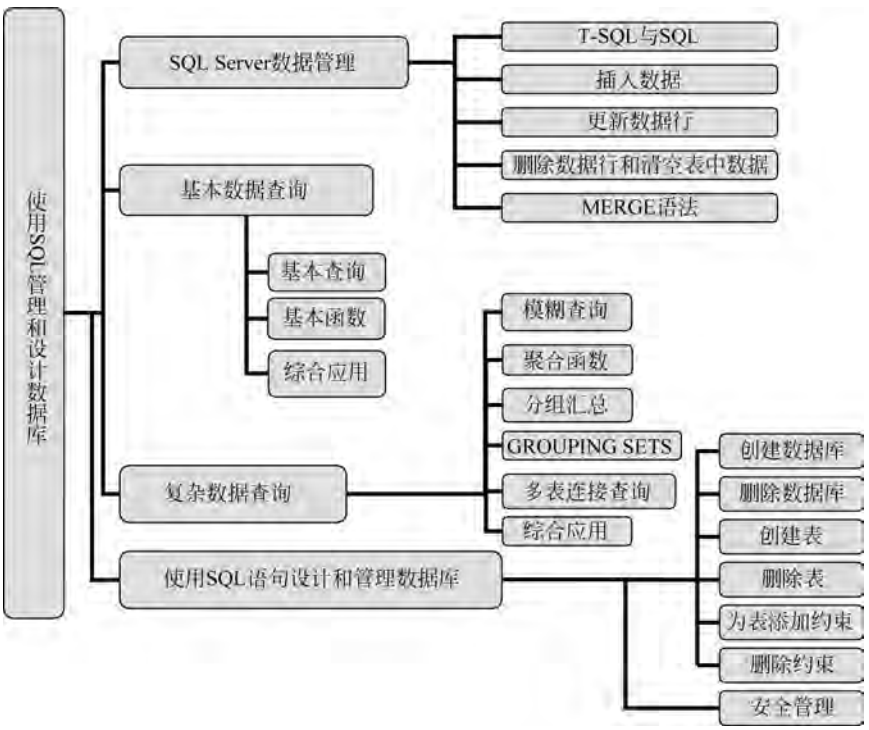
第5章 使用SQL管理和设计数据库

重点难点解析

典题例题

知识结构图



学习目标

了解 SQL Server 的数据管理功能

掌握基本数据查询方法

熟练掌握使用 SQL 语句设计和管理数据库

导入案例

使用软件产品提供的菜单服务就像到餐厅用餐，即便是第一次光临，也能在餐桌上摆放的菜单的帮助下找到自己想要的一切。软件语言如同顾客光顾久了的餐厅的外卖服务，只需拨通一个电话号码，就能吃到自己熟悉的餐点。如果与餐厅上下相处得十分融洽，或许顾

客还可以在特殊时候定制菜单，然后由外卖人员送上门。当然，不同的餐馆提供的服务不同，人性化服务的程度也各异。幸运的是，SQL Server 不仅提供了全面的菜单命令，帮助用户建设数据库、保护数据库、编辑数据库，还提供了更加灵活、全面的 Transact-SQL（T-SQL），帮助用户实现数据库中数据的编辑和各种效果的查询统计，以及数据库的建设和安全保护。本章主要介绍符合 SQL 标准部分的 T-SQL。

5.1 SQL Server 数据管理

SQL Server 数据管理包括对数据进行的增加、删除、修改和查找。数据管理可以由数据库软件通过菜单命令实现，但是数据库语言能够更加灵活地实现对数据的增、删、改、查。标准 SQL 是在关系数据库软件逐渐繁荣之后由国际标准化组织（ISO）提出的，并得到了几乎所有关系数据库软件的支持，虽然这些关系数据库的 SQL 各具特色，但是几乎都包含标准 SQL，并有所扩充，例如 SQL Server 提供的 Transact-SQL（简称 T-SQL）语言。

5.1.1 T-SQL 与 SQL

视频讲解

SQL 的全称是 Structured Query Language，即结构化查询语言，它是关系数据库的语言。SQL 包括数据定义语言（DDL）、数据操作语言（DML）和数据控制语言（DCL）三部分。

SQL 的作用主要有两大方面：一是可以替代企业管理器，灵活操作 SQL Server 数据库；二是作为嵌入式语言嵌入程序设计语言，使得应用程序可以使用 SQL 管理和访问数据库。实际上，从关系数据库的设计理论出发，关系数据库之所以称为关系数据库，主要原因是它采取二维表作为数据结构，满足数据操作和完整性约束；另外，它具有关系数据安全存储和管理的强大功能，与程序设计语言相独立，并为程序设计语言提供良好的访问接口。SQL 一方面可以帮助关系数据库自身灵活地操作数据库；另一方面可以帮助应用程序方便地管理数据。

用户可以使用 SQL 对数据库执行所有的操作，而且方便、灵活。SQL 多在应用程序中对数据库中的数据增、删、改、查时使用。

T-SQL 的全称是 Transact-SQL，它是 SQL 的加强版。T-SQL 的组成如下。

（1）DML：数据操作语言，用来查询、插入、删除和修改数据库中的数据，语句有 SELECT、INSERT、UPDATE、DELETE 等。

（2）DCL：数据控制语言，用来控制存取许可、存取权限等，语句有 GRANT、REVOKE 等。

（3）DDL：数据定义语言，用来建立数据库、数据库对象和定义其列，语句有 CREATE TABLE、DROP TABLE 等。

（4）变量说明、流程控制、功能函数：用来定义变量、判断、分支、循环等，函数包括日期函数、数学函数、字符函数、系统函数等。

在 SQL 语言中定义了运算符、通配符和逻辑运算符。SQL 中的运算符见表 5.1。

表 5.1　SQL 运算符

运　算　符	含　　义	运　算　符	含　　义	运　算　符	含　　义
=	等于	>=	大于或等于	!	非
>	大于	<=	小于或等于		
<	小于	<>	不等于		

SQL 中的通配符见表 5.2。

表 5.2　SQL 通配符

通　配　符	解　　释	示　　例
'_'	一个字符	A LIKE 'C_'
%	任意长度的字符串	B LIKE 'CO_%'
[ ]	括号中所指定范围内的一个字符	C LIKE '9W0[1-2]'
[^]	不在括号中所指定范围内的一个字符	D LIKE '%[A-D][^1-2]'

通配符通常与 LIKE 关键字一起使用，可以在检查约束中使用 LIKE，在后面的查询语句中还会经常使用到。

逻辑表达式见表 5.3。

表 5.3　逻辑表达式

逻辑运算符	说　　明	示　　例
AND	逻辑与	1 AND 1 =1; 1 AND 0 = 0; 0 AND 0 = 0;
OR	逻辑或	1 OR 1 = 1; 1OR 0 = 1; 0 OR 0 = 0;
NOT	逻辑非	NOT 1 = 0; NOT 0 = 1;

5.1.2　插入数据

视频讲解

1. 插入一条数据行

其语法如下，其中，中括号内可以省略。

```
INSERT [INTO] <表名> [列名] VALUES <值列表>
```

例如有表 student(sNo,sName,sAddress,sGrade,sEmail,sSex)，插入一条完整的记录的语句为：

```
INSERT INTO student(sNo,sName,sAddress,sGrade,sEmail,sSex)
VALUES('020110001','张黎','上海','2011','ZQC@Sohu.com',0)
```

2. 插入语句时的注意事项

插入语句时的注意事项如下。

(1) 每次插入一行数据，不可能只插入半行或者几列数据，因此插入的数据是否有效将按照整行的完整性要求来检验。

(2) 每个数据值的数据类型、精度和小数位数必须与相应的列匹配。

(3) 不能为标识列指定值，因为它的数字是自动增长的。

(4) 如果在设计表的时候就指定了某列不允许为空,则必须插入数据。

(5) 插入的数据项要符合检查约束的要求。

(6) 具有默认值的列,可以使用 DEFAULT(默认)关键字来代替插入的数值。

例:

```
INSERT INTO student(sName,sAddress,sGrade,sEmail,sSex)
VALUES('张莉',DEFAULT,6,'ZQC@Sohu.com',0)
```

插入一条数据示例如图 5.1 所示。

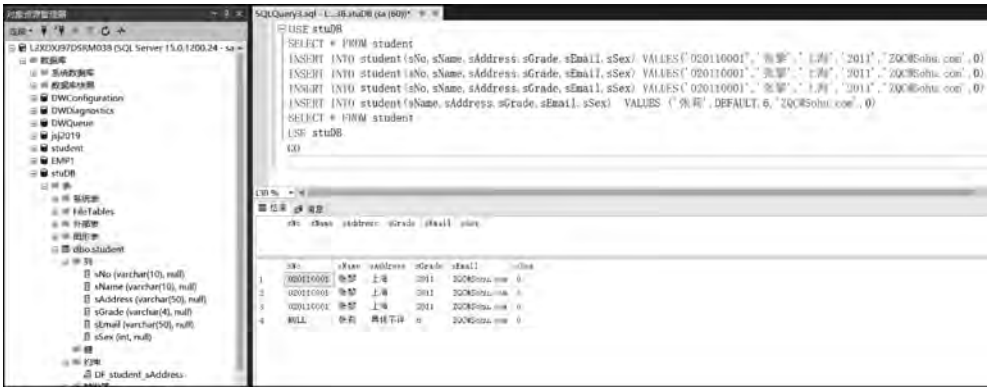


图 5.1　插入一条记录所用的数据库、表及 SQL 语句和查询结果

3. 插入多行数据

1) 从已知表向已知表插入若干满足条件的记录

其语法如下:

```
INSERT INTO <表名>(列名)
SELECT <列名>
FROM <源表名>
```

例:

```
INSERT INTO TongXunLu(姓名,地址,电子邮件)
SELECT sName,sAddress,sEmail
FROM student
```

2) 从已知表向未知表插入多行数据

其语法如下:

```
SELECT (列名)
INTO <新表名>
FROM <源表名>
```

注意:该语句只能执行一次。

例:

```
SELECT student.sName,student.sAddress,student.sEmail
INTO TongXunLu1 FROM student
```

3）在用 SELECT INTO 插入多行数据的时候插入新的标识列

其语法如下：

```
SELECT IDENTITY(数据类型,标识种子,标识增长量) AS 列名
INTO 新表 FROM 原始表
```

例：

```
SELECT student.sName,student.sAddress,student.sEmail, IDENTITY(int,1,1)
AS studentID INTO TongXunLu2 FROM student
```

