SQLite FORENSIC ANALYSIS

DAN O'DAY
LAKE COUNTY HIDTA

About Me

- Digital forensic examiner, Lake County HIDTA (since 2009)
- Training contractor, viaForensics (since 2012)
- Former adjunct instructor, Purdue University Calumet
- Background: US Army Signal Corps (also an amateur radio operator), criminal intelligence analyst
- Training includes DHS/FLETC, DOJ, Guidance Software, X-Ways, and more….
- M.S. Technology, Purdue University Calumet
- Accepted into Purdue University (West Lafayette) CERIAS interdisciplinary Ph.D. program in information security beginning Fall 2014

Time	Topic
0900 - 0930	**Introduction & System Setup**
0930 - 1000	**SQLite Basics & Core Concepts**
1000 - 1015	Break
1015 - 1035	**Core Concepts (continued)**
1035 - 1100	**Writing SQL Queries**
1100 - 1115	Break
1115 - 1135	**Writing SQL Queries (continued)**
1135 - 1200	**SQLite Deleted Record Recovery**
1200 - 1300	Lunch
1300 - 1400	**SQLite Forensic Analysis Lab**
1400 - 1415	Break
1415 - 1500	**HTCIA Business**

SYSTEM SETUP

- ☑ FIREFOX BROWSER
  - ☑ SQLITE MANAGER ADD-ON
- ☑ CHINOOK SAMPLE DATABASE
- ☑ PYTHON (2 & 3)
- ☑ SQLITE DELETED RECORDS PARSER

INSTALL Firefox + SQLITE MANAGER

- GO TO HTTP://CHINOOKDATABASE.CODEPLEX.COM
  - DOWNLOAD

Project Description

Chinook is a sample database available for SQL Server, Oracle, MySQL, etc. It can be created by running a single SQL script. Chinook database is an alternative to the Northwind database, being ideal for demos and testing ORM tools targeting single and multiple database servers.

What is New?

» Dec 10, 2012: Chinook database is now available on NuGet! See details here.

Search Wiki & Documentation

download

CURRENT	Chinook Database 1.4
DATE	Sun Dec 2, 2012 at 7:00 AM
STATUS	Stable
DOWNLOADS	15,051

DOWNLOAD CHINOOK SAMPLE DATABASE

hidtaisc — Python — 76×10

```
Last login: Mon Jun 23 11:57:06 on ttys000
HIDTAs-Mac-mini:~ hidtaisc$ python
Python 2.7.5 (default, Mar  9 2014, 22:15:05)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

hidtaisc — Python — 73×15

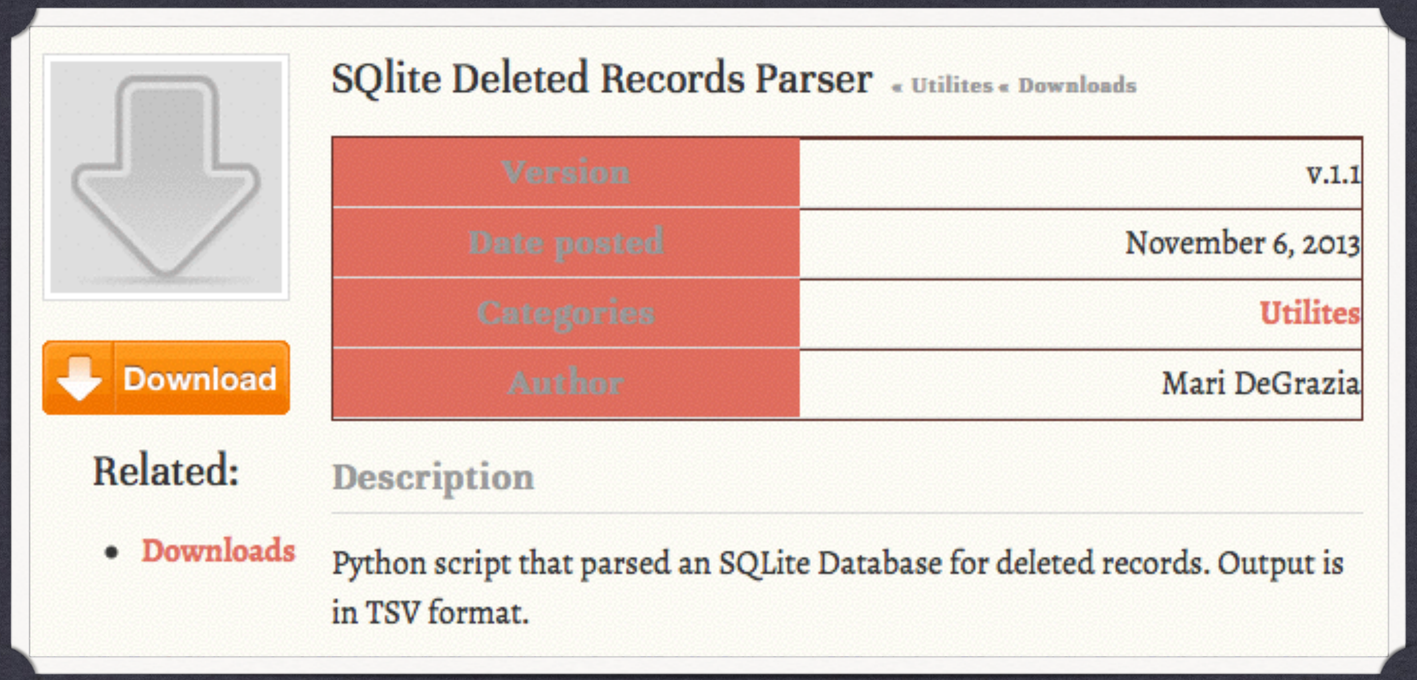
```
HIDTAs-Mac-mini:~ hidtaisc$ python3
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 00:54:21)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

```
Installing on OS X 10.8 (Mountain Lion) or later systems
========================================================

If you are attempting to install on an OS X 10.8+ system, you may
see a message that Python can't be installed because it is from an
unidentified developer.  This is because this Python installer
package is not yet compatible with the Gatekeeper security feature
introduced in OS X 10.8.  To allow Python to be installed, you
can override the Gatekeeper policy for this install.  In the Finder,
instead of double-clicking, control-click or right click the "Python"
installer package icon.  Then select "Open using ... Installer" from
the contextual menu that appears.
```

```
Python.org Python 3.4 and 2.7.x versions can both be installed and
will not conflict. Command names for Python 3 contain a 3 in them,
python3 (or python3.4), idle3 (or idle3.4), pip3 (or pip3.4), etc.
Python 2.7 command names contain a 2 or no digit: python2 (or
python2.7 or python), idle2 (or idle2.7 or idle), etc. If you want
```

INSTALL python™ 2 & 3

- GO TO HTTP://WWW.ARIZONA4N6.COM
  - DOWNLOADS
  - SQLITE DELETED RECORDS PARSER



SQlite Deleted Records Parser « Utilites « Downloads

Version	v.1.1
Date posted	November 6, 2013
Categories	Utilites
Author	Mari DeGrazia

Download

Related:

- Downloads

Description

Python script that parsed an SQLite Database for deleted records. Output is in TSV format.

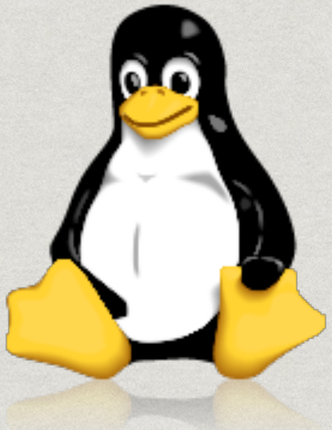
INSTALL SQLITE DELETED RECORDS PARSER

SQLITE BASICS

What is SQLite?

- Structured Query Language - Lite
- Relational database management engine
- Entire database is stored in a single file

Who Uses It?



symbian

LG webOS

Why SQLite?

- Most widely deployed database engine [1]
- Cross-platform compatible
- No server required, self-contained file, no dependencies
- No configuration/setup needed
- Public domain, F/OSS
- Why use a custom data storage format when this one is free and easy to use?
  - Supported by most programming languages
- Small code footprint, efficient use of memory, disk space, bandwidth, etc. (hence its used on mobile devices)

- [1] http://www.sqlite.org/mostdeployed.html

Relevant Distinctives

- Manifest (weakly) typed (plays well with dynamically-typed languages)
- Variable-length records
- Rollback journal and/or write-ahead log (WAL)
- Limited query language

SQLITE CORE CONCEPTS

Relational Database

- Database contains tables
- Tables contain records (rows of data)
  - Each record has a unique identifier called a primary key (PK / rowid)
- Records can have relationships with records in other tables by using their PK, which is then called a foreign key (FK)

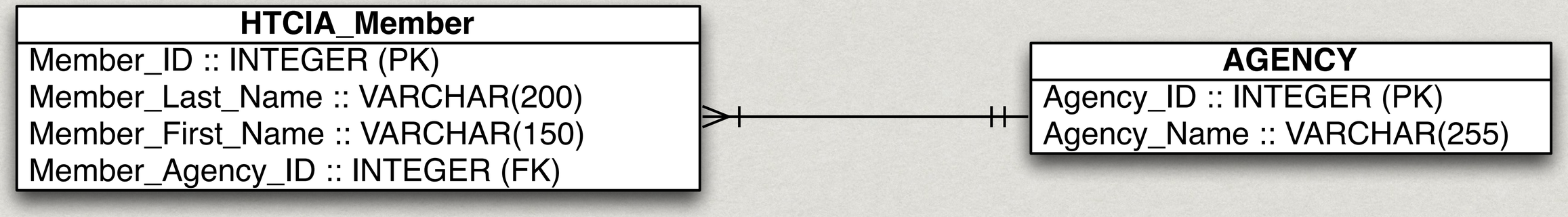
Data Types

- NULL
- Integer (signed integer, 1-8 bytes)
- Real (floating point value, 8-byte IEEE)
- Text (text string, encoded UTF-8 or 16)
- BLOB (binary large object)

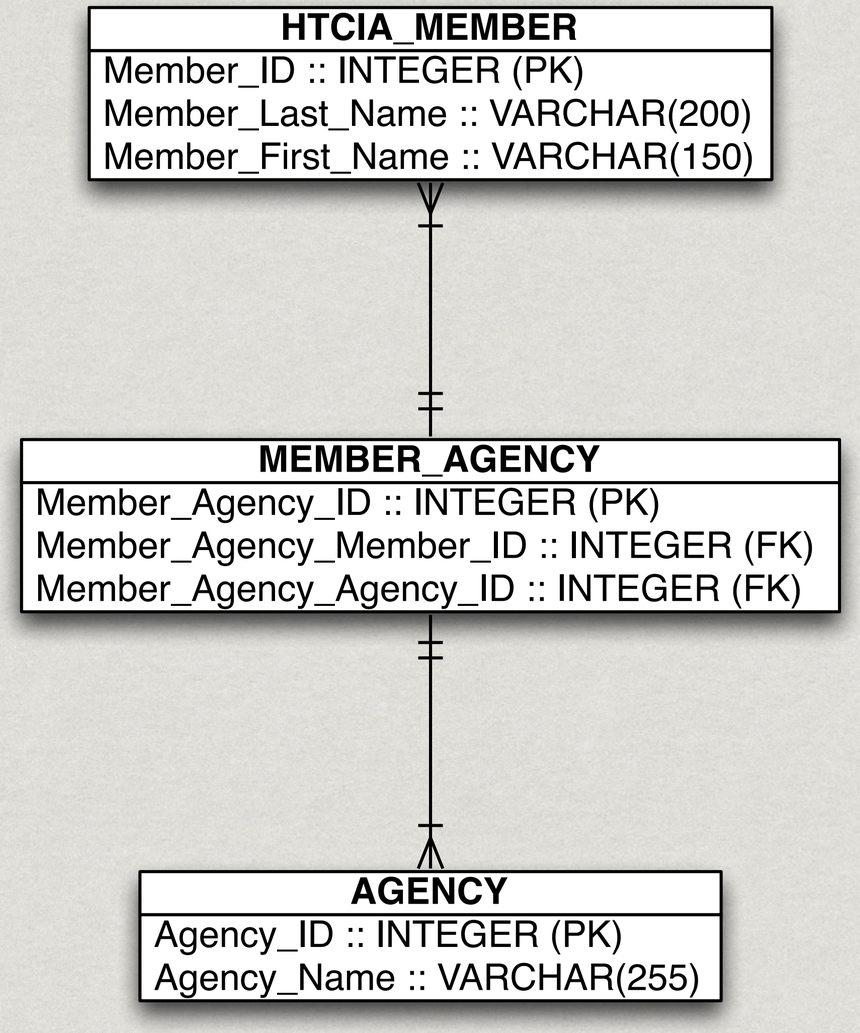
Single Table

HTCIA_Member
Member_ID :: INTEGER (PK)
Member_Last_Name :: VARCHAR(200)
Member_First_Name :: VARCHAR(150)
Member_Agency_Name :: VARCHAR(255)

