Chapter 2
Database Forensics

Nhien-An Le-Khac and Kim-Kwang Raymond Choo

2.1 Introduction to Databases

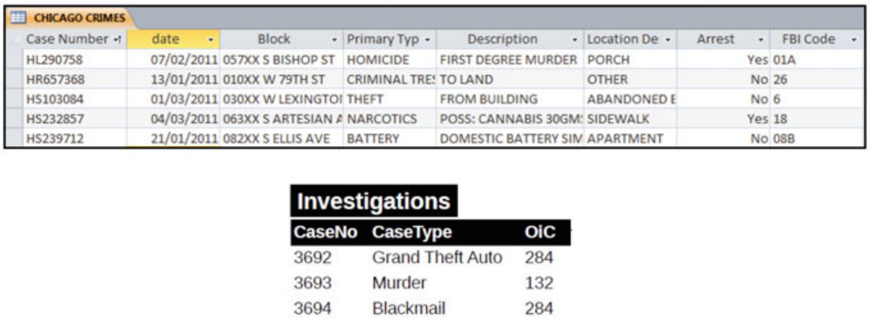
Today, investigators need databases to store and analyze forensic and criminal data (Fig. 2.1). Hence, they should design and build database solutions for investigations.

Besides, most parts of digital forensics today deal with extraction and collection of evidences from databases such as history or cookies information of browsers [1], account information, contact list or call logs of VoIP (Voice over Internet Protocol) application (Fig. 2.2) [2, 3] and social media apps [4, 5]. Therefore with a best understanding of database structure, investigator could retrieve evidences more efficient from variant types of data across electronic devices.

2.1.1 What Is a Database?

Database is not only tables as shown in previous examples. To define what a database is, we should know what is data? So, Eliot said: “Where is the wisdom? Lost in the knowledge. Where is the knowledge? Lost in the information” [6], and Mr. Celko, he said: “Where is the information? Lost in the data” [7]. Maybe the poet Eliot never wrote a computer program in his life but Mr. Celko did. However, we agree with both Eliot and Celko on their points about unofficial definition of knowledge, information and data. Actually, data is a representation of fact, figure or idea. Data normally refers to a collection of numbers, characters, image, etc.

N.-A. Le-Khac and K.-K. R. Choo, *A Practical Hands-on Approach to Database Forensics*, Studies in Big Data 116, https://doi.org/10.1007/978-3-031-16127-8_2



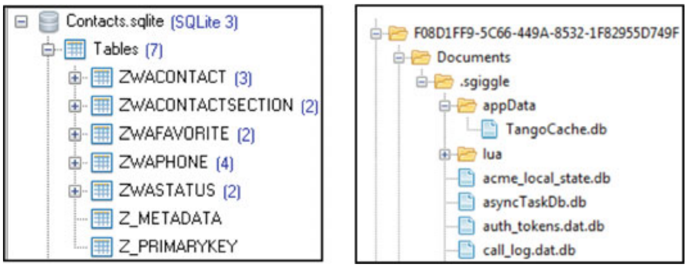
CHICAGO CRIMES

Case Number	date	Block	Primary Typ	Description	Location De	Arrest	FBI Code
HL290758	07/02/2011	057XX S BISHOP ST	HOMICIDE	FIRST DEGREE MURDER	PORCH	Yes	01A
HR657368	13/01/2011	010XX W 79TH ST	CRIMINAL TRE!	TO LAND	OTHER	No	26
HS103084	01/03/2011	030XX W LEXINGTOI	THEFT	FROM BUILDING	ABANDONED E	No	6
HS232857	04/03/2011	063XX S ARTESIAN A	NARCOTICS	POSS: CANNABIS 30GM!	SIDEWALK	Yes	18
HS239712	21/01/2011	082XX S ELLIS AVE	BATTERY	DOMESTIC BATTERY SIV	APARTMENT	No	08B

Investigations

CaseNo	CaseType	OiC
3692	Grand Theft Auto	284
3693	Murder	132
3694	Blackmail	284

Fig. 2.1 Examples of forensic and criminal databases



```
Contacts.sqlite (SQLite 3)
  Tables (7)
    ZWACONTACT (3)
    ZWACONTACTSECTION (2)
    ZWAFAVORITE (2)
    ZWAPHONE (4)
    ZWASTATUS (2)
    Z_METADATA
    Z_PRIMARYKEY
```

```
F08D1FF9-5C66-449A-8532-1F82955D749F
  Documents
    .sgiggle
      appData
        TangoCache.db
      lua
      acme_local_state.db
      asyncTaskDb.db
      auth_tokens.dat.db
      call_log.dat.db
```

Fig. 2.2 VoIP apps databases

Besides, when data has a relational connection, its meaning is an information. For example, Bob, 10 are data. “Bob” is “10” years old is an information. Knowledge is the appropriate collection of information such that it is to be useful. For example, when we have a collection of information such as “Bob is 10 years old”, “Alice is 5 years old”, “Rian is 6 years old”, etc. we can have a rule “80% of children in this room is less than 8 years old”. This rule is a knowledge.

Data is typically processed by human or stored and processed in computer as files. But how these files are organized so that users can access their information easily and effectively at any given time? It could be a database.

There are many way is define a database. Firstly, it is a collection of data, which is structurally stored in a computer system. A database can be considered a collection of related data which are describing the activities of relevant organizations. For instance, a school database contains information about the pupils, school, subjects, and rooms. This database also has the relationships between these objects such as school teaching subjects and the use of rooms for subjects. At the technical point of view, a database could be a tool that stores data, and allows users to create, read, update, and delete the relevant data per request [7].

Databases can be simple or very complicated [8]. We can have not only Universities’ databases as discussed previously but also many examples of databases in everyday uses. For example, a telephone company has a customer database with basic information about their clients such as first name, surname, address, city, postal code and phone number.

An example of large databases is databases existing in banking systems, library catalogues, hotel or airline reservation, social networking, etc. Besides, databases can also be used to store images, audio streams and videos digitally.

Law enforcement has been used databases to store and manage criminal data, which can help to identify and catch. Also, by suing the databases, law enforcement members can effectively gather relevant information to assist in investigations.

2.1.2 Database Management System (DBMS)

A Database Management System (DBMS) [9] is a software system specifically designed to handle and exploit databases at different scales. Most DBMSs are used to manage relational databases. But why we need a DBMS? Today, we live in a world experiencing information explosion. In order to manage efficiently the huge amount of data, we need DBMSs because a DBMS can provide:

- Data independence: A DBMS gives an abstract view of data representation and storage to the application programs.
- Efficient data access: A DBMS normally applies optimal techniques to handle and access data efficiently.
- Data integrity and security: A DBMS implements a variety of mechanisms to guarantee the integrity constraints on the data and to control the access the relevant data.
- Data administration: A DBMS allows the databases to share among several users or different user groups in an efficient way in terms of storing and retrieving.

A DBMS moreover can handles concurrent access and crash recovery as well as to reduce application development time. However, a DBMS has some drawbacks such as its overhead costs.

Some popular DBMSs in the market can be listed as Microsoft SQL Server [10], Oracle database (Oracle DB) [11], PostgreSQL [12] (open source) and MySQL [13] (open source).

There are four important elements of a DBMS: modeling language, data structures, data query language and transactions. A data model is a collection of data description at a high-level that hides details in low-level storage. A data model can be represented by data schemas. A modeling language is used to define the schema of each database stored in a DBMS, according to the data model. As mentioned previously, most DBMSs today are using the relational data model. There are moreover other data model such as: entity-relationship model, relational algebra, hierarchical model, network model and object data model.

Data structure relates to data types, relationships and constraints and the data structures allows DBMS to interact with the data align to their integrity.

A query language is a specialized language that allows to post a queries. A query is a question involving the data stored in a DBMS. For example, some queries can be posted for the University database such as “What is the student name of the student with an ID 1234?”, “How many students are enrolled in course COMP47370?” etc.

A transaction is an execution from the user program in a DBMS. It is also a basic unit of change in the DBMS.

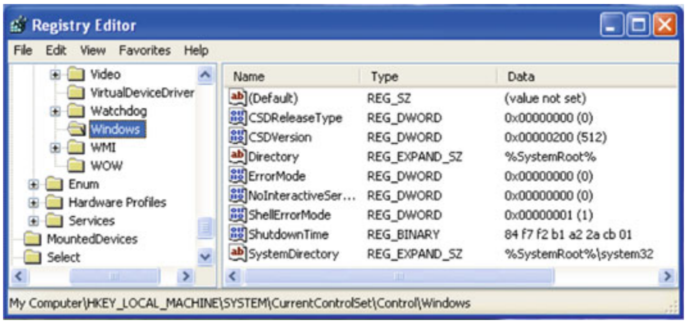
2.1.3 Database Types and Users

Let’s have a look at different types of a database [14]. First, it is a flat file. A flat file is simply file containing text. A flat file could be a database if we add structure in. For example, by separating values with commas, a flat file becomes a .csv file. Although flat files do not provide many services, they are simple and easy to understand. Flat files are also good places to store configuration settings such as .INI files.

In a hierarchical database, records’ relationships form a tree-like structure. One data record logically links to other data records. The structure of hierarchical database is simple and it is restricted to a one-to-many relationship. An example of a hierarchical database is the Windows system registry (Fig. 2.3).

A network database is not a database that is used over the network of computers. It relates to the database structure that consists of a collection of *nodes* connected by *links*. The nodes and links represent objects such as members of a social network. This database structure is uncommon and normally used to express many-to-many relationships [14].

The object oriented database is used to manage objects of varied types such as pictures, video clips, voice and text, etc. The object database management systems (ODBMS) provides special query syntax for accessing and retrieving objects from the database. This database type is popular for Web-based applications.



Name	Type	Data
(Default)	REG_SZ	(value not set)
CSDReleaseType	REG_DWORD	0x00000000 (0)
CSDVersion	REG_DWORD	0x00000200 (512)
Directory	REG_EXPAND_SZ	%SystemRoot%
ErrorMode	REG_DWORD	0x00000000 (0)
NoInteractiveSer...	REG_DWORD	0x00000000 (0)
ShellErrorMode	REG_DWORD	0x00000001 (1)
ShutdownTime	REG_BINARY	84 f7 f2 b1 a2 2a cb 01
SystemDirectory	REG_EXPAND_SZ	%SystemRoot%\system32

My Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Windows

Fig. 2.3 Example of windows registry

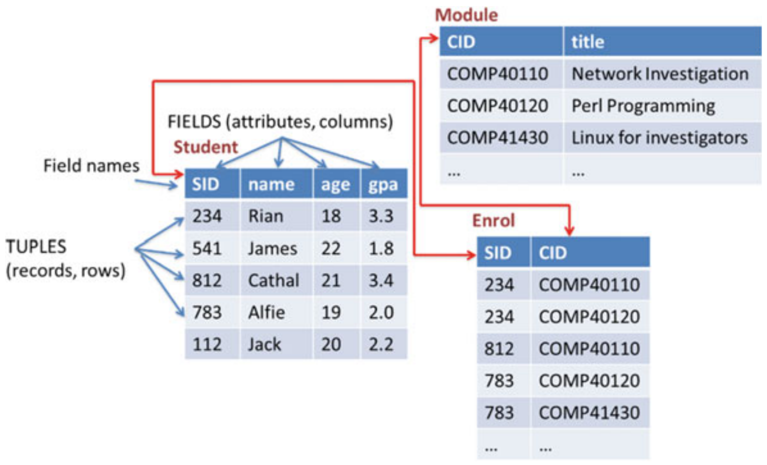
Regarding the database users, there are some popular roles in practice such as a database administrators (DBA), database designers, System Analysts and Application programmers and finally end users.

