SQLite 基础

SQLite 教程 | 菜鸟教程

1. SQLite 数据类型

1.1 SQLite 存储类

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 的存储类比数据类型更普遍。INTEGER 存储类，例如，包含6种不同的不同长度的整数数据类型。

1.2 SQLite 亲和（Affinity）类型

SQLite支持列的亲和类型概念。任何列仍然可以存储任何类型的数据，当数据插入时，该字段的数据将会优先采用亲缘类型作为该值的存储方式。SQLite目前的版本支持以下五种亲缘类型：

亲和类型	描述
TEXT	数值型数据在被插入之前，需要先被转换为文本格式，之后再插入到目标字段中。
NUMERIC	当文本数据被插入到亲缘性为NUMERIC的字段中时，如果转换操作不会导致数据信息丢失以及完全可逆，那么SQLite就会将该文本数据转换为INTEGER或REAL类型的数据，如

	果转换失败，SQLite仍会以TEXT方式存储该数据。对于NULL或BLOB类型的新数据，SQLite将不做任何转换，直接以NULL或BLOB的方式存储该数据。需要额外说明的是，对于浮点格式的常量文本，如"30000.0"，如果该值可以转换为INTEGER同时又不会丢失数值信息，那么SQLite就会将其转换为INTEGER的存储方式。
INTEGER	对于亲缘类型为INTEGER的字段，其规则等同于NUMERIC，唯一差别是在执行CAST表达式时。
REAL	其规则基本等同于NUMERIC，唯一的差别是不会将"30000.0"这样的文本数据转换为INTEGER存储方式。
NONE	不做任何的转换，直接以该数据所属的数据类型进行存储。

1.3 SQLite 亲和类型(Affinity)及类型名称

下表列出了当创建 SQLite3 表时可使用的各种数据类型名称，同时也显示了相应的亲和类型：

数据类型	亲和类型
INT、INTEGER、TINYINT、SMALLINT、MEDIUMINT、BIGINT、UNSIGNED BIG INT、INT2、INT8	INTEGER
CHARACTER(20)、VARCHAR(255)、VARYING CHARACTER(255)、NCHAR(55)、NATIVE CHARACTER(70)、NVARCHAR(100)、TEXT、CLOB	TEXT
BLOB、未指定类型	BLOB
REAL、DOUBLE、DOUBLE PRECISION、FLOAT	REAL
NUMERIC、DECIMAL(10,5)、BOOLEAN、DATE、DATETIME	NUMERIC

1.4 Boolean 数据类型

SQLite 没有单独的 Boolean 存储类。相反，布尔值被存储为整数 0（false）和 1（true）。

1.5 Date 与 Time 数据类型

SQLite 没有一个单独的用于存储日期和/或时间的存储类，但 SQLite 能够把日期和时间存储为 TEXT、REAL 或 INTEGER 值。

存储类	日期格式
TEXT	格式为 "YYYY-MM-DD HH:MM:SS.SSS" 的日期。
REAL	从公元前 4714 年 11 月 24 日格林尼治时间的正午开始算起的天数。
INTEGER	从 1970-01-01 00:00:00 UTC 算起的秒数。

您可以以任何上述格式来存储日期和时间，并且可以使用内置的日期和时间函数来自由转换不同格式。

2. 创建数据库

sqlite3 命令被用来创建新的 SQLite 数据库。不需要任何特殊的权限即可创建一个数据。

2.1 语法

```
$ sqlite3 DatabaseName.db
```

通常情况下，数据库名称在 RDBMS（关系数据库管理系统）内应该是唯一的。

也可以使用 **.open** 来建立新的数据库文件：

```
sqlite> .open test.db
```

2.2 实例

```
$ sqlite3 testDB.db
SQLite version 3.7.15.2 2013-01-09 11:53:05
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite>


// 一旦数据库被创建，就可以使用 SQLite 的 .databases 命令来检查它是否在数据库列表中
sqlite>.databases
seq  name             file
---  ---------------  ----------------------
0    main             /home/sqlite/testDB.db

```

```
14
15 使用 SQLite .quit 命令退出 sqlite 提示符
16 sqlite>.quit
17 $
```

2.3 .dump 命令

可以在命令提示符中使用 SQLite **.dump** 点命令来导出完整的数据库在一个文本文件中

```
1 $sqlite3 testDB.db .dump > testDB.sql
```

