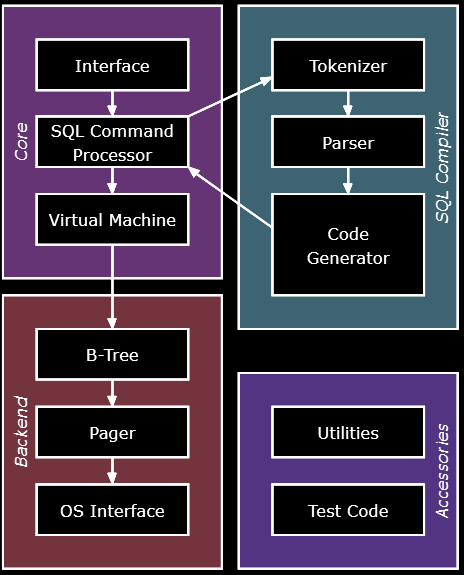
**資料庫系統概論 HW3**

10811020

陳光悦

**2-A**

1-a.



https://www.sqlite.org/arch.html

1-b

Interface：大部分的C code Interface可以在main.c、legacy.c和vdbeapi.c等檔案中找到。為了避免命名重複，SQLite library中的所有的external symbols皆以sqlite3為前綴。而那些給公共API則以sqlite3\_開頭。Extension API則還會再加上extension name，如sqlite3rbu\_或sqlite3session\_。

Tokenizer：當要解讀SQL語句的字串時，會先tokenizer分解成數個token。相關的code可以在tokenize.c找到。

Parser：之後parser會解讀token。SQLite是使用Lemon parser generator，不容易出錯，並且reentrant、thread-safe。

Code Generator：parser器將token組合成parse tree後，便由code generator解析成parse tree，並執行SQL。這些code generator的檔案包括：attach.c、auth.c、build.c、delete.c、expr.c、insert.c、pragma.c、select.c、trigger.c、update.c、vacuum.c、where.c、whereexpr.c。其中，expr.c負責處理SQL中表達式的部分，並生成相應可以被執行的指令。如：where\* .c負責生成SELECT、UPDATE和DELETE中的WHERE。而attach.c、delete.c、insert.c、select.c、trigger.c、update.c和vacuum.c則分别處理同名的SQL。同一個SQL語句可能會有無數個不同的方法來查詢出結果，特别是where\*.c和select.c中的邏輯，因此code generator還需要找出最佳的algorithm。

Bytecode Engine：code generator建立的bytecode program需要有一個virtual machine來執行。virtual machine的code可以在vdbe.c中找到，vdbe.h定義virtual machine與SQLite library之間的interface，vdbeInt.h定義virtual machine自己的structure和interface。vdbe\*.c為virtual machine的輔助code。vdbeaux.c包含virtual machine使用的interface modules。vdbeapi.c包含virtual machine使用的external interfaces，如sqlite3\_bind\_int()和sqlite3\_step()。vdbemem.c將變數（string、integer、floating point number、BLOB）儲存在Mem的internal object中。SQLite使用callbacks C code來實現SQL function。大部分的built-in SQL functions（abs()、count()、substr()等）可以在func.c中找到。Date和time轉換function可以在date.c中找到。其餘的function（coalesce()、typeof()）則直接由code generator生成bytecode。

B-Tree：SQLite的database用B-tree儲存在disk裡面，可以在btree.c中找到。每個table和database的index都使用不同的B-tree，所有的B-tree皆儲存在同一個disk file中。btree.h定義了B-tree subsystem與SQLite library的interface。

Page Cache：B-tree module從disk中固定大小的page請求訊息，預設的pages大小是4096 bytes。page cache負責讀取、寫入、緩存page，同時也提供database file的rollback、atomic commit abstraction、locking。B-tree driver從page cache請求特定的page，並在通知page cache當page修改、commit、rollback改變時。page cache可以在pager.c中找到，WAL mode則在wal.c中，in-memory caching則在pcache.c和pcache1.c。pager.h定義了page cache subsystem與SQLite library的interface。

OS Interface：SQLite為了在不同的operating system都可以運作，而使用VFS的abstract object，提供了在disk上打開、讀取、寫入和關閉file的方法。Unix的VFS在os\_unix.c中、Windows的VFS在os\_win.c中。

https://www.sqlite.org/arch.html

2-a.

