

SQL 语法速成手册



静默虚空 LV.5

2019年03月05日 18:49 · 阅读 14351

关注

本文针对关系型数据库的一般语法。限于篇幅，本文侧重说明用法，不会展开讲解特性、原理。

一、基本概念

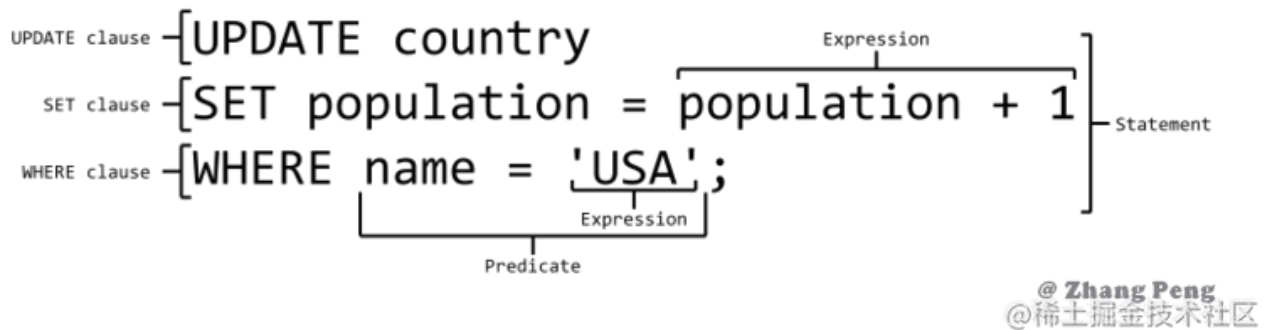
数据库术语

- **数据库 (database)** - 保存有组织的数据的容器（通常是一个文件或一组文件）。
- **数据表 (table)** - 某种特定类型数据的结构化清单。
- **模式 (schema)** - 关于数据库和表的布局及特性的信息。模式定义了数据在表中如何存储，包含存储什么样的数据，数据如何分解，各部分信息如何命名等信息。数据库和表都有模式。
- **列 (column)** - 表中的一个字段。所有表都是由一个或多个列组成的。
- **行 (row)** - 表中的一个记录。
- **主键 (primary key)** - 一列（或一组列），其值能够唯一标识表中每一行。

SQL 语法

SQL (Structured Query Language)，标准 SQL 由 ANSI 标准委员会管理，从而称为 ANSI SQL。各个 DBMS 都有自己的实现，如 PL/SQL、Transact-SQL 等。

SQL 语法结构



SQL 语法结构包括：

- **子句** - 是语句和查询的组成成分。（在某些情况下，这些都是可选的。）
- **表达式** - 可以产生任何标量值，或由列和行的数据库表
- **谓词** - 给需要评估的 SQL 三值逻辑（3VL）（true/false/unknown）或布尔真值指定条件，并限制语句和查询的效果，或改变程序流程。
- **查询** - 基于特定条件检索数据。这是 SQL 的一个重要组成部分。
- **语句** - 可以持久地影响纲要和数据，也可以控制数据库事务、程序流程、连接、会话或诊断。

SQL 语法要点

- **SQL 语句不区分大小写**，但是数据库表名、列名和值是否区分，依赖于具体的 DBMS 以及配置。

例如：**SELECT** 与 **select**、**Select** 是相同的。

- **多条 SQL 语句必须以分号（ ; ）分隔。**
- 处理 SQL 语句时，**所有空格都被忽略**。SQL 语句可以写成一行，也可以分写为多行。

sql 复制代码

-- 一行 SQL 语句

```
UPDATE user SET username='robot', password='robot' WHERE username = 'root';
```

-- 多行 SQL 语句

```
UPDATE user
SET username='robot', password='robot'
WHERE username = 'root';
```

- SQL 支持三种注释

```
## 注释1  
-- 注释2  
/* 注释3 */
```

SQL 分类

数据定义语言（DDL）

数据定义语言（Data Definition Language, DDL）是 SQL 语言集中负责数据结构定义与数据库对象定义的语言。

DDL 的主要功能是定义数据库对象。

DDL 的核心指令是 **CREATE**、**ALTER**、**DROP**。

数据操纵语言（DML）

数据操纵语言（Data Manipulation Language, DML）是用于数据库操作，对数据库其中的对象和数据运行访问工作的编程语句。

DML 的主要功能是访问数据，因此其语法都是以读写数据库为主。

DML 的核心指令是 **INSERT**、**UPDATE**、**DELETE**、**SELECT**。这四个指令合称 CRUD(Create, Read, Update, Delete)，即增删改查。

事务控制语言（TCL）

事务控制语言 (Transaction Control Language, TCL) 用于管理数据库中的事务。这些用于管理由 DML 语句所做的更改。它还允许将语句分组为逻辑事务。

TCL 的核心指令是 **COMMIT**、**ROLLBACK**。

数据控制语言（DCL）

数据控制语言 (Data Control Language, DCL) 是一种可对数据访问权进行控制的指令，它可以控制特定用户账户对数据表、查看表、预存程序、用户自定义函数等数据库对象的控制权。

