# 一、JOIN关键字

**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。**

## 1、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。

**"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。

**"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。

**2、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 |

**3、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 |

**4、不同的 JOIN**

下面列出了您可以使用的 JOIN 类型，以及它们之间的差异。

* INNER JOIN: 内连接，只有两个表都匹配才会显示记录；
* LEFT (OUTER) JOIN: 左（外）连接，即使右表中没有匹配，也从左表返回所有的行；
* RIGHT (OUTER) JOIN: 右（外）连接，即使左表中没有匹配，也从右表返回所有的行；
* FULL JOIN: 全连接，只要其中一个表中存在匹配，就返回行；

# 二、INNER JOIN 关键字

在两张表中存在至少一个匹配时，INNER JOIN 关键字返回行。

## 1、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 |

## 2、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。

# 三、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。

mysql中没有FULL JOIN,可以采用左连接+ union+右连接的 方式来实现；

**原始的表**

"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 |
| 5 | 34764 | 65 |

**2、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 LEFT JOIN Orders ON Persons.Id\_P=Orders.Id\_P

UNION

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

FROM Persons RIGHT JOIN Orders ON Persons.Id\_P=Orders.Id\_P

结果集：

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

注意：LEFT JOIN会从左表中返回所有行，RIGHT JOIN会从右表中返回所有行，UNION将两个结果集进行合并。

**3、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。

* **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。

* **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表:**

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

**Employees\_USA表:**

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

**4、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。

**5、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

**6、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 语句可用于创建表的备份复件。**

# 四、CREATE TABLE (SELECT) 语句

CREATE TABLE (SELECT) 语句从一个表中选取数据，然后把数据插入另一个表中。

CREATE TABLE (SELECT) 语句常用于创建表的备份复件或者用于对记录进行存档。

## 1、CREATE TABLE (SELECT) 语法

您可以把所有的列插入新表：

CREATE TABLE new\_table\_name (SELECT \* FROM old\_tablename)

或者只把希望的列插入新表：

CREATE TABLE new\_table\_name (SELECT ColumnName FROM old\_tablename)

## 2、CREATE TABLE (SELECT)实例 - 制作备份复件

下面的例子会制作 "Persons" 表的备份复件：

CREATE TABLE Persons\_backup(SELECT \* FROM Persons)

IN 子句可用于向另一个数据库中拷贝表：

CREATE TABLE database.Persons (SELECT \* 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：

CREATE TABLE Persons\_backup(SELECT LastName,FirstName

FROM Persons)

**3、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：

CREATE TABLE Persons\_backup (SELECT LastName,Firstname

FROM Persons WHERE City='Beijing')

**4、**CREATE TABLE(SELECT) **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：

CREATE TABLE Persons\_Order\_Backup

(SELECT Persons.LastName,Orders.OrderNo

FROM Persons

INNER JOIN Orders

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

# 五、CREATE TABLE 语句

CREATE TABLE 语句用于创建数据库中的表。

## 1、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。 |

## 2、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。

# 六、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。

## 1、建表时的Unique约束

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

CREATE TABLE Persons

(

Id\_P int NOT NULL,

LastNamevarchar(255) NOT NULL,

FirstNamevarchar(255),

Address varchar(255),

City varchar(255),

UNIQUE (Id\_P)

)

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)

)

**2、修改表的Unique约束**

当表已被创建时，如需在 "Id\_P" 列创建 UNIQUE 约束，请使用下列 SQL：

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：

ALTER TABLE Persons

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

**3、撤销 Unique 约束**

如需撤销 UNIQUE 约束，请使用下面的 SQL：

ALTER TABLE Persons

DROP INDEX uc\_PersonID

# 七、 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。

**1、建表时的主键约束**

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

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)

)

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：

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)

)

**2、修改表的主键约束**

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：

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：

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）。

**3、撤销 PRIMARY KEY 约束**

如需撤销 PRIMARY KEY 约束，请使用下面的 SQL：

ALTER TABLE Persons

DROP PRIMARY KEY

# 八、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。

**1、建表时的外键约束**

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

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)

)

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：

CREATE TABLE Orders

(

Id\_Oint NOT NULL,

OrderNoint NOT NULL,

Id\_Pint,

PRIMARY KEY (Id\_O),

CONSTRAINT fk\_PerOrders FOREIGN KEY (Id\_P)

REFERENCES Persons(Id\_P)

)

**2、修改表的外键约束**

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：

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：

ALTER TABLE Orders

ADD CONSTRAINT fk\_PerOrders

FOREIGN KEY (Id\_P)

REFERENCES Persons(Id\_P)