2.2 Relational Databases

2.2.1 Basic Concepts

In the relational data model, data is organized in tables [8, 9, 14]. A table has a set of records (or rows) and each record can have many attributes/fields (or columns). The simple structure of relational data model makes it easy to understand and easy to exploit with high-level languages such as Structured Query Language (SQL) [15] to query the data. Current popular relational DBMSs include Oracle DB, MS SQL Server, MySQL, DBs, etc.

Figure 2.4 illustrates an example of a relational database, which is a university database to explain the following concepts: *table*, *row* and *column*. Within this database will be three tables named *Student*, *Module* and *Enroll*. Table *Student* for example includes student information such as student id (SID), name, age and grade point average (*gpa*). Each information is described as a field (or an attribute, or a column) of this table. A column is also called an attribute. The set of the validate values of an attribute is called the attribute's domain. It relates to data type of this field. For example, the domain of the attribute age is integer, the domain of the attribute *name* is string and of the attribute *gpa* is real number. Besides, a table is sometime called a relation.



Student

SID	name	age	gpa
234	Rian	18	3.3
541	James	22	1.8
812	Cathal	21	3.4
783	Alfie	19	2.0
112	Jack	20	2.2

Module

CID	title
COMP40110	Network Investigation
COMP40120	Perl Programming
COMP41430	Linux for investigators
...	...

Enrol

SID	CID
234	COMP40110
234	COMP40120
812	COMP40110
783	COMP40120
783	COMP41430
...	...

Fig. 2.4 Example of tables in a relational database

Schema versus Instance

At the model level, there are two concepts related to relational database: schema and instance.

A schema describes the column head for the table that specifies how data to be structured at the logic level. The schema is also called meta-data. In fact, a schema species the name of table, all attribute (field) names, and the domain for each attribute. The schema is normally defined at the setup time of a relational database and it's rarely changed because of high cost for updating of instances.

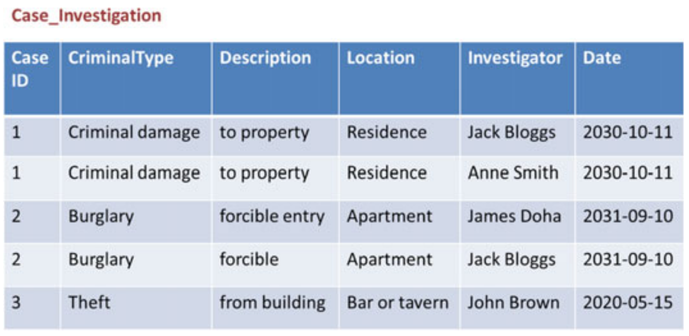
For example, for the Student table in previous example, we have the Student schema as you can see here, the name of this schema is also Student and this schema specifies the name and data type of each attribute.

A record/row in a table is called a table instance, it is also called a tuple. In fact, all tuples in a table have the same number of fields/attributes and there is no two tuples that are identical. That means a table contains a set of unique instances. Besides the order of these instances are not important. The value of an instance can be updated but it always conforms to its schema. In table *Student* for example (Fig. 2.4), there are five instance such as (234, Rian, 18, 2.3), (541, James, 22, 1.8), etc.

Keys and Index

Key is an important concept of relational database. A key is a combination of one or more fields that it can be used to identify records (rows) in a table. A key of a table is defined as a set of one or more attributes (or columns) in this table. For example, a *Student* table can use SID as a key to find students. If a student's ID is known, the relevant student's record can be found in the table. Hence, this table, the key is SID.

A compound/composite key is a key that has more than one attributes (or columns). For example, in table *Case_Investigation* (Fig. 2.5), it might be used the combination of ***CaseID*** and ***Investigator*** as a compound key. Of course there is an assumption that there is no duplication of investigator names in the department. Besides, in the previous example of University's database, it might be used the combination of SID and CID to look up enrollments.



Case_Investigation

Case ID	CriminalType	Description	Location	Investigator	Date
1	Criminal damage	to property	Residence	Jack Bloggs	2030-10-11
1	Criminal damage	to property	Residence	Anne Smith	2030-10-11
2	Burglary	forcible entry	Apartment	James Doha	2031-09-10
2	Burglary	forcible	Apartment	Jack Bloggs	2031-09-10
3	Theft	from building	Bar or tavern	John Brown	2020-05-15

Fig. 2.5 Case_Investigation table

A *superkey* is a set of one or more attributes in a table so that a record is unique. There are no two records that can have the same values. Hence, a *superkey* is also called a unique key. For example, in the *Enroll* table (Fig. 2.4), the SID and CID attributes together form a *superkey* because no two records have exactly the same SID and CID values. Besides, in the *Student* table, the *superkey* is SID.

There are more key definition in relational database. A *primary key* is a superkey and a table can have only one primary key. It should be noted that every records in a database has the own primary key.

Besides, a secondary key is used to lookup records but it does not guarantee the record uniqueness.

Another kind of key is the foreign key, which is used to refer to a primary key of another table.

Index is another important concept of the relational databases. This special database structure allows to find records quicker and easier by using one or more attributes' values. Note that indexes are not the same as keys. For example, as shown in Fig. 2.4, *Student* table holds student information: name, age, and gpa. This table also has the primary key SID. If students do not remember their student IDs, the student name can be used to search a student. If this table is indexed by name, users can quickly locate a student's record in two ways: by student ID or by name.

2.2.2 Database Design

Databases store vast quantities of information. Consider government social security databases for instance. Information is stored on every citizen in the entire nation. This information includes name, address, date of birth, income, tax status et cetera. Searching through all of this information can be very time consuming. If every record needed to be checked in the database it would be a very inefficient system. As mentioned above, there are two ways to make these searches faster: using keys or indexes. For example, the social security system gives each person in the nation a unique identifying number: a social security number. This number acts as a key. It is a unique number that identifies the person in question and can be searched for very quickly in the database. Rather than search for the name for example, Jim Murphy in the database users can search for the unique social security number.

There are three main steps in database design [9]. Gathering the required information, in other words to make sure that all of the necessary information can be stored in the database. These pieces of information will form the fields. This information should also be logically divided into tables. For each of the fields identified designers should select a datatype for this. Finally designers need to create keys or indexes to make retrieval quicker.

2.3 Structured Query Language (SQL)

2.3.1 SQL and SQLite

SQL, or Structured English Query Language [15] is the standard query language for relational DBMSs. There are different versions of SQL: SQL-86 or SQL-1 is the first standard version of SQL. SQL-2 is a revised and expanded version of SQL-86, it is also called SQL-92. The next version is a well-recognized standard SQL-99. The other standards such as SQL-3 have been proposed. However they are not fully endorsed by the industry.

SQL includes statements for data definitions, queries and updates. SQL uses the term table, row and column for the relation model.

For example, the CREATE command in SQL is for the data definition that is used to create schemas, tables, relations, domains, etc. The ALTER command is used to change the definition of a table such as adding or dropping a column, changing a column definition (name of column, data type), and adding or dropping table constraints. The DROP command is used to drop tables, domains or constraints. The SELECT command in SQL is a basic statement for retrieving information from database.

SQLite is public-domain, lightweight relational DBMS [16]. SQLite follows nearly entire SQL-92 standard. SQLite is designed by D. Richard Hipp and the first version was released in August 2000.

SQLite does not require a separate server as other DBMSs such as MySQL, SQOL server of Oracle to operate. That means users do not need to setup/configure a server. It is stability, ease to use as for instance, creating a SQLite database is as simple as opening a file. Moreover, the whole database are stored in a single file that can run on many platforms. SQLite can runs with a limit of hardware resource (CPU, RAM).

Recently, many applications are now storing log information in SQLite 3 database format. These include: Skype; WhatsApp; iOS; Google-Chrome; Mozilla Firefox and many more. So, the ability to extract this log information and handle it in meaningful ways is essential in forensics.

2.3.2 SQLite Basic Commands

The command line interface for SQLite available for all major. There are also some SQLite management commands that start with a dot as follows:

.tables: display all tables in the database
.schema: display the schema of tables created
.quit to exit the SQLite application

There are also some basic SQLite commands as follows[1]:

- Launch SQLite: sqlite3
- Create/open a SQLite file: sqlite3 <filename>
- Create a table: CREATE TABLE command
- Add a record into a table: INSERT INTO command
- Delete a table: DROP TABLE command
- Retrieve information: SELECT command
- Delete a record: DELETE command

Examples:

- Create/open `university` database:

  ```
  sqlite3 university
  ```

- Create `Student` table:

  ```
  CREATE TABLE Student (SID  INTEGER,
                         name  TEXT,
                        age   INTEGER,
                        gpa   REAL);
  ```

- Add a record into `Student` table:

  ```
  INSERT INTO Student VALUES (1, "Rian", 18, 3.3);
  ```

- Retrieve all students who have *gpa* greater than 3.68:

  ```
  SELECT * FROM Student WHERE gpa > 3.68;
  ```

- Delete student James:

  ```
  DELETE FROM Student WHERE name = 'James';
  ```

- Delete `Student` table:

  ```
  DROP TABLE Student.
  ```

2.4 Database Forensics

As mentioned in Sect. 2.1, there are many applications varied from web browsers to mobile apps, running on different operating systems and platforms that are using databases, especially SQLite databases to store relevant information such as searching history or cookies information of browsers, account information or call logs, which are important artifacts for any forensic investigation. This also means

[1] More details of SQLite commands' syntax and examples can be found in: https://www.sqlite.org/docs.html.

that investigators regularly encounter such artifacts from databases. Hence, how to conduct the database forensics from a given application? How to acquire and analysis artifacts?

Figure 2.6 illustrates a flow chart of the suggested forensic method that has three main phases: preparation, acquisition and analysis.

The investigator should explore the application in the first phase (Step 1, Fig. 2.6). The objective is to understand the application i.e. what can the application do? What information might be stored in its databases (e.g. SQLite files)? To achieve this objective, investigators should look for information related to the application, using application websites, open sources, published documents, forensic sites and blogs etc. Without knowledge of what the application does it is very difficult to know what to look for. Hence, in some cases, using the application (Step 1a, Fig. 2.6) is recommended to gain a better understanding of this application.

The second phase includes three steps: identify the database files, collect database information and extract relevant information from databases.

The objective of identifying the database files (Step 2, Fig. 2.6) is to locate all databases used by the application. Investigators can look at open sources on the internet. Databases of existing applications have probably been illustrated in published documents, relevant websites, blogs, etc. Moreover, Profiles, Preferences folders in applications could store location information. Another approach is to install the application in a virtual machine or a simulated platform. Investigators should take snapshots of (database) files existing before and after the installation and try to identify the new (database) files in the after snapshot. The outcome of this step is a list of all database files used by this application.

Next, investigators should collect as much as much information on the databases located (Step 3, Fig. 2.6). The relevant information of each database can be listed as a list of tables, the structure of all tables, the number of entries in all tables, the relationship between tables, index and key for each table, etc. To assist the collection information in this step, a good practice is to use the application by performing its common tasks (Step 3a, Fig. 2.6) then repeat Step 3 to identify which information created or updated, which table entries have been modified and examine these changes.

