



SQL 语言

前言

SQL是上世纪70年代末期，由IBM实验室开发的。SQL是Structured Query Language的缩写，即“结构化查询语言”，它当初是为IBM公司的DB2开发的，是一个功能强大的数据库语言。

SQL不同于其他的计算机语言，是一个非过程语言。非过程(Nonprocedural)意味着“作什么”而不是“怎么作”，比如：SQL描述出要检索、删除、插入的数据特征，而不是怎样进行这些操作。

两个标准化组织：ANSI(美国国家标准学会)和ISO(国际标准化组织)，提出了SQL的工业标准，目前最新的标准是ANSI-92。





SQL概述

SQL是操纵和检索关系数据库中数据的标准语言，使用SQL，程序员和数据库管理员可以：

- 修改数据库结构；
- 改变系统的安全性设置；
- 增加用户操作库和表的权利；
- 从库中查询信息；
- 更新库的内容。



增强 SQL

为了增强功能，所有的数据库产品都在一定程度上和ANSI标准有所不同，绝大多数系统提供了SQL的专有扩展功能，将SQL扩展成过程型语言。微软的数据库产品SQL Server所提供的**Transact-SQL(T-SQL)**语言就是扩展的SQL语言。



SQL产品

Oracle公司开发了第一个使用SQL的商品化RDBMS。常见关系数据库管理系统有：Oracle、 Sybase、 Microsoft SQL Server、 Access、 Ingres等。

资源

- SQL Server 联机丛书，可从微软网站单独下载。
- 搜索引擎：www.google.com
cn.yahoo.com
www.excite.com
www.sina.com.cn
www.sohu.com.cn





DBMS和RDBMS

- DBMS (Database Managerment System)
数据库管理系统
 - 层次型数据库
 - 网络形数据库
 - 关系型数据库
- RDBMS (Relation Database Managerment System)
关系数据库管理系统



关系型DBMS—RDBMS

1970年E.F.Codd博士写了一篇题为《大型共享数据银行的数据的一个关系模型》(A Relational Model of Data for Large Shared Data Banks)的学术论文,这篇论文奠定了关系数据库的基础。

Codd博士提出了定义关系数据库模型的十三条准则:

0. 一个关系型的DBMS必须能够通过它的关系能力来全面地管理数据库。
1. 信息准确。RDBMS的所有信息(包括表和列名)都应该用表中的值显式地表示。
2. 保证访问准则。在关系数据库中的每个值都能保证以表名、主码值和列名的组合访问。



关系数据库模型的十三条准则

3. 系统的空值支持策略。RDBMS在系统级上提供对空值(未知的或尚不能显式地定义具体的数值)处理的支持。空值和默认值不同, 并且是独立于任何域的。
4. 动态的、联机的关系型数据字典。数据库的描述和它的内容在逻辑级上表示成表, 并且可以用数据库语言查询。
5. 功能范围广泛的数据子语言。一个关系系统至少应提供一种具有严格语法定义的语言, 它的功能应很全面、很广泛, 支持数据定义、数据操纵、完整性规则、授权和事物处理。
6. 视图更新准则。所有理论上可更新的可更新的视图也应该允许由系统更新。



关系数据库模型的十三条准则

7. 集合级别上的插入、修改和删除。DBMS。
8. 数据物理独立性。当数据的物理存取方法或存取结构变化时，应用程序和其他特殊程序应保持逻辑上不受影响。
9. 功能范围广泛的数据子语言。一个关系系统至少应提供一种具有严格语法定义的语言，它的功能应很全面、很广泛，支持数据定义、数据操纵、完整性规则、授权和事物处理。
10. 视图更新准则。所有理论上可更新的可更新的视图也应该允许由系统更新。



客户机/服务器开发工具

- ◆ Microsoft 公司的 Visual Basic、Visual C++
- ◆ Borland 公司的 Delphi
- ◆ PowerSoft 公司的 PowerBuilder



数据类型

在 SQL 语言 中，每个列、局部变量、表达式和参数都有一个相关的数据类型，这是指定对象可持有的数据类型（整型、字符、money 等等）的特性。SQL Server 提供系统数据类型集，定义了可与 SQL Server 一起使用的所有数据类型。下面列出系统提供的数据类型集。

