SQLite

什么是SQLite

SQLite，是一款轻型的数据库，是遵守ACID的关系型数据库管理系统，它包含在一个相对小的C库中。它是D.RichardHipp建立的公有领域项目。它的设计目标是嵌入式的，而且目前已经在很多嵌入式产品中使用了它，它占用资源非常的低，在嵌入式设备中，可能只需要几百K的内存就够了。它能够支持Windows/Linux/Unix等等主流的操作系统，同时能够跟很多程序语言相结合，比如 Tcl、C#、PHP、Java等，还有ODBC接口，同样比起Mysql、PostgreSQL这两款开源的世界著名数据库管理系统来讲，它的处理速度比他们都快。SQLite第一个Alpha版本诞生于2000年5月。至2016年已经有16个年头，SQLite也迎来了一个版本 SQLite 3已经发布

> SQLite主要功能特性
>
> 1.ACID事务
>
> 2.零配置 – 无需安装和管理配置
>
> 3.储存在单一磁盘文件中的一个完整的数据库
>
> 4.数据库文件可以在不同字节顺序的机器间自由的共享
>
> 5.支持数据库大小至2TB
>
> 6.足够小, 大致13万行C代码, 4.43M
>
> 7.比一些流行的数据库在大部分普通数据库操作要快
>
> 8.独立: 没有额外依赖
>
> 9.源码完全的开源
>
> 10.支持多种开发语言，C, C++, PHP, Perl, Java, C#,Python, Ruby等

SQLite 安装

SQLite 的一个重要的特性是零配置的

在Windows上安装SQLite

- 请访问 SQLite 下载页面，从 Windows 区下载预编译的二进制文件。
- 您需要下载 **sqlite-tools-win32-*.zip** 和 **sqlite-dll-win32-*.zip** 压缩文件。
- 创建文件夹 C:\sqlite，并在此文件夹下解压上面两个压缩文件，将得到 sqlite3.def、sqlite3.dll 和 sqlite3.exe 文件。
- 添加 C:\sqlite 到 PATH 环境变量，最后在命令提示符下，使用 **sqlite3** 命令，将显示如下结果。

SQLite数据类型

每个存储在 SQLite 数据库中的值都具有以下存储类之一：

存储类	描述
NULL	值是一个 NULL 值。
INTEGER	值是一个带符号的整数，根据值的大小存储在 1、2、3、4、6 或 8 字节中。
REAL	值是一个浮点值，存储为 8 字节的 IEEE 浮点数字。
TEXT	值是一个文本字符串，使用数据库编码（UTF-8、UTF-16BE 或 UTF-16LE）存储。
BLOB	值是一个 blob 数据，完全根据它的输入存储。

SQLite运算符

SQLite 算术运算符

假设变量 a=10，变量 b=20，则：

运算符	描述	实例
+	加法 - 把运算符两边的值相加	a + b 将得到 30
-	减法 - 左操作数减去右操作数	a - b 将得到 -10
*	乘法 - 把运算符两边的值相乘	a * b 将得到 200
/	除法 - 左操作数除以右操作数	b / a 将得到 2
%	取模 - 左操作数除以右操作数后得到的余数	b % a will give 0

SQLite 比较运算符

假设变量 a=10，变量 b=20，则：

运算符	描述	实例
==	检查两个操作数的值是否相等，如果相等则条件为真。	(a == b) 不为真。
=	检查两个操作数的值是否相等，如果相等则条件为真。	(a = b) 不为真。
!=	检查两个操作数的值是否相等，如果不相等则条件为真。	(a != b) 为真。
<>	检查两个操作数的值是否相等，如果不相等则条件为真。	(a <> b) 为真。
>	检查左操作数的值是否大于右操作数的值，如果是则条件为真。	(a > b) 不为真。
<	检查左操作数的值是否小于右操作数的值，如果是则条件为真。	(a < b) 为真。
>=	检查左操作数的值是否大于等于右操作数的值，如果是则条件为真。	(a >= b) 不为真。
<=	检查左操作数的值是否小于等于右操作数的值，如果是则条件为真。	(a <= b) 为真。
!<	检查左操作数的值是否不小于右操作数的值，如果是则条件为真。	(a !< b) 为假。
!>	检查左操作数的值是否不大于右操作数的值，如果是则条件为真。	(a !> b) 为真。