The final step of this acquisition stage is to launch SQL queries to extract necessary data (artifacts) from all relevant tables (Step 4, Fig. 2.6) of the databases. Note that the storage of the acquired data and devices should follow the policy regarding the chain of custody and jurisdiction.

The acquired data are analyzed in Stage 3 (Step 5, Fig. 2.6). Note that investigators could re-run Step 4 (Fig. 2.6) if the acquired data outcome is un-sufficient for the investigation.

Finally, when the analysis phase is finished, investigators should conduct a review of the process and the actions of the previous steps to validate the investigation process. A formal report is produced in this phase (Step 6, Fig. 2.6) to record all the steps of the investigation, explain the findings etc.

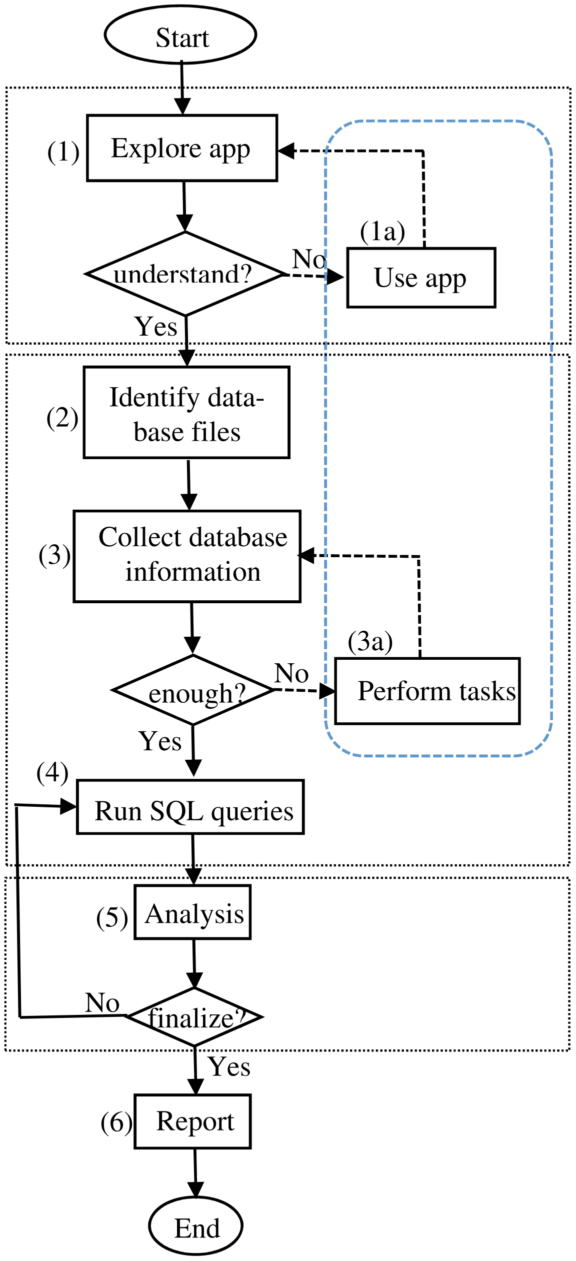


Fig. 2.6 Database forensic flowchart

2.5 Examples

This section presents the forensic investigation of databases from some popular applications as examples, more case studies of database forensics will be introduced in the following chapters.

2.5.1 IOS Database Investigation

Smartphones are widely used today and there are security risks associated with their use such as conducting a digital crime or becoming a victim of one. Hence, smartphones have an important role in the crime investigation [17]. Eventually, in crime scenes the importance of evidence out of the smartphone is increasing with the effect that more and more phones are seized and for a thoroughly investigation offered at the digital department of the police. In the early days the possible information out of a cell phone was phone calls, SMS and contacts. Nowadays the smartphone contains email, web browsing information, Chat, Instant Message, Documents, Photos, Videos, and plenty of applications used daily [18]. The most popular operating systems on smartphones today can be listed as iOS and Android.

iOS (or iPhone OS up to Version 3) is developed by Apple specifically for many Apple's products such as iPhone, iPad, iPod touch, Apple TV, etc. iOS revolutionized the way cell phones have been created. Like on other mobile platforms, most important information on the iOS devices are stored in SQLite databases that are used by both native and third-party applications. The popular databases in iOS are the Address Book, SMS, and Call History databases. Table 2.1 lists common iOS databases in different iOS versions.

Most evidence generated by native applications are located in the *Library* directory. In this directory, the *AddressBook* refers to the information related to the personal contacts that present in the Contact application. There are two databases of

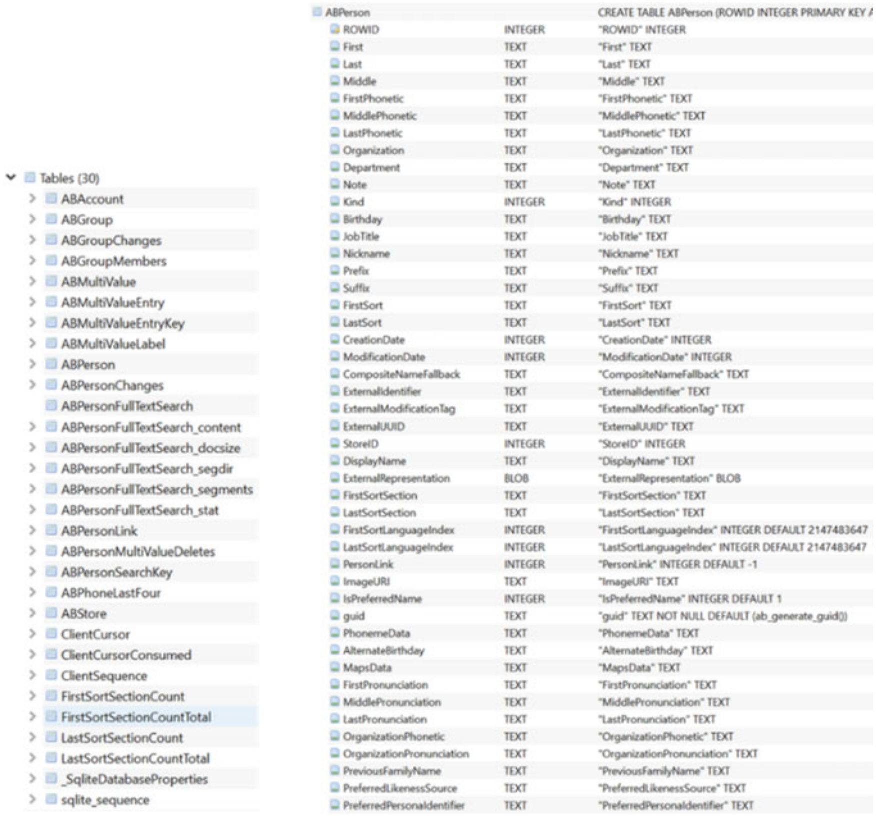
Table 2.1 List of common iOS databases

SQLite databases	iOS12	iOS13	iOS14
Addressbook.sqlite	√	√	√
Calendar.sqlitedb	√	√	√
Callhistory.storedata	√	√	√
consolidated.db	√	√	√
sms.db	√	√	√
notes.sqlite	√	√	√
photos.sqlite	√	√	√
voicemail.db	√	√	√
healthdb.sqlite	√	√	√
tcc.db	√	√	√

interest in this *AddressBook* directory: *AddressBook.sqlitedb* and *AddressBookImages.sqlitedb*. The *AddressBook.sqlitedb* contains the information for each contact such as name, surname, phone number, e-mail address, etc. The number of tables in this database depends on iOS version. On the left of Fig. 2.7 is a list of tables from the contact database *AddressBook.sqlitedb*, which can be found in the *Home/Library/AddressBook* folder of recent iOS versions. The tables of interest are mainly *ABPerson* and *ABMultiValue*. *ABPerson* table (Fig. 2.7, right) with 46 fields contains the name, organization, department, and other general information for each contact. *ABMultivalue* table contains phone numbers, email addresses, website URLs, and other data for the case a contact may have more than one.

Some multivalued entries contain multiple values. For example, an address consists of a city, state, zip code, and country code and these values can be found in *ABMultiValueEntry* table. This table has *parend_id* field, which corresponds to the *rowid* of *ABMultiValue* table.

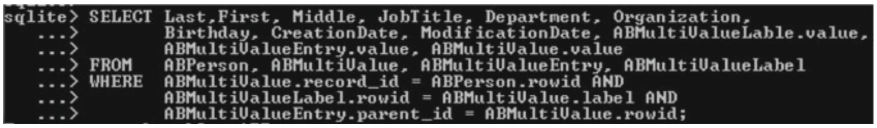
Figure 2.8 illustrates an example of retrieving some important information from tables of *AddressBook.sqlitedb* database.



Tables (30)
- ABAccount
- ABGroup
- ABGroupChanges
- ABGroupMembers
- ABMultiValue
- ABMultiValueEntry
- ABMultiValueEntryKey
- ABMultiValueLabel
- ABPerson
- ABPersonChanges
- ABPersonFullTextSearch
- ABPersonFullTextSearch_content
- ABPersonFullTextSearch_docsize
- ABPersonFullTextSearch_segdir
- ABPersonFullTextSearch_segments
- ABPersonFullTextSearch_stat
- ABPersonLink
- ABPersonMultiValueDeletes
- ABPersonSearchKey
- ABPhoneLastFour
- ABStore
- ClientCursor
- ClientCursorConsumed
- ClientSequence
- FirstSortSectionCount
- FirstSortSectionCountTotal
- LastSortSectionCount
- LastSortSectionCountTotal
- _SqliteDatabaseProperties
- sqlite_sequence

ABPerson		CREATE TABLE ABPerson (ROWID INTEGER PRIMARY KEY A
ROWID	INTEGER	"ROWID" INTEGER
First	TEXT	"First" TEXT
Last	TEXT	"Last" TEXT
Middle	TEXT	"Middle" TEXT
FirstPhonetic	TEXT	"FirstPhonetic" TEXT
MiddlePhonetic	TEXT	"MiddlePhonetic" TEXT
LastPhonetic	TEXT	"LastPhonetic" TEXT
Organization	TEXT	"Organization" TEXT
Department	TEXT	"Department" TEXT
Note	TEXT	"Note" TEXT
Kind	INTEGER	"Kind" INTEGER
Birthday	TEXT	"Birthday" TEXT
JobTitle	TEXT	"JobTitle" TEXT
Nickname	TEXT	"Nickname" TEXT
Prefix	TEXT	"Prefix" TEXT
Suffix	TEXT	"Suffix" TEXT
FirstSort	TEXT	"FirstSort" TEXT
LastSort	TEXT	"LastSort" TEXT
CreationDate	INTEGER	"CreationDate" INTEGER
ModificationDate	INTEGER	"ModificationDate" INTEGER
CompositeNameFallback	TEXT	"CompositeNameFallback" TEXT
ExternalIdentifier	TEXT	"ExternalIdentifier" TEXT
ExternalModificationTag	TEXT	"ExternalModificationTag" TEXT
ExternalUUID	TEXT	"ExternalUUID" TEXT
StoreID	INTEGER	"StoreID" INTEGER
DisplayName	TEXT	"DisplayName" TEXT
ExternalRepresentation	BLOB	"ExternalRepresentation" BLOB
FirstSortSection	TEXT	"FirstSortSection" TEXT
LastSortSection	TEXT	"LastSortSection" TEXT
FirstSortLanguageIndex	INTEGER	"FirstSortLanguageIndex" INTEGER DEFAULT 2147483647
LastSortLanguageIndex	INTEGER	"LastSortLanguageIndex" INTEGER DEFAULT 2147483647
PersonLink	INTEGER	"PersonLink" INTEGER DEFAULT -1
ImageURI	TEXT	"ImageURI" TEXT
IsPreferredName	INTEGER	"IsPreferredName" INTEGER DEFAULT 1
guid	TEXT	"guid" TEXT NOT NULL DEFAULT (ab_generate_guid())
PhonemeData	TEXT	"PhonemeData" TEXT
AlternateBirthday	TEXT	"AlternateBirthday" TEXT
MapsData	TEXT	"MapsData" TEXT
FirstPronunciation	TEXT	"FirstPronunciation" TEXT
MiddlePronunciation	TEXT	"MiddlePronunciation" TEXT
LastPronunciation	TEXT	"LastPronunciation" TEXT
OrganizationPhonetic	TEXT	"OrganizationPhonetic" TEXT
OrganizationPronunciation	TEXT	"OrganizationPronunciation" TEXT
PreviousFamilyName	TEXT	"PreviousFamilyName" TEXT
PreferredLikenessSource	TEXT	"PreferredLikenessSource" TEXT
PreferredPersonaIdentifier	TEXT	"PreferredPersonaIdentifier" TEXT

