，将会产生死锁。死锁是一种可能发生在任何多线程系统中的状态，而不仅发生在关系数据库管理系统中。多线程系统中的一个线程可能获取一个或多个资源（如锁）。如果正获取的资源当前为另一线程所拥有，则第一个线程可能必须等待拥有线程释放目标资源，这时就说等待线程在哪个特定资源上与拥有线程有相关性。

在数据库系统中，如果多个进程分别锁定了一个资源，并又要访问已经被锁定的资源，则此时就

会产生死锁，同时也会导致多个进程都处于等待的状态。在事务提交或回滚之前两个线程都不能释放资源，而且它们因为正等待对方拥有的资源而不能提交或回滚事务。

例如，事务 A 的线程 T1 具有 Supplier 表上的排他锁。事务 B 的线程 T2 具有 Part 表上的排他锁，并且之后需要 Supplier 表上的锁。事务 B 无法获得这一锁，因为事务 A 已拥有它。事务 B 被阻塞，等待事务 A。然而，事务 A 需要 Part 表的锁，但又无法获得锁，因为事务 B 将它锁定了。

程序示意图如图 15.10 所示。

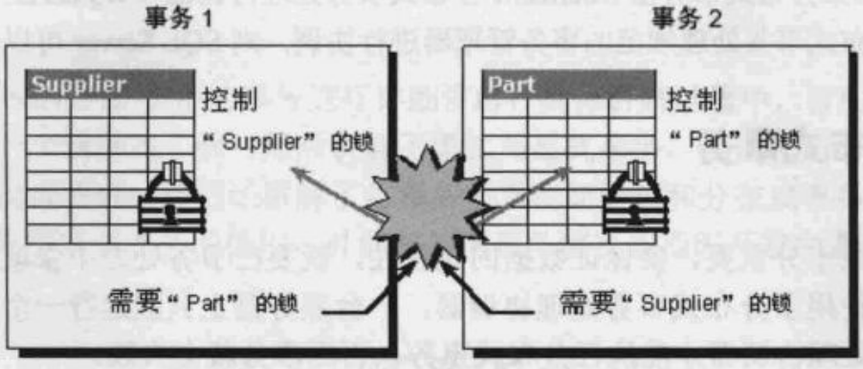


图 15.10　死锁示意图

在图 15.10 中，对于 Part 表锁资源，线程 T1 在线程 T2 上具有相关性。同样，对于 Supplier 表锁资源，线程 T2 在线程 T1 上具有相关性。因为这些相关性形成了一个循环，所以在线程 T1 和线程 T2 之间存在死锁。

说明

事务在提交或回滚之前不能释放持有的锁。因为事务需要对方控制的锁才能继续操作，所以它们不能提交或回滚。BEGIN TRANSACTION 与 COMMIT TRANSACTION 之间的语句，可以是任何对数据库进行修改的语句。

可以使用 LOCK_timeout 来设置程序请求锁定的最长等待时间，如果一个锁定请求等待超过了最长等待时间，那么该语句将被自动取消。LOCK_timeout 语句主要用于自定义锁超时。

语法：SET Lock_timeout[timeout_period]

参数 timeout_period 以 ms 为单位，值为-1（默认值）时表示没有超时期限（即无限期等待）。当锁等待超过超时值时，将返回错误。值为 0 时表示根本不等待，并且一遇到锁就返回信息。

【例 15.10】 将锁超时期限设置为 5000ms。（**实例位置：光盘\TM\sl\15\10**）

SQL 语句如下：

```
SET Lock_timeout 5000
```

15.5　分布式事务处理

视频讲解：光盘\TM\lx\15\分布式事务处理.mp4

在前面的学习中已经了解，事务是单个的工作单元，而分布式事务则是跨越两个或多个数据库的。

本节主要介绍分布式事务、如何创建分布式事务与分布式处理协调器。

15.5.1 分布式事务简介

在事务处理中，涉及一个以上数据库的事务被称为分布式事务。分布式事务跨越两个或多个称为资源管理器的服务器。如果分布式事务由 Microsoft 分布式事务处理协调器（MS DTC）这类事务管理器或其他支持 X/Open XA 分布式事务处理规范的事务管理器进行协调，则 SQL Server 可以作为资源管理器运行。

15.5.2 创建分布式事务

保证数据的完整性十分重要，要保证数据的完整性，就要在事务处理中保证事务的原子性，在分布式事务处理中主要使用了分布式事务处理协调器，一台服务器上只能运行一个处理协调器实例，必须启动了分布式事务处理协调器才能执行分布式事务。否则事务就会失败。

下面通过一个示例讲解如何创建一个分布式事务。

【例 15.11】 利用分布式事务对链接的远程数据源 MR 的 db_CSharp 数据库中的 Employee 表和本地 Employee 表进行修改。（**实例位置：光盘\TM\sl\15\11**）

SQL 语句如下：

```
Set Xact_Abort on
Begin DISTRIBUTED TRANSACTION
Update Employee set Name = '星星' where ID = 1
Update [MR].[ db_CSharp].[dbo].[ Employee] set Name   = '婷子' where ID = 1
COMMIT TRANSACTION
```

注意

本示例在执行分布式事务时，需启动服务项 Distributed Transaction Coordinator。

在上段代码中使用了 Xact_Abort 语句，此语句可实现当出现错误时回滚当前 Transact-SQL 命令，在 Xact_Abort 语句执行之后，任何运行时语句错误都将导致当前事务自动回滚。编译错误（如语法错误）不受 Xact_Abort 语句的影响。

说明

分布式事务处理要保证事务的原子性，即在事务执行过程中发生错误时，已更新操作必须可以回滚，否则事务数据库就会处于不一致状态。

15.5.3 分布式处理协调器

分布式事务处理协调器（DTC）系统服务负责协调跨计算机系统和资源管理器分布的事务，如数

据库、消息队列、文件系统和其他事务保护资源管理器。如果事务性组件是通过 COM+配置的，就需要 DTC 系统服务。消息队列（也称作 MSMQ）中的事务性队列和 SQL Server 跨多系统运行也需要 DTC 系统服务。

15.6　小　　结

本章主要对 SQL Server 2012 中的事务进行详细讲解，具体讲解过程中，首先介绍事务的概念，让读者对什么是事务有一个清晰的了解；然后讲解了隐式与显式事务、如何使用事务、事务的工作机制及并发事务的使用等高级内容；最后还讲解了与事务关系密切的锁和分布式事务处理等内容。通过本章的学习，读者应该熟练掌握事务的使用，并能够使用事务解决数据库开发中遇到的问题。

15.7　实践与练习

1．实现使用事务在 employee4 表中添加一条记录，并使用 COMMIT 语句结束事务。（**答案位置：光盘\TM\sl\15\12**）

2．使用事务将 student 表中 sno 为 201109008 的学生的 Sname 修改为“赵雪”。（**答案位置：光盘\TM\sl\15\13**）

3．使用一个嵌套事务对 Student 表插入一条记录。在事务 T_A 中嵌套事务 T_B，在 T_B 中执行插入操作，然后提交 T_B，再回滚 T_A。（**答案位置：光盘\TM\sl\15\14**）

第16章

维护 SQL Server 2012

（视频讲解：30 分钟）

数据库在使用的过程中，所有的对象（如表、视图和存储过程等）和数据都有可能根据需要随时进行更新，如果数据库出现突发的灾难性事件，导致数据丢失和损坏，后果将不堪设想，所以对数据库的维护工作将是数据库使用过程中一个重要的环节。

通过阅读本章，您可以：

- 熟悉数据库的脱机与联机
- 掌握如何分离和附加数据库
- 掌握如何导入和导出数据表
- 掌握如何备份和恢复数据库
- 了解如何收缩数据库和文件
- 熟悉如何将数据库或数据表生成脚本
- 了解如何执行脚本

16.1　脱机与联机数据库

视频讲解：光盘\TM\lx\16\脱机与联机数据库.mp4

如果需要暂时关闭某个数据库的服务，用户可以通过选择脱机的方式来实现，脱机后，在需要的时候可以对暂时关闭的数据库通过联机操作的方式重新启动服务。下面分别介绍如何实现数据库的脱机与联机操作。

16.1.1　脱机数据库

脱机数据库的具体步骤如下：

（1）启动 SQL Server Management Studio，并连接到 SQL Server2012 中的数据库，在“对象资源管理器”中展开“数据库”节点。

（2）鼠标右键单击要脱机的数据库 db_2012，在弹出的快捷菜单中选择“任务”/“脱机”命令，如图 16.1 所示，此时将弹出“使数据库脱机”对话框，如图 16.2 所示。

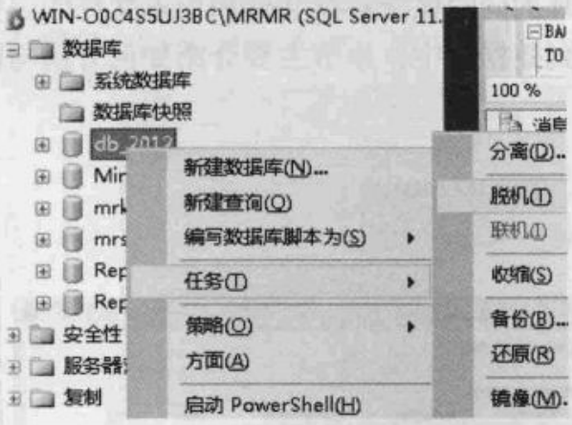


图 16.1　选择脱机数据库

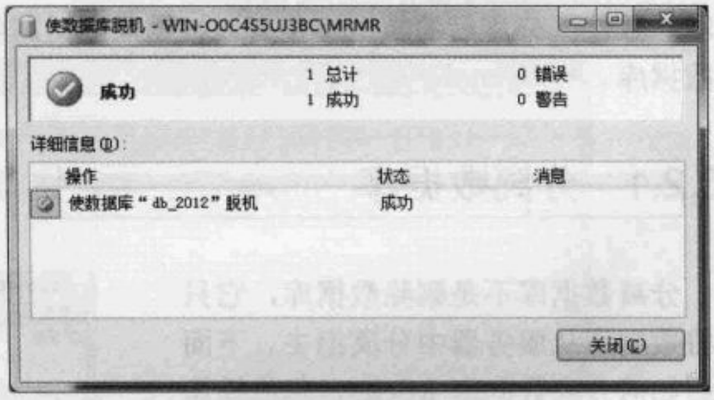


图 16.2　脱机成功

（3）脱机完成后，单击“关闭”按钮即可。

16.1.2　联机数据库

联机数据库的具体步骤如下：

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库，在“对象资源管理器”中展开“数据库”节点。

（2）鼠标右键单击要连接的数据库 db_2012，在弹出的快捷菜单中选择“任务”/“联机”命令，如图 16.3 所示，此时将弹出“使数据库联机”对话框，如图 16.4 所示。

（3）联机完成后，单击“关闭”按钮即可。

图 16.3　选择联机数据库

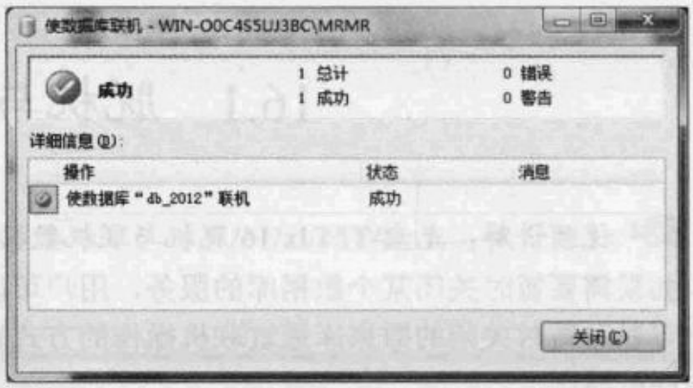


图 16.4　使数据库联机

16.2　分离和附加数据库

视频讲解：光盘\TM\lx\16\分离和附加数据库.mp4

使用分离和附加数据库的方法，可以实现对数据库的复制。对于 SQL Server 数据库来说，分离和附加的数据库，在执行速度和实现数据库的复制功能上更加方便、快捷。除了系统数据库以外，其余的数据库都可以从服务器的管理中分离出来，脱离服务器管理的同时保持数据文件和日志文件的完整性和一致性。分离后的数据库又可以根据需要重新将其附加到数据库中。本节主要介绍如何分离与附加数据库。

16.2.1　分离数据库

分离数据库不是删除数据库，它只是将数据库从服务器中分离出去。下面介绍如何分离数据库 MRKJ，具体操作步骤如下：

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库，在“对象资源管理器”中展开“数据库”节点。

（2）鼠标右键单击要分离的数据库 MRKJ，在弹出的快捷菜单中选择“任务”/“分离”命令，弹出“分离数据库”窗口，如图 16.5 所示。

（3）在“分离数据库”窗口中，“删除连接”表示是否断开与指定数据库的连接；“更新统计信息”表示在分离数据

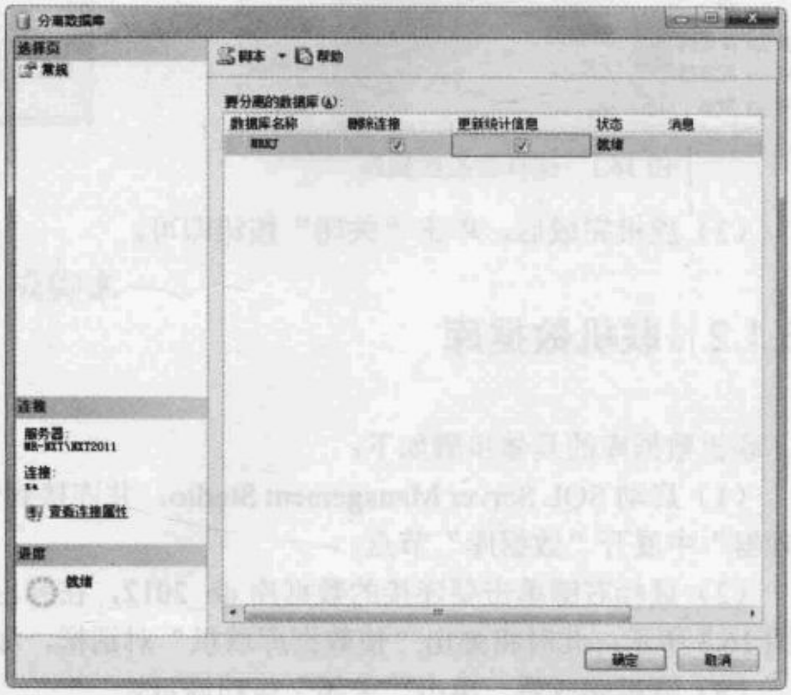


图 16.5　分离数据库

库之前是否更新过时的优化统计信息。这里选择“删除连接”和“更新统计信息”选项。

（4）单击“确定”按钮，完成数据库的分离操作。

16.2.2　附加数据库

与分离对应的就是附加操作，它可以将分离的数据库重新附加到数据库中，也可以附加其他服务器组中分离的数据库，但在附加数据库时，必须指定主数据文件（MDF 文件）的名称和物理位置。

下面附加数据库 MRKJ，具体操作步骤如下：

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库。

（2）鼠标右键单击数据库，在弹出的快捷菜单中选择“附加”命令，弹出“附加数据库”窗口，在“附加数据库”窗口中单击“添加”按钮，弹出“定位数据库文件”窗口，在该窗口中可以选择要附加数据库的位置，如图 16.6 所示。

图 16.6　定位数据库文件

（3）单击“确定”按钮，返回到“附加数据库”窗口，如图 16.7 所示。

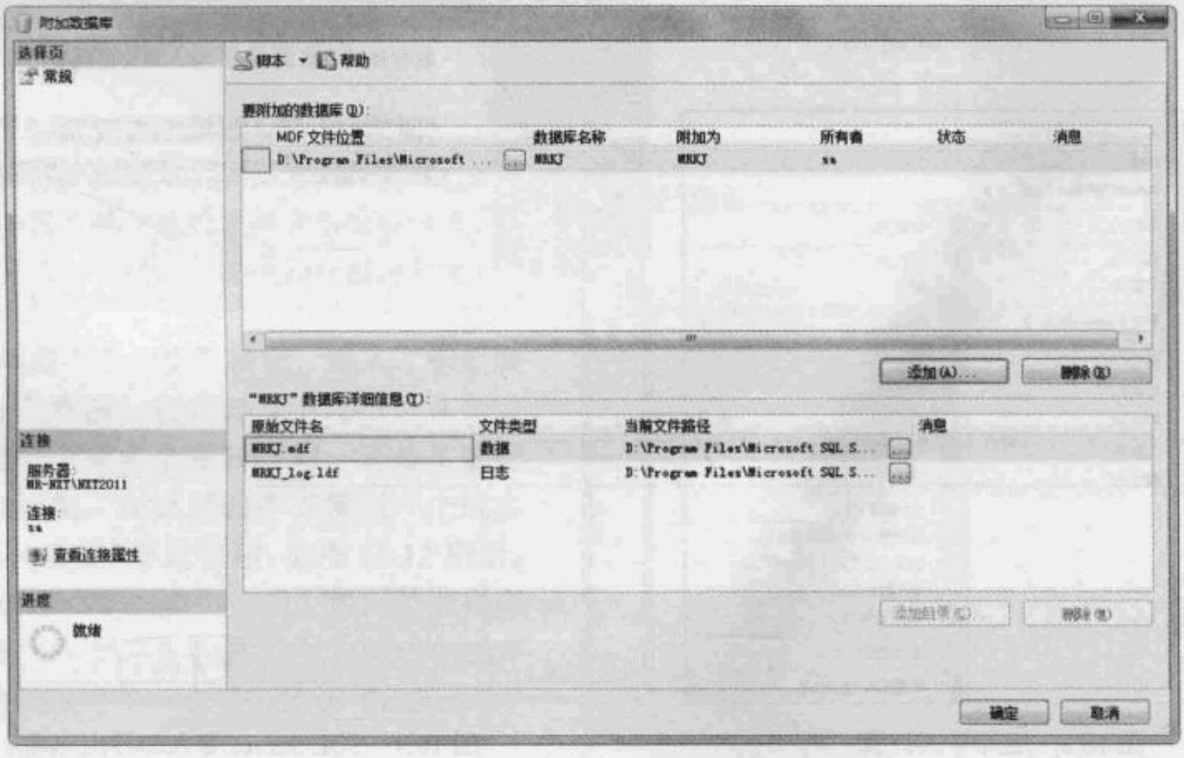


图 16.7　附加数据库

（4）单击“确定”按钮，完成数据库的附加操作。

16.3 导入/导出数据

视频讲解：光盘\TM\lx\16\导入/导出数据.mp4

SQL Server 2012 提供了强大的数据导入/导出功能，它可以在多种常用数据格式（数据库、电子表格和文本文件）之间导入和导出数据，为不同数据源间的数据转换提供了方便。本节主要介绍如何导入/导出数据表。

16.3.1 导入 SQL Server 数据表

导入数据是从 Microsoft SQL Server 的外部数据源中检索数据，然后将数据插入到 SQL Server 表的过程。下面主要介绍通过导入/导出向导将 SQL Server 数据库 db_2012 中的部分数据表导入到 SQL Server 数据库 MRKJ 中。具体操作步骤如下：

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库。在“对象资源管理器”中展开“数据库”节点。

（2）鼠标右键单击数据库 MRKJ，在弹出的快捷菜单中选择“任务”/“导入数据”命令，如图 16.8 所示，此时将弹出“SQL Server 导入和导出向导”窗口，如图 16.9 所示。

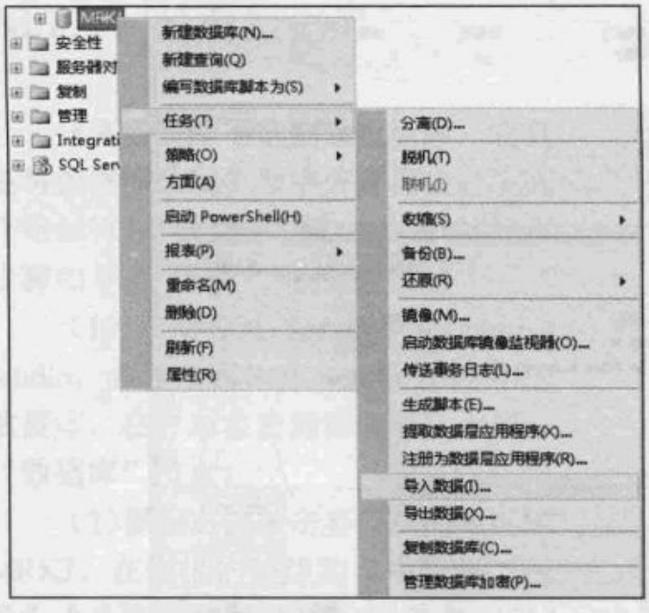


图 16.8 选择导入数据

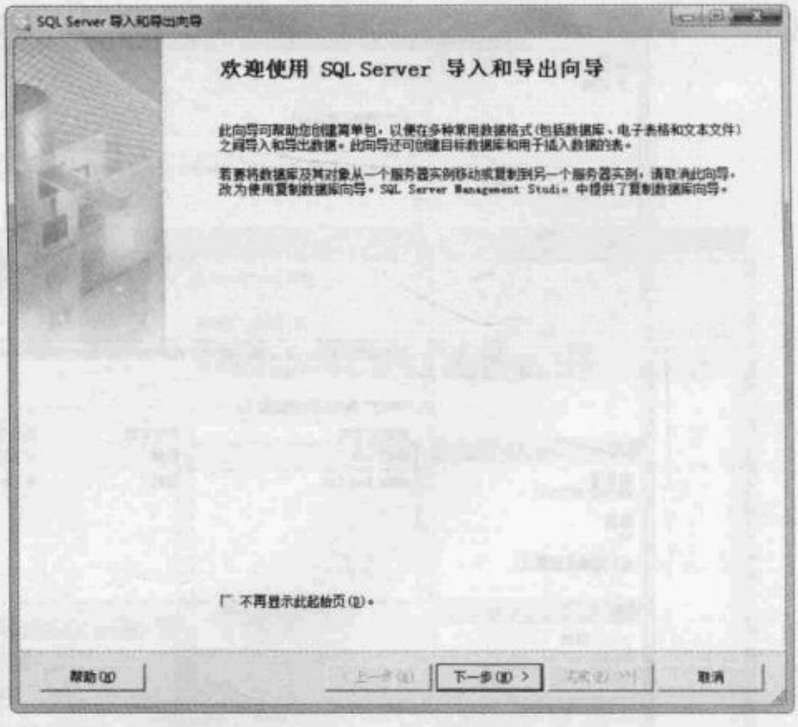


图 16.9 SQL Server 导入和导出向导

注意

将“SQL Server 导入和导出向导”中“不再显示此起始页”复选框选中，再选择“导入数据”命令时，就不再显示该页。

（3）单击“下一步”按钮，进入“选择数据源”界面，在该界面中首先选择数据源，然后选择服务器名称，再选择身份验证方式，最后选择导入数据的源数据库，这里选择 db_2012 数据库，如图 16.10 所示。

（4）单击“下一步”按钮，进入“选择目标”界面，在该界面中选择要将数据库复制到何处，如图 16.11 所示。

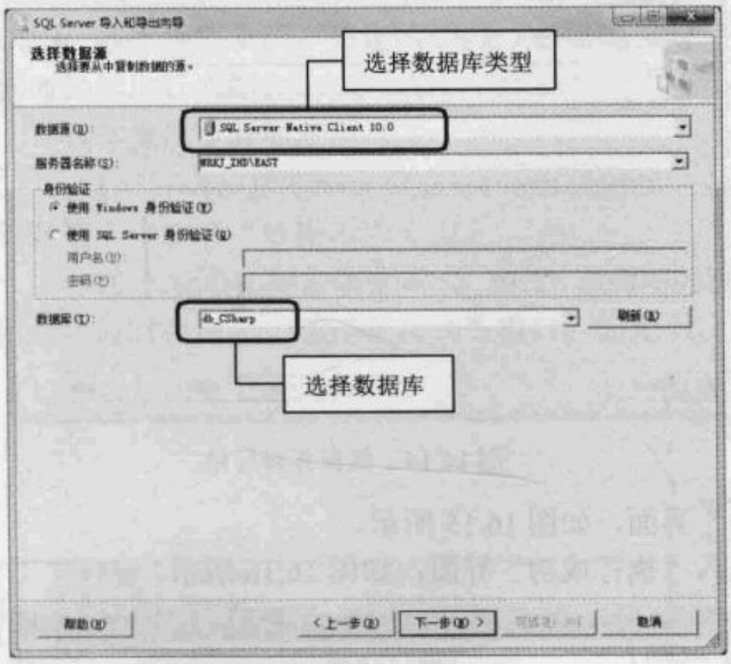


图 16.10　选择数据源

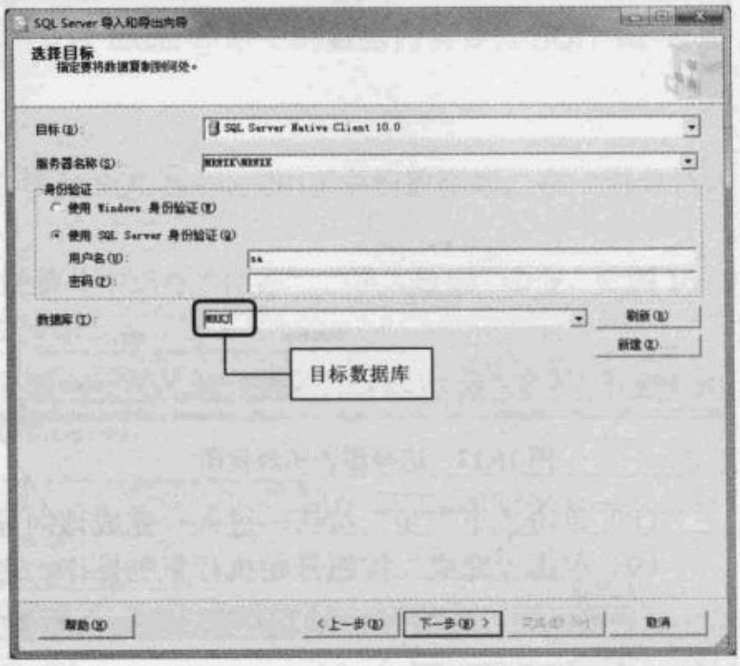


图 16.11　选择目标

说明

在选择要将数据库复制到何处时，首先需要输入服务器名称，然后选择身份验证方式，并输入用户名和密码，最后选择数据库即可。

（5）单击“下一步”按钮，进入“指定表复制或查询”界面，在该界面中选择是从指定数据源复制一个或多个表和视图，还是从数据源复制查询结果，在这里选中“复制一个或多个表或视图的数据”单选按钮，如图 16.12 所示。

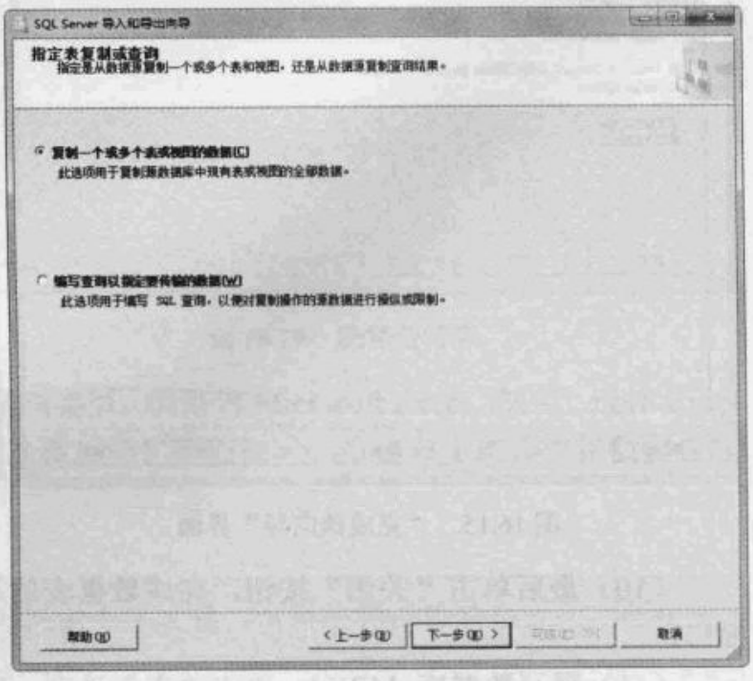


图 16.12　指定表复制或查询

（6）单击“下一步”按钮，进入“选择源表和源视图”界面，在该界面中选择一个或多个要复制的表或视图，这里选择 tb_Student 表，如图 16.13 所示。

（7）单击“下一步”按钮，进入“保存并运行包”界面，该界面窗口用于提示是否选择 SSIS 包，如图 16.14 所示。

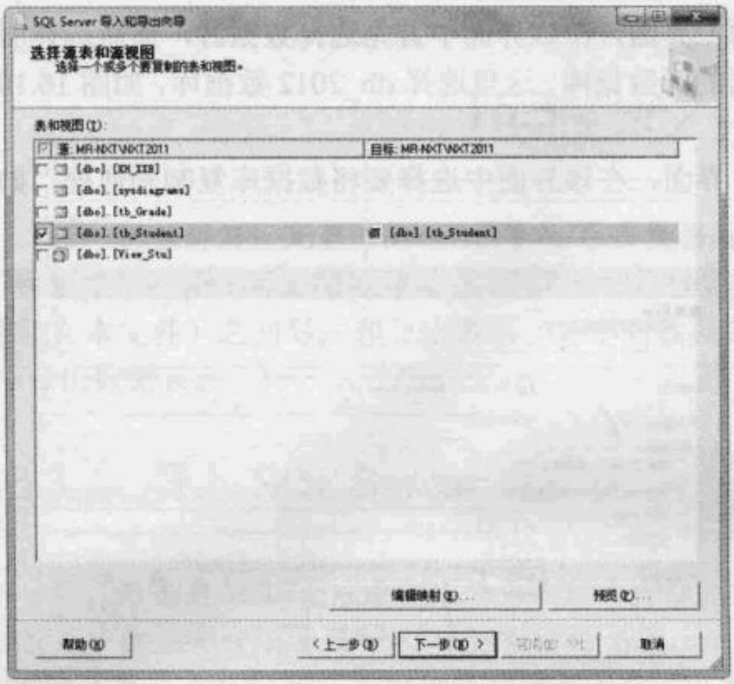


图 16.13　选择源表和源视图

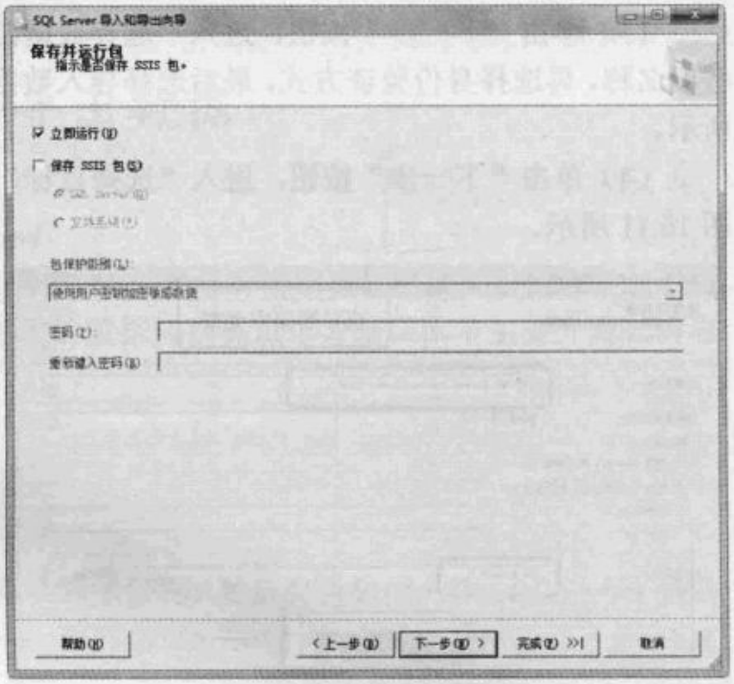


图 16.14　保存并运行包

（8）单击“下一步”按钮，进入“完成该向导”界面，如图 16.15 所示。

（9）单击“完成”按钮开始执行复制操作，进入“执行成功”界面，如图 16.16 所示。

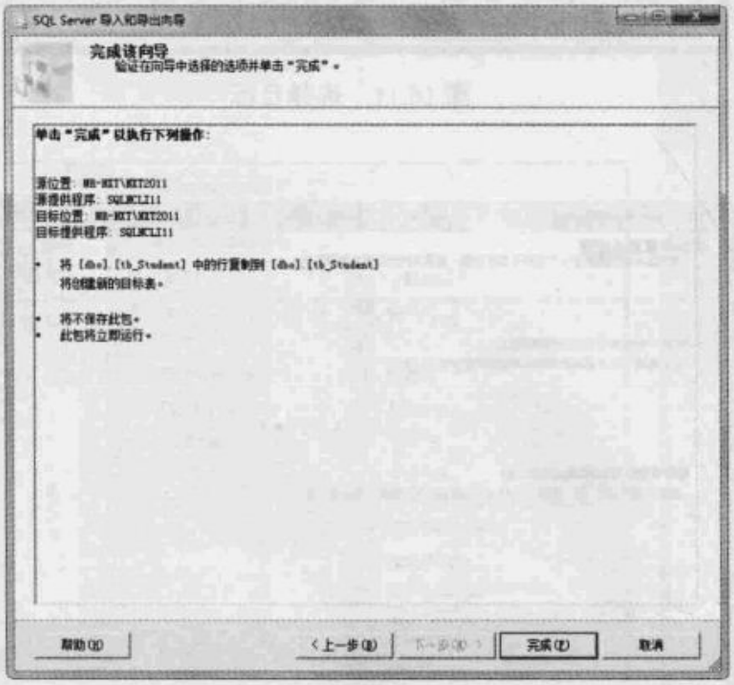


图 16.15　“完成该向导”界面

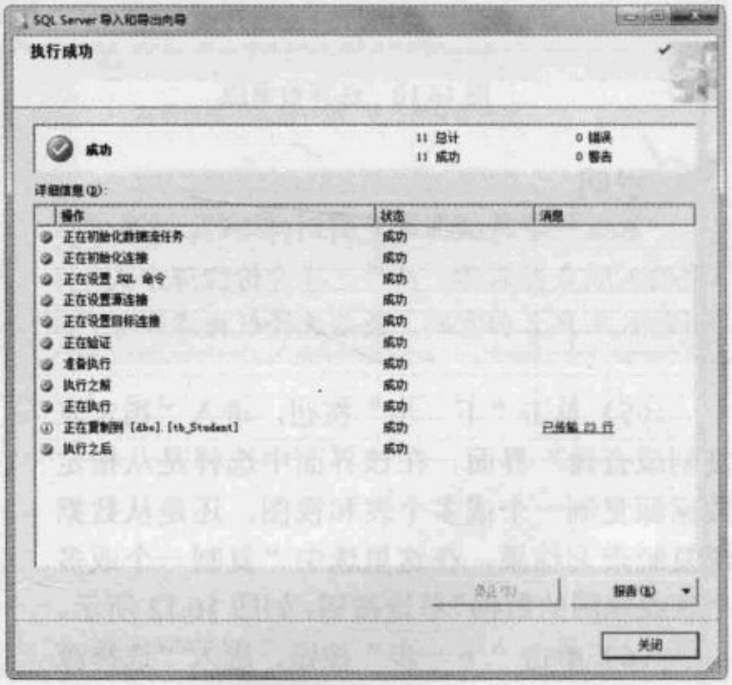


图 16.16　“执行成功”界面

（10）最后单击“关闭”按钮，完成数据表的导入操作。

（11）展开数据库 MRKJ，单击“表”选项，即可从数据库中查看从数据库 db_2012 中导入的数据表，如图 16.17 所示。

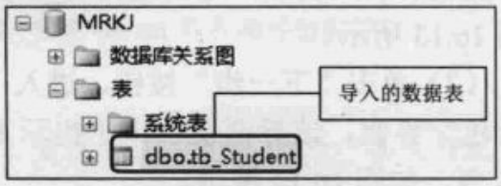


图 16.17　导入的数据表

16.3.2　导入其他数据源的数据

SQL Server 2012 除了支持 Access 和 SQL Server 数据源外，还支持其他形式的数据源，如 Microsoft Excel 电子表格、Microsoft FoxPro 数据库、dBase 或 Paradox 数据库、文本文件、大多数的 OLE DB 和 ODBC 数据源以及用户指定的 OLE DB 数据源等。本节以 Excel 表格中的数据内容导入 SQL Server 数据库为例进行介绍。

具体操作步骤如下：

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库。在“对象资源管理器”中展开“数据库”节点。

（2）鼠标右键单击数据库 db_2012，在弹出的快捷菜单中选择“任务”/“导入数据”命令，如图 16.18 所示，此时将弹出“选择数据源”窗口，如图 16.19 所示。

图 16.18　选择导入数据

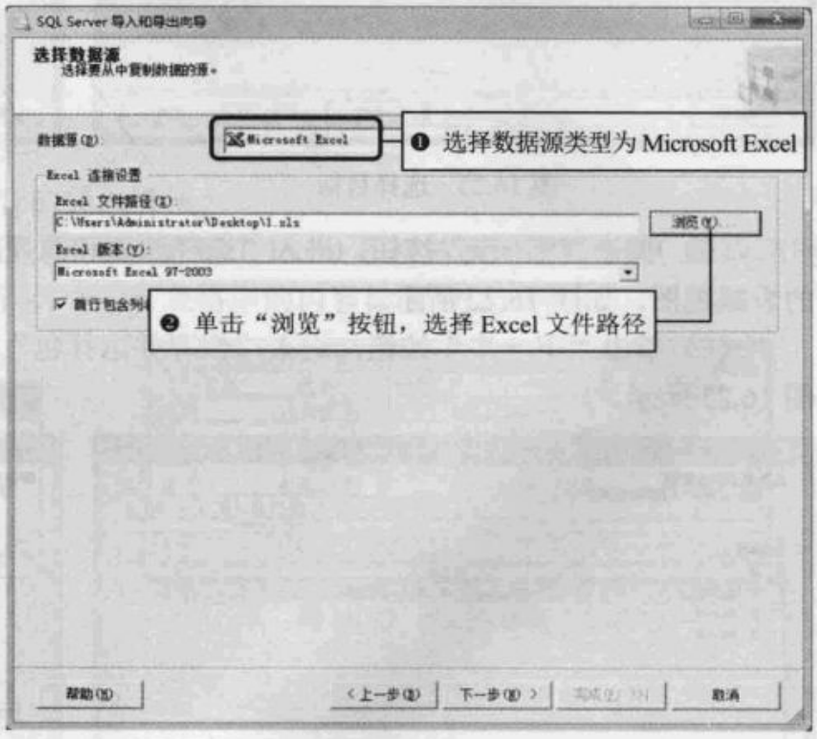


图 16.19　选择数据源

（3）在“选择数据源”窗口中，首先，选择数据源类型，类型为 Microsoft Excel，然后，选择 Excel 文件的路径。最后，单击“下一步”按钮，进入“选择目标”窗口中，在该窗口中选择要将数据库复制到何处，如图 16.20 所示。

说明

在选择要将数据库复制到何处时，首先需要输入服务器名称，然后选择身份验证方式，并输入用户名和密码，最后选择数据库即可。

（4）单击“下一步”按钮，进入“指定表复制或查询”界面，在该界面中选择是从指定数据源复

制一个或多个表和视图，还是从数据源复制查询结果，在这里选中“复制一个或多个表或视图的数据”单选按钮，如图 16.21 所示。

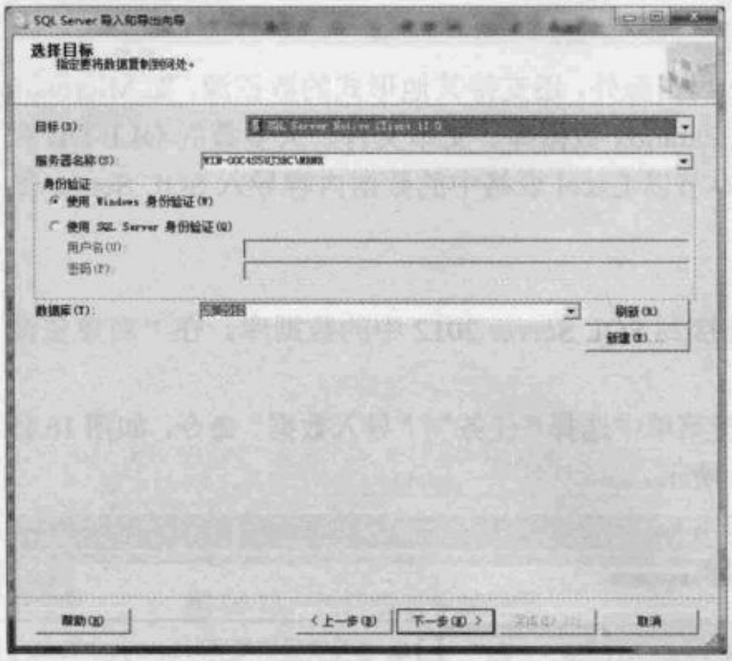


图 16.20　选择目标

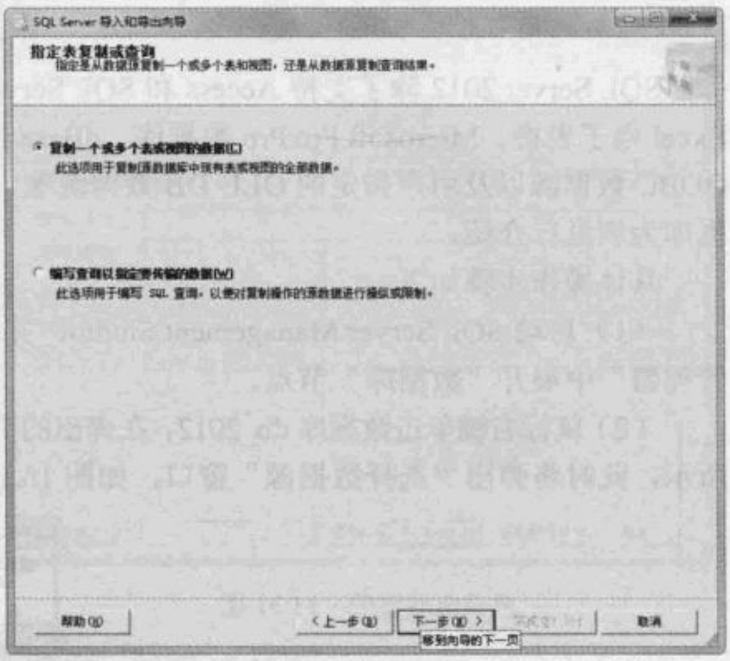


图 16.21　指定表复制或查询

（5）单击“下一步”按钮，进入“选择源表和源视图”界面，在该界面中选择一个或多个要复制的表或视图，如图 16.22 所示。

（6）单击“下一步”按钮，进入“保存并运行包”界面，该界面用于提示是否选择 SSIS 包，如图 16.23 所示。

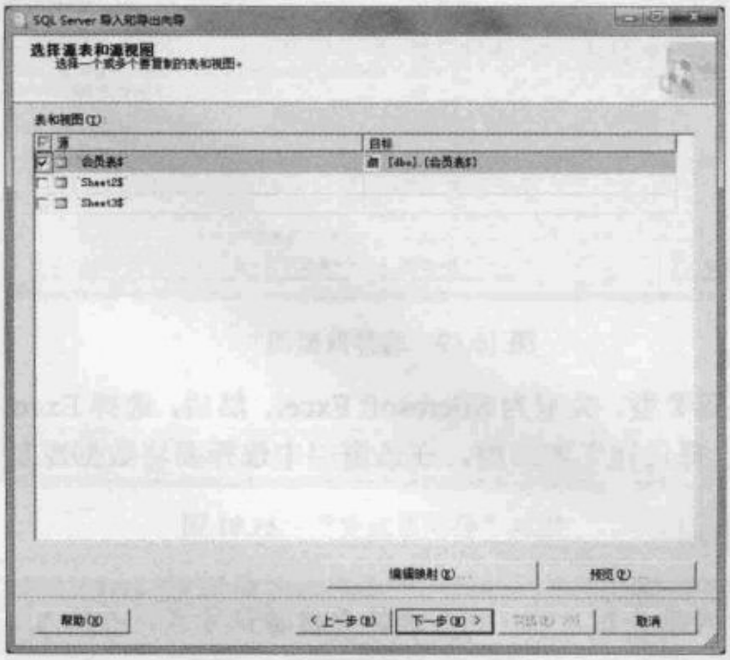


图 16.22　选择源表和源视图

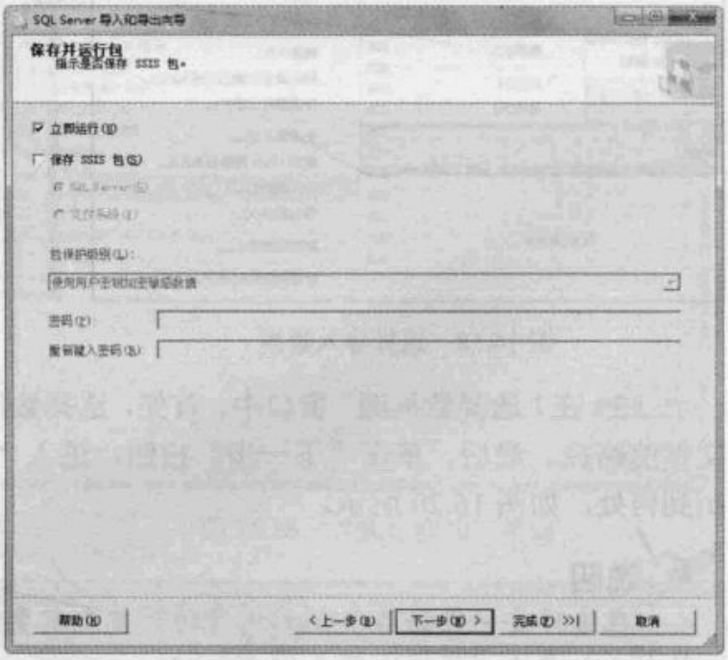


图 16.23　保存并运行包

（7）单击“下一步”按钮，进入“完成该向导”界面，如图 16.24 所示。

（8）单击“完成”按钮开始执行复制操作，进入“执行成功”界面，如图 16.25 所示。

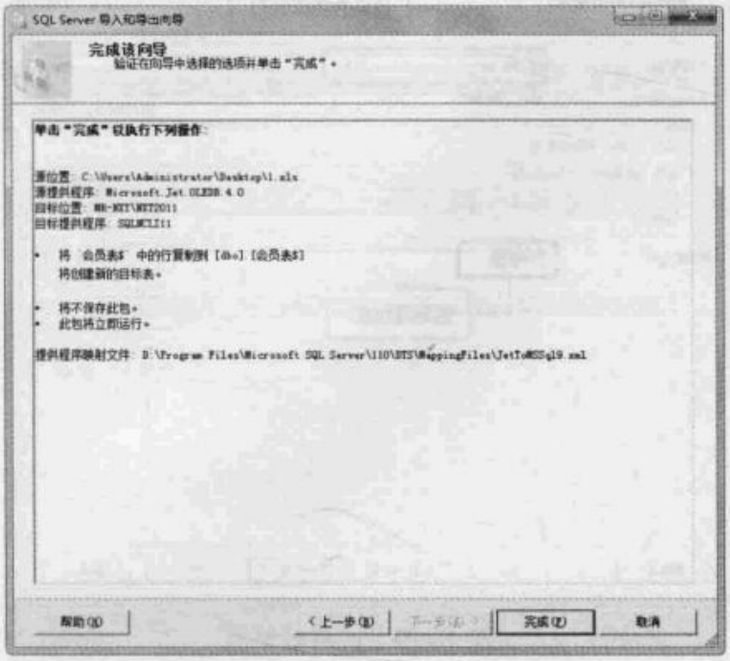


图 16.24　“完成该向导”界面

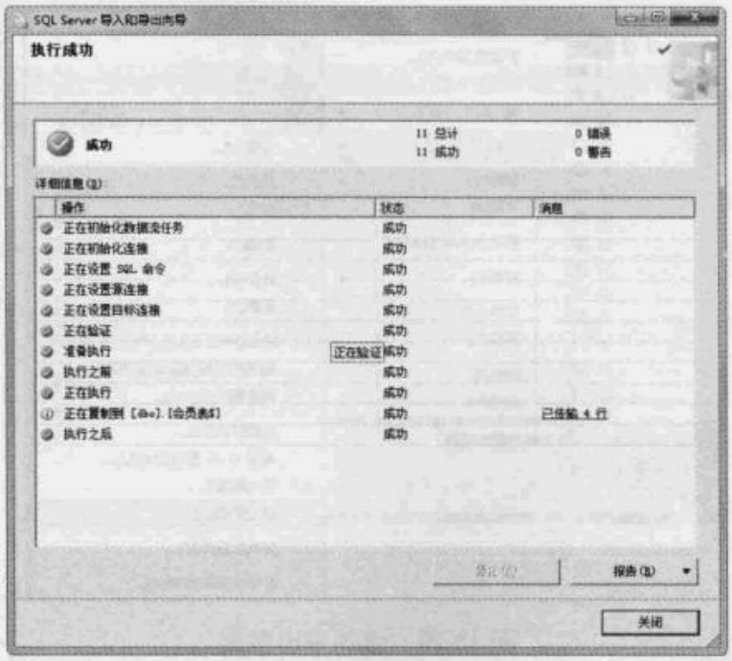
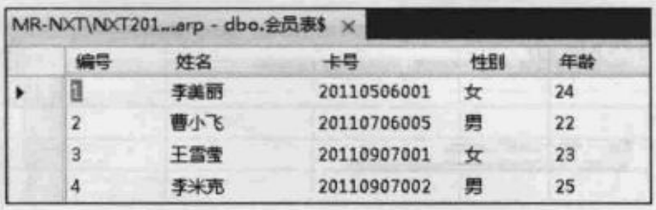


图 16.25　“执行成功”界面

（9）最后单击“关闭”按钮，完成数据表的导入操作。

（10）展开数据库 db_2012，打开“会员表”，可以看到在 Excel 表格转换的数据信息已经成功地导入到 SQL Server 数据库中了，如图 16.26 所示。Excel 表格中的内容如图 16.27 所示。



编号	姓名	卡号	性别	年龄
1	李美丽	20110506001	女	24
2	曹小飞	20110706005	男	22
3	王雪莹	20110907001	女	23
4	李米克	20110907002	男	25

图 16.26　导入的“会员表”中的数据

	A	B	C	D	E
1	编号	姓名	卡号	性别	年龄
2	1	李美丽	20110506001	女	24
3	2	曹小飞	20110706005	男	22
4	3	王雪莹	20110907001	女	23
5	4	李米克	20110907002	男	25

图 16.27　Excel 表格中的内容

16.3.3　导出 SQL Server 数据表

导出数据是将 SQL Server 实例中的数据析取为某些用户指定格式的过程，如将 SQL Server 表的内容复制到 Excel 表格中。

下面主要介绍通过导入/导出向导将 SQL Server 数据库 db_2012 中的部分数据表导出到 Excel 表格中。具体操作步骤如下：

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库。在“对象资源管理器”中展开“数据库”节点。

（2）鼠标右键单击数据库 db_2012，在弹出的快捷菜单中选择“任务”/“导出数据”命令，如图 16.28 所示，此时将弹出“选择数据源”界面，在该界面中选择要从中复制数据的源，如图 16.29 所示。

图 16.28　选择导出数据

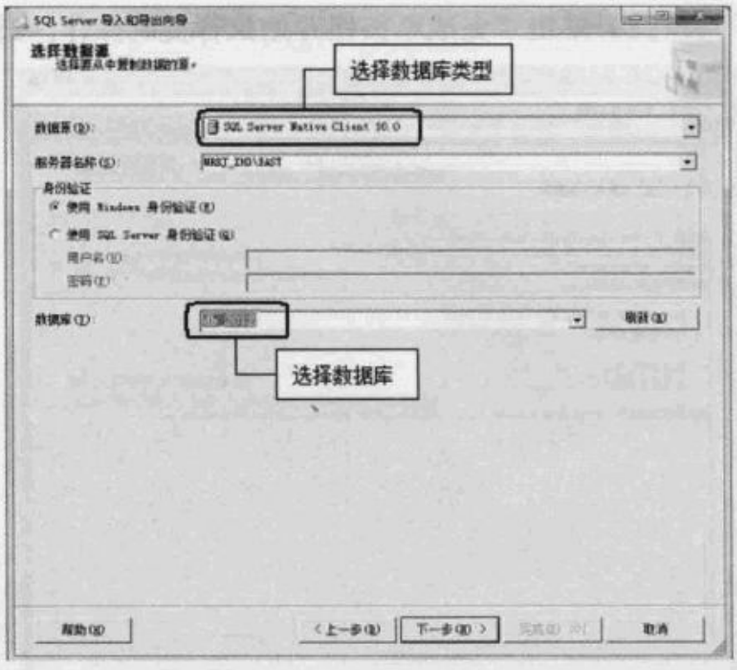


图 16.29　选择数据源

（3）单击“下一步”按钮，进入“选择数据源”界面，在该界面中选择要将数据库复制到何处，在该窗体中分别选择数据源类型和 Excel 文件的位置，如图 16.30 所示。

（4）单击“下一步”按钮，进入“指定表复制或查询”界面，在该界面中选择是从指定数据源复制一个或多个表和视图，还是从数据源复制查询结果，在这里选中“复制一个或多个表或视图的数据”单选按钮，如图 16.31 所示。

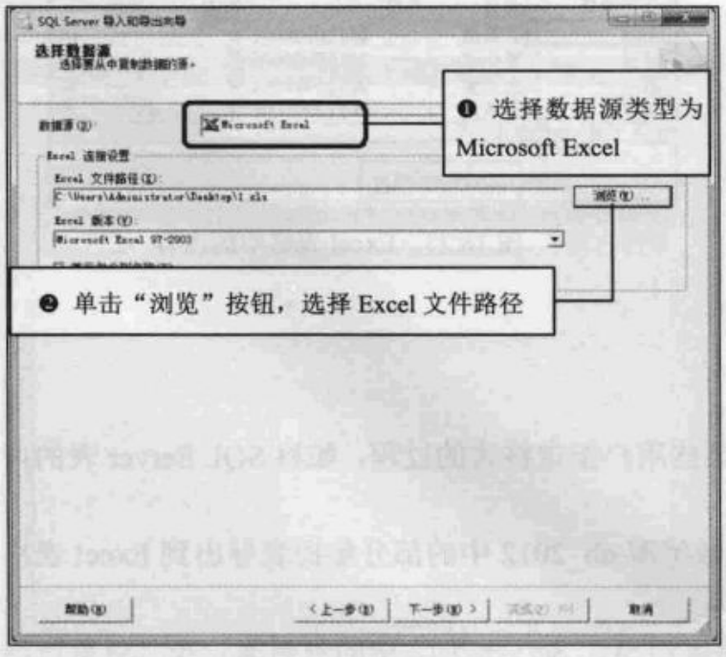


图 16.30　选择目标

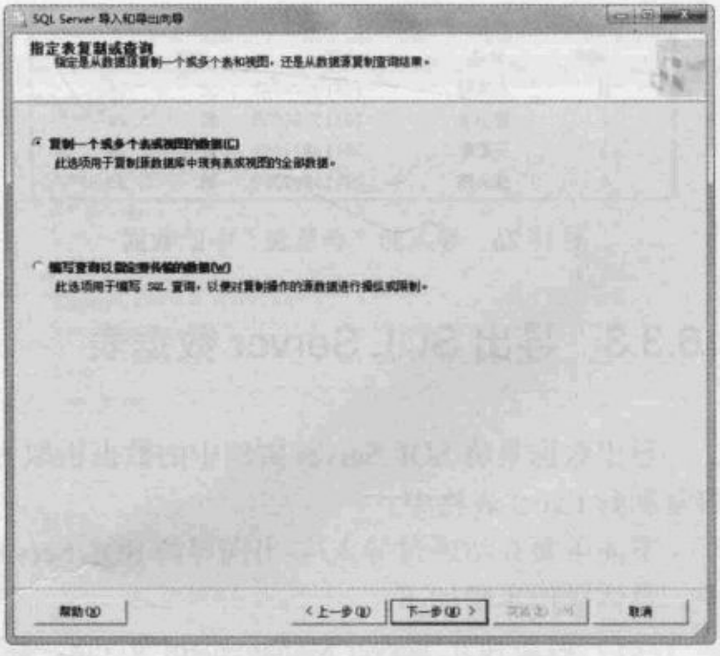


图 16.31　指定表复制或查询

（5）单击“下一步”按钮，进入“选择源表和源视图”界面，在该界面中选择一个或多个要复制的表或视图，这里选择 tb_Courset 表和 tb_Grade 表，如图 16.32 所示。

