## 数据类型

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

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

### MySQL 数据类型

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

**Text 类型：**

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

**Number 类型：**

|  |  |
| --- | --- |
| **数据类型** | **描述** |
| TINYINT(size) | 带符号-128到127 ，无符号0到255。 |
| SMALLINT(size) | 带符号范围-32768到32767，无符号0到65535, size 默认为 6。 |
| MEDIUMINT(size) | 带符号范围-8388608到8388607，无符号的范围是0到16777215。 size 默认为9 |
| INT(size) | 带符号范围-2147483648到2147483647，无符号的范围是0到4294967295。 size 默认为 11 |
| BIGINT(size) | 带符号的范围是-9223372036854775808到9223372036854775807，无符号的范围是0到18446744073709551615。size 默认为 20 |
| FLOAT(size,d) | 带有浮动小数点的小数字。在 size 参数中规定显示最大位数。在 d 参数中规定小数点右侧的最大位数。 |
| DOUBLE(size,d) | 带有浮动小数点的大数字。在 size 参数中规显示定最大位数。在 d 参数中规定小数点右侧的最大位数。 |
| DECIMAL(size,d) | 作为字符串存储的 DOUBLE 类型，允许固定的小数点。在 size 参数中规定显示最大位数。在 d 参数中规定小数点右侧的最大位数。 |

***注意：****以上的 size 代表的并不是存储在数据库中的具体的长度。*

*实际上int(size)所占多少存储空间并无任何关系。int(3)、int(4)、int(8) 在磁盘上都是占用 4 btyes 的存储空间。就是在显示给用户的方式有点不同外，int(M) 跟 int 数据类型是相同的。*

*例如：*

*1、int的值为10 （指定zerofill）*

int（9）显示结果为000000010

int（3）显示结果为010

**Date 类型：**

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

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

## select where



搜索 empno 等于 7900 的数据：

Select \* from emp where empno=7900;

条件：列，比较运算符，值

比较运算符包涵：= > < >= ,<=, !=,<> 表示（不等于）

Select \* from emp where ename='SMITH';

例子中的 SMITH 用单引号引起来，表示是字符串，字符串要区分大小写。

And:与 同时满足两个条件的值。

Select \* from emp where sal > 2000 and sal < 3000;

查询 EMP 表中 SAL 列中大于 2000 小于 3000 的值。

Or:或 满足其中一个条件的值

Select \* from emp where sal > 2000 or comm > 500;

查询 emp 表中 SAL 大于 2000 或 COMM 大于500的值。

Not:非 满足不包含该条件的值。

select \* from emp where not sal > 1500;

查询EMP表中 sal 小于等于 1500 的值。

逻辑运算的优先级：

() not and or

**1.空值判断： is null**

Select \* from emp where comm is null;

查询 emp 表中 comm 列中的空值。

**2.between and (在 之间的值)**

Select \* from emp where sal between 1500 and 3000;

查询 emp 表中 SAL 列中大于 1500 的小于 3000 的值。

注意：大于等于 1500 且小于等于 3000， 1500 为下限，3000 为上限，下限在前，上限在后，查询的范围包涵有上下限的值。

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

**因此，请检查您的数据库是如何处理 BETWEEN 操作符！**

**3.In**

Select \* from emp where sal in (5000,3000,1500);

查询 EMP 表 SAL 列中等于 5000，3000，1500 的值。

**4.like**

Like模糊查询

Select \* from emp where ename like 'M%';

查询 EMP 表中 Ename 列中有 M 的值，M 为要查询内容中的模糊信息。

* **%** 表示多个字值，**\_** 下划线表示一个字符；
* **M%** : 正则表达式，表示的意思为模糊查询信息为 M 开头的。
* **%M%** : 表示查询包含M的所有内容。
* **%M\_** : 表示查询以M在倒数第二位的所有内容。

