HÖGSKOLAN
DALARNA

Examensarbete

Kandidatexamen

Utveckling av metoder för utvinning av raderad data ur SQLite databaser

Development of methods for the extraction of deleted data from SQLite databases

Författare: Sebastian Zankl
Handledare: Hans Jones
Examinator: Pascal Rebreyend
Ämne/huvudområde: Datateknik
Poäng: 15 hp
Betygsdatum:

Examensarbete nr: E4151D

Högskolan Dalarna
791 88 Falun
Sweden
Tel 023-77 80 00

EXAMENSARBETE, C-nivå Datateknik

Program	Reg nr	Omfattning
Digitalbrott och eSäkerhet, 180 p	E4151D	15 hp
Namn Sebastian Zankl	Datum 2011-05-31	
Handledare Hans Jones	Examinator Pascal Rebreyend	
Företag/Institution Högskolan Dalarna	Kontaktperson vid företaget/institutionen	
Titel Utvinning av raderad data ur SQLite databaser		
Nyckelord Databas, SQLite, SQL, raderad data, Python, data utvinning		

Sammanfattning

SQLite är en databasmotor som används av många mjukvaror, som till exempel mjukvaror från Android, IOS (Apple), webbläsaren Mozilla Firefox, Internettelefoniprogrammet Skype med mera. Databaser innehåller ofta information som är intressant vid till exempel brottsutredningar vilket gör det mycket viktigt att kunna extrahera och granska all potentiell data som finns sparad i dem.

För tillfället kan de flesta program enbart visa den sparade (synliga) datan i SQLite databasfiler. Ofta är det dock den raderade datan som kan vara avgörande vid brottsutredningar. Därför handlar detta arbete om utvinning av raderad data ur dessa databaser.

Syftet med detta arbete var att utveckla metoder för att extrahera den raderade datan ur SQLite databaser och att implementera dessa i ett prototypprogram. Programmet utvecklades i programmeringsspråket Python. Databaserna som användes för att testa metoderna och programmet togs ifrån Android, iPhone och Mozilla Firefox.

Resultatet av arbetet blev tre metoder för att extrahera raderad data ur SQLite databaser, varav två implementerades i prototypprogrammet. De två metoderna som implementerades i programmet kan båda hitta kompletta records i en databas, vilket innebär kompletta rader i en databastabell. Metoden som inte implementerades i programmet letar efter delar av records som inte gick att hitta med någon av de två första metoderna.

Högskolan Dalarna
Röda Vägen 3, Borlänge
781 88 BORLÄNGE

Telefon: 023-77 80 00
Telefax: 023-77 80 50
URL: http://www.du.se/

HÖGSKOLAN DALARNA

DEGREE PROJECT
Computer Engineering

Programme	Reg number	Extent
Digital Crime and eSecurity	E4151D	15 ECTS
Name of student	Year-Month-Day	
Sebastian Zankl	2011-05-31	
Supervisor	Examiner	
Hans Jones	Pascal Rebreyend	
Company/Department	Supervisor at the Company/Department	
Högskolan Dalarna		
Title		
Extraction of deleted data from SQLite databases		
Keywords		
Database, SQLite, SQL, deleted data, Python, data extraction		

Summary

SQLite is a database engine which is used by a lot of software, for example software from Android, IOS (Apple), the web browser Mozilla Firefox, the Internet telephony program Skype etcetera. Databases often contain information which is interesting in for example crime investigations which makes the ability to extract and review all potential data that is saved in them very important.

Currently most programs can only show the saved (visible) data in SQLite databases but often it is the deleted data that can be crucial for crime investigations. That is why this thesis is about the extraction of deleted data from these databases.

The purpose of this thesis is to develop methods to extract the deleted data from SQLite databases and to implement these methods in a prototype program. The program was developed in the programming language Python. The databases which were used to test the methods and the program were taken from Android, iPhone and Mozilla Firefox.

The result of the thesis was three methods to extract deleted data from SQLite databases, of which two were implemented in a prototype program. The two methods that were implemented in the program can both find complete records in the database, which means complete rows in a database table. The method that wasn't implemented in the program searches for parts of records which cannot be found by any of the first two methods.

Högskolan Dalarna
Röda Vägen 3, Borlänge
781 88 BORLÄNGE

Telefon: 023-77 80 00
Telefax: 023-77 80 50
URL: http://www.du.se/

Innehållsförteckning

1 Inledning

1.1 Bakgrund

SQLite är en väldigt populär och utspridd databastyp. Det finns väldigt många program som klarar av att visa och administrera innehållet i en SQLite databas, som till exempel SQLite Maestro, SQLite2009 Pro Enterprise Manager, Adminer med mera (SQLite, datum okänt). Däremot är det väldigt få program som klarar av att hitta raderad data i SQLite databaser.

Några program som påstår sig kunna hitta raderad data i SQLite databaser är: FF3HR (Firefox 3 History Recoverer), SQLUn (SQLite Undeleter) och EPILOG. Dock verkar alla dessa program ha sina nackdelar. FF3HR tar enbart kompletta diskavbildningar som input och den letar enbart efter data i databaser som tillhör Firefox 3 (PenTestIT, 2009). SQLUn är fortfarande i stängd beta stadiet vilket gör det svårt att få tag på programvaran (Chirashi Security, 2010). En gratis textversion av EPILOG kan laddas ner ifrån utvecklarens hemsida, dock var detta inte möjligt när detta examensarbete påbörjades (CCL-Forensics, datum okänt).

På grund av ovanstående brister valdes att utveckla en mjukvara som kan extrahera raderad information ur SQLite databaser. Då många mjukvaror använder sig av SQLite kan en sådan lösning ge stor nytta. Exempel på mjukvaror som använder SQLite är mjukvaror från IOS (Apple) och Android, många webbläsare som Firefox och Safari, andra mjukvaror som Skype, Adobe och många andra. (SQLite, datum okänt)

1.2 Syfte

Under föregående rubrik nämndes att många mjukvaror använder SQLite databaser för att lagra data. Vid brottsundersökningar är det därför viktigt att kunna ta ut och granska denna. Brottslingar kan radera kontakter, SMS/MMS, webbhistorik och så vidare ur sina mobiltelefoner i syfte att försvåra utredningar. Därför är det viktigt att kunna extrahera raderad data ur SQLite databaser.

Syftet med examensarbetet var att utveckla metoder för att utvinna raderad data ur SQLite databaser. Dessa metoder skulle implementeras i ett prototypprogram som även skulle kunna ta ut den sparade (synliga) datan ur SQLite databaser.

1.3 Avgränsningar

Extrahering av data ur SQLite databaser är komplicerat. Därför begränsades det utvecklade programmet så att det enbart kan hantera de vanligaste databasformaten. Detta är också anledningen till att beskrivningen av SQLite's filformat i Bilaga A enbart inkluderar information som har använts för att utveckla programmet.

Funktioner programmet inte stödjer är:

- Stöd för analys av databaser med journalfiler
- Stöd för analys av enbart journalfiler utan tillhörande databas
- Stöd för analys av databaser med write-ahead log filer
- Stöd för analys av enbart write-ahead log filer utan tillhörande databas
- Stöd för högre schema layer filformat än ett
- Stöd för text som är avkodad i ett annat format än UTF-8

Begreppen journalfil, write-ahead log fil och schema layer filformat förklaras i ordlistan under Bilaga B.

Programmet kan dessutom inte tolka datan vilket innebär att till exempel en tidsstämpel som har sparats i Unix tid, alltså bara som ett heltal, inte kommer att omvandlas till ett riktigt datum. Datumet kommer istället att visas som ett heltal.

Anledningen till att stöd för dessa funktioner inte lades till var dels att de inte förekommer så ofta och dels att det inte fanns tillräckligt med tid att implementera dem.

1.4 Metod

För att uppnå examensarbetets mål delades det upp i två steg. Läsa och förstå SQLite's dokumentation om databasfilers uppbyggnad och format. Därefter utveckla ett program som kan extrahera både sparad (synlig) och raderad data ur databasen samt presentera den på ett bra sätt för användaren.

Arbetet inleddes med att läsa SQLite's dokumentation om databasernas filformat som nästan förklarade allt om hur databaserna är uppbyggda och fungerar. Information om SQLite och om databasernas filformat samlades via SQLite's hemsida: http://www.sqlite.org/. Deras hemsida har utmärkt allmän information om SQLite och även dokumentationen om både filformatet och alla funktioner är utmärkt. Då informationen kom direkt ifrån utvecklingsteamet som jobbar med SQLite är det nog bästa stället att hämta information om SQLite.

Därefter påbörjades utvecklingen av programmet vilket kodades i programmeringsspråket Python. Anledningen att Python valdes var främst att den är ett multi-plattformsspråk, det vill säga Python kan användas på flera olika operativsystem utan att koden måste ändras eller kompileras om. Python kan till exempel köras i Windows, Unix/Linux och Mac OS X (Python Software Foundation, datum okänt). Python är också för det mesta ett rätt enkelt programmeringsspråk vilket gjorde det lite lättare att skapa ett större, lite mer komplicerat, program.

Det finns även några nackdelar med att använda Python. Den största är att program skrivna i Python ofta exekvera långsammare än program skrivna i andra programmeringsspråk. Denna nackdel är oväsentligt när programmet ska gå igenom små databaser på några megabyte dock kommer det att ta lång tid att bearbeta stora databaser på flera hundra megabyte.

Rapporten skrevs till stor del samtidigt som programmet utvecklades. När programmet var färdigutvecklat utfördes ett antal tester för att försöka hitta så många programmeringsfel som möjligt och för att rätta till dem. Testerna användes också för att kontrollera vilka SQLite databaser programmet kan hantera, till exempel databaser ifrån Android, iPhone, Mozilla Firefox med mera. Dessutom skulle testerna visa om programmet eventuellt missar data. Slutligen lades resultatet av arbetet och testerna till i rapporten.

2 SQLite

2.1 Om SQLite

SQLite är en inbyggd SQL databas motor. Det som utmärker SQLite är att den arbetar utan extra serverprocess, inte behöver konfigureras, är transaktionell och kräver minimalt stöd ifrån externa bibliotek eller operativsystem. Transaktionell innebär att alla ändringar i en enda transaktion antingen sker fullt ut eller inte alls.

Jämfört med andra databaser som har en egen serverprocess så läser och skriver SQLite direkt till en vanlig fil. Dessa filer innehåller en komplett SQL databas med tabeller (tables), index, triggers och views. SQLite är också otroligt liten. Om alla funktioner aktiveras kan storleken på biblioteket fortfarande vara under 300KiB (kilobyte) och med minimala funktioner kan storleken minskas ner till mindre än 180KiB. Om det krävs kan SQLite även tvingas till att köras i minimalt stackutrymme och väldigt lite heap utrymme. Alla dessa attribut gör SQLite mycket lämpligt för inbyggda system eller mjukvara som sparar information i en databas där databasen inte kan ha en extra serverprocess, till exempel program som främst används av hemanvändare. (SQLite, datum okänt)

2.2 Överblick över filformatet

Informationen under denna rubrik ger en kort sammanfattning av SQLite's filformat. En mera utförlig beskrivning av filformatet finns tillgänglig under Bilaga A.

En SQLite databas består till stor del av B-Tree strukturer. Dessa är balanserade sökträd där varje nod har noll, en eller flera barnnoder. I B-Tree strukturer sparas all data i en databas. En databas är uppdelad i sidor (pages) som alla är lika stora och många av sidorna tillhör B-Tree strukturerna. Sidornas storlek är alltid en tvåpotens mellan 512 (2^9) och 65536 (2^{16}). Bild 1 (SQLite, datum okänt) visar ett exempel på en B-Tree struktur. Varje nod i strukturen är en sida (siffrorna i noderna är de sparade värdena inte sidnumren).

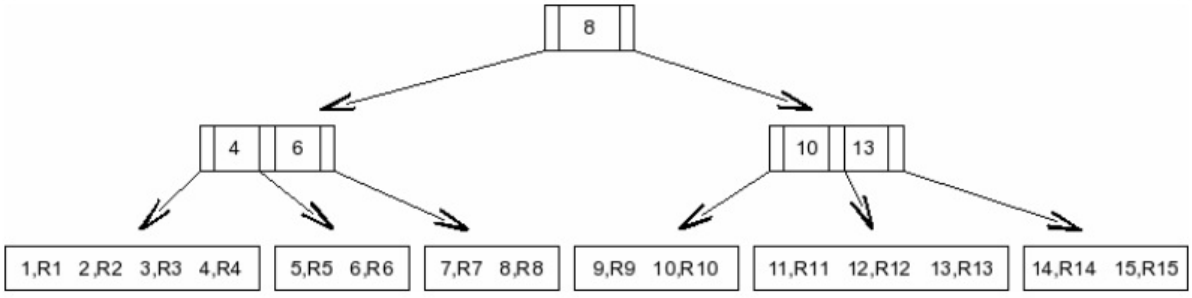

Bild 1 – Exempel på en B-Tree struktur

Den viktigaste B-Tree strukturen i en databas kallas schematabell. Den innehåller all information om alla schemaobjekt vilka kan vara tabeller, index, triggers och views. Tabeller och index kan innehålla data och består därför av B-Tree strukturer. Bild 2 är skapad i Microsoft PowerPoint och visar ett exempel på information som kan vara sparad på en rad i schematabellen. Exemplet visar bland annat typen av schemaobjektet ('table'), dess namn ('example') och dess rotsida ('5'). Rotsidan är den noden i B-Tree strukturen varifrån alla andra noder kan nås. I B-Tree exemplet i Bild 1 är rotsidan den översta som innehåller värdet åtta. Rotsidan för schematabellen är alltid databasens första sida.

```
'table', 'example', 'example', 5, 'CREATE TABLE example (_id integer primary key autoincrement,
name text, address text, telephone integer)'
```

Bild 2 – Exempel på sparad information i schematabellen

En B-Tree sida innehåller celler och varje cell kan antingen innehålla ett record eller sidnumret till en barnsida (child page). En barnsida är en sida som refereras till från en annan sida. En barnsida kan dock också vara en föräldersida (parent page) om även den refererar till en barnsida. Bild 3 visar en förenklad B-Tree sida där cellerna innehåller records. Bild 4 visar en förenklad B-Tree sida där cellerna innehåller sidnummer till barnsidor. Bild 3 och 4 är skapade i Microsoft PowerPoint.

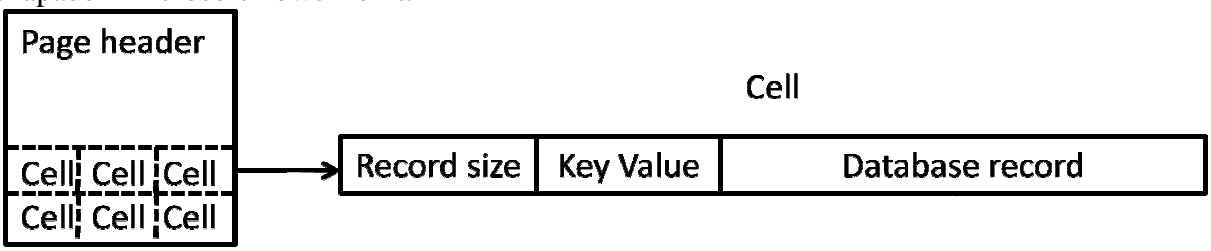

Bild 3 – Förenklad uppbyggnad på en B-Tree sida där cellerna innehåller records

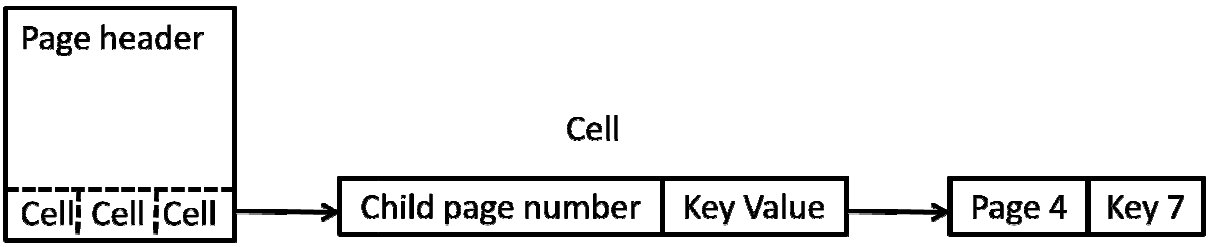

Bild 4 – Förenklad uppbyggnad på en B-Tree sida där cellerna innehåller sidnummer till barnsidor

Ett record motsvarar en rad i en tabell eller i ett index. För varje kolumn i motsvarande tabell eller index finns det ett värde i record:ens header. Dessa värden anger typ och storlek av ett inlägg (entries) i record:ens datadel. Bild 2 visade ett exempel på information som kan finnas sparad i schematabellen. Denna information fanns till exempel sparad i ett record. Bild 5 är skapad i Microsoft PowerPoint och visar ett records uppbyggnad.

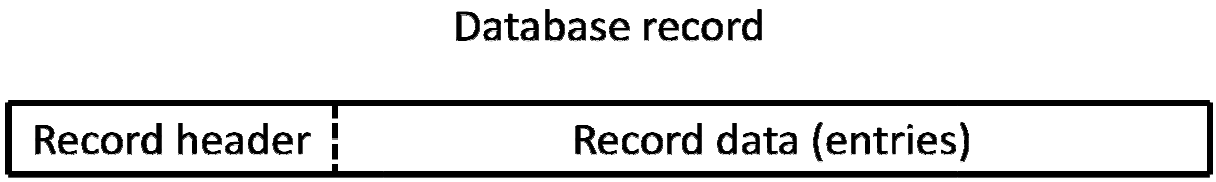

Bild 5 – Uppbyggnaden av records

Ibland är ett record för stor för att komplett kunna sparas på en B-Tree sida. I ett sådant fall sparas record:en i en överskottskedja (overflow chain). Det innebär att enbart ett record prefix sparas på sidan och att resterande record datan sparas på överskottssidor (overflow pages) som är ihopkopplade till en kedja. Bild 6 är skapad i Microsoft PowerPoint och visar ett exempel på ett sådant fall.

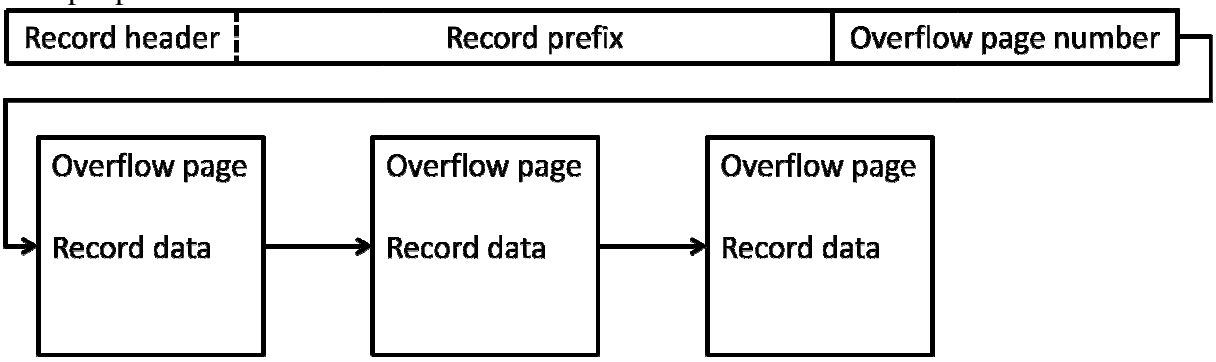

Bild 6 – Uppbyggnad av ett record som ingår i en överskottskedja

Bild 7 visar hur cellen med tillhörande record för exemplet i Bild 3 ser ut i en hexeditor. Den gröna markeringen visar record storlek (record size) och nyckelvärde (key value) som sparas i cellen (se exemplet i Bild 3). Den röda markeringen visar värdena i record header:n och den blåa markeringen visar datan i record:en.

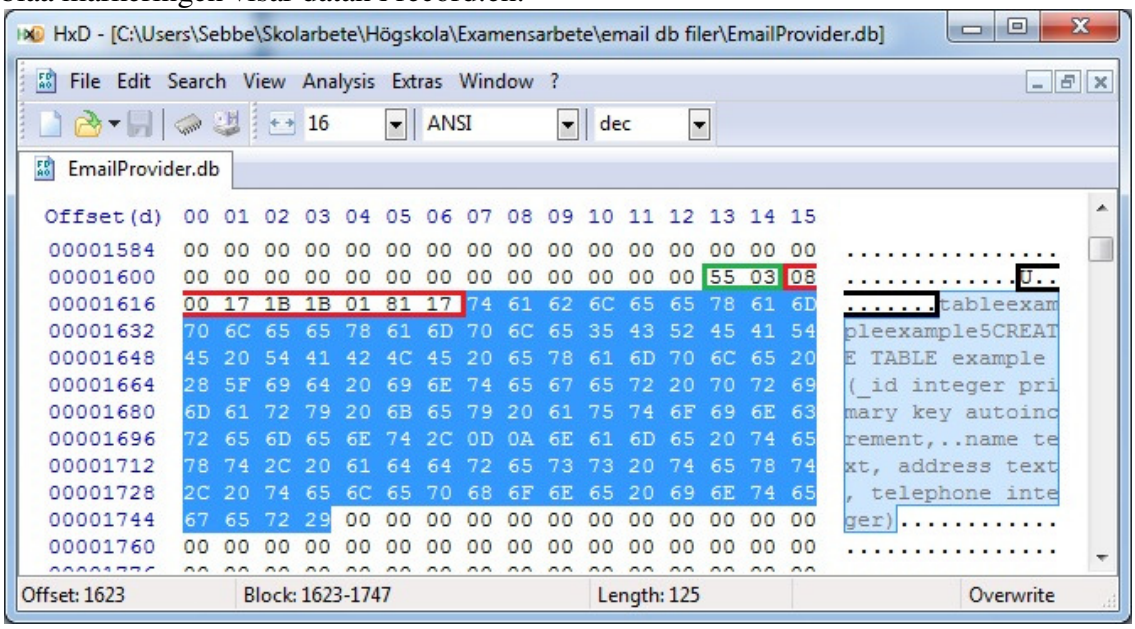

Bild 7 – Exempel på en B-Tree cell med tillhörande record

Andra sidor som finns i SQLite databaser är pointer map sidor, fria sidor (free pages) och locking sidan. Pointer map sidor bildar en uppslagstabell som sparar sidtypen och föräldersidan av alla sidor i databasen, med några undantag. Fria sidor är sidor som inte används för att spara någon giltig data och locking sidan är en speciell sida som börjar vid en gigabyte gränsen och som inte används till någonting. (SQLite, datum okänt)

Bild 8 är skapad i Microsoft PowerPoint och visar ett exempel på uppbyggnaden av en databas.