前 3 个例子的执行情况如图 5.2 所示。

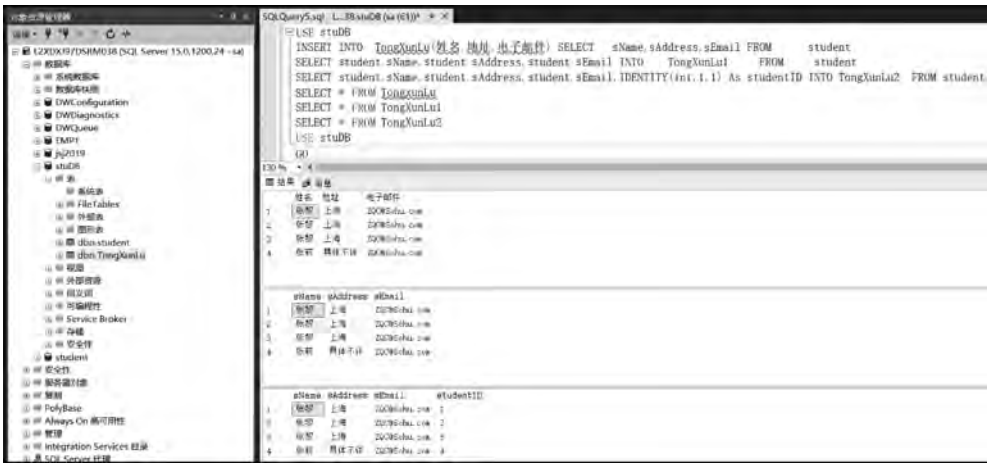


图 5.2　从已知表向其他表一次插入多条记录的 3 种方法的执行示例图

4）插入多行常值记录

其语法如下：

```
INSERT INTO <表名>(列名)
SELECT <列名> UNION
SELECT <列名> UNION
…
SELECT <列名>
```

例：

```
INSERT INTO student(sName,sGrade,sSex)
SELECT '测试女生 1',7,0 UNION
SELECT '测试女生 2',7,0 UNION
SELECT '测试女生 3',7,0 UNION
SELECT '测试女生 4',7,0 UNION
SELECT '测试女生 1',7,0 UNION
SELECT '测试男生 2',7,1 UNION
SELECT '测试男生 3',7,1 UNION
SELECT '测试男生 4',7,1 UNION
SELECT '测试男生 5',7,1
```

5）简化的插入多条记录的 SQL 语句

例：

```
INSERT INTO student(sName,sNo) VALUES ('Tom','123'),('John','456'),('Jack','569')
```

向已知表一次性插入多条常值记录的执行结果如图 5.3 所示。

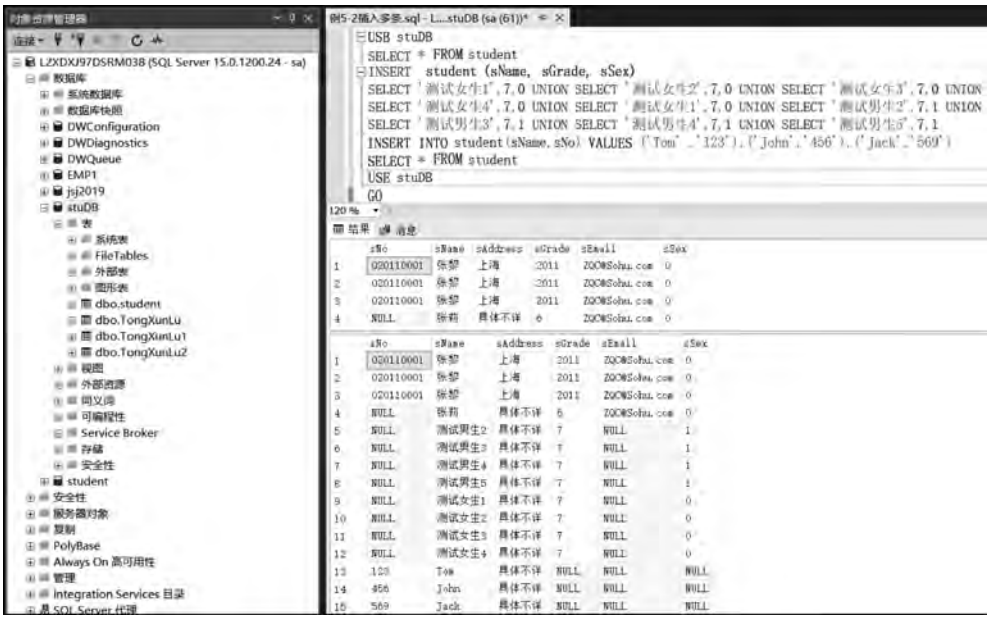


图 5.3　向已知表中插入多条常值记录的执行和查询结果

5.1.3　更新数据行

视频讲解

1. 更新表数据

其语法如下：

```
UPDATE <表名> SET <列名 1 = 更新值 1>, …,SET <列名 n = 更新值 n>
[WHERE <更新条件>]
```

例：用常量作更新值。

```
UPDATE student SET sSex = 0
```

例：有条件更新。

```
UPDATE student
SET sAddress = '北京'
WHERE sAddress = '上海'
```

例：用表达式更新。

```
UPDATE student
SET sSex = sSex + 1
WHERE sSex = 1
```

以上 3 种更新的执行结果如图 5.4 所示。

	sNo	sName	sAddress	sGrade	sEmail	sSex
1	020110001	张黎	上海	2011	ZQC@Sohu.com	0
2	020110001	张黎	上海	2011	ZQC@Sohu.com	0
3	020110001	张黎	上海	2011	ZQC@Sohu.com	0
4	NULL	张莉	具体不详	6	ZQC@Sohu.com	0
5	NULL	测试男生2	具体不详	7	NULL	1
6	NULL	测试男生3	具体不详	7	NULL	1
7	NULL	测试男生4	具体不详	7	NULL	1
8	NULL	测试男生5	具体不详	7	NULL	1
9	NULL	测试女生1	具体不详	7	NULL	0
10	NULL	测试女生2	具体不详	7	NULL	0

	sNo	sName	sAddress	sGrade	sEmail	sSex
1	020110001	张黎	北京	2011	ZQC@Sohu.com	1
2	020110001	张黎	北京	2011	ZQC@Sohu.com	1
3	020110001	张黎	北京	2011	ZQC@Sohu.com	1
4	NULL	张莉	具体不详	6	ZQC@Sohu.com	1
5	NULL	测试男生2	具体不详	7	NULL	1
6	NULL	测试男生3	具体不详	7	NULL	1
7	NULL	测试男生4	具体不详	7	NULL	1
8	NULL	测试男生5	具体不详	7	NULL	1
9	NULL	测试女生1	具体不详	7	NULL	1
10	NULL	测试女生2	具体不详	7	NULL	1

图 5.4　无条件和有条件更新数据的执行示例

2. 用其他表中的数据更新表数据

其语法如下：

```
UPDATE <表名 1> SET <表名 1.列名 = 表名 2.列名> FROM <表名 1>,<表名 2> [WHERE <更新条件>]
```

例：

```
UPDATE A SET A.stuName = B.stuName FROM A,B WHERE A.stuNo = '106'
UPDATE A SET A.stuName = B.stuName FROM A,B WHERE A.stuNo = B.stuNo
```

