**SQL 简介**

SQL 是用于访问和处理数据库的标准的计算机语言。

**SQL 是什么？**

* SQL，指结构化查询语言，全称是 Structured Query Language。
* SQL 让您可以访问和处理数据库。
* SQL 是一种 ANSI（American National Standards Institute 美国国家标准化组织）标准的计算机语言。

**SQL 能做什么？**

* SQL 面向数据库执行查询
* SQL 可从数据库取回数据
* SQL 可在数据库中插入新的记录
* SQL 可更新数据库中的数据
* SQL 可从数据库删除记录
* SQL 可创建新数据库
* SQL 可在数据库中创建新表
* SQL 可在数据库中创建存储过程
* SQL 可在数据库中创建视图
* SQL 可以设置表、存储过程和视图的权限

**SQL 是一种标准 - 但是...**

虽然 SQL 是一门 ANSI（American National Standards Institute 美国国家标准化组织）标准的计算机语言，但是仍然存在着多种不同版本的 SQL 语言。

然而，为了与 ANSI 标准相兼容，它们必须以相似的方式共同地来支持一些主要的命令（比如 SELECT、UPDATE、DELETE、INSERT、WHERE 等等）。

|  |  |
| --- | --- |
| **lamp** | **注释：**除了 SQL 标准之外，大部分 SQL 数据库程序都拥有它们自己的专有扩展！ |

**在您的网站中使用 SQL**

要创建一个显示数据库中数据的网站，您需要：

* RDBMS 数据库程序（比如 MS Access、SQL Server、MySQL）
* 使用服务器端脚本语言，比如 PHP 或 ASP
* 使用 SQL 来获取您想要的数据
* 使用 HTML / CSS

**RDBMS**

RDBMS 指关系型数据库管理系统，全称 Relational Database Management System。

RDBMS 是 SQL 的基础，同样也是所有现代数据库系统的基础，比如 MS SQL Server、IBM DB2、Oracle、MySQL 以及 Microsoft Access。

RDBMS 中的数据存储在被称为表的数据库对象中。

表是相关的数据项的集合，它由列和行组成。

**SQL 语法**

**数据库表**

一个数据库通常包含一个或多个表。每个表由一个名字标识（例如 "Customers" 或者 "Orders"）。表包含带有数据的记录（行）。

在本教程中，我们将使用众所周知的 Northwind 样本数据库（包含在 MS Access 和 MS SQL Server）。

下面是选自 "Customers" 表的数据：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |

上面的表包含五条记录（每一条对应一个客户）和七个列（CustomerID、CustomerName、ContactName、Address、City、PostalCode 和 Country）。

**SQL 语句**

您需要在数据库上执行的大部分工作都由 SQL 语句完成。

下面的 SQL 语句从 "Customers" 表中选取所有记录：

**实例**

SELECT \* FROM Customers;

在本教程中，我们将为您讲解各种不同的 SQL 语句。

**请记住...**

* SQL 对大小写不敏感：SELECT 与 select 是相同的。

**SQL 语句后面的分号？**

某些数据库系统要求在每条 SQL 语句的末端使用分号。

分号是在数据库系统中分隔每条 SQL 语句的标准方法，这样就可以在对服务器的相同请求中执行一条以上的 SQL 语句。

在本教程中，我们将在每条 SQL 语句的末端使用分号。

**一些最重要的 SQL 命令**

* **SELECT** - 从数据库中提取数据
* **UPDATE** - 更新数据库中的数据
* **DELETE** - 从数据库中删除数据
* **INSERT INTO** - 向数据库中插入新数据
* **CREATE DATABASE** - 创建新数据库
* **ALTER DATABASE** - 修改数据库
* **CREATE TABLE** - 创建新表
* **ALTER TABLE** - 变更（改变）数据库表
* **DROP TABLE** - 删除表
* **CREATE INDEX** - 创建索引（搜索键）
* **DROP INDEX** - 删除索引

# SQL SELECT 语句

SELECT 语句用于从数据库中选取数据。

## SQL SELECT 语句

SELECT 语句用于从数据库中选取数据。

结果被存储在一个结果表中，称为结果集。

### SQL SELECT 语法

SELECT column\_name,column\_name  
FROM table\_name;

与

SELECT \* FROM table\_name;

## 演示数据库

在本教程中，我们将使用众所周知的 Northwind 样本数据库。

下面是选自 "Customers" 表的数据：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |

## SELECT Column 实例

下面的 SQL 语句从 "Customers" 表中选取 "CustomerName" 和 "City" 列：

## 实例

SELECT CustomerName,City FROM Customers;

## SELECT \* 实例

下面的 SQL 语句从 "Customers" 表中选取所有列：

## 实例

SELECT \* FROM Customers;

## 结果集中的导航

大多数数据库软件系统都允许使用编程函数在结果集中进行导航，比如：Move-To-First-Record、Get-Record-Content、Move-To-Next-Record 等等。

类似这些编程函数不在本教程讲解之列。如需学习通过函数调用访问数据的知识，请访问我们的 [ADO 教程](http://www.w3cschool.cc/ado/ado-tutorial.html) 或者 [PHP 教程](http://www.w3cschool.cc/php/php-tutorial.html)。

# SQL SELECT DISTINCT 语句

SELECT DISTINCT 语句用于返回唯一不同的值。

## SQL SELECT DISTINCT 语句

在表中，一个列可能会包含多个重复值，有时您也许希望仅仅列出不同（distinct）的值。

DISTINCT 关键词用于返回唯一不同的值。

### SQL SELECT DISTINCT 语法

SELECT DISTINCT column\_name,column\_name  
FROM table\_name;

## 演示数据库

在本教程中，我们将使用众所周知的 Northwind 样本数据库。

下面是选自 "Customers" 表的数据：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |

## SELECT DISTINCT 实例

下面的 SQL 语句仅从 "Customers" 表的 "City" 列中选取唯一不同的值。从 Company" 列中仅选取唯一不同的值：

## 实例

SELECT DISTINCT City FROM Customers;

# SQL WHERE 子句

WHERE 子句用于过滤记录。

## SQL WHERE 子句

WHERE 子句用于提取那些满足指定标准的记录。

### SQL WHERE 语法

SELECT column\_name,column\_name  
FROM table\_name  
WHERE column\_name operator value;

## 演示数据库

在本教程中，我们将使用众所周知的 Northwind 样本数据库。

下面是选自 "Customers" 表的数据：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |

## WHERE 子句实例

下面的 SQL 语句从 "Customers" 表中选取国家为 "Mexico" 的所有客户：

## 实例

SELECT \* FROM Customers  
WHERE Country='Mexico';

## 文本字段 vs. 数值字段

SQL 使用单引号来环绕文本值（大部分数据库系统也接受双引号）。

如果是数值字段，请不要使用引号。

## 实例

SELECT \* FROM Customers  
WHERE CustomerID=1;

## WHERE 子句中的运算符

下面的运算符可以在 WHERE 子句中使用：

|  |  |
| --- | --- |
| **运算符** | **描述** |
| = | 等于 |
| <> | 不等于。**注释：**在 SQL 的一些版本中，该操作符可被写成 != |
| > | 大于 |
| < | 小于 |
| >= | 大于等于 |
| <= | 小于等于 |
| BETWEEN | 在某个范围内 |
| LIKE | 搜索某种模式 |
| IN | 指定针对某个列的多个可能值 |

# SQL AND & OR 运算符

AND & OR 运算符用于基于一个以上的条件对记录进行过滤。

## SQL AND & OR 运算符

如果第一个条件和第二个条件都成立，则 AND 运算符显示一条记录。

如果第一个条件和第二个条件中只要有一个成立，则 OR 运算符显示一条记录。

## 演示数据库

在本教程中，我们将使用众所周知的 Northwind 样本数据库。

下面是选自 "Customers" 表的数据：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |

## AND 运算符实例

下面的 SQL 语句从 "Customers" 表中选取国家为 "Germany" 且城市为 "Berlin" 的所有客户：

## 实例

SELECT \* FROM Customers  
WHERE Country='Germany'  
AND City='Berlin';

## OR 运算符实例

下面的 SQL 语句从 "Customers" 表中选取城市为 "Berlin" 或者 "München" 的所有客户：

## 实例

SELECT \* FROM Customers  
WHERE City='Berlin'  
OR City='München';

## 结合 AND & OR

您也可以把 AND 和 OR 结合起来（使用圆括号来组成复杂的表达式）。

下面的 SQL 语句从 "Customers" 表中选取国家为 "Germany" 且城市为 "Berlin" 或者 "München" 的所有客户：

## 实例

SELECT \* FROM Customers  
WHERE Country='Germany'  
AND (City='Berlin' OR City='München');

# SQL ORDER BY 关键字

ORDER BY 关键字用于对结果集进行排序。

## SQL ORDER BY 关键字

ORDER BY 关键字用于对结果集按照一个列或者多个列进行排序。

ORDER BY 关键字默认按照升序对记录进行排序。如果需要按照降序对记录进行排序，您可以使用 DESC 关键字。

### SQL ORDER BY 语法

SELECT column\_name,column\_name  
FROM table\_name  
ORDER BY column\_name,column\_name ASC|DESC;

## 演示数据库

在本教程中，我们将使用众所周知的 Northwind 样本数据库。

下面是选自 "Customers" 表的数据：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |

## ORDER BY 实例

下面的 SQL 语句从 "Customers" 表中选取所有客户，并按照 "Country" 列排序：

## 实例

SELECT \* FROM Customers  
ORDER BY Country;

## ORDER BY DESC 实例

下面的 SQL 语句从 "Customers" 表中选取所有客户，并按照 "Country" 列降序排序：

## 实例

SELECT \* FROM Customers  
ORDER BY Country DESC;

## ORDER BY Several Columns 实例

下面的 SQL 语句从 "Customers" 表中选取所有客户，并按照 "Country" 和 "CustomerName" 列排序：

## 实例

SELECT \* FROM Customers  
ORDER BY Country,CustomerName;

# SQL INSERT INTO 语句

INSERT INTO 语句用于向表中插入新记录。

## SQL INSERT INTO 语句

INSERT INTO 语句用于向表中插入新记录。

### SQL INSERT INTO 语法

INSERT INTO 语句可以有两种编写形式。

第一种形式无需指定要插入数据的列名，只需提供被插入的值即可：

INSERT INTO table\_name  
VALUES (value1,value2,value3,...);

第二种形式需要指定列名及被插入的值：

INSERT INTO table\_name (column1,column2,column3,...)  
VALUES (value1,value2,value3,...);

## 演示数据库

在本教程中，我们将使用众所周知的 Northwind 样本数据库。

下面是选自 "Customers" 表的数据：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 87 | Wartian Herkku | Pirkko Koskitalo | Torikatu 38 | Oulu | 90110 | Finland |
| 88 | Wellington Importadora | Paula Parente | Rua do Mercado, 12 | Resende | 08737-363 | Brazil |
| 89 | White Clover Markets | Karl Jablonski | 305 - 14th Ave. S. Suite 3B | Seattle | 98128 | USA |
| 90 | Wilman Kala | Matti Karttunen | Keskuskatu 45 | Helsinki | 21240 | Finland |
| 91 | Wolski | Zbyszek | ul. Filtrowa 68 | Walla | 01-012 | Poland |

## INSERT INTO 实例

假设我们要向 "Customers" 表中插入一个新行。

我们可以使用下面的 SQL 语句：

## 实例

INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)  
VALUES ('Cardinal','Tom B. Erichsen','Skagen 21','Stavanger','4006','Norway');

现在，选自 "Customers" 表的数据如下所示：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 87 | Wartian Herkku | Pirkko Koskitalo | Torikatu 38 | Oulu | 90110 | Finland |
| 88 | Wellington Importadora | Paula Parente | Rua do Mercado, 12 | Resende | 08737-363 | Brazil |
| 89 | White Clover Markets | Karl Jablonski | 305 - 14th Ave. S. Suite 3B | Seattle | 98128 | USA |
| 90 | Wilman Kala | Matti Karttunen | Keskuskatu 45 | Helsinki | 21240 | Finland |
| 91 | Wolski | Zbyszek | ul. Filtrowa 68 | Walla | 01-012 | Poland |
| 92 | Cardinal | Tom B. Erichsen | Skagen 21 | Stavanger | 4006 | Norway |

|  |  |
| --- | --- |
| **lamp** | **您是否注意到，我们没有向 CustomerID 字段插入任何数字？** CustomerID 列是自动更新的，表中的每条记录都有一个唯一的数字。 |

## 在指定的列插入数据

我们也可以在指定的列插入数据。

下面的 SQL 语句将插入一个新行，但是只在 "CustomerName"、"City" 和 "Country" 列插入数据（CustomerID 字段会自动更新）：

## 实例

INSERT INTO Customers (CustomerName, City, Country)  
VALUES ('Cardinal', 'Stavanger', 'Norway');

现在，选自 "Customers" 表的数据如下所示：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 87 | Wartian Herkku | Pirkko Koskitalo | Torikatu 38 | Oulu | 90110 | Finland |
| 88 | Wellington Importadora | Paula Parente | Rua do Mercado, 12 | Resende | 08737-363 | Brazil |
| 89 | White Clover Markets | Karl Jablonski | 305 - 14th Ave. S. Suite 3B | Seattle | 98128 | USA |
| 90 | Wilman Kala | Matti Karttunen | Keskuskatu 45 | Helsinki | 21240 | Finland |
| 91 | Wolski | Zbyszek | ul. Filtrowa 68 | Walla | 01-012 | Poland |
| 92 | Cardinal | null | null | Stavanger | null | Norway |

# SQL UPDATE 语句

UPDATE 语句用于更新表中的记录。

## SQL UPDATE 语句

UPDATE 语句用于更新表中已存在的记录。

### SQL UPDATE 语法

UPDATE table\_name  
SET column1=value1,column2=value2,...  
WHERE some\_column=some\_value;

|  |  |
| --- | --- |
| **lamp** | **请注意 SQL UPDATE 语句中的 WHERE 子句！** WHERE 子句规定哪条记录或者哪些记录需要更新。如果您省略了 WHERE 子句，所有的记录都将被更新！ |

## 演示数据库

在本教程中，我们将使用众所周知的 Northwind 样本数据库。

下面是选自 "Customers" 表的数据：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |

## SQL UPDATE 实例

假设我们要把客户 "Alfreds Futterkiste" 更新为一个新的联系人和城市。

我们使用下面的 SQL 语句：

## 实例

UPDATE Customers  
SET ContactName='Alfred Schmidt', City='Hamburg'  
WHERE CustomerName='Alfreds Futterkiste';

现在，选自 "Customers" 表的数据如下所示：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Alfred Schmidt | Obere Str. 57 | Hamburg | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |

## Update 警告！

在更新记录时要格外小心！在上面的实例中，如果我们省略了 WHERE 子句，如下所示：

UPDATE Customers  
SET ContactName='Alfred Schmidt', City='Hamburg';

"Customers" 表将如下所示：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Alfred Schmidt | Obere Str. 57 | Hamburg | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Alfred Schmidt | Avda. de la Constitución 2222 | Hamburg | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Alfred Schmidt | Mataderos 2312 | Hamburg | 05023 | Mexico |
| 4 | Around the Horn | Alfred Schmidt | 120 Hanover Sq. | Hamburg | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Alfred Schmidt | Berguvsvägen 8 | Hamburg | S-958 22 | Sweden |

