# MySQL

## 开始使用

- `表头(header)`:每一列的名称;
- `列(col)`:具有相同数据类型的数据的集合;
- `行(row)`:每一行用来描述某个人/物的具体信息;
- `值(value)`:行的具体信息,每个值必须与该列的数据类型相同;
- `键(key)`:表中用来识别某个特定的人\物的方法,键的值在当前列中具有唯一性。

### 登录MySQL

```
mysql -h 127.0.0.1 -u <用户名> -p<密码>.    # 默认用户名<root>, -p 是密码, 参数后面不需要空格
mysql -D 所选择的数据库名 -h 主机名 -u 用户名 -p
mysql> exit # 退出 使用 “quit;” 或 “\q;” 一样的效果
mysql> status;   # 显示当前mysql的version的各种信息
mysql> select version(); # 显示当前mysql的version信息
mysql> show global variables like 'port'; # 查看MySQL端口号
```

### 创建数据库

对于表的操作需要先进入库`use 库名;`

```
-- 创建一个名为 samp_db 的数据库, 数据库字符编码指定为 gbk
create database samp_db character set gbk;
drop database samp_db; -- 删除 库名为 samp_db 的库
show databases;            -- 显示数据库列表。
use samp_db;                -- 选择创建的数据库 samp_db
show tables;                -- 显示 samp_db 下面所有的表名字
describe 表名;              -- 显示数据表的结构
delete from 表名;          -- 清空表中记录
```

### 创建数据库表

> `CREATE TABLE 语法` 语句用于从表中选取数据。
>
> ```
> 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,
> ....
> );
> ```

```
-- 如果数据库中存在user_accounts表, 就把它从数据库中drop掉
DROP TABLE IF EXISTS `user_accounts`;
CREATE TABLE `user_accounts` (
   `id`                  int(100) unsigned NOT NULL AUTO_INCREMENT primary key,
   `password`          varchar(32)          NOT NULL DEFAULT '' COMMENT '用户密码',
```

```
    `reset_password` tinyint(32)          NOT NULL DEFAULT 0 COMMENT '用户类型：0-不需要重置密码；1-需要重置密码',
    `mobile`              varchar(20)          NOT NULL DEFAULT '' COMMENT '手机',
    `create_at`          timestamp(6)         NOT NULL DEFAULT CURRENT_TIMESTAMP(6),
    `update_at`          timestamp(6)         NOT NULL DEFAULT CURRENT_TIMESTAMP(6) ON UPDATE CURRENT_TIMESTAMP(6),
    -- 创建唯一索引，不允许重复
    UNIQUE INDEX idx_user_mobile(`mobile`)
)
ENGINE=InnoDB DEFAULT CHARSET=utf8
COMMENT='用户表信息';
```

数据类型的属性解释

- `NULL`：数据列可包含NULL值；
- `NOT NULL`：数据列不允许包含NULL值；
- `DEFAULT`：默认值；
- `PRIMARY KEY`：主键；
- `AUTO_INCREMENT`：自动递增，适用于整数类型；
- `UNSIGNED`：是指数值类型只能为正数；
- `CHARACTER SET name`：指定一个字符集；
- `COMMENT`：对表或者字段说明；

## 删除数据库表

> `DROP/TRUNCATE TABLE 语法` 语句用于删除数据库中的现有表。
>
> ```
> DROP TABLE 表名称;       -- 用于删除数据库中的现有表。
> TRUNCATE TABLE 表名称; -- 用于删除表内的数据，但不删除表本身。
> ```

```
-- 删除现有表 Shippers：
DROP TABLE Shippers;
-- 删除现有表 Shippers 表内的数据，不删除表：
TRUNCATE TABLE Shippers;
```