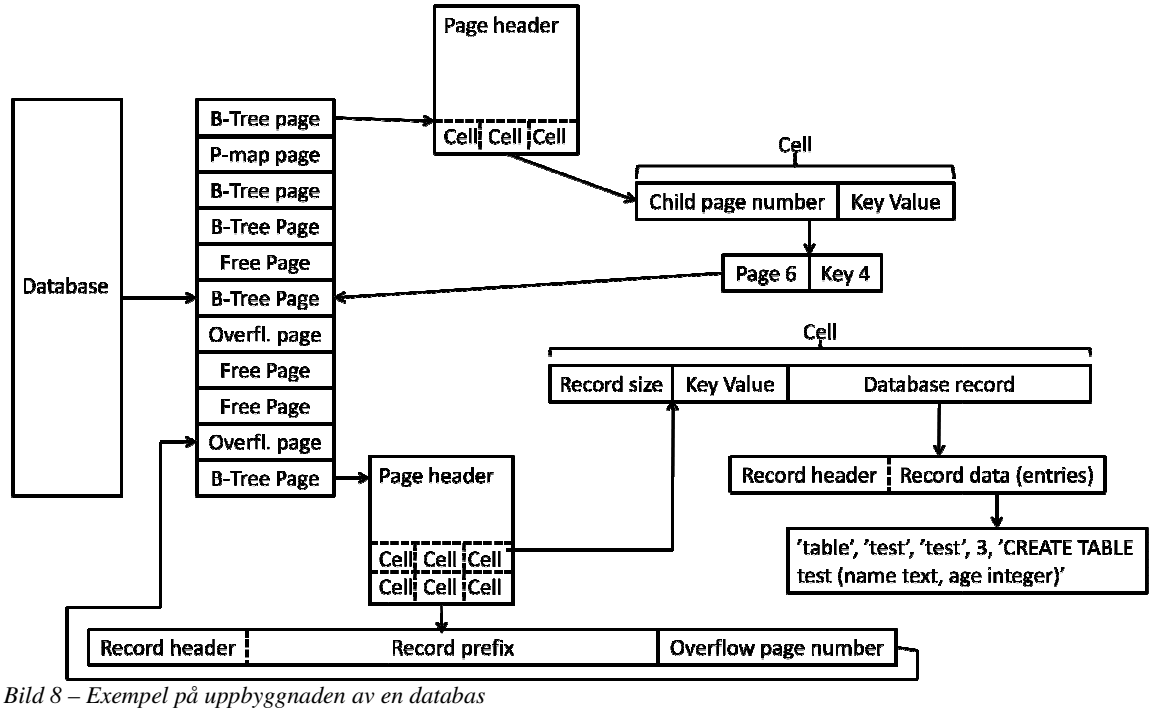

Bild 8 – Exempel på uppbyggnaden av en databas

3 Programmet

3.1 Överblick

Det utvecklade prototypprogrammet fick namnet SQLite Data Finder. Programmet exekverar i fem steg vilka är:

- Läsa in information ifrån databas header:n
- Extrahera schematabellen
- Extrahera all data ur tabeller och index
- Sökning efter raderad data
- Utskrift av all data

Programmets huvuddel utför inläsningen av header datan. De återstående fyra stegen utförs av andra funktioner. De viktigaste funktionerna i programmet är:

- ReadPage
- ReadRecord
- DeletedRecordSearch
- ReadDeletedEntry
- PrintTable

Dessa förklaras kortfattat under de efterföljande rubrikerna. I Bilaga C finns den kompletta koden till SQLite Data Finder.

3.1.1 ReadPage

Denna funktion används för att gå igenom informationen på en B-Tree sida. Funktionen behandlar alla celler som tillhör B-Tree sidan. Ifall cellen innehåller numret till en barnsida (child page) anropar funktionen en ny instans av sig själv för att gå igenom datan i barnsidan. Innehåller cellen ett record anropas ReadRecord funktionen. När alla sidor i en B-Tree struktur har gåtts igenom returnerar funktionen en lista som innehåller alla data som fanns sparad i sidornas records.

3.1.2 ReadRecord

Denna funktion används för att läsa ut datan i ett record. Funktionen granskar värdena i record header:n för att hitta typ och storlek på inläggen (entries) som finns sparade i record:ens datadel. Sedan läses alla inlägg in och en lista med header värdena och inläggen returneras. Anledningen till att även header värdena returneras är att det annars inte går att tilldela inläggen till rätt kolumn.

3.1.3 DeletedRecordSearch

Denna funktion letar igenom alla block av oanvänt utrymme. Funktionen kan använda sig utav både sökmetod ett och två som finns beskrivna under rubrikerna ”4.1 Återskapa records – record storlek matchning” och ”4.2 Återskapa records – record header typ matching”. Har en raderad record hittats anropas ReadDeletedEntry funktionen. När all oanvänt utrymme har sökts igenom returnerar funktionen en lista med record header värdena och record datan.

3.1.4 ReadDeletedEntry

Denna funktion läser ut datan ur raderade records och är ganska likt ReadRecord funktionen. Funktionen behöver dock inte granska record header:n då den får dess värden som ett argument vid anropet. När inläggen i record:ens datadel läses in kontrollerar funktionen att inte fler bytes läses in än vad det finns kvar i det aktuella block av oanvänt utrymme. Till slut returneras en lista med record datan.

3.1.5 PrintTable

Denna funktion är programmets utskriftsfunktion. Den skriver ut all tabell och index data dock inte den raderade datan. Funktionen försöker att ta reda på namnen och datatyperna av kolumnerna i en tabell eller ett index. Ifall den lyckas skapas en passande tabell där datan fylls i. Bild 9 visar ett exempel av en sådan tabell. Lyckas den inte skrivs tabellens eller indexets records ut rad för rad. Ett exempel på det visas i Bild 10.

sqlite_sequence

RowID	name	seq
1	HostAuth	2
2	Account	1
3	Mailbox	12
4	Message	25

Bild 9 – Exempel på en tabell skapad av utskriftsfunktionen

```
calls
------

1       ['15555215556', 1298209158820L, 1003, 1, 1, 'Friend', 1]
2       ['5558', 1298210424342L, 80, 2, 1, 0]
3       ['5558', 1298210524742L, 0, 2, 1, 0]
4       ['5558', 1298211204420L, 213, 2, 1, 0]
5       ['076123987463', 1298212589570L, 21, 1, 1, 'Jimmy', 2]
6       ['073156835154', 1298212908333L, 85, 2, 1, 'Timmy', 1]
7       ['073156835154', 1298213005540L, 551, 2, 1, 'Timmy', 1]
8       ['073156835154', 1298213566121L, 0, 2, 1, 'Timmy', 1]
9       ['073156835154', 1298213575142L, 225, 1, 1, 'Timmy', 1]
10      ['076123987463', 1298213830141L, 161, 2, 1, 'Jimmy', 2]
11      ['15555215556', 1298214006886L, 2315, 2, 1, 'Friend', 1]
```

Bild 10 – Exempel på hur datan skrivs ut ifall utskriftsfunktionen inte kunde skapa en tabell

3.2 Programmets brister

En del av programmets brister beskrevs redan under rubriken ”1.4 Avgränsningar”. Dessa är:

- Inget stöd för analys av databaser med journalfiler
- Inget stöd för analys av enbart journalfiler utan tillhörande databas
- Inget stöd för analys av databaser med write-ahead log filer
- Inget stöd för analys av enbart write-ahead log filer utan tillhörande databas
- Inget stöd för högre schema layer filformat än ett
- Inget stöd för text som är avkodad i ett annat format än UTF-8

Stöd för de första fyra punkterna uteslöts ifrån början. Anledningen till detta är att journal- och write-ahead log filer har en annan uppbyggnad än databasfiler. Att utveckla funktioner för att stödja dessa filer hade tagit för lång tid och därför fattades beslutet att utesluta dem.

Beslutet att utesluta de två sista punkterna fattades i slutet av programmets utvecklingsperiod. Den avsatta tiden för programutvecklingen började ta slut och viktigare funktioner hade inte fullt ut utvecklats. Detta ledde till beslutet att utesluta alla oviktiga funktioner då dessa inte krävs för att programmet ska kunna uppfylla sitt syfte.

Programmets utskriftsfunktion har också en del brister. Funktionen kan inte alltid skriva ut den extraherade datan i tabellform. En preliminär lösning har implementerats där tabell- eller indexdata skrivs ut radvis dock gör detta datan svårläst. En annan brist med denna funktion kan leda till att tabellerna som skapas av funktionen inte formateras korrekt. Det leder ofta till att datan blir oläsligt.

En annan brist är att sökmetoderna som letar efter raderad data i databasen inte genomsöker alla lediga områden. Sökmetoderna finns beskrivna under rubrikerna ”4.1 Återkskapa records – record storlek matchning” och ”4.2 Återskapa records – record header typ matching”. De områden sökmetoderna inte genomsöker är fria block (free blocks) mellan cellerna på B-Tree sidor. Dessutom kan programmet enbart läsa in ett record prefix av en raderad record. Dessa brister beror enbart på tidsbrist.

Sökmetod två presterar ofta inte lika bra som sökmetod ett. Metoden är beroende av en annan funktion som läser ut kolumnernas datatyper. Denna funktion fungerar inte alltid korrekt och är orsaken till att sökmetod två missar raderade records.

Programmet saknar också stöd för sökmetod tre som beskrivs under rubriken ”4.3 Återskapa fragmenterad data”. Denna sökmetod implementerades inte då det inte fanns tillräckligt med tid. Anledningen till att denna sökmetod valdes var att den inte kan hitta kompletta records och att den har störst sannolikhet för falska positiver.

4 Utvinning av raderad data ur SQLite databaser

Raderas data i SQLite tas referensen till en B-Tree cell bort men inte datan. En cell kan antingen hålla ett record, sidnumret (page number) till en nodsida (node page) eller både och.

Vid återskapning av raderad data i SQLite söks alltså B-Tree celler som inte längre refereras till eller record headers av raderade records. I vissa fall förekommer det dock att bara fragment av raderade records finns kvar. I ett sådant fall måste ett annat tillvägagångssätt användas för att hitta den raderade datan.

4.1 Återskapa records – record storlek matchning

B-Tree celler sparar alltid storleken på det record som är sparad i dem. Varje record har dessutom en header som innehåller information header:ns storlek och om typen och storleken av alla inlägg i record:en. Det innebär att ett records storlek också kan beräknas med hjälp av värdena i dess header. Vid en sökning efter raderade records är det därför möjligt att läsa ut respektive beräkna dessa värden ur eventuella celler och record headers och jämföra dem med varandra. Överensstämmer dem hittades med största sannolikhet en riktig record.

Bild 11 är skapad i Microsoft PowerPoint och visar ett exempel på hur en tabell B-Tree delträd nodcell (table B-Tree internal tree node cell) med tillhörande record ser ut. I exemplet kommer först värdet för record storleken, därefter nyckelvärdet (key value) och sist record:en med record header:n i början. Alla värden, bortsett från record datan, är sparade som ett heltal med varierande längd (se rubriken ”Variable length integer” i Bilaga A). Tabell 4 under rubriken ”Record format” i Bilaga A visar betydelserna av värdena i record header:n.

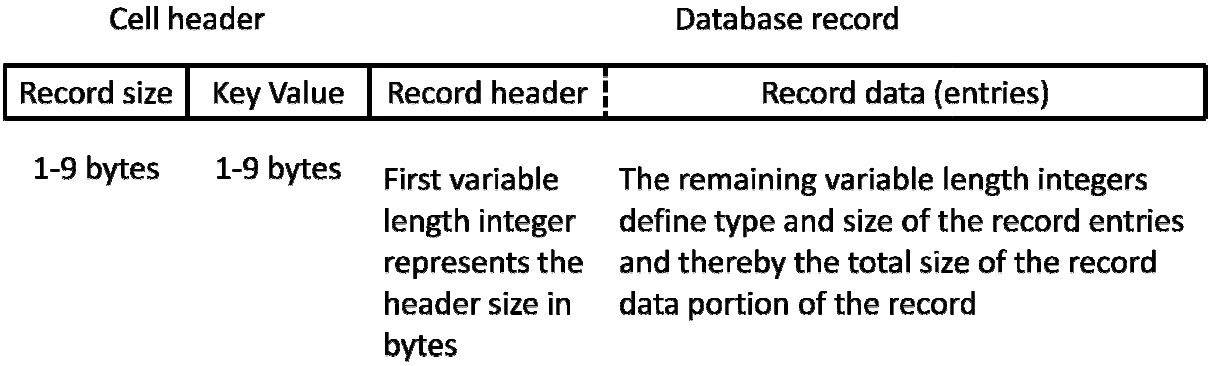

Bild 11 – Table B-Tree delträd nodcell med tillhörande record

För att minska antalet falska positiver är det möjligt att exkludera hittade records vars datatyper inte överensstämmer med befintliga tabeller eller index. Dock har detta nackdelen att raderade records som tillhörde tabeller eller index som inte längre finns också kommer att exkluderas.

Vid en sökning måste först en giltig cell hittas. En cell är giltig om sidnumret till en barnnod (enbart index B-Tree delträd nodceller) är:

- Lägre än två
- Inte är sidonumret till en pointer map sida
- Ett nummer som är större än antalet sidor i databasen

En B-Tree cell är också ogiltig om värdet för record storleken är:

- Lägre än två

- Större än databasens storlek i bytes

Tillhör cellen en table B-Tree cell så innehåller den också ett fält för ett nyckelvärde. I ett sådant fall är en cell också ogiltig om nyckelvärdet är lägre än ett.

Hittas en eventuell cell som är giltig måste värdena i den eventuella record header:n kontrolleras. En record header är ogiltig om första värdet är:

- Lägre än två
- Större än värdet för record storleken i cellen

En record header är också ogiltig om:

- Den totala storleken i bytes av alla värden i header:n inte motsvarar värdet för record header storleken.
- Något värde är tio eller elva

Har både en giltig cell och en giltig record header hittats beräknas och kontrolleras värdena för record storleken. Överensstämmer värdena har antagligen en riktig raderad record hittats.

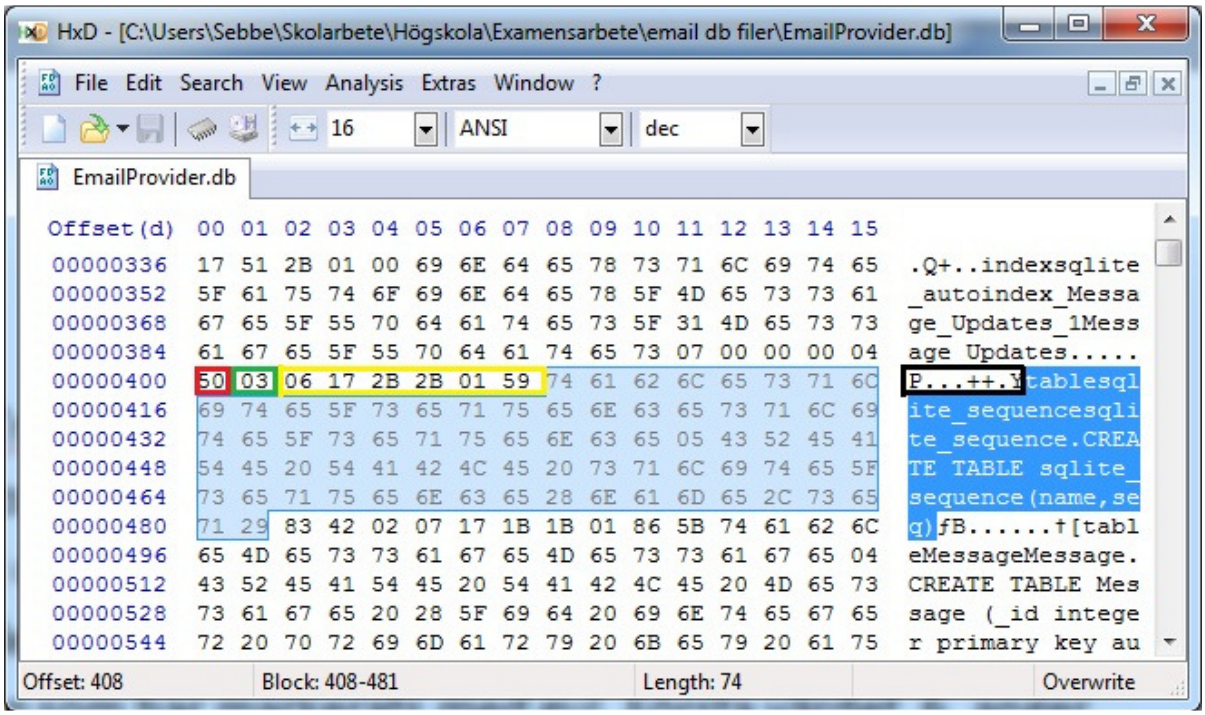

Bild 12 – Bild av en raderad cell med tillhörande record visad i en hexeditor

Bild 12 visar ett exempel av en raderad table B-Tree delträd nodcell med tillhörande record. De olika delarna av cell och record har markerats med olika färger. I den röda rutan visas record:ens storlek i bytes vilket i detta fall är 50 i hexadecimal form eller 80 i decimal form (alla värden är hexadecimala). I den gröna rutan finns nyckel värdet för denna record vilket i detta fall är 3. Därefter kommer record:en som inleds med record header:n. Record header:n har markerats med gul och record datan med blå.

Första värdet i record header:n anger dess storlek i bytes. I detta fall är det 6. De fem efterföljande värden anger typ och storlek för de inlägg som finns i record:en. I detta exempel

är de 23, 43, 43, 1 och 89 i decimal form (17, 2B, 2B, 01, 59 i hexdecimal form). Tolkas dessa värden som beskriven under rubriken ”Record format” i Bilaga A så har:

- Första inlägget typen text och längd 5 byte
- Andra inlägget typen text och längd 15 byte
- Tredje inlägget typen text och längd 15 byte
- Fjärde inlägget typen integer och längd 1 byte
- Femte inlägget typen text och längd 38 byte

Det ger en total record data storlek på 74 byte. Adderas sex byte för storleken av record header:n blir totala record storleken 80 byte. Detta överensstämmer med värdet för record storleken som fanns sparad i cellen.

Record:en i detta exempel tillhörde schematabellen och de fem inläggen är:

- table
- sqlite_sequence
- sqlite_sequence
- 5
- CREATE TABLE sqlite_sequence(name, seq)

4.1.1 Fördelar och nackdelar med denna metod

Fördelarna med denna metod är:

- Den kan hitta kompletta records
- Sannolikheten för falska positiver är relativt liten
- Datan kan eventuellt automatiskt tilldelas en befintlig tabell eller ett befintligt index

I vissa fall kan en raderad record automatiskt tilldelas en tabell eller ett index. Det kan göras genom att jämföra datatyperna av inläggen i record:en mot datatyperna av tabellernas och indexens kolumner. Finns det enbart en likhet är det möjligt att automatiskt tilldela den raderade record:en till en tabell eller ett index. Finns det dock inga eller flera likheter går record:en inte att automatiskt tilldela. Vid inga likheter är det också möjligt att record:en är ett falskt positiv.

Nackdelarna med denna metod är:

- Det finns ingen garanti att ett record kan återskapas komplett
- Den är mycket tidskrävande

Ett record kan enbart återskapas komplett om inte SQLite har skrivit över den med annan data. Har delar av record:en skrivits över är det omöjligt att återskapa dessa delar.

Denna metod behöver mycket tid för att leta igenom en databas då B-Tree celler har olika uppbyggnad. För varje måste metoden kontrollera om det går att bilda en giltig cell. Varje gång det går att bilda en giltig cell måste metoden också kontrollera om det går att bilda en giltig record header. I sämsta fallet kontrollerar metoden tre gånger samma sak utan resultat.

4.2 Återskapa records – record header typ matchning

Denna metod letar enbart efter giltiga record headers. När en giltig header har hittats jämförs inläggens antal och datatyper mot datatyperna av kolumner i befintliga tabeller och index. Definitionen av en giltig record header är fortfarande samma som i första metoden.

Risken för falska positiver med denna metod är betydligt högre än med första metoden speciellt om en tabell eller ett index har få inlägg. För att minimera denna risk kan eventuella records exkluderas om antalet inlägg i dem är för få.

Det är också möjligt att begränsa sökningen till records som tillhör en specifik tabell. Det kan uppnås genom att ange både ett minimum och ett maximum antal inlägg en hittad record måste ha.