# SQL DELETE 语句

DELETE 语句用于删除表中的记录。

## SQL DELETE 语句

DELETE 语句用于删除表中的行。

### SQL DELETE 语法

DELETE FROM table\_name  
WHERE some\_column=some\_value;

|  |  |
| --- | --- |
| **lamp** | **请注意 SQL DELETE 语句中的 WHERE 子句！** WHERE 子句规定哪条记录或者哪些记录需要删除。如果您省略了 WHERE 子句，所有的记录都将被删除！ |

## 演示数据库

在本教程中，我们将使用众所周知的 Northwind 样本数据库。

下面是选自 "Customers" 表的数据：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |

## SQL DELETE 实例

假设我们要从 "Customers" 表中删除客户 "Alfreds Futterkiste"。

我们使用下面的 SQL 语句：

## 实例

DELETE FROM Customers  
WHERE CustomerName='Alfreds Futterkiste' AND ContactName='Maria Anders';

现在，"Customers" 表如下所示：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |

## 删除所有数据

您可以在不删除表的情况下，删除表中所有的行。这意味着表结构、属性、索引将保持不变：

DELETE FROM table\_name;  
  
or  
  
DELETE \* FROM table\_name;

**注释：**在删除记录时要格外小心！因为您不能重来！

# SQL SELECT TOP 子句

## SQL SELECT TOP 子句

SELECT TOP 子句用于规定要返回的记录的数目。

SELECT TOP 子句对于拥有数千条记录的大型表来说，是非常有用的。

**注释：**并非所有的数据库系统都支持 SELECT TOP 子句。

### SQL Server / MS Access 语法

SELECT TOP number|percent column\_name(s)  
FROM table\_name;

## MySQL 和 Oracle 中的 SQL SELECT TOP 是等价的

### MySQL 语法

SELECT column\_name(s)  
FROM table\_name  
LIMIT number;

### 实例

SELECT \*  
FROM Persons  
LIMIT 5;

### Oracle 语法

SELECT column\_name(s)  
FROM table\_name  
WHERE ROWNUM <= number;

### 实例

SELECT \*  
FROM Persons  
WHERE ROWNUM <=5;

## 演示数据库

在本教程中，我们将使用众所周知的 Northwind 样本数据库。

下面是选自 "Customers" 表的数据：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |

## SQL SELECT TOP 实例

下面的 SQL 语句从 "Customers" 表中选取头两条记录：

## 实例

SELECT TOP 2 \* FROM Customers;

## SQL SELECT TOP PERCENT 实例

下面的 SQL 语句从 "Customers" 表中选取前面 50% 的记录：

## 实例

SELECT TOP 50 PERCENT \* FROM Customers;

# SQL LIKE 操作符

LIKE 操作符用于在 WHERE 子句中搜索列中的指定模式。

## SQL LIKE 操作符

LIKE 操作符用于在 WHERE 子句中搜索列中的指定模式。

### SQL LIKE 语法

SELECT column\_name(s)  
FROM table\_name  
WHERE column\_name LIKE pattern;

## 演示数据库

在本教程中，我们将使用众所周知的 Northwind 样本数据库。

下面是选自 "Customers" 表的数据：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |

## SQL LIKE 操作符实例

下面的 SQL 语句选取 City 以字母 "s" 开始的所有客户：

## 实例

SELECT \* FROM Customers  
WHERE City LIKE 's%';

**提示：**"%" 符号用于在模式的前后定义通配符（缺省字母）。您将在下一章中学习更多有关通配符的知识。

下面的 SQL 语句选取 City 以字母 "s" 结尾的所有客户：

## 实例

SELECT \* FROM Customers  
WHERE City LIKE '%s';

下面的 SQL 语句选取 Country 包含模式 "land" 的所有客户：

## 实例

SELECT \* FROM Customers  
WHERE Country LIKE '%land%';

通过使用 NOT 关键字，您可以选取不匹配模式的记录。

下面的 SQL 语句选取 Country 不包含模式 "land" 的所有客户：

## 实例

SELECT \* FROM Customers  
WHERE Country NOT LIKE '%land%';

# SQL 通配符

通配符可用于替代字符串中的任何其他字符。

## SQL 通配符

在 SQL 中，通配符与 SQL LIKE 操作符一起使用。

SQL 通配符用于搜索表中的数据。

在 SQL 中，可使用一下通配符：

|  |  |
| --- | --- |
| **通配符** | **描述** |
| % | 替代 0 个或多个字符 |
| \_ | 替代一个字符 |
| [charlist] | 字符列中的任何单一字符 |
| [^charlist] or [!charlist] | 不在字符列中的任何单一字符 |

## 演示数据库

在本教程中，我们将使用众所周知的 Northwind 样本数据库。

下面是选自 "Customers" 表的数据：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |

## 使用 SQL % 通配符

下面的 SQL 语句选取 City 以字母 "ber" 开始的所有客户：

## 实例

SELECT \* FROM Customers  
WHERE City LIKE 'ber%';

下面的 SQL 语句选取 City 包含模式 "es" 的所有客户：

## 实例

SELECT \* FROM Customers  
WHERE City LIKE '%es%';

## 使用 SQL \_ 通配符

下面的 SQL 语句选取 City 以一个任意字符开始，然后是 "erlin" 的所有客户：

## 实例

SELECT \* FROM Customers  
WHERE City LIKE '\_erlin';

下面的 SQL 语句选取 City 以 "L" 开始，然后是一个任意字符，然后是 "n"，然后是一个任意字符，然后是 "on" 的所有客户：

## 实例

SELECT \* FROM Customers  
WHERE City LIKE 'L\_n\_on';

## 使用 SQL [charlist] 通配符

下面的 SQL 语句选取 City 以 "b"、"s" 或 "p" 开始的所有客户：

## 实例

SELECT \* FROM Customers  
WHERE City LIKE '[bsp]%';

下面的 SQL 语句选取 City 以 "a"、"b" 或 "c" 开始的所有客户：

## 实例

SELECT \* FROM Customers  
WHERE City LIKE '[a-c]%';

下面的 SQL 语句选取 City 不以 "b"、"s" 或 "p" 开始的所有客户：

## 实例

SELECT \* FROM Customers  
WHERE City LIKE '[!bsp]%';

# SQL IN 操作符

## IN 操作符

IN 操作符允许您在 WHERE 子句中规定多个值。

### SQL IN 语法

SELECT column\_name(s)  
FROM table\_name  
WHERE column\_name IN (value1,value2,...);

## 演示数据库

在本教程中，我们将使用众所周知的 Northwind 样本数据库。

下面是选自 "Customers" 表的数据：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |

## IN 操作符实例

下面的 SQL 语句选取 City 为 "Paris" 或 "London" 的所有客户：

## 实例

SELECT \* FROM Customers  
WHERE City IN ('Paris','London');

# SQL BETWEEN 操作符

BETWEEN 操作符用于选取介于两个值之间的数据范围内的值。

## SQL BETWEEN 操作符

BETWEEN 操作符选取介于两个值之间的数据范围内的值。这些值可以是数值、文本或者日期。

### SQL BETWEEN 语法

SELECT column\_name(s)  
FROM table\_name  
WHERE column\_nameBETWEEN value1 AND value2;

## 演示数据库

在本教程中，我们将使用众所周知的 Northwind 样本数据库。

下面是选自 "Products" 表的数据：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ProductID** | **ProductName** | **SupplierID** | **CategoryID** | **Unit** | **Price** |
| 1 | Chais | 1 | 1 | 10 boxes x 20 bags | 18 |
| 2 | Chang | 1 | 1 | 24 - 12 oz bottles | 19 |
| 3 | Aniseed Syrup | 1 | 2 | 12 - 550 ml bottles | 10 |
| 4 | Chef Anton's Cajun Seasoning | 1 | 2 | 48 - 6 oz jars | 22 |
| 5 | Chef Anton's Gumbo Mix | 1 | 2 | 36 boxes | 21.35 |

## BETWEEN 操作符实例

下面的 SQL 语句选取价格介于 10 和 20 之间的所有产品：

## 实例

SELECT \* FROM Products  
WHERE Price BETWEEN 10 AND 20;

## NOT BETWEEN 操作符实例

如需显示不在上面实例范围内的产品，请使用 NOT BETWEEN：

## 实例

SELECT \* FROM Products  
WHERE Price NOT BETWEEN 10 AND 20;

## 带有 IN 的 BETWEEN 操作符实例

下面的 SQL 语句选取价格介于 10 和 20 之间但 CategoryID 不为 1、2 或 3 的所有产品：

## 实例

SELECT \* FROM Products  
WHERE (Price BETWEEN 10 AND 20)  
AND NOT CategoryID IN (1,2,3);

## 带有文本值的 BETWEEN 操作符实例

下面的 SQL 语句选取 ProductName 以介于 'C' 和 'M' 之间字母开始的所有产品：

## 实例

SELECT \* FROM Products  
WHERE ProductName BETWEEN 'C' AND 'M';

## 带有文本值的 NOT BETWEEN 操作符实例

下面的 SQL 语句选取 ProductName 不以介于 'C' 和 'M' 之间字母开始的所有产品：

## 实例

SELECT \* FROM Products  
WHERE ProductName NOT BETWEEN 'C' AND 'M';

## 示例表

下面是选自 "Orders" 表的数据：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| 10248 | 90 | 5 | 7/4/1996 | 3 |
| 10249 | 81 | 6 | 7/5/1996 | 1 |
| 10250 | 34 | 4 | 7/8/1996 | 2 |
| 10251 | 84 | 3 | 7/9/1996 | 1 |
| 10252 | 76 | 4 | 7/10/1996 | 2 |

## 带有日期值的 BETWEEN 操作符实例

下面的 SQL 语句选取 OrderDate 介于 '04-July-1996' 和 '09-July-1996' 之间的所有订单：

## 实例

SELECT \* FROM Orders  
WHERE OrderDate BETWEEN #07/04/1996# AND #07/09/1996#;

|  |  |
| --- | --- |
| **lamp** | **请注意，在不同的数据库中，BETWEEN 操作符会产生不同的结果！** 在某些数据库中，BETWEEN 选取介于两个值之间但不包括两个测试值的字段。 在某些数据库中，BETWEEN 选取介于两个值之间且包括两个测试值的字段。 在某些数据库中，BETWEEN 选取介于两个值之间且包括第一个测试值但不包括最后一个测试值的字段。  **因此，请检查您的数据库是如何处理 BETWEEN 操作符！** |

# SQL 别名（Aliases）

通过使用 SQL，可以为表名称或列名称指定别名（Alias）。

## SQL 别名（Aliases）

通过使用 SQL，可以为表名称或列名称指定别名（Alias）。

基本上，创建别名是为了让列名称的可读性更强。

### 列的 SQL Alias 语法

SELECT column\_name AS alias\_name  
FROM table\_name;

### 表的 SQL Alias 语法

SELECT column\_name(s)  
FROM table\_nameAS alias\_name;

## 演示数据库

在本教程中，我们将使用众所周知的 Northwind 样本数据库。

下面是选自 "Customers" 表的数据：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |

下面是选自 "Orders" 表的数据：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| 10643 | 1 | 6 | 1997-08-25 | 1 |
| 10644 | 88 | 3 | 1997-08-25 | 2 |
| 10645 | 34 | 4 | 1997-08-26 | 1 |

## 列的 Alias 实例

下面的 SQL 语句指定了两个别名，一个是 CustomerName 列的别名，一个是 ContactName 列的别名。**提示：**如果列名称包含空格，要求使用双引号或方括号：

## 实例

SELECT CustomerName AS Customer, ContactName AS [Contact Person]  
FROM Customers;

在下面的 SQL 语句中，我们把四个列（Address、City、PostalCode 和 Country）结合在一起，并创建一个名为 "Address" 的别名：

## 实例

SELECT CustomerName, Address+', '+City+', '+PostalCode+', '+Country AS Address  
FROM Customers;

## 表的 Alias 实例

下面的 SQL 语句选取来自客户 "Alfreds Futterkiste" 的所有订单。我们使用 "Customers" 和 "Orders" 表，并分别为它们指定表别名 "c" 和 "o"（通过使用别名让 SQL 更简短）：

## 实例

SELECT o.OrderID, o.OrderDate, c.CustomerName  
FROM Customers AS c, Orders AS o  
WHERE c.CustomerName='Alfreds Futterkiste';

不带别名的相同的 SQL 语句：

## 实例

SELECT Orders.OrderID, Orders.OrderDate, Customers.CustomerName  
FROM Customers, Orders  
WHERE Customers.CustomerName='Alfreds Futterkiste';

在下面的情况下，使用别名很有用：

* 在查询中涉及超过一个表
* 在查询中使用了函数
* 列名称很长或者可读性差
* 需要把两个列或者多个列结合在一起

# SQL 连接（Joins）

SQL join 用于把来自两个或多个表的行结合起来。

## SQL JOIN

SQL JOIN 子句用于把来自两个或多个表的行结合起来，基于这些表之间的共同字段。

最常见的 JOIN 类型：**SQL INNER JOIN（简单的 JOIN）**。 SQL INNER JOIN 从多个表中返回满足 JOIN 条件的所有行。

让我们看看选自 "Orders" 表的数据：

|  |  |  |
| --- | --- | --- |
| **OrderID** | **CustomerID** | **OrderDate** |
| 10308 | 2 | 1996-09-18 |
| 10309 | 37 | 1996-09-19 |
| 10310 | 77 | 1996-09-20 |

然后，看看选自 "Customers" 表的数据：

|  |  |  |  |
| --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mexico |

请注意，"Orders" 表中的 "CustomerID" 列指向 "Customers" 表中的客户。上面这两个表是通过 "CustomerID" 列联系起来的。

然后，如果我们运行下面的 SQL 语句（包含 INNER JOIN）：

## 实例

SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate  
FROM Orders  
INNER JOIN Customers  
ON Orders.CustomerID=Customers.CustomerID;

运行结果如下所示：

|  |  |  |
| --- | --- | --- |
| **OrderID** | **CustomerName** | **OrderDate** |
| 10308 | Ana Trujillo Emparedados y helados | 9/18/1996 |
| 10365 | Antonio Moreno Taquería | 11/27/1996 |
| 10383 | Around the Horn | 12/16/1996 |
| 10355 | Around the Horn | 11/15/1996 |
| 10278 | Berglunds snabbköp | 8/12/1996 |

## 不同的 SQL JOIN

在我们继续讲解实例之前，我们先列出您可以使用的不同的 SQL JOIN 类型：

* **INNER JOIN**：如果表中有至少一个匹配，则返回行
* **LEFT JOIN**：即使右表中没有匹配，也从左表返回所有的行
* **RIGHT JOIN**：即使左表中没有匹配，也从右表返回所有的行
* **FULL JOIN**：只要其中一个表中存在匹配，则返回行

# SQL INNER JOIN 关键字

## SQL INNER JOIN 关键字

INNER JOIN 关键字在表中存在至少一个匹配时返回行。

### SQL INNER JOIN 语法

SELECT column\_name(s)  
FROM table1  
INNER JOIN table2  
ON table1.column\_name=table2.column\_name;

或：

SELECT column\_name(s)  
FROM table1  
JOIN table2  
ON table1.column\_name=table2.column\_name;

**注释：**INNER JOIN 与 JOIN 是相同的。



## 演示数据库

在本教程中，我们将使用众所周知的 Northwind 样本数据库。

下面是选自 "Customers" 表的数据：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |

选自 "Orders" 表的数据：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| 10308 | 2 | 7 | 1996-09-18 | 3 |
| 10309 | 37 | 3 | 1996-09-19 | 1 |
| 10310 | 77 | 8 | 1996-09-20 | 2 |