选取 name 不包含模式 "oo" 的所有客户：

SELECT \* FROM Websites WHERE name **NOT** LIKE '%oo%';

select \* from username where 用户名 like '段\_%' -- 会查出来段煜 段鑫

select \* from username where 用户名 like '段\\_%' **escape '\'**

-- 通过 \转义,只能查出来 段\_煜

**5 正则表达式**

MySQL 中使用 **REGEXP** 或 **NOT REGEXP** 运算符 (或 RLIKE 和 NOT RLIKE) 来操作正则表达式。

下面的 SQL 语句选取 name 以 "G"、"F" 或 "s" 开始的所有网站：

SELECT \* FROM Websites  
WHERE name REGEXP '^[GFs]';

WHERE 子句并不一定带比较运算符，当不带运算符时，会执行一个隐式转换。当 0 时转化为 false，1 转化为 true。例如：

SELECT studentNO FROM student WHERE 0

则会返回一个空集，因为每一行记录 WHERE 都返回 false。

SELECT studentNO FROM student WHERE 1

返回 student 表所有行中 studentNO 列的值。因为每一行记录 WHERE 都返回 true。

## Order by

order by A,B 这个时候都是默认按升序排列

order by A desc,B 这个时候 A 降序，B 升序排列

order by A ,B desc 这个时候 A 升序，B 降序排列

即 **desc** 或者 **asc** 只对它紧跟着的第一个列名有效，其他不受影响，仍然是默认的升序。

## INSERT INTO

INSERT INTO 语句可以有两种编写形式。

第一种形式无需指定要插入数据的列名，只需提供被插入的值即可：

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

没有指定要插入数据的列名的形式需要列出插入行的每一列数据:

第二种形式需要指定列名及被插入的值：

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

## UPDATE 语法

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

WHERE 子句规定哪条记录或者哪些记录需要更新。如果您省略了 WHERE 子句，所有的记录都将被更新！

执行没有 WHERE 子句的 UPDATE 要慎重，再慎重。

在 MySQL 中可以通过设置 **sql\_safe\_updates** 这个自带的参数来解决，当该参数开启的情况下，你必须在update 语句后携带 where 条件，否则就会报错。

**set sql\_safe\_updates=1;** 表示开启该参数

假设我们要从 "Websites" 表中把 "菜鸟教程" 的 alexa 排名更新为 5000，country 改为 USA。

UPDATE Websites SET alexa='5000', country='USA' WHERE name='菜鸟教程';

## DELETE 语法

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

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

假设我们要从 "Websites" 表中删除网站名为 "百度" 且国家为 CN 的网站 。

DELETE FROM Websites WHERE name='百度' AND country='CN';

## SQL 别名

通过使用 SQL，可以为表名称或列名称指定别名。

基本上，创建别名是为了让列名称的可读性更强。

### 列的 SQL 别名语法

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

### 表的 SQL 别名语法

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

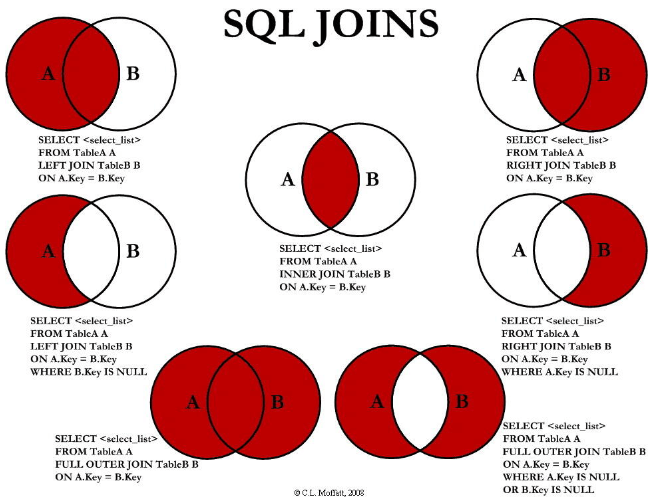
把三个列（url、alexa 和 country）结合在一起，并创建一个名为 "site\_info" 的别名：

SELECT name, CONCAT(url, ', ', alexa, ', ', country) AS site\_info  
FROM Websites;

