**SQL教程**

**SQL(Structured Query Language**结构化查询语言**)是用于访问和处理数据库的标准的计算机语言。**

**在本教程中，您将学到如何使用 SQL 访问和处理数据系统中的数据，这类数据库包括：Oracle, Sybase, SQL Server, DB2, Access 等等。**

**基础教程**

**SQL 简介**

**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...**

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

**注释：**CSS(Cascading Style Sheets) 层叠样式表

**RDBMS**

RDBMS (Relational Database Management System)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**

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。

**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 |

**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。

**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**

**WHERE 子句用于规定选择的标准。**

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**

**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 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**

**ORDER BY 语句用于对结果集进行排序。**

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**

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**

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**

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

**高级教程**

**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。

**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**

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**

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）**

**通过使用 SQL，可以为列名称和表名称指定别名（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**

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**

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**

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**

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**

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

**SELECT INTO 语句**

**SELECT INTO 语句可用于创建表的备份复件。**

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**

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。

**Constraints (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。

**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 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**

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 "Id\_P" 列创建 UNIQUE 约束，请使用下列 SQL：

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

ALTER TABLE Persons

ADD UNIQUE (Id\_P)

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 (Id\_P,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**

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。

**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 pk\_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**

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

(

Id\_O int NOT NULL,

OrderNo int NOT NULL,

Id\_P int,

PRIMARY KEY (Id\_O),

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

)

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

CREATE TABLE Orders

(

Id\_O 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

(

Id\_O int NOT NULL,

OrderNo int NOT NULL,

Id\_P int,

PRIMARY KEY (Id\_O),

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**

**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

**MySQL:**

ALTER TABLE Persons

DROP CHECK chk\_Person

**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。

**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

**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)

## DROP(撤消)

**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 |

## 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。**

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**

下列 SQL 语句把 "Persons" 表中的 "P\_Id" 列定义为 auto-increment 主键：

CREATE TABLE Persons

(

P\_Id int NOT NULL AUTO\_INCREMENT,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255),

PRIMARY KEY (P\_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，请使用下列 SQL 语法：

ALTER TABLE Persons AUTO\_INCREMENT=100

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

INSERT INTO Persons (FirstName,LastName)

VALUES ('Bill','Gates')

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

**用于 SQL Server 的语法**

下列 SQL 语句把 "Persons" 表中的 "P\_Id" 列定义为 auto-increment 主键：

CREATE TABLE Persons

(

P\_Id int PRIMARY KEY IDENTITY,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255)

)

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

默认地，IDENTITY 的开始值是 1，每条新记录递增 1。

要规定 "P\_Id" 列以 20 起始且递增 10，请把 identity 改为 IDENTITY(20,10)

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

INSERT INTO Persons (FirstName,LastName)

VALUES ('Bill','Gates')

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

**用于 Access 的语法**

下列 SQL 语句把 "Persons" 表中的 "P\_Id" 列定义为 auto-increment 主键：

CREATE TABLE Persons

(

P\_Id int 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。

要规定 "P\_Id" 列以 20 起始且递增 10，请把 autoincrement 改为 AUTOINCREMENT(20,10)

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

INSERT INTO Persons (FirstName,LastName)

VALUES ('Bill','Gates')

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

**用于 Oracle 的语法**

在 Oracle 中，代码稍微复杂一点。

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

请使用下面的 CREATE SEQUENCE 语法：

CREATE SEQUENCE seq\_person

MINVALUE 1

START WITH 1

INCREMENT BY 1

CACHE 10

上面的代码创建名为 seq\_person 的序列对象，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 (P\_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。"P\_Id" 的赋值是来自 seq\_person 序列的下一个数字。"FirstName" 会被设置为 "Bill"，"LastName" 列会被设置为 "Gates"。

## VIEW（视图）

**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**

**什么是视图？**

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。我们可以向视图添加 SQL 函数、WHERE 以及 JOIN 语句，我们也可以提交数据，就像这些来自于某个单一的表。

**注释：**数据库的设计和结构不会受到视图中的函数、where 或 join 语句的影响。

**SQL CREATE VIEW 语法**

CREATE VIEW view\_name AS

SELECT column\_name(s)

FROM table\_name

WHERE condition

**注释：**视图总是显示最近的数据。每当用户查询视图时，数据库引擎通过使用 SQL 语句来重建数据。

**SQL CREATE VIEW 实例**

可以从某个查询内部、某个存储过程内部，或者从另一个视图内部来使用视图。通过向视图添加函数、join 等等，我们可以向用户精确地提交我们希望提交的数据。

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。这个视图使用下列 SQL 创建：

CREATE VIEW [Current Product List] AS

SELECT ProductID,ProductName

FROM Products

WHERE Discontinued=No

我们可以查询上面这个视图：

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]