Ofta är det också viktigt att betrakta ”NULL” värden som wildcards. Detta beror på att värdet i en kolumn kan vara ”NULL” dock är kolumnens datatyp fortfarande någonting annat. I ett sådant fall skulle denna metod exkludera hittade records trots att de är giltiga.

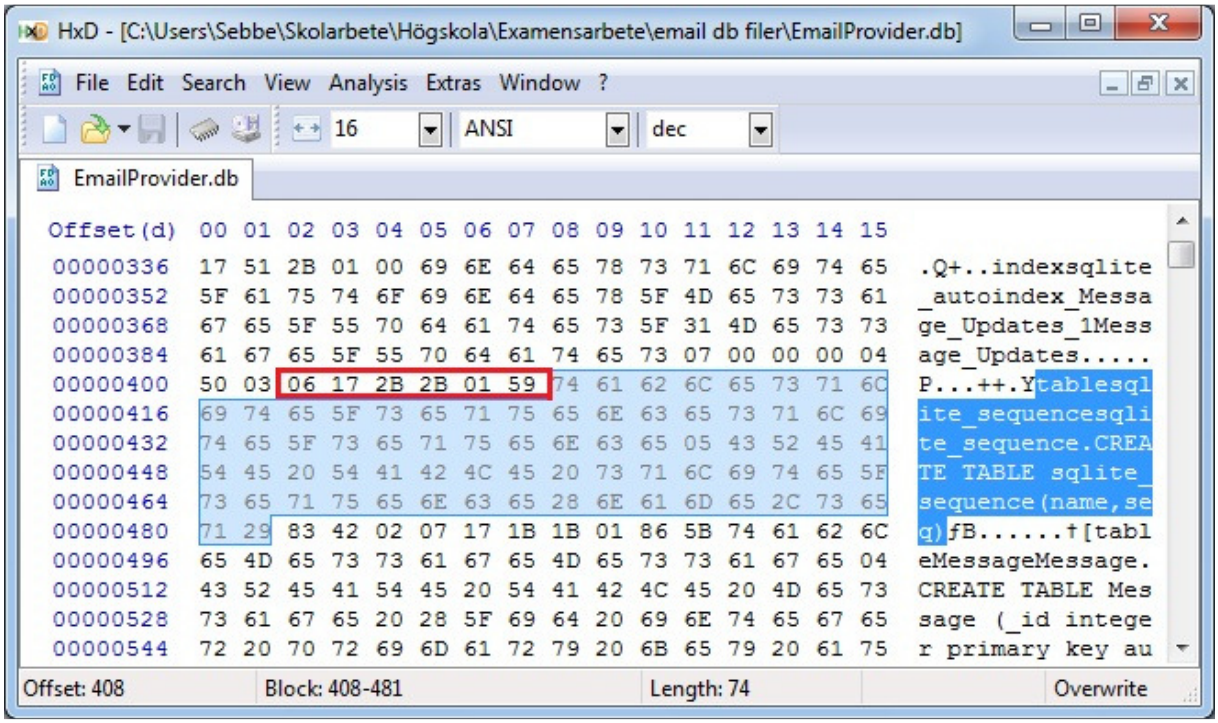

Bild 13 - Bild av en raderad record visad i en hexeditor

Exemplet som visas i Bild 13 är samma som föregående. Metod två börjar med att läsa in record header:ns storlek i bytes, vilket i detta fall är 6. Sedan läses de resterande fem bytes in. Resultatet blir återigen att:

- Första inlägget har typen text och längd 5 byte
- Andra inlägget har typen text och längd 15 byte
- Tredje inlägget har typen text och längd 15 byte
- Fjärde inlägget har typen integer och längd 1 byte
- Femte inlägget har typen text och längd 38 byte

Datatyperna för denna record är: text, text, text, integer och text. Dessa datatyper överensstämmer med datatyperna i schematabellen. Detta innebär att den raderade record:en antagligen är riktig och tillhör schematabellen.

4.2.1 Fördelar och nackdelar med denna metod

Fördelarna med denna metod är:

- Den kan hitta kompletta records
- Hittade records kan automatiskt tilldelas minst en tabell eller ett index
- Det krävs ingen cell header för att denna metod ska fungera

Då denna metod inte kräver någon cell header är den mycket lämpligt för att leta efter raderade records bland fria block (se rubrik ”B-Tree page format” i Bilaga A) i cell content area. Detta beror på att SQLite skriver över de fyra första byte:n i varje free block. Hade en cell header befunnit sig i början av ett fritt block (free block) hade den skrivits över och metod ett hade inte längre kunnat hitta den tillhörande record:en. Metod två kan dock fortfarande hitta denna record.

Denna metod har enbart nackdelen att risken för falska positiver är större. Även tidsåtgången för denna metod är rätt hög dock inte alls lika hög som för första metoden.

4.3 Återskapa fragmenterad data

I vissa fall saknas både en komplett cell- och record header. I ett sådant fall fungerar inte de två första metoderna. Det finns dock ytterligare ett sätt att hitta raderad data i SQLite databaser.

Det är möjligt att utföra en sökning efter strängar i databasen. Denna metod kräver ingen cell- eller record header men har nackdelen att enbart text kan återskapas. Det är teoretiskt sätt också möjligt att utföra en sökning efter numeriska värden dock är sannolikheten för genomslag mycket liten och risken för ett oläsligt resultat mycket hög.

För att kunna hitta strängar kräver denna metod en definition av värden som ska betraktas som tecken. Det skulle också vara lämpligt att definiera en minimum längd på hittade strängar då dessa måste ha en viss längd för att vara meningsfulla.

4.3.1 Fördelar och nackdelar med denna metod

Denna metod har enbart den fördelen att inga cell- eller record headers krävs. Därför lämpar sig denna metod främst för att hitta data de första två metoderna inte kan hitta.

Nackdelarna med denna metod är:

- Mycket hög risk för falska positiver
- Risk för svårläst resultat
- Mer eller mindre omöjligt att automatiskt tilldela datan till tabeller eller index

Vid sökningar efter strängar inkluderas ofta mycket skräpdata i resultatet. Det gör att resultatet blir svårläst och att skräpdatan måste sållas bort.

4.4 Begränsning av genomsökningen

För att spara tid och undvika dubbletter i resultatet är det viktigt att avgränsa en sökning till områden i databasen som är lediga. Det kan uppnås genom att spara början av alla lediga områden i en lista och längden av dessa områden i en annan.

Det finns tre typer av sidor som kan innehålla lediga områden. Dessa sidor är:

- B-Tree sidor
- Överskottssidor
- Free-list sidor

På B-Tree sidor finns det dels ett oanvänt utrymme mellan cell offset area och cell content area och dels fria block mellan cellerna i cell content area. Det finns också fragment mellan cellerna dock är dessa maximalt tre byte stora och kan därför exkluderas ur sökningen. Free-list sidor delas upp i två typer: trunk sidor (trunk page) och löv sidor (leaf page). Löv sidor används inte till att spara någon befintlig data och är därför ett enda ledigt område. En trunk sida och även en överskottssida innehåller enbart ett ledigt område om sidan är sist i kedjan. Det lediga området sträcker sig i så fall från datans slut till slutet av sidan.

Det kan också vara av fördel att utöka listorna under en sökning med områden som tas upp av raderad data som redan har hittats. På så sätt förhindras att de olika metoderna hittar samma data.

5 Slutsatser

5.1 Resultat

Resultatet av projektet blev tre metoder för att hitta raderad data i SQLite databaser och ett prototypprogram där två av dessa metoder implementerades. De två metoderna som implementerades i programmet kan båda hitta kompletta records och tredje funktionen kan hitta all data av typen text de första två missade.

5.1.1 Testning av programmet

Testerna utfördes med SQLite databaser som togs ifrån ett Android operativsystem som kördes i en emulator, ifrån en iPhone 3GS och ifrån Mozilla Firefox 3. Testerna användes för att kontrollera vilka databaser programmet kunde hantera, om det uppstod några fel vid exekveringen, hur snabbt programmet arbetar och hur resultatet ser ut.

Databaserna som användes vid testerna var de som innehåller data om kontakter, sms/mms, webbhistorik, E-post med mera. Programmet kunde hantera alla dessa databaser. Ibland uppstod det dock fel vid exekveringen vilket ledde till att programmet fick avslutas. Dessa fel var dock enbart programmeringsfel.

Testerna visade hur programmet presterar tidsmässigt. Testdatabasernas storlek varierade mellan några kilobyte och nästan fem megabyte. Om inga sökningar efter raderad data utfördes exekverade programmet vid alla tester mycket snabbt. När enbart metod ett användes för att söka efter raderad data ökades exekveringstiden kraftigt, ibland upp till fem sekunder. Metod två ökade också exekveringstiden mycket dock var den snabbare än metod ett. En sökning med båda metoderna resulterade därför nästan alltid i mycket långa exekveringstider, ibland över fem sekunder långa. Dessa långa exekveringstider beror på att varje sökfunktion måste söka igenom alla lediga områden byte för byte.

Resultatet blev att metod ett fungerar mycket bra och ofta hitta all raderad data. Metod två fungerar inte lika bra. Detta beror dock ofta på att funktionen som läser ut kolumnernas datatyper inte alltid fungerar korrekt. Då metod två behöver kolumnernas datatyper för att kunna jämföra dessa med datatyperna i eventuella record headers förklarar detta metodens dåliga prestation.

5.2 Problem

Problem som uppstod under detta arbetes gång hade främst med programmeringen att göra då det var en stor del av projektet.

Det första problemet som uppstod handlade om att hitta information om hur SQLite arbetar och hur databas images är uppbyggda. Detta problem kunde lösas då SQLite har en mycket bra och omfattande dokumentation om både filformatet och andra funktioner. Den beskriver allt som behöver vetas för att utveckla egna applikationer mot SQLite databaser.

Ett annat problem uppstod vid omvandlingen av heltal med varierande längd. Det tog mycket längre tid än förväntad innan programmet korrekt kunde omvandla ett heltal med varierande längd till sitt originalvärde. Detta berodde på att det var svårt att utveckla en bra lösning för omvandlingen. Problemet löstes genom att ta isär första värdet och att återuppbygga originalvärdet bit för bit utan alla kontrollbits.

Det uppstod också problem med att hantera överskottskedjor. Anledningen till det var att de befintliga funktionerna för att läsa in records inte kunde användas. Det ledde till att vissa förändringar på befintliga funktioner fick göras och att nya funktioner fick utvecklas. Dessa funktioner fick dessutom ändras och förbättras flera gånger innan de fungerade korrekt.

Under utvecklingen av utskriftsfunktionen uppstod det också problem, dock var detta förväntad och inplanerad. Problemet bestod i att få utskriften av tabellerna att bli bra.

Hanteringen av argumentinläsningen skapade också en del problem. Det berodde på att vissa optioner enbart ska vara tillgängliga om en viss sökmetod efter raderad data har valts. Det finns dock inget bra sätt att hantera detta på. Problemet kunde slutligen lösas genom att analysera de grundläggande argumenten först och sedan eventuella optioner som tillhör en specifik sökmetod.

Det uppstod även ett problem vid utvecklingen av sökfunktionerna då tidsåtgången underskattades vilket ledde till att tidsplanen inte kunde hållas. Skrivandet av rapporten gick dock snabbare än förväntat vilket vägde upp tidsförlusten.

5.3 Vidareutveckling

Det utvecklade programmet SQLite Data Finder är för tillfället enbart en prototyp som fortfarande har en del programmeringsfel och många brister. Det finns alltså mycket kvar att förbättra.

Det första som kan förbättras är alla brister som nämns under rubriken ”3.2 Programmets brister”. När det har gjorts kan det läggas till ytterligare funktioner som skulle förbättra programmet.

En tänkbar funktion skulle kunna vara en utskrift av de extraherade värdena till en kommaseparerad fil (CSV fil). Dessa filer kan läsas av andra program och dessa kan sedan använda den sparade datan på något sätt. Det skulle till exempel vara möjligt att utveckla ett program som läser in datan i dessa filer och presenterar denna grafiskt. Ett sådant program skulle kunna ha många funktioner som SQLite Data Finder inte kan ha, till exempel en sökfunktion.

En annan funktion som skulle kunna läggas till är en sök- och karvningsfunktion som letar efter SQLite databaser i diskavbildningar eller binära filer. Det är inte så lätt att karva ut dessa databaser då de saknar footer. Dock är det möjligt att beräkna databasens storlek genom att hitta antalet sidor i databasen.

5.4 Reflektioner

Detta arbete var väldigt intressant och lärorikt. Speciellt inom Python programmering och uppbyggnaden av SQLite databaser medförde arbetet utökade kunskaper. Det var också en värdefull erfarenhet att utföra ett så stort projekt.

Det är lite synd att programmet har så många brister. Från början var det tänkt att den skulle kunna hantera alla databaser. Det visade det sig dock snabbt att det var alldeles för lite tid för att hinna med att lägga till stöd för alla funktioner och därför valdes att utelämna de som inte var så viktiga.

Tidsmässigt gick det inte alls som det var planerat från början. Först var det tänkt att databasfiler skulle analysera för att förstå hur datan lagras. Det användes dock istället SQLite's dokumentation om filformatet. Detta gjorde dock att arbetet gick betydligt snabbare och utan dokumentationen hade det varit tvivelaktigt om arbetets mål verkligen hade uppnåtts. Efter detta gjordes vissa ändringar i tidsplanen som resulterade i att allt därefter gick som planerat.

6 Referensförteckning

6.1 Internetreferenser

SQLite. *About SQLite*. [online] Tillgänglig vid: <http://www.sqlite.org/about.html> [Hämtat den 6 april 2011].

CCL-Forensics. *EPILOG - SQLite database analysis tool*. [online] Tillgänglig vid: <http://www.ccl-forensics.com/Research-tools/epilog-sqlite-database-analysis-tool.html> [Hämtat den 5 maj 2011].

PenTestIT, 2009. *ff3hr: The Firefox 3 History Recoverer!*. [online] Tillgänglig vid: <http://www.pentestit.com/2009/11/09/ff3hr-firefox-3-history-recoverer/> [Hämtat den 5 maj 2011].

Python Software Foundation. *Python Programming Language – Official Website*. [online] Tillgänglig vid: <http://www.python.org/> [Hämtat den 5 maj 2011].

Chirashi Security, 2010. *Recover Deleted Data from SQLite Databases*. [online] Tillgänglig vid: <http://chirashi.zensay.com/2010/11/recover-deleted-data-from-sqlite-databases/> [Hämtat den 5 maj 2011].

SQLite. *SQLite Database File Format*. [online] Tillgänglig vid: <http://www.sqlite.org/fileformat.html> [Hämtat den 4 april 2011].

SQLite. *sqlite – Management Tools*. [online] Tillgänglig vid: <http://www.sqlite.org/cvstrac/wiki?p=ManagementTools> [Hämtat den 5 maj 2011].

SQLite. *Well Known Users of SQLite*. [online] Tillgänglig vid: <http://www.sqlite.org/famous.html> [Hämtat den 5 maj 2011].

6.2 Bildrefenser

SQLite. *Figure 7 - Table B-Tree Tree Structure*. [bild online] Tillgänglig vid: <http://www.sqlite.org/fileformat.html> [Hämtat den 11 maj 2011].

SQLite. *Figure 3 - Index B-Tree Tree Structure*. [bild online] Tillgänglig vid: <http://www.sqlite.org/fileformat.html> [Hämtat den 11 maj 2011].

SQLite. *Figure 4 - Index B-Tree Page Data*. [bild online] Tillgänglig vid: <http://www.sqlite.org/fileformat.html> [Hämtat den 11 maj 2011].

SQLite. *Figure 5 - Small Record Index B-Tree Cell*. [bild online] Tillgänglig vid: <http://www.sqlite.org/fileformat.html> [Hämtat den 11 maj 2011].

SQLite. *Figure 6 - Large Record Index B-Tree Cell*. [bild online] Tillgänglig vid: <http://www.sqlite.org/fileformat.html> [Hämtat den 11 maj 2011].

SQLite. *Figure 8 - Table B-Tree Internal Node Cell*. [bild online] Tillgänglig vid: <http://www.sqlite.org/fileformat.html> [Hämtat den 11 maj 2011].

SQLite. *Figure 9 - Table B-Tree Small Record Leaf Node Cell*. [bild online] Tillgänglig vid: <http://www.sqlite.org/fileformat.html> [Hämtat den 11 maj 2011].

SQLite. *Figure 10 - Table B-Tree Large Record Leaf Node Cell*. [bild online] Tillgänglig vid: <http://www.sqlite.org/fileformat.html> [Hämtat den 11 maj 2011].

SQLite. *Figure 12 - Free List Trunk Page Format*. [bild online] Tillgänglig vid: <http://www.sqlite.org/fileformat.html> [Hämtat den 12 maj 2011].

6.3 Tabellrefenser

SQLite. *SQLite Database File Format*. [online] Tillgänglig vid: <http://www.sqlite.org/fileformat.html> [Hämtat den 11 maj 2011].

Bilagor

Bilaga A, SQLite databas filformat

Databas header

SQLite databas header:n är 100 byte stor och börjar alltid med sekvensen ”SQLite format 3” följd av en null-terminator byte. Denna sekvens följs av ett antal värden som innehåller information SQLite använder för att kunna läsa och skriva till databasen. Några exempel på informationen som lagras i databas header:n är:

- Sidstorleken/page size (sidor/pages förklaras under nästa rubrik)
- Schema layer filformat
- Ifall databasen är auto-vacuum kapabel
- Text avkodningen

En komplett lista över all information lagrad i databas header:n finns i Tabell 1.[1] Alla värden i både databas header:n och resten av databasen är sparade i big-endian formatet. (SQLite, datum okänt)

Tabell 1 – Beskrivning av SQLite databas header:n

Byte Range	Byte Size	Description	Reqs
16..17	2	Database *page size* in bytes. See section 2.2.2 for details.	H30190
18	1	File-format "write version". Currently, this field is always set to 1. If a value greater than 1 is read by SQLite, then the library will only open the file for read-only access. This field and the next one are intended to be used for forwards compatibility, should the need ever arise. If in the future a version of SQLite is created that uses a file format that may be safely read but not written by older versions of SQLite, then this field will be set to a value greater than 1 to prevent older SQLite versions from writing to a file that uses the new format.	H30040
19	1	File-format "read version". Currently, this field is always set to 1. If a value greater than 1 is read by SQLite, then the library will refuse to open the database Like the "write version" described above, this field exists to facilitate some degree of forwards compatibility, in case it is ever required. If a version of SQLite created in the future uses a file format that may not be safely read by older SQLite versions, then this field will be set to a value greater than 1.	H30040
20	1	Number of bytes of unused space at the end of each database page. Usually this field is set to 0. If it is non-zero, then it contains the number of bytes that are left unused at the end of every database page (see section 2.2.2 for a description of a database page).	H30040
21	1	Maximum fraction of an index tree page to use for embedded content. This value is used to determine the maximum size of a B-Tree cell to store as embedded content on a page that is part of an index B-Tree. Refer to section 2.3.3.4 for details.	H30040
22	1	Minimum fraction of an index B-Tree page to use for embedded content when an entry uses one or more overflow pages. This value is used to determine the portion of a B-Tree cell that requires one or more overflow pages to store as embedded content on a page that is part of an index B-Tree. Refer to section 2.3.3.4 for details.	H30040
23	1	Minimum fraction of an table B-Tree leaf page to use for embedded content when an entry uses one or more overflow pages. This value is used to determine the portion of a B-Tree cell that requires one or more overflow pages to store as embedded content on a page that is a leaf of a table B-Tree. Refer to section 2.3.4.3 for details.	H30040
24..27	4	The file change counter. Each time a database transaction is committed, the value of the 32-bit unsigned integer stored in this field is incremented. SQLite uses this field to test the validity of its internal cache. After unlocking the *database file*, SQLite may retain a portion of the file cached in memory. However, since the file is unlocked, another process may use SQLite to modify the contents of the file, invalidating the internal cache of the first process. When the file is relocked, the first process can check if the value of the file change counter has been modified since the file was unlocked. If it has not, then the internal cache may be assumed to be valid and may be reused.	H33040
28..31	4	The in-header database size. This field holds the logical size of the *database file* in pages. This field is only valid if it is nonzero and if the file change counter at offset 24 exactly matches the version-valid-for at offset 92. The in-header database size will only be valid when the database was last written by SQLite version 3.7.0 or later. If the in-header database size is valid, then it is used as the logical size of the database. If the in-header database size is not valid, then the actual *database file* size is examined to determine the logical database size.	
32..35	4	Page number of first freelist trunk page. For more details, refer to section 2.4.	H31320
36..39	4	Number of free pages in the *database file*. For more details, refer to section 2.4.	H31310
40..43	4	The schema version. Each time the database schema is modified (by creating or deleting a database table, index, trigger or view) the value of the 32-bit unsigned integer stored in this field is incremented.	H33050
44..47	4	Schema layer file-format. This value is similar to the "read-version" and "write-version" fields at offsets 18 and 19 of the *database header*. If SQLite encounters a database with a schema layer file-format value greater than the file-format that it understands (currently 4), then SQLite will refuse to access the database. Usually, this value is set to 1. However, if any of the following file-format features are used, then the schema layer file-format must be set to the corresponding value or greater: 2. Implicit NULL values at the end of table records (see section 2.3.4.1). 3. Implicit default (non-NULL) values at the end of table records (see section 2.3.4.1). 4. Descending indexes (see section 2.3.3.2) and Boolean values in database records (see section 2.3.2, serial types 8 and 9). *TODO: Turns out SQLite can be tricked into violating this. If you delete all tables from a database and then VACUUM the database, the schema layer file-format field somehow gets set to 0.*	H30120
48..51	4	Default pager cache size. This field is used by SQLite to store the recommended pager cache size to use for the database.	H30130
52..55	4	For auto-vacuum capable databases, the numerically largest root-page number in the database. Since page 1 is always the root-page of the schema table (section 2.2.3), this value is always non-zero for auto-vacuum databases. For non-auto-vacuum databases, this value is always zero.	H30140, H30141
56..59	4	(constant) Database text encoding. A value of 1 means all text values are stored using UTF-8 encoding. 2 indicates little-endian UTF-16 text. A value of 3 means that the database contains big-endian UTF-16 text.	H30150
60..63	4	The user-cookie value. A 32-bit integer value available to the user for read/write access.	H30160
64..67	4	The incremental-vacuum flag. In non-auto-vacuum databases this value is always zero. In auto-vacuum databases, this field is set to 1 if "incremental vacuum" mode is enabled. If incremental vacuum mode is not enabled, then the *database file* is reorganized so that it contains no free pages (section 2.4) at the end of each database transaction. If incremental vacuum mode is enabled, then the reorganization is not performed until explicitly requested by the user.	H30171
92..95	4	The version-valid-for integer. This is a copy of the file change counter (offset 24) for when the SQLite version number in offset 96 was written. This integer is also used to determine if the in-header database size (offset 28) is valid.	