Fig. 2.7 *AddressBook.sqlitedb* tables (left) and list of fields of *ABPerson* table (right)



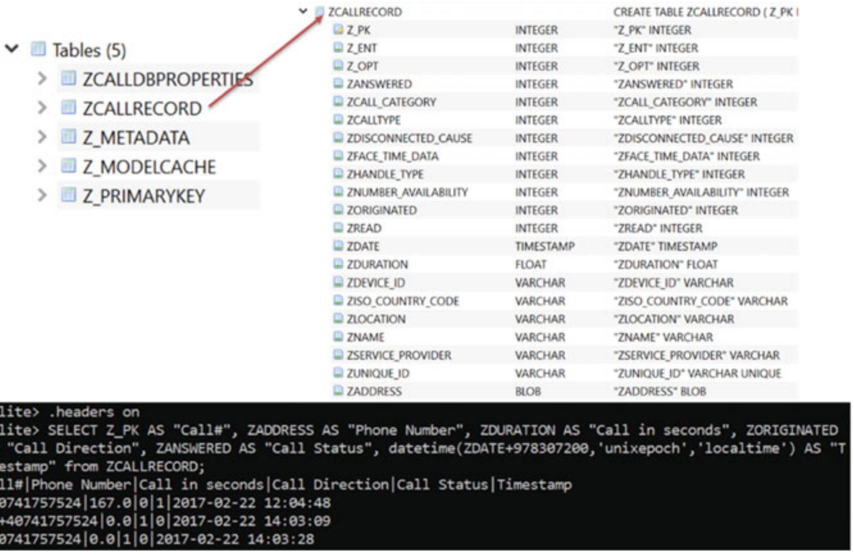
```
sqlite> SELECT Last,First, Middle, JobTitle, Department, Organization,
   ...>        Birthday, CreationDate, ModificationDate, ABMultiValueLable.value,
   ...>        ABMultiValueEntry.value, ABMultiValue.value
   ...> FROM   ABPerson, ABMultiValue, ABMultiValueEntry, ABMultiValueLabel
   ...> WHERE  ABMultiValue.record_id = ABPerson.rowid AND
   ...>        ABMultiValueLabel.rowid = ABMultiValue.label AND
   ...>        ABMultiValueEntry.parent_id = ABMultiValue.rowid;
```

Fig. 2.8 Exploring a suspect's contact information with a SQLite query

AddressBookImages.sqlitedb contains images associated to a given contact. In this database, the important table is *ABFullSizeImage.*

The Call History information is stored in the Call History database: *callhistory.storedata* that contains each of the missed, placed, and received calls, etc. on the device. So, this information helps to find tracks about incoming, outgoing and missed calls with time and date occurred and their duration. Tables contained within this database are listed in Fig. 2.9 (left) where ZCALLRECORD (right) is the important one.

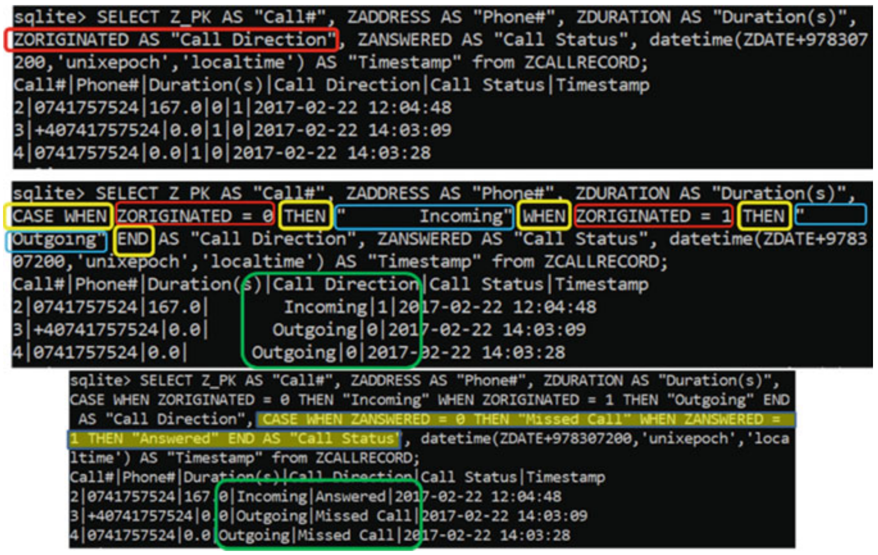
Figure 2.9 (bottom) shows an example of exploring some important fields of ZCALLRECORD table such as the call sequence id, phone number, the duration of the call, the call direction, the call status and the call timestamp. The headers command is used in this example to display the column headers in our query results. The value of its parameter must be either ON or OFF. Note that the call direction showed in this example is a number i.e. 0, 1. To display this information in a more comprehensive format for the query result, ZORIGINATED field is used with the value 0 is "Incoming" and value 1 is "Outgoing" with the SQLite CASE command



ZCALLRECORD		CREATE TABLE ZCALLRECORD (Z_PK I
Z_PK	INTEGER	"Z_PK" INTEGER
Z_ENT	INTEGER	"Z_ENT" INTEGER
Z_OPT	INTEGER	"Z_OPT" INTEGER
ZANSWERED	INTEGER	"ZANSWERED" INTEGER
ZCALL_CATEGORY	INTEGER	"ZCALL_CATEGORY" INTEGER
ZCALLTYPE	INTEGER	"ZCALLTYPE" INTEGER
ZDISCONNECTED_CAUSE	INTEGER	"ZDISCONNECTED_CAUSE" INTEGER
ZFACE_TIME_DATA	INTEGER	"ZFACE_TIME_DATA" INTEGER
ZHANDLE_TYPE	INTEGER	"ZHANDLE_TYPE" INTEGER
ZNUMBER_AVAILABILITY	INTEGER	"ZNUMBER_AVAILABILITY" INTEGER
ZORIGINATED	INTEGER	"ZORIGINATED" INTEGER
ZREAD	INTEGER	"ZREAD" INTEGER
ZDATE	TIMESTAMP	"ZDATE" TIMESTAMP
ZDURATION	FLOAT	"ZDURATION" FLOAT
ZDEVICE_ID	VARCHAR	"ZDEVICE_ID" VARCHAR
ZISO_COUNTRY_CODE	VARCHAR	"ZISO_COUNTRY_CODE" VARCHAR
ZLOCATION	VARCHAR	"ZLOCATION" VARCHAR
ZNAME	VARCHAR	"ZNAME" VARCHAR
ZSERVICE_PROVIDER	VARCHAR	"ZSERVICE_PROVIDER" VARCHAR
ZUNIQUE_ID	VARCHAR	"ZUNIQUE_ID" VARCHAR UNIQUE
ZADDRESS	BLOB	"ZADDRESS" BLOB

```
sqlite> .headers on
sqlite> SELECT Z_PK AS "Call#", ZADDRESS AS "Phone Number", ZDURATION AS "Call in seconds", ZORIGINATED
AS "Call Direction", ZANSWERED AS "Call Status", datetime(ZDATE+978307200,'unixepoch','localtime') AS "T
imestamp" from ZCALLRECORD;
Call#|Phone Number|Call in seconds|Call Direction|Call Status|Timestamp
2|0741757524|167.0|0|1|2017-02-22 12:04:48
3|+40741757524|0.0|1|0|2017-02-22 14:03:09
4|0741757524|0.0|1|0|2017-02-22 14:03:28
```

Fig. 2.9 *callhistory.storedata* tables (left) and list of fields of ZCALLRECORD table (right). SQLite query to explore ZCALLRECORD table (bottom)



```
sqlite> SELECT Z_PK AS "Call#", ZADDRESS AS "Phone#", ZDURATION AS "Duration(s)",
ZORIGINATED AS "Call Direction", ZANSWERED AS "Call Status", datetime(ZDATE+978307
200,'unixepoch','localtime') AS "Timestamp" from ZCALLRECORD;
Call#|Phone#|Duration(s)|Call Direction|Call Status|Timestamp
2|0741757524|167.0|0|1|2017-02-22 12:04:48
3|+40741757524|0.0|1|0|2017-02-22 14:03:09
4|0741757524|0.0|1|0|2017-02-22 14:03:28
```

```
sqlite> SELECT Z_PK AS "Call#", ZADDRESS AS "Phone#", ZDURATION AS "Duration(s)",
CASE WHEN ZORIGINATED = 0 THEN "        Incoming" WHEN ZORIGINATED = 1 THEN "
Outgoing" END AS "Call Direction", ZANSWERED AS "Call Status", datetime(ZDATE+9783
07200,'unixepoch','localtime') AS "Timestamp" from ZCALLRECORD;
Call#|Phone#|Duration(s)|Call Direction|Call Status|Timestamp
2|0741757524|167.0|        Incoming|1|2017-02-22 12:04:48
3|+40741757524|0.0|       Outgoing|0|2017-02-22 14:03:09
4|0741757524|0.0|       Outgoing|0|2017-02-22 14:03:28
```

```
sqlite> SELECT Z_PK AS "Call#", ZADDRESS AS "Phone#", ZDURATION AS "Duration(s)",
CASE WHEN ZORIGINATED = 0 THEN "Incoming" WHEN ZORIGINATED = 1 THEN "Outgoing" END
 AS "Call Direction", CASE WHEN ZANSWERED = 0 THEN "Missed Call" WHEN ZANSWERED =
1 THEN "Answered" END AS "Call Status", datetime(ZDATE+978307200,'unixepoch','loca
ltime') AS "Timestamp" from ZCALLRECORD;
Call#|Phone#|Duration(s)|Call Direction|Call Status|Timestamp
2|0741757524|167.0|Incoming|Answered|2017-02-22 12:04:48
3|+40741757524|0.0|Outgoing|Missed Call|2017-02-22 14:03:09
4|0741757524|0.0|Outgoing|Missed Call|2017-02-22 14:03:28
```