3. 允许使用复合赋值操作符

例：

```
UPDATE C SET score += 2 WHERE stuNo LIKE '10_'
```

执行前面两个例题的结果如图 5.5 所示。

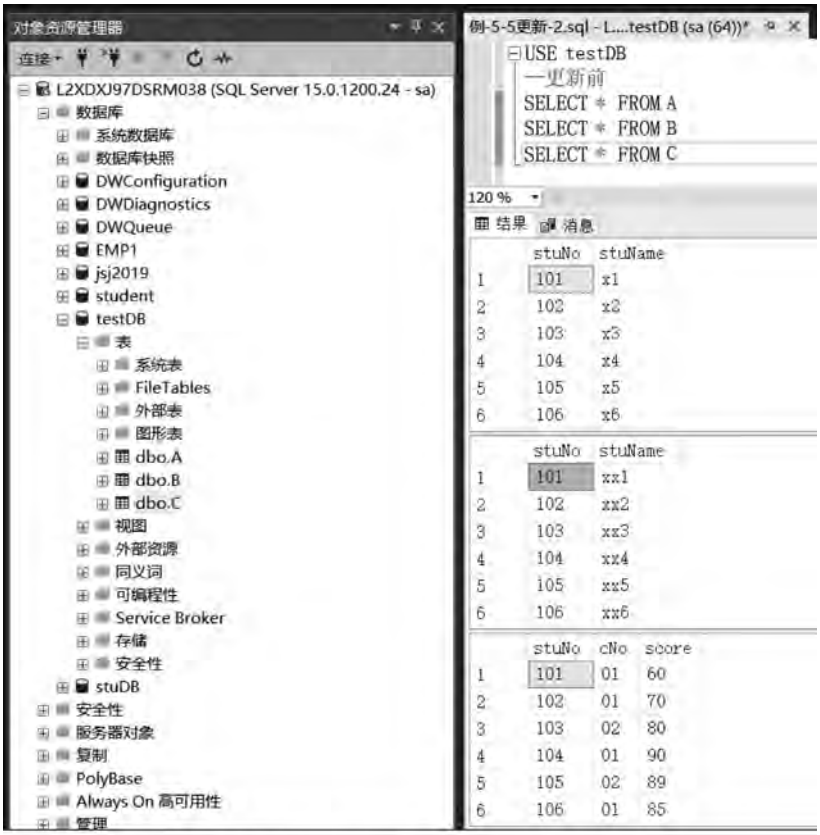
5.1.4　删除数据行和清空表中数据

视频讲解

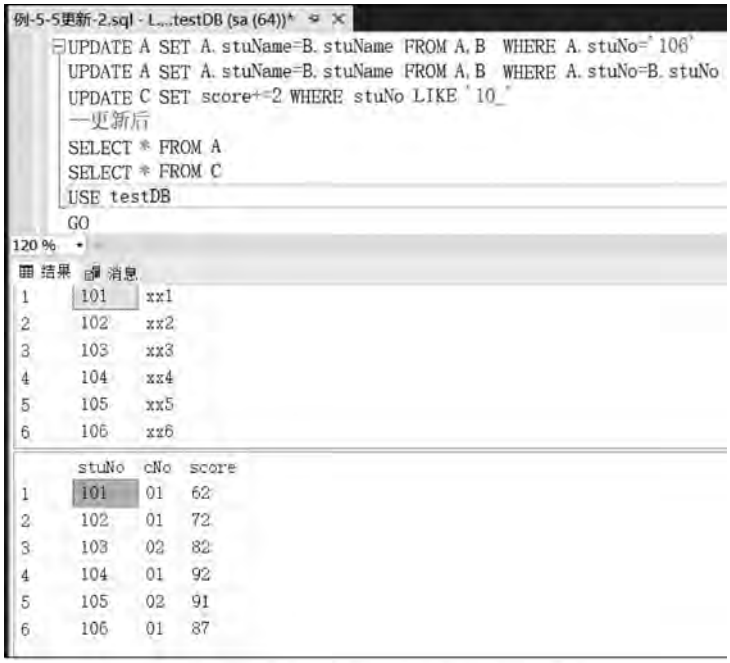
1. 删除数据行

其语法如下：

```
DELETE FROM <表名> [WHERE <删除条件>]
```



(a) testDB数据库中表A、B、C的原始结构和数据界面



(b) 执行修改语句及修改后的表A、B的结果

图 5.5　使用其他表改变本表数据以及使用复合赋值表达式修改数据的执行及结果

例：

```
DELETE FROM student WHERE sName = '张青'
```

2. 清空表中数据

其语法如下：

```
TRUNCATE TABLE <表名>
```

5.1.5 MERGE 语法

视频讲解

MERGE 语法是在一条语句中同时执行插入、更新、删除这 3 个操作。其操作原理是根据与源表连接的结果对目标表执行插入、更新或删除操作。例如，根据在另一个表中找到的差异在一个表中插入、更新或删除行，可以将两个表进行同步。

MERGE 语法包含 4 个主要子句，其中，MERGE 子句用于指定进行 INSERT、DELETE 和 UPDATA 操作的目标表或视图；USING 子句用于指定要与目标数据连接的数据源；ON 子句用于指定目标数据与数据源连接位置的匹配条件；WHEN 子句用于指定额外的过滤条件和数据更新逻辑。

例：

```
USE testDB
-- 创建一个订单表
CREATE TABLE Orders
(
  orderID int,
  customerID char(10)
)
GO
-- 往订单表中添加两行记录
INSERT INTO Orders VALUES(1,'2012010101'),(2,'2012010102')
-- 复制订单表中的第一条数据到新建表 Orders2
SELECT * INTO Orders2 FROM Orders WHERE orderID = 1
-- 显示两个表的初值
SELECT * FROM Orders
SELECT * FROM Orders2
-- 将 Orders2 表的数据进行更新
UPDATE Orders2 SET customerID = '2012010103'
-- 合并两个表
MERGE Orders AS o1
USING Orders2 AS o2
ON o2.orderID = o1.orderID
WHEN MATCHED
THEN UPDATE SET o1.customerID = o2.customerID          -- 如果匹配到了,就更新掉目标表
WHEN NOT MATCHED
THEN INSERT VALUES(o2.orderID,o2.customerID)           -- 如果匹配不到,就插入
WHEN NOT MATCHED BY SOURCE THEN DELETE;                -- 如果来源表无法匹配到,就删除
-- 显示修改后的表
SELECT * FROM Orders
SELECT * FROM Orders2
GO
```

MERGE 例题的执行结果如图 5.6 所示。

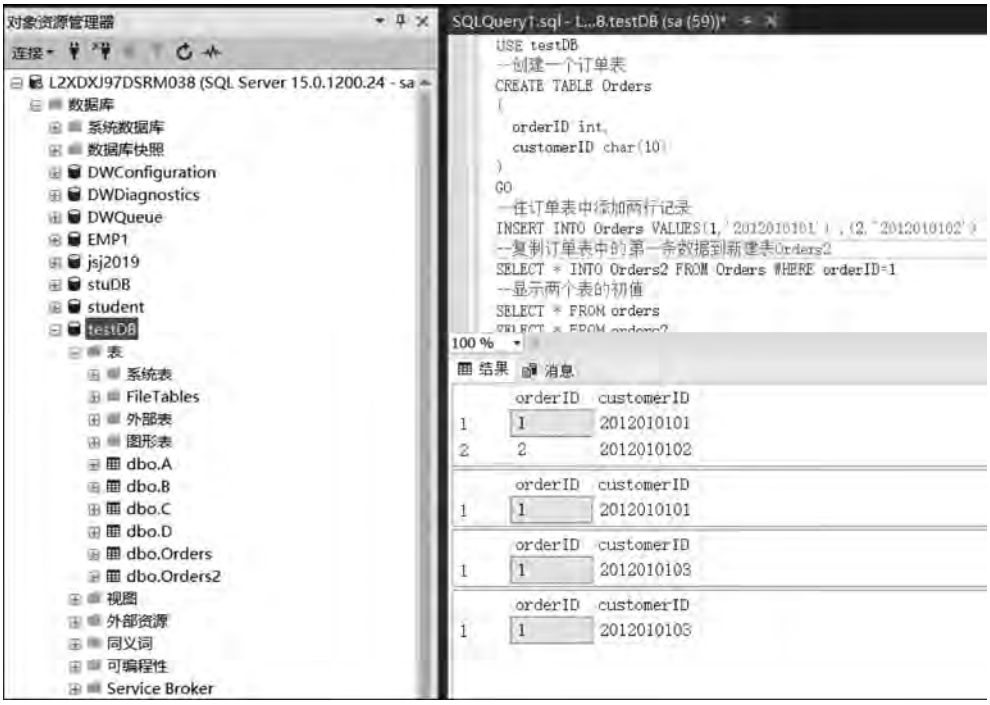


图 5.6　MERGE 例题的执行结果

5.2　基本数据查询

视频讲解

5.2.1　基本查询

1. 查询基本语法

```
SELECT      <列名>
FROM        <表名>
[WHERE      <查询条件表达式>]
[ORDER BY   <排序的列名>[ASC 或 DESC]]
```

例：

```
USE stuD B
SELECT sNo, sName, sAddress
FROM student
WHERE sSex = 1
ORDER BY sNo DESC
```

2. 查询所有行和列

例：

```
SELECT * FROM student
```

3. 查询部分行和列

例 1：

```
SELECT sNo,sName,sAddress FROM student
WHERE sAddress = '北京'
```

例 2：

```
SELECT sNo,sName,sAddress FROM student
WHERE sAddress <> '上海'
```

以上 3 个例题的执行结果如图 5.7 所示。

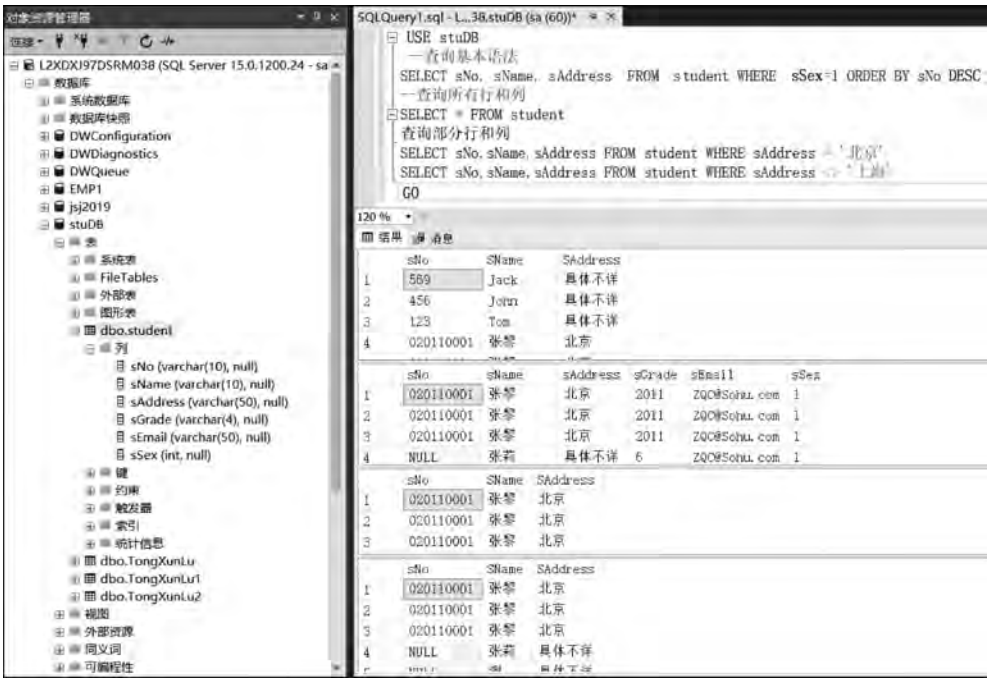


图 5.7　执行基本查询例题的结果

4. 数据查询：列名

1）使用 AS 命名列

例 1：

```
SELECT sNo AS 学生编号,sName AS 学生姓名,sAddress AS 学生地址
FROM student
WHERE sAddress <> '北京'
```

例 2：

```
SELECT sAddress + ' and ' + sEmail AS '地址及邮箱' FROM student
```

2）使用＝来命名列

例：

```
SELECT '地址及邮箱' = sAddress + ' and ' + sEmail FROM student
```

3）使用常量列

例：

```
SELECT 姓名 = sName,地址 = sAddress, '河北新龙' AS 学校名称 FROM student
```

命名列的例题的执行结果如图 5.8 所示。

图 5.8　命名列例题的执行结果

4）判断一行中的数据项是否为空

例：

```
SELECT sName FROM student WHERE sEmail IS NULL
```

执行结果如图 5.9 所示。

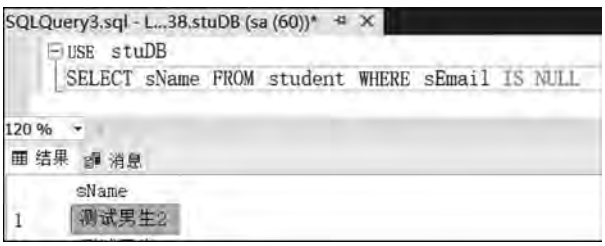


图 5.9　查询 NULL 列的执行结果

5. 数据查询：限制行数

1）限制固定行数

例：

```
SELECT TOP 5 sName, sAddress FROM student WHERE sSex = 1
```

2）返回百分之多少行

例：

```
SELECT TOP 20 PERCENT sName, sAddress FROM student WHERE sSex = 1
```

执行结果如图 5.10 所示。

图 5.10　限制行数查询界面

6. 数据查询：排序

1）升序排列

ASC 是默认升序排列命令，可以省略。

例：

```
SELECT * FROM stuInfo ORDER BY stuNo ASC
```

2）降序排列

DESC 是降序排列命令。

例：

```
SELECT * FROM stuInfo ORDER BY stuNo DESC
```

3）按照表达式排序

例：

```
SELECT stuNo AS 学员编号,(score * 0.9 + 5) AS 综合成绩
```

```
FROM scores WHERE(score * 0.9 + 5)> 60 ORDER BY score
```