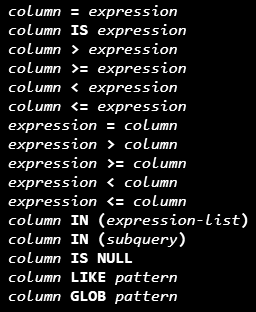
Core：當我們下達一個SQL指令時，它會被先tokenizer分解成token，並將這些token一一交給parser。相關的code可以在tokenize.c找到。其中，sqlite3GetToken()、sqlite3RunParser()兩個function分別負責tokenizer與parser SQL指令。  
 SQL Complier：parser器將token組合成parse tree後，便由code generator解析成parse tree，並執行SQL。這些code generator的檔案包括：attach.c、auth.c、build.c、delete.c、expr.c、insert.c、pragma.c、select.c、trigger.c、update.c、vacuum.c、where.c、whereexpr.c。其中，expr.c負責處理SQL中表達式的部分，並生成相應可以被執行的指令。如：where\* .c負責生成SELECT、UPDATE和DELETE中的WHERE。而attach.c、delete.c、insert.c、select.c、trigger.c、update.c和vacuum.c則分别處理同名的SQL。

Backend：SQLite是透過B-tree來進行實作，相關的code可在btree.c找到。其中，資料庫中的每個table和index，都是使用獨立的B-tree。B-tree的預設pages大小是4096位元組。相關的code可以在pager.c中找到。

https://www.sqlite.org/arch.html

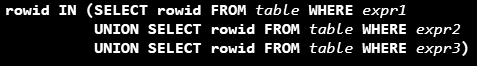
2-b-i

WHERE：WHERE clause可以拆成不同的terms，以AND或OR連接。為了提高效率，這些terms會被分析，當他們滿足下列的條件時，則可利用index來找到滿足這些條件的數據。



這些index在被建立時，可以指定某些特定的column來作為index的關鍵column（ 　）。

 BETWEEN：如果WHERE clause的term包含的形式，則可替換成的virtual terms，但不會生成額外的byte-code。如果此時兩個virtual terms皆符合index的條件，那原始的BETWEEN clause則可被省略。

 OR：如果WHERE clause的term是被用OR連接，則有兩種不同的方式處理。其一，可以被改寫為 　。其二，可以被改寫為 。

LIKE：如果WHERE clause中有LIKE或GLOB，當符合下述條件時，即可利用index來範圍搜尋的方式優化。

1. LIKE或GLOB的右側必須是string literal或parameter bound to a string literal，並且不是以wildcard character開頭。
2. LIKE或GLOB的左側是indexed column的名稱或右側不能是「-」或數字開頭。
3. LIKE和GLOB的built-in functions不能與sqlite3\_create\_function()裡的一樣。
4. GLOB必需使用built-in BINARY collating sequence。
5. LIKE如果是case\_sensitive\_like mode，則column必需使用BINARY collating sequence；反之，則使用built-in NOCASE collating sequence。
6. 如果使用ESCAPE，則ESCAPE的character必須是ASCII或UTF-8。

https://www.sqlite.org/optoverview.html

2-b-ii

SQLite使用nested loop連接操作，每個表都被一個loop連接。每個loop上可能使用一個或多個index來加快查詢，或者是loop可以是full table scan，來掃過table中的每一的row。因此query planning可以拆成兩個subtask：