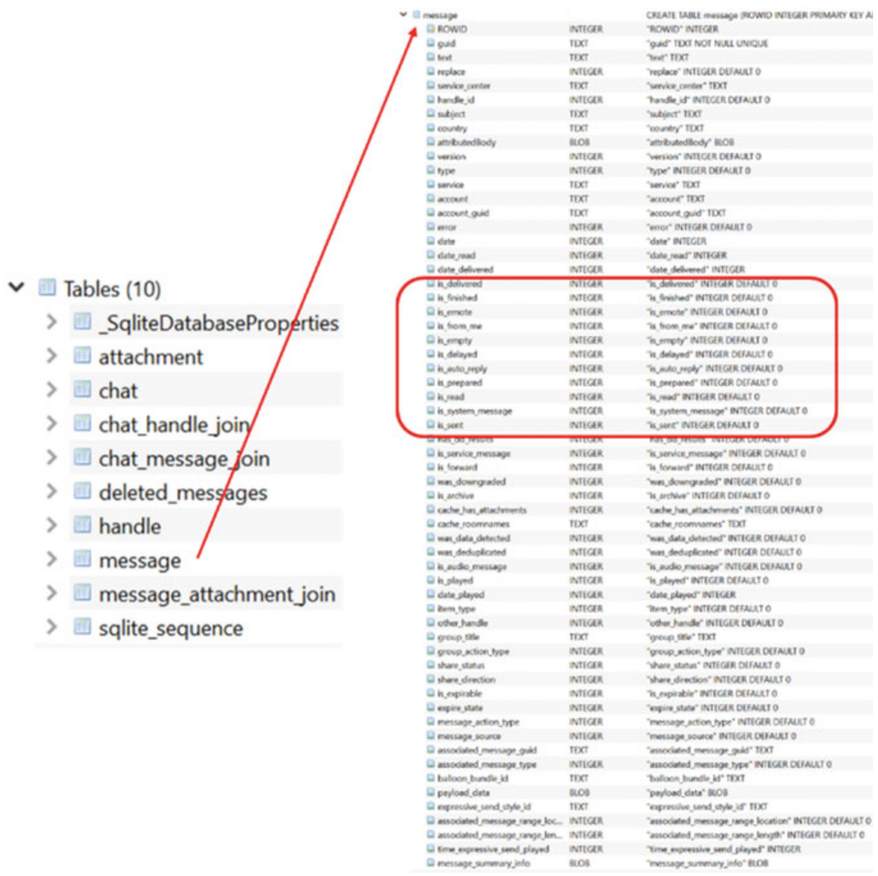
Fig. 2.10 More examples of SQLite queries to explore ZCALLRECORD table

(Fig. 2.10). Similarly, in the last example, the CASE command for the call status ZANSWERED field is used, with value 0—a missed call and 1—a call answered (Fig. 2.10).

Messages are one of the most significant data items to be recovered from iOS devices. The database, which is used to storedsuch information in iOs is *sms.db* that can be found in the *Home/Library/SMS* folder and the important table is *message* table (Fig. 2.11). This table contains the message, date and time, and whether the message was sent or received, etc. For example, this table uses specific fields for each status such as *is_sent*, *is_read*, *is_delievered*, etc. and the value of each specific field is either 0 (default) or 1 (Fig. 2.11).

From the forensic point of views, there are also other interesting sqlite databases in iOS such as *notes*, *photos*, *voicemail*, *healthdb* and *tcc.db* (Table 2.1). *tcc.db* database for example, tcc means Transparency Consent and Control system, it contains all the prevailing settings for privacy controls, including the allow lists which are displayed in the Privacy tab of the Security and Privacy pane. The *healthdb.sqlite* database contains all information collected or received by the Health app. The *voicemail.db* database contains voicemail entries. The *photos.sqlite* database stores a lot of photo asset information including location, date, time, etc.

Note database *NoteStore.sqlite* is also interesting because some users tend to store passwords and other important info in notes. Figure 2.12 illustrates tables in the *NoteStore.sqlite* database, and the important tables are ZNOTE and ZNOTEBODY. Some important fields are ZAUTHOR that contains the author's email address, ZTITLE contains the title of the note and ZCONTENT field of the ZNOTEBODY table has the content of the note. However, most of the notes' contains in the recent iOS version



Tables (10)
- _SqliteDatabaseProperties
- attachment
- chat
- chat_handle_join
- chat_message_join
- deleted_messages
- handle
- message
- message_attachment_join
- sqlite_sequence

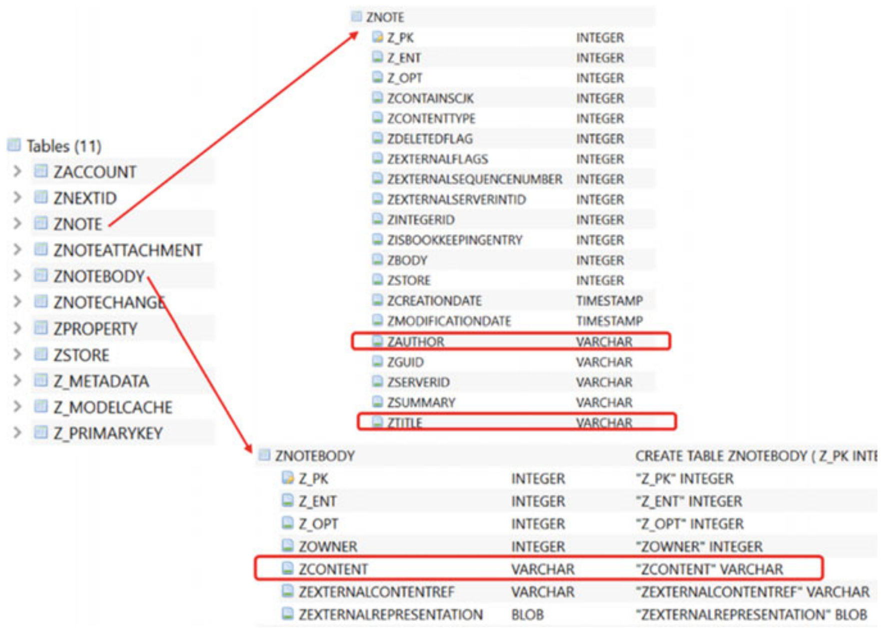
message		CREATE TABLE message (ROWID INTEGER PRIMARY KEY A
ROWID	INTEGER	"ROWID" INTEGER
guid	TEXT	"guid" TEXT NOT NULL UNIQUE
text	TEXT	"text" TEXT
replace	INTEGER	"replace" INTEGER DEFAULT 0
service_center	TEXT	"service_center" TEXT
handle_id	INTEGER	"handle_id" INTEGER DEFAULT 0
subject	TEXT	"subject" TEXT
country	TEXT	"country" TEXT
attributedBody	BLOB	"attributedBody" BLOB
version	INTEGER	"version" INTEGER DEFAULT 0
type	INTEGER	"type" INTEGER DEFAULT 0
service	TEXT	"service" TEXT
account	TEXT	"account" TEXT
account_guid	TEXT	"account_guid" TEXT
error	INTEGER	"error" INTEGER DEFAULT 0
date	INTEGER	"date" INTEGER
date_read	INTEGER	"date_read" INTEGER
date_delivered	INTEGER	"date_delivered" INTEGER
is_delivered	INTEGER	"is_delivered" INTEGER DEFAULT 0
is_finished	INTEGER	"is_finished" INTEGER DEFAULT 0
is_emote	INTEGER	"is_emote" INTEGER DEFAULT 0
is_from_me	INTEGER	"is_from_me" INTEGER DEFAULT 0
is_empty	INTEGER	"is_empty" INTEGER DEFAULT 0
is_delayed	INTEGER	"is_delayed" INTEGER DEFAULT 0
is_auto_reply	INTEGER	"is_auto_reply" INTEGER DEFAULT 0
is_prepared	INTEGER	"is_prepared" INTEGER DEFAULT 0
is_read	INTEGER	"is_read" INTEGER DEFAULT 0
is_system_message	INTEGER	"is_system_message" INTEGER DEFAULT 0
is_sent	INTEGER	"is_sent" INTEGER DEFAULT 0
has_dd_results	INTEGER	"has_dd_results" INTEGER DEFAULT 0
is_service_message	INTEGER	"is_service_message" INTEGER DEFAULT 0
is_forward	INTEGER	"is_forward" INTEGER DEFAULT 0
was_downgraded	INTEGER	"was_downgraded" INTEGER DEFAULT 0
is_archive	INTEGER	"is_archive" INTEGER DEFAULT 0
cache_has_attachments	INTEGER	"cache_has_attachments" INTEGER DEFAULT 0
cache_roomnames	TEXT	"cache_roomnames" TEXT
was_data_detected	INTEGER	"was_data_detected" INTEGER DEFAULT 0
was_deduplicated	INTEGER	"was_deduplicated" INTEGER DEFAULT 0
is_audio_message	INTEGER	"is_audio_message" INTEGER DEFAULT 0
is_played	INTEGER	"is_played" INTEGER DEFAULT 0
date_played	INTEGER	"date_played" INTEGER
item_type	INTEGER	"item_type" INTEGER DEFAULT 0
other_handle	INTEGER	"other_handle" INTEGER DEFAULT 0
group_title	TEXT	"group_title" TEXT
group_action_type	INTEGER	"group_action_type" INTEGER DEFAULT 0
share_status	INTEGER	"share_status" INTEGER DEFAULT 0
share_direction	INTEGER	"share_direction" INTEGER DEFAULT 0
is_expirable	INTEGER	"is_expirable" INTEGER DEFAULT 0
expire_state	INTEGER	"expire_state" INTEGER DEFAULT 0
message_action_type	INTEGER	"message_action_type" INTEGER DEFAULT 0
message_source	INTEGER	"message_source" INTEGER DEFAULT 0
associated_message_guid	TEXT	"associated_message_guid" TEXT
associated_message_type	INTEGER	"associated_message_type" INTEGER DEFAULT 0
balloon_bundle_id	TEXT	"balloon_bundle_id" TEXT
payload_data	BLOB	"payload_data" BLOB
expressive_send_style_id	TEXT	"expressive_send_style_id" TEXT
associated_message_range_loc...	INTEGER	"associated_message_range_location" INTEGER DEFAULT 0
associated_message_range_len...	INTEGER	"associated_message_range_length" INTEGER DEFAULT 0
time_expressive_send_played	INTEGER	"time_expressive_send_played" INTEGER
message_summary_info	BLOB	"message_summary_info" BLOB

Fig. 2.11 *sms.db* tables (left) and list of fields of *message* table (right)

can be found in *NoteStore.sqlite* database that can be synchronized with the iCloud account and the note content is normally encrypted.

Figure 2.13 lists tables in the *Photos.sqlite* database, and an important table is ZGENERICASSET. There are many fields in this ZGENERICASSET table. Some important fields are ZDATECREATED, which is the timestamp of the image, ZLATITUDE and ZLONGITUDE, which are the location where the photo is taken, ZFILENAME and so on. Figure 2.13 also has an example of SQLite query to show all photo names with the timestamp and location.

Finally, Fig. 2.14 shows tables of *voicemail.db* database with an example of a SQLite query to get a list of voicemails sorted by date.