4）按照新命名列排序

例：

```
SELECT stuNo + '.' + cNo AS [学号.科目号]
FROM scores UNION
SELECT stuNo + '.' + cNo AS [学号.科目号]
FROM scores2
ORDER BY [学号.科目号] DESC
```

5）按多列排序

例：

```
SELECT stuNo AS 学员编号, score AS 成绩
FROM scores
WHERE score > 60
ORDER BY score,cNo
```

排序查询例题的执行结果如图 5.11 所示。

120 %

结果 消息

	stuNo	stuName	stuAge	stuSeat	stuAddress
1	2019000001	x1	19	1	北京
2	2019000002	x2	19	2	具体不详
3	2019000003	x3	20	4	上海
4	2019000004	x4	19	5	具体不详

	stuNo	stuName	stuAge	stuSeat	stuAddress
1	2019000004	x4	19	5	具体不详
2	2019000003	x3	20	4	上海
3	2019000002	x2	19	2	具体不详
4	2019000001	x1	19	1	北京

	学员编号	综合成绩
1	2019000001	68.0
2	2019000003	68.0
3	2019000002	77.0
4	2019000001	77.0

	学号.科目号
1	2019000003.0003
2	2019000003.0002
3	2019000003.0001
4	2019000002.0003
5	2019000002.0002

	学员编号	成绩
1	2019000003	70
2	2019000001	70
3	2019000001	80
4	2019000002	80
5	2019000001	86

图 5.11　排序查询例题的执行结果

7. DISTINCT：去掉重复记录行

例：在成绩表 scores(stuNo,cNo,score)中查询参加考试的学生的考号。

```
SELECT DISTINCT stuNo FROM scores
```

其执行结果如图 5.12 所示。

图 5.12　DISTINCT 查询结果

5.2.2　基本函数

视频讲解

在查询中使用一些系统定义的函数可以令操作效果事半功倍，本节主要介绍字符串函数、日期函数、数学函数和系统函数。

字符串函数见表 5.4。

表 5.4　字符串函数

函　数　名	描　　述	举　　例
CHARINDEX	用来寻找一个指定的字符串在另一个字符串中的起始位置	SELECT CHARINDEX('ACCP','My Accp Course',1) 返回：4
LEN	返回传递给它的字符串长度	SELECT LEN('SQL Server 课程') 返回：12
LOWER	把传递给它的字符串转换为小写	SELECT LOWER('SQL Server 课程') 返回：sql server 课程
UPPER	把传递给它的字符串转换为大写	SELECT UPPER('sql server 课程') 返回：SQL SERVER 课程
LTRIM	清除字符左边的空格	SELECT LTRIM('周智宇　') 返回：周智宇(右边的空格保留)

续表

函 数 名	描 述	举 例
RTRIM	清除字符右边的空格	SELECT RTRIM('周智宇 ') 返回：周智宇(左边的空格保留)
RIGHT	从字符串右边返回指定数目的字符	SELECT RIGHT('买买提.吐尔松',3) 返回：吐尔松
REPLACE	替换一个字符串中的字符	SELECT REPLACE('莫乐可切.杨可','可','兰') 返回：莫乐兰切.杨兰
STUFF	在一个字符串中删除指定长度的字符，并在该位置插入一个新的字符串	SELECT STUFF('ABCDEFG',2,3,'我的音乐我的世界') 返回：A 我的音乐我的世界 EFG

日期函数见表 5.5。

表 5.5 日期函数

函 数 名	描 述	举 例
GETDATE	取得当前的系统日期	SELECT GETDATE() 返回：今天的日期
DATEADD	将指定的数值添加到指定的日期部分后的日期	SELECT DATEADD(mm,4,'01/01/99') 返回：以当前的日期格式返回 05/01/99
DATEDIFF	两个日期之间的间隔	SELECT DATEDIFF(mm,'01/01/99','05/01/99') 返回：4
DATENAME	日期中指定日期部分的字符串形式	SELECT DATENAME(dw,'01/01/2000') 返回：Saturday
DATEPART	日期中指定日期部分的整数形式	SELECT DATEPART(day,'01/15/2000') 返回：15

数学函数见表 5.6。

表 5.6 数学函数

函 数 名	描 述	举 例
ABS	取数值表达式的绝对值	SELECT ABS(－43) 返回：43
CEILING	返回大于或等于指定表达式的最小整数	SELECT CEILING(43.5) 返回：44
FLOOR	取小于或等于指定表达式的最大整数	SELECT FLOOR(43.5) 返回：43
POWER	取数值表达式的幂值	SELECT POWER(5,2) 返回：25
ROUND	将数值表达式四舍五入为指定精度	SELECT ROUND(43.543,1) 返回：43.5

续表

函 数 名	描 述	举 例
SIGN	对于正数返回＋1，对于负数返回－1，对于 0 返回 0	SELECT SIGN(－43) 返回：－1
SQRT	取浮点表达式的平方根	SELECT SQRT(9) 返回：3

系统函数见表 5.7。

表 5.7 系统函数

函 数 名	描 述	举 例
CONVERT	用来转变数据类型	SELECT CONVERT(varchar(5),12345) 返回：字符串 12345
CURRENT_USER	返回当前用户的名字	SELECT CURRENT_USER 返回：登录的用户名
DATALENGTH	返回用于指定表达式的字节数	SELECT DATALENGTH('哆啦 A 梦') 返回：7
HOST_NAME	返回当前用户所登录的计算机的名字	SELECT HOST_NAME() 返回：所登录的计算机的名字
SYSTEM_USER	返回当前所登录的用户名	SELECT SYSTEM_USER 返回：当前所登录的用户名
USER_NAME	从给定的用户 ID 返回用户名	SELECT USER_NAME(1) 返回：从任意数据库中返回“dbo”

5.2.3 综合应用

视频讲解

【例 5.1】 某公司做了一批手机充值卡，充值卡密码是随机生成的，现在出现一个问题，即充值卡密码里面的“o 和 0”(哦和零)、“l 和 1”(哎哦和一)，用户反映看不清楚。公司决定把存储在数据库里密码中的所有“哦”都改成“零”，把所有“l”都改成“1”。请编写 SQL 语句实现以上要求。其中数据库表名为 Card，密码字段名为 PassWord。

分析：这是更新操作，需要使用 UPDATE 语句。

因为涉及字符串的替换，需要使用到 SQL Server 中的函数 REPLACE()。

答：

```
UPDATE Card SET PassWord = REPLACE(密码,'o','0')
UPDATE Card SET PassWord = REPLACE(密码,'l','1')
```

或者写成一条语句：

```
UPDATE Card SET PassWord = REPLACE(REPLACE(密码,'o','0'),'l','1')
```

【例 5.2】 在数据库表中有以下字符数据：

```
1-1、1-2、1-3、1-10、1-11、1-108、1-18、1-31、1-15、2-1、2-2
```

现在希望通过 SQL 语句进行排序，并且首先按照前半部分的数字进行排序，然后再按照后半部分的数字进行排序，输出要排成这样：

```
1-1、1-2、1-3、1-10、1-11、1-15、1-18、1-31、1-108、2-1、2-2
```

数据库表名为 ArticleNo，字段名为 ListNumber。

分析：这是查询操作，需要使用 SELECT 语句；需要用到 ORDER BY 进行排序，并且在 ORDER BY 的排序列中也需要重新计算出排序的数字。

前半部分的数字，可以先找到“-”符号的位置，然后取其左半部分，最后再使用 CONVERT 函数将其转换为数字：

```
CONVERT(int, LEFT(ListNumber, CHARINDEX('-', ListNumber)-1))
```

后半部分的数字，可以先找到“-”符号的位置，然后把从第一个位置到该位置的全部字符替换为空格，最后再使用 CONVERT 函数将其转换为数字：

```
CONVERT(int, STUFF(ListNumber,1, CHARINDEX('-', ListNumber), ''))
```

答：

```
SELECT ListNumber
FROM ArticleNo
ORDER BY
CONVERT(int, LEFT(ListNumber, CHARINDEX('-', ListNumber)-1)),
CONVERT(int, STUFF(ListNumber,1, CHARINDEX('-', ListNumber), ''))
```

思考：还有其他的办法吗？

5.3 复杂数据查询

视频讲解

5.3.1 模糊查询

人们希望查询更加智能化，只要提供较少的线索，就可以进行相关数据的查询。模糊查询是较好的查询实现技术。

(1) LIKE：在查询时，字段中的内容并不一定与查询内容完全匹配，使用 LIKE 和通配符。

例：

```
SELECT stuName AS 姓名 FROM stuInfo WHERE stuName LIKE 'x%'
```

(2) IS NULL：把某一字段中内容为空的记录查询出来。

例：

```
SELECT stuName AS 姓名,stuAddress AS 地址
FROM stuInfo WHERE stuAddress IS NULL
```

(3) BETWEEN AND：把某一字段中内容在特定范围内的记录查询出来。

例：

```
SELECT stuNo, score FROM scores
WHERE score BETWEEN 60 AND 80
```

(4) IN：把某一字段中内容与所列出的查询内容列表匹配的记录查询出来。

例：

```
SELECT stuName AS 学员姓名,stuAddress AS 地址
FROM stuInfo WHERE stuAddress IN('北京','广州','上海')
```