One-To-Many Relationship



Many-To-Many Relationship



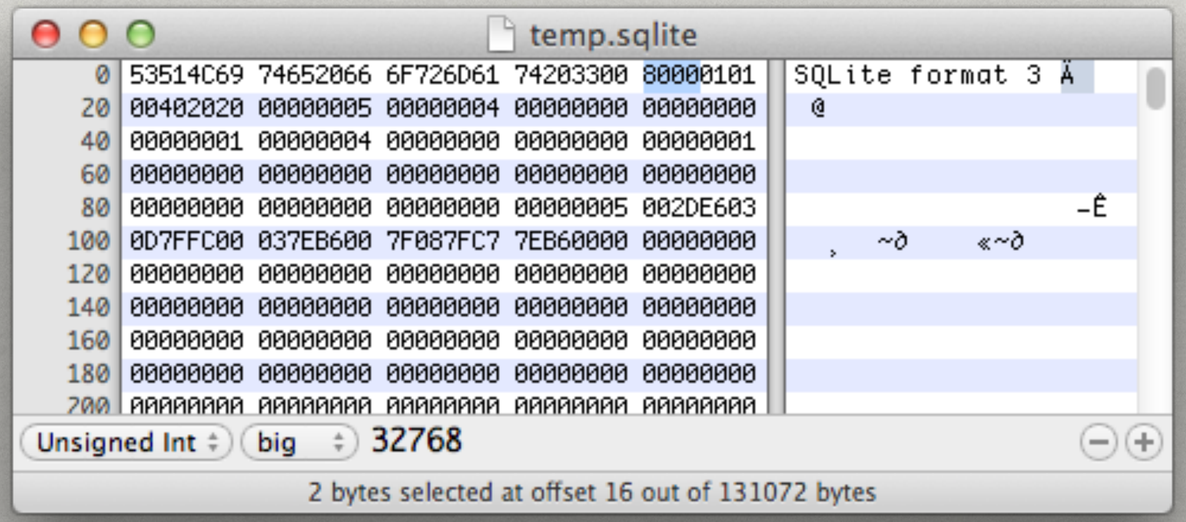
SQLite Database File Format

- Database (db) file consists of one or more **pages**
- When parsing any multi-byte integer value in SQLite, read it using big endian

http://www.sqlite.org/fileformat.html

Page Size

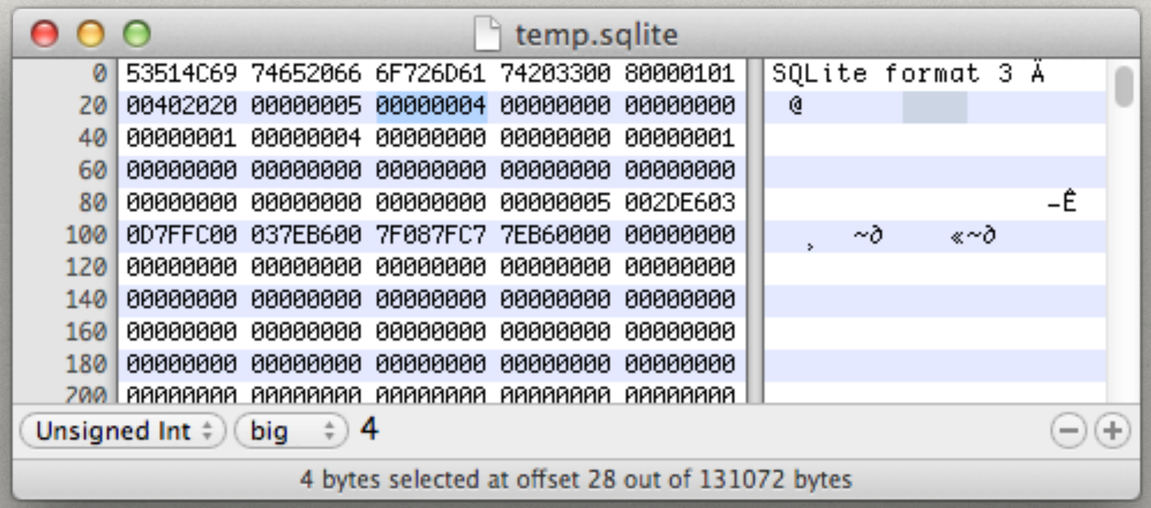
- Page size is a power of two between 512 and 65536 (inclusive)
- Pages within a db file are all the same size
- Page size is a 2-byte integer (big endian) at byte offset 16
- Maximum # pages = 2147483646
- Thus a SQLite db can grow to approximately 140 TB



http://www.sqlite.org/fileformat.html

Total # of Pages

- Total # of pages is stored as a 4-byte integer at byte offset 28



http://www.sqlite.org/fileformat.html

Page Uses

- Every page in the database has a single purpose and is one of the following (although not all db's have all of these):
  - Lock-byte page
  - Freelist page
    - Trunk & leaves
  - B-tree page
    - Table b-tree interior & leaves
    - Index b-tree interior & leaves
  - Payload overflow page
  - Pointer map page

http://www.sqlite.org/fileformat.html

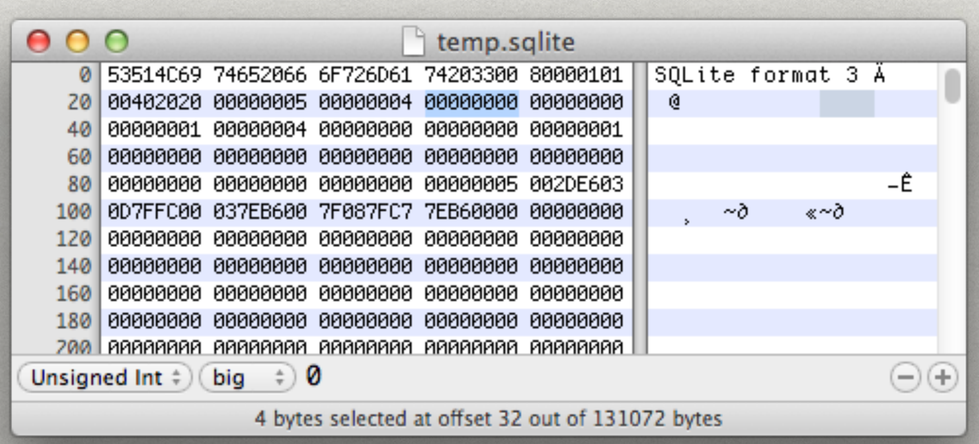
Lock-Byte Page

- Set aside for OS-specific (VFS) implementation of file locking (not used by SQLite)

http://www.sqlite.org/fileformat.html

Freelist Page(s)

- Pages not in active use
- Maintains a list of page #'s where DELETED data *may* reside *within the logical db file*
- Organized as linked list of free list trunk pages containing page #'s for 0+ free list leaf pages
- # of freelist pages stored as 4-byte integer at byte offset 32



http://www.sqlite.org/fileformat.html

Freelist Trunk & Leaf Pages

- Trunk page
  - Contains array of 4-byte integers (array will store as many integers as will fit within a page)
  - First integer contains page # of next free list trunk page (or 0 if it is the last freelist trunk page)
  - Second integer is the # of leaf page pointers to follow
- Leaf page
  - Contains no information currently used by db (SQLite doesn’t read or write)

http://www.sqlite.org/fileformat.html

B-Tree Pages

- Binary trees
  - Key/data storage
    - Most keys are variable integers (varints), between 1 and 9 bytes
  - SQLite has two types:

    1. Table b-trees (data stored in leaves, key corresponds to rowid)
    2. Index b-trees (only keys stored, no data)

http://www.sqlite.org/fileformat.html

B-Tree Pages

- B-tree page is either:
  - Interior page (keys with pointers to child b-tree pages, i.e. page #)
    - Key + pointer on left = “cell”
  - Leaf page (contains keys, and if table b-tree each key will have associated data)

http://www.sqlite.org/fileformat.html

Payload

- “Payload” of a cell = arbitrary length section (for index b-trees this is the key, for table b-tree leaf pages this is the content, and interior table b-trees have no payload)
- When payload exceeds threshold, only first few bytes are stored on b-tree page and balance is stored in a linked list of content overflow pages
- First four bytes of payload overflow page stores page # of next page in the chain

http://www.sqlite.org/fileformat.html

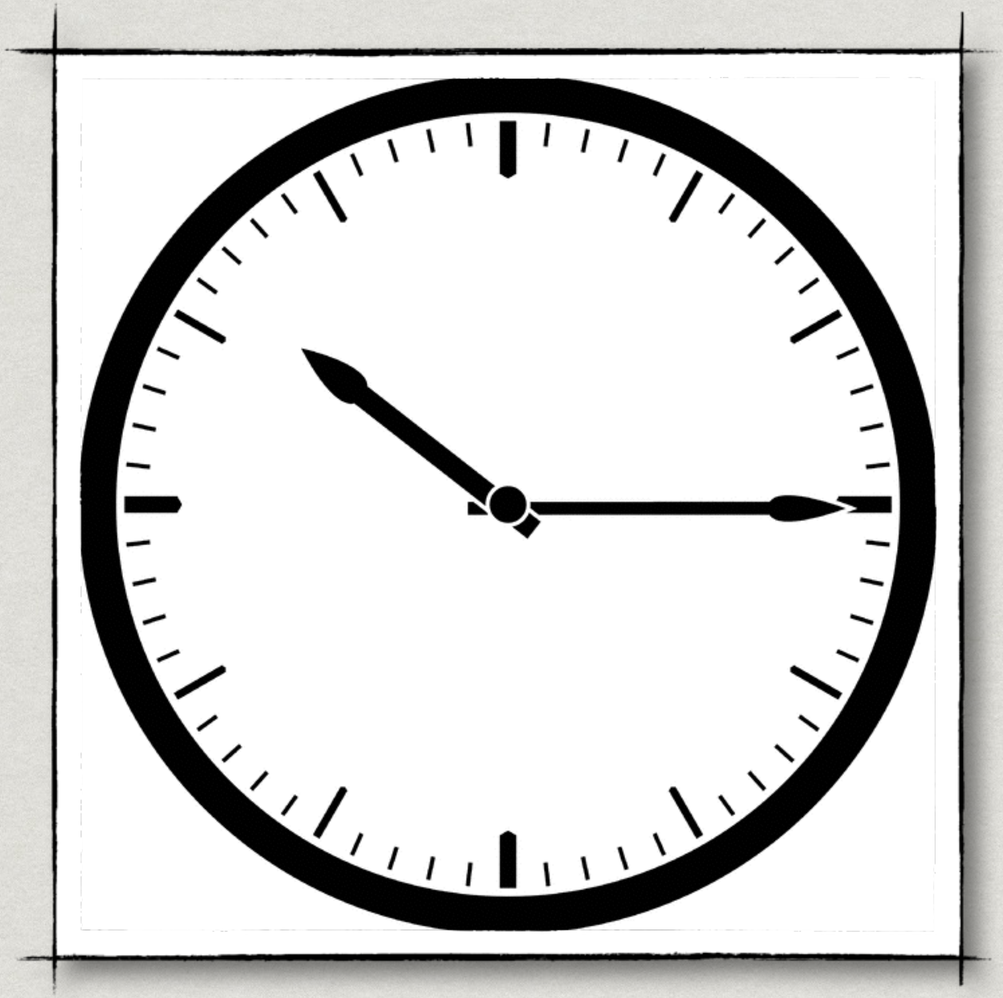
Overflow

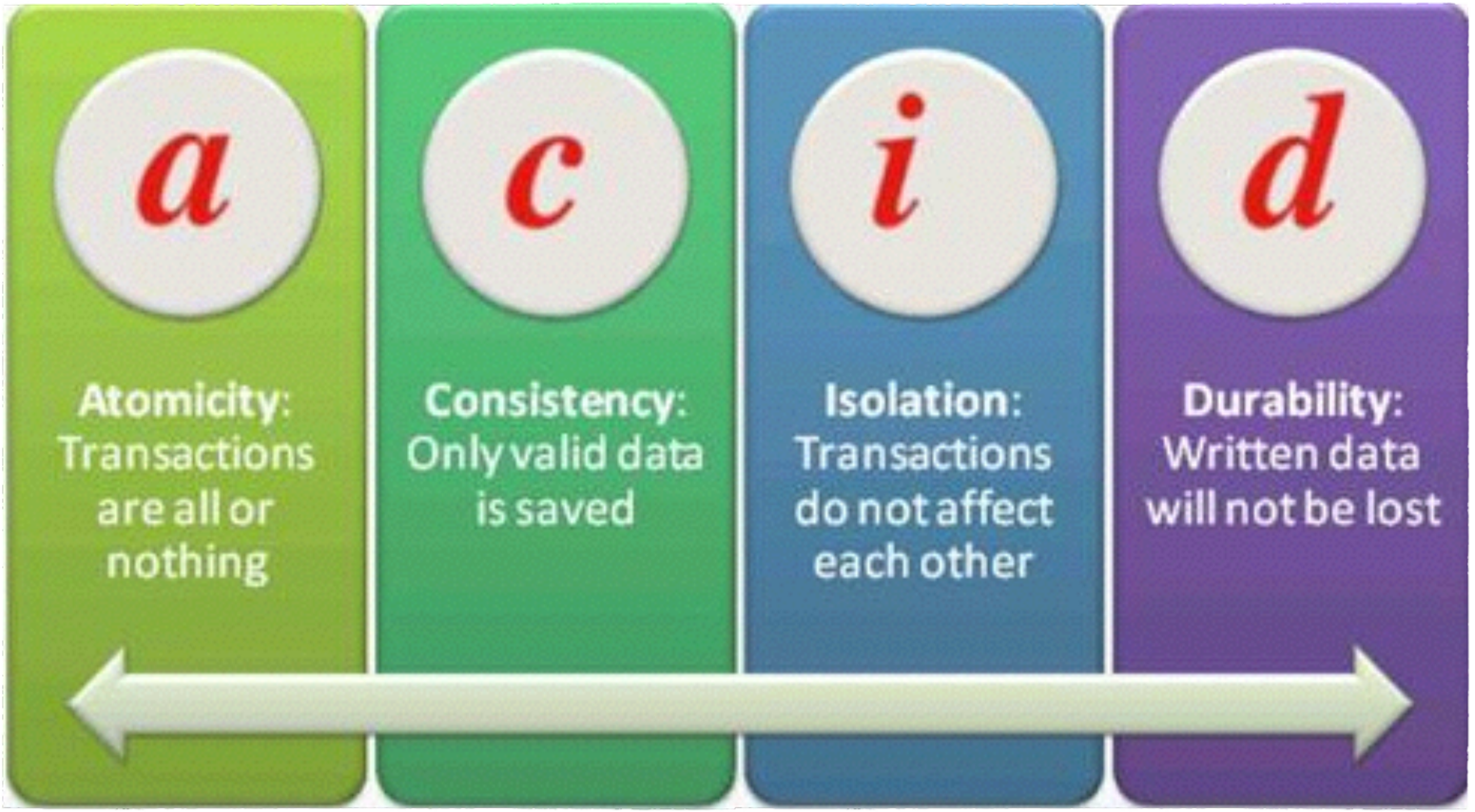
- Check out http://forensicsfromthesausagefactory.blogspot.com/2011/07/sqlite-overflow-pages-and-other-loose.html
- Explains getting Safari cached thumbnails from overflow pages