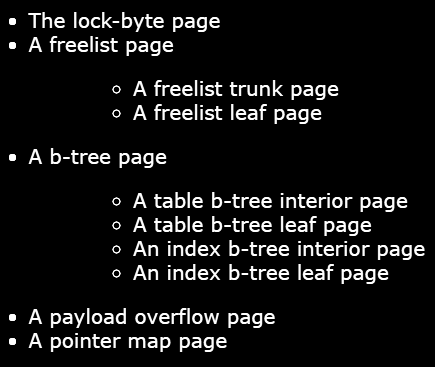
1. 選擇不同順序的nested loop
2. 為每個loop選擇合適的index

https://www.sqlite.org/queryplanner-ng.html

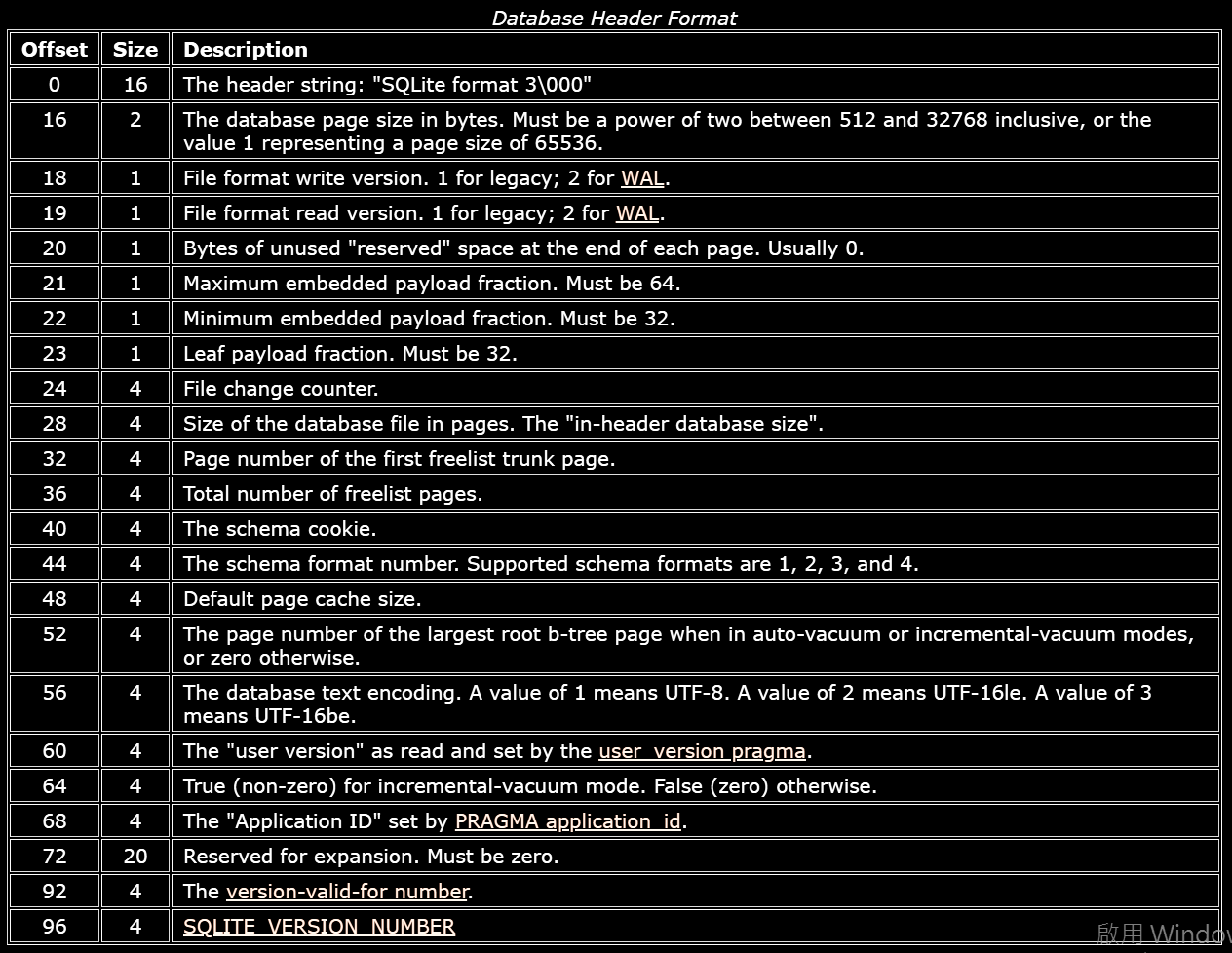
3-a

完整的SQLite資料庫通常是存放在disk中的單一文件夾中，文件夾的名字為main database file。而在transaction期間，SQLite並不會馬上將資料寫入main database file中，而是會先寫入rollback journal的文件夾中（write-ahead log (WAL) mode）。而當資料要修改時，未修改的原始内容先被寫入rollback journal中，如此可以降低資料庫在寫入時可能存在的損壞風險。如果在transaction完成前應用程序或主機電腦發生崩潰，此時就可以從rollback journal中恢復main database file，此時的rollback journal可被稱為hot journal或hot WAL file。

main database file由一個或多個page組成，大小為 512～65536 byte，所有位於同一資料庫內的page大小皆相同。每一個page都有單一的用途，如下：



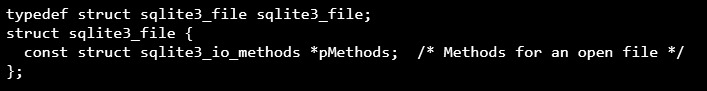
每個page的前100 bytes為database file header，包含了所有關於此page存取資料的資訊，以big-endian的方式儲存。再之後才是page中真正存取的資料內容。database file header的內容如下：