## SQL INNER JOIN 实例

下面的 SQL 语句将返回有下订单的所有客户：

## 实例

SELECT Customers.CustomerName, Orders.OrderID  
FROM Customers  
INNER JOIN Orders  
ON Customers.CustomerID=Orders.CustomerID  
ORDER BY Customers.CustomerName;

**注释：**INNER JOIN 关键字在表中存在至少一个匹配时返回行。如果 "Customers" 表中的行在 "Orders" 中没有匹配，则不会列出这些行。

# SQL LEFT JOIN 关键字

## SQL LEFT JOIN 关键字

LEFT JOIN 关键字从左表（table1）返回所有的行，即使右表（table2）中没有匹配。如果右表中没有匹配，则结果为 NULL。

### SQL LEFT JOIN 语法

SELECT column\_name(s)  
FROM table1  
LEFT JOIN table2  
ON table1.column\_name=table2.column\_name;

或：

SELECT column\_name(s)  
FROM table1  
LEFT OUTER JOIN table2  
ON table1.column\_name=table2.column\_name;

**注释：**在某些数据库中，LEFT JOIN 称为 LEFT OUTER JOIN。



## 演示数据库

在本教程中，我们将使用众所周知的 Northwind 样本数据库。

下面是选自 "Customers" 表的数据：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |

选自 "Orders" 表的数据：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| 10308 | 2 | 7 | 1996-09-18 | 3 |
| 10309 | 37 | 3 | 1996-09-19 | 1 |
| 10310 | 77 | 8 | 1996-09-20 | 2 |

## SQL LEFT JOIN 实例

下面的 SQL 语句将返回所有客户及他们的订单（如果有的话）：

## 实例

SELECT Customers.CustomerName, Orders.OrderID  
FROM Customers  
LEFT JOIN Orders  
ON Customers.CustomerID=Orders.CustomerID  
ORDER BY Customers.CustomerName;

**注释：**LEFT JOIN 关键字从左表（Customers）返回所有的行，即使右表（Orders）中没有匹配。

# SQL RIGHT JOIN 关键字

## SQL RIGHT JOIN 关键字

RIGHT JOIN 关键字从右表（table2）返回所有的行，即使左表（table1）中没有匹配。如果左表中没有匹配，则结果为 NULL。

### SQL RIGHT JOIN 语法

SELECT column\_name(s)  
FROM table1  
RIGHT JOIN table2  
ON table1.column\_name=table2.column\_name;

或：

SELECT column\_name(s)  
FROM table1  
RIGHT OUTER JOIN table2  
ON table1.column\_name=table2.column\_name;

**注释：**在某些数据库中，RIGHT JOIN 称为 RIGHT OUTER JOIN。



## 演示数据库

在本教程中，我们将使用众所周知的 Northwind 样本数据库。

下面是选自 "Orders" 表的数据：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| 10308 | 2 | 7 | 1996-09-18 | 3 |
| 10309 | 37 | 3 | 1996-09-19 | 1 |
| 10310 | 77 | 8 | 1996-09-20 | 2 |

选自 "Employees" 表的数据：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EmployeeID** | **LastName** | **FirstName** | **BirthDate** | **Photo** | **Notes** |
| 1 | Davolio | Nancy | 12/8/1968 | EmpID1.pic | Education includes a BA in psychology..... |
| 2 | Fuller | Andrew | 2/19/1952 | EmpID2.pic | Andrew received his BTS commercial and.... |
| 3 | Leverling | Janet | 8/30/1963 | EmpID3.pic | Janet has a BS degree in chemistry.... |

## SQL RIGHT JOIN 实例

下面的 SQL 语句将返回所有员工及他们处理的订单：

## 实例

SELECT Orders.OrderID, Employees.FirstName  
FROM Orders  
RIGHT JOIN Employees  
ON Orders.EmployeeID=Employees.EmployeeID  
ORDER BY Orders.OrderID;

**注释：**RIGHT JOIN 关键字从右表（Employees）返回所有的行，即使左表（Orders）中没有匹配。

# SQL FULL OUTER JOIN 关键字

## SQL FULL OUTER JOIN 关键字

FULL OUTER JOIN 关键字只要左表（table1）和右表（table2）其中一个表中存在匹配，则返回行.

FULL OUTER JOIN 关键字结合了 LEFT JOIN 和 RIGHT JOIN 的结果。

### SQL FULL OUTER JOIN 语法

SELECT column\_name(s)  
FROM table1  
FULL OUTER JOIN table2  
ON table1.column\_name=table2.column\_name;



## 演示数据库

在本教程中，我们将使用众所周知的 Northwind 样本数据库。

下面是选自 "Customers" 表的数据：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |

选自 "Orders" 表的数据：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| 10308 | 2 | 7 | 1996-09-18 | 3 |
| 10309 | 37 | 3 | 1996-09-19 | 1 |
| 10310 | 77 | 8 | 1996-09-20 | 2 |

## SQL FULL OUTER JOIN 实例

下面的 SQL 语句选取所有的客户及所有的订单：

SELECT Customers.CustomerName, Orders.OrderID  
FROM Customers  
FULL OUTER JOIN Orders  
ON Customers.CustomerID=Orders.CustomerID  
ORDER BY Customers.CustomerName;

选自结果集中的数据如下所示：

|  |  |
| --- | --- |
| **CustomerName** | **OrderID** |
| Alfreds Futterkiste |  |
| Ana Trujillo Emparedados y helados | 10308 |
| Antonio Moreno Taquería | 10365 |
|  | 10382 |
|  | 10351 |

**注释：**FULL OUTER JOIN 关键字返回左表（Customers）和右表（Orders）中所有的行。如果 "Customers" 表中的行在 "Orders" 中没有匹配或者 "Orders" 表中的行在 "Customers" 表中没有匹配，也会列出这些行。

# SQL UNION 操作符

SQL UNION 操作符合并两个或多个 SELECT 语句的结果。

## SQL UNION 操作符

UNION 操作符用于合并两个或多个 SELECT 语句的结果集。

请注意，UNION 内部的每个 SELECT 语句必须拥有相同数量的列。列也必须拥有相似的数据类型。同时，每个 SELECT 语句中的列的顺序必须相同。

### SQL UNION 语法

SELECT column\_name(s) FROM table1  
UNION  
SELECT column\_name(s) FROM table2;

**注释：**默认地，UNION 操作符选取不同的值。如果允许重复的值，请使用 UNION ALL。

### SQL UNION ALL 语法

SELECT column\_name(s) FROM table1  
UNION ALL  
SELECT column\_name(s) FROM table2;

**注释：**UNION 结果集中的列名总是等于 UNION 中第一个 SELECT 语句中的列名。

## 演示数据库

在本教程中，我们将使用众所周知的 Northwind 样本数据库。

下面是选自 "Customers" 表的数据：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |

选自 "Suppliers" 表的数据：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **SupplierID** | **SupplierName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Exotic Liquid | Charlotte Cooper | 49 Gilbert St. | Londona | EC1 4SD | UK |
| 2 | New Orleans Cajun Delights | Shelley Burke | P.O. Box 78934 | New Orleans | 70117 | USA |
| 3 | Grandma Kelly's Homestead | Regina Murphy | 707 Oxford Rd. | Ann Arbor | 48104 | USA |

## SQL UNION 实例

下面的 SQL 语句从 "Customers" 和 "Suppliers" 表中选取所有**不同的**城市（只有不同的值）：

## 实例

SELECT City FROM Customers  
UNION  
SELECT City FROM Suppliers  
ORDER BY City;

**注释：**UNION 不能用于列出两个表中所有的城市。如果一些客户和供应商来自同一个城市，每个城市只会列出一次。UNION 只会选取不同的值。请使用 UNION ALL 来选取重复的值！

## SQL UNION ALL 实例

下面的 SQL 语句使用 UNION ALL 从 "Customers" 和 "Suppliers" 表中选取**所有的**城市（也有重复的值）：

## 实例

SELECT City FROM Customers  
UNION ALL  
SELECT City FROM Suppliers  
ORDER BY City;

## 带有 WHERE 的 SQL UNION ALL

下面的 SQL 语句使用 UNION ALL 从 "Customers" 和 "Suppliers" 表中选取**所有的**德国城市（也有重复的值）：

## 实例

SELECT City, Country FROM Customers  
WHERE Country='Germany'  
UNION ALL  
SELECT City, Country FROM Suppliers  
WHERE Country='Germany'  
ORDER BY City;

# SQL SELECT INTO 语句

通过 SQL，您可以从一个表复制信息到另一个表。

SELECT INTO 语句从一个表复制数据，然后把数据插入到另一个新表中。

## SQL SELECT INTO 语句

SELECT INTO 语句从一个表复制数据，然后把数据插入到另一个新表中。

### SQL SELECT INTO 语法

我们可以复制所有的列插入到新表中：

SELECT \*  
INTO newtable [IN externaldb]  
FROM table1;

或者只复制希望的列插入到新表中：

SELECT column\_name(s)  
INTO newtable [IN externaldb]  
FROM table1;

|  |  |
| --- | --- |
| **lamp** | **提示：**新表将会使用 SELECT 语句中定义的列名称和类型进行创建。您可以使用 AS 子句来应用新名称。 |

## SQL SELECT INTO 实例

创建 Customers 的备份复件：

SELECT \*  
INTO CustomersBackup2013  
FROM Customers;

请使用 IN 子句来复制表到另一个数据库中：

SELECT \*  
INTO CustomersBackup2013 IN 'Backup.mdb'  
FROM Customers;

只复制一些列插入到新表中：

SELECT CustomerName, ContactName  
INTO CustomersBackup2013  
FROM Customers;

只复制德国的客户插入到新表中：

SELECT \*  
INTO CustomersBackup2013  
FROM Customers  
WHERE Country='Germany';

复制多个表中的数据插入到新表中：

SELECT Customers.CustomerName, Orders.OrderID  
INTO CustomersOrderBackup2013  
FROM Customers  
LEFT JOIN Orders  
ON Customers.CustomerID=Orders.CustomerID;

**提示：**SELECT INTO 语句可用于通过另一种模式创建一个新的空表。只需要添加促使查询没有数据返回的 WHERE 子句即可：

SELECT \*  
INTO newtable  
FROM table1  
WHERE 1=0;

# SQL INSERT INTO SELECT 语句

通过 SQL，您可以从一个表复制信息到另一个表。

INSERT INTO SELECT 语句从一个表复制数据，然后把数据插入到一个已存在的表中。

## SQL INSERT INTO SELECT 语句

INSERT INTO SELECT 语句从一个表复制数据，然后把数据插入到一个已存在的表中。目标表中任何已存在的行都不会受影响。

### SQL INSERT INTO SELECT 语法

我们可以从一个表中复制所有的列插入到另一个已存在的表中：

INSERT INTO table2  
SELECT \* FROM table1;

或者我们可以只复制希望的列插入到另一个已存在的表中：

INSERT INTO table2  
(column\_name(s))  
SELECT column\_name(s)  
FROM table1;

## 演示数据库

在本教程中，我们将使用众所周知的 Northwind 样本数据库。

下面是选自 "Customers" 表的数据：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |

选自 "Suppliers" 表的数据：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **SupplierID** | **SupplierName** | **ContactName** | **Address** | **City** | **Postal Code** | **Country** | **Phone** |
| 1 | Exotic Liquid | Charlotte Cooper | 49 Gilbert St. | Londona | EC1 4SD | UK | (171) 555-2222 |
| 2 | New Orleans Cajun Delights | Shelley Burke | P.O. Box 78934 | New Orleans | 70117 | USA | (100) 555-4822 |
| 3 | Grandma Kelly's Homestead | Regina Murphy | 707 Oxford Rd. | Ann Arbor | 48104 | USA | (313) 555-5735 |

## SQL INSERT INTO SELECT 实例

只复制 "Suppliers" 中的一些列插入到 "Customers" 中：

## 实例

INSERT INTO Customers (CustomerName, Country)  
SELECT SupplierName, Country FROM Suppliers;

只复制德国的供应商插入到 "Customers" 中：

## 实例

INSERT INTO Customers (CustomerName, Country)  
SELECT SupplierName, Country FROM Suppliers  
WHERE Country='Germany';

# SQL CREATE DATABASE 语句

## SQL CREATE DATABASE 语句

CREATE DATABASE 语句用于创建数据库。

### SQL CREATE DATABASE 语法

CREATE DATABASE dbname;

## SQL CREATE DATABASE 实例

下面的 SQL 语句创建一个名为 "my\_db" 的数据库：

CREATE DATABASE my\_db;

数据库表可以通过 CREATE TABLE 语句来添加。

# SQL CREATE TABLE 语句

## SQL CREATE TABLE 语句

CREATE TABLE 语句用于创建数据库中的表。

表由行和列组成，每个表都必须有个表名。

### SQL CREATE TABLE 语法

CREATE TABLE table\_name  
(  
column\_name1 data\_type(size),  
column\_name2 data\_type(size),  
column\_name3 data\_type(size),  
....  
);

column\_name 参数规定表中列的名称。

data\_type 参数规定列的数据类型（例如 varchar、integer、decimal、date 等等）。

size 参数规定表中列的最大长度。

**提示：**如需了解 MS Access、MySQL 和 SQL Server 中可用的数据类型，请访问我们完整的 [数据类型参考手册](http://www.w3cschool.cc/sql/sql-datatypes.html)。

## SQL CREATE TABLE 实例

现在我们想要创建一个名为 "Persons" 的表，包含五列：PersonID、LastName、FirstName、Address 和 City。

我们使用下面的 CREATE TABLE 语句：

## 实例

CREATE TABLE Persons  
(  
PersonID int,  
LastName varchar(255),  
FirstName varchar(255),  
Address varchar(255),  
City varchar(255)  
);

PersonID 列的数据类型是 int，包含整数。

LastName、FirstName、Address 和 City 列的数据类型是 varchar，包含字符，且这些字段的最大长度为 255 个字符。

空的 "Persons" 表如下所示：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PersonID** | **LastName** | **FirstName** | **Address** | **City** |
|  |  |  |  |  |

**提示：**可使用 INSERT INTO 语句向空表写入数据。

# SQL 约束（Constraints）

## SQL 约束（Constraints）

SQL 约束用于规定表中的数据规则。

如果存在违反约束的数据行为，行为会被约束终止。

约束可以在创建表时规定（通过 CREATE TABLE 语句），或者在表创建之后规定（通过 ALTER TABLE 语句）。

### SQL CREATE TABLE + CONSTRAINT 语法

CREATE TABLE table\_name  
(  
column\_name1 data\_type(size) constraint\_name,  
column\_name2 data\_type(size) constraint\_name,  
column\_name3 data\_type(size) constraint\_name,  
....  
);

在 SQL 中，我们有如下约束：

* **NOT NULL** - 指示某列不能存储 NULL 值。
* **UNIQUE** - 保证某列的每行必须有唯一的值。
* **PRIMARY KEY** - NOT NULL 和 UNIQUE 的结合。确保某列（或两个列多个列的结合）有唯一标识，有助于更容易更快速地找到表中的一个特定的记录。
* **FOREIGN KEY** - 保证一个表中的数据匹配另一个表中的值的参照完整性。
* **CHECK** - 保证列中的值符合指定的条件。
* **DEFAULT** - 规定没有给列赋值时的默认值。