精确数字，近似数字，字符串，Unicode 字符串，二进制字符串，其它数据类型。

有关数据类型的细节请看SQL Server 联机丛书。



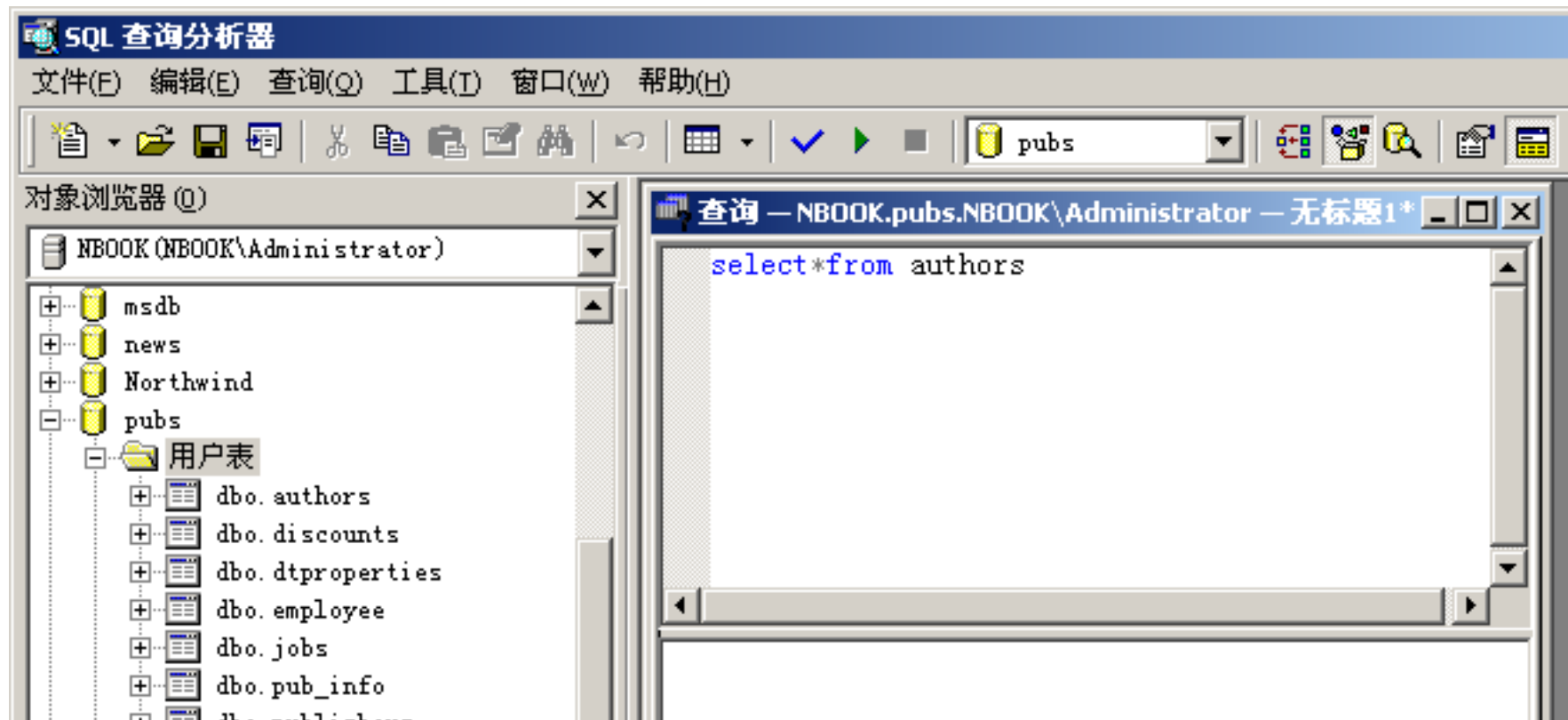
表、行、列

- ◆ **行**：SQL 表中构成表中的水平行的元素集合。表中的每一行代表由该表建模的对象的一个出现，并存储该对象所有特性的值。例如，在 Northwind 示例数据库中，Employees 表建立 Northwind Traders 公司职员模型。表中的第一行记录职员 ID 为 1 的职员的全部信息（例如姓名和头衔）。
- ◆ **列**：SQL 表中每一行中的区域，存储用该表制作模型的某个对象特性的数据值。例如，Northwind 示例数据库中的 Employees 表建立 Northwind Traders 公司职员模型。Employees 表的每个行中的 LastName 列存储该行所代表的职员的姓氏，就象窗口或表单中的 Last Name 字段包含姓氏一样。

SQL Server服务管理器



使用查询分析器





查询分析器连接 SQL Server

◆ 本地连接

- (local)
- .
- 主机名 (用ipconfig/all查询主机名)
- IP地址

◆ 远程连接

- 主机名
- IP地址



第一单元

查 询



错误信息

◆ select

服务器：消息 170，级别 15，状态 1，行 1
第 1 行： 'select' 附近有语法错误。



第一个查询

◆ `select * from authors`

星号*告诉数据库返回由from指定的表的所有列，返回顺序由数据库决定。大小写不影响查询结果。

◆ `select au_id, au_lname, au_fname,
phone, address, city, state, zip,
contract from authors`

与前一条SQL语句有相同的结果。



改变列的顺序

♦ `select au_id, phone, au_lname,
au_fname, address, city, state, zip,
contract from authors`

现在你有了控制各列输出顺序的能力。



选择个别列

如果你只对某些列感兴趣，比如只想检索
au_lname, au_fname, phone, address, 就应该
这样写SQL语句：

◆ `select au_lname, au_fname,
phone, address from authors`

现在你已经得到你感兴趣的列了。



选择另一张表

如果你需要另一张表中的信息，该怎么办呢？

◆ `select * from discounts`



结果不重复的查询

观察下面查询的输出：

- ◆ `select city from authors`

你会发现有相同的城市，两个人住在同一个城市这很正常，但如果你想看看到底有那几个城市在表中怎么办呢？试试这个：

- ◆ `select distinct city from authors`

SQL语法中还有一个和distinct对应的关键字all

- ◆ `select all city from authors`

你会发现查询结果和上面的第一个SQL语句相同，既然如此，谁还想自找麻烦呢？



第二单元

表达式、条件和操作符



表达式

表达式用于返回一个值。它包括：字符串、数字、布尔表达式。其实你已经用过表达式了，跟在`select`和`from`中间的东西都是表达式。

表达式是符号与运算符的组合。简单的表达式可以是一个常量、变量、列或标量函数。可以用运算符将两个或更多的简单表达式联接起来组成复杂的表达式。



条件



Where子句

Where子句的语法: where <搜索条件>
先来回顾一下我们前面所学的“第一个查询”

◆ `select * from authors`

你会看到, 结果返回了数据库pubs中表authors的所有内容, 如果你想查某一作者的情况, 可以键入 :

◆ `select * from authors where`

`au_lname = 'Smith'`

我们得到了想要的结果。



操作符

操作符是条件表达式中用于精确表示你想从数据库中查询出什么数据的运算符号。操作符大致分为六类：算术操作符、比较操作符、字符操作符、逻辑操作符、集合操作符和其他操作符。



算术操作符(+)

加号有两种不同的用法，可以作算术操作符，也可以作字符操作符，我们先用它作算术操作符。键入：

- ◆ `select discounttype,stor_id,lowqty,highqty,discount from discounts`

现在我们把所有的折扣增加0.5元，可以键入：

- ◆ `select discounttype,stor_id,lowqty,highqty,discount+0.5 from discounts`

我们得到了想要的结果。SQL允许你将已有的列进行组合或计算，以建立虚拟列和导出字段，原始表并不发生变化。



算术操作符(+) 续

观察一下discount+0.5字段的标题是(无列名), 太不好听了, 我们把它改一改。键入:

```
♦ select discounttype,stor_id,lowqty,  
highqty,discount+0.5 新折扣 from  
discounts
```

很好, 我们可以重新命名列标题。重新命名任意字段列标题的语法格式: 列名 **别名** (注意它们之间有空格)
加号还可以作为字符操作符, 后面你将看到这种用法。



建立虚拟列

我们想把原来的折扣和新折扣加以对照，键入：

```
♦ select discounttype,stor_id,lowqty,  
highqty, discount 旧折扣,discount+0.5  
新折扣 from discounts
```

太好了！我们不仅建立了一个新的列，还可以重新命名任意列标题。



算术操作符(-)

减号也有两种用法：可以用某列减去一个常数，或者用一列减去另一列。另一种用法是改变某数的符号。

我们先来改变一列数的符号，键入：

◆ `select discounttype, stor_id, lowqty, highqty, discount-3 from discounts`

用一列减去另一列，键入：

◆ `select discounttype, stor_id, lowqty, highqty, discount 旧折扣, discount+4 新折扣, discount+4-discount 折扣差 from discounts`



算术操作符(-) 续

如果不小心在数据类型是字符串的字段使用了减号:

♦ `select -discounttype, stor_id,
lowqty, highqty, discount from
discounts`

你将看到类似下面的结果:

服务器: 消息 403, 级别 16, 状态 1, 行 1

**对数据类型而言运算符无效。运算符为 minus, 类型为
varchar。**



算术操作符(/)