Pages och page typer

SQLite databaser är uppdelade i sidor (pages). Alla sidor i en databas har samma storlek och storleken är alltid en tvåpotens mellan 512 (2^9) och 65536 (2^{16}). Sida ett innehåller databas

[1] SQLite, datum okänt

header:n och är schematabellens rotsida (root page). Schematabellen beskrivs närmare under nästa rubrik.

Det finns fem huvudtyper av sidor:

- B-Tree sidor
- Överskottssidor (overflow pages)
- Fria sidor (free pages)
- Pointer map sidor
- Locking sidan

B-Tree sidor tillhör B-Tree strukturer och kan delas upp i rotsidor, delträdsidor och lövsidor. B-Tree strukturer beskrivs närmare under rubriken ”B-tree strukturer”.

Överskottssidor förklaras under rubriken ”Overflow chains”, fria sidor under rubriken ”Free pages” och pointer map sidor under rubriken ”Pointer map pages”.

Locking sidan börjar vid en gigabyte gränsen (byte 2^{30}) och förekommer enbart en gång i databasen. Denna sida används aldrig av SQLite. (SQLite, datum okänt)

Schematabellen

Schematabellen innehåller data om alla schemaobjekt, förutom sig själv, som finns sparade i databasen. Schemaobjekt kan vara tabeller, virtuella tabeller, index, triggers och views.

Tabeller och index i SQLite databaser består av records. Dessa beskrivs närmare under rubriken ”Record format”. Ett record motsvarar en rad i en tabell eller ett index. Varje record i schematabellen innehåller information om ett schemaobjekt i databasen och består av fem värden.

Första värdet är en sträng och beskriver schemaobjektets typ. Typen kan antingen vara table (tabell), index, trigger eller view. Andra fältet är också en sträng och innehåller schemaobjektets namn. Även tredje värdet är en sträng och anger namnet av tabellen schemaobjektet är associerad med. Fjärde fältet är ett integer värde och anger numret till objektets rotsida. Femte fältet är en sträng och innehåller SQL kommandot som användes för att skapa schemaobjektet. (SQLite, datum okänt)

Tabell 2 innehåller en komplett beskrivning av värdena i schematabellens records.[2]

Tabell 2 – Beskrivning av de fem värdena varje record i schematabellen innehåller

Field	Description
Schema item type.	A string value. One of "table", "index", "trigger" or "view", according to the schema item type. Entries associated with UNIQUE or PRIMARY KEY clauses have this field set to "index".
Schema item name.	A string value. The name of the database schema item (table, index, trigger or view) associated with this record, if any. Entries associated with UNIQUE or PRIMARY KEY clauses have this field set to a string of the form "sqlite_autoindex_<name>_<idx>" where <name> is the name of the SQL table and <idx> is an integer value.
Associated table name.	A string value. For "table" or "view" records this is a copy of the second (previous) value. For "index" and "trigger" records, this field is set to the name of the associated database table.
The "root page" number.	For "trigger" and "view" records, as well as "table" records associated with virtual tables, this is set to integer value 0. For other "table" and "index" records (including those associated with UNIQUE or PRIMARY KEY clauses), this field contains the root page number (an integer) of the B-Tree structure that contains the table or index data.
The SQL statement.	A string value. The SQL statement used to create the schema item (i.e. the complete text of an SQL "CREATE TABLE" statement). This field contains an empty string for table entries associated with PRIMARY KEY or UNIQUE clauses. *TODO:Refer to some document that describes these SQL statements more precisely.*

[2] SQLite, datum okänt

B-Tree strukturer

B-Tree strukturer är balanserade sökträd där varje nod har noll, en eller flera barnnoder. Större delen av databasen består av dessa strukturer. Schematabellen är ett exempel på en B-Tree struktur. Varje B-Tree struktur kan bestå av en eller flera sidor där varje sida är en knutpunkt (node) i det binära trädet.

En B-Tree struktur består alltid av en rotsida (root page). Rotsidan är den sidan i det binära trädet varifrån det är möjligt att återfinna alla andra sidor. Bild 14 och Bild 15 (SQLite, datum okänt) visar exempel på B-Tree strukturer. I Bild 14 har rotsidan värdet åtta och i Bild 15 har den värdet R18.

En B-Tree struktur kan även, men måste inte, bestå av delträdsidor (internal tree node page) och lövsidor (leaf node page). Både Bild 14 och Bild 15 har två delträdssidor. Dessa ligger på nivån nedanför respektive rotsida. Sidorna i nedersta nivån är lövsidorna och alla lövsidor ligger alltid på samma nivå.

B-Tree strukturer delas också in i två typer: table (tabell) B-Tree strukturer och index B-Tree strukturer. Det som främst skiljer dessa åt är vilka värden som sparas på delträd- och lövsidor. (SQLite, datum okänt)

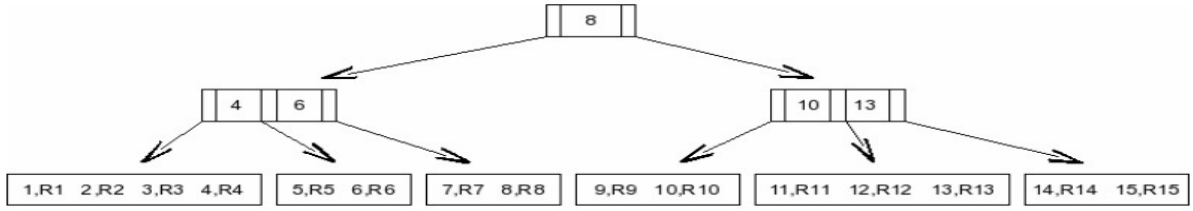

Bild 14 – Exempel på en table B-Tree struktur

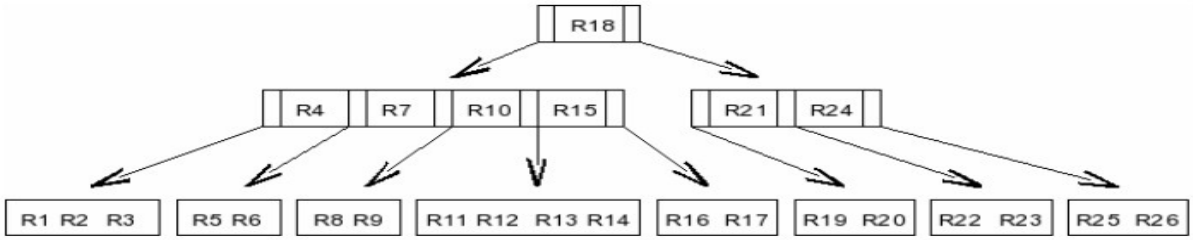

Bild 15 – Exempel på en index B-Tree struktur

B-Tree page format

En B-Tree sida (page) börjar alltid med ett sidhuvud (page header). För delträdssidor är denna tolv bytes stor och för lövsidor 8 bytes. Första byte:n anger sidans typ och kan vara:

- 2 (0x02) för index B-Tree delträdssidor
- 10 (0x0A) för index B-Tree lövsidor
- 5 (0x05) för tabell B-Tree delträdssidor
- 13 (0x0D) för tabell B-Tree lövsidor

Ifall sidan är en delträdssida anger de sista fyra byte:n i header:n numret till den högra barnsidan (child page). Tabell 3 förklarar utförligt index B-Tree sidhuvudets bytes och deras

betydelse.[3] Det enda som skiljer ett index B-Tree sidhuvud mot ett tabell B-Tree sidhuvud är värdet sparad i första byte:n. (SQLite, datum okänt)

Tabell 3 – Beskrivning av index B-Tree sidhuvudet

Byte Range	Byte Size	Description
0	1	B-Tree page flags. For an index B-Tree internal tree node page, this is set to 0x02. For a leaf node page, 0x0A.
1..2	2	Byte offset of first block of free space on this page. If there are no free blocks on this page, this field is set to 0.
3..4	2	Number of cells (entries) on this page.
5..6	2	Byte offset of the first byte of the cell content area (see figure 4), relative to the start of the page. If this value is zero, then it should be interpreted as 65536.
7	1	Number of fragmented free bytes on page.
8..11	4	Page number of rightmost child-page (the child-page that heads the sub-tree in which all records are larger than all records stored on this page). This field is not present for leaf node pages.

Efter B-Tree sidhuvudet kommer cell offset array:n. I denna sparas filpositionerna till alla celler på sidan som 2-byte stora heltalsvärden. Alla celler är sorterade i stigande ordning enligt record:en sparad i respektive cell. Records som tillhör tabell B-Trees har nyckelvärden och är sorterade efter dessa. Index B-Tree records är sorterade efter databas records. Ett record anses vara större om den innehåller fler inlägg (entries). Ifall antalet inlägg är lika avgör en jämförelsefunktion i SQLite vilket record som är större.

Efter cell offset area kommer ett block av oanvänt utrymme. Detta block tar upp all utrymme som sidhuvudet, cell offset area och cell content area inte tar upp.

Sista delen av en B-Tree sida är cell content area. Denna innehåller en cell för varje record sparad på sidan. Är sidan av typen tabell delträd innehåller varje cell istället numret till en barnsida.

Mellan cellerna på en sida kan det också finnas block av oanvänt utrymme. Dessa block kan antingen vara fragment eller fria block (free block). Fragment är maximalt tre bytes stora och det totala antalet fragmenterade bytes på en sida är aldrig större än 255. Free blocks är minst fyra byte stora och är ihopkopplade till en lista. De fyra första byte:n på varje fritt block används för att spara nästa blocks filposition (offset) och blockets storlek i bytes. (SQLite, datum okänt)

Alla filpositioner som anger en position på samma sida är relativa till sidans början. Om till exempel sidans första byte är på filposition 4096 och en cells filposition angavs som 130 så innebär det att cellens första byte börja på filposition 4096 + 130, alltså 4226.

Bild 16 (SQLite, datum okänt) ger en kort överblick över hur B-Tree sidor är uppbyggda.

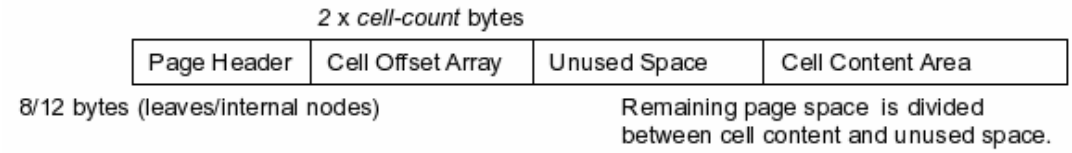

Bild 16 – Överblick över B-Tree siduppbyggnaden [B3]

[3] SQLite, datum okänt

B-Tree cell format

Det finns många faktorer som avgör formatet på en B-Tree cell. Typen av B-Tree strukturen och sidtypen cellen tillhör är de främsta faktorerna. Dock spelar även den sparade record:ens storlek en roll.

De värden som kan finnas i en B-Tree cell är:

- Record storlek (heltal med varierande längd 1-9 bytes)
- Numret till en barnsida (4 bytes)
- Ett nyckelvärde (heltal med varierande längd 1-9 bytes)
- En databas record (varierande storlek)
- En databas record prefix (varierande storlek)
- Numret till första sidan i en överskottskedja/overflow chain (4 bytes)

(SQLite, datum okänt)

Bild 17 till och med 21 (SQLite, datum okänt) visar hur de olika B-Tree cellerna är uppbyggda.

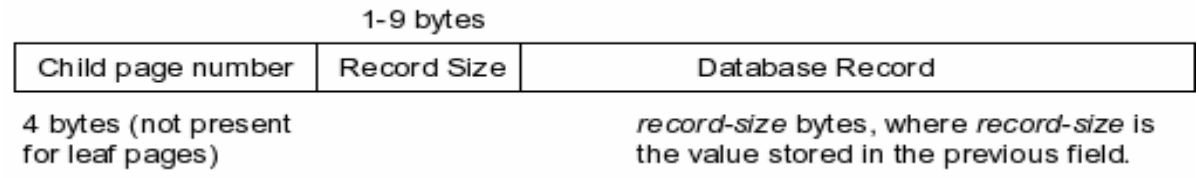

Bild 17 – Överblick över index B-Tree cell uppbyggnaden med en komplett record [B4]

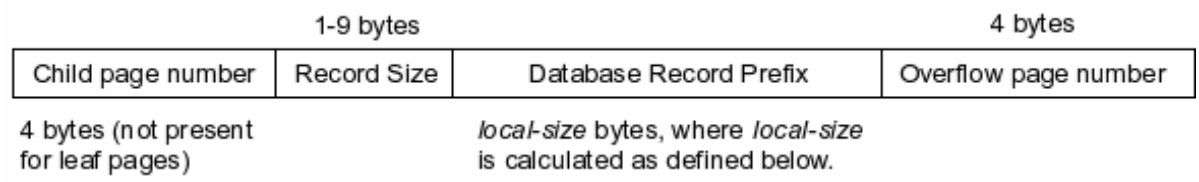

Bild 18 – Överblick över index B-Tree cell uppbyggnaden med ett record prefix [B5]

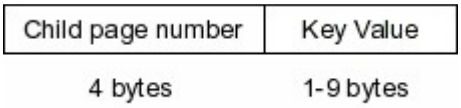

Bild 19 – Överblick över tabell B-Tree delträdssida cell uppbyggnaden [B6]

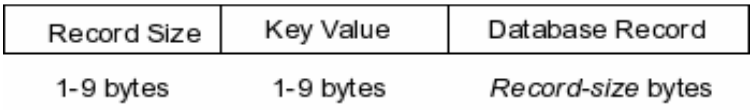

Bild 20 – Överblick över table B-Tree lövsida cell uppbyggnaden med en komplett record [B7]

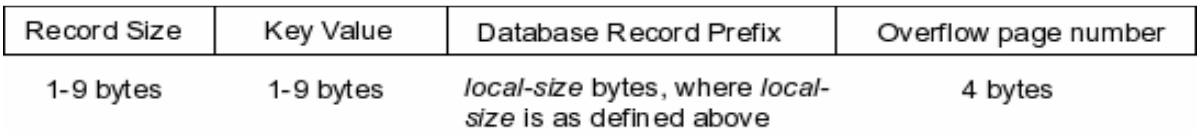

Bild 21 – Överblick över table B-Tree lövsida cell uppbyggnaden med ett record prefix [B8]

I vissa fall förekommer det att ett record är för stor för att kunna sparas komplett på en sida. I ett sådant fall måste record:en fördelas på en överskottskedja. På den aktuella sidan sparas ett record prefix vars storlek beräknas med en speciell formel. I anslutningen till recod prefixen sparas sedan sidnumret till första sidan i överskottskedjan. (SQLite, datum okänt)

Alla formler SQLite använder sig utav finns beskrivna i SQLite's dokumentation om databasernas filformat på http://www.sqlite.org/fileformat.html.

Record format

Varje rad i en tabell eller i ett index motsvaras av en SQLite record. Records är uppdelade i en record header och en datadel. Första värdet i record header:n anger dess storlek i bytes och de efterföljande värdena beskriver typen och storleken av inläggen (entries) i record:ens datadel. Alla värden i record header:n är heltal med varierande längd.

De vanligaste datatyperna i ett record är NULL, heltal (integer), reella tal (float), binary large objects (blob) och text (string). Tabell 4 beskriver alla datatyper och vilka värden de representeras av.[4] Dessutom beskriver den hur inläggens storlek beräknas med hjälp av värdena i record header:n. (SQLite, datum okänt)

Tabell 4 – Innebörd av värden i SQLite record header:n

Header Value	Data type and size
0	An SQL NULL value (type SQLITE_NULL). This value consumes zero bytes of space in the record's data area.
1	An SQL integer value (type SQLITE_INTEGER), stored as a big-endian 1-byte signed integer.
2	An SQL integer value (type SQLITE_INTEGER), stored as a big-endian 2-byte signed integer.
3	An SQL integer value (type SQLITE_INTEGER), stored as a big-endian 3-byte signed integer.
4	An SQL integer value (type SQLITE_INTEGER), stored as a big-endian 4-byte signed integer.
5	An SQL integer value (type SQLITE_INTEGER), stored as a big-endian 6-byte signed integer.
6	An SQL integer value (type SQLITE_INTEGER), stored as an big-endian 8-byte signed integer.
7	An SQL real value (type SQLITE_FLOAT), stored as an 8-byte IEEE floating point value.
8	The literal SQL integer 0 (type SQLITE_INTEGER). The value consumes zero bytes of space in the record's data area. Values of this type are only present in databases with a schema file format (the 32-bit integer at byte offset 44 of the *database header*) value of 4 or greater.
9	The literal SQL integer 1 (type SQLITE_INTEGER). The value consumes zero bytes of space in the record's data area. Values of this type are only present in databases with a schema file format (the 32-bit integer at byte offset 44 of the *database header*) value of 4 or greater.
bytes * 2 + 12	Even values greater than or equal to 12 are used to signify a blob of data (type SQLITE_BLOB) (*n*-12)/2 bytes in length, where *n* is the integer value stored in the record header.
bytes * 2 + 13	Odd values greater than 12 are used to signify a string (type SQLITE_TEXT) (*n*-13)/2 bytes in length, where *n* is the integer value stored in the record header.

Variable length integer

SQLite använder i vissa fall ett speciellt format på heltalsvärden (integer). Dessa värden kallas variable length integer och betyder på svenska heltal mer varierande längd. Heltal med varierande längd används alltid när storleken på ett heltal inte är förutbestämd.

Ett heltal med varierande längd kan vara en till nio bytes stor. Första bit:en i varje byte används som ett kontrollvärde. En etta indikerar att aktuell byte inte är den sista som tillhör värdet. En nolla indikerar istället att aktuella byte:n är den sista som tillhör värdet. Det enda undantaget är 9-byte heltal med varierande längd. Dessa har första bit:en i de första åtta byte:n satt till ett och sista byte:n innehåller ingen kontrollbit. Tabell 5 illustrerar hur de olika bitmönstren ser ut.[5]

Ett heltal med varierande längd omvandlas till sitt originalvärde genom att alla kontrollbits tas bort. Därefter måste eventuellt nollor infogas i början av bitsekvensen tills antalet bits är en multipel av åtta. Detta måste göras då det annars eventuellt inte finns tillräckligt många bits för att bilda alla bytes. (SQLite, datum okänt)

[4] SQLite, datum okänt
[5] SQLite, datum okänt

Tabell 5 – Uppbyggnad av variable length integer formatet

Bytes	Value Range	Bit Pattern
1	7 bit	0xxxxxxx
2	14 bit	1xxxxxxx 0xxxxxxx
3	21 bit	1xxxxxxx 1xxxxxxx 0xxxxxxx
4	28 bit	1xxxxxxx 1xxxxxxx 1xxxxxxx 0xxxxxxx
5	35 bit	1xxxxxxx 1xxxxxxx 1xxxxxxx 1xxxxxxx 0xxxxxxx
6	42 bit	1xxxxxxx 1xxxxxxx 1xxxxxxx 1xxxxxxx 1xxxxxxx 0xxxxxxx
7	49 bit	1xxxxxxx 1xxxxxxx 1xxxxxxx 1xxxxxxx 1xxxxxxx 1xxxxxxx 0xxxxxxx
8	56 bit	1xxxxxxx 1xxxxxxx 1xxxxxxx 1xxxxxxx 1xxxxxxx 1xxxxxxx 1xxxxxxx 0xxxxxxx
9	64 bit	1xxxxxxx 1xxxxxxx 1xxxxxxx 1xxxxxxx 1xxxxxxx 1xxxxxxx 1xxxxxxx 1xxxxxxx xxxxxxxx

Overflow chains

Overflow chains, eller på svenska överskottskedjor, används för att spara records som är för stora för att kunna sparas på en enda sida. En sådan record delas upp i ett record prefix och en resterande del. Record prefixen sparas på B-Tree sidan där cellen som record:en tillhör finns. Den resterande delen av record:en fördelas på en eller flera överskottssidor (overflow page).

En överskottskedja består av ett record prefix och en eller flera överskottssidor. Direkt efter record prefixen finns numret (page number) till första överskottssidan som ett 4-byte stort heltal. Alla överskottssidor börjar med numret till nästa sida i kedjan vilket också är sparad i som ett 4-byte stort heltal. Sista sidan indikeras genom att detta nummer är satt till noll. All resterande utrymme på en överskottssida används för att spara record data. (SQLite, datum okänt)

Free pages

Free pages, eller fria sidor på svenska, är sidor som har tagits bort från B-Tree strukturer och som för tillfället inte används för att spara någon giltig data.