在下面的章节，我们会详细讲解每一种约束。

**SQL NOT NULL 约束**

在默认的情况下，表的列接受 NULL 值。

**SQL NOT NULL 约束**

NOT NULL 约束强制列不接受 NULL 值。

NOT NULL 约束强制字段始终包含值。这意味着，如果不向字段添加值，就无法插入新记录或者更新记录。

下面的 SQL 强制 "P\_Id" 列和 "LastName" 列不接受 NULL 值：

CREATE TABLE Persons  
(  
P\_Id int NOT NULL,  
LastName varchar(255) NOT NULL,  
FirstName varchar(255),  
Address varchar(255),  
City varchar(255)  
)

**SQL UNIQUE 约束**

**SQL UNIQUE 约束**

UNIQUE 约束唯一标识数据库表中的每条记录。

UNIQUE 和 PRIMARY KEY 约束均为列或列集合提供了唯一性的保证。

PRIMARY KEY 约束拥有自动定义的 UNIQUE 约束。

请注意，每个表可以有多个 UNIQUE 约束，但是每个表只能有一个 PRIMARY KEY 约束。

**CREATE TABLE 时的 SQL UNIQUE 约束**

下面的 SQL 在 "Persons" 表创建时在 "P\_Id" 列上创建 UNIQUE 约束：

**MySQL：**

CREATE TABLE Persons  
(  
P\_Id int NOT NULL,  
LastName varchar(255) NOT NULL,  
FirstName varchar(255),  
Address varchar(255),  
City varchar(255),  
UNIQUE (P\_Id)  
)

**SQL Server / Oracle / MS Access：**

CREATE TABLE Persons  
(  
P\_Id int NOT NULL UNIQUE,  
LastName varchar(255) NOT NULL,  
FirstName varchar(255),  
Address varchar(255),  
City varchar(255)  
)

如需命名 UNIQUE 约束，并定义多个列的 UNIQUE 约束，请使用下面的 SQL 语法：

**MySQL / SQL Server / Oracle / MS Access：**

CREATE TABLE Persons  
(  
P\_Id int NOT NULL,  
LastName varchar(255) NOT NULL,  
FirstName varchar(255),  
Address varchar(255),  
City varchar(255),  
CONSTRAINT uc\_PersonID UNIQUE (P\_Id,LastName)  
)

**ALTER TABLE 时的 SQL UNIQUE 约束**

当表已被创建时，如需在 "P\_Id" 列创建 UNIQUE 约束，请使用下面的 SQL：

**MySQL / SQL Server / Oracle / MS Access：**

ALTER TABLE Persons  
ADD UNIQUE (P\_Id)

如需命名 UNIQUE 约束，并定义多个列的 UNIQUE 约束，请使用下面的 SQL 语法：

**MySQL / SQL Server / Oracle / MS Access：**

ALTER TABLE Persons  
ADD CONSTRAINT uc\_PersonID UNIQUE (P\_Id,LastName)

**撤销 UNIQUE 约束**

如需撤销 UNIQUE 约束，请使用下面的 SQL：

**MySQL：**

ALTER TABLE Persons  
DROP INDEX uc\_PersonID

**SQL Server / Oracle / MS Access：**

ALTER TABLE Persons  
DROP CONSTRAINT uc\_PersonID

**SQL PRIMARY KEY 约束**

**SQL PRIMARY KEY 约束**

PRIMARY KEY 约束唯一标识数据库表中的每条记录。

主键必须包含唯一的值。

主键列不能包含 NULL 值。

每个表都应该有一个主键，并且每个表只能有一个主键。

**CREATE TABLE 时的 SQL PRIMARY KEY 约束**

下面的 SQL 在 "Persons" 表创建时在 "P\_Id" 列上创建 PRIMARY KEY 约束：

**MySQL：**

CREATE TABLE Persons  
(  
P\_Id int NOT NULL,  
LastName varchar(255) NOT NULL,  
FirstName varchar(255),  
Address varchar(255),  
City varchar(255),  
PRIMARY KEY (P\_Id)  
)

**SQL Server / Oracle / MS Access：**

CREATE TABLE Persons  
(  
P\_Id int NOT NULL PRIMARY KEY,  
LastName varchar(255) NOT NULL,  
FirstName varchar(255),  
Address varchar(255),  
City varchar(255)  
)

如需命名 PRIMARY KEY 约束，并定义多个列的 PRIMARY KEY 约束，请使用下面的 SQL 语法：

**MySQL / SQL Server / Oracle / MS Access：**

CREATE TABLE Persons  
(  
P\_Id int NOT NULL,  
LastName varchar(255) NOT NULL,  
FirstName varchar(255),  
Address varchar(255),  
City varchar(255),  
CONSTRAINT pk\_PersonID PRIMARY KEY (P\_Id,LastName)  
)

**注释：**在上面的实例中，只有一个主键 PRIMARY KEY（pk\_PersonID）。然而，pk\_PersonID 的值是由两个列（P\_Id 和 LastName）组成的。

**ALTER TABLE 时的 SQL PRIMARY KEY 约束**

当表已被创建时，如需在 "P\_Id" 列创建 PRIMARY KEY 约束，请使用下面的 SQL：

**MySQL / SQL Server / Oracle / MS Access：**

ALTER TABLE Persons  
ADD PRIMARY KEY (P\_Id)

如需命名 PRIMARY KEY 约束，并定义多个列的 PRIMARY KEY 约束，请使用下面的 SQL 语法：

**MySQL / SQL Server / Oracle / MS Access：**

ALTER TABLE Persons  
ADD CONSTRAINT pk\_PersonID PRIMARY KEY (P\_Id,LastName)

**注释：**如果您使用 ALTER TABLE 语句添加主键，必须把主键列声明为不包含 NULL 值（在表首次创建时）。

**撤销 PRIMARY KEY 约束**

如需撤销 PRIMARY KEY 约束，请使用下面的 SQL：

**MySQL：**

ALTER TABLE Persons  
DROP PRIMARY KEY

**SQL Server / Oracle / MS Access：**

ALTER TABLE Persons  
DROP CONSTRAINT pk\_PersonID

**SQL FOREIGN KEY 约束**

**SQL FOREIGN KEY 约束**

一个表中的 FOREIGN KEY 指向另一个表中的 PRIMARY KEY。

让我们通过一个实例来解释外键。请看下面两个表：

"Persons" 表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P\_Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

"Orders" 表：

|  |  |  |
| --- | --- | --- |
| **O\_Id** | **OrderNo** | **P\_Id** |
| 1 | 77895 | 3 |
| 2 | 44678 | 3 |
| 3 | 22456 | 2 |
| 4 | 24562 | 1 |

请注意，"Orders" 表中的 "P\_Id" 列指向 "Persons" 表中的 "P\_Id" 列。

"Persons" 表中的 "P\_Id" 列是 "Persons" 表中的 PRIMARY KEY。

"Orders" 表中的 "P\_Id" 列是 "Orders" 表中的 FOREIGN KEY。

FOREIGN KEY 约束用于预防破坏表之间连接的行为。

FOREIGN KEY 约束也能防止非法数据插入外键列，因为它必须是它指向的那个表中的值之一。

**CREATE TABLE 时的 SQL FOREIGN KEY 约束**

下面的 SQL 在 "Orders" 表创建时在 "P\_Id" 列上创建 FOREIGN KEY 约束：

**MySQL：**

CREATE TABLE Orders  
(  
O\_Id int NOT NULL,  
OrderNo int NOT NULL,  
P\_Id int,  
PRIMARY KEY (O\_Id),  
FOREIGN KEY (P\_Id) REFERENCES Persons(P\_Id)  
)

**SQL Server / Oracle / MS Access：**

CREATE TABLE Orders  
(  
O\_Id int NOT NULL PRIMARY KEY,  
OrderNo int NOT NULL,  
P\_Id int FOREIGN KEY REFERENCES Persons(P\_Id)  
)

如需命名 FOREIGN KEY 约束，并定义多个列的 FOREIGN KEY 约束，请使用下面的 SQL 语法：

**MySQL / SQL Server / Oracle / MS Access：**

CREATE TABLE Orders  
(  
O\_Id int NOT NULL,  
OrderNo int NOT NULL,  
P\_Id int,  
PRIMARY KEY (O\_Id),  
CONSTRAINT fk\_PerOrders FOREIGN KEY (P\_Id)  
REFERENCES Persons(P\_Id)  
)

**ALTER TABLE 时的 SQL FOREIGN KEY 约束**

当 "Orders" 表已被创建时，如需在 "P\_Id" 列创建 FOREIGN KEY 约束，请使用下面的 SQL：

**MySQL / SQL Server / Oracle / MS Access：**

ALTER TABLE Orders  
ADD FOREIGN KEY (P\_Id)  
REFERENCES Persons(P\_Id)

如需命名 FOREIGN KEY 约束，并定义多个列的 FOREIGN KEY 约束，请使用下面的 SQL 语法：

**MySQL / SQL Server / Oracle / MS Access：**

ALTER TABLE Orders  
ADD CONSTRAINT fk\_PerOrders  
FOREIGN KEY (P\_Id)  
REFERENCES Persons(P\_Id)

**撤销 FOREIGN KEY 约束**

如需撤销 FOREIGN KEY 约束，请使用下面的 SQL：

**MySQL：**

ALTER TABLE Orders  
DROP FOREIGN KEY fk\_PerOrders

**SQL Server / Oracle / MS Access：**

ALTER TABLE Orders  
DROP CONSTRAINT fk\_PerOrders

**SQL CHECK 约束**

**SQL CHECK 约束**

CHECK 约束用于限制列中的值的范围。

如果对单个列定义 CHECK 约束，那么该列只允许特定的值。

如果对一个表定义 CHECK 约束，那么此约束会基于行中其他列的值在特定的列中对值进行限制。

**CREATE TABLE 时的 SQL CHECK 约束**

下面的 SQL 在 "Persons" 表创建时在 "P\_Id" 列上创建 CHECK 约束。CHECK 约束规定 "P\_Id" 列必须只包含大于 0 的整数。

**MySQL：**

CREATE TABLE Persons  
(  
P\_Id int NOT NULL,  
LastName varchar(255) NOT NULL,  
FirstName varchar(255),  
Address varchar(255),  
City varchar(255),  
CHECK (P\_Id>0)  
)

**SQL Server / Oracle / MS Access：**

CREATE TABLE Persons  
(  
P\_Id int NOT NULL CHECK (P\_Id>0),  
LastName varchar(255) NOT NULL,  
FirstName varchar(255),  
Address varchar(255),  
City varchar(255)  
)

如需命名 CHECK 约束，并定义多个列的 CHECK 约束，请使用下面的 SQL 语法：

**MySQL / SQL Server / Oracle / MS Access：**

CREATE TABLE Persons  
(  
P\_Id int NOT NULL,  
LastName varchar(255) NOT NULL,  
FirstName varchar(255),  
Address varchar(255),  
City varchar(255),  
CONSTRAINT chk\_Person CHECK (P\_Id>0 AND City='Sandnes')  
)

**ALTER TABLE 时的 SQL CHECK 约束**

当表已被创建时，如需在 "P\_Id" 列创建 CHECK 约束，请使用下面的 SQL：

**MySQL / SQL Server / Oracle / MS Access:**

ALTER TABLE Persons  
ADD CHECK (P\_Id>0)

如需命名 CHECK 约束，并定义多个列的 CHECK 约束，请使用下面的 SQL 语法：

**MySQL / SQL Server / Oracle / MS Access：**

ALTER TABLE Persons  
ADD CONSTRAINT chk\_Person CHECK (P\_Id>0 AND City='Sandnes')

**撤销 CHECK 约束**

如需撤销 CHECK 约束，请使用下面的 SQL：

**SQL Server / Oracle / MS Access：**

ALTER TABLE Persons  
DROP CONSTRAINT chk\_Person

**MySQL：**

ALTER TABLE Persons  
DROP CHECK chk\_Person

**SQL DEFAULT 约束**

**SQL DEFAULT 约束**

DEFAULT 约束用于向列中插入默认值。

如果没有规定其他的值，那么会将默认值添加到所有的新记录。

**CREATE TABLE 时的 SQL DEFAULT 约束**

下面的 SQL 在 "Persons" 表创建时在 "City" 列上创建 DEFAULT 约束：

**My SQL / SQL Server / Oracle / MS Access：**

CREATE TABLE Persons  
(  
P\_Id int NOT NULL,  
LastName varchar(255) NOT NULL,  
FirstName varchar(255),  
Address varchar(255),  
City varchar(255) DEFAULT 'Sandnes'  
)

通过使用类似 GETDATE() 这样的函数，DEFAULT 约束也可以用于插入系统值：

CREATE TABLE Orders  
(  
O\_Id int NOT NULL,  
OrderNo int NOT NULL,  
P\_Id int,  
OrderDate date DEFAULT GETDATE()  
)

**ALTER TABLE 时的 SQL DEFAULT 约束**

当表已被创建时，如需在 "City" 列创建 DEFAULT 约束，请使用下面的 SQL：

**MySQL：**

ALTER TABLE Persons  
ALTER City SET DEFAULT 'SANDNES'

**SQL Server / MS Access：**

ALTER TABLE Persons  
ALTER COLUMN City SET DEFAULT 'SANDNES'

**Oracle：**

ALTER TABLE Persons  
MODIFY City DEFAULT 'SANDNES'

**撤销 DEFAULT 约束**

如需撤销 DEFAULT 约束，请使用下面的 SQL：

**MySQL：**

ALTER TABLE Persons  
ALTER City DROP DEFAULT

**SQL Server / Oracle / MS Access：**

ALTER TABLE Persons  
ALTER COLUMN City DROP DEFAULT

# SQL CREATE INDEX 语句

CREATE INDEX 语句用于在表中创建索引。

在不读取整个表的情况下，索引使数据库应用程序可以更快地查找数据。

## 索引

您可以在表中创建索引，以便更加快速高效地查询数据。

用户无法看到索引，它们只能被用来加速搜索/查询。

**注释：**更新一个包含索引的表需要比更新一个没有索引的表花费更多的时间，这是由于索引本身也需要更新。因此，理想的做法是仅仅在常常被搜索的列（以及表）上面创建索引。

### SQL CREATE INDEX 语法

在表上创建一个简单的索引。允许使用重复的值：

CREATE INDEX index\_name  
ON table\_name (column\_name)

### SQL CREATE UNIQUE INDEX 语法

在表上创建一个唯一的索引。不允许使用重复的值：唯一的索引意味着两个行不能拥有相同的索引值。Creates a unique index on a table. Duplicate values are not allowed:

CREATE UNIQUE INDEX index\_name  
ON table\_name (column\_name)

**注释：**用于创建索引的语法在不同的数据库中不一样。因此，检查您的数据库中创建索引的语法。

## CREATE INDEX 实例

下面的 SQL 语句在 "Persons" 表的 "LastName" 列上创建一个名为 "PIndex" 的索引：

CREATE INDEX PIndex  
ON Persons (LastName)

如果您希望索引不止一个列，您可以在括号中列出这些列的名称，用逗号隔开：

CREATE INDEX PIndex  
ON Persons (LastName, FirstName)

