# SQL 简介

**SQL 是用于访问和处理数据库的标准的计算机语言。**

**什么是 SQL？**

* SQL 指结构化查询语言
* SQL 使我们有能力访问数据库
* SQL 是一种 ANSI 的标准计算机语言

**编者注：**ANSI，美国国家标准化组织

**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纪录
* 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...**

SQL 是一门 ANSI 的标准计算机语言，用来访问和操作数据库系统。SQL 语句用于取回和更新数据库中的数据。SQL 可与数据库程序协同工作，比如 MS Access、DB2、Informix、MS SQL Server、Oracle、Sybase 以及其他数据库系统。

不幸地是，存在着很多不同版本的 SQL 语言，但是为了与 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关键词（比如 SELECT、UPDATE、DELETE、INSERT、WHERE 等等）。

**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私有扩展！

**在您的网站中使用 SQL**

要创建发布数据库中数据的网站，您需要以下要素：

* RDBMS 数据库程序（比如 MS Access, SQL Server, MySQL）
* 服务器端脚本语言（比如 PHP 或 ASP）
* SQL
* HTML / CSS

**RDBMS**

RDBMS 指的是关系型数据库管理系统。

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（tables）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“客户”或者“订单”）。表包含带有数据的记录（行）。

下面的例子是一个名为 "Persons" 的表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Adams | John | Oxford Street | London |
| 2 | Bush | George | Fifth Avenue | New York |
| 3 | Carter | Thomas | Changan Street | Beijing |

上面的表包含三条记录（每一条对应一个人）和五个列（Id、姓、名、地址和城市）。

**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。

下面的语句从表中选取 LastName 列的数据：

SELECT LastName FROM Persons

结果集类似这样：

|  |
| --- |
| **LastName** |
| Adams |
| Bush |
| Carter |

在本教程中，我们将为您讲解各种不同的 SQL 语句。

**重要事项**

一定要记住，**SQL 对大小写不敏感**！

**SQL 语句后面的分号？**

某些数据库系统要求在每条 SQL 命令的末端使用分号。在我们的教程中不使用分号。

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语句。

如果您使用的是 MS Access 和 SQL Server 2000，则不必在每条 SQL 语句之后使用分号，不过某些数据库软件要求必须使用分号。

**SQL DML 和 DDL**

可以把 SQL 分为两个部分：数据操作语言 (DML) 和 数据定义语言 (DDL)。

SQL (结构化查询语言)是用于执行查询的语法。但是 SQL 语言也包含用于更新、插入和删除记录的语法。

查询和更新指令构成了 SQL 的 DML 部分：

* **SELECT** - 从数据库表中获取数据
* **UPDATE** - 更新数据库表中的数据
* **DELETE** - 从数据库表中删除数据
* **INSERT INTO** - 向数据库表中插入数据

SQL 的数据定义语言 (DDL) 部分使我们有能力创建或删除表格。我们也可以定义索引（键），规定表之间的链接，以及施加表间的约束。

SQL 中最重要的 DDL 语句:

* **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

# SELECT 语句

* [Previous Page](http://www.w3school.com.cn/sql/sql_syntax.asp)
* [Next Page](http://www.w3school.com.cn/sql/sql_distinct.asp)

**本章讲解 SELECT 和 SELECT \* 语句。**

**SQL SELECT 语句**

SELECT 语句用于从表中选取数据。

结果被存储在一个结果表中（称为结果集）。

**SQL SELECT 语法**

SELECT 列名称 FROM 表名称

以及：

SELECT \* FROM 表名称

**注释：**SQL 语句对大小写不敏感。SELECT 等效于 select。

**SQL SELECT 实例**

如需获取名为 "LastName" 和 "FirstName" 的列的内容（从名为 "Persons" 的数据库表），请使用类似这样的 SELECT 语句：

SELECT LastName,FirstName FROM Persons

**"Persons" 表:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Adams | John | Oxford Street | London |
| 2 | Bush | George | Fifth Avenue | New York |
| 3 | Carter | Thomas | Changan Street | Beijing |

**结果：**

|  |  |
| --- | --- |
| **LastName** | **FirstName** |
| Adams | John |
| Bush | George |
| Carter | Thomas |

**SQL SELECT \* 实例**

现在我们希望从 "Persons" 表中选取所有的列。

请使用符号 \* 取代列的名称，就像这样：

SELECT \* FROM Persons

**提示：**星号（\*）是选取所有列的快捷方式。

**结果：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Adams | John | Oxford Street | London |
| 2 | Bush | George | Fifth Avenue | New York |
| 3 | Carter | Thomas | Changan Street | Beijing |

**在结果集（result-set）中导航**

由 SQL 查询程序获得的结果被存放在一个结果集中。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。

# SELECT DISTINCT 语句

**本章讲解 SELECT DISTINCT 语句。**

**SQL SELECT DISTINCT 语句**

在表中，可能会包含重复值。这并不成问题，不过，有时您也许希望仅仅列出不同（distinct）的值。

关键词 DISTINCT 用于返回唯一不同的值。

**语法：**

SELECT DISTINCT 列名称 FROM 表名称

**使用 DISTINCT 关键词**

如果要从 "Company" 列中选取所有的值，我们需要使用 SELECT 语句：

SELECT Company FROM Orders

**"Orders"表：**

|  |  |
| --- | --- |
| **Company** | **OrderNumber** |
| IBM | 3532 |
| W3School | 2356 |
| Apple | 4698 |
| W3School | 6953 |

**结果：**

|  |
| --- |
| **Company** |
| IBM |
| W3School |
| Apple |
| W3School |

请注意，在结果集中，W3School 被列出了两次。

如需从 Company" 列中仅选取唯一不同的值，我们需要使用 SELECT DISTINCT 语句：

SELECT DISTINCT Company FROM Orders

**结果：**

|  |
| --- |
| **Company** |
| IBM |
| W3School |
| Apple |

现在，在结果集中，"W3School" 仅被列出了一次。

# WHERE 子句

* [Previous Page](http://www.w3school.com.cn/sql/sql_distinct.asp)
* [Next Page](http://www.w3school.com.cn/sql/sql_and_or.asp)

**WHERE 子句用于规定选择的标准。**

**WHERE 子句**

如需有条件地从表中选取数据，可将 WHERE 子句添加到 SELECT 语句。

**语法**

SELECT 列名称 FROM 表名称 WHERE 列 运算符 值

下面的运算符可在 WHERE 子句中使用：

|  |  |
| --- | --- |
| **操作符** | **描述** |
| = | 等于 |
| <> | 不等于 |
| > | 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 |

**注释：**在某些版本的 SQL 中，操作符 <> 可以写为 !=。

**使用 WHERE 子句**

如果只希望选取居住在城市 "Beijing" 中的人，我们需要向 SELECT 语句添加 WHERE 子句：

SELECT \* FROM Persons WHERE City='Beijing'

**"Persons" 表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **LastName** | **FirstName** | **Address** | **City** | **Year** |
| Adams | John | Oxford Street | London | 1970 |
| Bush | George | Fifth Avenue | New York | 1975 |
| Carter | Thomas | Changan Street | Beijing | 1980 |
| Gates | Bill | Xuanwumen 10 | Beijing | 1985 |