Fria sidor delas upp i två typer: free-list trunk sidor (trunk page) och free-list lövsidor (leaf page). Free-list trunk sidor bildar, likt överskottssidor, en kedja och används för att spara sidonummer till free-list lövsidor. Free-list lövsidor används däremot till ingenting. Sidnumret till första free-list trunk sida och antalet fria sidor finns sparade i databas header:n (se Tabell 1).

Bild 22 (SQLite, datum okänt) illustrerar uppbyggnaden av free-list trunk sidor. De första fyra byte:n anger sidnumret till nästa trunk sida i kedjan. Är detta nummer noll indikerar det att denna sida är sista i kedjan. Nästa fyra byte anger hur många lövsidnummer en trunk sida innehåller och alla resterande utrymme används för att lagra dessa sidnummer. (SQLite, datum okänt)

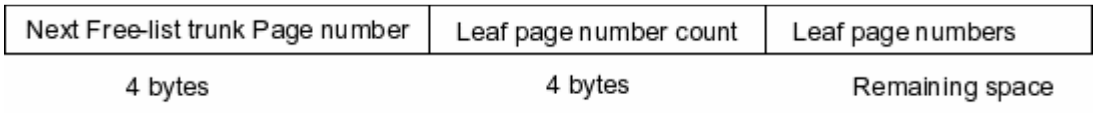

Bild 22 – Formatet av free-list trunk sidor [B9]

Pointer map pages

Pointer map sidor finns enbart i databaser som är auto-vacuum kapabla. Ifall databasen är auto-vacuum kapabel är sida två i databasen alltid en pointer map sida. Sidnumren av alla

andra pointer map sidor kan beräknas med en speciell formel som kan hittas i SQLite's dokumentation om databasernas filformat på http://www.sqlite.org/fileformat.html.

Pointer map sidor anger typen och föräldersida (parent page) för alla sidor i databasen. De enda undantagen är pointer map sidor, locking sidan och sida ett. En barnsidas föräldersida är sidan som refererade till den.

En pointer map sida innehåller ett inlägg (entry) för alla sidor från och med efterföljande sida fram till nästa pointer map sida. Första inlägget på en pointer map sida innehåller typen och föräldersidan för nästa sida, andra inlägget för nästnästa sida och så vidare. Varje inlägg är fem byte stor där första byte:n anger typen och de återstående fyra byte:n sidnumret till föräldersidan. Tabell 6 förklarar hur värdena i inläggens första byte ska tolkas.[6] (SQLite, datum okänt)

Tabell 6 – Pointer map page inlägg typvärden och deras beskrivning

Page Type	Byte Value	Description
B-Tree Root Page	0x01	The page is the root page of a table or index B-Tree structure. There is no parent page number in this case, the value stored in the pointer map lookup table is always zero.
Free Page	0x02	The page is part of the free page list (section 2.4). There is no parent page in this case, zero is stored in the lookup table instead of a parent page number.
Overflow type 1	0x03	The page is the first page in an overflow chain. The parent page is the B-Tree page containing the B-Tree cell to which the overflow chain belongs.
Overflow type 2	0x04	The page is part of an overflow chain, but is not the first page in that chain. The parent page is the previous page in the overflow chain linked-list.
B-Tree Page	0x05	The page is part of a table or index B-Tree structure, and is not an overflow page or root page. The parent page is the page containing the parent tree node in the B-Tree structure.

[6] SQLite, datum okänt

Bilaga B, Ordlista

Auto-vacuum: En databas kan antingen vara auto-vacuum kapabel eller inte. Är en databas auto-vacuum kapabel finns det en speciell modus som, om aktiverad, ser till att det inte finns några fria sidor i databasen efter en databas transaktion. Är denna modus inte aktiverad måste användaren uttryckligen efterfråga en sådan åtgärd.

Balanserad sökträd: Ett balanserat sökträd är en träddatastruktur som håller data sorterad och som tillåter sökningar, sekventiell åtkomst, att data läggs till och raderas. I toppen på trädet är roten som sedan grenar ut sig till delträd och slutligen löv.

Beta: Beta kallas det stadiet i ett programs utveckling där programmet är så långt utvecklad att det kan börja testas för att hitta och fixa programmeringsfel. Det finns öppen (open) och stängd (closed) beta. Vid öppen beta kan vem som helst ladda ner en kopia av programmet för att testa det och vid stängd beta är det enbart utvalda personer som har fått tillgång till programmet.

Big-endian: Endian beskriver byteordningen i ett värde som sträcker sig över flera byte. I big-endian kommer högsta byte:n först och lägsta byte:n sist. Detta kan jämföras med decimalsystemet där hundratal kommer först, sedan tiotal och sist ental. I little-endian är detta exakt tvärtom.

Binary large object (blob): Blobs är binär data som har sparats i en enda stor klump i en databas. Exempel på blobs är ofta bilder, audio- och annan multimedia data.

Databas: En databas är en organiserad samling av data. För att en samling av data ska klassificeras som en databas måste datan dock vara hanterad. Det innebär att det ska vara möjligt att söka, lägga till, ändra och radera data på ett enkelt sätt.

Footer: Footer kallas ett avsnitt av data i slutet på en fil eller ett objekt. En fil footer brukar ha en sekvens av värden som alltid finns i varje fil footer som tillhör samma filtyp.

Header: Header kallas ett avsnitt av data i början på en fil eller ett objekt som innehåller information om hur resten av filen eller objektet ska tolkas. En fil header brukar ha en sekvens av värden som alltid finns i varje fil header som tillhör samma filtyp.

Heap utrymme: Ett gemensamt minnesutrymme för program där minne för programmen allokeras dynamisk.

Inbyggda system:	Inbyggda system är datorliknande system som ingår i enheter vilka har en eller ett fåtal specifika funktioner. Exempel på inbyggda system är mobiltelefoner, routrar och GPS.
Journalfil:	Används för att förhindra dataförlust. När ny data ska skrivas till en databasfil säkerhetskopieras de sidorna som ska uppdateras; sedan utförs uppdateringen. Ifall uppdateringsprocessen avbryts av någon anledning kan databasfilen återställas till föregående version med hjälp av journalfilen.
NULL:	Inom datateknik har en variabel eller ett objekt som är NULL inget värde. Till exempel kan en variabel av typen heltal (int) vara noll eller en variabel av typen textsträng (string) vara noll tecken lång, alltså ””, dock är det inte samma sak som NULL. Är en variabel eller ett objekt NULL är den/det odefinierad.
Rotsida:	Sidan i en B-Tree struktur varifrån alla andra sidor i strukturen kan hittas. Se också definitionen av balanserad sökträd.
Schema layer filformat:	Bestämmer hur records sparas och vilka funktioner som stöds. Högre schema layer filformat stödjer fler funktioner.
SQLite index:	Ett index sparar värdena ur en eller flera kolumner i en tabell.
SQLite tabell (table):	En SQLite tabell är en vanlig tabell som innehåller data.
SQLite trigger:	En trigger är kod som automatiskt exekveras vid särskilda händelser på en viss tabell eller view.
SQLite view:	En view är en sparad förfrågan på en tabell. En förfrågan kan till exempel, i en tabell som lagrar personers namn och adress, plocka ut och visa alla rader där personens namn är Per Persson.
Stack utrymme:	Ett minnesutrymme för program där minne för programmen allokeras statisk.
Unix tid:	En representation av datum och tid som antalet sekunder som förflutit sedan midnatt 1 januari 1970.
UTF-8:	En typ av textavkodningssystem. I dessa system representeras varje tecken av någonting annat. I detta fall representeras tecknen av en binär sekvens, det vill säga sekvens av ettor och nollor.
Write-ahead log (WAL) fil:	Har samma syfte som journalfiler fungerar dock lite annorlunda. Istället för att säkerhetskopiera den gamla datan

till en fil och sedan uppdatera databasfilen skrivs den nya datan till en fil och databasfilen uppdateras senare.