# SQL 撤销索引、撤销表以及撤销数据库

通过使用 DROP 语句，可以轻松地删除索引、表和数据库。

## DROP INDEX 语句

DROP INDEX 语句用于删除表中的索引。

### 用于 MS Access 的 DROP INDEX 语法：

DROP INDEX index\_name ON table\_name

### 用于 MS SQL Server 的 DROP INDEX 语法：

DROP INDEX table\_name.index\_name

### 用于 DB2/Oracle 的 DROP INDEX 语法：

DROP INDEX index\_name

### 用于 MySQL 的 DROP INDEX 语法：

ALTER TABLE table\_name DROP INDEX index\_name

## DROP TABLE 语句

DROP TABLE 语句用于删除表。

DROP TABLE table\_name

## DROP DATABASE 语句

DROP DATABASE 语句用于删除数据库。

DROP DATABASE database\_name

## TRUNCATE TABLE 语句

如果我们仅仅需要删除表内的数据，但并不删除表本身，那么我们该如何做呢？

请使用 TRUNCATE TABLE 语句：

TRUNCATE TABLE table\_name

# SQL ALTER TABLE 语句

## ALTER TABLE 语句

ALTER TABLE 语句用于在已有的表中添加、删除或修改列。

### SQL ALTER TABLE 语法

如需在表中添加列，请使用下面的语法:

ALTER TABLE table\_name  
ADD column\_name datatype

如需删除表中的列，请使用下面的语法（请注意，某些数据库系统不允许这种在数据库表中删除列的方式）：

ALTER TABLE table\_name  
DROP COLUMN column\_name

要改变表中列的数据类型，请使用下面的语法：

**SQL Server / MS Access：**

ALTER TABLE table\_name  
ALTER COLUMN column\_name datatype

**My SQL / Oracle：**

ALTER TABLE table\_name  
MODIFY COLUMN column\_name datatype

## SQL ALTER TABLE 实例

请看 "Persons" 表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P\_Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

现在，我们想在 "Persons" 表中添加一个名为 "DateOfBirth" 的列。

我们使用下面的 SQL 语句：

ALTER TABLE Persons  
ADD DateOfBirth date

请注意，新列 "DateOfBirth" 的类型是 date，可以存放日期。数据类型规定列中可以存放的数据的类型。如需了解 MS Access、MySQL 和 SQL Server 中可用的数据类型，请访问我们完整的 [数据类型参考手册](http://www.w3cschool.cc/sql/sql-datatypes.html)。

现在，"Persons" 表将如下所示：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **P\_Id** | **LastName** | **FirstName** | **Address** | **City** | **DateOfBirth** |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |  |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |  |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |  |

## 改变数据类型实例

现在，我们想要改变 "Persons" 表中 "DateOfBirth" 列的数据类型。

我们使用下面的 SQL 语句：

ALTER TABLE Persons  
ALTER COLUMN DateOfBirth year

请注意，现在 "DateOfBirth" 列的类型是 year，可以存放 2 位或 4 位格式的年份。

## DROP COLUMN 实例

接下来，我们想要删除 "Person" 表中的 "DateOfBirth" 列。

我们使用下面的 SQL 语句：

ALTER TABLE Persons  
DROP COLUMN DateOfBirth

现在，"Persons" 表将如下所示：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P\_Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

**SQL AUTO INCREMENT 字段**

Auto-increment 会在新记录插入表中时生成一个唯一的数字。

**AUTO INCREMENT 字段**

我们通常希望在每次插入新记录时，自动地创建主键字段的值。

我们可以在表中创建一个 auto-increment 字段。

**用于 MySQL 的语法**

下面的 SQL 语句把 "Persons" 表中的 "ID" 列定义为 auto-increment 主键字段：

CREATE TABLE Persons  
(  
ID int NOT NULL AUTO\_INCREMENT,  
LastName varchar(255) NOT NULL,  
FirstName varchar(255),  
Address varchar(255),  
City varchar(255),  
PRIMARY KEY (ID)  
)

MySQL 使用 AUTO\_INCREMENT 关键字来执行 auto-increment 任务。

默认地，AUTO\_INCREMENT 的开始值是 1，每条新记录递增 1。

要让 AUTO\_INCREMENT 序列以其他的值起始，请使用下面的 SQL 语法：

ALTER TABLE Persons AUTO\_INCREMENT=100

要在 "Persons" 表中插入新记录，我们不必为 "ID" 列规定值（会自动添加一个唯一的值）：

INSERT INTO Persons (FirstName,LastName)  
VALUES ('Lars','Monsen')

上面的 SQL 语句会在 "Persons" 表中插入一条新记录。"ID" 列会被赋予一个唯一的值。"FirstName" 列会被设置为 "Lars"，"LastName" 列会被设置为 "Monsen"。

**用于 SQL Server 的语法**

下面的 SQL 语句把 "Persons" 表中的 "ID" 列定义为 auto-increment 主键字段：

CREATE TABLE Persons  
(  
ID int IDENTITY(1,1) PRIMARY KEY,  
LastName varchar(255) NOT NULL,  
FirstName varchar(255),  
Address varchar(255),  
City varchar(255)  
)

MS SQL Server 使用 IDENTITY 关键字来执行 auto-increment 任务。

在上面的实例中，IDENTITY 的开始值是 1，每条新记录递增 1。

**提示：**要规定 "ID" 列以 10 起始且递增 5，请把 identity 改为 IDENTITY(10,5)。

要在 "Persons" 表中插入新记录，我们不必为 "ID" 列规定值（会自动添加一个唯一的值）：

INSERT INTO Persons (FirstName,LastName)  
VALUES ('Lars','Monsen')

上面的 SQL 语句会在 "Persons" 表中插入一条新记录。"ID" 列会被赋予一个唯一的值。"FirstName" 列会被设置为 "Lars"，"LastName" 列会被设置为 "Monsen"。

**用于 Access 的语法**

下面的 SQL 语句把 "Persons" 表中的 "ID" 列定义为 auto-increment 主键字段：

CREATE TABLE Persons  
(  
ID Integer PRIMARY KEY AUTOINCREMENT,  
LastName varchar(255) NOT NULL,  
FirstName varchar(255),  
Address varchar(255),  
City varchar(255)  
)

MS Access 使用 AUTOINCREMENT 关键字来执行 auto-increment 任务。

默认地，AUTOINCREMENT 的开始值是 1，每条新记录递增 1。

**提示：**要规定 "ID" 列以 10 起始且递增 5，请把 autoincrement 改为 AUTOINCREMENT(10,5)。

要在 "Persons" 表中插入新记录，我们不必为 "ID" 列规定值（会自动添加一个唯一的值）：

INSERT INTO Persons (FirstName,LastName)  
VALUES ('Lars','Monsen')

上面的 SQL 语句会在 "Persons" 表中插入一条新记录。"ID" 列会被赋予一个唯一的值。"FirstName" 列会被设置为 "Lars"，"LastName" 列会被设置为 "Monsen"。

**语法 for Oracle**

在 Oracle 中，代码稍微复杂一点。

您必须通过 sequence 对象（该对象生成数字序列）创建 auto-increment 字段。

请使用下面的 CREATE SEQUENCE 语法：

CREATE SEQUENCE seq\_person  
MINVALUE 1  
START WITH 1  
INCREMENT BY 1  
CACHE 10

上面的代码创建一个名为 seq\_person 的 sequence 对象，它以 1 起始且以 1 递增。该对象缓存 10 个值以提高性能。cache 选项规定了为了提高访问速度要存储多少个序列值。

要在 "Persons" 表中插入新记录，我们必须使用 nextval 函数（该函数从 seq\_person 序列中取回下一个值）：

INSERT INTO Persons (ID,FirstName,LastName)  
VALUES (seq\_person.nextval,'Lars','Monsen')

上面的 SQL 语句会在 "Persons" 表中插入一条新记录。"ID" 列会被赋值为来自 seq\_person 序列的下一个数字。"FirstName"列 会被设置为 "Lars"，"LastName" 列会被设置为 "Monsen"。

# SQL 视图（Views）

视图是可视化的表。

本章讲解如何创建、更新和删除视图。

## SQL CREATE VIEW 语句

在 SQL 中，视图是基于 SQL 语句的结果集的可视化的表。

视图包含行和列，就像一个真实的表。视图中的字段就是来自一个或多个数据库中的真实的表中的字段。

您可以向视图添加 SQL 函数、WHERE 以及 JOIN 语句，也可以呈现数据，就像这些数据来自于某个单一的表一样。

### SQL CREATE VIEW 语法

CREATE VIEW view\_name AS  
SELECT column\_name(s)  
FROM table\_name  
WHERE condition

**注释：**视图总是显示最新的数据！每当用户查询视图时，数据库引擎通过使用视图的 SQL 语句重建数据。

## SQL CREATE VIEW 实例

样本数据库 Northwind 拥有一些被默认安装的视图。

视图 "Current Product List" 会从 "Products" 表列出所有正在使用的产品（未停产的产品）。这个视图使用下面的 SQL 创建：

CREATE VIEW [Current Product List] AS  
SELECT ProductID,ProductName  
FROM Products  
WHERE Discontinued=No

我们可以像这样查询上面这个视图：

SELECT \* FROM [Current Product List]

Northwind 样本数据库的另一个视图会选取 "Products" 表中所有单位价格高于平均单位价格的产品：

CREATE VIEW [Products Above Average Price] AS  
SELECT ProductName,UnitPrice  
FROM Products  
WHERE UnitPrice>(SELECT AVG(UnitPrice) FROM Products)

我们可以像这样查询上面这个视图：

SELECT \* FROM [Products Above Average Price]

Northwind 样本数据库的另一个视图会计算在 1997 年每个种类的销售总数。请注意，这个视图会从另一个名为 "Product Sales for 1997" 的视图那里选取数据：

CREATE VIEW [Category Sales For 1997] AS  
SELECT DISTINCT CategoryName,Sum(ProductSales) AS CategorySales  
FROM [Product Sales for 1997]  
GROUP BY CategoryName

我们可以像这样查询上面这个视图：

SELECT \* FROM [Category Sales For 1997]

我们也可以向查询添加条件。现在，我们仅仅需要查看 "Beverages" 类的销售总数：

SELECT \* FROM [Category Sales For 1997]  
WHERE CategoryName='Beverages'

## SQL 更新视图

您可以使用下面的语法来更新视图：

### SQL CREATE OR REPLACE VIEW 语法

CREATE OR REPLACE VIEW view\_name AS  
SELECT column\_name(s)  
FROM table\_name  
WHERE condition

现在，我们希望向 "Current Product List" 视图添加 "Category" 列。我们将通过下列 SQL 更新视图：

CREATE VIEW [Current Product List] AS  
SELECT ProductID,ProductName,Category  
FROM Products  
WHERE Discontinued=No

## SQL 撤销视图

您可以通过 DROP VIEW 命令来删除视图。

### SQL DROP VIEW 语法

DROP VIEW view\_name

# SQL Date 函数

## SQL 日期（Dates）

Note当我们处理日期时，最难的任务恐怕是确保所插入的日期的格式，与数据库中日期列的格式相匹配。

只要您的数据包含的只是日期部分，运行查询就不会出问题。但是，如果涉及时间部分，情况就有点复杂了。

在讨论日期查询的复杂性之前，我们先来看看最重要的内建日期处理函数。

## MySQL Date 函数

下面的表格列出了 MySQL 中最重要的内建日期函数：

|  |  |
| --- | --- |
| **函数** | **描述** |
| [NOW()](http://www.w3cschool.cc/sql/func-now.html) | 返回当前的日期和时间 |
| [CURDATE()](http://www.w3cschool.cc/sql/func-curdate.html) | 返回当前的日期 |
| [CURTIME()](http://www.w3cschool.cc/sql/func-curtime.html) | 返回当前的时间 |
| [DATE()](http://www.w3cschool.cc/sql/func-date.html) | 提取日期或日期/时间表达式的日期部分 |
| [EXTRACT()](http://www.w3cschool.cc/sql/func-extract.html) | 返回日期/时间的单独部分 |
| [DATE\_ADD()](http://www.w3cschool.cc/sql/func-date-add.html) | 向日期添加指定的时间间隔 |
| [DATE\_SUB()](http://www.w3cschool.cc/sql/func-date-sub.html) | 从日期减去指定的时间间隔 |
| [DATEDIFF()](http://www.w3cschool.cc/sql/func-datediff-mysql.html) | 返回两个日期之间的天数 |
| [DATE\_FORMAT()](http://www.w3cschool.cc/sql/func-date-format.html) | 用不同的格式显示日期/时间 |

## SQL Server Date 函数

下面的表格列出了 SQL Server 中最重要的内建日期函数：

|  |  |
| --- | --- |
| **函数** | **描述** |
| [GETDATE()](http://www.w3cschool.cc/sql/func-getdate.html) | 返回当前的日期和时间 |
| [DATEPART()](http://www.w3cschool.cc/sql/func-datepart.html) | 返回日期/时间的单独部分 |
| [DATEADD()](http://www.w3cschool.cc/sql/func-dateadd.html) | 在日期中添加或减去指定的时间间隔 |
| [DATEDIFF()](http://www.w3cschool.cc/sql/func-datediff.html) | 返回两个日期之间的时间 |
| [CONVERT()](http://www.w3cschool.cc/sql/func-convert.html) | 用不同的格式显示日期/时间 |

## SQL Date 数据类型

**MySQL** 使用下列数据类型在数据库中存储日期或日期/时间值：

* DATE - 格式：YYYY-MM-DD
* DATETIME - 格式：YYYY-MM-DD HH:MM:SS
* TIMESTAMP - 格式：YYYY-MM-DD HH:MM:SS
* YEAR - 格式：YYYY 或 YY

**SQL Server** 使用下列数据类型在数据库中存储日期或日期/时间值：

* DATE - 格式：YYYY-MM-DD
* DATETIME - 格式：YYYY-MM-DD HH:MM:SS
* SMALLDATETIME - 格式：YYYY-MM-DD HH:MM:SS
* TIMESTAMP - 格式：唯一的数字

**注释：**当您在数据库中创建一个新表时，需要为列选择数据类型！

如需了解所有可用的数据类型，请访问我们完整的 [数据类型参考手册](http://www.w3cschool.cc/sql/sql-datatypes.html)。

## SQL 日期处理

Note如果不涉及时间部分，那么我们可以轻松地比较两个日期！

假设我们有如下的 "Orders" 表：

|  |  |  |
| --- | --- | --- |
| **OrderId** | **ProductName** | **OrderDate** |
| 1 | Geitost | 2008-11-11 |
| 2 | Camembert Pierrot | 2008-11-09 |
| 3 | Mozzarella di Giovanni | 2008-11-11 |
| 4 | Mascarpone Fabioli | 2008-10-29 |

现在，我们希望从上表中选取 OrderDate 为 "2008-11-11" 的记录。

我们使用下面的 SELECT 语句：

SELECT \* FROM Orders WHERE OrderDate='2008-11-11'

结果集如下所示：

|  |  |  |
| --- | --- | --- |
| **OrderId** | **ProductName** | **OrderDate** |
| 1 | Geitost | 2008-11-11 |
| 3 | Mozzarella di Giovanni | 2008-11-11 |

现在，假设 "Orders" 表如下所示（请注意 "OrderDate" 列中的时间部分）：

|  |  |  |
| --- | --- | --- |
| **OrderId** | **ProductName** | **OrderDate** |
| 1 | Geitost | 2008-11-11 13:23:44 |
| 2 | Camembert Pierrot | 2008-11-09 15:45:21 |
| 3 | Mozzarella di Giovanni | 2008-11-11 11:12:01 |
| 4 | Mascarpone Fabioli | 2008-10-29 14:56:59 |

