Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.28, NO.3, Jun. 2018 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2018.28.3.625

Realm 데이터베이스의 삭제된 레코드 복구 기법

김 준 기,† 한 재 혁, 최 종 현, 이 상 진‡
고려대학교 정보보호연구원

The Method of Recovery for Deleted Record of Realm Database

Junki Kim,† Jaehyeok Han, Jong-Hyun Choi, Sangjin Lee‡
Institute of Cyber Security & Privacy (ICSP), Korea University

요 약

Realm은 모바일 기기에서 주로 사용되는 SQLite를 대체하기 위해 개발된 오픈 소스 데이터베이스이다. 데이터베이스에 저장되는 데이터는 사용자의 행위를 파악하고 모바일 기기의 동작 여부를 확인하는 데 도움이 될 수 있어 모바일 기기를 대상으로 하는 디지털 포렌식 분석 과정에서 반드시 확인되어야 한다. 뿐만 아니라, 사용자가 의도적으로 데이터베이스에 저장된 데이터 삭제와 같은 안티 포렌식 기법을 사용할 수 있으므로 데이터베이스에서 삭제된 레코드를 복구하는 방법에 대한 연구가 필요하다. 본 논문은 Realm 데이터베이스 파일의 구조와 레코드의 저장 및 삭제 과정을 분석한 결과를 바탕으로 삭제된 후 덮어 쓰여지지 않은 레코드의 복구 기법을 제시하였다.

ABSTRACT

Realm is an open source database developed to replace SQLite, which is commonly used in mobile devices. The data stored in the database must be checked during the digital forensic analysis process for mobile devices because it can help to understand the behavior of the user and whether the mobile device is operating or not. In addition, since the user can intentionally use anti-forensic techniques such as deleting data stored in the database, research on how to recover deleted records is needed. In this paper, we propose a method to recover records that have not been overwritten after deletion based on the analysis of the structure and record and deletion process of the Realm database file.

Keywords: Realm, Mobile Database, Recovery

I. 서 론

최근 모바일 기기에서 동작하는 애플리케이션들은 생성하고 관리해야 할 데이터의 양이 증가함에 따라 이를 효율적으로 관리하기 위해 데이터베이스를 사용한다. 애플리케이션이 주로 생성하는 데이터로는 동작에 필요한 환경 설정 정보, 사용자의 사용 기록, 위치 정보 등이 있다. 이러한 데이터들은 모바일 기기를 대상으로 디지털 포렌식 분석을 진행할 때 사용자의 행위를 파악하고 모바일 기기의 동작 여부를 확인하는 등 분석 과정에 도움이 될 수 있으므로 데이터베이스를 분석해 데이터를 추출하는 것은 중요하다. 또한, 사용자가 의도적으로 데이터베이스에 저장된 데이터를 삭제하는 등의 안티 포렌식 기법을 사용할 수 있으므로 데이터베이스에서 삭제된 데이터를 복구하는 것도 중요하다.

Realm은 모바일 환경에서 사용되는 C++ 기반의 오픈 소스 데이터베이스로 SQLite를 대체하기 위해 개발되어 2014년 7월에 발표되었다. Realm은 모바일 환경의 특성에 맞게 작은 크기와 빠른 속도를 특징으로 가지며[1], SQLite와 유사한 기능과 성능을 보여 최근에는 SQLite를 대체하기 시작했다. 또

Received(04. 19. 2018), Modified(05. 03. 2018), Accepted(05. 15. 2018)
† 주저자, jjun1207@korea.ac.kr
‡ 교신저자, sangjin@korea.ac.kr(Corresponding author)

Table 1. List of mobile database

Name	Android/iOS	Type of data stored	Ranking[2]
SQLite	Android/iOS	Relational	11
Couchbase Lite	Android/iOS	JSON Documents / NoSQL db	24
Interbase	Android/iOS	Relational	44
Realm	Android/iOS	Object Database	50
SQL Anywhere	Android/iOS	Relational	59
Oracle Berkeley DB	Android/iOS	Relational and Key-value store	82
LevelDB	Android/iOS	Key-value pairs / NoSQL db	83
Snappy DB	Android	Key-value pairs / NoSQL db	241
ForestDB	Android/iOS	Key-value pairs / NoSQL db	-
UnQLite	Android/iOS	Key-value paris / document store / NoSQL db	-