**结果：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **LastName** | **FirstName** | **Address** | **City** | **Year** |
| Carter | Thomas | Changan Street | Beijing | 1980 |
| Gates | Bill | Xuanwumen 10 | Beijing | 1985 |

**引号的使用**

请注意，我们在例子中的条件值周围使用的是单引号。

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**，请不要使用引号。

**文本值：**

这是正确的：

SELECT \* FROM Persons WHERE FirstName='Bush'

这是错误的：

SELECT \* FROM Persons WHERE FirstName=Bush

**数值：**

这是正确的：

SELECT \* FROM Persons WHERE Year>1965

这是错误的：

SELECT \* FROM Persons WHERE Year>'1965'

# AND & OR 运算符

* [Previous Page](http://www.w3school.com.cn/sql/sql_where.asp)
* [Next Page](http://www.w3school.com.cn/sql/sql_orderby.asp)

**AND 和 OR 运算符用于基于一个以上的条件对记录进行过滤。**

**AND 和 OR 运算符**

AND 和 OR 可在 WHERE 子语句中把两个或多个条件结合起来。

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。

**原始的表 (用在例子中的)：**

|  |  |  |  |
| --- | --- | --- | --- |
| **LastName** | **FirstName** | **Address** | **City** |
| Adams | John | Oxford Street | London |
| Bush | George | Fifth Avenue | New York |
| Carter | Thomas | Changan Street | Beijing |
| Carter | William | Xuanwumen 10 | Beijing |

**AND 运算符实例**

使用 AND 来显示所有姓为 "Carter" 并且名为 "Thomas" 的人：

SELECT \* FROM Persons WHERE FirstName='Thomas' AND LastName='Carter'

**结果：**

|  |  |  |  |
| --- | --- | --- | --- |
| **LastName** | **FirstName** | **Address** | **City** |
| Carter | Thomas | Changan Street | Beijing |

**OR 运算符实例**

使用 OR 来显示所有姓为 "Carter" 或者名为 "Thomas" 的人：

SELECT \* FROM Persons WHERE firstname='Thomas' OR lastname='Carter'

**结果：**

|  |  |  |  |
| --- | --- | --- | --- |
| **LastName** | **FirstName** | **Address** | **City** |
| Carter | Thomas | Changan Street | Beijing |
| Carter | William | Xuanwumen 10 | Beijing |

**结合 AND 和 OR 运算符**

我们也可以把 AND 和 OR 结合起来（使用圆括号来组成复杂的表达式）:

SELECT \* FROM Persons WHERE (FirstName='Thomas' OR FirstName='William')

AND LastName='Carter'

**结果：**

|  |  |  |  |
| --- | --- | --- | --- |
| **LastName** | **FirstName** | **Address** | **City** |
| Carter | Thomas | Changan Street | Beijing |
| Carter | William | Xuanwumen 10 | Beijing |

# ORDER BY 子句

* [Previous Page](http://www.w3school.com.cn/sql/sql_and_or.asp)
* [Next Page](http://www.w3school.com.cn/sql/sql_insert.asp)

**ORDER BY 语句用于对结果集进行排序。**

**ORDER BY 语句**

ORDER BY 语句用于根据指定的列对结果集进行排序。

ORDER BY 语句默认按照升序对记录进行排序。

如果您希望按照降序对记录进行排序，可以使用 DESC 关键字。

**原始的表 (用在例子中的)：**

Orders 表:

|  |  |
| --- | --- |
| **Company** | **OrderNumber** |
| IBM | 3532 |
| W3School | 2356 |
| Apple | 4698 |
| W3School | 6953 |

**实例 1**

以字母顺序显示公司名称：

SELECT Company, OrderNumber FROM Orders ORDER BY Company

**结果：**

|  |  |
| --- | --- |
| **Company** | **OrderNumber** |
| Apple | 4698 |
| IBM | 3532 |
| W3School | 6953 |
| W3School | 2356 |

**实例 2**

以字母顺序显示公司名称（Company），并以数字顺序显示顺序号（OrderNumber）：

SELECT Company, OrderNumber FROM Orders ORDER BY Company, OrderNumber

结果：

|  |  |
| --- | --- |
| **Company** | **OrderNumber** |
| Apple | 4698 |
| IBM | 3532 |
| W3School | 2356 |
| W3School | 6953 |

**实例 3**

以逆字母顺序显示公司名称：

SELECT Company, OrderNumber FROM Orders ORDER BY Company DESC

**结果：**

|  |  |
| --- | --- |
| **Company** | **OrderNumber** |
| W3School | 6953 |
| W3School | 2356 |
| IBM | 3532 |
| Apple | 4698 |

**实例 4**

以逆字母顺序显示公司名称，并以数字顺序显示顺序号：

SELECT Company, OrderNumber FROM Orders ORDER BY Company DESC, OrderNumber ASC

**结果：**

|  |  |
| --- | --- |
| **Company** | **OrderNumber** |
| W3School | 2356 |
| W3School | 6953 |
| IBM | 3532 |
| Apple | 4698 |

**注意：**在以上的结果中有两个相等的公司名称 (W3School)。只有这一次，在第一列中有相同的值时，第二列是以升序排列的。如果第一列中有些值为 nulls 时，情况也是这样的。

# INSERT INTO 语句

* [Previous Page](http://www.w3school.com.cn/sql/sql_orderby.asp)
* [Next Page](http://www.w3school.com.cn/sql/sql_update.asp)

**INSERT INTO 语句**

INSERT INTO 语句用于向表格中插入新的行。

**语法**

INSERT INTO 表名称 VALUES (值1, 值2,....)

我们也可以指定所要插入数据的列：

INSERT INTO table\_name (列1, 列2,...) VALUES (值1, 值2,....)

**插入新的行**

**"Persons" 表：**

|  |  |  |  |
| --- | --- | --- | --- |
| **LastName** | **FirstName** | **Address** | **City** |
| Carter | Thomas | Changan Street | Beijing |

**SQL 语句：**

INSERT INTO Persons VALUES ('Gates', 'Bill', 'Xuanwumen 10', 'Beijing')

**结果：**

|  |  |  |  |
| --- | --- | --- | --- |
| **LastName** | **FirstName** | **Address** | **City** |
| Carter | Thomas | Changan Street | Beijing |
| Gates | Bill | Xuanwumen 10 | Beijing |

**在指定的列中插入数据**

**"Persons" 表：**

|  |  |  |  |
| --- | --- | --- | --- |
| **LastName** | **FirstName** | **Address** | **City** |
| Carter | Thomas | Changan Street | Beijing |
| Gates | Bill | Xuanwumen 10 | Beijing |

**SQL 语句：**

INSERT INTO Persons (LastName, Address) VALUES ('Wilson', 'Champs-Elysees')

**结果：**

|  |  |  |  |
| --- | --- | --- | --- |
| **LastName** | **FirstName** | **Address** | **City** |
| Carter | Thomas | Changan Street | Beijing |
| Gates | Bill | Xuanwumen 10 | Beijing |
| Wilson |  | Champs-Elysees |  |