Tables (11)
- ZACCOUNT
- ZNEXTID
- ZNOTE
- ZNOTEATTACHMENT
- ZNOTEBODY
- ZNOTECHANGE
- ZPROPERTY
- ZSTORE
- Z_METADATA
- Z_MODELCACHE
- Z_PRIMARYKEY

ZNOTE	
Z_PK	INTEGER
Z_ENT	INTEGER
Z_OPT	INTEGER
ZCONTAINSCJK	INTEGER
ZCONTENTTYPE	INTEGER
ZDELETEDFLAG	INTEGER
ZEXTERNALFLAGS	INTEGER
ZEXTERNALSEQUENCENUMBER	INTEGER
ZEXTERNALSERVERINTID	INTEGER
ZINTEGERID	INTEGER
ZISBOOKKEEPINGENTRY	INTEGER
ZBODY	INTEGER
ZSTORE	INTEGER
ZCREATIONDATE	TIMESTAMP
ZMODIFICATIONDATE	TIMESTAMP
ZAUTHOR	VARCHAR
ZGUID	VARCHAR
ZSERVERID	VARCHAR
ZSUMMARY	VARCHAR
ZTITLE	VARCHAR

ZNOTEBODY		CREATE TABLE ZNOTEBODY (Z_PK INT
Z_PK	INTEGER	"Z_PK" INTEGER
Z_ENT	INTEGER	"Z_ENT" INTEGER
Z_OPT	INTEGER	"Z_OPT" INTEGER
ZOWNER	INTEGER	"ZOWNER" INTEGER
ZCONTENT	VARCHAR	"ZCONTENT" VARCHAR
ZEXTERNALCONTENTREF	VARCHAR	"ZEXTERNALCONTENTREF" VARCHAR
ZEXTERNALREPRESENTATION	BLOB	"ZEXTERNALREPRESENTATION" BLOB

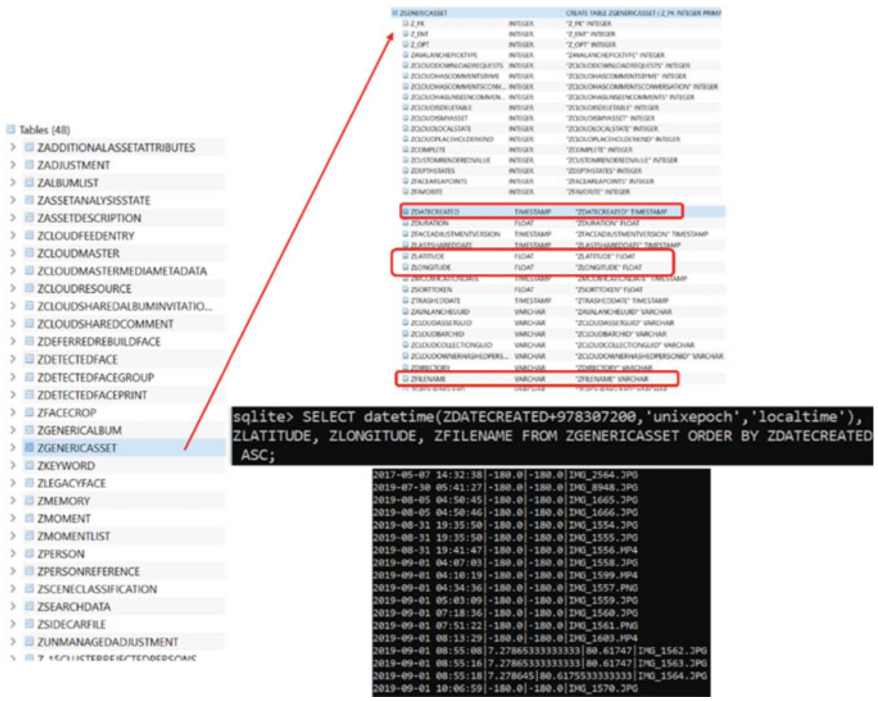
Fig. 2.12 *NoteStore.sqlite* tables (left) and list of fields of ZNOTE table (right) and ZNOTEBODY table (bottom)

2.5.2 *WhatsApp Database Forensics*

WhatsApp is an Instant Messaging and Voice over IP (VoIP) app developed by Brian Acton and Jan Koum. Users can use WhatsApp to exchange instant messages, images, video and audio media messages. Today, there are around two billion WhatsApp registered users active monthly. Hence it becomes an important source of forensic investigation [19].

WhatsApp is available on different devices such as iPhone, iPad, Android phones and tablets and also with different platforms such as Windows, Mac, iOS, Android, etc.

On iOS systems, the manual file system analysis initially shown that the WhatsApp files are in the directory *group.net.whatsapp.WhatsApp.shared/*. The activity and contact information are stored in SQLite database files. For example, the communication activity is stored in *ChatStorage.sqlite* database and the contact information is stored in the *ContactsV2.sqlite* database. There are other artefacts in *net.whatsapp.WhatsApp/Documents/* folder. The *ChatStorage.sqlite* database (Fig. 2.15) might be of interest in an investigation. Its most significant tables are ZWACHATSESSION, ZWAGROUPINFO, ZWAGROUPMEMBER, ZWAMEDIAITEM and ZWAMESSAGE. ZWACHATSESSION table contains a list of unique conversations started with different contacts or groups. ZWAGROUPINFO table stores a list of group conversations. ZWAGROUPMEMBER table has a list of



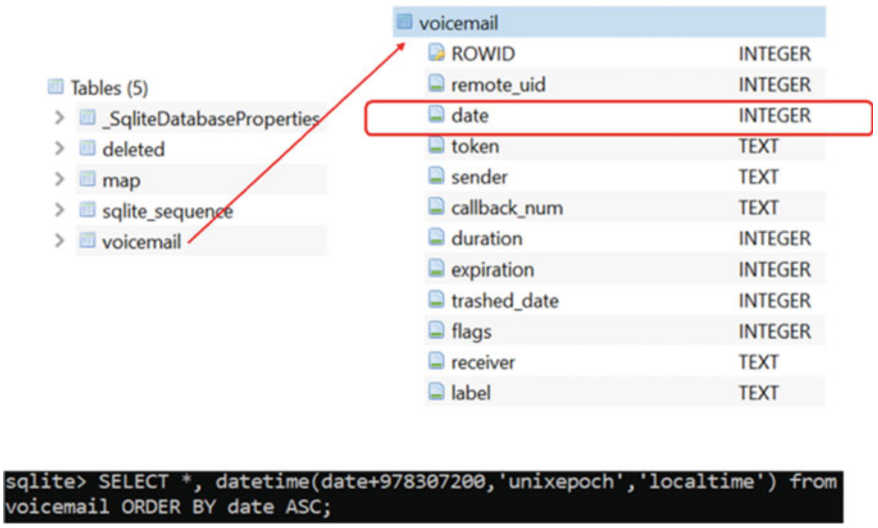
Tables (48)
- ZADDITIONALASSETATTRIBUTES
- ZADJUSTMENT
- ZALBUMLIST
- ZASSETANALYSISSTATE
- ZASSETDESCRIPTION
- ZCLOUDFEEDENTRY
- ZCLOUDMASTER
- ZCLOUDMASTERMEDIAMETADATA
- ZCLOUDRESOURCE
- ZCLOUDSHAREDALBUMINVITATIO...
- ZCLOUDSHAREDCOMMENT
- ZDEFERREDREBUILDFACE
- ZDETECTEDFACE
- ZDETECTEDFACEGROUP
- ZDETECTEDFACEPRINT
- ZFACECROP
- ZGENERICALBUM
- ZGENERICASSET
- ZKEYWORD
- ZLEGACYFACE
- ZMEMORY
- ZMOMENT
- ZMOMENTLIST
- ZPERSON
- ZPERSONREFERENCE
- ZSCENECLASSIFICATION
- ZSEARCHDATA
- ZSIDECARFILE
- ZUNMANAGEDADJUSTMENT

ZGENERICASSET		CREATE TABLE ZGENERICASSET (Z_PK INTEGER PRIM
Z_PK	INTEGER	"Z_PK" INTEGER
Z_ENT	INTEGER	"Z_ENT" INTEGER
Z_OPT	INTEGER	"Z_OPT" INTEGER
ZAVALANCHEPICKTYPE	INTEGER	"ZAVALANCHEPICKTYPE" INTEGER
ZCLOUDDOWNLOADREQUESTS	INTEGER	"ZCLOUDDOWNLOADREQUESTS" INTEGER
ZCLOUDHASCOMMENTSBYME	INTEGER	"ZCLOUDHASCOMMENTSBYME" INTEGER
ZCLOUDHASCOMMENTSCONV...	INTEGER	"ZCLOUDHASCOMMENTSCONVERSATION" INTEGER
ZCLOUDHASUNSEENCOMMEN...	INTEGER	"ZCLOUDHASUNSEENCOMMENTS" INTEGER
ZCLOUDISDELETABLE	INTEGER	"ZCLOUDISDELETABLE" INTEGER
ZCLOUDISMYASSET	INTEGER	"ZCLOUDISMYASSET" INTEGER
ZCLOUDLOCALSTATE	INTEGER	"ZCLOUDLOCALSTATE" INTEGER
ZCLOUDPLACEHOLDERKIND	INTEGER	"ZCLOUDPLACEHOLDERKIND" INTEGER
ZCOMPLETE	INTEGER	"ZCOMPLETE" INTEGER
ZCUSTOMRENDEREDVALUE	INTEGER	"ZCUSTOMRENDEREDVALUE" INTEGER
ZDEPTHSTATES	INTEGER	"ZDEPTHSTATES" INTEGER
ZFACEAREAPOINTS	INTEGER	"ZFACEAREAPOINTS" INTEGER
ZFAVORITE	INTEGER	"ZFAVORITE" INTEGER
ZDATECREATED	TIMESTAMP	"ZDATECREATED" TIMESTAMP
ZDURATION	FLOAT	"ZDURATION" FLOAT
ZFACEADJUSTMENTVERSION	TIMESTAMP	"ZFACEADJUSTMENTVERSION" TIMESTAMP
ZLASTSHAREDDATE	TIMESTAMP	"ZLASTSHAREDDATE" TIMESTAMP
ZLATITUDE	FLOAT	"ZLATITUDE" FLOAT
ZLONGITUDE	FLOAT	"ZLONGITUDE" FLOAT
ZMODIFICATIONDATE	TIMESTAMP	"ZMODIFICATIONDATE" TIMESTAMP
ZSORTTOKEN	FLOAT	"ZSORTTOKEN" FLOAT
ZTRASHEDDATE	TIMESTAMP	"ZTRASHEDDATE" TIMESTAMP
ZAVALANCHEUUID	VARCHAR	"ZAVALANCHEUUID" VARCHAR
ZCLOUDASSETGUID	VARCHAR	"ZCLOUDASSETGUID" VARCHAR
ZCLOUDBATCHID	VARCHAR	"ZCLOUDBATCHID" VARCHAR
ZCLOUDCOLLECTIONGUID	VARCHAR	"ZCLOUDCOLLECTIONGUID" VARCHAR
ZCLOUDOWNERHASHEDPERS...	VARCHAR	"ZCLOUDOWNERHASHEDPERSONID" VARCHAR
ZDIRECTORY	VARCHAR	"ZDIRECTORY" VARCHAR
ZFILENAME	VARCHAR	"ZFILENAME" VARCHAR

```
sqlite> SELECT datetime(ZDATECREATED+978307200,'unixepoch','localtime'),
ZLATITUDE, ZLONGITUDE, ZFILENAME FROM ZGENERICASSET ORDER BY ZDATECREATED
 ASC;
```

```
2017-05-07 14:32:38|-180.0|-180.0|IMG_2564.JPG
2019-07-30 05:41:27|-180.0|-180.0|IMG_8948.JPG
2019-08-05 04:50:45|-180.0|-180.0|IMG_1665.JPG
2019-08-05 04:50:46|-180.0|-180.0|IMG_1666.JPG
2019-08-31 19:35:50|-180.0|-180.0|IMG_1554.JPG
2019-08-31 19:35:50|-180.0|-180.0|IMG_1555.JPG
2019-08-31 19:41:47|-180.0|-180.0|IMG_1556.MP4
2019-09-01 04:07:03|-180.0|-180.0|IMG_1558.JPG
2019-09-01 04:10:19|-180.0|-180.0|IMG_1599.MP4
2019-09-01 04:34:36|-180.0|-180.0|IMG_1557.PNG
2019-09-01 05:03:09|-180.0|-180.0|IMG_1559.JPG
2019-09-01 07:18:36|-180.0|-180.0|IMG_1560.JPG
2019-09-01 07:51:22|-180.0|-180.0|IMG_1561.PNG
2019-09-01 08:13:29|-180.0|-180.0|IMG_1603.MP4
2019-09-01 08:55:08|7.27865333333333|80.61747|IMG_1562.JPG
2019-09-01 08:55:16|7.27865333333333|80.61747|IMG_1563.JPG
2019-09-01 08:55:18|7.278645|80.6175533333333|IMG_1564.JPG
2019-09-01 10:06:59|-180.0|-180.0|IMG_1570.JPG
```