（6）单击“下一步”按钮，进入“保存并运行包”界面，该界面用于提示是否选择 SSIS 包，如图 16.33 所示。

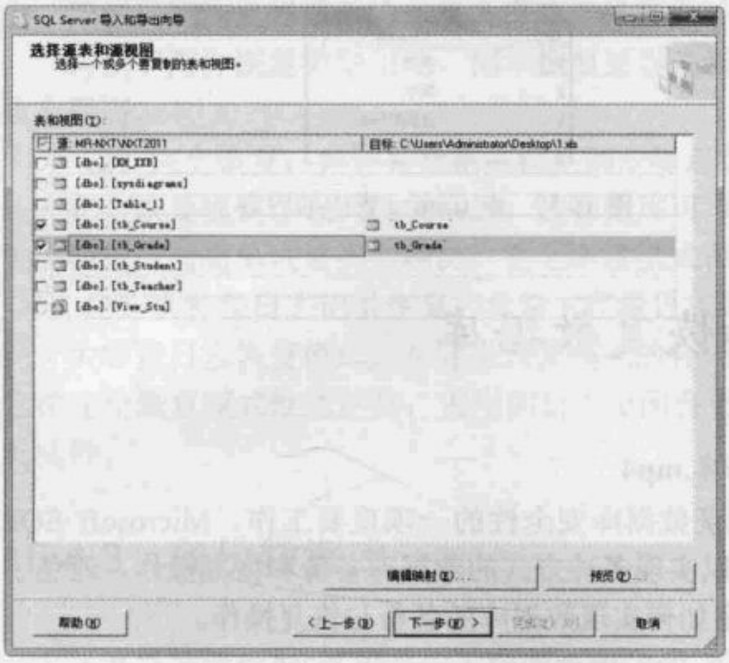


图 16.32　选择源表和源视图

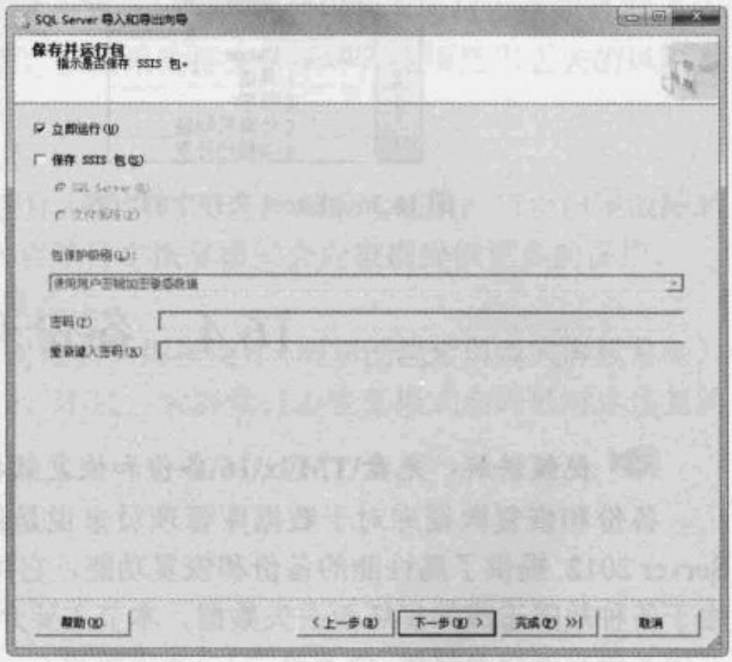


图 16.33　保存并运行包

（7）单击“下一步”按钮，进入“完成该向导”界面，如图 16.34 所示。

（8）单击“完成”按钮开始执行复制操作，进入“执行成功”界面，如图 16.35 所示。

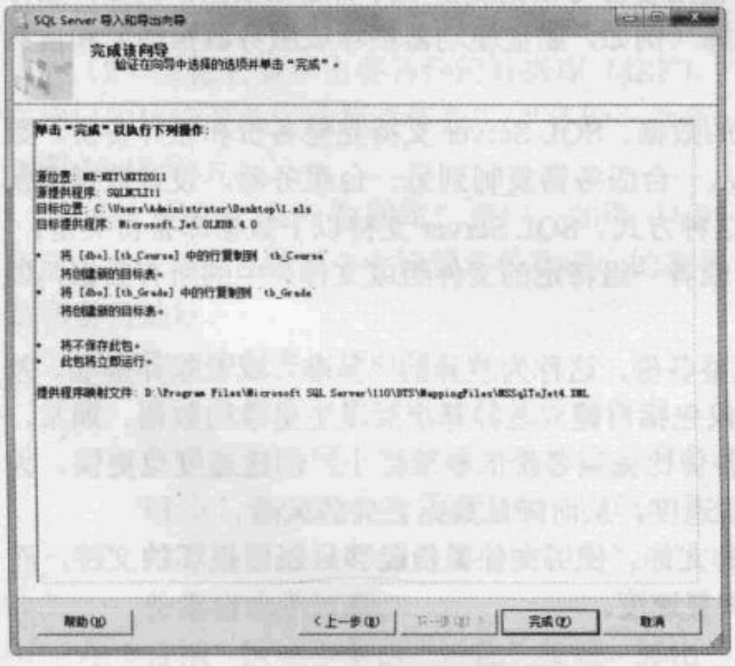


图 16.34　“完成该向导”界面

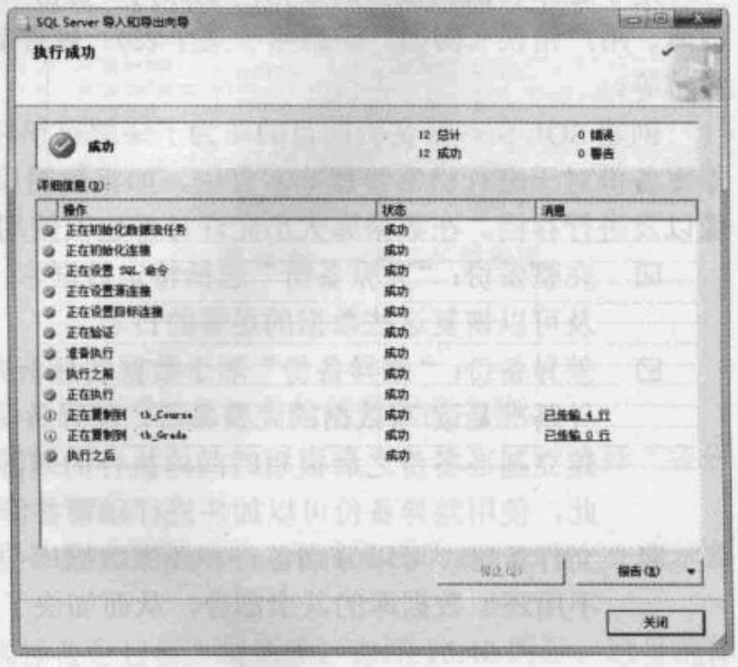


图 16.35　“执行成功”界面

（9）最后单击“关闭”按钮，完成数据表的导入操作。

（10）打开 1.xls，即可查看从数据库 db_2012 中导入的数据表中的内容，如图 16.36 所示，图 16.37 为 tb_Course 表中的内容。

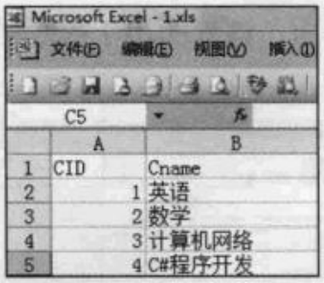


图 16.36　Excel 文件中的内容

CID	Cname
1	英语
2	数学
3	计算机网络
4	C#程序开发

图 16.37　tb_Course 表中的内容

16.4　备份和恢复数据库

视频讲解：光盘\TM\lx\16\备份和恢复数据库.mp4

备份和恢复数据库对于数据库管理员来说是保证数据库安全性的一项重要工作。Microsoft SQL Server 2012 提供了高性能的备份和恢复功能，它可以实现多种方式的数据库备份和恢复操作，避免了由于各种故障造成的损坏而丢失数据。本节主要介绍如何实现数据库的备份与恢复操作。

16.4.1　备份类型

用于还原和恢复数据的数据副本称为“备份”。使用备份可以在发生故障后还原数据。例如，媒体故障、用户错误（例如，误删除了某个表）、硬件故障（例如，磁盘驱动器损坏或服务器报废）和自然灾难等。

创建 SQL Server 备份的目的是为了还原已损坏的数据。SQL Server 支持完整备份和差异备份。数据库备份对于进行日常管理非常有用，如将数据库从一台服务器复制到另一台服务器，设置数据库镜像以及进行存档。在数据库大小允许时都建议使用这种方式。SQL Server 支持以下数据库备份类型。

- ☑ 完整备份：“完整备份”包括特定数据库（或者一组特定的文件组或文件）中的所有数据，以及可以恢复这些数据的足够的日志。
- ☑ 差异备份：“差异备份”基于数据的最新完整备份。这称为差异的“基准”或者差异基准。差异基准是读/写数据的完整备份。差异备份仅包括自建立差异基准后发生更改的数据。通常，建立基准备份之后很短时间内执行的差异备份比完整备份的基准更小，创建速度也更快。因此，使用差异备份可以加快进行频繁备份的速度，从而降低数据丢失的风险。
- ☑ 文件备份：可以分别备份和还原数据库中的文件。使用文件备份能够只还原损坏的文件，而不用还原数据库的其余部分，从而加快了恢复速度。

16.4.2　恢复模式

恢复模式旨在控制事务日志维护。有 3 种恢复模式：简单恢复模式、完整恢复模式和大容量日志恢复模式。通常，数据库使用完整恢复模式或简单恢复模式。

（1）简单恢复：允许将数据库恢复到最新的备份。

简单恢复仅用于测试和开发数据库或包含的大部分数据为只读的数据库。简单恢复所需的管理最少，数据只能恢复到最近的完整备份或差异备份，不备份事务日志，且使用的事务日志空间最小。

与以下两种恢复类型相比，简单恢复更容易管理，但如果数据文件损坏，出现数据丢失的风险系数会很高。

（2）完全恢复：允许将数据库恢复到故障点状态。

完全恢复提供了最大的灵活性，使数据库可以恢复到早期时间点，在最大范围内防止出现故障时丢失数据。与简单恢复类型相比，完全恢复模式和大容量日志恢复模式会向数据提供更多的保护。

（3）大容量日志记录恢复：允许大容量日志记录操作。

大容量日志恢复模式是对完全恢复模式的补充。对某些大规模操作（例如创建索引或大容量复制），它比完全恢复模式性能更高，占用的日志空间会更少。不过，大容量日志恢复模式会降低时点恢复的灵活性。

16.4.3　备份数据库

“备份数据库”任务可执行不同类型的 SQL Server 数据库备份（完整备份、差异备份和文件备份）。

下面以备份数据库 MRKJ 为例介绍如何备份数据库。具体操作步骤如下：

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库。在“对象资源管理器”中展开“数据库”节点。

（2）鼠标右键单击要备份的数据库 MRKJ，在弹出的快捷菜单中选择“任务”/“备份”命令，如图 16.38 所示。

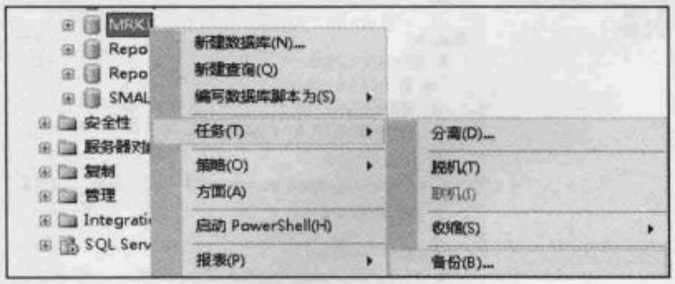


图 16.38　选择备份数据库

（3）进入“备份数据库”窗口，如图 16.39 所示。在“常规”选项卡中设置备份数据库的数据源和备份地址。

在该窗口中设置以下几项：

- ☑ 在“数据库”列表框中验证数据库名，如果需要也可以更改备份的数据库名称。
- ☑ 在“备份类型”列表框中选择数据库备份的类型，这里选择“完整”备份。同时选择“备份组件”选项组中的“数据库”选项，备份整个数据库。
- ☑ 根据需要通过“备份集过期时间”选项设置备份的过期天数。取值范围为 0～9999，0 表示备份集将永不过期。

在“目标”区域中单击“添加”按钮，弹出“选择备份目标”对话框，如图 16.40 所示，这里选择“文件名”选项，单击其后的“浏览”按钮 ...，选择文件名及其路径。

（4）单击“确定”按钮，返回到“备份数据库”窗口。打开“选项”选项卡，如图 16.41 所示。这里在“覆盖介质”区域中选择“备份到现有介质集”/“追加到现有备份集”选项，把备份文件追加到指定介质上，同时保留以前的所有备份。

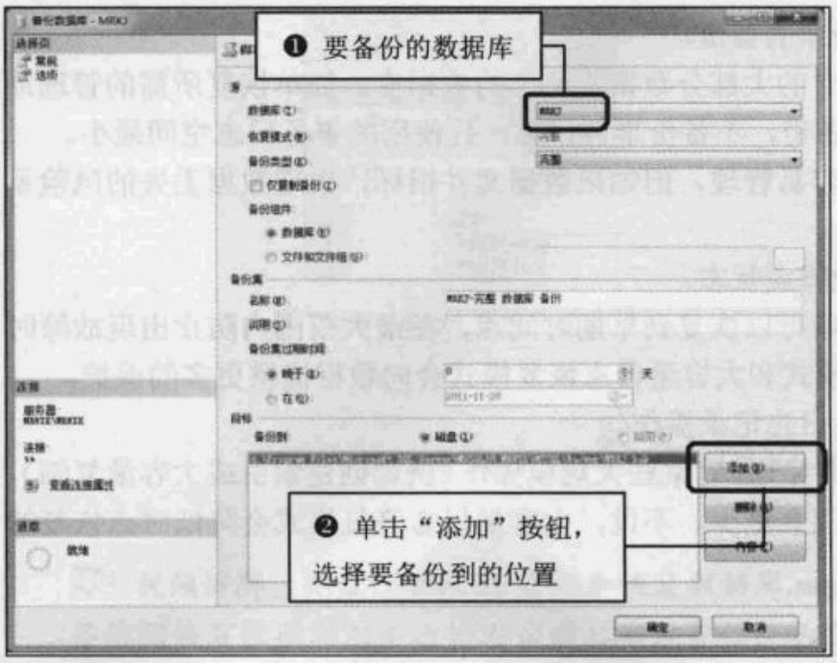


图 16.39　备份数据库

图 16.40　选择备份目标

（5）单击“确定”按钮，系统提示备份成功的提示信息，如图 16.42 所示。单击“确定”按钮后即可完成数据库的完整备份。

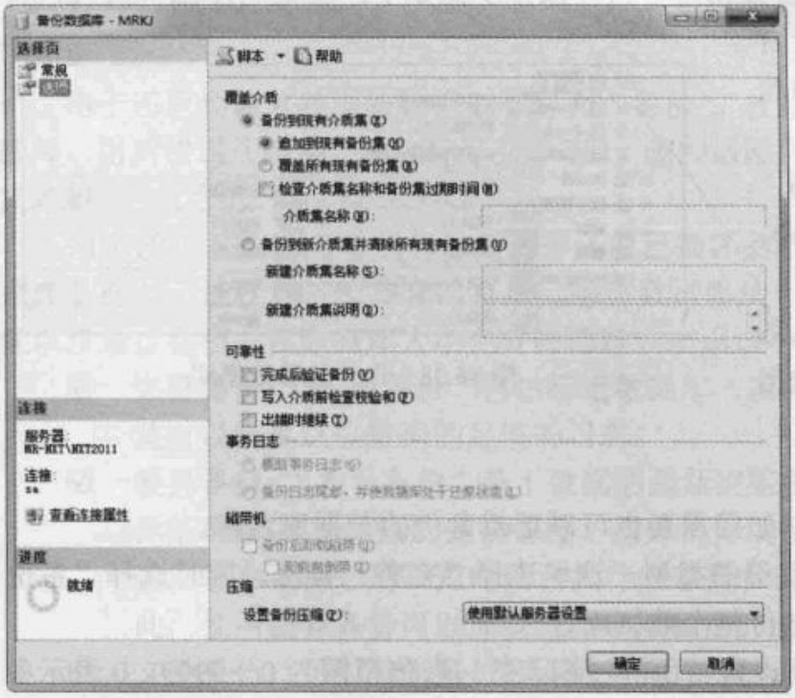


图 16.41　备份数据库

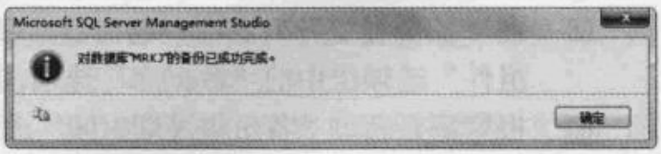


图 16.42　提示信息

16.4.4　恢复数据库

执行数据库备份的目的是便于进行数据恢复。如果发生机器错误、用户操作错误等，用户就可以对备份过的数据库进行恢复。

下面以恢复数据库 MRKJ 为例介绍如何恢复数据库。具体操作步骤如下：

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库。在“对象资源管理器”中展开“数据库”节点。

（2）鼠标右键单击要恢复的数据库 MRKJ，在弹出的快捷菜单中选择“任务”/“还原”/“数据库”命令，如图 16.43 所示。

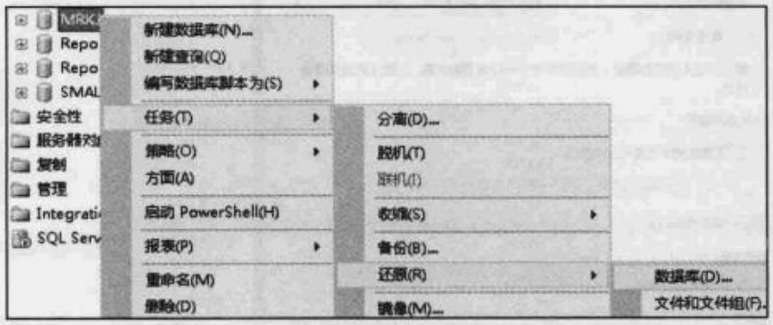


图 16.43　选择还原数据库

（3）进入“还原数据库”窗口，在该窗口的“常规”选项卡中设置还原的目标和源数据库，在该窗口中保留默认设置即可，如图 16.44 所示。

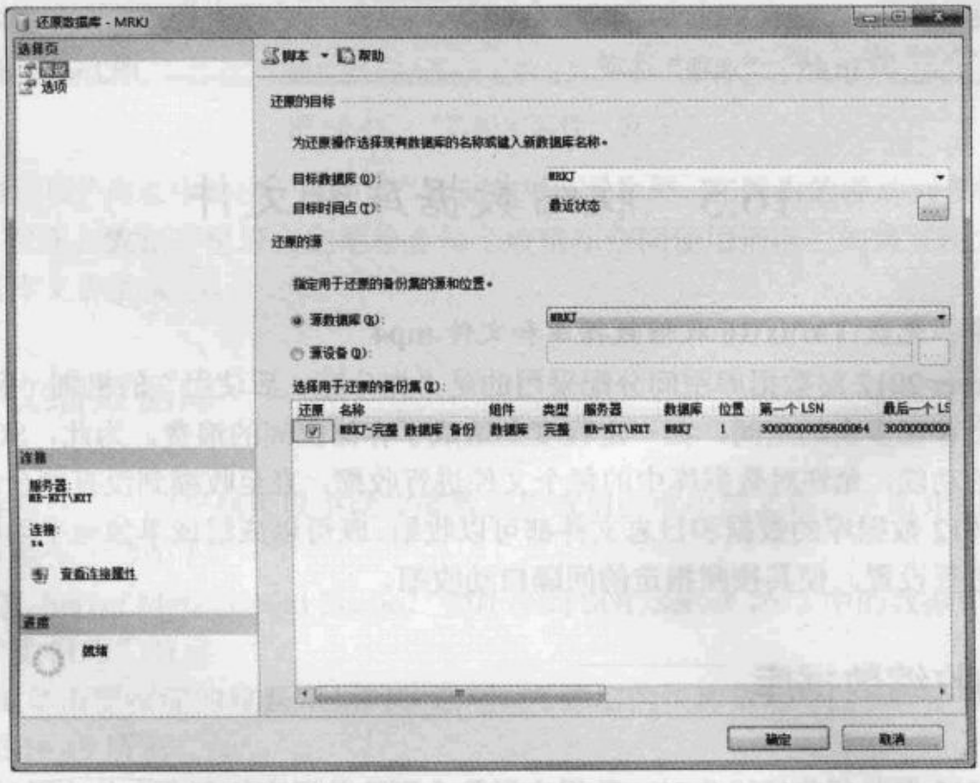


图 16.44　还原数据库

（4）打开“选项”选项卡，设置还原操作时采用的形式以及恢复完成后的状态，如图 16.45 所示。这里在“还原选项”区域中选中“覆盖现有数据库”复选框，以便在恢复时覆盖现有数据库及其相关文件。

（5）单击“确定”按钮，系统提示还原成功的提示信息，如图 16.46 所示。单击“确定”按钮后即可完成数据库的还原操作。

图 16.45 “选项”选项卡

图 16.46 提示信息

16.5 收缩数据库和文件

视频讲解：光盘\TM\lx\16\收缩数据库和文件.mp4

由于 SQL Server 2012 对数据库空间分配采用的是“先分配、后使用”的机制，所以数据库在使用的过程中就可能会存在多余的空间，在一定程度上造成了存储空间的浪费。为此，SQL Server 2012 提供了收缩数据库的功能，允许对数据库中的每个文件进行收缩，直至收缩到没有剩余的可用空间为止。

SQL Server 2012 数据库的数据和日志文件都可以收缩。既可以成组或单独地手动收缩数据库文件；也可以对数据库进行设置，使其按照指定的间隔自动收缩。

16.5.1 自动收缩数据库

SQL Server 2012 在执行收缩操作时，数据库引擎会删除数据库的每个文件中已经分配但还没有使用的页，收缩后的数据库空间将自动减少。下面介绍如何自动收缩数据库。具体操作步骤如下：

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库。在“对象资源管理器”中展开“数据库”节点。

（2）鼠标右键单击指定的数据库 MRKJ，在弹出的快捷菜单中选择“属性”命令，进入“数据库属性”窗口，打开“选项”选项卡，如图 16.47 所示。

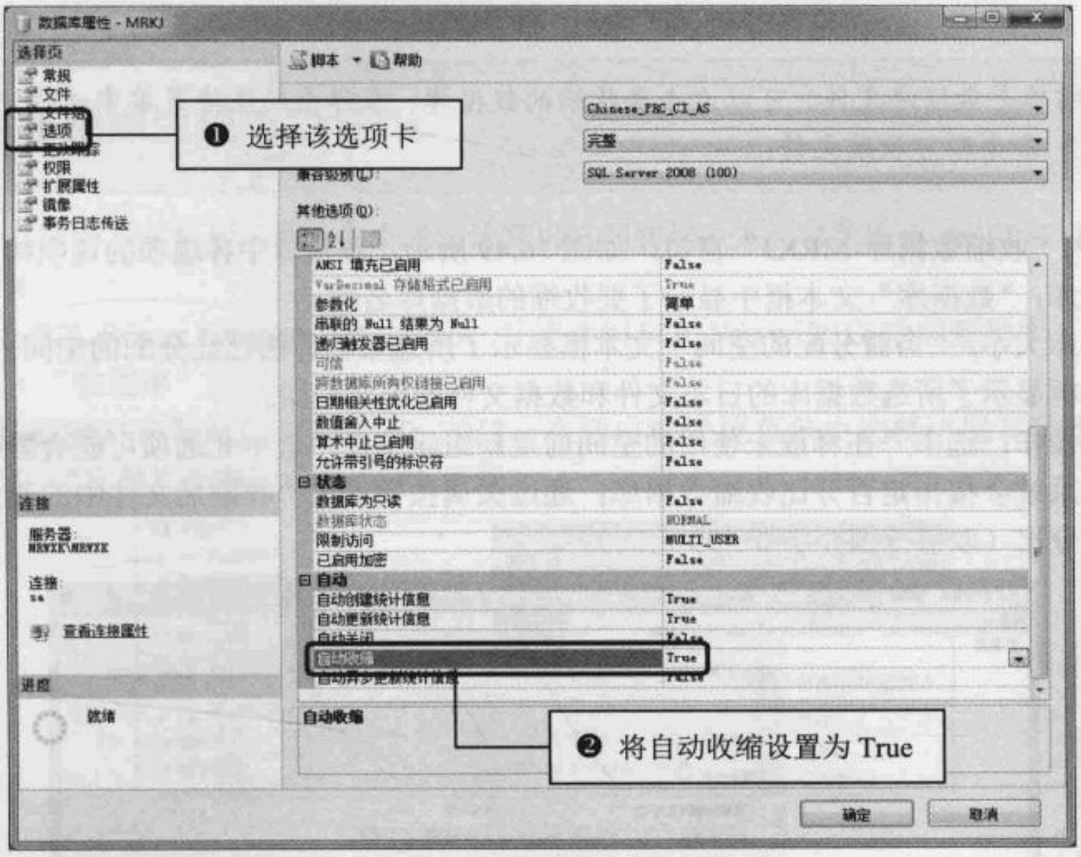


图 16.47　“数据库属性”窗口

（3）在“其他选项”列表中单击“自动”/“自动收缩”文本框，在弹出的浮动列表框中选择 TRUE，然后单击“确定”按钮。数据库引擎会定期检查每个数据库空间使用情况，如果发现大量闲置的空间，就会自动收缩数据库文件的大小。

16.5.2　手动收缩数据库

除了自动收缩数据库，用户也可以手动收缩数据库或数据库中的文件。下面介绍如何手动收缩数据库 MRKJ。具体操作步骤如下：

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库。在“对象资源管理器”中展开“数据库”节点。

（2）鼠标右键单击要收缩的数据库 MRKJ 选项，在弹出的快捷菜单中选择“任务”/“收缩”/“数据库”命令，如图 16.48 所示。

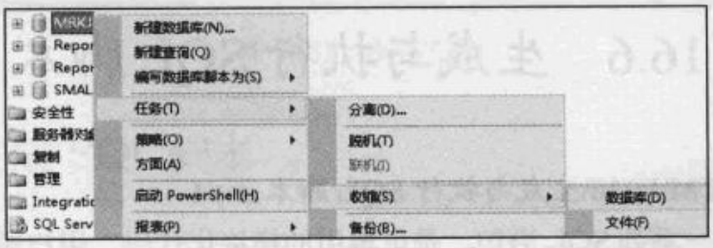


图 16.48　选择收缩数据库

注意

若要收缩单个数据库文件，可以右击要收缩的数据库，在弹出的快捷菜单中选择“任务”/“收缩”/“文件”命令即可收缩文件。

（3）进入“收缩数据库-MRKJ”窗口，如图 16.49 所示。该窗口中各选项的说明如下。

☑ 数据库：“数据库”文本框中显示了要收缩的数据库名称。

☑ 数据库大小：“当前分配的空间”文本框显示了所选数据库的已经分配的空间；“可用空间”文本框显示了所选数据库的日志文件和数据文件的可用空间。

☑ 收缩操作：选中“在释放未使用的空间前重新组织文件，选中此选项可能会影响性能”复选框，系统会按指定百分比收缩数据库；通过微调按钮设置“收缩后文件中的最大可用空间”的百分比（取值范围介于 0～99 之间）。

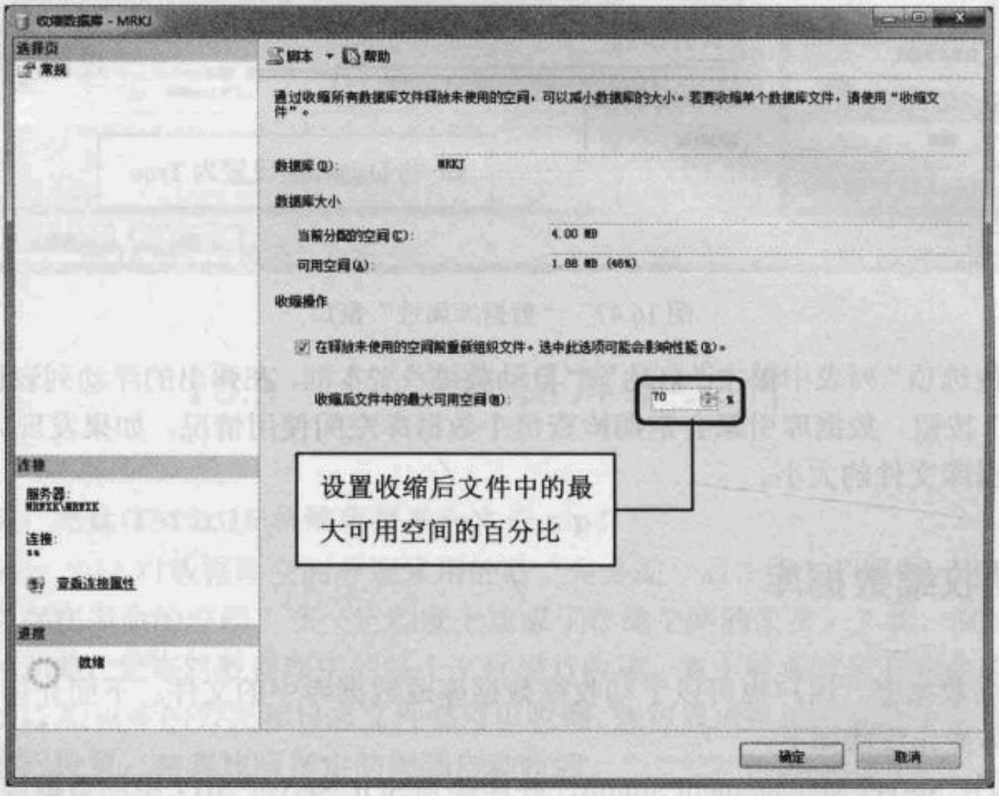


图 16.49　手动收缩数据库

（4）设置完成后单击“确定”按钮进行数据库收缩操作。

16.6　生成与执行 SQL 脚本

视频讲解：光盘\TM\lx\16\生成与执行 SQL 脚本.mp4

脚本是存储在文件中的一系列 SQL 语句，是可再用的模块化代码。用户通过 SQL Server Management Studio 可以对指定文件中的脚本进行修改、分析和执行。

本节主要介绍如何将数据库、数据表生成脚本，以及如何执行脚本。

16.6.1　将数据库生成 SQL 脚本

数据库在生成脚本文件后，可以在不同的计算机之间传送。下面将数据库 db_2012 生成脚本文件。具体操作步骤如下：

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库。在“对象资源管理器”中展开“数据库”节点。

（2）鼠标右键单击指定的数据库 db_2012，在弹出的快捷菜单中选择“编写数据库脚本为”/“CREATE 到”/“文件”命令，如图 16.50 所示。

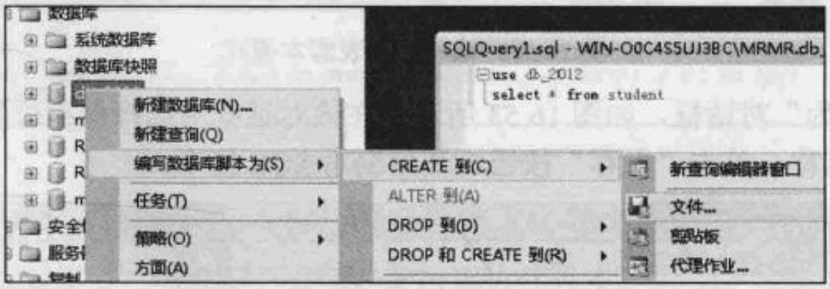


图 16.50　编写数据库脚本模式

（3）进入“另存为”对话框，如图 16.51 所示。在该对话框中选择保存位置，在“文件名”文本框中写入相应的脚本名称。单击“保存”按钮，开始编写 SQL 脚本。

图 16.51　保存文件

16.6.2　将数据表生成 SQL 脚本

除了将数据库生成脚本文件以外，用户还可以根据需要将指定的数据表生成脚本文件。下面将数

据库 db_2012 中的数据表 tb_Student 生成脚本文件。具体操作步骤如下：

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库。在“对象资源管理器”中展开“数据库”节点。

（2）展开指定的数据库 db_2012 /“表”选项。

（3）鼠标右键单击数据表 tb_Student，在弹出的快捷菜单中选择“编写表脚本为”/“CREATE 到”/“文件”命令，如图 16.52 所示。

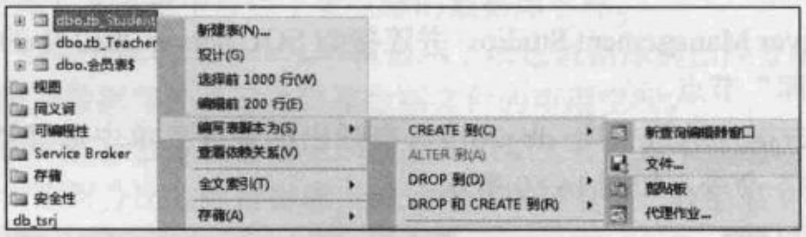


图 16.52　编写数据表脚本模式

（4）进入“另存为”对话框，如图 16.53 所示。在该对话框中选择保存位置，在“文件名”文本框中写入相应的脚本名称，单击“保存”按钮，开始编写 SQL 脚本。

图 16.53　保存文件

16.6.3　执行 SQL 脚本

脚本文件生成以后，用户可以通过 SQL Server Management Studio 对指定的脚本文件进行修改，然后执行该脚本文件。执行 SQL 脚本文件的具体操作步骤如下：

（1）启动 SQL Server Management Studio，并连接到 SQL Server 2012 中的数据库。在“对象资源管理器”中展开“数据库”节点。

（2）选择“文件”/“打开”/“文件”命令，弹出“打开文件”对话框，从中选择保存过的脚本

文件，单击“打开”按钮，脚本文件就被加载到 SQL Server Management Studio 中了，如图 16.54 所示。

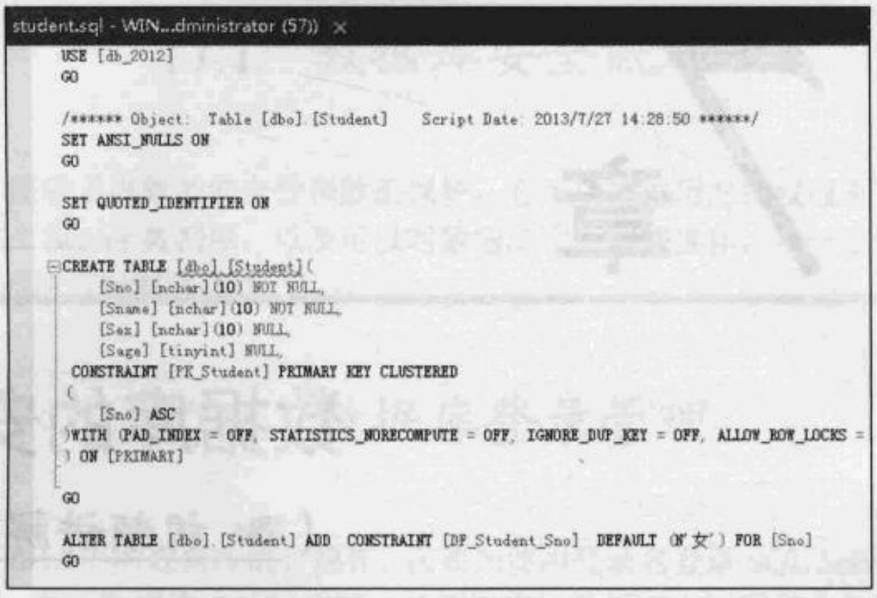


图 16.54　脚本文件

（3）在打开的脚本文件中可以对代码进行修改。修改完成后，可以按 Ctrl+F5 键或单击 ✓ 按钮对脚本语言进行分析，然后使用 F5 键或单击 ! 执行(X) 按钮执行脚本。

16.7　小　　结

本章介绍 SQL Server 2012 中对数据库及数据表的维护管理。读者应熟练掌握脱机与联机数据库、分离和附加数据库、导入和导出数据表、备份和恢复数据库等操作，能够执行将数据库或数据表生成脚本的操作。

16.8　实践与练习

1．通过对 sysdatabases 表执行 SELECT 语句，获取当前 SQL Server 实例中所有数据库信息，使用 syslogins 表验证登录账户，通过查询语句查询登录账户名不为 sa 的数据库。（**答案位置：光盘\TM\sl\16\1**）

2．完整备份数据库 M1，将其备份到 G 盘，然后使用 RESTORE DATABASE 从 G 盘上的备份中还原数据库 M1。（**答案位置：光盘\TM\sl\16\2**）

第17章

数据库的安全机制

（视频讲解：13分钟）

安全性对于任何一个数据库管理系统来说都非常重要，本章将对 SQL Server 2012 数据库的安全性进行详细讲解，包括数据库的登录管理、用户及权限管理等。

通过阅读本章，您可以：

- 熟悉 SQL Server 的两种登录验证模式
- 掌握如何创建以 SQL Server 方式登录的登录名
- 掌握如何更改登录名密码
- 熟悉使用 SQL 语句管理登录名
- 掌握如何为用户设置访问权限

17.1　数据库安全概述

SQL Server 2012 提供了内置的安全性和数据保护，它可以根据用户的权限不同，来决定用户是否可以登录到当前的 SQL Server 数据库，以及可以对数据库实现哪些操作，在一定程度上避免了数据因使用不当或非法访问而造成泄漏和破坏。

17.2　数据库登录管理

要对 SQL Server 2012 中的数据库进行操作，需要先使用登录名登录 SQL Server 2012，然后再对数据库进行操作，然而，在对数据库进行操作时，其所操作的数据库中还要存在与登录名相应的数据库用户，本节将介绍登录名的创建与删除，更改登录用户的验证方式等。

17.2.1　选择验证模式

验证模式指数据库服务器如何处理用户名与密码，SQL Server 2012 的验证方式包括 Windows 验证模式与混合验证模式。用户可根据需要选择相应的验证模式。

☑　Windows 验证模式

Windows 验证模式是 SQL Server 2012 使用 Windows 操作系统中的信息验证账户名和密码。这是默认的身份验证模式，比混合验证模式安全。Windows 验证使用 Kerberos 安全协议，通过强密码的复杂性验证提供密码策略强制，提供账户锁定与密码过期功能。

☑　混合验证模式

允许用户使用 Windows 身份验证或 SQL Server 身份验证进行连接。通过 Windows 用户账户连接的用户可以使用 Windows 验证的受信任连接。

17.2.2　管理登录账号

在 SQL Server 2012 中有两个登录账户：一类是登录服务器的登录名；另外一类是使用数据库的用户账号。登录名是指能登录到 SQL Server 的账号，它属于服务器的层面，本身并不能让用户访问服务器中的数据库，而登录者要使用服务器中的数据库时，必须要有用户账号才能存取数据库。本节介绍如何创建、修改和删除服务器登录名。

管理员可以通过 SQL Server Management Studio 工具对 SQL Server 2012 中的登录名进行创建、修改、删除等管理。

1. 创建登录名

在 SQL Server 2012 中可以创建的登录账户有两种，一种是 SQL Server 标准登录账户，如 sa 账户；另一种是 Windows 系统账户登录 SQL Server 2012，如 Administrator 账户。创建登录名的步骤如下：

☑ 创建标准登录账户

（1）使用 Microsoft SQL Server Management Studio 连接到需要创建标准登录账户的 SQL Server 2012。

（2）单击“服务器名”/“安全性”/“登录名”展开所连接的服务器，并在登录名界面列中单击鼠标右键，在弹出的快捷菜单中选择“新建登录名”命令，如图 17.1 所示。

图 17.1 Microsoft SQL Server Management Studio 中展开服务器后

（3）在“登录名-新建”窗口中的“登录名”处输入创建的登录名，并选中“SQL Server 身份验证”单选按钮，此时在“密码”及“确认密码”文本框中可以输入创建的登录名登录时所用的密码，如图 17.2 所示。

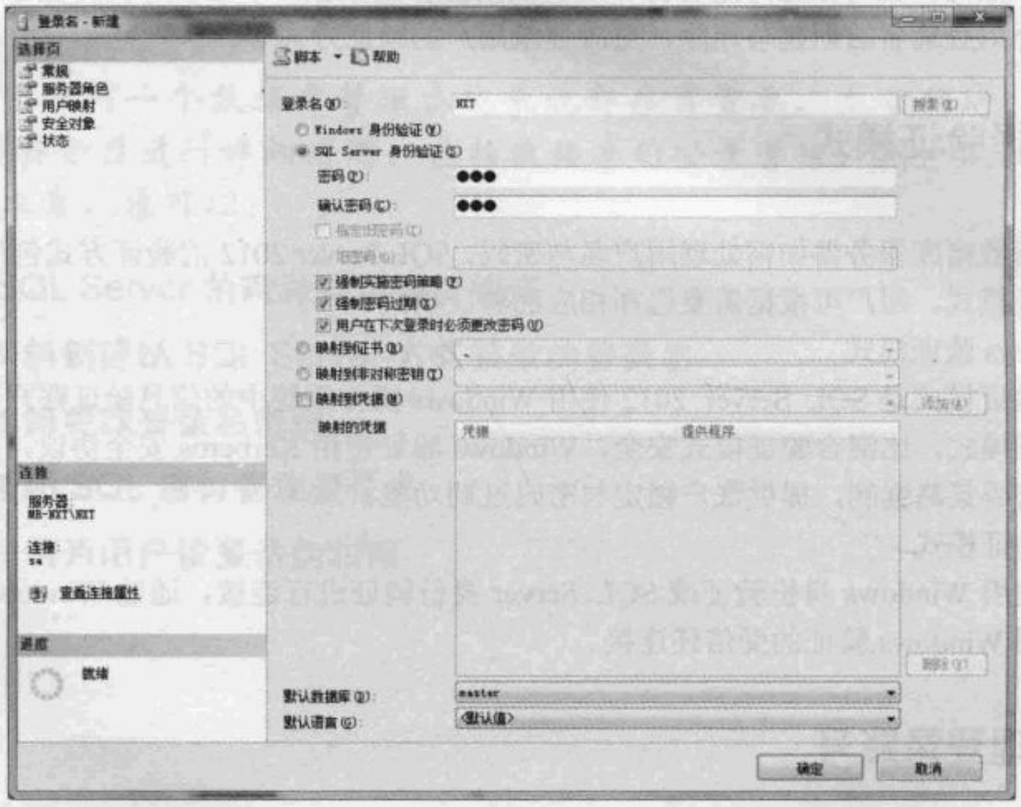


图 17.2 “登录名-新建”窗口

（4）输入要创建的登录名与密码后，单击“确定”按钮即可完成创建标准登录账户。

☑ 创建 Windows 系统账户登录 SQL Server 2012

（1）按照创建标准登录账户的方法打开“登录名-新建”窗口，选中“Windows 身份验证”单选按钮，单击“搜索”按钮。

（2）在弹出的“选择用户或组”对话框中，单击“对象类型”按钮，弹出“对象类型”对话框，如图 17.3 所示，在此对话框中可以选择查找对象的类型。

（3）单击“确定”按钮，在弹出的“选择用户或组”对话框中，单击“位置”按钮，打开“位置”对话框，如图 17.4 所示，在此对话框中选择进行搜索的位置。

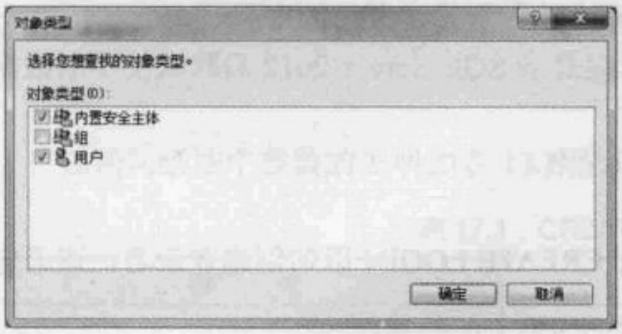


图 17.3　选择对象类型

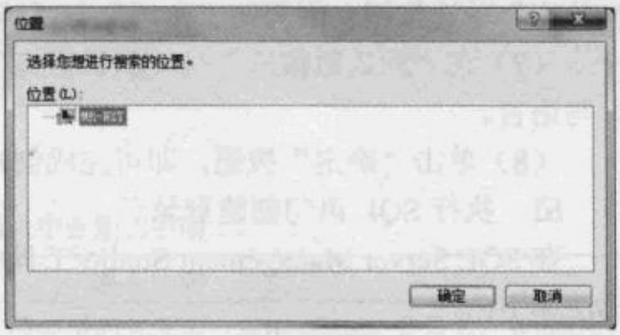


图 17.4　搜索位置

（4）单击“确定”按钮，弹出“选择用户或组”对话框，在文本框内输入要选择的对象名，如图 17.5 所示。

注意

对象名称可以是用户名、计算机名或者组对话框。

（5）单击“确定”按钮进行查找，将创建的系统用户对象添加到“登录名-新建”窗口中的“登录名处”，如图 17.6 所示。

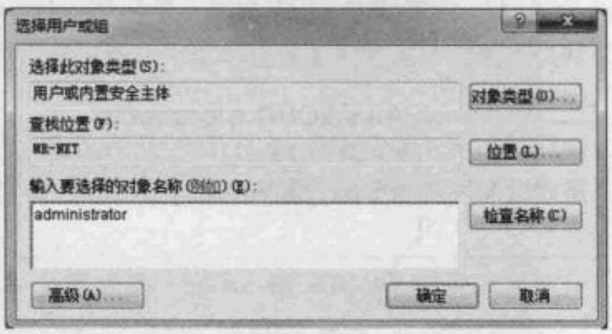


图 17.5　“选择用户或组”对话框

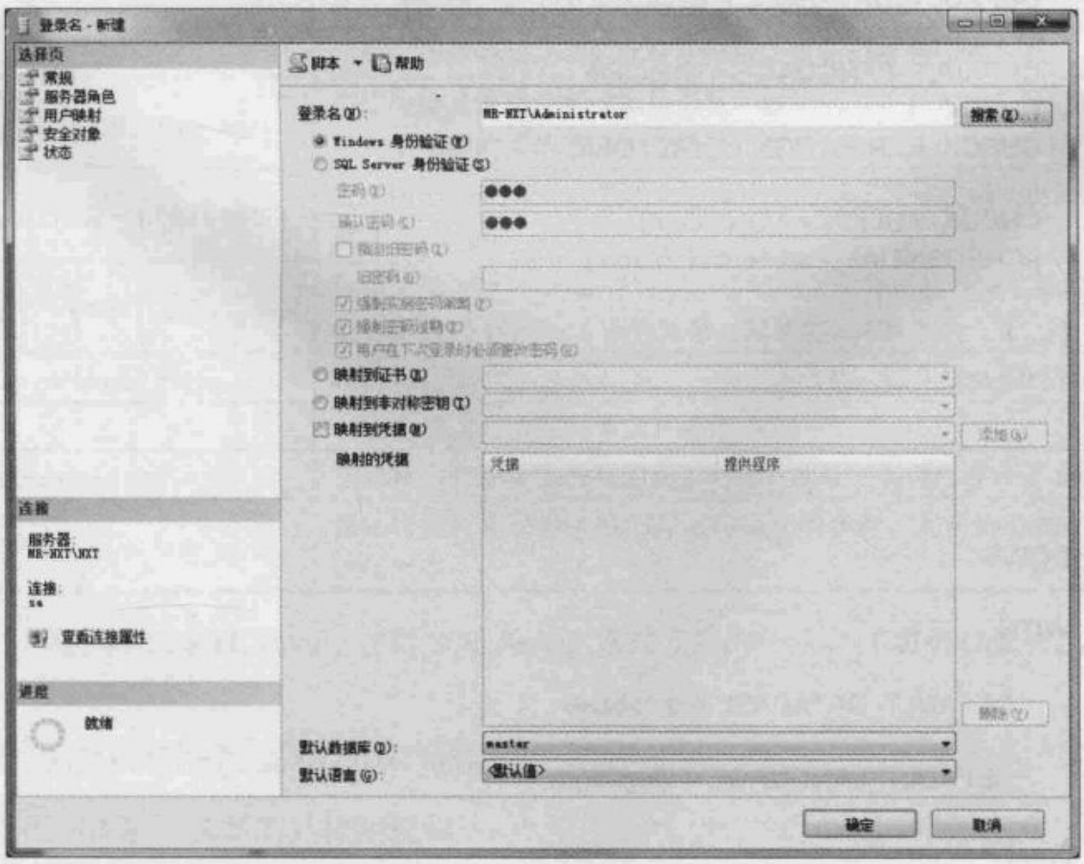


图 17.6　显示创建的登录名

（6）在“登录名-新建”窗口中输入所创建登录名的名称。若选中“Windows 身份验证”单选按钮，可通过单击“搜索”按钮，查找并添加 Windows 操作系统中的用户名称；若选中“SQL Server 身份验证”单选按钮，则需在“密码”与“确认密码”文本框中输入登录时采用的密码。

（7）在“默认数据库”与“默认语言”中选择该登录名登录 SQL Server 2012 后默认使用的数据库与语言。

（8）单击“确定”按钮，即可完成创建 SQL Server 登录名。

☑ 执行 SQL 语句创建登录名

在 SQL Server Management Studio 工具中也可通过执行 CREATE LOGIN 语句创建登录名。该语句语法如下：

```
CREATE LOGIN login_name
  {
    WITH
      <
        PASSWORD = 'password'
        [ HASHED ]
        [ MUST_CHANGE ]
        [
          ,
          <
            SID = sid
            |
            DEFAULT_DATABASE = database
            |
            DEFAULT_LANGUAGE = language
            |
            CHECK_EXPIRATION = { ON | OFF}
            |
            CHECK_POLICY = { ON | OFF}
            [ CREDENTIAL = credential_name ]
          >
          [ ,... ]
        ]
      >
    |
    FROM
    <
      WINDOWS
        [
          WITH
            <
              DEFAULT_DATABASE = database
              |
              DEFAULT_LANGUAGE = language
            >
          [ ,... ]
        ]
```

```
    |
    CERTIFICATE certname
    |
    ASYMMETRIC KEY asym_key_name
    >
}
```

该语句语法中参数的说明如表 17.1 所示。

表 17.1　CREATE LOGIN 语句语法中参数的说明

参　数	说　明
login_name	指定创建的登录名。有 4 种类型的登录名：SQL Server 登录名、Windows 登录名、证书映射登录名和非对称密钥映射登录名。如果从 Windows 域账户映射 login_name，则 login_name 必须用方括号（[ ]）括起来
PASSWORD = 'password'	仅适用于 SQL Server 登录名。指定正在创建的登录名的密码。此值提供时可能已经过哈希运算
HASHED	仅适用于 SQL Server 登录名。指定在 PASSWORD 参数后输入的密码已经过哈希运算。如果未选择此选项，则在将作为密码输入的字符串存储到数据库之前，对其进行哈希运算
MUST_CHANGE	仅适用于 SQL Server 登录名。如果包括此选项，则 SQL Server 将在首次使用新登录名时提示用户输入新密码
SID = sid	仅适用于 SQL Server 登录名。指定新 SQL Server 登录名的 GUID。如果未选择此选项，则 SQL Server 将自动指派 GUID
DEFAULT_DATABASE = database	指定将指派给登录名的默认数据库。默认设置为 master 数据库
DEFAULT_LANGUAGE = language	指定将指派给登录名的默认语言，默认语言设置为服务器的当前默认语言。即使服务器的默认语言发生更改，登录名的默认语言仍保持不变
CHECK_EXPIRATION = { ON | OFF }	仅适用于 SQL Server 登录名。指定是否对此登录名强制实施密码过期策略。默认值为 OFF
CHECK_POLICY = { ON | OFF }	仅适用于 SQL Server 登录名。指定应对此登录名强制实施运行 SQL Server 的计算机的 Windows 密码策略。默认值为 ON
CREDENTIAL = credential_name	将映射到新 SQL Server 登录名的凭据名称。该凭据必须已存在于服务器中
WINDOWS	指定将登录名映射到 Windows 登录名
CERTIFICATE certname	指定将与此登录名关联的证书名称。此证书必须已存在于 master 数据库中
ASYMMETRIC KEY asym_key_name	指定将与此登录名关联的非对称密钥的名称。此密钥必须已存在于 master 数据库中

【例 17.1】使用 CREATE 语句创建以 SQL Server 方式登录的登录名。（**实例位置：光盘\TM\sl\17\1**）代码如下：

```
CREATE LOGIN Mr WITH PASSWORD = 'MrSoft'
```

执行 SQL 语句创建登录名的具体步骤如下：

（1）通过“开始”/“所有程序”/ Microsoft SQL Server 2012 / SQL Server Management Studio 菜单

启动 SQL Server Management Studio 工具。

（2）通过弹出的“连接到服务器”对话框，输入服务器名称，并选择登录服务器使用的身份验证模式，输入用户名与密码，单击“连接”按钮连接到服务器中。

（3）单击工具栏中的 新建查询(N) 按钮，打开“查询编辑器”。该编辑器可以用来创建和运行 Transact-SQL 脚本，如图 17.7 所示。

（4）在查询编辑器内编辑创建登录名的 SQL 语句。通过 F5 键执行编辑的 SQL 语句，完成创建登录名操作，如图 17.8 所示。

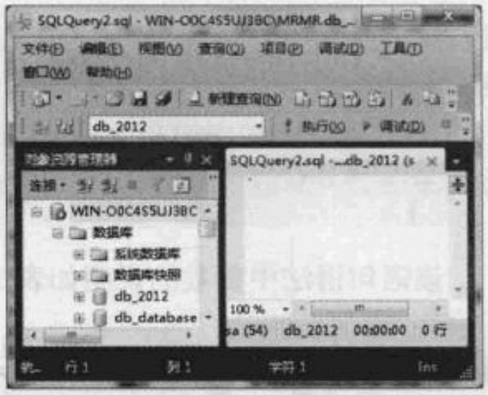


图 17.7　查询编辑器

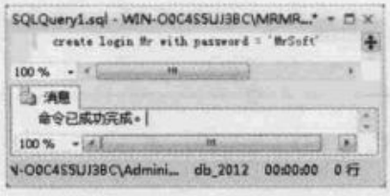


图 17.8　执行 SQL 语句创建登录名

2. 修改登录名

☑ 手动修改登录名

（1）通过“开始”/“所有程序”/Microsoft SQL Server 2012/SQL Server Management Studio 菜单启动 SQL Server Management Studio 工具。

（2）通过弹出的“连接到服务器”对话框，输入服务器名称，并选择登录服务器使用的身份验证模式，输入用户名与密码，单击“连接”按钮连接到服务器中。

（3）单击“对象资源管理器”中的⊞号，依次展开服务器名称/“安全性”/“登录名”。

（4）选择“登录名”下需要修改的登录名，单击鼠标右键，在弹出的快捷菜单中选择“属性”命令，如图 17.9 所示。

（5）在弹出的“登录属性”窗口中修改有关该登录名的信息，如图 17.10 所示，单击“确定”按钮即可完成修改。

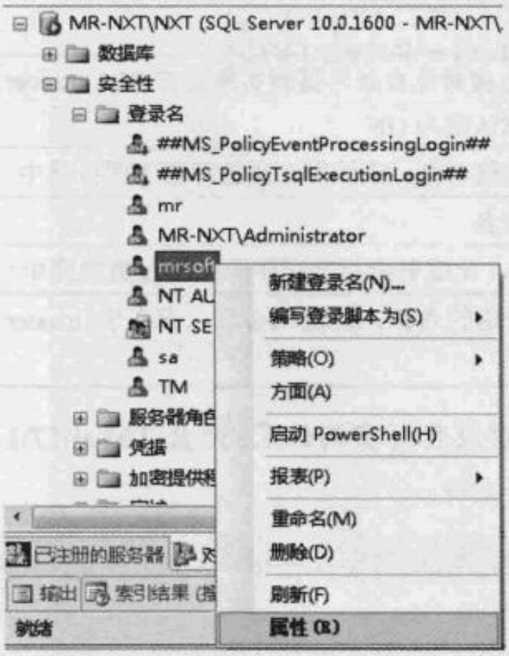


图 17.9　修改登录名

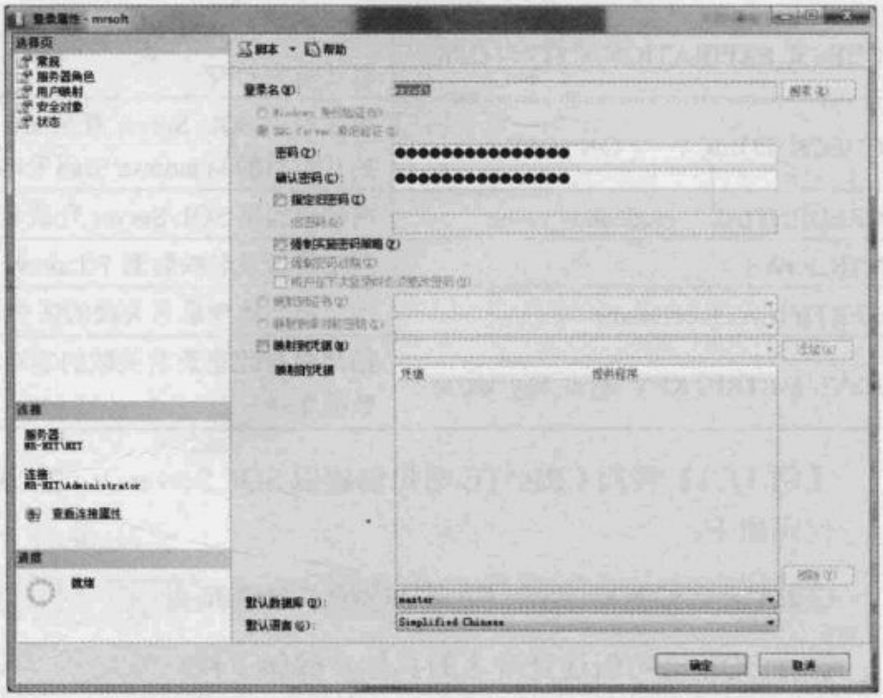


图 17.10　“登录属性”窗口

☑　执行 SQL 语句修改登录名

通过执行 ALTER LOGIN 语句，也可以修改更改 SQL Server 登录名的属性。该语句语法如下：

```
ALTER LOGIN login_name
  {
    <
      ENABLE | DISABLE
    >
    |
    WITH
      <
        PASSWORD = 'password'
        [
          OLD_PASSWORD = 'oldpassword'
          | <MUST_CHANGE | UNLOCK>
          [ <MUST_CHANGE | UNLOCK> ]
        ]
        | DEFAULT_DATABASE = database
        | DEFAULT_LANGUAGE = language
        | NAME = login_name
        | CHECK_POLICY = { ON | OFF }
        | CHECK_EXPIRATION = { ON | OFF }
        | CREDENTIAL = credential_name
        | NO CREDENTIAL
      >
      [,... ]
  }
```

该语句语法中参数的说明如表 17.2 所示。

表 17.2　ALTER LOGIN 语句语法参数的说明

参　数	说　明
login_name	指定正在更改的 SQL Server 登录的名称
ENABLE | DISABLE	启用或禁用此登录
PASSWORD = 'password'	仅适用于 SQL Server 登录账户。指定正在更改的登录的密码
OLD_PASSWORD = 'oldpassword'	仅适用于 SQL Server 登录账户。要指派新密码的登录的当前密码
MUST_CHANGE	仅适用于 SQL Server 登录账户。如果包括此选项，则 SQL Server 将在首次使用已更改的登录时提示输入更新的密码
UNLOCK	仅适用于 SQL Server 登录账户。指定应解锁被锁定的登录
DEFAULT_DATABASE = database	指定将指派给登录的默认数据库
DEFAULT_LANGUAGE = language	指定将指派给登录的默认语言
NAME = login_name	正在重命名的登录的新名称。如果是 Windows 登录，则与新名称对应的 Windows 主体的 SID 必须匹配与 SQL Server 中的登录相关联的 SID。SQL Server 登录的新名称不能包含反斜杠字符（\）
CHECK_POLICY = { ON | OFF }	仅适用于 SQL Server 登录账户。指定应对此登录账户强制实施运行 SQL Server 的计算机的 Windows 密码策略。默认值为 ON

续表

参　数	说　明
CHECK_EXPIRATION = { ON | OFF }	仅适用于 SQL Server 登录账户。指定是否对此登录账户强制实施密码过期策略。默认值为 OFF
CREDENTIAL = credential_name	将映射到 SQL Server 登录的凭据的名称。该凭据必须已存在于服务器中
NO CREDENTIAL	删除登录到服务器凭据的当前所有映射

【例 17.2】使用 ALTER 语句更改 SQL Server 登录方式的登录名密码。（**实例位置：光盘\TM\sl\17\2**）

代码如下：

```
ALTER LOGIN Mr WITH PASSWORD = 'MrSoft'
```

执行 SQL 语句修改登录名属性的具体步骤如下：

（1）通过“开始”/“所有程序”/ Microsoft SQL Server 2012 / SQL Server Management Studio 菜单启动 SQL Server Management Studio 工具。