# UPDATE 语句

* [Previous Page](http://www.w3school.com.cn/sql/sql_insert.asp)
* [Next Page](http://www.w3school.com.cn/sql/sql_delete.asp)

**Update 语句**

Update 语句用于修改表中的数据。

**语法：**

UPDATE 表名称 SET 列名称 = 新值 WHERE 列名称 = 某值

**Person:**

|  |  |  |  |
| --- | --- | --- | --- |
| **LastName** | **FirstName** | **Address** | **City** |
| Gates | Bill | Xuanwumen 10 | Beijing |
| Wilson |  | Champs-Elysees |  |

**更新某一行中的一个列**

我们为 lastname 是 "Wilson" 的人添加 firstname：

UPDATE Person SET FirstName = 'Fred' WHERE LastName = 'Wilson'

**结果：**

|  |  |  |  |
| --- | --- | --- | --- |
| **LastName** | **FirstName** | **Address** | **City** |
| Gates | Bill | Xuanwumen 10 | Beijing |
| Wilson | Fred | Champs-Elysees |  |

**更新某一行中的若干列**

我们会修改地址（address），并添加城市名称（city）：

UPDATE Person SET Address = 'Zhongshan 23', City = 'Nanjing'

WHERE LastName = 'Wilson'

**结果：**

|  |  |  |  |
| --- | --- | --- | --- |
| **LastName** | **FirstName** | **Address** | **City** |
| Gates | Bill | Xuanwumen 10 | Beijing |
| Wilson | Fred | Zhongshan 23 | Nanjing |

# DELETE 语句

* [Previous Page](http://www.w3school.com.cn/sql/sql_update.asp)
* [Next Page](http://www.w3school.com.cn/sql/sql_top.asp)

**DELETE 语句**

DELETE 语句用于删除表中的行。

**语法**

DELETE FROM 表名称 WHERE 列名称 = 值

**Person:**

|  |  |  |  |
| --- | --- | --- | --- |
| **LastName** | **FirstName** | **Address** | **City** |
| Gates | Bill | Xuanwumen 10 | Beijing |
| Wilson | Fred | Zhongshan 23 | Nanjing |

**删除某行**

"Fred Wilson" 会被删除：

DELETE FROM Person WHERE LastName = 'Wilson'

**结果:**

|  |  |  |  |
| --- | --- | --- | --- |
| **LastName** | **FirstName** | **Address** | **City** |
| Gates | Bill | Xuanwumen 10 | Beijing |

**删除所有行**

可以在不删除表的情况下删除所有的行。这意味着表的结构、属性和索引都是完整的：

DELETE FROM table\_name

或者：

DELETE \* FROM table\_name

# TOP 子句

**TOP 子句**

TOP 子句用于规定要返回的记录的数目。

对于拥有数千条记录的大型表来说，TOP 子句是非常有用的。

**注释：**并非所有的数据库系统都支持 TOP 子句。

**SQL Server 的语法：**

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

**例子**

SELECT \*

FROM Persons

LIMIT 5

**Oracle 语法**

SELECT column\_name(s)

FROM table\_name

WHERE ROWNUM <= number

**例子**

SELECT \*

FROM Persons

WHERE ROWNUM <= 5

**原始的表 (用在例子中的)：**

Persons 表:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Adams | John | Oxford Street | London |
| 2 | Bush | George | Fifth Avenue | New York |
| 3 | Carter | Thomas | Changan Street | Beijing |
| 4 | Obama | Barack | Pennsylvania Avenue | Washington |

**SQL TOP 实例**

现在，我们希望从上面的 "Persons" 表中选取头两条记录。

我们可以使用下面的 SELECT 语句：

SELECT TOP 2 \* FROM Persons

**结果：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Adams | John | Oxford Street | London |
| 2 | Bush | George | Fifth Avenue | New York |

**SQL TOP PERCENT 实例**

现在，我们希望从上面的 "Persons" 表中选取 50% 的记录。

我们可以使用下面的 SELECT 语句：

SELECT TOP 50 PERCENT \* FROM Persons

结果：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Adams | John | Oxford Street | London |
| 2 | Bush | George | Fifth Avenue | New York |

# LIKE 操作符

**LIKE 操作符用于在 WHERE 子句中搜索列中的指定模式。**

**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 操作符语法**

SELECT column\_name(s)

FROM table\_name

WHERE column\_name LIKE pattern

**原始的表 (用在例子中的)：**

Persons 表:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Adams | John | Oxford Street | London |
| 2 | Bush | George | Fifth Avenue | New York |
| 3 | Carter | Thomas | Changan Street | Beijing |

**LIKE 操作符实例**

**例子 1**

现在，我们希望从上面的 "Persons" 表中选取居住在以 "N" 开始的城市里的人：

我们可以使用下面的 SELECT 语句：

SELECT \* FROM Persons

WHERE City LIKE 'N%'

**提示：**"%" 可用于定义通配符（模式中缺少的字母）。

**结果集：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **LastName** | **FirstName** | **Address** | **City** |
| 2 | Bush | George | Fifth Avenue | New York |

**例子 2**

接下来，我们希望从 "Persons" 表中选取居住在以 "g" 结尾的城市里的人：

我们可以使用下面的 SELECT 语句：

SELECT \* FROM Persons

WHERE City LIKE '%g'

**结果集：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **LastName** | **FirstName** | **Address** | **City** |
| 3 | Carter | Thomas | Changan Street | Beijing |

**例子 3**

接下来，我们希望从 "Persons" 表中选取居住在包含 "lon" 的城市里的人：

我们可以使用下面的 SELECT 语句：

SELECT \* FROM Persons

WHERE City LIKE '%lon%'

**结果集：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Adams | John | Oxford Street | London |

**例子 4**

通过使用 NOT 关键字，我们可以从 "Persons" 表中选取居住在**不包含** "lon" 的城市里的人：

我们可以使用下面的 SELECT 语句：

SELECT \* FROM Persons

WHERE City NOT LIKE '%lon%'

**结果集：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **LastName** | **FirstName** | **Address** | **City** |
| 2 | Bush | George | Fifth Avenue | New York |
| 3 | Carter | Thomas | Changan Street | Beijing |

# 通配符

* [Previous Page](http://www.w3school.com.cn/sql/sql_like.asp)
* [Next Page](http://www.w3school.com.cn/sql/sql_in.asp)

**在搜索数据库中的数据时，您可以使用 SQL 通配符。**

**SQL 通配符**

在搜索数据库中的数据时，SQL 通配符可以替代一个或多个字符。

SQL 通配符必须与 LIKE 运算符一起使用。

在 SQL 中，可使用以下通配符：

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

**原始的表 (用在例子中的)：**

Persons 表:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Adams | John | Oxford Street | London |
| 2 | Bush | George | Fifth Avenue | New York |
| 3 | Carter | Thomas | Changan Street | Beijing |