除号只有一种用法:

♦ `select discounttype, stor_id, lowqty, highqty, discount/3 from discounts`

或者:

♦ `select discounttype, stor_id, lowqty, highqty, discount 旧折扣, discount+4 新折扣, (discount+4)/discount 折扣比 from discounts`



算术操作符(*)

乘号也只有一种用法:

♦ `select discounttype, stor_id, lowqty, highqty, discount*3 from discounts`

或者:

♦ `select discounttype, stor_id, lowqty, highqty, discount 旧折扣, discount+4 新折扣, (discount+4)*discount 折扣乘 from discounts`



算术操作符(%)

模运算返回除法操作的余数：

◆ `select discounttype, stor_id, lowqty, lowqty%3 模, highqty, discount from discounts`

后面我们将会看到，模运算还可以作为函数。



算术操作符优先级

先乘除后加减，括号优先。



比较操作符

比较操作符比较两个表达式并返回如下三个值之一，
TRUE，FALSE或NULL。

重要 不能将空值用于区分表中两行所需的信息（例如，外键或主键）。

如果数据出现空值，则逻辑运算符和比较运算符有可能返回 TRUE 或 FALSE 以外的第三种结果 UNKNOWN。
需要三值逻辑是导致许多应用程序出错之源。



NULL

空值NULL的概念：在数据库术语中，如果某记录的某字段里没有数据，则该字段的值就是NULL。NULL并不意味着字段包含一个0值或长度为0的字符串。0值是一个整数，而长度为0的字符串也是字符串的一个具体值。NULL意味着什么也没有！操作符`is null`用来检测变量是否为空值。

重要 为了减少对已有查询或报表的维护和可能的影响，建议尽量少使用空值。对查询和数据修改语句进行规划，使空值的影响降到最小。



Is null 操作符

我们来找出一列中值为NULL的记录:

◆ `select * from discounts where lowqty
is null`

执行的不错, 现在如果用等号代替 `is null`:

◆ `select * from discounts where lowqty
= null`

会怎样呢? 试试看吧, 什么也没有, 因为 `lowqty =
null` 的比较结果是FALSE。

还有一个操作符是IS NOT NULL, 你一看就明白是什么意思。



比较操作符(=)

我们早就用过等号了:

◆ `select * from discounts where discount = 5`

上面字段discount的数据类型是数字型, 对字符串类型的字段记得加上单引号:

◆ `select * from discounts where discounttype = 'Volume Discount'`



比较操作符(>和>=)

大于号像这样工作:

♦ `select * from discounts where discount > 5`

如果要包括 5:

♦ `select * from discounts where discount >= 5`

字符也可以比较:

♦ `select * from discounts where discounttype >= 'Vo'`



比较操作符(<和<=)

小于号像这样工作:

♦ `select * from discounts where discount < 6.7`

如果要包括 6.7:

♦ `select * from discounts where discount <= 6.7`

字符也可以比较:

♦ `select * from discounts where discounttype <= 'Vo'`



比较操作符(<>或!=)

不等于可以这样写:

♦ `select * from discounts where discount <> 6.7`

还可以这样写:

♦ `select * from discounts where discount != 6.7`

字符也可以不等于:

♦ `select * from discounts where discounttype <> ' Volume Discount '`



字符操作符(like和%)

如果你想查找不十分精确的数据, like 很好用:

♦ `select * from authors where au_lname like 'St%'`

上面的操作选出au_lname的第一个字母是s的记录, 试试小写

♦ `select * from authors where au_lname like 'st%'`

大小写没关系。

%是通配符, 代表多个字符。%也可以多个使用, 见下面例子。



字符操作符()

下划线是单个字符通配符:

◆ `select * from authors where zip like '946_9'`

多个下划线使用:

◆ `select * from authors where zip like '9_6_8'`

与%混合使用:

◆ `select * from authors where phone like '%9_6_8%'`



字符操作符(+)

连接符 (+) 用于连接两个字符串:

♦ `select au_id, au_lname, au_fname,
au_lname+au_fname from authors`

我们在前面已经学过算术操作符+的用法。



逻辑操作符(and)

与 (and) 用于连接符两个表达式。只有当两个表达式都为TRUE时, and才返回TRUE, 否则返回FALSE。

◆ `select * from discounts where discount>=5 and discount<10`

很容易使用。



逻辑操作符(or)

或 (or) 也用于连接符两个表达式。当两个表达式中有一个为TRUE时, or就返回TRUE, 只有两个表达式都为FALSE时才返回FALSE。

◆ `select * from discounts where discount=5 or discount=6.7`
也很容易使用。



逻辑操作符(not)

非(not)是反运算。如果条件为TRUE, 则not运算后返回FALSE, 反之返回TRUE。

◆ `select * from discounts where not discount = 5`

也很容易使用。



操作符(in)

in可以简化你已经学过的一些查询，或者说你不会用**in**也没关系，用前面学过的知识可以满足你的要求。我们看前面的一个例子：

◆ `select * from discounts where discount=5 or discount=6.7`

我们用**in**来实现。

◆ `select * from discounts where discount in (5,6.7)`

不仅语句更短了，而且更容易阅读，**in**也可用于字符类型的字段。



操作符(between)

between也用来简化你已经学过的一些查询，或者说你不会用between也没关系，用前面学过的知识可以满足你的要求。我们看前面的一个例子：

◆ `select * from discounts where discount>=5 and discount<10`

我们用between来实现。

◆ `select * from discounts where discount between 5 and 10`

不仅语句更短了，而且更容易阅读，between也可用于字符类型的字段。



第三单元

函 数



函 数

- ◆ 聚集函数;
- ◆ 日期和时间函数;
- ◆ 数学函数;
- ◆ 字符函数;
- ◆ 数据类型转换函数;
- ◆ 其他函数。

函数非常多，我们不能一一举例，只能作一个示范性的介绍。



聚集函数(count)

count函数返回符合select语句中的查询条件的行数

- - ◆ `select count(*) from discounts
where discount>=5 and discount<10`



聚集函数(sum)

sum函数返回一列中所有值的和。

◆ `select sum(discount) from
discounts where discount>=5 and
discount<10`

只对数值类型字段有效。



聚集函数(avg)

avg函数计算一列的平均值。

♦ `select avg(discount) from
discounts where discount>=5 and
discount<10`

只对数值类型字段有效。



聚集函数(max)

max函数找到一列的最大值。

◆ `select max(discount) from
discounts where discount>=5 and
discount<10`

可用于字符型字段。



聚集函数(min)

min函数找到一系列的最小值。

◆ `select min(discount) from discounts where discount >= 5 and discount < 10`

可用于字符型字段。

max函数和min函数可以一块儿使用，查出值所在的范围。

◆ `select min(discount), max(discount) from discounts`



聚集函数(var)

var函数计算标准差的平方—方差。

◆ `select var(discount) from discounts`

只对数值类型字段有效。



聚集函数(stddev)

stddev函数计算标准差的平方—方差。

◆ `select stddev(discount) from discounts`

只对数值类型字段有效。



日期函数(年、月、日)