DCL 的核心指令是 **GRANT** 、 **REVOKE** 。

DCL 以控制用户的访问权限为主，因此其指令作法并不复杂，可利用 DCL 控制的权限有：
CONNECT 、 **SELECT** 、 **INSERT** 、 **UPDATE** 、 **DELETE** 、 **EXECUTE** 、 **USAGE** 、 **REFERENCES** 。

根据不同的 DBMS 以及不同的安全性实体，其支持的权限控制也有所不同。

(以下为 DML 语句用法)

二、增删改查

增删改查，又称为 CRUD，数据库基本操作中的基本操作。

插入数据

- **INSERT INTO** 语句用于向表中插入新记录。

插入完整的行

```
INSERT INTO user
VALUES (10, 'root', 'root', 'xxxx@163.com');
```

sql 复制代码

插入行的一部分

```
INSERT INTO user(username, password, email)
VALUES ('admin', 'admin', 'xxxx@163.com');
```

sql 复制代码

插入查询出来的数据

```
INSERT INTO user(username)
SELECT name
FROM account;
```

sql 复制代码

更新数据

- **UPDATE** 语句用于更新表中的记录。

[ini 复制代码](#)

```
UPDATE user
SET username='robot', password='robot'
WHERE username = 'root';
```

删除数据

- **DELETE** 语句用于删除表中的记录。
- **TRUNCATE TABLE** 可以清空表，也就是删除所有行。

删除表中的指定数据

[sql 复制代码](#)

```
DELETE FROM user
WHERE username = 'robot';
```

清空表中的数据

[sql 复制代码](#)

```
TRUNCATE TABLE user;
```

查询数据

- **SELECT** 语句用于从数据库中查询数据。
- **DISTINCT** 用于返回唯一不同的值。它作用于所有列，也就是说所有列的值都相同才算相同。
- **LIMIT** 限制返回的行数。可以有两个参数，第一个参数为起始行，从 0 开始；第二个参数为返回的总行数。
 - **ASC** : 升序（默认）
 - **DESC** : 降序

查询单列

[sql 复制代码](#)

```
SELECT prod_name
FROM products;
```

查询多列

[sql 复制代码](#)

```
SELECT prod_id, prod_name, prod_price
FROM products;
```

查询所有列

[css 复制代码](#)

```
ELECT *
FROM products;
```

查询不同的值

[sql 复制代码](#)

```
SELECT DISTINCT
vend_id FROM products;
```

限制查询结果

[sql 复制代码](#)

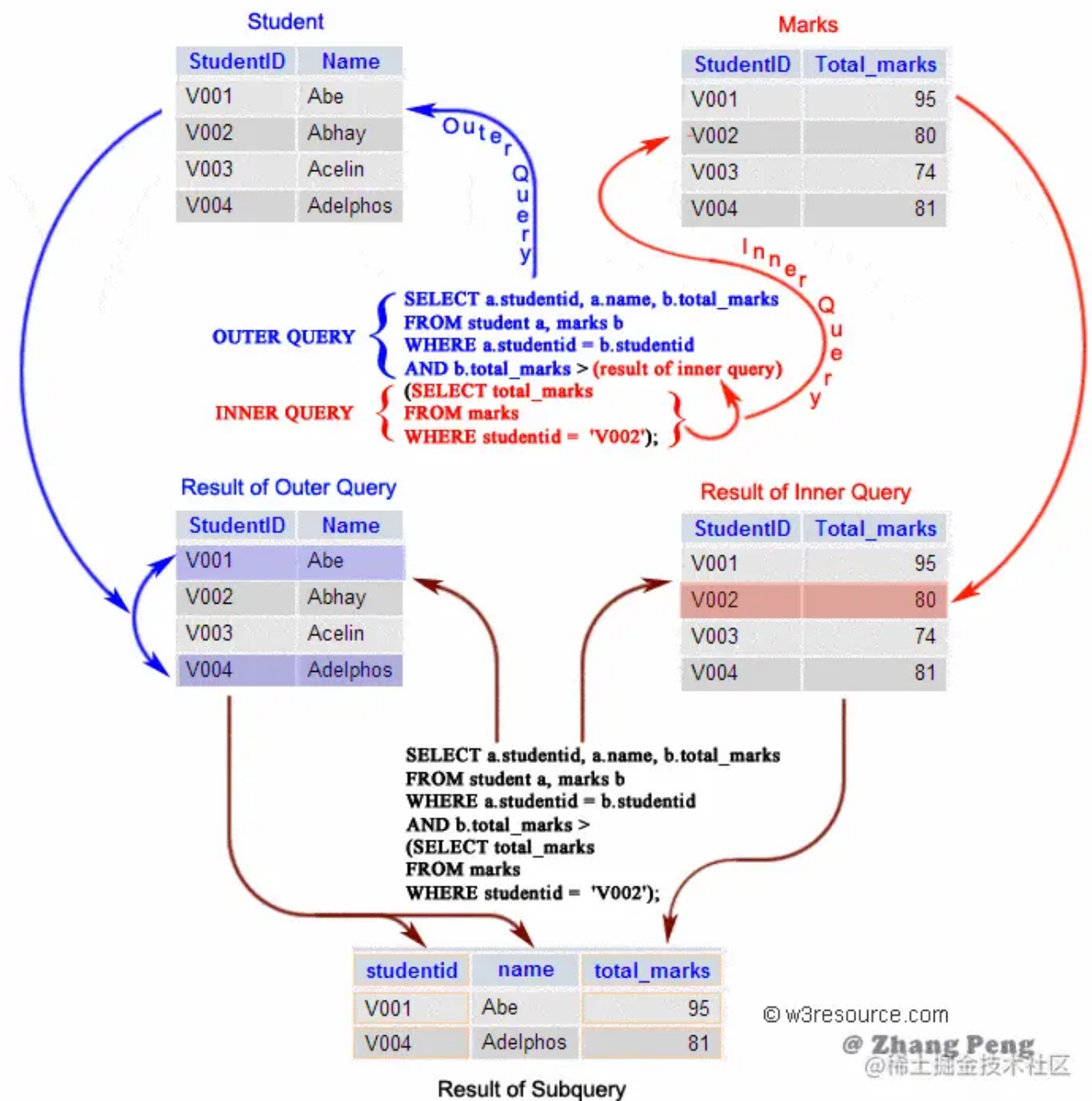
```
-- 返回前 5 行
SELECT * FROM mytable LIMIT 5;
SELECT * FROM mytable LIMIT 0, 5;
-- 返回第 3 ~ 5 行
SELECT * FROM mytable LIMIT 2, 3;
```