另一个来自 Northwind 数据库的视图实例会计算在 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" 类的全部销量：

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 Syntax

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 Syntax

DROP VIEW view\_name

## Date 函数

**SQL 日期**

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，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()](https://www.w3school.com.cn/sql/func_now.asp) | 返回当前的日期和时间 |
| [CURDATE()](https://www.w3school.com.cn/sql/func_curdate.asp) | 返回当前的日期 |
| [CURTIME()](https://www.w3school.com.cn/sql/func_curtime.asp) | 返回当前的时间 |
| [DATE()](https://www.w3school.com.cn/sql/func_date.asp) | 提取日期或日期/时间表达式的日期部分 |
| [EXTRACT()](https://www.w3school.com.cn/sql/func_extract.asp) | 返回日期/时间的单独部分 |
| [DATE\_ADD()](https://www.w3school.com.cn/sql/func_date_add.asp) | 给日期添加指定的时间间隔 |
| [DATE\_SUB()](https://www.w3school.com.cn/sql/func_date_sub.asp) | 从日期减去指定的时间间隔 |
| [DATEDIFF()](https://www.w3school.com.cn/sql/func_datediff_mysql.asp) | 返回两个日期之间的天数 |
| [DATE\_FORMAT()](https://www.w3school.com.cn/sql/func_date_format.asp) | 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()](https://www.w3school.com.cn/sql/func_getdate.asp) | 返回当前日期和时间 |
| [DATEPART()](https://www.w3school.com.cn/sql/func_datepart.asp) | 返回日期/时间的单独部分 |
| [DATEADD()](https://www.w3school.com.cn/sql/func_dateadd.asp) | 在日期中添加或减去指定的时间间隔 |
| [DATEDIFF()](https://www.w3school.com.cn/sql/func_datediff.asp) | 返回两个日期之间的时间 |
| [CONVERT()](https://www.w3school.com.cn/sql/func_convert.asp) | 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

**SQL 日期处理**

如果不涉及时间部分，那么我们可以轻松地比较两个日期！

假设我们有下面这个 "Orders" 表：

|  |  |  |
| --- | --- | --- |
| **OrderId** | **ProductName** | **OrderDate** |
| 1 | computer | 2008-12-26 |
| 2 | printer | 2008-12-26 |
| 3 | electrograph | 2008-11-12 |
| 4 | telephone | 2008-10-19 |

现在，我们希望从上表中选取 OrderDate 为 "2008-12-26" 的记录。

我们使用如下 SELECT 语句：

SELECT \* FROM Orders WHERE OrderDate='2008-12-26'

结果集：

|  |  |  |
| --- | --- | --- |
| **OrderId** | **ProductName** | **OrderDate** |
| 1 | computer | 2008-12-26 |
| 3 | electrograph | 2008-12-26 |

现在假设 "Orders" 类似这样（请注意 "OrderDate" 列中的时间部分）：

|  |  |  |
| --- | --- | --- |
| **OrderId** | **ProductName** | **OrderDate** |
| 1 | computer | 2008-12-26 16:23:55 |
| 2 | printer | 2008-12-26 10:45:26 |
| 3 | electrograph | 2008-11-12 14:12:08 |
| 4 | telephone | 2008-10-19 12:56:10 |

如果我们使用上面的 SELECT 语句：

SELECT \* FROM Orders WHERE OrderDate='2008-12-26'

那么我们得不到结果。这是由于该查询不含有时间部分的日期。

**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！

## NULL 值

**NULL 值是遗漏的未知数据。**

**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。

**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：

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

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 =, <, 或者 <>。

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：

|  |  |  |
| --- | --- | --- |
| **LastName** | **FirstName** | **Address** |
| Adams | John |  |
| Carter | Thomas |  |

**提示：**请始终使用 IS NULL 来查找 NULL 值。

**SQL IS NOT NULL**

我们如何选取在 "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：

|  |  |  |
| --- | --- | --- |
| **LastName** | **FirstName** | **Address** |
| Bush | George | Fifth Avenue |

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

## 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 | computer | 699 | 25 | 15 |
| 2 | printer | 365 | 36 |  |
| 3 | telephone | 280 | 159 | 57 |

假如 "UnitsOnOrder" 是可选的，而且可以包含 NULL 值。

我们使用如下 SELECT 语句：

SELECT ProductName,UnitPrice\*(UnitsInStock+UnitsOnOrder)

FROM Products

在上面的例子中，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，则不利于计算，因此如果值是 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，就像这样：