给定日期取年、月、日。

```
♦ SELECT "Year Number" =  
YEAR('03/12/2003'), "Month Number" =  
MONTH('03/12/2003'), DAY('03/12/2003'  
) AS 'Day Number'
```

注意不同的列标题的定义方法，比较一下我们以前学过的别名。



日期函数(年、月、日)

与上页等价的写法。

```
♦ SELECT "Year Number" =  
DATEPART(yy, '03/12/2003'), "Month  
Number" = DATEPART(mm,  
'03/12/2003'), "Day Number" =  
DATEPART(dd, '03/12/2003')
```

注意不同的列标题的定义方法，比较一下我们以前学过的别名。



日期函数(系统日期)

从系统日期中取年、月、日。

```
♦ SELECT "Year Number" =  
YEAR (getdate()) , "Month Number" =  
MONTH (getdate()) , DAY (getdate()) AS  
'Day Number'
```




日期函数(访问字段)

从日期字段中取年、月、日。

```
♦ SELECT "Year Number" =  
YEAR(hire_date), "Month Number" =  
MONTH(hire_date), DAY(hire_date) AS  
'Day Number' from employee
```



数学函数

我们经常要用到数学函数，例如计算discount列的指数：

◆ `SELECT discount, exp(discount)`
`from discounts`

还有很多数学函数，例如：ABS、CEILING、DEGREES、FLOOR、POWER、RADIANS、SIGN、EXP、LOG、LOG10、SQUARE、SQRT和三角函数，这里不再列举，可查阅SQL Server 联机丛书。



字符函数(chr)

将 `int ASCII` 代码转换为字符的字符串函数，例如：

- ◆ `SELECT char(65)`

整数表达式是介于 0 和 255 之间的整数。如果整数表达式不在此范围内，将返回 `NULL` 值。

对于 `Unicode` 编码，整数表达式可以大于255，例如：

- ◆ `SELECT nchar(65)`
- ◆ `SELECT nchar(26578)`



字符函数(len)

len函数返回给定字符串表达式的字符（而不是字节）个数，其中不包含尾随空格。

◆ `SELECT discount, len(discount)`
`from discounts`

len函数可用于数字和字符，例如：

◆ `SELECT * , len(discounttype) from`
`discounts`

还有很多字符函数，可查阅SQL Server 联机丛书



数据类型转换

数据类型转换有两种：

隐性转换对于用户是不可见的。 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。

显式转换使用 CAST 或 CONVERT 函数。

详细用法可查阅SQL Server 联机丛书。



其他函数

还有一些函数，如：配置函数、系统函数等，可查阅
SQL Server 联机丛书



第四单元

子 句



子 句

- ◆ select
- ◆ where
- ◆ order by
- ◆ group by
- ◆ having



Select语句的一般语法

```
SELECT [ ALL | DISTINCT ]
      [ TOP n [ PERCENT ] [ WITH TIES ] ]
      < select_list >
< select_list > ::=
      {
          *
          | { table_name | view_name |
            table_alias }. *
          | { column_name | expression |
            IDENTITYCOL | ROWGUIDCOL }
          [ [ AS ] column_alias ]
          | column_alias = expression
      }
      [ , ...n ]
```



Select语句的一般语法(续)

从上页我们看到了select语法的复杂性，实际上其他SQL语句的语法也很复杂，迄今为止，我们一直避免这种复杂的语法图，因为对于初学者来说，复杂的语法难以理解，所以我们一直在用例子解释特定的语法点。下面我们还这样讲，详细的语法可查阅SQL Server联机丛书。



Select语句的子句

- ◆ 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。虽然 SELECT 语句的完整语法较复杂，但是其主要的子句可归纳如下：

- ◆ 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和where子句

指定查询返回的列。你已经上百次地使用过Select子句和where子句了，以后你还要更多地使用它们。



从混沌到有序 Order by子句

如果你需要把查询结果按顺序显示，可以使用order by子句：

- ◆ `SELECT * from titles`

记录是按录入顺序显示的。下面按价格从小到大排序：

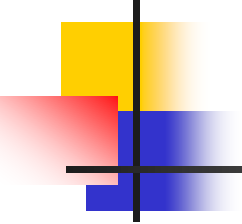
- ◆ `SELECT * from titles order by price`

按价格从大到小排序：

- ◆ `SELECT * from titles order by price`

`desc`

从小到大是升序排列，关键字是`asc`，系统默认排序为升序，所以`asc`可以省略。从大到小是降序排列，关键字是`desc`。



Order by子句(续)

order by子句也可以用来对字符型数据排序，排序规则较为复杂，可查阅SQL Server联机丛书。

- ◆ `SELECT * from titles order by title asc`

order by子句也可以按多列进行排序：

- ◆ `SELECT * from titles order by title, type, price`

注意优先级：

- ◆ `SELECT * from titles order by price, type, title`



Group by子句

指定用来放置输出行的组，并且如果 `SELECT` 子句 `<select list>` 中包含聚合函数，则计算每组的汇总值。指定 `GROUP BY` 时，选择列表中任一非聚合表达式内的所有列都应包含在 `GROUP BY` 列表中，或者 `GROUP BY` 表达式必须与选择列表表达式完全匹配。

说明 如果未指定 `ORDER BY` 子句，则使用 `GROUP BY` 子句返回的组没有任何特定的顺序。**建议始终使用 `ORDER BY` 子句指定特定的数据顺序。**



Group by子句

在前面你已经学会了使用聚集函数 (count, sum, avg, min, max等), 先看下面:

- ◆ `SELECT * from titleview`

注意: titleview是视图, 视图的概念我们将在后面讲解
我们计算一下总价格:

- ◆ `SELECT sum(price) from titleview`

我们使用多个聚集函数, 按每个人分组计算价格:

- ◆ `SELECT au_lname, sum(price) 分组和,
count(au_lname) 计数 from titleview
group by au_lname`



子句组合

SQL变的越来越有用了。在做项目时，实际情况往往很复杂，我们还可以将子句组合起来使用：

♦ `SELECT au_lname, sum(price) 分组和,
count(au_lname) 计数 from titleview
group by au_lname order by au_lname
desc`



having子句

having子句给用在group by子句中的数据加限制条件。HAVING 通常与 GROUP BY 子句一起使用。如果不使用 GROUP BY 子句, HAVING 的行为与 WHERE 子句一样。

