一、什么是SQLite

1.什么是SQLite

SQLite是一个进程内的库，实现了自给自足的、无服务器的、零配置的、事务性的 SQL 数据库引擎。它是一个零配置的数据库，这意味着与其他数据库一样，您不需要在系统中配置。嵌入式数据库，可以被用作做缓存，例如Firefox, Chrome, 微信等用作客户端缓存数据库，客户端缓存了数据，就不需要去服务端取数据了，减少了服务器压力，缓存数据库的叫法就是这么来的。

就像其他数据库，SQLite 引擎不是一个独立的进程，可以按应用程序需求进行静态或动态连接。SQLite 直接访问其存储文件。

2.为什么要用SQLite

- 不需要一个单独的服务器进程或操作的系统（无服务器的）。
- SQLite 不需要配置，这意味着不需要安装或管理。
- 一个完整的 SQLite 数据库是存储在一个单一的跨平台的磁盘文件。
- SQLite 是非常小的，是轻量级的，完全配置时小于 400KiB，省略可选功能配置时小于250KiB。
- SQLite 是自给自足的，这意味着不需要任何外部的依赖。
- SQLite 事务是完全兼容 ACID 的，允许从多个进程或线程安全访问。
- SQLite 支持 SQL92（SQL2）标准的大多数查询语言的功能。
- SQLite 使用 ANSI-C 编写的，并提供了简单和易于使用的 API。
- SQLite 可在 UNIX（Linux, Mac OS-X, Android, iOS）和 Windows（Win32, WinCE, WinRT）中运行。

从 2000 年 5 月 29 日开始，SQLite 就选择了 C 语言。直到今天，C 也是实现 SQLite 这样软件库的最佳语言。

C 语言是实现 SQLite 最好的语言的原因包括：

- 性能。
- 兼容性。
- 低依赖性。
- 稳定性。

性能

像 SQLite 这样被密集使用的基础库需要有很好的性能（SQLite 确实很快，可以看看 Internal Versus External BLOBs 和 35% Faster Than The Filesystem 两篇文章）。

C 语言很适合写这样有性能要求的程序。C 语言有时被称为「便携式汇编语言」，让开发者能尽可能的接近底层硬件编码，同时保证跨平台的便携性。

当然，也有其他的编程语言声称和 C 一样快或者更快，但没有一个能和 C 一样通用。

兼容性

目前几乎所有的系统都可以调用由 C 语言编写的库。

比如，用 Java 编写的 Android 应用能通过 adapter 来使用 SQLite。如果 SQLite 是用 Java 编写的，这对于 Android 肯定会更方便。但在 iPhone 上应用是 Objective-C 或者 Swift 编写的，这两种语言都没办法调用 Java 库。因此，如果 SQLite 选择用 Java 编写，那在 iPhone 上就没办法用了。

低依赖性

用 C 来编写库不会在运行时有太多的依赖。在最小的配置下，SQLite 只需要 C 标准库里的：memcmp()、memcpy()、memmove()、memset()、strcmp()、strlen()、strncmp()在更复杂的配置下，SQLite 可能还会用到 malloc()，free() 和一些操作系统接口来打开、读取、写入和关闭文件。但即使这样，依赖的数量也非常小。

稳定性

这个稳定性是指语言的稳定性。C 语言可能是老旧又无聊，但却正好很适合开发像 SQLite 这样更注重长期稳定的模块。

2.SQLite局限性

在 SQLite 中，SQL92 不支持的特性如下所示：

特性	描述
RIGHT OUTER JOIN	只实现了 LEFT OUTER JOIN。
FULL OUTER JOIN	只实现了 LEFT OUTER JOIN。
ALTER TABLE	支持 RENAME TABLE 和 ALTER TABLE 的 ADD COLUMN variants 命令，不支持 DROP COLUMN、ALTER COLUMN、ADD CONSTRAINT。
Trigger 支持	支持 FOR EACH ROW 触发器，但不支持 FOR EACH STATEMENT 触发器。
VIEWs	在 SQLite 中，视图是只读的。您不可以在视图上执行 DELETE、INSERT 或 UPDATE 语句。
GRANT 和 REVOKE	可以应用的唯一的访问权限是底层操作系统的正常文件访问权限。