（2）通过弹出的“连接到服务器”对话框，输入服务器名称，并选择登录服务器使用的身份验证模式，输入用户名与密码，单击“连接”按钮连接到服务器中。

（3）单击工具栏中的 新建查询(N) 按钮，打开“Transact-SQL 查询编辑器”。

（4）在 Transact-SQL 查询编辑器内编辑修改登录名的 SQL 语句。通过 F5 键执行编辑的 SQL 语句，完成修改登录名操作，如图 17.11 所示。

3. 删除登录名

（1）使用 Microsoft SQL Server Management Studio 连接到需要删除登录名的 SQL Server 2012。

（2）选择服务器名/“安全性”/“登录名”展开所连接的服务器，并在登录名界面列中选择需要删除的登录名，单击鼠标右键，在弹出的快捷菜单中选择“删除”命令，如图 17.12 所示。

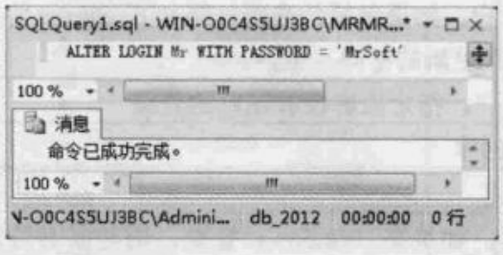


图 17.11　执行 SQL 语句修改登录名属性

图 17.12　在 Microsoft SQL Server Management Studio 中选择要删除的登录名

（3）在弹出的“删除对象”窗口中单击“确定”按钮，即可删除该登录名，如图 17.13 所示。

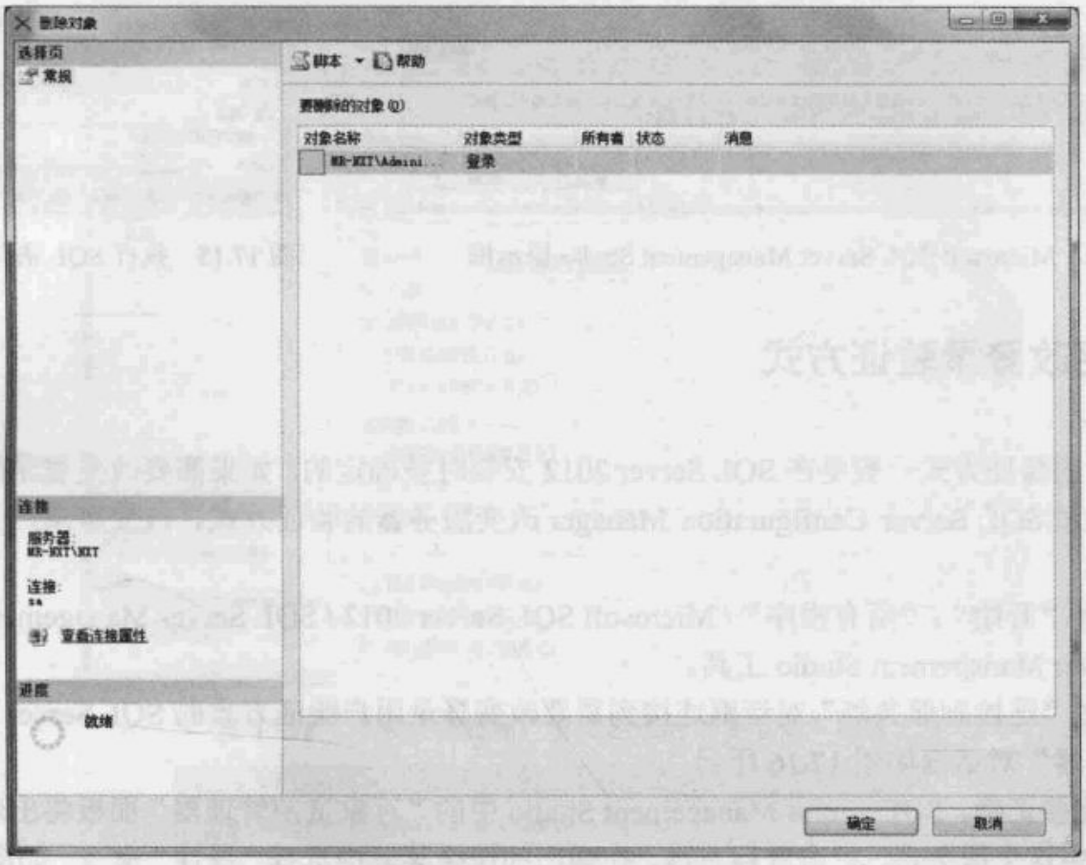


图 17.13　“删除对象”窗口

（4）单击“确定”按钮，在弹出的 Microsoft SQL Server Management Studio 提示框中单击“确定”按钮，即可完成登录名的删除，如图 17.14 所示。

☑ 执行 SQL 语句删除登录名

通过执行 DROP LOGIN 语句可以将 SQL Server 2012 中的登录名删除，该语句语法如下：

```
DROP LOGIN login_name                    //login_name 为指定要删除的登录名
```

【例 17.3】 使用 DROP 语句删除“Mr”登录名。（**实例位置：光盘\TM\sl\17\3**）

代码如下：

```
DROP LOGIN Mr
```

执行 SQL 语句删除登录名的具体步骤如下：

（1）通过“开始”/“所有程序”/ Microsoft SQL Server 2012 / SQL Server Management Studio 菜单启动 SQL Server Management Studio 工具。

（2）通过弹出的“连接到服务器”对话框，输入服务器名称，并选择登录服务器使用的身份验证模式，输入用户名与密码，单击“连接”按钮连接到服务器中。

（3）单击工具栏中的 新建查询(N) 按钮，打开 Transact-SQL 查询编辑器。

（4）在查询编辑器内编辑删除登录名的 SQL 语句。通过 F5 键执行编辑的 SQL 语句，完成删除登录名操作，如图 17.15 所示。

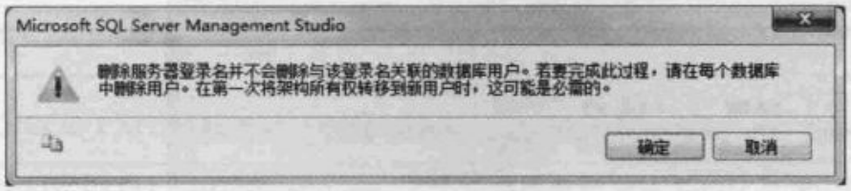


图 17.14　Microsoft SQL Server Management Studio 提示框

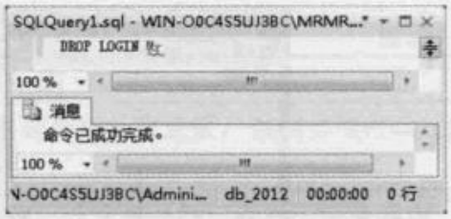


图 17.15　执行 SQL 语句删除登录名

17.2.3　更改登录验证方式

登录用户的验证方式一般是在 SQL Server 2012 安装时被确定的。如果需要改变登录用户的验证方式，只可以通过 SQL Server Configuration Manager 改变服务器的验证方式。改变登录用户验证方式的步骤如下：

（1）选择“开始”/“所有程序”/ Microsoft SQL Server 2012 / SQL Server Management Studio 菜单启动 SQL Server Management Studio 工具。

（2）通过“连接到服务器”对话框连接到需要改变登录用户验证方式的 SQL Server 2012 服务器。“连接到服务器”对话框如图 17.16 所示。

（3）若连接正确，SQL Server Management Studio 中的“对象资源管理器”面板将出现刚刚所连接的服务器。选中这个服务器，单击鼠标右键，在弹出的快捷菜单中选择“属性”命令，如图 17.17 所示。

图 17.16　“连接到服务器”对话框

图 17.17　SQL Server Management Studio

（4）在弹出的“服务器属性”窗口中的“选项页”区域中选择“安全性”，如图 17.18 所示。

（5）在“服务器身份验证”框架内重新选择登录用户的验证方式。选择完成后单击“确定”按钮，这时会弹出 Microsoft SQL Server Management Studio 提示框，提示重新启动 SQL Server 后所做的更改才会生效。Microsoft SQL Server Management Studio 提示框如图 17.19 所示。

（6）单击 Microsoft SQL Server Management Studio 提示框中的“确定”按钮后，重新启动 SQL Server，即可更改登录用户验证方式。

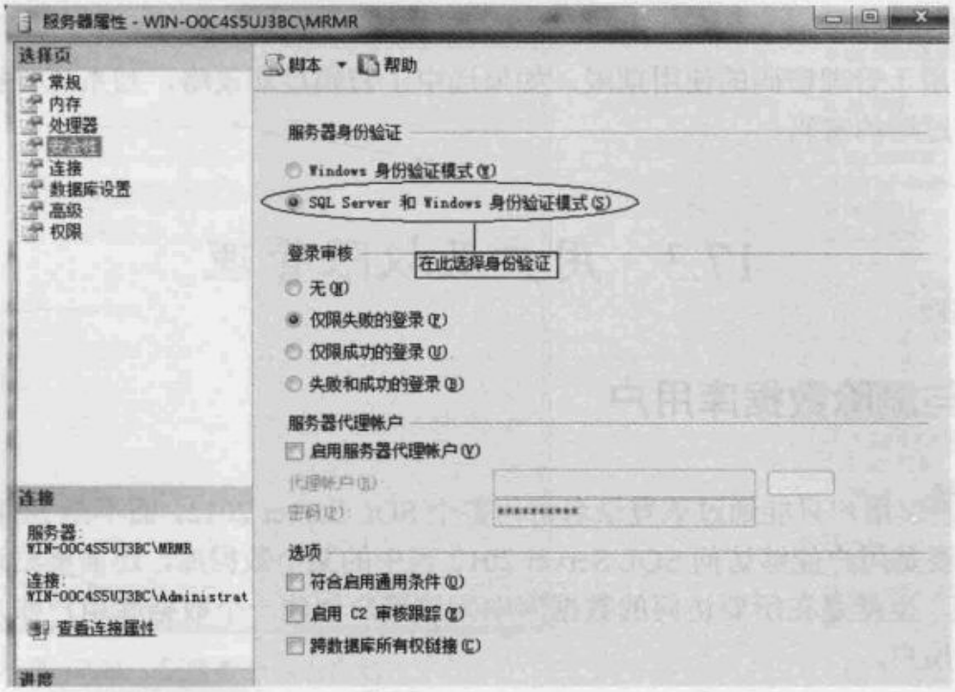


图 17.18　“服务器属性”窗口显示的“安全性”页面

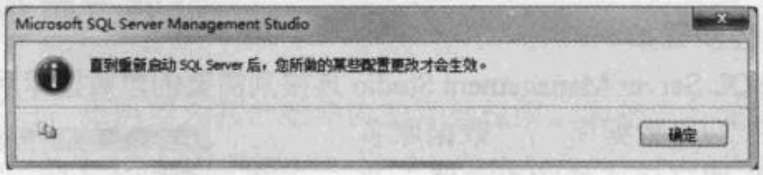


图 17.19　Microsoft SQL Server Management Studio 提示框

17.2.4　设置密码

SQL Server 中的密码最多可包含 128 个字符，其中包括字母、符号和数字。由于在 Transact-SQL 语句（以下简称 SQL 语句）中经常使用登录名、用户名、角色和密码，所以必须用英文双引号（"）或方括号（[ ]）分隔某些符号，例如，SQL Server 登录名、用户、角色或密码中含有空格、以空格开头、以$或@字符开头等，都需要在 Transact-SQL 语句中使用分隔符。

SQL Server 2012 运行在 Windows Server 2003 或更高的操作环境时，可以使用 Windows 的密码机制。Windows Server 2003 密码中使用的复杂性策略和过期策略可以应用于 SQL Server 内部。

（1）密码复杂性策略。

密码复杂性策略通过增加可用的密码数量来阻止强力攻击，实施该策略时，密码必须符合以下原则：

- ☑ 密码不得包含全部或“部分”用户账户名。部分账户名是指 3 个或 3 个以上两端用“空白”（空格、制表符、回车符等）或“-”、“_”、“#”等字符分隔的连续字母数字字符。
- ☑ 密码长度至少为 6 个字符。
- ☑ 密码包含英文大写字母（A～Z）、英文小写字母（a～z）、10 个基本数字（0～9）、非字母数字（例如，!、$、#或%）等 4 类字符中的 3 类。

（2）密码过期策略。

密码过期策略用于管理密码的使用期限。如果选中了密码过期策略，则系统将提醒用户更改旧密码和账户，并禁用过期的密码。

17.3 用户及权限管理

17.3.1 创建与删除数据库用户

登录名创建之后，用户只能通过该登录名访问整个 SQL Server 2012，而不是 SQL Server 2012 当中的某个数据库。若要是用户能够访问 SQL Server 2012 当中的某个数据库，还需要给这个用户授予访问某个数据库的权限，也就是在所要访问的数据库中为该用户创建一个数据库用户账户。本节介绍如何创建及删除数据库用户。

1. 创建数据库用户

创建数据库用户的具体步骤如下：

（1）使用 Microsoft SQL Server Management Studio 连接到需要创建数据库用户的 SQL Server 2012。

（2）单击“服务器名”/“数据库”/“数据库名称”/“安全性”/“用户”展开所连接的服务器，并在用户界面列中单击鼠标右键，在弹出的快捷菜单中选择“新建用户”命令，如图 17.20 所示。

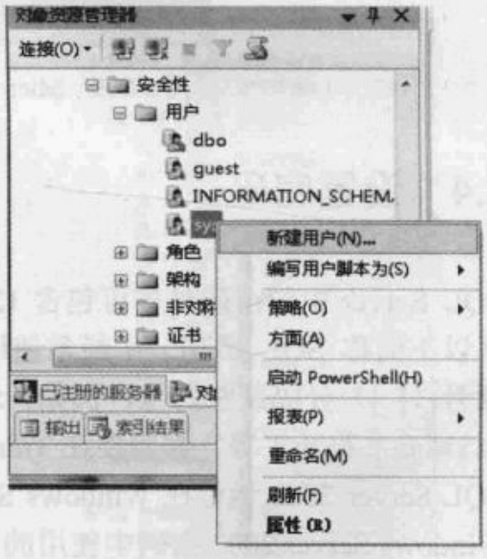


图 17.20 Microsoft SQL Server Management Studio

（3）在弹出的“数据库用户-新建”窗口中输入操作数据库的用户名，以及登录服务器的登录名，并选择其相应的架构与数据库中的角色。例如，新建用户名为“mr”，并分配架构与角色如图 17.21 所示。

2. 删除数据库用户

删除数据库用户的具体步骤如下：

（1）通过“开始”/“程序”/ Microsoft SQL Server 2012 / SQL Server Management Studio 菜单启动 SQL Server Management Studio 工具。

（2）通过弹出的“连接到服务器”对话框，输入服务器名称，并选择登录服务器使用的身份验证模式，输入用户名与密码，单击“连接”按钮连接到服务器中。

（3）单击“对象资源管理器”中的⊞号，依次展开服务器名称/“数据库”/数据库名称/“安全性”/“用户”，在删除的用户上单击鼠标右键，选择快捷菜单中的“删除”命令，如图 17.22 所示。

（4）通过“删除”菜单命令，打开“删除对象”窗口，在“删除对象”窗口中确认删除的用户名称，单击“确定”按钮即可将该用户删除。

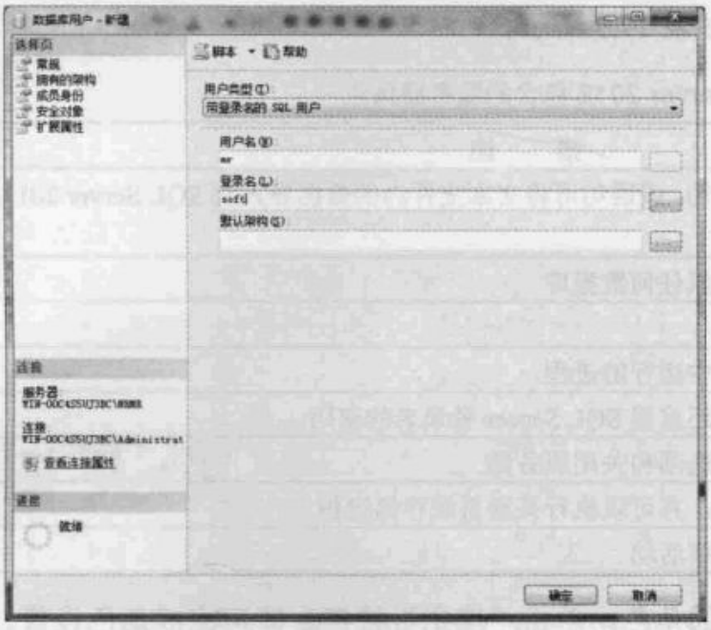


图 17.21　创建数据库用户名

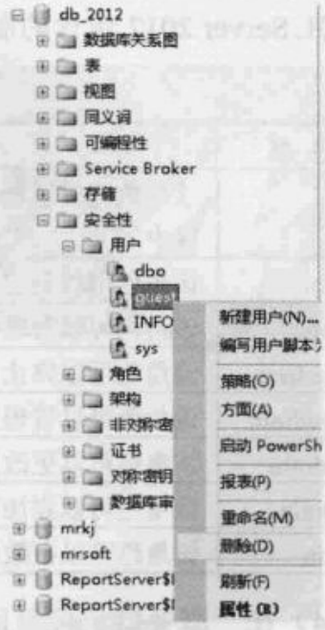


图 17.22　删除用户

17.3.2　设置服务器角色权限

创建完相应的登录名后，还需要为其分配相应的管理权限。为登录名设置角色权限的步骤如下：

（1）使用 Microsoft SQL Server Management Studio 连接到需要分配角色权限的 SQL Server 2012。

（2）单击“服务器名”/“安全性”/“登录名”展开所连接的服务器，选择需要设置权限的登录名，单击鼠标右键，在弹出的快捷菜单中选择“属性”命令，打开“登录属性”窗口，如图 17.23 所示。

（3）在“登录属性”窗口中的“选项页”区域中选择“服务器角色”，如图 17.24 所示。“服务器角色”页面包含的角色都是 SQL Server 2012 固有的，不允许改变。这些角色的权限涵盖了 SQL Server 2012 管理中的各个方面。

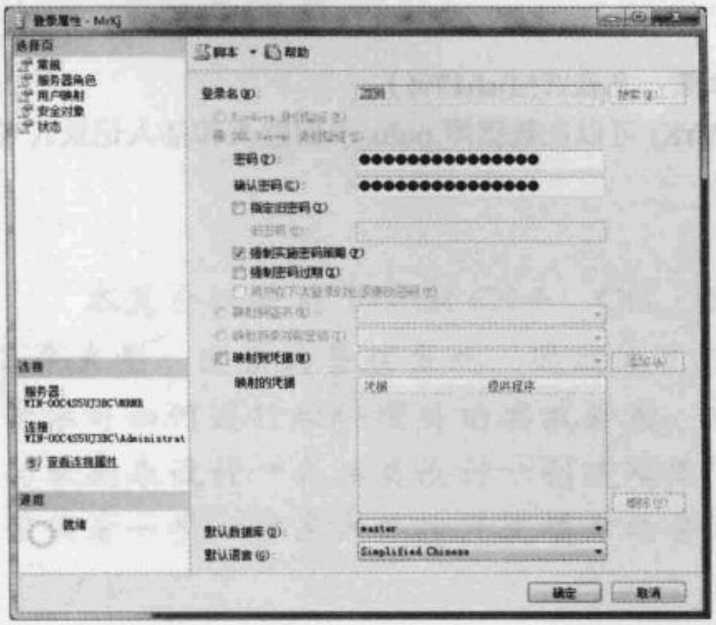


图 17.23　“登录属性”窗口

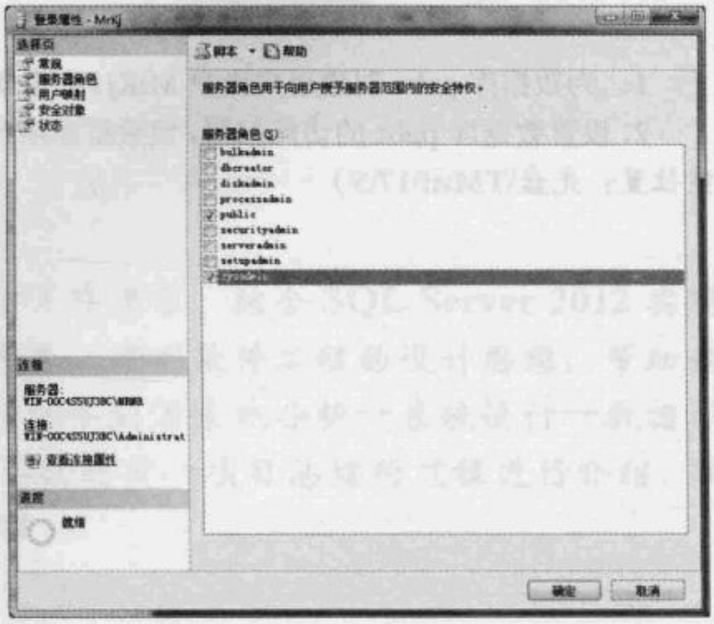


图 17.24　“登录属性”窗口显示的“服务器角色”页面

SQL Server 2012 包含的服务器角色说明如表 17.3 所示。

表 17.3　SQL Server 2012 包含的服务器角色

角 色 名	描　　述
bulkadmin	该角色可以运行 BULK INSERT 语句。该语句可将文本文件内的数据导入到 SQL Server 2012 的数据库中
dbcreator	该角色可以创建、更改、删除和还原任何数据库
diskadmin	该角色可以管理磁盘文件
processadmin	该角色可以终止在数据库引擎实例中运行的进程
securityadmin	该角色可以管理登录名及其属性，还重置 SQL Server 登录名的密码
serveradmin	该角色可以更改服务器范围的配置选项和关闭服务器
setupadmin	该角色可以添加和删除连接服务器，并可以执行某些系统存储过程
sysadmin	该角色可以在数据库引擎中执行任何活动

（4）在“服务器角色”区域中选中相应的角色，单击“确定”按钮，即可完成角色设置。

17.4　小　　结

本章介绍了加强 SQL Server 2012 安全管理的方式，例如，SQL Server 身份验证、创建数据库用户、SQL Server 角色和 SQL Server 权限等。读者应该熟悉两种 SQL Server 身份验证模式，并能够进行创建和管理登录账户、为数据库指定用户、为 SQL Server 角色添加或删除用户等操作。

17.5　实践与练习

1．为数据库 pubs 创建用户账户 MrKj。（**答案位置：光盘\TM\sl\17\4**）

2．设置数据库 pubs 的访问权限，使数据库用户 MrKj 可以在数据库 pubs 中创建表和插入记录。（**答案位置：光盘\TM\sl\17\5**）

项目实战

本篇分别使用 Visual C++、C#、Java 3 种语言，结合 SQL Server 2012 实现 3 个大型、完整的管理系统，通过这 3 个项目，运用软件工程的设计思想，帮助读者学习如何进行软件项目的实践开发。书中按照编写系统分析→系统设计→数据库与数据表设计→公共类设计→创建项目→实现项目→项目总结的过程进行介绍，带领读者一步一步亲身体验开发项目的全过程。

第18章

Visual C++ + SQL Server 实现图书管理系统

（视频讲解：66 分钟）

随着现代图书流通市场竞争愈演愈烈，如何以一种便捷的管理方式加快图书流通信息的反馈速度，降低图书库存占用、缩短资金周转时间、提高工作效率，已经成为能否增强图书企业竞争力的关键问题。在计算机信息技术高速发展的今天，人们已经意识到原有的人工管理方式已经远远不能满足需要，而使用计算机信息系统来管理很明显是如今最有效率的一种手段。本章将通过使用 Visual C++ 6.0 + SQL Server 2012 技术开发一个图书管理系统。

通过阅读本章，您可以：

- 掌握自定义控件的使用
- 掌握菜单绘制方法
- 掌握对数据库的操作方法
- 掌握在对话框程序中调用文档/视图结构打印

18.1　系 统 概 述

视频讲解：光盘\TM\lx\18\系统概述.mp4

信息技术的飞速发展给图书企业的管理带来了全新的变革，采用图书管理系统对图书企业的经营运作进行全程管理，不仅可以使企业摆脱以往人工管理产生的一系列问题，而且可以提高管理效率、降低管理成本、增加经济效益。通过管理系统对图书企业的发展进行规划，可以收集大量关键可靠的数据。企业决策层通过分析这些数据，可以做出合理决策，及时调整，使之能够更好地遵循市场的销售规律，适应市场的变化，从而让企业能够在激烈的行业竞争中占据一席之地。

18.2　系 统 设 计

视频讲解：光盘\TM\lx\18\系统设计.mp4

在开发管理系统时，首先要明确待开发系统的系统目标，即需要实现哪些功能，然后绘制系统功能结构图和业务流程图，这样可以使开发人员在创建项目以前就对项目有一定的了解，从而开发出用户需要的项目。

18.2.1　系统目标

对于图书管理系统，必须要满足使用方便、操作灵活和安全性好等设计需求。本系统在设计时应满足以下几个目标：

- ☑ 采用人机对话的操作方式，界面设计美观友好，操作灵活、方便、快捷、准确，数据存储安全可靠。
- ☑ 实现对图书信息、仓库信息、柜台信息和供应商信息等的管理功能。
- ☑ 可以对图书的定价、库存和销售等信息进行管理。
- ☑ 提供入库查询和销售查询。
- ☑ 提供对查询结果的报表打印。
- ☑ 系统最大限度地实现了易维护性和易操作性。
- ☑ 系统运行稳定、安全可靠。

18.2.2　系统功能结构

图书管理系统功能结构图如图 18.1 所示。

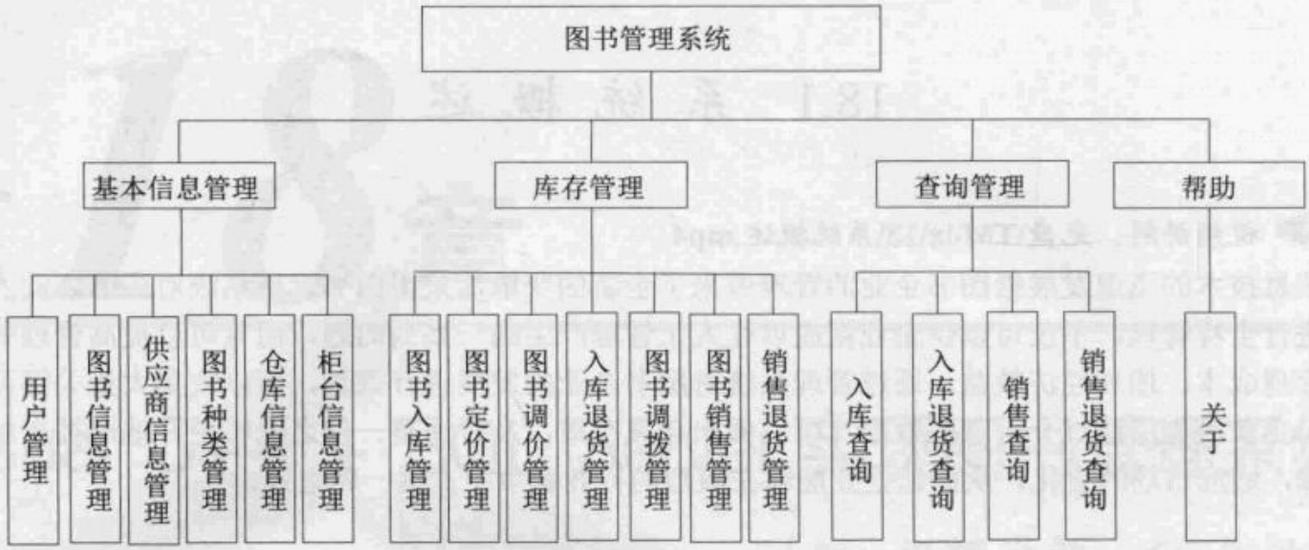


图 18.1　图书管理系统功能结构图

18.2.3　业务流程图

图书管理系统的业务流程图如图 18.2 所示。

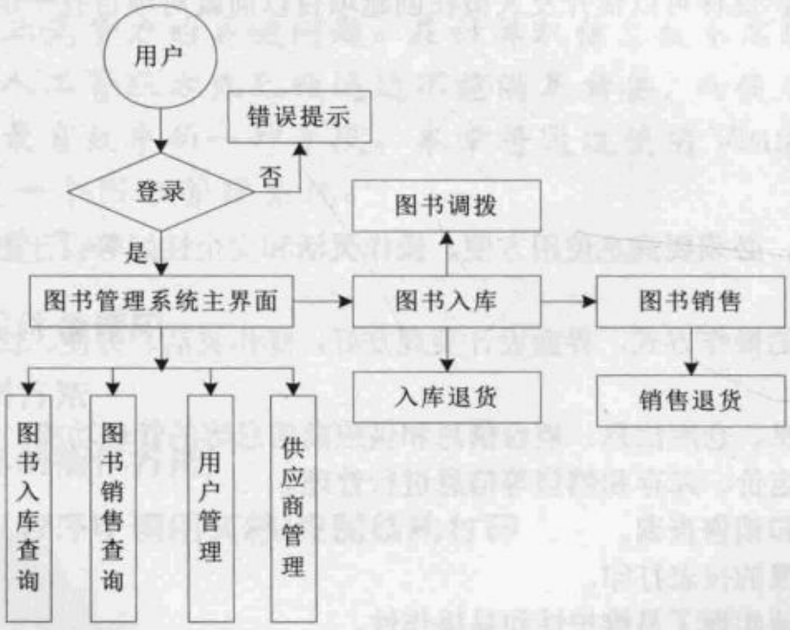


图 18.2　图书管理系统业务流程图

18.3　数据库设计

视频讲解：光盘\TM\lx\18\数据库设计.mp4

要开发一个好的管理系统，数据库的设计是必不可少的。本节从数据库分析入手，逐步介绍数据库和数据表的创建过程。

18.3.1　数据库分析

在图书管理系统中涉及大量的图书信息、图书入库及销售数据，为了更好地对其进行管理，在设计该系统时选择 Microsoft SQL Server 2012 数据库来满足系统的要求，数据库名称为 BookManage。在数据库中创建 19 张表用于存储各种不同信息，如图 18.3 所示。

图 18.3　图书管理系统中使用的数据表

18.3.2　主要数据表结构

下面给出数据库中主要表的结构。

1. tb_bookinfo（图书信息表）

表 tb_bookinfo 主要用于存储图书的书名、作者和售价等信息，结构如表 18.1 所示。

表 18.1　tb_bookinfo 表

字 段 名	数 据 类 型	主　键	描　述
bookname	varchar(30)		书籍名称
shortname	varchar(10)		简码
barcode	varchar(30)	是	条形码
author	varchar(30)		作者
bookconcern	varchar(30)		出版社
price	money		价格
memo	varchar(100)		备注
kind	varchar(30)		种类

2. tb_providerinfo（供应商信息表）

表 tb_providerinfo 用于存储供应商信息，包括供应商名称及联系方式等，结构如表 18.2 所示。

表 18.2 tb_providerinfo 表

字 段 名	数 据 类 型	主 键	描 述
provider	varchar(50)	是	供应商名称
corporation	varchar(30)		法人
principal	varchar(10)		负责人
phone	varchar(30)		联系电话
addr	varchar(50)		地址
web	varchar(50)		网址
e_mail	varchar(30)		电子邮件

3. tb_counterbook（柜台图书表）

表 tb_counterbook 用于存储柜台的图书信息，包括图书数量及条形码等，结构如表 18.3 所示。

表 18.3 tb_counterbook 表

字 段 名	数 据 类 型	主 键	描 述
barcode	varchar(30)		条形码
counter	varchar(30)		柜台
booknum	float		数量

4. tb_instorage_main（图书入库主表）

表 tb_instorage_main 用于存储图书的入库信息，包括入库单号和入库时间等，结构如表 18.4 所示。

表 18.4 tb_instorage_main 表

字 段 名	数 据 类 型	主 键	描 述
ID	varchar(30)	是	入库单号
provider	varchar(50)		供应商
operator	varchar(30)		操作员
rebate	float		折扣
sumtotal	money		总计
paymoney	money		应付金额
factmoney	money		实付金额
intime	datetime		入库时间

5. tb_sell_main（商品销售主表）

表 tb_sell_main 用于存储图书的销售信息，包括销售单号和销售时间等，结构如表 18.5 所示。

表 18.5 tb_sell_main 表

字 段 名	数 据 类 型	主 键	描 述
sellID	varchar(30)	是	销售单号
customer	varchar(30)		顾客

续表

字 段 名	数 据 类 型	主　键	描　述
operator	varchar(30)		操作员
rebate	float		折扣
sumtotal	money		总计
paymoney	money		应付金额
factmoney	money		实付金额
intime	datetime		销售时间
counter	varchar(30)		柜台

6. tb_bookmove_main（图书调拨主表）

表 tb_bookmove_main 用于存储图书的调拨信息，包括调拨单号和调拨时间等，结构如表 18.6 所示。

表 18.6　tb_bookmove_main 表

字 段 名	数 据 类 型	主　键	描　述
moveID	varchar(30)	是	调拨单号
storage	varchar(30)		仓库名称
counter	varchar(30)		柜台名称
operator	varchar(30)		操作员
movetime	datetime		调拨时间

18.4　创 建 工 程

视频讲解：光盘\TM\lx\18\创建工程.mp4

图书管理系统采用对话框结构进行设计，创建工程的步骤如下：

（1）启动 Visual C++ 6.0，选择 File/New 命令，打开 New 对话框，如图 18.4 所示。

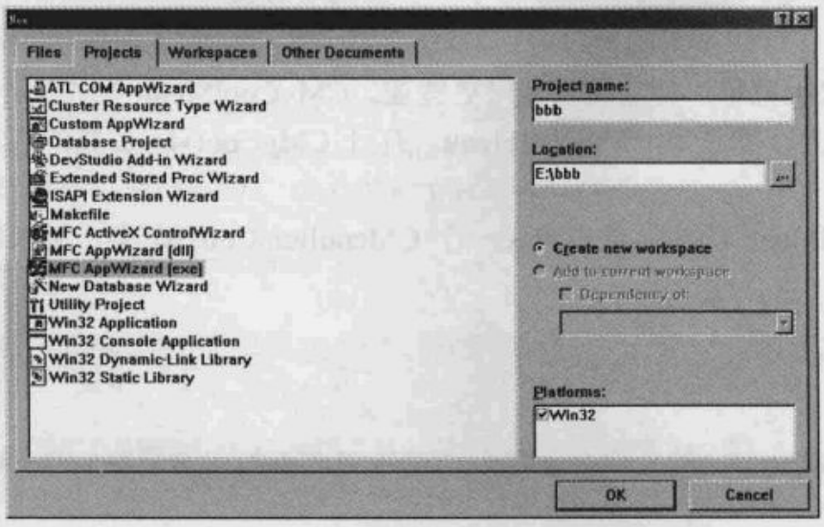


图 18.4　New 对话框

（2）在列表框中选择 MFC AppWizard[exe]选项，单击 OK 按钮，弹出 MFC AppWizard-Step 1 对话框，选中 Dialog based 单选按钮，创建一个基于对话框的应用程序，如图 18.5 所示。

（3）单击 Finish 按钮创建应用程序，新工程如图 18.6 所示。

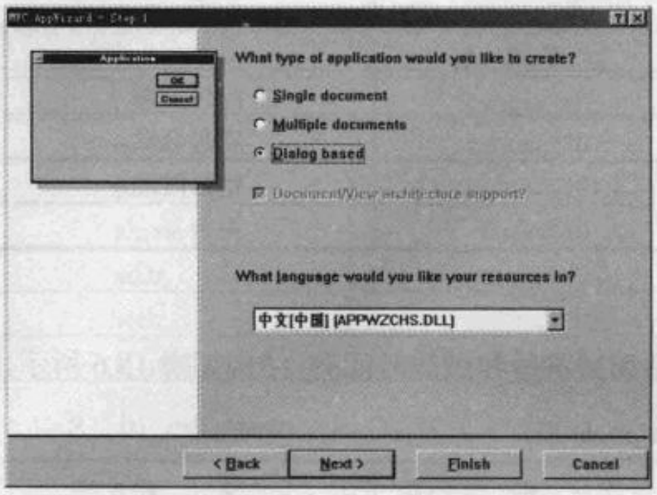


图 18.5　应用程序创建向导

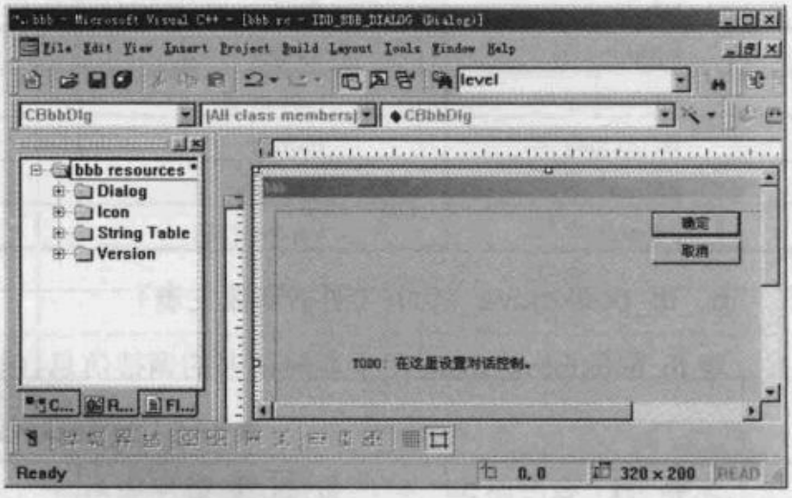


图 18.6　新工程

18.5　公共类设计

视频讲解：光盘\TM\lx\18\公共类设计.mp4

在开发应用程序时，可以将界面绘制的相关操作以及对一些控件的设置封装在自定义类中，以便于在开发程序时调用，这样可以提高代码的重用性。本系统创建了 CMyCoolMenu 类，用于绘制菜单、美化界面；还定义了 CKeyEdit 类和 CCustomGrid 类，这两个类是控件类，用来简化程序操作，提高代码的重用性。

18.5.1　自绘菜单类 CMyCoolMenu

自绘菜单类用来绘制菜单，使系统看起来更美观。CMyCoolMenu 类的设计如下：

（1）创建一个新类，类名为 CMyCoolMenu。打开 CMyCoolMenu 类的头文件，为该类添加基类 CMenu。

（2）在 CMyCoolMenu 的头文件中定义一个 CMenuItemContext 类，用来保存菜单项信息。代码如下：

```
class CMenuItemContext
{
public:
    int     nMenuID;            //-2:顶层菜单条；-1:弹出菜单；0:分隔条；>0:一般的菜单
    CString strText;            //菜单文本
};
```

（3）在 CMyCoolMenu 的头文件中声明成员变量，代码如下：

```
CMenuItemContext lpMenu[100];                                    //菜单项信息
int index;                                                       //菜单索引
```

注意

粗体字部分定义的数组的元素数量一定不要小于所有菜单项之和。

（4）为 CMyCoolMenu 类添加方法。

① AttachMenu 方法。

AttachMenu 方法用于将当前窗口的菜单与自定义的菜单关联，代码如下：

```
BOOL CMyCoolMenu::AttachMenu(HMENU hMenu,CSize sz)
{
    Attach(hMenu);                                               //关联菜单
    ChangeMenuStyle(hMenu, TRUE);                                //改变主菜单风格
    return TRUE;
}
```

② ChangeMenuStyle 方法。

ChangeMenuStyle 方法用于改变主菜单的风格，代码如下：

```
BOOL CMyCoolMenu::ChangeMenuStyle(HMENU hMenu, BOOL bTop = FALSE)
{
    CMenu *pMenu = CMenu::FromHandle(hMenu);                     //获得菜单句柄
    if(pMenu != NULL)                                            //判断句柄是否为空
    {
        for(UINT i = 0; i < pMenu->GetMenuItemCount(); i ++)     //获得菜单数
        {
            lpMenu[index].nMenuID = pMenu->GetMenuItemID(i);     //获得菜单项 ID
            if(lpMenu[index].nMenuID < 0 && bTop)                //判断是否是顶层菜单
            {
                lpMenu[index].nMenuID = -2;                      //标识为顶层菜单
            }
            pMenu->GetMenuString(i, lpMenu[index].strText, MF_BYPOSITION);   //获得菜单项文本
            pMenu->ModifyMenu(i, MF_OWNERDRAW|MF_BYPOSITION |MF_STRING,
                lpMenu[index].nMenuID,LPCTSTR(&lpMenu[index]));  //设置菜单风格
            CMenu *pSubMenu = pMenu->GetSubMenu(i);              //获取弹出式菜单项
            if(pSubMenu && lpMenu[index].nMenuID != -2 && !bTop) //判断是否是弹出菜单
            {
                lpMenu[index].nMenuID = -1;                      //标识为弹出菜单
            }
            index += 1;
            if(pSubMenu)
            {
                ChangeMenuStyle(pSubMenu->GetSafeHmenu());       //递归改变菜单风格
            }
        }
    }
```

```
    return TRUE;
}
```

说明

在 ChangeMenuStyle 方法中递归修改子菜单的风格。

③ DrawTop 方法。

DrawTop 方法用于绘制顶层菜单，代码如下：

```
void CMyCoolMenu::DrawTop(CDC *pDC,CRect rect,BOOL bSelected)
{
    if(bSelected)                                                    //判断菜单项是否选中
    {
        pDC->SelectStockObject(BLACK_PEN);                           //设置画笔颜色
        pDC->Rectangle(&rect);                                       //绘制矩形区域
        rect.DeflateRect(1, 1);                                      //设置区域大小
        pDC->FillSolidRect(&rect, RGB(185, 185, 255));               //向矩形中填充颜色
    }
    else
    {
        CRect rtWnd;                                                 //声明区域对象
        AfxGetMainWnd()->GetClientRect(&rtWnd);                      //获得客户区域
        CRect rcGray(rect);                                          //设置区域
        rcGray.left = 300;                                           //设置区域左边位置
        rcGray.right = rtWnd.right + 4;                              //设置区域右边位置
        pDC->FillSolidRect(&rcGray,RGB(255, 176, 55));               //填充设置的区域
        CRect rcWhite(rect);                                         //获得菜单区域
        pDC->FillSolidRect(&rcWhite,RGB(255, 176, 55));              //填充菜单区域
    }
}
```

④ DrawBestRect 方法。

DrawBestRect 方法用于绘制菜单效果，代码如下：

```
void CMyCoolMenu::DrawBestRect(CDC *pDC, CRect rect, COLORREF cr1,COLORREF cr2, BOOL bHor)
{
    int r1 = GetRValue(cr1);                                         //设置颜色值
    int g1 = GetGValue(cr1);                                         //设置颜色值
    int b1 = GetBValue(cr1);                                         //设置颜色值
    int r2 = GetRValue(cr2);                                         //设置颜色值
    int g2 = GetGValue(cr2);                                         //设置颜色值
    int b2 = GetBValue(cr2);                                         //设置颜色值
    float dr = ((float)(r2 - r1))/(float)(rect.Width());             //设置 R 颜色值
    float dg = ((float)(g2 - g1))/(float)(rect.Width());             //设置 G 颜色值
    float db = ((float)(b2 - b1))/(float)(rect.Width());             //设置 B 颜色值
    for(int i = rect.left; i < rect.right; i ++)                     //循环设置颜色渐变
    {
        int r = r1 + (int)(dr*((float)(i - rect.left)));             //设置 R 颜色值
        int g = g1 + (int)(dg*((float)(i - rect.left)));             //设置 G 颜色值
```

```
            int b = b1 + (int)(db*((float)(i - rect.left)));          //设置 B 颜色值
            CPen pen(PS_SOLID, 1, RGB(r, g, b));                     //设置画笔颜色
            CPen *old = pDC->SelectObject(&pen);                     //选入画笔
            pDC->MoveTo(i, rect.top);                                //画线起始点
            pDC->LineTo(i, rect.bottom);                             //画线终点
        }
}
```

⑤ DrawBGColor 方法。

DrawBGColor 方法用于绘制菜单项的背景，代码如下：

```
void CMyCoolMenu::DrawBGColor(CDC* pDC, CRect rect, BOOL bSelected)
{
    if(bSelected)                                                    //菜单被选中
    {
        pDC->SelectStockObject(NULL_BRUSH);                          //设置画刷
        pDC->SelectStockObject(BLACK_PEN);                           //设置画笔
        pDC->Rectangle(&rect);                                       //绘制矩形
        rect.DeflateRect(1, 1);                                      //设置区域
        DrawBestRect(pDC,rect,100,255,TRUE);                         //绘制菜单效果
    }
    else                                                             //菜单未被选中
    {
        CRect rcGray(rect);                                          //设置区域
        rcGray.right = rcGray.left + Public_Area;                    //设置区域右边
        pDC->FillSolidRect(&rcGray, RGB(0, 0, 0));                   //填充区域
        CRect rcWhite(rect);                                         //设置区域
        rcWhite.left = rcGray.right;                                 //设置区域左边
        pDC->FillSolidRect(&rcWhite,RGB(255, 255, 255));             //填充区域
    }
}
```

⑥ DrawText 方法。

DrawText 方法用于绘制菜单文本，代码如下：

```
void CMyCoolMenu::DrawText(CDC* pDC, CRect rect, CString sText)
{
    pDC->DrawText(sText, &rect, DT_LEFT | DT_VCENTER | DT_SINGLELINE);   //绘制菜单文本
}
```

说明

绘制菜单项文本时，设置文本左对齐以及垂直方向居中对齐。

⑦ DrawItem 方法。

DrawItem 方法用于绘制每一个菜单项，代码如下：

```
void CMyCoolMenu::DrawItem(LPDRAWITEMSTRUCT  lpDIS)
{
    CDC* pDC = CDC::FromHandle(lpDIS->hDC);                          //获得菜单上下文
```

```
VERIFY(pDC);pDC->SetBkMode(TRANSPARENT);                                    //设置背景透明
CRect rcItem = lpDIS->rcItem;                                               //获得菜单项区域
UINT uState   = lpDIS->itemState;                                           //获得菜单项状态
if(lpDIS->itemData == NULL)   return;                                       //如果无数据不绘制
CString strText = ((CMenuItemContext*)(lpDIS->itemData))->strText;          //获得菜单文本
UINT nMenuID      = ((CMenuItemContext*)(lpDIS->itemData))->nMenuID;        //获得菜单 ID
CRect rcText(rcItem);                                                       //设置文本区域
switch(nMenuID)
{
case -2:                                                                    // -2:顶层菜单条
      if(uState&ODS_SELECTED)                                               //选中菜单时
      {
            DrawTop(pDC,rcItem,TRUE);                                       //绘制选中菜单
      }
      else                                                                  //未选中时
      {
            DrawTop(pDC,rcItem,FALSE);                                      //绘制顶层菜单
      }
      DrawText(pDC,rcItem,strText);                                         //绘制菜单文本
      break;
case -1:                                                                    //-1:弹出菜单
      strText= "\t    "+strText;                                           //设置菜单文本
      if(uState&ODS_SELECTED)                                               //菜单选中时
            DrawBestRect(pDC,rcItem,RGB(255,0,255),RGB(255,255,255),TRUE); //绘制选中效果
      Else                                                                  //未选中时
            DrawTop(pDC,rcItem,TRUE);                                       //绘制菜单项
      DrawText(pDC,rcText,strText);                                         //绘制菜单文本
      break;
case 0:                                                                     //0:分隔条
      rcText.top      = rcText.Height()/2+rcText.top ;                      //设置分隔条上边
      rcText.bottom = rcText.top +2;                                        //设置分隔条下边
      rcText.left   += 2;                                                   //设置分隔条左边
      rcText.right -= 2;                                                    //设置分隔条右边
      pDC->Draw3dRect(rcText,RGB(64,0,128),RGB(255,255,255));               //绘制分隔条
      break;
default:                                                                    //>0:一般的菜单
      if(uState&ODS_SELECTED)                                               //被选择
      {
            rcItem.left   -= 20;                                            //设置菜单项左边
            rcItem.right -= 0;                                              //设置菜单项右边
            DrawBGColor(pDC,rcItem,TRUE);                                   //绘制背景
      }
      else
      {
            rcItem.left   -= 20;                                            //设置菜单项左边
            rcItem.right -= 0;                                              //设置菜单项右边
            DrawBGColor(pDC,rcItem,TRUE);                                   //绘制背景
      }
      if(uState&ODS_CHECKED)                                                //当前状态是否被选中
```

```
        {
            DrawText(pDC,rcText,"    "+strText);                    //当前状态被选中
            break;
        }
        else
        {
            DrawText(pDC,rcText,"      "+strText);                  //未选取
            break;
        }
        break;
    }
}
```

⑧ MeasureItem 方法。

MeasureItem 方法用于设置每一个菜单项的风格，代码如下：

```
void CMyCoolMenu::MeasureItem(LPMEASUREITEMSTRUCT lpMIS)
{
    lpMIS->itemWidth = 130;                                          //获得菜单项宽度
    lpMIS->itemHeight = 26;                                          //获得菜单项高度
    UINT nMenuID= ((CMenuItemContext*)(lpMIS->itemData))->nMenuID;   //获得菜单 ID
    switch(nMenuID)
    {
    case -2:                                                         //-2:顶层菜单条
        //设置菜单项宽度
        lpMIS->itemWidth =((CMenuItemContext*)(lpMIS->itemData))->strText.GetLength()*5;
        break;
    case -1:                                                         //-1:弹出菜单
        //设置菜单项宽度
        lpMIS->itemWidth =((CMenuItemContext*)(lpMIS->itemData))->strText.GetLength()*10;
        break;
    case 0:                                                          //0:分隔条
        lpMIS->itemHeight =10;                                       //设置分隔条高度
        break;
    default:                                                         //>0:一般的菜单
        break;
    }
}
```

18.5.2　自定义编辑框类 CKeyEdit

CKeyEdit 类派生于编辑框类，使用该类可以限制字符输入，并设置编辑框中的文本颜色。CKeyEdit 类设计如下。

（1）以 CEdit 类为基类派生一个 CKeyEdit 类，在该类的头文件中声明变量。代码如下：

```
CFont       m_font;                                              //文本字体
bool        IsNumber;                                            //是否是数字
COLORREF    color;                                               //文本颜色
```

（2）处理 WM_CHAR 消息，在该消息中限制字符的输入。代码如下：

```
void CKeyEdit::OnChar(UINT nChar, UINT nRepCnt, UINT nFlags)
{
    if (IsNumber)                                                    //判断是否为数字编辑框
        //允许减号、小数点、数字和删除键的输入操作
        if (((nChar <45)||(nChar>46)&&(nChar<48)||(nChar > 57))&& (nChar != 8))
        {
            nChar = 0;
            Beep(100,100);                                           //提示音
        }
        else
            CEdit::OnChar(nChar, nRepCnt, nFlags);
    else
        CEdit::OnChar(nChar, nRepCnt, nFlags);
}
```

注意

如果是数字编辑框，则只允许减号、小数点、数字和删除键的输入操作。

（3）处理 WM_CREATE 消息，在该消息的处理函数中设置文本字体。代码如下：

```
int CKeyEdit::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CEdit::OnCreate(lpCreateStruct) == -1)
        return -1;
    LOGFONT font;                                                    //声明字体对象
    font.lfHeight = -12;                                             //设置高度
    font.lfWidth = 0;                                                //设置宽度
    font.lfItalic = 0;                                               //不是斜体
    font.lfStrikeOut = 0;                                            //不带删除线
    font.lfEscapement = 0;                                           //偏离垂线与 x 轴的夹角
    font.lfUnderline = 0;                                            //不带下划线
    font.lfWeight = 400;                                             //设置字体的磅数
    font.lfCharSet = 134;                                            //设置字体的字符集
    strcpy(font.lfFaceName,"宋体");                                  //设置字体的名称
    m_font.DeleteObject();                                           //释放设备环境
    m_font.CreateFontIndirect(&font);                                //初始化字体
    SetFont(&m_font);                                                //设置字体
    return 0;
}
```

（4）添加 SetEditTextColor 方法，获得要设置的文本颜色。代码如下：

```
void CKeyEdit::SetEditTextColor(COLORREF Color)
{
    color = Color;                                                   //获得颜色
    Invalidate();                                                    //刷新窗口
}
```

（5）处理 WM_CTLCOLOR_REFLECT 消息，设置文本颜色。代码如下：

```
HBRUSH CKeyEdit::CtlColor(CDC* pDC, UINT nCtlColor)
{
    pDC->SetTextColor(color);                                    //设置文本颜色
    CBrush brush(color);                                         //设置画刷颜色
    return brush;
}
```

（6）处理 EN_KILLFOCUS 消息，当失去焦点时将编辑框中的数据添加到列表视图相应的单元格中，并隐藏控件。代码如下：

```
void CKeyEdit::OnKillfocus()
{
    CWnd* temp = NULL;
    temp = GetParent();                                          //获得父窗口句柄
    if (temp != NULL)
    {
        if (temp->IsKindOf(RUNTIME_CLASS(CCustomGrid))==true)    //父窗口是列表视图
        {
            CString str;                                         //声明字符串
            GetWindowText(str);                                  //获得控件内文本
            ((CCustomGrid*)(temp))->SetItemText(((CCustomGrid*)(temp))->row,
                ((CCustomGrid*)(temp))->col,str);                //显示在相应的单元格中
            ShowWindow(SW_HIDE);                                 //隐藏控件
        }
    }
}
```

18.5.3　自定义列表视图类 CCustomGrid

为了方便用户输入多种图书，可采用表格形式进行输入。程序中利用列表视图控件输入商品信息，但 MFC 提供的列表视图控件 CListCtrl 只能显示数据，而不能编辑。为了使列表视图控件能够编辑，这里改写了 CListCtrl 类，从该类派生一个子类 CCustomGrid，使 CCustomGrid 类具有编辑功能。CCustomGrid 类的设计如下。

（1）以 CListCtrl 类为基类派生一个 CCustomGrid 类，在该类的头文件中声明变量。代码如下：

```
bool        showedit;                                            //显示编辑框
int         row;                                                 //记录行
int         col;                                                 //记录列
CKeyEdit    edit;                                                //CKeyEdit 类对象
```

（2）在 PreSubclassWindow 虚方法中设置列表视图的风格，并创建编辑框控件。代码如下：

```
void CCustomGrid::PreSubclassWindow()
{
    ModifyStyle(LVS_EDITLABELS,0);                               //设置允许编辑
    ModifyStyle(0,LVS_REPORT);                                   //设置报表风格
    SetExtendedStyle(LVS_EX_FULLROWSELECT|LVS_EX_HEADERDRAGDROP|
        LVS_EX_GRIDLINES|LVS_EX_ONECLICKACTIVATE|LVS_EX_FLATSB);//设置扩展风格
```

```
    edit.Create(WS_CHILD|WS_CLIPSIBLINGS|WS_EX_TOOLWINDOW|WS_BORDER,
        CRect(0,40,10,50),this,1001);                                    //创建编辑框
    CListCtrl::PreSubclassWindow();
}
```

（3）添加 ShowEdit 方法，用于在列表中显示编辑框。代码如下：

```
void CCustomGrid::ShowEdit()
{
    CRect rect;                                                          //记录当前单元格的坐标
    GetSubItemRect(row,col,LVIR_LABEL,rect);                             //获得单元格区域
    CString str;                                                         //声明字符串
    str = GetItemText(row,col);                                          //获得单元格的文本
    edit.MoveWindow(rect);                                               //将编辑框移动到单元格
    edit.SetWindowText(str);                                             //设置编辑框显示文本
    edit.ShowWindow(SW_SHOW);                                            //显示编辑框
    edit.SetSel(0,100);                                                  //选中文本
    edit.SetFocus();                                                     //设置焦点
    edit.SetReadOnly(!showedit);                                         //设置是否可编辑
    UpdateWindow();                                                      //更新窗口
}
```

注意

在 ShowEdit 方法中，在列表视图控件的指定单元格中显示编辑框。

（4）处理 WM_LBUTTONDOWN 消息，当鼠标左键按下时获得单元格所在的行、列信息。代码如下：

```
void CCustomGrid::OnLButtonDown(UINT nFlags, CPoint point)
{
    CListCtrl::OnLButtonDown(nFlags, point);
    LVHITTESTINFO p;
    p.pt = point;                                                        //获得鼠标位置
    if (SubItemHitTest(&p)!= -1)                                         //获得鼠标所在单元格
    {
        col = p.iSubItem;                                                //获得列
        row = p.iItem;                                                   //获得行
        ShowEdit();                                                      //显示编辑框
    }
}
```

18.6 启动界面的设计

视频讲解：光盘\TM\lx\18\启动界面的设计.mp4

启动界面也称为欢迎界面，即在应用程序启动时一闪而过的窗体界面。启动界面可以为用户提示一定的信息，用户无须对其进行任何操作。

启动界面是应用程序中最先显示给用户的一个界面，主要用于数据加载的延时。在数据加载时显

示一个这样的界面，可以避免用户由于等待时间过长而产生焦虑。图书管理系统的启动界面如图 18.7 所示。

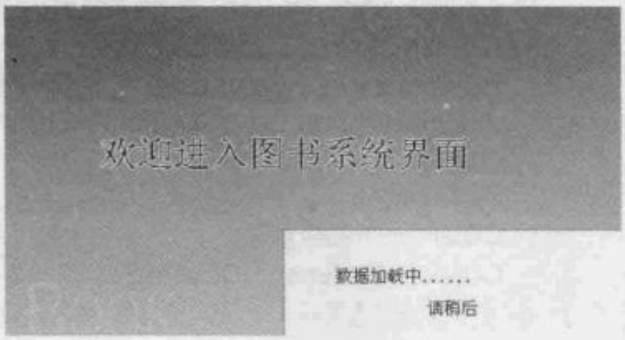


图 18.7　图书管理系统的启动界面

18.6.1　启动界面设计

（1）新建一个对话框资源，打开对话框的属性窗口，选择 Styles 选项卡，将 Border 属性设置为 None。

（2）在工作区窗口中选择 RecourceView 选项卡，导入一个位图资源。

（3）向对话框中添加 1 个图像控件，并通过图像控件显示导入的位图资源。

18.6.2　启动界面的淡入/淡出效果

（1）在对话框的头文件中定义常量，代码如下：

```
#define AW_BLEND                0x00080000                    //淡入/淡出
#define AW_HIDE                 0x00010000                    //隐藏
```

（2）在对话框的 OnInitDialog 函数中设置对话框显示时的淡入/淡出效果，代码如下：

```
BOOL CStartUp::OnInitDialog()
{
    CDialog::OnInitDialog();
    CenterWindow();                                           //对话框处于中心位置
    DWORD dwStyle = AW_BLEND;                                 //设置淡入/淡出风格
    HINSTANCE hInst = LoadLibrary("User32.DLL");              //导入动态链接库 User32.DLL
    typedef BOOL(WINAPI MYFUNC(HWND,DWORD,DWORD));            //函数结构
    MYFUNC* AnimateWindow;                                    //声明函数
    AnimateWindow = (MYFUNC *)::GetProcAddress(hInst,"AnimateWindow");
    AnimateWindow(this->m_hWnd,750,dwStyle);                  //调用函数
    FreeLibrary(hInst);                                       //释放动态链接库
    SetTimer(1,2000,NULL);                                    //设置定时器
return TRUE;
}
```