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

FROM Products

或者我们可以使用 COALESCE() 函数，就像这样：

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

FROM Products

## SQL 数据类型

**Microsoft Access、MySQL 以及 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 到 32,767 之间的数字。 | 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 | 允许你创建一个可从下列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：文本、数字和日期/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.) | 允许你输入可能值的列表。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：这些值是按照你输入的顺序存储的。  可以按照此格式输入可能的值：ENUM('X','Y','Z') |
| SET | 与 ENUM 类似，SET 最多只能包含 64 个列表项，不过 SET 可存储一个以上的值。 |

**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。在括号中规定最大位数。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。在括号中规定最大位数。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。 |

\* 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) 至今的描述来存储。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**

**Character 字符串：**

|  |  |  |
| --- | --- | --- |
| **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。 | n |
| varchar(n) | 可变长度的字符串。最多 8,000 个字符。 |  |
| varchar(max) | 可变长度的字符串。最多 1,073,741,824 个字符。 |  |
| text | 可变长度的字符串。最多 2GB 字符数据。 |  |

**Unicode 字符串：**

|  |  |  |
| --- | --- | --- |
| **数据类型** | **描述** | **存储** |
| nchar(n) | 固定长度的 Unicode 数据。最多 4,000 个字符。 |  |
| nvarchar(n) | 可变长度的 Unicode 数据。最多 4,000 个字符。 |  |
| nvarchar(max) | 可变长度的 Unicode 数据。最多 536,870,912 个字符。 |  |
| ntext | 可变长度的 Unicode 数据。最多 2GB 字符数据。 |  |

**Binary 类型：**

|  |  |  |
| --- | --- | --- |
| **数据类型** | **描述** | **存储** |
| bit | 允许 0、1 或 NULL |  |
| binary(n) | 固定长度的二进制数据。最多 8,000 字节。 |  |
| varbinary(n) | 可变长度的二进制数据。最多 8,000 字节。 |  |
| varbinary(max) | 可变长度的二进制数据。最多 2GB 字节。 |  |
| image | 可变长度的二进制数据。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 | 允许从 -32,768 到 32,767 的所有数字。 | 2 字节 |
| int | 允许从 -2,147,483,648 到 2,147,483,647 的所有数字。 | 4 字节 |
| bigint | 允许介于 -9,223,372,036,854,775,808 和 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 和 214,748.3647 之间的货币数据。 | 4 字节 |
| money | 介于 -922,337,203,685,477.5808 和 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。 参数 n 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 bytes |
| datetime2 | 从 1753 年 1 月 1 日 到 9999 年 12 月 31 日，精度为 100 纳秒。 | 6-8 bytes |
| smalldatetime | 从 1900 年 1 月 1 日 到 2079 年 6 月 6 日，精度为 1 分钟。 | 4 bytes |
| 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 bytes |
| datetimeoffset | 与 datetime2 相同，外加时区偏移。 | 8-10 bytes |
| 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 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 服务器 – RDBMS

**现代的 SQL 服务器构建在 RDBMS 之上。**

## DBMS - 数据库管理系统（Database Management System）

数据库管理系统是一种可以访问数据库中数据的计算机程序。

DBMS 使我们有能力在数据库中提取、修改或者存贮信息。

不同的 DBMS 提供不同的函数供查询、提交以及修改数据。

## RDBMS - 关系数据库管理系统（Relational Database Management System）

关系数据库管理系统 (RDBMS) 也是一种数据库管理系统，其数据库是根据数据间的关系来组织和访问数据的。

20 世纪 70 年代初，IBM 公司发明了 RDBMS。

RDBMS 是 SQL 的基础，也是所有现代数据库系统诸如 Oracle、SQL Server、IBM DB2、Sybase、MySQL 以及 Microsoft Access 的基础。

**函数**

**SQL 拥有很多可用于计数和计算的内建函数。**

**函数的语法**

内建 SQL 函数的语法是：

SELECT function(列) FROM 表

**函数的类型**

在 SQL 中，基本的函数类型和种类有若干种。函数的基本类型是：

* Aggregate 函数
* Scalar 函数

**合计函数（Aggregate functions）**

Aggregate 函数的操作面向一系列的值，并返回一个单一的值。

**注释：**如果在 SELECT 语句的项目列表中的众多其它表达式中使用 SELECT 语句，则这个 SELECT 必须使用 GROUP BY 语句！

**"Persons" table (在大部分的例子中使用过)**

|  |  |
| --- | --- |
| **Name** | **Age** |
| Adams, John | 38 |
| Bush, George | 33 |
| Carter, Thomas | 28 |