面的命令将转换整个 **testDB.db** 数据库的内容到 SQLite 的语句中，并将其转储到 ASCII 文本文件 **testDB.sql** 中。您可以通过简单的方式从生成的 testDB.sql 恢复

```
1 $sqlite3 testDB.db < testDB.sql
```

3. 附加（Attach）数据库

当在同一时间有多个数据库可用，您想使用其中的任何一个。SQLite 的 **ATTACH DATABASE** 语句是用来选择一个特定的数据库，使用该命令后，所有的 SQLite 语句将在附加的数据库下执行。

3.1 语法

ATTACH DATABASE 语句的基本语法如下：

```
1 ATTACH DATABASE file_name AS database_name;
```

如果数据库尚未被创建，上面的命令将创建一个数据库，如果数据库已存在，则把数据库文件名称与逻辑数据库 'Alias-Name' 绑定在一起。

打开的数据库和使用 ATTACH附加进来的数据库的必须位于同一文件夹下。

3.2 实例

如果想附加一个现有的数据库 **testDB.db**，则 ATTACH DATABASE 语句将如下所示：

```
sqlite> ATTACH DATABASE 'testDB.db' as 'TEST';
```

使用 SQLite **.database** 命令来显示附加的数据库。

```
sqlite> .database
seq  name             file
---  ---------------  ----------------------
0    main             /home/sqlite/testDB.db
2    test             /home/sqlite/testDB.db
```

数据库名称 **main** 和 **temp** 被保留用于主数据库和存储临时表及其他临时数据对象的数据库。这两个数据库名称可用于每个数据库连接，且不应该被用于附加，否则将得到一个警告消息，如下所示：

```
sqlite>  ATTACH DATABASE 'testDB.db' as 'TEMP';
Error: database TEMP is already in use
sqlite>  ATTACH DATABASE 'testDB.db' as 'main';
Error: database main is already in use;
```

4. 分离（Detach）数据库

DETACH DATABASE *语句是用来把命名数据库从一个数据库连接分离和游离出来，连接是之前使用 ATTACH 语句附加的。如果同一个数据库文件已经被附加上多个别名，DETACH 命令将只断开给定名称的连接，而其余的仍然有效。您无法分离* ***main*** *或* ***temp*** *数据库。*

如果数据库是在内存中或者是临时数据库，则该数据库将被摧毁，且内容将会丢失。

4.1 语法

```
DETACH DATABASE 'Alias-Name';
```

4.2 实例

已经创建了一个数据库，并给它附加了 'test' 和 'currentDB'，使用 .database 命令，我们可以看到：

```
sqlite>.databases
seq  name             file
```

```
3  ---  ---------------  ----------------------
4  0    main             /home/sqlite/testDB.db
5  2    test             /home/sqlite/testDB.db
6  3    currentDB        /home/sqlite/testDB.db
```

把 'currentDB' 从 testDB.db 中分离出来

```
1  sqlite> DETACH DATABASE 'currentDB';
```

检查当前附加的数据库，您会发现，testDB.db 仍与 'test' 和 'main' 保持连接

```
1  sqlite>.databases
2  seq  name             file
3  ---  ---------------  ----------------------
4  0    main             /home/sqlite/testDB.db
5  2    test             /home/sqlite/testDB.db
```

5. 创建表

5.1 语法

CREATE TABLE 语句的基本语法

```
1  CREATE TABLE database_name.table_name(
2     column1 datatype  PRIMARY KEY(one or more columns),
3     column2 datatype,
4     column3 datatype,
5     .....
6     columnN datatype,
7  );
```

5.2 实例

建了一个 COMPANY 表，ID 作为主键，NOT NULL 的约束表示在表中创建纪录时这些字段不能为 NULL：

```
sqlite> CREATE TABLE COMPANY(
   ID INT PRIMARY KEY     NOT NULL,
   NAME           TEXT    NOT NULL,
   AGE            INT     NOT NULL,
   ADDRESS        CHAR(50),
   SALARY         REAL
);
```

再创建一个表

```
sqlite> CREATE TABLE DEPARTMENT(
   ID INT PRIMARY KEY      NOT NULL,
   DEPT           CHAR(50) NOT NULL,
   EMP_ID         INT      NOT NULL
);
```