注意

在设置淡入/淡出效果时需要加载 User32.DLL 动态链接库。

（3）处理 WM_TIMER 消息，在定时器中以淡入/淡出效果关闭启动界面。代码如下：

```
void CStartUp::OnTimer(UINT nIDEvent)
{
    DWORD dwStyle = AW_BLEND;                                 //设置淡入/淡出风格
    HINSTANCE hInst=LoadLibrary("User32.DLL");                //导入动态链接库 User32.DLL
```

```
    typedef BOOL(WINAPI MYFUNC(HWND,DWORD,DWORD));          //函数结构
    MYFUNC* AnimateWindow;                                  //声明函数
    AnimateWindow=(MYFUNC *)::GetProcAddress(hInst,"AnimateWindow");
    AnimateWindow(this->GetSafeHwnd(),750,AW_HIDE | dwStyle);  //调用函数
    FreeLibrary(hInst);                                     //释放动态链接库
    KillTimer(1);                                           //关闭定时器
    CDialog::OnOK();                                        //关闭对话框
    CDialog::OnTimer(nIDEvent);
}
```

18.7 登录对话框设计

视频讲解：光盘\TM\lx\18\登录对话框设计.mp4

本模块使用的数据表：tb_operator

为了防止非法用户进入系统，程序中设计了一个系统登录对话框。在程序启动界面之后显示登录对话框，进行用户身份验证，如果用户输入的用户名和密码不正确，将禁止其进入系统。系统登录对话框如图 18.8 所示。

图 18.8 系统登录对话框

18.7.1 登录对话框的界面设计

（1）新建一个对话框资源，将对话框的 Caption 属性修改为“系统登录”。

（2）向对话框中添加 1 个图像控件、两个编辑框控件和两个按钮控件，控件的属性设置如表 18.7 所示。

表 18.7 登录对话框中控件的属性设置

控件 ID	设 置 属 性	对 应 变 量
IDC_STATIC	Type：Bitmap	无
IDC_EDIT1	无	CEdit m_user
IDC_EDIT2	Password	CEdit m_password
IDC_BUTTON1	Bitmap	CButton m_confirm
IDC_BUTTON2	Bitmap	CButton m_cancel

（3）在工作区窗口中选择 RecourceView 选项卡，导入 3 个位图资源。

18.7.2 设置按钮显示位图

在为按钮控件选择了 Bitmap 属性后，还需要通过代码为控件设置显示的图片资源。代码如下：

```
BOOL CDlgLogin::OnInitDialog()
{
    CDialog::OnInitDialog();
    CStartUp dlg;                                          //声明启动界面模块对象
    dlg.DoModal();                                         //显示启动界面模块
    m_confirm.SetBitmap(LoadBitmap(AfxGetInstanceHandle(),
    MAKEINTRESOURCE(IDB_BITMAP5)));                        //设置位图
    m_cancel.SetBitmap(LoadBitmap(AfxGetInstanceHandle(),
    MAKEINTRESOURCE(IDB_BITMAP6)));                        //设置位图
    return TRUE;
}
```

18.7.3　设置按 Enter 键移动焦点

可以在 PreTranslateMessage 虚方法中设置按 Enter 键移动焦点，代码如下：

```
BOOL CDlgLogin::PreTranslateMessage(MSG* pMsg)
{
    if ((pMsg->message ==WM_KEYDOWN)&&(pMsg->hwnd!= m_confirm.m_hWnd)
        &&(pMsg->hwnd!= m_cancel.m_hWnd))
        if (pMsg->wParam ==13)                             //如果是 Enter 键
            pMsg->wParam = 9;                              //改为 Tab 键
    return CDialog::PreTranslateMessage(pMsg);
}
```

注意

截获键盘键按下消息，并判断当前窗口是否为按钮，如果是则将 Enter 键按下的消息改为 Tab 键，实现按 Enter 键移动焦点的功能。

18.7.4　设置“登录”按钮功能

处理“登录”按钮的单击事件，在该事件的处理函数中判断登录用户名和密码是否正确，如果正确则登录，否则弹出错误提示。代码如下：

```
void CDlgLogin::OnButton1()
{
    CString c_user,c_password;                             //声明字符串
    m_user.GetWindowText(c_user);                          //获得输入的用户名
    m_password.GetWindowText(c_password);                  //获得输入的密码
    if (c_user.IsEmpty() || c_password.IsEmpty())          //用户名和密码为空时
    {
        MessageBox("用户名称或密码不能为空", "用户登录信息");    //弹出错误提示
        return;
    }
```

```
    CString sql;                                                    //声明字符串
    sql.Format("select * from tb_operator where name = '%s' and \
        password = '%s'",c_user,c_password);                        //设置字符串
    m_pRs->Open((_variant_t)sql,m_pCon.GetInterfacePtr(),adOpenKeyset,
        adLockOptimistic,adCmdText);                                //打开记录集
    if (m_pRs->RecordCount>0)                                       //用户名和密码正确时
    {
        Flag = true;                                                //设置登录
        user = m_pRs->GetCollect("name").bstrVal;                   //获得用户名
        password = m_pRs->GetCollect("password").bstrVal;           //获得密码
        EndDialog(0);
    }
    else                                                            //用户名和密码不正确
    {
        user = "";                                                  //设置用户名为空
        password = "";                                              //设置密码为空
        MessageBox("用户名或密码不正确.","提示",64);                 //弹出错误提示
        return;
    }
}
```

18.8 主窗体设计

视频讲解：光盘\TM\lx\18\主窗体设计.mp4

主窗体是程序操作过程中必不可少的，是人机交互中的重要环节。通过主窗体，用户可以调用系统相关的各子模块，快速实现各个功能。图书管理系统中，主窗体被分为 3 个部分：最上面是系统菜单栏，可以通过菜单调用系统中的所有子窗体；菜单栏下面是工具栏，工具栏以按钮的形式使用户能够方便地调用最常用的子窗体；窗体中间的区域是一个和程序主题相关的背景图片。主窗体运行效果如图 18.9 所示。

18.8.1 菜单设计

（1）在工作区窗口中选择 RecourceView 选项卡，创建一个菜单资源，菜单资源设计如图 18.10 所示。

（2）打开主窗口的属性窗口，在 Menu 组合框中选择 IDR_MENU1，在程序运行时显示菜单。

（3）在主窗口的头文件中声明一个 CMyCoolMenu 类 m_menu 对象。

（4）在 OnInitDialog 函数中关联菜单，代码如下：

```
m_menu.AttachMenu(GetMenu()->GetSafeHmenu(),CSize(16,15));
```

（5）处理 WM_DRAWITEM 消息绘制菜单，代码如下：

```
void CBbbDlg::OnDrawItem(int nIDCtl, LPDRAWITEMSTRUCT lpDrawItemStruct)
{
```

```
    m_menu.DrawItem(lpDrawItemStruct);                    //绘制菜单
    CDialog::OnDrawItem(nIDCtl, lpDrawItemStruct);        //调用基类的方法
}
```

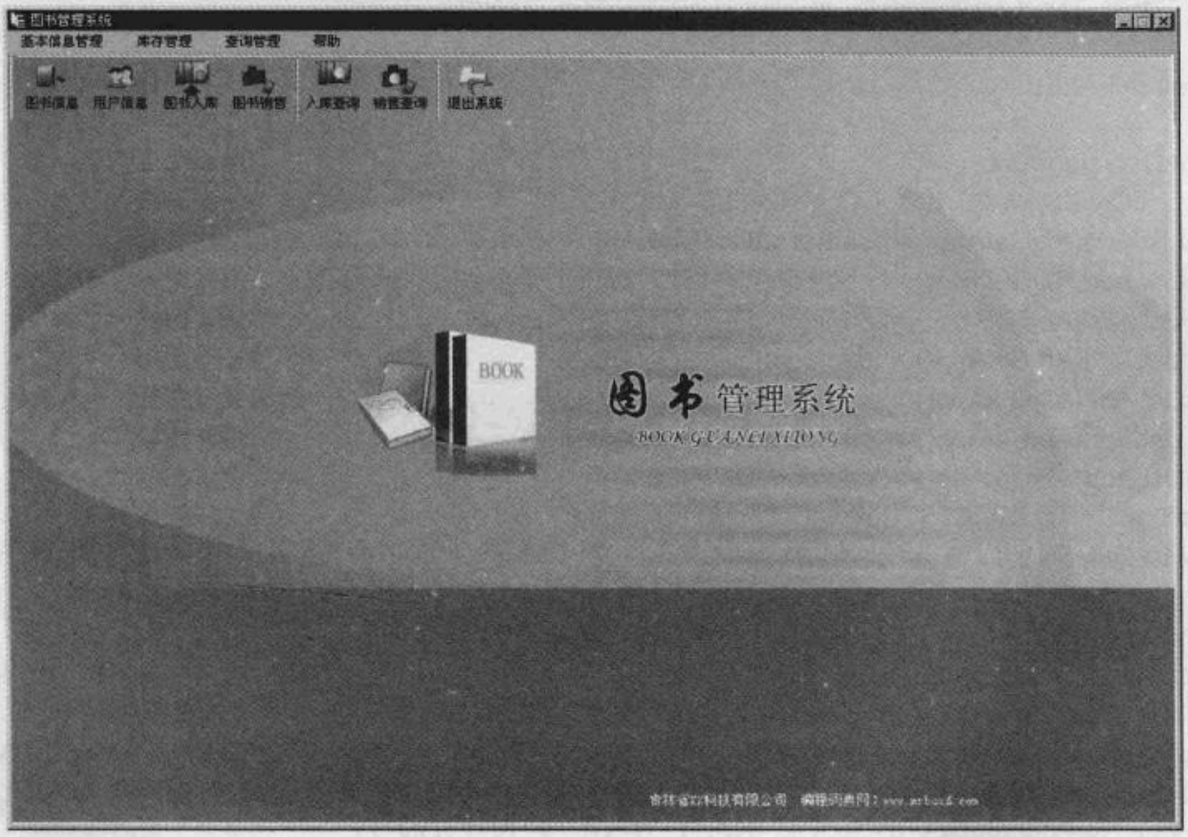


图 18.9　主窗体

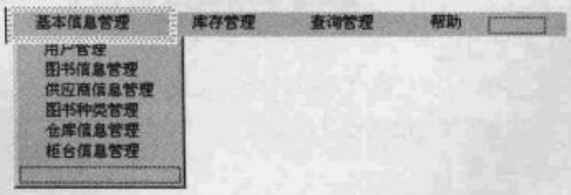


图 18.10　菜单资源

（6）处理 WM_MEASUREITEM 消息，设置菜单项风格。代码如下：

```
void CBbbDlg::OnMeasureItem(int nIDCtl, LPMEASUREITEMSTRUCT lpMeasureItemStruct)
{
    m_menu.MeasureItem(lpMeasureItemStruct);              //设置菜单项风格
    CDialog::OnMeasureItem(nIDCtl, lpMeasureItemStruct);  //调用基类的方法
}
```

18.8.2　工具栏设计

（1）在工作区窗口选择 RecourceView 选项卡，导入 1 个位图和 7 个图标资源。

（2）在主窗口的头文件中声明一个 CToolBar 类对象 m_ToolBar 和一个 CReBar 类对象 m_Rebar。

（3）在主窗口的 OnInitDialog 函数中创建工具栏，代码如下：

```
if(!m_ToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE | CBRS_TOP
    | CBRS_TOOLTIPS |CBRS_FLYBY | CBRS_SIZE_DYNAMIC))          //创建工具栏
{
    return FALSE;
}
CImageList ImageList;                                          //声明图像列表对象
ImageList.Create(32, 32, ILC_COLOR24|ILC_MASK,2,2);            //创建图像列表
ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICON1));               //设置图标
ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICON2));               //设置图标
ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICON3));               //设置图标
ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICON4));               //设置图标
ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICON5));               //设置图标
ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICON6));               //设置图标
ImageList.Add(AfxGetApp()->LoadIcon(IDI_ICON7));               //设置图标
m_ToolBar.GetToolBarCtrl().SetImageList(&ImageList);           //关联图像列表
ImageList.Detach();                                            //释放图像列表
m_ToolBar.SetButtons(NULL, 10);                                //设置工具栏按钮
m_ToolBar.SetButtonInfo(0, ID_BOOK_INFO, TBSTYLE_BUTTON, 0);   //设置工具按钮信息
m_ToolBar.SetButtonText(0, "图书信息");                        //设置工具栏按钮文本
m_ToolBar.SetButtonInfo(1, ID_OPERATOR_M, TBSTYLE_BUTTON , 1); //设置工具按钮信息
m_ToolBar.SetButtonText(1, "用户信息");                        //设置工具栏按钮文本
m_ToolBar.SetButtonInfo(2, ID_SEPARATOR, TBBS_SEPARATOR , 1);  //设置分隔条
m_ToolBar.SetButtonInfo(3, ID_BOOK_INPUT, TBSTYLE_BUTTON, 2);  //设置工具按钮信息
m_ToolBar.SetButtonText(3, "图书入库");                        //设置工具栏按钮文本
m_ToolBar.SetButtonInfo(4, ID_SALE_M, TBSTYLE_BUTTON , 3);     //设置工具按钮信息
m_ToolBar.SetButtonText(4, "图书销售");                        //设置工具栏按钮文本
m_ToolBar.SetButtonInfo(5, ID_SEPARATOR, TBBS_SEPARATOR , 5);  //设置分隔条
m_ToolBar.SetButtonInfo(6, ID_INPUT_QUERY, TBSTYLE_BUTTON , 4); //设置工具按钮信息
m_ToolBar.SetButtonText(6, "入库查询");                        //设置工具栏按钮文本
m_ToolBar.SetButtonInfo(7, ID_SALE_QUERY, TBSTYLE_BUTTON, 5);  //设置工具按钮信息
m_ToolBar.SetButtonText(7, "销售查询");                        //设置工具栏按钮文本
m_ToolBar.SetButtonInfo(8, ID_SEPARATOR, TBBS_SEPARATOR , 6);  //设置分隔条
m_ToolBar.SetButtonInfo(9, IDCANCEL, TBSTYLE_BUTTON, 6);       //设置工具按钮信息
m_ToolBar.SetButtonText(9, "退出系统");                        //设置工具栏按钮文本
m_ToolBar.SetSizes(CSize(60,60), CSize(32,32));                //设置按钮的大小
m_Rebar.Create(this);                                          //创建容器
m_Rebar.AddBar(&m_ToolBar);                                    //设置目标工具栏
m_Rebar.RedrawWindow();                                        //刷新窗口
REBARBANDINFO info;                                            // REBARBANDINFO 结构
info.cbSize = sizeof(info);                                    //设置结构大小
info.fMask = RBBIM_BACKGROUND;                                 //设置标记
m_ToolBar.ModifyStyle(0,TBSTYLE_TRANSPARENT);                  //修改工具栏风格
info.hbmBack = LoadBitmap(AfxGetInstanceHandle(),MAKEINTRESOURCE(IDB_BITMAP3));   //加载图片
m_Rebar.GetReBarCtrl().SetBandInfo(0,&info);                   //访问工具栏
RepositionBars(AFX_IDW_CONTROLBAR_FIRST, AFX_IDW_CONTROLBAR_LAST, 0);       //显示工具栏
```

注意

将工具栏按钮 ID 设置为 ID_SEPARATOR，可以将按钮改为分隔条。

18.8.3　主窗体界面设计

（1）在工作区窗口中选择 RecourceView 选项卡，导入一个位图资源。

（2）添加 1 个图像控件，并使用图像控件显示添加的位图资源。

18.9　“基本信息管理”模块设计

视频讲解：光盘\TM\lx\18\“基本信息管理”模块设计.mp4

本模块使用的数据表：tb_operator、tb_bookkinds、tb_bookinfo、tb_providerinfo、tb_storageinfo、tb_counterinfo

在“基本信息管理”模块中可以设置用户信息、图书信息、供应商信息、图书种类、仓库信息和柜台信息。用户在编辑信息后，可以通过“添加”按钮将编辑的信息保存到数据库中相应的表中；通过“修改”按钮修改数据库中相应表的数据；通过“删除”按钮删除数据库中相应表的数据；通过“退出”按钮退出基本信息管理模块。“基本信息管理”模块如图 18.11 所示。

图 18.11　“基本信息管理”模块

18.9.1　“基本信息管理”模块界面设计

（1）新建一个对话框资源，将对话框资源的 Caption 属性修改为“基本信息管理”。

（2）向对话框中添加 1 个标签控件和 4 个按钮控件。

（3）添加 6 个对话框资源，设置对话框资源的 Style 属性为 Child，去掉 Title bar 属性。

（4）向每个资源中添加相应的控件，控件的添加详见光盘中的程序。

18.9.2 设置选项卡

在“基本信息管理”模块中包含 6 个选项卡（也称为标签页），每个选项卡都对相应的数据表进行操作，这里以“图书信息”选项卡为例进行介绍。

1. 设置列表视图控件

在“图书信息”选项卡的 OnInitDialog 函数中设置列表视图控件的风格和行、列信息，代码如下：

```
BOOL CDlgBookInfo1::OnInitDialog()
{
    CDialog::OnInitDialog();
    m_list.ModifyStyle(LVS_EDITLABELS,0);                          //设置列表可编辑
    m_list.ModifyStyle(0,LVS_REPORT);                              //设置为报表格式
    m_list.ModifyStyle(0,LVS_SHOWSELALWAYS);                       //高亮显示选中项
    m_list.SetExtendedStyle(LVS_EX_FULLROWSELECT                   //允许整行选中
        |LVS_EX_HEADERDRAGDROP                                     //允许整列拖动
        |LVS_EX_GRIDLINES                                          //画出网格线
        |LVS_EX_ONECLICKACTIVATE                                   //单击选中项
        |LVS_EX_FLATSB);                                           //扁平风格显示滚动条
    //添加列
    m_list.InsertColumn(0,"书籍名称");
    m_list.InsertColumn(1,"助记码");
    m_list.InsertColumn(2,"条形码");
    m_list.InsertColumn(3,"作者");
    m_list.InsertColumn(4,"出版社");
    m_list.InsertColumn(5,"价格");
    m_list.InsertColumn(6,"备注");
    m_list.InsertColumn(7,"种类");
    //设置列宽度
    m_list.SetColumnWidth(0,100);
    m_list.SetColumnWidth(1,60);
    m_list.SetColumnWidth(2,80);
    m_list.SetColumnWidth(3,60);
    m_list.SetColumnWidth(4,80);
    m_list.SetColumnWidth(5,60);
    m_list.SetColumnWidth(6,80);
    m_list.SetColumnWidth(7,60);
    AddBookkinds();                                                //加载图书种类
    LoadBookInfo();                                                //加载图书信息
    return TRUE;
}
```

2．加载图书种类

在设置图书信息前需要加载图书种类，通过添加 AddBookkinds 函数来实现。代码如下：

```
void CDlgBookInfo1::AddBookkinds()
{
    m_kinds.ResetContent();                                          //清空组合框中的数据
    m_pRs->raw_Close();                                              //关闭记录集
    CString sql;                                                     //声明字符串
    sql.Format("select * from tb_bookkinds");                        //设置查询语句
    //打开记录集
    m_pRs->Open((_variant_t)sql,m_pCon.GetInterfacePtr(),adOpenKeyset,adLockOptimistic,adCmdText);
    while (! m_pRs->adoEOF)                                          //记录集不为空时循环
    {
        //循环向组合框中插入数据
        m_kinds.AddString((TCHAR*)(_bstr_t)m_pRs->GetFields()->GetItem((long)0)->Value);
        m_pRs->MoveNext();                                           //向下移动记录集指针
    }
}
```

3．判断信息是否为空

在对数据进行操作时，为了避免误操作，需要检查数据信息是否为空，可以通过添加 InfoIsNull 函数来实现。代码如下：

```
bool CDlgBookInfo1::InfoIsNull()
{
    CWnd * temp;
    temp = this;                                                     //获得窗口指针
    CString text;                                                    //声明字符串
    for (int index = 0; index <8;index++)
    {
        temp = this->GetNextDlgTabItem(temp);                        //获得下一个 Tab 键控件句柄
        temp->GetWindowText(text);                                   //获得控件文本
        if (text.IsEmpty())                                          //判断文本是否为空
            return true;
    }
    return false;
}
```

4．加载图书信息

向对话框中添加 LoadBookInfo 函数，用于加载图书信息。代码如下：

```
void CDlgBookInfo1::LoadBookInfo()
{
    m_pRs->raw_Close();                                              //关闭记录集
    CString sql;                                                     //声明字符串
    sql.Format("select * from tb_bookinfo");                         //设置查询语句
    //打开记录集
    m_pRs->Open((_variant_t)sql,m_pCon.GetInterfacePtr(),adOpenKeyset,adLockOptimistic,adCmdText);
```

```
    int row = 0;
    m_list.DeleteAllItems();                                          //清空列表中的数据
    while (!m_pRs->adoEOF)                                            //记录集不为空时循环
    {
        m_list.InsertItem(100,"");                                    //插入行
        for (int col = 0;col <8;col++)                                //循环插入列
        {
            m_list.SetItemText(row,col,(TCHAR*)(_bstr_t)m_pRs->GetFields()->
                GetItem((long)col)->Value);                           //设置列文本
        }
        row += 1;                                                     //设置下一行
        m_pRs->MoveNext();                                            //向下移动记录集指针
    }
}
```

5．添加图书信息

添加 AddBookInfo 函数，用于向数据表中添加图书信息。代码如下：

```
void CDlgBookInfo1::AddBookInfo()
{
    if (InfoIsNull())                                                 //判断数据是否为空
    {
        MessageBox("图书信息不能为空.","提示",MB_OK|MB_ICONINFORMATION);
        return;
    }
    CString c_barcode;                                                //声明字符串
    m_barcode.GetWindowText(c_barcode);                               //获得条形码
    if (BarcodeIsExist((_bstr_t)c_barcode))                           //判断条形码是否存在
    {
        MessageBox("条形码已经存在.","提示",MB_OK|MB_ICONINFORMATION);
        return;
    }
    CString c_name,c_shortname,c_author,c_press,c_price,c_memo,c_kinds;//声明字符串
    //获得编辑框中的数据
    m_bookname.GetWindowText(c_name);
    m_shortcode.GetWindowText(c_shortname);
    m_author.GetWindowText(c_author);
    m_public.GetWindowText(c_press);
    m_price.GetWindowText(c_price);
    m_memo.GetWindowText(c_memo);
    m_kinds.GetWindowText(c_kinds);
    CString sql;                                                      //声明字符串
    sql.Format("Insert into tb_bookinfo values ( '%s','%s','%s','%s',\
        '%s',%f,'%s','%s')",c_name,c_shortname,c_barcode,c_author,
        c_press,atof(c_price),c_memo,c_kinds);                        //设置插入语句
    try
    {
        m_pRs->raw_Close();                                           //关闭记录集
        m_pRs->Open((_variant_t)sql,m_pCon.GetInterfacePtr(),
            adOpenKeyset,adLockOptimistic,adCmdText);                 //打开记录集
```

```
        MessageBox("操作成功.","提示",MB_OK|MB_ICONINFORMATION);//弹出成功提示
        ClearInterface();                                          //清空控件数据
        LoadBookInfo();                                            //加载图书信息
    }
    catch(...)                                                     //捕捉可能出现的错误
    {
        MessageBox("操作失败.","提示",MB_OK|MB_ICONERROR);          //弹出错误提示
    }
}
```

6. 修改图书信息

添加 UpdateBookInfo 函数，用于将修改后的图书信息保存到数据表中。代码如下：

```
void CDlgBookInfo1::UpdateBookInfo()
{
    if (!InfoIsNull())                                             //数据不为空
    {
        if (m_list.GetSelectionMark()!=-1)                         //当前选中的列表项存在
        {
            //确定修改数据
            if (MessageBox("确实要修改数据吗?","提示",MB_YESNO|MB_ICONINFORMATION)==IDYES)
            {
                CString c_barcode;                                 //声明字符串
                //获得条形码
                c_barcode = m_list.GetItemText(m_list.GetSelectionMark(),2);
                CString c_name,c_newcode,c_shortname,c_author;     //声明字符串
                CString c_press,c_price,c_memo,c_kinds;            //声明字符串
                //获得编辑框中的数据
                m_bookname.GetWindowText(c_name);
                m_barcode.GetWindowText(c_newcode);
                m_shortcode.GetWindowText(c_shortname);
                m_author.GetWindowText(c_author);
                m_public.GetWindowText(c_press);
                m_price.GetWindowText(c_price);
                m_memo.GetWindowText(c_memo);
                m_kinds.GetWindowText(c_kinds);
                CString sql;                                       //声明字符串
                sql.Format("update tb_bookinfo set bookname = '%s',shortname \
                    = '%s',barcode = '%s',author = '%s',bookconcern = '%s',\
                    price = %f,memo = '%s',kind = '%s' where barcode = '%s'"
                    ,c_name,c_shortname,c_newcode,c_author,c_press,
                    atof(c_price),c_memo,c_kinds,c_barcode);       //设置修改语句
                m_pRs->raw_Close();                                //关闭记录集
                try
                {
                    m_pRs->Open((_variant_t)sql,m_pCon.GetInterfacePtr(),
                        adOpenKeyset,adLockOptimistic,adCmdText); //打开记录集
                    ClearInterface();                              //清空控件中的数据
                    LoadBookInfo();                                //加载图书信息
```

```
                MessageBox("操作成功.","提示",64);                    //提示成功
            }
            catch(...)                                                //捕捉可能存在的错误
            {
                //提示操作失败
                MessageBox("操作失败.","提示",MB_OK|MB_ICONERROR);
            }
        }
    }
    else                                                              //没有选中列表项时
        //提示选择要修改的列表项
        MessageBox("请选择欲修改的信息.","提示",MB_OK|MB_ICONINFORMATION);
}
else                                                                  //数据为空时
    //提示信息不完整
    MessageBox("基础信息不完整.","提示",MB_OK|MB_ICONINFORMATION);
}
```

7．删除图书信息

添加 DeleteBookInfo 方法，用于删除指定的图书信息。代码如下：

```
void CDlgBookInfo1::DeleteBookInfo()
{
    if (m_list.GetSelectionMark()!=-1)                                //当前选中的列表项存在
    {
        //确定删除记录
        if (MessageBox("确实要删除数据吗?","提示",MB_YESNO|MB_ICONINFORMATION)==IDYES)
        {
            int i = m_list.GetSelectionMark();                        //获得当前选中的列表项索引
            CString c_barcode = m_list.GetItemText(i,2);              //获得条形码
            CString sql;                                              //声明字符串
            sql.Format("delete tb_bookinfo where barcode= '%s'",c_barcode);//设置查询语句
            m_pRs->raw_Close();                                       //关闭记录集
            try
            {
                m_pRs->Open((_variant_t)sql,m_pCon.GetInterfacePtr(),
                    adOpenKeyset,adLockOptimistic,adCmdText);         //打开记录集
                ClearInterface();                                     //清空控件中的数据
                LoadBookInfo();                                       //加载图书信息
                MessageBox("操作成功.","提示",64);                    //提示成功
            }
            catch(...)                                                //捕捉可能出现的错误
            {
                //提示操作失败
                MessageBox("操作失败.","提示",MB_OK|MB_ICONERROR);
            }
        }
    }
    else                                                              //没有选中列表项时
        //提示选择要删除信息
```

```
        MessageBox("请选择欲删除的信息.","提示",MB_OK|MB_ICONINFORMATION);
}
```

18.9.3　初始化标签控件

在“基本信息管理”模块的 OnInitDialog 函数中汇总初始化标签控件，代码如下：

```
BOOL CDlgBaseInfoM::OnInitDialog()
{
    CDialog::OnInitDialog();
    //设置标签控件标题及对应显示的对话框
    m_tab.AddPage("用户信息", &m_dlg1, IDD_OPERATOR1);
    m_tab.AddPage("图书信息", &m_dlg2, IDD_BOOK_INFO1);
    m_tab.AddPage("供应商信息", &m_dlg3, IDD_PROVIDER_INFO1);
    m_tab.AddPage("图书种类", &m_dlg4, IDD_BOOK_KINDS1);
    m_tab.AddPage("仓库信息", &m_dlg5, IDD_STORE_INFO1);
    m_tab.AddPage("柜台信息", &m_dlg6, IDD_DESK_INOF1);
    if (nSelect >= 0)                                    //有标签页被选中
        m_tab.Show(nSelect);                             //设置标签显示选中页
    else                                                 //没有标签页被选中
        m_tab.Show();                                    //标签默认显示页
    return TRUE;
}
```

注意

在对话框初始化时添加标签页，将不同的模块操作集成到同一窗口中。

18.9.4　设置按钮功能

在“基本信息管理”模块中，“添加”、“修改”和“删除”按钮都是根据选中的标签页来调用相应对话框中的添加信息、修改信息和删除信息的方法，这里以“添加”按钮为例进行介绍。代码如下：

```
void CDlgBaseInfoM::OnButAdd()
{
    int cur = m_tab.GetCurSel();                         //获得当前选中标签页索引
    switch (cur)                                         //根据索引调用相应的方法
    {
        case 0:                                          //用户信息
            m_dlg1.AddOperator();                        //添加用户信息
            break;
        case 1:                                          //图书信息
            m_dlg2.AddBookInfo();                        //添加图书信息
            break;
        case 2:                                          //供应商信息
            m_dlg3.AddProvider();                        //添加供应商信息
            break;
```

```
        case 3:                                          //图书种类
            m_dlg4.AddBookKind();                        //添加图书种类信息
            break;
        case 4:                                          //仓库信息
            m_dlg5.AddStorageInfo();                     //添加仓库信息
            break;
        case 5:                                          //柜台信息
            m_dlg6.AddCounterInfo();                     //添加柜台信息
            break;
    }
}
```

18.10 “库存信息管理”模块设计

视频讲解：光盘\TM\lx\18\“库存信息管理”模块设计.mp4

本模块使用的数据表：tb_storageinfo、tb_counterinfo、tb_bookinfo、tb_bookmove_sub、tb_bookstorage、tb_counterbook、tb_fixprice、tb_providerinfo、tb_instock_sub、tb_sell_sub、tb_cancelsell_sub

在“库存信息管理”模块中可以进行图书入库管理、图书定价管理、图书调价管理、入库退货管理、图书调拨管理、图书销售管理和销售退货管理。用户在编辑信息后，可以通过“添加”按钮将编辑的信息保存到数据库中相应的表中；通过“取消”按钮撤销操作；通过“退出”按钮退出“库存信息管理”模块。“库存信息管理”模块如图 18.12 所示。

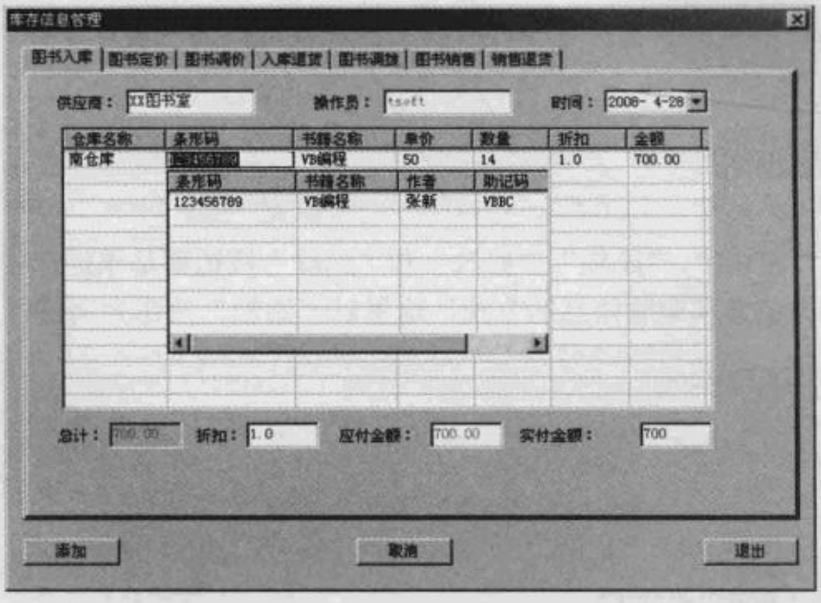


图 18.12 “库存信息管理”模块

18.10.1 “库存信息管理”模块界面设计

（1）新建一个对话框资源，将对话框资源的 Caption 属性修改为“库存信息管理”。

（2）向对话框中添加 1 个标签控件和 3 个按钮控件。

（3）添加 7 个对话框资源，设置对话框资源的 Style 属性为 Child，去掉 Title bar 属性。

（4）向每个资源中添加相应的控件，控件的添加详见光盘中的程序。

18.10.2　设置选项卡

在“库存信息管理”模块中包含 7 个选项卡（也称为标签页），每个选项卡都对相应的数据表进行操作，这里以“图书入库”选项卡为例进行介绍。

1．设置列表视图控件

在“图书入库”选项卡的 OnInitDialog 函数中设置列表视图控件的风格和行、列信息，代码如下：

```
BOOL CDlgBookInput2::OnInitDialog()
{
    CDialog::OnInitDialog();
    LoadProvider();                                                    //添加供应商
    LoadStorage();                                                     //添加仓库信息
    m_storagelist.SetParent(&m_list);
    m_operator.SetWindowText(user);                                    //读取操作员名称
    SetEditColor(RGB(255,0,0));                                        //设置编辑框颜色
    m_auxilist.ModifyStyle(LVS_EDITLABELS,0);                          //禁止编辑标题
    m_auxilist.SetExtendedStyle(LVS_EX_FULLROWSELECT|LVS_EX_ONECLICKACTIVATE
        |LVS_EX_INFOTIP|LVS_EX_SUBITEMIMAGES|LVS_EX_GRIDLINES );       //设置列表风格
    //向表格中添加列
    m_list.InsertColumn(0,"仓库名称",LVCFMT_LEFT,80);
    m_list.InsertColumn(1,"条形码",LVCFMT_LEFT,100);
    m_list.InsertColumn(2,"书籍名称",LVCFMT_LEFT,80);
    m_list.InsertColumn(3,"单价",LVCFMT_LEFT,60);
    m_list.InsertColumn(4,"数量",LVCFMT_LEFT,60);
    m_list.InsertColumn(5,"折扣",LVCFMT_LEFT,60);
    m_list.InsertColumn(6,"金额",LVCFMT_LEFT,60);
    m_auxilist.InsertColumn(0,"条形码",LVCFMT_LEFT,100);
    m_auxilist.InsertColumn(1,"书籍名称",LVCFMT_LEFT,80);
    m_auxilist.InsertColumn(2,"作者",LVCFMT_LEFT,60);
    m_auxilist.InsertColumn(3,"助记码",LVCFMT_LEFT,60);
    m_auxilist.InsertColumn(4,"默认价格",LVCFMT_LEFT,60);
    //添加空行
    m_list.InsertItem(1,"");
    m_list.SetItemText(0,5,"1.0");
    m_auxilist.SetParent(&m_list);
    m_rebate.SetWindowText("1.0");
    m_rebate.IsNumber = true;
    m_factmoney.IsNumber = true;
    m_providerlist.SetParent(&m_list);
    return TRUE;
}
```

2. 加载供应商信息

添加 LoadProvider 方法，该方法用于加载供应商信息。代码如下：

```
void CDlgBookInput2::LoadProvider()
{
    m_providerlist.SetRedraw(FALSE);                                    //不改变外观
    m_providerlist.ResetContent();                                      //删除所有的数据
    CString sql;                                                        //声明字符串
    sql = "select provider from tb_providerinfo";                       //设置查询语句
    m_pRs->raw_Close();                                                 //关闭记录集
    //打开记录集
    m_pRs->Open((_bstr_t)sql,m_pCon.GetInterfacePtr(),adOpenKeyset,adLockOptimistic,adCmdText);
    while (!m_pRs->adoEOF)
    {
        //向列表中添加数据
        m_providerlist.AddString((TCHAR *)(_bstr_t)m_pRs->GetFields()->GetItem("provider")->Value);
        m_pRs->MoveNext();                                              //向下移动记录集指针
    }
    m_providerlist.SetRedraw(TRUE);                                     //重绘控件
    m_providerlist.Invalidate();                                        //刷新窗口
}
```

3. 添加新行

添加 AddNewRow 函数，用于在列表中添加新行。代码如下：

```
void CDlgBookInput2::AddNewRow()
{
    int counts = m_list.GetItemCount();                                 //获得列表项数
        if (m_list.row == counts-1)                                     //当前行为最后一行
        {
            m_list.InsertItem(100,"");                                  //插入新行
            m_list.row+=1;                                              //行加 1
            m_list.SetItemText(m_list.row,5,"1.0");                     //设置显示文本
            m_list.col = 0;                                             //设置为第 0 列
            m_list.ShowEdit();                                          //显示编辑框
        }
        else                                                            //不是最后一行
        {
            m_list.row+=1;                                              //行加 1
            m_list.col = 0;                                             //设置为第 0 列
            m_list.ShowEdit();                                          //显示编辑框
        }
}
```

4. 计算金额

添加 CalculateMoney 函数，该函数用于格式化金额，将字符串转换为浮点型数据。代码如下：

```
float CDlgBookInput2::CalculateMoney()
{
```

```
    float money,temp;                                          //设置浮点型变量
    money = 0.0;                                               //初始化金额
    CString c_money;                                           //声明字符串
    int rowcounts = m_list.GetItemCount();                     //获得列表行数
    for (int i = 0;i<rowcounts;i++)                            //根据列表行数循环
    {
        c_money = m_list.GetItemText(i,6);                     //获得列表项文本
        if (!c_money.IsEmpty())                                //当文本不为空时
        {
            temp = atof(c_money);                              //转为浮点数
            money += temp;
        }
    }
    return money;                                              //返回金额
}
```

5. 设置选择列表

添加 DoEditKeyDown 函数，当用户设置供应商时显示供应商列表。代码如下：

```
void CDlgBookInput2::DoEditKeyDown(UINT nChar, UINT CtrlID)
{
    if ((nChar ==34)&&(CtrlID ==IDC_PROVIDER))                 //按 PageDown 键
    {
        CRect rect,rect1;                                      //声明区域对象
        m_providerlist.GetWindowRect(rect1);                   //获得列表的区域
        m_provider.GetWindowRect(rect);                        //获得编辑框的区域
        ScreenToClient(rect);                                  //屏幕坐标转客户坐标
        m_providerlist.ShowWindow(SW_SHOW);                    //显示提示列表
        m_providerlist.BringWindowToTop();                     //将提示列表置顶
        SetWindowPos(&m_providerlist,rect.left,rect.bottom,rect.right-rect.left,
            rect1.bottom-rect1.top,SWP_SHOWWINDOW);            //移动到指定位置
        m_providerlist.SetSel(0);                              //设置选中项
        m_providerlist.SetFocus();                             //设置焦点
    }
}
```

6. 显示提示列表

添加 EditChange 方法，在输入条形码和书籍名称时显示提示列表，并在用户输入数量后自动计算金额。代码如下：

```
void CDlgBookInput2::EditChange()
{
    CString str;                                               //声明字符串
    m_list.edit.GetWindowText(str);                            //获得编辑框文本
        CString sql = "";                                      //声明字符串
        switch (m_list.col)                                    //获得列表当前列索引
        {
        case 1 :                                               //条形码列
                if (!str.IsEmpty())                            //文本不为空
```

```
            sql.Format("select barcode,bookname,author,shortname,\
            price from tb_bookinfo where barcode like '%s%%' or \
            shortname like '%s%%' ",str,str);                          //设置查询字符串
        break;
case 2 :
        if (!str.IsEmpty())                                            //文本不为空
            sql.Format("select barcode,bookname,author,shortname,\
            price from tb_bookinfo where bookname like '%s%%' or \
            shortname like '%s%%'",str,str);                           //设置查询字符串
        break;
case 3:
        CString c_price,c_num,c_rebate,c_money;                        //声明字符串
        float f_price,f_num,f_rebate,f_money;                          //声明浮点型变量
        c_price = str;                                                 //获得单价
        c_num = m_list.GetItemText(m_list.row,4);                      //获得数量
        c_rebate = m_list.GetItemText(m_list.row,5);                   //获得折扣
        if (c_price.IsEmpty()||c_num.IsEmpty()||c_rebate.IsEmpty())    //如果数据为空
        {
            m_list.SetItemText(m_list.row,6,"");        //如果单价、数量、折扣为空，金额将为空
        }
        else
        {
            f_price = atof(c_price);                                   //格式化单价
            f_num = atof(c_num);                                       //格式化数量
            f_rebate = atof(c_rebate);                                 //格式化折扣
            f_money = f_price*f_num*f_rebate;                          //计算金额
            c_money.Format("%10.2f",f_money);                          //设置金额字符串
            c_money.TrimLeft();                                        //去掉空格
            m_list.SetItemText(m_list.row,6,c_money);                  //设置金额列文本
        }
        str.Format("%10.2f",CalculateMoney());                         //格式化金额
        str.TrimLeft();                                                //去掉空格
        m_summoney.SetWindowText(str);                                 //设置应付金额
        break;
case 4:
        CString c_price,c_num,c_rebate,c_money;                        //声明字符串
        float f_price,f_num,f_rebate,f_money;                          //声明浮点型变量
        c_price = m_list.GetItemText(m_list.row,3);                    //获得单价
        c_num = str;                                                   //获得数量
        c_rebate = m_list.GetItemText(m_list.row,5);                   //获得折扣
        if (c_price.IsEmpty()||c_num.IsEmpty()||c_rebate.IsEmpty())    //如果书籍为空
        {
            m_list.SetItemText(m_list.row,6,"");        //如果单价、数量、折扣为空，金额将为空
        }
        else                                                           //书籍不为空
        {
            f_price = atof(c_price);                                   //格式化单价
            f_num = atof(c_num);                                       //格式化数量
            f_rebate = atof(c_rebate);                                 //格式化折扣
            f_money = f_price*f_num*f_rebate;                          //计算金额
            c_money.Format("%10.2f",f_money);                          //设置金额字符串
```

```
                c_money.TrimLeft();                                    //去掉空格
                m_list.SetItemText(m_list.row,6,c_money);              //设置金额列文本
            }
            str.Format("%10.2f",CalculateMoney());                     //格式化金额
            str.TrimLeft();                                            //去掉空格
            m_summoney.SetWindowText(str);                             //设置应付金额
            break;
case 5:
            CString c_price,c_num,c_rebate,c_money;                    //声明字符串
            float f_price,f_num,f_rebate,f_money;                      //声明浮点型变量
            c_price = m_list.GetItemText(m_list.row,3);                //获得单价
            c_num = m_list.GetItemText(m_list.row,4);                  //获得数量
            c_rebate = str;                                            //获得折扣
            if (c_price.IsEmpty()||c_num.IsEmpty()||c_rebate.IsEmpty())  //如果书籍为空
            {
                m_list.SetItemText(m_list.row,6,"");        //如果单价、数量、折扣为空，金额将为空
            }
            else                                                       //书籍不为空
            {
                f_price = atof(c_price);                               //格式化单价
                f_num = atof(c_num);                                   //格式化数量
                f_rebate = atof(c_rebate);                             //格式化折扣
                f_money = f_price*f_num*f_rebate;                      //计算金额
                c_money.Format("%10.2f",f_money);                      //设置金额字符串
                c_money.TrimLeft();                                    //去掉空格
                m_list.SetItemText(m_list.row,6,c_money);              //设置金额列文本
            }
            str.Format("%10.2f",CalculateMoney());                     //格式化金额
            str.TrimLeft();                                            //去掉空格
            m_summoney.SetWindowText(str);                             //设置应付金额
            break;
default:
    break;
}
if (! sql.IsEmpty())                                                   //查询语句不为空
{
    m_pRs->raw_Close();                                                //关闭记录集
    m_pRs->Open((_bstr_t)sql,m_pCon.GetInterfacePtr(),adOpenKeyset,
        adLockOptimistic,adCmdText);                                   //打开记录集
    if (m_pRs->RecordCount>0)                                          //记录集不为空
    {
        m_auxilist.DeleteAllItems();                                   //删除提示列表中的数据
        int i=0;                                                       //设置初始行
        while(! m_pRs->adoEOF)
        {
            m_auxilist.InsertItem(100,"");                             //插入行
            for (int m = 0;m<5;m++)
                m_auxilist.SetItemText(i,m,(TCHAR*)(_bstr_t)m_pRs->
                    GetFields()->GetItem((long)m)->Value);             //设置列显示文本
            m_pRs->MoveNext();                                         //向下移动记录集指针
            i +=1;                                                     //行加 1
```

```
                }
                ShowListInfo();                                    //显示提示列表
            }
            else                                                   //记录集为空
                m_auxilist.ShowWindow(SW_HIDE);                    //隐藏提示列表
        }
        else                                                       //查询语句为空
            m_auxilist.ShowWindow(SW_HIDE);                        //隐藏提示列表
}
```

7．设置按键消息处理

在 PreTranslateMessage 虚方法中设置鼠标左键和键盘按键在按下时处理的方法，使程序操作起来更加简单、快捷。代码如下：

```
BOOL CDlgBookInput2::PreTranslateMessage(MSG* pMsg)
{
    if (pMsg->message ==WM_KEYDOWN)
    {
        //在“供应商”编辑框获得焦点时，按 PageDown 键显示供应商提示列表
        if ((pMsg->wParam == 34)&&(pMsg->hwnd==m_provider.m_hWnd))
        {
            DoEditKeyDown(34,IDC_PROVIDER);
        }
        //表格中编辑框获得焦点时，按 Enter 键控制单元格焦点移动
        if ((pMsg->wParam == 13)&&(pMsg->hwnd==m_list.edit.m_hWnd))
        {
            MoveFocus();
            return true;
        }
        //表格中获得焦点时，按 Enter 键添加新行
        if ((pMsg->wParam == 13)&&(pMsg->hwnd == m_list.m_hWnd))
        {
            AddNewRow();
            return true;
        }
        //表格中获得焦点时，按 Delete 键删除行
        if ((pMsg->wParam ==VK_DELETE)&&(pMsg->hwnd == m_list.m_hWnd))
        {
            DeleteCurRow();
            return true;
        }
        //列表框获得焦点时，按 Enter 键会将当前数据显示在编辑框中
        else if ((pMsg->hwnd ==m_providerlist.m_hWnd)&&(pMsg->wParam == 13))
        {
            OnDblclkProviderlist();
            return true;
        }
        //辅助录入表格获得焦点时，按 Enter 键会将当前数据显示在表格中
        else if ((pMsg->hwnd ==m_auxilist.m_hWnd)&&(pMsg->wParam == 13))
        {
```

```
            OnDblclkList2(NULL,NULL);
            return true;
        }
        //在“仓库名称”编辑框获得焦点时，按 PageDown 键显示仓库提示列表
        else if ((pMsg->wParam ==13)&&(pMsg->hwnd == m_storagelist.m_hWnd))
        {
            OnDblclkStoragelist();
            return true;
        }
        else if (pMsg->wParam ==13)                                    //如果是 Enter 键
        {
            pMsg->wParam = 9;                                          //改为 Tab 键
            //return;
        }
        else if (pMsg->wParam == VK_ESCAPE)                            //屏蔽 Esc 键
            return true;
    }
    //表格中编辑框按键时的事件，判断是否限制字符输入
    //如果用户按 PageDown 键，将使辅助录入表格获得焦点
    if ((pMsg->message == WM_KEYDOWN)&&(pMsg->hwnd == m_list.edit.m_hWnd))
    {
        LimitEdit();
        if (pMsg->wParam == 34)                                        //按 PageDown 键
        {
            ShowListinfo();                                            //显示供应商提示
            ShowStorage();                                             //显示仓库提示
        }
        else if (pMsg->wParam ==VK_DELETE)                             //按 Delete 键
            DeleteCurRow();                                            //删除表格当前行
    }
    if ((pMsg->message == WM_KEYUP)&&(pMsg->hwnd == m_list.edit.m_hWnd)) //表格中编辑框改变时
    {
        EditChange();
    }
    //在窗体中控件获得焦点时，使提示列表不可见
    if (pMsg->message==WM_LBUTTONDOWN)                                 //鼠标左键按下
    {
        if (pMsg->hwnd!= m_auxilist.m_hWnd)
            m_auxilist.ShowWindow(SW_HIDE);                            //供应商提示不可见
        if (pMsg->hwnd != m_storagelist.m_hWnd)
            m_storagelist.ShowWindow(SW_HIDE);                         //仓库提示不可见
    }
    //在单击表格时，根据当前列判断是否显示编辑框
    if ((pMsg->message ==WM_LBUTTONDOWN)&&(pMsg->hwnd ==m_list.m_hWnd))
    {
        CString tempID,tempname;                                       //声明字符串
        LVHITTESTINFO pos;                                             //声明 LVHITTESTINFO
        pos.pt.x = LOWORD(pMsg->lParam);
        pos.pt.y = HIWORD(pMsg->lParam);
        pos.flags = LVHT_ABOVE;
```

```
        int row,col;                                          //声明整型变量
        row = -1;                                             //初始行
        col = -1;                                             //初始列
        if (m_list.SubItemHitTest(&pos)>=0)                   //当前位置有视图项
        {
            m_list.SetFocus();                                //设置视图焦点
            row = pos.iItem;                                  //设置行
            col = pos.iSubItem;                               //设置列
            tempID = m_list.GetItemText(row,1);               //获得控件 ID
            tempname = m_list.GetItemText(row,2);             //获得文本
            if (col==6)                                       //单击金额列不显示编辑框
                return true;
            m_list.showedit =true;                            //设置显示编辑框
            if ((tempname.IsEmpty()==false)&&(col == 1))
                m_list.showedit =false;                       //不显示编辑框
            else if((tempID.IsEmpty()==false)&&(col == 2))
                m_list.showedit =false;                       //不显示编辑框
        }
    }
    return CDialog::PreTranslateMessage(pMsg);                //调用基类方法
}
```

说明

数据的添加、删除方法可参照 18.9.3 节进行设置。

18.11 “查询管理”模块设计

视频讲解：光盘\TM\lx\18\“查询管理”模块设计.mp4

本模块使用的数据表：tb_cancelinstock_sub、tb_bookinfo、tb_instock_sub、tb_cancelsell_sub、tb_cancelsell_main、tb_sell_sub、tb_sell_main

在“查询管理”模块中可以进行入库查询、入库退货查询、销售查询和销售退货查询。用户在设置查询条件后，可以通过“查询”按钮将数据表中符合条件的记录显示在表中；通过“取消”按钮撤销操作；通过“打印”按钮打印报表；通过“退出”按钮退出“查询管理”模块。“查询管理”模块如图 18.13 所示。

18.11.1 “查询管理”模块界面设计

（1）新建一个对话框资源，将对话框资源的 Caption 属性修改为“查询管理”。

（2）向对话框中添加 1 个标签控件和 4 个按钮控件。

（3）添加 4 个对话框资源，设置对话框资源的 Style 属性为 Child，去掉 Title bar 属性。

（4）向每个资源中添加相应的控件，控件的添加详见光盘中的程序。

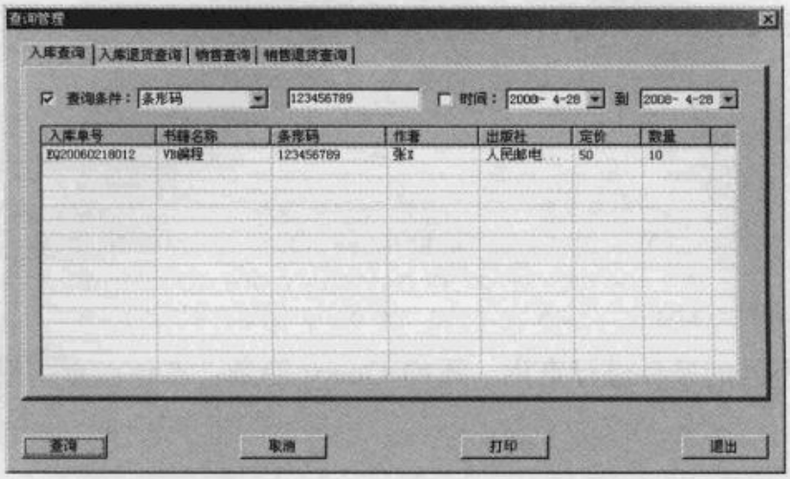


图 18.13　“查询管理”模块

18.11.2　设置选项卡

在“查询管理”模块中包含 4 个选项卡，每个选项卡都对相应的数据表进行操作，这里以“入库查询”选项卡为例进行介绍。

1．设置列表视图控件

在“入库查询”选项卡的 OnInitDialog 函数中设置列表视图控件的风格和行、列信息，代码如下：

```
BOOL CDlgInputQuery3::OnInitDialog()
{
    CDialog::OnInitDialog();
    m_list.SetExtendedStyle(LVS_EX_FULLROWSELECT|LVS_EX_ONECLICKACTIVATE|
        LVS_EX_INFOTIP|LVS_EX_SUBITEMIMAGES|LVS_EX_GRIDLINES ); //设置列表风格
    //向表格中添加列
    m_list.InsertColumn(0,"入库单号");
    m_list.InsertColumn(1,"书籍名称");
    m_list.InsertColumn(2,"条形码");
    m_list.InsertColumn(3,"作者");
    m_list.InsertColumn(4,"出版社");
    m_list.InsertColumn(5,"定价");
    m_list.InsertColumn(6,"数量");
    //设置列宽度
    m_list.SetColumnWidth(0,100);
    m_list.SetColumnWidth(1,100);
    m_list.SetColumnWidth(2,100);
    m_list.SetColumnWidth(3,80);
    m_list.SetColumnWidth(4,80);
    m_list.SetColumnWidth(5,60);
    m_list.SetColumnWidth(6,60);
    //设置字符串
    arrays[1] ="图书名称";
    arrays[2] ="条形码";
    arrays[3] ="作者";
```

```
    arrays[4] ="出版社";
    arrays[5] ="价格";
    arrays[6] ="数量";
    arrays[0] ="入库单号";
    CancelQuery();                                                    //初始化控件信息
    return TRUE;
}
```