## 连接(JOIN)

用于把来自两个或多个表的行结合起来，基于这些表之间的共同字段

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

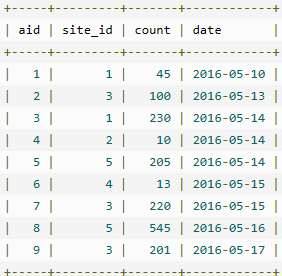


"Websites" 表的数据



"access\_log" 网站访问记录表的数据。

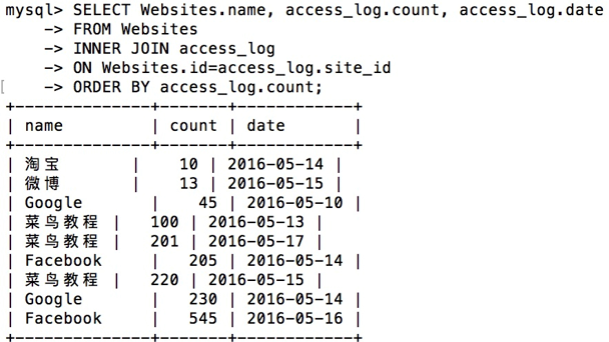
"Websites" 表中的 "**id**" 列指向 "access\_log" 表中的字段 "**site\_id**"。上面这两个表是通过 "site\_id" 列联系起来的



### INNER JOIN

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

返回所有网站的访问记录



### LEFT JOIN

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

返回所有网站及他们的访问量（如果有的话）



### RIGHT JOIN

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

返回网站的访问记录。把 access\_log 作为左表，Websites 作为右表



### FULL OUTER JOIN

FULL OUTER JOIN 关键字只要左表（table1）和右表（table2）其中一个表中存在匹配，则返回行. FULL OUTER JOIN 关键字结合了 LEFT JOIN 和 RIGHT JOIN 的结果。

MySQL中不支持 FULL OUTER JOIN，你可以在 SQL Server 测试以下实例。

SELECT Websites.name, access\_log.count, access\_log.date  
FROM Websites  
FULL OUTER JOIN access\_log  
ON Websites.id=access\_log.site\_id  
ORDER BY access\_log.count DESC;

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

## UNION 操作符

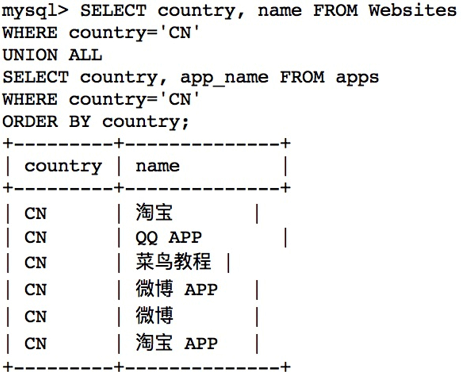
UNION 操作符用于合并两个或多个 SELECT 语句的结果集。

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

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

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

使用 UNION ALL 从 "Websites" 和 "apps" 表中选取**所有的**中国(CN)的数据（也有重复的值）



## 复制表

### SELECT INTO

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

*MySQL 数据库不支持 SELECT ... INTO 语句，但支持* [*INSERT INTO ... SELECT*](https://www.runoob.com/sql/sql-insert-into-select.html) *。*

复制多个表中的数据插入到新表中：

SELECT Websites.name, access\_log.count, access\_log.date  
INTO WebsitesBackup2016  
FROM Websites  
LEFT JOIN access\_log  
ON Websites.id=access\_log.site\_id;

### INSERT INTO SELECT 语句

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

只复 第一条 的 APP 数据到 "Websites" 中：

INSERT INTO Websites (name, country)  
SELECT app\_name, country FROM apps  
WHERE id=1;

**select into from** 要求目标表不存在，因为在插入时会自动创建；**insert into select from** 要求目标表存在。

1. 复制表结构及其数据：

create table table\_name\_new as select \* from table\_name\_old

2. 只复制表结构：

create table table\_name\_new as select \* from table\_name\_old where 1=2;

或者：

create table table\_name\_new like table\_name\_old

3. 只复制表数据：

如果两个表结构一样：

insert into table\_name\_new select \* from table\_name\_old

如果两个表结构不一样：

insert into table\_name\_new(column1,column2...) select column1,column2... from table\_name\_old

## 创建数据库和表

创建一个名为 "my\_db" 的数据库：

CREATE DATABASE my\_db;