SQLite 逻辑运算符

运算符	描述
AND	AND 运算符允许在一个 SQL 语句的 WHERE 子句中的多个条件的存在。
BETWEEN	BETWEEN 运算符用于在给定最小值和最大值范围内的一系列值中搜索值。
EXISTS	EXISTS 运算符用于在满足一定条件的指定表中搜索行的存在。
IN	IN 运算符用于把某个值与一系列指定列表的值进行比较。
NOT IN	IN 运算符的对立面，用于把某个值与不在一系列指定列表的值进行比较。
LIKE	LIKE 运算符用于把某个值与使用通配符运算符的相似值进行比较。
GLOB	GLOB 运算符用于把某个值与使用通配符运算符的相似值进行比较。GLOB 与 LIKE 不同之处在于，它是大小写敏感的。
NOT	NOT 运算符是所用的逻辑运算符的对立面。比如 NOT EXISTS、NOT BETWEEN、NOT IN，等等。**它是否定运算符。**
OR	OR 运算符用于结合一个 SQL 语句的 WHERE 子句中的多个条件。
IS NULL	NULL 运算符用于把某个值与 NULL 值进行比较。
IS	IS 运算符与 = 相似。
IS NOT	IS NOT 运算符与 != 相似。
||	连接两个不同的字符串，得到一个新的字符串。
UNIQUE	UNIQUE 运算符搜索指定表中的每一行，确保唯一性（无重复）。

SQLite 位运算符

下表中列出了 SQLite 语言支持的位运算符。假设变量 A=60，变量 B=13，则：

运算符	描述	实例
&	如果同时存在于两个操作数中，二进制 AND 运算符复制一位到结果中。	(A & B) 将得到 12，即为 0000 1100
|	如果存在于任一操作数中，二进制 OR 运算符复制一位到结果中。	(A | B) 将得到 61，即为 0011 1101
~	二进制补码运算符是一元运算符，具有"翻转"位效应，即0变成1，1变成0。	(~A) 将得到 -61，即为 1100 0011，一个有符号二进制数的补码形式。
<<	二进制左移运算符。左操作数的值向左移动右操作数指定的位数。	A << 2 将得到 240，即为 1111 0000
>>	二进制右移运算符。左操作数的值向右移动右操作数指定的位数。	A >> 2 将得到 15，即为 0000 1111

基本的sql语法

DDL 数据定义语言

命令	描述
CREATE	创建一个新的表，一个表的视图，或者数据库中的其他对象。
ALTER	修改数据库中的某个已有的数据库对象，比如一个表。
DROP	删除整个表，或者表的视图，或者数据库中的其他对象。

DML 数据操作语言

命令	描述
INSERT	创建一条记录。
UPDATE	修改记录。
DELETE	删除记录。

DQL 数据查询语言

命令	描述
SELECT	从一个或多个表中检索某些记录。

SQLite创建表

SQLite 的 **CREATE TABLE** 语句用于在任何给定的数据库创建一个新表。创建基本表，涉及到命名表、定义列及每一列的数据类型。

```
CREATE TABLE database_name.table_name(
   column1 datatype  PRIMARY KEY(one or more columns),
   column2 datatype,
   column3 datatype,
   .....
   columnN datatype,
);
--示例
create table user(
    id int primary key not null,
    name text not null,
    age int not null,
    weight real not null
);

--使用.tables查看
.tables
```

SQLite修改已有的表

用来重命名已有的表的 **ALTER TABLE**

```
ALTER TABLE database_name.table_name RENAME TO new_table_name;
--示例
alter table user rename to old_user;
```

用来在已有的表中添加一个新的列的 **ALTER TABLE** 的基本语法如下：

```
ALTER TABLE database_name.table_name ADD COLUMN column_def...;
--示例
alter table user old_user add column sex char(1);
```

SQLite删除表

SQLite 的 **DROP TABLE** 语句用来删除表定义及其所有相关数据、索引、触发器、约束和该表的权限规范。

> 使用此命令时要特别注意，因为一旦一个表被删除，表中所有信息也将永远丢失。

```
DROP TABLE database_name.table_name;
--示例
drop table user;
```