.tables 命令来验证表是否已成功创建，该命令用于列出附加数据库中的所有表。

```
sqlite>.tables
COMPANY     DEPARTMENT
```

.schema 命令得到表的完整信息

```
sqlite>.schema COMPANY
CREATE TABLE COMPANY(
   ID INT PRIMARY KEY     NOT NULL,
   NAME           TEXT    NOT NULL,
   AGE            INT     NOT NULL,
   ADDRESS        CHAR(50),
   SALARY         REAL
);

```

6. 删除表

***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。*

6.1 语法

```
DROP TABLE database_name.table_name;
```

6.2 实例

```
sqlite>DROP TABLE COMPANY;
```

7. Insert 语句

INSERT INTO 语句用于向数据库的某个表中添加新的数据行。

7.1 语法

INSERT INTO 语句有两种基本语法

```
INSERT INTO TABLE_NAME [(column1, column2, column3,...columnN)]
VALUES (value1, value2, value3,...valueN);
```

column1, column2,...columnN 是要插入数据的表中的列的名称。

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。但要确保值的顺序与列在表中的顺序一致。SQLite 的 INSERT INTO 语法如下：

```
INSERT INTO TABLE_NAME VALUES (value1,value2,value3,...valueN);
```

7.2 实例

在 COMPANY 表中创建六个记录：

```
INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (1, 'Paul', 32, 'California', 20000.00 );

```

```
INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (2, 'Allen', 25, 'Texas', 15000.00 );

INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (3, 'Teddy', 23, 'Norway', 20000.00 );

INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (4, 'Mark', 25, 'Rich-Mond ', 65000.00 );

INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (5, 'David', 27, 'Texas', 85000.00 );

INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (6, 'Kim', 22, 'South-Hall', 45000.00 );

```

使用第二种语法在 COMPANY 表中创建一个记录

```
INSERT INTO COMPANY VALUES (7, 'James', 24, 'Houston', 10000.00 );
```

上面的所有语句将在 COMPANY 表中创建下列记录。

```
ID          NAME        AGE         ADDRESS     SALARY
----------  ----------  ----------  ----------  ----------
1           Paul        32          California  20000.0
2           Allen       25          Texas       15000.0
3           Teddy       23          Norway      20000.0
4           Mark        25          Rich-Mond   65000.0
5           David       27          Texas       85000.0
6           Kim         22          South-Hall  45000.0
7           James       24          Houston     10000.0
```

7.3 使用一个表来填充另一个表

可以通过在一个有一组字段的表上使用 select 语句，填充数据到另一个表中。

```
INSERT INTO first_table_name [(column1, column2, ... columnN)]
   SELECT column1, column2, ...columnN
   FROM second_table_name
   [WHERE condition];
```

8. Select 语句

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。

8.1 语法

```
SELECT column1, column2, columnN FROM table_name;
```

column1, column2...是表的字段，他们的值即是您要获取的。如果您想获取所有可用的字段

```
SELECT * FROM table_name;
```

8.2 实例

使用 SELECT 语句获取并显示所有这些记录。在这里，前两个命令被用来设置正确格式化的输出。

```
sqlite>.header on
sqlite>.mode column
sqlite> SELECT * FROM COMPANY;
```

如果只想获取 COMPANY 表中指定的字段，则使用下面的查询：

```
sqlite> SELECT ID, NAME, SALARY FROM COMPANY;
```

上面的查询会产生以下结果：

```
ID          NAME        SALARY
----------  ----------  ----------
1           Paul        20000.0
2           Allen       15000.0
3           Teddy       20000.0
4           Mark        65000.0
5           David       85000.0
```

```
8 6           Kim         45000.0
9 7           James       10000.0
```

8.3 设定输出列的宽度

有时，由于要显示的列的默认宽度导致 **.mode column**，这种情况下，输出被截断。此时，您可以使用 **.width num, num....** 命令设置显示列的宽度，如下所示：

```
1 sqlite>.width 10, 20, 10
2 sqlite>SELECT * FROM COMPANY;
```