Bilaga C, SQLite Data Finder koden

```
'''
Created on 13 apr 2011

@author: Sebastian Zankl
'''


# Takes the number of remaining bytes in the current page of the
# overflow chain and the page number of the next page (or -1 to
# indicate that the current page in the chain is the first in the
# overflow chain) in the overflow chain as arguments.
# This function changes the position in the input file to the
# offset of the new page in the overflow chain and reads the
# number of the next page in the chain.
# Returns the number of the next page in the overflow chain or
# zero if the new page is the last in the chain.
def NextOverflowPage (overflowPage):
    # This checks if the current page in the overflow chain is the
    # first page in the chain. In that case the number of the next
    # page in the chain is stored at the end of the record prefix.
    if (overflowPage == -1):
        overflowPage = int(input.read(4).encode('hex'), 16)

    # Here the page in the overflow chain is switched to the new
    # page and the number of the next page in the chain is read,
    # since it is always stored in the beginning of the overflow
    # page except for the first page. If the number of the next
    # page is zero the current page is the last in the chain.
    input.seek((overflowPage-1) * pageSize)
    overflowPage = int(input.read(4).encode('hex'), 16)

    return overflowPage

# Takes the length of the entry in bytes, the number of remaining
# bytes in the current page of the overflow chain and the page
# number of the next page in the overflow chain as arguments.
# This function reads the contents of an entry. It is only used if
# the contents of the current record are stored in an overflow chain.
# Returns the data contained in the entry (or in some cases a part
# of it), the number of remaining bytes in the current page of
# the overflow chain and the page number of the next page in the
# overflow chain.
def ReadEntry (bytes, overflow, overflowPage):
    # If more bytes need to be read than there are bytes left to
    # read until the end of the current page of the overflow chain
    # is reached then as many bytes as remain on the current page
    # are read and the same amount is subtracted from the number
    # of bytes to read. Also the value of overflow is set to zero.
    # If there are enough bytes left on the current page then the
    # required number of bytes are read and the same amount is
    # subtracted from overflow.
    if (bytes > overflow):
        data = input.read(overflow)
```

```
            bytes -= overflow
            overflow = 0
        else:
            data = input.read(bytes)
            overflow -= bytes

            if (args.mode != "0" and overflowPage == 0 and overflow >= minCharCount):
                freeSpaceStart.append(input.tell())
                freeSpaceSize.append(overflow)

        # If overflow has reached zero and the end of the overflow
        # chain has not been reached yet then the page needs to be
        # switched to the next page in the chain.
        if (overflow == 0 and overflowPage != 0):
            overflowPage = NextOverflowPage(overflowPage)
            overflow = (pageSize-unusedBytes-4)

            # If more bytes need to be read than there are bytes
            # left to read until the end of the current page of
            # the overflow chain is reached then ReadEntry is
            # called again to create a recursive loop that
            # continues until the last page in the chain is
            # reached and all bytes have been read. If there are
            # enough bytes left on the current page then the
            # required number of bytes are read and the same
            # amount is subtracted from overflow. This will also
            # end the recursive loop.
            if (bytes > overflow):
                list = ReadEntry(bytes, overflow, overflowPage)
                data += list[0]
                overflow = list[1]
                overflowPage = list[2]
            else:
                data += input.read(bytes)
                overflow -= bytes

        return [data, overflow, overflowPage]

    def GetValueDeleted (maxBytes):
        # Here the length (in bytes) of the variable length integer
        # is determined.
        list = IntLenDeleted(maxBytes)
        bytes = list[0]
        value = int(list[1], 16)
        maxBytes = list[2]

        if (maxBytes != 0):
            if (bytes > 1):
                value = IntValue(value, bytes)

            return [value, bytes]
        else:
            return [0, maxBytes]

    def IntLenDeleted (maxBytes):
```

```
    counter = 0
    hexStr = ""

    byte = int(input.read(1).encode('hex'), 16)
    maxBytes -= 1

    if (maxBytes > 0):
        # Here the bytes belonging to the variable length integer are
        # counted.
        while (counter <= 8 and byte >= 128):
            counter += 1

            if (byte <= 15):
                hexStr += "0" + hex(byte)[2:]
            else:
                hexStr += hex(byte)[2:]

            byte = int(input.read(1).encode('hex'), 16)
            maxBytes -= 1

            if (maxBytes == 0):
                break

    counter += 1

    # Values below ten will be converted into 1, 2 and so on
    # instead of 01, 02 and so on. That is why a zero is
    # inserted before the value in those cases.
    if (byte <= 15):
        hexStr += "0" + hex(byte)[2:]
    else:
        hexStr += hex(byte)[2:]

    return [counter, hexStr, maxBytes]

# Takes the number of remaining bytes in the current page of the
# overflow chain and the page number of the next page in the
# overflow chain as arguments.
# This function is used to simplify the reading and conversion of
# variable length integers.
# Returns the real value of a variable length integer, its length
# in bytes, the number of remaining bytes in the current page of
# the overflow chain and the page number of the next page in the
# overflow chain.
def GetValueOverflow (overflow, overflowPage):
    # Here the length (in bytes) of the variable length integer
    # is determined.
    list = IntLenOverflow(overflow, overflowPage)
    bytes = list[0]
    value = int(list[1], 16)
    overflow = list[2]
    overflowPage = list[3]

    # If the variable length integer is longer than one byte it
    # must be converted into its real value else no conversion
```

```
    # will be needed.
    if (bytes > 1):
        value = IntValue(value, bytes)

    return [value, bytes, overflow, overflowPage]

# Takes the number of remaining bytes in the current page of the
# overflow chain and the page number of the next page in the
# overflow chain as arguments.
# This function is used to determine how many bytes a certain
# record header entry consumes and also to return the entry value
# as a hex-string.
# Returns the number of bytes the integer consumes, the hex-value
# of the integer as a string, the number of remaining bytes in the
# current page of the overflow chain and the page number of the
# next page in the overflow chain.
def IntLenOverflow (overflow, overflowPage):
    counter = 0
    hexStr = ""

    byte = int(input.read(1).encode('hex'), 16)
    overflow -= 1

    # Checks if the end of the current page in the overflow chain
    # has been reached and, if yes, switches to the next overflow
    # page in the chain and resets the value of overflow.
    if (overflow == 0 and overflowPage != 0):
        overflowPage = NextOverflowPage(overflowPage)
        overflow = (pageSize-unusedBytes-4)

    # Here the bytes belonging to the variable length integer are
    # counted.
    while (counter <= 8 and byte >= 128):
        counter += 1

        if (byte <= 15):
            hexStr += "0" + hex(byte)[2:]
        else:
            hexStr += hex(byte)[2:]

        byte = int(input.read(1).encode('hex'), 16)
        overflow -= 1

        # Same as before.
        if (overflow == 0 and overflowPage != 0):
            overflowPage = NextOverflowPage(overflowPage)
            overflow = (pageSize-unusedBytes-4)

    counter += 1

    # Values below ten will be converted into 1, 2 and so on
    # instead of 01, 02 and so on. That is why a zero is
    # inserted before the value in those cases.
    if (byte <= 15):
        hexStr += "0" + hex(byte)[2:]
```

```
    else:
        hexStr += hex(byte)[2:]

    return [counter, hexStr, overflow, overflowPage]

# Takes a boolean value as argument.
# This function is used to simplify the reading and conversion of
# variable length integers.
# Returns either a list containing the real value of a variable
# length integer and its length in bytes or just the real value
# of the variable length integer.
def GetValue (retBytes):
    # Here the length (in bytes) of the variable length integer
    # is determined.
    bytes = IntLen()

    # If the variable length integer is longer than one byte it
    # must be converted into its real value else it can simply
    # be read.
    if (bytes > 1):
        value = IntValue(int(input.read(bytes).encode('hex'), 16), bytes)
    else:
        value = int(input.read(bytes).encode('hex'), 16)

    # In most cases only the real value of the variable length
    # integer needs to be returned but in some cases even its
    # length needs to be returned.
    if (retBytes == True):
        return [value, bytes]
    else:
        return value

# Takes no arguments.
# This function is used to determine how many bytes a certain
# record header entry consumes (integers can have different
# length in bytes in SQLite depending on the size of the integer).
# Returns the number of bytes the integer consumes.
def IntLen ():
    counter = 0
    byte = int(input.read(1).encode('hex'), 16)

    # The first bit in each byte indicates if that byte is the last.
    # A zero indicates that this is the last byte belonging to the
    # integer. If the bytes value is equal or greater than 128 the
    # first bit is one. The integer cannot be longer than 9 bytes.
    while (counter <= 8 and byte >= 128):
        counter += 1
        byte = int(input.read(1).encode('hex'), 16)

    # The counter must always be increased by one after the
    # if-statement since the last byte belonging to an integer
    # never is counted in the if-statement.
    counter += 1

    # The position in the file needs to be reset by the amount of
```

```
    # the counter.
    input.seek(input.tell()-counter)

    return counter

def StringToFloat(string):
    s = "".join(string)
    f = struct.unpack(">d", s)[0]

    return f

# Takes a string containing one byte in binary format as argument.
# This function converts a string containing one byte in binary
# format to the corresponding integer value.
# Returns the corresponding integer.
def BinStrToInt (binStr):
    return (int(binStr[0]) + int(binStr[1]) * 2 + int(binStr[2]) * 4 + int(binStr[3]) * 8 + int(binStr[4]) *
16 + int(binStr[5]) * 32 + int(binStr[6]) * 64 + int(binStr[7]) * 128)

# Takes two integers as arguments. The first one is the variable
# length integer and the second one is the amount of bytes the
# variable length integer consumed.
# This function converts an SQLite variable length integer to the
# original integer.
# Returns the integer that the SQLite variable length integer
# was representing. See http://www.sqlite.org/fileformat.html
# for a description of the SQLite variable length integer format.
def IntValue (integer, bytes):
    i = 1
    j = 1
    temp = ''
    byteList = []
    byte = ''
    intStr = ''

    # The ninth byte can be read as two normal hex values.
    if (bytes == 9):
        temp = str(hex(integer)[2:-1]) # -1 from [2:-1] has to be removed if python 3.0 or higher is used
        temp = temp[16] + temp[17]
        bytes -= 1
        integer = integer >> 8

    # This checks if the last bit in the integer is a zero
    # or a one and appends the value of the bit to a string.
    # Then all bits in the integer are shifted one bit to the
    # right and the last bit is checked again. Every eighth bit
    # is skipped and after eight bits have been written to the
    # string, it is saved to a list and reset.
    while (i < (bytes * 8)):
        if ((i % 8) != 0):
            if ((integer % 2) == 0):
                byte += str(0)
            else:
                byte += str(1)
```

```
            if ((j % 8) == 0):
                byteList.insert(0, byte)
                byte = ''

            j += 1

        i += 1
        integer = integer >> 1

    i = 0

    # If the number of bytes is less than 8 the number of bits
    # will not be a multiple of 8 and zeros need to be added to
    # the beginning of the first byte.
    if (bytes != 8):
        while (i < bytes):
            byte += "0"
            i += 1
        byteList.insert(0, byte)

    # Converts the binary string for each byte into the
    # corresponding integer and then the corresponding hex value.
    for byte in byteList:
        value = BinStrToInt(byte)

        # Values below ten will be converted into 1, 2 and so on
        # instead of 01, 02 and so on. That is why a zero is
        # inserted before the value in those cases.
        if (value <= 15):
            intStr += "0" + hex(value)[2:]
        else:
            intStr += hex(value)[2:]

    # If the integer was nine bytes long the last byte needs to
    # be appended and if the four first bits combined have the
    # value "f" then the integer needs to be converted into a
    # negative value.
    if (temp != ''):
        intStr += temp
        if (intStr[0] == "f"):
            temp = long("FFFFFFFFFFFFFFFF", 16) - long(intStr, 16)
            return  (temp - (temp * 2 + 1))

    return long(intStr, 16)

# Takes the record prefix length in bytes or if there
# is none -1 as argument.
# This function reads a certain record in the database.
# Returns a list with the number of record entries, type and
# size of the entries and the data contained in the entries.
def ReadRecord (overflow):
    i = 0
    dataTypeSize = []
    data = []
```

```
# If overflow is greater than -1 the record is spread over
# several pages and overflow represents the length of the
# record prefix in bytes stored on the current page.
if (overflow == -1):
    # Here the record header size (in bytes) is read.
    list =  GetValue(True)
    recordHeaderSize = list[0]
    bytes = list[1]

    # The values in the record header are variable length
    # integers which means that the size of each header entry
    # needs to be determined and the value needs to be converted.
    while (i < (recordHeaderSize-bytes)):
        list = GetValue(True)
        dataTypeSize.append(list[0])
        i += list[1]

    # Depending on the type and size of the record a
    # different amount of bytes is read.
    for entry in dataTypeSize:
        if (entry == 1):
            data.append(int(input.read(1).encode('hex'), 16))
        elif (entry == 2):
            data.append(int(input.read(2).encode('hex'), 16))
        elif (entry == 3):
            data.append(int(input.read(3).encode('hex'), 16))
        elif (entry == 4):
            data.append(int(input.read(4).encode('hex'), 16))
        elif (entry == 5):
            data.append(int(input.read(6).encode('hex'), 16))
        elif (entry == 6):
            data.append(int(input.read(8).encode('hex'), 16))
        elif (entry == 7):
            data.append(StringToFloat(input.read(8)))
        elif (entry >= 12 and (entry % 2) == 0):
            data.append(input.read((entry - 12) / 2))
        elif (entry > 12 and (entry % 2) == 1):
            data.append(input.read((entry - 13) / 2))
else:
    # The variable overflowPage is used to determine which
    # page the next one is in the overflow page chain. It
    # is set to -1 so that the function NextOverflowPage
    # knows that the current page is the first page in the
    # chain. See comment in NextOverflowPage why this matters.
    overflowPage = -1
    list = GetValueOverflow(overflow, overflowPage)
    recordHeaderSize = list[0]
    bytes = list[1]
    overflow = list[2]
    overflowPage = list[3]

    # A special version of the GetValue function is used
    # which makes sure that the page is switched in case the
    # end of the record prefix or the end of the overflow
    # page is reached.
```

```
while (i < (recordHeaderSize-bytes)):
    list = GetValueOverflow(overflow, overflowPage)
    dataTypeSize.append(list[0])
    i += list[1]
    overflow = list[2]
    overflowPage = list[3]

# The ReadEntry function is used to read the data from
# the record. This function, like the GetValueOverflow
# function, makes sure that the page is switched in case
# the end of the record prefix or the end of the overflow
# page is reached.
for entry in dataTypeSize:
    if (entry == 1):
        list = ReadEntry(1, overflow, overflowPage)
        data.append(int(list[0].encode('hex'), 16))
        overflow = list[1]
        overflowPage = list[2]
    elif (entry == 2):
        list = ReadEntry(2, overflow, overflowPage)
        data.append(int(list[0].encode('hex'), 16))
        overflow = list[1]
        overflowPage = list[2]
    elif (entry == 3):
        list = ReadEntry(3, overflow, overflowPage)
        data.append(int(list[0].encode('hex'), 16))
        overflow = list[1]
        overflowPage = list[2]
    elif (entry == 4):
        list = ReadEntry(4, overflow, overflowPage)
        data.append(int(list[0].encode('hex'), 16))
        overflow = list[1]
        overflowPage = list[2]
    elif (entry == 5):
        list = ReadEntry(6, overflow, overflowPage)
        data.append(int(list[0].encode('hex'), 16))
        overflow = list[1]
        overflowPage = list[2]
    elif (entry == 6):
        list = ReadEntry(8, overflow, overflowPage)
        data.append(int(list[0].encode('hex'), 16))
        overflow = list[1]
        overflowPage = list[2]
    elif (entry == 7):
        list = ReadEntry(8, overflow, overflowPage)
        data.append(StringToFloat(list[0]))
        overflow = list[1]
        overflowPage = list[2]
    elif (entry >= 12 and (entry % 2) == 0):
        list = ReadEntry(((entry - 12) / 2), overflow, overflowPage)
        data.append(list[0])
        overflow = list[1]
        overflowPage = list[2]
    elif (entry > 12 and (entry % 2) == 1):
        list = ReadEntry(((entry - 13) / 2), overflow, overflowPage)
```

```
            data.append(list[0])
            overflow = list[1]
            overflowPage = list[2]

    record = [dataTypeSize, data]

    return record

# Takes the size of the record and its type (either table or
# index) as arguments.
# This function calculates the size of the record prefix (in
# bytes) that has been stored on the first page of an overflow
# chain. The formulas are directly taken from the SQLite
# documentation at http://www.sqlite.org/fileformat.html.
# Returns the length of the record prefix in bytes.
def CalcOverflow (recordSize, type):
    # This calculates the number of bytes that can be used on
    # any database page (should always be the same as the
    # page size since the value at offset twenty in the
    # database header, the value saved to the variable
    # unusedBytes, in a well-formed database always contains
    # the value zero).
    usableSize = pageSize - unusedBytes

    # The formulas to calculate minLocal and maxLocal are
    # slightly different for pages associated with tables
    # then for pages associated with indices. (minFractionIndex
    # and minFractionTable should always be 32 since the values
    # at offsets 22 and 23, the values saved to those variables,
    # in a well-formed database always contain the value 32 and
    # maxFractionIndex should always be 64 since the value at
    # offset 21, the value saved to this variable, in a
    # well-formed database always contains the value 64)
    if (type == "table"):
        minLocal = (usableSize - 12) * minFractionTable / 255 - 23
        maxLocal = usableSize - 35
    else:
        minLocal = (usableSize - 12) * minFractionIndex / 255 - 23
        maxLocal = (usableSize -12) * maxFractionIndex / 255 - 23

    localSize = minLocal + (recordSize - minLocal) % (usableSize - 4)

    if (localSize > maxLocal):
        localSize = minLocal

    return localSize

# Takes the start offset of a page as argument.
# This function reads the contents of a page.
# Returns a list containing the data found in the current page,
# parts of the data contained in a B-Tree structure or the
# entire data found in the B-Tree structure (this depends on
# how deep in the B-Tree structure the current page is).
def ReadPage (offset):
    input.seek(offset)
```

```
    # The variable currentOffset is used to determine at what
    # offset, relative to the start of the page, the cell
    # offset array is located in the page. It is set to eight
    # since the header of leaf pages is eight bytes long.
    currentOffset = 8
    i = 0
    values = []
    rightmostChild = 0

    # Here the page header information is read.
    pageType = int(input.read(1).encode('hex'), 16)
    firstFreeSpace = int(input.read(2).encode('hex'), 16)            # currently not used in the program,
comment this line
    #input.seek(offset+2)                                          # and uncomment this one if you want to save
memory
    numOfCells = int(input.read(2).encode('hex'), 16)
    cellContentArea = int(input.read(2).encode('hex'), 16)

    if (cellContentArea == 0):
        cellContentArea = 65536

    numOfFragments = int(input.read(1).encode('hex'), 16)            # currently not used in the program,
comment this line
    # input.seek(offset+1)                                          # and uncomment this one if you want to save
memory

    # If offset is 100 it means the current page is the root
    # page of the schema table. This also means currentOffset
    # needs to be increased by 100 and offset set to zero since
    # the values of the offsets of all cells in the page are
    # relative to the start of the page.
    if (offset == 100):
        offset = 0
        currentOffset += 100

    # If the page is of type internal leaf node then four
    # additional bytes need to be read and currentOffset needs
    # to be increased by four.
    if (pageType == 2 or pageType == 5):
        rightmostChild = int(input.read(4).encode('hex'), 16)
        currentOffset += 4

    if (args.mode != "0" and ((cellContentArea-currentOffset)-(numOfCells*2)) >= minCharCount):
        freeSpaceStart.append(input.tell()+(numOfCells*2))
        freeSpaceSize.append((cellContentArea-currentOffset)-(numOfCells*2))

    # This will loop as many times as there are left (as in the
    # direction) child pages.
    while (i < numOfCells):
        # This finds the offset, relative to the start of the page,
        # of the next cell in the page.
        input.seek(offset+currentOffset)
        cellOffset = int(input.read(2).encode('hex'), 16)
```

```
        if (pageType == 2):
            # The page number of the next child page and the size of
            # the record in the current cell are read.
            input.seek(offset + cellOffset)
            childPage = int(input.read(4).encode('hex'), 16)
            recordSize = GetValue(False)

            # This determines if the current record is small enough to
            # entirely fit on the current page. If not, the function
            # CalcOverflow is called to determine the size of the
            # record prefix stored on the first page (the current page)
            # of the overflow chain.
            if (recordSize > (((pageSize-unusedBytes)-12)*maxFractionIndex/255-23)):
                overflow = CalcOverflow(recordSize, "index")
            else:
                overflow = -1

            # Here first the data contained in the child page is read,
            # saved to a list and then each entry in this list is
            # appended to the variable values. Then the data
            # contained in the current record is read and appended to
            # the variable values.
            list = ReadPage(((childPage-1) * pageSize))

            for entry in list:
                values.append(entry)

            values.append(ReadRecord(overflow))

            # currentOffset needs to be increased by two since the
            # values stored in the cell offset array are stored as
            # 2-byte big-endian integers.
            currentOffset += 2
        elif (pageType == 10):
            # The size of the record in the current cell is read.
            input.seek(offset + cellOffset)
            recordSize = GetValue(False)

            # Same as above.
            if (recordSize > (((pageSize-unusedBytes)-12)*maxFractionIndex/255-23)):
                overflow = CalcOverflow(recordSize, "index")
            else:
                overflow = -1

            # Here the data contained in the current record is read
            # and appended to the variable values.
            values.append(ReadRecord(overflow))

            # Same as above.
            currentOffset += 2
        elif (pageType == 5):
            # The key value is read.
            input.seek(offset+cellOffset+4)                    # if you comment the next line you also need
to comment this one
```

```
            keyValue = GetValue(False)                    # this is actually not used for anything in the
program, comment this line if you want to save memory

            # Here the data contained in the child page is read and
            # appended to the variable values.
            input.seek(offset+cellOffset)

            list = ReadPage((((int(input.read(4).encode('hex'), 16)-1) * pageSize)))

            for entry in list:
                values.append(entry)

            # Same as above.
            currentOffset += 2
        else:
            input.seek(offset+cellOffset)

            recordSize = GetValue(False)
            keyValue = GetValue(False)

            # Same as above.
            if (recordSize > ((pageSize-unusedBytes)-35)):
                overflow = CalcOverflow(recordSize, "table")
            else:
                overflow = -1

            # Here the data contained in the current record is read and
            # appended to the variable values together with its key value.
            record = ReadRecord(overflow)
            record.insert(0, keyValue)
            values.append(record)

            # Same as above.
            currentOffset += 2

        i += 1

    # If there is a rightmost child page this will read its contents.
    if (rightmostChild != 0 and (pageType == 2 or pageType == 5)):
        list = ReadPage(((rightmostChild-1) * pageSize))

        for entry in list:
            values.append(entry)

    return values

def PrintLine(output, length, columnWidth):
    i = 0
    line = "----------------------------------------------------------------------------------------------------"
    string = ""

    while (i < length):
        if (i < (length-1)):
            string += "%s-|-" % line[:columnWidth[i]]
        else:
```

```
            string += "%s-\n" % line[:columnWidth[i]]

        i += 1

    return string

def PrintTable(item, key, type):
    i = 0
    loops = 0
    blobCount = 1
    moreData = 0

    try:
        if (type == "schema"):
            tableName = "schema_table"
            columnNames = ["RowID", "Schema item type", "Schema item name", "Associated table
name", "Root page number", "SQL statement"]
            columnWidth = [19, 16, 46, 30, 19, 100]
            columnData = [0, 0, 0, 0, 0, 0]

            string = tableName + "\n"
            string += PrintLine(output, 1, [len(tableName)])
            string += "\n"
        elif (type == "table"):
            columnNames = ["RowID"]
            columnWidth = [19]
            columnData = [0]
            namesTypes = []
            index = 0
            j = 0
            k = 0
            zero = 0
            sizeCounter = 0

            sqlStatement = schemaTable[key][2][4]
            sqlStatement = sqlStatement.split("(", 1)
            tableName = schemaTable[key][2][1]

            string = tableName + "\n"
            string += PrintLine(output, 1, [len(tableName)])
            string += "\n"

            if (item != -1):
                if (tableName == "sqlite_sequence"):
                    columnNames.append("name")
                    columnNames.append("seq")
                    columnWidth.append(30)
                    columnWidth.append(19)
                else:
                    sqlStatement = sqlStatement[1].split(",")

                    for sequence in sqlStatement:
                        if (sequence.__contains__("primary key") or sequence.__contains__("PRIMARY
KEY")):
                            sqlStatement.remove(sequence)
```

```
                    break
                elif (sequence.__contains__("id integer unique") or sequence.__contains__("ID
INTEGER UNIQUE")):
                    sqlStatement.remove(sequence)
                    break
                elif (sequence.__contains__("autoincrement") or
sequence.__contains__("AUTOINCREMENT")):
                    sqlStatement.remove(sequence)
                    break

                index += 1

            for entry in item:
                if (index != len(sqlStatement)):
                    entry[1].__delitem__(index)

            sqlStatement[len(sqlStatement)-1] = sqlStatement[len(sqlStatement)-1].strip(")")