Pointer Map (Ptrmap) Page(s)

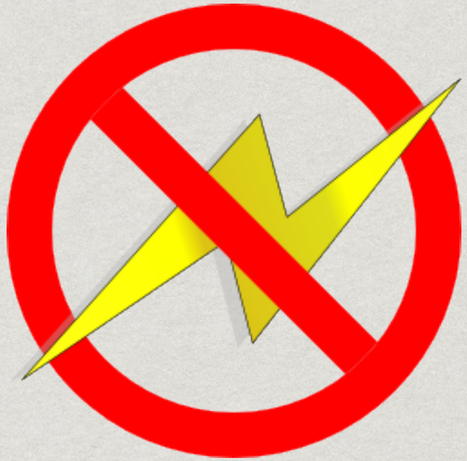
- Extra pages inserted into db to make VACUUM more efficient
- All other page types in db have pointers from parent to child
  - Ptrmap is exact opposite: pointers from child to parent
- Page 2 is a Ptrmap when AUTO_VACUUM is enabled (cf. http://forensicsfromthesausagefactory.blogspot.com/2011/05/sqlite-pointer-maps-pages.html)

http://www.sqlite.org/fileformat.html




a
Atomicity:
Transactions are all or nothing
c
Consistency:
Only valid data is saved
i
Isolation:
Transactions do not affect each other
d
Durability:
Written data will not be lost

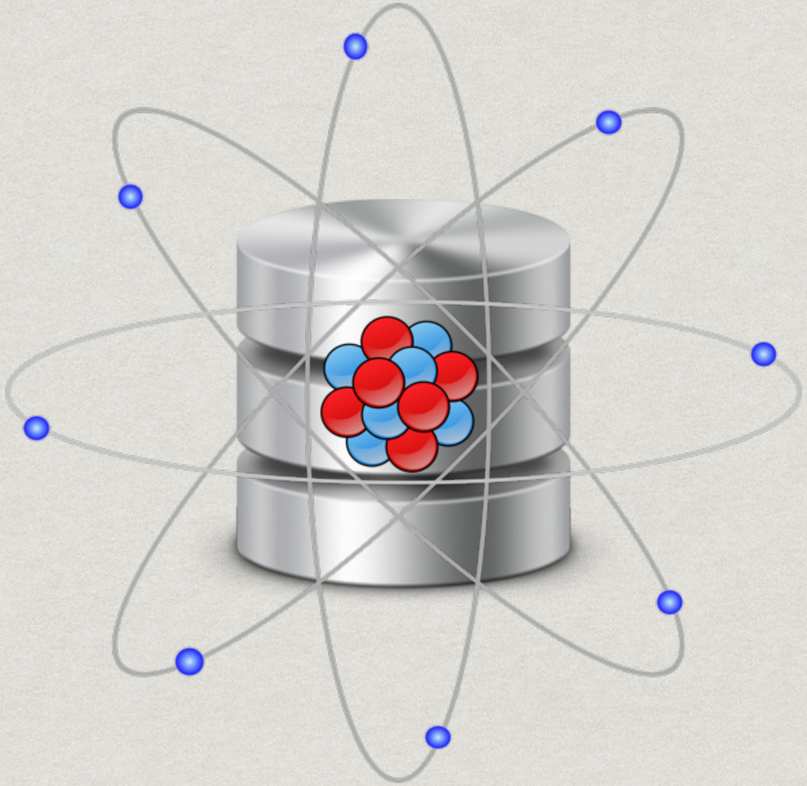
Atomicity & Durability



Atomic Commit

Atomic Commit “Modes”

1. **Rollback journal**
2. **Write-ahead log (WAL)**



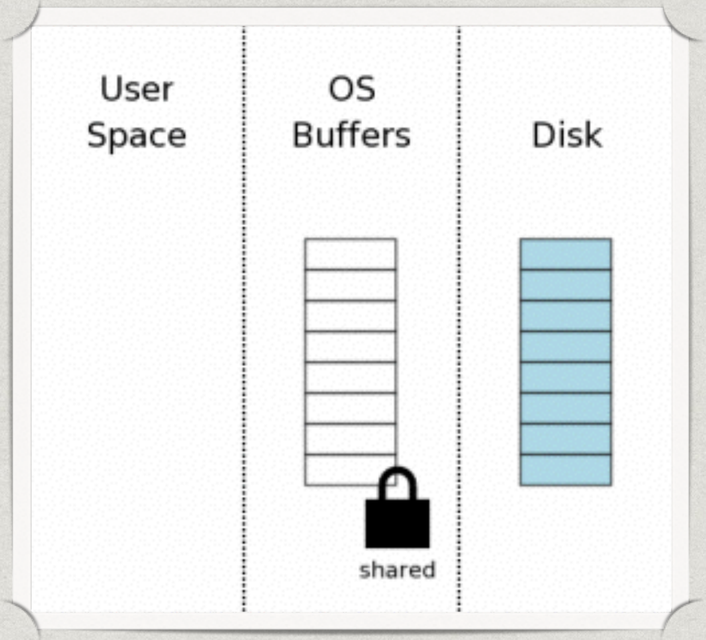
What happens when writing to a db with a rollback journal?

- Creates rollback journal file with original data being altered
- Stored in same folder as db with '-journal' filename suffix
- Allows restoration of database to original state
- Default behavior of SQLite

http://www.sqlite.org/atomiccommit.html

Rollback Journal

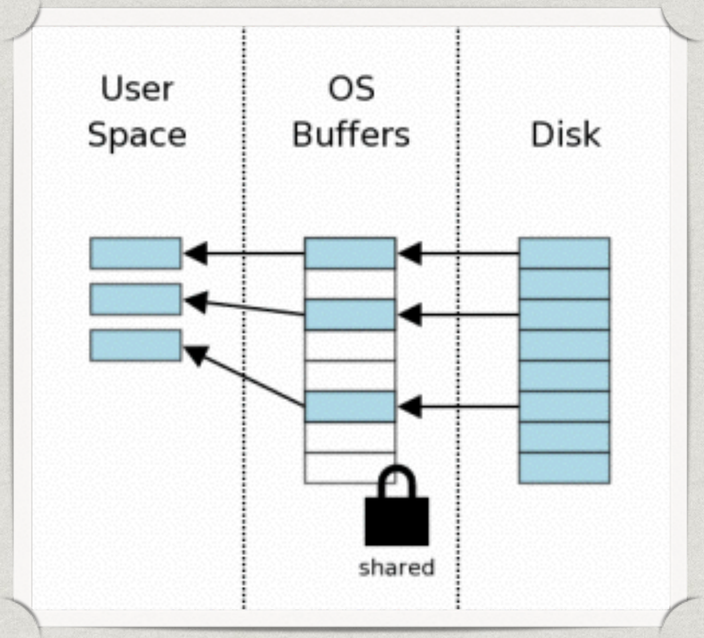
- “Shared lock” placed on database file - allows multiple reads but prevents writing while reading

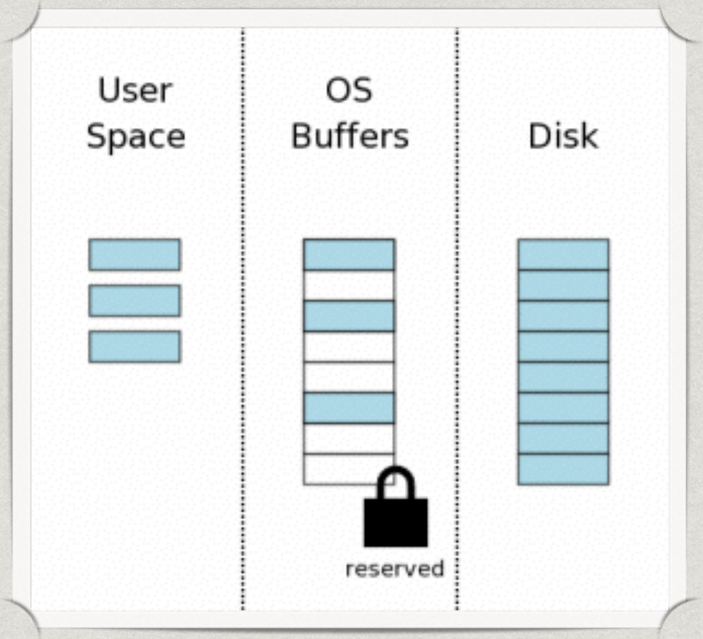


http://www.sqlite.org/atomiccommit.html

Rollback Journal

- Only a subset of pages in db file are read (but always writes a complete ‘sector’ of data)
  - Freelist pages are not journaled
- “Reserved” lock placed on db file preventing any writes from other processes (signals intent to write)

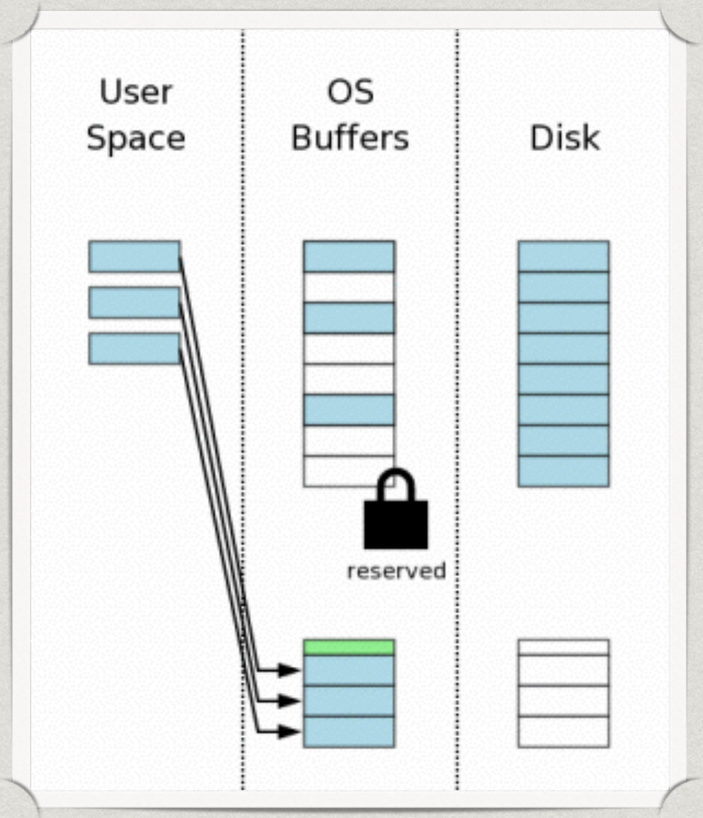




http://www.sqlite.org/atomiccommit.html

Rollback Journal

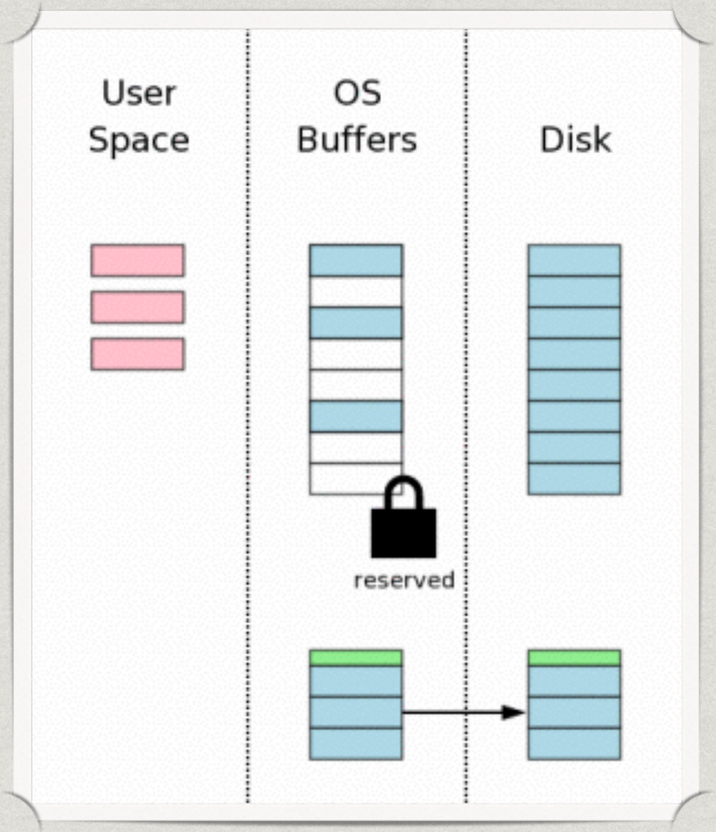
- Rollback journal file is created
  - Most OS's do not write this file to disk initially, stored in cache only
- Header contains original db file size
- Pages are modified in user space (RAM)