.width 命令设置第一列的宽度为 10，第二列的宽度为 20，第三列的宽度为 10。因此上述 SELECT 语句将得到以下结果：

```
1 ID          NAME                  AGE         ADDRESS     SALARY
2 ----------  --------------------  ----------  ----------  ----------
3 1           Paul                  32          California  20000.0
4 2           Allen                 25          Texas       15000.0
5 3           Teddy                 23          Norway      20000.0
6 4           Mark                  25          Rich-Mond   65000.0
7 5           David                 27          Texas       85000.0
8 6           Kim                   22          South-Hall  45000.0
9 7           James                 24          Houston     10000.0
```

8.4 Schema 信息

因为所有的**点命令**只在 SQLite 提示符中可用，所以当您进行带有 SQLite 的编程时，您要使用下面的带有 **sqlite_master** 表的 SELECT 语句来列出所有在数据库中创建的表：

```
1 sqlite> SELECT tbl_name FROM sqlite_master WHERE type = 'table';
```

假设在 testDB.db 中已经存在唯一的 COMPANY 表，则将产生以下结果：

```
1 tbl_name
2 ----------
3 COMPANY
```

可以列出关于 COMPANY 表的完整信息

```
sqlite> SELECT sql FROM sqlite_master WHERE type = 'table' AND tbl_name =
'COMPANY';
```

假设在 testDB.db 中已经存在唯一的 COMPANY 表，则将产生以下结果：

```
CREATE TABLE COMPANY(
   ID INT PRIMARY KEY     NOT NULL,
   NAME           TEXT    NOT NULL,
   AGE            INT     NOT NULL,
   ADDRESS        CHAR(50),
   SALARY         REAL
)
```

9. 运算符

9.1 SQLite 运算符是什么?

运算符是一个保留字或字符，主要用于 SQLite 语句的 WHERE 子句中执行操作，如比较和算术运算。

运算符用于指定 SQLite 语句中的条件，并在语句中连接多个条件。

- 算术运算符
- 比较运算符
- 逻辑运算符
- 位运算符

9.2 SQLite 算术运算符

假设变量 a=10，变量 b=20

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 将得到 0

实例

```
sqlite> .mode line
sqlite> select 10 + 20;
10 + 20 = 30


sqlite> select 10 - 20;
10 - 20 = -10


sqlite> select 10 * 20;
10 * 20 = 200


sqlite> select 10 / 5;
10 / 5 = 2


sqlite> select 12 %  5;
12 %  5 = 2
```

9.3 SQLite 比较运算符

假设变量 a=10，变量 b=20

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。

实例

COMPANY 表有以下记录：

```
ID          NAME        AGE         ADDRESS     SALARY
----------  ----------  ----------  ----------  ----------
1           Paul        32          California  20000.0
2           Allen       25          Texas       15000.0
3           Teddy       23          Norway      20000.0
4           Mark        25          Rich-Mond   65000.0
5           David       27          Texas       85000.0
6           Kim         22          South-Hall  45000.0
7           James       24          Houston     10000.0
```

下面的实例演示了各种 SQLite 比较运算符的用法。WHERE 子句是用来设置 SELECT 语句的条件语句。

下面的 SELECT 语句列出了 SALARY 大于 50,000.00 的所有记录：

```
sqlite> SELECT * FROM COMPANY WHERE SALARY > 50000;
ID          NAME        AGE         ADDRESS     SALARY
----------  ----------  ----------  ----------  ----------
4           Mark        25          Rich-Mond   65000.0
5           David       27          Texas       85000.0
```

9.4 SQLite 逻辑运算符

下面是 SQLite 中所有的逻辑运算符列表。

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）。

实例

下面的实例演示了 SQLite 逻辑运算符的用法。

下面的 SELECT 语句列出了 AGE 大于等于 25 **且**工资大于等于 65000.00 的所有记录：

```
sqlite> SELECT * FROM COMPANY WHERE AGE >= 25 AND SALARY >= 65000;
ID          NAME        AGE         ADDRESS     SALARY
----------  ----------  ----------  ----------  ----------
4           Mark        25          Rich-Mond   65000.0
5           David       27          Texas       85000.0
```

下面的 SELECT 语句列出了 NAME 以 'Ki' 开始的所有记录，'Ki' 之后的字符不做限制：

```
sqlite> SELECT * FROM COMPANY WHERE NAME LIKE 'Ki%';
ID          NAME        AGE         ADDRESS     SALARY
----------  ----------  ----------  ----------  ----------
```

```
4  6           Kim         22          South-Hall  45000.0
```