            for word in sqlStatement:
                list = word.split()

                if (len(list) > 1):
                    namesTypes.append(list[0])
                    namesTypes.append(list[1])
                else:
                    namesTypes.append(list[0])

                    if (item[0][1][i] == 1):
                        namesTypes.append("integer")
                    elif (item[0][1][i] == 2):
                        namesTypes.append("integer")
                    elif (item[0][1][i] == 3):
                        namesTypes.append("integer")
                    elif (item[0][1][i] == 4):
                        namesTypes.append("integer")
                    elif (item[0][1][i] == 5):
                        namesTypes.append("integer")
                    elif (item[0][1][i] == 6):
                        namesTypes.append("integer")
                    elif (item[0][1][i] == 7):
                        namesTypes.append("float")
                    elif (item[0][1][i] >= 12 and (item[0][1][i] % 2) == 0):
                        namesTypes.append("blob")
                    elif (item[0][1][i] >= 12 and (item[0][1][i] % 2) == 1):
                        namesTypes.append("text")

                    i += 1

            i = 0

            while (i < len(namesTypes)):
                columnNames.append(namesTypes[i])
                columnData.append(0)

                if (namesTypes[i+1].lower() == "text" or namesTypes[i+1].lower() == "string"):
```

```
                    if (item[0][1][i/2] == 0):
                        columnWidth.append(20)
                    else:
                        while (j < 10 and j < len(item)):
                            while (k < (i/2) and k < len(item[0][1])):
                                if (item[j][1][k] == 0):
                                    zero += 1
                                k += 1

                            k = 0

                            if (len(item[j][2][i/2-zero]) > 50):
                                sizeCounter += 1

                            j += 1
                            zero = 0

                        j = 0

                        if (sizeCounter >= 4):
                            columnWidth.append(100)
                        else:
                            columnWidth.append(50)

                        sizeCounter = 0
                elif (namesTypes[i+1].lower() == "integer" or namesTypes[i+1].lower() == "float" or
namesTypes[i+1].lower() == "long"):
                    if (item[0][1][i/2] == 0):
                        columnWidth.append(20)
                    else:
                        columnWidth.append(22)
                elif (namesTypes[i+1].lower() == "blob"):
                    if (item[0][1][i/2] == 0):
                        columnWidth.append(20)
                    else:
                        columnWidth.append(30)

                i += 2

            i = 0
    elif (type == "index"):
        columnNames = ["RowID"]
        columnWidth = [19]
        columnData = [0]
        names = []
        index = 0
        j = 0
        zero = 0
        sizeCounter = 0

        indexName = schemaTable[key][2][1]

        if (indexName[:17] != "sqlite_autoindex_"):
            sqlStatement = schemaTable[key][2][4]
            sqlStatement = sqlStatement.split("(", 1)
```

```
        if (item != -2):
            sqlStatement = sqlStatement[1].split(",")
            sqlStatement[len(sqlStatement)-1] = sqlStatement[len(sqlStatement)-1].strip(")")

            for word in sqlStatement:
                list = word.split()

                for entry in list:
                    names.append(entry)

            onTable = schemaTable[key][2][2]

            for entry in tableColumns:
                if (entry.__contains__(onTable)):
                    break

                index += 1

            i = index

            for name in names:
                index = tableColumns[i][1].index(name)
                columnWidth.append(tableColumns[i][1][index+1])
                columnNames.append(name)
                columnData.append(0)

            i = 0

    string = indexName + "\n"
    string += PrintLine(output, 1, [len(indexName)])
    string += "\n"

if (item > -1):
    while (i < len(columnNames)):
        if (i < (len(columnNames)-1)):
            string += str.center(columnNames[i], columnWidth[i]) + " | "
        else:
            string += str.center(columnNames[i], columnWidth[i]) + " \n"

        i += 1

    i = 0
    j = 0

    string += PrintLine(output, len(columnNames), columnWidth)

    if (type == "index"):
        for entry in item:
            entry[0].__delitem__(len(entry[0])-1)
            entry.insert(0, entry[1][len(entry[1])-1])
            entry[2].__delitem__(len(entry[2])-1)

    for entry in item:
        string += "%*d | " % (columnWidth[0], entry[0])
```

```
            for dataType in entry[1]:
                if (dataType > 12 and (dataType % 2) == 1):
                    if (len(entry[2][i]) < columnWidth[j+1]):
                        if (j+1 < (len(entry[1]))):
                            string += "%-*s | " % (columnWidth[j+1], entry[2][i])
                        else:
                            string += "%-*s " % (columnWidth[j+1], entry[2][i])
                    else:
                        if (j+1 < (len(entry[1]))):
                            string += "%-*s | " % (columnWidth[j+1], entry[2][i][:columnWidth[j+1]])
                        else:
                            string += "%-*s " % (columnWidth[j+1], entry[2][i][:columnWidth[j+1]])

                        columnData[j+1] = entry[2][i][columnWidth[j+1]:]
                elif (dataType > 0 and dataType < 7):
                    if (j+1 < (len(entry[1]))):
                        string += "%*d | " % (columnWidth[j+1], entry[2][i])
                    else:
                        string += "%*d " % (columnWidth[j+1], entry[2][i])
                elif (entry[1][i] == 7):
                    if (j+1 < (len(entry[1]))):
                        string += "%*.5f | " % ((columnWidth[j+1]-5), entry[2][i])
                    else:
                        string += "%*.5f " % ((columnWidth[j+1]-5), entry[2][i])
                elif (dataType > 12 and (dataType % 2) == 0):
                    try:
                        blob = open(tableName + "-blob" + str(blobCount) + ".bin", 'wb')
                        blob.write(entry[2][i])
                        blobName = (tableName + "-blob" + str(blobCount) + ".bin")

                        if (len(entry[2][i]) < columnWidth[j+1]):
                            if (j+1 < len(entry[1])):
                                string += "%-*s | " % (columnWidth[j+1], blobName)
                            else:
                                string += "%-*s " % (columnWidth[j+1], blobName)
                        else:
                            if (j+1 < len(entry[1])):
                                string += "%-*s | " % (columnWidth[j+1], blobName[:columnWidth[j+1]])
                            else:
                                string += "%-*s " % (columnWidth[j+1], blobName[:columnWidth[j+1]])

                            columnData[j+1] = blobName[columnWidth[j+1]:]
                    except:
                        print "Kunde inte skriva blobdata f\xf6r blob" + str(blobCount) + " i " + tableName +
" till en fil!"
                        blobName = ("blob" + str(blobCount))

                        if (len(entry[2][i]) < columnWidth[j+1]):
                            if (j+1 < (len(entry[1]))):
                                string += "%-*s | " % (columnWidth[j+1], blobName)
                            else:
                                string += "%-*s " % (columnWidth[j+1], blobName)
                        else:
                            if (j+1 < (len(entry[1]))):
```

```
                string += "%-*s | " % (columnWidth[j+1], blobName[:columnWidth[j+1]])
            else:
                string += "%-*s " % (columnWidth[j+1], blobName[:columnWidth[j+1]])

            columnData[j+1] = blobName[columnWidth[j+1]:]
        finally:
            blob.close()
            blobCount += 1
    else:
        if (j+1 < (len(entry[1]))):
            string += "%*s | " % (columnWidth[j+1], "")
        else:
            string += "%*s " % (columnWidth[j+1], "")

    if (dataType != 0):
        i += 1

    j += 1

i = 0
j = 0
zero = 0

for column in columnData:
    if (column != 0):
        moreData += 1

while (moreData > 0):
    string += "\n"

    for data in columnData:
        if (data != 0):
            if (len(data) < columnWidth[i]):
                if (i < (len(entry[2])-1)):
                    string += "%-*s | " % (columnWidth[i], data)
                else:
                    string += "%-*s " % (columnWidth[i], data)

                columnData[i] = 0
                moreData -= 1
            else:
                if (i < (len(entry[2])-1)):
                    string += "%-*s | " % (columnWidth[i], data[:columnWidth[i]])
                else:
                    string += "%-*s " % (columnWidth[i], data[:columnWidth[i]])

                columnData[i] = data[columnWidth[i]:]
        else:
            if (i < (len(entry[2]))):
                string += "%*s | " % (columnWidth[i], "")
            else:
                string += "%*s " % (columnWidth[i], "")

        i += 1
```

```
                i = 0

            if (loops < (len(item)-1)):
                string += "\n"
                string += PrintLine(output, len(columnNames), columnWidth)

            loops += 1
        elif (item == -1):
            string += "Tabellen \xe4r tom."
        elif (item == -2):
            string += "Indexet \xe4r tom."

        string += "\n\n\n"

        output.write(string)

        if (type == "table"):
            i = 1

            if (len(columnNames) < 3 or tableName == "sqlite_sequence"):
                j = 0
            else:
                j = 1

            while (i <= len(namesTypes)):
                namesTypes[i] = columnWidth[j]
                i += 2
                j += 1

            return [tableName, namesTypes]
    except:
        if (type == "table"):
            tableName = schemaTable[key][2][1]

            output.write(tableName + "\n")
            output.write(PrintLine(output, 1, [len(tableName)]))
            output.write("\n")

            if (item != -1):
                for entry in item:
                    output.write(str(entry[0]) + "\t" + str(entry[2]) + "\n")
            else:
                output.write("Tabellen \xe4r tom.")

            output.write("\n\n\n")
        elif (type == "index"):
            indexName = schemaTable[key][2][1]

            output.write(indexName + "\n")
            output.write(PrintLine(output, 1, [len(indexName)]))
            output.write("\n")

            if (item != -2):
                for entry in item:
                    output.write(str(entry[1]) + "\n")
```

```
            else:
                output.write("Indexet \xe4r tom.")

            output.write("\n\n\n")

def FindIndexDataTypes(statement, tableTypes, tableStatement):
    index = 0
    i = 0
    j = 0
    datatypes = []
    columnNames = []

    list = statement.split("(")
    list = list[1].split(",")
    list[len(list)-1] = list[len(list)-1].strip(")")

    for name in list:
        columnNames.append(name.strip())

    list = tableStatement.split("(")
    list2 = list[0].split()

    if (list2[2] == "sqlite_sequence"):
        if (len(columnNames) == 1):
            return ["text", "integer"]
        else:
            return ["text", "text", "integer"]
    else:
        list = list[1].split(",")

        if (tableTypes.__contains__("null")):
            index = tableTypes.index("null")
            list.__delitem__(index)

        list2 = []

        for word in list:
            list2.append(word.split())

        while (i < len(list2)):
            try:
                if (list2[i][0] == columnNames[j]):
                    datatypes.append(list2[i][1])
                    j += 1

                    if (j == len(columnNames)):
                        break
            except:
                datatypes.append("")

            i += 1

        datatypes.append("integer")

        return datatypes
```

```
def FindTableDataTypes(statement):
    index = 0
    i = 0
    datatypes = []

    list = statement.split("(", 1)
    list2 = list[0].split()

    if (list2[2] == "sqlite_sequence"):
        return ["text", "text"]
    else:
        list = list[1].split(",")

        while (index < len(list)):
            if (list[index].__contains__("primary key") or list[index].__contains__("PRIMARY KEY")):
                list.__delitem__(index)
                break
            elif (list[index].__contains__("id integer unique") or list[index].__contains__("ID INTEGER
UNIQUE")):
                list.__delitem__(index)
                break
            elif (list[index].__contains__("autoincrement") or
list[index].__contains__("AUTOINCREMENT")):
                list.__delitem__(index)
                break

            index += 1

        list[len(list)-1] = list[len(list)-1].strip(")")
        temp = list

        while (i < len(temp)):
            list = temp[i].split()
            if (len(list) > 1):
                datatypes.append(list[1].lower())
            i += 1

        if (index < len(datatypes)):
            datatypes.insert(index, "null")

        return datatypes

def ReadDeletedEntry(bytes, maxBytes):
    if (bytes > maxBytes):
        data = input.read(maxBytes)
        maxBytes = 0
    else:
        data = input.read(bytes)
        maxBytes -= bytes

    return [data, maxBytes]

def ReadRecordData(values, start, maxBytes):
    i = 0
```

```
dataTypeSize = []
data = []

while (start < (len(values))):
    dataTypeSize.append(values[start])

    start += 1

while (i < len(dataTypeSize)):
    if (dataTypeSize[i] == 1):
        list = ReadDeletedEntry(1, maxBytes)
        data.append(int(list[0].encode('hex'), 16))
        maxBytes = list[1]

        if (maxBytes == 0):
            break
    elif (dataTypeSize[i] == 2):
        list = ReadDeletedEntry(2, maxBytes)
        data.append(int(list[0].encode('hex'), 16))
        maxBytes = list[1]

        if (maxBytes == 0):
            break
    elif (dataTypeSize[i] == 3):
        list = ReadDeletedEntry(3, maxBytes)
        data.append(int(list[0].encode('hex'), 16))
        maxBytes = list[1]

        if (maxBytes == 0):
            break
    elif (dataTypeSize[i] == 4):
        list = ReadDeletedEntry(4, maxBytes)
        data.append(int(list[0].encode('hex'), 16))
        maxBytes = list[1]

        if (maxBytes == 0):
            break
    elif (dataTypeSize[i] == 5):
        list = ReadDeletedEntry(6, maxBytes)
        data.append(int(list[0].encode('hex'), 16))
        maxBytes = list[1]

        if (maxBytes == 0):
            break
    elif (dataTypeSize[i] == 6):
        list = ReadDeletedEntry(8, maxBytes)
        data.append(int(list[0].encode('hex'), 16))
        maxBytes = list[1]

        if (maxBytes == 0):
            break
    elif (dataTypeSize[i] == 7):
        list = ReadDeletedEntry(8, maxBytes)
        data.append(StringToFloat(list[0]))
        maxBytes = list[1]
```

```
            if (maxBytes == 0):
                break
        elif (dataTypeSize[i] >= 12 and (dataTypeSize[i] % 2) == 0):
            list = ReadDeletedEntry(((dataTypeSize[i]-12) / 2), maxBytes)
            data.append(list[0])
            maxBytes = list[1]

            if (maxBytes == 0):
                break
        elif (dataTypeSize[i] >= 12 and (dataTypeSize[i] % 2) == 1):
            list = ReadDeletedEntry(((dataTypeSize[i]-13) / 2), maxBytes)
            data.append(list[0])
            maxBytes = list[1]

            if (maxBytes == 0):
                break

        i += 1

    if (maxBytes == 0):
        i += 1

        while (i < len(dataTypeSize)):
            if (dataTypeSize[i] >= 12 or (dataTypeSize[i] > 0 and dataTypeSize[i] < 8)):
                data.append("")

            i += 1

    return [maxBytes, dataTypeSize, data]

def CalcRecordSize(values, start):
    recordSize = 0

    while (start < len(values)):
        if (values[start] == 1):
            recordSize += 1
        elif (values[start] == 2):
            recordSize += 2
        elif (values[start] == 3):
            recordSize += 3
        elif (values[start] == 4):
            recordSize += 4
        elif (values[start] == 5):
            recordSize += 6
        elif (values[start] == 6 or values[start] == 7):
            recordSize += 8
        elif (values[start] >= 12 and (values[start] % 2) == 0):
            recordSize += (values[start]-12) / 2
        elif (values[start] >= 12 and (values[start] % 2) == 1):
            recordSize += (values[start]-13) / 2

        start += 1

    return recordSize
```

```
def CheckDataTypes(values):
    i = 1
    count = 0
    types = []

    while (i < len(values)):
        if (values[i] == 0 and args.wildcardNull == True):
            types.append("*")
        elif (values[i] == 0 and args.wildcardNull == False):
            types.append("null")
        elif (values[i] == 1):
            types.append("integer")
        elif (values[i] == 2):
            types.append("integer")
        elif (values[i] == 3):
            types.append("integer")
        elif (values[i] == 4):
            types.append("integer")
        elif (values[i] == 5):
            types.append("integer")
        elif (values[i] == 6):
            types.append("integer")
        elif (values[i] == 7):
            types.append("float")
        elif (values[i] >= 12 and (values[i] % 2) == 0):
            types.append("blob")
        elif (values[i] >= 12 and (values[i] % 2) == 1):
            types.append("text")

        i += 1

    i = 0

    for item in itemDataTypes:
        for type in item[1]:
            if (i >= len(types)):
                count = 0
                break
            elif (types[i] == type or types[i] == "*"):
                count += 1
            else:
                count = 0
                break

            i += 1

        if (count == len(item[1]) and count == len(types)):
            return True
        else:
            i = 0
            count = 0

    return False
```

```
def IsPointerMapPage(value):
    for page in pointerMapPages:
        if (page == value):
            return True

    return False

def DeletedRecordSearch(mode):
    i = 0
    j = 0
    hit = False
    values = []
    records = []

    while (i < len(freeSpaceStart)):
        if (mode == 1):
            while (j < freeSpaceSize[i]):
                input.seek(freeSpaceStart[i]+j)

                list = GetValueDeleted((freeSpaceSize[i]-j))
                values.append(list[0])
                bytes = list[1]

                if (values[0] >= 2 and values[0] < (dbSize*pageSize) and bytes < freeSpaceSize[i] and
list[1] != 0):
                    list = GetValueDeleted((freeSpaceSize[i]-bytes-j))
                    values.append(list[0])
                    bytes += list[1]

                    if (values[1] > 0 and bytes < freeSpaceSize[i] and list[1] != 0):
                        list = GetValueDeleted((freeSpaceSize[i]-bytes-j))
                        values.append(list[0])
                        bytes += list[1]
                        headerSizeBytes = list[1]

                        if (values[2] >= 2 and values[2] <= values[0] and values[2] <
(headerSizeBytes+(itemMaxColumnCount*8)) and bytes < freeSpaceSize[i] and headerSizeBytes !=
0):
                            k = 0

                            while (k < (values[2]-headerSizeBytes)):
                                list = GetValueDeleted((freeSpaceSize[i]-bytes-j))

                                if (list[1] == 0):
                                    break

                                values.append(list[0])
                                k += list[1]

                            bytes += k

                            for value in values[3:]:
                                if (value == 10 or value == 11):
                                    values[2] = 0
                                    break
```

```
                    if (values[2] == (k+headerSizeBytes) and values[0] ==
(values[2]+CalcRecordSize(values, 3)) and bytes <= freeSpaceSize[i]) and (len(values)-3) <=
itemMaxColumnCount and list[1] != 0:
                        list = ReadRecordData(values, 3, (freeSpaceSize[i]-bytes-j))
                        readBytes = (freeSpaceSize[i]-bytes-j) - list[0]
                        records.append([values[1], [list[1], list[2]]])

                        hit = True
                        totalSize = bytes + readBytes

                        if (readBytes == (freeSpaceSize[i]-bytes-j)):
                            readBytes = -1
                    else:
                        input.seek(freeSpaceStart[i]+j)
                else:
                    input.seek(freeSpaceStart[i]+j)
            else:
                input.seek(freeSpaceStart[i]+j)
        else:
            input.seek(freeSpaceStart[i]+j)

        values = []

        list = GetValueDeleted((freeSpaceSize[i]-j))
        values.append(list[0])
        bytes = list[1]

        if (hit == False and values[0] >= 2 and values[0] < (dbSize*pageSize) and bytes <
freeSpaceSize[i] and list[1] != 0):

            list = GetValueDeleted((freeSpaceSize[i]-bytes-j))
            values.append(list[0])
            bytes += list[1]
            headerSizeBytes = list[1]

            if (values[1] >= 5 and values[1] <= values[0] and values[1] <
(headerSizeBytes+(itemMaxColumnCount*8)) and bytes < freeSpaceSize[i] and list[1] != 0):
                k = 0

                while (k < (values[1]-headerSizeBytes)):
                    list = GetValueDeleted((freeSpaceSize[i]-bytes-j))

                    if (list[1] == 0):
                        break

                    values.append(list[0])
                    k += list[1]

                bytes += k

                for value in values[2:]:
                    if (value == 10 or value == 11):
                        values[1] = 0
                        break
```

```
                if (values[1] == (k+headerSizeBytes) and values[0] ==
(values[1]+CalcRecordSize(values, 2)) and bytes <= freeSpaceSize[i] and (len(values)-2) <=
itemMaxColumnCount and list[1] != 0):
                    list = ReadRecordData(values, 3, (freeSpaceSize[i]-bytes-j))
                    readBytes = (freeSpaceSize[i]-bytes-j) - list[0]
                    records.append([list[1], list[2]])

                    hit = True
                    totalSize = bytes + readBytes

                    if (readBytes == (freeSpaceSize[i]-bytes-j)):
                        readBytes = -1
                else:
                    input.seek(freeSpaceStart[i]+j)
            else:
                input.seek(freeSpaceStart[i]+j)
        else:
            input.seek(freeSpaceStart[i]+j)

        values = []

        if ((freeSpaceSize[i]-j) > 4):
            values.append(int(input.read(4).encode('hex'), 16))
            bytes = 4
        else:
            values.append(0)
            bytes = 0

        if (hit == False and values[0] > 1 and values[0] < dbSize and IsPointerMapPage(values[0])
== False and bytes < freeSpaceSize[i]):
            list = GetValueDeleted((freeSpaceSize[i]-bytes-j))
            values.append(list[0])
            bytes += list[1]

            if (values[1] >= 2 and values[1] < (dbSize*pageSize) and bytes < freeSpaceSize[i] and
list[1] != 0):
                list = GetValueDeleted((freeSpaceSize[i]-bytes-j))
                values.append(list[0])
                bytes += list[1]
                headerSizeBytes = list[1]

                if (values[2] >= 5 and values[2] <= values[1] and values[2] <
(headerSizeBytes+(itemMaxColumnCount*8)) and bytes < freeSpaceSize[i] and headerSizeBytes !=
0):
                    while (k < (values[2]-headerSizeBytes)):
                        list = GetValueDeleted((freeSpaceSize[i]-bytes-j))

                        if (list[1] == 0):
                            break

                        values.append(list[0])
                        k += list[1]

                    bytes += k
```

```
                k = 0

                for value in values[3:]:
                    if (value == 10 or value == 11):
                        values[2] = 0
                        break

                if (values[2] == (k+headerSizeBytes) and values[1] ==
(values[2]+CalcRecordSize(values, 3)) and bytes <= freeSpaceSize[i] and (len(values)-3) <=
itemMaxColumnCount and list[1] != 0):
                    list = ReadRecordData(values, 3, (freeSpaceSize[i]-bytes-j))
                    readBytes = (freeSpaceSize[i]-bytes-j) - list[0]
                    records.append([list[1], list[2]])

                    hit = True
                    totalSize = bytes + readBytes

                    if (readBytes == (freeSpaceSize[i]-bytes-j)):
                        readBytes = -1
                else:
                    input.seek(freeSpaceStart[i]+j)
            else:
                input.seek(freeSpaceStart[i]+j)
        else:
            input.seek(freeSpaceStart[i]+j)
    else:
        input.seek(freeSpaceStart[i]+j)

    values = []

    if (hit == False):
        j += 1
    else:
        if (readBytes == -1):
            freeSpaceStart.__delitem__(i)
            freeSpaceSize.__delitem__(i)

            if (i >= len(freeSpaceStart)):
                break
        else:
            temp = freeSpaceSize[i]
            freeSpaceSize[i] = j

            if (freeSpaceSize[i] < minCharCount):
                if (temp < minCharCount):
                    freeSpaceStart.__delitem__(i)
                    freeSpaceSize.__delitem__(i)

                    if (i >= len(freeSpaceStart)):
                        break
                else:
                    freeSpaceStart.insert(i+1, (freeSpaceStart[i]+j+totalSize))
                    freeSpaceSize.insert(i+1, (temp-(j+totalSize)))
                    freeSpaceStart.__delitem__(i)
                    freeSpaceSize.__delitem__(i)
```

```
                else:
                    freeSpaceStart.insert(i+1, (freeSpaceStart[i]+j+totalSize))
                    freeSpaceSize.insert(i+1, (temp-(j+totalSize)))
                    i += 1

            j = 0
            hit = False
    else:
        while (j < freeSpaceSize[i]):
            input.seek(freeSpaceStart[i]+j)
            validHeader = False

            list = GetValueDeleted((freeSpaceSize[i]-j))
            values.append(list[0])
            bytes = list[1]
            headerSizeBytes = list[1]

            if (values[0] < (headerSizeBytes+(itemMaxColumnCount*8)) and values[0] >= 3 and bytes
< freeSpaceSize[i] and headerSizeBytes != 0):
                k = 0

                while (k < (values[0]-headerSizeBytes)):
                    list = GetValueDeleted((freeSpaceSize[i]-bytes-k-j))

                    if (list[1] == 0):
                        values[0] = -1
                        break

                    values.append(list[0])
                    k += list[1]

                bytes += k

                for value in values[1:]:
                    if (value == 10 or value == 11):
                        bytes = 0
                        break

                if (values[0] == bytes and (len(values)-1) >= args.entryMin and (len(values)-1) <=
itemMaxColumnCount):
                    validHeader = CheckDataTypes(values)

                if (validHeader == True and bytes <= freeSpaceSize[i]):
                    list = ReadRecordData(values, 1, (freeSpaceSize[i]-bytes-j))
                    readBytes = (freeSpaceSize[i]-bytes-j) - list[0]
                    records.append([list[1], list[2]])

                    hit = True
                    totalSize = bytes + readBytes

                    if (readBytes == (freeSpaceSize[i]-bytes-j)):
                        readBytes = -1

                if (validHeader == False):
                    input.seek(freeSpaceStart[i]+j)
```

```
            else:
                input.seek(freeSpaceStart[i]+j)

            values = []

            if (hit == False):
                j += 1
            else:
                if (readBytes == -1):
                    freeSpaceStart.__delitem__(i)
                    freeSpaceSize.__delitem__(i)

                    if (i >= len(freeSpaceStart)):
                        break
                else:
                    temp = freeSpaceSize[i]
                    freeSpaceSize[i] = j

                    if (freeSpaceSize[i] < minCharCount):
                        if (temp < minCharCount):
                            freeSpaceStart.__delitem__(i)
                            freeSpaceSize.__delitem__(i)

                            if (i >= len(freeSpaceStart)):
                                break
                        else:
                            freeSpaceStart.insert(i+1, (freeSpaceStart[i]+j+totalSize))
                            freeSpaceSize.insert(i+1, (temp-(j+totalSize)))
                            freeSpaceStart.__delitem__(i)
                            freeSpaceSize.__delitem__(i)
                    else:
                        freeSpaceStart.insert(i+1, (freeSpaceStart[i]+j+totalSize))