Fig. 2.13 *Photos.sqlite* tables (left) and list of fields of ZGENERICASSET table (top right). SQLite query to show all photo names with the timestamp and location (bottom right)

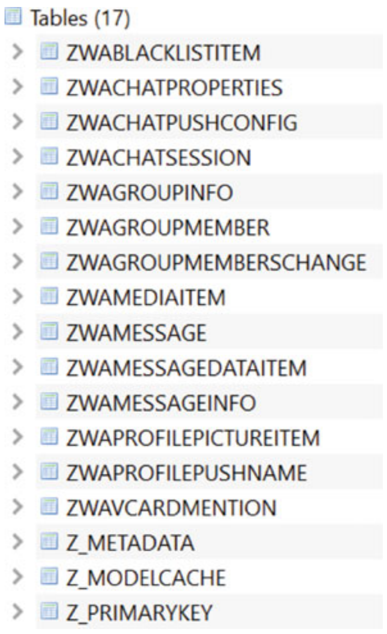


Tables (5)
- _SqliteDatabaseProperties
- deleted
- map
- sqlite_sequence
- voicemail

voicemail	
ROWID	INTEGER
remote_uid	INTEGER
date	INTEGER
token	TEXT
sender	TEXT
callback_num	TEXT
duration	INTEGER
expiration	INTEGER
trashed_date	INTEGER
flags	INTEGER
receiver	TEXT
label	TEXT

```
sqlite> SELECT *, datetime(date+978307200,'unixepoch','localtime') from
voicemail ORDER BY date ASC;
```

Fig. 2.14 Tables of *voicemail.db* (left), list of fields of *voicemail* table (right). SQLite query to explore *voicemail* table (bottom)



Tables (17)
- ZWABLACKLISTITEM
- ZWACHATPROPERTIES
- ZWACHATPUSHCONFIG
- ZWACHATSESSION
- ZWAGROUPINFO
- ZWAGROUPMEMBER
- ZWAGROUPMEMBERSCHANGE
- ZWAMEDIAITEM
- ZWAMESSAGE
- ZWAMESSAGEDATAITEM
- ZWAMESSAGEINFO
- ZWAPROFILEPICTUREITEM
- ZWAPROFILEPUSHNAME
- ZWAVCARDMENTION
- Z_METADATA
- Z_MODELCACHE
- Z_PRIMARYKEY

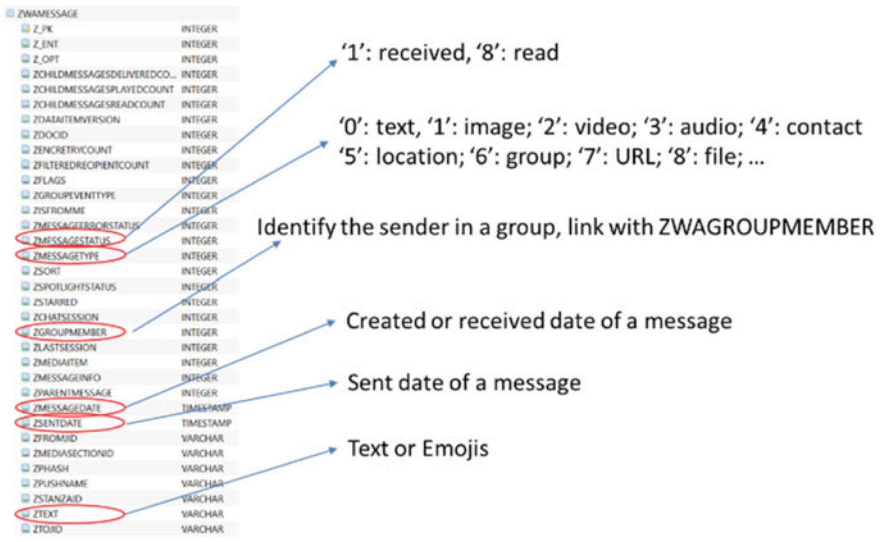
Fig. 2.15 List of tables in *ChatStorage.sqlite* database

contacts participating in group conversations. ZWAMEDIAITEM table contains a list of exchanged media items and finally ZWAMESSAGE is an important one, it stores a list of messages exchanged.

Figure 2.16 illustrates more details of ZWAMESSAGE table. The interesting fields are: ZMESSAGESTATUS contains the message status. For example, value '1' is 'received', value '8' is 'read'. ZMESSAGETYPE is the message type, with '0' is text, '1' is image; '2' is video; '3' is audio; '4' is contact; '5' is location; '6' is group; '7' is URL; '8' is file; etc. ZGROUPMEMBER stores a value, which is a primary key in ZWAGROUPMEMBER table, it links with ZWAGROUPMEMBER to identify the sender in a group. ZMESSAGEDATE contains the created date or the received date of a message. ZSENTDATE contains the sent date of a message. And finally ZTEXT contains text or Emojis.

Another interesting WhatsApp database is *Contacts.sqlite* with the most important table is ZWAADDRESSBOOKCONTACT (Fig. 2.17). In this table, we can find the list of all contacts found on the phone with the phone numbers and the contact's profile. Some important fields are ZFULLNAME stores the Contact's full name. ZPHONENUMBER stores the contact's phone number. ZWHATSAPPID stores the contact's number in WhatsApp id format, i.e. the phone number in international format.

On the Android systems, WhatsApp artifacts can be found in the following folders: *data/com.whatsapp/databases/msgstore.db; data/com.whatsapp/databases/wa.db; data/com.whatsapp/...*



ZWAMESSAGE

Z_PK	INTEGER
Z_ENT	INTEGER
Z_OPT	INTEGER
ZCHILDMESSAGESDELIVEREDCO...	INTEGER
ZCHILDMESSAGESPLAYEDCOUNT	INTEGER
ZCHILDMESSAGESREADCOUNT	INTEGER
ZDATAITEMVERSION	INTEGER
ZDOCID	INTEGER
ZENCRETRYCOUNT	INTEGER
ZFILTEREDRECIPIENTCOUNT	INTEGER
ZFLAGS	INTEGER
ZGROUPEVENTTYPE	INTEGER
ZISFROMME	INTEGER
ZMESSAGEERRORSTATUS	INTEGER
ZMESSAGESTATUS	INTEGER
ZMESSAGETYPE	INTEGER
ZSORT	INTEGER
ZSPOTLIGHTSTATUS	INTEGER
ZSTARRED	INTEGER
ZCHATSESSION	INTEGER
ZGROUPMEMBER	INTEGER
ZLASTSESSION	INTEGER
ZMEDIAITEM	INTEGER
ZMESSAGEINFO	INTEGER
ZPARENTMESSAGE	INTEGER
ZMESSAGEDATE	TIMESTAMP
ZSENTDATE	TIMESTAMP
ZFROMJID	VARCHAR
ZMEDIASECTIONID	VARCHAR
ZPHASH	VARCHAR
ZPUSHNAME	VARCHAR
ZSTANZAID	VARCHAR
ZTEXT	VARCHAR
ZTOJID	VARCHAR

'1': received, '8': read

'0': text, '1': image; '2': video; '3': audio; '4': contact
'5': location; '6': group; '7': URL; '8': file; ...

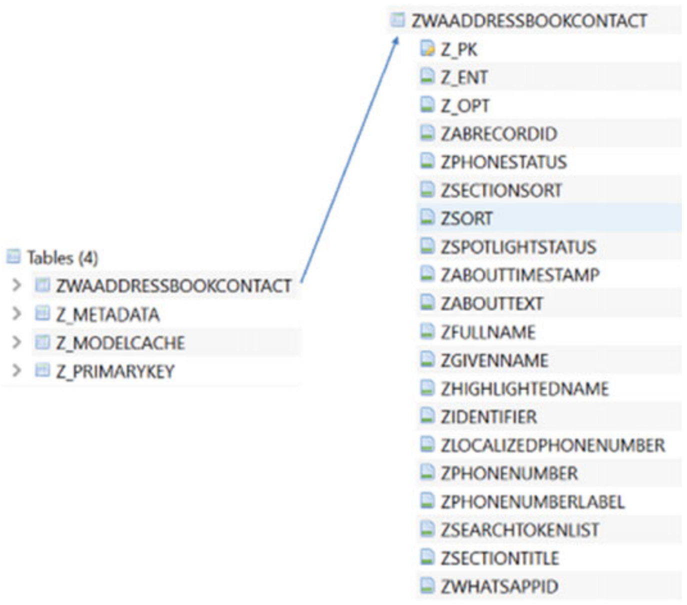
Identify the sender in a group, link with ZWAGROUPMEMBER