下面的 SELECT 语句列出了 AGE 的值为 25 或 27 的所有记录：

```
1  sqlite> SELECT * FROM COMPANY WHERE AGE IN ( 25, 27 );
2  ID          NAME        AGE         ADDRESS     SALARY
3  ----------  ----------  ----------  ----------  ----------
4  2           Allen       25          Texas       15000.0
5  4           Mark        25          Rich-Mond   65000.0
6  5           David       27          Texas       85000.0
```

下面的 SELECT 语句列出了 AGE 的值在 25 与 27 之间的所有记录：

```
1  sqlite> SELECT * FROM COMPANY WHERE AGE BETWEEN 25 AND 27;
2  ID          NAME        AGE         ADDRESS     SALARY
3  ----------  ----------  ----------  ----------  ----------
4  2           Allen       25          Texas       15000.0
5  4           Mark        25          Rich-Mond   65000.0
6  5           David       27          Texas       85000.0
```

下面的 SELECT 语句使用 SQL 子查询，子查询查找 SALARY > 65000 的带有 AGE 字段的所有记录，后边的 WHERE 子句与 EXISTS 运算符一起使用，列出了外查询中的 AGE 存在于子查询返回的结果中的所有记录：

```
1  sqlite> SELECT AGE FROM COMPANY
2          WHERE EXISTS (SELECT AGE FROM COMPANY WHERE SALARY > 65000);
3  AGE
4  ----------
5  32
6  25
7  23
8  25
9  27
10 22
11 24
```

下面的 SELECT 语句使用 SQL 子查询，子查询查找 SALARY > 65000 的带有 AGE 字段的所有记录，后边的 WHERE 子句与 > 运算符一起使用，列出了外查询中的 AGE 大于子查询返回的结果中的年龄的所有记录：

```
sqlite> SELECT * FROM COMPANY
        WHERE AGE > (SELECT AGE FROM COMPANY WHERE SALARY > 65000);
ID          NAME        AGE         ADDRESS     SALARY
----------  ----------  ----------  ----------  ----------
1           Paul        32          California  20000.0
```

9.5 SQLite 位运算符

位运算符作用于位，并逐位执行操作。真值表 & 和 | 如下：

p	q	p & q	p | q
0	0	0	0
0	1	0	1
1	1	1	1
1	0	0	1

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

实例

```
sqlite> .mode line
sqlite> select 60 | 13;
60 | 13 = 61

sqlite> select 60 & 13;
60 & 13 = 12
```

```
sqlite>  select  (~60);
(~60) = -61

sqlite>  select  (60 << 2);
(60 << 2) = 240

sqlite>  select  (60 >> 2);
(60 >> 2) = 15
```

10. 表达式

表达式是一个或多个值、运算符和计算值的SQL函数的组合。

SQL 表达式与公式类似，都写在查询语言中。您还可以使用特定的数据集来查询数据库。

10.1 语法

假设 SELECT 语句的基本语法如下：

```
SELECT column1, column2, columnN
FROM table_name
WHERE [CONDITION | EXPRESSION];
```

有不同类型的 SQLite 表达式，具体讲解如下：

10.2 布尔表达式

SQLite 的布尔表达式在匹配单个值的基础上获取数据。语法如下：

```
SELECT column1, column2, columnN
FROM table_name
WHERE SINGLE VALUE MATCHING EXPRESSION;
```

下面的实例演示了 SQLite 布尔表达式的用法：

```
sqlite> SELECT * FROM COMPANY WHERE SALARY = 10000;
ID          NAME        AGE         ADDRESS     SALARY
```

```
3 ----------  ----------  ----------  ----------  ----------
4 4           James        24          Houston    10000.0
```

10.3 数值表达式

这些表达式用来执行查询中的任何数学运算。语法如下：

```
1 SELECT numerical_expression as  OPERATION_NAME
2 [FROM table_name WHERE CONDITION] ;
```

在这里，numerical_expression 用于数学表达式或任何公式。下面的实例演示了 SQLite 数值表达式的用法：

```
1 sqlite> SELECT (15 + 6) AS ADDITION
2 ADDITION = 21
```