◇并发访问的锁机制 SQLite在并发（包括多进程和多线程）读写方面的性能一直不太理想。数据库可能会被写操作独占，从而导致其它读写操作阻塞或出错。 ◇SQL标准支持不全 在它的官方网站上，具体列举了不支持哪些SQL92标准。我个人感觉比较不爽的是不支持外键约束。 ◇网络文件系统（以下简称NFS）

3.SQLite命令

与关系数据库进行交互的标准 SQLite 命令类似于 SQL。命令包括 CREATE、SELECT、INSERT、UPDATE、DELETE 和 DROP。这些命令基于它们的操作性质可分为以下几种：

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。

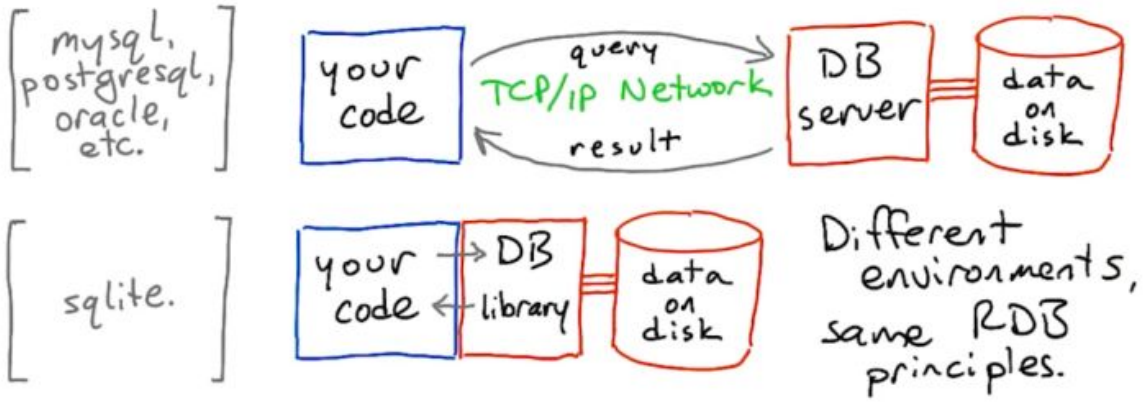
4.SQLite文件结构介绍

5.SQLite的应用

- 谷歌浏览器（C:\Users\17646\AppData\Local\Google\Chrome\User Data\Default下的Login Data文件）
- 网易邮箱大师（C:\Users\17646\AppData\Local\Netease\MailMaster\data中的文件）
- 微信（在手机上保存聊天记录就是用的SQLite）

6.SQLite与常用数据库的比较

Mysql, Sql Server之类的数据库通常是客户端通过网络传输从远程数据库服务器端读取数据；sqlite则是被集成在客户端程序（如网页浏览器），用于存取本地缓存数据，应用通过数据库接口直接从本地磁盘读取数据。服务架构和应用场景的不同赋予了sqlite特殊的使命。



sqlite工具中 常用命令集合

命令	说明
.databases	查看数据库信息
.tables	查看数据中的表
.schema TableName	查看表的完整信息
sqlite3 DatabaseName.db	用sqlite3工具连接数据库
.quit	退出
sqlite3 testDB.db .dump > testDB.sql	导出完整的数据库到指定文件
sqlite3 testDB.db < testDB.sql	从文件中恢复数据

二、DDL

1.首先创建数据库

```
sqlite3 DatabaseName.db
```

2.创建表

```
CREATE TABLE COMPANY(
   ID INT PRIMARY KEY      NOT NULL,
   NAME           TEXT     NOT NULL,
```

```
   AGE             INT      NOT NULL,
   ADDRESS         CHAR(50),
   SALARY          REAL
);

CREATE TABLE table_name(
   column1 INTEGER AUTOINCREMENT,
   column2 datatype,
   column3 datatype,
   .....
   columnN datatype,
);
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。

```

3.删除表

```
DROP TABLE COMPANY;
```

4.Alter修改表命令

```
# 修改表名
ALTER TABLE COMPANY RENAME TO OLD_COMPANY;
# 添加列
ALTER TABLE OLD_COMPANY ADD COLUMN SEX char(1);
```

三、DML

1.INSERT

```
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 );
```

```
15
16 INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY)
17 VALUES (6, 'Kim', 22, 'South-Hall', 45000.00 );
```

四、DQL

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。

下面是一个实例，使用 SELECT 语句获取并显示所有这些记录。在这里，前三个命令被用来设置正确格式化的输出。

```
1 .header on
2 .mode column
3 SELECT * FROM COMPANY;
```