http://www.sqlite.org/atomiccommit.html

Rollback Journal

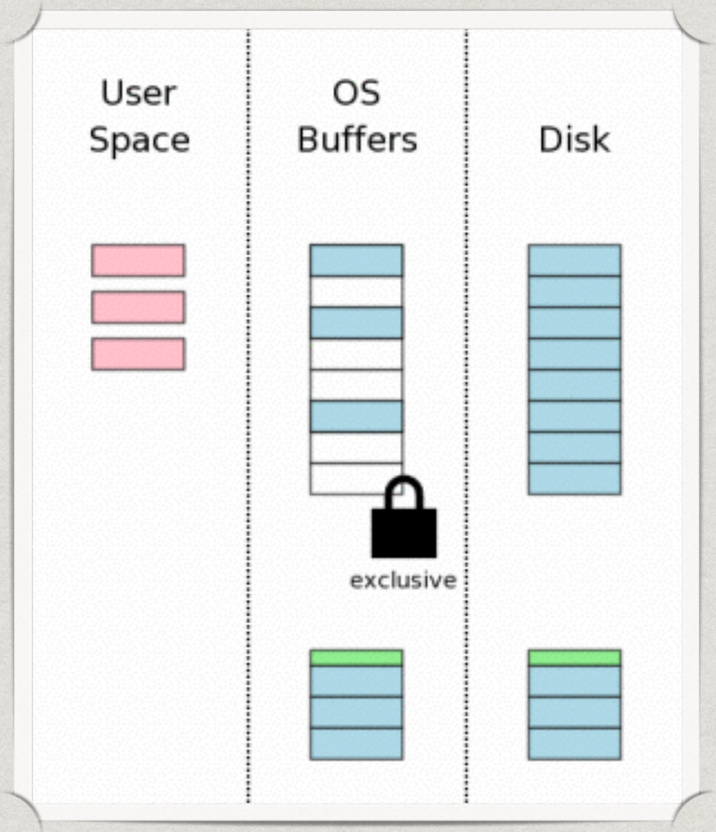
- Flush contents of journal file to non-volatile storage (forces write to disk)
- Crucial step so db can survive unexpected power loss



http://www.sqlite.org/atomiccommit.html

Rollback Journal

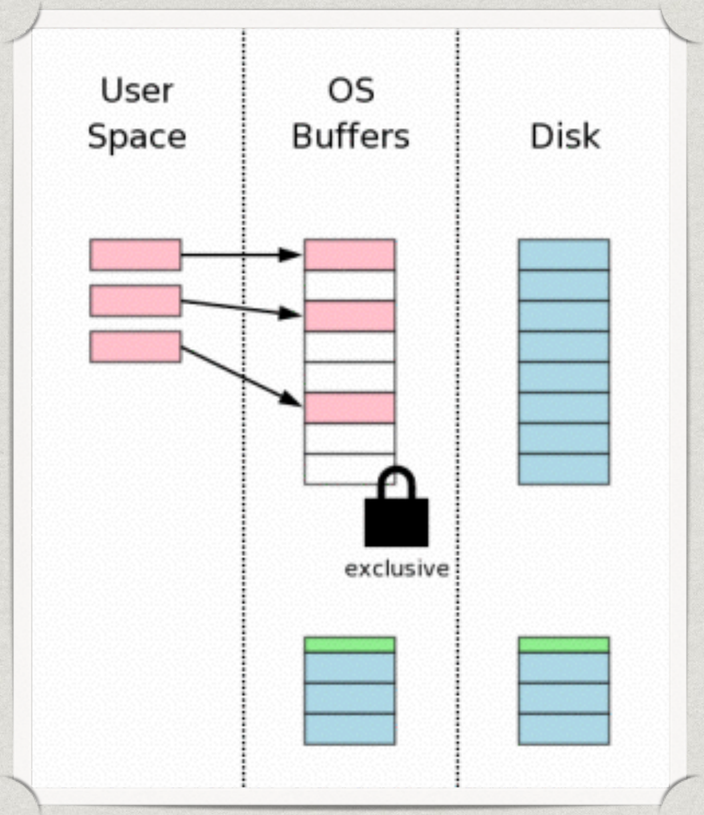
- “Exclusive lock” obtained
  - “Pending lock” issued first which allows any existing ‘shared locks’ to finish but allows no new locks (preventing ‘writer starvation’)

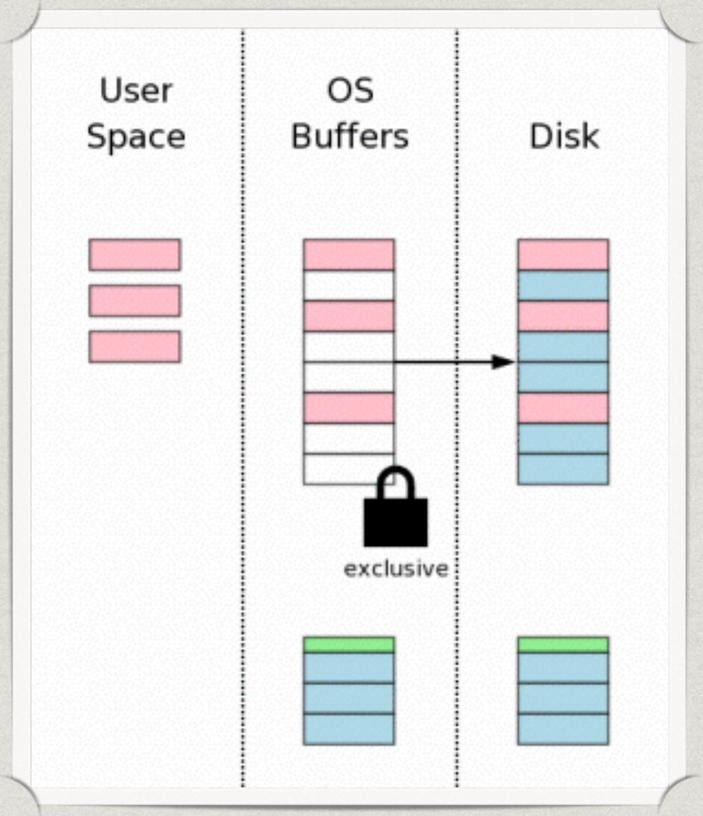


http://www.sqlite.org/atomiccommit.html

Rollback Journal

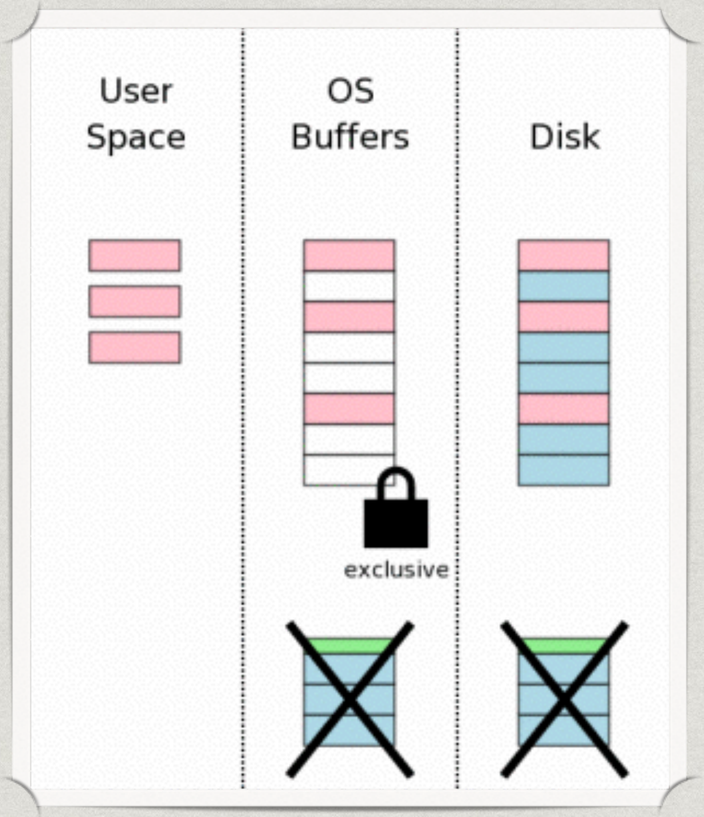
- Changes are written to db file and then flushed to disk

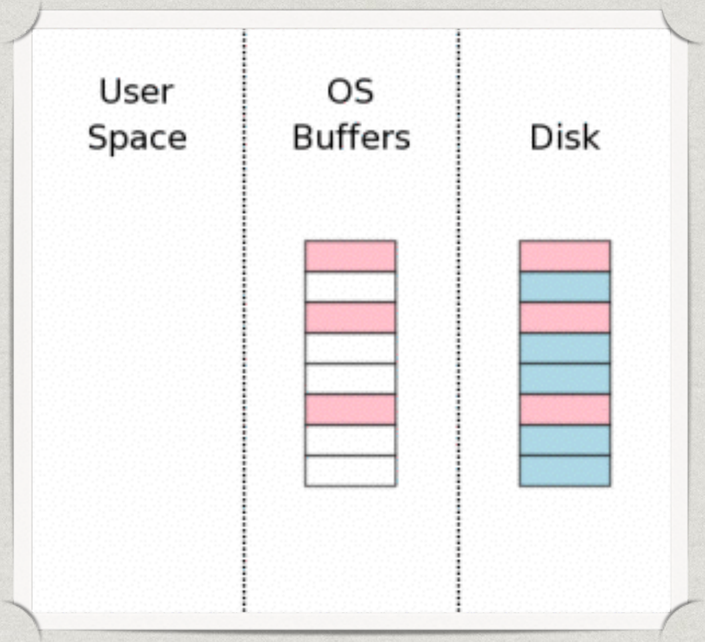




Rollback Journal

- Rollback journal is deleted (transaction now considered ‘committed’)
- Lock is released





Rollback Journal

- Deleting files is ‘expensive’ on many file systems
- SQLite can be configured so that either:

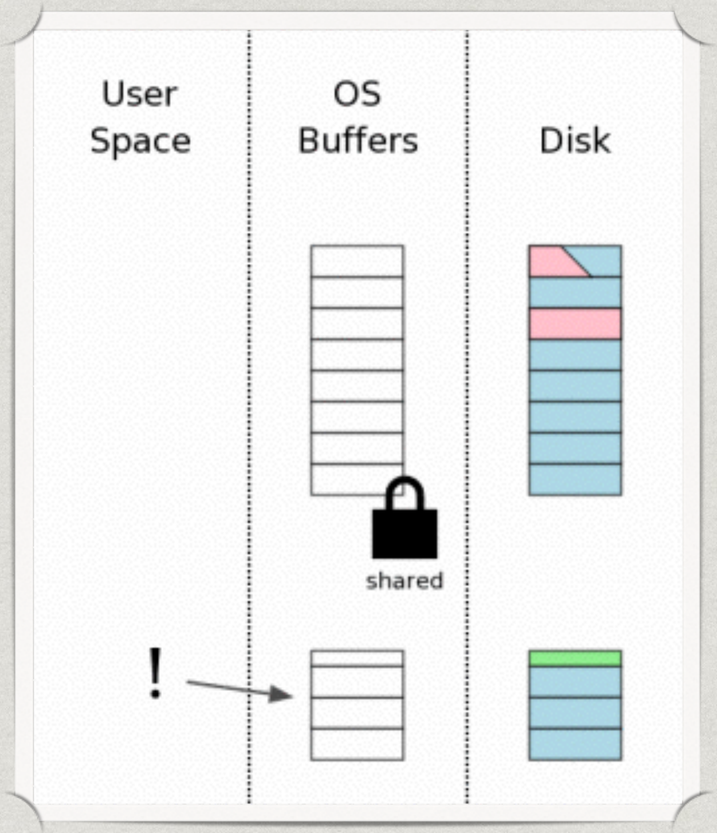
  1. Journal header is zeroed out (and the next time a journal is needed, the existing one is simply overwritten rather than creating a ‘new’ file)

  2. Journal file is truncated to 0 byte file size

http://www.sqlite.org/atomiccommit.html

“Hot” Rollback Journal

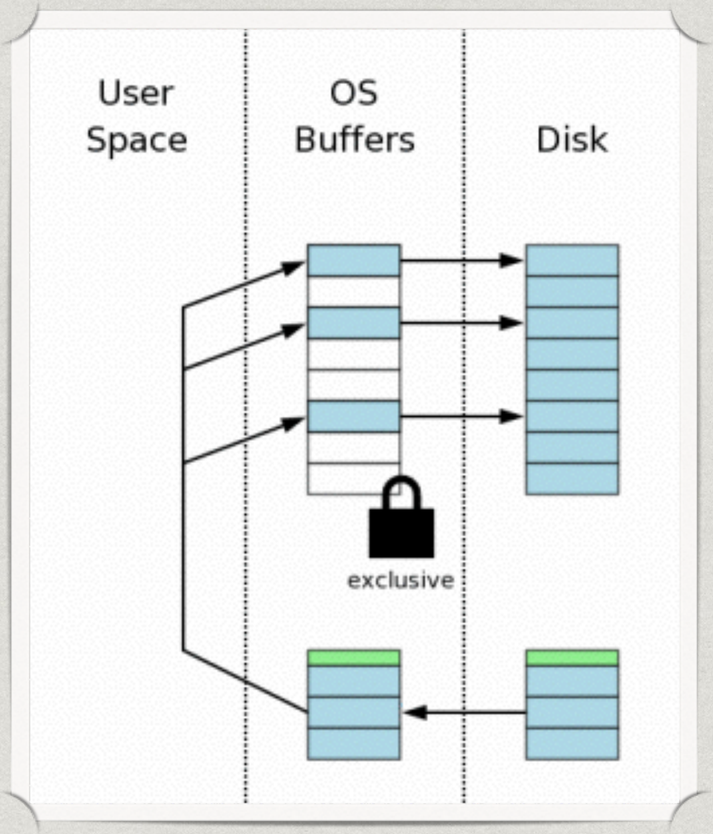
- Something went wrong, data only partially written and/or not written when it should have been
- Database is inconsistent and must be repaired



http://www.sqlite.org/atomiccommit.html

“Hot” Rollback Journal

- Incomplete changes are rolled back
  - Exclusive lock obtained
  - Content from rollback journal written to db
  - Truncate db file back to original size
  - Delete ‘hot’ journal



http://www.sqlite.org/atomiccommit.html

Write-Ahead Log (WAL)

- Allows concurrent reads and writes
- Faster for applications with more writes than reads, but only for smaller transactions
- More efficient use of disk I/O operations
- Page size cannot be changed
- Requires shared memory by all processes using db (-shm files with no persistent content), i.e. can’t be used over network
- Often can’t be read on read-only media

http://www.sqlite.org/wal.html

WAL

- Inversion of rollback journal process
  - Original content preserved in db file
  - Changes appended to WAL file
  - Transaction considered ‘committed’ when special record appended to WAL (even if change actually has not yet been written to db file)
  - Allows multiple transactions to occur ‘simultaneously’

http://www.sqlite.org/wal.html

WAL

- Checkpointing
  - Transfers all transactions appended to WAL to original db file
  - Automatically occurs when WAL reaches threshold of 1000 pages (by default)
  - When WAL is committed to disk, new appended transactions begin overwriting the WAL file

http://www.sqlite.org/wal.html

WAL

- wal-index in shared memory tracks reads to ensure proper data is returned at time requested, factoring in ‘commits’
  - Read request checks wal-index to see if page requested is in WAL
    - If not, page from original db file is returned
    - If so, most recent version of page is returned from WAL file

http://www.sqlite.org/wal.html

Temporary Files Created by SQLite

- Rollback journals (-journal)
- WAL files (-wal)
- Shared-memory files (-shm)
- Statement journals (given random name, location may not be in same directory as db, used to rollback single statement for larger transactions, look for etilqs)
- TEMP databases (gets its own db file and respective rollback journal)
- Master journals (only when single transaction makes changes to multiple databases that are using the same connection, i.e. ATTACH)
- Transient indices/databases (usually associated with VACUUM)

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

```
> SELECT * FROM users WHERE clue > 0;
0 rows returned
```

WRITING SQL QUERIES

- GO TO HTTP://CHINOOKDATABASE.CODEPLEX.COM
  - DOWNLOAD

Project Description

Chinook is a sample database available for SQL Server, Oracle, MySQL, etc. It can be created by running a single SQL script. Chinook database is an alternative to the Northwind database, being ideal for demos and testing ORM tools targeting single and multiple database servers.

