

# SQL教程

---

## 目录

- 开始使用
  - 登录MySQL
  - 创建数据库
  - 创建数据库表
- 增删改查
  - SELECT
  - UPDATE
  - INSERT
  - DELETE
- WHERE
- AND 和 OR
  - AND
  - OR
- ORDER BY
- IN
- NOT
- UNION
- AS
- JOIN
- SQL 函数
  - COUNT
  - MAX
- 触发器
- 添加索引
  - 普通索引(INDEX)
  - 主键索引(PRIMARY key)
  - 唯一索引(UNIQUE)
  - 全文索引(FULLTEXT)
  - 添加多列索引
  - 建立索引的时机
- 创建后表的修改
  - 添加列
  - 修改列
  - 删除列
  - 重命名表
  - 清空表数据

- 删除整张表
- 删除整个数据库
- 其它实例
  - SQL删除重复记录
- 参考手册

## 开始使用

我下面所有的SQL语句是基于MySQL 5.6+运行。

MySQL 为关系型数据库(Relational Database Management System), 一个关系型数据库由一个或数个表格组成, 如图所示的一个表格:

The diagram shows a table with the following structure and annotations:

id	name	parent_id	level
1	上海市	0	1
2	北京市	0	1
3	湖北省	0	2
4	江苏省	0	1
5	重庆市	0	2

Annotations in the diagram:

- 键(key)**: Points to the 'id' column.
- 表头(header)**: Points to the first row of the table.
- 列(col)**: Points to the 'level' column.
- 行(row)**: Points to the third row of the table.
- 值(value)**: Points to the 'name' column.

- 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
```

```
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

使用 create table 语句可完成对表的创建, create table 的常见形式: 语法: create table 表名称(列声明);

-- 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；

## 增删改查

### SELECT

SELECT 语句用于从表中选取数据。

语法：SELECT 列名称 FROM 表名称

语法：SELECT \* FROM 表名称

-- 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);
```

-- 从表 Persons 选取 LastName 列的数据

```
SELECT LastName FROM Persons
```

-- 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

-- 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 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,...)

语法: INSERT INTO 表名称 (列1, 列2,...) VALUES (值1, 值2,...)

-- 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 语句用于删除表中的行。

语法: DELETE FROM 表名称 WHERE 列名称 = 值

-- 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 子句用于规定选择的标准。

语法：SELECT 列名称 FROM 表名称 WHERE 列 运算符 值

-- 从表 Persons 中选出 Year 字段大于 1965 的数据

```
SELECT * FROM Persons WHERE Year>1965
```

## AND 和 OR

AND – 如果第一个条件和第二个条件都成立；

OR – 如果第一个条件和第二个条件中只要有一个成立；

## AND

-- 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

-- 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'
```

## ORDER BY

语句默认按照升序对记录进行排序。

ORDER BY – 语句用于根据指定的列对结果集进行排序。

DESC – 按照降序对记录进行排序。

ASC – 按照顺序对记录进行排序。

-- Company在表Orders中为字母，则会以字母顺序显示公司名称

```
SELECT Company, OrderNumber FROM Orders ORDER BY Company
```

-- 后面跟上 DESC 则为降序显示

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

-- Company以降序显示公司名称，并OrderNumber以顺序显示

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

## IN

IN – 操作符允许我们在 WHERE 子句中规定多个值。

IN – 操作符用来指定范围，范围中的每一条，都进行匹配。IN取值规律，由逗号分割，全部放置括号中。语法：SELECT "字段名"FROM "表格名"WHERE "字段名" IN ('值一', '值二', ...);

-- 从表 Persons 选取 字段 LastName 等于 Adams、Carter

```
SELECT * FROM Persons WHERE LastName IN ('Adams','Carter')
```

## NOT

NOT – 操作符总是与其他操作符一起使用，用在要过滤的前面。

```
SELECT vend_id, prod_name FROM Products WHERE NOT vend_id = 'DLL01'  
ORDER BY prod_name;
```

## UNION

UNION – 操作符用于合并两个或多个 SELECT 语句的结果集。

-- 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;
```

## AS

as – 可理解为：用作、当成，作为；别名  
一般是重命名列名或者表名。

语法：select column\_1 as 列1,column\_2 as 列2 from table as 表

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

-- 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  
-- 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

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: 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)



```
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
FROM Persons
INNER JOIN Orders
ON Persons.Id_P = Orders.Id_P
ORDER BY Persons.LastName;
```

## SQL 函数

### COUNT

COUNT 让我们能够数出在表格中有多少笔资料被选出来。

语法：SELECT COUNT("字段名") FROM "表格名";

-- 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;
```

### MAX

MAX 函数返回一列中的最大值。NULL 值不包括在计算中。

语法：SELECT MAX("字段名") FROM "表格名"

-- 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
```

## 触发器

语法：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 )

-- 一直接创建索引
CREATE INDEX index_user ON user(title)

-- 一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)

-- 一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))
)

-- 一删除索引
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username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 telephone:

```
alter table students change tel telephone 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 from "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。

-- 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;

-- 删除 workmates 表:

```
drop table workmates;
```

## 删除整个数据库

语法：drop database 数据库名;

-- 删除 samp\_db 数据库:

```
drop database samp_db;
```