三、子查询

子查询是嵌套在较大查询中的 SQL 查询。子查询也称为**内部查询**或**内部选择**，而包含子查询的语句也称为**外部查询**或**外部选择**。

- 子查询可以嵌套在 **SELECT** , **INSERT** , **UPDATE** 或 **DELETE** 语句内或另一个子查询中。
- 子查询通常会在另一个 **SELECT** 语句的 **WHERE** 子句中添加。
- 您可以使用比较运算符, 如 **>** , **<** , 或 **=** 。比较运算符也可以是多行运算符, 如 **IN** , **ANY** 或 **ALL** 。
- 子查询必须被圆括号 **()** 括起来。

- 内部查询首先在其父查询之前执行，以便可以将内部查询的结果传递给外部查询。执行过程可以参考下图：



子查询的子查询

sql 复制代码

```
SELECT cust_name, cust_contact
FROM customers
WHERE cust_id IN (SELECT cust_id
                  FROM orders
                  WHERE order_num IN (SELECT order_num
                                      FROM orderitems
                                      WHERE prod_id = 'RGAN01'));
```

WHERE

- **WHERE** 子句用于过滤记录，即缩小访问数据的范围。
- **WHERE** 后跟一个返回 **true** 或 **false** 的条件。
- **WHERE** 可以与 **SELECT**，**UPDATE** 和 **DELETE** 一起使用。
- 可以在 **WHERE** 子句中使用的操作符

运算符	描述
=	等于
<>	不等于。注释：在 SQL 的一些版本中，该操作符可被写成 !=
>	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
IN	指定针对某个列的多个可能值

SELECT 语句中的 WHERE 子句

[ini 复制代码](#)

```
SELECT * FROM Customers
WHERE cust_name = 'Kids Place';
```

UPDATE 语句中的 WHERE 子句

[ini 复制代码](#)

```
UPDATE Customers
SET cust_name = 'Jack Jones'
WHERE cust_name = 'Kids Place';
```

DELETE 语句中的 WHERE 子句


```
DELETE FROM Customers
WHERE cust_name = 'Kids Place';
```

IN 和 BETWEEN

- **IN** 操作符在 **WHERE** 子句中使用，作用是在指定的几个特定值中任选一个值。
- **BETWEEN** 操作符在 **WHERE** 子句中使用，作用是选取介于某个范围内的值。

IN 示例

sql 复制代码

```
SELECT *
FROM products
WHERE vend_id IN ('DLL01', 'BRS01');
```

BETWEEN 示例

sql 复制代码

```
SELECT *
FROM products
WHERE prod_price BETWEEN 3 AND 5;
```

AND、OR、NOT

- **AND**、**OR**、**NOT** 是用于对过滤条件的逻辑处理指令。
- **AND** 优先级高于 **OR**，为了明确处理顺序，可以使用 **()**。
- **AND** 操作符表示左右条件都要满足。
- **OR** 操作符表示左右条件满足任意一个即可。
- **NOT** 操作符用于否定一个条件。

AND 示例

ini 复制代码

```
SELECT prod_id, prod_name, prod_price
FROM products
WHERE vend_id = 'DLL01' AND prod_price <= 4;
```

OR 示例

ini 复制代码

```
SELECT prod_id, prod_name, prod_price
FROM products
```

```
WHERE vend_id = 'DLL01' OR vend_id = 'BRS01';
```

NOT 示例

[sql 复制代码](#)

```
SELECT *  
FROM products  
WHERE prod_price NOT BETWEEN 3 AND 5;
```