SQLite Insert语法

SQLite 的 **INSERT INTO** 语句用于向数据库的某个表中添加新的数据行。

```
INSERT INTO TABLE_NAME [(column1, column2, column3,...columnN)]
VALUES (value1, value2, value3,...valueN);
--如果要为表中的所有列添加值，您也可以不需要在 SQLite 查询中指定列名称。
INSERT INTO TABLE_NAME VALUES (value1,value2,value3,...valueN);
--示例
insert into user(id,name,age,weight) values(1,'zero',18,120);
--所有的列都添加值，只需确保值的顺序与列在表中的顺序一致
insert into user(id,name,age,weight) values(2,'lance',17,140);
```

SQLite Update语法

SQLite 的 **UPDATE** 查询用于修改表中已有的记录。可以使用带有 WHERE 子句的 UPDATE 查询来更新选定行，否则所有的行都会被更新。

> 可以使用 AND 或 OR 运算符来结合 N 个数量的条件

```
UPDATE table_name
SET column1 = value1, column2 = value2...., columnN = valueN
WHERE [condition];
--示例
update user
set age = 23
where name = 'zero';
```

SQLite Delete语法

SQLite 的 **DELETE** 查询用于删除表中已有的记录。可以使用带有 WHERE 子句的 DELETE 查询来删除选定行，否则所有的记录都会被删除。

> 可以使用 AND 或 OR 运算符来结合 N 个数量的条件

```
DELETE FROM table_name
WHERE [condition];
--示例
delete from user where id = 2;
```

SQLite Select语法

SQLite 的 **SELECT** 语句用于从 SQLite 数据库表中获取数据，以结果表的形式返回数据。这些结果表也被称为结果集。

```
SELECT column1, column2, columnN FROM table_name;
--查询所有行
SELECT * FROM table_name;
--示例
select id,name,age from user;
--查询所有行
select * from user;
```

SQLite Where 语法

SQLite的 **WHERE** 子句用于指定从一个表或多个表中获取数据的条件。

如果满足给定的条件，即为真（true）时，则从表中返回特定的值。您可以使用 WHERE 子句来过滤记录，只获取需要的记录。

WHERE 子句不仅可用在 SELECT 语句中，它也可用在 UPDATE、DELETE 语句中，等等

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition]
--示例
select * from user where age >18;
select * from user where age >18 and age < 32;
select * from user where age  is not null;
select * from user where age >18 or weight >120;
```

SQLite Like语法

SQLite 的 **LIKE** 运算符是用来匹配通配符指定模式的文本值。如果搜索表达式与模式表达式匹配，LIKE 运算符将返回真（true），也就是 1。这里有两个通配符与 LIKE 运算符一起使用：

- 百分号 （%）
- 下划线 （_）

百分号（%）代表零个、一个或多个数字或字符。下划线（_）代表一个单一的数字或字符。这些符号可以被组合使用

语句	描述
where weight like '120%'	查找以 120 开头的任意值
where weight like '%3%'	查找任意位置包含 3 的任意值
where weight like '_21%'	查找第二位和第三位为 21 的任意值
where weight like '1%%'	查找以 1 开头，且长度至少为 3 个字符的任意值
where weight like '%2'	查找以 2 结尾的任意值
where weight like '_2%3'	查找第二位为 2，且以 3 结尾的任意值
where weight like '1__3'	查找长度为 4 位数，且以 1 开头以 3 结尾的任意值

SQLite Glob 语法

SQLite 的 **GLOB** 运算符是用来匹配通配符指定模式的文本值。如果搜索表达式与模式表达式匹配，GLOB 运算符将返回真（true），也就是 1。与 LIKE 运算符不同的是，GLOB 是大小写敏感的，对于下面的通配符，它遵循 UNIX 的语法。

- 星号 （*）
- 问号 （?）

星号（*）代表零个、一个或多个数字或字符。问号（?）代表一个单一的数字或字符。这些符号可以被组合使用。

语句	描述
where weight glob '120*'	查找以 120 开头的任意值
where weight glob '*3*'	查找任意位置包含 3 的任意值
where weight glob '?21*'	查找第二位和第三位为 21 的任意值
where weight glob '1??'	查找以 1 开头，且长度至少为 3 个字符的任意值
where weight glob '*2'	查找以 2 结尾的任意值
where weight glob '?2*3'	查找第二位为 2，且以 3 结尾的任意值
where weight glob '1??3'	查找长度为 4 位数，且以 1 开头以 3 结尾的任意值

SQLite Limit 语法

SQLite 的 **LIMIT** 子句用于限制由 SELECT 语句返回的数据数量。 带有 LIMIT 子句的 SELECT 语句的基本语法如下：