2．设置查询功能

在查询时可以根据不同的条件进行查询，添加 Query 函数，在该函数中根据用户设置的条件进行查询。代码如下：

```
void CDlgInputQuery3::Query()
{
    if ((m_check1.GetCheck() ==false)&&(m_check2.GetCheck()==false))      //两个复选框都未选中
    {
        MessageBox("请设置查询条件","提示",64);                           //提示设置查询条件
        return;
    }
    if ( (m_check1.GetCheck()==true)&&(m_check2.GetCheck()==false))       //选中“查询条件”复选框
    {
        CString c_field,c_value;                                          //声明字符串
        m_fields.GetWindowText(c_field);                                  //获得查询字段
        m_value.GetWindowText(c_value);                                   //获得查询值
        if (c_field.IsEmpty()||(c_value.IsEmpty()))                       //如果数据为空
        {
            MessageBox("请设置查询条件","提示",64);                       //弹出提示
            return;
        }
        CString sql;                                                      //声明字符串
        m_list.DeleteAllItems();                                          //清空列表
        switch (m_fields.GetCurSel())                                     //获得组合框选项索引
        {
            case 0:                                                       //按书名查询
            {
                c_field = "bookname";                                     //设置查询字段
                sql.Format(" select a.instockid,b.bookname,b.barcode,\
                    b.author,b.bookconcern,a.unitprice,a.numbers from \
                    tb_instock_sub a inner join tb_bookinfo b on \
                    a.barcode = b.barcode and b.%s = '%s' ",c_field,c_value);//设置查询语句
                break;
            }
            case 1:
            {
                c_field = "barcode";                                      //按条形码查询
                sql.Format(" select a.instockid,b.bookname,b.barcode,\
                    b.author,b.bookconcern,a.unitprice,a.numbers from \
                    tb_instock_sub a inner join tb_bookinfo b on \
                    a.barcode = b.barcode and b.%s = '%s' ",c_field,c_value); //设置查询语句
```

```
                break;
            }
            case 2:
            {
                c_field = "instockid";                                          //按入库单号查询
                sql.Format(" select a.instockid,b.bookname,b.barcode,\
                    b.author,b.bookconcern,a.unitprice,a.numbers from \
                    tb_instock_sub a inner join tb_bookinfo b on \
                    a.barcode = b.barcode and a.%s = '%s' ",c_field,c_value);   //设置查询语句
                break;
            }
            case 3:
            {
                c_field = "Operator" ;                                          //按操作员查询
                sql.Format(" select a.instockid,b.bookname,b.barcode,\
                    b.author,b.bookconcern,a.unitprice,a.numbers from \
                    tb_instock_sub a inner join tb_bookinfo b on \
                    a.barcode = b.barcode inner join tb_instorage_main \
                    c   on a.instockid = c.id and   c.%s = '%s' ",c_field,c_value); //设置查询语句
                break;
            }
            case 4:
            {
                c_field = "provider" ;                                          //按供应商查询
                sql.Format(" select a.instockid,b.bookname,b.barcode,\
                    b.author,b.bookconcern,a.unitprice,a.numbers from \
                    tb_instock_sub a inner join tb_bookinfo b on \
                    a.barcode = b.barcode inner join tb_instorage_main \
                    c   on a.instockid = c.id and   c.%s = '%s' ",c_field,c_value); //设置查询语句
                break;
            }
        }
        m_pRs->raw_Close();                                                     //关闭记录集
        m_pRs->Open((_variant_t)sql,m_pCon.GetInterfacePtr(),
            adOpenKeyset,adLockOptimistic,adCmdText);                           //打开记录集
        int row = 0;                                                            //初始行
        while (! m_pRs->adoEOF)                                                 //记录集不为空循环
        {
            CString temp;                                                       //声明字符串
            m_list.InsertItem(1000,"");                                         //插入行
            for (int i = 0;i<7;i++)                                             //循环列
            {
                temp = (TCHAR *)(_bstr_t)m_pRs->GetFields()->GetItem((long)i)->Value;//获得数据
                m_list.SetItemText(row,i,temp);                                 //设置列显示文本
            }
            m_pRs->MoveNext();                                                  //向下移动记录集指针
            row +=1;                                                            //设置行加 1
        }
    }
    else if ((m_check1.GetCheck()==false)&&(m_check2.GetCheck()==true))         //选中“时间”复选框
```

```
{
    m_list.DeleteAllItems();                                          //清空列表
    CString sql,c_starttime,c_endtime;                                //声明字符串
    m_strtime.GetWindowText(c_starttime);                             //获得起始时间
    m_endtime.GetWindowText(c_endtime);                               //获得结束时间
    sql.Format(" select a.instockid,b.bookname,b.barcode,b.author,\
        b.bookconcern,a.unitprice,a.numbers from tb_instock_sub a \
        inner join tb_bookinfo b on a.barcode = b.barcode inner join \
        tb_instorage_main c on a.instockid = c.id and c.intime between \
        '%s' and '%s' ",c_starttime,c_endtime);                       //设置查询语句
    m_pRs->raw_Close();                                               //关闭记录集
    m_pRs->Open((_variant_t)sql,m_pCon.GetInterfacePtr(),
        adOpenKeyset,adLockOptimistic,adCmdText);                     //打开记录集
    int row = 0;                                                      //初始行
    while (! m_pRs->adoEOF)                                           //记录集不为空
    {
        CString temp;                                                 //声明字符串
        m_list.InsertItem(1000,"");                                   //插入行
        for (int i = 0;i<7;i++)                                       //循环列
        {
            temp = (TCHAR *)(_bstr_t)m_pRs->GetFields()->GetItem((long)i)->Value;//获得数据
            m_list.SetItemText(row,i,temp);                           //设置列显示文本
        }
        m_pRs->MoveNext();                                            //向下移动记录集指针
        row +=1;                                                      //设置行加 1
    }
}
else                                                                  //两个复选框都选中
{
    CString c_field,c_value;                                          //声明字符串
    m_fields.GetWindowText(c_field);                                  //获得查询字段
    m_value.GetWindowText(c_value);                                   //获得查询值
    if (c_field.IsEmpty()||(c_value.IsEmpty()))  {                    //如果数据为空
        MessageBox("请设置查询条件","提示",64);                       //弹出提示
        return;
    }
    CString sql,c_starttime,c_endtime;                                //声明字符串
    m_strtime.GetWindowText(c_starttime);                             //获得起始时间
    m_endtime.GetWindowText(c_endtime);                               //获得结束时间
    if (c_field.IsEmpty()||(c_value.IsEmpty()))                       //数据为空
    {
        MessageBox("请设置查询条件","提示",64);                       //提示设置查询条件
        return;
    }
    m_list.DeleteAllItems();                                          //清空列表数据
    switch (m_fields.GetCurSel())  {                                  //获得组合框选项索引
        case 0:                                                       //按书名和时间查询
        {
            c_field = "bookname";
            sql.Format(" select a.instockid,b.bookname,b.barcode,b.author,\
                b.bookconcern,a.unitprice,a.numbers from tb_instock_sub a \
```

```
                inner join tb_bookinfo b on a.barcode = b.barcode and b.%s \
                = '%s' inner join tb_instorage_main c on a.instockid = c.id \
                and c.intime between '%s' and '%s' ",c_field,c_value,
                c_starttime,c_endtime);                                    //设置查询语句
            break;
        }
        case 1:
        {
            c_field = "barcode";                                           //按条形码和时间查询
            sql.Format(" select a.instockid,b.bookname,b.barcode,b.author,\
                b.bookconcern,a.unitprice,a.numbers from tb_instock_sub \
                a inner join tb_bookinfo b on a.barcode = b.barcode and \
                b.%s = '%s' inner join tb_instorage_main c on a.instockid \
                = c.id and c.intime between '%s' and '%s' ",c_field,
                c_value,c_starttime,c_endtime);                            //设置查询语句
            break;
        }
        case 2:
        {
            c_field = "instockid";                                         //按入库单号查询
            sql.Format(" select a.instockid,b.bookname,b.barcode,b.author,\
                b.bookconcern,a.unitprice,a.numbers from tb_instock_sub \
                a inner join tb_bookinfo b on a.barcode = b.barcode and \
                a.%s = '%s' inner join tb_instorage_main c on a.instockid \
                = c.id and c.intime between '%s' and '%s' ",c_field,
                c_value,c_starttime,c_endtime);                            //设置查询语句
            break;
        }
        case 3:
        {
            c_field = "Operator" ;                                         //按用户和时间查询
            sql.Format(" select a.instockid,b.bookname,b.barcode,b.author,\
                b.bookconcern,a.unitprice,a.numbers from tb_instock_sub \
                a inner join tb_bookinfo b on a.barcode = b.barcode inner \
                join tb_instorage_main c on a.instockid = c.id and c.%s = \
                '%s' and c.intime between '%s' and '%s' ",c_field,
                c_value,c_starttime,c_endtime);                            //设置查询语句
            break;
        }
        case 4:
        {
            c_field = "provider" ;                                         //按供应商和时间查询
            sql.Format(" select a.instockid,b.bookname,b.barcode,b.author,\
                b.bookconcern,a.unitprice,a.numbers from tb_instock_sub \
                a inner join tb_bookinfo b on a.barcode = b.barcode   inner \
                join tb_instorage_main c on a.instockid = c.id and   c.%s = \
                '%s' and   c.intime between '%s' and '%s' ",c_field,
                c_value,c_starttime,c_endtime);                            //设置查询语句
            break;
        }
    }
```

```
        m_pRs->raw_Close();                                                    //关闭记录集
        m_pRs->Open((_variant_t)sql,m_pCon.GetInterfacePtr(),
            adOpenKeyset,adLockOptimistic,adCmdText);                          //打开记录集
        int row = 0;                                                           //初始行
        while (! m_pRs->adoEOF)      {                                         //记录集不为空
            CString temp;                                                      //声明字符串
            m_list.InsertItem(1000,"");                                        //插入行
            for (int i = 0;i<7;i++)                                            //循环列
            {
                temp = (TCHAR *)(_bstr_t)m_pRs->GetFields()->GetItem((long)i)->Value;//获得数据
                m_list.SetItemText(row,i,temp);                                //设置列显示文本
            }
            m_pRs->MoveNext();                                                 //向下移动记录集指针
            row +=1;                                                           //设置行加 1
        }
    }
}
```

18.12 小　　结

本章运用软件工程的设计思想，通过一个完整的图书管理系统为读者详细讲解了一个管理系统的开发流程。通过本章的学习，读者可以了解 Visual C++ 6.0 的应用程序是如何开发的，以及如何通过自定义控件来简化程序，为以后独自开发程序打下坚实的基础。

第19章

C# + SQL Server 实现企业人事管理系统

（视频讲解：93 分钟）

人事管理是现代企业管理工作不可缺少的一部分，是推动企业走向科学化、规范化的必要条件。员工是企业生存的主要元素，员工的增减、变动将直接影响到企业的整体运作。企业员工越多、分工越细、联系越密，所要做的统计工作就越多，人事管理的难度就越大。随着企业的不断壮大，自动化的企业人事管理系统就显得非常必要，本章将通过使用 C# 4.5 + SQL Server 2012 技术开发一个企业人事管理系统。

通过阅读本章，您可以：

- 掌握如何用自定义方法对不同的数据表进行添加、修改的操作
- 掌握如何用自定义方法快速实现多条件的查询
- 掌握如何在数据库中读取或写入图片
- 掌握如何将数据信息以自定义表格的形式插入到 Word 中
- 掌握如何将数据信息导出到 Excel 中
- 掌握如何向 Word 中插入数据库中的图片

19.1 系 统 概 述

视频讲解：光盘\TM\lx\19\系统概述.mp4

基于其他企业人事管理软件的不足，要求能够制作一个可以方便、快捷地对职工信息进行添加、修改、删除的操作，并且可以在数据库中存储相应职工的照片。为了能够更好地存储职工信息，可以将职工信息添加到 Word 文档或者 Excel 表格，这样，不但便于保存，还可以通过 Word 文档或者 Excel 表格进行打印。

19.2 系 统 设 计

视频讲解：光盘\TM\lx\19\系统设计.mp4

19.2.1 系统目标

根据企业对人事管理的要求，制定企业人事管理系统的目标如下。

- ☑ 操作简单方便、界面简洁美观。
- ☑ 在查看员工信息时，可以对当前员工的家庭情况、培训情况进行添加、修改、删除操作。
- ☑ 方便快捷地全方位数据查询。
- ☑ 按照指定的条件对员工进行统计。
- ☑ 可以将员工信息以表格的形式导出到 Word 文档中以便进行打印。
- ☑ 可以将员工信息导出到 Excel 表格中以便进行打印。
- ☑ 灵活的数据备份、还原及清空功能。
- ☑ 由于该系统的使用对象较多，所以要有较好的权限管理。
- ☑ 能够在当前运行的系统中重新进行登录。
- ☑ 系统运行稳定、安全可靠。

19.2.2 系统功能结构

企业人事管理系统的功能结构如图 19.1 所示。

19.2.3 系统业务流程图

企业人事管理系统的业务流程如图 19.2 所示。

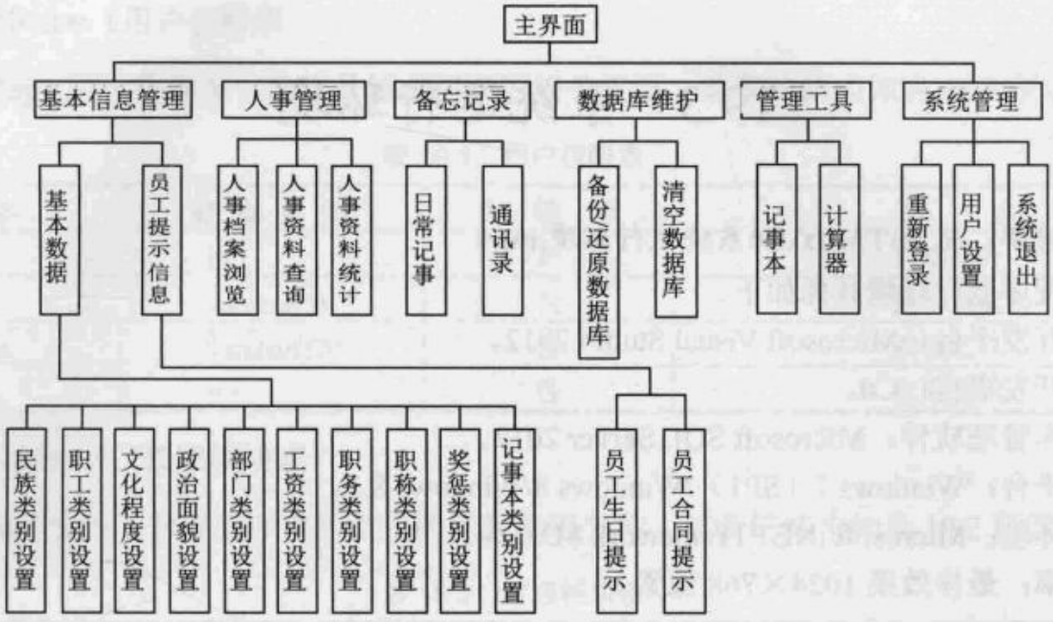


图 19.1　企业人事管理系统功能结构

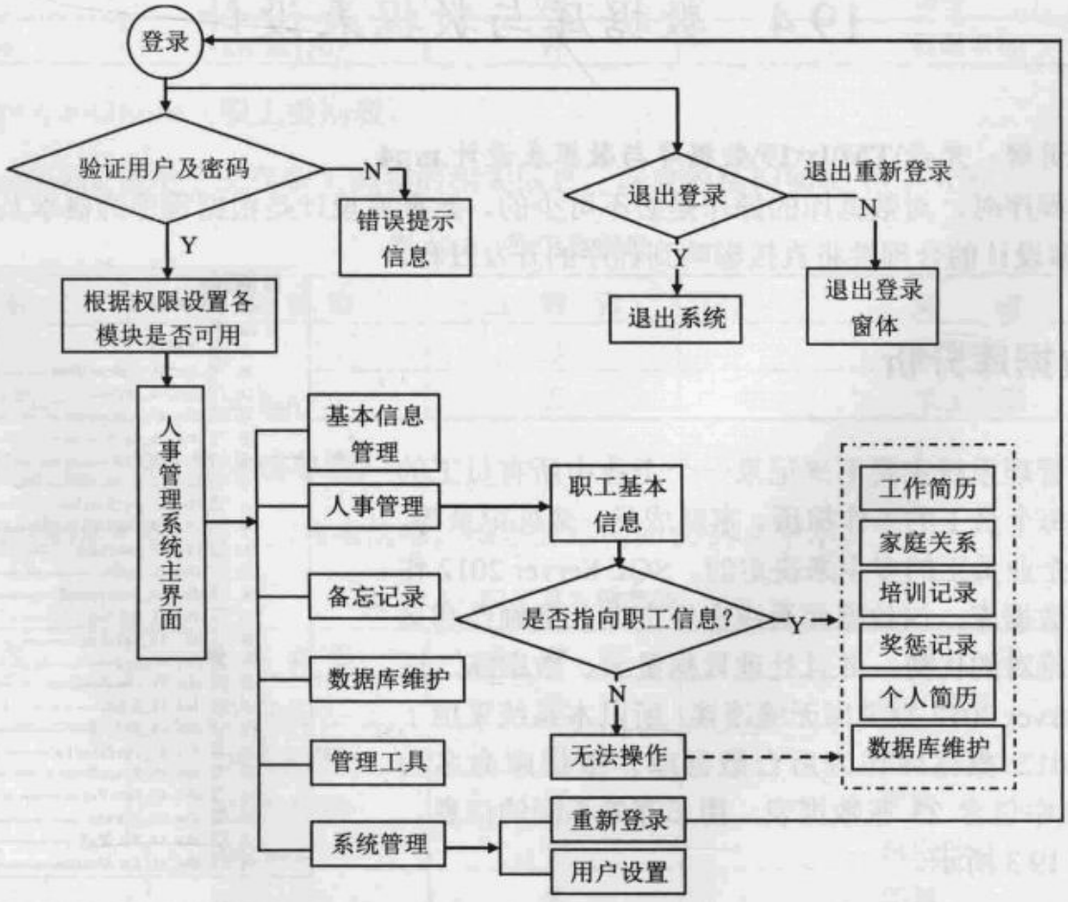


图 19.2　企业人事管理系统的业务流程

注意

在制作项目前，必须根据其实现目标制作业务流程图。

19.3 系统运行环境

视频讲解：光盘\TM\lx\19\系统运行环境.mp4

本系统的程序运行环境具体如下。

- ☑ 系统开发平台：Microsoft Visual Studio 2012。
- ☑ 系统开发语言：C#。
- ☑ 数据库管理软件：Microsoft SQL Server 2012。
- ☑ 运行平台：Windows 7（SP1）/Windows 8/Windows 8.1。
- ☑ 运行环境：Microsoft .NET Framework SDK v4.5。
- ☑ 分辨率：最佳效果 1024×768 像素。

19.4 数据库与数据表设计

视频讲解：光盘\TM\lx\19\数据库与数据表设计.mp4

开发应用程序时，对数据库的操作是必不可少的。数据库设计是根据程序的需求及其实现功能所制定的，数据库设计的合理性将直接影响到程序的开发过程。

19.4.1 数据库分析

企业人事管理系统主要用来记录一个企业中所有员工的基本信息以及每个员工的工作简历、家庭成员、奖惩记录等，数据量是根据企业员工的多少来决定的。SQL Server 2012 作为目前最新的数据库，该数据库系统在安全性、准确性和运行速度方面有绝对的优势，并且处理数据量大、效率高，而且可与 SQL Server 2012 数据库无缝连接，所以本系统采用了 SQL Server 2012 数据库作为后台数据库。数据库命名为 db_PWMS，其中包含 23 张数据表，用于存储不同的信息，详细信息如图 19.3 所示。

图 19.3 企业人事管理系统中用到的数据表

19.4.2 主要数据表结构

由于篇幅有限，所以其他数据表的创建过程不再介绍，相信读者能够举一反三，结合下面给出的数据表结构，其他数据表的创建也不成问题。

1. tb_UserPope（用户权限表）

表 tb_UserPope 用于保存每个操作员使用程序的相关权限，该表的结构如表 19.1 所示。

表 19.1　用户权限表

字 段 名	数 据 类 型	主 键 否	描　述
AutoID	int	是	自动编号
ID	varchar(5)	否	操作员编号
PopeName	varchar(50)	否	权限名称
Pope	int	否	权限标识

2. tb_PopeModel（权限模块表）

tb_PopeModel 表用于保存程序中所涉及的所有权限名称，该表的结构如表 19.2 所示。

表 19.2　权限模块表

字 段 名	数 据 类 型	主 键 否	描　述
ID	int	是	编号
PopeName	varchar(50)	否	权限名称

3. tb_EmployeeGenre（职工类别表）

tb_EmployeeGenre 表用于保存职工类别的相关信息，该表的结构如表 19.3 所示。

表 19.3　职工类别表

字 段 名	数 据 类 型	主 键 否	描　述
ID	int	是	编号
EmployeeName	varchar(20)	否	职工类型

4. tb_Staffbasic（职工基本信息表）

tb_Staffbasic 表用于保存职工的基本信息，该表的结构如表 19.4 所示。

表 19.4　职工基本信息表

字 段 名	数 据 类 型	主 键 否	描　述
ID	varchar(5)	是	职工编号
StaffName	varchar(20)	否	职工姓名
Folk	varchar(20)	否	民族
Birthday	datetime	否	出生日期
Age	int	否	年龄
Culture	varchar(14)	否	文化程度
Marriage	varchar(4)	否	婚姻
Sex	varchar(4)	否	性别
Visage	varchar(14)	否	政治面貌

续表

字 段 名	数 据 类 型	主 键 否	描 述
IDCard	varchar(20)	否	身份证号
Workdate	datetime	否	单位工作时间
WorkLength	int	否	工龄
Employee	varchar(20)	否	职工类型
Business	varchar(10)	否	职务类型
Laborage	varchar(10)	否	工资类别
Branch	varchar(14)	否	部门类别
Duthcall	varchar(14)	否	职称类别
Phone	varchar(14)	否	电话
Handset	varchar(11)	否	手机
School	varchar(24)	否	毕业学校
Speciality	varchar(20)	否	主修专业
GraduateDate	datetime	否	毕业时间
Address	varchar(50)	否	家庭地址
Photo	image	否	个人照片
BeAware	varchar(30)	否	省
City	varchar(30)	否	市
M_Pay	float	否	月工资
Bank	varchar(20)	否	银行账号
Pact_B	datetime	否	合同起始日期
Pact_E	datetime	否	合同结束日期
Pact_Y	float	否	合同年限

5. tb_Family（家庭关系表）

tb_Family 表用于保存家庭关系的相关信息，该表的结构如表 19.5 所示。

表 19.5 家庭关系表

字 段 名	数 据 类 型	主 键 否	描 述
ID	varchar(5)	是	编号
Stu_ID	varchar(5)	否	职工编号
LeagueName	varchar(20)	否	家庭成员名称
Nexus	varchar(10)	否	与本人的关系
BirthDate	datetime	否	出生日期
WorkUnit	varchar(24)	否	工作单位
Business	varchar(10)	否	职务
Visage	varchar(10)	否	政治面貌
phone	varchar(14)	否	电话号码

6. tb_WorkResume（工作简历表）

tb_WorkResume 表用于保存工作简历的相关信息，该表的结构如表 19.6 所示。

表 19.6　工作简历表

字 段 名	数 据 类 型	主 键 否	描　述
ID	varchar(5)	是	编号
Stu_ID	varchar(5)	否	职工编号
BeginDate	datetime	否	开始时间
EndDate	datetime	否	结束时间
WorkUnit	varchar(24)	否	工作单位
Branch	varchar(14)	否	部门
Business	varchar(14)	否	职务

7. tb_RANDP（奖惩表）

tb_RANDP 表用于保存职工奖惩记录的信息，该表的结构如表 19.7 所示。

表 19.7　奖惩表

字 段 名	数 据 类 型	主 键 否	描　述
ID	varchar(5)	是	编号
Stu_ID	varchar(5)	否	职工编号
RPKind	varchar(20)	否	奖惩种类
RPDate	datetime	否	奖惩时间
SealMan	varchar(10)	否	批准人
QuashDate	datetime	否	撤销时间
QuashWhys	varchar(50)	否	撤销原因

8. tb_Individual（个人简历表）

tb_Individual 表用于保存职工个人简历的信息，该表的结构如表 19.8 所示。

表 19.8　个人简历表

字 段 名	数 据 类 型	主 键 否	描　述
ID	varchar(5)	是	编号
Memo	text	否	内容

9. tb_DayWordPad（日常记事本）

tb_DayWordPad 表用于保存人事方面的一些日常事情，该表的结构如表 19.9 所示。

表 19.9　日常记事本表

字 段 名	数 据 类 型	主 键 否	描　述
ID	int	是	编号
BlotterDate	datetime	否	记事时间
BlotterSort	varchar(20)	否	记事类别
Motif	varchar(20)	否	主题
Wordpa	text	否	内容

10. tb_TrainNote（培训记录表）

tb_TrainNote 表用于保存职员培训记录的相关信息，该表的结构如表 19.10 所示。

表 19.10 培训记录表

字 段 名	数 据 类 型	主 键 否	描 述
ID	varchar(5)	是	编号
Stu_ID	varchar(5)	否	职工编号
TrainFashion	varchar(20)	否	培训方式
BeginDate	datetime	否	培训开始时间
EndDate	datetime	否	培训结束时间
Speciality	varchar(20)	否	培训专业
TrainUnit	varchar(30)	否	培训单位
CultureMemo	varchar(50)	否	培训内容
Charge	float	否	费用
Effect	varchar(20)	否	效果

11. tb_AddressBook（通讯录）

tb_AddressBook 表用于保存职员的其他联系信息，该表的结构如表 19.11 所示。

表 19.11 通讯录表

字 段 名	数 据 类 型	主 键 否	描 述
ID	varchar(5)	是	编号
Name	varchar(20)	否	职工姓名
Sex	varchar(4)	否	性别
Phone	varchar(13)	否	家庭电话
QQ	varchar(15)	否	QQ 号
WorkPhone	varchar(13)	否	工作电话
E-mail	varchar(32)	否	邮箱地址
Handset	varchar(11)	否	手机号

说明

由于篇幅有限，这里只列举了重要的数据表的结构，其他的数据表结构可参见光盘中的数据库文件。

19.4.3 数据表逻辑关系

为了使读者能够更好地了解职工基本信息表与其他各表之间的关系，在这里给出数据表关系图，如图 19.4 所示。通过图 19.4 的表关系可以看出，职工基本信息表的一些字段，可以在相关联的表中获取指定的值，并通过职工基本信息表的 ID 值与家庭关系表、培训记录表、奖惩表等建立关系。

为了使读者能够更好地理解登录表与用户权限表、权限模板表之间的关系，下面给出其表间关系

图，如图 19.5 所示。通过图 19.5 可以看出，在用户登录时，可以根据用户 ID 在用户权限表中调用相关的权限。当添加用户时，可以通过权限模板表中的信息，将权限名称自动添加到用户权限表中，以方便在前台中对用户进行添加操作。

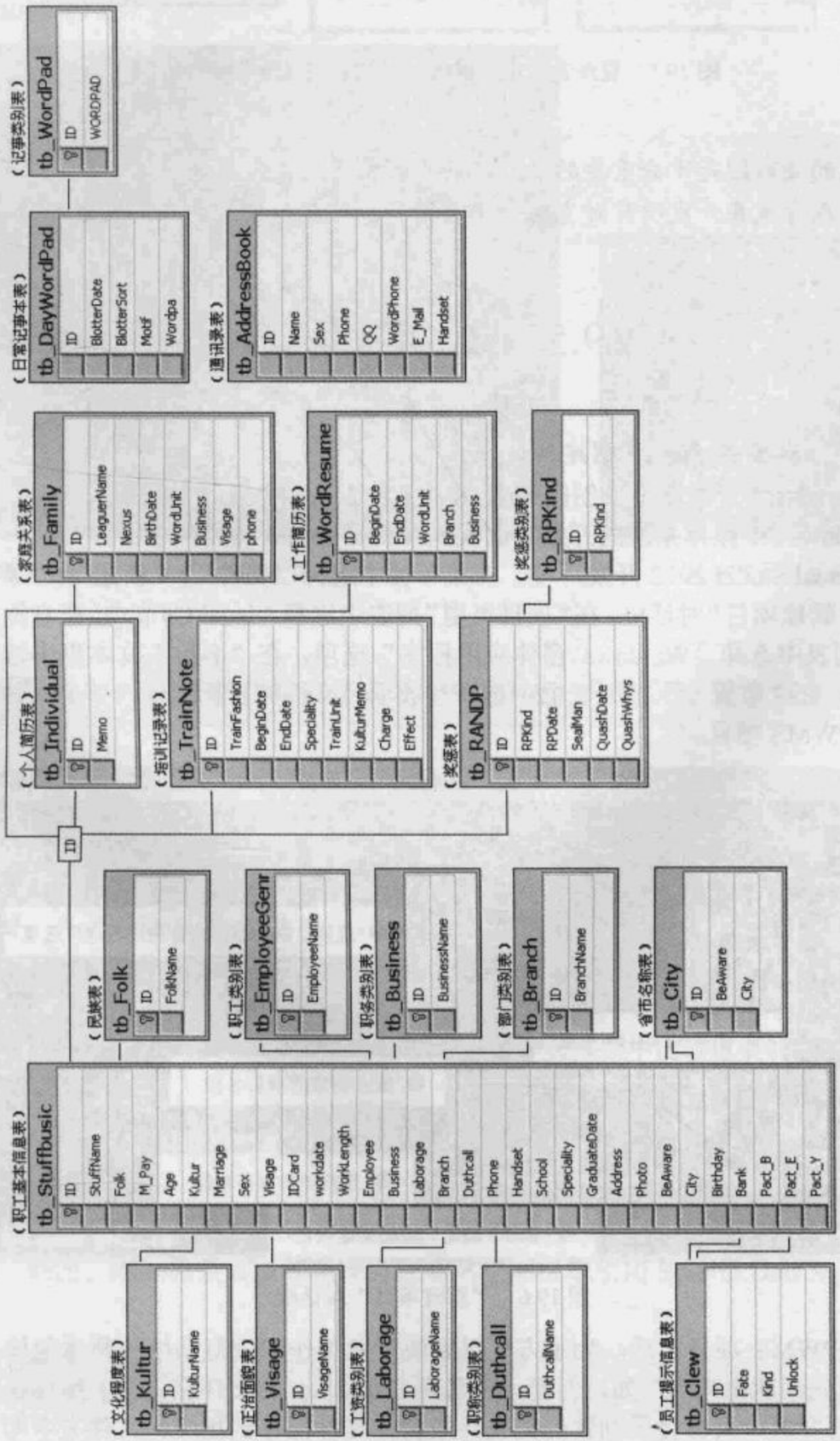


图 19.4　职工基本信息表与各表之间的关系

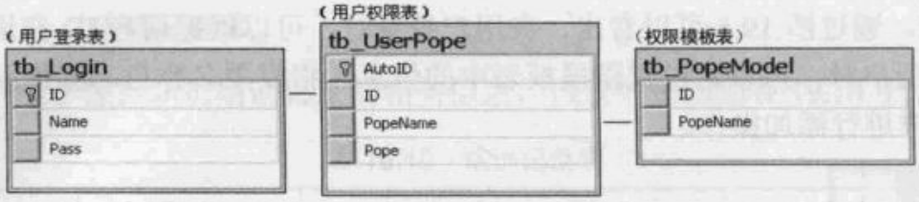


图 19.5　登录表与用户权限表、权限模板表之间的关系

说明

制作数据表的关系图是十分重要的，只有将各表之间的关系制定清楚，才可以通过表关系制作相应的触发器，或者是在开发项目时当对一个表进行操作后，相应的关系表也会随之改变。

19.5　创 建 项 目

视频讲解：光盘\TM\lx\19\创建项目.mp4

在 Visual Studio 2012 开发环境中创建 PWMS 项目的具体步骤如下：

（1）在 Windows 8.1 操作系统的开始界面找到 Visual Studio 2012，单击打开。

（2）打开 Visual Studio 2012 开发环境，在菜单栏中选择“文件”/“新建”/“项目”命令，打开如图 19.6 所示的“新建项目”对话框。在“项目类型”列表中选择 Visual C#节点，在右侧的“Visual Studio 已安装的模板”列表中选择“Windows 窗体应用程序”选项，在“名称”文本框中输入项目名称，这里输入“PWMS”，在“位置”下拉列表框中选择存放项目文件的目标地址，单击“确定”按钮，即可创建一个空白的 PWMS 项目。

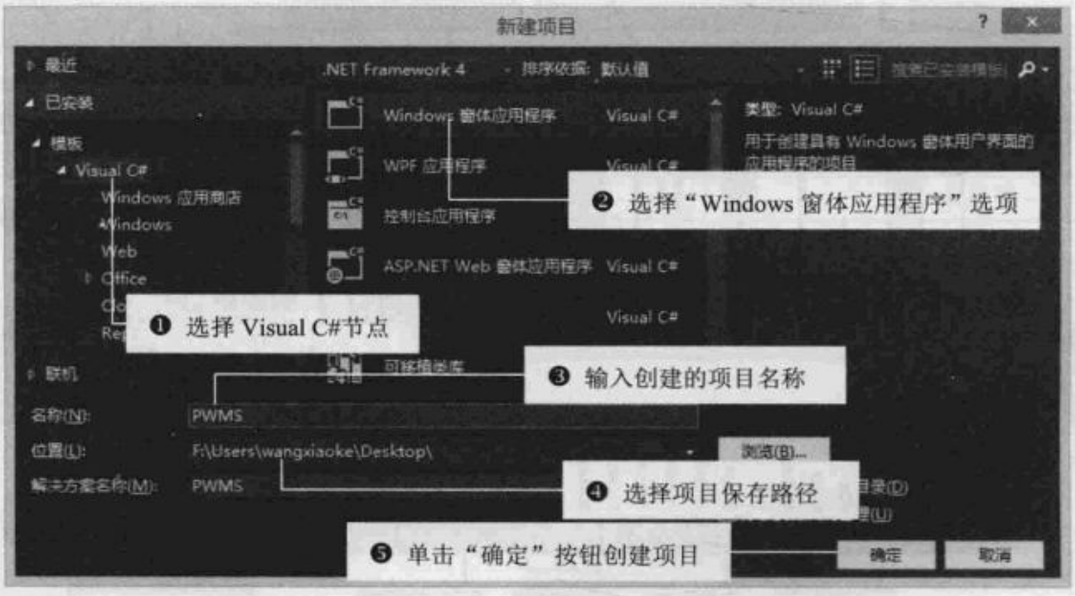


图 19.6　“新建项目”对话框

（3）创建完 PWMS 项目之后，为了方便以后的开发工作和规范系统的整体架构，可以把系统中可能用到的文件夹先创建出来（例如，创建一个名为 DataClass 的文件夹，用于保存程序中用到的数据库文件），这样在开发时，只需将所创建的类文件或窗体文件保存到相应的文件夹中即可。在项目中创

建文件夹非常简单，只需选中当前项目，单击鼠标右键，在弹出的快捷菜单中选择“添加”/“新建文件夹”命令即可，如图 19.7 所示。

（4）按照以上步骤，依次创建企业人事管理系统中可能用到的文件夹，并重命名。下面给出创建完成后的效果，如图 19.8 所示。

图 19.7　选择“添加”/“新建文件夹”命令

图 19.8　文件夹组织结构图

说明

为了使项目结构更加清晰，在项目中创建指定的文件夹，并在文件夹中创建相应的类、窗体等。

19.6　公共类设计

视频讲解：光盘\TM\lx\19\公共类设计.mp4

在开发应用程序时，可以将数据库的相关操作以及对一些控件的设置、遍历等封装在自定义类中，以便于在开发程序时调用，这样可以提高代码的重用性。本系统创建了 MyMeans 和 MyModule 两个公共类，分别存放在 DataClass 和 ModuleClass 文件夹中。下面对这两个公共类中比较重要的方法进行详细讲解。

19.6.1　MyMeans 公共类

该类封装了本系统中所有与数据库连接的方法，可以通过该类的方法与数据库建立连接，并对数据信息进行添加、修改、删除以及读取等操作。在命名空间区域引用 System.Data.SqlClient 命名空间，主要代码如下：

```
using System.Data.SqlClient;
namespace PWMS.DataClass
{
```

```
class MyMeans
{
    #region  全局变量
    public static string Login_ID = "";                          //定义全局变量，记录当前登录的用户编号
    public static string Login_Name = "";                        //定义全局变量，记录当前登录的用户名
    //定义静态全局变量，记录“基础信息”各窗体中的表名、SQL 语句以及要添加和修改的字段名
    public static string Mean_SQL = "", Mean_Table = "", Mean_Field = "";
    //定义一个 SqlConnection 类型的静态公共变量 My_con，用于判断数据库是否连接成功
    public static SqlConnection My_con;
    //定义 SQL Server 2012 连接字符串，用户在使用时，将 Data Source 改为自己的 SQL Server 2012 服务器名
    public static string M_str_sqlcon = "Data Source=XIAOKE;Database=db_PWMS;User
id=sa;PWD=";
    public static int Login_n = 0;                               //用户登录与重新登录的标识
    //存储职工基本信息表中的 SQL 语句
    public static string AllSql = "Select * from tb_Staffbasic";
    #endregion

    …自定义方法，如 getcon、con_close、getcom 等方法
  }
}
```

下面对 MyMeans 类中的自定义方法进行详细介绍。

注意

在项目中连接 SQL Server 2012 数据库是用本机名称，如果在其他计算机上运行该项目，应用本地计算机的名称进行连接。

1. getcon 方法

该方法是用 static 定义的静态方法，其功能是建立与数据库的连接，然后通过 SqlConnection 对象的 Open 方法打开与数据库的连接，并返回 SqlConnection 对象的信息。代码如下：

```
public static SqlConnection getcon()
{
    My_con = new SqlConnection(M_str_sqlcon);                    //用 SqlConnection 对象与指定的数据库相连接
    My_con.Open();                                               //打开数据库连接
    return My_con;                                               //返回 SqlConnection 对象的信息
}
```

2. con_close 方法

该方法的主要功能是对数据库操作后，通过该方法判断是否与数据库连接。如果连接，则关闭数据库连接。代码如下：

```
public void con_close()
{
    if (My_con.State == ConnectionState.Open) {                  //判断是否打开与数据库的连接
        My_con.Close();                                          //关闭数据库的连接
        My_con.Dispose();                                        //释放 My_con 变量的所有空间
    }
}
```

3．getcom 方法

该方法的主要功能是用 SqlDataReader 对象以只读的方式读取数据库中的信息，并以 SqlDataReader 对象进行返回，其中，SQLstr 参数表示传递的 SQL 语句。代码如下：

```
public SqlDataReader getcom(string SQLstr)
{
    getcon();                                                    //打开与数据库的连接
    //创建一个 SqlCommand 对象，用于执行 SQL 语句
    SqlCommand My_com = My_con.CreateCommand();
    My_com.CommandText = SQLstr;                                 //获取指定的 SQL 语句
    SqlDataReader My_read = My_com.mp4cuteReader();              //执行 SQL 语句，生成一个 SqlDataReader 对象
    return My_read;
}
```

4．getsqlcom 方法

该方法的主要功能是通过 SqlCommand 对象执行数据库中的添加、修改和删除的操作，并在执行完后，关闭与数据库的连接，其中，SQLstr 参数表示传递的 SQL 语句。代码如下：

```
public void getsqlcom(string SQLstr)
{
    getcon();                                                    //打开与数据库的连接
    //创建一个 SqlCommand 对象，用于执行 SQL 语句
    SqlCommand SQLcom = new SqlCommand(SQLstr, My_con);
    SQLcom.mp4cuteNonQuery();                                    //执行 SQL 语句
    SQLcom.Dispose();                                            //释放所有空间
    con_close();                                                 //调用 con_close 方法，关闭数据库连接
}
```

5．getDataSet 方法

该方法的主要功能是通过 SqlCommand 对象执行数据库中的添加、修改和删除的操作，并在执行完后，关闭与数据库的连接，其中，SQLstr 参数表示传递的 SQL 语句。代码如下：

```
public DataSet getDataSet(string SQLstr, string tableName)
 {
    getcon();                                                    //打开与数据库的连接
    SqlDataAdapter SQLda = new SqlDataAdapter(SQLstr, My_con);
    DataSet My_DataSet = new DataSet();                          //创建 DataSet 对象
    SQLda.Fill(My_DataSet, tableName);
    con_close();                                                 //关闭数据库的连接
    return My_DataSet;                                           //返回 DataSet 对象的信息
 }
```

说明

为了可以在项目中对不同的数据表进行操作，可以将数据库的连接、断开，数据表的添加、修改、删除、查询用指定的方法进行封装，以便于重复调用。

19.6.2 MyModule 公共类

该类将系统中所有窗体的动态调用以及动态生成添加、修改、删除和查询的 SQL 语句等全部封装到了指定的自定义方法中，以便在开发程序时进行重复调用，这样可以大大简化程序的开发过程。由于该类中应用了可视化组件的基类和对数据库进行操作的相关对象，所以在命名空间区域引用 System.Windows.Forms 和 using System.Data.SqlClient 命名空间。主要代码如下：

```
//以下是添加的命名空间
using System.Windows.Forms;
using System.Data;
using System.Data.SqlClient;
namespace PWMS.ModuleClass
{
    class MyModule
    {
        #region  公共变量
        //声明 MyMeans 类的一个对象，以调用其方法
        DataClass.MyMeans MyDataClass = new PWMS.DataClass.MyMeans();
        public static string ADDs = "";                          //用来存储添加或修改的 SQL 语句
        public static string FindValue = "";                     //存储查询条件
        public static string Address_ID = "";                    //存储通讯录添加修改时的 ID 编号
        public static string User_ID = "";                       //存储用户的 ID 编号
        public static string User_Name = "";                     //存储用户名
        #endregion
        …自定义方法，如 Show_Form、TreeMenuF、Part_SaveClass 等方法
    }
}
```

因篇幅有限，下面只对几个比较重要的方法进行介绍。

1. Show_Form 方法

该方法通过 FrmName 参数传递的窗体名称，调用相应的子窗体，因本系统中存在公共窗体，也就是在同一个窗体模块中可以显示不同的窗体，所以用参数 n 来进行标识。调用公共窗体，实际上就是通过不同的 SQL 语句，在显示窗体时以不同的数据进行显示，以实现不同窗体的显示效果。主要代码如下：

```
#region  窗体的调用
/// <summary>
///窗体的调用
/// </summary>
/// <param name="FrmName">调用窗体的 Text 属性值</param>
/// <param name="n">标识</param>
public void Show_Form(string FrmName, int n)
{
    if (n == 1)
    {
```

```
if (FrmName == "人事档案管理")                    //判断当前要打开的窗体
{
    PerForm.F_ManFile FrmManFile = new PWMS.PerForm.F_ManFile();
    FrmManFile.Text = "人事档案管理";             //设置窗体名称
    FrmManFile.ShowDialog();                      //显示窗体
    FrmManFile.Dispose();
}
if (FrmName == "人事资料查询")
{
    PerForm.F_Find FrmFind = new PWMS.PerForm.F_Find();
    FrmFind.Text = "人事资料查询";
    FrmFind.ShowDialog();
    FrmFind.Dispose();
}
if (FrmName == "人事资料统计")
{
    PerForm.F_Stat FrmStat = new PWMS.PerForm.F_Stat();
    FrmStat.Text = "人事资料统计";
    FrmStat.ShowDialog();
    FrmStat.Dispose();
}
if (FrmName == "员工生日提示")
{
    InfoAddForm.F_ClewSet FrmClewSet = new PWMS.InfoAddForm.F_ClewSet();
    FrmClewSet.Text = "员工生日提示";             //设置窗体名称
    //设置窗体的 Tag 属性，用于在打开窗体时判断窗体的显示类型
    FrmClewSet.Tag = 1;
    FrmClewSet.ShowDialog();                      //显示窗体
    FrmClewSet.Dispose();
}
if (FrmName == "员工合同提示")
{
    InfoAddForm.F_ClewSet FrmClewSet = new PWMS.InfoAddForm.F_ClewSet();
    FrmClewSet.Text = "员工合同提示";
    FrmClewSet.Tag = 2;
    FrmClewSet.ShowDialog();
    FrmClewSet.Dispose();
}
if (FrmName == "日常记事")
{
    PerForm.F_WordPad FrmWordPad = new PWMS.PerForm.F_WordPad();
    FrmWordPad.Text = "日常记事";
    FrmWordPad.ShowDialog();
    FrmWordPad.Dispose();
}
if (FrmName == "通讯录")
{
    PerForm.F_AddressList FrmAddressList = new PWMS.PerForm.F_AddressList();
    FrmAddressList.Text = "通讯录";
    FrmAddressList.ShowDialog();
```

```
            FrmAddressList.Dispose();
        }
        if (FrmName == "备份/还原数据库")
        {
            PerForm.F_HaveBack FrmHaveBack = new PWMS.PerForm.F_HaveBack();
            FrmHaveBack.Text = "备份/还原数据库";
            FrmHaveBack.ShowDialog();
            FrmHaveBack.Dispose();
        }
        if (FrmName == "清空数据库")
        {
            PerForm.F_ClearData FrmClearData = new PWMS.PerForm.F_ClearData();
            FrmClearData.Text = "清空数据库";
            FrmClearData.ShowDialog();
            FrmClearData.Dispose();
        }
        if (FrmName == "重新登录")
        {
            F_Login FrmLogin = new F_Login();
            FrmLogin.Tag = 2;
            FrmLogin.ShowDialog();
            FrmLogin.Dispose();
        }
        if (FrmName == "用户设置")
        {
            PerForm.F_User FrmUser = new PWMS.PerForm.F_User();
            FrmUser.Text = "用户设置";
            FrmUser.ShowDialog();
            FrmUser.Dispose();
        }
        if (FrmName == "计算器")
        {
            System.Diagnostics.Process.Start("calc.mp4");
        }
        if (FrmName == "记事本")
        {
            System.Diagnostics.Process.Start("notepad.mp4");
        }
        if (FrmName == "系统帮助")
        {
            System.Diagnostics.Process.Start("readme.doc");
        }
    }
    if (n == 2)
    {
        String FrmStr = "";                                          //记录窗体名称
        if (FrmName == "民族类别设置")                                //判断要打开的窗体
        {
            DataClass.MyMeans.Mean_SQL = "select * from tb_Folk";   //SQL 语句
            DataClass.MyMeans.Mean_Table = "tb_Folk";                //表名
```

```
        DataClass.MyMeans.Mean_Field = "FolkName";                    //添加、修改数据的字段名
        FrmStr = FrmName;
    }
    if (FrmName == "职工类别设置")
    {
        DataClass.MyMeans.Mean_SQL = "select * from tb_EmployeeGenre";
        DataClass.MyMeans.Mean_Table = "tb_EmployeeGenre";
        DataClass.MyMeans.Mean_Field = "EmployeeName";
        FrmStr = FrmName;
    }
    if (FrmName == "文化程度设置")
    {
        DataClass.MyMeans.Mean_SQL = "select * from tb_Culture";
        DataClass.MyMeans.Mean_Table = "tb_Culture";
        DataClass.MyMeans.Mean_Field = "CultureName";
        FrmStr = FrmName;
    }
    if (FrmName == "政治面貌设置")
    {
        DataClass.MyMeans.Mean_SQL = "select * from tb_Visage";
        DataClass.MyMeans.Mean_Table = "tb_Visage";
        DataClass.MyMeans.Mean_Field = "VisageName";
        FrmStr = FrmName;
    }
    if (FrmName == "部门类别设置")
    {
        DataClass.MyMeans.Mean_SQL = "select * from tb_Branch";
        DataClass.MyMeans.Mean_Table = "tb_Branch";
        DataClass.MyMeans.Mean_Field = "BranchName";
        FrmStr = FrmName;
    }
    if (FrmName == "工资类别设置")
    {
        DataClass.MyMeans.Mean_SQL = "select * from tb_Laborage";
        DataClass.MyMeans.Mean_Table = "tb_Laborage";
        DataClass.MyMeans.Mean_Field = "LaborageName";
        FrmStr = FrmName;
    }
    if (FrmName == "职务类别设置")
    {
        DataClass.MyMeans.Mean_SQL = "select * from tb_Business";
        DataClass.MyMeans.Mean_Table = "tb_Business";
        DataClass.MyMeans.Mean_Field = "BusinessName";
        FrmStr = FrmName;
    }
    if (FrmName == "职称类别设置")
    {
        DataClass.MyMeans.Mean_SQL = "select * from tb_Duthcall";
        DataClass.MyMeans.Mean_Table = "tb_Duthcall";
        DataClass.MyMeans.Mean_Field = "DuthcallName";
```

```
            FrmStr = FrmName;
        }
        if (FrmName == "奖惩类别设置")
        {
            DataClass.MyMeans.Mean_SQL = "select * from tb_RPKind";
            DataClass.MyMeans.Mean_Table = "tb_RPKind";
            DataClass.MyMeans.Mean_Field = "RPKind";
            FrmStr = FrmName;
        }
        if (FrmName == "记事本类别设置")
        {
            DataClass.MyMeans.Mean_SQL = "select * from tb_WordPad";
            DataClass.MyMeans.Mean_Table = "tb_WordPad";
            DataClass.MyMeans.Mean_Field = "WordPad";
            FrmStr = FrmName;
        }
        InfoAddForm.F_Basic FrmBasic = new PWMS.InfoAddForm.F_Basic();
        FrmBasic.Text = FrmStr;   //设置窗体名称
        FrmBasic.ShowDialog();   //显示调用的窗体
        FrmBasic.Dispose();
    }
}
#endregion
```

说明

在开发项目时，窗体的调用是必不可少的，可以将窗体的调用过程写在一个方法中，并通过窗体名称显示指定的窗体。

2. GetMenu 方法

该方法的主要功能是将 MenuStrip 菜单中的菜单项按照级别动态添加到 TreeView 控件的相应节点中，其中，treeV 参数表示要添加节点的 TreeView 控件，MenuS 参数表示要获取信息的 MenuStrip 菜单。主要代码如下：

```
#region  将 StatusStrip 控件中的信息添加到 treeView 控件中
/// <summary>
///读取菜单中的信息
/// </summary>
/// <param name="treeV">TreeView 控件</param>
/// <param name="MenuS">MenuStrip 控件</param>
public void GetMenu(TreeView treeV, MenuStrip MenuS)
{
    //遍历 MenuStrip 组件中的一级菜单项
    for (int i = 0; i < MenuS.Items.Count; i++)
    {
        //将一级菜单项的名称添加到 TreeView 组件的根节点中，并设置当前节点的子节点 newNode1
        TreeNode newNode1 = treeV.Nodes.Add(MenuS.Items[i].Text);
        //将当前菜单项的所有相关信息存入到 ToolStripDropDownItem 对象中
```

```
        ToolStripDropDownItem newmenu = (ToolStripDropDownItem)MenuS.Items[i];
        //判断当前菜单项中是否有二级菜单项
        if (newmenu.HasDropDownItems && newmenu.DropDownItems.Count > 0)
            for (int j = 0; j < newmenu.DropDownItems.Count; j++)                    //遍历二级菜单项
            {
                //将二级菜单名称添加到 TreeView 组件的子节点 newNode1 中，并设置当前节点的子节点
                TreeNode newNode2 = newNode1.Nodes.Add(newmenu.DropDownItems[j].Text);
                //将当前菜单项的所有相关信息存入到 ToolStripDropDownItem 对象中
                ToolStripDropDownItem newmenu2 = (ToolStripDropDownItem)newmenu.DropDownItems[j];
                //判断二级菜单项中是否有三级菜单项
                if (newmenu2.HasDropDownItems && newmenu2.DropDownItems.Count > 0)
                    for (int p = 0; p < newmenu2.DropDownItems.Count; p++)   //遍历三级菜单项
                        //将三级菜单名称添加到 TreeView 组件的子节点 newNode2 中
                        newNode2.Nodes.Add(newmenu2.DropDownItems[p].Text);
            }
    }
}
#endregion
```

说明

在设置节点时，同一级别上的每个节点必须具有唯一的 Value 属性值。

3. Clear_Control 方法

该方法的主要功能是清空可视化控件集中指定控件的文本信息及图片，主要用于在添加数据信息时，对相应文本框进行清空。其中，Con 参数表示可视化控件的控件集合。主要代码如下：

```
#region  遍历清空指定的控件
/// <summary>
///清空所有控件下的所有控件
/// </summary>
/// <param name="Con">可视化控件</param>
public void Clear_Control(Control.ControlCollection Con)
{
    foreach (Control C in Con){                                    //遍历可视化组件中的所有控件
        if (C.GetType().Name == "TextBox")                         //判断是否为 TextBox 控件
            if (((TextBox)C).Visible == true)                      //判断当前控件是否为显示状态
                ((TextBox)C).Clear();                              //清空当前控件
        if (C.GetType().Name == "MaskedTextBox")                   //判断是否为 MaskedTextBox 控件
            if (((MaskedTextBox)C).Visible == true)                //判断当前控件是否为显示状态
                ((MaskedTextBox)C).Clear();                        /清空当前控件
        if (C.GetType().Name == "ComboBox")                        //判断是否为 ComboBox 控件
            if (((ComboBox)C).Visible == true)                     //判断当前控件是否为显示状态
                ((ComboBox)C).Text = "";                           //清空当前控件的 Text 属性值
        if (C.GetType().Name == "PictureBox")                      //判断是否为 PictureBox 控件
            if (((PictureBox)C).Visible == true)                   //判断当前控件是否为显示状态
                ((PictureBox)C).Image = null;                      //清空当前控件的 Image 属性
    }
}
#endregion
```

4. Part_SaveClass 方法

该方法的主要功能是通过部分控件名 BoxName 与 i 值（数字）相结合，在可视化控件集中查找指定的控件，并根据 Sarr 参数中的字段名，组合成添加或修改语句，将生成后的语句存储在公共变量 ADDs 中。主要代码如下：

```
#region  保存添加或修改的信息
/// <summary>
///保存添加或修改的信息
/// </summary>
/// <param name="Sarr">数据表中的所有字段</param>
/// <param name="ID1">第一个字段值</param>
/// <param name="ID2">第二个字段值</param>
/// <param name="Contr">指定控件的数据集</param>
/// <param name="BoxName">要搜索的控件名称</param>
/// <param name="TableName">数据表名称</param>
/// <param name="n">控件的个数</param>
/// <param name="m">标识，用于判断是添加还是修改</param>
public void Part_SaveClass(string Sarr, string ID1, string ID2, Control.ControlCollection Contr, string
BoxName, string TableName, int n, int m)
{
    string tem_Field = "", tem_Value = "";
    int p = 2;
    if (m == 1){                                          //当 m 为 1 时，表示添加数据信息
        if (ID1 != "" && ID2 == ""){                      //根据参数值判断添加的字段
            tem_Field = "ID";
            tem_Value = "'" + ID1 + "'";
            p = 1;
        }
        else{
            tem_Field = "Sta_id,ID";
            tem_Value = "'" + ID1 + "','" + ID2 + "'";
        }
    }
    else
        if (m == 2){                                      //当 m 为 2 时，表示修改数据信息
            if (ID1 != "" && ID2 == ""){                  //根据参数值判断添加的字段
                tem_Value = "ID='" + ID1 + "'";
                p = 1;
            }
            else
                tem_Value = "Sta_ID='" + ID1 + "',ID='" + ID2 + "'";
        }

    if (m > 0){                                           //生成部分添加、修改语句
        string[] Parr = Sarr.Split(Convert.ToChar(','));
        for (int i = p; i < n; i++)
        {
            //通过 BoxName 参数获取要进行操作的控件名称
```

```
            string sID = BoxName + i.ToString();
            foreach (Control C in Contr){                          //遍历控件集中的相关控件
                if (C.GetType().Name == "TextBox" | C.GetType().Name == "MaskedTextBox" |
C.GetType(). Name == "ComboBox")
                    if (C.Name == sID){                            //如果在控件集中找到相应的组件
                        string Ctext = C.Text;
                    if (C.GetType().Name == "MaskedTextBox")       //如果当前是 MaskedTextBox 控件
                        Ctext = Date_Format(C.Text);               //对当前控件的值进行格式化
                    if (m == 1){                                   //组合 SQL 语句中 insert 的相关语句
                        tem_Field = tem_Field + "," + Parr[i];
                        if (Ctext == "")
                            tem_Value = tem_Value + "," + "NULL";
                        else
                            tem_Value = tem_Value + "," + "'" + Ctext + "'";
                    }
                    if (m == 2)
                    {                                              //组合 SQL 语句中 update 的相关语句
                        if (Ctext=="")
                            tem_Value = tem_Value + "," + Parr[i] + "=NULL";
                        else
                            tem_Value = tem_Value + "," + Parr[i] + "='" + Ctext + "'";
                    }
                }
            }
        }
        ADDs = "";
        if (m == 1)                                                //生成 SQL 的添加语句
            ADDs = "insert into " + TableName + " (" + tem_Field + ") values(" + tem_Value + ")";
        if (m == 2)                                                //生成 SQL 的修改语句
            if (ID2 == "")                                         //根据 ID2 参数，判断修改语句的条件
                ADDs = "update " + TableName + " set " + tem_Value + " where ID='" + ID1 + "'";
            else
                ADDs = "update " + TableName + " set " + tem_Value + " where ID='" + ID2 + "'";
    }
}
#endregion
```

Part_SaveClass 方法中的参数说明如表 19.12 所示。

表 19.12　Part_SaveClass 方法中的参数说明

参　数　值	描　　述
Sarr	要添加或修改表的部分字段名称，字段名必须以“,”号分隔
ID1	数据表中的 ID 字段名，在修改表时，可用于条件字段
ID2	数据表中的职工编号字段名，可以为空
Contr	可视化控件集，用于在该控件集中查找控件信息
BoxName	获取控件的部分名称，用于查找相关控件
TableName	要进行添加、修改的数据表名称
n	控件集中要获取控件信息的个数
m	标识，用于判断是生成添加语句；还是修改语句

注意

在 Part_SaveClass 方法中所查找的控件名，必须以 BoxName_i 格式命名（如 Word_1）。

5. Find_Grids 方法

该方法的主要功能是查找指定可视化控件集中控件名包含 TName 参数值的所有控件，并根据控件名称获取相应表的字段名。当查找的控件为 TextBox 时，根据当前控件的部分名称查找相应的 ComboBox 控件（用来记录逻辑运算符），通过 ANDSign 参数将具有相关性的控件组合成查询条件，存入到公共变量 FindValue 中。主要代码如下：

```
#region  组合查询条件
/// <summary>
///根据控件是否为空组合查询条件
/// </summary>
/// <param name="GBox">GroupBox 控件的数据集</param>
/// <param name="TName">获取信息控件的部分名称</param>
/// <param name="ANDSign">查询关系</param>
public void Find_Grids(Control.ControlCollection GBox, string TName, string ANDSign)
{
    string sID = "";                                                    //定义局部变量
    if (FindValue.Length>0)
        FindValue = FindValue + ANDSign;
            foreach (Control C in GBox){                                //遍历控件集上的所有控件
        //判断是否为遍历的控件
        if (C.GetType().Name == "TextBox" | C.GetType().Name == "ComboBox"){
            if (C.GetType().Name == "ComboBox" && C.Text!=""){          //当指定控件不为空时
                sID = C.Name;
                 //当 TName 参数是当前控件名中的部分信息时
                if (sID.IndexOf(TName) > -1){
                    //用 "_" 符号分隔当前控件的名称，获取相应的字段名
                    string[] Astr = sID.Split(Convert.ToChar('_'));
                    //生成查询条件
                    FindValue = FindValue + "(" + Astr[1] + " = '" + C.Text + "')" + ANDSign;
                }
                    }
            //如果当前为 TextBox 控件，并且控件不为空
            if (C.GetType().Name == "TextBox" && C.Text != "")
            {
                sID = C.Name;                                           //获取当前控件的名称
                //判断 TName 参数值是否为当前控件名的子字符串
                if (sID.IndexOf(TName) > -1)
                {
                    string[] Astr = sID.Split(Convert.ToChar('_'));
                    //以 "_" 为分隔符，将控件名存入到一维数组中
                    string m_Sign = "";                                 //用于记录逻辑运算符
                    string mID = "";                                    //用于记录字段名
                    if (Astr.Length > 2)                                //当数组的元素个数大于 2 时
                        mID = Astr[1] + "_" + Astr[2];                  //将最后两个元素组成字段名
                    else
```

```
                    mID = Astr[1];                                    //获取当前条件所对应的字段名称
                foreach (Control C1 in GBox)                          //遍历控件集
                {
                    if (C1.GetType().Name == "ComboBox")              //判断是否为 ComboBox 组件
                    //判断当前组件名是否包含条件组件的部分文件名
                    if ((C1.Name).IndexOf(mID) > -1)
                    {
                        if (C1.Text == "")                            //当查询条件为空时
                            break;                                    //退出本次循环
                        else
                        {
                            m_Sign = C1.Text;                         //将条件值存储到 m_Sgin 变量中
                            break;
                        }
                    }
                }
                if (m_Sign != "")                                     //当该条件不为空时
                    //组合 SQL 语句的查询条件
                    FindValue = FindValue + "(" + mID + m_Sign + C.Text + ")" + ANDSign;
            }
        }
    }
    if (FindValue.Length > 0) //当存储查询条件的变量不为空时，删除逻辑运算符 AND 和 OR
    {
        if (FindValue.IndexOf("AND") > -1)                            //判断是否用 AND 连接条件
            FindValue = FindValue.Substring(0, FindValue.Length - 4);
        if (FindValue.IndexOf("OR") > -1)                             //判断是否用 OR 连接条件
            FindValue = FindValue.Substring(0, FindValue.Length - 3);
    }
    else
        FindValue = "";
}
#endregion
```

Find_Grids 方法中的参数说明如表 19.13 所示。

表 19.13　Find_Grids 方法中的参数说明

参　数　值	描　　述
GBox	用于查找的控件集
TName	获取控件的部分名称，用于查找相关控件
ANDSign	逻辑运算符 AND 或 OR

注意

在 Find_Grids 方法中所查找的条件控件 ComboBox 或 TextBox，必须以“TName+相应字段名”命名（如查找民族类别的控件，其控件名为 Find_Folk，Find_就是传递的 TName 参数值，Folk 为相应表的字段名）。存储逻辑运算符的 ComboBox 控件必须以“相应表的字段名+_Sign”命名（如当 TextBox 控件名为 Find_Age 时，相应的 ComboBox 控件名为 Age_Sign）。这样便于根据控件名称进行组合。

6. GetAutocoding 方法

该方法的主要功能是在添加数据时自动获取添加数据的编号。其实现过程是通过表名和 ID 字段在表中查找最大的 ID 值，并将 ID 值加 1 进行返回。当表中无记录时，返回“0001”。TableName 参数表示进行自动编号的表名，ID 参数表示数据表的编号字段。主要代码如下：

```
#region  自动编号
/// <summary>
/// 在添加信息时自动计算编号
/// </summary>
/// <param name="TableName">表名</param>
/// <param name="ID">字段名</param>
/// <returns>返回 String 对象</returns>
public String GetAutocoding(string TableName, string ID)
{
    //查找指定表中 ID 号为最大的记录
    SqlDataReader MyDR = MyDataClass.getcom("select max(" + ID + ") NID from " + TableName);
    int Num = 0;
    if (MyDR.HasRows)                                    //当查找到记录时
    {
        MyDR.Read();                                     //读取当前记录
        if (MyDR[0].ToString() == "")
            return "0001";
        Num = Convert.ToInt32(MyDR[0].ToString());       //将当前找到的最大编号转换成整数
        ++Num;                                           //最大编号加
        string s = string.Format("{0:0000}", Num);       //将整数值转换成指定格式的字符串
        return s;                                        //返回自动生成的编号
    }
    else
    {
        return "0001";                                   //当数据表没有记录时，返回
    }
}
#endregion
```