**使用 % 通配符**

**例子 1**

现在，我们希望从上面的 "Persons" 表中选取居住在以 "Ne" 开始的城市里的人：

我们可以使用下面的 SELECT 语句：

SELECT \* FROM Persons

WHERE City LIKE 'Ne%'

**结果集：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **LastName** | **FirstName** | **Address** | **City** |
| 2 | Bush | George | Fifth Avenue | New York |

**例子 2**

接下来，我们希望从 "Persons" 表中选取居住在包含 "lond" 的城市里的人：

我们可以使用下面的 SELECT 语句：

SELECT \* FROM Persons

WHERE City LIKE '%lond%'

**结果集：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Adams | John | Oxford Street | London |

**使用 \_ 通配符**

**例子 1**

现在，我们希望从上面的 "Persons" 表中选取名字的第一个字符之后是 "eorge" 的人：

我们可以使用下面的 SELECT 语句：

SELECT \* FROM Persons

WHERE FirstName LIKE '\_eorge'

**结果集：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **LastName** | **FirstName** | **Address** | **City** |
| 2 | Bush | George | Fifth Avenue | New York |

**例子 2**

接下来，我们希望从 "Persons" 表中选取的这条记录的姓氏以 "C" 开头，然后是一个任意字符，然后是 "r"，然后是任意字符，然后是 "er"：

我们可以使用下面的 SELECT 语句：

SELECT \* FROM Persons

WHERE LastName LIKE 'C\_r\_er'

**结果集：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **LastName** | **FirstName** | **Address** | **City** |
| 3 | Carter | Thomas | Changan Street | Beijing |

**使用 [charlist] 通配符**

**例子 1**

现在，我们希望从上面的 "Persons" 表中选取居住的城市以 "A" 或 "L" 或 "N" 开头的人：

我们可以使用下面的 SELECT 语句：

SELECT \* FROM Persons

WHERE City LIKE '[ALN]%'

**结果集：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Adams | John | Oxford Street | London |
| 2 | Bush | George | Fifth Avenue | New York |

**例子 2**

现在，我们希望从上面的 "Persons" 表中选取居住的城市**不以** "A" 或 "L" 或 "N" 开头的人：

我们可以使用下面的 SELECT 语句：

SELECT \* FROM Persons

WHERE City LIKE '[!ALN]%'

**结果集：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **LastName** | **FirstName** | **Address** | **City** |
| 3 | Carter | Thomas | Changan Street | Beijing |

# IN 操作符

**IN 操作符**

IN 操作符允许我们在 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,...)

**原始的表 (在实例中使用：)**

Persons 表:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Adams | John | Oxford Street | London |
| 2 | Bush | George | Fifth Avenue | New York |
| 3 | Carter | Thomas | Changan Street | Beijing |

**IN 操作符实例**

现在，我们希望从上表中选取姓氏为 Adams 和 Carter 的人：

我们可以使用下面的 SELECT 语句：

SELECT \* FROM Persons

WHERE LastName IN ('Adams','Carter')

**结果集：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Adams | John | Oxford Street | London |
| 3 | Carter | Thomas | Changan Street | Beijing |

# BETWEEN 操作符

* [Previous Page](http://www.w3school.com.cn/sql/sql_in.asp)
* [Next Page](http://www.w3school.com.cn/sql/sql_alias.asp)

**BETWEEN 操作符在 WHERE 子句中使用，作用是选取介于两个值之间的数据范围。**

**BETWEEN 操作符**

操作符 BETWEEN ... AND 会选取介于两个值之间的数据范围。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\_name

BETWEEN value1 AND value2

**原始的表 (在实例中使用：)**

Persons 表:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Adams | John | Oxford Street | London |
| 2 | Bush | George | Fifth Avenue | New York |
| 3 | Carter | Thomas | Changan Street | Beijing |
| 4 | Gates | Bill | Xuanwumen 10 | Beijing |

**BETWEEN 操作符实例**

如需以字母顺序显示介于 "Adams"（包括）和 "Carter"（不包括）之间的人，请使用下面的 SQL：

SELECT \* FROM Persons

WHERE LastName

BETWEEN 'Adams' AND 'Carter'

**结果集：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Adams | John | Oxford Street | London |
| 2 | Bush | George | Fifth Avenue | New York |

**重要事项：**不同的数据库对 BETWEEN...AND 操作符的处理方式是有差异的。某些数据库会列出介于 "Adams" 和 "Carter" 之间的人，但不包括 "Adams" 和 "Carter" ；某些数据库会列出介于 "Adams" 和 "Carter" 之间并包括 "Adams" 和 "Carter" 的人；而另一些数据库会列出介于 "Adams" 和 "Carter" 之间的人，包括 "Adams" ，但不包括 "Carter" 。

所以，请检查你的数据库是如何处理 BETWEEN....AND 操作符的！

**实例 2**

如需使用上面的例子显示范围之外的人，请使用 NOT 操作符：

SELECT \* FROM Persons

WHERE LastName

NOT BETWEEN 'Adams' AND 'Carter'

**结果集：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **LastName** | **FirstName** | **Address** | **City** |
| 3 | Carter | Thomas | Changan Street | Beijing |
| 4 | Gates | Bill | Xuanwumen 10 | Beijing |

# Alias（别名）

* [Previous Page](http://www.w3school.com.cn/sql/sql_between.asp)
* [Next Page](http://www.w3school.com.cn/sql/sql_join.asp)

**通过使用 SQL，可以为列名称和表名称指定别名（Alias）。**

**SQL Alias**

**表的 SQL Alias 语法**

SELECT column\_name(s)

FROM table\_name

AS alias\_name

**列的 SQL Alias 语法**

SELECT column\_name AS alias\_name

FROM table\_name

**Alias 实例: 使用表名称别名**

假设我们有两个表分别是："Persons" 和 "Product\_Orders"。我们分别为它们指定别名 "p" 和 "po"。

现在，我们希望列出 "John Adams" 的所有定单。

我们可以使用下面的 SELECT 语句：

SELECT po.OrderID, p.LastName, p.FirstName

FROM Persons AS p, Product\_Orders AS po

WHERE p.LastName='Adams' AND p.FirstName='John'

不使用别名的 SELECT 语句：

SELECT Product\_Orders.OrderID, Persons.LastName, Persons.FirstName

FROM Persons, Product\_Orders

WHERE Persons.LastName='Adams' AND Persons.FirstName='John'

从上面两条 SELECT 语句您可以看到，别名使查询程序更易阅读和书写。

**Alias 实例: 使用一个列名别名**

**表 Persons:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Adams | John | Oxford Street | London |
| 2 | Bush | George | Fifth Avenue | New York |
| 3 | Carter | Thomas | Changan Street | Beijing |

**SQL:**

SELECT LastName AS Family, FirstName AS Name

FROM Persons

**结果：**

|  |  |
| --- | --- |
| **Family** | **Name** |
| Adams | John |
| Bush | George |
| Carter | Thomas |