https://www.sqlite.org/fileformat2.html

3-b

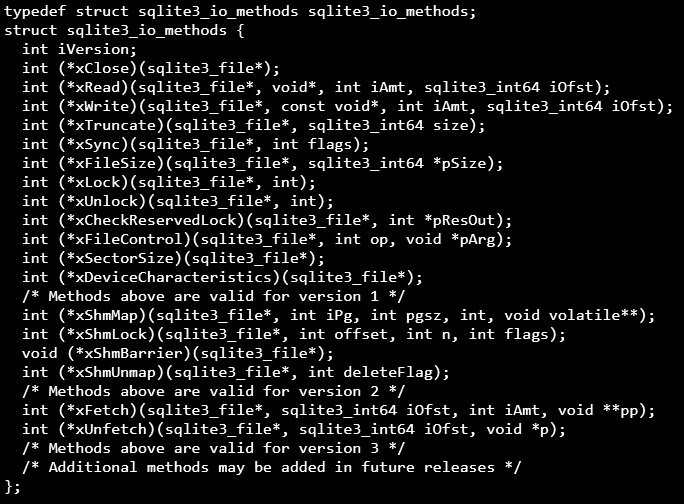
VFS為SQLite提供file的打開、讀取、寫入、關閉等方法。



https://www.sqlite.org/c3ref/file.html



https://www.sqlite.org/c3ref/vfs.html



https://www.sqlite.org/c3ref/io\_methods.html

open：SQLite在打開file時，會使用xOpen。根據database的設定， xOpen初始化sqlite3\_file結構並設置pMethods。

read：讀取數據時，會使用xRead。根據iOfst決定開始讀取的位置，再根據iAmt決定讀取多少data，並將其存儲在buffer中。

write：寫入數據時，會使用xWrite。根據iOfst決定開始寫入的位置，再根據iAmt決定寫入多少data。

close：關閉file時，會使用xClose。釋放file所佔用的任何資源。

https://www.sqlite.org/vfs.html

4-a

SQLite 3.0.0以locking and journaling mechanism來實現concurrency control。pager會使用一些特定的modules（os\_unix.c或os\_win.c等等），與operating system進行互動，為operating system services提供一致的abstraction。