有几个内置的函数，比如 avg()、sum()、count()，等等，执行被称为对一个表或一个特定的表列的汇总数据计算。

```
1 sqlite> SELECT COUNT(*) AS "RECORDS" FROM COMPANY;
2 RECORDS = 7
```

10.4 日期表达式

日期表达式返回当前系统日期和时间值，这些表达式将被用于各种数据操作。

```
1 sqlite>  SELECT CURRENT_TIMESTAMP;
2 CURRENT_TIMESTAMP = 2013-03-17 10:43:35
```

current_timestamp 得到的时间，时区不对，要想得到本地时间，试试下面这个：

```
1 sqlite> select datetime('now','localtime');
2 datetime('now','localtime') = 2018-09-13 16:38:32
```

11. Where 子句

***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。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

11.1 语法

带有 WHERE 子句的 SELECT 语句的基本语法如下：

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition]
```

11.2 实例

还可以使用比较或逻辑运算符指定条件，比如 >、<、=、LIKE、NOT，等等。

下面的 SELECT 语句列出了 AGE 大于等于 25 **且**工资大于等于 65000.00 的所有记录：

```
sqlite> SELECT * FROM COMPANY WHERE AGE >= 25 AND SALARY >= 65000;
ID          NAME        AGE         ADDRESS     SALARY
----------  ----------  ----------  ----------  ----------
4           Mark        25          Rich-Mond   65000.0
5           David       27          Texas       85000.0
```

下面的 SELECT 语句列出了 AGE 不为 NULL 的所有记录

```
sqlite>  SELECT * FROM COMPANY WHERE AGE IS NOT NULL;
```

下面的 SELECT 语句列出了 NAME 以 'Ki' 开始的所有记录，'Ki' 之后的字符不做限制：

```
sqlite> SELECT * FROM COMPANY WHERE NAME LIKE 'Ki%';
ID          NAME        AGE         ADDRESS     SALARY
----------  ----------  ----------  ----------  ----------
6           Kim         22          South-Hall  45000.0
```

下面的 SELECT 语句使用 SQL 子查询，子查询查找 SALARY > 65000 的带有 AGE 字段的所有记录，后边的 WHERE 子句与 EXISTS 运算符一起使用，列出了外查询中的 AGE 存在于子查询返回的结果中的所有记录：

```
sqlite> SELECT AGE FROM COMPANY
        WHERE EXISTS (SELECT AGE FROM COMPANY WHERE SALARY > 65000);
AGE
----------
32
25
23
25
27
22
24
```

下面的 SELECT 语句使用 SQL 子查询，子查询查找 SALARY > 65000 的带有 AGE 字段的所有记录，后边的 WHERE 子句与 > 运算符一起使用，列出了外查询中的 AGE 大于子查询返回的结果中的年龄的所有记录：

```
sqlite> SELECT * FROM COMPANY
        WHERE AGE > (SELECT AGE FROM COMPANY WHERE SALARY > 65000);
ID          NAME        AGE         ADDRESS     SALARY
----------  ----------  ----------  ----------  ----------
1           Paul        32          California  20000.0
```

12. AND/OR 运算符

***AND** 和 **OR** 运算符用于编译多个条件来缩小在 SQLite 语句中所选的数据。这两个运算符被称为连接运算符。*

12.1 AND 运算符

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。

12.1.1 语法

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition1] AND [condition2]...AND [conditionN];
```

12.1.2 实例

12.2 OR 运算符

12.2.1 语法

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition1] OR [condition2]...OR [conditionN]
```

12.2.2 实例

13. Update 语句

***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。*

13.1 语法

带有 WHERE 子句的 UPDATE 查询的基本语法如下：

```
UPDATE table_name
SET column1 = value1, column2 = value2...., columnN = valueN
WHERE [condition];
```

13.2 实例

更新 ID 为 6 的客户地址：

```
sqlite> UPDATE COMPANY SET ADDRESS = 'Texas' WHERE ID = 6;
```

14. Delete 语句

***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。*

14.1 语法

带有 WHERE 子句的 DELETE 查询的基本语法如下：

```
DELETE FROM table_name
WHERE [condition];
```

14.2 实例

它会删除 ID 为 7 的客户：

```
sqlite> DELETE FROM COMPANY WHERE ID = 7;
```

15. Like 子句

***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。

15.1 语法

% 和 _ 的基本语法如下：