# JOIN

**SQL join 用于根据两个或多个表中的列之间的关系，从这些表中查询数据。**

**Join 和 Key**

有时为了得到完整的结果，我们需要从两个或更多的表中获取结果。我们就需要执行 join。

数据库中的表可通过键将彼此联系起来。主键（Primary Key）是一个列，在这个列中的每一行的值都是唯一的。在表中，每个主键的值都是唯一的。这样做的目的是在不重复每个表中的所有数据的情况下，把表间的数据交叉捆绑在一起。

请看 "Persons" 表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id\_P** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Adams | John | Oxford Street | London |
| 2 | Bush | George | Fifth Avenue | New York |
| 3 | Carter | Thomas | Changan Street | Beijing |

请注意，"Id\_P" 列是 Persons 表中的的主键。这意味着没有两行能够拥有相同的 Id\_P。即使两个人的姓名完全相同，Id\_P 也可以区分他们。

接下来请看 "Orders" 表：

|  |  |  |
| --- | --- | --- |
| **Id\_O** | **OrderNo** | **Id\_P** |
| 1 | 77895 | 3 |
| 2 | 44678 | 3 |
| 3 | 22456 | 1 |
| 4 | 24562 | 1 |
| 5 | 34764 | 65 |

请注意，"Id\_O" 列是 Orders 表中的的主键，同时，"Orders" 表中的 "Id\_P" 列用于引用 "Persons" 表中的人，而无需使用他们的确切姓名。

请留意，"Id\_P" 列把上面的两个表联系了起来。

**引用两个表**

我们可以通过引用两个表的方式，从两个表中获取数据：

谁订购了产品，并且他们订购了什么产品？

SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo

FROM Persons, Orders

WHERE Persons.Id\_P = Orders.Id\_P

结果集：

|  |  |  |
| --- | --- | --- |
| **LastName** | **FirstName** | **OrderNo** |
| Adams | John | 22456 |
| Adams | John | 24562 |
| Carter | Thomas | 77895 |
| Carter | Thomas | 44678 |

**SQL JOIN - 使用 Join**

除了上面的方法，我们也可以使用关键词 JOIN 来从两个表中获取数据。

如果我们希望列出所有人的定购，可以使用下面的 SELECT 语句：

SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo

FROM Persons

INNER JOIN Orders

ON Persons.Id\_P = Orders.Id\_P

ORDER BY Persons.LastName

结果集：

|  |  |  |
| --- | --- | --- |
| **LastName** | **FirstName** | **OrderNo** |
| Adams | John | 22456 |
| Adams | John | 24562 |
| Carter | Thomas | 77895 |
| Carter | Thomas | 44678 |

**不同的 SQL JOIN**

除了我们在上面的例子中使用的 INNER JOIN（内连接），我们还可以使用其他几种连接。

下面列出了您可以使用的 JOIN 类型，以及它们之间的差异。

* 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，就返回行

# INNER JOIN 关键字

**SQL INNER JOIN 关键字**

在表中存在至少一个匹配时，INNER JOIN 关键字返回行。

**INNER JOIN 关键字语法**

SELECT column\_name(s)

FROM table\_name1

INNER JOIN table\_name2

ON table\_name1.column\_name=table\_name2.column\_name

**注释：**INNER JOIN 与 JOIN 是相同的。

**原始的表 (用在例子中的)：**

"Persons" 表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id\_P** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Adams | John | Oxford Street | London |
| 2 | Bush | George | Fifth Avenue | New York |
| 3 | Carter | Thomas | Changan Street | Beijing |

"Orders" 表：

|  |  |  |
| --- | --- | --- |
| **Id\_O** | **OrderNo** | **Id\_P** |
| 1 | 77895 | 3 |
| 2 | 44678 | 3 |
| 3 | 22456 | 1 |
| 4 | 24562 | 1 |
| 5 | 34764 | 65 |

**内连接（INNER JOIN）实例**

现在，我们希望列出所有人的定购。

您可以使用下面的 SELECT 语句：

SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo

FROM Persons

INNER JOIN Orders

ON Persons.Id\_P=Orders.Id\_P

ORDER BY Persons.LastName

结果集：

|  |  |  |
| --- | --- | --- |
| **LastName** | **FirstName** | **OrderNo** |
| Adams | John | 22456 |
| Adams | John | 24562 |
| Carter | Thomas | 77895 |
| Carter | Thomas | 44678 |

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

# LEFT JOIN 关键字

LEFT JOIN 关键字会从左表 (table\_name1) 那里返回所有的行，即使在右表 (table\_name2) 中没有匹配的行。

**LEFT JOIN 关键字语法**

SELECT column\_name(s)

FROM table\_name1

LEFT JOIN table\_name2

ON table\_name1.column\_name=table\_name2.column\_name

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

**原始的表 (用在例子中的)：**

"Persons" 表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id\_P** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Adams | John | Oxford Street | London |
| 2 | Bush | George | Fifth Avenue | New York |
| 3 | Carter | Thomas | Changan Street | Beijing |

"Orders" 表：

|  |  |  |
| --- | --- | --- |
| **Id\_O** | **OrderNo** | **Id\_P** |
| 1 | 77895 | 3 |
| 2 | 44678 | 3 |
| 3 | 22456 | 1 |
| 4 | 24562 | 1 |
| 5 | 34764 | 65 |

**左连接（LEFT JOIN）实例**

现在，我们希望列出所有的人，以及他们的定购 - 如果有的话。

您可以使用下面的 SELECT 语句：

SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo

FROM Persons

LEFT JOIN Orders

ON Persons.Id\_P=Orders.Id\_P

ORDER BY Persons.LastName

结果集：

|  |  |  |
| --- | --- | --- |
| **LastName** | **FirstName** | **OrderNo** |
| Adams | John | 22456 |
| Adams | John | 24562 |
| Carter | Thomas | 77895 |
| Carter | Thomas | 44678 |
| Bush | George |  |

LEFT JOIN 关键字会从左表 (Persons) 那里返回所有的行，即使在右表 (Orders) 中没有匹配的行。

# RIGHT JOIN 关键字

* [Previous Page](http://www.w3school.com.cn/sql/sql_join_left.asp)
* [Next Page](http://www.w3school.com.cn/sql/sql_join_full.asp)

**SQL RIGHT JOIN 关键字**

RIGHT JOIN 关键字会右表 (table\_name2) 那里返回所有的行，即使在左表 (table\_name1) 中没有匹配的行。

**RIGHT JOIN 关键字语法**

SELECT column\_name(s)

FROM table\_name1

RIGHT JOIN table\_name2

ON table\_name1.column\_name=table\_name2.column\_name

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

**原始的表 (用在例子中的)：**

"Persons" 表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id\_P** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Adams | John | Oxford Street | London |
| 2 | Bush | George | Fifth Avenue | New York |
| 3 | Carter | Thomas | Changan Street | Beijing |

"Orders" 表：

|  |  |  |
| --- | --- | --- |
| **Id\_O** | **OrderNo** | **Id\_P** |
| 1 | 77895 | 3 |
| 2 | 44678 | 3 |
| 3 | 22456 | 1 |
| 4 | 24562 | 1 |
| 5 | 34764 | 65 |