# 增删改查

## SELECT

> `SELECT 语法` 语句用于从表中选取数据。
>
> ```
> SELECT 列名称1, 列名称2, ... FROM 表名称;
> SELECT * FROM 表名称;
> ```

```
-- 从 Customers 表中选择 CustomerName 和 City 列：
SELECT CustomerName, City FROM Customers;
-- 从 Customers 表中选择所有列：
SELECT * FROM Customers;
-- 表 station 取个别名叫 s，表 station 中不包含 字段 id=13 或者 14 的，并且 id 不等于 4 的 查询出来，只显示 id
SELECT s.id from station s WHERE id in (13,14) and id not in (4);
```

```sql
-- 从表 users 选取 id=3 的数据，并只拉一条数据(据说能优化性能)
SELECT * FROM users where id=3 limit 1
-- 结果集中会自动去重复数据
SELECT DISTINCT Company FROM Orders
-- 表 Persons 字段 Id_P 等于 Orders 字段 Id_P 的值，
-- 结果集显示 Persons表的 LastName、FirstName字段，Orders表的OrderNo字段
SELECT p.LastName, p.FirstName, o.OrderNo FROM Persons p, Orders o WHERE p.Id_P = o.Id_P

-- gbk 和 utf8 中英文混合排序最简单的办法
-- ci是 case insensitive, 即 “大小写不敏感”
SELECT tag, COUNT(tag) from news GROUP BY tag order by convert(tag using gbk) collate gbk_chinese_ci;
SELECT tag, COUNT(tag) from news GROUP BY tag order by convert(tag using utf8) collate utf8_unicode_ci;
```

## UPDATE

Update 语法 语句用于修改表中的数据。

```sql
UPDATE 表名称 SET 列名称1 = 值1, 列名称2 = 值2, ... WHERE 条件;
```

```sql
-- update语句设置字段值为另一个结果取出来的字段
UPDATE user set name = (SELECT name from user1 WHERE user1 .id = 1 )
WHERE id = (SELECT id from user2 WHERE user2 .name='小苏');
-- 更新表 orders 中 id=1 的那一行数据更新它的 title 字段
UPDATE `orders` set title='这里是标题' WHERE id=1;
```

## INSERT

INSERT 语法 用于向表格中插入新的行。

```sql
INSERT INTO 表名称 (列名称1, 列名称2, 列名称3, ...) VALUES (值1, 值2, 值3, ...);
INSERT INTO 表名称 VALUES (值1, 值2, 值3, ...);
```

```sql
-- 向表 Persons 插入一条字段 LastName = JSLite 字段 Address = shanghai
INSERT INTO Persons (LastName, Address) VALUES ('JSLite', 'shanghai');
-- 向表 meeting 插入 字段 a=1 和字段 b=2
INSERT INTO meeting SET a=1,b=2;
--
-- SQL实现将一个表的数据插入到另外一个表的代码
-- 如果只希望导入指定字段，可以用这种方法：
-- INSERT INTO 目标表 (字段1, 字段2, ...) SELECT 字段1, 字段2, ... FROM 来源表;
INSERT INTO orders (user_account_id, title) SELECT m.user_id, m.title FROM meeting m where m.id=1;

-- 向表 charger 插入一条数据，已存在就对表 charger 更新 `type`,`update_at` 字段;
INSERT INTO `charger` (`id`,`type`,`create_at`,`update_at`) VALUES (3,2,'2017-05-18 11:06:17','2017-05-18 11:06:17') ON DUPLICATE KEY UPDATE `id`=VALUES(`id`), `type`=VALUES(`type`), `update_at`=VALUES(`update_at`);
```

### DELETE

`DELETE 语法` 语句用于删除表中的现有记录。

```
DELETE FROM 表名称 WHERE 条件;
```

```
-- 在不删除table_name表的情况下删除所有的行，清空表。
DELETE FROM table_name
-- 或者
DELETE * FROM table_name
-- 删除 Person 表字段 LastName = 'JSLite'
DELETE FROM Person WHERE LastName = 'JSLite'
-- 删除 表meeting id 为2和3的两条数据
DELETE from meeting where id in (2,3);
```