LIKE

- **LIKE** 操作符在 **WHERE** 子句中使用，作用是确定字符串是否匹配模式。
- 只有字段是文本值时才使用 **LIKE**。
- **LIKE** 支持两个通配符匹配选项：**%** 和 **_**。
- 不要滥用通配符，通配符位于开头处匹配会非常慢。
- **%** 表示任何字符出现任意次数。
- **_** 表示任何字符出现一次。

% 示例

[sql 复制代码](#)

```
SELECT prod_id, prod_name, prod_price  
FROM products  
WHERE prod_name LIKE '%bean bag%';
```

_ 示例

[sql 复制代码](#)

```
SELECT prod_id, prod_name, prod_price  
FROM products  
WHERE prod_name LIKE '__ inch teddy bear';
```

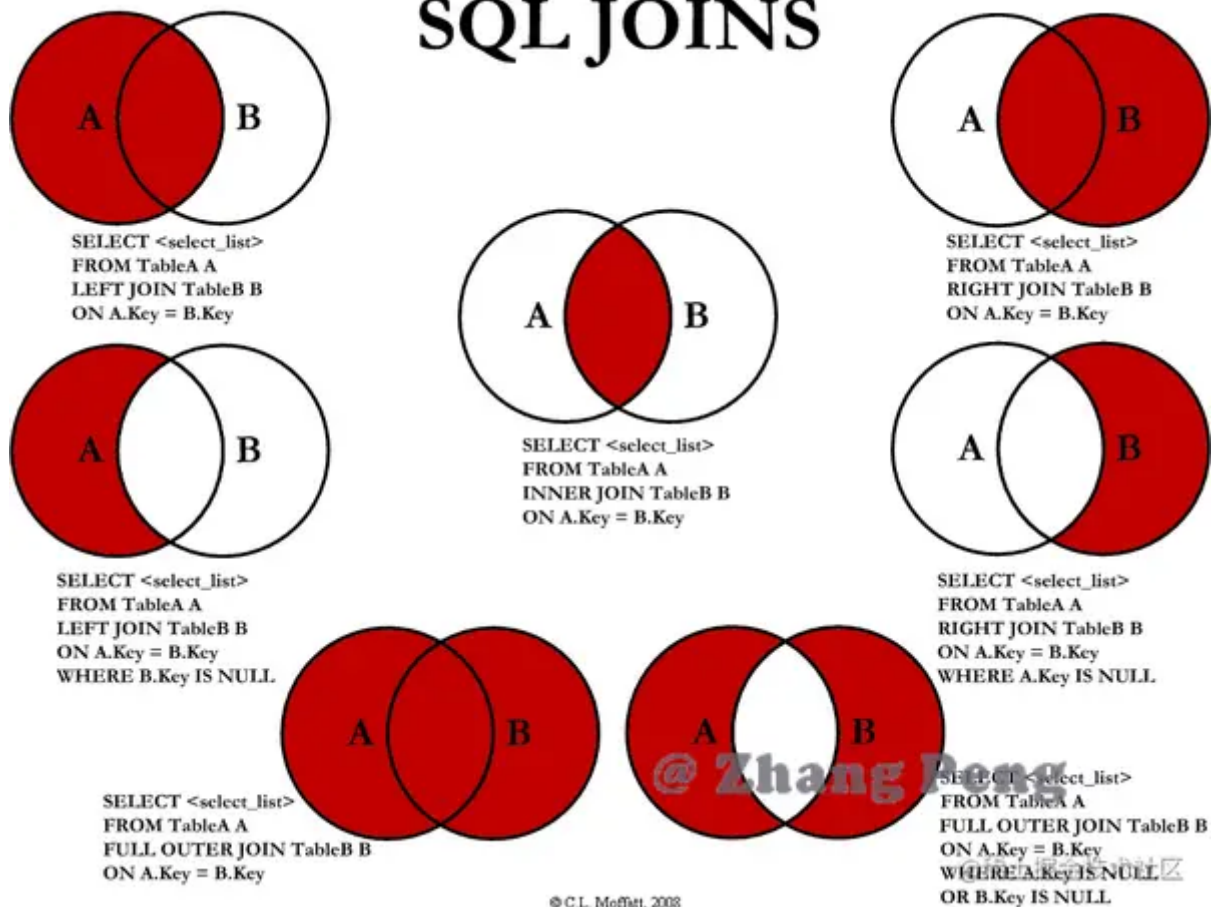
四、连接和组合

连接 (JOIN)

- 如果一个 **JOIN** 至少有一个公共字段并且它们之间存在关系，则该 **JOIN** 可以在两个或多个表上工作。
- 连接用于连接多个表，使用 **JOIN** 关键字，并且条件语句使用 **ON** 而不是 **WHERE**。