**右连接（RIGHT JOIN）实例**

现在，我们希望列出所有的定单，以及定购它们的人 - 如果有的话。

您可以使用下面的 SELECT 语句：

SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo

FROM Persons

RIGHT JOIN Orders

ON Persons.Id\_P=Orders.Id\_P

ORDER BY Persons.LastName

结果集：

|  |  |  |
| --- | --- | --- |
| **LastName** | **FirstName** | **OrderNo** |
| Adams | John | 22456 |
| Adams | John | 24562 |
| Carter | Thomas | 77895 |
| Carter | Thomas | 44678 |
|  |  | 34764 |

RIGHT JOIN 关键字会从右表 (Orders) 那里返回所有的行，即使在左表 (Persons) 中没有匹配的行。

# FULL JOIN 关键字

* [Previous Page](http://www.w3school.com.cn/sql/sql_join_right.asp)
* [Next Page](http://www.w3school.com.cn/sql/sql_union.asp)

**SQL FULL JOIN 关键字**

只要其中某个表存在匹配，FULL JOIN 关键字就会返回行。

**FULL JOIN 关键字语法**

SELECT column\_name(s)

FROM table\_name1

FULL JOIN table\_name2

ON table\_name1.column\_name=table\_name2.column\_name

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

**原始的表 (用在例子中的)：**

"Persons" 表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id\_P** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Adams | John | Oxford Street | London |
| 2 | Bush | George | Fifth Avenue | New York |
| 3 | Carter | Thomas | Changan Street | Beijing |

"Orders" 表：

|  |  |  |
| --- | --- | --- |
| **Id\_O** | **OrderNo** | **Id\_P** |
| 1 | 77895 | 3 |
| 2 | 44678 | 3 |
| 3 | 22456 | 1 |
| 4 | 24562 | 1 |
| 5 | 34764 | 65 |

**全连接（FULL JOIN）实例**

现在，我们希望列出所有的人，以及他们的定单，以及所有的定单，以及定购它们的人。

您可以使用下面的 SELECT 语句：

SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo

FROM Persons

FULL JOIN Orders

ON Persons.Id\_P=Orders.Id\_P

ORDER BY Persons.LastName

结果集：

|  |  |  |
| --- | --- | --- |
| **LastName** | **FirstName** | **OrderNo** |
| Adams | John | 22456 |
| Adams | John | 24562 |
| Carter | Thomas | 77895 |
| Carter | Thomas | 44678 |
| Bush | George |  |
|  |  | 34764 |

FULL JOIN 关键字会从左表 (Persons) 和右表 (Orders) 那里返回所有的行。如果 "Persons" 中的行在表 "Orders" 中没有匹配，或者如果 "Orders" 中的行在表 "Persons" 中没有匹配，这些行同样会列出。

# UNION 和 UNION ALL 操作符

* [Previous Page](http://www.w3school.com.cn/sql/sql_join_full.asp)
* [Next Page](http://www.w3school.com.cn/sql/sql_select_into.asp)

**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 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，每条 SELECT 语句中的列的顺序必须相同。

**SQL UNION 语法**

SELECT column\_name(s) FROM table\_name1

UNION

SELECT column\_name(s) FROM table\_name2

**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 table\_name1

UNION ALL

SELECT column\_name(s) FROM table\_name2

另外，UNION 结果集中的列名总是等于 UNION 中第一个 SELECT 语句中的列名。

**下面的例子中使用的原始表：**

**Employees\_China:**

|  |  |
| --- | --- |
| **E\_ID** | **E\_Name** |
| 01 | Zhang, Hua |
| 02 | Wang, Wei |
| 03 | Carter, Thomas |
| 04 | Yang, Ming |

**Employees\_USA:**

|  |  |
| --- | --- |
| **E\_ID** | **E\_Name** |
| 01 | Adams, John |
| 02 | Bush, George |
| 03 | Carter, Thomas |
| 04 | Gates, Bill |

**使用 UNION 命令**

**实例**

列出所有在中国和美国的不同的雇员名：

SELECT E\_Name FROM Employees\_China

UNION

SELECT E\_Name FROM Employees\_USA

**结果**

|  |
| --- |
| **E\_Name** |
| Zhang, Hua |
| Wang, Wei |
| Carter, Thomas |
| Yang, Ming |
| Adams, John |
| Bush, George |
| Gates, Bill |

**注释：**这个命令无法列出在中国和美国的所有雇员。在上面的例子中，我们有两个名字相同的雇员，他们当中只有一个人被列出来了。UNION 命令只会选取不同的值。

**UNION ALL**

UNION ALL 命令和 UNION 命令几乎是等效的，不过 UNION ALL 命令会列出所有的值。

SQL Statement 1

UNION ALL

SQL Statement 2

**使用 UNION ALL 命令**

**实例：**

列出在中国和美国的所有的雇员：

SELECT E\_Name FROM Employees\_China

UNION ALL

SELECT E\_Name FROM Employees\_USA

**结果**

|  |
| --- |
| **E\_Name** |
| Zhang, Hua |
| Wang, Wei |
| Carter, Thomas |
| Yang, Ming |
| Adams, John |
| Bush, George |
| Carter, Thomas |
| Gates, Bill |

# SELECT INTO 语句

* [Previous Page](http://www.w3school.com.cn/sql/sql_union.asp)
* [Next Page](http://www.w3school.com.cn/sql/sql_create_db.asp)

**SQL SELECT INTO 语句可用于创建表的备份复件。**

**SELECT INTO 语句**

SELECT INTO 语句从一个表中选取数据，然后把数据插入另一个表中。

SELECT INTO 语句常用于创建表的备份复件或者用于对记录进行存档。

**SQL SELECT INTO 语法**

您可以把所有的列插入新表：

SELECT \*

INTO new\_table\_name [IN externaldatabase]

FROM old\_tablename

或者只把希望的列插入新表：

SELECT column\_name(s)

INTO new\_table\_name [IN externaldatabase]

FROM old\_tablename

**SQL SELECT INTO 实例 - 制作备份复件**

下面的例子会制作 "Persons" 表的备份复件：

SELECT \*

INTO Persons\_backup

FROM Persons

IN 子句可用于向另一个数据库中拷贝表：

SELECT \*

INTO Persons IN 'Backup.mdb'

FROM Persons

如果我们希望拷贝某些域，可以在 SELECT 语句后列出这些域：

SELECT LastName,FirstName

INTO Persons\_backup

FROM Persons

**SQL SELECT INTO 实例 - 带有 WHERE 子句**

我们也可以添加 WHERE 子句。

下面的例子通过从 "Persons" 表中提取居住在 "Beijing" 的人的信息，创建了一个带有两个列的名为 "Persons\_backup" 的表：

SELECT LastName,Firstname

INTO Persons\_backup

FROM Persons

WHERE City='Beijing'