## WHERE

`WHERE 语法` 用于仅提取满足指定条件的记录

```
SELECT 列名称, 列名称, ... FROM 表名称 WHERE 条件1;
```

```
-- 从表 Persons 中选出 Year 字段大于 1965 的数据
SELECT * FROM Persons WHERE Year>1965
-- 从 Customers 表中选择 Country = Mexico 的所有数据：
SELECT * FROM Customers WHERE Country='Mexico';
-- 从 Customers 表中选择 CustomerID = 1 的所有数据：
SELECT * FROM Customers WHERE CustomerID=1;
```

## AND, OR 和 NOT

`WHERE` 子句可以与 `AND`、`OR` 和 `NOT` 运算符组合使用。

`AND` 和 `OR` 运算符用于根据多个条件过滤记录：

- 如果由 `AND` 分隔的所有条件都为 `TRUE`，则 `AND` 运算符将显示一条记录。
- 如果由 `OR` 分隔的任何条件为 `TRUE`，则 `OR` 运算符将显示一条记录。

如果条件不为真，`NOT` 运算符将显示一条记录。

### AND

`AND 语法`

```
SELECT 列名称, 列名称, ... FROM 表名称 WHERE 条件1 AND 条件2 AND 条件3 ...;
```

```sql
-- 删除 meeting 表字段
-- id=2 并且 user_id=5 的数据  和
-- id=3 并且 user_id=6 的数据
DELETE from meeting where id in (2,3) and user_id in (5,6);

-- 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';
```

### OR

> OR 语法
>
> ```sql
> SELECT 列名称1, 列名称2, ... FROM 表名称 WHERE 条件1 OR 条件2 OR 条件3 ...;
> ```

```sql
-- 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'
```

### NOT

> NOT 语法
>
> ```sql
> SELECT 列名称1, 列名称2, ... FROM 表名称 WHERE NOT 条件2;
> ```

```sql
-- 从 Customers 表中选择 Country 不是 Germany 的所有字段：
SELECT * FROM Customers WHERE NOT Country='Germany';
```

### AND & OR & NOT

```sql
-- 从 Customers 表中选择所有字段，其中 Country 为 Germany 且城市必须为 Berlin 或 München（使用括号构成复杂表达式）：
SELECT * FROM Customers WHERE Country='Germany' AND (City='Berlin' OR City='München');
-- 从 Customers 表中选择 Country 不是 Germany 和 NOT "USA" 的所有字段：
SELECT * FROM Customers WHERE NOT Country='Germany' AND NOT Country='USA';
```

## ORDER BY

> ORDER BY 语法 用于按升序或降序对结果集进行排序。
>
> ```sql
> SELECT 列名称1, 列名称2, ... FROM 表名称 ORDER BY 列名称1, 列名称2, ... ASC|DESC;
> ```
>
> 默认按 ASC 升序对记录进行排序。要按降序对记录进行排序，请使用 DESC 关键字。

```sql
-- 从 Customers 表中选择所有字段，按 Country 列排序：
SELECT * FROM Customers ORDER BY Country;
-- 从 Orders 表中选择 Company, OrderNumber 字段，按 Company 列排序：
SELECT Company, OrderNumber FROM Orders ORDER BY Company
-- 从 Orders 表中选择 Company, OrderNumber 字段，按 Company 列降序排序：
SELECT Company, OrderNumber FROM Orders ORDER BY Company DESC
-- 从 Orders 表中选择 Company, OrderNumber 字段，按 Company 列降序排序，并 OrderNumber 以顺序显示：
SELECT Company, OrderNumber FROM Orders ORDER BY Company DESC, OrderNumber ASC
```

## GROUP BY

`GROUP BY 语法` 将具有相同值的行分组到汇总行中

```sql
SELECT 列名称(s)
FROM 表名称
WHERE 条件
GROUP BY 列名称(s)
ORDER BY 列名称(s);
```

```sql
-- 列出了 Orders 每个发货人 Shippers 发送的订单 Orders 数量
SELECT Shippers.ShipperName, COUNT(Orders.OrderID) AS NumberOfOrders FROM Orders
LEFT JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID
GROUP BY ShipperName;
```