如果我们使用和上面一样的 SELECT 语句：

SELECT \* FROM Orders WHERE OrderDate='2008-11-11'

那么我们将得不到结果！这是由于该查询的日期不含有时间部分。

**提示：**如果您希望使查询简单且更易维护，那么请不要在日期中使用时间部分！

**SQL NULL 值**

NULL 值代表遗漏的未知数据。

默认地，表的列可以存放 NULL 值。

本章讲解 IS NULL 和 IS NOT NULL 操作符。

**SQL NULL 值**

如果表中的某个列是可选的，那么我们可以在不向该列添加值的情况下插入新记录或更新已有的记录。这意味着该字段将以 NULL 值保存。

NULL 值的处理方式与其他值不同。

NULL 用作未知的或不适用的值的占位符。

Note**注释：**无法比较 NULL 和 0；它们是不等价的。

**SQL 的 NULL 值处理**

请看下面的 "Persons" 表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P\_Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Hansen | Ola |  | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari |  | Stavanger |

假如 "Persons" 表中的 "Address" 列是可选的。这意味着如果在 "Address" 列插入一条不带值的记录，"Address" 列会使用 NULL 值保存。

那么我们如何测试 NULL 值呢？

无法使用比较运算符来测试 NULL 值，比如 =、< 或 <>。

我们必须使用 IS NULL 和 IS NOT NULL 操作符。

**SQL IS NULL**

我们如何仅仅选取在 "Address" 列中带有 NULL 值的记录呢？

我们必须使用 IS NULL 操作符：

SELECT LastName,FirstName,Address FROM Persons  
WHERE Address IS NULL

结果集如下所示：

|  |  |  |
| --- | --- | --- |
| **LastName** | **FirstName** | **Address** |
| Hansen | Ola |  |
| Pettersen | Kari |  |

Note**提示：**请始终使用 IS NULL 来查找 NULL 值。

**SQL IS NOT NULL**

我们如何仅仅选取在 "Address" 列中不带有 NULL 值的记录呢？

我们必须使用 IS NOT NULL 操作符：

SELECT LastName,FirstName,Address FROM Persons  
WHERE Address IS NOT NULL

结果集如下所示：

|  |  |  |
| --- | --- | --- |
| **LastName** | **FirstName** | **Address** |
| Svendson | Tove | Borgvn 23 |

在下一节中，我们了解 ISNULL()、NVL()、IFNULL() 和 COALESCE() 函数。

**SQL NULL 函数**

**SQL ISNULL()、NVL()、IFNULL() 和 COALESCE() 函数**

请看下面的 "Products" 表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P\_Id** | **ProductName** | **UnitPrice** | **UnitsInStock** | **UnitsOnOrder** |
| 1 | Jarlsberg | 10.45 | 16 | 15 |
| 2 | Mascarpone | 32.56 | 23 |  |
| 3 | Gorgonzola | 15.67 | 9 | 20 |

假如 "UnitsOnOrder" 是可选的，而且可以包含 NULL 值。

我们使用下面的 SELECT 语句：

SELECT ProductName,UnitPrice\*(UnitsInStock+UnitsOnOrder)  
FROM Products

在上面的实例中，如果有 "UnitsOnOrder" 值是 NULL，那么结果是 NULL。

微软的 ISNULL() 函数用于规定如何处理 NULL 值。

NVL()、IFNULL() 和 COALESCE() 函数也可以达到相同的结果。

在这里，我们希望 NULL 值为 0。

下面，如果 "UnitsOnOrder" 是 NULL，则不会影响计算，因为如果值是 NULL 则 ISNULL() 返回 0：

**SQL Server / MS Access**

SELECT ProductName,UnitPrice\*(UnitsInStock+ISNULL(UnitsOnOrder,0))  
FROM Products

**Oracle**

Oracle 没有 ISNULL() 函数。不过，我们可以使用 NVL() 函数达到相同的结果：

SELECT ProductName,UnitPrice\*(UnitsInStock+NVL(UnitsOnOrder,0))  
FROM Products

**MySQL**

MySQL 也拥有类似 ISNULL() 的函数。不过它的工作方式与微软的 ISNULL() 函数有点不同。

在 MySQL 中，我们可以使用 IFNULL() 函数，如下所示：

SELECT ProductName,UnitPrice\*(UnitsInStock+IFNULL(UnitsOnOrder,0))  
FROM Products

或者我们可以使用 COALESCE() 函数，如下所示：

SELECT ProductName,UnitPrice\*(UnitsInStock+COALESCE(UnitsOnOrder,0))  
FROM Products

**SQL 通用数据类型**

数据类型定义列中存放的值的种类。

**SQL 通用数据类型**

数据库表中的每个列都要求有名称和数据类型。Each column in a database table is required to have a name and a data type.

SQL 开发人员必须在创建 SQL 表时决定表中的每个列将要存储的数据的类型。数据类型是一个标签，是便于 SQL 了解每个列期望存储什么类型的数据的指南，它也标识了 SQL 如何与存储的数据进行交互。

下面的表格列出了 SQL 中通用的数据类型：

|  |  |
| --- | --- |
| **数据类型** | **描述** |
| CHARACTER(n) | 字符/字符串。固定长度 n。 |
| VARCHAR(n) 或 CHARACTER VARYING(n) | 字符/字符串。可变长度。最大长度 n。 |
| BINARY(n) | 二进制串。固定长度 n。 |
| BOOLEAN | 存储 TRUE 或 FALSE 值 |
| VARBINARY(n) 或 BINARY VARYING(n) | 二进制串。可变长度。最大长度 n。 |
| INTEGER(p) | 整数值（没有小数点）。精度 p。 |
| SMALLINT | 整数值（没有小数点）。精度 5。 |
| INTEGER | 整数值（没有小数点）。精度 10。 |
| BIGINT | 整数值（没有小数点）。精度 19。 |
| DECIMAL(p,s) | 精确数值，精度 p，小数点后位数 s。例如：decimal(5,2) 是一个小数点前有 3 位数小数点后有 2 位数的数字。 |
| NUMERIC(p,s) | 精确数值，精度 p，小数点后位数 s。（与 DECIMAL 相同） |
| FLOAT(p) | 近似数值，尾数精度 p。一个采用以 10 为基数的指数计数法的浮点数。该类型的 size 参数由一个指定最小精度的单一数字组成。 |
| REAL | 近似数值，尾数精度 7。 |
| FLOAT | 近似数值，尾数精度 16。 |
| DOUBLE PRECISION | 近似数值，尾数精度 16。 |
| DATE | 存储年、月、日的值。 |
| TIME | 存储小时、分、秒的值。 |
| TIMESTAMP | 存储年、月、日、小时、分、秒的值。 |
| INTERVAL | 由一些整数字段组成，代表一段时间，取决于区间的类型。 |
| ARRAY | 元素的固定长度的有序集合 |
| MULTISET | 元素的可变长度的无序集合 |
| XML | 存储 XML 数据 |

**SQL 数据类型快速参考手册**

然而，不同的数据库对数据类型定义提供不同的选择。

下面的表格显示了各种不同的数据库平台上一些数据类型的通用名称：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **数据类型** | **Access** | **SQLServer** | **Oracle** | **MySQL** | **PostgreSQL** |
| *boolean* | Yes/No | Bit | Byte | N/A | Boolean |
| *integer* | Number (integer) | Int | Number | Int Integer | Int Integer |
| *float* | Number (single) | Float Real | Number | Float | Numeric |
| *currency* | Currency | Money | N/A | N/A | Money |
| *string (fixed)* | N/A | Char | Char | Char | Char |
| *string (variable)* | Text (<256) Memo (65k+) | Varchar | Varchar Varchar2 | Varchar | Varchar |
| *binary object* | OLE Object Memo | Binary (fixed up to 8K) Varbinary (<8K) Image (<2GB) | Long Raw | Blob Text | Binary Varbinary |

|  |  |
| --- | --- |
| **lamp** | **注释：**在不同的数据库中，同一种数据类型可能有不同的名称。即使名称相同，尺寸和其他细节也可能不同！ **请总是检查文档！** |

**SQL 用于各种数据库的数据类型**

Microsoft Access、MySQL 和 SQL Server 所使用的数据类型和范围。

**Microsoft Access 数据类型**

|  |  |  |
| --- | --- | --- |
| **数据类型** | **描述** | **存储** |
| Text | 用于文本或文本与数字的组合。最多 255 个字符。 |  |
| Memo | Memo 用于更大数量的文本。最多存储 65,536 个字符。**注释：**无法对 memo 字段进行排序。不过它们是可搜索的。 |  |
| Byte | 允许 0 到 255 的数字。 | 1 字节 |
| Integer | 允许介于 -32,768 与 32,767 之间的全部数字。 | 2 字节 |
| Long | 允许介于 -2,147,483,648 与 2,147,483,647 之间的全部数字。 | 4 字节 |
| Single | 单精度浮点。处理大多数小数。 | 4 字节 |
| Double | 双精度浮点。处理大多数小数。 | 8 字节 |
| Currency | 用于货币。支持 15 位的元，外加 4 位小数。**提示：**您可以选择使用哪个国家的货币。 | 8 字节 |
| AutoNumber | AutoNumber 字段自动为每条记录分配数字，通常从 1 开始。 | 4 字节 |
| Date/Time | 用于日期和时间 | 8 字节 |
| Yes/No | 逻辑字段，可以显示为 Yes/No、True/False 或 On/Off。在代码中，使用常量 True 和 False （等价于 1 和 0）。**注释：**Yes/No 字段中不允许 Null 值 | 1 比特 |
| Ole Object | 可以存储图片、音频、视频或其他 BLOBs（Binary Large OBjects）。 | 最多 1GB |
| Hyperlink | 包含指向其他文件的链接，包括网页。 |  |
| Lookup Wizard | 允许您创建一个可从下拉列表中进行选择的选项列表。 | 4 字节 |

**MySQL 数据类型**

在 MySQL 中，有三种主要的类型：Text（文本）、Number（数字）和 Date/Time（日期/时间）类型。

**Text 类型：**

|  |  |
| --- | --- |
| **数据类型** | **描述** |
| CHAR(size) | 保存固定长度的字符串（可包含字母、数字以及特殊字符）。在括号中指定字符串的长度。最多 255 个字符。 |
| VARCHAR(size) | 保存可变长度的字符串（可包含字母、数字以及特殊字符）。在括号中指定字符串的最大长度。最多 255 个字符。**注释：**如果值的长度大于 255，则被转换为 TEXT 类型。 |
| TINYTEXT | 存放最大长度为 255 个字符的字符串。 |
| TEXT | 存放最大长度为 65,535 个字符的字符串。 |
| BLOB | 用于 BLOBs（Binary Large OBjects）。存放最多 65,535 字节的数据。 |
| MEDIUMTEXT | 存放最大长度为 16,777,215 个字符的字符串。 |
| MEDIUMBLOB | 用于 BLOBs（Binary Large OBjects）。存放最多 16,777,215 字节的数据。 |
| LONGTEXT | 存放最大长度为 4,294,967,295 个字符的字符串。 |
| LONGBLOB | 用于 BLOBs (Binary Large OBjects)。存放最多 4,294,967,295 字节的数据。 |
| ENUM(x,y,z,etc.) | 允许您输入可能值的列表。可以在 ENUM 列表中列出最大 65535 个值。如果列表中不存在插入的值，则插入空值。  **注释：**这些值是按照您输入的顺序排序的。  可以按照此格式输入可能的值： ENUM('X','Y','Z') |
| SET | 与 ENUM 类似，不同的是，SET 最多只能包含 64 个列表项且 SET 可存储一个以上的选择。 |

**Number 类型：**

|  |  |
| --- | --- |
| **数据类型** | **描述** |
| TINYINT(size) | -128 到 127 常规。0 到 255 无符号\*。在括号中规定最大位数。 |
| SMALLINT(size) | -32768 到 32767 常规。0 到 65535 无符号\*。在括号中规定最大位数。 |
| MEDIUMINT(size) | -8388608 到 8388607 普通。0 to 16777215 无符号\*。在括号中规定最大位数。 |
| INT(size) | -2147483648 到 2147483647 常规。0 到 4294967295 无符号\*。在括号中规定最大位数。 |
| BIGINT(size) | -9223372036854775808 到 9223372036854775807 常规。0 到 18446744073709551615 无符号\*。在括号中规定最大位数。 |
| FLOAT(size,d) | 带有浮动小数点的小数字。在 size 参数中规定最大位数。在 d 参数中规定小数点右侧的最大位数。 |
| DOUBLE(size,d) | 带有浮动小数点的大数字。在 size 参数中规定最大位数。在 d 参数中规定小数点右侧的最大位数。 |
| DECIMAL(size,d) | 作为字符串存储的 DOUBLE 类型，允许固定的小数点。在 size 参数中规定最大位数。在 d 参数中规定小数点右侧的最大位数。 |

\*这些整数类型拥有额外的选项 UNSIGNED。通常，整数可以是负数或正数。如果添加 UNSIGNED 属性，那么范围将从 0 开始，而不是某个负数。

**Date 类型：**

|  |  |
| --- | --- |
| **数据类型** | **描述** |
| DATE() | 日期。格式：YYYY-MM-DD  **注释：**支持的范围是从 '1000-01-01' 到 '9999-12-31' |
| DATETIME() | \*日期和时间的组合。格式：YYYY-MM-DD HH:MM:SS  **注释：**支持的范围是从 '1000-01-01 00:00:00' 到 '9999-12-31 23:59:59' |
| TIMESTAMP() | \*时间戳。TIMESTAMP 值使用 Unix 纪元('1970-01-01 00:00:00' UTC) 至今的秒数来存储。格式：YYYY-MM-DD HH:MM:SS  **注释：**支持的范围是从 '1970-01-01 00:00:01' UTC 到 '2038-01-09 03:14:07' UTC |
| TIME() | 时间。格式：HH:MM:SS  **注释：**支持的范围是从 '-838:59:59' 到 '838:59:59' |
| YEAR() | 2 位或 4 位格式的年。  **注释：**4 位格式所允许的值：1901 到 2155。2 位格式所允许的值：70 到 69，表示从 1970 到 2069。 |

\*即便 DATETIME 和 TIMESTAMP 返回相同的格式，它们的工作方式很不同。在 INSERT 或 UPDATE 查询中，TIMESTAMP 自动把自身设置为当前的日期和时间。TIMESTAMP 也接受不同的格式，比如 YYYYMMDDHHMMSS、YYMMDDHHMMSS、YYYYMMDD 或 YYMMDD。

**SQL Server 数据类型**

**String 类型：**

|  |  |  |
| --- | --- | --- |
| **数据类型** | **描述** | **存储** |
| char(n) | 固定长度的字符串。最多 8,000 个字符。 | Defined width |
| varchar(n) | 可变长度的字符串。最多 8,000 个字符。 | 2 bytes + number of chars |
| varchar(max) | 可变长度的字符串。最多 1,073,741,824 个字符。 | 2 bytes + number of chars |
| text | 可变长度的字符串。最多 2GB 文本数据。 | 4 bytes + number of chars |
| nchar | 固定长度的 Unicode 字符串。最多 4,000 个字符。 | Defined width x 2 |
| nvarchar | 可变长度的 Unicode 字符串。最多 4,000 个字符。 |  |
| nvarchar(max) | 可变长度的 Unicode 字符串。最多 536,870,912 个字符。 |  |
| ntext | 可变长度的 Unicode 字符串。最多 2GB 文本数据。 |  |
| bit | 允许 0、1 或 NULL |  |
| binary(n) | 固定长度的二进制字符串。最多 8,000 字节。 |  |
| varbinary | 可变长度的二进制字符串。最多 8,000 字节。 |  |
| varbinary(max) | 可变长度的二进制字符串。最多 2GB。 |  |
| image | 可变长度的二进制字符串。最多 2GB。 |  |