数据库表可以通过 CREATE TABLE 语句来添加

创建一个名为 "Persons" 的表，包含五列：PersonID、LastName、FirstName、Address 和 City

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

## SELECT TOP, LIMIT, ROWNUM 子句

并非所有的数据库系统都支持 SELECT TOP 语句。 MySQL 支持 LIMIT 语句来选取指定的条数数据， Oracle 可以使用 ROWNUM 来选取。

--前5行

select top 5 \* from table

--后5行

select top 5 \* from table order by id desc

### MySQL 语法

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

SELECT \*  
FROM Persons  
LIMIT 5;

### Oracle 语法

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

SELECT \*  
FROM Persons  
WHERE ROWNUM <=5;

## 约束constraint（非空，唯一，主键，外键，检查）

SQL 约束用于规定表中的数据规则。

如果存在违反约束的数据行为，行为会被约束终止。

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

### SQL CREATE TABLE + CONSTRAINT 语法

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

在 SQL 中，我们有如下约束：

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

**■ 非空 NOT NULL**

要求此项目数据内容不能为空

1.建表时直接追加

create table user{  
'user\_id' varchar(8)**not null** COMMENT '用户id',  
'user\_name' varchar(32)**not null** COMMENT '用户名',  
'password' varchar(128)  COMMENT '密码',  
primary key ('user\_id') ,  
unique key ('user\_name')  
} ENGINE=InnoDB DEFAULT CHARSET=utf8;

2.对已存在的表增加非空约束

ALTER table user MODIFY(‘password’ varchar(128) not Null);

3.对已存在的表删除非空约束

ALTER table user MODIFY(‘password’ varchar(128) Null);

**■ 唯一性约束 unique key**

字段或字段的组合不能有重复值（不包括null）

1.建表时直接追加

create table user{  
'user\_id' varchar(8) not null COMMENT '用户id',  
'user\_name' varchar(32) not null COMMENT '用户名',  
'password' varchar(128)  COMMENT '密码',  
primary key ('user\_id') ,  
unique key ('user\_name')  
} ENGINE=InnoDB DEFAULT CHARSET=utf8;

或

create table user{  
'user\_id' varchar(8) not null COMMENT '用户id',  
'user\_name' varchar(32) not null  uniqueCOMMENT '用户名',  
'password' varchar(128)  COMMENT '密码',  
primary key ('user\_id')   
} ENGINE=InnoDB DEFAULT CHARSET=utf8;

2.对已存在的表增加唯一性约束

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

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

ALTER TABLE Persons ADD UNIQUE (P\_Id)

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

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

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

4、撤销 UNIQUE 约束：

**MySQL：**

ALTER TABLE Persons DROP INDEX uc\_PersonID

**SQL Server / Oracle / MS Access：**

ALTER TABLE Persons DROP CONSTRAINT uc\_PersonID

**■ 主键约束 PRIMARY KEY**

一个表中只能有一个主键（可以是单列，也可以是多个字段的组合）

主键：对应的字段既要是not null也要是unique key；尽量是单列；对系统[无意义](https://www.baidu.com/s?wd=%E6%97%A0%E6%84%8F%E4%B9%89&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)的数据；不要更新主键；不要是动态数据（如时间戳）；尽量是自动生成数据，

用途：仅是唯一标识行

1.建表时直接追加

create table user{  
'user\_id' varchar(8) not null primary key COMMENT '用户id',  
'user\_name' varchar(32) not null COMMENT '用户名',  
'password' varchar(128)  COMMENT '密码',  
unique key ('user\_name')  
} ENGINE=InnoDB DEFAULT CHARSET=utf8;

或

create table user{  
'user\_id' varchar(8) not null COMMENT '用户id',  
'user\_name' varchar(32) not null COMMENT '用户名',  
'password' varchar(128)  COMMENT '密码',  
primary key ('user\_id') ,  
unique key ('user\_name')  
} ENGINE=InnoDB DEFAULT CHARSET=utf8;

如需命名 PRIMARY KEY 约束，并定义多个列的 PRIMARY KEY 约束：

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

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

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

2.对已存在的表增加主键约束

ALTER TABLE user ADD constraint  pkid primary key ('user\_id') ;