## IN

`IN 语法` 运算符允许您在 WHERE 子句中指定多个值。运算符是多个 OR 条件的简写。

```sql
SELECT 列名称(s) FROM 表名称 WHERE 列名称 IN (值1, 值2, ...);
SELECT 列名称(s) FROM 表名称 WHERE 列名称 IN (SELECT STATEMENT);
```

```sql
-- 从表 Persons 选取 字段 LastName 等于 Adams、Carter
SELECT * FROM Persons WHERE LastName IN ('Adams','Carter')
-- 从表 Customers 选取 Country 值为 'Germany', 'France', 'UK' 的所有数据
SELECT * FROM Customers WHERE Country IN ('Germany', 'France', 'UK');
-- 从表 Customers 选取 Country 值不为 'Germany', 'France', 'UK' 的所有数据
SELECT * FROM Customers WHERE Country NOT IN ('Germany', 'France', 'UK');
-- 从表 Customers 选取与 Suppliers 表 Country 字段相同的所有数据：
SELECT * FROM Customers WHERE Country IN (SELECT Country FROM Suppliers);
```

## UNION

`UNION 语法` 操作符用于合并两个或多个 SELECT 语句的结果集

```
SELECT 列名称(s) FROM 表名称1
UNION
SELECT 列名称(s) FROM 表名称2;
```

```
-- 列出所有在中国表（Employees_China）和美国（Employees_USA）的不同的雇员名
SELECT E_Name FROM Employees_China UNION SELECT E_Name FROM Employees_USA

-- 列出 meeting 表中的 pic_url，
-- station 表中的 number_station 别名设置成 pic_url 避免字段不一样报错
-- 按更新时间排序
SELECT id,pic_url FROM meeting UNION ALL SELECT id,number_station AS pic_url FROM station  ORDER BY update_at;
-- 通过 UNION 语法同时查询了 products 表 和 comments 表的总记录数，并且按照 count 排序
SELECT 'product' AS type, count(*) as count FROM `products` union select 'comment' as type, count(*) as count FROM `comments` order by count;
```

## BETWEEN

BETWEEN 语法 运算符选择给定范围内的值

```
SELECT 列名称(s) FROM 表名称 WHERE 列名称 BETWEEN 值1 AND 值2;
```

```
-- 选择 Products 表中 Price 字段在 10 到 20 之间的所有：
SELECT * FROM Products WHERE Price BETWEEN 10 AND 20;
```

## AS

AS 语法 用于为表或表中的列(字段)提供临时名称(别名)。

```
SELECT 列名称 AS 别名 FROM 表名称;
SELECT 列名称(s) FROM 表名称 AS 别名;
```

```
-- 创建两个别名，一个用于 CustomerID 的 ID 别名列，一个用于 CustomerName  的 Customer 别名列：
SELECT CustomerID AS ID, CustomerName AS Customer FROM Customers;

-- 这句意思是查找所有 Employee 表里面的数据，并把 Employee 表格命名为 emp。
-- 当你命名一个表之后，你可以在下面用 emp 代替 Employee.
-- 例如 SELECT * FROM emp.
SELECT * FROM Employee AS emp

-- 列出表 Orders 字段 OrderPrice 列最大值，
-- 结果集列不显示 OrderPrice 显示 LargestOrderPrice
SELECT MAX(OrderPrice) AS LargestOrderPrice FROM Orders

-- 显示表 users_profile 中的 name 列
SELECT t.name from (SELECT * from users_profile a) AS t;

-- 表 user_accounts 命名别名 ua，表 users_profile 命名别名 up
```

```sql
-- 满足条件 表 user_accounts 字段 id 等于 表 users_profile 字段 user_id
-- 结果集只显示mobile、name两列
SELECT ua.mobile,up.name FROM user_accounts as ua INNER JOIN users_profile as up ON ua.id = up.user_id;
```

# JOIN

JOIN 子句用于根据两个或多个表之间的相关列组合来自两个或多个表的行。

- `JOIN`：如果表中有至少一个匹配，则返回行
- `INNER JOIN`:在表中存在至少一个匹配时，INNER JOIN 关键字返回行。
- `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(MySQL 是不支持的，通过 `LEFT JOIN + UNION + RIGHT JOIN` 的方式 来实现)