- **JOIN** 保持基表（结构和数据）不变。
- **JOIN** 有两种连接类型：内连接和外连接。
- 内连接又称等值连接，使用 **INNER JOIN** 关键字。在没有条件语句的情况下返回笛卡尔积。
 - 自连接可以看成内连接的一种，只是连接的表是自身而已。
- 自然连接是把同名列通过 = 测试连接起来的，同名列可以有多个。
- 内连接 vs 自然连接
 - 内连接提供连接的列，而自然连接自动连接所有同名列。
- 外连接返回一个表中的所有行，并且仅返回来自次表中满足连接条件的那些行，即两个表中的列是相等的。外连接分为左外连接、右外连接、全外连接（Mysql 不支持）。
 - 左外连接就是保留左表没有关联的行。
 - 右外连接就是保留右表没有关联的行。
- 连接 vs 子查询
 - 连接可以替换子查询，并且比子查询的效率一般会更快。

SQL JOINS



内连接 (INNER JOIN)

[sql 复制代码](#)

```
SELECT vend_name, prod_name, prod_price
FROM vendors INNER JOIN products
ON vendors.vend_id = products.vend_id;
```

自连接

[ini 复制代码](#)

```
SELECT c1.cust_id, c1.cust_name, c1.cust_contact
FROM customers c1, customers c2
WHERE c1.cust_name = c2.cust_name
AND c2.cust_contact = 'Jim Jones';
```

自然连接 (NATURAL JOIN)

[sql 复制代码](#)

```
SELECT *
FROM Products
NATURAL JOIN Customers;
```

左连接 (LEFT JOIN)

[sql 复制代码](#)

```
SELECT customers.cust_id, orders.order_num
FROM customers LEFT JOIN orders
ON customers.cust_id = orders.cust_id;
```

右连接 (RIGHT JOIN)

[sql 复制代码](#)

```
SELECT customers.cust_id, orders.order_num
FROM customers RIGHT JOIN orders
ON customers.cust_id = orders.cust_id;
```

组合 (UNION)

- **UNION** 运算符将两个或更多查询的结果组合起来，并生成一个结果集，其中包含来自 **UNION** 中参与查询的提取行。
- **UNION** 基本规则
 - 所有查询的列数和列顺序必须相同。
 - 每个查询中涉及表的列的数据类型必须相同或兼容。

- 通常返回的列名取自第一个查询。
- 默认会去除相同行，如果需要保留相同行，使用 **UNION ALL**。
- 只能包含一个 **ORDER BY** 子句，并且必须位于语句的最后。
- 应用场景
 - 在一个查询中从不同的表返回结构数据。
 - 对一个表执行多个查询，按一个查询返回数据。

组合查询

sql 复制代码

```
SELECT cust_name, cust_contact, cust_email
FROM customers
WHERE cust_state IN ('IL', 'IN', 'MI')
UNION
SELECT cust_name, cust_contact, cust_email
FROM customers
WHERE cust_name = 'Fun4All';
```

JOIN vs UNION

- JOIN vs UNION
 - **JOIN** 中连接表的列可能不同，但在 **UNION** 中，所有查询的列数和列顺序必须相同。
 - **UNION** 将查询之后的行放在一起（垂直放置），但 **JOIN** 将查询之后的列放在一起（水平放置），即它构成一个笛卡尔积。

五、函数



注意：不同数据库的函数往往各不相同，因此不可移植。本节主要以 Mysql 的函数为例。

文本处理

函数	说明
LEFT() 、 RIGHT()	左边或者右边的字符
LOWER() 、 UPPER()	转换为小写或者大写
LTRIM() 、 RTIM()	去除左边或者右边的空格
LENGTH()	长度
SOUNDEX()	转换为语音值

其中， **SOUNDEX()** 可以将一个字符串转换为描述其语音表示的字母数字模式。

```
SELECT *
FROM mytable
WHERE SOUNDEX(col1) = SOUNDEX('apple')
```

scss 复制代码

日期和时间处理

- 日期格式： YYYY-MM-DD
- 时间格式： HH:MM:SS

函 数	说 明
AddDate()	增加一个日期（天、周等）
AddTime()	增加一个时间（时、分等）
CurDate()	返回当前日期
CurTime()	返回当前时间
Date()	返回日期时间的日期部分
DateDiff()	计算两个日期之差
Date_Add()	高度灵活的日期运算函数
Date_Format()	返回一个格式化的日期或时间串
Day()	返回一个日期的天数部分
DayOfWeek()	对于一个日期，返回对应的星期几
Hour()	返回一个时间的小时部分
Minute()	返回一个时间的分钟部分
Month()	返回一个日期的月份部分
Now()	返回当前日期和时间
Second()	返回一个时间的秒部分
Time()	返回一个日期时间的时间部分
Year()	返回一个日期的年份部分

```
mysql> SELECT NOW();
```

csharp 复制代码

```
2018-4-14 20:25:11
```

yaml 复制代码

数值处理

函数	说明
SIN()	正弦
COS()	余弦
TAN()	正切
ABS()	绝对值
SQRT()	平方根
MOD()	余数
EXP()	指数
PI()	圆周率
RAND()	随机数

汇总

函 数	说 明
AVG()	返回某列的平均值
COUNT()	返回某列的行数
MAX()	返回某列的最大值
MIN()	返回某列的最小值
SUM()	返回某列值之和