一個database file可以處於下述五種鎖定狀態的其中一種：

1. UNLOCKED：database上沒有任何lock，既不能read，也不能write，視為默認狀態。
2. SHARED：database可以read，但不能write。可以多個processes同時持有SHARED lock。
3. RESERVED：process計劃之後的某個時刻write database file，但目前只是在read，而非write。同一時間只能active一個RESERVED lock。但可以多個SHARED lock與一個RESERVED lock同時存在。
4. PENDING：表示process希望盡快write database file，只是在等所有剩下的SHARED lock被清除，以便獲得EXCLUSIVE lock。當有PENDING lock時，則不允許database新增其他的SHARED lock，但允許現有的SHARED lock繼續存在。
5. EXCLUSIVE：可以write database file。文件上只允許有一個EXCLUSIVE lock，而且此時不允許有任何其他類型的lock存在。

operating system的interface layer可以識別和追蹤這五種lock，而page module只會追蹤其中四種。因為在這四種lock中，PENDING lock只是EXCLUSIVE lock的臨時過渡階段而已，因此page module並不會不追蹤PENDING lock。

當process想要修改database file（不是WAL mode）時，他會先在rollback journal中記錄原始未修改過的內容。rollback journal是一個普通的disk file，與database file位於相同的目錄或文件夾中，名稱的前綴與database file相同，後綴為-journal。rollback journal還記錄database的初始大小，因此如果數據庫文件增長，則可以在rollback時截斷回到其原始大小。

當rollback journal需要rollback恢復database的內容時，就被稱為hot journal。當process正在更新database到一半時，如果發生operating system崩潰或斷電，就會導致更新無法完成，此時就需要建立hot journal以救回database裡面的內容。因此如果直到process完成時都沒有發生異常狀況，就永遠不會出現hot journal。

https://www.sqlite.org/lockingv3.html

4-b-i

當要修改database時，相關的database file會進行鎖定，防止誤改到其他的database file。

如果是讀取操作時，會給以SHARED lock，且可以同時多個processes進行；寫入操作時，會先給RESERVED lock，告訴processes將有database file要進行寫入的動作，而當要真正開始寫入時，會先給PENDING lock，再給EXCLUSIVE lock。

4-b-ii

當我們對同一個database進行兩個transaction，都是試圖去更新database。理論上應該要等第一個transaction完成，才能進行下一個transaction。但如果第二個transaction並沒有等第一個transaction完成，就直接開始進行第二個transation，此時就會發生nondeterministic。

因此在對database進行修改時，應該要有locking mechanism，避免一個transaction尚未完成，就開始另一個transation。此外還有WAL module，也可以確保在系統在崩潰或斷電的情況下，所做到一半的更改可以被恢復，從而減少資料損壞的風險。

**2-B** Temporary Files Used By SQLite

SQLite在處理database的過程中，會產生下述九種臨時文件。

1. Rollback journals

rollback journal實現了atomic commit和rollback的功能，與database file位於相同的目錄中，並與database file的名稱相同，只是在最後加上-journal。通常在transaction開始時創建，並在transaction commit或rollback後刪除。如果沒有rollback journal，則當transaction在進行的過程中發生中斷，就無法rollback，回朔到transaction之前的database file。

1. Super-journals

super-journal是當一個transaction中對使用ATTACH statement添加到單個database連接的多個database進行修改時，作為atomic commit過程的一部分而使用的。super-journal與database file位於相同的目錄中，具有隨機生成的後綴。super-journal包含在transaction期間更改的所有附加auxiliary database的名稱。在transaction commit後，才會刪除super-journal。在transaction中，super-journal使得database中的所有更改，要麼全部rollback，要麼全部commit。super-journal僅為涉及多個database file的commit操作所創建，其中至少要有兩個database滿足以下所有要求：

1. transaction有對該database做修改
2. PRAGMA synchronous設置不是OFF
3. PRAGMA journal\_mode不是OFF、MEMORY或WAL
4. Write-ahead Log (WAL) files

當在WAL mode運行時，WAL files的功能與rollback journal相似，皆是實現了atomic commit和rollback。且與rollback journal一樣，位於database file的相同目錄中，並與database file的名稱相同，只是在最後加上-wal。當database的第一個connection被打開時，WAL files就會被創建，等到database的最後一個connection被關閉後，才會被刪除。但如果最後一個連接並不是正常的被關閉，則WAL file仍會保留在filesystem中，直到下一次打開database時，才會自動清除。