```
SELECT column_list
FROM table_name
WHERE column LIKE 'XXXX%'

or

```

```
SELECT column_list
FROM table_name
WHERE column LIKE '%XXXX%'

or

SELECT column_list
FROM table_name
WHERE column LIKE 'XXXX_'

or

SELECT column_list
FROM table_name
WHERE column LIKE '_XXXX'

or

SELECT column_list
FROM table_name
WHERE column LIKE '_XXXX_'
```

15.2 实例

下面一些实例演示了 带有 '%' 和 '_' 运算符的 LIKE 子句不同的地方：

语句	描述
WHERE SALARY LIKE '200%'	查找以 200 开头的任意值
WHERE SALARY LIKE '%200%'	查找任意位置包含 200 的任意值
WHERE SALARY LIKE '_00%'	查找第二位和第三位为 00 的任意值
WHERE SALARY LIKE '2_%_%'	查找以 2 开头，且长度至少为 3 个字符的任意值
WHERE SALARY LIKE '%2'	查找以 2 结尾的任意值
WHERE SALARY LIKE '_2%3'	查找第二位为 2，且以 3 结尾的任意值
WHERE SALARY LIKE '2___3'	查找长度为 5 位数，且以 2 开头以 3 结尾的任意值

它显示 COMPANY 表中 AGE 以 2 开头的所有记录：

```
sqlite> SELECT * FROM COMPANY WHERE AGE  LIKE '2%';
```

16. Glob 子句

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。

- `*`：匹配零个、一个或多个数字或字符。
- `?`：代表一个单一的数字或字符。
- `[...]`：匹配方括号内指定的字符之一。例如，`[abc]` 匹配 "a"、"b" 或 "c" 中的任何一个字符。
- `[^...]`：匹配不在方括号内指定的字符之一。例如，`[^abc]` 匹配不是 "a"、"b" 或 "c" 中的任何一个字符的字符。

以上这些符号可以被组合使用。

16.1 语法

`*` 和 `?` 的基本语法如下：

```
SELECT FROM table_name
WHERE column GLOB 'XXXX*'

or

SELECT FROM table_name
WHERE column GLOB '*XXXX*'

or

SELECT FROM table_name
WHERE column GLOB 'XXXX?'

or

SELECT FROM table_name
WHERE column GLOB '?XXXX'

or

SELECT FROM table_name
WHERE column GLOB '?XXXX?'

or

SELECT FROM table_name
```

```
27 WHERE column GLOB '????'
```

16.2 实例

下面一些实例演示了 带有 '*' 和 '?' 运算符的 GLOB 子句不同的地方：

语句	描述
WHERE SALARY GLOB '200*'	查找以 200 开头的任意值
WHERE SALARY GLOB '*200*'	查找任意位置包含 200 的任意值
WHERE SALARY GLOB '?00*'	查找第二位和第三位为 00 的任意值
WHERE SALARY GLOB '2??'	查找以 2 开头，且长度为 3 个字符的任意值，例如，它可能匹配 "200"、"2A1"、"2B2" 等值。
WHERE SALARY GLOB '*2'	查找以 2 结尾的任意值
WHERE SALARY GLOB '?2*3'	查找第二位为 2，且以 3 结尾的任意值
WHERE SALARY GLOB '2???3'	查找长度为 5 位数，且以 2 开头以 3 结尾的任意值

[...] 表达式用于匹配方括号内指定的字符集中的任何一个字符。

实例 1：匹配以 "A" 或 "B" 开头的产品名称。

```
1 SELECT * FROM products WHERE product_name LIKE '[AB]%';
```

[^...] 表达式用于匹配不在方括号内指定的字符集中的任何字符。

实例 1：匹配不以 "X" 或 "Y" 开头的产品代码。

```
1 SELECT * FROM products WHERE product_code LIKE '[^XY]%';
```

实例 2：匹配不包含数字字符的用户名。

```
1 SELECT * FROM users WHERE username LIKE '[^0-9]%';
```

17. Limit 子句

LIMIT 子句用于限制由 SELECT 语句返回的数据数量。

17.1 语法

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]
```

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]
```

SQLite 引擎将返回从下一行开始直到给定的 OFFSET 为止的所有行，如下面的最后一个实例所示。

17.2 实例

它限制了您想要从表中提取的行数：