AVG() 会忽略 NULL 行。

使用 DISTINCT 可以让汇总函数值汇总不同的值。


```
SELECT AVG(DISTINCT col1) AS avg_col  
FROM mytable
```

六、排序和分组

ORDER BY

- **ORDER BY** 用于对结果集进行排序。
 - **ASC** : 升序 (默认)
 - **DESC** : 降序
- 可以按多个列进行排序, 并且为每个列指定不同的排序方式

指定多个列的排序方向

```
SELECT * FROM products  
ORDER BY prod_price DESC, prod_name ASC;
```

sql 复制代码

GROUP BY

- **GROUP BY** 子句将记录分组到汇总行中。
- **GROUP BY** 为每个组返回一个记录。
- **GROUP BY** 通常还涉及聚合: COUNT, MAX, SUM, AVG 等。
- **GROUP BY** 可以按一列或多列进行分组。
- **GROUP BY** 按分组字段进行排序后, **ORDER BY** 可以以汇总字段来进行排序。

分组

```
SELECT cust_name, COUNT(cust_address) AS addr_num  
FROM Customers GROUP BY cust_name;
```

vbnet 复制代码

分组后排序

```
SELECT cust_name, COUNT(cust_address) AS addr_num  
FROM Customers GROUP BY cust_name  
ORDER BY cust_name DESC;
```

vbnet 复制代码

HAVING

- **HAVING** 用于对汇总的 **GROUP BY** 结果进行过滤。
- **HAVING** 要求存在一个 **GROUP BY** 子句。
- **WHERE** 和 **HAVING** 可以在相同的查询中。
- **HAVING** vs **WHERE**
 - **WHERE** 和 **HAVING** 都是用于过滤。
 - **HAVING** 适用于汇总的组记录；而 **WHERE** 适用于单个记录。

使用 WHERE 和 HAVING 过滤数据

[sql 复制代码](#)

```
SELECT cust_name, COUNT(*) AS num
FROM Customers
WHERE cust_email IS NOT NULL
GROUP BY cust_name
HAVING COUNT(*) >= 1;
```

(以下为 DDL 语句用法)

七、数据定义

DDL 的主要功能是定义数据库对象（如：数据库、数据表、视图、索引等）。

数据库 (DATABASE)

创建数据库

[复制代码](#)

```
CREATE DATABASE test;
```

删除数据库

[复制代码](#)

```
DROP DATABASE test;
```

选择数据库

[复制代码](#)

```
USE test;
```

数据表 (TABLE)

创建数据表

普通创建

[sql 复制代码](#)

```
CREATE TABLE user (  
  id int(10) unsigned NOT NULL COMMENT 'Id',  
  username varchar(64) NOT NULL DEFAULT 'default' COMMENT '用户名',  
  password varchar(64) NOT NULL DEFAULT 'default' COMMENT '密码',  
  email varchar(64) NOT NULL DEFAULT 'default' COMMENT '邮箱'  
) COMMENT='用户表';
```

根据已有的表创建新表

[sql 复制代码](#)

```
CREATE TABLE vip_user AS  
SELECT * FROM user;
```

删除数据表

[sql 复制代码](#)

```
DROP TABLE user;
```

修改数据表

添加列

[sql 复制代码](#)

```
ALTER TABLE user  
ADD age int(3);
```

删除列

[sql 复制代码](#)

```
ALTER TABLE user
DROP COLUMN age;
```

修改列

[sql 复制代码](#)

```
ALTER TABLE `user`
MODIFY COLUMN age tinyint;
```

添加主键

[sql 复制代码](#)

```
ALTER TABLE user
ADD PRIMARY KEY (id);
```

删除主键

[sql 复制代码](#)

```
ALTER TABLE user
DROP PRIMARY KEY;
```

视图 (VIEW)

- 定义
 - 视图是基于 SQL 语句的结果集的可视化的表。
 - 视图是虚拟的表，本身不包含数据，也就不能对其进行索引操作。对视图的操作和对普通表的操作一样。
- 作用
 - 简化复杂的 SQL 操作，比如复杂的联结；
 - 只使用实际表的一部分数据；
 - 通过只给用户访问视图的权限，保证数据的安全性；
 - 更改数据格式和表示。

创建视图

[sql 复制代码](#)

```
CREATE VIEW top_10_user_view AS
SELECT id, username
FROM user
WHERE id < 10;
```

删除视图

[sql 复制代码](#)

```
DROP VIEW top_10_user_view;
```

索引 (INDEX)

- 作用
 - 通过索引可以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查询。
- 注意
 - 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。
- 唯一索引
 - 唯一索引表明此索引的每一个索引值只对应唯一的数据记录。

创建索引

[sql 复制代码](#)

```
CREATE INDEX user_index  
ON user (id);
```

创建唯一索引

[sql 复制代码](#)

```
CREATE UNIQUE INDEX user_index  
ON user (id);
```

删除索引

[sql 复制代码](#)

```
ALTER TABLE user  
DROP INDEX user_index;
```

约束

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）。
- 约束类型
 - **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。
 - **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。