模糊查询执行的结果如图 5.13 所示。

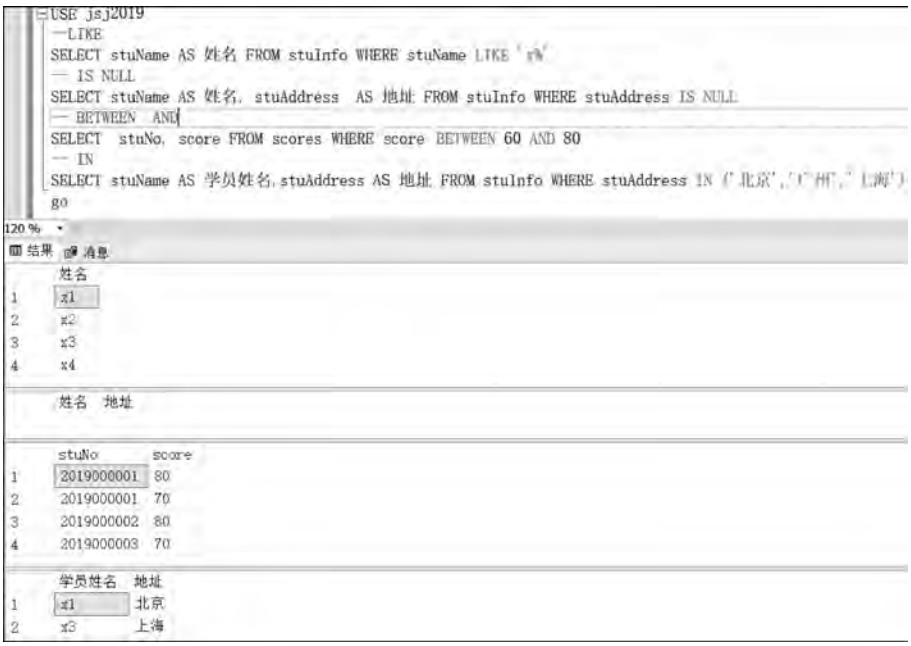


图 5.13　模糊查询的执行结果

5.3.2　聚合函数

视频讲解

(1) SUM：

例：

```
SELECT SUM(score) FROM scores WHERE cNo = '0001'
```

以下写法是错误的：

```
SELECT SUM(score),stuNo FROM scores WHERE cNo = '0001'
```

(2) AVG：

例：

```
SELECT AVG(score) AS 平均成绩 FROM scores WHERE score >= 60
```

(3) MAX、MIN：

例：

```
SELECT AVG(score) AS 平均成绩, MAX(score) AS 最高分,
MIN(score) AS 最低分 FROM scores WHERE score >= 60
```

(4) COUNT：

例：

```
SELECT COUNT( * ) AS 及格人数 FROM scores WHERE score >= 60
```

(5) 注意事项：

聚合函数不单独出现在条件语句中，只与返回结果值数目一致的列一起查询。

聚合函数的执行结果如图 5.14 所示。

图 5.14　聚合函数的执行结果

5.3.3　分组汇总

视频讲解

基本语法：

```
SELECT [<列名 x>],[聚合函数] FROM <表名>
WHERE 条件
GROUP BY <列名 x>
HAVING 条件
```

其中，WHERE 子句从数据源中去掉不符合其搜索条件的数据；GROUP BY 子句搜集数据行到各个组中，统计函数为各个组计算统计值；HAVING 子句去掉不符合其组搜索条件的各组数据行。

例：

```
SELECT        部门编号, COUNT( * )
```

```
FROM        员工信息表
WHERE       工资 >= 2000
GROUP BY    部门编号
HAVING      COUNT( * ) > 1
```

5.3.4 GROUPING SETS

视频讲解

从 SQL Server 2008 开始,可以使用 GROUPING SETS 进行分组及分组汇总。

例如有数据库 sale、表 sales(productID, productName, saleAmout, saleMonth),数据表的内容显示如图 5.15 所示。

	productID	productName	saleAmout	saleMonth
1	13	吸尘器	1201	2
2	10	微波炉	1290	3
3	16	洗衣机	1500	1
4	17	洗衣机	1501	2
5	4	电视机	1502	3
6	18	洗衣机	1540	3
7	7	热水器	1901	3
8	1	电冰箱	1943	3
9	14	吸尘器	2201	3
10	8	热水器	2401	2
11	5	电视机	2500	1
12	6	电视机	2510	2
13	9	热水器	2901	1
14	2	电冰箱	2913	2
15	3	电冰箱	2943	1
16	15	吸尘器	3201	1
17	11	微波炉	3230	2
18	12	微波炉	3290	1

图 5.15　按照销售量升序显示 sales 表中的数据

使用 SQL 语句分组显示各产品的销售总量:

```
SELECT productName AS 产品名,SUM(saleAmout) AS 销售总量 FROM sales
GROUP BY
GROUPING SETS
(
  (productName)
);
```

效果等同于:

```
SELECT productName AS 产品名,SUM(saleAmout) AS 销售总量 FROM sales
GROUP BY productName
```

结果如图 5.16 所示。

图 5.16　显示每个产品的销售总量

但是如果使用两个 GROUPING SETS 分组，就可以实现分组汇总了：

```
SELECT productName AS 产品名,SUM(saleAmout) AS 销售总量 FROM sales
GROUP BY
GROUPING SETS
(
  (productName),
  ()
);
```

结果如图 5.17 所示。

图 5.17　显示每个产品的销售总量和所有产品的销售总和

当然，可以给分组汇总结果起个名字，例如“汇总”，而不必用 NULL 显示：

```
SELECT [sales.Rep] = case
WHEN productName IS NULL THEN '汇总'
ELSE productName
```

```
END,
SUM(saleAmout) AS 销售总量 FROM sales
GROUP BY
GROUPING SETS
(
  (productName),
  ()
)
```

结果如图 5.18 所示。

图 5.18　显示每个产品的销售总量和带名字的所有产品的销售总和

另外还可以进行三级甚至四级分类汇总，例如将产品按月份分类汇总显示：

```
SELECT productName AS 产品名, SUM(saleAmout) AS 销售总量, saleMonth AS 销售月份 FROM sales
GROUP BY
GROUPING SETS
(
    (saleMonth,productName),
    (saleMonth),
    ()
);
```

结果如图 5.19 所示。

值得一提的是，CUBE 和 ROLLUP 是在 SQL Server 2005 中新增的 GROUP BY 扩展，用来创建分组汇总。例如上面的实现效果可以由以下 SQL 语句实现：

```
SELECT productName AS 产品名, SUM(saleAmout) AS 销售总量, saleMonth AS 销售月份
FROM sales GROUP BY
saleMonth,productName WITH ROLLUP
```

```
USE sale
SELECT productName AS 产品名, SUM(saleAmout)AS 销售总量, saleMonth AS 销售月份 FROM sales GROUP BY
GROUPING SETS
(  (saleMonth,productName),(saleMonth),() );
GO
```

	产品名	销售总量	销售月份
1	电冰箱	2943	1
2	电视机	2500	1
3	热水器	2901	1
4	微波炉	3290	1
5	吸尘器	3201	1
6	洗衣机	1500	1
7	NULL	16335	1
8	电冰箱	2913	2
9	电视机	2510	2
10	热水器	2401	2
11	微波炉	3230	2
12	吸尘器	1201	2
13	洗衣机	1501	2
14	NULL	13756	2
15	电冰箱	1943	3
16	电视机	1502	3
17	热水器	1901	3
18	微波炉	1290	3
19	吸尘器	2201	3
20	洗衣机	1540	3
21	NULL	10377	3
22	NULL	40468	NULL

图 5.19　三级分类汇总

执行结果如图 5.20 所示。

```
USE sale
SELECT productName AS 产品名, SUM(saleAmout) AS 销售总量, saleMonth AS 销售月份
FROM sales GROUP BY
saleMonth,productName  WITH  ROLLUP
GO
```

	产品名	销售总量	销售月份
1	电冰箱	2943	1
2	电视机	2500	1
3	热水器	2901	1
4	微波炉	3290	1
5	吸尘器	3201	1
6	洗衣机	1500	1
7	NULL	16335	1
8	电冰箱	2913	2
9	电视机	2510	2
10	热水器	2401	2
11	微波炉	3230	2
12	吸尘器	1201	2
13	洗衣机	1501	2
14	NULL	13756	2
15	电冰箱	1943	3
16	电视机	1502	3
17	热水器	1901	3
18	微波炉	1290	3
19	吸尘器	2201	3
20	洗衣机	1540	3
21	NULL	10377	3
22	NULL	40468	NULL

图 5.20　使用 ROLLUP 分组汇总

5.3.5 多表连接查询

视频讲解

1. 内连接

1）两表内连接

```
SELECT    <表名.列名>  FROM   左表
[INNER] JOIN  右表
ON    左表.列 = 右表.列
```

例：查询参加考试的学生的姓名、考试科目号码、考试成绩。

```
SELECT  S.stuName,C.cNo,C.score
FROM    scores AS C
INNER JOIN  stuInfo AS S
ON    C.stuNo  =  S.stuNo
```

2）三表内连接

例：查询参加考试的学生的姓名、考试科目名称、考试成绩。

```
SELECT
S.stuName AS 姓名, CS.cName AS 课程, C.score AS 成绩
FROM stuInfo AS S
INNER JOIN scores AS C ON(S.stuNo  =  C.stuNo)
INNER JOIN courseInfo AS CS ON(CS.cNo  =  C.cNo)
```

2. 多表连接查询

```
SELECT   <表名.列名> FROM 表 1,表 2 WHERE 左表.列 = 右表.列
```

例：

```
SELECT stuInfo.stuName, scores.cNo, scores.score
FROM stuInfo,scores WHERE stuInfo.stuNo = scores.stuNo
```

多表连接查询例题的执行结果如图 5.21 所示。

图 5.21　多表连接查询例题的执行结果

3. 外连接

1）左外连接(LEFT JOIN)

例：查询所有学生的考试情况。

```
SELECT  S.stuName,C.cNo,C.score FROM stuInfo AS S
 LEFT JOIN  scores AS C
 ON    C.stuNo = S.stuNo
```

2）右外连接(RIGHT JOIN)

例：

```
SELECT S.stuName,C.cNo,C.score  FROM stuInfo  AS S
RIGHT  JOIN  scores AS C
ON    C.stuNo = S.stuNo
```