```
SELECT column1, column2, columnN
FROM table_name
LIMIT [no of rows]
--示例
select * from user limit 3;
```

下面是 LIMIT 子句与 OFFSET 子句一起使用时的语法：

```
SELECT column1, column2, columnN
FROM table_name
LIMIT [no of rows] OFFSET [row num]
--示例
select * from user limit 3 offset 2;
```

SQLite 引擎将返回从下一行开始直到给定的 OFFSET 为止的所有行

> 分页查询
>
> select * from user order by id limit 10 offset 0;//offset代表从第几条记录“之后”开始查询，limit表明查询多少条结果

> 运用： sqlitecmd.CommandText = String.Format("select * from GuestInfo order by GuestId limit {0} offset {0}*{1}", size, index-1);//size:每页显示条数，index页码

SQLite Order By语法

SQLite 的 **ORDER BY** 子句是用来基于一个或多个列按升序或降序顺序排列数据。

```
SELECT column-list
FROM table_name
[WHERE condition]
[ORDER BY column1, column2, .. columnN] [ASC | DESC];
--示例
select * from user order by age,name desc;
--有多个排序字段时，优先按照前面的排序，如果前面的排序中有值相同的，后面的字段才起作用，在这些值相同的记录中，按照后续字段排序
```

SQLite Group By

SQLite 的 **GROUP BY** 子句用于与 SELECT 语句一起使用，来对相同的数据进行分组。 在 SELECT 语句中，GROUP BY 子句放在 **WHERE 子句之后**，放在 **ORDER BY 子句之前。**

```
SELECT column-list
FROM table_name
WHERE [ conditions ]
GROUP BY column1, column2....columnN
ORDER BY column1, column2....columnN
--示例
select name sum(weight) from user group by name;
select name sum(weight) from user group by name order by name desc;
```

SQLite Having 语法

HAVING 子句允许指定条件来过滤将出现在最终结果中的分组结果。 WHERE 子句在所选列上设置条件，而 HAVING 子句则在由 GROUP BY 子句创建的分组上设置条件。

```
SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY
```

在一个查询中，HAVING 子句必须放在 GROUP BY 子句之后，必须放在 ORDER BY 子句之前。下面是包含 HAVING 子句的 SELECT 语句的语法：

```
SELECT column1, column2
FROM table1, table2
WHERE [ conditions ]
GROUP BY column1, column2
HAVING [ conditions ]
ORDER BY column1, column2
--示例
select * from user group by name having count(name) > 2;
```

SQLite Distinct 关键字

SQLite 的 **DISTINCT** 关键字与 SELECT 语句一起使用，来消除所有重复的记录，并只获取唯一一次记录。

有可能出现一种情况，在一个表中有多个重复的记录。当提取这样的记录时，DISTINCT 关键字就显得特别有意义，它只获取唯一一次记录，而不是获取重复记录。

```
SELECT DISTINCT column1, column2,.....columnN
FROM table_name
WHERE [condition]
--示例
select distinct name from user;
```

SQLite高级用法

SQLite 约束

约束是在表的数据列上强制执行的规则。这些是用来限制可以插入到表中的数据类型。这确保了数据库中数据的准确性和可靠性。

约束可以是列级或表级。列级约束仅适用于列，表级约束被应用到整个表。

以下是在 SQLite 中常用的约束。

- **NOT NULL 约束**：确保某列不能有 NULL 值。
- **DEFAULT 约束**：当某列没有指定值时，为该列提供默认值。
- **UNIQUE 约束**：确保某列中的所有值是不同的。
- **PRIMARY Key 约束**：唯一标识数据库表中的各行/记录。
- **CHECK 约束**：CHECK 约束确保某列中的所有值满足一定条件。

SQLite Autoincrement（自动递增）

SQLite 的 **AUTOINCREMENT** 是一个关键字，用于表中的字段值自动递增。我们可以在创建表时在特定的列名称上使用 **AUTOINCREMENT** 关键字实现该字段值的自动增加。

关键字 **AUTOINCREMENT** 只能用于整型（INTEGER）字段。