                        freeSpaceSize.insert(i+1, (temp-(j+totalSize)))
                        i += 1

                j = 0
                hit = False

        j = 0
        i += 1

    return records


def FindFreePages(nextPage):
    i = 0
    currentPage = nextPage
    input.seek(nextPage)

    nextPage = int(input.read(4).encode('hex'), 16)

    freePageCount = int(input.read(4).encode('hex'), 16)

    if ((freePageCount*4) < pageSize and (pageSize-(freePageCount*4)) >= minCharCount):
        freeSpaceStart.append(input.tell()+(freePageCount*4))
```

```
        freeSpaceSize.append((currentPage+pageSize)-(input.tell()+(freePageCount*4)))

    while (i < freePageCount):
        offset = (int(input.read(4).encode('hex'), 16) - 1) * pageSize
        freeSpaceStart.append(offset)
        freeSpaceSize.append(pageSize)
        i += 1

    if (nextPage != 0):
        FindFreePages((nextPage-1)*pageSize)


def ModeValue(mode):
    i = 0
    values = []
    zero = False

    if (len(mode) > 3):
        msg = "Du kan enbart v\xe4lja metod 0 eller en kombination av metoderna 1, 2 och 3!"
        raise argparse.ArgumentTypeError(msg)

    while (i < len(mode)):
        if (mode[i] < "0" or mode[i] > "3"):
            msg = "Metod %s finns inte!" % mode[i]
            raise argparse.ArgumentTypeError(msg)
        elif (mode[i] == "0"):
            zero = True

        if (values.__contains__(int(mode[i]))):
            msg = "Varje metod kan enbart v\xe4ljas en g\xe5ng!"
            raise argparse.ArgumentTypeError(msg)

        values.append(int(mode[i]))

        i += 1

    if (zero == True and len(values) > 1):
        msg = "Du kan enbart v\xe4lja metod 0 eller en kombination av metoderna 1, 2 och 3!"
        raise argparse.ArgumentTypeError(msg)
    elif (zero == True):
        return "0"
    else:
        mode = ""

        for value in values:
            if (value == 1):
                mode += "1"

        for value in values:
            if (value == 2):
                mode += "2"

        for value in values:
            if (value == 3):
                mode += "3"
```

```
        return mode

def CheckMax(max):
    try:
        max = int(max)
    except:
        msg = "Maximum antalet karakt\xe4rer m\xe5ste vara en siffra!"
        raise argparse.ArgumentTypeError(msg)

    if (argMin[0] >= int(max)):
        msg = "Maximum antalet karakt\xe4rer m\xe5ste vara st\xf6rre \xe4n minimum antalet
karakt\xe4rer (standard f\xf6r minimum \xe4r 3)!"
        raise argparse.ArgumentTypeError(msg)
    else:
        return int(max)

def CheckMin(min):
    try:
        min = int(min)
    except:
        msg = "Minimum antalet karakt\xe4rer m\xe5ste vara en siffra!"
        raise argparse.ArgumentTypeError(msg)

    if (min < 1):
        msg = "Minimum antalet karakt\xe4rer m\xe5ste vara st\xf6rre \xe4n noll!"
        raise argparse.ArgumentTypeError(msg)
    else:
        argMin[0] = min

        return min

def CheckEntryMin(min):
    try:
        min = int(min)
    except:
        msg = "Minimum antalet entries m\xe5ste vara en siffra!"
        raise argparse.ArgumentTypeError(msg)

    if (min < 1):
        msg = "Minimum antalet entries m\xe5ste vara st\xf6rre \xe4n noll!"
        raise argparse.ArgumentTypeError(msg)

    return min

def CheckWildcardNull(bool):
    if (bool == "true" or bool == "True" or bool == "*"):
        return True
    elif (bool == "false" or bool == "False" or bool == "!*"):
        return False
    else:
        msg = "Du m\xe5ste ange antingen true/True/* eller false/False/!* !"
        raise argparse.ArgumentTypeError(msg)

# Here the execution of the program starts.
```

```
import argparse
import hashlib
import os
import struct

# This part checks the arguments the program receives and parses them.
# If the wrong type or not enough arguments are specified then a warning
# and usage instructions are displayed. This part also handles the help
# and version options.
parser = argparse.ArgumentParser(description = 'Letar efter data i en SQLite databas (f\xf6r
tillf\xe4llet \xe4r programmet bara en prototyp som ska visa att metod 1 och 2 fungerar).')
parser.add_argument('infile', help ='input fil (SQLite databas fil)')
parser.add_argument('outfile', help = 'output fil')
parser.add_argument('-m', '--mode', dest = 'mode', metavar = '0123', type = ModeValue, required =
True, help = 'Ange vilken metod som ska anv\xe4ndas f\xf6r att extrahera raderat data. 0 = ingen
s\xf6kning efter raderat data (kan inte kombineras med de andra alternativen), 1 = Leta efter record
headers med passande cell headers, 2 = Leta efter record headers som passar till befintliga tabeller och
index, 3 = Leta efter str\xe4ngar (alternativen 1 t.o.m. 3 kan kombineras med varandra).')
parser.add_argument('-v', '--version', action = 'version', version = 'SQLite Data Finder v0.4', help =
'Visar programmets version.')
args = parser.parse_known_args()

if (args[0].mode[0] == "1" or args[0].mode[0] == "2" or (len(args[0].mode) > 1 and args[0].mode[1]
== "2")):
    parser.add_argument('-ic', '--max_item_columns', dest = 'itemMaxColumnCount', metavar = '1-
65536', default = 0, type = int, choices = xrange(0, 65537), help = 'Anger maximala antalet kolumner
en tabell eller ett index f\xe5r ha. 0 inneb\xe4r att programmet sj\xe4lv f\xf6rs\xf6ker lista ut det
st\xf6rsta antalet kolumner tabellerna och indexen har.')
if (args[0].mode[0] == "2" or (len(args[0].mode) > 1 and args[0].mode[1] == "2")):
    parser.add_argument('-e', '--entry_minimum', dest = 'entryMin', metavar = 'positiv number', default
= 4, type = CheckEntryMin , help = 'Minsta antalet entries en eventuell record m\xe5ste ha f\xf6r att
metod 2 inte ska ignorera den.')
    parser.add_argument('-w', '--wildcard_null', dest = 'wildcardNull', metavar = 'true/false', default =
True, type = CheckWildcardNull, help = 'Best\xe4mmer om null v\xe4rden ska behandlas som
wildcard eller null n\xe4r matchning med metod 2 utf\xf6rs.')
if (args[0].mode[len(args[0].mode)-1] == "3"):
    argMin = [3]
    group = parser.add_mutually_exclusive_group(required = True)
    group.add_argument('-c', '--charset', dest = 'charset', metavar = 'aA0!', choices = 'aA0!')
    group.add_argument('-cf', '--charsetfile', dest = 'charsetFile', metavar = '<filename>')
    parser.add_argument('-cmin', '--char_count_min', dest = 'minCharCount', metavar = 'positiv
number', default = 3, type = CheckMin, help = 'Antalet tecken en str\xe4ng minst m\xe5ste ha f\xf6r
att metod 3 ska l\xe4sa in den.')
    parser.add_argument('-cmax', '--char_count_max', dest = 'maxCharCount', metavar = 'positiv
number', default = 0, type = CheckMax, help = 'Antalet tecken som maximalt l\xe4ses in av metod 3.
M\xe5ste vara st\xf6rre \xe4n cmin.')

args = parser.parse_args()

if (args.mode[len(args.mode)-1] == "3" and len(args.mode) == 1):
    minCharCount = args.minCharCount
elif (args.mode[len(args.mode)-1] == "3"):
    if (args.minCharCount <= 5):
        minCharCount = args.minCharCount
    else:
```

```
        minCharCount = 5
else:
    minCharCount = 5

print "\xd6ppnar fil...",

# This variable is used to determine if something went wrong during the
# execution of the program or the analysis of the database file.
failed = False

# Here the database file (the argument "infile") is opened and any
# eventual exception is caught. The variable failed is set to true
# so that the program knows something went wrong and the user is
# also informed that the file could'nt be opened.
try:
    input = open(args.infile, 'rb')

    # Creates a sha512 hash algorithm object.
    sha512 = hashlib.new('sha512')

    # Updates the hash algorithm object with parts of the
    # input file until the entire file has been given as input
    # to the hash algorithm.
    for part in iter(lambda: input.read(8*sha512.block_size), ''):
        sha512.update(part)

    input.seek(0)
except:
    failed = True
    print "misslyckades!"

# Here the program checks if it was able to open the specified file.
# If it was the user is informed about the success and the execution
# continues. If it wasn't the variable failed is set to "" (nothing)
# and the execution won't be continued.
if (failed == True):
    print "\nKunde inte \xf6ppna filen.\nAvbryter exekvering."
    failed = ""
elif (failed == False):
    print "klar"
    print "Kontrollerar headern...",

    # Reads the first 16 bytes from the file, which will be used to
    # determine if the file is a SQLite version 3.0 or later database.
    firstLine = input.readline(16)

    # This try-catch statement catches all exceptions. If an exception
    # is raised the user is informed that something went wrong and the
    # variable failed is set to true.
    try:
        # The output file is opened.
        output = open(args.outfile, 'w')

        # This if-statement checks if the first line in the file has
        # the right format, if not the output file is closed and
```

```
        # execution terminated.
        if (firstLine == "SQLite format 3\x00"):
            # Retrieves the size of the pages in the database.
            pageSize = int(input.read(2).encode('hex'), 16)

            # Retrieves the file format write and read version. These are
            # currently always one. These fields will be used for forward
            # compatibility. In case a future version of SQLite uses a
            # different file format these fields will be used to determine
            # if older versions may safely read or write this file format.
            writeVersion = int(input.read(1).encode('hex'), 16)              # currently not used in the
program, comment this line
            readVersion = int(input.read(1).encode('hex'), 16)              # and this line
            #input.seek(20)                                    # and uncomment this line if you want to save
memory

            # Retrieves the value for the number of bytes of unused
            # space at the end of each database page. Should always
            # be zero in a well-formed database.
            unusedBytes = int(input.read(1).encode('hex'), 16)

            # Retrieves the maximum and minimum fraction of an index
            # B-Tree and the minimum fraction of a table B-Tree. The
            # maximum fraction should always be 64 and the both of the
            # minimum fractions should always be 32 in a well-formed
            # database.
            maxFractionIndex = int(input.read(1).encode('hex'), 16)
            minFractionIndex = int(input.read(1).encode('hex'), 16)
            minFractionTable = int(input.read(1).encode('hex'), 16)

            # Retrieves the file change counter. This value is change
            # every time a database transaction is committed.
            changeCounter = int(input.read(4).encode('hex'), 16)

            # Retrieves the logical size of the database in pages. This
            # field is only valid when it is nonzero and if the values at
            # offsets 24 (changeCounter) and 92 (versionValidFor) exactly
            # match each other.
            dbSize = int(input.read(4).encode('hex'), 16)

            # Retrieves the page number of the first free-list trunk page
            # and the number of free pages in the database.
            firstFreelistTrunkPage = int(input.read(4).encode('hex'), 16)
            NumOfreePages = int(input.read(4).encode('hex'), 16)            # currently not used in the
program, comment this line
            #input.seek(40)                                    # and uncomment this line if you want to save
memory

            # Retrieves the schema version and schema layer file-format.
            # The program can currently only handle database files with a
            # schema layer file format value of one. If the value is
            # something else the program will still try to extract as much
            # data as possible.
            schemaVersion = int(input.read(4).encode('hex'), 16)
```

```
        schemaLayerFileFormat = int(input.read(4).encode('hex'), 16)        # currently not used in the
program, comment this line
        #input.seek(52)                                     # and uncomment this one if you want to save
memory
        #input.seek(48)                                     # use this instead of the line before if you
want to read the value for the pager cache size.

        # Retrieves the recommended pager cache size to use for the
        # database.
        pagerCacheSize = int(input.read(4).encode('hex'), 16)             # currently not used in the
program, comment this line
        #input.seek(52)                                     # and uncomment this one if you want to save
memory (does bot need to be uncommented if the
                                                # previous seek(52) statement was uncommented.

        # Retrieves the numerically largest root page number for
        # auto-vacuum capable databases. This is always zero for
        # non-auto-vacuum databases and at least one for auto-vacuum
        # databases.
        autoVacuum = int(input.read(4).encode('hex'), 16)

        # Retrieves the value for the text encoding. A one means the
        # text in the database was stored using UTF-8 encoding, two
        # means little-endian UTF-16 text and three means big-endian
        # UTF-16 text. Currently the program can only handle databases
        # with UTF-8 text encoding.
        textEncoding = int(input.read(4).encode('hex'), 16)

        # Retrieves the value for the user cookie.
        userCookie = int(input.read(4).encode('hex'), 16)                 # currently not used in the
program, comment this line
        #input.seek(64)                                     # and uncomment this one if you want to save
memory

        # Retrieves the value for incremental vacuum mode. Always
        # zero in non-auto-vacuum databases and also if incremental
        # vacuum mode is not enabled in auto-vacuum databases. One if
        # enabled.
        IncrementalVacuumMode = int(input.read(4).encode('hex'), 16)

        # Retrieves the version-valid-for integer and the version number
        # of SQLite that last wrote to the database file.
        input.seek(92)
        versionValidFor = int(input.read(4).encode('hex'), 16)
        versionNumber = int(input.read(4).encode('hex'), 16)

        # This part writes the acquired information to the output file.
        output.write("SQLite databas filnamn: " + args.infile + "\n")
        output.write("SHA-512 hash: " + str(sha512.hexdigest()) + "\n")
        output.write("\nHeader information\n")
        output.write("------------------\n")

        if (versionNumber != 0):
            majorVersionNumber = versionNumber / 1000000
            minorVersionNumber = versionNumber - (majorVersionNumber * 1000000)
```

```
            releaseNumber = minorVersionNumber % 1000
            minorVersionNumber = minorVersionNumber / 1000
            output.write("Version: " + str(majorVersionNumber) + "." + str(minorVersionNumber) + "."
+ str(releaseNumber) + "\n")

            if (changeCounter != versionValidFor):
               dbSize = 0
         else:
            output.write("Version: L\xe4gre \xe4n 3.7.0\n")
            dbSize = 0

         if (dbSize == 0):
            dbSize = os.path.getsize(args.infile) / pageSize

         if (textEncoding == 1):
            output.write("Text encoding: UTF-8\n")
         elif (textEncoding == 2):
            output.write("Text encoding: little-endian UTF-16 (currently not supported)\n")
         elif (textEncoding == 3):
            output.write("Text encoding: big-endian UTF-16 (currently not supported)\n")
         else:
            output.write("Text encoding: ok\xe4nd\n")

         output.write("Page size: " + str(pageSize) + " bytes\n")
         output.write("Databas storlek: " + str(dbSize) + " pages\n")
         output.write("Change counter: " + str(changeCounter) + "\n")
         output.write("Schema version: " + str(schemaVersion) + "\n")

         if (autoVacuum != 0):
            output.write("Auto-vacuum: ja\n")
            if (IncrementalVacuumMode == 1):
               output.write("Incremental vacuum: ja\n")
            else:
               output.write("Incremental vacuum: nej\n")
         else:
            output.write("Auto-vacuum: nej\n")
      else:
         failed = True
         print "misslyckades!"
   except:
      failed = True
      print "misslyckades!"
   finally:
      if (failed == True):
         output.close()

   # If failed is set to true the user is informed that the execution has
   # been stopped and failed is set to "". Otherwise the execution continues.
   if (failed == True):
      print "\nIngen giltig SQLite header har hittats.\nKontrollera om det finns en fil med samma namn
som databasfilen vilken dessutom har \"-journal\" bifogat i slutet",
      print "(SQLite Data Finder st\xf6djer inte databaser som har journal filer)."
      print "Annars \xe4r SQLite versionen f\xf6r gammal (f\xf6re SQLite version 3) eller filen
inneh\xe5ller ingen giltig SQLite databas image.\nAvbryter exekvering."
      failed = ""
```

```
    elif (failed == False):
        try:
            print "klar"
            print "Extraherar schema/master tabell...",

            freeSpaceStart = []
            freeSpaceSize = []

            # Here the schema/master table of the database is read. Page
            # one of the database is always the root page of the
            # schema/master table and since the database header takes up
            # the first 100 bytes (0-99) of page one the page header starts
            # at offset 100.
            schemaTable = ReadPage(100)

            tables = []
            indices = []
            triggers = []
            views = []

            # Here the entries in the schema table are parsed and saved to
            # lists for tables, indices, triggers and views respectively.
            # The variable schemaTable contains a list of all schema items.
            # Each schema item is saved as a list where the first value is
            # the key value and the second the actual data for that schema
            # item. The second value in that list always specifies the item
            # type.
            for entry in schemaTable:
                if (entry[2][0] == "table" or entry[2][0] == "TABLE" or entry[2][0] == "t a b l e" or
entry[2][0] == "T A B L E"):
                    tables.append(entry)
                elif (entry[2][0] == "index" or entry[2][0] == "INDEX" or entry[2][0] == "i n d e x" or
entry[2][0] == "I N D E X"):
                    indices.append(entry)
                elif (entry[2][0] == "trigger" or entry[2][0] == "TRIGGER" or entry[2][0] == "t r i g g e r"
or entry[2][0] == "T R I G G E R"):
                    triggers.append(entry)
                else:
                    views.append(entry)

            # These two lists will simplify the access to table and index
            # root pages.
            tableRootPages = []
            IndexRootPages = []
            i = 0

            # Here the names and root pages of the tables are extracted.
            if (len(tables) != 0):
                while (i < len(tables)):
                    # If the root page value is zero then the table is a
                    # virtual table. In that case the entire entry of the
                    # virtual table is removed from the list tables.
                    if (tables[i][2][3] != 0):
                        tableRootPages.append(tables[i][2][3])
                        i += 1
```

```
            else:
                tables.remove(tables[i])

        i = 0

        # Here the root pages of the indices are extracted.
        if (len(indices) != 0):
            for entry in indices:
                IndexRootPages.append(entry[2][3])
    except:
        failed = True
        print "misslyckades!"
    finally:
        if (failed == True):
            output.close()

# If failed is set to true the user is informed that the execution has
# been stopped and failed is set to "". Otherwise the execution continues.
if (failed == True):
    print "Det gick inte att extraherar schema/master tabell.\nAvbryter exekvering."
    failed = ""
elif (failed == False):
        print "klar"
        print "Extraherar data ur tabeller och index...",

        tableData = []
        indexData = []

        # Here the data is extracted from each table and saved to the
        # list tableData.
        for page in tableRootPages:
            tableData.append(ReadPage((page-1)*pageSize))

        # Here the data is extracted from each index and saved to the
        # list indexData.
        for page in IndexRootPages:
            indexData.append(ReadPage((page-1)*pageSize))

# If failed is set to true the user is informed that the execution has
# been stopped and failed is set to "". Otherwise the execution continues.
if (failed == True):
    print "Det gick inte att extraherar data ur tabeller och index.\nAvbryter exekvering."
    failed = ""
elif (failed == False and args.mode != "0"):
    print "klar"
    print "Extraherar raderat data ur databasen...",

    if (firstFreelistTrunkPage != 0):
        FindFreePages((firstFreelistTrunkPage-1) * pageSize)

    i = 0
    j = 1

    if (args.mode[0] == "1" or args.mode[0] == "2"):
        itemDataTypes = [["schem_table", ["text", "text", "text", "integer", "text"]]]
```

```
        if ((args.mode[0] == "1" or args.mode[0] == "2" or (len(args.mode) > 1 and args.mode[1] ==
"2")) and args.itemMaxColumnCount == 0):
            itemMaxColumnCount = 5
        elif (args.mode[0] == "1" or args.mode[0] == "2" or (len(args.mode) > 1 and args.mode[1] ==
"2")):
            itemMaxColumnCount = args.itemMaxColumnCount

        for entry in tables:
            itemDataTypes.append([entry[2][1], FindTableDataTypes(entry[2][4])])

            if ((args.mode[0] == "1" or args.mode[0] == "2" or (len(args.mode) > 1 and args.mode[1]
== "2")) and args.itemMaxColumnCount == 0 and itemMaxColumnCount <
len(itemDataTypes[len(itemDataTypes)-1][1])):
                itemMaxColumnCount = len(itemDataTypes[len(itemDataTypes)-1][1])

        for entry in indices:
            if (len(entry[2]) == 5):
                while (i < len(itemDataTypes)):
                    if (itemDataTypes[i].__contains__(entry[2][2])):
                        tableTypes = itemDataTypes[i][1]
                        tableStatement = tables[i-1][2][4]
                        break

                    i += 1

                itemDataTypes.append([entry[2][1], FindIndexDataTypes(entry[2][4], tableTypes,
tableStatement)])

                i = 0

    while (i < len(args.mode)):
        if (args.mode[i] == "1"):
            if (autoVacuum != 0):
                numEntries = (pageSize - unusedBytes) / 5
                pointerMapPages = [2]

                while (pointerMapPages[j-1] < dbSize):
                    pointerMapPages.append(2 + (j * numEntries))
                    j += 1

                pointerMapPages.__delitem__(j-1)
            else:
                pointerMapPages = []

            mode1 = DeletedRecordSearch(1)
        elif (args.mode[i] == "2"):
            mode2 = DeletedRecordSearch(2)
        else:
            print "St\xf6d f\xf6r metod 3 har inte implementerats \xe4n! ...",

        i += 1

    # If failed is set to true the user is informed that the execution has
    # been stopped and failed is set to "". Otherwise the execution continues.
```

```
if (failed == True):
    print "Det gick inte att extraherar raderat data ur databasen.\nAvbryter exekvering."
    failed = ""
elif (failed == False):
    #try:
    print "klar"
    print "Skriver datan till outputfilen...",

    i = 0
    j = 0
    tableColumns = []

    print >> output, "\n\n\n"
    PrintTable(schemaTable, 0, "schema")
    print >> output, "Tables\n------\n\n"

    while (i < len(tables)):
        for entry in schemaTable:
            if (entry.__contains__(tables[i][0])):
                key = j
                break

            j += 1

        j = 0

        if (len(tableData[i]) != 0):
            tableColumns.append(PrintTable(tableData[i], key, "table"))
        else:
            tableColumns.append(PrintTable(-1, key, "table"))
        i += 1

    i = 0

    print >> output, "Indices\n-------\n\n"

    while (i < len(indices)):
        for entry in schemaTable:
            if (entry.__contains__(indices[i][0])):
                key = j
                break

            j += 1

        j = 0

        if (len(indexData[i]) != 0):
            PrintTable(indexData[i], key, "index")
        else:
            PrintTable(-2, key, "index")
        i += 1

    if (args.mode != 0):
        i = 0
        zero = 0
```

```
counter = 1

print >> output, "Deleted data\n----------\n\n"

if (args.mode[0] == "1"):

    print >> output, "Metod 1\n-------\n\n"

    if (len(mode1) != 0):
        for item in mode1:
            if (type(item[0]).__name__ == "list"):
                print >> output, "%s" % item[0][1]

                while (i < len(item[0])):
                    if (item[0][i] >= 12 and (item[0][i] % 2) == 0):
                        try:
                            blob = open("deleted-blob" + str(counter) + ".bin", 'wb')
                            blob.write(item[1][i-zero])
                            blob.close()
                            counter += 1
                        except:
                            print >> output, "Kunde inte skriva deleted-blob" + str(counter)
                            counter += 1
                    elif (item[0][i] == 0):
                        zero += 1

                    i += 1

                zero = 0
                i = 0
            else:
                print >> output, "%s" % item[1][1]

                while (i < len(item[1][0])):
                    if (item[1][0][i] >= 12 and (item[1][0][i] % 2) == 0):
                        try:
                            blob = open("deleted-blob" + str(counter) + ".bin", 'wb')
                            blob.write(item[1][1][i-zero])
                            blob.close()
                            counter += 1
                        except:
                            print >> output, "Kunde inte skriva deleted-blob" + str(counter)
                            counter += 1
                    elif (item[1][0][i] == 0):
                        zero += 1

                    i += 1

                zero = 0
                i = 0

        print >> output, "\n\n"
    else:
        print >> output, "Metod 1 hittade ingen raderat data.\n\n"
```

```
        if (args.mode[0] == "2" or (len(args.mode) > 1 and args.mode[1] == "2")):
            print >> output, "Metod 2\n-------\n\n"

            if (len(mode2) != 0):
                for item in mode2:
                    print >> output, "%s" % item[1]

                    while (i < len(item[0])):
                        if (item[0][i] >= 12 and (item[0][i] % 2) == 0):
                            try:
                                blob = open("deleted-blob" + str(counter) + ".bin", 'wb')
                                blob.write(item[1][i-zero])
                                blob.close()
                                counter += 1
                            except:
                                print >> output, "Kunde inte skriva deleted-blob" + str(counter)
                                counter += 1
                        elif (item[0][i] == 0):
                            zero += 1

                        i += 1

                    zero = 0
                    i = 0

                print >> output, "\n\n"
            else:
                print >> output, "Metod 2 hittade ingen raderat data.\n\n"

        # Method 3 is not supported yet!
        #if (args.mode[len(args.mode)-1] == "3"):
            #print >> output, "Metod 3\n-------\n\n"
#       except:
#           failed = True
#           print "misslyckades!"
#       finally:
    output.close()

    # If failed is set to true the user is informed that the execution has
    # been stopped and failed is set to "". Otherwise the execution continues.
    if (failed == True):
        print "Det gick inte att datan till outputfilen.\nAvbryter exekvering."
        failed = ""
    elif (failed == False):
        print "klar"

    if (failed == False):
        # Creates another sha512 hash algorithm object which will be used to check
        # if the input file has been corrupted somehow during execution.
        hashCheckSha512 = hashlib.new('sha512')

        input.seek(0)

        for part in iter(lambda: input.read(8*sha512.block_size), ''):
            hashCheckSha512.update(part)
```

```
        # Compares the two hashes against each other and gives a warning in case they
        # do not match.
        if (sha512.hexdigest() == hashCheckSha512.hexdigest()):
            print "\nHash kontroll av input filen gav korrekt resultat."
            print "Exekveringen avslutades utan fel."
        else:
            print "\nHash kontroll av input filen gav inkorrekt resultat!"
            print "Varning! Hashen f\xf6r input filen har \xe4ndrats sedan exekveringen startades."
            print "Resultatet av genoms\xf6kningen \xe4r ogiltig!"
            print "Exekveringen avslutades med fel!"

    # The input file is being closed.
    input.close()

# End of program
```