**MS Access 中的合计函数**

|  |  |
| --- | --- |
| **函数** | **描述** |
| [AVG(column)](https://www.w3school.com.cn/sql/sql_func_avg.asp) | 返回某列的平均值 |
| [COUNT(column)](https://www.w3school.com.cn/sql/sql_func_count.asp) | 返回某列的行数（不包括 NULL 值） |
| [COUNT(\*)](https://www.w3school.com.cn/sql/sql_func_count_ast.asp) | 返回被选行数 |
| FIRST(column) | 返回在指定的域中第一个记录的值 |
| LAST(column) | 返回在指定的域中最后一个记录的值 |
| [MAX(column)](https://www.w3school.com.cn/sql/sql_func_max.asp) | 返回某列的最高值 |
| [MIN(column)](https://www.w3school.com.cn/sql/sql_func_min.asp) | 返回某列的最低值 |
| STDEV(column) |  |
| STDEVP(column) |  |
| [SUM(column)](https://www.w3school.com.cn/sql/sql_func_sum.asp) | 返回某列的总和 |
| VAR(column) |  |
| VARP(column) |  |

**在 SQL Server 中的合计函数**

|  |  |
| --- | --- |
| **函数** | **描述** |
| [AVG(column)](https://www.w3school.com.cn/sql/sql_func_avg.asp) | 返回某列的平均值 |
| BINARY\_CHECKSUM |  |
| CHECKSUM |  |
| CHECKSUM\_AGG |  |
| [COUNT(column)](https://www.w3school.com.cn/sql/sql_func_count.asp) | 返回某列的行数（不包括NULL值） |
| [COUNT(\*)](https://www.w3school.com.cn/sql/sql_func_count_ast.asp) | 返回被选行数 |
| [COUNT(DISTINCT column)](https://www.w3school.com.cn/sql/sql_func_count_distinct.asp) | 返回相异结果的数目 |
| [FIRST(column)](https://www.w3school.com.cn/sql/sql_func_first.asp) | 返回在指定的域中第一个记录的值（SQLServer2000 不支持） |
| [LAST(column)](https://www.w3school.com.cn/sql/sql_func_last.asp) | 返回在指定的域中最后一个记录的值（SQLServer2000 不支持） |
| [MAX(column)](https://www.w3school.com.cn/sql/sql_func_max.asp) | 返回某列的最高值 |
| [MIN(column)](https://www.w3school.com.cn/sql/sql_func_min.asp) | 返回某列的最低值 |
| STDEV(column) |  |
| STDEVP(column) |  |
| [SUM(column)](https://www.w3school.com.cn/sql/sql_func_sum.asp) | 返回某列的总和 |
| VAR(column) |  |
| VARP(column) |  |

**Scalar 函数**

Scalar 函数的操作面向某个单一的值，并返回基于输入值的一个单一的值。

**MS Access 中的 Scalar 函数**

|  |  |
| --- | --- |
| **函数** | **描述** |
| UCASE(c) | 将某个域转换为大写 |
| LCASE(c) | 将某个域转换为小写 |
| MID(c,start[,end]) | 从某个文本域提取字符 |
| LEN(c) | 返回某个文本域的长度 |
| INSTR(c,char) | 返回在某个文本域中指定字符的数值位置 |
| LEFT(c,number\_of\_char) | 返回某个被请求的文本域的左侧部分 |
| RIGHT(c,number\_of\_char) | 返回某个被请求的文本域的右侧部分 |
| ROUND(c,decimals) | 对某个数值域进行指定小数位数的四舍五入 |
| MOD(x,y) | 返回除法操作的余数 |
| NOW() | 返回当前的系统日期 |
| FORMAT(c,format) | 改变某个域的显示方式 |
| DATEDIFF(d,date1,date2) | 用于执行日期计算 |

## AVG 函数

**定义和用法**

AVG 函数返回数值列的平均值。NULL 值不包括在计算中。

**SQL AVG() 语法**

SELECT AVG(column\_name) FROM table\_name

**SQL AVG() 实例**

我们拥有下面这个 "Orders" 表：

|  |  |  |  |
| --- | --- | --- | --- |
| **O\_Id** | **OrderDate** | **OrderPrice** | **Customer** |
| 1 | 2008/12/29 | 1000 | Bush |
| 2 | 2008/11/23 | 1600 | Carter |
| 3 | 2008/10/05 | 700 | Bush |
| 4 | 2008/09/28 | 300 | Bush |
| 5 | 2008/08/06 | 2000 | Adams |
| 6 | 2008/07/21 | 100 | Carter |