```
create table student(
    id int primary key autoincrement, --主键 自增
    name text not null unique, --不允许null 不允许同名
    score real default 0, -- 设置默认值
    age int check(age > 18) --age必须大于18
);
```

SQLite Join语法

SQLite 的 **Join** 子句用于结合两个或多个数据库中表的记录。JOIN 是一种通过共同值来结合两个表中字段的手段。

SQL 定义了三种主要类型的连接：

- 交叉连接 - CROSS JOIN
- 内连接 - INNER JOIN
- 外连接 - OUTER JOIN

交叉连接 - CROSS JOIN

交叉连接（CROSS JOIN）把第一个表的每一行与第二个表的每一行进行匹配。如果两个输入表分别有 x 和 y 行，则结果表有 x*y 行。由于交叉连接（CROSS JOIN）有可能产生非常大的表，使用时必须谨慎，只在适当的时候使用它们。

> 交叉连接的操作，它们都返回被连接的两个表所有数据行的笛卡尔积，返回到的数据行数等于第一个表中符合查询条件的数据行数乘以第二个表中符合查询条件的数据行数。

下面是交叉连接（CROSS JOIN）的语法：

```
SELECT ... FROM table1 CROSS JOIN table2 ...
--示例
create table class(
    id int primary key autoincrement,
    user_id int not null,
    class_name text not null
);
select id, name, class_name from user cross join class;
```

内连接 - INNER JOIN

内连接（INNER JOIN）根据连接谓词结合两个表（table1 和 table2）的列值来创建一个新的结果表。查询会把 table1 中的每一行与 table2 中的每一行进行比较，找到所有满足连接谓词的行的匹配对。当满足连接谓词时，A 和 B 行的每个匹配对的列值会合并成一个结果行。

内连接（INNER JOIN）是最常见的连接类型，是默认的连接类型。INNER 关键字是可选的。

下面是内连接（INNER JOIN）的语法：

```
SELECT ... FROM table1 [INNER] JOIN table2 ON conditional_expression ...
--示例
select id, name, class_name from user inner join class
on user.id = class.user_id;
```

外连接 - OUTER JOIN

外连接（OUTER JOIN）是内连接（INNER JOIN）的扩展。虽然 SQL 标准定义了三种类型的外连接：LEFT、RIGHT、FULL，但 SQLite 只支持 **左外连接（LEFT OUTER JOIN）**。

外连接（OUTER JOIN）声明条件的方法与内连接（INNER JOIN）是相同的，使用 ON、USING 或 NATURAL 关键字来表达。最初的结果表以相同的方式进行计算。一旦主连接计算完成，外连接（OUTER JOIN）将从一个或两个表中任何未连接的行合并进来，外连接的列使用 NULL 值，将它们附加到结果表中。

下面是左外连接（LEFT OUTER JOIN）的语法：

```
SELECT ... FROM table1 LEFT OUTER JOIN table2 ON conditional_expression ...
--示例
select id, name, class_name from user left outer join class
on user.id = class.user_id;
```

SQLite Unions 语法

SQLite的 **UNION** 子句/运算符用于合并两个或多个 SELECT 语句的结果，不返回任何重复的行。

为了使用 UNION，每个 SELECT 被选择的列数必须是相同的，相同数目的列表达式，相同的数据类型，并确保它们有相同的顺序，但它们不必具有相同的长度。

- **UNION**

```
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]

UNION

SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
--示例
select id ,name, class_name from user inner join class
on user.id = class.user_id
union
select id ,name, class_name from user left outer join class
on user.id = class.user_id;
```

- **UNION ALL**

UNION ALL 运算符用于结合两个 SELECT 语句的结果，包括重复行。 适用于 UNION 的规则同样适用于 UNION ALL 运算符。

```
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]