外连接的执行结果如图 5.22 所示。

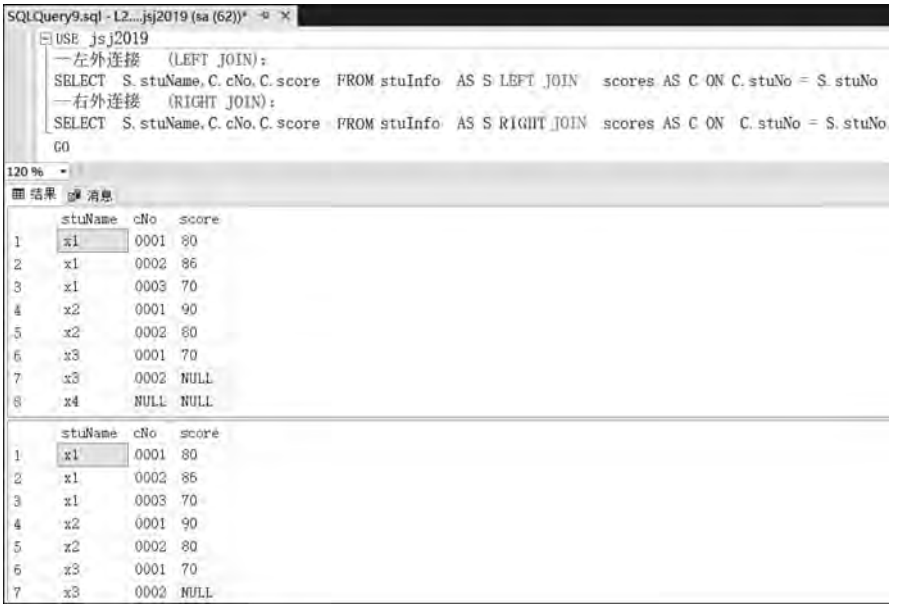


图 5.22　外连接例题的执行结果

思考：为什么以上两个例子的执行结果有所不同，它们的查询结果分别是什么含义？

4. UNION

两个结构相同的表的连接查询，相同列合并、记录行做或运算。

语法：

```
SELECT <列 1>,<…>,<列 n> FROM 表 1 UNION SELECT <列 1>,<…>,<列 n> FROM 表 2
```

例：在数据库 jsj2019 中有两个结构相同的表，即表 scores(stuNo，cNo，score)和表 scores2(stuNo，cNo，score)，求“SELECT stuNo，cNo FROM scores UNION SELECT stuNo，cNo FROM scores2”，具体如图 5.23 所示。

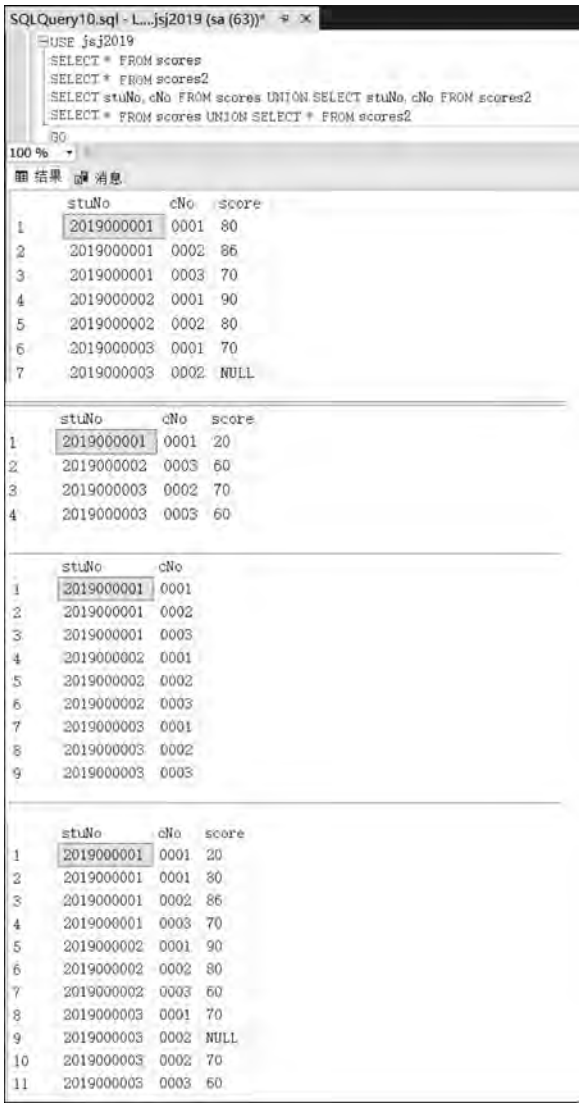


图 5.23　UNION 练习

5.3.6　综合应用

视频讲解

【例 5.3】 在给发掘出来的远古时代的谷物种子做育种实验的过程中，种子每 3 粒一组，组内每粒种子的培育方法不同。现要求查看所有组内培育方法相同的种子的发育值的平均值。

数据库表名为 TABLE1、字段名为 A、主键字段为 IDKEY(标识列，种子：1；增长量：1)。

分析：可以依靠标识列的值来进行判断和选取，但是因为操作过程中数据行可能存在增加、修改和删除，所以标识列的数据值未必完全有序，也就不“完全可靠”。例如标识列的值为 3，但并不一定是第三行，因为如果第二行被删除了，它就是第二行。

根据前面使用过的 SELECT…INTO，可以创建一张新表，顺便创建新的标识列，然后在新的标识列上执行除 3 取余判断。

判断依据：标识列值%3 等于 0、标识列值%3 等于 1 和标识列值%3 等于 2。

答：

```
SELECT        A, IDENTITY(int,1,1) AS ID
INTO          TABLE2
FROM          TABLE1

SELECT        AVG(A) AS  1 号种子发育平均值
FROM          TABLE2
WHERE         ID % 3  = 1

SELECT        AVG(A) AS  2 号种子发育平均值
FROM          TABLE2
WHERE         ID % 3  = 2

SELECT        AVG(A) AS  3 号种子发育平均值
FROM          TABLE2
WHERE         ID % 3  = 0
```

【例 5.4】 有学生基本信息表 stuInfo(stuNo,stuName)和学生成绩表 scores(stuNo,cNo,score)。现要求建立新表 stuAllInfo(stuNo,stuName,cNo,score)，存储所有学生的考试信息。

分析：这是数据插入的操作，因此要使用 INSERT 语句来进行。

参加考试的学生的考试信息在 scores 表里，所以可以使用 INSERT INTO…SELECT 结构。

但是还要插入 stuName 数据项，所以要用多表连接查询。

另外，还有学生可能没有参加任何一科的考试，所以根本不在 scores 表中。

等值的多表连接和不等值的多表连接都不能找到这些学生。

在前面的连接查询中，使用 INNER JOIN…ON 可以找出所有参加考试的学生的信息，编写以下 T-SQL：

```
SELECT stuInfo.stuNo, stuName, cNo, score FROM stuInfo
INNER JOIN  scores ON stuInfo.stuNo = scores.stuNo
```

但是如何把未参加任何一科考试的学生也显示在其中？如下可以吗？

```
SELECT stuInfo.stuNo, stuName, cNo, score FROM stuInfo
INNER JOIN  scores ON stuInfo.stuNo <> scores.stuNo
```

以上把“=”简单地改为“<>”，不仅不能找出未参加考试的学生，而且所找到的项很多，也没有意义，所以这种方法不可行。这也说明一点：内连接查询的基础是 ON 后面的等值比较，非等值比较就不是内连接查询了。

考虑前面学习过的左外连接查询，能够查询出左表中存在而相关表中不存在的数据项，所以可以使用如下语句查询出所有考生。

```
SELECT stuInfo.stuNo, stuName, cNo, score FROM stuInfo
LEFT OUT JOIN   scores ON stuInfo.stuNo = scores.stuNo
```

最后，使用子查询创建表 stuAllInfo。

答：

```
SELECT stuInfo.stuNo, stuName, cNo, score
INTO stuAllInfo
FROM stuInfo
LEFT JOIN   scores ON stuInfo.stuNo = scores.stuNo
SELECT * FROM stuAllInfo
```

在查询分析器中选择“工具”→“选项”，将“结果”下的“默认结果目标”修改为“显示为文本”。本例的执行结果如图 5.24 所示。

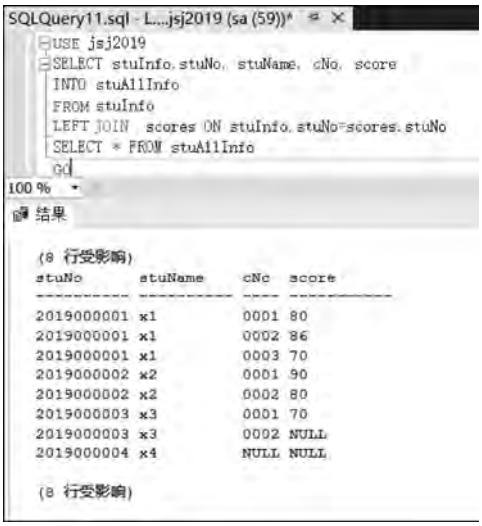


图 5.24　例 5.4 的执行结果

5.4　使用 SQL 语句设计和管理数据库

视频讲解

5.4.1　创建数据库

T-SQL 创建数据库的语法：

```
CREATE DATABASE 数据库名
 ON [PRIMARY]
 (
  <数据文件参数> [,…n]  [<文件组参数>]
 )
 [LOG ON]
 (
  <日志文件参数> [,…n]
 )
```

其中,[ ]表示可选参数。

这里创建一个数据库,其只包含一个主文件组,见例 5.5。

【例 5.5】 创建数据库 studentDB,保存在"D:\project"下,数据文件增长率为 15%、初始大小为 5MB,日志文件初始大小为 2MB、文件增长率按 1MB 自动增长。

```
CREATE DATABASE studentDB
  ON  PRIMARY                                    -- 默认就属于 PRIMARY 主文件组,可省略
(
 name = 'studentDB',                             -- 主数据文件的逻辑名
 filename = 'D:\project\studentDB.mdf',          -- 主数据文件的物理名
 size = 5MB,                                     -- 主数据文件的初始大小
 maxsize = 100MB,                                -- 主数据文件增长的最大值
 filegrowth = 15 %                               -- 主数据文件的增长率
)
LOG ON
(
 name = 'studentDB_log',
 filename = 'D:\project\studentDB_log.ldf',
 size = 2MB,
 filegrowth = 1MB
)
GO
```