What is New?

» Dec 10, 2012: Chinook database is now available on NuGet! See details here.

Search Wiki & Documentation

download

CURRENT	Chinook Database 1.4
DATE	Sun Dec 2, 2012 at 7:00 AM
STATUS	Stable
DOWNLOADS	15,051

DOWNLOAD CHINOOK SAMPLE DATABASE

Single Table

HTCIA_Member
Member_ID :: INTEGER (PK)
Member_Last_Name :: VARCHAR(200)
Member_First_Name :: VARCHAR(150)
Member_Agency_Name :: VARCHAR(255)

SELECT * FROM HTCIA_MEMBER;

Member_ID	Member_Last_Name	Member_First_Name	Member_Agency_Name
1	O’Day	Dan	Lake County HIDTA
2	Rademacher	Karl	Aon Corporation
3	Shih	Steve	GSU
4	Doe	John	ABC, Inc.
5	Smith	Mike	XYZ Company
6	Katz	Bob	ABC, Inc.

Structure | Browse & Search | Execute SQL | DB Settings

Enter SQL

```
SELECT * FROM Customer
```

Run SQL | Actions | Last Error: not an error

Custom...	FirstName	LastName	Company	Address	City	State	Country	PostalC...	Phone
1	Luís	Gonçalves	Embraer ...	Av. Briga...	São José ...	SP	Brazil	12227-...	+55 (12)...
2	Leonie	Köhler		Theodor...	Stuttgart		Germany	70174	+49 071...
3	François	Tremblay		1498 ru...	Montréal	QC	Canada	H2G 1A7	+1 (514)...
4	Bjørn	Hansen		Ullevålsv...	Oslo		Norway	0171	+47 22 ...
5	František	Wichterl...	JetBrains...	Klanova ...	Prague		Czech R...	14700	+420 2 ...
6	Helena	Holý		Rilská 3...	Prague		Czech R...	14300	+420 2 ...
7	Astrid	Gruber		Rotentur...	Vienne		Austria	1010	+43 01 ...
8	Daan	Peeters		Grétrystr...	Brussels		Belgium	1000	+32 02 ...
9	Kara	Nielsen		Sønder B...	Copenha...		Denmark	1720	+453 33...
10	Eduardo	Martins	Woodsto...	Rua Dr. ...	São Paulo	SP	Brazil	01007-...	+55 (11)...

Returned: 59

```
SELECT Member_Last_Name, Member_Agency_Name
FROM HTCIA_MEMBER;
```

Member_Last_Name	Member_Agency_Name
O’Day	Lake County HIDTA
Rademacher	Aon Corporation
Shih	GSU
Doe	ABC, Inc.
Smith	XYZ Company
Katz	ABC, Inc.

Enter SQL

```
SELECT FirstName, LastName FROM Customer
```

Run SQL | Actions | Last Error:

FirstName	LastName
Luís	Gonçalves
Leonie	Köhler
François	Tremblay
Bjørn	Hansen
František	Wichterlová
Helena	Holý
Astrid	Gruber
Daan	Peeters
Kara	Nielsen
Eduardo	Martins

```
SELECT Member_Last_Name, Member_Agency_Name
FROM HTCIA_MEMBER ORDER BY Member_Last_Name;
```

Member_Last_Name	Member_Agency_Name
Doe	ABC, Inc.
Katz	ABC, Inc.
O’Day	Lake County HIDTA
Rademacher	Aon Corporation
Shih	GSU
Smith	XYZ Company

Enter SQL

```
SELECT LastName, FirstName FROM Customer ORDER BY LastName, FirstName
```

Run SQL | Actions | Last Error: not an error

LastName	FirstName
Almeida	Roberto
Barnett	Julia
Bernard	Camille
Brooks	Michelle
Brown	Robert
Chase	Kathy
Cunningham	Richard
Dubois	Marc
Fernandes	João
Francis	Edward

```
SELECT DISTINCT Member_Agency_Name
FROM HTCIA_MEMBER ORDER BY Member_Agency_Name;
```

Member_Agency_Name
ABC, Inc.
Aon Corporation
GSU
Lake County HIDTA
XYZ Company

How many countries are the customers from?

```
SELECT DISTINCT Country FROM Customer ORDER BY Country
```

Run SQL | Actions | Last Error: not an error

Country
Argentina
Australia
Austria
Belgium
Brazil
Canada
Chile
Czech Republic
Denmark
Finland

Number of Rows Returned: 24

```
SELECT
  Member_ID,
  Member_Last_Name,
  Member_Agency_Name
FROM HTCIA_MEMBER
WHERE Member_ID > 4
ORDER BY Member_Last_Name;
```

Member_ID	Member_Last_Name	Member_Agency_Name
6	Katz	ABC, Inc.
5	Smith	XYZ Company

```
SELECT
  Member_ID,
  Member_Last_Name,
  Member_Agency_Name
FROM HTCIA_MEMBER
WHERE Member_Agency_Name='ABC, Inc.'
ORDER BY Member_Last_Name DESC;
```

Member_ID	Member_Last_Name	Member_Agency_Name
6	Katz	ABC, Inc.
4	Doe	ABC, Inc.

```
SELECT
  Member_ID,
  Member_Last_Name,
  Member_Agency_Name
FROM HTCIA_MEMBER
WHERE Member_Agency_Name='ABC, Inc.'
  AND Member_ID=6
ORDER BY Member_Last_Name DESC;
```

Member_ID	Member_Last_Name	Member_Agency_Name
6	Katz	ABC, Inc.

```
SELECT
  Member_ID,
  Member_Last_Name,
  Member_Agency_Name
FROM HTCIA_MEMBER
WHERE Member_ID > 4
  OR Member_Agency_Name='ABC, Inc.'
ORDER BY Member_Last_Name;
```

Member_ID	Member_Last_Name	Member_Agency_Name
4	Doe	ABC, Inc.
6	Katz	ABC, Inc.
5	Smith	XYZ Company

```
SELECT
  Member_ID,
  Member_Last_Name,
  Member_Agency_Name
FROM HTCIA_MEMBER
WHERE Member_Last_Name LIKE ‘Kat%’;
```

Member_ID	Member_Last_Name	Member_Agency_Name
6	Katz	ABC, Inc.

```
SELECT
  Member_ID AS id,
  Member_Last_Name AS last,
  Member_Agency_Name AS agency
FROM HTCIA_MEMBER
WHERE last LIKE ‘Kat%’;
```

id	last	agency
6	Katz	ABC, Inc.

WHERE Clause Operators

Operator	Description
=	Equal (also IS)
<>	Not equal (also != and IS NOT)
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	Specify multiple possible values for a column

Wildcard Operators

Operator	Description
%	Substitute for 0+ characters
_	Substitute for 1 character (underscore)

Other Operator

Operator	Description
||	String concatenation operator

Basic (Aggregate) Functions

Operator	Description
avg(n)	Average of all non-NULL values in n column
count(n)	Count instances of given non-NULL column
lower(x)	Returns x as lowercase ASCII string
upper(x)	Returns x as uppercase ASCII string
max(n)	Returns maximum value in column n
min(n)	Returns minimum value in column n
sum(n)	Sums all non-NULL values (prone to errors if bad or unexpected input is given)
total(n)	Sums all non-NULL values and returns floating point value

http://www.sqlite.org/lang_aggfunc.html#minggunc

Show customer’s full name and country as only two columns

String concatenation

```
select FirstName || ' ' || LastName as Name, Country from Customer
```

Run SQL | Actions | Last Error: not an error

Name	Country
Luís Gonçalves	Brazil
Leonie Köhler	Germany
François Tremblay	Canada
Bjørn Hansen	Norway
František Wichterlová	Czech Republic
Helena Holý	Czech Republic
Astrid Gruber	Austria
Daan Peeters	Belgium
Kara Nielsen	Denmark
Eduardo Martins	Brazil

Number of Rows Returned: 59

Which customers have Hotmail email addresses?

```
SELECT CustomerId, LastName, FirstName, Country, Email FROM Customer WHERE Email LIKE '%@hotmail.com' ORDER BY LastName, FirstName
```

Run SQL | Actions | Last Error: not an error

CustomerId	LastName	FirstName	Country	Email
21	Chase	Kathy	USA	kachase@hotmail.com
26	Cunningham	Richard	USA	ricunningham@hotmail.com
41	Dubois	Marc	France	marc.dubois@hotmail.com
52	Jones	Emma	United Kingdom	emma_jones@hotmail.com

4 rows returned

How many customers live in Canada, Austria, **<u>or</u>** Belgium?

```
SELECT * FROM Customer WHERE Country = 'Austria' OR Country = 'Belgium' OR Country = 'Canada'
```

Run SQL | Actions | Last Error: not an error

Custom...	FirstName	LastName	Company	Address	City	State	Country	PostalC...
3	François	Tremblay		1498 ru...	Montréal	QC	Canada	H2G 1A7
7	Astrid	Gruber		Rotentur...	Vienne		Austria	1010
8	Daan	Peeters		Grétrystr...	Brussels		Belgium	1000
14	Mark	Philips	Telus	8210 11...	Edmonton	AB	Canada	T6G 2C7
15	Jennifer	Peterson	Rogers C...	700 W P...	Vancouver	BC	Canada	V6C 1G8
29	Robert	Brown		796 Dun...	Toronto	ON	Canada	M6J 1V1
30	Edward	Francis		230 Elgi...	Ottawa	ON	Canada	K2P 1L7
31	Martha	Silk		194A Ch...	Halifax	NS	Canada	B3S 1C5
32	Aaron	Mitchell		696 Osb...	Winnipeg	MB	Canada	R3L 2B9
33	Ellie	Sullivan		5112 48...	Yellowkn...	NT	Canada	X1A 1N6

Number of Rows Returned: 10

```
SELECT COUNT(CustomerId) AS Total FROM Customer WHERE Country IN ('Austria', 'Belgium', 'Canada')
```

Run SQL | Actions | Last Error: not an error

Total
10

Which customers live in Brazil **or** have a last name that begins with the letter ‘p’ **and** ends in the letter ’s’?

```
SELECT * FROM Customer WHERE Country = 'Brazil' OR (LastName LIKE 'P%' AND LastName LIKE '%s')
```

Run SQL | Actions | Last Error: not an error

CustomerId	FirstName	LastName	Company	Address	City	State	Country
1	Luís	Gonçalves	Embraer -...	Av. Brigad...	São José d...	SP	Brazil
8	Daan	Peeters		Grétrystra...	Brussels		Belgium
10	Eduardo	Martins	Woodstoc...	Rua Dr. Fa...	São Paulo	SP	Brazil
11	Alexandre	Rocha	Banco do ...	Av. Paulist...	São Paulo	SP	Brazil
12	Roberto	Almeida	Riotur	Praça Pio ...	Rio de Jan...	RJ	Brazil
13	Fernanda	Ramos		Qe 7 Bloc...	Brasília	DF	Brazil
14	Mark	Philips	Telus	8210 111 ...	Edmonton	AB	Canada