## INNER JOIN

> `INNER JOIN 语法` 选择在两个表中具有匹配值的记录。
>
> ```sql
> SELECT 列名称(s)
> FROM 表1
> INNER JOIN 表2
> ON 表1.列名称 = 表2.列名称;
> ```

```sql
-- 选择包含 Customers 的所有 Orders:
SELECT Orders.OrderID, Customers.CustomerName FROM Orders INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;

-- [JOIN 三张表] 选择包含 Customers 和 Shippers 的所有 Orders:
SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName
FROM ((Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID)
INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);
```

## LEFT JOIN

> `LEFT JOIN 语法` 返回左表 (表1) 中的所有记录，以及右表 (表2) 中的匹配记录
>
> ```sql
> SELECT 列名称(s)
> FROM 表1
> LEFT JOIN 表2
> ON 表1.列名称 = 表2.列名称;
> ```

```sql
-- 将选择所有 Customers 以及他们可能拥有的任何 Orders:
SELECT Customers.CustomerName, Orders.OrderID FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;
```

### RIGHT JOIN

`RIGHT JOIN 语法` 返回右表 (表2) 中的所有记录，以及左表 (表1) 中的匹配记录

```
SELECT 列名称(s)
FROM 表1
RIGHT JOIN 表2
ON 表1.列名称 = 表2.列名称;
```

```
-- 返回所有 Employees 以及他们可能下的任何 Orders:
SELECT Orders.OrderID, Employees.LastName, Employees.FirstName FROM Orders
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Orders.OrderID;
```

### FULL OUTER JOIN

`FULL OUTER JOIN 语法` 当左（表1）或右（表2）表记录中存在匹配时，关键字返回所有记录

```
SELECT 列名称(s)
FROM 表1
FULL OUTER JOIN 表2
ON 表1.列名称 = 表2.列名称
WHERE 条件;
```

## SQL 函数

### COUNT

`COUNT 语法` 返回与指定条件匹配的行数

```
SELECT COUNT(列名称) FROM 表名称 WHERE 条件;
```

```
-- 表 Store_Information 有几笔 store_name 栏不是空白的资料。
-- "IS NOT NULL" 是 "这个栏位不是空白" 的意思。
SELECT COUNT (Store_Name) FROM Store_Information WHERE Store_Name IS NOT NULL;
-- 获取 Persons 表的总数
SELECT COUNT(1) AS totals FROM Persons;
-- 获取表 station 字段 user_id 相同的总数
select user_id, count(*) as totals from station group by user_id;
```

### AVG

`AVG 语法` 返回数值列的平均值

```
SELECT AVG(列名称) FROM 表名称 WHERE 条件;
```

```sql
-- 查找 Products 表中所的 Price 平均值：
SELECT AVG(Price) FROM Products;
```

### SUM

SUM 语法 返回数值列的总和

```sql
SELECT SUM(列名称) FROM 表名称 WHERE 条件;
```

```sql
-- 查找 OrderDetails 表中 Quantity 字段的总和：
SELECT SUM(Quantity) FROM OrderDetails;
```

### MAX

MAX 语法 返回所选列的最大值

```sql
SELECT MIN(列名称) FROM 表名称 WHERE 条件;
```

```sql
-- 列出表 Orders 字段 OrderPrice 列最大值，
-- 结果集列不显示 OrderPrice 显示 LargestOrderPrice
SELECT MAX(OrderPrice) AS LargestOrderPrice FROM Orders
```

### MIN

MIN 语法 返回所选列的最小值

```sql
SELECT MIN(列名称) FROM 表名称 WHERE 条件;
```

```sql
-- 查找 Products 表中 Price 字段最小值，并命名 SmallestPrice 别名：
SELECT MIN(Price) AS SmallestPrice FROM Products;
```