执行结果如图 5.25 所示。

图 5.25 例 5.5 的执行结果

接下来创建一个数据库,其中包含多个数据文件和多个日志文件,即包含主文件组和从文件组,见例 5.6。

【例 5.6】 在"D:\"下创建数据库 DB,其包含一个主数据文件和一个从数据文件,数据文件增长率都按 10%自动增长,初始大小都为 1MB;日志文件增长率都按 1MB 自动增长,初始大小都为 1MB。

```
CREATE DATABASE DB
ON
(
  name = 'db1',
  filename = 'D:\db1.mdf',
  size = 1,
  filegrowth = 10 %
)
,
(
  name = 'db2',
  filename = 'D:\db2.ndf',
  size = 1,
  filegrowth = 10 %
)
LOG ON
(
  name = 'db1_log',
  filename = 'D:\db1_log.ldf',
  size = 1,
  filegrowth = 1
),
(
  name = 'db2_log',
  filename = 'D:\db2_log.ldf',
  size = 1,
  filegrowth = 1
)
GO
```

多个数据文件的好处是，如果硬盘满了，希望买个硬盘继续存放数据，这时就可以将一个数据文件放在 D 盘，将另一个数据文件放在另一个硬盘，例如 H 盘等。

例 5.6 的执行结果如图 5.26 所示。

图 5.26　例 5.6 的执行结果

5.4.2 删除数据库

删除数据库的语法：

```
DROP DATABASE 数据库名
```

例：判断是否已经存在数据库 stuDB，若存在，删除重建。

分析：新建的数据库信息在数据库的 sys.databases 视图中可以找到，所以只需要查看 master 数据库的 sys.databases 视图的 name 列即可。

答：

```
USE master                    -- 设置当前数据库为 master,以便访问 sys.databases 系统视图
GO
IF  EXISTS(SELECT * FROM  sys.databases WHERE  name = 'stuDB')
   DROP DATABASE stuDB
CREATE  DATABASE  stuDB
ON (
 …
)
LOG ON
(
 …
)
GO
```

注意：对于 EXISTS(查询语句)，如果查询语句返回一条以上的记录，表示存在满足条件的记录，返回 true，否则返回 false。

5.4.3 创建表

视频讲解

使用 SQL 语句创建数据表的基本步骤与使用表设计器创建是基本一致的，首先确定表中有哪些列，之后确定每列的数据类型，最后给表添加各种约束，包括创建表与表之间的关系。虽然使用表设计器直观、简单，但使用 SQL 语句创建数据表的效率要更高一些。

创建表的语法：

```
CREATE TABLE  表名
 (
     字段 1 数据类型 列的特征,
     字段 2 数据类型 列的特征,
     …
)
```

注意：

(1) 数据类型：数据表的字段，一般都要求在数据类型后加“()”，并在其中声明长度，例如 char、varchar 等，但 int、smallint、float、datetime、image、bit 和 money 类型不需要声明字段的长度。

(2) 列的特征：包括该列是否为空(NULL)、是否为标识列(自动编号)、是否有默认值、

是否为主键等。

【例 5.7】 创建学员信息表 stuInfo,具体要求如表 5.8 所示。

表 5.8 stuInfo 表

字　　段	类　　型	描　　述
stuNo	char(6)	非空
stuName	varchar(20)	非空
stuAge	int	非空
stuID	numeric(18,0)	身份证号
stuSeat	smallint	标识列
stuAddress	text	

实现代码如下:

```
USE studentDB                                  --将当前数据库设置为 studentDB
GO
CREATE  TABLE  stuInfo                         /*-创建学员信息表-*/
(
 stuName  varchar(20)  NOT  NULL ,             --姓名,非空(必填)
 stuNo   char(6)  NOT  NULL,                   --学号,非空(必填)
 stuAge  int  NOT  NULL,                       --年龄,int 类型默认为 4 字节
 stuID  numeric(18,0),                         --身份证号
 stuSeat   smallint  IDENTITY(1,1),            --座位号,自动编号
 stuAddress   text                             --住址,允许为空,即可选输入
)
GO
```

注意:

numeric(18,0)代表 18 位数字,小数位数为 0。

有些类型不必规定长度,大家要记住,例如 int、smallint、datetime。

【例 5.8】 创建学员程序设计成绩表 stuMarks,如表 5.9 所示。

表 5.9 stuMarks 表

字　　段	类　　型	描　　述
examNo	char(7)	考号,非空
stuNo	char(6)	学号,非空
writtenExam	int	笔试成绩,非空
labExam	int	机试成绩,非空

实现代码如下:

```
CREATE TABLE stuMarks
(
 examNo  char(7)  NOT NULL,                    --考号
 stuNo  char(6)  NOT NULL,                     --学号
 writtenExam  int  NOT NULL,                   --笔试成绩
 labExam  int  NOT NULL                        --机试成绩
)
GO
```

5.4.4 删除表

删除表的语法：

```
DROP TABLE 表名
```

【例 5.9】 如果当前数据库中已存在 stuInfo 表，此次创建时系统将提示出错。如何解决呢？

分析：当表中存在 stuInfo 表时，在 studentDB 数据库的系统视图 sys.objects 中检查 name 列即可。

答：

```
USE studentDB                    -- 将当前数据库设置为 studentDB,以便在 studentDB 数据库中建表
GO
IF EXISTS(SELECT * FROM  sys.objects  WHERE  name = 'stuInfo')
    DROP  TABLE  stuInfo
CREATE  TABLE  stuInfo       /* - 创建学员信息表 - */
(
…
)
GO
```

视频讲解

5.4.5 为表添加约束

SQL Server 中常用的约束类型如下。

- 主键约束(Primary Key Constraint)：要求主键列数据唯一，并且不允许为空。
- 唯一约束(Unique Constraint)：要求该列唯一，允许为空，但只能出现一个空值。
- 检查约束(Check Constraint)：某列的取值范围限制、格式限制等，例如有关年龄的约束。
- 默认约束(Default Constraint)：某列的默认值，例如男性学员较多，性别默认为“男”。
- 外键约束(Foreign Key Constraint)：用于在两表间建立关系，需要指定引用主表的那一列。

添加约束的语法：

```
ALTER TABLE 表名
     ADD CONSTRAINT 约束名   约束类型   具体的约束说明
```

约束名的取名规则推荐采用约束类型_约束字段。

- 主键(Primary Key)约束：例如 PK_stuNo。
- 唯一(Unique)约束：例如 UQ_stuID。
- 默认(Default)约束：例如 DF_stuAddress。
- 检查(Check)约束：例如 CK_stuAge。

• 外键(Foreign Key)约束：例如 FK_stuNo。

【例 5.10】 在 stuInfo 表(见表 5.8)上添加约束：①添加主键约束(stuNo 作为主键)；②添加唯一约束(因为每人的身份证号全国唯一)；③添加默认约束(如果地址不填，默认为“地址不详”)；④添加检查约束，要求年龄只能在 15～40 岁；⑤添加外键约束(主表 stuInfo 和从表 stuMarks 建立关系，关联字段为 stuNo)。

实现代码如下：

① 添加主键约束：

```
ALTER TABLE stuInfo
    ADD CONSTRAINT PK_stuNo PRIMARY KEY(stuNo)
```

② 添加唯一约束：

```
ALTER TABLE stuInfo
    ADD CONSTRAINT UQ_stuID UNIQUE(stuID)
```

③ 添加默认约束：

```
ALTER TABLE stuInfo
   ADD CONSTRAINT DF_stuAddress
       DEFAULT('地址不详') FOR stuAddress
```

④ 添加检查约束：

```
ALTER TABLE stuInfo
   ADD CONSTRAINT CK_stuAge
       CHECK(stuAge BETWEEN 15 AND 40)
```

⑤ 添加外键约束：

```
ALTER TABLE stuMarks
   ADD CONSTRAINT FK_stuNo
       FOREIGN KEY(stuNo) REFERENCES stuInfo(stuNo)
GO
```

各类表约束除了可以在数据表创建完毕之后添加以外，也可以在创建表时添加，见例 5.11。

【例 5.11】 使用 SQL 语句在创建表 stuInfo(见表 5.8)和 stuMarks(见表 5.9)的同时添加如例 5.10 中的约束。

实现代码如下：

```
USE studentDB                                    --将当前数据库设置为 studentDB
GO
CREATE  TABLE  stuInfo                           /*-创建学员信息表-*/
(
 stuName  varchar(20)  NOT  NULL,                --姓名,非空(必填)
 stuNo   char(6)  PRIMARY KEY,                   --学号,非空(必填)
 stuAge  int  CHECK(stuAge BETWEEN 15 AND 40),   --年龄
 stuID  numeric(18,0),                           --身份证号
 stuSeat   smallint  IDENTITY(1,1),              --座位号,自动编号
 stuAddress   text   DEFAULT('地址不详')          --住址,允许为空,即可选输入
 UNIQUE(stuID)
```

```
)
GO

CREATE TABLE stuMarks                                    /*创建学员成绩表 stuMarks*/
(
 examNo   char(7),                                       --考号
 stuNo   char(6),                                        --学号
 writtenExam   int,                                      --笔试成绩
 labExam   int,                                          --机试成绩
 PRIMARY KEY(examNo),
 FOREIGN KEY(stuNo) REFERENCES stuInfo(stuNo)
)
GO
```

例 5.11 的执行结果如图 5.27 所示。

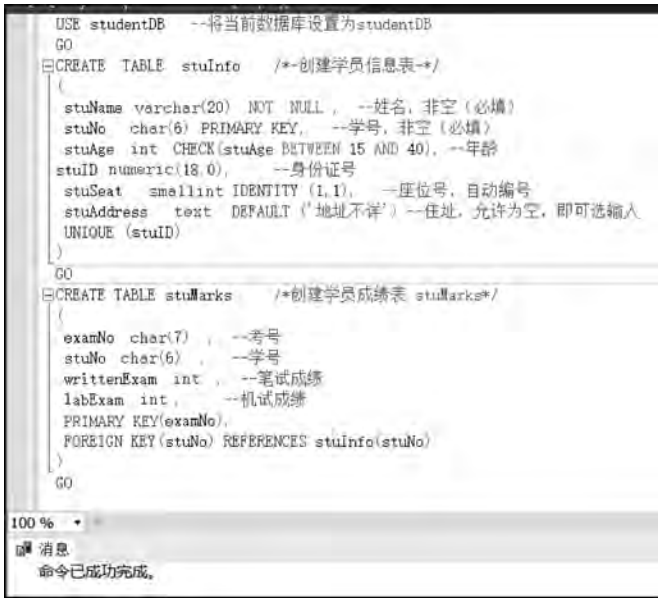


图 5.27　在 studentDB 库中创建表的同时添加约束

5.4.6　删除约束

如果错误地添加了约束，还可以删除约束。删除约束的语法如下：

```
ALTER TABLE 表名
      DROP CONSTRAINT 约束名
```

例 1：删除 stuInfo 表中的地址默认约束 DF_stuAddress。

```
ALTER  TABLE  stuInfo
     DROP  CONSTRAINT  DF_stuAddress
```