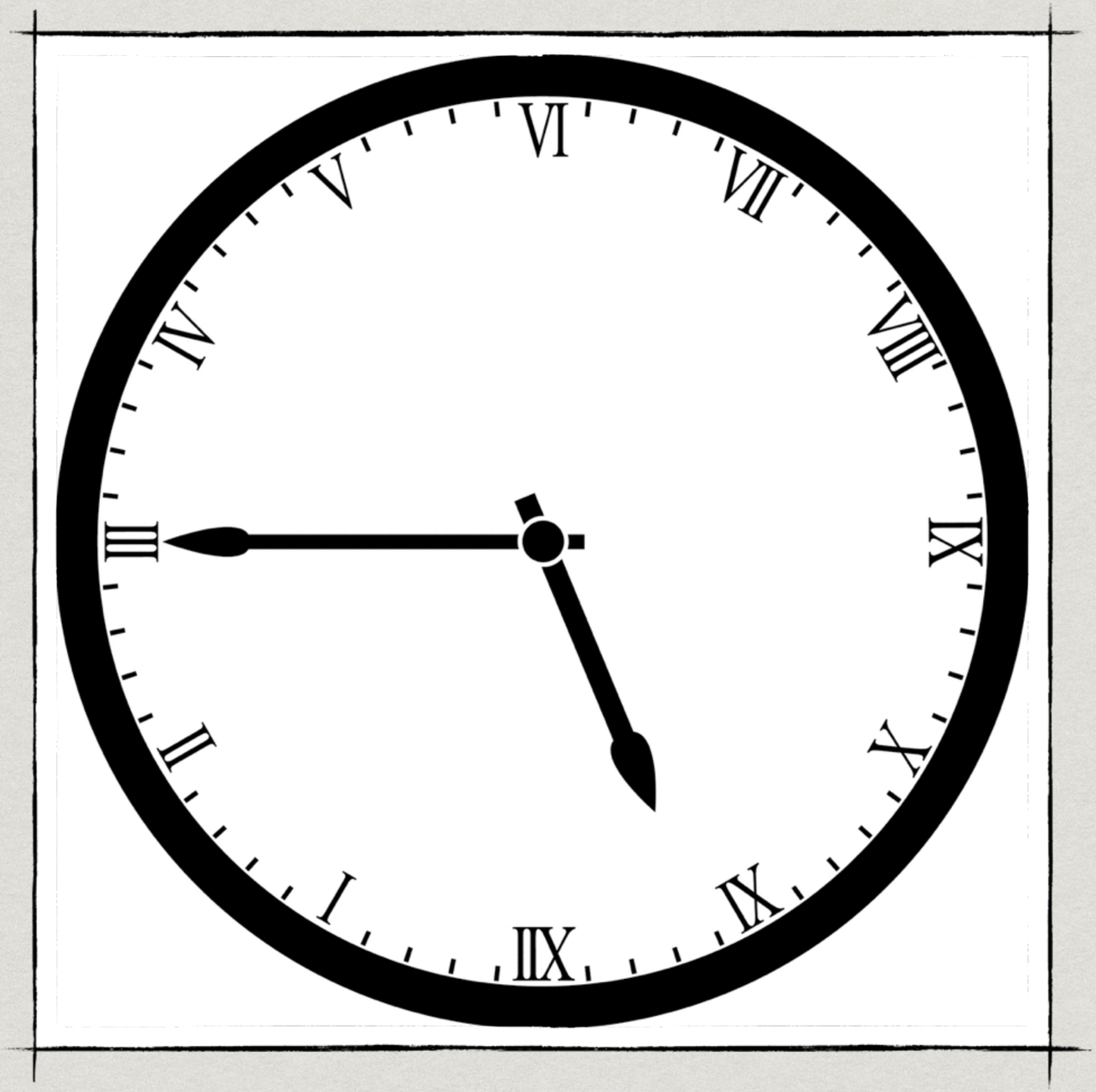
Number of Rows Returned: 7

```
SELECT * FROM Customer WHERE Country = 'Brazil' OR LastName LIKE 'P%s'
```

Run SQL | Actions | Last Error: not an error

CustomerId	FirstName	LastName	Company	Address	City	State	Country
1	Luís	Gonçalves	Embraer -...	Av. Brigad...	São José d...	SP	Brazil
8	Daan	Peeters		Grétrystra...	Brussels		Belgium
10	Eduardo	Martins	Woodstoc...	Rua Dr. Fa...	São Paulo	SP	Brazil
11	Alexandre	Rocha	Banco do ...	Av. Paulist...	São Paulo	SP	Brazil
12	Roberto	Almeida	Riotur	Praça Pio ...	Rio de Jan...	RJ	Brazil
13	Fernanda	Ramos		Qe 7 Bloc...	Brasília	DF	Brazil
14	Mark	Philips	Telus	8210 111 ...	Edmonton	AB	Canada

Number of Rows Returned: 7


VI
VII
VIII
IX
X
XI
XII
I
II
III
IV
V

CASE Statements

- We know type values:
  - 1 is Sent
  - 2 is Received
  - 3 is an Error
- We know flag values:
  - R is Read
  - U is Unread

select * from test

Run SQL | Actions | Last E

id	type	flag
1	1	R
2	1	R
3	1	R
4	1	U
5	2	U
6	2	R
7	2	R

```
select id, case type when 1 then 'Sent' when 2 then 'Received' when 3 then 'Error' else 'Unknown' end as [Call Type],
case flag when 'U' then 'Unread' when 'R' then 'Read' else 'Unknown' end as [Read Status] from test
```

Run SQL | Actions | Last Error: not an error

id	Call Type	Read Status
1	Sent	Read
2	Sent	Read
3	Sent	Read
4	Sent	Unread
5	Received	Unread
6	Received	Read
7	Received	Read

Number of Rows Returned: 7

Parsing Date/Time

- SELECT DATETIME(*column_name*, ‘unixepoch’, [‘localtime’])
- iOS (Epoch: 1/1/2001): Add 978307200 seconds to timestamp
- Android (Standard UNIX Epoch, 1/1/1970, but uses milliseconds): Divide by 1000

TABLE sms

Search | Show All | Add | Duplicate | Edit | Delete

id	timestamp	read	body	locked
1	1290551001726	0	We are getting panda t...	0
2	1290550829974	1	Don't wait for me.	0
3	1290550771692	1	Will you be home for di...	0
4	1290548868055	1	Hi would any of u want...	0
5	1290547793048	1	SCHERERVILLEPD: Toda...	0
6	1290538423418	1	U mean thurs? 1pm =)	0
7	1290520654773	1	What time is dinner to...	0
8	1290520251282	1	So do I... let me know i...	0
9	1290492302264	1	Need Christmas ideas f...	0
10	1290454774582	1	V.CNN BREAKING Stam...	0
11	1290448127472	1	V.CNN BREAKING Ingm...	0
12	1290391197105	1	V.NCAABKBL FINAL Gol...	0
13	1290360442388	1	We're next on the list	0
14	1290289526794	1	The FIRST AID merit ba...	0

```
select id, datetime(timestamp/1000, 'unixepoch') as date from sms
```

Run SQL | Actions | Last Error: not an error

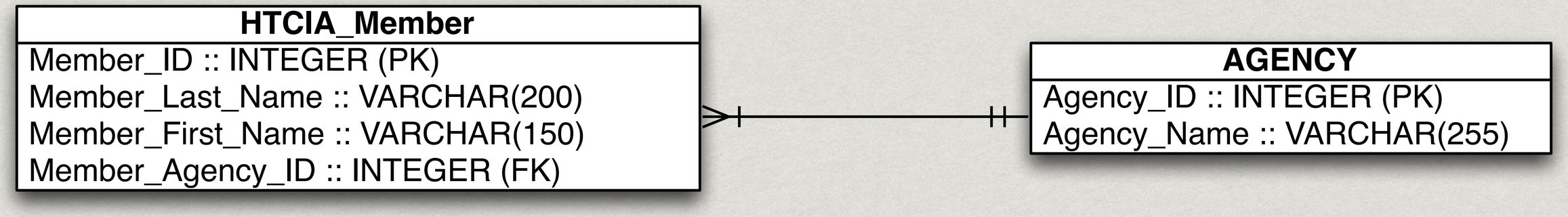
id	date
1	2010-11-23 22:23:21
2	2010-11-23 22:20:29
3	2010-11-23 22:19:31
4	2010-11-23 21:47:48
5	2010-11-23 21:29:53
6	2010-11-23 18:53:43
7	2010-11-23 13:57:34
8	2010-11-23 13:50:51
9	2010-11-23 06:05:02
10	2010-11-22 19:39:34
11	2010-11-22 17:48:47
12	2010-11-22 01:59:57
13	2010-11-21 17:27:22
14	2010-11-20 21:45:26

```
select id, datetime(timestamp/1000, 'unixepoch') as date, case read when 1 then 'Read' else 'Unread' end as Status, body, case locked when 1 then 'True' else 'False' end as Locked from sms
```

Run SQL | Actions | Last Error: not an error

id	date	Status	body	Locked
1	2010-11-23 22:23:21	Unread	We are getting panda then...	False
2	2010-11-23 22:20:29	Read	Don't wait for me.	False
3	2010-11-23 22:19:31	Read	Will you be home for dinner?	False
4	2010-11-23 21:47:48	Read	Hi would any of u want to ...	False
5	2010-11-23 21:29:53	Read	SCHERERVILLEPD: Today S...	False
6	2010-11-23 18:53:43	Read	U mean thurs? 1pm =)	False
7	2010-11-23 13:57:34	Read	What time is dinner tomorr...	False
8	2010-11-23 13:50:51	Read	So do I... let me know if yo...	False
9	2010-11-23 06:05:02	Read	Need Christmas ideas for ...	False
10	2010-11-22 19:39:34	Read	V.CNN BREAKING Stamped...	False
11	2010-11-22 17:48:47	Read	V.CNN BREAKING Ingmar G...	False
12	2010-11-22 01:59:57	Read	V.NCAABKBL FINAL Golden...	False
13	2010-11-21 17:27:22	Read	We're next on the list	False
14	2010-11-20 21:45:26	Read	The FIRST AID merit badge...	False

One-To-Many Relationship



SELECT * FROM AGENCY;

Agency_ID	Agency_Name
1	Lake County HIDTA
2	Aon Corporation
3	ABC, Inc.
4	GSU
5	XYZ Company

SELECT * FROM HTCIA_MEMBER;

Member_ID	Member_Last_Name	Member_First_Name	Member_Agency_ID
1	O’Day	Dan	1
2	Rademacher	Karl	2
3	Shih	Steve	4
4	Doe	John	3
5	Smith	Mike	5
6	Katz	Bob	3

```
SELECT m.Member_Last_Name AS LN, a.Agency_Name
FROM HTCIA_MEMBER m
JOIN AGENCY a
  ON m.Member_Agency_ID = a.Agency_ID
ORDER BY LN;
```

LN	Agency_Name
Doe	ABC, Inc.
Katz	ABC, Inc.
O’Day	Lake County HIDTA
Rademacher	Aon Corporation
Shih	GSU
Smith	XYZ Company

Show each customer’s last name alongside the last name of their respective sales representative

Enter SQL

```
SELECT c.LastName AS [Customer Last Name], e.LastName AS [Employee Last Name] FROM Customer c
JOIN Employee e ON c.SupportRepId = e.EmployeeId
```

Run SQL | Actions | Last Error: not an error

Customer Last Name	Employee Last Name
Gonçalves	Peacock
Köhler	Johnson
Tremblay	Peacock
Hansen	Park
Wichterlová	Park
Holý	Johnson
Gruber	Johnson
Peeters	Park
Nielsen	Park
Martins	Park

Number of Rows Returned: 59

JOIN Statements

- Returns all rows from multiple tables where the JOIN condition is met
- Default JOIN is an INNER JOIN
- SQLite limits JOIN functionality
  - No RIGHT nor FULL OUTER JOIN

INNER JOIN
(default JOIN)

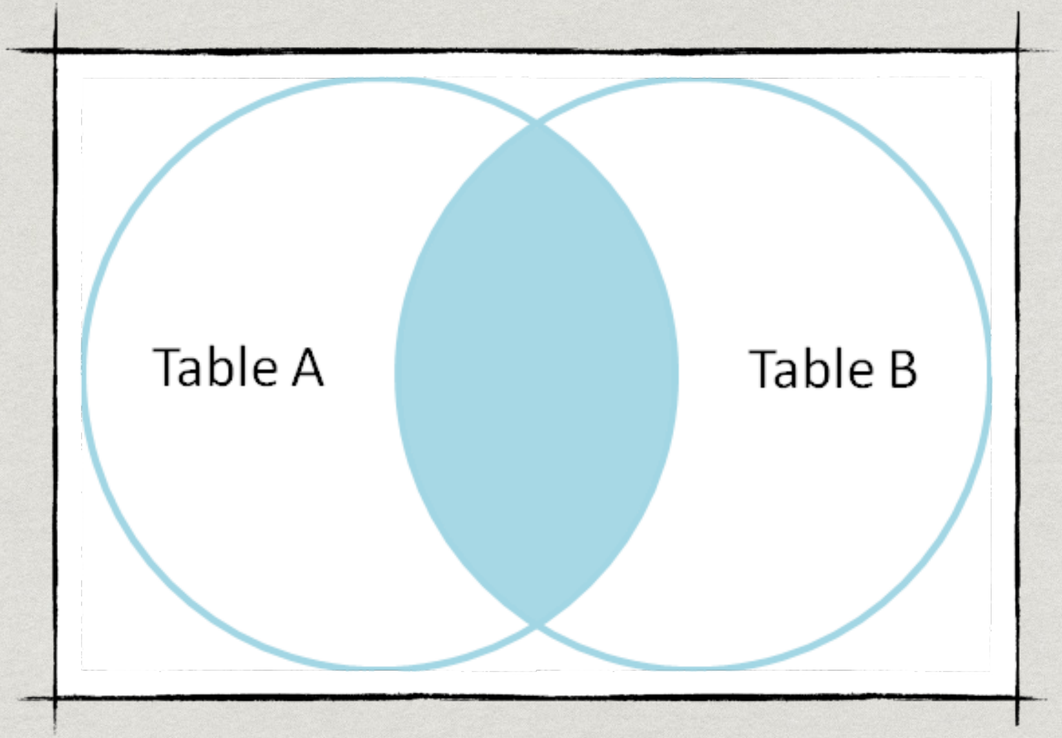


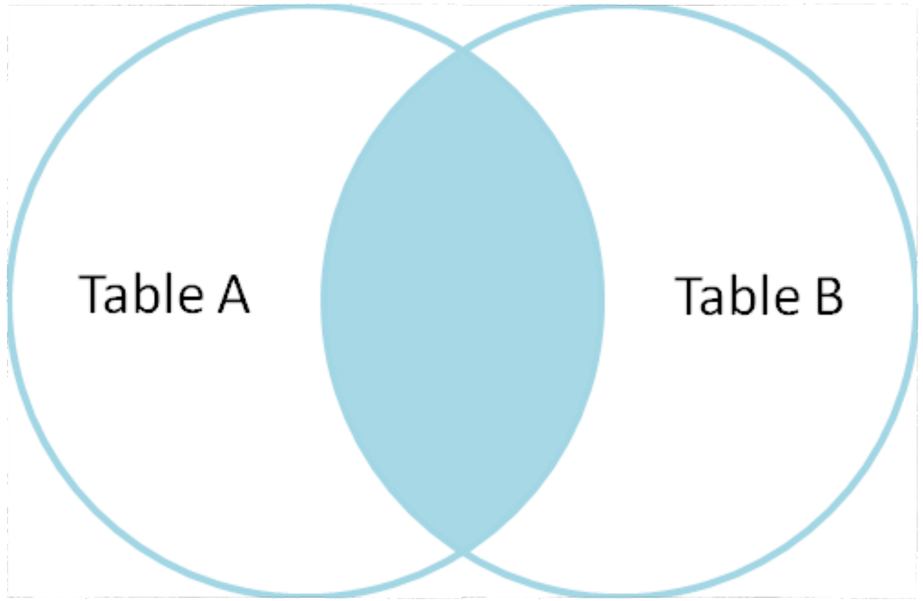
TABLE A Search Show All

id	b_id	name
1	4	HTCIA
2	1	Dan
3	3	The realism of the TV show CSI
4	2	4n6
5		Null Test

TABLE B

id	name
1	is awesome
2	is great
3	is horrible
4	rocks

INNER JOIN
(default JOIN)




```
select A.name, B.name from A join B on A.b_id = B.id
```

Run SQL | Actions | Last Error: not an error

name	name
HTCIA	rocks
Dan	is awesome
The realism of the TV show CSI	is horrible
4n6	is great

Number of Rows Returned: 4

What about the record in table A with NULL in the b_id column?