## 触发器

语法:

```sql
create trigger <触发器名称>
{ before | after}            -- 之前或者之后出发
insert | update | delete   -- 指明了激活触发程序的语句的类型
on <表名>                          -- 操作哪张表
for each row                    -- 触发器的执行间隔，for each row 通知触发器每隔一行执行一次动作，而不是对整个表执行一次。
<触发器SQL语句>
```

```
delimiter $
CREATE TRIGGER set_userdate BEFORE INSERT
on `message`
for EACH ROW
BEGIN
   set @statu = new.status; -- 声明复制变量 statu
   if @statu = 0 then          -- 判断 statu 是否等于 0
      UPDATE `user_accounts` SET status=1 WHERE openid=NEW.openid;
   end if;
END
$
DELIMITER ; -- 恢复结束符号
```

OLD和NEW不区分大小写

- NEW 用NEW.col_name，没有旧行。在DELETE触发程序中，仅能使用OLD.col_name，没有新行。
- OLD 用OLD.col_name来引用更新前的某一行的列

# 添加索引

## 普通索引(INDEX)

> 语法：ALTER TABLE `表名字` ADD INDEX 索引名字 ( `字段名字` )

```
-- -直接创建索引
CREATE INDEX index_user ON user(title)
-- -修改表结构的方式添加索引
ALTER TABLE table_name ADD INDEX index_name ON (column(length))
-- 给 user 表中的 name 字段 添加普通索引(INDEX)
ALTER TABLE `user` ADD INDEX index_name (name)
-- -创建表的时候同时创建索引
CREATE TABLE `user` (
      `id` int(11) NOT NULL AUTO_INCREMENT ,
      `title` char(255) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL ,
      `content` text CHARACTER SET utf8 COLLATE utf8_general_ci NULL ,
      `time` int(10) NULL DEFAULT NULL ,
      PRIMARY KEY (`id`),
      INDEX index_name (title(length))
)
-- -删除索引
DROP INDEX index_name ON table
```

## 主键索引(PRIMARY key)

> 语法：ALTER TABLE `表名字` ADD PRIMARY KEY ( `字段名字` )

```
-- 给 user 表中的 id字段 添加主键索引(PRIMARY key)
ALTER TABLE `user` ADD PRIMARY key (id);
```

## 唯一索引(UNIQUE)

> 语法：ALTER TABLE `表名字` ADD UNIQUE ( `字段名字` )

```
-- 给 user 表中的 creattime 字段添加唯一索引(UNIQUE)
ALTER TABLE `user` ADD UNIQUE (creattime);
```

## 全文索引(FULLTEXT)

> 语法：ALTER TABLE `表名字` ADD FULLTEXT ( `字段名字` )

```
-- 给 user 表中的 description 字段添加全文索引(FULLTEXT)
ALTER TABLE `user` ADD FULLTEXT (description);
```

## 添加多列索引

> 语法：
> ALTER TABLE `table_name` ADD INDEX index_name ( `column1` , `column2` , `column3` )

```
-- 给 user 表中的 name、city、age 字段添加名字为name_city_age的普通索引(INDEX)
ALTER TABLE user ADD INDEX name_city_age (name(10),city,age);
```

## 建立索引的时机

在 `WHERE` 和 `JOIN` 中出现的列需要建立索引，但也不完全如此：

- MySQL只对 `<` , `<=` , `=` , `>` , `>=` , `BETWEEN` , `IN` 使用索引
- 某些时候的 `LIKE` 也会使用索引。
- 在 `LIKE` 以通配符%和_开头作查询时，MySQL不会使用索引。

```
-- 此时就需要对city和age建立索引，
-- 由于mytable表的userame也出现在了JOIN子句中，也有对它建立索引的必要。
SELECT t.Name
FROM mytable t LEFT JOIN mytable m ON t.Name=m.username
WHERE m.age=20 AND m.city='上海';

SELECT * FROM mytable WHERE username like'admin%'; -- 而下句就不会使用：
SELECT * FROM mytable WHERE Name like'%admin'; -- 因此，在使用LIKE时应注意以上的区别。
```