♦ SELECT au_lname, sum(price) 分组和,
count(au_lname) 计数 from titleview
group by au_lname having
sum(price)>20 order by sum(price)

HAVING可以让你在比较表达式中使用聚集函数, 而WHERE则不行。



正确使用where和having子句

WHERE和ORDER BY子句经常用于单行的查询，就像这样：

◆ `SELECT * from titleview where price>20 order by au_lname desc`

GROUP BY和HAVING一般用于合计，像上页的例子。

如果将WHERE和ORDER BY、 GROUP BY和HAVING这两组子句结合起来，就会产生料想不到的结果，你可以试试看。也许你要的正是这种混乱的局面。



第一阶段总结

到现在为止，凭你所学的知识，你已经能够很有效地使用单个表了。下面我们将扩展范围，包括对多个进行操作。



第五单元

创建和操纵表



简介

迄今为止，你已经学习了用各种条件从数据库中检索数据。但是，你一定觉得困惑，在查询之前，这些数据是怎样写入数据库的？SQL“结构化查询语言”就像它的名字一样，只能从数据库中查询数据吗？实际上SQL能做的比这多的多，你可以用SQL建库、建表、增加数据、删除数据、合并数据，并根据数据库中数据的变化激发触发器活动。可是没有找到更好的单词代替SQL，如果将这种语言称为“结构化查询增加修改删除连接存储触发器查询语言”是很麻烦的。所以，我们还叫它SQL吧，当然，你现在已经知道它的功能要远远大于它的字面意思。



数据定义语句

数据定义语句用来创建数据库、创建表；修改表结构；删除表、删除数据库。实现这些功能的语句被称为**数据定义语句**，主要包括：

- ◆ CREATE DATABASE
- ◆ CREATE TABLE
- ◆ ALTER TABLE
- ◆ DROP TABLE
- ◆ DROP DATABASE

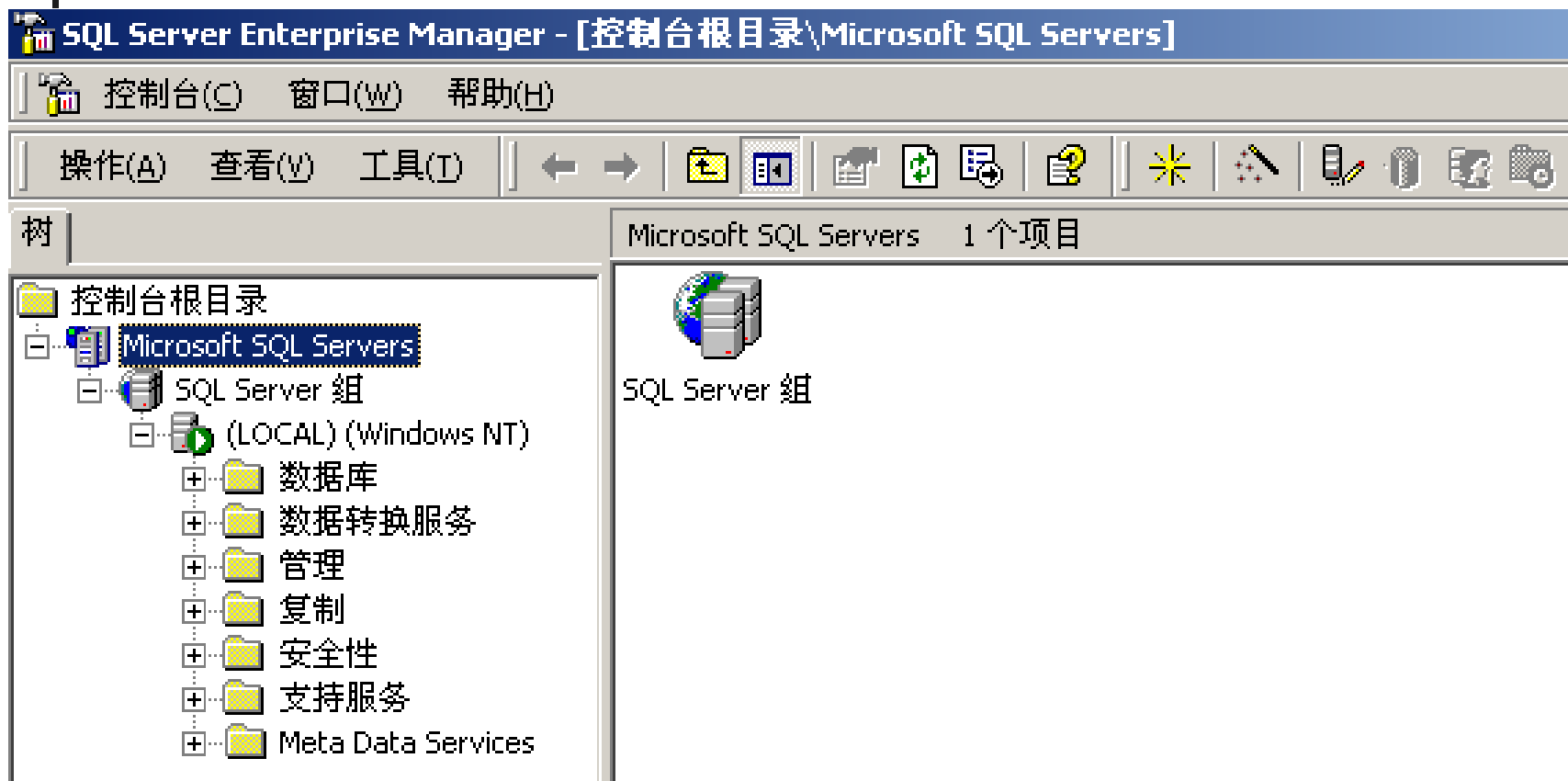
数据定义语句用来完成数据库的管理任务。



你的第一个数据库项目

我们的目标是建立一个学生成绩管理系统。我们首先建立一个数据库`student`，再在`student`中建立两个表：学生档案和学生成绩。

使用企业管理器





创建数据库

在开始任何数据库项目的时候，第一个数据库管理的步骤就是创建数据库。尽管现在的数据库管理系统提供图形界面工具来完成数据库的管理工作，但是，学会用SQL代码来完成数据库管理任务还是很重要，例如，当我们建立一个自动安装系统时，就需要用SQL代码所形成的文件。下面我们用两种方法创建数据库。

- ◆ 用企业管理器创建数据库student (演示)

用SQL语句创建数据库student (注意选择master数据库，注意刷新)

- ◆ `create database student`



删除数据库

和创建数据库一样，我们可以用两种方法删除数据库。

- ◆ 用企业管理器删除数据库student（演示）

用SQL语句删除数据库student（注意选择master数据库，注意刷新）

- ◆ `drop database student`

创建表 “学生档案”