UNION ALL

SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
```

SQLite 索引（Index）

索引（Index）是一种特殊的查找表，数据库搜索引擎用来加快数据检索。简单地说，索引是一个指向表中数据的指针。一个数据库中的索引与一本书的索引目录是非常相似的。

拿汉语字典的目录页（索引）打比方，我们可以按拼音、笔画、偏旁部首等排序的目录（索引）快速查找到需要的字。

索引有助于加快 SELECT 查询和 WHERE 子句，但它会减慢使用 UPDATE 和 INSERT 语句时的数据输入。索引可以创建或删除，但不会影响数据。

使用 CREATE INDEX 语句创建索引，它允许命名索引，指定表及要索引的一列或多列，并指示索引是升序排列还是降序排列。

索引也可以是唯一的，与 UNIQUE 约束类似，在列上或列组合上防止重复条目。

CREATE INDEX 命令

CREATE INDEX 的基本语法如下：

```
CREATE INDEX index_name ON table_name;
--示例
```

- 单列索引

单列索引是一个只基于表的一个列上创建的索引。基本语法如下：

```
CREATE INDEX index_name
ON table_name (column_name);
--示例
create index age_index on user(age);
--查看索引
.indices user
```

- 唯一索引

使用唯一索引不仅是为了性能，同时也为了数据的完整性。唯一索引不允许任何重复的值插入到表中。基本语法如下：

```
CREATE UNIQUE INDEX index_name
on table_name (column_name);
--示例
create unique index name_index on user(name);
```

- 组合索引

组合索引是基于一个表的两个或多个列上创建的索引。基本语法如下：

```
CREATE INDEX index_name
on table_name (column1, column2);
--示例
create index compex_index on user(name,age);
```

是否要创建一个单列索引还是组合索引，要考虑到您在作为查询过滤条件的 WHERE 子句中使用非常频繁的列。

如果值使用到一个列，则选择使用单列索引。如果在作为过滤的 WHERE 子句中有两个或多个列经常使用，则选择使用组合索引。

- 隐式索引

隐式索引是在创建对象时，由数据库服务器自动创建的索引。索引自动创建为主键约束和唯一约束

DROP INDEX 命令

一个索引可以使用 SQLite 的 **DROP** 命令删除。当删除索引时应特别注意，因为性能可能会下降或提高。

基本语法如下：

```
DROP INDEX index_name;
--示例
drop index age_index;
```

什么情况下要避免使用索引?

虽然索引的目的在于提高数据库的性能，但这里有几个情况需要避免使用索引。使用索引时，应重新考虑下列准则：

- 索引不应该使用在较小的表上。
- 索引不应该使用在有频繁的大批量的更新或插入操作的表上。
- 索引不应该使用在含有大量的 NULL 值的列上。
- 索引不应该使用在频繁操作的列上。

SQLite 子查询

子查询或称为内部查询、嵌套查询，指的是在 SQLite 查询中的 WHERE 子句中嵌入查询语句。 一个 SELECT 语句的查询结果能够作为另一个语句的输入值。

子查询可以与 SELECT、INSERT、UPDATE 和 DELETE 语句一起使用，可伴随着使用运算符如 =、<、>、>=、<=、IN、BETWEEN 等。

以下是子查询必须遵循的几个规则：

- 子查询必须用括号括起来。
- 子查询在 SELECT 子句中只能有一个列，除非在主查询中有多列，与子查询的所选列进行比较。
- ORDER BY 不能用在子查询中，虽然主查询可以使用 ORDER BY。可以在子查询中使用 GROUP BY，功能与 ORDER BY 相同。

- 子查询返回多于一行，只能与多值运算符一起使用，如 IN 运算符。
- BETWEEN 运算符不能与子查询一起使用，但是，BETWEEN 可在子查询内使用。

SELECT 语句中的子查询使用

子查询通常与 SELECT 语句一起使用。基本语法如下：

```
SELECT column_name [, column_name ]
FROM   table1 [, table2 ]
WHERE  column_name OPERATOR
      (SELECT column_name [, column_name ]
      FROM table1 [, table2 ]
      [WHERE])
--示例
select * from user
       where id in (select id from user where age > 18);
```

INSERT 语句中的子查询使用

子查询也可以与 INSERT 语句一起使用。INSERT 语句使用子查询返回的数据插入到另一个表中。在子查询中所选择的数据可以用任何字符、日期或数字函数修改。

基本语法如下：

```
INSERT INTO table_name [ (column1 [, column2 ]) ]
           SELECT [ *|column1 [, column2 ]
           FROM table1 [, table2 ]
           [ WHERE VALUE OPERATOR ]
--示例
insert into user_bk
            select * from user
            where id in (select id from user) ;
```

UPDATE 语句中的子查询使用

子查询可以与 UPDATE 语句结合使用。当通过 UPDATE 语句使用子查询时，表中单个或多个列被更新。

基本语法如下：