**例子 1**

现在，我们希望计算 "OrderPrice" 字段的平均值。

我们使用如下 SQL 语句：

SELECT AVG(OrderPrice) AS OrderAverage FROM Orders

结果集类似这样：

|  |
| --- |
| **OrderAverage** |
| 950 |

**例子 2**

现在，我们希望找到 OrderPrice 值高于 OrderPrice 平均值的客户。

我们使用如下 SQL 语句：

SELECT Customer FROM Orders

WHERE OrderPrice>(SELECT AVG(OrderPrice) FROM Orders)

结果集类似这样：

|  |
| --- |
| **Customer** |
| Bush |
| Carter |
| Adams |

## COUNT() 函数

**COUNT() 函数返回匹配指定条件的行数。**

**SQL COUNT() 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。

**SQL COUNT(column\_name) 实例**

我们拥有下列 "Orders" 表：

|  |  |  |  |
| --- | --- | --- | --- |
| **O\_Id** | **OrderDate** | **OrderPrice** | **Customer** |
| 1 | 2008/12/29 | 1000 | Bush |
| 2 | 2008/11/23 | 1600 | Carter |
| 3 | 2008/10/05 | 700 | Bush |
| 4 | 2008/09/28 | 300 | Bush |
| 5 | 2008/08/06 | 2000 | Adams |
| 6 | 2008/07/21 | 100 | Carter |

现在，我们希望计算客户 "Carter" 的订单数。

我们使用如下 SQL 语句：

SELECT COUNT(Customer) AS CustomerNilsen FROM Orders

WHERE Customer='Carter'

以上 SQL 语句的结果是 2，因为客户 Carter 共有 2 个订单：

|  |
| --- |
| **CustomerNilsen** |
| 2 |

SQL COUNT(\*) 实例

如果我们省略 WHERE 子句，比如这样：

SELECT COUNT(\*) AS NumberOfOrders FROM Orders

结果集类似这样：

|  |
| --- |
| **NumberOfOrders** |
| 6 |

这是表中的总行数。

**SQL COUNT(DISTINCT column\_name) 实例**

现在，我们希望计算 "Orders" 表中不同客户的数目。

我们使用如下 SQL 语句：

SELECT COUNT(DISTINCT Customer) AS NumberOfCustomers FROM Orders

结果集类似这样：

|  |
| --- |
| **NumberOfCustomers** |
| 3 |

这是 "Orders" 表中不同客户（Bush, Carter 和 Adams）的数目。

## FIRST() 函数

FIRST() 函数返回指定的字段中第一个记录的值。

**提示：**可使用 ORDER BY 语句对记录进行排序。

**SQL FIRST() 语法**

SELECT FIRST(column\_name) FROM table\_name

**SQL FIRST() 实例**

我们拥有下面这个 "Orders" 表：

|  |  |  |  |
| --- | --- | --- | --- |
| **O\_Id** | **OrderDate** | **OrderPrice** | **Customer** |
| 1 | 2008/12/29 | 1000 | Bush |
| 2 | 2008/11/23 | 1600 | Carter |
| 3 | 2008/10/05 | 700 | Bush |
| 4 | 2008/09/28 | 300 | Bush |
| 5 | 2008/08/06 | 2000 | Adams |
| 6 | 2008/07/21 | 100 | Carter |

现在，我们希望查找 "OrderPrice" 列的第一个值。

我们使用如下 SQL 语句：

SELECT FIRST(OrderPrice) AS FirstOrderPrice FROM Orders

结果集类似这样：

|  |
| --- |
| **FirstOrderPrice** |
| 1000 |

## LAST() 函数

LAST() 函数返回指定的字段中最后一个记录的值。

**提示：**可使用 ORDER BY 语句对记录进行排序。

**SQL LAST() 语法**

SELECT LAST(column\_name) FROM table\_name

**SQL LAST() 实例**

我们拥有下面这个 "Orders" 表：

|  |  |  |  |
| --- | --- | --- | --- |
| **O\_Id** | **OrderDate** | **OrderPrice** | **Customer** |
| 1 | 2008/12/29 | 1000 | Bush |
| 2 | 2008/11/23 | 1600 | Carter |
| 3 | 2008/10/05 | 700 | Bush |
| 4 | 2008/09/28 | 300 | Bush |
| 5 | 2008/08/06 | 2000 | Adams |
| 6 | 2008/07/21 | 100 | Carter |