索引的注意事项

- 索引不会包含有NULL值的列
- 使用短索引
- 不要在列上进行运算 索引会失效

# 创建后表的修改

## 添加列

> 语法：`alter table 表名 add 列名 列数据类型 [after 插入位置];`

示例:

```
-- 在表students的最后追加列 address:
alter table students add address char(60);
-- 在名为 age 的列后插入列 birthday:
alter table students add birthday date after age;
-- 在名为 number_people 的列后插入列 weeks:
alter table students add column `weeks` varchar(5) not null default "" after `number_people`;
```

## 修改列

> 语法：`alter table 表名 change 列名称 列新名称 新数据类型;`

```
-- 将表 tel 列改名为 telphone:
alter table students change tel telphone char(13) default "-";
-- 将 name 列的数据类型改为 char(16):
alter table students change name name char(16) not null;
-- 修改 COMMENT 前面必须得有类型属性
alter table students change name name char(16) COMMENT '这里是名字';
-- 修改列属性的时候 建议使用modify,不需要重建表
-- change用于修改列名字，这个需要重建表
alter table meeting modify `weeks` varchar(20) NOT NULL DEFAULT '' COMMENT '开放日期 周一到周日：0~6，间隔用英文逗号隔开';
-- `user`表的`id`列，修改成字符串类型长度50，不能为空，`FIRST`放在第一列的位置
alter table `user` modify COLUMN `id` varchar(50) NOT NULL FIRST ;
```

## 删除列

> 语法：`alter table 表名 drop 列名称;`

```
-- 删除表students中的 birthday 列:
alter table students drop birthday;
```

## 重命名表

> 语法：`alter table 表名 rename 新表名;`

```
-- 重命名 students 表为 workmates:
alter table students rename workmates;
```

## 清空表数据

> 方法一：`delete from 表名;`
> 方法二：`truncate table "表名";`

- `DELETE:` 1. DML语言;2. 可以回退;3. 可以有条件的删除;
- `TRUNCATE:` 1. DDL语言;2. 无法回退;3. 默认所有的表内容都删除;4. 删除速度比delete快。

```sql
-- 清空表为 workmates 里面的数据，不删除表。
delete from workmates;
-- 删除workmates表中的所有数据，且无法恢复
truncate table workmates;
```

### 删除整张表

> 语法：`drop table 表名;`

```sql
-- 删除 workmates 表:
drop table workmates;
```

### 删除整个数据库

> 语法：`drop database 数据库名;`

```sql
-- 删除 samp_db 数据库:
drop database samp_db;
```

## 其它实例

### SQL删除重复记录

**转载**

```sql
-- 查找表中多余的重复记录，重复记录是根据单个字段（peopleId）来判断
select * from people where peopleId in (select peopleId from people group by peopleId having count(peopleId) > 1)
-- 删除表中多余的重复记录，重复记录是根据单个字段（peopleId）来判断，只留有rowid最小的记录
delete from people
where peopleId in (select peopleId from people group by peopleId having count(peopleId) > 1)
and rowid not in (select min(rowid) from people group by peopleId having count(peopleId )>1)
-- 查找表中多余的重复记录（多个字段）
select * from vitae a
where (a.peopleId,a.seq) in (select peopleId,seq from vitae group by peopleId,seq having count(*) > 1)
-- 删除表中多余的重复记录（多个字段），只留有rowid最小的记录
delete from vitae a
where (a.peopleId,a.seq) in (select peopleId,seq from vitae group by peopleId,seq having count(*) > 1) and rowid not in (select min(rowid) from vitae group by peopleId,seq having count(*)>1)
-- 查找表中多余的重复记录（多个字段），不包含rowid最小的记录
select * from vitae a
where (a.peopleId,a.seq) in (select peopleId,seq from vitae group by peopleId,seq having count(*) > 1) and rowid not in (select min(rowid) from vitae group by peopleId,seq having count(*)>1)
```

## 参考手册

- http://www.w3school.com.cn/sql/index.asp
- http://www.1keydata.com/cn/sql/sql-count.php