한, 지속적인 업데이트가 이루어지고 사용되는 범위가 점차 확대되고 있으므로[2] 앞으로 더욱 많은 분야에서 사용될 것으로 예상된다. 이러한 이유로 Realm은 모바일 기기를 대상으로 수행하는 디지털 포렌식 분석 과정에서 수집될 가능성이 커졌지만, Realm 데이터베이스 파일을 수집하여 저장된 데이터를 추출하거나 삭제된 데이터를 복구하는 기술은 연구되어있지 않다. 따라서 Realm 데이터베이스 파일에서 정상 데이터를 추출하고 삭제된 데이터를 복구하는 기법에 관한 연구가 필요하다.

II. Realm 데이터베이스

2.1 Realm 데이터베이스 개요

Realm은 모바일 데이터베이스의 한 종류로 SQLite를 대체하기 위한 목적으로 개발된 오픈 소스 데이터베이스이다. 모바일 데이터베이스는 주로 캐시를 저장하거나 애플리케이션에서 생성한 주요 데이터를 저장하는 데이터 저장소로 사용되며, 일반적인 데이터베이스와 다르게 낮은 성능, 제한된 배터리와 같이 물리적인 한계가 있는 환경에서 사용되기 때문에 작은 데이터베이스 파일 크기와 시스템 자원을 최소한으로 사용하도록 설계된 특징을 가진다. 현재 개발되어 사용되고 있는 모바일 데이터베이스 목록은 Table 1과 같다[3, 4]. Realm은 안드로이드와 iOS 모두에서 사용할 수 있으며, Java, Object-C, Swift, React Native, Xamarin, C# 등에서 사용할 수 있는 라이브러리를 제공함으로써 다양한 프로그래밍 언어를 사용하여 Realm을 사용하는 모바일 애플리케이션을 개발할 수 있다.

Realm은 객체 데이터베이스로 데이터베이스 구조를 포함한 데이터베이스에 저장되는 모든 데이터를 객체 형식으로 저장하며, 저장되는 객체들을 B+트리로 관리한다. Fig. 1은 Realm의 논리적인 구조를 나타낸 것으로써, 관계형 데이터베이스에서 사용하는 테이블과 유사한 형태로 데이터를 관리한다는 것을 알 수 있다. Realm이 사용하는 데이터베이스 모델인 객체 데이터베이스는 데이터베이스에 저장하는 정보를 객체지향(Object-Oriented) 프로그래밍에서 사용하는 객체 형식으로 저장하며, 복잡한 데이터를 저장하고 관리하기 위한 목적으로 개발되었다. 객체 데이터베이스는 관계형 데이터베이스와 다르게 데이터를 행과 열로 표현하지 않고 데이터 사이의 복

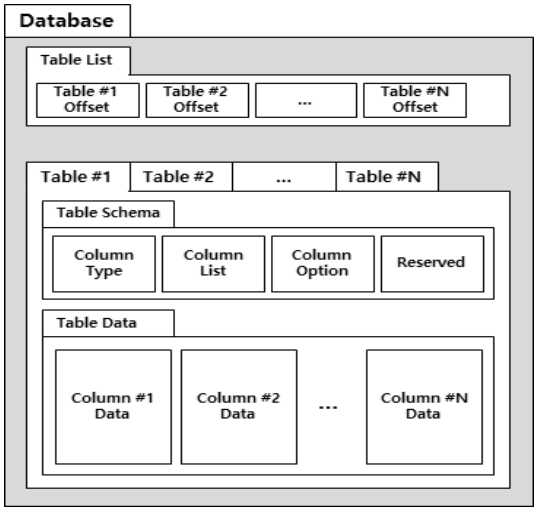

Fig. 1. Logical Structure of the Realm

잡한 관계를 포인터 형태로 저장하므로 일대다(One to many) 또는 다대다(Many to many) 관계를 나타낼 수 있어 매우 복잡한 데이터를 처리하는 응용 프로그램에 사용될 수 있다[5].