**Number 类型：**

|  |  |  |
| --- | --- | --- |
| **数据类型** | **描述** | **存储** |
| tinyint | 允许从 0 到 255 的所有数字。 | 1 字节 |
| smallint | 允许介于 -32,768 与 32,767 的所有数字。 | 2 字节 |
| int | 允许介于 -2,147,483,648 与 2,147,483,647 的所有数字。 | 4 字节 |
| bigint | 允许介于 -9,223,372,036,854,775,808 与 9,223,372,036,854,775,807 之间的所有数字。 | 8 字节 |
| decimal(p,s) | 固定精度和比例的数字。  允许从 -10^38 +1 到 10^38 -1 之间的数字。  p 参数指示可以存储的最大位数（小数点左侧和右侧）。p 必须是 1 到 38 之间的值。默认是 18。  s 参数指示小数点右侧存储的最大位数。s 必须是 0 到 p 之间的值。默认是 0。 | 5-17 字节 |
| numeric(p,s) | 固定精度和比例的数字。  允许从 -10^38 +1 到 10^38 -1 之间的数字。  p 参数指示可以存储的最大位数（小数点左侧和右侧）。p 必须是 1 到 38 之间的值。默认是 18。  s 参数指示小数点右侧存储的最大位数。s 必须是 0 到 p 之间的值。默认是 0。 | 5-17 字节 |
| smallmoney | 介于 -214,748.3648 与 214,748.3647 之间的货币数据。 | 4 字节 |
| money | 介于 -922,337,203,685,477.5808 与 922,337,203,685,477.5807 之间的货币数据。 | 8 字节 |
| float(n) | 从 -1.79E + 308 到 1.79E + 308 的浮动精度数字数据。  n 参数指示该字段保存 4 字节还是 8 字节。float(24) 保存 4 字节，而 float(53) 保存 8 字节。n 的默认值是 53。 | 4 或 8 字节 |
| real | 从 -3.40E + 38 到 3.40E + 38 的浮动精度数字数据。 | 4 字节 |

**Date 类型：**

|  |  |  |
| --- | --- | --- |
| **数据类型** | **描述** | **存储** |
| datetime | 从 1753 年 1 月 1 日 到 9999 年 12 月 31 日，精度为 3.33 毫秒。 | 8 字节 |
| datetime2 | 从 1753 年 1 月 1 日 到 9999 年 12 月 31 日，精度为 100 纳秒。 | 6-8 字节 |
| smalldatetime | 从 1900 年 1 月 1 日 到 2079 年 6 月 6 日，精度为 1 分钟。 | 4 字节 |
| date | 仅存储日期。从 0001 年 1 月 1 日 到 9999 年 12 月 31 日。 | 3 bytes |
| time | 仅存储时间。精度为 100 纳秒。 | 3-5 字节 |
| datetimeoffset | 与 datetime2 相同，外加时区偏移。 | 8-10 字节 |
| timestamp | 存储唯一的数字，每当创建或修改某行时，该数字会更新。timestamp 值基于内部时钟，不对应真实时间。每个表只能有一个 timestamp 变量。 |  |

**其他数据类型：**

|  |  |
| --- | --- |
| **数据类型** | **描述** |
| sql\_variant | 存储最多 8,000 字节不同数据类型的数据，除了 text、ntext 以及 timestamp。 |
| uniqueidentifier | 存储全局唯一标识符 (GUID)。 |
| xml | 存储 XML 格式化数据。最多 2GB。 |
| cursor | 存储对用于数据库操作的指针的引用。 |
| table | 存储结果集，供稍后处理。 |

# SQL 函数

SQL 拥有很多可用于计数和计算的内建函数。

## SQL Aggregate 函数

SQL Aggregate 函数计算从列中取得的值，返回一个单一的值。

有用的 Aggregate 函数：

* AVG() - 返回平均值
* COUNT() - 返回行数
* FIRST() - 返回第一个记录的值
* LAST() - 返回最后一个记录的值
* MAX() - 返回最大值
* MIN() - 返回最小值
* SUM() - 返回总和

## SQL Scalar 函数

SQL Scalar 函数基于输入值，返回一个单一的值。

有用的 Scalar 函数：

* UCASE() - 将某个字段转换为大写
* LCASE() - 将某个字段转换为小写
* MID() - 从某个文本字段提取字符
* LEN() - 返回某个文本字段的长度
* ROUND() - 对某个数值字段进行指定小数位数的四舍五入
* NOW() - 返回当前的系统日期和时间
* FORMAT() - 格式化某个字段的显示方式

**提示：**在下面的章节，我们会详细讲解 Aggregate 函数和 Scalar 函数。

# SQL AVG() 函数

## AVG() 函数

AVG() 函数返回数值列的平均值。

### SQL AVG() 语法

SELECT AVG(column\_name) FROM table\_name

## 演示数据库

在本教程中，我们将使用众所周知的 Northwind 样本数据库。

下面是选自 "Products" 表的数据：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ProductID** | **ProductName** | **SupplierID** | **CategoryID** | **Unit** | **Price** |
| 1 | Chais | 1 | 1 | 10 boxes x 20 bags | 18 |
| 2 | Chang | 1 | 1 | 24 - 12 oz bottles | 19 |
| 3 | Aniseed Syrup | 1 | 2 | 12 - 550 ml bottles | 10 |
| 4 | Chef Anton's Cajun Seasoning | 2 | 2 | 48 - 6 oz jars | 21.35 |
| 5 | Chef Anton's Gumbo Mix | 2 | 2 | 36 boxes | 25 |

## SQL AVG() 实例

下面的 SQL 语句从 "Products" 表的 "Price" 列获取平均值：

## 实例

SELECT AVG(Price) AS PriceAverage FROM Products;

下面的 SQL 语句选择价格高于平均价格的 "ProductName" 和 "Price" 记录：

## 实例

SELECT ProductName, Price FROM Products  
WHERE Price>(SELECT AVG(Price) FROM Products);

# SQL COUNT() 函数

COUNT() 函数返回匹配指定条件的行数。

### SQL COUNT(column\_name) 语法

COUNT(column\_name) 函数返回指定列的值的数目（NULL 不计入）：

SELECT COUNT(column\_name) FROM table\_name;

### SQL COUNT(\*) 语法

COUNT(\*) 函数返回表中的记录数：

SELECT COUNT(\*) FROM table\_name;

### SQL COUNT(DISTINCT column\_name) 语法

COUNT(DISTINCT column\_name) 函数返回指定列的不同值的数目：

SELECT COUNT(DISTINCT column\_name) FROM table\_name;

**注释：**COUNT(DISTINCT) 适用于 ORACLE 和 Microsoft SQL Server，但是无法用于 Microsoft Access。

## 演示数据库

在本教程中，我们将使用众所周知的 Northwind 样本数据库。

下面是选自 "Orders" 表的数据：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| 10265 | 7 | 2 | 1996-07-25 | 1 |
| 10266 | 87 | 3 | 1996-07-26 | 3 |
| 10267 | 25 | 4 | 1996-07-29 | 1 |

## SQL COUNT(column\_name) 实例

下面的 SQL 语句计算 "Orders" 表中 "CustomerID"=7 的订单数：

## 实例

SELECT COUNT(CustomerID) AS OrdersFromCustomerID7 FROM Orders  
WHERE CustomerID=7;

## SQL COUNT(\*) 实例

下面的 SQL 语句计算 "Orders" 表中的订单总数：

## 实例

SELECT COUNT(\*) AS NumberOfOrders FROM Orders;

## SQL COUNT(DISTINCT column\_name) 实例

下面的 SQL 语句计算 "Orders" 表中不同客户的数目：

## 实例

SELECT COUNT(DISTINCT CustomerID) AS NumberOfCustomers FROM Orders;

# SQL FIRST() 函数

## FIRST() 函数

FIRST() 函数返回指定的列中第一个记录的值。

### SQL FIRST() 语法

SELECT FIRST(column\_name) FROM table\_name;

**注释：**只有 MS Access 支持 FIRST() 函数。

## SQL Server、MySQL 和 Oracle 中的 SQL FIRST() 工作区

### SQL Server 语法

SELECT TOP 1 column\_name FROM table\_nameORDER BYcolumn\_nameASC;

### 实例

SELECT TOP 1 CustomerName FROM Customers  
ORDER BY CustomerID ASC;

### MySQL 语法

SELECT column\_name FROM table\_name  
ORDER BY column\_name ASC  
LIMIT 1;

### 实例

SELECT CustomerName FROM Customers  
ORDER BY CustomerID ASC  
LIMIT 1;

### Oracle 语法

SELECT column\_nameFROM table\_name  
ORDER BY column\_nameASC  
WHERE ROWNUM <=1;

### 实例

SELECT CustomerName FROM Customers  
ORDER BY CustomerID ASC  
WHERE ROWNUM <=1;

## 演示数据库

在本教程中，我们将使用众所周知的 Northwind 样本数据库。

下面是选自 "Customers" 表的数据：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |

## SQL FIRST() 实例

下面的 SQL 语句选取 "Customers" 表的 "CustomerName" 列中第一个记录的值：

## 实例

SELECT FIRST(CustomerName) AS FirstCustomer FROM Customers;

# SQL LAST() 函数

## LAST() 函数

LAST() 函数返回指定的列中最后一个记录的值。

### SQL LAST() 语法

SELECT LAST(column\_name) FROM table\_name;

**注释：**只有 MS Access 支持 LAST() 函数。

## SQL Server、MySQL 和 Oracle 中的 SQL LAST() 工作区

### SQL Server 语法

SELECT TOP 1 column\_name FROM table\_nameORDER BYcolumn\_nameDESC;

### 实例

SELECT TOP 1 CustomerName FROM Customers  
ORDER BY CustomerID DESC;

### MySQL 语法

SELECT column\_name FROM table\_name  
ORDER BY column\_name DESC  
LIMIT 1;

### 实例

SELECT CustomerName FROM Customers  
ORDER BY CustomerID DESC  
LIMIT 1;

### Oracle 语法

SELECT column\_nameFROM table\_name  
ORDER BY column\_nameDESC  
WHERE ROWNUM <=1;

### 实例

SELECT CustomerName FROM Customers  
ORDER BY CustomerID DESC  
WHERE ROWNUM <=1;

## 演示数据库

在本教程中，我们将使用众所周知的 Northwind 样本数据库。

下面是选自 "Customers" 表的数据：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |

## SQL LAST() Example

下面的 SQL 语句选取 "Customers" 表的 "CustomerName" 列中最后一个记录的值：

## 实例

SELECT LAST(CustomerName) AS LastCustomer FROM Customers;

# SQL MAX() 函数

## MAX() 函数

MAX() 函数返回指定列的最大值。

### SQL MAX() 语法

SELECT MAX(column\_name) FROM table\_name;

## 演示数据库

在本教程中，我们将使用众所周知的 Northwind 样本数据库。

下面是选自 "Products" 表的数据：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ProductID** | **ProductName** | **SupplierID** | **CategoryID** | **Unit** | **Price** |
| 1 | Chais | 1 | 1 | 10 boxes x 20 bags | 18 |
| 2 | Chang | 1 | 1 | 24 - 12 oz bottles | 19 |
| 3 | Aniseed Syrup | 1 | 2 | 12 - 550 ml bottles | 10 |
| 4 | Chef Anton's Cajun Seasoning | 2 | 2 | 48 - 6 oz jars | 21.35 |
| 5 | Chef Anton's Gumbo Mix | 2 | 2 | 36 boxes | 25 |

## SQL MAX() 实例

下面的 SQL 语句从 "Products" 表的 "Price" 列获取最大值：

## 实例

SELECT MAX(Price) AS HighestPrice FROM Products;

# SQL MIN() Function

## MIN() 函数

MIN() 函数返回指定列的最小值。

### SQL MIN() 语法

SELECT MIN(column\_name) FROM table\_name;

## 演示数据库

在本教程中，我们将使用众所周知的 Northwind 样本数据库。

下面是选自 "Products" 表的数据：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ProductID** | **ProductName** | **SupplierID** | **CategoryID** | **Unit** | **Price** |
| 1 | Chais | 1 | 1 | 10 boxes x 20 bags | 18 |
| 2 | Chang | 1 | 1 | 24 - 12 oz bottles | 19 |
| 3 | Aniseed Syrup | 1 | 2 | 12 - 550 ml bottles | 10 |
| 4 | Chef Anton's Cajun Seasoning | 2 | 2 | 48 - 6 oz jars | 21.35 |
| 5 | Chef Anton's Gumbo Mix | 2 | 2 | 36 boxes | 25 |

## SQL MIN() 实例

下面的 SQL 语句从 "Products" 表的 "Price" 列获取最小值：

## 实例

SELECT MIN(Price) AS SmallestOrderPrice FROM Products;

# SQL SUM() Function

## SUM() 函数

SUM() 函数返回数值列的总数。

### SQL SUM() 语法

SELECT SUM(column\_name) FROM table\_name;

## 演示数据库

在本教程中，我们将使用众所周知的 Northwind 样本数据库。

下面是选自 "OrderDetails" 表的数据：

|  |  |  |  |
| --- | --- | --- | --- |
| **OrderDetailID** | **OrderID** | **ProductID** | **Quantity** |
| 1 | 10248 | 11 | 12 |
| 2 | 10248 | 42 | 10 |
| 3 | 10248 | 72 | 5 |
| 4 | 10249 | 14 | 9 |
| 5 | 10249 | 51 | 40 |

## SQL SUM() 实例

下面的 SQL 语句查找 "OrderDetails" 表的 "Quantity" 字段的总数：

## 实例

SELECT SUM(Quantity) AS TotalItemsOrdered FROM OrderDetails;

# SQL GROUP BY 语句

Aggregate 函数常常需要添加 GROUP BY 语句。

## GROUP BY 语句

GROUP BY 语句用于结合 Aggregate 函数，根据一个或多个列对结果集进行分组。

### SQL GROUP BY 语法

SELECT column\_name, aggregate\_function(column\_name)  
FROM table\_name  
WHERE column\_name operator value  
GROUP BY column\_name;

## 演示数据库

在本教程中，我们将使用众所周知的 Northwind 样本数据库。

下面是选自 "Orders" 表的数据：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| 10248 | 90 | 5 | 1996-07-04 | 3 |
| 10249 | 81 | 6 | 1996-07-05 | 1 |
| 10250 | 34 | 4 | 1996-07-08 | 2 |

选自 "Shippers" 表的数据：

|  |  |  |
| --- | --- | --- |
| **ShipperID** | **ShipperName** | **Phone** |
| 1 | Speedy Express | (503) 555-9831 |
| 2 | United Package | (503) 555-3199 |
| 3 | Federal Shipping | (503) 555-9931 |