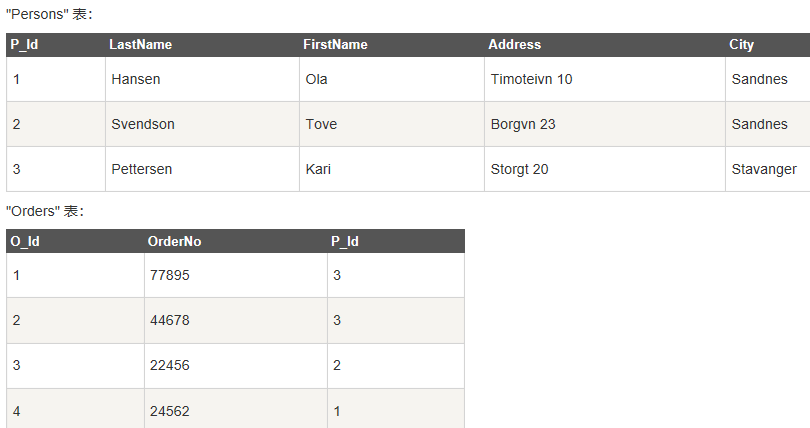
3.删除主键

Alter table user Drop Constraint PrimaryKey(默认名);

Alter table user Drop Constraint pkid (主键约束的别名);

**■ 外键约束 FOREIGN KEY**

注意事项：子表列的值必须从主表得到，不能为null；主表字段的数据被子表参照使用后，数据不能删除。



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

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

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

FOREIGN KEY 约束用于预防破坏表之间连接的行为。

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

1.建表时直接追加

**MySQL：**

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

**SQL Server / Oracle / MS Access：**

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

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

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

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

2.对已存在的表增加外键约束

ALTER TABLE note ADD constraint  `FK\_note\_userid` FOREIGN KEY (`user\_id`) REFERENCES `user` (`user\_id`);

**■ 检查约束 check**

自定义硬性的检查条件，不满足时DML语句不执行。

ALTER TABLE note ADD CONSTRAINT 'check\_notebook\_id' CHECK('notebook\_id' >100);

## CHECK 约束

CHECK 约束用于限制列中的值的范围。

如果对单个列定义 CHECK 约束，那么该列只允许特定的值。

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

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

**MySQL：**

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

**SQL Server / Oracle / MS Access：**

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

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

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

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

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

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

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

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

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

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

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。

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

CREATE TABLE Orders

(

    O\_Id int NOT NULL,

    OrderNo int NOT NULL,

    P\_Id int,

    OrderDate date DEFAULT GETDATE()

)

## CREATE INDEX

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。

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

### SQL CREATE INDEX 语法

在表上创建一个简单的索引。允许使用重复的值：

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

### SQL CREATE UNIQUE INDEX 语法

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

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

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

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

CREATE INDEX PIndex  
ON Persons (LastName)

如果您希望索引不止一个列，您可以在括号中列出这些列的名称，用逗号隔开：

CREATE INDEX PIndex  
ON Persons (LastName, FirstName)

## Drop撤销索引、撤销表以及撤销数据库

DROP TABLE 语句用于删除表。

DROP TABLE table\_name

DROP DATABASE 语句用于删除数据库。

DROP DATABASE database\_name

如果我们仅仅需要删除表内的数据，但并不删除表本身，请使用 TRUNCATE TABLE 语句：

TRUNCATE TABLE table\_name

### 用于 MySQL 的 DROP INDEX 语法：

ALTER TABLE table\_name DROP INDEX index\_name

## ALTER TABLE 语句

ALTER TABLE 语句用于在已有的表中添加、删除或修改列。

如需在表中添加列，请使用下面的语法:

ALTER TABLE table\_name  
ADD column\_name datatype

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

ALTER TABLE table\_name  
DROP COLUMN column\_name

要改变表中列的数据类型，请使用下面的语法：

**SQL Server / MS Access：**

ALTER TABLE table\_name  
ALTER COLUMN column\_name datatype

**My SQL / Oracle：**

ALTER TABLE table\_name  
MODIFY COLUMN column\_name datatype

**AUTO INCREMENT 字段**

我们通常希望在每次插入新记录时，自动地创建主键字段的值。可以在表中创建一个 auto-increment 字段。

### 用于 MySQL 的语法

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

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

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

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

ALTER TABLE Persons AUTO\_INCREMENT=100

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

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

## 视图（Views）

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

1、视图隐藏了底层的表结构，简化了数据访问操作，客户端不再需要知道底层表的结构及其之间的关系。

2、视图提供了一个统一访问数据的接口。（即可以允许用户通过视图访问数据的安全机制，而不授予用户直接访问底层表的权限）

3、从而加强了安全性，使用户只能看到视图所显示的数据。

4、视图还可以被嵌套，一个视图中可以嵌套另一个视图。

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

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

我们可以像这样查询上面这个视图：

SELECT \* FROM [Current Product List]

DROP VIEW 命令来删除视图。

DROP VIEW view\_name

更新视图：

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

## 日期

**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 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。

无法比较 NULL 和 0；它们是不等价的

无法使用比较运算符来测试 NULL 值，比如 =、< 或 <>。

我们必须使用 IS NULL 和 IS NOT NULL 操作符。



仅仅选取在 "Address" 列中不带有 NULL 值的记录

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

由于NULL影响运算，在 MySQL 中，我们可以使用 IFNULL() 函数，或者我们可以使用 COALESCE() 函数

-如果alexa列为null值，则赋予0，否则，取原值

select id,name,url,ifnull(alexa,0)from websites;

select id,name,url,COALESCE(alexa,0) from websites;

## SQL 函数

### 平均值

选择访问量高于平均访问量的 "site\_id" 和 "count"：

SELECT site\_id, count FROM access\_log  
WHERE count > (SELECT **AVG(**count**)** FROM access\_log);

### 数量

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(\*) 函数返回表中的记录数：

SELECT COUNT(\*) FROM table\_name;

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;

计算 "access\_log" 表中 "site\_id"=3 的总访问量：

SELECT COUNT(count) AS nums FROM access\_log  
WHERE site\_id=3;

### 最大值和最小值

SELECT MAX(column\_name) FROM table\_name

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

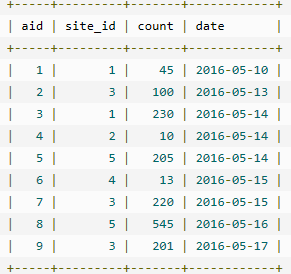
### 总和

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

### 对结果集进行分组

统计 access\_log 各个 site\_id 的访问量

SELECT site\_id, SUM(access\_log.count) AS nums  
FROM access\_log **GROUP BY** site\_id;



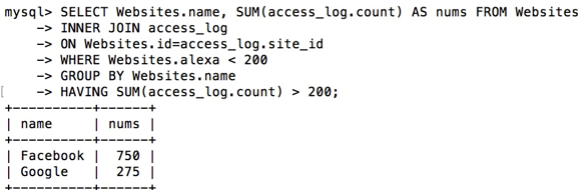
### HAVING 语法

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

HAVING 子句可以让我们筛选分组后的各组数据。

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;

查找总访问量大于 200 的网站，并且 alexa 排名小于 200



### 大写UCASE() 和 小写LCASE()

从 "Websites" 表中选取 "name" 和 "url" 列，并把 "name" 列的值转换为大写：

SELECT UCASE(name) AS site\_title, url  
FROM Websites;

### MID() 函数用于从文本字段中提取字符

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。 |

从 "Websites" 表的 "name" 列中提取前 4 个字符：

SELECT MID(name,1,4) AS ShortTitle  
FROM Websites;

### LEN() 函数返回文本字段中值的长度

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

从 "Websites" 表中选取 "name" 和 "url" 列中值的长度：

SELECT name, LENGTH(url) as LengthOfURL  
FROM Websites;

### ROUND() 函数用于把数值字段舍入为指定的小数位数

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。 |

### Now()

面的 SQL 语句从 "Websites" 表中选取 name，url，及当天日期：

SELECT name, url, Now() AS date  
FROM Websites;



### FORMAT() 函数用于对字段的显示进行格式化

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。 |