TABLE A

Search | Show All

id	b_id	name
1	4	HTCIA
2	1	Dan
3	3	The realism of the TV show CSI
4	2	4n6
5		Null Test

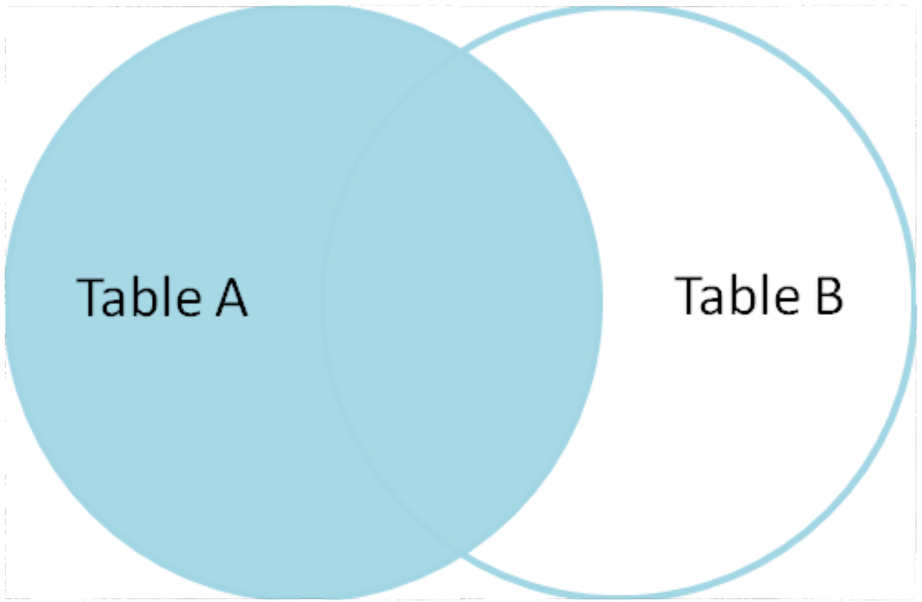
LEFT OUTER JOIN

```
select A.name, B.name from A left outer join B on A.b_id = B.id
```

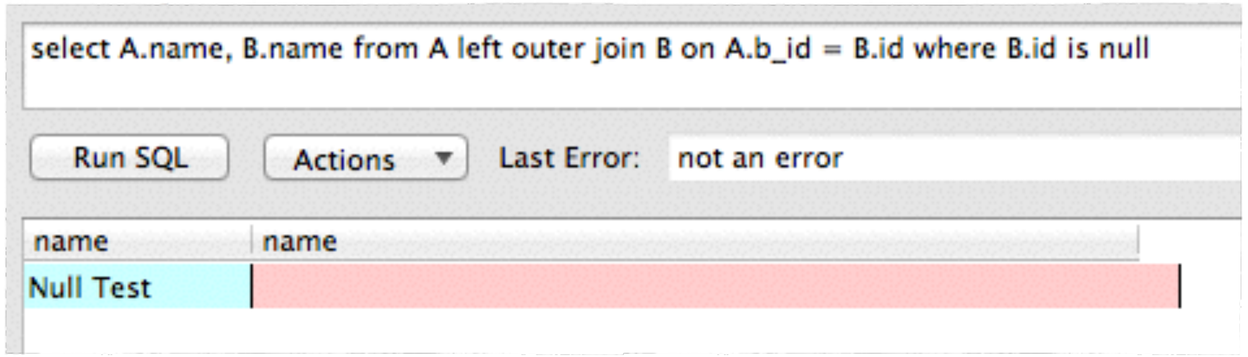
Run SQL | Actions | Last Error: not an error

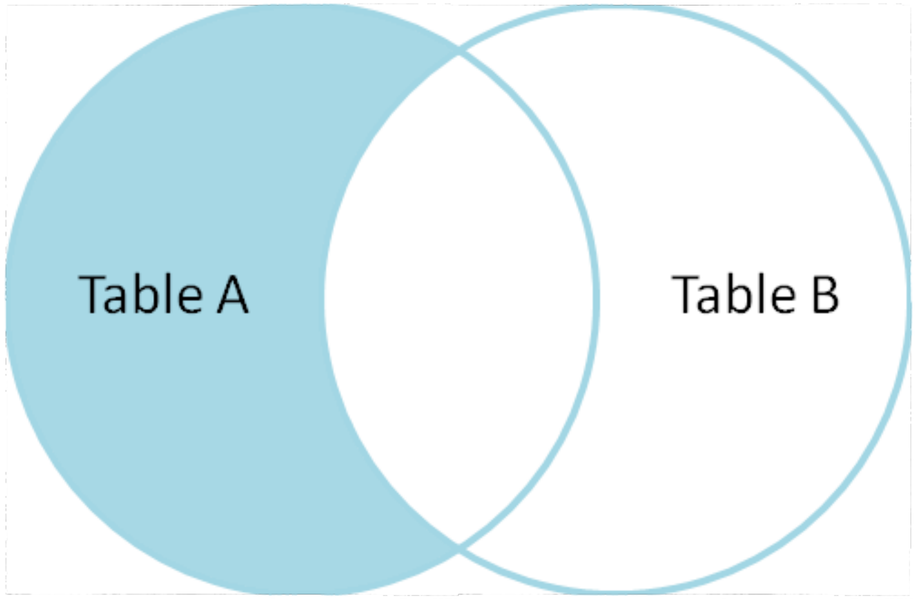
name	name
HTCIA	rocks
Dan	is awesome
The realism of the TV show CSI	is horrible
4n6	is great
Null Test	

Number of Rows Returned: 5



LEFT OUTER JOIN
(exclude records on right)





Which Artist produced the Album entitled “Jagged Little Pill”?

```
select Artist.Name, Album.Title from Album join Artist on Album.ArtistId = Artist.ArtistId
where Album.Title like 'Jagged L%'
```

Run SQL | Actions | Last Error: not an error

Name	Title
Alanis Morissette	Jagged Little Pill

Who does each employee report to?

Recursive JOIN: Joining a table to itself

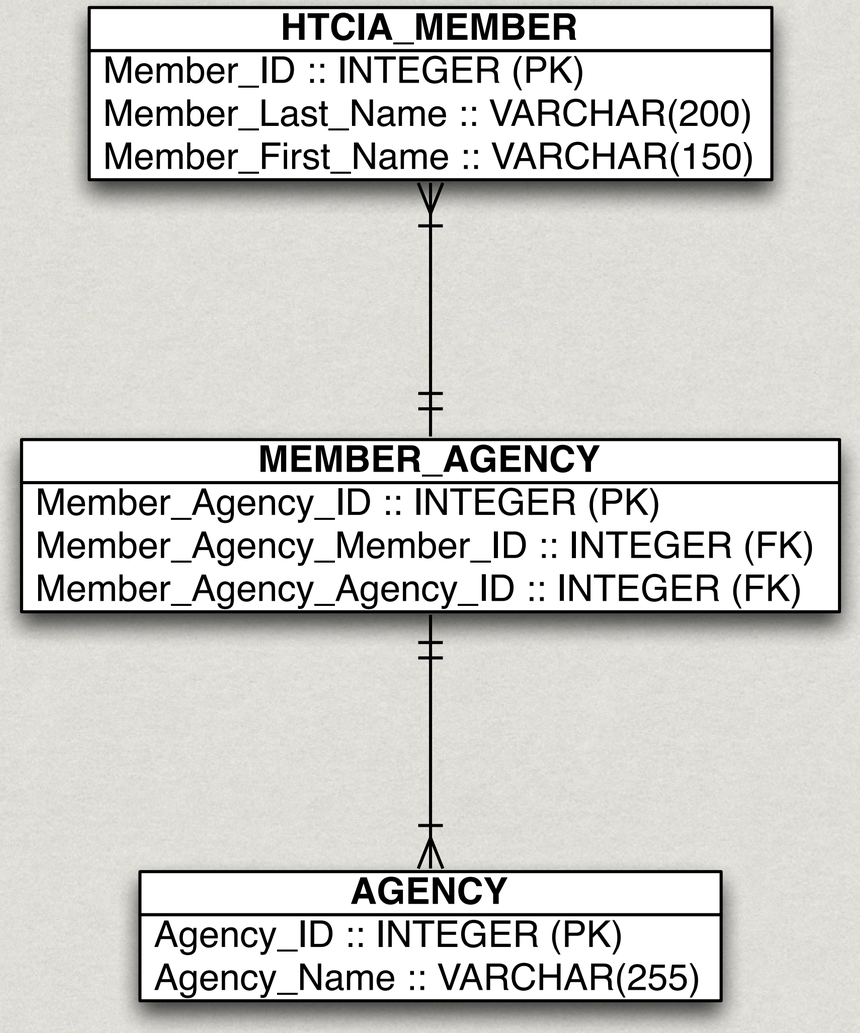
```
select e.LastName || ', ' || e.FirstName as Employee, b.LastName || ', ' || b.FirstName as Boss from Employee e
left outer join Employee b on e.ReportsTo = b.EmployeeId order by Boss, Employee
```

Run SQL | Actions | Last Error: not an error

Employee	Boss
Adams, Andrew	
Edwards, Nancy	Adams, Andrew
Mitchell, Michael	Adams, Andrew
Johnson, Steve	Edwards, Nancy
Park, Margaret	Edwards, Nancy
Peacock, Jane	Edwards, Nancy
Callahan, Laura	Mitchell, Michael
King, Robert	Mitchell, Michael

Number of Rows Returned: 8

Many-To-Many Relationship



HTCIA_MEMBER

Member_ID	Member_Last_Name	Member_First_Name
1	O'Day	Dan

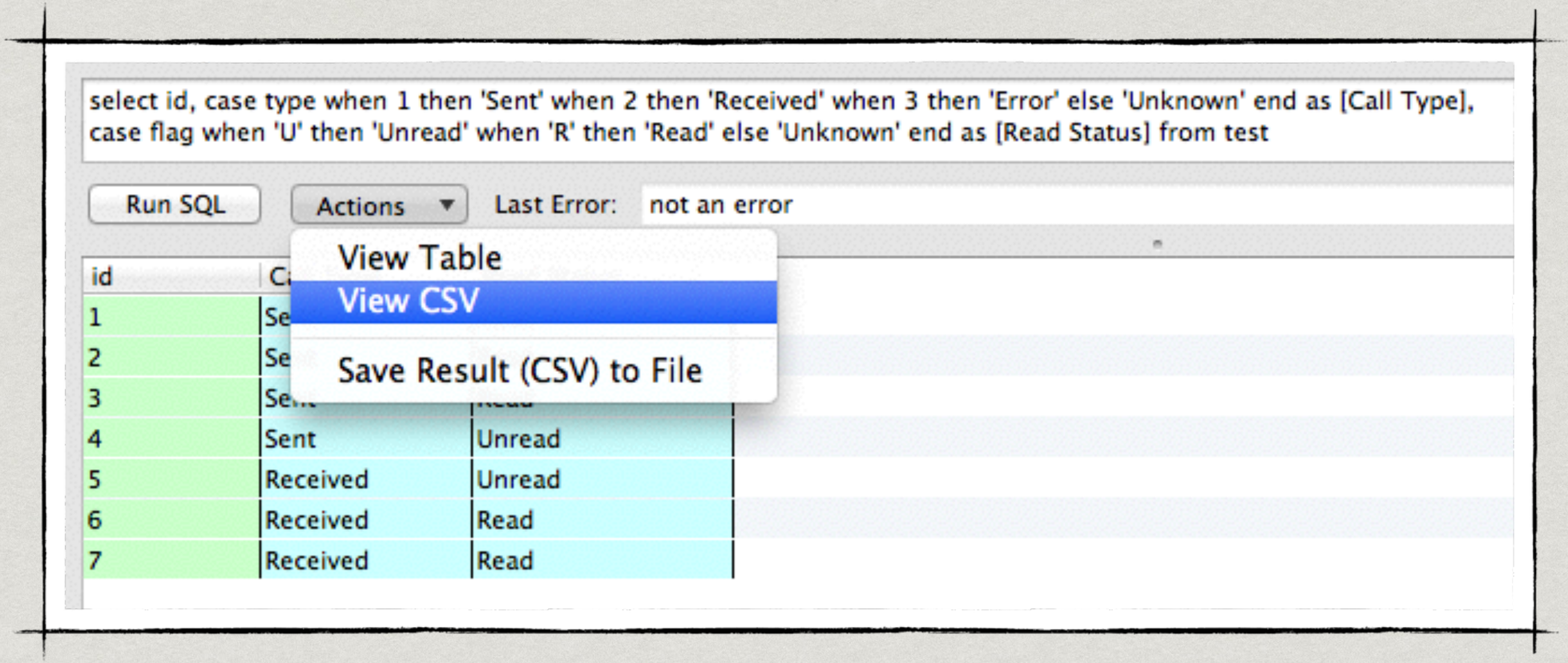
AGENCY

Agency_ID	Agency_Name
1	Lake County HIDTA
2	viaForensics
3	Purdue University

MEMBER_AGENCY

Member_Agency_ID	Member_Agency_Member_ID	Member_Agency_Agency_ID
1	1	3
2	1	1
3	1	2

Exporting Data To A Report



DELETED RECORD RECOVERY

What are the odds?