现在，我们希望查找 "OrderPrice" 列的最后一个值。

我们使用如下 SQL 语句：

SELECT LAST(OrderPrice) AS LastOrderPrice FROM Orders

结果集类似这样：

|  |
| --- |
| **LastOrderPrice** |
| 100 |

## MAX() 函数

MAX 函数返回一列中的最大值。NULL 值不包括在计算中。

**SQL MAX() 语法**

SELECT MAX(column\_name) FROM table\_name

**注释：**MIN 和 MAX 也可用于文本列，以获得按字母顺序排列的最高或最低值。

**SQL MAX() 实例**

我们拥有下面这个 "Orders" 表：

|  |  |  |  |
| --- | --- | --- | --- |
| **O\_Id** | **OrderDate** | **OrderPrice** | **Customer** |
| 1 | 2008/12/29 | 1000 | Bush |
| 2 | 2008/11/23 | 1600 | Carter |
| 3 | 2008/10/05 | 700 | Bush |
| 4 | 2008/09/28 | 300 | Bush |
| 5 | 2008/08/06 | 2000 | Adams |
| 6 | 2008/07/21 | 100 | Carter |

现在，我们希望查找 "OrderPrice" 列的最大值。

我们使用如下 SQL 语句：

SELECT MAX(OrderPrice) AS LargestOrderPrice FROM Orders

结果集类似这样：

|  |
| --- |
| **LargestOrderPrice** |
| 2000 |

## MIN() 函数

MIN 函数返回一列中的最小值。NULL 值不包括在计算中。

**SQL MIN() 语法**

SELECT MIN(column\_name) FROM table\_name

**注释：**MIN 和 MAX 也可用于文本列，以获得按字母顺序排列的最高或最低值。

**SQL MIN() 实例**

我们拥有下面这个 "Orders" 表：

|  |  |  |  |
| --- | --- | --- | --- |
| **O\_Id** | **OrderDate** | **OrderPrice** | **Customer** |
| 1 | 2008/12/29 | 1000 | Bush |
| 2 | 2008/11/23 | 1600 | Carter |
| 3 | 2008/10/05 | 700 | Bush |
| 4 | 2008/09/28 | 300 | Bush |
| 5 | 2008/08/06 | 2000 | Adams |
| 6 | 2008/07/21 | 100 | Carter |

现在，我们希望查找 "OrderPrice" 列的最小值。

我们使用如下 SQL 语句：

SELECT MIN(OrderPrice) AS SmallestOrderPrice FROM Orders

结果集类似这样：

|  |
| --- |
| **SmallestOrderPrice** |
| 100 |

## SUM() 函数

SUM 函数返回数值列的总数（总额）。

**SQL SUM() 语法**

SELECT SUM(column\_name) FROM table\_name

**SQL SUM() 实例**

我们拥有下面这个 "Orders" 表：

|  |  |  |  |
| --- | --- | --- | --- |
| **O\_Id** | **OrderDate** | **OrderPrice** | **Customer** |
| 1 | 2008/12/29 | 1000 | Bush |
| 2 | 2008/11/23 | 1600 | Carter |
| 3 | 2008/10/05 | 700 | Bush |
| 4 | 2008/09/28 | 300 | Bush |
| 5 | 2008/08/06 | 2000 | Adams |
| 6 | 2008/07/21 | 100 | Carter |

现在，我们希望查找 "OrderPrice" 字段的总数。

我们使用如下 SQL 语句：

SELECT SUM(OrderPrice) AS OrderTotal FROM Orders

结果集类似这样：

|  |
| --- |
| **OrderTotal** |
| 5700 |

## GROUP BY 语句

**合计函数 (比如 SUM) 常常需要添加 GROUP BY 语句。**

## GROUP BY 语句

GROUP BY 语句用于结合合计函数，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

**SQL GROUP BY 实例**

我们拥有下面这个 "Orders" 表：

|  |  |  |  |
| --- | --- | --- | --- |
| **O\_Id** | **OrderDate** | **OrderPrice** | **Customer** |
| 1 | 2008/12/29 | 1000 | Bush |
| 2 | 2008/11/23 | 1600 | Carter |
| 3 | 2008/10/05 | 700 | Bush |
| 4 | 2008/09/28 | 300 | Bush |
| 5 | 2008/08/06 | 2000 | Adams |
| 6 | 2008/07/21 | 100 | Carter |

现在，我们希望查找每个客户的总金额（总订单）。

我们想要使用 GROUP BY 语句对客户进行组合。

我们使用下列 SQL 语句：

SELECT Customer,SUM(OrderPrice) FROM Orders

GROUP BY Customer

结果集类似这样：

|  |  |
| --- | --- |
| **Customer** | **SUM(OrderPrice)** |
| Bush | 2000 |
| Carter | 1700 |
| Adams | 2000 |