例 2：删除 stuInfo 表中的身份证号唯一约束 UQ_stuID。

```
ALTER  TABLE  stuInfo
     DROP  CONSTRAINT  UQ_stuID
```

5.4.7 安全管理

视频讲解

在第 4 章已经接触过如何使用企业管理器进行数据库权限设置，这里学习如何使用 SQL 语句来实现相应功能。

之前提过数据库安全管理的三道防火线，大家不妨按以下解释理解：

SQL Server 的安全模型正如一个防卫森严的小区，如果想进入自己的房间，需要闯三关。第一关，需要通过小区的门卫检查，进入小区；第二关，到了所在的单元楼门前，还需要单元楼门的钥匙或门铃密码；第三关，进了单元楼门后，还需要自己房间的钥匙。

大家回忆一下 SQL Server 的三层安全模型，它非常类似于小区的三层验证关口。第一关，需要登录到 SQL Server 系统，即需要登录账户；第二关，需要访问某个数据库（相当于单元楼），即需要成为该数据库的用户；第三关，需要访问数据库中的表（相当于打开房间），即需要数据库管理员自己授权，例如授予增加、修改、删除、查询等权限。

1. 登录验证的两种方式

- SQL 身份验证：适合于非 Windows 平台的用户或 Internet 用户，需要提供账户和密码。
- Windows 身份验证：适合于 Windows 平台用户，不需要提供密码和 Windows 集成验证。

登录账户相应有两种，即 SQL 账户和 Windows 账户。

2. 创建登录账户

（1）添加 Windows 登录账户：

```
EXEC sp_grantlogin   '域名\用户名'
```

例：

```
EXEC sp_grantlogin   'jbtraining\S26301'
```

（2）添加 SQL 登录账户：

```
EXEC sp_addlogin  用户名,密码
```

例：

```
EXEC sp_addlogin  'zhangsan', '1234'
```

注意：EXEC 表示调用存储过程，存储过程类似于 C 语言的函数。内置的系统管理员账户 sa，其密码在安装的时候初设，要求尽量复杂。

3. 创建数据库用户

创建数据库用户需要调用系统存储过程 sp_grantdbaccess，其用法为：

```
EXEC sp_grantdbaccess '登录账户名','数据库用户名'
```

其中，“数据库用户名”为可选参数，默认为登录账户，即数据库用户默认和登录账户同名。

例：在 studentDB 数据库中添加两个用户。

```
USE studentDB
GO
EXEC sp_grantdbaccess  'jbtraining\S26301', 'S26301DBUser'
EXEC sp_grantdbaccess  'zhangsan', 'zhangsanDBUser'
```

创建登录只是通过了第一关,还需要创建指定数据库的用户,打开数据库这道“单元楼门”。

数据库用户名可以省略,默认和登录名相同。

数据库用户如下。

- dbo 用户:表示数据库的所有者(DB Owner),注意无法删除 dbo 用户,此用户始终出现在每个数据库中。
- guest 用户:适用于没有数据库用户的登录账号访问,每个数据库可有也可删除。

系统内置的数据库用户如下。

- dbo 用户:表示数据库的主人。一般来说,谁创建了数据库,谁就是数据库的主人,但是可以转让,就像转让房屋产权证一样。
- guest 用户:若某人不是某个公司的员工,则该人进入该公司就是作为一个来宾(guest)。

数据库中的 guest 用户含义一样:如果某人登录到 SQL Server 中,希望访问某个数据库,但又不是该数据库的用户,那么当他访问时 SQL Server 就认为该人作为 guest 用户的身份访问数据库,至于该人作为 guest 用户访问该数据库能不能访问呢?那就要看管理员的授权了。

如果管理员给 guest 用户授予了访问的权限,那么该人就能访问,否则就不能访问。

4. 向数据库用户授权

打个比方:对于某间房屋来说,房屋的权限是指房产出售权(房主)、转租权(可能是租房人有事不住了,但又没到期)或只能居住(租房人)。

对于数据库来说,指的是数据库的增(INSERT)、删(DELETE)、改(UPDATE)、查(SELECT)权限以及后续将要学习的执行权限等。

授权的语法为:

```
GRANT 权限 [ON  表名 ]  TO  数据库用户
USE  studentDB
GO
/* -- 为 zhangsanDBUser 分配对 stuInfo 表的 SELECT、INSERT、UPDATE 权限 -- */
GRANT SELECT, INSERT, UPDATE  ON  stuInfo  TO  zhangsanDBUser
/* -- 为 S26301DBUser 分配建表的权限 -- */
GRANT  CREATE  TABLE  TO  S26301DBUser
```

小　　结

SQL(结构化查询语言)是数据库能够识别的通用指令集。

SQL Server 中的通配符经常和 LIKE 结合使用,用于进行不精确的限制。

WHERE 用于限制条件,其后紧跟条件表达式。

若一次插入多行数据，可以使用 INSERT INTO…SELECT…、SELECT…INTO…或者 UNION 关键字来实现。

在使用 UPDATE 更新数据时，一般都有限制条件。在使用 DELETE 删除数据时，不能删除被外键值所引用的数据行。

查询将逐行筛选表中的数据，最后把符合要求的记录重新组合成“记录集”，记录集的结构类似于表结构。

判断一行中的数据项是否为空，使用 IS NULL；使用 ORDER BY 进行查询记录集的排序，并且可以按照多个列进行排序。

在查询中可以使用常量、表达式、运算符；在查询中使用函数，能够像在程序中那样处理查询得到的数据项。

使用 LIKE、BETWEEN、IN 关键字能够进行模糊查询，即条件不明确的查询。

聚合函数能够对列生成一个单一的值，对于分析和统计非常有用。

分组查询是针对表中不同的组分类统计和输出，GROUP BY 子句通常结合聚合函数一起使用。HAVING 子句能够在分组的基础上再次进行筛选。

在多个表之间通常使用连接查询。最常见的连接查询是内连接(INNER JOIN)查询，通常会在相关表之间提取引用列的数据项。

数据库的物理实现一般包括创建数据库、创建表、添加各种约束、创建数据库的登录账户并授权。在创建数据库或表时一般需要预先检测是否存在该对象，数据库从 master 系统数据库的 sys.databases 视图中查询，表从该数据库的系统视图表 sys.objects 中查询。

访问 SQL Server 某个数据库中的某个表需要三层验证，即是否为 SQL Server 的登录账户，是否为该数据库的用户，是否有足够的权限访问该表。

课 后 题

1. 执行 SQL 语句“SELECT * FROM stuInfo WHERE stuNo LIKE '010[^0]%[A,B,C]%'”，可能会查询出的 stuNo 是(　　)。

A. 01053090A　　B. 01003090A01　　C. 01053090D09　　D. 0101A01

2. 使用以下(　　)可以进行模糊查询。

A. OR　　B. NOT BETWEEN

C. NOT IN　　D. LIKE

3. 对于成绩表 scores(stuNo,cNo,score)，以下(　　)语句返回成绩表中的最低分。

A. SELECT MAX(score) FROM scores

B. SELECT TOP 1 score FROM scores ORDER BY score ASC

C. SELECT MIN(score) FROM scores

D. SELECT TOP 1 score FROM scores ORDER BY score DESC

4. 有订单表 orders(customerID,orderMoney)，orderMoney 代表单次订购额，下面(　　)语句可以查询每个客户的订购次数和每个客户的订购总金额。

A. SELECT customerID,COUNT(DISTINCT(customerID)),SUM(orderMoney) FROM orders GROUP BY customerID

B. SELECT customerID,COUNT(DISTINCT(customerID)),SUM(orderMoney) FROM orders ORDER BY customerID

C. SELECT customerID, COUNT (customerID), SUM (orderMoney) FROM orders ORDER BY customerID

D. SELECT customerID, COUNT (customerID), SUM (orderMoney) FROM orders GROUP BY customerID

5. 有学生信息表 stuInfo(stuNo,stuName,stuSex,stuAge,stuEmail,stuAddress),以下(　　)语句能查出未填写 Email 信息的同学。

A. SELECT * FROM stuInfo WHERE stuEmail ="

B. SELECT * FROM stuInfo WHERE stuEmail =NULL

C. SELECT * FROM stuInfo WHERE stuEmail is NULL

D. SELECT * FROM stuInfo WHERE stuEmail="

上 机 题

1. 在 NetBar 的数据库表 Card 中为字段 ID 增加约束,ID 字段的格式限制如下:

- 只能是 8 位数字;
- 前两位是 0;
- 第 3～4 位为数字;
- 第 5 位为下画线;
- 第 6～8 位为字母。

2. 为 NetBar 的数据库表 Card 增加如表 5.10 所示的数据行。

表 5.10 增加数据行

ID	PassWord	Balance	UserName
030101	abc	100	均军
030102	abd	200	李开
030104	abe	300	朱军

对数据库执行增加、修改和删除数据的操作,使其数据行变为如表 5.11 所示。

表 5.11 执行操作后的数据行

ID	PassWord	Balance	UserName
030101	030101abc	98	均军
030104	abe	44	朱军
030105	ccd	100	何柳
030106	zhang	134	张君

3. 在前面的 NetBar 数据库中,编写查询语句实现以下要求。

- 由于最近屡次发生卡密码丢失事件,所以机房规定密码与姓名或者卡号不能一样。请编写 SQL 语句,查出密码与姓名或者卡号一样的人的姓名,以方便通知。

- 编号为 B02 的计算机坏了，请通过查询得到在这台计算机上最近一次上机的人的卡号。
- 为了提高上机率，上个月举行了优惠活动，即周六和周日每小时上机的费用为半价。请统一更新一下数据表中的费用信息。
- 编写查询显示本月上机时间最长的前三名用户的卡号。

4. 在前面的 NetBar 数据库中，一位家长想看看他儿子这个月的上机次数，已知他儿子的卡号为 0023030104，请编写 SQL 语句查询以下内容：

(1) 查询 24 小时之内上机的人员姓名列表。

(2) 查询本周的上机人员的姓名、计算机名、总费用，并按姓名进行分组。

(3) 查询卡号第 6 位和第 7 位是“BC”的人员的消费情况，并显示其姓名及费用汇总。