我们用两种方法创建表。(注意避免 “空” - NULL)

- ◆ 用企业管理器创建表 “学生档案” (演示)

使用默认值 (DEFAULT) 约束可以使得客户端编程方便一些，默认值可以是常量、函数、系统函数、空值 (NULL) 等，默认值不能用于 timestamp 列和 IDENTITY 列。

- ◆ 用SQL语句创建表 “学生档案” (注意选择 student 数据库，注意刷新)

下页是创建 “学生档案” 的SQL代码。

创建表“学生档案” (续)

```
if exists (select * from dbo.sysobjects where id =  
object_id(N'[dbo].[学生档案]') and OBJECTPROPERTY(id,  
N'IsUserTable') = 1)  
drop table [dbo].[学生档案]  
GO
```

```
CREATE TABLE [dbo].[学生档案] (  
    [s_id] [int] IDENTITY (1, 1) NOT NULL ,  
    [学号] [varchar] (10) COLLATE Chinese_PRC_CI_AS NULL ,  
    [姓名] [varchar] (10) COLLATE Chinese_PRC_CI_AS NULL ,  
    [性别] [char] (2) COLLATE Chinese_PRC_CI_AS NULL ,  
    [班级] [varchar] (50) COLLATE Chinese_PRC_CI_AS NULL  
) ON [PRIMARY]  
GO
```



SQL代码的自动生成

手工书写SQL代码虽不是很难，但也要花费一些时间，我们可以用企业管理器自动生成已建对象的SQL语言代码。
例：生成单个表的SQL脚本代码。

◆ 打开企业管理器 – 选择一个表，例如：学生档案
– 右击表名 – 所有任务 – 打开“生成 SQL 脚本”对话框 – 确定 – 选择路径和文件名 – 保存

。

在查询分析器中运行看看，你发现没有默认值约束的定义，主键定义也没有，就是我们前面用过的。怎么办？在打开的“生成 SQL 脚本”对话框中，选择 选项 卡，选中表脚本选项 的4个复选框，生成新的SQL脚本代码。



SQL代码的自动生成(续)

```
if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[学生档案]') and OBJECTPROPERTY(id, N'IsUserTable') = 1)
drop table [dbo].[学生档案]
GO
```

```
CREATE TABLE [dbo].[学生档案] (
    [s_id] [int] IDENTITY (1, 1) NOT NULL ,
    [学号] [varchar] (10) COLLATE Chinese_PRC_CI_AS NULL ,
    [姓名] [varchar] (10) COLLATE Chinese_PRC_CI_AS NULL ,
    [性别] [char] (2) COLLATE Chinese_PRC_CI_AS NULL ,
    [班级] [varchar] (50) COLLATE Chinese_PRC_CI_AS NULL
) ON [PRIMARY]
GO
```

```
ALTER TABLE [dbo].[学生档案] WITH NOCHECK ADD
    CONSTRAINT [PK_学生档案] PRIMARY KEY CLUSTERED
    (
        [s_id]
    ) ON [PRIMARY]
GO
```

```
ALTER TABLE [dbo].[学生档案] ADD
    CONSTRAINT [DF_学生档案_学号] DEFAULT (') FOR [学号],
    CONSTRAINT [DF_学生档案_姓名] DEFAULT (') FOR [姓名],
    CONSTRAINT [DF_学生档案_性别] DEFAULT (') FOR [性别],
    CONSTRAINT [DF_学生档案_班级] DEFAULT (') FOR [班级]
GO
```

“学生档案”的样本数据



学号	姓名	性别	班级
001	王小童	男	初二一班
002	张柳风	女	初二一班
003	紫云飞	女	初二三班
004	黄天龙	男	初二二班

在 查询分析器 中验证你的工作。

创建表 “学生成绩”

我们用两种方法创建表。（注意避免“空” - NULL）

- ◆ 用企业管理器创建表 “学生成绩”

用SQL语句创建表 “学生成绩”（注意选择student数据库，注意刷新）

- ◆ 下页是创建 “学生成绩” 的SQL代码

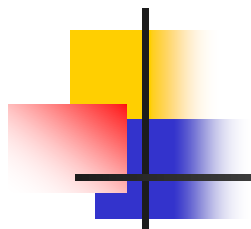
创建表“学生成绩” (续)

```
if exists (select * from dbo.sysobjects where id =  
object_id(N'[dbo].[学生成绩]') and OBJECTPROPERTY(id,  
N'IsUserTable') = 1)  
drop table [dbo].[学生成绩]  
GO
```

```
CREATE TABLE [dbo].[学生成绩] (  
    [s_id] [int] NULL ,  
    [学习科目] [varchar] (50) COLLATE Chinese_PRC_CI_AS NULL ,  
    [学习成绩] [tinyint] NULL  
) ON [PRIMARY]  
GO
```

```
ALTER TABLE [dbo].[学生成绩] ADD  
    CONSTRAINT [DF_学生成绩_s_id] DEFAULT (0) FOR [s_id],  
    CONSTRAINT [DF_学生成绩_学习科目] DEFAULT (') FOR [学习科目],  
    CONSTRAINT [DF_学生成绩_学习成绩] DEFAULT (0) FOR [学习成绩]  
GO
```

“学生成绩”的样本数据



S_id	语文	数学	英语
1	78	100	87
2	85	92	95
3	65	89	86
4	98	67	75



数据备份和恢复

数据备份

- ◆ 右击student – 所有任务 – 备份数据库，打开 SQL Server 备份 对话框 – 单击 添加 按钮，选择路径和文件名，选择 “重写现有媒体” – 确定

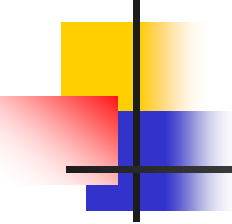
数据恢复

- ◆ 右击student – 所有任务 – 还原数据库，打开 还原数据库 对话框 – 单击 “从设备” 单选按钮 – 单击 “选择设备” 按钮，添加路径和文件名 – 确定



第六单元

连 接 (JOIN)



连 接

- ◆ 内连接;
- ◆ 外连接;
- ◆ 左连接;
- ◆ 右连接;
- ◆ 等值连接;
- ◆ 非等值连接。



表关系

在关系数据库中，关系能防止冗余的数据。例如，如果正在设计一个数据库来跟踪有关书的信息，而每本书的信息（如书名、出版日期和出版商）都保存在一个名为 titles 的表中。同时还有一些想保存的有关出版商的信息，例如出版商的电话号码、地址和邮政编码。如果将所有这些信息都保存在 titles 表中，则对于某个出版商出版的每本书，出版商的电话号码将是重复的。