很棒吧，对不对？

让我们看一下如果省略 GROUP BY 会出现什么情况：

SELECT Customer,SUM(OrderPrice) FROM Orders

结果集类似这样：

|  |  |
| --- | --- |
| **Customer** | **SUM(OrderPrice)** |
| Bush | 5700 |
| Carter | 5700 |
| Bush | 5700 |
| Bush | 5700 |
| Adams | 5700 |
| Carter | 5700 |

上面的结果集不是我们需要的。

那么为什么不能使用上面这条 SELECT 语句呢？解释如下：上面的 SELECT 语句指定了两列（Customer 和 SUM(OrderPrice)）。"SUM(OrderPrice)" 返回一个单独的值（"OrderPrice" 列的总计），而 "Customer" 返回 6 个值（每个值对应 "Orders" 表中的每一行）。因此，我们得不到正确的结果。不过，您已经看到了，GROUP BY 语句解决了这个问题。

**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，就像这样：

SELECT Customer,OrderDate,SUM(OrderPrice) FROM Orders

GROUP BY Customer,OrderDate

## HAVING 子句

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

**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

**SQL HAVING 实例**

我们拥有下面这个 "Orders" 表：

|  |  |  |  |
| --- | --- | --- | --- |
| **O\_Id** | **OrderDate** | **OrderPrice** | **Customer** |
| 1 | 2008/12/29 | 1000 | Bush |
| 2 | 2008/11/23 | 1600 | Carter |
| 3 | 2008/10/05 | 700 | Bush |
| 4 | 2008/09/28 | 300 | Bush |
| 5 | 2008/08/06 | 2000 | Adams |
| 6 | 2008/07/21 | 100 | Carter |

现在，我们希望查找订单总金额少于 2000 的客户。

我们使用如下 SQL 语句：

SELECT Customer,SUM(OrderPrice) FROM Orders

GROUP BY Customer

HAVING SUM(OrderPrice)<2000

结果集类似：

|  |  |
| --- | --- |
| **Customer** | **SUM(OrderPrice)** |
| Carter | 1700 |

现在我们希望查找客户 "Bush" 或 "Adams" 拥有超过 1500 的订单总金额。

我们在 SQL 语句中增加了一个普通的 WHERE 子句：

SELECT Customer,SUM(OrderPrice) FROM Orders

WHERE Customer='Bush' OR Customer='Adams'

GROUP BY Customer

HAVING SUM(OrderPrice)>1500

结果集：

|  |  |
| --- | --- |
| **Customer** | **SUM(OrderPrice)** |
| Bush | 2000 |
| Adams | 2000 |

## 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

**SQL UCASE() 实例**

我们拥有下面这个 "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 |

现在，我们希望选取 "LastName" 和 "FirstName" 列的内容，然后把 "LastName" 列转换为大写。

我们使用如下 SQL 语句：

SELECT UCASE(LastName) as LastName,FirstName FROM Persons

结果集类似这样：

|  |  |
| --- | --- |
| **LastName** | **FirstName** |
| ADAMS | John |
| BUSH | George |
| CARTER | Thomas |

## 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

**SQL LCASE() 实例**

我们拥有下面这个 "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 |

现在，我们希望选取 "LastName" 和 "FirstName" 列的内容，然后把 "LastName" 列转换为小写。

我们使用如下 SQL 语句：

SELECT LCASE(LastName) as LastName,FirstName FROM Persons

结果集类似这样：

|  |  |
| --- | --- |
| **LastName** | **FirstName** |
| adams | John |
| bush | George |
| carter | Thomas |

## 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。 |

**SQL MID() 实例**

我们拥有下面这个 "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 |

现在，我们希望从 "City" 列中提取前 3 个字符。

我们使用如下 SQL 语句：

SELECT MID(City,1,3) as SmallCity FROM Persons

结果集类似这样：

|  |
| --- |
| **SmallCity** |
| Lon |
| New |
| Bei |

## 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

**SQL LEN() 实例**

我们拥有下面这个 "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 |

现在，我们希望取得 "City" 列中值的长度。

我们使用如下 SQL 语句：

SELECT LEN(City) as LengthOfCity FROM Persons

结果集类似这样：

|  |
| --- |
| **LengthOfCity** |
| 6 |
| 8 |
| 7 |

## 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。 |

**SQL ROUND() 实例**

我们拥有下面这个 "Products" 表：

|  |  |  |  |
| --- | --- | --- | --- |
| **Prod\_Id** | **ProductName** | **Unit** | **UnitPrice** |
| 1 | gold | 1000 g | 32.35 |
| 2 | silver | 1000 g | 11.56 |
| 3 | copper | 1000 g | 6.85 |