选自 "Employees" 表的数据：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EmployeeID** | **LastName** | **FirstName** | **BirthDate** | **Photo** | **Notes** |
| 1 | Davolio | Nancy | 1968-12-08 | EmpID1.pic | Education includes a BA.... |
| 2 | Fuller | Andrew | 1952-02-19 | EmpID2.pic | Andrew received his BTS.... |
| 3 | Leverling | Janet | 1963-08-30 | EmpID3.pic | Janet has a BS degree.... |

## SQL GROUP BY 实例

现在我们想要查找每个送货员配送的订单数目。

下面的 SQL 语句按送货员进行订单分类统计：

## 实例

SELECT Shippers.ShipperName,COUNT(Orders.OrderID) AS NumberOfOrders FROM Orders  
LEFT JOIN Shippers  
ON Orders.ShipperID=Shippers.ShipperID  
GROUP BY ShipperName;

## GROUP BY 一个以上的列

我们也可以对一个以上的列应用 GROUP BY 语句，如下所示：

## 实例

SELECT Shippers.ShipperName, Employees.LastName,  
COUNT(Orders.OrderID) AS NumberOfOrders  
FROM ((Orders  
INNER JOIN Shippers  
ON Orders.ShipperID=Shippers.ShipperID)  
INNER JOIN Employees  
ON Orders.EmployeeID=Employees.EmployeeID)  
GROUP BY ShipperName,LastName;

# SQL HAVING 子句

## HAVING 子句

在 SQL 中增加 HAVING 子句原因是，WHERE 关键字无法与 Aggregate 函数一起使用。

### SQL HAVING 语法

SELECT column\_name, aggregate\_function(column\_name)  
FROM table\_name  
WHERE column\_name operator value  
GROUP BY column\_name  
HAVING aggregate\_function(column\_name) operator value;

## 演示数据库

在本教程中，我们将使用众所周知的 Northwind 样本数据库。

下面是选自 "Orders" 表的数据：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| 10248 | 90 | 5 | 1996-07-04 | 3 |
| 10249 | 81 | 6 | 1996-07-05 | 1 |
| 10250 | 34 | 4 | 1996-07-08 | 2 |

选自 "Employees" 表的数据：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EmployeeID** | **LastName** | **FirstName** | **BirthDate** | **Photo** | **Notes** |
| 1 | Davolio | Nancy | 1968-12-08 | EmpID1.pic | Education includes a BA.... |
| 2 | Fuller | Andrew | 1952-02-19 | EmpID2.pic | Andrew received his BTS.... |
| 3 | Leverling | Janet | 1963-08-30 | EmpID3.pic | Janet has a BS degree.... |

## SQL HAVING 实例

现在我们想要查找订单总数少于 2000 的客户。

我们使用下面的 SQL 语句：

下面的 SQL 语句查找注册超过 10 个订单的员工：

## 实例

SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders FROM (Orders  
INNER JOIN Employees  
ON Orders.EmployeeID=Employees.EmployeeID)  
GROUP BY LastName  
HAVING COUNT(Orders.OrderID) > 10;

现在我们想哟啊查找员工 "Davolio" 或 "Fuller" 是否拥有超过 25 个订单。

我们在 SQL 语句中增加一个普通的 WHERE 子句：

## 实例

SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders FROM Orders  
INNER JOIN Employees  
ON Orders.EmployeeID=Employees.EmployeeID  
WHERE LastName='Davolio' OR LastName='Fuller'  
GROUP BY LastName  
HAVING COUNT(Orders.OrderID) > 25;

# SQL UCASE() 函数

## UCASE() 函数

UCASE() 函数把字段的值转换为大写。

### SQL UCASE() 语法

SELECT UCASE(column\_name) FROM table\_name;

### 用于 SQL Server 的语法

SELECT UPPER(column\_name) FROM table\_name;

## 演示数据库

在本教程中，我们将使用众所周知的 Northwind 样本数据库。

下面是选自 "Customers" 表的数据：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |

## SQL UCASE() 实例

下面的 SQL 语句从 "Customers" 表中选取 "CustomerName" 和 "City" 列，并把 "CustomerName" 列的值转换为大写：

## 实例

SELECT UCASE(CustomerName) AS Customer, City  
FROM Customers;

# SQL LCASE() 函数

## LCASE() 函数

LCASE() 函数把字段的值转换为小写。

### SQL LCASE() 语法

SELECT LCASE(column\_name) FROM table\_name;

### 用于 SQL Server 的语法

SELECT LOWER(column\_name) FROM table\_name;

## 演示数据库

在本教程中，我们将使用众所周知的 Northwind 样本数据库。

下面是选自 "Customers" 表的数据：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |

## SQL LCASE() 实例

下面的 SQL 语句从 "Customers" 表中选取 "CustomerName" 和 "City" 列，并把 "CustomerName" 列的值转换为小写：

## 实例

SELECT LCASE(CustomerName) AS Customer, City  
FROM Customers;

# SQL MID() 函数

## MID() 函数

MID() 函数用于从文本字段中提取字符。

### SQL MID() 语法

SELECT MID(column\_name,start[,length]) FROM table\_name;

|  |  |
| --- | --- |
| **参数** | **描述** |
| column\_name | 必需。要提取字符的字段。 |
| start | 必需。规定开始位置（起始值是 1）。 |
| length | 可选。要返回的字符数。如果省略，则 MID() 函数返回剩余文本。 |

## 演示数据库

在本教程中，我们将使用众所周知的 Northwind 样本数据库。

下面是选自 "Customers" 表的数据：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |

## SQL MID() 实例

下面的 SQL 语句从 "Customers" 表的 "City" 列中提取前 4 个字符：

## 实例

SELECT MID(City,1,4) AS ShortCity  
FROM Customers;

# SQL LEN() 函数

## LEN() 函数

LEN() 函数返回文本字段中值的长度。

### SQL LEN() 语法

SELECT LEN(column\_name) FROM table\_name;

## 演示数据库

在本教程中，我们将使用众所周知的 Northwind 样本数据库。

下面是选自 "Customers" 表的数据：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |
| 4 | Around the Horn | Thomas Hardy | 120 Hanover Sq. | London | WA1 1DP | UK |
| 5 | Berglunds snabbköp | Christina Berglund | Berguvsvägen 8 | Luleå | S-958 22 | Sweden |

## SQL LEN() 实例

下面的 SQL 语句从 "Customers" 表中选取 "CustomerName" 和 "Address" 列中值的长度：

## 实例

SELECT CustomerName,LEN(Address) as LengthOfAddress  
FROM Customers;

# SQL ROUND() 函数

## ROUND() 函数

ROUND() 函数用于把数值字段舍入为指定的小数位数。

### SQL ROUND() 语法

SELECT ROUND(column\_name,decimals) FROM table\_name;

|  |  |
| --- | --- |
| **参数** | **描述** |
| column\_name | 必需。要舍入的字段。 |
| decimals | 必需。规定要返回的小数位数。 |

## 演示数据库

在本教程中，我们将使用众所周知的 Northwind 样本数据库。

下面是选自 "Products" 表的数据：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ProductID** | **ProductName** | **SupplierID** | **CategoryID** | **Unit** | **Price** |
| 1 | Chais | 1 | 1 | 10 boxes x 20 bags | 18 |
| 2 | Chang | 1 | 1 | 24 - 12 oz bottles | 19 |
| 3 | Aniseed Syrup | 1 | 2 | 12 - 550 ml bottles | 10 |
| 4 | Chef Anton's Cajun Seasoning | 2 | 2 | 48 - 6 oz jars | 21.35 |
| 5 | Chef Anton's Gumbo Mix | 2 | 2 | 36 boxes | 25 |

## SQL ROUND() 实例

下面的 SQL 语句从 "Products" 表中选取产品名称和价格舍入为最接近的整数：。提取前 4 个字符：

## 实例

SELECT ProductName, ROUND(Price,0) AS RoundedPrice  
FROM Products;

# SQL NOW() 函数

## NOW() 函数

NOW() 函数返回当前系统的日期和时间。

### SQL NOW() 语法

SELECT NOW() FROM table\_name;

## 演示数据库

在本教程中，我们将使用众所周知的 Northwind 样本数据库。

下面是选自 "Products" 表的数据：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ProductID** | **ProductName** | **SupplierID** | **CategoryID** | **Unit** | **Price** |
| 1 | Chais | 1 | 1 | 10 boxes x 20 bags | 18 |
| 2 | Chang | 1 | 1 | 24 - 12 oz bottles | 19 |
| 3 | Aniseed Syrup | 1 | 2 | 12 - 550 ml bottles | 10 |
| 4 | Chef Anton's Cajun Seasoning | 2 | 2 | 48 - 6 oz jars | 21.35 |
| 5 | Chef Anton's Gumbo Mix | 2 | 2 | 36 boxes | 25 |

## SQL NOW() 实例

下面的 SQL 语句从 "Products" 表中选取产品名称以及当天的价格：

## 实例

SELECT ProductName, Price, Now() AS PerDate  
FROM Products;

# SQL FORMAT() 函数

## FORMAT() 函数

FORMAT() 函数用于对字段的显示进行格式化。

### SQL FORMAT() 语法

SELECT FORMAT(column\_name,format) FROM table\_name;

|  |  |
| --- | --- |
| **参数** | **描述** |
| column\_name | 必需。要格式化的字段。 |
| format | 必需。规定格式。 |

## 演示数据库

在本教程中，我们将使用众所周知的 Northwind 样本数据库。

下面是选自 "Products" 表的数据：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ProductID** | **ProductName** | **SupplierID** | **CategoryID** | **Unit** | **Price** |
| 1 | Chais | 1 | 1 | 10 boxes x 20 bags | 18 |
| 2 | Chang | 1 | 1 | 24 - 12 oz bottles | 19 |
| 3 | Aniseed Syrup | 1 | 2 | 12 - 550 ml bottles | 10 |
| 4 | Chef Anton's Cajun Seasoning | 2 | 2 | 48 - 6 oz jars | 21.35 |
| 5 | Chef Anton's Gumbo Mix | 2 | 2 | 36 boxes | 25 |

## SQL FORMAT() 实例

下面的 SQL 语句从 "Products" 表中选取产品名称以及当天（格式化为 YYYY-MM-DD）的价格：

## 实例

SELECT ProductName, Price, FORMAT(Now(),'YYYY-MM-DD') AS PerDate  
FROM Products;

**来自 W3CSchool 的 SQL 快速参考**

|  |  |
| --- | --- |
| **SQL 语句** | **语法** |
| AND / OR | SELECT column\_name(s) FROM table\_name WHERE condition AND|OR condition |
| ALTER TABLE | ALTER TABLE table\_name  ADD column\_name datatype  or  ALTER TABLE table\_name  DROP COLUMN column\_name |
| AS (alias) | SELECT column\_name AS column\_alias FROM table\_name  or  SELECT column\_name FROM table\_name AS table\_alias |
| BETWEEN | SELECT column\_name(s) FROM table\_name WHERE column\_name BETWEEN value1 AND value2 |
| CREATE DATABASE | CREATE DATABASE database\_name |
| CREATE TABLE | CREATE TABLE table\_name ( column\_name1 data\_type, column\_name2 data\_type, column\_name2 data\_type, ... ) |
| CREATE INDEX | CREATE INDEX index\_name ON table\_name (column\_name)  or  CREATE UNIQUE INDEX index\_name ON table\_name (column\_name) |
| CREATE VIEW | CREATE VIEW view\_name AS SELECT column\_name(s) FROM table\_name WHERE condition |
| DELETE | DELETE FROM table\_name WHERE some\_column=some\_value  or  DELETE FROM table\_name  (**Note:**Deletes the entire table!!)  DELETE \* FROM table\_name  (**Note:**Deletes the entire table!!) |
| DROP DATABASE | DROP DATABASE database\_name |
| DROP INDEX | DROP INDEX table\_name.index\_name (SQL Server) DROP INDEX index\_name ON table\_name (MS Access) DROP INDEX index\_name (DB2/Oracle) ALTER TABLE table\_name DROP INDEX index\_name (MySQL) |
| DROP TABLE | DROP TABLE table\_name |
| GROUP BY | SELECT column\_name, aggregate\_function(column\_name) FROM table\_name WHERE column\_name operator value GROUP BY column\_name |
| HAVING | SELECT column\_name, aggregate\_function(column\_name) FROM table\_name WHERE column\_name operator value GROUP BY column\_name HAVING aggregate\_function(column\_name) operator value |
| IN | SELECT column\_name(s) FROM table\_name WHERE column\_name IN (value1,value2,..) |
| INSERT INTO | INSERT INTO table\_name VALUES (value1, value2, value3,....)  *or*  INSERT INTO table\_name (column1, column2, column3,...) VALUES (value1, value2, value3,....) |
| INNER JOIN | SELECT column\_name(s) FROM table\_name1 INNER JOIN table\_name2  ON table\_name1.column\_name=table\_name2.column\_name |
| LEFT JOIN | SELECT column\_name(s) FROM table\_name1 LEFT JOIN table\_name2  ON table\_name1.column\_name=table\_name2.column\_name |
| RIGHT JOIN | SELECT column\_name(s) FROM table\_name1 RIGHT JOIN table\_name2  ON table\_name1.column\_name=table\_name2.column\_name |
| FULL JOIN | SELECT column\_name(s) FROM table\_name1 FULL JOIN table\_name2  ON table\_name1.column\_name=table\_name2.column\_name |
| LIKE | SELECT column\_name(s) FROM table\_name WHERE column\_name LIKE pattern |
| ORDER BY | SELECT column\_name(s) FROM table\_name ORDER BY column\_name [ASC|DESC] |
| SELECT | SELECT column\_name(s) FROM table\_name |
| SELECT \* | SELECT \* FROM table\_name |
| SELECT DISTINCT | SELECT DISTINCT column\_name(s) FROM table\_name |
| SELECT INTO | SELECT \* INTO new\_table\_name [IN externaldatabase] FROM old\_table\_name  *or*  SELECT column\_name(s) INTO new\_table\_name [IN externaldatabase] FROM old\_table\_name |
| SELECT TOP | SELECT TOP number|percent column\_name(s) FROM table\_name |
| TRUNCATE TABLE | TRUNCATE TABLE table\_name |
| UNION | SELECT column\_name(s) FROM table\_name1 UNION SELECT column\_name(s) FROM table\_name2 |
| UNION ALL | SELECT column\_name(s) FROM table\_name1 UNION ALL SELECT column\_name(s) FROM table\_name2 |
| UPDATE | UPDATE table\_name SET column1=value, column2=value,... WHERE some\_column=some\_value |
| WHERE | SELECT column\_name(s) FROM table\_name WHERE column\_name operator value |

**SQL 主机**

**SQL 主机**

如果您想要您的网站存储数据在数据库并从数据库显示数据，您的 Web 服务器必须能使用 SQL 语言访问数据库系统。

如果您的 Web 服务器托管在互联网服务提供商（ISP，全称 Internet Service Provider），您必须寻找 SQL 主机计划。

最常见的 SQL 主机数据库是 MySQL、MS SQL Server 和 MS Access。

您可以在 Windows 和 Linux/UNIX 操作系统上运行 SQL 主机数据库。

下面是操作系统上对应运行的数据库系统的概览。

**MS SQL Server**

只在 Windows OS 上运行。

**MySQL**

在 Windows, Mac OS X 和 Linux/UNIX 操作系统上运行。

**MS Access**（只建议用于小型网站）

只在 Windows OS 上运行。

如需学习有关 Web 主机的知识，请访问我们的 [主机教程](http://www.w3cschool.cc/sql/sql-hosting.html)。