7. TreeMenuF 方法

该方法是在单击 TreeView 控件的节点时被调用，其主要功能是通过所选节点的文本名称，在 MenuStrip 控件中进行遍历查找。如果找到，并且为可用状态，则通过 Show_Form 方法动态调用相关的窗体。代码如下：

```
#region  用 TreeView 控件调用 StatusStrip 控件下各菜单的单击事件
/// <summary>
///用 TreeView 控件调用 StatusStrip 控件下各菜单的单击事件
/// </summary>
/// <param name="MenuS">MenuStrip 控件</param>
/// <param name="e">TreeView 控件的 TreeNodeMouseClickEventArgs 类</param>
public void TreeMenuF(MenuStrip MenuS, TreeNodeMouseClickEventArgs e)
{
    string Men = "";
    for (int i = 0; i < MenuS.Items.Count; i++)          //遍历 MenuStrip 控件中的主菜单项
```

```
    {
        Men = ((ToolStripDropDownItem)MenuS.Items[i]).Name;        //获取主菜单项的名称
        //如果 MenuStrip 控件的菜单项没有子菜单
        if (Men.IndexOf("Menu") == -1)
        {
            //当节点名称与菜单项名称相等时
            if (((ToolStripDropDownItem)MenuS.Items[i]).Text == e.Node.Text)
            //判断当前菜单项是否可用
            if (((ToolStripDropDownItem)MenuS.Items[i]).Enabled == false)
            {
                MessageBox.Show("当前用户无权限调用" + "\"" + e.Node.Text + "\"" + "窗体");
                break;
            }
            else
                //调用相应的窗体
                Show_Form(((ToolStripDropDownItem)MenuS.Items[i]).Text.Trim(), 1);
        }
        ToolStripDropDownItem newmenu = (ToolStripDropDownItem)MenuS.Items[i];
        //遍历二级菜单项
        if (newmenu.HasDropDownItems && newmenu.DropDownItems.Count > 0)
            for (int j = 0; j < newmenu.DropDownItems.Count; j++)
            {
                Men = newmenu.DropDownItems[j].Name;                //获取二级菜单项的名称
                if (Men.IndexOf("Menu") == -1)
                {
                    if ((newmenu.DropDownItems[j]).Text == e.Node.Text)
                        if ((newmenu.DropDownItems[j]).Enabled == false)
                        {
                            MessageBox.Show("当前用户无权限调用" + "\"" + e.Node.Text + "\"" + "窗体");
                            break;
                        }
                        else
                            Show_Form((newmenu.DropDownItems[j]).Text.Trim(), 1);
                    }
                    ToolStripDropDownItem newmenu2 = (ToolStripDropDownItem)newmenu.DropDownItems[j];
                    //遍历三级菜单项
                    if (newmenu2.HasDropDownItems && newmenu2.DropDownItems.Count > 0)
                        for (int p = 0; p < newmenu2.DropDownItems.Count; p++)
                        {
                            if ((newmenu2.DropDownItems[p]).Text == e.Node.Text)
                                if ((newmenu2.DropDownItems[p]).Enabled == false)
                                {
                                    MessageBox.Show("当前用户无权限调用" + "\"" + e.Node.Text + "\"" + "窗体");
                                    break;
                                }
                                else
                                    if ((newmenu2.DropDownItems[p]).Text.Trim() == "员工生日提示" ||
(newmenu2.DropDownItems[p]).Text.Trim() == "员工合同提示")
                                        Show_Form((newmenu2.DropDownItems[p]).Text.Trim(), 1);
                                    else
```

```
                                        Show_Form((newmenu2.DropDownItems[p]).Text.Trim(), 2);
                    }
                }
            }
        }
#endregion
```

说明

ToolStripDropDownItem 类主要用于将指定的项装载到下拉容器中，可以通过将 DropDown 属性设置为 ToolStripDropDown，以及设置 ToolStripDropDown 的 Items 属性来执行此操作。可通过 DropDownItems 属性访问已添加的下拉项。

8. MainPope 方法

该方法的主要功能是通过当前登录用户的名称，在权限用户表中查询当前用户的所有权限，并根据权限设置菜单栏中各菜单项的可用状态。其中，MenuS 参数是要设置的菜单栏控件，UName 参数为当前用户的名称。代码如下：

```
#region  根据用户权限设置主窗体菜单
/// <summary>
///根据用户权限设置菜单是否可用
/// </summary>
/// <param name="MenuS">MenuStrip 控件</param>
/// <param name="UName">当前登录用户名</param>
public void MainPope(MenuStrip MenuS, String UName)
{
    string Str = "";
    string MenuName = "";
    DataSet DSet = MyDataClass.getDataSet("select ID from tb_Login where Name='" + UName + "'", "tb_Login");
                                                                    //获取当前登录用户的信息
    string UID = Convert.ToString(DSet.Tables[0].Rows[0][0]);       //获取当前用户编号
    DSet = MyDataClass.getDataSet("select ID,PopeName,Pope from tb_UserPope where ID='" + UID + "'", "tb_UserPope");
                                                                    //获取当前用户的权限信息
    bool bo = false;
    for (int k = 0; k < DSet.Tables[0].Rows.Count; k++)             //遍历当前用户的权限名称
    {
        Str = Convert.ToString(DSet.Tables[0].Rows[k][1]);          //获取权限名称
        if (Convert.ToInt32(DSet.Tables[0].Rows[k][2]) == 1)        //判断权限是否可用
            bo = true;
        else
            bo = false;
        for (int i = 0; i < MenuS.Items.Count; i++)                 //遍历菜单栏中的一级菜单项
        {
            //记录当前菜单项下的所有信息
            ToolStripDropDownItem newmenu = (ToolStripDropDownItem)MenuS.Items[i];
            //如果当前菜单项有子级菜单项
            if (newmenu.HasDropDownItems && newmenu.DropDownItems.Count > 0)
                for (int j = 0; j < newmenu.DropDownItems.Count; j++)   //遍历二级菜单项
```

```
                {
                    MenuName = newmenu.DropDownItems[j].Name;        //获取当前菜单项的名称
                    if (MenuName.IndexOf(Str) > -1)                  //如果包含权限名称
                        newmenu.DropDownItems[j].Enabled = bo;       //根据权限设置可用状态
                        //记录当前菜单项的所有信息
                        ToolStripDropDownItem newmenu2 = (ToolStripDropDownItem)newmenu.
                        DropDownItems[j];
                    //如果当前菜单项有子级菜单项
                    if (newmenu2.HasDropDownItems && newmenu2.DropDownItems.Count > 0)
                        //遍历三级菜单项
                        for (int p = 0; p < newmenu2.DropDownItems.Count; p++)
                        {
                            //获取当前菜单项的名称
                            MenuName = newmenu2.DropDownItems[p].Name;
                            if (MenuName.IndexOf(Str) > -1)          //如果包含权限名称
                            newmenu2.DropDownItems[p].Enabled = bo; //根据权限设置可用状态
                        }
                }
            }
        }
    }
#endregion
```

9. Amend_Pope 方法

该方法的主要功能是修改指定用户的权限，其中，GBox 参数是包含权限复选框的容器控件，TID 参数为当前用户的编号。代码如下：

```
#region  修改指定用户权限
/// <summary>
///修改指定用户的权限
/// </summary>
/// <param name="GBox">GroupBox 控件的数据集</param>
/// <param name="TID">获取用户编号</param>
public void Amend_Pope(Control.ControlCollection GBox, string TID)
{
    string CheckName = "";
    int tt = 0;                                                  //定义一个变量，用来表示是否拥有权限
    foreach (Control C in GBox)                                  //循环查找 GroupBox 包含的控件
    {
        if (C.GetType().Name == "CheckBox")                      //判断控件类型是不是 CheckBox
        {
            if (((CheckBox)C).Checked)                           //判断复选框是否选中
                tt = 1;
            else
                tt = 0;
            CheckName = C.Name;
            string[] Astr = CheckName.Split(Convert.ToChar('_')); //截取复选框的名称，并存放到一个数组中
            //修改用户权限
            MyDataClass.getsqlcom("update tb_UserPope set Pope=" + tt + " where (ID='" + TID + "') and
```

```
(PopeName='" + Astr[1].Trim() + "')");
            }
        }
    }
#endregion
```

说明

(CheckBox) Control 主要是将 Control 强制转换成 CheckBox 类，在强制转换前必须通过 Control 的 GetType()方法的 Name 属性获取其控件类型。如果类型不对，则触发异常。

19.7 登录模块设计

视频讲解：光盘\TM\lx\19\登录模块设计.mp4

本模块使用的数据表：tb_Login

登录模块主要通过输入正确的用户名和密码进入主窗体，它可以提高程序的安全性，保护数据资料不外泄。登录模块运行结果如图 19.9 所示。

图 19.9 系统登录

19.7.1 设计登录窗体

新建一个 Windows 窗体，命名为 F_Login.cs，主要用于实现系统的登录功能，将窗体的 FormBorderStyle 属性设置为 None，以便去掉窗体的标题栏。F_Login 窗体用到的主要控件如表 19.14 所示。

表 19.14 登录窗体用到的主要控件

控件类型	控件 ID	主要属性设置	用途
TextBox	textName	无	输入登录用户名
	textPass	PasswordChar 属性设置为“*”	输入登录用户密码
Button	butLogin	Text 属性设置为“登录”	登录
	butClose	Text 属性设置为“取消”	取消
PictureBox	pictureBox1	SizeMode 属性设置为 StretchImage	显示登录窗体的背景图片

19.7.2 按 Enter 键时移动鼠标焦点

当用户在“用户名”文本框中输入值，并按下 Enter 键时，将鼠标焦点移动到“密码”文本框中。当在“密码”文本框中输入值，并按下 Enter 键时，将鼠标焦点移动到“登录”按钮上。实现代码如下：

```
private void textName_KeyPress(object sender, KeyPressEventArgs e)
{
```

```
    if (e.KeyChar == '\r')                                    //判断是否按下 Enter 键
        textPass.Focus();                                     //将鼠标焦点移动到“密码”文本框
}
private void textPass_KeyPress(object sender, KeyPressEventArgs e)
{
    if (e.KeyChar == '\r')                                    //判断是否按下 Enter 键
        butLogin.Focus();                                     //将鼠标焦点移动到“登录”按钮
}
```

说明

KeyPressEventArgs 指定在用户按键时撰写的字符。例如，当用户按 Shift+A 键时，KeyChar 属性返回一个大写字母 A。

19.7.3　登录功能的实现

当用户输入用户名和密码后，单击“登录”按钮进行登录。在“登录”按钮的 Click 事件中，首先判断用户名和密码是否为空。如果为空，则弹出提示框，通知用户将登录信息填写完整。否则将判断用户名和密码是否正确，如果正确，则进入本系统。详细代码如下：

```
private void butLogin_Click(object sender, EventArgs e)
{
    if (textName.Text != "" & textPass.Text != "")
    {
        //用自定义方法 getcom()在 tb_Login 数据表中查找是否有当前登录用户
        SqlDataReader temDR = MyClass.getcom("select * from tb_Login where Name='" + textName.Text.Trim()
+ "' and Pass='" + textPass.Text.Trim() + "'");
        bool ifcom = temDR.Read();                            //必须用 Read()方法读取数据
        //当有记录时，表示用户名和密码正确
        if (ifcom)
        {
            DataClass.MyMeans.Login_Name = textName.Text.Trim(); //将用户名记录到公共变量中
            DataClass.MyMeans.Login_ID = temDR.GetString(0);  //获取当前操作员编号
            DataClass.MyMeans.My_con.Close();                 //关闭数据库连接
            DataClass.MyMeans.My_con.Dispose();               //释放所有资源
            DataClass.MyMeans.Login_n = (int)(this.Tag);      //记录当前窗体的 Tag 属性值
            this.Close();                                     //关闭当前窗体
        }
        else
        {
            MessageBox.Show("用户名或密码错误！", "提示", MessageBoxButtons.OK, MessageBoxIcon.
Information);
            textName.Text = "";
            textPass.Text = "";
        }
        MyClass.con_close();                                  //关闭数据库连接
    }
```

```
    else
        MessageBox.Show("请将登录信息填写完整！", "提示", MessageBoxButtons.OK, MessageBoxIcon.
Information);
}
```

19.8 系统主窗体设计

视频讲解：光盘\TM\lx\19\系统主窗体设计.mp4

本模块使用的数据表：tb_UserPope

主窗体是程序操作过程中必不可少的，它是人机交互中的重要环节。通过主窗体，用户可以调用系统相关的各子模块，快速掌握本系统中所实现的各个功能。企业人事管理系统中，当登录窗体验证成功后，用户将进入主窗体。主窗体被分为 4 个部分，最上面是系统菜单栏，可以通过它调用系统中的所有子窗体。菜单栏下面是工具栏，它以按钮的形式使用户能够方便地调用最常用的子窗体。窗体的左边是一个树状导航菜单，该导航菜单中的各节点是根据菜单栏中的项自动生成的。窗体的最下面用状态栏显示当前登录的用户名。主窗体运行结果如图 19.10 所示。

图 19.10 企业人事管理系统主窗体

19.8.1 设计菜单栏

菜单栏运行效果如图 19.11 所示。

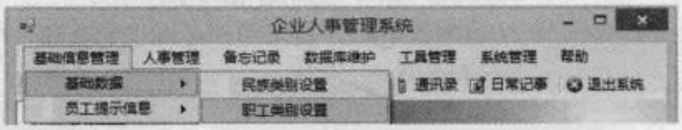


图 19.11 菜单栏运行效果

本系统的菜单栏是通过 MenuStrip 控件实现的，设计菜单栏的具体步骤如下：

（1）从工具箱中拖放一个 MenuStrip 控件置于企业人事管理系统的主窗体中。

（2）为菜单栏中的各个菜单项设置菜单名称，如图 19.12 所示。在输入菜单名称时，系统会自动产生输入下一个菜单名称的提示。

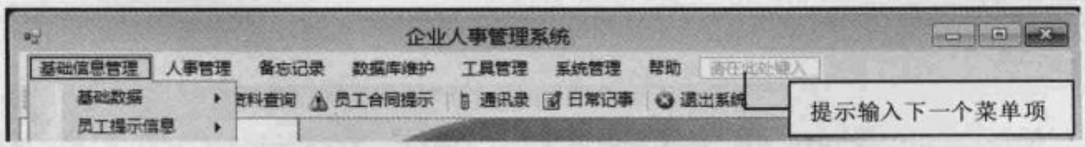


图 19.12 为菜单栏添加项

（3）选中菜单项，单击其“属性”窗口中的 DropDownItems 属性后面的按钮，弹出“项集合编辑器”对话框，如图 19.13 所示。该对话框中可以为菜单项设置 Name 名称，也可以继续通过单击其 DropDownItems 属性后面的按钮添加子项。

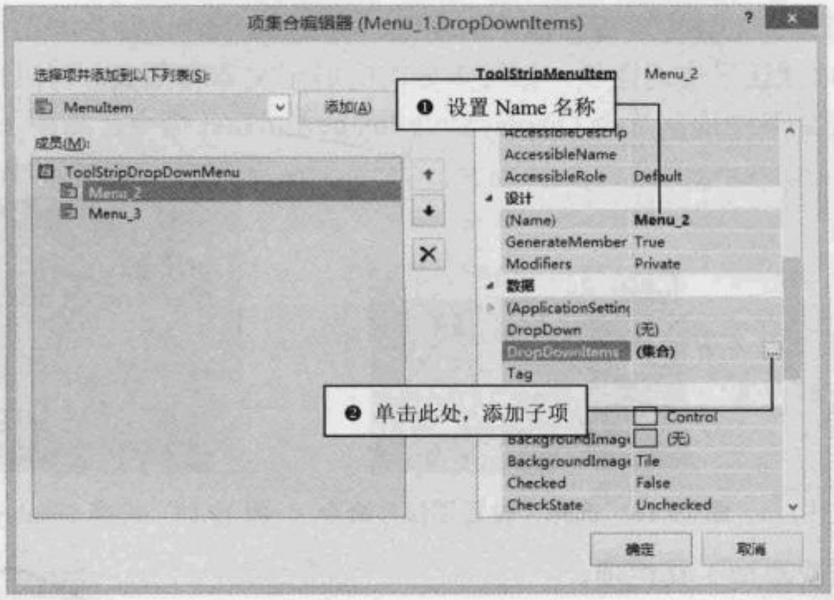


图 19.13　为菜单栏中的项命名并添加子项

菜单栏设计完成之后，单击菜单栏中的各菜单项调用相应的子窗体，为了使程序的制作过程更加简便，将所有子窗体的调用封装到 MyModule 公共类的 Show_Form 方法中，只需要获取当前调用窗体的名称及标识，即可调用相应的窗体。下面以单击“人事管理”/“人事档案管理”菜单项为例进行说明。代码如下：

```
private void Tool_Staffbasic_Click(object sender, EventArgs e)
{
    //用 MyModule 公共类中的 Show_Form()方法调用各窗体
    MyMenu.Show_Form(sender.ToString().Trim(), 1);
}
```

说明

sender.ToString().Trim()表示获取当前对象的 Text 属性值，即当前单击菜单项的文本。如果调用的是“基础信息管理”/“基础数据”下的子菜单项，则把 Show_Form 方法中的 1 改为 2，因为“基础数据”菜单下的所有子菜单项调用的是一个公共窗体。

19.8.2　设计工具栏

工具栏运行效果如图 19.14 所示。

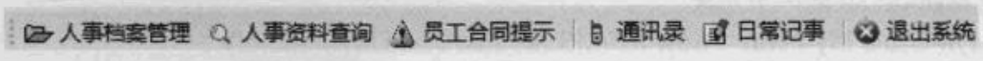


图 19.14　工具栏运行效果

本系统的工具栏是通过 ToolStrip 控件实现的，设计工具栏的具体步骤如下：

（1）从工具箱中拖放一个 ToolStrip 控件置于企业人事管理系统的主窗体中，单击 ToolStrip 控件后面的下拉按钮，可以选择为工具栏添加哪种控件，如图 19.15 所示。

（2）为工具栏添加完控件之后，选中添加的工具栏项，单击鼠标右键，在弹出的快捷菜单中选择

“设置图像”命令，可以为工具栏项设置显示的图像，如图 19.16 所示。

（3）工具栏中的项默认只实现图像，如果需要同时显示文本和图像，可以选中工具栏项，单击鼠标右键，在弹出的快捷菜单中依次选择 DisplayStyle/ImageAndText 命令，如图 19.17 所示。

图 19.15　为工具栏添加控件

图 19.16　选择“设置图像”命令

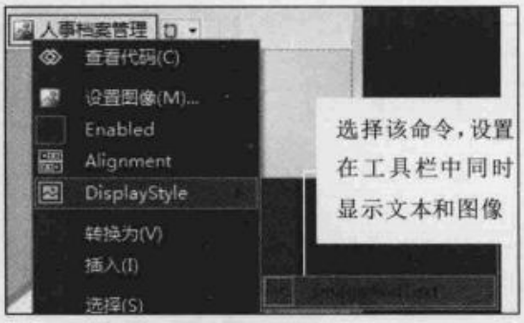


图 19.17　选择 DisplayStyle/ImageAndText 命令

按照以上步骤，依次添加工具栏项。

工具栏主要是为用户提供一种方便的操作系统常用功能的方式，它在实现时主要调用菜单栏中相应菜单项的 Click 事件即可。例如，“人事档案管理”工具栏项的 Click 事件代码如下：

```
private void Button_Staffbasic_Click(object sender, EventArgs e)
{
    if (Tool_Staffbasic.Enabled==true)
        Tool_Staffbasic_Click(sender, e);          //调用人事档案管理菜单项的单击事件
    else
        MessageBox.Show("当前用户无权限调用" + "\"" + ((ToolStripButton)sender).Text + "\"" + "窗体");
}
```

说明

单击（Click）事件中的 sender 参数是事件源，用于引用引发事件的实例。

19.8.3　设计导航菜单

导航菜单运行效果如图 19.18 所示。

本系统的导航菜单是通过 TreeView 控件实现的，导航菜单中的项根据菜单栏自动生成，它主要调用了公共类 MyModule 下的 GetMenu 方法。代码如下：

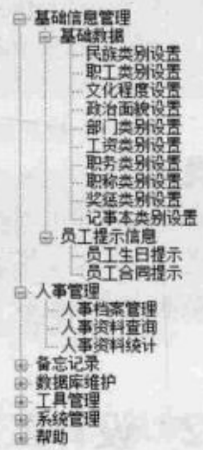


图 19.18　导航菜单运行效果

```
//实例化公共类 MyModule 的一个对象
ModuleClass.MyModule MyMenu = new PWMS.ModuleClass.MyModule();
MyMenu.GetMenu(treeView1, menuStrip1);          //使用菜单栏中的项填充导航菜单
```

当使用树状导航菜单的下拉列表打开相应的子窗体时，可以在 TreeView 控件的节点单击事件（NodeMouseClick）中调用相应的子窗体。代码如下：

```
private void treeView1_NodeMouseClick(object sender, TreeNodeMouseClickEventArgs e)
{
```

```
    if (e.Node.Text.Trim() == "系统退出")          //如果当前节点的文本为“系统退出”
    {
        Application.Exit();                        //关闭应用程序
    }
    MyMenu.TreeMenuF(menuStrip1, e);               //用 MyModule 公共类中的 TreeMenuF()方法调用各窗体
}
```

说明

TreeMenuF 方法是 MyModule 公共类中定义的，用来通过当前节点的文本信息在 menuStrip1 控件中进行遍历查找。如果找到，并且为可用状态，则调用相应窗体；否则弹出“当前用户无权限调用×××窗体”对话框。

19.8.4　设计状态栏

状态栏运行效果如图 19.19 所示。

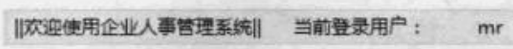


图 19.19　状态栏运行效果

本系统的状态栏是通过 StatusStrip 控件实现的，设计状态栏的具体步骤如下：

（1）从工具箱中拖放一个 StatusStrip 控件置于企业人事管理系统的主窗体中，单击 StatusStrip 控件后面的下拉按钮，可以选择为状态栏添加哪种控件，如图 19.20 所示。

图 19.20　为状态栏添加控件

（2）本系统中的状态栏主要显示欢迎信息和当前登录的用户，因此这里用到 3 个 StatusLabel 控件，其中前两个 StatusLabel 控件的 Text 属性分别设置为“||欢迎使用企业人事管理系统||”和“当前登录用户：”，第 3 个 StatusLabel 控件用来显示当前登录的用户名。状态栏设计完成之后的效果如图 19.21 所示。

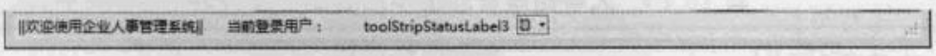


图 19.21　状态栏设计效果

在状态栏中显示当前登录用户名的实现代码如下：

```
statusStrip1.Items[2].Text = DataClass.MyMeans.Login_Name;      //在状态栏显示当前登录的用户名
```

19.9　人事档案管理模块设计

视频讲解：光盘\TM\lx\19\人事档案管理模块设计.mp4

本模块使用的数据表：tb_Folk、tb_Culture、tb_Visage、tb_EmployeeGenre、tb_Business、tb_Laborage、tb_Branch、tb_Duthcall、tb_City、tb_Staffbasic、tb_WorkResume、tb_Family、tb_TrainNote、tb_RANDP、tb_Individual

人事档案管理窗体是用来对职工的基本信息、家庭情况、工作简历、培训记录等进行浏览以及添加、修改、删除的操作。在主窗体中，可以通过菜单栏中的“人事管理”/“人事档案管理”调用人事

档案浏览窗体，也可以通过工具栏中的“人事档案管理”按钮或导航菜单中的下拉列表进行调用。人事档案管理窗体由 4 部分组成，分别由分类查询、“浏览”按钮、职工名称表和信息操作组成。其中，分类查询主要是通过职工的类别，对职工进行简单查询。“浏览”按钮是通过按钮对职工名称表进行浏览。职工名称表用来显示当前所记录的所有职工名称。信息操作是用来对职工的相关信息进行添加、修改、删除、浏览等操作，并可以将职工的基本信息在 Word 文档或者 Excel 表格中以自定义表格的形式进行显示。人事档案管理窗体运行结果如图 19.22 所示。

图 19.22 “人事档案管理”窗体

说明

由于人事档案管理模块中有多个面板，但它们实现的功能大部分是相同的，因此下面以“职工基本信息”面板为例进行讲解。

19.9.1 设计人事档案管理窗体

新建一个 Windows 窗体，命名为 F_MainFile.cs，主要用于对企业的人事档案信息进行管理。F_MainFile 窗体用到的主要控件如表 19.15 所示。

表 19.15 人事档案管理窗体用到的主要控件

控件类型	控件 ID	主要属性设置	用途
abl TextBox	S_0	将其 ReadOnly 属性设置为 True	自动生成职工编号
	S_1	无	输入职工姓名
	S_4	无	输入年龄
	S_9	无	输入身份证号
	S_11	无	输入工龄
	S_25	无	输入月工资
	S_26	无	输入银行账号
	S_29	无	输入合同年限
	S_17	无	输入电话号码
	S_18	无	输入手机号码
	S_19	无	输入毕业学校
	S_20	无	输入主修专业
	S_22	无	输入家庭地址
	textBox1	无	显示当前查看的记录是第几条

续表

控 件 类 型	控件 ID	主要属性设置	用　途
MaskedTextBox	S_3	无	输入职工出生日期
	S_10	无	输入工作时间
	S_27	无	输入合同开始日期
	S_28	无	输入合同结束日期
	S_21	无	输入毕业时间
ComboBox	comboBox1	其 Items 属性设置参见图 19.23	选择查询类型
	comboBox2	无	选择查询条件
	S_2	无	选择民族
	S_7	在其 Items 属性中添加两项，分别为“男”和“女”	选择性别
	S_6	在其 Items 属性中添加两项，分别为“已”和“未”	选择婚姻状态
	S_5	无	选择文化程度
	S_8	无	选择政治面貌
	S_23	无	选择省份
	S_24	无	选择市
	S_14	无	选择工资类别
	S_13	无	选择职务类别
	S_15	无	选择部门类别
	S_16	无	选择职称类别
	S_12	无	选择职工类别
Button	button1	无	查看所有员工信息
	N_First	无	查看第一条记录
	N_Previous	无	查看上一条记录
	N_Next	无	查看下一条记录
	N_Cauda	无	查看最后一条记录
	Img_Save	将其 Enabled 属性设置为 FALSE	选择职工头像
	Img_Clear	将其 Enabled 属性设置为 FALSE	清除职工头像
	Sta_Table	无	将职工信息导出到 Word 文档中
	Sub_Excel	无	将职工信息导出到 Excel 表格中
	Sta_Add	无	清空各文本框及下拉列表，以执行添加操作
	Sta_Amend	无	将“保存”按钮设置为可用以执行修改操作
	Sta_Delete	无	删除选中的职工信息
	Sta_Cancel	将其 Enabled 属性设置为 FALSE	将各按钮的状态恢复到初始化时的状态
	Sta_Save	将其 Enabled 属性设置为 FALSE	执行职工添加或修改操作
OpenFileDialog	openFileDialog1	无	打开选择职工头像的对话框

续表

控 件 类 型	控件 ID	主要属性设置	用　途
PictureBox	S_Photo	将其 SizeMode 属性设置为 StretchImage	显示选择的职工头像
DataGridView	dataGridView1	将其 SelectionMode 属性设置为 FullRowSelect	显示职工编号和姓名信息
TabControl	tabControl1	添加 6 个面板，并分别将其 Text 属性设置为“职工基本信息”、“工作简历”、“家庭关系”、“培训记录”、“奖惩记录”和“个人简历”	显示人事档案管理窗体中的各个控制面板

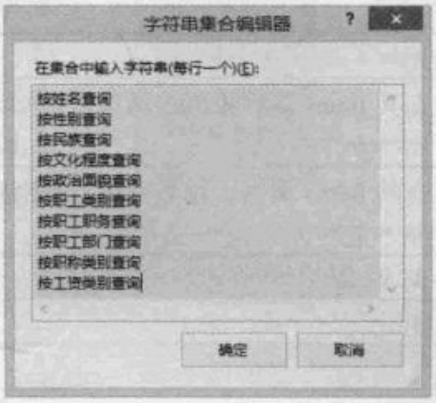


图 19.23　“查询类型”下拉列表 Items 属性设置

19.9.2　添加/修改人事档案信息

单击“添加”按钮，首先调用 MyModule 公共类中的 Clear_Control 方法，将指定控件集下的控件进行清空，然后根据表名和 ID 字段调用 MyModule 公共类中的 GetAutocoding 方法进行自动编号。代码如下：

```
private void Sta_Add_Click(object sender, EventArgs e)
{
    MyMC.Clear_Control(tabControl1.TabPages[0].Controls);      //清空职工基本信息的相应文本框
    S_0.Text = MyMC.GetAutocoding("tb_Staffbasic", "ID");      //自动添加编号
    hold_n = 1;                                                //用于记录添加操作的标识
    MyMC.Ena_Button(Sta_Add, Sta_Amend, Sta_Cancel, Sta_Save, 0, 0, 1, 1);
    groupBox5.Text = "当前正在添加信息";
    Img_Clear.Enabled = true;                                  //使图片选择按钮为可用状态
    Img_Save.Enabled = true;
}
```

单击“修改”按钮，该按钮的功能只是用 hold_n 标识记录当前为修改状态，并修改其他相关按钮的可用状态。代码如下：

```
private void Sta_Amend_Click(object sender, EventArgs e)
{
    hold_n = 2;                                                //用于记录修改操作的标识
    MyMC.Ena_Button(Sta_Add, Sta_Amend, Sta_Cancel, Sta_Save, 0, 0, 1, 1);
```

```
    groupBox5.Text = "当前正在修改信息";
    Img_Clear.Enabled = true;                                    //使图片选择按钮为可用状态
    Img_Save.Enabled = true;
}
```

说明

自定义变量 hold_n 是用于添加和修改操作的标识，如果 hold_n 值不为 1 或 2 时，将不做任何操作。

单击“保存”按钮，根据 hold_n 标识判断执行的是添加操作还是修改操作，并调用“取消”按钮的单击事件功能，将各按钮的状态恢复到初始状态。代码如下：

```
private void Stu_Save_Click(object sender, EventArgs e)
{
    if (tabControl1.SelectedTab.Name == "tabPage6")              //如果当前是“个人简历”选项卡
    {
        //通过 MyMeans 公共类中的 getcom 方法查询当前职工是否添加了个人简历
        SqlDataReader Read_Memo = MyDataClass.getcom("Select * from tb_Individual where ID='" + tem_ID + "'");
        if (Read_Memo.Read())                                    //如果有记录
            //将当前设置的个人简历进行修改
            MyDataClass.getsqlcom("update tb_Individual set Memo='" + Ind_Mome.Text + "' where ID='" +
tem_ID + "'");
        else
            //如果没有记录，则进行添加操作
            MyDataClass.getsqlcom("insert into tb_Individual (ID,Memo) values('" + tem_ID + "','" +
Ind_Mome. Text + "')");
    }
    else //如果当前是“职工基本信息”选项卡
    {
        //定义字符串变量，并存储“职工基本信息表”中的所有字段
        string All_Field = "ID,StuffName,Folk,Birthday,Age,Culture,Marriage,Sex,Visage,IDCard,Workdate, WorkLength,
Employee,Business,Laborage,Branch,Duthcall,Phone,Handset,School,Speciality,GraduateDate,Address,BeAw
are,City,M_Pay, Bank,Pact_B,Pact_E,Pact_Y";
        if (hold_n == 1 || hold_n == 2)                          //判断当前是添加还是修改操作
        {
            ModuleClass.MyModule.ADDs = "";                      //清空 MyModule 公共类中的 ADDs 变量
            //用 MyModule 公共类中的 Part_SaveClass()方法组合添加或修改的 SQL 语句
            MyMC.Part_SaveClass(All_Field, S_0.Text.Trim(), "", tabControl1.TabPages[0].Controls,
"S_", "tb_Staffbasic", 30, hold_n);
            //如果 ADDs 变量不为空，则通过 MyMeans 公共类中的 getsqlcom 方法执行添加、修改操作
            if (ModuleClass.MyModule.ADDs != "")
                MyDataClass.getsqlcom(ModuleClass.MyModule.ADDs);
        }
        if (Ima_n > 0)                                           //如果图片标识大于 0
        {
            //通过 MyModule 公共类中的 SaveImage()方法将图片存入数据库中
            MyMC.SaveImage(S_0.Text.Trim(), imgBytesIn);
        }
```

```
            Sta_Cancel_Click(sender, e);                    //调用“取消”按钮的单击事件
        }
    }
```

在添加和修改人事档案信息时，当为职工选择头像后，需要将选择的头像转换成字节数组，然后再存放到数据库中。将头像转换成字节数组的实现代码如下：

```
#region  将图片转换成字节数组
public void Read_Image(OpenFileDialog openF, PictureBox MyImage)
{
    openF.Filter = "*.jpg|*.jpg|*.bmp|*.bmp";                  //指定 OpenFileDialog 控件打开的文件格式
    if (openF.ShowDialog(this) == DialogResult.OK)              //如果打开了图片文件
    {
        try
        {
            //将图片文件存入到 PictureBox 控件中
            MyImage.Image = System.Drawing.Image.FromFile(openF.FileName);
            string strimg = openF.FileName.ToString();          //记录图片的所在路径
            //将图片以文件流的形式进行保存
            FileStream fs = new FileStream(strimg, FileMode.Open, FileAccess.Read);
            BinaryReader br = new BinaryReader(fs);
            imgBytesIn = br.ReadBytes((int)fs.Length);          //将流读入到字节数组中
        }
        catch
        {
            MessageBox.Show("您选择的图片不能被读取或文件类型不对！", "错误", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
            S_Photo.Image = null;
        }
    }
}
#endregion
```

说明

BinaryReader 类是用特定的编码将基元数据类型读作二进制值。如果该流为 null，或是已关闭，将触发异常。

19.9.3 删除人事档案信息

单击“删除”按钮，将职工基本信息表中的当前记录全部删除，同时根据当前记录的编号，删除工作简历表、家庭关系表、培训记录表、奖惩记录表和个人简历表中的相关记录。代码如下：

```
private void Stu_Delete_Click(object sender, EventArgs e)
{
    if (dataGridView1.RowCount < 2) //判断 dataGridView1 控件中是否有记录
    {
```

```
        MessageBox.Show("数据表为空，不可以删除。");
        return;
    }
    //删除职工信息表中的当前记录及其他相关表中的信息
    MyDataClass.getsqlcom("Delete tb_Staffbasic where ID='" + S_0.Text.Trim() + "'");
    MyDataClass.getsqlcom("Delete tb_WorkResume where Stu_ID='" + S_0.Text.Trim() + "'");
    MyDataClass.getsqlcom("Delete tb_Family where Sta_ID='" + S_0.Text.Trim() + "'");
    MyDataClass.getsqlcom("Delete tb_TrainNote where Sta_ID='" + S_0.Text.Trim() + "'");
    MyDataClass.getsqlcom("Delete tb_RANDP where Sta_ID='" + S_0.Text.Trim() + "'");
    MyDataClass.getsqlcom("Delete tb_WorkResume where Sta_ID='" + S_0.Text.Trim() + "'");
    MyDataClass.getsqlcom("Delete tb_Individual where ID='" + S_0.Text.Trim() + "'");
    Sta_Cancel_Click(sender, e);                                    //调用“取消”按钮的单击事件
}
```

19.9.4　单条件查询人事档案信息

单条件查询人事档案信息运行效果如图 19.24 所示。

图 19.24　单条件查询人事档案信息运行效果

当在“查询类型”下拉列表框中选择查询的类型时，“查询条件”下拉列表框中的值随之改变，然后在“查询条件”下拉列表框中选择要查询的内容，系统根据选择的查询条件调用自定义方法 Condition_Lookup 在数据库中的相关记录，并显示在 DataGridView 控件中。单条件查询人事档案信息的实现代码如下：

```
private void comboBox1_TextChanged(object sender, EventArgs e)
{
    //向 comboBox2 控件中添加相应的查询条件
    switch (comboBox1.SelectedIndex)
    {
        case 0:
        {
            //职工姓名
            MyMC.CityInfo(comboBox2, "select distinct StuffName from tb_Staffbasic", 0);
            tem_Field = "StuffName";
            break;
        }
        case 1:                                                     //性别
        {
            comboBox2.Items.Clear();
            comboBox2.Items.Add("男");
            comboBox2.Items.Add("女");
            tem_Field = "Sex";
            break;
        }
        case 2:
        {
            MyMC.CoPassData(comboBox2, "tb_Folk");          //民族类别
            tem_Field = "Folk";
            break;
```

```
            }
        case 3:
            {
                MyMC.CoPassData(comboBox2, "tb_Culture");                    //文化程度
                tem_Field = "Culture";
                break;
            }
        case 4:
            {
                MyMC.CoPassData(comboBox2, "tb_Visage");                     //政治面貌
                tem_Field = "Visage";
                break;
            }
        case 5:
            {
                MyMC.CoPassData(comboBox2, "tb_EmployeeGenre");              //职工类别
                tem_Field = "Employee";
                break;
            }
        case 6:
            {
                MyMC.CoPassData(comboBox2, "tb_Business");                   //职务类别
                tem_Field = "Business";
                break;
            }
        case 7:
            {
                MyMC.CoPassData(comboBox2, "tb_Branch");                     //部门类别
                tem_Field = "Branch";
                break;
            }
        case 8:
            {
                MyMC.CoPassData(comboBox2, "tb_Duthcall");                   //职称类别
                tem_Field = "Duthcall";
                break;
            }
        case 9:
            {
                MyMC.CoPassData(comboBox2, "tb_Laborage");                   //工资类别
                tem_Field = "Laborage";
                break;
            }
    }
}
private void comboBox2_TextChanged(object sender, EventArgs e)
{
    try
    {
        tem_Value = comboBox2.SelectedItem.ToString();
        Condition_Lookup(tem_Value);
```

```
    }
    catch
    {
        comboBox2.Text = "";
        MessageBox.Show("只能以选择方式查询。");
    }
}
```

实现单条件查询人事档案信息时，用到了自定义方法 Condition_Lookup，该方法用来根据指定的条件查找职工信息，并显示在 DataGridView 控件中。Condition_Lookup 方法的实现代码如下：

```
#region  按条件显示“职工基本信息”表的内容
/// <summary>
/// 通过公共变量动态进行查询
/// </summary>
/// <param name="C_Value">条件值</param>
public void Condition_Lookup(string C_Value)
{
    MyDS_Grid = MyDataClass.getDataSet("Select * from tb_Staffbasic where " + tem_Field + "='" +
tem_Value + "'", "tb_Staffbasic");
    dataGridView1.DataSource = MyDS_Grid.Tables[0];
    textBox1.Text = Grid_Inof(dataGridView1);                    //显示职工信息表的当前记录
}
#endregion
```

19.9.5　逐条查看人事档案信息

“浏览按钮”区域中的 4 个按钮主要实现逐条查看人事档案信息功能，其运行效果如图 19.25 所示。

图 19.25　逐条查看人事档案信息运行效果

当单击图 19.25 中的 4 个按钮时，程序根据单击按钮的 ID 判断将要执行“第一条”、“上一条”、“下一条”和“最后一条”4 项操作中的哪项操作。“浏览按钮”区域中的 4 个按钮的实现代码如下：

```
private void N_First_Click(object sender, EventArgs e)                    //第一条
{
    int ColInd = 0;
    //判断 DataGridView 控件的当前单元格的列索引
    if (dataGridView1.CurrentCell.ColumnIndex == -1 || dataGridView1.CurrentCell.ColumnIndex>1)
        ColInd = 0;
    else
        ColInd = dataGridView1.CurrentCell.ColumnIndex;
    if ((((Button)sender).Name) == "N_First")                              //判断当前单击的是不是“第一条”
    {
        dataGridView1.CurrentCell = this.dataGridView1[ColInd, 0];   //将当前控件的索引设置为 0
        MyMC.Ena_Button(N_First, N_Previous, N_Next, N_Cauda, 0, 0, 1, 1);
    }
    if ((((Button)sender).Name) == "N_Previous")                           //判断当前单击的是不是“上一条”
    {
```

```
        if (dataGridView1.CurrentCell.RowIndex == 0)                    //判断当前行的索引是否为 0
        {
            //调用公共类中的方法设置 4 个按钮的状态
            MyMC.Ena_Button(N_First, N_Previous, N_Next, N_Cauda, 0, 0, 1, 1);
        }
        else
        {
//重新给当前单元格赋值
            dataGridView1.CurrentCell = this.dataGridView1[ColInd, dataGridView1.CurrentCell.RowIndex - 1];
            MyMC.Ena_Button(N_First, N_Previous, N_Next, N_Cauda, 1, 1, 1, 1);
        }
    }
    if ((((Button)sender).Name) == "N_Next")                            //判断当前单击的是不是“下一条”
{
        //判断当前行索引是不是最后一行
        if (dataGridView1.CurrentCell.RowIndex == dataGridView1.RowCount-2)
        {
            //调用公共类中的方法设置 4 个按钮的状态
            MyMC.Ena_Button(N_First, N_Previous, N_Next, N_Cauda, 1, 1, 0, 0);
        }
        else
        {
//重新给当前单元格赋值
            dataGridView1.CurrentCell = this.dataGridView1[ColInd, dataGridView1.CurrentCell.RowIndex + 1];
            MyMC.Ena_Button(N_First, N_Previous, N_Next, N_Cauda, 1, 1, 1, 1);
        }
    }
    if ((((Button)sender).Name) == "N_Cauda")                           //判断当前单击的是不是“最后一条”
{
        //将当前单元格索引设置为最后一行
        dataGridView1.CurrentCell = this.dataGridView1[ColInd, dataGridView1.RowCount - 2];
        MyMC.Ena_Button(N_First, N_Previous, N_Next, N_Cauda, 1, 1, 0, 0);
    }
}
private void N_Previous_Click(object sender, EventArgs e)              //上一条
{
    N_First_Click(sender, e);
}
private void N_Next_Click(object sender, EventArgs e)                  //下一条
{
    N_First_Click(sender, e);
}
private void N_Cauda_Click(object sender, EventArgs e)                 //最后一条
{
    N_First_Click(sender, e);
}
```

说明

在设置具有焦点的单元格时，可以用 dataGridView1[列数,行数]来指定单元格的位置。

19.9.6　将人事档案信息导出为 Word 文档

将人事档案信息导出为 Word 文档，如图 19.26 所示。

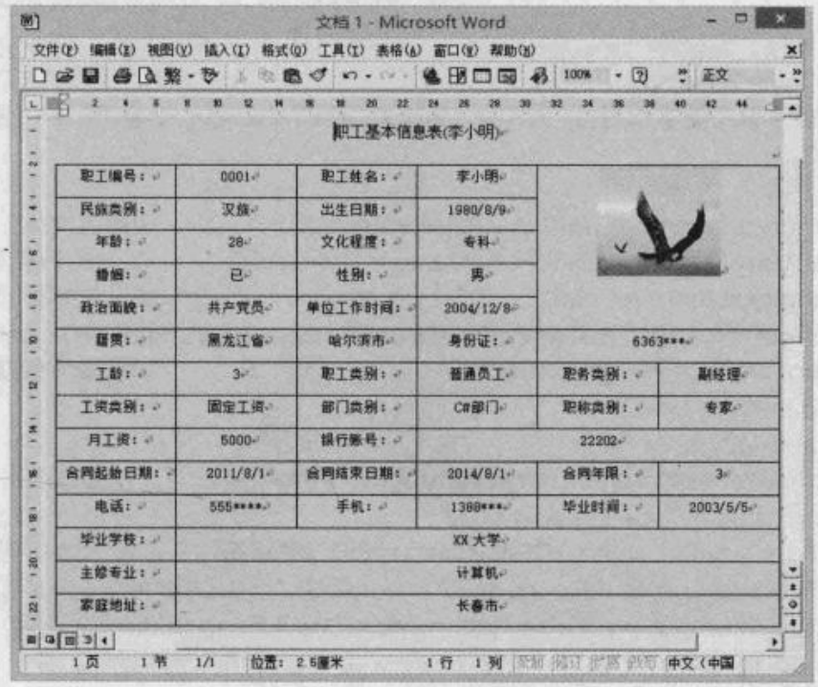


图 19.26　导出的 Word 文档

为了便于职工信息的存储及打印，单击“导出 Word”按钮，可以将职工信息以表格的形式存入 Word 文档中。将人事档案信息导出为 Word 文档的实现代码如下：

```
private void but_Table_Click(object sender, EventArgs e)
{
    object Nothing = System.Reflection.Missing.Value;
    object missing = System.Reflection.Missing.Value;
    //创建 Word 文档
    Microsoft.Office.Interop.Word.Application wordApp = new Microsoft.Office.Interop.Word.Application();
    Microsoft.Office.Interop.Word.Document wordDoc = wordApp.Documents.Add(ref Nothing, ref Nothing, ref
Nothing, ref Nothing);
    wordApp.Visible = true;
    //设置文档宽度
    wordApp.Selection.PageSetup.LeftMargin = wordApp.CentimetersToPoints(float.Parse("2"));
    wordApp.ActiveWindow.ActivePane.HorizontalPercentScrolled = 11;
    wordApp.Selection.PageSetup.RightMargin = wordApp.CentimetersToPoints(float.Parse("2"));
    Object start = Type.Missing;
    Object end = Type.Missing;
    PictureBox pp = new PictureBox();                                    //新建一个 PictureBox 控件
    int p1 = 0;
    for (int i = 0; i < MyDS_Grid.Tables[0].Rows.Count; i++)
    {
        try
```

```
        {
            ShowData_Image((byte[])(MyDS_Grid.Tables[0].Rows[i][23]), pp);
            pp.Image.Save(@"D:\22.bmp");                                    //将图片存入到指定的路径
        }
        catch
        {
            p1 = 1;
        }
        object rng = Type.Missing;
        string strInfo = "职工基本信息表" + "(" + MyDS_Grid.Tables[0].Rows[i][1].ToString() + ")";
        start = 0;
        end = 0;
        wordDoc.Range(ref start, ref end).InsertBefore(strInfo);            //插入文本
        wordDoc.Range(ref start, ref end).Font.Name = "Verdana";            //设置字体
        wordDoc.Range(ref start, ref end).Font.Size = 20;                   //设置字体大小
        wordDoc.Range(ref start, ref end).ParagraphFormat.Alignment = Microsoft.Office.Interop.Word.
WdParagraphAlignment.wdAlignParagraphCenter;                                //设置字体居中
        start = strInfo.Length;
        end = strInfo.Length;
        wordDoc.Range(ref start, ref end).InsertParagraphAfter();           //插入回车
        object missingValue = Type.Missing;
        object location = strInfo.Length; //如果 location 超过已有字符的长度会出错。要比“明细表”串多一个字符
        Microsoft.Office.Interop.Word.Range rng2 = wordDoc.Range(ref location, ref location);
        Microsoft.Office.Interop.Word.Table tab = wordDoc.Tables.Add(rng2, 14, 6, ref missingValue, ref
missingValue);
        tab.Rows.HeightRule = Microsoft.Office.Interop.Word.WdRowHeightRule.wdRowHeightAtLeast;
        tab.Rows.Height = wordApp.CentimetersToPoints(float.Parse("0.8"));
        tab.Range.Font.Size = 10;
        tab.Range.Font.Name = "宋体";
        //设置表格样式
        tab.Borders.InsideLineStyle = Microsoft.Office.Interop.Word.WdLineStyle.wdLineStyleSingle;
        tab.Borders.InsideLineWidth = Microsoft.Office.Interop.Word.WdLineWidth.wdLineWidth050pt;
        tab.Borders.InsideColor = Microsoft.Office.Interop.Word.WdColor.wdColorAutomatic;
        wordApp.Selection.ParagraphFormat.Alignment = Microsoft.Office.Interop.Word.WdParagraphAlignment.
wdAlignParagraphRight;                                                      //设置右对齐
        //第 5 行显示
        tab.Cell(1, 5).Merge(tab.Cell(5, 6));
        //第 6 行显示
        tab.Cell(6, 5).Merge(tab.Cell(6, 6));
        //第 9 行显示
        tab.Cell(9, 4).Merge(tab.Cell(9, 6));
        //第 12 行显示
        tab.Cell(12, 2).Merge(tab.Cell(12, 6));
        //第 13 行显示
        tab.Cell(13, 2).Merge(tab.Cell(13, 6));
        //第 14 行显示
        tab.Cell(14, 2).Merge(tab.Cell(14, 6));
        //第 1 行赋值
        tab.Cell(1, 1).Range.Text = "职工编号：";
        tab.Cell(1, 2).Range.Text = MyDS_Grid.Tables[0].Rows[i][0].ToString();
        tab.Cell(1, 3).Range.Text = "职工姓名：";
```

```
            tab.Cell(1, 4).Range.Text = MyDS_Grid.Tables[0].Rows[i][1].ToString();
            //插入图片
            if (p1 == 0)
            {
                string FileName = @"D:\22.bmp";                                    //图片所在路径
                object LinkToFile = false;
                object SaveWithDocument = true;
                object Anchor = tab.Cell(1, 5).Range;                              //指定图片插入的区域
                //将图片插入到单元格中
                tab.Cell(1, 5).Range.InlineShapes.AddPicture(FileName, ref LinkToFile, ref SaveWithDocument,
ref Anchor);
            }
            p1 = 0;
            //第 2 行赋值
            tab.Cell(2, 1).Range.Text = "民族类别：";
            tab.Cell(2, 2).Range.Text = MyDS_Grid.Tables[0].Rows[i][2].ToString();
            tab.Cell(2, 3).Range.Text = "出生日期：";
            try
            {
                tab.Cell(2, 4).Range.Text = Convert.ToString(Convert.ToDateTime(MyDS_Grid.Tables[0].Rows[i][3]).
ToShortDateString());
            }
            catch { tab.Cell(2, 4).Range.Text = ""; }
            //第 3 行赋值
            tab.Cell(3, 1).Range.Text = "年龄：";
            tab.Cell(3, 2).Range.Text = Convert.ToString(MyDS_Grid.Tables[0].Rows[i][4]);
            tab.Cell(3, 3).Range.Text = "文化程度：";
            tab.Cell(3, 4).Range.Text = MyDS_Grid.Tables[0].Rows[i][5].ToString();
            //第 4 行赋值
            tab.Cell(4, 1).Range.Text = "婚姻：";
            tab.Cell(4, 2).Range.Text = MyDS_Grid.Tables[0].Rows[i][6].ToString();
            tab.Cell(4, 3).Range.Text = "性别：";
            tab.Cell(4, 4).Range.Text = MyDS_Grid.Tables[0].Rows[i][7].ToString();
            //第 5 行赋值
            tab.Cell(5, 1).Range.Text = "政治面貌：";
            tab.Cell(5, 2).Range.Text = MyDS_Grid.Tables[0].Rows[i][8].ToString();
            tab.Cell(5, 3).Range.Text = "单位工作时间：";
            try
            {
                tab.Cell(5, 4).Range.Text = Convert.ToString(Convert.ToDateTime(MyDS_Grid.Tables[0].Rows[0][10]).
ToShortDateString());
            }
            catch { tab.Cell(5, 4).Range.Text = ""; }
            //第 6 行赋值
            tab.Cell(6, 1).Range.Text = "籍贯：";
            tab.Cell(6, 2).Range.Text = MyDS_Grid.Tables[0].Rows[i][24].ToString();
            tab.Cell(6, 3).Range.Text = MyDS_Grid.Tables[0].Rows[i][25].ToString();
            tab.Cell(6, 4).Range.Text = "身份证：";
            tab.Cell(6, 5).Range.Text = MyDS_Grid.Tables[0].Rows[i][9].ToString();
            //第 7 行赋值
            tab.Cell(7, 1).Range.Text = "工龄：";
```

```
        tab.Cell(7, 2).Range.Text = Convert.ToString(MyDS_Grid.Tables[0].Rows[i][11]);
        tab.Cell(7, 3).Range.Text = "职工类别：";
        tab.Cell(7, 4).Range.Text = MyDS_Grid.Tables[0].Rows[i][12].ToString();
        tab.Cell(7, 5).Range.Text = "职务类别：";
        tab.Cell(7, 6).Range.Text = MyDS_Grid.Tables[0].Rows[i][13].ToString();
        //第 8 行赋值
        tab.Cell(8, 1).Range.Text = "工资类别：";
        tab.Cell(8, 2).Range.Text = MyDS_Grid.Tables[0].Rows[i][14].ToString();
        tab.Cell(8, 3).Range.Text = "部门类别：";
        tab.Cell(8, 4).Range.Text = MyDS_Grid.Tables[0].Rows[i][15].ToString();
        tab.Cell(8, 5).Range.Text = "职称类别：";
        tab.Cell(8, 6).Range.Text = MyDS_Grid.Tables[0].Rows[i][16].ToString();
        //第 9 行赋值
        tab.Cell(9, 1).Range.Text = "月工资：";
        tab.Cell(9, 2).Range.Text = Convert.ToString(MyDS_Grid.Tables[0].Rows[i][26]);
        tab.Cell(9, 3).Range.Text = "银行账号：";
        tab.Cell(9, 4).Range.Text = MyDS_Grid.Tables[0].Rows[i][27].ToString();
        //第 10 行赋值
        tab.Cell(10, 1).Range.Text = "合同起始日期：";
        try
        {
            tab.Cell(10, 2).Range.Text = Convert.ToString(Convert.ToDateTime(MyDS_Grid.Tables[0].Rows[i][28]).
ToShortDateString());
        }
        catch { tab.Cell(10, 2).Range.Text = ""; }
        tab.Cell(10, 3).Range.Text = "合同结束日期：";
        try
        {
            tab.Cell(10, 4).Range.Text = Convert.ToString(Convert.ToDateTime(MyDS_Grid.Tables[0].Rows[i][29]).
ToShortDateString());
        }
        catch { tab.Cell(10, 4).Range.Text = ""; }
        tab.Cell(10, 5).Range.Text = "合同年限：";
        tab.Cell(10, 6).Range.Text = Convert.ToString(MyDS_Grid.Tables[0].Rows[i][30]);
        //第 11 行赋值
        tab.Cell(11, 1).Range.Text = "电话：";
        tab.Cell(11, 2).Range.Text = MyDS_Grid.Tables[0].Rows[i][17].ToString();
        tab.Cell(11, 3).Range.Text = "手机：";
        tab.Cell(11, 4).Range.Text = MyDS_Grid.Tables[0].Rows[i][18].ToString();
        tab.Cell(11, 5).Range.Text = "毕业时间：";
        try
        {
            tab.Cell(11, 6).Range.Text = Convert.ToString(Convert.ToDateTime(MyDS_Grid.Tables[0].Rows[i][21]).
ToShortDateString());
        }
        catch { tab.Cell(11, 6).Range.Text = ""; }
        //Convert.ToString(MyDS_Grid.Tables[0].Rows[i][21]);
        //第 12 行赋值
        tab.Cell(12, 1).Range.Text = "毕业学校：";
        tab.Cell(12, 2).Range.Text = MyDS_Grid.Tables[0].Rows[i][19].ToString();
        //第 13 行赋值
```

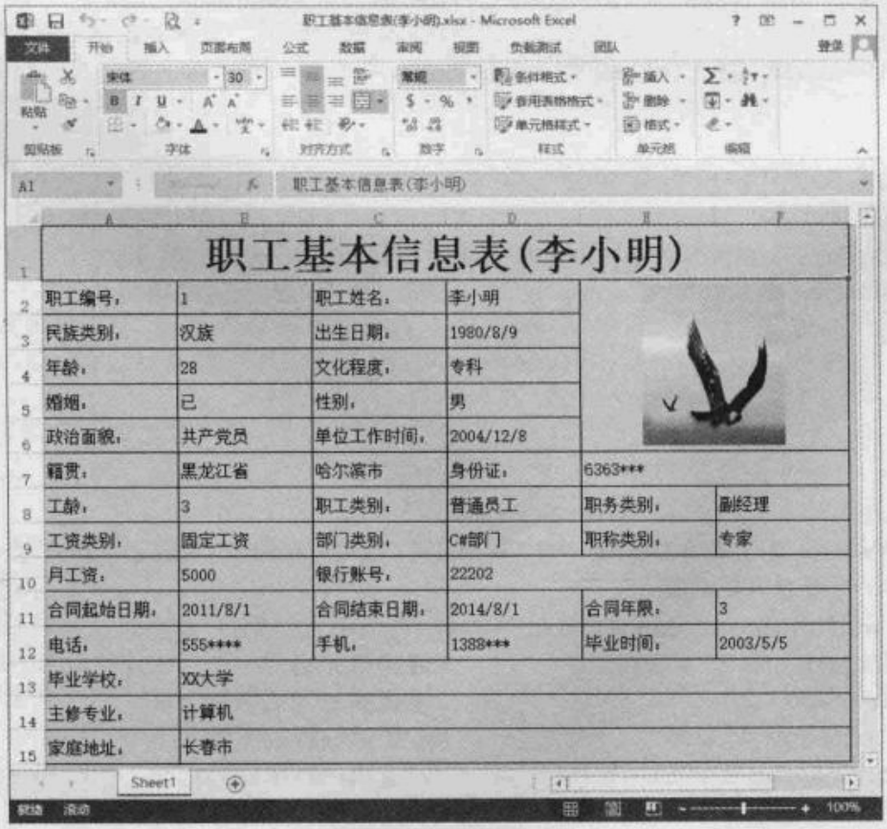
```
            tab.Cell(13, 1).Range.Text = "主修专业：";
            tab.Cell(13, 2).Range.Text = MyDS_Grid.Tables[0].Rows[i][20].ToString();
            //第 14 行赋值
            tab.Cell(14, 1).Range.Text = "家庭地址：";
            tab.Cell(14, 2).Range.Text = MyDS_Grid.Tables[0].Rows[i][22].ToString();
            wordDoc.Range(ref start, ref end).InsertParagraphAfter();                    //插入回车
            wordDoc.Range(ref start, ref end).ParagraphFormat.Alignment = Microsoft.Office.Interop.Word.
WdParagraphAlignment.wdAlignParagraphCenter;          //设置字体居中
        }
        #endregion
}
```

说明

在 C#中如果想要对 Word 文档进行操作，必须对 Word 进行引用。其添加步骤为：首先，在“解决方案资源管理器”中的“引用”上单击鼠标右键，在弹出的快捷菜单中选择“添加引用”命令；然后，在打开的“引用管理器”窗体中选择“程序集”/“扩展”；最后，在该选择卡中选择 Microsoft.Office.Interop.Word，单击“确定”按钮即可。

19.9.7　将人事档案信息导出为 Excel 表格

将人事档案信息导出为 Excel 表格，如图 19.27 所示。



职工基本信息表(李小明)