```
UPDATE table
SET column_name = new_value
[ WHERE OPERATOR [ VALUE ]
   (SELECT COLUMN_NAME
   FROM TABLE_NAME)
   [ WHERE) ]
--示例
 update user set weight = 150
 where age in (select age from user_bk where age >20);
```

DELETE 语句中的子查询使用

子查询可以与 DELETE 语句结合使用，就像上面提到的其他语句一样。

基本语法如下：

```
DELETE FROM TABLE_NAME
[ WHERE OPERATOR [ VALUE ]
   (SELECT COLUMN_NAME
   FROM TABLE_NAME)
   [ WHERE) ]
--示例
 delete from user
 where age in (select age from user_bk where age >20);
```

不常用的语法

SQLite 事务（Transaction）

事务（Transaction）是一个对数据库执行工作单元。事务（Transaction）是以逻辑顺序完成的工作单位或序列，可以是由用户手动操作完成，也可以是由某种数据库程序自动完成。

事务（Transaction）是指一个或多个更改数据库的扩展。例如，如果您正在创建一个记录或者更新一个记录或者从表中删除一个记录，那么您正在该表上执行事务。重要的是要控制事务以确保数据的完整性和处理数据库错误。

实际上，您可以把许多的 SQLite 查询联合成一组，把所有这些放在一起作为事务的一部分进行执行。

事务的属性

事务（Transaction）具有以下四个标准属性，通常根据首字母缩写为 ACID：

- **原子性（Atomicity）：**确保工作单位内的所有操作都成功完成，否则，事务会在出现故障时终止，之前的操作也会回滚到以前的状态。
- **一致性（Consistency）:** 确保数据库在成功提交的事务上正确地改变状态。
- **隔离性（Isolation）：**使事务操作相互独立和透明。
- **持久性（Durability）：**确保已提交事务的结果或效果在系统发生故障的情况下仍然存在。

事务控制

使用下面的命令来控制事务：

- **BEGIN TRANSACTION**：开始事务处理。
- **COMMIT**：保存更改，或者可以使用 **END TRANSACTION** 命令。
- **ROLLBACK**：回滚所做的更改。

事务控制命令只与 DML 命令 INSERT、UPDATE 和 DELETE 一起使用。他们不能在创建表或删除表时使用，因为这些操作在数据库中是自动提交的。

BEGIN TRANSACTION 命令

事务（Transaction）可以使用 BEGIN TRANSACTION 命令或简单的 BEGIN 命令来启动。此类事务通常会持续执行下去，直到遇到下一个 COMMIT 或 ROLLBACK 命令。不过在数据库关闭或发生错误时，事务处理也会回滚。以下是启动一个事务的简单语法：

```
BEGIN;

or

BEGIN TRANSACTION;
```

COMMIT 命令

COMMIT 命令是用于把事务调用的更改保存到数据库中的事务命令。

COMMIT 命令把自上次 COMMIT 或 ROLLBACK 命令以来的所有事务保存到数据库。

COMMIT 命令的语法如下：

```
COMMIT;

or

END TRANSACTION;
```

ROLLBACK 命令

ROLLBACK 命令是用于撤消尚未保存到数据库的事务的事务命令。

ROLLBACK 命令只能用于撤销自上次发出 COMMIT 或 ROLLBACK 命令以来的事务。

ROLLBACK 命令的语法如下：