- iOS
  - Good recovery of allocated db content. Prior to 4S also good recovery of latent/unallocated SQLite files
- Android
  - Good recovery but LOTS of repetition thanks to flash file systems
  - FULL AUTO_VACUUM MODE!!!
- Standard File Systems
  - Good recovery of allocated db content and latent if not overwritten

VACUUM

- Defragmentation plus elimination of *logical* unused ‘space’ in db file
- When a large amount of data is deleted from the db, it leaves behind many ‘free’ pages (and so the db is larger than it needs to be)
- Frequent writes can cause the db to become fragmented
- Contents of entire db are copied into temp db then original is overwritten
  - When overwriting original, a rollback journal or WAL is used as normal

http://www.sqlite.org/lang_vacuum.html

Why Full Auto-Vacuum Mode is a big deal on the Android SMS/MMS DB

- Scenario: bad guy sees police coming, deletes a bunch of texts
  - Deleted records now moved to end of db and file is truncated
  - His buddy sends him an innocent text (“Yo”)
  - DB file begins overwriting deleted records (and media / file system may wipe out entire allocation block depending on how it handles file slack)

Overview

- When information is deleted the pages where it was stored are added to list of free pages so that they can be reused the next time data is inserted
- When an entire page is not deleted, ‘freeblocks’ come to exist (unallocated space in a page below/between allocated records). They have 4-byte headers pointing to the next freeblock.
- If db has AUTO_VACUUM = TRUE then all free pages are moved to end of db file and db is truncated after *every* commit
- Deleted records reside in freeblocks and unallocated space in the logical db file

Approaches

- Carve for databases
  - Easy-to-find header, but no footer
  - File length not stored within file, must check each page in ‘chain’
  - Must parse b-trees and other SQLite specific structures
  - **Focus on finding table b-tree pages in the freelist**
- Forget about recovering entire db’s, carve for individual SQLite records in unallocated space / file slack of latent files
  - OK for text messages and other single-table data, but not for relational data where multiple tables are needed
  - No schema / data structure

Approaches

- Vacuum one copy of the database and compare the differences between it and an un-vacuumed copy
- Most approaches used in mobile device forensics are proprietary to tool manufacturers

Using Python

- Speeds up the process (hand-parsing varints and b-trees is not fun)
  - DISCLAIMER: This is not a Python class (I gave that training at HTCIA a couple years ago)
- Focus on finding unallocated/free table b-tree leaf pages

Find_B-Tree_Leaf_Pages.py

```python
# HTCIA Training: July 15, 2014
#
# credit must be given to Linux Sleuthing (other credits listed in slides):
# http://linuxsleuthing.blogspot.com/2013/09/recovering-data-from-deleted-sqlite.html

from struct import unpack

with open('Chinook_Sqlite_deleted_lots.sqlite', 'rb') as f:
    data = f.read()

pageSize = unpack('>h', data[16:18])[0] # reads 2-byte page size beginning at byte offset 16

pageList = []

for offset in range(0, len(data), pageSize):
    if data[offset] == 13: # table b-tree leaf pages' first byte is \x0D (hex), i.e. 13 (decimal)
        pageList.append(offset)

print(pageList)
```

sqlite — bash — 101×46

```
t0huwav0hu:sqlite danoday$ python3 Find_B-Tree_Leaf_Pages.py
[4096, 5120, 7168, 8192, 10240, 12288, 13312, 14336, 17408, 19456, 24576, 34816, 44032, 45056, 46080,
 49152, 50176, 51200, 53248, 54272, 55296, 56320, 57344, 58368, 63488, 64512, 65536, 68608, 69632, 70
656, 71680, 74752, 75776, 76800, 77824, 78848, 79872, 80896, 81920, 82944, 83968, 89088, 94208, 95232
, 96256, 97280, 98304, 100352, 103424, 104448, 106496, 107520, 108544, 109568, 111616, 112640, 115712
, 116736, 118784, 119808, 121856, 122880, 124928, 125952, 128000, 129024, 132096, 134144, 135168, 137
216, 138240, 139264, 142336, 144384, 145408, 147456, 148480, 151552, 152576, 154624, 155648, 157696,
158720, 161792, 162816, 163840, 164864, 165888, 166912, 169984, 172032, 173056, 174080, 175104, 17612
8, 180224, 181248, 182272, 183296, 184320, 186368, 187392, 191488, 193536, 194560, 195584, 196608, 19
8656, 200704, 202752, 204800, 205824, 206848, 208896, 209920, 211968, 212992, 215040, 217088, 219136,
 220160, 222208, 223232, 224256, 225280, 228352, 230400, 231424, 233472, 234496, 236544, 239616, 2406
40, 242688, 243712, 244736, 246784, 247808, 249856, 251904, 253952, 254976, 256000, 258048, 259072, 2
61120, 264192, 265216, 266240, 268288, 269312, 270336, 272384, 273408, 275456, 278528, 279552, 280576
, 281600, 283648, 284672, 288768, 290816, 292864, 293888, 296960, 405504, 407552, 409600, 410624, 412
672, 413696, 414720, 416768, 417792, 420864, 423936, 425984, 427008, 428032, 429056, 433152, 435200,
436224, 437248, 439296, 441344, 443392, 444416, 445440, 446464, 450560, 452608, 453632, 454656, 45568
0, 456704, 457728, 459776, 460800, 462848, 463872, 464896, 465920, 466944, 467968, 468992, 470016, 47
1040, 472064, 473088, 474112, 475136, 476160, 477184, 478208, 479232, 480256, 481280, 482304, 483328,
 488448, 489472, 490496, 491520, 492544, 493568, 494592, 495616, 496640, 498688, 500736, 501760, 5027
84, 503808, 504832, 505856, 506880, 508928, 509952, 512000, 513024, 514048, 515072, 516096, 517120, 5
18144, 519168, 526336, 527360, 530432, 532480, 534528, 537600, 539648, 542720, 545792, 547840, 549888
, 552960, 556032, 557056, 560128, 562176, 565248, 567296, 569344, 572416, 629760, 632832, 634880, 637
952, 641024, 643072, 650240, 651264, 654336, 658432, 665600, 666624, 671744, 677888, 683008, 689152,
690176, 697344, 703488, 707584, 711680, 718848, 721920, 723968, 728064, 737280, 744448, 746496, 75059
2, 754688, 758784, 762880, 766976, 770048, 774144, 779264, 782336, 788480, 792576, 796672, 800768, 80
7936, 812032, 818176, 823296, 826368, 829440, 836608, 842752, 846848, 847872, 852992, 858112, 862208,
 864256, 870400, 876544, 880640, 1049600, 1052672, 1055744, 1062912, 1065984]
t0huwav0hu:sqlite danoday$
```

Find_B-Tree_Leaf_Pages.py × | Find_Unallocated_Space.py ×

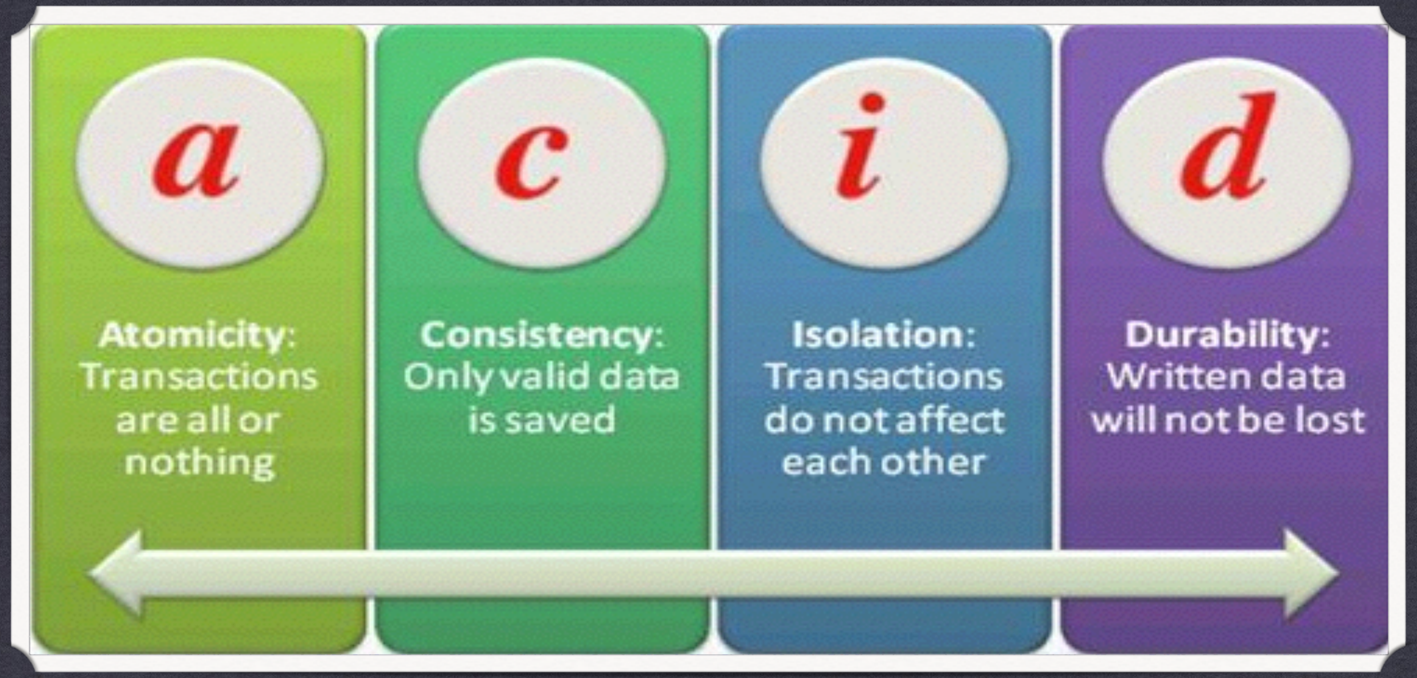
```python
# HTCIA Training: July 15, 2014
#
# credit must be given to Linux Sleuthing (other credits listed in slides):
# http://linuxsleuthing.blogspot.com/2013/09/recovering-data-from-deleted-sqlite.html

from struct import unpack

with open('Chinook_Sqlite_deleted_lots.sqlite', 'rb') as f:
    data = f.read()

pageSize = unpack('>h', data[16:18])[0] # reads 2-byte page size beginning at byte offset 16

pageList = []

# find all table b-tree leaf pages
for offset in range(0, len(data), pageSize):
    if data[offset] == 13: # table b-tree leaf pages' first byte is \x0D (hex), i.e. 13 (decimal)
        pageList.append(offset)

# now find unallocated space
for offset in pageList:
    page = data[offset: offset + pageSize] # advance to next page

    pageHeader = unpack('>bhhhb', page[:8]) # read first 8 bytes big-endian with specified integer format
    pageByte, fbOffset, cellQty, cellOffset, freebytes = pageHeader

    # get unallocated
    start = 8 + cellQty * 2
    end = cellOffset-start
    unalloc = page[start:end]
    print(offset, unalloc, sep=',')

    # get freeblocks, if any
    if fbOffset > 0:
        while fbOffset != 0:
            start, size = unpack('>hh', page[fbOffset: fbOffset + 4])
            freeblock = page[fbOffset: fbOffset + size]
            print(offset, freeblock, sep = ',')
            fbOffset = start
```

Better Yet! Someone did it all for us!

Thanks Mari DeGrazia (arizona4n6@gmail.com)!

http://az4n6.blogspot.com.es/2013/11/python-parser-to-recover-deleted-sqlite.html

- GO TO HTTP://WWW.ARIZONA4N6.COM
  - DOWNLOADS
  - SQLITE DELETED RECORDS PARSER



INSTALL SQLITE DELETED RECORDS PARSER

report.txt

```
Unallocated, Offset 4108 Length 535
Data:


Unallocated, Offset 5130 Length 389
Data:
ã

Unallocated, Offset 7180 Length 194
Data:


Unallocated, Offset 8250 Length 617
Data:


Unallocated, Offset 10250 Length 472
Data:
fi

Unallocated, Offset 12300 Length 303
Data:
âJJ9
M'indexsqlite_autoindex_PlaylistTrack_1PlaylistTrackÅ/  Ç1tablePlaylistPlaylistCREATE TABLE [Playlist]
(
    [PlaylistId] INTEGER  NOT NULL,
    [Name] NVARCHAR(120),
    CONSTRAINT [PK_Playlist] PRIMARY KEY  ([PlaylistId])
)

Unallocated, Offset 13330 Length 877
Data:


Unallocated, Offset 14380 Length 671
Data:


Unallocated, Offset 17422 Length 342
Data:
`

Unallocated, Offset 19472 Length 40
Data:


Unallocated, Offset 24602 Length 116
Data:
3~/'Å3indexIPK_PlaylistTrackPlaylistTrackCREATE UNIQUE INDEX [IPK_PlaylistTrack] ON [Pl

Unallocated, Offset 34842 Length 30
Data:
x='ÅindexIFK_Pla

Unallocated, Offset 44108 Length 62
Data:
ãããããããããããããã
```

Use:

```
-f /home/sanforensics/smsmms.db -o report.tsv\n\
-f /home/sanforensics/smssms.db -r -o report.txt \n"
```

```
danoday$ python sqlparse.py -f Chinook_Sqlite_deleted_lots.sqlite -r -o report.txt
```

SQLITE LAB

Queries

- Determine how much money Leonie Köhler has been charged through invoices for music
- Create a CSV report containing all the track names grouped by artist, showing the album title, media type, and genre for each track
- How many tracks are in the ‘Music Videos’ playlist?
- What tracks were purchased in invoice #19?