```
sqlite> SELECT * FROM COMPANY LIMIT 6;
```

但是，在某些情况下，可能需要从一个特定的偏移开始提取记录。下面是一个实例，从第三位开始提取 3 个记录：

```
sqlite> SELECT * FROM COMPANY LIMIT 3 OFFSET 2;
```

18. Order By

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。

18.1 语法

ORDER BY 子句的基本语法如下：

```
SELECT column-list
FROM table_name
[WHERE condition]
[ORDER BY column1, column2, .. columnN] [ASC | DESC];
```

- **ASC** 默认值，从小到大，升序排列
- **DESC** 从大到小，降序排列

您可以在 ORDER BY 子句中使用多个列，确保您使用的排序列在列清单中：

```
SELECT
   select_list
FROM
   table
ORDER BY
    column_1 ASC,
    column_2 DESC;
```

column_1 与 column_2 如果后面不指定排序规则，默认为 ASC 升序，以上语句按 column_1 升序，column_2 降序读取，等价如下语句：

```
SELECT
   select_list
FROM
   table
ORDER BY
    column_1,
    column_2 DESC;
```

18.2 实例

COMPANY 表有以下记录：

```
ID          NAME        AGE         ADDRESS     SALARY
----------  ----------  ----------  ----------  ----------
1           Paul        32          California  20000.0
2           Allen       25          Texas       15000.0
3           Teddy       23          Norway      20000.0
4           Mark        25          Rich-Mond   65000.0
5           David       27          Texas       85000.0
```

```
8 6           Kim         22          South-Hall  45000.0
9 7           James       24          Houston     10000.0
```

它会将结果按 SALARY 升序排序：

```
1 sqlite> SELECT * FROM COMPANY ORDER BY SALARY ASC;
```

19. Group By

***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。

19.1 语法

下面给出了 GROUP BY 子句的基本语法。GROUP BY 子句必须放在 WHERE 子句中的条件之后，必须放在 ORDER BY 子句之前。

```
1 SELECT column-list
2 FROM table_name
3 WHERE [ conditions ]
4 GROUP BY column1, column2....columnN
5 ORDER BY column1, column2....columnN
```

可以在 GROUP BY 子句中使用多个列。确保您使用的分组列在列清单中。

19.2 实例

使用下面的 INSERT 语句在 COMPANY 表中另外创建三个记录：

```
1 INSERT INTO COMPANY VALUES (8, 'Paul', 24, 'Houston', 20000.00 );
2 INSERT INTO COMPANY VALUES (9, 'James', 44, 'Norway', 5000.00 );
3 INSERT INTO COMPANY VALUES (10, 'James', 45, 'Texas', 5000.00 );
```

现在，我们的表具有重复名称的记录，如下所示：

```
1 ID          NAME        AGE         ADDRESS     SALARY
```

```
----------  ----------  ----------  ----------  ----------
1           Paul        32          California  20000.0
2           Allen       25          Texas       15000.0
3           Teddy       23          Norway      20000.0
4           Mark        25          Rich-Mond   65000.0
5           David       27          Texas       85000.0
6           Kim         22          South-Hall  45000.0
7           James       24          Houston     10000.0
8           Paul        24          Houston     20000.0
9           James       44          Norway      5000.0
10          James       45          Texas       5000.0
```

如果您想了解每个客户的工资总额，则可使用 GROUP BY 查询，如下所示：

```
sqlite> SELECT NAME, SUM(SALARY) FROM COMPANY GROUP BY NAME ORDER BY NAME;
```

20. Having 子句

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。

20.1 语法

下面是 HAVING 子句在 SELECT 查询中的位置：

```
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
```

20.2 实例

它将显示名称计数小于 2 的所有记录：

```
sqlite > SELECT * FROM COMPANY GROUP BY name HAVING count(name) < 2;
```

21. Distinct 关键字

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。*

21.1 语法

用于消除重复记录的 DISTINCT 关键字的基本语法如下：

```
SELECT DISTINCT column1, column2,.....columnN
FROM table_name
WHERE [condition]
```

21.2 实例

SELECT 查询中使用 DISTINCT 关键字：

```
sqlite> SELECT DISTINCT name FROM COMPANY;
```