1. Shared-memory files

在WAL mode運行時，與同一個database file有關聯的所有SQLite database connection需要共享一些內存，用來儲存WAL file的index。因此只有在WAL mode時，才會有shared-memory file。shared-memory file與database file位於相同的目錄中，名稱與database file相同，是在最後加上-shm。shared-memory file的惟一目的，是WAL mode訪問同一個database的多個process時，提供一個shared-memory。因此如果VFS能夠提供一個shared-memory的訪問方法，則可以替代shared-memory file。例如，當PRAGMA locking\_mode設置為EXCLUSIVE時，此時只有一個process能夠訪問database file，則shared-memory即可分配到heap中，而不是創建一個shared-memory file。在創建WAL file時，shared-memory file也會同時被創建，並且在刪除WAL file時會被刪除。在WAL file恢復期間，則根據正在恢復的WAL file的內容，shared-memory file會被從頭開始重新創建。

1. Statement journals

statement journal是用在一個冗長的transaction中，rollback部分結果。例如，假設有一個UPDATE statement試圖修改database中的100 rows。但在修改前50 rows後，遇到constraint violation，導致此UPDATE statement無法繼續進行。此時statement journal只需要rollback前50 rows的更改，使得數據庫恢復到UPDATE statement開始前的狀態。statement journal只會在UPDATE或INSERT statement時被建立，因為只有兩個statement才有可能發生修改多rows，並且會觸發constraint violation或RAISE exception，需要恢復部分的結果。如果UPDATE或INSERT不包含在BEGIN...COMMIT 中，並且在同一database connection上沒有其他的active statements，則也不會statement journal，因為此時使用rollback journal即可達到目的。statement journal的名稱是隨機的，並且不一定與main database位於同一目錄。會在transaction結束時自動刪除。statement journal的大小與UPDATE或INSERT statement所更改的大小成正相關。

1. TEMP databases

CREATE TEMP TABLE syntax所創建的table，與其相關的index、trigger、view，都存儲在個別的temporary database file中。temporary database file在首次遇到CREATE TEMP TABLE syntax時創建。這個單獨的temporary database file也有相應的rollback journal。temporary database file與其rollback journal，只有在使用CREATE TEMP TABLE syntax時才會被創建。當使用sqlite3\_close()關閉database connection時，temporary database file就會自動刪除。temporary database file與使用ATTACH statement添加的auxiliary database files非常相似。但temporary database會在database connection關閉時自動刪除。temporary database的設定永遠是synchronous=OFF和journal\_mode=PERSIST PRAGMA，且temporary database不能用於DETACH，其他的process也不能ATTACH temporary database。

1. Materializations of views and subqueries

當query中有subquery時，就會需要將結果存儲在一個temporary table中。SQLite中的query optimizer會嘗試避免temporary table的產生，但很多時候還是無法避免。此時，這些temporary table將會分別存儲在他們個別的temporary file中，並在query結束時自動刪除。

1. Transient indices

SQLite可以利用transient index來實現一些SQL的功能，例如：

1. ORDER BY或GROUP BY clause
2. aggregate query中的DISTINCT
3. 使用UNION、EXCEPT、INTERSECT合併的SELECT statement

這些transient index都會存儲在各自的temporary file中，並且在這些的statement使用結束時自動刪除。

1. Transient databases used by VACUUM

VACUUM是一個在SQLite database中用於優化和壓縮存儲空間的指令。當database中的table被修改或刪除時，SQLite可能會保留已使用的空間，而不會立即回收。這就會導致database file佔用不必要的磁盤空間。通過VACUUM，SQLite將重新組織database file，並回收被刪除或修改過的空間。這樣可以壓縮database file的大小，釋放未使用的存儲空間，並提高database的性能。而VACUUM的工作原理是，創建一個temporary file，然後將database的內容完整地重建到temporary file中，接著將temporary file的內容複製回原始的database file中，並刪除temporary file。這個過程可以清理和壓縮database file，並將其大小恢復到適合的大小。而在執行VACUUM時，會將database變為read only，直到VACUUM完成。因此在執行VACUUM前，需要確保沒有其他transaction在使用該database。

https://www.sqlite.org/tempfiles.html