创建表时使用约束条件：

sql 复制代码

```
CREATE TABLE Users (  
  Id INT(10) UNSIGNED NOT NULL AUTO_INCREMENT COMMENT '自增Id',  
  Username VARCHAR(64) NOT NULL UNIQUE DEFAULT 'default' COMMENT '用户名',  
  Password VARCHAR(64) NOT NULL DEFAULT 'default' COMMENT '密码',  
  Email VARCHAR(64) NOT NULL DEFAULT 'default' COMMENT '邮箱地址',  
  Enabled TINYINT(4) DEFAULT NULL COMMENT '是否有效',  
  PRIMARY KEY (Id)  
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8mb4 COMMENT='用户表';
```

(以下为 TCL 语句用法)

八、事务处理

- 不能回退 SELECT 语句，回退 SELECT 语句也没意义；也不能回退 CREATE 和 DROP 语句。
- **MySQL** 默认是隐式提交，每执行一条语句就把这条语句当成一个事务然后进行提交。当出现 **START TRANSACTION** 语句时，会关闭隐式提交；当 **COMMIT** 或 **ROLLBACK** 语句执行后，事务会自动关闭，重新恢复隐式提交。
- 通过 **set autocommit=0** 可以取消自动提交，直到 **set autocommit=1** 才会提交；autocommit 标记是针对每个连接而不是针对服务器的。
- 指令

- **START TRANSACTION** - 指令用于标记事务的起始点。
- **SAVEPOINT** - 指令用于创建保留点。
- **ROLLBACK TO** - 指令用于回滚到指定的保留点；如果没有设置保留点，则回退到 **START TRANSACTION** 语句处。
- **COMMIT** - 提交事务。

sql 复制代码

```
-- 开始事务
START TRANSACTION;

-- 插入操作 A
INSERT INTO `user`
VALUES (1, 'root1', 'root1', 'xxxx@163.com');

-- 创建保留点 updateA
SAVEPOINT updateA;

-- 插入操作 B
INSERT INTO `user`
VALUES (2, 'root2', 'root2', 'xxxx@163.com');

-- 回滚到保留点 updateA
ROLLBACK TO updateA;

-- 提交事务, 只有操作 A 生效
COMMIT;
```

(以下为 DCL 语句用法)

九、权限控制

- GRANT 和 REVOKE 可在几个层次上控制访问权限：
 - 整个服务器，使用 GRANT ALL 和 REVOKE ALL；
 - 整个数据库，使用 ON database.*；
 - 特定的表，使用 ON database.table；
 - 特定的列；
 - 特定的存储过程。
- 新创建的账户没有任何权限。
- 账户用 username@host 的形式定义，username@% 使用的是默认主机名。
- MySQL 的账户信息保存在 mysql 这个数据库中。

[sql 复制代码](#)

```
USE mysql;  
SELECT user FROM user;
```

创建账户

[sql 复制代码](#)

```
CREATE USER myuser IDENTIFIED BY 'mypassword';
```

修改账户名

[sql 复制代码](#)

```
UPDATE user SET user='newuser' WHERE user='myuser';  
FLUSH PRIVILEGES;
```

删除账户

[sql 复制代码](#)

```
DROP USER myuser;
```

查看权限

[ini 复制代码](#)

```
SHOW GRANTS FOR myuser;
```

授予权限

[sql 复制代码](#)

```
GRANT SELECT, INSERT ON *.* TO myuser;
```

删除权限

[sql 复制代码](#)

```
REVOKE SELECT, INSERT ON *.* FROM myuser;
```

更改密码

[ini 复制代码](#)

```
SET PASSWORD FOR myuser = 'mypass';
```


十、存储过程

- 存储过程可以看成是对一系列 SQL 操作的批处理；
- 使用存储过程的好处
 - 代码封装，保证了一定的安全性；
 - 代码复用；
 - 由于是预先编译，因此具有很高的性能。
- 创建存储过程
 - 命令行中创建存储过程需要自定义分隔符，因为命令行是以 `;` 为结束符，而存储过程中也包含了分号，因此会错误把这部分分号当成是结束符，造成语法错误。
 - 包含 in、out 和 inout 三种参数。
 - 给变量赋值都需要用 `select into` 语句。
 - 每次只能给一个变量赋值，不支持集合的操作。

创建存储过程

[sql 复制代码](#)

```
DROP PROCEDURE IF EXISTS `proc_adder`;
DELIMITER ;;
CREATE DEFINER=`root`@`localhost` PROCEDURE `proc_adder`(IN a int, IN b int, OUT sum int)
BEGIN
    DECLARE c int;
    if a is null then set a = 0;
    end if;

    if b is null then set b = 0;
    end if;

    set sum = a + b;
END
;;
DELIMITER ;
```