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 # 设置列宽度
2 .width 10, 20, 10
3 # 查询
4 SELECT * FROM COMPANY;
```

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

```
1  # 从COMPANY中查询 AGE >= 25 AND SALARY >= 65000
2  SELECT * FROM COMPANY WHERE AGE >= 25 AND SALARY >= 65000;
3  # 列出了 AGE 不为 NULL 的所有记录
4  SELECT * FROM COMPANY WHERE AGE IS NOT NULL;
5  # 列出了 NAME 以 'Ki' 开始的所有记录
6  SELECT * FROM COMPANY WHERE NAME LIKE 'Ki%';
7  # 列出了 AGE 的值为 25 或 27 的所有记录
8  SELECT * FROM COMPANY WHERE AGE IN ( 25, 27 );
9  # 查询年龄比薪资65000高的人的所有记录
10 SELECT * FROM COMPANY
11      WHERE AGE > (SELECT AGE FROM COMPANY WHERE SALARY > 65000);
```

Glob子句：

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 SALARY GLOB '200*'	查找以 200 开头的任意值
WHERE SALARY GLOB '200'	查找值为200的记录
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，且长度至少为 3 个字符的任意值
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

Limit子句：

SQLite 的 **LIMIT** 子句用于限制由 SELECT 语句返回的数据数量。

```
# 查询前6条
SELECT * FROM COMPANY LIMIT 6;
# 分页：表示从第3条开始提取记录，再向后偏移2位（效果就是取3条记录）
SELECT * FROM COMPANY LIMIT 3 OFFSET 2;
```

Order by：

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

# 按照 SALARY 升序
SELECT * FROM COMPANY ORDER BY SALARY ASC;
```

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 NAME, SUM(SALARY) FROM COMPANY GROUP BY NAME;

# 在插入几条数据
INSERT INTO COMPANY VALUES (11, 'Paul', 24, 'Houston', 20000.00 );
INSERT INTO COMPANY VALUES (12, 'James', 44, 'Norway', 5000.00 );
INSERT INTO COMPANY VALUES (13, 'James', 45, 'Texas', 5000.00 );

# group by和order by组合使用
SELECT NAME, SUM(SALARY)
         FROM COMPANY GROUP BY NAME ORDER BY NAME DESC;
```

Having:

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。

```
# 显示名称计数小于 2 的所有记录（与传统SQL有差别），在sql标准中select中的条件是要求再group by中的值
SELECT * FROM COMPANY GROUP BY name HAVING count(name) < 2;
```

Distinct:

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。

```
# 获取name列表
SELECT name FROM COMPANY;
# 去重
SELECT DISTINCT name FROM COMPANY;
```

Join:

```
# 准备数据
CREATE TABLE DEPARTMENT(
   ID INT PRIMARY KEY      NOT NULL,
   DEPT           CHAR(50) NOT NULL,
   EMP_ID         INT      NOT NULL
);
INSERT INTO DEPARTMENT (ID, DEPT, EMP_ID)
VALUES (1, 'IT Billing', 1 );

INSERT INTO DEPARTMENT (ID, DEPT, EMP_ID)
VALUES (2, 'Engineering', 2 );

INSERT INTO DEPARTMENT (ID, DEPT, EMP_ID)
VALUES (3, 'Finance', 7 );
```

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。

```
SELECT EMP_ID, NAME, DEPT FROM COMPANY CROSS JOIN DEPARTMENT;
```

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。

```
SELECT EMP_ID, NAME, DEPT FROM COMPANY INNER JOIN DEPARTMENT
        ON COMPANY.ID = DEPARTMENT.EMP_ID;
```

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。

```
SELECT EMP_ID, NAME, DEPT FROM COMPANY LEFT OUTER JOIN DEPARTMENT
        ON COMPANY.ID = DEPARTMENT.EMP_ID;

SELECT EMP_ID, NAME, DEPT FROM DEPARTMENT  LEFT OUTER JOIN COMPANY
        ON COMPANY.ID = DEPARTMENT.EMP_ID;
```

事务：

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。

```
# 开始事务
BEGIN;
# 删除数据
DELETE FROM COMPANY WHERE AGE = 25;
# 回滚数据
ROLLBACK;
```

常用函数：

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。

日期时间：

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。

序号	函数	实例
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。

五、Java中的实现