또한, Realm은 레코드를 삽입할 때 행 단위(Row-based)가 아닌 열 단위(Column-oriented)로 저장하는 특징이 있다(Fig. 2). 열 단위로 레코드를 삽입하는 방식은 행 단위로 레코드를 삽입하는 방식과 비교하였을 때 하드웨어 자원 사용량을 크게 줄일 수 있고, 쿼리를 효율적으로 처리할 수 있다는 장점이 있다[1].

데이터베이스 모델에서는 여러 개의 프로세스가 동시에 하나의 데이터베이스로 접근할 때 데이터의 일관성이 깨질 수 있다는 문제를 해결해야 한다. 이를 해결하기 위해서 일반적으로 데이터베이스 파일에 쓰기 제한(Lock) 기능을 적용한다. 쓰기 제한 기능은 데이터베이스 파일에 새로운 값을 기록하고자 할 때 하나의 프로세스가 데이터베이스 파일에 접근하면 다른 프로세스가 동시에 접근하지 못하도록 하는 방법이다. 그러나 이 기능은 데이터베이스 파일의 데이터를 조작(삽입, 삭제, 수정)할 때 오버헤드가 발생한다는 단점이 있다.

Realm은 이러한 단점을 보완하기 위해 COW(Copy on write) 방식을 사용함으로써 데이터의 일관성을 유지하면서 신속하게 데이터베이스 접근을 처리할 수 있다. COW 방식은 쓰기 동작(레코드 삽입, 삭제, 수정)이 발생하면 기존의 데이터를 수정하는 것이 아니라, 기존의 데이터의 복사본을 생성한 후 복사본에

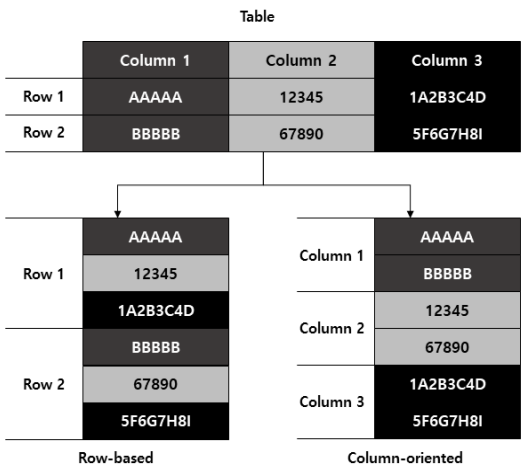

Fig. 2. Row-based vs Column-oriented

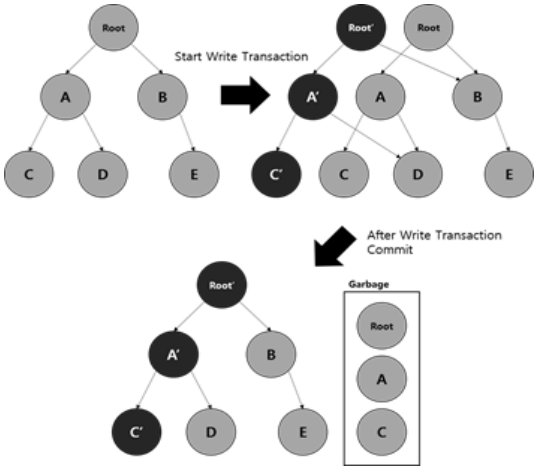

Fig. 3. The Process of the COW(Copy on Write)

서 쓰기 동작을 수행하고 쓰기 동작이 완료되면 데이터의 연결을 원본에서 수정된 복사본으로 바꾸는 기법으로 Fig. 3과 같이 동작한다[6, 7].

2.2 응용 및 활용 범위

현재 많은 모바일 애플리케이션들(예: 네이버 앱, 네이버 파파고, 11번가, 카카오 맵, 비트윈, 네이버 폴라, 리디북스, 피키캐스트, 알리바바, 트위터 비디오 앱(Vine) 등)이 사용자 행위에 따른 데이터를 Realm을 사용하여 저장한다. Table 2는 각 애플리케이션에서 저장하는 데이터를 정리한 표이다[8].

III. Realm 데이터베이스 파일 구조 분석

3.1 Realm 데이터베이스 파일

데이터베이스 파일에서 정상 데이터 영역과 삭제된 데이터 영역을 구분하고 삭제된 레코드를 복원하기 위해서는 데이터베이스 파일의 구조를 파악해야 하며, 본 논문에서는 Realm 개발진이 공개한 코드[7]를 이용하여 데이터베이스 파일 구조와 레코드 삽입 및 삭제 과정을 분석하였다.