现在，我们希望把名称和价格舍入为最接近的整数。

我们使用如下 SQL 语句：

SELECT ProductName, ROUND(UnitPrice,0) as UnitPrice FROM Products

结果集类似这样：

|  |  |
| --- | --- |
| **ProductName** | **UnitPrice** |
| gold | 32 |
| silver | 12 |
| copper | 7 |

## NOW() 函数

NOW 函数返回当前的日期和时间。

**提示：**如果您在使用 Sql Server 数据库，请使用 getdate() 函数来获得当前的日期时间。

**SQL NOW() 语法**

SELECT NOW() FROM table\_name

**SQL NOW() 实例**

我们拥有下面这个 "Products" 表：

|  |  |  |  |
| --- | --- | --- | --- |
| **Prod\_Id** | **ProductName** | **Unit** | **UnitPrice** |
| 1 | gold | 1000 g | 32.35 |
| 2 | silver | 1000 g | 11.56 |
| 3 | copper | 1000 g | 6.85 |

现在，我们希望显示当天的日期所对应的名称和价格。

我们使用如下 SQL 语句：

SELECT ProductName, UnitPrice, Now() as PerDate FROM Products

结果集类似这样：

|  |  |  |
| --- | --- | --- |
| **ProductName** | **UnitPrice** | **PerDate** |
| gold | 32.35 | 12/29/2008 11:36:05 AM |
| silver | 11.56 | 12/29/2008 11:36:05 AM |
| copper | 6.85 | 12/29/2008 11:36:05 AM |

## 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。 |

**SQL FORMAT() 实例**

我们拥有下面这个 "Products" 表：

|  |  |  |  |
| --- | --- | --- | --- |
| **Prod\_Id** | **ProductName** | **Unit** | **UnitPrice** |
| 1 | gold | 1000 g | 32.35 |
| 2 | silver | 1000 g | 11.56 |
| 3 | copper | 1000 g | 6.85 |

现在，我们希望显示每天日期所对应的名称和价格（日期的显示格式是 "YYYY-MM-DD"）。

我们使用如下 SQL 语句：

SELECT ProductName, UnitPrice, FORMAT(Now(),'YYYY-MM-DD') as PerDate

FROM Products

结果集类似这样：

|  |  |  |
| --- | --- | --- |
| **ProductName** | **UnitPrice** | **PerDate** |
| gold | 32.35 | 12/29/2008 |
| silver | 11.56 | 12/29/2008 |
| copper | 6.85 | 12/29/2008 |

**SQL 快速参考**

|  |  |
| --- | --- |
| **语句** | **语法** |
| AND / OR | SELECT column\_name(s) FROM table\_name WHERE condition AND|OR condition |
| ALTER TABLE (add column) | ALTER TABLE table\_name  ADD column\_name datatype |
| ALTER TABLE (drop column) | ALTER TABLE table\_name  DROP COLUMN column\_name |
| AS (alias for column) | SELECT column\_name AS column\_alias FROM table\_name |
| AS (alias for table) | 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 INDEX | CREATE INDEX index\_name ON table\_name (column\_name) |
| CREATE TABLE | CREATE TABLE table\_name ( column\_name1 data\_type, column\_name2 data\_type, ....... ) |
| CREATE UNIQUE INDEX | 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 FROM | DELETE FROM table\_name  (**Note:** Deletes the entire table!!)  *or*  DELETE FROM table\_name WHERE condition |
| DROP DATABASE | DROP DATABASE database\_name |
| DROP INDEX | DROP INDEX table\_name.index\_name |
| DROP TABLE | DROP TABLE table\_name |
| GROUP BY | SELECT column\_name1,SUM(column\_name2) FROM table\_name GROUP BY column\_name1 |
| HAVING | SELECT column\_name1,SUM(column\_name2) FROM table\_name GROUP BY column\_name1 HAVING SUM(column\_name2) condition value |
| IN | SELECT column\_name(s) FROM table\_name WHERE column\_name IN (value1,value2,..) |
| INSERT INTO | INSERT INTO table\_name VALUES (value1, value2,....)  *or*  INSERT INTO table\_name (column\_name1, column\_name2,...) VALUES (value1, value2,....) |
| 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 (used to create backup copies of tables) | SELECT \* INTO new\_table\_name FROM original\_table\_name  *or*  SELECT column\_name(s) INTO new\_table\_name FROM original\_table\_name |
| TRUNCATE TABLE (deletes only the data inside the table) | TRUNCATE TABLE table\_name |
| UPDATE | UPDATE table\_name SET column\_name=new\_value [, column\_name=new\_value] WHERE column\_name=some\_value |
| WHERE | SELECT column\_name(s) FROM table\_name WHERE condition |