**SQL SELECT INTO 实例 - 被连接的表**

从一个以上的表中选取数据也是可以做到的。

下面的例子会创建一个名为 "Persons\_Order\_Backup" 的新表，其中包含了从 Persons 和 Orders 两个表中取得的信息：

SELECT Persons.LastName,Orders.OrderNo

INTO Persons\_Order\_Backup

FROM Persons

INNER JOIN Orders

ON Persons.Id\_P=Orders.Id\_P

# CREATE DATABASE 语句

* [Previous Page](http://www.w3school.com.cn/sql/sql_select_into.asp)
* [Next Page](http://www.w3school.com.cn/sql/sql_create_table.asp)

**CREATE DATABASE 语句**

CREATE DATABASE 用于创建数据库。

**SQL CREATE DATABASE 语法**

CREATE DATABASE database\_name

**SQL CREATE DATABASE 实例**

现在我们希望创建一个名为 "my\_db" 的数据库。

我们使用下面的 CREATE DATABASE 语句：

CREATE DATABASE my\_db

可以通过 CREATE TABLE 来添加数据库表。

# CREATE TABLE 语句

**CREATE TABLE 语句**

CREATE TABLE 语句用于创建数据库中的表。

**SQL CREATE TABLE 语法**

CREATE TABLE 表名称

(

列名称1 数据类型,

列名称2 数据类型,

列名称3 数据类型,

....

)

数据类型（data\_type）规定了列可容纳何种数据类型。下面的表格包含了SQL中最常用的数据类型：

|  |  |
| --- | --- |
| **数据类型** | **描述** |
| * integer(size) * int(size) * smallint(size) * tinyint(size) | 仅容纳整数。在括号内规定数字的最大位数。 |
| * decimal(size,d) * numeric(size,d) | 容纳带有小数的数字。  "size" 规定数字的最大位数。"d" 规定小数点右侧的最大位数。 |
| char(size) | 容纳固定长度的字符串（可容纳字母、数字以及特殊字符）。  在括号中规定字符串的长度。 |
| varchar(size) | 容纳可变长度的字符串（可容纳字母、数字以及特殊的字符）。  在括号中规定字符串的最大长度。 |
| date(yyyymmdd) | 容纳日期。 |

**SQL CREATE TABLE 实例**

本例演示如何创建名为 "Person" 的表。

该表包含 5 个列，列名分别是："Id\_P"、"LastName"、"FirstName"、"Address" 以及 "City"：

CREATE TABLE Persons

(

Id\_P int,

LastName varchar(255),

FirstName varchar(255),

Address varchar(255),

City varchar(255)

)

Id\_P 列的数据类型是 int，包含整数。其余 4 列的数据类型是 varchar，最大长度为 255 个字符。

空的 "Persons" 表类似这样：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id\_P** | **LastName** | **FirstName** | **Address** | **City** |
|  |  |  |  |  |

可使用 INSERT INTO 语句向空表写入数据。

# 约束 (Constraints)

**SQL 约束**

约束用于限制加入表的数据的类型。

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

我们将主要探讨以下几种约束：

* NOT NULL
* UNIQUE
* PRIMARY KEY
* FOREIGN KEY
* CHECK
* DEFAULT

**注释：**在下面的章节，我们会详细讲解每一种约束。

# NOT NULL 约束

**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，就无法插入新纪录或者更新记录。

下面的 SQL 语句强制 "Id\_P" 列和 "LastName" 列不接受 NULL 值：

CREATE TABLE Persons

(

Id\_P int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255)

)

# 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 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。

**SQL UNIQUE Constraint on CREATE TABLE**

下面的 SQL 在 "Persons" 表创建时在 "Id\_P" 列创建 UNIQUE 约束：

**MySQL:**

CREATE TABLE Persons

(

Id\_P int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255),

UNIQUE (Id\_P)

)

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

CREATE TABLE Persons

(

Id\_P int NOT NULL UNIQUE,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255)

)

如果需要命名 UNIQUE 约束，以及为多个列定义 UNIQUE 约束，请使用下面的 SQL 语法：

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

CREATE TABLE Persons

(

Id\_P int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255),

CONSTRAINT uc\_PersonID UNIQUE (Id\_P,LastName)

)

**SQL UNIQUE Constraint on ALTER TABLE**

当表已被创建时，如需在 "P\_Id" 列创建 UNIQUE 约束，请使用下列 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

# PRIMARY KEY 约束

* [Previous Page](http://www.w3school.com.cn/sql/sql_unique.asp)
* [Next Page](http://www.w3school.com.cn/sql/sql_foreignkey.asp)

**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。

每个表应该都一个主键，并且每个表只能有一个主键。

**SQL PRIMARY KEY Constraint on CREATE TABLE**

下面的 SQL 在 "Persons" 表创建时在 "Id\_P" 列创建 PRIMARY KEY 约束：

**MySQL:**

CREATE TABLE Persons

(

Id\_P int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255),

PRIMARY KEY (Id\_P)

)

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

CREATE TABLE Persons

(

Id\_P int NOT NULL PRIMARY KEY,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255)

)

如果需要命名 PRIMARY KEY 约束，以及为多个列定义 PRIMARY KEY 约束，请使用下面的 SQL 语法：

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

CREATE TABLE Persons

(

Id\_P int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255),

CONSTRAINT uc\_PersonID PRIMARY KEY (Id\_P,LastName)

)

**SQL PRIMARY KEY Constraint on ALTER TABLE**

如果在表已存在的情况下为 "Id\_P" 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 (Id\_P)

如果需要命名 PRIMARY KEY 约束，以及为多个列定义 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 (Id\_P,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

# FOREIGN KEY 约束

* [Previous Page](http://www.w3school.com.cn/sql/sql_primarykey.asp)
* [Next Page](http://www.w3school.com.cn/sql/sql_check.asp)

**SQL FOREIGN KEY 约束**

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

让我们通过一个例子来解释外键。请看下面两个表：

"Persons" 表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id\_P** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Adams | John | Oxford Street | London |
| 2 | Bush | George | Fifth Avenue | New York |
| 3 | Carter | Thomas | Changan Street | Beijing |

"Orders" 表：

|  |  |  |
| --- | --- | --- |
| **Id\_O** | **OrderNo** | **Id\_P** |
| 1 | 77895 | 3 |
| 2 | 44678 | 3 |
| 3 | 22456 | 1 |
| 4 | 24562 | 1 |

请注意，"Orders" 中的 "Id\_P" 列指向 "Persons" 表中的 "Id\_P" 列。

"Persons" 表中的 "Id\_P" 列是 "Persons" 表中的 PRIMARY KEY。

"Orders" 表中的 "Id\_P" 列是 "Orders" 表中的 FOREIGN KEY。

FOREIGN KEY 约束用于预防破坏表之间连接的动作。

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。

**SQL FOREIGN KEY Constraint on CREATE TABLE**

下面的 SQL 在 "Orders" 表创建时为 "Id\_P" 列创建 FOREIGN KEY：

**MySQL:**

CREATE TABLE Orders

(

O\_Id int NOT NULL,

OrderNo int NOT NULL,

Id\_P int,

PRIMARY KEY (O\_Id),

FOREIGN KEY (Id\_P) REFERENCES Persons(Id\_P)

)

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

CREATE TABLE Orders

(

O\_Id int NOT NULL PRIMARY KEY,

OrderNo int NOT NULL,

Id\_P int FOREIGN KEY REFERENCES Persons(Id\_P)