职工编号：	1	职工姓名：	李小明		
民族类别：	汉族	出生日期：	1980/8/9		
年龄：	28	文化程度：	专科		
婚姻：	已	性别：	男		
政治面貌：	共产党员	单位工作时间：	2004/12/8		
籍贯：	黑龙江省	哈尔滨市	身份证：	6363***	
工龄：	3	职工类别：	普通员工	职务类别：	副经理
工资类别：	固定工资	部门类别：	C#部门	职称类别：	专家
月工资：	5000	银行账号：	22202		
合同起始日期：	2011/8/1	合同结束日期：	2014/8/1	合同年限：	3
电话：	555****	手机：	1388***	毕业时间：	2003/5/5
毕业学校：	XX大学				
主修专业：	计算机				
家庭地址：	长春市				

图 19.27　导出的 Excel 表格

为了便于职工信息的存储及打印，单击“导出 Excel”按钮，可以将职工信息导入 Excel 表格中。将人事档案信息导出为 Excel 表格的实现代码如下：

```
private void Sub_Excel_Click(object sender, EventArgs e)
{
    object rng = Type.Missing;
    //创建 Excel 对象
    Microsoft.Office.Interop.Excel.Application excel = new Microsoft.Office.Interop.Excel.Application();
    Microsoft.Office.Interop.Excel.Workbook workbook = excel.Application.Workbooks.Add(Microsoft.Office.
Interop.Excel.XlWBATemplate.xlWBATWorksheet);
    Microsoft.Office.Interop.Excel.Worksheet worksheet = (Microsoft.Office.Interop.Excel.Worksheet)(workbook.
Worksheets[1]);
    Microsoft.Office.Interop.Excel.Range range = null;
    //获取除第一行之外的所有单元格范围
    range = worksheet.get_Range(excel.Cells[2, 1], excel.Cells[15, 6]);
    range.ColumnWidth = 15;                          //设置单元格宽度
    range.RowHeight = 25;                            //设置单元格高度
    range.Borders.LineStyle = 1;                     //设置边框线的宽度
    //设置边框线的样式
    range.BorderAround2(1, Microsoft.Office.Interop.Excel.XlBorderWeight.xlThin, Microsoft.Office.Interop.Excel.
XlColorIndex.xlColorIndexAutomatic, Color.Black, Type.Missing);
     range.Font.Size = 12;                           //设置字体大小
     range.Font.Name = "宋体";                       //设置字体
     //设置对齐格式为左对齐
     range.HorizontalAlignment = Microsoft.Office.Interop.Excel.XlVAlign.xlVAlignJustify;
     PictureBox pp = new PictureBox();               //新建一个 PictureBox 控件
     int p1 = 0;                                     //定义一个标识，用来标识是否存在照片
     for (int i = 0; i < MyDS_Grid.Tables[0].Rows.Count; i++)
     {
         try
         {
             //获取照片
             ShowData_Image((byte[])(MyDS_Grid.Tables[0].Rows[i][23]), pp);
             pp.Image.Save(@"D:\22.bmp");            //将图片存入到指定的路径
         }
         catch
         {
             p1 = 1;
         }
         //设置标题名称
         string strInfo = "职工基本信息表" + "(" + MyDS_Grid.Tables[0].Rows[i][1].ToString() + ")";
         //设置第 1 行要合并的表格
         range = worksheet.get_Range(excel.Cells[1, 1], excel.Cells[1, 6]);
         range.Merge();                              //合并单元格
         range.Font.Size = 30;                       //设置第一行的字体大小
         range.Font.Name = "宋体";                   //设置第一行的字体
         range.Font.FontStyle = "Bold";              //设置第一行字体为粗体
         //设置标题居中显示
         range.HorizontalAlignment = Microsoft.Office.Interop.Excel.XlVAlign.xlVAlignCenter;
```

```
            excel.Cells[1, 1] = strInfo;                    //设置标题
            //第 2 行到第 6 行的合并范围，用来显示照片
            range = worksheet.get_Range(excel.Cells[2, 5], excel.Cells[6, 6]);
            range.Merge(true);
            //第 7 行显示
            range = worksheet.get_Range(excel.Cells[7, 5], excel.Cells[7, 6]);
            range.Merge(true);
            //第 10 行显示
            range = worksheet.get_Range(excel.Cells[10, 4], excel.Cells[10, 6]);
            range.Merge(true);
            //第 13 行显示
            range = worksheet.get_Range(excel.Cells[13, 2], excel.Cells[13, 6]);
            range.Merge(true);
            //第 14 行显示
            range = worksheet.get_Range(excel.Cells[14, 2], excel.Cells[14, 6]);
            range.Merge(true);
            //第 15 行显示
            range = worksheet.get_Range(excel.Cells[15, 2], excel.Cells[15, 6]);
            range.Merge(true);
            //第 1 行赋值
            excel.Cells[2, 1] = "职工编号：";
            excel.Cells[2, 2] = MyDS_Grid.Tables[0].Rows[i][0].ToString();
            excel.Cells[2, 3] = "职工姓名：";
            excel.Cells[2, 4] = MyDS_Grid.Tables[0].Rows[i][1].ToString();
            //插入照片
            if (p1 == 0)
            {
                string FileName = @"D:\22.bmp";      //照片所在路径
                range = worksheet.get_Range(excel.Cells[2, 5], excel.Cells[6, 5]);
                range.Merge();
worksheet.Shapes.AddPicture(FileName,Microsoft.Office.Core.MsoTriState.msoFalse,Microsoft.Office.Core.Ms
oTriState.msoTrue,418, 43, 100, 115);
            }
            p1 = 0;
            //第 2 行赋值
            excel.Cells[3, 1] = "民族类别：";
            excel.Cells[3, 2] = MyDS_Grid.Tables[0].Rows[i][2].ToString();
            excel.Cells[3, 3] = "出生日期：";
            try
            {
                excel.Cells[3,  4]  =  Convert.ToString(Convert.ToDateTime(MyDS_Grid.Tables[0].Rows[i][3]).
ToShortDateString());
            }
            catch { excel.Cells[3, 4] = ""; }
            //第 3 行赋值
            excel.Cells[4, 1] = "年龄：";
            excel.Cells[4, 2] = Convert.ToString(MyDS_Grid.Tables[0].Rows[i][4]);
            excel.Cells[4, 3] = "文化程度：";
            excel.Cells[4, 4] = MyDS_Grid.Tables[0].Rows[i][5].ToString();
```

```
            //第 4 行赋值
            excel.Cells[5, 1] = "婚姻：";
            excel.Cells[5, 2] = MyDS_Grid.Tables[0].Rows[i][6].ToString();
            excel.Cells[5, 3] = "性别：";
            excel.Cells[5, 4] = MyDS_Grid.Tables[0].Rows[i][7].ToString();
            //第 5 行赋值
            excel.Cells[6, 1] = "政治面貌：";
            excel.Cells[6, 2] = MyDS_Grid.Tables[0].Rows[i][8].ToString();
            excel.Cells[6, 3] = "单位工作时间：";
            try
            {
                excel.Cells[6, 4] = Convert.ToString(Convert.ToDateTime(MyDS_Grid.Tables[0].Rows[0][10]).
ToShortDateString());
            }
            catch { excel.Cells[6, 4] = ""; }
            //第 6 行赋值
            excel.Cells[7, 1] = "籍贯：";
            excel.Cells[7, 2] = MyDS_Grid.Tables[0].Rows[i][24].ToString();
            excel.Cells[7, 3] = MyDS_Grid.Tables[0].Rows[i][25].ToString();
            excel.Cells[7, 4] = "身份证：";
            excel.Cells[7, 5] = MyDS_Grid.Tables[0].Rows[i][9].ToString();
            //第 7 行赋值
            excel.Cells[8, 1] = "工龄：";
            excel.Cells[8, 2] = Convert.ToString(MyDS_Grid.Tables[0].Rows[i][11]);
            excel.Cells[8, 3] = "职工类别：";
            excel.Cells[8, 4] = MyDS_Grid.Tables[0].Rows[i][12].ToString();
            excel.Cells[8, 5] = "职务类别：";
            excel.Cells[8, 6] = MyDS_Grid.Tables[0].Rows[i][13].ToString();
            //第 8 行赋值
            excel.Cells[9, 1] = "工资类别：";
            excel.Cells[9, 2] = MyDS_Grid.Tables[0].Rows[i][14].ToString();
            excel.Cells[9, 3] = "部门类别：";
            excel.Cells[9, 4] = MyDS_Grid.Tables[0].Rows[i][15].ToString();
            excel.Cells[9, 5] = "职称类别：";
            excel.Cells[9, 6] = MyDS_Grid.Tables[0].Rows[i][16].ToString();
            //第 9 行赋值
            excel.Cells[10, 1] = "月工资：";
            excel.Cells[10, 2] = Convert.ToString(MyDS_Grid.Tables[0].Rows[i][26]);
            excel.Cells[10, 3] = "银行账号：";
            excel.Cells[10, 4] = MyDS_Grid.Tables[0].Rows[i][27].ToString();
            //第 10 行赋值
            excel.Cells[11, 1] = "合同起始日期：";
            try
            {
                excel.Cells[11, 2] = Convert.ToString(Convert.ToDateTime(MyDS_Grid.Tables[0].Rows[i][28]).
ToShortDateString());
            }
            catch { excel.Cells[11, 2] = ""; }
            excel.Cells[11, 3] = "合同结束日期：";
```

```
            try
            {
                excel.Cells[11, 4] = Convert.ToString(Convert.ToDateTime(MyDS_Grid.Tables[0].Rows[i][29]).
ToShortDateString());
            }
            catch { excel.Cells[11, 4] = ""; }
            excel.Cells[11, 5] = "合同年限：";
            excel.Cells[11, 6] = Convert.ToString(MyDS_Grid.Tables[0].Rows[i][30]);
            //第 11 行赋值
            excel.Cells[12, 1] = "电话：";
            excel.Cells[12, 2] = MyDS_Grid.Tables[0].Rows[i][17].ToString();
            excel.Cells[12, 3] = "手机：";
            excel.Cells[12, 4] = MyDS_Grid.Tables[0].Rows[i][18].ToString();
            excel.Cells[12, 5] = "毕业时间：";
            try
            {
                excel.Cells[12, 6] = Convert.ToString(Convert.ToDateTime(MyDS_Grid.Tables[0].Rows[i][21]).
ToShortDateString());
            }
            catch { excel.Cells[12, 6] = ""; }
            //Convert.ToString(MyDS_Grid.Tables[0].Rows[i][21]);
            //第 12 行赋值
            excel.Cells[13, 1] = "毕业学校：";
            excel.Cells[13, 2] = MyDS_Grid.Tables[0].Rows[i][19].ToString();
            //第 13 行赋值
            excel.Cells[14, 1] = "主修专业：";
            excel.Cells[14, 2] = MyDS_Grid.Tables[0].Rows[i][20].ToString();
            //第 14 行赋值
            excel.Cells[15, 1] = "家庭地址：";
            excel.Cells[15, 2] = MyDS_Grid.Tables[0].Rows[i][22].ToString();
            if (!System.IO.File.Exists("D:\\" + strInfo + ".xlsx"))
                worksheet.SaveAs("D:\\" + strInfo + ".xlsx", Type.Missing, Type.Missing, Type.Missing, Type.Missing,
Type.Missing, Type.Missing, Type.Missing, Type.Missing, Type.Missing);
            else
                worksheet.Copy(Type.Missing, Type.Missing);
            workbook.Save();                                    //保存工作表
            workbook.Close(false, Type.Missing, Type.Missing);  //关闭工作表
            MessageBox.Show("基本信息表导出到 Excel 成功，位置: D:\\" + strInfo + ".xlsx", "提示");
        }
    }
```

说明

在 C#中如果想要对 Excel 进行操作，必须对 Excel 进行引用。其添加步骤为：首先，在“解决方案资源管理器”中的“引用”上单击鼠标右键，在弹出的快捷菜单中选择“添加引用”命令；然后，在打开的“引用管理器”窗体中选择 COM /“类型库”；最后，在该选择卡中选择 Microsoft Excel 版本号 Object Library，单击“确定”按钮即可。

19.10 用户设置模块设计

视频讲解：光盘\TM\lx\19\用户设置模块设计.mp4

本模块使用的数据表：tb_Login、tb_UserPope

用户设置模块主要对企业人事管理系统中的用户信息进行管理，包括对用户信息的添加、修改和删除等操作，还可以为指定的用户设置操作权限。另外，如果要对管理员信息进行修改、删除和设置操作权限等操作，系统会提示不能对管理员进行操作。“用户设置”窗口运行结果如图 19.28 所示。

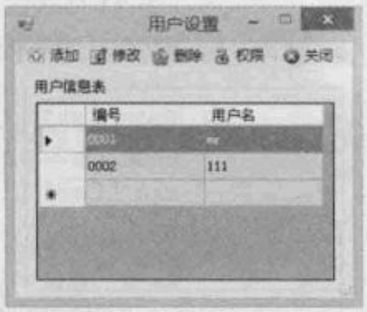


图 19.28 “用户设置”窗口

19.10.1 设计用户设置窗体

新建一个 Windows 窗体，命名为 F_User.cs，主要用于对该系统的用户信息进行管理。F_User 窗体用到的主要控件如表 19.16 所示。

表 19.16 用户设置窗体用到的主要控件

控件类型	控件 ID	主要属性设置	用途
ToolStrip	toolStrip1	添加 5 个 ToolStripButton 按钮，并分别命名为 tool_UserAdd、tool_UserAmend、tool_UserDelete、tool_UserPopedom 和 tool_Close	作为该窗体中的工具栏
DataGridView	dataGridView1	将其 SelectionMode 属性设置为 FullRowSelect	显示用户信息

19.10.2 添加/修改用户信息

添加用户信息和修改用户信息窗体的运行效果分别如图 19.29 和图 19.30 所示。

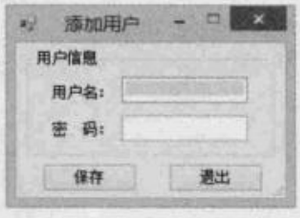


图 19.29 添加用户信息

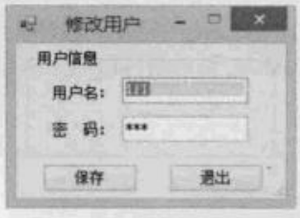


图 19.30 修改用户信息

在 F_User 窗体中单击工具栏中的“添加”/“修改”按钮，实例化 F_UserAdd 窗体的一个对象，并分别为该对象的 Tag 属性赋值为 1 和 2，以标识在 F_UserAdd 窗体中将执行哪种操作。工具栏中的“添加”/“修改”按钮的实现代码如下：

```
private void tool_UserAdd_Click(object sender, EventArgs e)
{
```

```
    //实例化 F_UserAdd 窗体类对象
    PerForm.F_UserAdd FrmUserAdd = new F_UserAdd();
    //设置 F_UserAdd 窗体的 Tag 属性为 1，以标识执行添加操作
    FrmUserAdd.Tag = 1;
    FrmUserAdd.Text = tool_UserAdd.Text + "用户";                    //设置 F_UserAdd 窗体的标题
    FrmUserAdd.ShowDialog(this);                                     //以对话框形式显示窗体
}
private void tool_UserAmend_Click(object sender, EventArgs e)
{
    if (ModuleClass.MyModule.User_ID.Trim() == "0001")               //判断选择的是不是超级用户
    {
        MessageBox.Show("不能修改超级用户。");
        return;
    }
    //实例化 F_UserAdd 窗体类对象
    PerForm.F_UserAdd FrmUserAdd = new F_UserAdd();
    //设置 F_UserAdd 窗体的 Tag 属性为 2，以标识执行修改操作
    FrmUserAdd.Tag = 2;
    FrmUserAdd.Text = tool_UserAmend.Text + "用户";                  //设置 F_UserAdd 窗体的标题
    FrmUserAdd.ShowDialog(this);                                     //以对话框形式显示窗体
}
```

在 F_UserAdd 窗体中单击“保存”按钮，判断“用户名”文本框和“密码”文本框是否为空。如果为空，则弹出提示信息；否则，根据该窗体的 Tag 属性值判断是执行用户添加操作，还是执行用户修改操作。“保存”按钮的实现代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    if (text_Name.Text == "" && text_Pass.Text == "")                //判断用户名和密码是否为空
    {
        MessageBox.Show("请将用户名和密码添加完整。");
        return;
    }
    DSet = MyDataClass.getDataSet("select Name from tb_Login where Name='" + text_Name.Text + "'",
"tb_Login");
    //判断窗体的 Tag 属性是否为 2，以执行修改操作
    if ((int)this.Tag == 2 && text_Name.Text == ModuleClass.MyModule.User_Name)
    {
        MyDataClass.getsqlcom("update tb_Login set Name='" + text_Name.Text + "',Pass='" + text_Pass.
Text + "' where ID='" + ModuleClass.MyModule.User_ID + "'");
        return;
    }
    if (DSet.Tables[0].Rows.Count > 0)                               //判断用户是否已经存在
    {
        MessageBox.Show("当前用户名已存在，请重新输入。");            //弹出提示信息
        text_Name.Text = "";
        text_Pass.Text = "";
        return;
    }
```

```
    //判断窗体的 Tag 属性是否为 1，以执行添加操作
    if ((int)this.Tag == 1)
    {
        AutoID = MyMC.GetAutocoding("tb_Login", "ID");          //自动生成编号
        //调用公共类中的方法添加用户信息
        MyDataClass.getsqlcom("insert into tb_Login (ID,Name,Pass) values('" + AutoID + "','" + text_Name.
Text + "','" + text_Pass.Text + "')");
        MyMC.ADD_Pope(AutoID, 0);                               //为新添加的用户设置权限
        MessageBox.Show("添加成功。");
    }
    else
    {
        //调用公共类中的方法修改用户信息
        MyDataClass.getsqlcom("update tb_Login set Name='" + text_Name.Text + "',Pass='" + text_Pass.
Text + "' where ID='" + ModuleClass.MyModule.User_ID + "'");
        //判断新添加的用户编号是否与登录用户的编号相同
        if (ModuleClass.MyModule.User_ID == DataClass.MyMeans.Login_ID)
            DataClass.MyMeans.Login_Name = text_Name.Text;  //设置登录用户名为“用户名”文本框的值
        MessageBox.Show("修改成功。");
    }
    this.Close();                                               //关闭当前窗体
}
```

19.10.3 删除用户基本信息

在 F_User 窗体中单击工具栏中的“删除”按钮，判断要删除的用户是不是管理员。如果是，则弹出提示信息，提示不能修改管理员信息；否则，删除选中的用户信息，同时删除其权限信息。工具栏中“删除”按钮的实现代码如下：

```
private void tool_UserDelete_Click(object sender, EventArgs e)
{
    if (ModuleClass.MyModule.User_ID != "")
    {
        if (ModuleClass.MyModule.User_ID.Trim() == "0001")      //判断要删除的用户是不是超级用户
        {
            MessageBox.Show("不能删除超级用户。");
            return;
        }
        //删除用户信息
        MyDataClass.getsqlcom("Delete tb_Login where ID='" + ModuleClass.MyModule.User_ID.Trim() + "'");
        //删除用户权限信息
        MyDataClass.getsqlcom("Delete tb_UserPope where ID='" + ModuleClass.MyModule.User_ID.Trim() + "'");
        //在数据库中查找所有用户信息，并将结果存储在 DataSet 数据集中
        MyDS_Grid = MyDataClass.getDataSet("select ID as 编号,Name as 用户名 from tb_Login", "tb_Login");
        dataGridView1.DataSource = MyDS_Grid.Tables[0];         //为 DataGridView 控件设置数据源
    }
    else
```

```
            MessageBox.Show("无法删除空数据表。");
}
```

说明

在对数据表中的数据进行删除后，必须对窗体中的数据进行更新，以显示最新的数据内容。

19.10.4　设置用户操作权限

图 19.31　设置用户操作权限的运行效果

在 F_User 窗体中单击工具栏中的“权限”按钮，弹出 F_UserPope 窗体，如图 19.31 所示。

“用户权限设置”窗口中可以设置用户的权限，在该窗口中选中要拥有权限的复选框，单击“保存”按钮，调用 MyModule 公共类中的 Amend_Pope 方法为用户设置权限，同时将 MyMeans 公共类中的静态变量 Login_n 设置为 2，以便在调用“重新登录”窗体时，使用新设置的权限对其进行初始化。设置用户操作权限的实现代码如下：

```
private void User_Save_Click(object sender, EventArgs e)
{
    //调用公共类的 Amend_Pope 方法为指定的用户设置权限
    MyMC.Amend_Pope(groupBox2.Controls, ModuleClass.MyModule.User_ID);
    //判断登录用户的编号是否与修改的用户编号相同
    if (DataClass.MyMeans.Login_ID == ModuleClass.MyModule.User_ID)
        //将静态变量 Login_n 设置为 2，以便在调用“重新登录”窗体时，使用新设置的权限对其进行初始化
        DataClass.MyMeans.Login_n = 2;
}
```

19.11　数据库维护模块设计

视频讲解：光盘\TM\lx\19\数据库维护模块设计.mp4

数据库维护模块主要对企业人事管理系统中的数据信息进行备份和还原操作，其运行结果如图 19.32 所示。

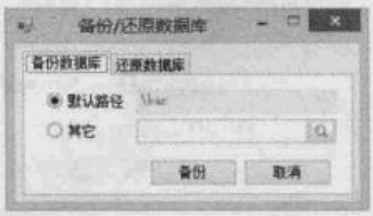


图 19.32　“备份/还原数据库”窗口

19.11.1　设计数据库维护窗体

新建一个 Windows 窗体，命名为 F_HaveBack.cs，主要用于对该系统的数据进行备份和还原。F_HaveBack 窗体用到的主要控件如表 19.17 所示。

表 19.17　数据库维护窗体用到的主要控件

控 件 类 型	控件 ID	主要属性设置	用　途
TextBox	textBox1	将 Text 属性设置为“\bar”	备份数据库的默认路径
	textBox2	无	指定备份数据库的路径
	textBox3	无	输入备份文件的路径名
Button	button1	无	执行数据库备份操作
	button2	无	选择要存放备份文件的路径
	button3	无	关闭当前窗体
	button4	无	选择备份文件的存放路径
	button5	无	执行数据库还原操作
	button6	无	关闭当前窗体
RadioButton	radioButton1	将 Checked 属性设置为 TRUE	是否将备份文件放到默认路径中
	radioButton2	无	是否指定新的备份文件路径
OpenFileDialog	openFileDialog1	无	选择备份文件的对话框
FolderBrowserDialog	folderBrowserDialog1	无	选择存放备份文件路径的对话框
TabControl	tabControl1	添加两个选项卡，并分别将其 Text 属性设置为“备份数据库”和“还原数据库”	显示“备份数据库”和“还原数据库”两个选项卡

19.11.2　备份数据库

在“备份数据库”选项卡中单击“备份”按钮，程序首先判断是将备份文件存放到默认路径下，还是存放到用户选择的路径下，然后对数据库文件进行备份。备份数据库的实现代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    string Str_dar = "";
    if (radioButton1.Checked == true)                          //判断默认路径是否选中
        Str_dar = System.Environment.CurrentDirectory + "\\bar\\";
    if (radioButton2.Checked == true)                          //判断自定义路径是否选中
        Str_dar = textBox2.Text+ "\\";
    if (textBox2.Text == "" & radioButton2.Checked == true)
    {
        MessageBox.Show("请选择备份数据库文件的路径。");
        return;
    }
    try
    {
        //定义数据库备份的 SQL 语句
        Str_dar = "backup database db_PWMS to disk='" + Str_dar+System.DateTime.Now.ToShortDateString().
Replace("/","")+MyMC.Time_Format(System.DateTime.Now.ToString())+".bak" + "'";
        //调用公共类中的方法执行数据库备份操作
```

```
            MyDataClass.getsqlcom(Str_dar);
            MessageBox.Show("数据备份成功！", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message, "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
    }
```

说明

在对数据库中的数据进行备份时，必须关闭当前数据库的所有连接。

19.11.3　还原数据库

还原数据库运行效果如图 19.33 所示。

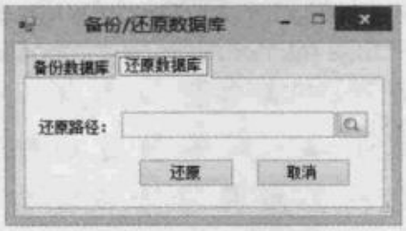


图 19.33　还原数据库运行效果

在“还原数据库”选项卡中单击“还原”按钮，程序首先调用 kill 命令将与 db_PWMS 数据库有关的进程全部强行关闭，然后重新备份该数据库的日志文件，同时对该数据库进行还原操作。还原数据库的实现代码如下：

```
private void button5_Click(object sender, EventArgs e)
{
    if (textBox3.Text == "")                                    //判断备份文件路径是否为空
    {
        MessageBox.Show("请选择备份数据库文件的路径。");
        return;
    }
    try
    {
        //判断数据库连接状态是否打开
        if (DataClass.MyMeans.My_con.State == ConnectionState.Open)
        {
            DataClass.MyMeans.My_con.Close();                   //关闭数据库连接
        }
        string DateStr = "Data Source=mrwxk\\wxk;Database=master;User id=sa;PWD=";
        SqlConnection conn = new SqlConnection(DateStr);        //实例化 SqlConnection 连接对象
        conn.Open();                                            //打开数据库连接
        //------------------杀掉所有连接 db_PWMS 数据库的进程----------
        string strSQL = "select spid from master..sysprocesses where dbid=db_id( 'db_PWMS') ";
```

```
        SqlDataAdapter Da = new SqlDataAdapter(strSQL, conn);        //实例化 SqlDataAdapter 类对象
        DataTable spidTable = new DataTable();                       //实例化 DataTable 对象
        Da.Fill(spidTable);                                          //填充 DataTable 数据表
        SqlCommand Cmd = new SqlCommand();                           //实例化 SqlCommand 对象
        Cmd.CommandType = CommandType.Text;                          //设置 SqlCommand 命令的类型
        Cmd.Connection = conn;                                       //设置 SqlCommand 命令的连接对象
        //循环访问 DataTable 数据表中的行
        for (int iRow = 0; iRow < spidTable.Rows.Count ; iRow++)
        {
            Cmd.CommandText = "kill " + spidTable.Rows[iRow][0].ToString();  //强行关闭用户进程
            Cmd.mp4cuteNonQuery();                                   //执行 SqlCommand 命令
        }
        conn.Close();                                                //关闭数据库连接
        conn.Dispose();                                              //释放数据库连接资源
        //重新连接数据库
        SqlConnection Tem_con = new SqlConnection(DataClass.MyMeans.M_str_sqlcon);
        Tem_con.Open();                                              //打开数据库连接
        //使用数据库还原语句实例化 SqlCommand 对象
        SqlCommand SQLcom = new SqlCommand("backup log db_PWMS to disk='"
            + textBox3.Text.Trim() + "'use master restore database db_PWMS from disk='"
            + textBox3.Text.Trim() + "'", Tem_con);
        SQLcom.mp4cuteNonQuery();                                    //执行数据库还原操作
        SQLcom.Dispose();                                            //释放 SqlCommand 对象
        Tem_con.Close();                                             //关闭数据库连接
        Tem_con.Dispose();                                           //释放数据库连接资源
        MessageBox.Show("数据还原成功！", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
        MyDataClass.con_open();
        MyDataClass.con_close();
        MessageBox.Show("为了避免数据丢失，在数据库还原后将关闭整个系统。");
        Application.Exit();                                          //退出当前应用程序
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "提示", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}
```

19.12 小　　结

本章根据软件的开发流程，对企业人事管理系统的开发过程进行了详细讲解。通过对本章的学习，读者应该能够掌握如何用自定义方法对多个不同的数据表进行添加、修改、删除以及多字段组合查询等操作。另外，还应该掌握如何将数据库中的信息导出到 Word 文档中，以方便打印。

第20章

Java + SQL Server 实现企业进销存管理系统

（视频讲解：73 分钟）

进销存管理系统是促进企业发展的重要组成部分，是商业企业经营管理中的核心环节，也是一个企业能否取得效益的关键，如果能够做到合理采购、及时销售、库存量最小、减少积压，那么企业就能取得最佳的经济效益。在现代社会中，大多数事业、企业单位，特别是中小型企业，实现信息化管理是首要任务。只有实现信息化管理，才能提高工作效率和企业的管理水平。市场经济快速多变，竞争激烈，企业采用信息化管理进货、库存、销售等诸多环节也已成为趋势及必然。

本章将使用 Java Swing 技术和 SQL Server 2012 数据库开发跨平台的企业进销存管理系统，该系统是典型的 MIS（管理信息系统），主要包括创建并维护后台数据库和前端应用程序的开发两个方面。

通过阅读本章，您可以：

- 学会如何使用 JDBC 技术操作 SQL Server 2012 数据库
- 学会如何使用 Swing 技术的高级布局管理器
- 学会如何实现数据库的备份与恢复
- 学会如何使用 Swing 菜单栏与工具栏
- 学会如何使用 Desktop 类实现系统资源的关联
- 掌握内部窗体的各种操作

20.1 系 统 概 述

视频讲解：光盘\TM\lx\20\系统概述.mp4

企业进销存管理系统的主要工作是对企业的进货、销售和库存以信息化的方式进行管理，最大限度地减少各个环节中可能出现的错误，有效减少盲目采购、降低采购成本、合理控制库存、减少资金占用并提高市场灵敏度，使企业能够合理安排进、销、存的每个关键步骤，提升企业市场竞争力。针对经营管理中存在的问题，吉林省铭泰××有限公司对产品的进销存合理化提出了更高的要求（概括地讲，用户对进销存系统的需求具有普遍性）。

20.2 系 统 设 计

视频讲解：光盘\TM\lx\20\系统设计.mp4

20.2.1 系统目标

根据企业对进销存管理的要求，制定企业进销存管理系统的目标如下：

- ☑ 灵活的人机交互界面，操作简单方便，界面简洁美观。
- ☑ 键盘操作，快速响应。
- ☑ 对进货和销售提供相应的退货管理功能。
- ☑ 实现各种查询，如多条件查询、模糊查询等。
- ☑ 可以随时修改系统口令。
- ☑ 灵活的数据备份、还原功能。
- ☑ 系统最大限度地实现了易安装性、易维护性和易操作性。
- ☑ 系统运行稳定、安全可靠。

20.2.2 系统功能结构

铭泰企业进销存管理系统的功能结构如图 20.1 所示。

20.2.3 系统业务流程图

铭泰企业进销存管理系统的业务流程图如图 20.2 所示。

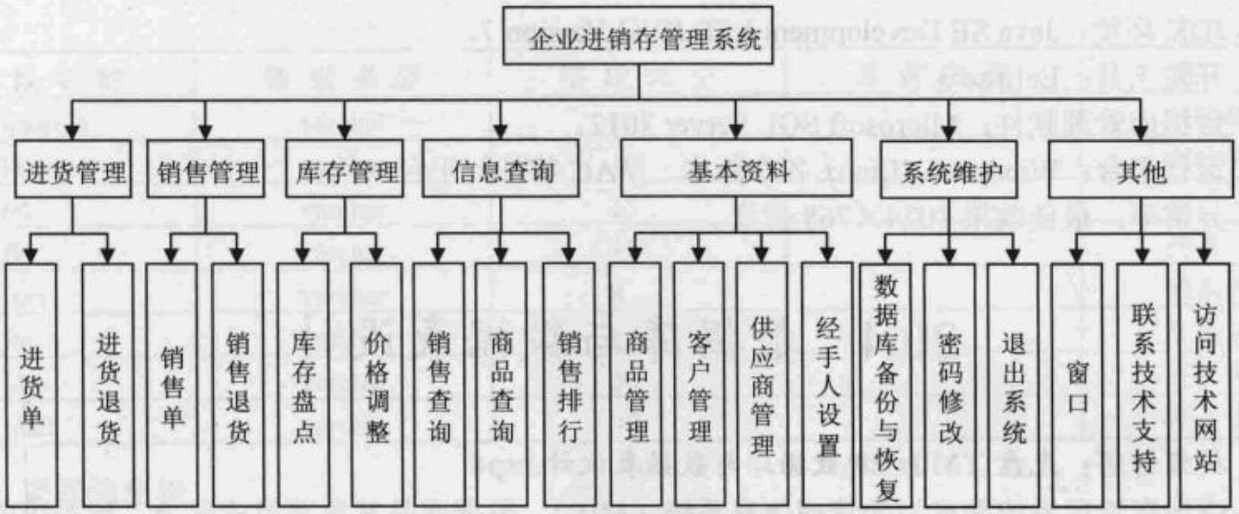


图 20.1　铭泰企业进销存管理系统功能结构图

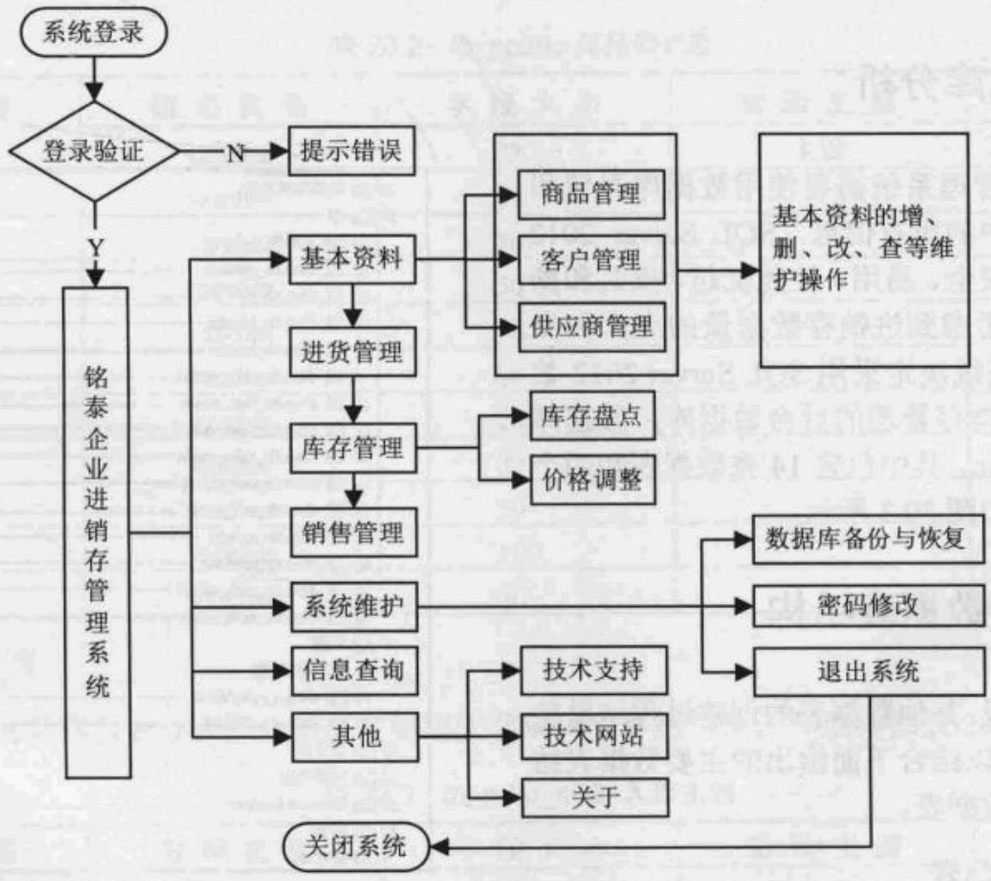


图 20.2　铭泰企业进销存管理系统的业务流程图

20.3　开 发 环 境

视频讲解：光盘\TM\lx\20\开发环境.mp4

本系统的软件开发环境如下。

☑　操作系统：Windows 7。

- ☑ JDK 环境：Java SE Development KIT(JDK) Version 7。
- ☑ 开发工具：Eclipse 3.7。
- ☑ 数据库管理软件：Microsoft SQL Server 2012。
- ☑ 运行平台：Windows、Linux 各个版本、MAC 等任何平台。
- ☑ 分辨率：最佳效果 1024×768 像素。

20.4 数据库与数据表设计

视频讲解：光盘\TM\lx\20\数据库与数据表设计.mp4

企业进销存管理系统是典型的管理信息系统（MIS），数据库是其重要组成部分。该系统的数据库设计是根据需求分析以及系统功能结构制定的。

20.4.1 数据库分析

企业进销存管理系统需要使用数据库存储和管理进销存过程中的所有信息。SQL Server 2012 数据库系统具有安全、易用、性能优越、安装和操作简便等优点。考虑到进销存数据量的庞大和安全性的保障，本系统决定采用 SQL Server 2012 数据库系统作为进销存管理的后台数据库，数据库名称为 db_database。其中包含 14 张数据表和两个视图，详细信息如图 20.3 所示。

图 20.3 铭泰企业进销存管理系统中用到的数据表和视图

20.4.2 主要数据表结构

由于篇幅有限，其他数据表的创建过程这里就不介绍了，读者可以结合下面给出的主要数据表结构，创建其他的数据表。

1. 供应商信息表

供应商信息表的名称为 tb_gysinfo，主要用于存储供应商详细信息，其结构如表 20.1 所示。

表 20.1 tb_gysinfo 供应商信息表

字段名称	数据类型	字段大小	是否主键	说明
id	varchar	32	主键	供应商编号
name	varchar	50		供应商名称
jc	varchar	20		供应商简称

续表

字段名称	数据类型	字段大小	是否主键	说　明
address	varchar	100		供应商地址
bianma	varchar	10		邮政编码
tel	varchar	15		电话
fax	varchar	15		传真
lian	varchar	8		联系人
ltel	varchar	15		联系电话
yh	varchar	50		开户银行
mail	varchar	30		电子信箱

2. 商品信息表

商品信息表的名称为 tb_spinfo，主要用于存储商品详细信息，其结构如表 20.2 所示。

表 20.2　tb_spinfo 商品信息表

字段名称	数据类型	字段大小	是否主键	说　明
id	varchar	32	主键	商品编号
spname	varchar	50		商品名称
jc	varchar	30		商品简称
cd	varchar	50		产地
dw	varchar	10		商品计量单位
gg	varchar	10		商品规格
bz	varchar	20		包装
ph	varchar	32		批号
pzwh	varchar	50		批准文号
memo	varchar	100		备注
gysname	varchar	50		供应商名称

3. 入库主表

入库主表的名称为 tb_ruku_main，主要用于存储入库单据信息，其结构如表 20.3 所示。

表 20.3　tb_ruku_main 入库主表

字段名称	数据类型	字段大小	是否主键	说　明
rkID	varchar	32	主键	入库编号
pzs	float	8		品种数量
je	money	8		总计金额
ysjl	varchar	50		验收结论
gysname	varchar	100		供应商名称
rkdate	datetime	8		入库时间
czy	varchar	30		操作员
jsr	varchar	30		经手人
jsfs	varchar	10		结算方式

4. 入库明细表

入库明细表的名称为 tb_ruku_detail，主要用于存储入库详细信息，其结构如表 20.4 所示。

表 20.4 tb_ruku_detail 入库明细表

字段名称	数据类型	字段大小	是否主键	说明
id	varchar	50	主键	流水号
rkID	varchar	30		入库编号
spid	varchar	50		商品编号
dj	money	8		单价
sl	float	8		数量

5. 销售主表

销售主表的名称为 tb_sell_main，主要用于存储销售单据信息，其结构如表 20.5 所示。

表 20.5 tb_sell_main 销售主表

字段名称	数据类型	字段大小	是否主键	说明
sellID	varchar	30	主键	销售编号
pzs	float	8		销售品种数
je	money	8		总计金额
ysjl	varchar	50		验收结论
khname	varchar	100		客户名称
xsdate	datetime	8		销售日期
czy	varchar	30		操作员
jsr	varchar	30		经手人
jsfs	varchar	10		结算方式

6. 销售明细表

销售明细表的名称为 tb_sell_detail，主要用于存储销售详细信息，其结构如表 20.6 所示。

表 20.6 tb_sell_detail 销售明细表

字段名称	数据类型	字段大小	是否主键	说明
id	varchar	50	主键	流水号
sellID	varchar	50		销售编号
spid	varchar	50		商品编号
dj	money	8		销售单价
sl	float	8		销售数量

20.5 创建项目

视频讲解：光盘\TM\lx\20\创建项目.mp4

铭泰企业进销存管理系统是使用 Eclipse 开发的一个 Java 项目。在 Eclipse 开发环境中创建“进销

存管理系统”项目的具体步骤如下：

（1）启动 Eclipse，在 Eclipse 工作台中选择“文件”/“新建”/“Java 项目”命令，如图 20.4 所示。

（2）弹出“新建 Java 项目”窗口，在“项目名”文本框中输入新建项目的名称，在“位置”栏中选择项目的创建位置，可以选择默认位置（即当前工作空间），也可以单击“浏览”按钮指定具体的位置，这里采用默认位置，然后在 JRE 栏中选择该项目所使用的 JRE（Java SE Runtime Environment，Java 运行环境）；接下来在“项目布局”栏中选中“为源文件和类文件创建单独的文件夹”单选按钮；最后还可以为该项目指定工作集，这里不指定任何工作集，单击“完成”按钮，如图 20.5 所示。

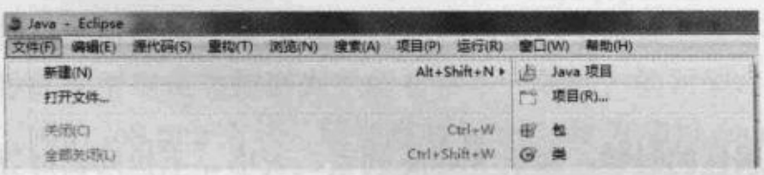


图 20.4　选择“新建”命令

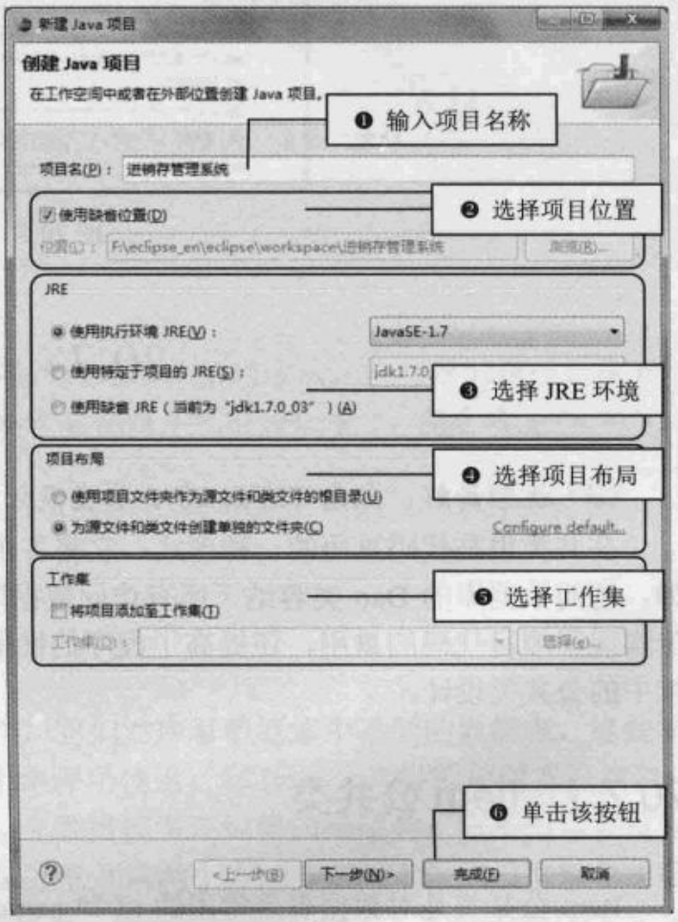


图 20.5　“新建 Java 项目”窗口

20.6　系统文件夹组织结构

视频讲解：光盘\TM\lx\20\系统文件夹组织结构.mp4

由于铭泰企业进销存管理系统是由团队进行开发的，所以在编写项目代码之前，需要制定好项目的系统文件夹组织结构，如不同的 Java 包存放不同的窗体、公共类、数据模型、工具类或者图片资源等，这样不但可以保证团队开发的一致性，也可以规范系统的整体架构。创建完系统中可能用到的文件夹或 Java 包之后，在开发时，只需将所创建的类文件或资源文件保存到相应的文件夹中即可。铭泰企业进销存管理系统的文件夹组织结构如图 20.6 所示。

图 20.6　文件夹组织结构

20.7　公共类设计

视频讲解：光盘\TM\lx\20\公共类设计.mp4

公共类也是代码重用的一种形式，它将各个功能模块经常调用的方法提取到共用的 Java 类中，例如，访问数据库的 Dao 类容纳了所有访问数据库的方法，并同时管理着数据库的连接和关闭。这样不但实现了项目代码的重用，还提高了程序的性能和代码的可读性。本节将介绍铭泰企业进销存管理系统中的公共类设计。

20.7.1　Item 公共类

Item 公共类是对数据表最常用的 id 和 name 属性的封装，用于 Swing 列表、表格、下拉列表框等组件的赋值。该类重写了 toString()方法，在该方法中只输出 name 属性，所以 Item 类在 Swing 组件显示文本时只包含名称信息，不会连带着 id 属性。但是，在获取组件的内容时，获取的是 Item 类的对象，从该对象中可以很容易地获取 id 属性，然后通过该属性到数据库中获取唯一的数据。Item 公共类的程序代码如下：

```
package com.lzw;
public class Item {
    private String id;                                  //id 编号属性
    private String name;                                //name 属性
    public Item() {                                     //默认构造方法
    }
    public Item(String id, String name) {               //包含所有属性初始化的构造方法
        this.id = id;
        this.name = name;
```

```
    }
    public String getId() {                    //获取 id 属性的方法
        return id;
    }
    public void setId(String id) {             //设置 id 属性的方法
        this.id = id;
    }
    public String getName() {                  //获取 name 属性的方法
        return name;
    }
    public void setName(String name) {         //设置 name 属性的方法
        this.name = name;
    }
    public String toString() {                 //重写 toString()方法，只输出 name 属性
        return getName();
    }
}
```

注意

代码中相应的 get×××方法和 set×××方法，是访问不同属性的方法，而相应的属性则被声明为 private 私有属性，这样就实现了属性的封装。在本章其他程序代码的介绍中，将省略 get××× 方法和 set××× 方法的程序代码。

20.7.2　数据模型公共类

在 com.lzw.dao.model 包中存放的是数据模型公共类，它们对应着数据库中不同的数据表，这些模型将被访问数据库的 Dao 类和程序中各个模块甚至各个组件所使用。和 Item 公共类的使用方法类似，数据模型也是对数据表中所有字段（属性）的封装，但是数据模型是纯粹的模型类，它不但需要重写父类的 toString 方法，还要重写 hashCode 方法和 equals 方法（这两个方法分别用于生成模型对象的哈希代码和判断模型对象是否相同）。模型类主要用于存储数据，并通过相应的 get××× 方法和 set××× 方法实现不同属性的访问原则。现在以商品数据表为例，介绍它所对应的模型类的关键代码。

```
package com.lzw.dao.model;
public class TbSpinfo implements java.io.Serializable {
    private String id;              //id 属性
    private String spname;          //商品名称
    private String jc;              //商品简称
    private String cd;              //产地
    private String dw;              //单位
    private String gg;              //规格
    private String bz;              //包装
    private String ph;              //批号
    private String pzwh;            //批准文号
    private String memo;            //备注
```

```
    private String gysname;                    //供应商名称
    public TbSpinfo() {                        //默认的构造方法
    }
    …//省略 getXXX()方法和 setXXX()方法
    public String toString() {                 //重写 toString 方法
        return getSpname();
    }
    @Override
    public int hashCode() {                    //重写生成 hashCode 的方法
        final int PRIME = 31;
        int result = 1;
        result = PRIME * result + ((bz == null) ? 0 : bz.hashCode());
        result = PRIME * result + ((cd == null) ? 0 : cd.hashCode());
        result = PRIME * result + ((dw == null) ? 0 : dw.hashCode());
        result = PRIME * result + ((gg == null) ? 0 : gg.hashCode());
        result = PRIME * result
                + ((gysname == null) ? 0 : gysname.hashCode());
        result = PRIME * result + ((id == null) ? 0 : id.hashCode());
        result = PRIME * result + ((jc == null) ? 0 : jc.hashCode());
        result = PRIME * result + ((memo == null) ? 0 : memo.hashCode());
        result = PRIME * result + ((ph == null) ? 0 : ph.hashCode());
        result = PRIME * result + ((pzwh == null) ? 0 : pzwh.hashCode());
        result = PRIME * result
                + ((spname == null) ? 0 : spname.hashCode());
        return result;
    }
    @Override
    public boolean equals(Object obj) {        //重写 equals 方法
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        final TbSpinfo other = (TbSpinfo) obj;
        if (bz == null) {
            if (other.bz != null)
                return false;
        } else if (!bz.equals(other.bz))
            return false;
        if (cd == null) {
            if (other.cd != null)
                return false;
        } else if (!cd.equals(other.cd))
        …//省略部分判断代码
        } else if (!spname.equals(other.spname))
            return false;
        return true;
    }
}
```

其他模型类的定义和商品模型类的定义方法类似，其属性内容就是数据表中相应的字段。com.lzw.dao.model 包中包含的数据模型类如表 20.7 所示。

表 20.7　com.lzw.dao.model 包中的数据模型类

类　名	说　明	类　名	说　明
TbGysinfo	供应商数据表模型类	TbRukuMain	进货主表模型类
TbJsr	经手人数据表模型类	TbSellDetail	销售详细信息数据表模型类
TbKhinfo	客户数据表模型类	TbSellMain	销售主表模型类
TbKucun	库存数据表模型类	TbSpinfo	商品信息数据表模型类
TbRkthDetail	进货退货详细数据表模型类	TbXsthDetail	销售退货详细信息数据表模型类
TbRkthMain	进货退货主数据表模型类	TbXsthMain	销售退货主表模型类
TbRukuDetail	进货详细信息数据表模型类		

说明

表中所有模型类都定义了对应数据表字段的属性，并提供了访问相应属性的 get×××()方法和 set×××()方法。

20.7.3　Dao 公共类

Dao 的全称是 Data Access Object，即数据访问对象。本项目中应用该名称作为数据库访问类的名称，在该类中实现了数据库的驱动、连接、关闭和多个操作数据库的方法，这些方法包括不同数据表的操作方法。在介绍具体的数据库访问方法之前，先来看一下 Dao 类的定义，也就是数据库驱动和连接的代码。关键代码如下：

```
package com.lzw.dao;
import java.sql.*;                    //导入其他类包
import java.sql.Date;
import java.util.*;
import com.lzw.Item;
import com.lzw.dao.model.*;
public class Dao {
    //定义数据库驱动类的名称
    protected static String dbClassName = "net.sourceforge.jtds.jdbc.Driver";
    //定义访问数据库的 URL
    protected static String dbUrl = "jdbc:jtds:sqlserver://localhost:1433/"
            + "db_database;SelectMethod=Cursor";
    //定义访问数据库的用户名
    protected static String dbUser = "sa";
    //定义访问数据库的密码
    protected static String dbPwd = "";
    //声明数据库的连接对象
    public static Connection conn = null;
```

```
    static {                                          //在静态代码段中初始化 Dao 类，实现数据库的驱动和连接
        try {
            if (conn == null) {
                Class.forName(dbClassName).newInstance();
                conn = DriverManager.getConnection(dbUrl, dbUser, dbPwd);
            }
        } catch (Exception ee) {
            ee.printStackTrace();
        }
    }
    private Dao() {                                   //封闭构造方法，禁止创建 Dao 类的实例对象
    }
}
```

初始化代码

Dao 类中的所有数据库操作方法都使用 static 关键字定义为静态方法，所以 Dao 类不需要创建对象，可以直接调用类中的所有数据库操作方法。下面对 Dao 类中关键的自定义方法进行详细介绍。

1. getKhInfo(Item item)方法

该方法用于获取客户信息，方法的返回值是 TbKhinfo 类的对象，即客户信息的数据模型。方法将接收一个 Item 类的实例对象，通过该对象获取客户的 ID 编号，然后从数据库中获取该 ID 编号的数据信息，并封装到客户信息的数据模型中，这个数据模型最后会作为方法的返回值，返回给方法的调用者。该方法的关键代码如下：

```
//读取客户信息
public static TbKhinfo getKhInfo(Item item) {
    //获取 item 对象的 name 属性
    String where = "khname='" + item.getName() + "'";
    if (item.getId() != null)
        where = "id='" + item.getId() + "'";      //获取 item 对象的 id 属性
    TbKhinfo info = new TbKhinfo();               //创建客户信息数据模型
    ResultSet set = findForResultSet("select * from tb_khinfo where "+ where);
    try {
        if (set.next()) {                         //封装数据到数据模型中
            info.setId(set.getString("id").trim());
            info.setKhname(set.getString("khname").trim());
            info.setJian(set.getString("jian").trim());
            info.setAddress(set.getString("address").trim());
            info.setBianma(set.getString("bianma").trim());
            info.setFax(set.getString("fax").trim());
            info.setHao(set.getString("hao").trim());
            info.setLian(set.getString("lian").trim());
            info.setLtel(set.getString("ltel").trim());
            info.setMail(set.getString("mail").trim());
            info.setTel(set.getString("tel").trim());
            info.setXinhang(set.getString("xinhang").trim());
        }
    } catch (SQLException e) {
```

```
            e.printStackTrace();
        }
        return info;                                    //将数据模型作为返回值
    }
```

注意

Item 就是之前所讲到的公共类，它用于封装 id 和 name 属性。

2. getGysInfo(Item item)方法

该方法用于获取供应商信息，方法的返回值是 TbGysinfo 类的对象，即供应商数据表的模型对象。方法将接收 Item 类的对象作为参数，从 item 对象中获取供应商的编号和名称，然后从数据库中获取该编号的供应商信息并封装到供应商数据模型对象中，最后将该模型对象作为方法的返回值返回给方法的调用者。该方法的关键代码如下：

```
//读取指定供应商信息
public static TbGysinfo getGysInfo(Item item) {
    String where = "name='" + item.getName() + "'";      //从 item 对象获取 id 信息
    if (item.getId() != null)
        where = "id='" + item.getId() + "'";             //从 item 对象获取 name 信息
    TbGysinfo info = new TbGysinfo();                    //创建供应商数据模型
    ResultSet set = findForResultSet("select * from tb_gysinfo where "
            + where);                                    //查询数据
    try {
        if (set.next()) {                                //将供应商信息封装到数据模型中
            info.setId(set.getString("id").trim());
            info.setAddress(set.getString("address").trim());
            info.setBianma(set.getString("bianma").trim());
            info.setFax(set.getString("fax").trim());
            info.setJc(set.getString("jc").trim());
            info.setLian(set.getString("lian").trim());
            info.setLtel(set.getString("ltel").trim());
            info.setMail(set.getString("mail").trim());
            info.setName(set.getString("name").trim());
            info.setTel(set.getString("tel").trim());
            info.setYh(set.getString("yh").trim());
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return info;                                         //将供应商数据模型返回给调用者
}
```

（将供应商封装到数据模型中）

3. getSpInfo(Item item)方法

该方法用于获取商品信息，该方法的返回值是 TbSpinfo 类的对象，即商品数据表的模型对象。方法的参数是 Item 公共类的对象 item。该对象封装了商品的 id 和 name 属性，通过这两个属性可以从数据库中获取指定编号的商品信息。该方法将获取到的商品信息封装为商品数据模型，然后返回给方法

的调用者。该方法的关键代码如下：

```
//读取商品信息
public static TbSpinfo getSpInfo(Item item) {
    String where = "spname='" + item.getName() + "'";          //获取商品名称
    if (item.getId() != null)
        where = "id='" + item.getId() + "'";                   //获取商品编号
    ResultSet rs = findForResultSet("select * from tb_spinfo where "+ where);
    TbSpinfo spInfo = new TbSpinfo();                          //创建商品数据模型对象
    try {
        if (rs.next()) {
            spInfo.setId(rs.getString("id").trim());
            spInfo.setBz(rs.getString("bz").trim());
            spInfo.setCd(rs.getString("cd").trim());
            spInfo.setDw(rs.getString("dw").trim());
            spInfo.setGg(rs.getString("gg").trim());
            spInfo.setGysname(rs.getString("gysname").trim());
            spInfo.setJc(rs.getString("jc").trim());
            spInfo.setMemo(rs.getString("memo").trim());
            spInfo.setPh(rs.getString("ph").trim());
            spInfo.setPzwh(rs.getString("pzwh").trim());
            spInfo.setSpname(rs.getString("spname").trim());
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return spInfo;                                             //返回商品数据模型对象
}
```

将商品信息封装到数据模型中

4．getLogin(String name, String password)方法

该方法用于判断登录用户的用户名与密码是否正确，方法的返回值是 boolean 布尔类型，接收的参数有 name 和 password，分别是用户名与密码信息。该方法将在一条 SQL 语句中获取指定用户名和密码的数据。如果用户名和密码正确，则返回 TRUE 值，否则返回 FALSE 值。关键代码如下：

```
public static boolean getLogin(String name, String password)
        throws SQLException {
    ResultSet rs = findForResultSet("select * from tb_userlist where name='"
            + name + "' and pass='" + password + "'");          //执行 SQL 查询
    return rs.next();
}
```

注意

该方法没有对 SQL 注入进行相应的防范处理。

5．insertSellInfo(TbSellMain sellMain)方法

该方法用于添加销售信息到数据库中，它将在事务中完成对销售主表、销售明细表和库存表的添

加与保存操作。基于事务的安全原则，如果对任何一个数据表的操作失败，将导致整个事务回滚，恢复到之前的数据状态。因此，该方法执行前后可以保证数据库的完整性不被破坏，同时完成对销售信息的添加业务。在 JDBC 中使用事务的关键是调用 Connection 类的 setAutoCommit 方法设置自动提交模式为 FALSE，完成业务之后，再调用 commit 方法手动提交事务。关键代码如下：

```
//在事务中添加销售信息
public static boolean insertSellInfo(TbSellMain sellMain) {
    try {
        boolean autoCommit = conn.getAutoCommit();
        conn.setAutoCommit(false);
        //添加销售主表记录
        insert("insert into tb_sell_main values('" + sellMain.getSellId()
                + "','" + sellMain.getPzs() + "'," + sellMain.getJe()
                + ",'" + sellMain.getYsjl() + "','" + sellMain.getKhname()
                + "','" + sellMain.getXsdate() + "','" + sellMain.getCzy()
                + "','" + sellMain.getJsr() + "','" + sellMain.getJsfs()
                + "')");
        Set<TbSellDetail> rkDetails = sellMain.getTbSellDetails();
        for (Iterator<TbSellDetail> iter = rkDetails.iterator(); iter
                .hasNext();) {
            TbSellDetail details = iter.next();
            //添加销售详细表记录
            insert("insert into tb_sell_detail values('"
                    + sellMain.getSellId() + "','" + details.getSpid()
                    + "'," + details.getDj() + "," + details.getSl() + ")");
            //修改库存表记录
            Item item = new Item();
            item.setId(details.getSpid());
            TbSpinfo spInfo = getSpInfo(item);
            if (spInfo.getId() != null && !spInfo.getId().isEmpty()) {
                TbKucun kucun = getKucun(item);
                if (kucun.getId() != null && !kucun.getId().isEmpty()) {
                    int sl = kucun.getKcsl() - details.getSl();
                    update("update tb_kucun set kcsl=" + sl + " where id='"
                            + kucun.getId() + "'");
                }
            }
        }
        conn.commit();
        conn.setAutoCommit(autoCommit);
    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
    return true;
}
```