一个更好的解决方案是，单独在一个名为 publishers 的表中只保存一次出版商信息。然后在 titles 表中设置指针，以引用 publishers 表中的项。



数据完整性

存储在数据库中的所有数据值均正确的状态。如果数据库中存储有不正确的数据值，则该数据库称为已丧失数据完整性。

强制数据完整性可确保数据库中的数据质量。例如，如果输入了 `employee_id` 值为 123 的职员，那么该数据库不应允许其他职员使用同一 ID 值。如果计划将 `employee_rating` 列的值范围设定为从 1 到 5，则数据库不应接受 6。如果表有一 `dept_id` 列，该列存储职员的部门编号，则数据库应只允许接受公司中的有效部门编号。



数据完整性的分类

对表进行计划有两个重要步骤：标识列的有效值和确定如何强制列中的数据完整性。数据完整性有四种类型：

- 实体完整性
- 域完整性
- 引用完整性
- 用户定义完整性

▪ 参考SQL Server联机丛书—数据完整性，概述。



数据同步

若要确保数据同步，可以在 titles 表和 publishers 表之间强制**引用完整性**。引用完整性关系能确保某个表中的信息与另一个表中的信息相匹配。例如，titles 表中的每个书名必须和 publishers 表的特定出版商相关联。不能在数据库中添加数据库中不存在的出版商的书名。

引用完整性

引用完整性是一种规则系统，这些规则可确保相关表中各行间关系的有效性，并确保不会意外删除或更改相关的数据。

在强制引用完整性时必须遵循以下规则：

- 如果在相关表的主键中不存在某个值，则不能在相关表的外键列中输入该值。但是，可以在外键列中输入空值。例如，在 employee 表中没有包括某职员，则不能指明分配给该职员的工作，但是可在 employee 表的 job_id 列输入空值来指明没有给该职员分配工作。
- 如果在相关表中存在与某行匹配的行，则不能从主表中删除该行。例如，如果在 employee 表中给多个职员分配了由 jobs 表中某行所代表的工作时，则不能删除该行。
- 当主表的某行有相关行时，则不能更改主键值。例如，如果将 jobs 表中的一项工作分配给某职员，则不能从 employee 表中删除该职员。

。

- 当满足下述所有条件时，可以设置引用完整性：
- 主表中相匹配的列是主键或具有唯一约束。
- 相关列具有相同的数据类型和长度。



唯一-UNIQUE 约束

强制非主键上的实体完整性的约束。UNIQUE 约束确保未输入重复值，并创建一个索引以增强性能。



键、键列

键：唯一标识行 (PRIMARY KEY)、定义两表之间的关系 (FOREIGN KEY) 或用于生成索引的一个列或一组列。

键列：由主键、外键或索引键引用的列。



主键

主键 (PK)：唯一标识表中的所有行的一个列或一组列。主键不允许空值。不能存在具有相同的主键值的两个行，因此主键值总是唯一标识单个行。表中可以有不止一个键唯一标识行，每个键都称作候选键。只有一个候选键可以选作表的主键，所有其它候选键称作备用键。尽管表不要求具有主键，但定义主键是很好的做法。在规范化的表中，每行中的所有数据值都完全依赖于主键。例如，在以 `EmployeeID` 作为主键的规范化的 `employee` 表中，所有列都应包含与某个特定职员相关的数据。该表不具有 `DepartmentName` 列，因为部门的名称依赖于部门 ID，而不是职员 ID。



外键

外键 (FK)：列或列的组合，其值与同一个表或另一个表中的主键 (PK) 或唯一键相匹配。也称作参照键。



表关系类型



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。在大多数情况下，关系将一个表中为每个行提供唯一标识符的主键与另一个表中外键内的项相匹配。例如，通过在 titles 表的 title_id 列（主键）和 sales 表的 title_id 列（外键）之间创建一个关系，可以使销售额与特定的销售书名相关联。

表与表之间存在三种类型的关系。所创建的关系类型取决于相关联的列是如何定义的。

一对一关系

在一对一关系中，表 A 中的一行最多只能与表 B 中的一行相匹配，反之亦然。如果两个相关列都是主键或具有唯一约束，则创建的是一对一关系。

这种关系不常见，因为这种方式的大部分相关信息都在一个表中。使用一对一关系可以是为了：

- 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。
- 一对一关系的主键方由键  符号表示。外键方也由键  符号表示。

一对多关系

一对多关系是最常见的关系类型。该关系中第一个表中的单个行可以与第二个表中的一个或多个行相关（匹配），但第二个表中的一个行只可以与第一个表中的一个行相关。例如，publishers 表和 titles 表是一对多的关系：每一个出版商可出版许多书，但每一本书只能有一个出版商。

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。

一对多关系中的主键方由一个键  符号表示。关系中的外键方由一个无穷大 ∞ 符号表示。



多对多关系

在多对多关系中，表 A 中的一行可与表 B 中的多行相匹配，反之亦然。通过定义称为连接表的第三方表创建这样的关系，该连接表的主键包括表 A 和表 B 中的外键。例如，authors 表和 titles 表是多对多关系，该关系通过从这些表中的每个表与 titleauthors 表的一对多关系定义。titleauthors 表的主键由 au_id 列（authors 表的主键）和 title_id 列（titles 表的主键）组成。

连接表：建立其它表之间的关系的表。连接表包含引用构成该关系的表的外键。例如，OrderParts 连接表可以通过具有 Orders 表和 Parts 表的外键来展示每个订单运送的部件。



连接两个表

如果我们需要的信息来自两个表，可以这样：

◆ **SELECT** **dbo.学生档案.学号**, **dbo.学生成绩.学习科目** **FROM** **dbo.学生成绩** **CROSS JOIN** **dbo.学生档案**

这就是两个表的连接，

仔细观察一下结果，找出其中的规律。

上面演示的就是所谓的交叉 (CROSS) 连接，它虽然不怎么有用，但它用来演示所有连接的。

手工在视图工具中输入下面SQL语句，看看执行结果。

◆ **SELECT * FROM 学生档案,学生成绩**



视图(VIEW)和交叉连接

视图 (VIEW) 是一种虚拟表，视图可以封装复杂的查询，视图通过 SQL 语句由基表导出，视图可以像真实表一样使用。我们可以通过视图来较快地学习连接。

♦ 先修改 学生档案 表，去掉主键设置，在企业管理器中做 学生档案 和 学生成绩 两个表的视图

仔细观察一下结果，你会发现前面这就是两个表的交叉 (CROSS) 连接。



内联接

通过比较源表间共享的列的值从多个源表检索行的操作。内联接排除来自不具有与其它源表中的行匹配的行的源表的行。



视图(VIEW)和内连接

我们通过视图来学习内连接。

♦ 先修改 学生档案 表，加上主键设置，在企业管理器中做 学生档案 和 学生成绩 两个表的视图

这就是两个表的内 (INNER) 连接，其SQL语句如下：

♦ `SELECT dbo.学生档案.学号, dbo.学生档案.姓名, dbo.学生档案.性别, dbo.学生档案.班级, dbo.学生成绩.学习科目, dbo.学生成绩.学习成绩 FROM dbo.学生成绩 INNER JOIN dbo.学生档案 ON dbo.学生成绩.s_id = dbo.学生档案.s_id`