**3、撤销 FOREIGN KEY 约束**

如需撤销 FOREIGN KEY 约束，请使用下面的 SQL：

ALTER TABLE Orders

DROP FOREIGN KEY fk\_PerOrders

**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。**

# 九、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(针对海量数据的情况)。

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用更多的时间，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。

## 1、 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。

## 2、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)

**3、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)

**4、DROP INDEX 语法**

可以对表中建好的索引进行删除，可以使用以下的语句：

DROP INDEX index\_name

ON table\_name

**5、DROP INDEX实例**

如果已经在表product上建立了索引product\_index，可以采用下面的语句进行删除：

DROP INDEX product\_index on product;

# 十、ALTER TABLE 语句

ALTER TABLE 语句用于在已有的表中添加、修改或删除列。

## 1、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 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

modify 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 |

## 2、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 |  |

## 3、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

MODIFY 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。

## 4、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 |

# 十一、AUTO INCREMENT 语句

**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。

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,

LastNamevarchar(255) NOT NULL,

FirstNamevarchar(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"。

# 十二、SQL函数

Mysql 拥有很多可用于计数和计算的内建函数。

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 函数的操作面向一系列的值，并返回一个单一的值。如：

* COUNT()函数：用来统计记录的条数；
* SUM()函数:是求和函数；
* AVG()函数:是求平均值的函数 ;
* MAX()函数是求最大值的函数
* MIN()函数是求最小值的函数

**注释：**group by 语句经常和合计函数配合使用来进行求和、计数、求平均值、求最大值和求最小值。

**"Persons" 表：**

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

### 标量函数

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。如：

* abs：该函数返回一个数值表达式的绝对值。如abs(-123);
* addtime：把两个时间表达式加起来。如addtime('100:00:00','200:02:04');
* concat：该函数合并两个字符串的值。
* convert：该函数转换参数1的数据类型为参数2指定的类型。如：convert('12.56',unsigned integer);

## 1、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。

### 1、Date 函数

下面的表格列出了 MySQL 中最重要的内建日期函数：

|  |  |
| --- | --- |
| **函数** | **描述** |
| [NOW()](http://www.w3school.com.cn/sql/func_now.asp) | 返回当前的日期和时间 |
| [CURDATE()](http://www.w3school.com.cn/sql/func_curdate.asp) | 返回当前的日期 |
| [CURTIME()](http://www.w3school.com.cn/sql/func_curtime.asp) | 返回当前的时间 |
| [DATE(exp)](http://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  **SELECT DATE('2018-09-07');** |
| [DATE\_ADD()](http://www.w3school.com.cn/sql/func_date_add.asp) | 给日期添加指定的时间间隔,如：  SELECT DATE\_ADD('2018-05-12',INTERVAL 2 MINUTE) |
| [DATE\_SUB()](http://www.w3school.com.cn/sql/func_date_sub.asp) | 从日期减去指定的时间间隔，如：  SELECT DATE\_SUB('2018-05-12',INTERVAL 2 DAY) |
| [DATEDIFF()](http://www.w3school.com.cn/sql/func_datediff_mysql.asp) | 返回两个日期之间的天数，如：  SELECT DATEDIFF('2018-05-12','2018-05-10') |
| [DATE\_FORMAT()](http://www.w3school.com.cn/sql/func_date_format.asp) | 用不同的格式显示日期/时间，如：  SELECT DATE\_FORMAT(NOW(),'%b %d %Y %h:%i %p') |

### 2、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

### 3、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。这是由于该查询不到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！

## 2、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。

### AVG() 语法

SELECT AVG(column\_name) FROM table\_name

### 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 |

## 3、COUNT() 函数

**COUNT() 函数返回匹配指定条件的行数。**

**COUNT() 语法**

### COUNT(column\_name) 语法

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

SELECT COUNT(column\_name) FROM table\_name

### COUNT(\*) 语法

COUNT(\*) 函数返回表中的记录数：

SELECT COUNT(\*) FROM table\_name

### 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。

**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。

**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。

## 4、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。

### GROUP BY 语法

SELECT column\_name, aggregate\_function(column\_name)

FROM table\_name

WHERE column\_name operator value

GROUP BY column\_name

### 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。

## 5、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。

### 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

### 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 |

# 十三、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。

## 1、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。 |

## 2、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。

## 3、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。

# 十四、视图

## 1、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。

## 2、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。

## 3、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'

## 4、更新视图

您可以使用下面的语法来更新视图：

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

**5、撤销视图**

您可以通过 DROP VIEW 命令来删除视图。

DROP VIEW view\_name