)

如果需要命名 FOREIGN KEY 约束，以及为多个列定义 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,

Id\_P int,

PRIMARY KEY (O\_Id),

CONSTRAINT fk\_PerOrders FOREIGN KEY (Id\_P)

REFERENCES Persons(Id\_P)

)

**SQL FOREIGN KEY Constraint on ALTER TABLE**

如果在 "Orders" 表已存在的情况下为 "Id\_P" 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 (Id\_P)

REFERENCES Persons(Id\_P)

如果需要命名 FOREIGN KEY 约束，以及为多个列定义 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 (Id\_P)

REFERENCES Persons(Id\_P)

**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

# CHECK 约束

* [Previous Page](http://www.w3school.com.cn/sql/sql_foreignkey.asp)
* [Next Page](http://www.w3school.com.cn/sql/sql_default.asp)

**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，那么此约束会在特定的列中对值进行限制。

**SQL CHECK Constraint on CREATE TABLE**

下面的 SQL 在 "Persons" 表创建时为 "Id\_P" 列创建 CHECK 约束。CHECK 约束规定 "Id\_P" 列必须只包含大于 0 的整数。

**My SQL:**

CREATE TABLE Persons

(

Id\_P int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255),

CHECK (Id\_P>0)

)

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

CREATE TABLE Persons

(

Id\_P int NOT NULL CHECK (Id\_P>0),

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255)

)

如果需要命名 CHECK 约束，以及为多个列定义 CHECK 约束，请使用下面的 SQL 语法：

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

CREATE TABLE Persons

(

Id\_P int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255),

CONSTRAINT chk\_Person CHECK (Id\_P>0 AND City='Sandnes')

)

**SQL CHECK Constraint on ALTER TABLE**

如果在表已存在的情况下为 "Id\_P" 列创建 CHECK 约束，请使用下面的 SQL：

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

ALTER TABLE Persons

ADD CHECK (Id\_P>0)

如果需要命名 CHECK 约束，以及为多个列定义 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 (Id\_P>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

# 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纪录。

**SQL DEFAULT Constraint on CREATE TABLE**

下面的 SQL 在 "Persons" 表创建时为 "City" 列创建 DEFAULT 约束：

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

CREATE TABLE Persons

(

Id\_P 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

(

Id\_O int NOT NULL,

OrderNo int NOT NULL,

Id\_P int,

OrderDate date DEFAULT GETDATE()

)

**SQL DEFAULT Constraint on ALTER TABLE**

如果在表已存在的情况下为 "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 / Oracle / MS Access:**

ALTER TABLE Persons

ALTER COLUMN City SET 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

# CREATE INDEX 语句

* [Previous Page](http://www.w3school.com.cn/sql/sql_default.asp)
* [Next Page](http://www.w3school.com.cn/sql/sql_drop.asp)

**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)

**注释：**"column\_name" 规定需要索引的列。

**SQL CREATE UNIQUE INDEX 语法**

在表上创建一个唯一的索引。唯一的索引意味着两个行不能拥有相同的索引值。

CREATE UNIQUE INDEX index\_name

ON table\_name (column\_name)

**CREATE INDEX 实例**

本例会创建一个简单的索引，名为 "PersonIndex"，在 Person 表的 LastName 列：

CREATE INDEX PersonIndex

ON Person (LastName)

如果您希望以**降序**索引某个列中的值，您可以在列名称之后添加保留字 **DESC**：

CREATE INDEX PersonIndex

ON Person (LastName DESC)

假如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 PersonIndex

ON Person (LastName, FirstName)

# 撤销索引、表以及数据库

* [Previous Page](http://www.w3school.com.cn/sql/sql_create_index.asp)
* [Next Page](http://www.w3school.com.cn/sql/sql_alter.asp)

**通过使用 DROP 语句，可以轻松地删除索引、表和数据库。**

**SQL DROP INDEX 语句**

我们可以使用 DROP INDEX 命令删除表格中的索引。

**用于 Microsoft SQLJet (以及 Microsoft Access) 的语法:**

DROP INDEX index\_name ON table\_name

**用于 MS SQL Server 的语法:**

DROP INDEX table\_name.index\_name

**用于 IBM DB2 和 Oracle 语法:**

DROP INDEX index\_name

**用于 MySQL 的语法:**

ALTER TABLE table\_name DROP INDEX index\_name

**SQL DROP TABLE 语句**

DROP TABLE 语句用于删除表（表的结构、属性以及索引也会被删除）：

DROP TABLE 表名称

**SQL DROP DATABASE 语句**

DROP DATABASE 语句用于删除数据库：

DROP DATABASE 数据库名称

**SQL TRUNCATE TABLE 语句**

如果我们仅仅需要除去表内的数据，但并不删除表本身，那么我们该如何做呢？

请使用 TRUNCATE TABLE 命令（仅仅删除表格中的数据）：

TRUNCATE TABLE 表名称

# ALTER TABLE 语句

**ALTER TABLE 语句**

ALTER TABLE 语句用于在已有的表中添加、修改或删除列。

**SQL ALTER TABLE 语法**

如需在表中添加列，请使用下列语法:

ALTER TABLE table\_name

ADD column\_name datatype

要删除表中的列，请使用下列语法：

ALTER TABLE table\_name

DROP COLUMN column\_name

**注释：**某些数据库系统不允许这种在数据库表中删除列的方式 (DROP COLUMN column\_name)。

要改变表中列的数据类型，请使用下列语法：

ALTER TABLE table\_name

ALTER COLUMN column\_name datatype

**原始的表 (用在例子中的)：**

Persons 表:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Adams | John | Oxford Street | London |
| 2 | Bush | George | Fifth Avenue | New York |
| 3 | Carter | Thomas | Changan Street | Beijing |

**SQL ALTER TABLE 实例**

现在，我们希望在表 "Persons" 中添加一个名为 "Birthday" 的新列。

我们使用下列 SQL 语句：

ALTER TABLE Persons

ADD Birthday date

请注意，新列 "Birthday" 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。

新的 "Persons" 表类似这样：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Id** | **LastName** | **FirstName** | **Address** | **City** | **Birthday** |
| 1 | Adams | John | Oxford Street | London |  |
| 2 | Bush | George | Fifth Avenue | New York |  |
| 3 | Carter | Thomas | Changan Street | Beijing |  |

**改变数据类型实例**

现在我们希望改变 "Persons" 表中 "Birthday" 列的数据类型。

我们使用下列 SQL 语句：

ALTER TABLE Persons

ALTER COLUMN Birthday year

请注意，"Birthday" 列的数据类型是 year，可以存放 2 位或 4 位格式的年份。

**DROP COLUMN 实例**

接下来，我们删除 "Person" 表中的 "Birthday" 列：

ALTER TABLE Person

DROP COLUMN Birthday

Persons 表会成为这样:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Adams | John | Oxford Street | London |
| 2 | Bush | George | Fifth Avenue | New York |
| 3 | Carter | Thomas | Changan Street | Beijing |