将一个表连接到它本身

将一个表连接到它自身是个经常使用的技术，用来找出有重复字段的记录。

♦ `SELECT dbo.学生档案.学号, dbo.学生档案.姓名 FROM dbo.学生档案 INNER JOIN
dbo.学生档案 学生档案_1 ON dbo.学生档案.姓名 = 学生档案_1.姓名 AND
dbo.学生档案.学号 <> 学生档案_1.学号`

等值连接。

非等值连接。



外联接

一种联接，包括满足搜索条件的联接表的所有行，甚至包括在所联接的表中没有匹配行的表中的行。对于当一个表中的行与另一个表中的行不匹配时所返回的结果集行，将为解析到没有相应行的表中的所有结果集列提供 NULL 值。



外联接-左连接

在s_id列上联接 学生档案 表和 学生成绩 表。结果只显示相匹配的数据。若要在结果中包括所有的学生信息，而不管 学生成绩 表中是否有关联的记录，可以使用 SQL-92 左向外联接 LEFT OUTER JOIN 。下面是 Transact-SQL 左向外联接的查询和结果：

```
♦ SELECT dbo.学生档案.学号, dbo.学生档案.姓名, dbo.
学生档案.性别, dbo.学生档案.班级, dbo.学生成绩.学习科目,
dbo.学生成绩.学习成绩, dbo.学生成绩.s_id FROM dbo.学
生档案 LEFT OUTER JOIN dbo.学生成绩 ON dbo.学生档案
.s_id = dbo.学生成绩.s_id
```

在视图中观察设计和结果。



外联接-右连接

在s_id列上联接 学生档案 表和 学生成绩 表。结果只显示相匹配的数据。若要在结果中包括所有的学生信息，而不管 学生成绩 表中是否有关联的记录，可以使用SQL-92 右向外联接运算符 RIGHT OUTER JOIN。不管第一个表中是否有匹配的数据，结果将包含第二个表中的所有行。例：

```
♦ SELECT dbo.学生档案.学号, dbo.学生档案.姓名,  
dbo.学生档案.性别, dbo.学生档案.班级, dbo.学生成绩.  
学习科目, dbo.学生成绩.学习成绩, dbo.学生成绩.  
s_id FROM dbo.学生档案 RIGHT OUTER JOIN  
dbo.学生成绩 ON dbo.学生档案.s_id = dbo.学生成绩.  
s_id
```



外联接-全连接

若要通过在联接结果中包括不匹配的行保留不匹配信息，请使用完整外部联接。Microsoft® SQL Server™ 2000 提供完整外部联接运算符 FULL OUTER JOIN，不管另一个表是否有匹配的值，此运算符都包括两个表中的所有行。

假设在 s_id 列上联接 学生档案 表和 学生成绩 表。结果只显示相匹配的数据。SQL-92 FULL OUTER JOIN 运算符指明：不管表中是否有匹配的数据，结果将包括两个表中的所有行。例：

♦ SELECT dbo.学生档案.学号, dbo.学生档案.姓名, dbo.学生档案.性别, dbo.学生档案.班级, dbo.学生成绩.学习科目, dbo.学生成绩.学习成绩, dbo.学生成绩.s_id FROM dbo.学生档案 FULL OUTER JOIN dbo.学生成绩 ON dbo.学生档案.s_id = dbo.学生成绩.s_id



第七单元

数据操纵



数据操纵语句简介

在上一单元，你已经学习了用数据定义语句创建数据库及其相关的表。在这个单元，我们学习怎样往表中增加、修改和删除数据，完成这些功能的语句就是**数据操纵语句**。数据操纵语句有三个：

- ◆ INSERT
- ◆ UPDATE
- ◆ DELETE

它们用来操纵数据库中的表中的数据。



数据操纵语句—INSERT

INSERT语句用于将数据录入到数据库中。它可以划分为如下两语句：

- ◆ **INSERT...VALUES**
- 和**
- ◆ **INSERT...SELECT**



INSERT...VALUES

INSERT语句以一次一记录的方式录入数据，对于只和几条记录打交道的小规模操作很有用。它的语法如下：

◆ INSERT INTO 表名 (列1, 列2...)
VALUES (值1, 值2...)

这是INSERT...VALUES语句的基本格式，用于给表增加一条记录，在使用时，必须遵守以下三条规则：

- ◆ 用于插入的值必须和相应字段的数据类型一致。
- ◆ 插入数据的大小尺寸必须在相应列的数据范围之内。

例如，一个有80个字符的字符串不能插入只有40个字符长度的列。

- ◆ 在VALUES后面的值序列必须和前面的列的序列一一对应。



INSERT语句

INSERT语句以一次一记录的方式录入数据，对于只和几条记录打交道的小规模操作很有用。它的语法如下：

- ◆ INSERT INTO 表名 (列1, 列2...)
VALUES (值1, 值2...)

和

- ◆ INSERT...SELECT

例

- ◆ INSERT INTO 学生档案 (学号, 姓名, 性别)
VALUES ('009', '李刚', '男')



UPDATE语句

UPDATE语句用于改变现有记录中字段的值，它的语法如下：

◆ UPDATE 表名 SET 列1=值1,列2=值2 WHERE 搜索条件

例

◆ UPDATE 学生档案 SET 班级='初二一班'

注意：因为省略了 WHERE 子句，表中的每条记录的相应字段都被更新。所以，一般情况下UPDATE语句应带 WHERE 子句。例

◆ UPDATE 学生档案 SET 班级='初二三班'
WHERE s_id=2



DELETE语句

DELETE语句从表中删除记录，它的语法如下：

- ◆ DELETE FROM 表名 WHERE 条件

例

- ◆ DELETE FROM 学生档案 WHERE s_id=2

注意：和UPDATE一样，如果省略了 WHERE 子句，表中的所有记录将被删除。所以，一般情况下DELETE语句应带WHERE 子句。例

- ◆ DELETE FROM 学生档案



本讲义主要参考资料

- ◆ 14天自学教程—SQL; Bryan Morgan, Jeef Perkins 著; 邓洪涛, 陈越 译。—北京: 清华大学出版社, 1997
- ◆ SQL联机丛书。