```
ROLLBACK;
```

SQLite 触发器（Trigger）

SQLite **触发器（Trigger）**是数据库的回调函数，它会在指定的数据库事件发生时自动执行/调用。以下是关于 SQLite 的触发器（Trigger）的要点：

- SQLite 的触发器（Trigger）可以指定在特定的数据库表发生 DELETE、INSERT 或 UPDATE 时触发，或在一个或多个指定表的列发生更新时触发。
- SQLite 只支持 FOR EACH ROW 触发器（Trigger），没有 FOR EACH STATEMENT 触发器（Trigger）。因此，明确指定 FOR EACH ROW 是可选的。
- WHEN 子句和触发器（Trigger）动作可能访问使用表单 **NEW.column-name** 和 **OLD.column-name** 的引用插入、删除或更新的行元素，其中 column-name 是从与触发器关联的表的列的名称。
- 如果提供 WHEN 子句，则只针对 WHEN 子句为真的指定行执行 SQL 语句。如果没有提供 WHEN 子句，则针对所有行执行 SQL 语句。
- BEFORE 或 AFTER 关键字决定何时执行触发器动作，决定是在关联行的插入、修改或删除之前或者之后执行触发器动作。
- 当触发器相关联的表删除时，自动删除触发器（Trigger）。
- 要修改的表必须存在于同一数据库中，作为触发器被附加的表或视图，且必须只使用 **tablename**，而不是 **database.tablename**。
- 一个特殊的 SQL 函数 RAISE() 可用于触发器程序内抛出异常。

创建 **触发器（Trigger）** 的基本语法如下：

```
CREATE  TRIGGER trigger_name [BEFORE|AFTER] event_name
ON table_name
BEGIN
 -- 触发器逻辑....
END;
```

在这里，**event_name** 可以是在所提到的表 **table_name** 上的 *INSERT*、*DELETE* 和 *UPDATE* 数据库操作。您可以在表名后选择指定 FOR EACH ROW。

以下是在 UPDATE 操作上在表的一个或多个指定列上创建触发器（Trigger）的语法：

```
CREATE  TRIGGER trigger_name [BEFORE|AFTER] UPDATE OF column_name
ON table_name
BEGIN
 -- 触发器逻辑....
END;
--示例
create table log(
    id int not null,
    entry_date text not null
);
create trigger user_log after insert
on user
begin
    insert into log(id,entry_date) values(new.ID,datetime('now'))
```

列出触发器（TRIGGERS）

您可以从 **sqlite_master** 表中列出所有触发器，如下所示：

```
sqlite> SELECT name FROM sqlite_master
WHERE type = 'trigger';
```

删除触发器（TRIGGERS）

下面是 DROP 命令，可用于删除已有的触发器：

```
DROP TRIGGER trigger_name;
```

sqlite函数

SQLite 日期 & 时间

SQLite 支持以下五个日期和时间函数：

序号	函数	实例
1	date(timestring, modifier, modifier, ...)	以 YYYY-MM-DD 格式返回日期。
2	time(timestring, modifier, modifier, ...)	以 HH:MM:SS 格式返回时间。
3	datetime(timestring, modifier, modifier, ...)	以 YYYY-MM-DD HH:MM:SS 格式返回。
4	julianday(timestring, modifier, modifier, ...)	这将返回从格林尼治时间的公元前 4714 年 11 月 24 日正午算起的天数。
5	strftime(format, timestring, modifier, modifier, ...)	这将根据第一个参数指定的格式字符串返回格式化的日期。具体格式见下边讲解。

SQLite 常用函数

SQLite 有许多内置函数用于处理字符串或数字数据。下面列出了一些有用的 SQLite 内置函数，且所有函数都是大小写不敏感，这意味着您可以使用这些函数的小写形式或大写形式或混合形式。欲了解更多详情，请查看 SQLite 的官方文档：

序号	函数 & 描述
1	**SQLite COUNT 函数** SQLite COUNT 聚集函数是用来计算一个数据库表中的行数。
2	**SQLite MAX 函数** SQLite MAX 聚合函数允许我们选择某列的最大值。
3	**SQLite MIN 函数** SQLite MIN 聚合函数允许我们选择某列的最小值。
4	**SQLite AVG 函数** SQLite AVG 聚合函数计算某列的平均值。
5	**SQLite SUM 函数** SQLite SUM 聚合函数允许为一个数值列计算总和。
6	**SQLite RANDOM 函数** SQLite RANDOM 函数返回一个介于 -9223372036854775808 和 +9223372036854775807 之间的伪随机整数。
7	**SQLite ABS 函数** SQLite ABS 函数返回数值参数的绝对值。
8	**SQLite UPPER 函数** SQLite UPPER 函数把字符串转换为大写字母。
9	**SQLite LOWER 函数** SQLite LOWER 函数把字符串转换为小写字母。
10	**SQLite LENGTH 函数** SQLite LENGTH 函数返回字符串的长度。
11	**SQLite sqlite_version 函数** SQLite sqlite_version 函数返回 SQLite 库的版本。