Created or received date of a message

Sent date of a message

Text or Emojis

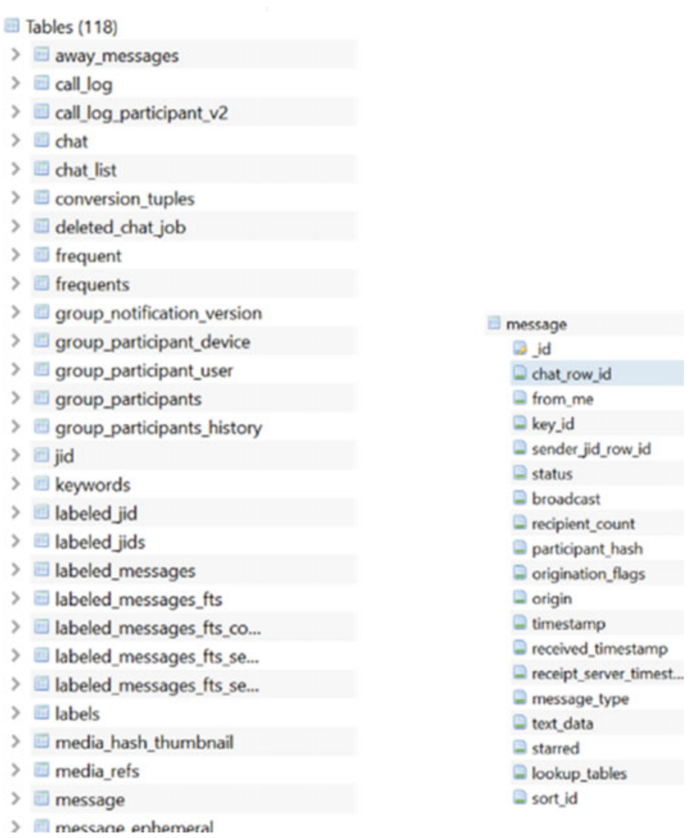
Fig. 2.16 ZWAMESSAGE table



Tables (4)
- ZWAADDRESSBOOKCONTACT
- Z_METADATA
- Z_MODELCACHE
- Z_PRIMARYKEY

ZWAADDRESSBOOKCONTACT
- Z_PK
- Z_ENT
- Z_OPT
- ZABRECORDID
- ZPHONESTATUS
- ZSECTIONSORT
- ZSORT
- ZSPOTLIGHTSTATUS
- ZABOUTTIMESTAMP
- ZABOUTTEXT
- ZFULLNAME
- ZGIVENNAME
- ZHIGHLIGHTEDNAME
- ZIDENTIFIER
- ZLOCALIZEDPHONENUMBER
- ZPHONENUMBER
- ZPHONENUMBERLABEL
- ZSEARCHTOKENLIST
- ZSECTIONTITLE
- ZWHATSAPPID

Fig. 2.17 *Contacts.sqlite* tables (left) and fields of ZWAADDRESSBOOKCONTACT table



Tables (118)
- away_messages
- call_log
- call_log_participant_v2
- chat
- chat_list
- conversion_tuples
- deleted_chat_job
- frequent
- frequents
- group_notification_version
- group_participant_device
- group_participant_user
- group_participants
- group_participants_history
- jid
- keywords
- labeled_jid
- labeled_jids
- labeled_messages
- labeled_messages_fts
- labeled_messages_fts_co...
- labeled_messages_fts_se...
- labeled_messages_fts_se...
- labels
- media_hash_thumbnail
- media_refs
- message
- message_ephemeral

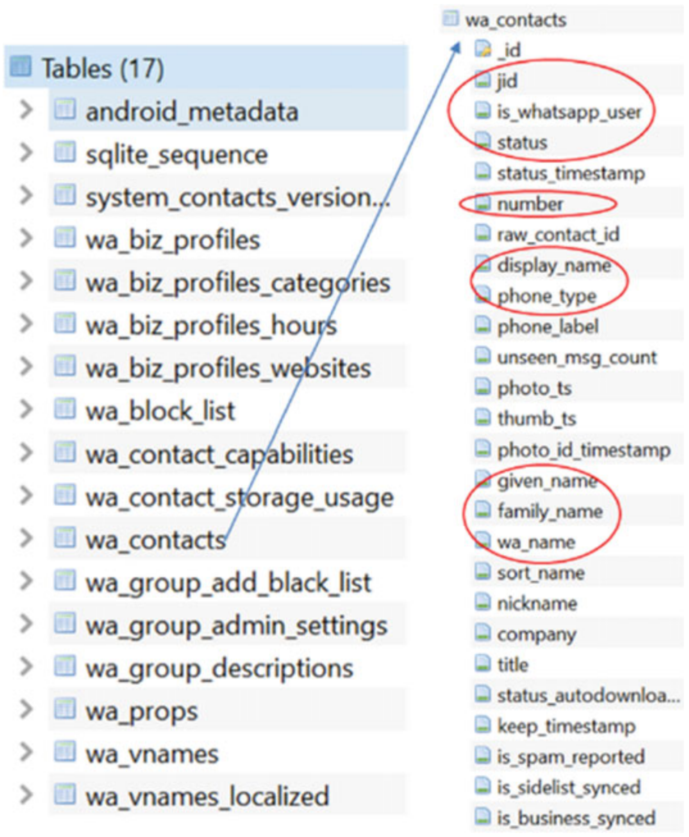
message
- _id
- chat_row_id
- from_me
- key_id
- sender_jid_row_id
- status
- broadcast
- recipient_count
- participant_hash
- origination_flags
- origin
- timestamp
- received_timestamp
- receipt_server_timest...
- message_type
- text_data
- starred
- lookup_tables
- sort_id

Fig. 2.18 *msgstore.db* tables (left) and fields of *message* table (right)

The folder ‘databases’ was identified as storing the most valuable information in order to reconstruct the communication history in WhatsApp, which were the two main database files *wa.db* and *msgstore.db* that contained all the exchanged attachments, such as images, video and contact cards. Figure 2.18 illustrates the structure of database and fields of its *message* table.

The *wa.db* database has one important table *wacontacts* (Fig. 2.19). This table contains a list of all contacts found on the phone (not only WhatsApp contacts). It has the following attributes:

- jid: The contact or group WhatsApp id in < phonenumber> @s.whatsapp.net format
- is_whatsapp_user: Boolean if the phone number is a registered WhatsApp user
- status: The status text of the contact
- number: The contact’s phone number
- display_name: The contact’s display name in the contact list
- phone_type: Number mapped to phone type, i.e. mobile, home etc.



Tables (17)
- android_metadata
- sqlite_sequence
- system_contacts_version...
- wa_biz_profiles
- wa_biz_profiles_categories
- wa_biz_profiles_hours
- wa_biz_profiles_websites
- wa_block_list
- wa_contact_capabilities
- wa_contact_storage_usage
- wa_contacts
- wa_group_add_black_list
- wa_group_admin_settings
- wa_group_descriptions
- wa_props
- wa_vnames
- wa_vnames_localized

wa_contacts
- _id
- jid
- is_whatsapp_user
- status
- status_timestamp
- number
- raw_contact_id
- display_name
- phone_type
- phone_label
- unseen_msg_count
- photo_ts
- thumb_ts
- photo_id_timestamp
- given_name
- family_name
- wa_name
- sort_name
- nickname
- company
- title
- status_autodownloa...
- keep_timestamp
- is_spam_reported
- is_sidelist_synced
- is_business_synced

Fig. 2.19 *wa.db* tables (left) and fields of *wa_contacts* table (right)

- given_name: Contact's first name
- family_name: Contact's family name
- wa_name: Contact's WhatsApp name.

We refer readers to other related research efforts on WhatsApp forensics, such as those outlined in [19, 20].

2.6 Summary

This chapter present background of databases and database investigation that are necessary to follow case studies in following chapters. Basic concepts of databases including relational databases, database design and SQL language were introduced. This chapter also described a process for investigating application databases. Finally, two examples of database forensics were illustrated to show how to examine databases from iOS and from an instant message application. The following chapters will present case studies where database investigation techniques are applied in acquire relevant artifacts from different applications of forensic cases.

Acknowledgements Authors would like to thank Ranul Thantilage, who helped us to prepare relevant SQLite databases used in this chapter.

References

1. Warren, C., El-Sheikh, E., & Le-Khac, N.-A. (2017). Privacy preserving internet browsers—forensic analysis of browzar. In K. Daimi, et al. (Eds.), *Computer and network security essentials* (pp. 369–388 (18 pages)). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-319-58424-9_21
2. Sgaras, C., Tahar Kechadi, M., & Le-Khac, N.-A. (2015). *Forensics acquisition and analysis of instant messaging and VoIP applications. Lecture Notes in Computer Science* (Vol. 8915, pp. 188–199 (12 pages)). https://doi.org/10.1007/978-3-319-20125-2_16
3. Schipper, G. C., Seelt, R., & Le-Khac, N.-A. (2021). Forensic analysis of matrix protocol and Riot.im application. *Forensic Science International: Digital Investigation, 36*, 301118. https://doi.org/10.1016/j.fsidi.2021.301118
4. Thantilage, R., & Le-Khac, N.-A. (2020). Forensic analysis of e-dating applications based on iPhone backups. In G. Peterson & S. Shenoi (Eds.), *Advances in digital forensics XVI. Digital forensics 2020. IFIP advances in information and communication technology* (Vol. 589). Springer. https://doi.org/10.1007/978-3-030-56223-6_12
5. Voorst, R. V., Kechadi, T., & Le-Khac, N.-A. (2015). Forensics acquisition of IMVU: a case study. *Journal of Association of Digital Forensics, Security and Law, 10*(4), 69–78 (10 pages). https://doi.org/10.15394/jdfsl.2015.1212
6. Eliot, T. S. (1934). The Rock, Hardcourt, Brace and Company Inc. https://archive.org/details/in.ernet.dli.2015.3608/mode/2up
7. Celko, J. (1999). *Data and databases: Concepts in practice, Morgan Kaufmann* (1st ed.). ISBN-13: 978-1558604322.
8. Connolly, T., & Begg, C. (2010). *Database systems: A practice; approach to design, implementation and management* (5th ed.). Addison Wesley.
9. Coronel, C., et al. (2020). *Database principles: Fundamentals of design, implementation and management* (3rd ed.). Cengage Learning EMEA. ISBN-13: 978-1473768048.
10. Assaf, W., et al. (2018). *SQL server 2017 administration inside out* (1st ed.). Microsoft Press. ISBN-13: 978-1509305216.
11. Freeman, R. G. (2013). *Oracle database 12c new features* (1st ed.). McGraw Hill.
12. Matthew, N., & Stones, R. (2005). *Beginning databases with PostgreSQL: From novice to professional* (2nd Corrected ed., Corr. 3rd printing ed.). A Press. ISBN-13: 978-1590594780.
13. Grippa, V. M., & Kuzmichev, S. (2021). *Learning MySQL: Get a handle on your data* (2nd ed.). O'Reilly Media, Inc. ISBN-13: 978-1492085928.
14. Coronel, C., & Morris, S. (2018). *Database systems: Design, implementation, & management* (13th ed.). *Cengage learning* (13th ed.). ISBN-13: 978-1337627900.
15. Rockoff, L. (2016). *The language of SQL* (2nd ed.). Addison-Wesley Professional.
16. Kreibich, J. A. (2010). *Using SQLite* (1st ed.). O'Reilly Media. ISBN-13: 978-0596521189.
17. Aouad, L., Kechadi, M.-T., Trentesaux, J., & Le-Khac, N.-A. (2012). An open framework for smartphone evidence acquisition. In P. Gilbert & S. Sujeet (Eds.), *Advances in digital forensics VIII* (pp. 159–166 (8 pages)). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-33962-2

18. Thantilage, R., & Le-Khac, N.-A. (2019). Framework for the retrieval of social media and instant messaging evidence from volatile memory. In *18th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (IEEE TrustCom-19)* (CORE Rank A).
19. Cents, R., & Le-Khac, N.-A. (2020). Towards a new approach to identify WhatsApp messages. In *19th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (IEEE TrustCom-20)* (CORE Rank A).
20. Wijnberg, D., & Le-Khac, N.-A. (2021). Identifying interception possibilities for WhatsApp communication. *Forensic Science International: Digital Investigation, 38*(Supplement), 301132. https://doi.org/10.1016/j.fsidi.2021.301132