操作销售主表

操作库存表

注意

在开始 JDBC 事务之前最好使用 getAutoCommit 获取原有的事务提交模式并保存，在执行 commit 方法提交事务之后，记得使用 setAutoCommit 方法恢复原有的提交模式。

Dao 类中还有关于进货、进货退货和销售退货业务的 insert×××方法，其实现方法和该方法类似，这里就不再介绍了，相信读者能够举一反三，理解其他方法的业务逻辑。

6．restoreOrBackup(String sql)方法

该方法用于执行数据库备份与恢复的 SQL 语句。由于这类数据库操作涉及多用户操作、数据库连接共享等多种可能导致备份与恢复语句无法执行的问题，所以该方法首先设置数据库的选项为单用户连接模式，再执行相应的 SQL 语句，这样就可以避免数据库恢复或备份时失败。该方法的关键代码如下：

```
public static int restoreOrBackup(String sql) throws Exception {
    int rs = 0;
    if (conn != null) {
        conn.close();
    }
    //连接到系统数据库
    conn = DriverManager.getConnection(
            "jdbc:jtds:sqlserver://localhost:1433/master", dbUser,
            dbPwd);
    Statement stmt = conn.createStatement();
    //定义单用户连接模式的 SQL 语句
    String single = "alter database db_database set single_user"
            + " with rollback immediate " + sql;
    //执行数据库备份或恢复的 SQL 语句
    rs = stmt.mp4cuteUpdate(single);
    stmt.close();
    conn.close(); //关闭数据库连接
    //恢复原有数据库连接
    conn = DriverManager.getConnection(dbUrl, dbUser, dbPwd);
    return rs;
}
```

7．checkLogin(String userStr, String passStr)方法

该方法用于判断登录用户的用户名与密码是否正确，方法的返回值是 boolean 布尔类型，接收的参数有 userStr 和 passStr，分别是用户名与密码信息。该方法将在一条 SQL 语句中获取指定用户名和密码的数据。如果用户名和密码正确，则返回 TRUE 值，否则返回 FALSE 值。关键代码如下：

```
public static boolean checkLogin(String userStr, String passStr)
        throws SQLException {
    ResultSet rs = findForResultSet("select * from tb_userlist where name='"
            + userStr + "' and pass='" + passStr + "'");
    if (rs == null)
        return false;
```

```
        return rs.next();
    }
```

20.8　系统登录模块设计

视频讲解：光盘\TM\lx\20\系统登录模块设计.mp4

本模块使用的数据表：tb_userlist

现在开始设计项目开发的第一步——系统登录模块。该模块的设计与实现方法比较普遍，非常适合该项目的起步设计工作，读者可以从最简单的自由布局开始，逐渐了解项目开发所需要的各种高级布局管理器和高级组件应用。同时，系统登录也是项目必须开发的模块，它是系统的安全门，只有提供正确的用户名和登录口令之后，才能够进入铭泰企业进销存管理系统进行进销存管理操作。本系统的登录用户名是 Tsoft，密码是 111。登录模块的界面效果如图 20.7 所示。

图 20.7　系统登录

20.8.1　设计登录窗体

登录模块的窗体设计由两部分组成，一部分是登录窗体；另一部分是窗体中带背景图片的内容面板。下面来看一下登录窗体的设计。

1．创建内容面板

所有组件都要布置在窗体的内容面板上，而登录模块的内容面板使用了背景图片来美化窗体界面，这就需要继承 Swing 的 JPanel 类编写自己的面板类，然后将该面板类作为窗体的内容面板。程序代码如下：

```
package com.lzw.login;
import java.awt.*;
import java.net.URL;
import javax.swing.*;
public class LoginPanel extends JPanel {
    public int width, height;
    private Image img;
    public LoginPanel() {                                    //在构造方法中创建背景图片
        super();
        URL url = getClass().getResource("/res/login.jpg");  //获取图片的 URL
        img = new ImageIcon(url).getImage();                 //初始化 img 对象
    }
    protected void paintComponent(Graphics g) {              //重写父类的组件绘制方法
        super.paintComponent(g);
        g.drawImage(img, 0, 0, this);                        //在面板左上角开始绘制背景图片
    }
}
```

2. 创建登录窗体

创建 LoginDialog 类，该类继承 JFrame 类，成为一个窗体。设置窗体的标题为“系统登录”，设置内容面板为 LoginPanel 类的对象。该窗体用于布置各种组件，来实现系统登录的界面。窗体用到的主要控件如表 20.8 所示。

表 20.8　登录窗体用到的主要控件

组 件 类 型	组件 ID	主要属性设置	说　明
JTextField	userField	无	登录“用户名”文本框
JPasswordField	passwordField	无	登录“密码”文本框
JButton	loginButton	Text 属性设置为“登录”	“登录”按钮
	exitButton	Text 属性设置为“退出”	“退出”按钮

20.8.2　“密码”文本框的回车事件

在系统登录窗体的“密码”文本框中添加了按键事件监听器，它在获取到“密码”文本框输入的回车字符时将执行登录事件，也就是说在“密码”文本框输入密码后，按 Enter 键将执行与单击“登录”按钮相同的业务逻辑。“密码”文本框程序代码如下：

```
private JPasswordField getPasswordField() {
    if (passwordField == null) {
        passwordField = new JPasswordField();
        passwordField.setBounds(new Rectangle(143, 69, 125, 22));
        passwordField.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyTyped(java.awt.event.KeyEvent e) {
                if(e.getKeyChar()=='\n')//如果按键字符是换行符 '\n'
                    loginButton.doClick();//执行“登录”按钮的 doClick()方法
            }
        });
    }
    return passwordField;
}
```

20.8.3　“登录”按钮的事件处理

“登录”按钮用于执行用户名和密码的验证工作，如果验证用户名和密码有效，则启动系统，否则禁止进入系统。

在“登录”按钮的动作事件监听器中，首先获取用户输入的用户名与密码信息，然后调用 Dao 类的 checkLogin 方法，如果该方法返回 TRUE 则登录成功，否则禁止用户登录，并提示输入的用户名与密码无法登录系统。程序关键代码如下：

```
private JButton getLoginButton() {//初始化“登录”按钮的方法
    if (loginButton == null) {
```

```
        loginButton = new JButton();
        loginButton.setBounds(new Rectangle(109, 114, 48, 20));
        loginButton.setIcon(new ImageIcon(getClass().getResource(
                "/res/loginButton.jpg")));
        //添加按钮的动作监听器
        loginButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                try {
                    //设置本地系统外观样式
                    UIManager.setLookAndFeel(UIManager
                            .getSystemLookAndFeelClassName());
                    userStr = userField.getText();//获取用户名
                    //获取密码
                    String passStr = new String(passwordField.getPassword());
                    if (!Dao.checkLogin(userStr, passStr)) {
                        JOptionPane.showMessageDialog(LoginDialog.this,
                                "用户名与密码无法登录", "登录失败",
                                JOptionPane.ERROR_MESSAGE);
                        return;
                    }
                } catch (Exception e1) {
                    e1.printStackTrace();
                }
                mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                mainFrame.setVisible(true);
                //设置状态栏的操作员
                mainFrame.getCzyStateLabel().setText(userStr);
                setVisible(false);//隐藏登录窗体
            }
        });
    }
    return loginButton;
}
```

（验证用户登录信息；显示主窗体）

说明

checkLogin()方法是在 20.7 节公共类中所讲的 Dao 方法，用于验证登录的用户名与密码是否有效。

20.9　系统主窗体设计

视频讲解：光盘\TM\lx\20\系统主窗体设计.mp4

主窗体是人机交互的主体，用户通过主窗体中提供的各种菜单、表格、文本框、子窗体等组件进行管理操作。本系统主界面采用的是 MID（即“多文档界面”），类似于 Word 应用程序，可以同时打开多个子窗体进行操作。可以对打开的功能窗体进行各种操作，如窗口平铺、全部还原、全部关闭，并在菜单中列出当前打开子窗体的名称，如图 20.8 所示。

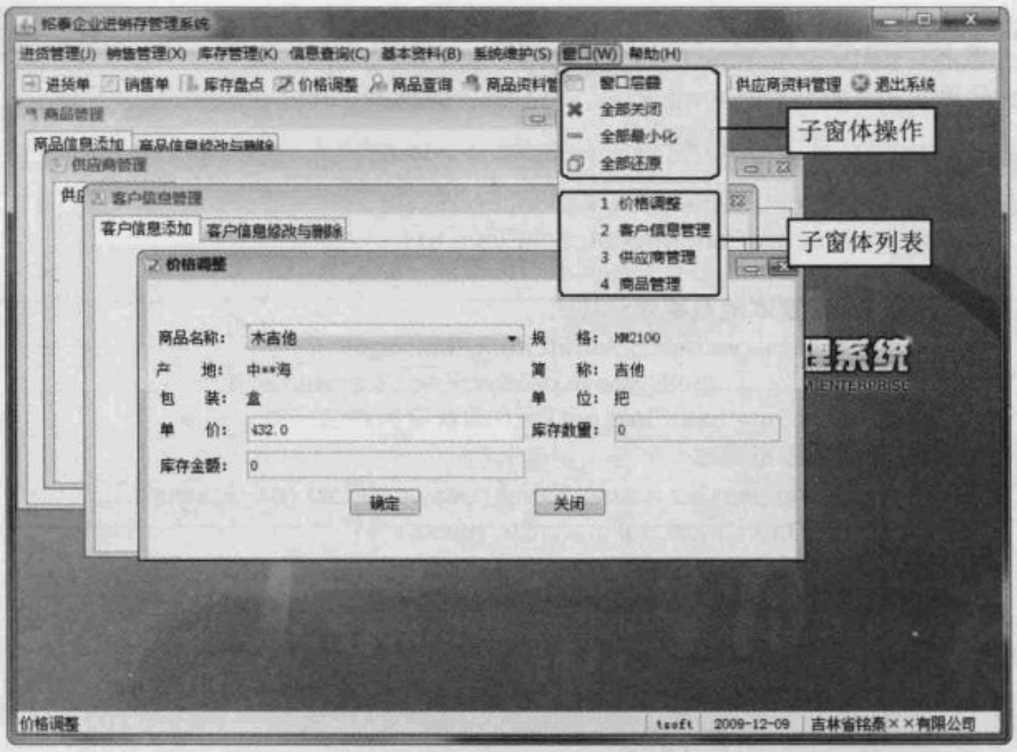


图 20.8　企业进销存管理系统主窗体

20.9.1　设计菜单栏

本系统的菜单栏是由 MenuBar 类实现的，该类是一个自定义菜单栏类，它继承 JMenuBar 类成为 Swing 的菜单栏组件。下面以创建进货单菜单为例，介绍一下菜单栏组件的创建步骤。

（1）创建 MenuBar 类，该类继承 javax.swing.JMenuBar 类，并且在该类中定义一个私有的成员变量，类型为 JMenu，用于表示菜单对象，关键代码如下：

```
public class MenuBar extends JMenuBar {
    private JMenu jinhuo_Menu = null;
}
```

（2）编写一个名称为 getJinhuo_Menu 的方法，该方法的返回值为一个 JMenu 对象，也就是一个菜单对象。在该方法中，当进货菜单对象为 null 时，创建一个菜单对象，并为其设置菜单名和快捷键，关键代码如下：

```
/**
 * 初始化进货管理菜单的方法
 *
 * @return javax.swing.JMenu
 */
public JMenu getJinhuo_Menu() {
    if (jinhuo_Menu == null) {
        jinhuo_Menu = new JMenu();                    //创建一个菜单对象
        jinhuo_Menu.setText("进货管理(J)");            //设置菜单名称
```

```
            jinhuo_Menu.setMnemonic(KeyEvent.VK_J);    //设置快捷键
        }
        return jinhuo_Menu;
    }
```

（3）编写一个初始化菜单栏界面的方法 initialize，在该方法中，首先设置组件的尺寸，然后调用 JMenuBar 对象的 add 方法向菜单栏中添加一个菜单，关键代码如下：

```
/**
 * 初始化菜单栏界面的方法
 *
 */
private void initialize() {
    this.setSize(new Dimension(600, 24));
    add(getJinhuo_Menu());
}
```

（4）编写以下构造方法，用于调用初始化菜单栏界面。

```
public MenuBar(JDesktopPane desktopPanel, JLabel label) {
    super();
    initialize();                                    //调用初始化菜单栏界面的方法
}
```

说明

至此，就可以创建一个“进货管理”菜单，运行效果如图 20.9 所示。接下来还需要为该菜单添加菜单项。

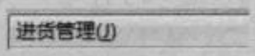


图 20.9　“进货管理”菜单

（5）在 MenuBar 类中再创建一个 JMenuItem 类型的成员变量 jinhuoItem，表示进货菜单项，关键代码如下：

```
private JMenuItem jinhuoItem = null;
```

（6）编写一个名称为 getJinhuoItem 的方法，该方法的返回值为一个 JMenuItem 对象，也就是一个菜单项对象。在该方法中，当“进货单”菜单项对象为 null 时，创建一个菜单项对象，并为其设置菜单项名、图标和动作事件监听器，关键代码如下：

```
/**
 * 初始化（进货单）菜单项的方法
 *
 * @return javax.swing.JMenuItem
 */
public JMenuItem getJinhuoItem() {
    if (jinhuoItem == null) {
        jinhuoItem = new JMenuItem();
```

```
            jinhuoItem.setText("进货单");
            jinhuoItem.setIcon(new ImageIcon(getClass().getResource(
                    "/res/icon/jinhuodan.png")));
            jinhuoItem.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    //编写用于打开进货单窗口的代码
                }
            });
        }
        return jinhuoItem;
    }
```

注意

组件的图标资源或图片资源应尽量使用 Swing 支持的 JPG、GIF 或 PNG 格式。

（7）按照步骤（5）和步骤（6）的方法再创建一个“进货退货”菜单项对象，名称为 jinhuo_tuihuoItem。

（8）在 getJinhuo_Menu 方法中，应用 JMenu 对象的 add 方法向菜单中添加菜单项，关键代码如下：

```
jinhuo_Menu.add(getJinhuoItem());
jinhuo_Menu.add(getJinhuo_tuihuoItem());
```

至此，就完成了“进货管理”菜单的创建，最后的运行效果如图 20.10 所示。

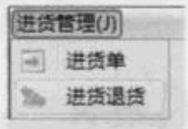


图 20.10　创建完成的“进货管理”菜单

20.9.2　设计工具栏

工具栏用于放置常用命令按钮，如进货单、销售单、库存盘点等。本系统的工具栏界面如图 20.11 所示。

图 20.11　工具栏界面

向本系统中添加工具栏的方法和添加菜单栏的方法类似，也需要继承 Swing 的 JTool 组件编写自己的工具栏。当然，读者也可以根据自己的思路直接使用 Swing 的 JTool 组件。本系统为实现代码重用，所以重新定义了工具栏组件。组件的 initialize 方法用于初始化工具栏的程序界面。关键代码如下：

```
private void initialize() {                                    //初始化工具栏界面的方法
    setSize(new Dimension(600, 24));
    setBorder(BorderFactory.createEtchedBorder(EtchedBorder.LOWERED));
    add(createToolButton(menuBar.getJinhuoItem()));    //添加指定的工具栏按钮
    add(createToolButton(menuBar.getXiaoshou_danItem()));
    add(createToolButton(menuBar.getKucun_pandianItem()));
    add(createToolButton(menuBar.getJiage_tiaozhengItem()));
    add(createToolButton(menuBar.getShangpin_chaxunItem()));
    add(createToolButton(menuBar.getShangpin_guanliItem()));
    add(createToolButton(menuBar.getKehu_guanliItem()));
    add(createToolButton(menuBar.getGys_guanliItem()));
```

```
        add(createToolButton(menuBar.getExitItem()));
    }
```

另外还定义了 createToolButton 方法来创建工具栏按钮，该方法实现了高度的代码重用，只要将相应的菜单项作为参数传递给这个方法就可以自动创建新的工具栏按钮。关键代码如下：

```
private JButton createToolButton(final JMenuItem item) {
    JButton button = new JButton();                    //创建按钮
    button.setText(item.getText());                    //设置按钮名称
    button.setToolTipText(item.getText());             //设置按钮提示文本
    button.setIcon(item.getIcon());                    //设置按钮图标
    button.setFocusable(false);
    //添加按钮动作监听器
    button.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent e) {
            item.doClick();                            //执行按钮的单击动作
        }
    });
    return button;
}
```

20.9.3　设计状态栏

本系统的状态栏显示了当前选择的功能窗体、登录用户名、当前日期和本系统所属公司，即版权所有者等信息，如图 20.12 所示。

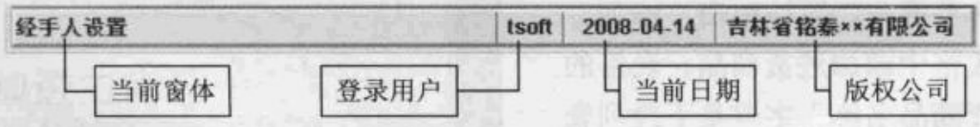


图 20.12　状态栏界面

该状态栏是由 JPanel 面板、JLabel 标签和 JSeparator 分隔条组件组成的。相关组件的说明如表 20.9 所示。

表 20.9　状态栏相关组件说明

组 件 类 型	组件 ID	主要属性设置	说　明
JLabel	stateLabel	将 text 属性设置为“当前没有选定窗体”	显示当前窗体信息
	czyStateLabel	无	显示登录用户名信息
	nowDateLabel	无	显示当前日期信息
	nameLabel	设置 text 属性为“吉林省铭泰××有限公司”	显示版权公司信息
JPanel	statePanel	设置 layout 属性为 GridBagLayout 布局管理器。constraint 属性为 South	状态栏面板
JSeparator	jSeparator	设置 Orientation 为 JSeparator.VERTICAL	分隔符
	jSeparator2	设置 Orientation 为 JSeparator.VERTICAL	分隔符

主窗体中的 getStateLabel 方法用于初始化当前窗体状态标签 stateLabel，如果该标签已经初始化将直接返回标签组件的对象。修改该方法的权限修饰符为 public，使其他类可以访问该标签，从而改变标签内容。关键代码如下：

```
public JLabel getStateLabel() {                    //获取当前窗体状态标签
    if (stateLabel == null) {
        stateLabel = new JLabel();
        stateLabel.setText("当前没有选定窗体");
    }
    return stateLabel;
}
```

20.10 进货单模块设计

视频讲解：光盘\TM\lx\20\进货单模块设计.mp4

本模块使用的数据表：tb_ruku_main、tb_ruku_detail、tb_kucun、tb_gysinfo、tb_spinfo、tb_jsr

进货单模块负责添加企业的进货信息，它根据进货人员提供的单据，将采购商品的名称、编号、产地、规格、单价和数量等信息记录到数据库的库存表中。进货单窗体界面如图 20.13 所示。

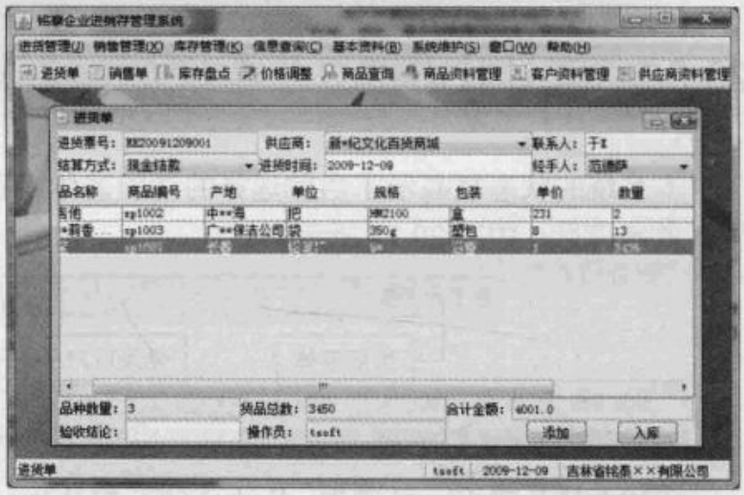


图 20.13 进货单窗体界面

在进货单窗体界面中，可以单击“添加”按钮向进货单的表格中添加进货商品。表格的第一列，也就是“商品名称”字段是下拉列表框组件，其内容根据“供应商”下拉列表框而定，可以通过该组件选择商品名称，其他表格字段（商品信息）会自动添加。

说明

“进货管理”菜单中包含“进货单”和“进货退货”两个菜单项，由于实现方式基本相同，所以这里以进货单模块为例进行讲解。

20.10.1 设计进货单窗体

在 Eclipse 中选择“文件”/“新建”/“其他”命令，在弹出的“新建”对话框中，选择 WindowBuilder/Swing Designer 节点，创建 JInternalFrame 内部窗体类，命名为 JinHuoDan_IFrame。该窗体中所用到的关键组件如

表 20.10 所示。

表 20.10　进货单窗体用到的关键组件

组 件 类 型	组件 ID	主要属性设置	用　途
JTextField	idField	设置 editable 属性为 FALSE	显示进货单编号
	lxrField	无	显示联系人
	jhsjField	无	显示进货时间信息
	czyField	设置 editable 属性为 FALSE	显示操作员
	pzslField	设置 editable 属性为 FALSE	显示品种数量信息
	hpzsField	设置 editable 属性为 FALSE	货品总数
	hjjeField	设置 editable 属性为 FALSE	合计金额
	ysjlField	无	验收结论
JComboBox	jsfsComboBox	无	选择结算方式
	spComboBox	无	选择商品
	gysComboBox	无	选择供应商
	jsrComboBox	无	选择经手人
JButton	tjButton	设置 text 属性为“添加”	“添加”按钮
	rukuButton	设置 text 属性为“入库”	“入库”按钮
JPanel	jContentPane	设置 BorderLayout 布局管理器	内容面板
	topPanel	设置 GridBagLayout 布局管理器	上层面板
	bottomPanel	设置 GridBagLayout 布局管理器	下层面板
JScrollPane	tablePane	无	表格的滚动面板
JTable	table	设置 autoResizeMode 属性为 Off	显示商品列表的表格

20.10.2　添加进货商品

在进货单窗体中单击“添加”按钮，会在表格中添加一个空行，可以在该空行的第一个字段选择商品名称，其他的字段信息会根据选择的商品自动填充。这就需要为“添加”按钮编写 ActionListener 动作监听器，在该监听器中实现相应的操作。“添加”按钮的初始化由 getTjButton 方法完成，该方法在初始化“添加”按钮时，为按钮添加了动作事件监听器。关键代码如下：

```
private JButton getTjButton() {
    if (tjButton == null) {
        tjButton = new JButton();
        tjButton.setText("添加");
        tjButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                //初始化票号
                java.sql.Date date = new java.sql.Date(jhsjDate.getTime());
                jhsjField.setText(date.toString());
                String maxId = Dao.getRuKuMainMaxId(date);
                idField.setText(maxId);
```

```
                stopTableCellEditing();    //结束表格中编辑单元
                // 如果表格中不包含空行，就添加新行
                for (int i = 0; i < table.getRowCount(); i++) {
                    TbSpinfo info = (TbSpinfo) table.getValueAt(i, 0);
                }
                DefaultTableModel model = (DefaultTableModel) table.getModel();
                model.addRow(new Vector());
                initSpBox();
            }
        });
    }
    return tjButton;
}
```

为表格添加新的空行

在添加空行后，还需要调用 initSpBox 方法初始化商品下拉框。在该方法中，首先获取指定供应商提供的商品信息，并清空商品下拉框，然后添加一个空的列表项，再通过 for 循环将表格中已经添加的商品 ID 保存到 List 集合中，最后将没有添加到表格中的商品加到商品下拉框中。initSpBox 方法的具体代码如下：

```
private void initSpBox() {
    List<String> list = new ArrayList<>();
    ResultSet set = Dao.query("select * from tb_spinfo where gysName='"
            + gysComboBox.getSelectedItem() + "'");
    spComboBox.removeAllItems();
    spComboBox.addItem(new TbSpinfo());
    for (int i = 0; table != null && i < table.getRowCount(); i++) {
        TbSpinfo tmpInfo = (TbSpinfo) table.getValueAt(i, 0);
        if (tmpInfo != null && tmpInfo.getId() != null){
            list.add(tmpInfo.getId());
        }
    }

    try {
        while (set.next()) {
            TbSpinfo spinfo = new TbSpinfo();
            spinfo.setId(set.getString("id").trim());
            //如果表格中已存在同样商品，商品下拉框中就不再添加该商品
            if (list.contains(spinfo.getId()))
                continue;
            spinfo.setSpname(set.getString("spname").trim());
            spinfo.setCd(set.getString("cd").trim());
            spinfo.setJc(set.getString("jc").trim());
            spinfo.setDw(set.getString("dw").trim());
            spinfo.setGg(set.getString("gg").trim());
            spinfo.setBz(set.getString("bz").trim());
            spinfo.setPh(set.getString("ph").trim());
            spinfo.setPzwh(set.getString("pzwh").trim());
            spinfo.setMemo(set.getString("memo").trim());
            spinfo.setGysname(set.getString("gysname").trim());
            spComboBox.addItem(spinfo);
```

```
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
```

20.10.3　进货统计

在 bottomPanel 面板中布置了多个文本框，用于统计品种数量、货品总数、合计金额等商品信息。在添加进货商品之后，要实现商品信息的自动统计，就要在 table 表格的 PropertyChangeListener 事件监听器中编写统计代码。这里将统计代码编写为 ComputeInfo 方法，然后在事件监听器中调用。为表格添加事件监听器的关键代码如下：

```
//添加匿名的事件监听器
table.addPropertyChangeListener(new PropertyChangeListener() {
    public void propertyChange(java.beans.PropertyChangeEvent e) {
        if ((e.getPropertyName().equals("tableCellEditor"))) {      //判断事件类型
            ComputeInfo();                                          //执行统计方法
        }
    }
});
```

当 table 表格发生属性改变事件时，事件监听器首先会检测发生的事件类型，也就是判断发生了哪种更改属性的事件，如果事件类型是 tableCellEditor，则说明属于表格编辑事件，这时应该针对表格的修改事件去调用 ComputeInfo 方法执行商品进货的统计业务并将结果显示在相应的组件上。ComputeInfo 方法的关键代码如下：

```
/**
 * @author 李钟尉 <br>
 *          事件处理器，该处理器用于计算货品总数、合计金额等信息。
 */
private final void ComputeInfo() {
        //计算代码
        int rows = table.getRowCount();
        int count = 0;
        double money = 0.0;
        //计算品种数量
        TbSpinfo column = null;
        Object valueAt = table.getValueAt(rows - 1, 0);
        if(!(valueAt instanceof TbSpinfo))
          return;
        if (rows > 0)
          column = (TbSpinfo) valueAt;
        if (rows > 0 && (column == null || column.getId().isEmpty()))
            rows--;
        //计算货品总数和合计金额
        for (int i = 0; i < rows; i++) {
```

```
            String column7 = (String) table.getValueAt(i, 7);
            String column6 = (String) table.getValueAt(i, 6);
            int c7 = (column7 == null || column7.isEmpty()) ? 0 : Integer
                    .parseInt(column7);
            float c6 = (column6 == null || column6.isEmpty()) ? 0 : Float
                    .parseFloat(column6);
            count += c7;
            money += c6 * c7;
        }
        pzslField.setText(rows + "");
        hpzsField.setText(count + "");
        hjjeField.setText(money + "");
}
```

20.10.4 商品入库

在添加了进货单中的所有商品后，单击“入库”按钮可以将这些商品添加到数据库中。这需要在“入库”按钮的初始化方法中，为按钮添加 ActionListener 动作监听器，在监听器中实现商品入库的业务逻辑。getRukuButton 方法是“入库”按钮的初始化方法，该方法将判断“入库”按钮对象是否初始化，如果已经初始化就直接将按钮对象返回给方法的调用者，否则先对按钮进行初始化，然后返回该按钮对象。在初始化“入库”按钮的过程中为按钮添加了动作事件监听器，在该事件监听器中首先调用 stopTableCellEditing 方法停止正在编辑的表格单元，然后获取进货单的品种数量、结算方式、合计金额、经手人、操作员、进货票号、验收结论等信息，并对关键信息进行判断，防止用户忘记填写这些关键信息。最后，创建进货主表的模型对象、进货详细表的模型对象和库存表的模型对象，使用进货单窗体中的信息初始化这些模型对象，并把它们通过 Dao 公共类的 insertRukuInfo 方法保存到数据库中。程序关键代码如下：

```
private JButton getRukuButton() {
    if (rukuButton == null) {
        rukuButton = new JButton();
        rukuButton.setText("入库");
        rukuButton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent e) {
                stopTableCellEditing();                                    //结束表格中没有编写的单元
                String pzsStr = pzslField.getText();                       //品种数量
                String jeStr = hjjeField.getText();                        //合计金额
                //结算方式
                String jsfsStr = jsfsComboBox.getSelectedItem().toString();
                String jsrStr = jsrComboBox.getSelectedItem() + "";        //经手人
                String czyStr = jsrComboBox.getSelectedItem() + "";        //操作员
                String rkDate = jhsjField.getText();                       //入库时间
                String ysjlStr = ysjlField.getText().trim();               //验收结论
                String id = idField.getText();                             //票号
                //供应商名称
                String gysName = gysComboBox.getSelectedItem() + "";
                if (jsrStr == null || jsrStr.isEmpty()) {
```

收集进货单的相关信息

```
                JOptionPane.showMessageDialog(JinHuoDan_IFrame.this,
                        "请填写经手人");
                return;
            }
            if (ysjlStr == null || ysjlStr.isEmpty()) {
                JOptionPane.showMessageDialog(JinHuoDan_IFrame.this,
                        "填写验收结论");
                return;
            }
            if (table.getRowCount() <= 0) {
                JOptionPane.showMessageDialog(JinHuoDan_IFrame.this,
                        "添加入库商品");
                return;
            }
            TbRukuMain ruMain = new TbRukuMain(id, pzsStr, jeStr,
                    ysjlStr, gysName, rkDate, czyStr, jsrStr, jsfsStr);
            Set<TbRukuDetail> set = ruMain.getTabRukuDetails();
            int rows = table.getRowCount();
            for (int i = 0; i < rows; i++) {
                TbSpinfo spinfo = (TbSpinfo) table.getValueAt(i, 0);
                if (spinfo == null || spinfo.getId() == null
                        || spinfo.getId().isEmpty())
                    continue;
                String djStr = (String) table.getValueAt(i, 6);
                String slStr = (String) table.getValueAt(i, 7);
                Double dj = Double.valueOf(djStr);
                Integer sl = Integer.valueOf(slStr);
                TbRukuDetail detail = new TbRukuDetail();
                detail.setTabSpinfo(spinfo.getId());
                detail.setTabRukuMain(ruMain.getRkId());
                detail.setDj(dj);
                detail.setSl(sl);
                set.add(detail);
            }
            //将数据模型保存到数据库中
            boolean rs = Dao.insertRukuInfo(ruMain);
            if (rs) {
                JOptionPane.showMessageDialog(JinHuoDan_IFrame.this,"入库完成");
                //移除全部表格行
                DefaultTableModel model = (DefaultTableModel) table.getModel();
                for(int i=model.getRowCount()-1;i>=0;i--){
                    model.removeRow(i);                    //移除指定行
                }
                pzslField.setText("0");
                hpzsField.setText("0");
                hjjeField.setText("0");
            }
        }
    });
```

创建并初始化进货详细表数据模型

```
        }
        return rukuButton;
    }
```

说明

addActionListener 方法的参数使用了匿名类实现动作监听器接口，这是 Swing 的事件处理最常用的方式，读者应该熟练掌握。

20.11 销售单模块设计

视频讲解：光盘\TM\lx\20\销售单模块设计.mp4

本模块使用的数据表：tb_sell_main、tb_sell_detail、tb_kucun、tb_khinfo、tb_spinfo、tb_jsr

商品销售是进销存管理中的重要环节之一，进货商品在入库之后就可以开始销售了。销售单模块主要负责根据经手人的销售单据，操作进销存管理系统的库存商品和记录销售信息，方便以后查询和统计。销售单窗体的运行效果如图 20.14 所示。

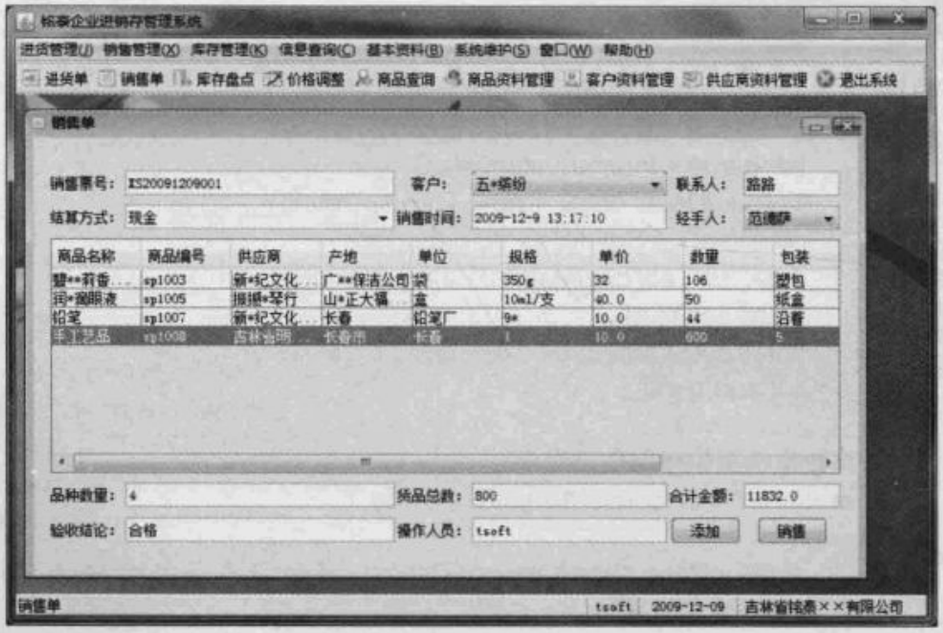


图 20.14 销售单窗体界面

20.11.1 设计销售单窗体

创建 JInternalFrame 内部窗体类，命名为 XiaoShouDan。该窗体主要用于处理商品销售的业务逻辑，它所用到的关键组件如表 20.11 所示。

表 20.11　进货单窗体用到的关键组件

组 件 类 型	组件 ID	主要属性设置	用　途
JTextField	sellDate	设置 Focusable 属性为 FALSE	显示记录销售单的日期和时间
	lian	无	显示联系人
	piaoHao	无	显示销售票号
JTextField	pzs	设置 Focusable 属性为 FALSE	显示品种数量
	hpzs	设置 Focusable 属性为 FALSE	显示货品总数
	hjje	设置 Focusable 属性为 FALSE	显示合计金额信息
	ysjl	设置 Focusable 属性为 FALSE	输入验收结论
	czy	无	显示操作员
JComboBox	kehu	无	选择客户
	jsr	无	选择经手人
	jsfs	无	选择结算方式
	sp	无	选择商品
JButton	tjButton	设置 text 属性为“添加”	“添加”按钮
	sellButton	设置 text 属性为“销售”	“销售”按钮
JScrollPane	tablePane	无	表格的滚动面板
JTable	table	设置 autoResizeMode 属性为 Off	显示商品列表的表格

20.11.2　添加销售商品

在销售单窗体中单击“添加”按钮，将向 table 表格中添加新的空行，操作员可以在空行的第一列字段的商品下拉列表框中选择销售的商品，这个下拉列表框和进货单窗体的不同，它不是根据供应商字段确定选择框内容，而是包含数据库中所有可以销售的商品。要实现添加销售商品功能，需要为“添加”按钮添加动作监听器，在监听器中实现相应的业务逻辑。关键代码如下：

```
JButton tjButton = new JButton("添加");                    //创建“添加”按钮
tjButton.addActionListener(new ActionListener() {          //添加
    public void actionPerformed(ActionEvent e) {
        initPiaoHao();                                     //初始化票号
        stopTableCellEditing();                            //结束表格中正在编辑的单元
        //如果表格中不包含空行，就再添加新行
        for (int i = 0; i < table.getRowCount(); i++) {
            TbSpinfo info = (TbSpinfo) table.getValueAt(i, 0);
            if (table.getValueAt(i, 0) == null)
                return;
        }
        DefaultTableModel model = (DefaultTableModel) table.getModel();
        model.addRow(new Vector());                        //添加新的空行
    }
});
```

在该监听器中调用了 initPiaoHao 方法初始化销售票号，该票号就是销售单在数据库中的 id 编号。

initPiaoHao 方法首先创建 java.sql 包中 Date 类的对象，该对象包含当前日期；然后调用 Dao 类的 getSellMainMaxId 方法获取数据库销售主表中的最大 ID 编号；最后，将该 ID 编号更新到 piaoHao 文本框中。

```
private void initPiaoHao() {
    //获取 Date 对象
    Date date = new Date(System.currentTimeMillis());
    String maxId = Dao.getSellMainMaxId(date);          //获取票号
    piaoHao.setText(maxId);                             //更新界面组件
}
```

20.11.3 销售统计

和进货单的统计功能类似，销售单也需要统计功能，统计的内容包括货品数量、品种数量、合计金额等信息，实现方式也是通过 table 表格的事件监听器来处理相应的统计业务，但是销售单窗体使用的不是 PropertyChangeListener 属性改变事件监听器，而是使用 ContainerListener 容器监听器。关键代码如下：

```
table = new JTable();                                   //初始化表格对象
table.setAutoResizeMode(JTable.AUTO_RESIZE_OFF); //取消自动调整列宽
//添加 ContainerListener 容器监听器
table.addContainerListener(new computeInfo());
```

computeInfo 类是销售单窗体的内部类，该类实现 ContainerListener 接口成为容器监听器，该监听器将 table 表格视为容器，当表格添加新行和删除行时，将触发 ContainerEvent 容器事件，监听器将对该事件进行相应的业务处理，完成本次销售信息的统计。关键代码如下：

```
//在事件中计算品种数量、货品总数、合计金额
private final class computeInfo implements ContainerListener {
    public void componentRemoved(ContainerEvent e) {
        clearEmptyRow();                                //清除空行
        int rows = table.getRowCount();                 //品种数量
        int count = 0;                                  //货品总数
        double money = 0.0;                             //合计金额
        TbSpinfo column = null;
        if (rows > 0)
            column = (TbSpinfo) table.getValueAt(rows - 1, 0);
        if (rows > 0 && (column == null || column.getId().isEmpty()))
            rows--;
        for (int i = 0; i < rows; i++) {                //遍历表格，统计销售信息
            String column7 = (String) table.getValueAt(i, 7);
            String column6 = (String) table.getValueAt(i, 6);
            int c7 = (column7 == null || column7.isEmpty()) ? 0 : Integer
                    .valueOf(column7);
            Double c6 = (column6 == null || column6.isEmpty()) ? 0
                    : Double.valueOf(column6);
```

```
                count += c7;
                money += c6 * c7;
            }
            pzs.setText(rows + "");                          //更新“品种数量”文本框中的内容
            hpzs.setText(count + "");                        //更新“货品总数”文本框中的内容
            hjje.setText(money + "");                        //更新“合计金额”文本框中的内容
        }
        public void componentAdded(ContainerEvent e) {
        }
}
```

20.11.4　商品销售

在销售单窗体中添加完销售商品之后，单击“销售”按钮，将完成本次销售单的销售业务。系统会记录本次销售信息，并从库存表中扣除销售的商品数量。这些业务处理都是在“销售”按钮的动作监听器中完成的，该监听器需要获取销售单窗体中的所有销售信息和商品信息，将所有商品信息封装为销售明细表的模型对象，并将这些模型对象放到一个集合中，然后调用 Dao 公共类的 insertSellInfo 方法将该集合与销售主表的模型对象保存到数据库中。程序关键代码如下：

```
//单击“销售”按钮保存进货信息
JButton sellButton = new JButton("销售");
sellButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        stopTableCellEditing();                              //结束表格中正在编辑的单元
        clearEmptyRow();                                     //清除空行
        String hpzsStr = hpzs.getText();                     //货品总数
        String pzsStr = pzs.getText();                       //品种数量
        String jeStr = hjje.getText();                       //合计金额
        String jsfsStr = jsfs.getSelectedItem().toString();  //结算方式
        String jsrStr = jsr.getSelectedItem() + "";          //经手人
        String czyStr = czy.getText();                       //操作员
        String rkDate = jhsjDate.toLocaleString();           //销售时间
        String ysjlStr = ysjl.getText().trim();              //验收结论
        String id = piaoHao.getText                          //票号
        String kehuName = kehu.getSelectedItem().toString(); //供应商名称
        if (jsrStr == null || jsrStr.isEmpty()) {
            JOptionPane.showMessageDialog(XiaoShouDan.this, "请填写经手人");
            return;
        }
        if (ysjlStr == null || ysjlStr.isEmpty()) {
            JOptionPane.showMessageDialog(XiaoShouDan.this, "填写验收结论");
            return;
        }
        if (table.getRowCount() <= 0) {
            JOptionPane.showMessageDialog(XiaoShouDan.this, "添加销售商品");
            return;
        }
```

```
            //创建销售主表的模型对象
            TbSellMain sellMain = new TbSellMain(id, pzsStr, jeStr,
                    ysjlStr, kehuName, rkDate, czyStr, jsrStr, jsfsStr);
            //获取销售明细表的集合
            Set<TbSellDetail> set = sellMain.getTbSellDetails();
            int rows = table.getRowCount();
            for (int i = 0; i < rows; i++) {                          //初始化销售明细表集合
                //创建销售明细表模型对象
                TbSpinfo spinfo = (TbSpinfo) table.getValueAt(i, 0);
                //初始化销售明细表模型
                String djStr = (String) table.getValueAt(i, 6);
                String slStr = (String) table.getValueAt(i, 7);
                Double dj = Double.valueOf(djStr);
                Integer sl = Integer.valueOf(slStr);
                TbSellDetail detail = new TbSellDetail();
                detail.setSpid(spinfo.getId());
                detail.setTbSellMain(sellMain.getSellId());
                detail.setDj(dj);
                detail.setSl(sl);
                set.add(detail);
            }
            //调用 Dao 类的 insertSellInfo()方法
            boolean rs = Dao.insertSellInfo(sellMain);
            if (rs) {
                JOptionPane.showMessageDialog(XiaoShouDan.this, "销售完成");
                DefaultTableModel dftm = new DefaultTableModel();
                table.setModel(dftm);
                initTable();
                pzs.setText("0");
                hpzs.setText("0");
                hjje.setText("0");
            }
        }
});
```

20.12 库存盘点模块设计

视频讲解：光盘\TM\lx\20\库存盘点模块设计.mp4

本模块使用的数据表：tb_kucun、tb_spinfo

库存盘点模块主要负责计算库存管理人员的商品盘点数量和库存数量的损益。程序界面将提示当前日期和库存商品的品种数量，并在表格中显示所有库存商品，在表格的“盘点数量”一列中输入相应商品的盘点数量，“损益数量”字段会自动计算该商品的剩余商品数量，如果该数量为正数，则说明库存数量多于盘点数量，如图 20.15 所示。

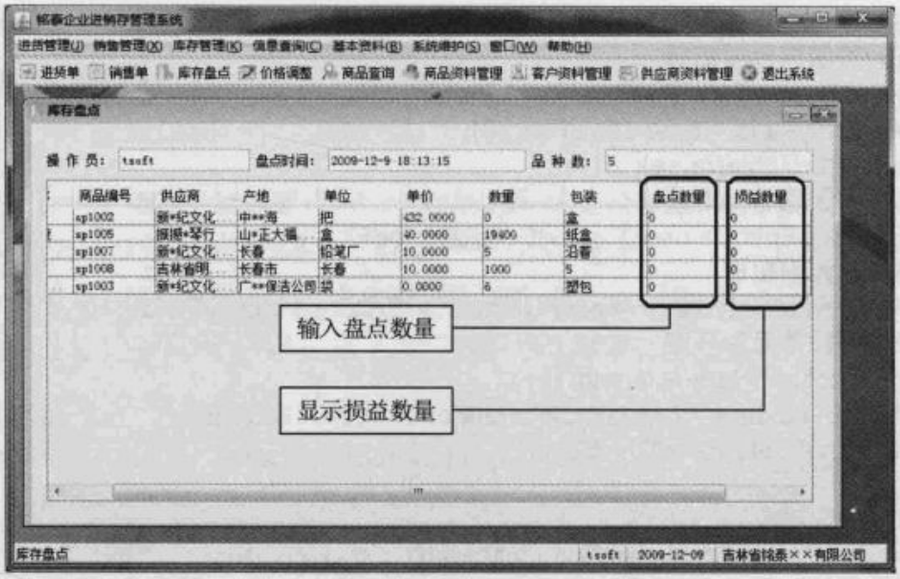


图 20.15　库存盘点窗体界面

20.12.1　设计库存盘点窗体

创建 JInternalFrame 内部窗体类，命名为 KuCunPanDian。该窗体主要用于计算库存管理中的损益结果，它所用到的关键组件如表 20.12 所示。

表 20.12　进货单窗体用到的关键组件

组 件 类 型	组件 ID	主要属性设置	用　途
JTextField	pdsj	设置 Focusable 属性为 FALSE	显示盘点时间
	pzs	无	显示品种数
	czy	无	显示操作员
JScrollPane	tablePane	无	表格的滚动面板
JTable	table	设置 autoResizeMode 属性为 Off	显示商品列表的表格

20.12.2　读取库存商品

本模块窗体的商品表格 table 组件用于显示库存中的所有商品信息，这需要在 initTable 方法中初始化表格字段名，并调用 Dao 类的 getKucunInfos 方法读取库存数据中的所有商品列表，添加到 table 商品表格组件中。关键代码如下：

```
private void initTable() {
    //定义表格字段名数组
    String[] columnNames = { "商品名称", "商品编号", "供应商", "产地", "单位",
            "规格", "单价","数量", "包装", "盘点数量", "损益数量" };
    //获取表格模型
    DefaultTableModel tableModel = (DefaultTableModel) table.getModel();
    tableModel.setColumnIdentifiers(columnNames);     //设置表格字段名
    //设置盘点字段只接收数字输入
    final JTextField pdField = new JTextField(0);          //创建盘点文本框
```

```
        pdField.setEditable(false);
        //添加文本框的按键事件监听器
        pdField.addKeyListener(new PanDianKeyAdapter(pdField));
        JTextField readOnlyField = new JTextField(0);
        readOnlyField.setEditable(false);
        //使用盘点文本框创建单元编辑器
        DefaultCellEditor pdEditor = new DefaultCellEditor(pdField);
        //创建其他表格单元编辑器
        DefaultCellEditor readOnlyEditor = new DefaultCellEditor(readOnlyField);
        //设置所有表格编辑单元为只读
        for (int i = 0; i < columnNames.length; i++) {
            TableColumn column = table.getColumnModel().getColumn(i);
            column.setCellEditor(readOnlyEditor);
        }
        //获取盘点数量字段对象
        TableColumn pdColumn = table.getColumnModel().getColumn(9);
        //获取损益数量字段对象
        TableColumn syColumn = table.getColumnModel().getColumn(10);
        //设置盘点数量字段使用盘点文本框作为编辑器
        pdColumn.setCellEditor(pdEditor);
        //设置损益数量字段使用损益文本框作为编辑器
        syColumn.setCellEditor(readOnlyEditor);
        //下面的代码用于初始化表格内容
        List kcInfos = Dao.getKucunInfos();                    //获取库存商品列表
        for (int i = 0; i < kcInfos.size(); i++) {             //遍历商品列表
            List info = (List) kcInfos.get(i);                 //获取单个商品信息
            Item item = new Item();                            //创建 Item 公共类的对象
            item.setId((String) info.get(0));                  //封装商品信息为 Item 对象
            item.setName((String) info.get(1));
            TbSpinfo spinfo = Dao.getSpInfo(item);             //获取该商品的数据模型对象
            Object[] row = new Object[columnNames.length];
            //判断商品 ID 编号
            if (spinfo.getId() != null && !spinfo.getId().isEmpty()) {
                row[0] = spinfo.getSpname();
                row[1] = spinfo.getId();
                row[2] = spinfo.getGysname();
                row[3] = spinfo.getCd();
                row[4] = spinfo.getDw();
                row[5] = spinfo.getGg();
                row[6] = info.get(2).toString();
                row[7] = info.get(3).toString();
                row[8] = spinfo.getBz();
                row[9] = 0;
                row[10] = 0;
                tableModel.addRow(row);                        //为 table 表格添加一行
                String pzsStr = pzs.getText();                 //计算品种数
                int pzsInt = Integer.parseInt(pzsStr);
                pzsInt++;
                pzs.setText(pzsInt + "");
            }
        }
    }
```

将库存商品添加到表格中

20.12.3　统计损益数量

商品表格组件需要在用户输入盘点数量时，自动计算并更新损益单元格的内容，也就是使用库存商品实际数量减去用户输入的盘点数量。实现自动计算功能最好的方式，就是为表格组件的“盘点数量”编辑器的编辑组件添加按键监听器，使用该按键监听器可以限制用户只能输入数字信息，同时还可以在按键事件发生时进行损益统计。该监听器的关键代码如下：

```
//盘点字段的按键监听器
private class PanDianKeyAdapter extends KeyAdapter {
    private final JTextField field;
    private PanDianKeyAdapter(JTextField field) {
        this.field = field;
    }
    public void keyTyped(KeyEvent e) {                    //限制盘点数量只能输入数字字符
        if (("0123456789" + (char) 8).indexOf(e.getKeyChar() + "") < 0) {
            e.consume();
        }
        field.setEditable(true);
    }
    public void keyReleased(KeyEvent e) {                 //计算损益数量
        String pdStr = field.getText();                   //获取盘点数量
        String kcStr = "0";
        int row = table.getSelectedRow();                 //获取 table 组件的当前选择行
        if (row >= 0) {
            //获取该行的第 7 列单元内容，即库存数量
            kcStr = (String) table.getValueAt(row, 7);
        }
        try {
            int pdNum = Integer.parseInt(pdStr);          //获取盘点数量
            int kcNum = Integer.parseInt(kcStr);          //获取库存数量
            if (row >= 0) {
                //计算并更新损益单元格的内容
                table.setValueAt(kcNum - pdNum, row, 10);
            }
            if (e.getKeyChar() != 8)
                field.setEditable(false);
        } catch (NumberFormatException e1) {
            field.setText("0");
        }
    }
}
```

20.13 数据库备份与恢复模块设计

视频讲解：光盘\TM\lx\20\数据库备份与恢复模块设计.mp4

数据库备份与恢复模块可以增强系统安全性。及时备份系统数据，如果发生意外可以恢复最近时间段的数据库内容，将损失降低到最小程度。铭泰企业进销存管理系统数据库备份与恢复模块的窗体界面如图 20.16 所示。

图 20.16 数据库备份与恢复窗体

20.13.1 设计窗体

创建 JInternalFrame 内部窗体类，命名为 BackupAndRestore。该窗体主要用于备份和恢 复系统的数据库文件，它所用到的关键组件如表 20.13 所示。

表 20.13 数据库备份与恢复窗体用到的关键组件

组 件 类 型	组件 ID	主要属性设置	用 途
JTextField	backupTextField	无	备份数据库的文件路径文本框
	restoreTextField	无	恢复数据库的文件路径文本框
JPanel	backupPanel	设置 border 属性使用 Title 边框，边框的标题设置为“数据库备份” 设置 layout 属性使用 GridBagLayout 布局管理器	备份功能的组件面板
	restorePanel	设置 border 属性使用 Title 边框，边框的标题设置为“数据库恢复” 设置 layout 属性使用 GridBagLayout 布局管理器	恢复功能的组件面板
JButton	backupButton	设置 Text 属性为“备份(K)” 设置 mnemonic 属性为“VK_K”	“备份”按钮
	browseButton1	设置 Text 属性为“浏览(B)” 设置 mnemonic 属性为“VK_B”	备份功能的“浏览”按钮
	browseButton2	设置 Text 属性为“浏览(W)” 设置 mnemonic 属性为“VK_W”	恢复功能的“浏览”按钮
	restoreButton	设置 Text 属性为“恢复(R)” 设置 mnemonic 属性为“VK_R”	“恢复”按钮

20.13.2　文件浏览

数据库的备份和恢复功能都需要使用“浏览”按钮选择数据库文件的位置。本模块的窗体界面中有两个“浏览”按钮，分别用于选择数据库备份文件和数据库恢复文件的位置。这两个“浏览”按钮的 ActionListener 动作监听器实现方法是相同的，监听器中通过 JFileChooser 文件选择器组件打开文件选择对话框，选择数据库备份文件的位置。由于实现方法相同，这里以数据库恢复的“浏览”按钮为例进行讲解。程序关键代码如下：

```
private JButton getBrowseButton2() {                          //“浏览”按钮的初始化方法
    if (browseButton2 == null) {                              //如果按钮未初始化
        browseButton2 = new JButton();                        //创建按钮
        browseButton2.setText("浏览(W)……");                    //设置按钮文本
        browseButton2.setMnemonic(KeyEvent.VK_W);             //设置按钮助记符
        //添加动作监听器
        browseButton2.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                //创建文件选择器组件
                JFileChooser dirChooser=new JFileChooser(".");
                //打开文件选择对话框
                int option=dirChooser.showOpenDialog(BackupAndRestore.this);
                if(option==JFileChooser.APPROVE_OPTION){
                    //获取用户选择文件
                    File selFile = dirChooser.getSelectedFile();
                    //设置文本框的文件路径
                    restoreTextField.setText(selFile.getAbsolutePath());
                }
            }
        });
    }
    return browseButton2;                                     //返回初始化的按钮组件
}
```

20.13.3　备份数据库

通过“浏览”按钮选择或者直接在文本框中输入数据库备份文件的路径之后单击“备份”按钮，将系统当前数据库内容备份到文件中，为数据库预留多个备份。“备份”按钮的动作监听器将通过 Dao 类的 restoreOrBackup 方法执行备份数据库的 SQL 语句，如果在此期间程序抛出异常，将以对话框的方式提示用户错误信息，否则提示“备份成功”。初始化“备份”按钮的关键代码如下：

```
private JButton getBackupButton() {                           //“备份”按钮的初始化方法
    if (backupButton == null) {                               //判断“备份”按钮是否未初始化
        backupButton = new JButton();                         //创建按钮组件
        backupButton.setText("备份(K)");                       //设置按钮文本
```

```
        backupButton.setMnemonic(KeyEvent.VK_K);                    //设置按钮助记符
        //添加动作事件监听器
        backupButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String path = backupTextField.getText();            //获取备份文件的路径
                File backupFile = new File(path);                   //创建 File 文件对象
                //声明备份数据库的 SQL 语句
                String sql = "backup database db_Database to DISK='"
                        + backupFile.getAbsolutePath() + "'";
                try {
                    //调用 Dao 类的 restoreOrBackup( )方法
                    Dao.restoreOrBackup(sql);
                } catch (Exception e1) {
                    e1.printStackTrace();
                    String message = e1.getMessage();               //获取异常信息
                    int index = message.lastIndexOf(')');
                    message = message.substring(index + 1);
                    JOptionPane.showMessageDialog(BackupAndRestore.this,
                            message);                               //显示错误提示对话框
                    return;
                }
                JoptionPane                                         //显示对话框
                        .showMessageDialog(BackupAndRestore.this, "备份成功");
            }
        });
    }
    return backupButton;                                            //返回初始化后的组件
}
```

20.13.4 恢复数据库

如果系统由于任何不可避免的原因导致程序无法运行，或者数据库系统损坏，可以在另一台计算机上安装铭泰企业进销存管理系统和数据库系统，然后在本模块的数据库恢复功能界面，通过“浏览”按钮选择备份在硬盘或其他移动设备上的数据库备份文件，并单击“恢复”按钮，就可以使程序恢复正常。“恢复”按钮的动作事件监听器将调用 Dao 类的 restoreOrBackup 方法执行还原数据库的 SQL 语句，如果在此期间程序抛出异常，将以对话框的方式提示用户错误信息，否则提示“恢复成功”。初始化“恢复”按钮的关键代码如下：

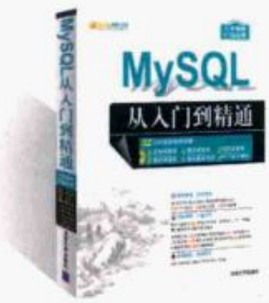
```
private JButton getRestoreButton() {                                //“恢复”按钮的初始化方法
    if (restoreButton == null) {                                    //判断是否未初始化
        restoreButton = new JButton();                              //创建按钮组件
        restoreButton.setText("恢复(R)");                           //设置按钮文本
        restoreButton.setMnemonic(KeyEvent.VK_R);                   //设置按钮助记符
        //添加动作事件监听器
        restoreButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
```

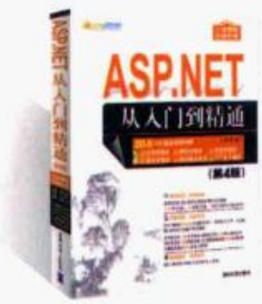
```
                    //获取数据库备份文件路径
                    String path = restoreTextField.getText();
                    if (path == null || path.isEmpty())
                        return;
                    File restoreFile = new File(path);                    //创建 File 文件对象
                    String sql = "restore database db_Database from DISK='"
                            + restoreFile.getAbsolutePath()
                            + "' WITH REPLACE";                           //声明 SQL 语句
                    try {
                        //调用 Dao 类的 restoreOrBackup()方法
                        Dao.restoreOrBackup(sql);
                    } catch (Exception e1) {
                        e1.printStackTrace();
                        String message = e1.getMessage();                 //获取异常信息
                        int index = message.lastIndexOf(')');
                        message = message.substring(index + 1);
                        JOptionPane.showMessageDialog(                    //显示错误提示对话框
                                BackupAndRestore.this, message);
                        return;
                    }
                    JOptionPane.showMessageDialog(
                            BackupAndRestore.this, "恢复成功");
                }
            });
        }
        return restoreButton;                                             //返回初始化的按钮组件
    }
```

20.14　小　　结

本章重点介绍了铭泰企业进销存管理系统中关键模块的开发过程、项目的运行和打包，以及程序开发中常见的问题与解决方法。通过对本章的学习，读者应该能够掌握 Swing 的各种常用布局管理器，熟练使用菜单栏和工具栏，掌握 JDBC 技术和项目的打包与发布。

软件开发视频大讲堂
RUANJIANKAIFASHIPINDAJIANGTANG





清华大学出版社数字出版网站

WQBook 书文局泉

www.wqbook.com

ISBN 978-7-302-45821-0

定价：79.80元

（附1DVD，含视频讲解、实例资源库、模块资源库、项目资源库、测试题库系统、面试资源库等）