使用存储过程

[less 复制代码](#)

```
set @b=5;
call proc_adder(2,@b,@s);
select @s as sum;
```

十一、游标

- 游标（cursor）是一个存储在 DBMS 服务器上的数据库查询，它不是一条 SELECT 语句，而是被该语句检索出来的结果集。
- 在存储过程中使用游标可以对一个结果集进行移动遍历。
- 游标主要用于交互式应用，其中用户需要对数据集中的任意行进行浏览和修改。
- 使用游标的四个步骤：
 - 声明游标，这个过程没有实际检索出数据；
 - 打开游标；
 - 取出数据；
 - 关闭游标；

sql 复制代码

```

DELIMITER $

CREATE PROCEDURE getTotal()
BEGIN
    DECLARE total INT;
    -- 创建接收游标数据的变量
    DECLARE sid INT;
    DECLARE sname VARCHAR(10);
    -- 创建总数变量
    DECLARE sage INT;
    -- 创建结束标志变量
    DECLARE done INT DEFAULT false;
    -- 创建游标
    DECLARE cur CURSOR FOR SELECT id,name,age from cursor_table where age>30;
    -- 指定游标循环结束时的返回值
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = true;
    SET total = 0;
    OPEN cur;
    FETCH cur INTO sid, sname, sage;
    WHILE(NOT done)
    DO
        SET total = total + 1;
        FETCH cur INTO sid, sname, sage;
    END WHILE;

    CLOSE cur;
    SELECT total;
END $

DELIMITER ;

-- 调用存储过程
call getTotal();

```

十二、触发器


触发器是一种与表操作有关的数据库对象，当触发器所在表上出现指定事件时，将调用该对象，即表的操作事件触发表上的触发器的执行。

可以使用触发器来进行审计跟踪，把修改记录到另外一张表中。

MySQL 不允许在触发器中使用 CALL 语句，也就是不能调用存储过程。

BEGIN 和 END

当触发器的触发条件满足时，将会执行 BEGIN 和 END 之间的触发器执行动作。

 注意：在 MySQL 中，分号 ; 是语句结束的标识符，遇到分号表示该段语句已经结束，MySQL 可以开始执行了。因此，解释器遇到触发器执行动作中的分号后就开始执行，然后会报错，因为没有找到和 BEGIN 匹配的 END。

这时就会用到 DELIMITER 命令（DELIMITER 是定界符，分隔符的意思）。它是一条命令，不需要语句结束标识，语法为：DELIMITER new_delemiter。new_delemiter 可以设为 1 个或多个长度的符号，默认的是分号 ;，我们可以把它修改为其他符号，如 \$ - DELIMITER \$。在这之后的语句，以分号结束，解释器不会有什么反应，只有遇到了 \$，才认为是语句结束。注意，使用完之后，我们还应该记得把它给修改回来。

NEW 和 OLD

- MySQL 中定义了 NEW 和 OLD 关键字，用来表示触发器的所在表中，触发了触发器的那一行数据。
- 在 INSERT 型触发器中，NEW 用来表示将要（BEFORE）或已经（AFTER）插入的新数据；
- 在 UPDATE 型触发器中，OLD 用来表示将要或已经被修改的原数据，NEW 用来表示将要或已经修改为新数据；
- 在 DELETE 型触发器中，OLD 用来表示将要或已经被删除的原数据；
- 使用方法：NEW.columnName（columnName 为相应数据表某一列名）

创建触发器

提示：为了理解触发器的要点，有必要先了解一下创建触发器的指令。

CREATE TRIGGER 指令用于创建触发器。

语法：

sql 复制代码

```
CREATE TRIGGER trigger_name
trigger_time
trigger_event
ON table_name
FOR EACH ROW
BEGIN
    trigger_statements
END;
```

说明：

- trigger_name: 触发器名
- trigger_time: 触发器的触发时机。取值为 **BEFORE** 或 **AFTER** 。
- trigger_event: 触发器的监听事件。取值为 **INSERT**、**UPDATE** 或 **DELETE** 。
- table_name: 触发器的监听目标。指定在哪张表上建立触发器。
- FOR EACH ROW: 行级监视，Mysql 固定写法，其他 DBMS 不同。
- trigger_statements: 触发器执行动作。是一条或多条 SQL 语句的列表，列表内的每条语句都必须用分号 **;** 来结尾。

示例：

sql 复制代码

```
DELIMITER $
CREATE TRIGGER `trigger_insert_user`
AFTER INSERT ON `user`
FOR EACH ROW
BEGIN
    INSERT INTO `user_history`(user_id, operate_type, operate_time)
    VALUES (NEW.id, 'add a user', now());
END $
DELIMITER ;
```

查看触发器

ini 复制代码

```
SHOW TRIGGERS;
```

删除触发器

[sql 复制代码](#)

```
DROP TRIGGER IF EXISTS trigger_insert_user;
```

(完)

参考资料

- BenForta. SQL 必知必会 [M]. 人民邮电出版社, 2013.
- [『浅入深出』MySQL 中事务的实现](#)
- [MySQL 的学习--触发器](#)
- [维基百科词条 - SQL](#)
- www.sitesbay.com/sql/index
- [SQL Subqueries](#)
- [Quick breakdown of the types of joins](#)
- [SQL UNION](#)
- [SQL database security](#)
- [Mysql 中的存储过程](#)

分类： 后端 标签： [SQL](#)

安装掘金浏览器插件

多内容聚合浏览、多引擎快捷搜索、多工具便捷提效、多模式随心畅享，你想要的，这里都有！

[前往安装](#)