Realm 데이터베이스 파일은 'realm'을 확장자로 가지며, 파일 시그니처는 'T-DB' (0x54 2D 44 42)이다. 데이터베이스 파일의 구조는 Table 3과 같이 트리 루트 오프셋, 파일 시그니처, 파일 속성, 객체 컨테이너로 구성된다. 데이터베이스가 저장하고 있는 데이터를 찾아가기 위해서는 가장 먼저 루트에

Table 2. Applications that use the Realm database

Application Name	Platform	Contents in the database file	
		Realm	SQLite
Naver	Android	Application Usage History	Search History, Bookmark, App Environment Setting
	iOS		
Naver Papago	Android	Translation History	App Environment Setting
	iOS	Translation History	-
11st	iOS	Shopping History	App Environment Setting
Kakao Map	Android	Search History	App Environment Setting
	iOS	Search History	-
JikBang	Android	Search History	App Environment Setting
	iOS	Search History, Location Information	-

접근해야 한다. Realm의 경우 쓰기 동작에서 COW 방식을 사용하기 때문에 원본과 복사본에 대한 두 개의 트리를 가지며, 파일 속성(File Attribute)값을 참조하여 루트로 접근하기 위한 오프셋 값을 확인해야 한다. 파일 속성의 하위 1바이트 값이 '0'인 경우에는 첫 번째 트리 루트 오프셋(Tree Root Offset #1)을 참조하고, '1'인 경우에는 두 번째 트리 루트 오프셋(Tree Root Offset #2)을 참조한다. 객체 컨테이너(Object Container)에는 데이터베이스의 모든 데이터가 객체 형식으로 저장되어 있다.

객체는 Realm에서 데이터를 저장하는 단위로써, 컬럼 타입(Column Type), 컬럼 리스트(Column List) 또는 컬럼 데이터(Column Data) 등을 저장하는 구조체이다. Table 4는 객체의 구조를 분석한 결과이며 객체 시그니처(Object Signature) 'AAAA' (0x41 41 41 41), 객체 타입(Object Type), 객체 데이터의 크기 또는 개수(Count or Length of Data), 객체 데이터(Object Data)로 구성된다. 컬럼 데이터로 접근하기 위해서는 먼저 참조할 트리 루트 오프셋을 결정하고 테이블 스키마를 구성하는 객체들과 테이블 데이터 객체들을 해석한 후에 컬럼 데이터로 접근할 수 있다. 관계형 데이터베이스의 레코드와 같은 형태로 값을 읽기 위해서는 모든 컬럼 데이터의 값을 순서에 맞춰 읽어야 한다.

Table 3. Structure of Realm Database file

Offset	Size	Field
0x00	8	Tree Root Offset #1
0x08	8	Tree Root Offset #2
0x10	4	File Signature 'T-DB'
0x14	4	File Attribute
0x18	Variable Length	Object Container

3.2 트리 구조를 이용한 데이터베이스 구성

Realm 데이터베이스 파일의 구조를 나타내는 트리는 Fig. 4와 같이 테이블 정보(Table Information) 객체의 오프셋과 테이블 배열(Table Array) 객체의 오프셋으로 구성된다. 트리 루트 노드의 첫 번째 오프셋으로 이동하면 데이터베이스에 존재하는 테이블의 목록을 확인할 수 있고, 두 번째 오프셋으로 이동하면 테이블 정보가 저장된 테이블 객체들의 오프셋 목록을 확인할 수 있다. 데이터베이스에 존재하는 테이블의 목록은 테이블 정보 객체에서만 확인할 수 있으므로 데이터베이스를 분석하는 과정에서 반드시 확인하여야 한다.

테이블 객체의 구조는 Fig. 5와 같이 테이블 스키마(Table Schema) 객체의 오프셋과 데이터 스토리지(Data Storage) 객체의 오프셋으로 구성된다. 테이블 스키마 객체에는 테이블을 구성하고 있는

Table 4. Structure of Object

Offset	Size	Field
0x00	4	Object Signature 'AAAA'
0x04	2	Object Type
0x06	2	Count or Length of Data
0x08	Variable Length	Object Data

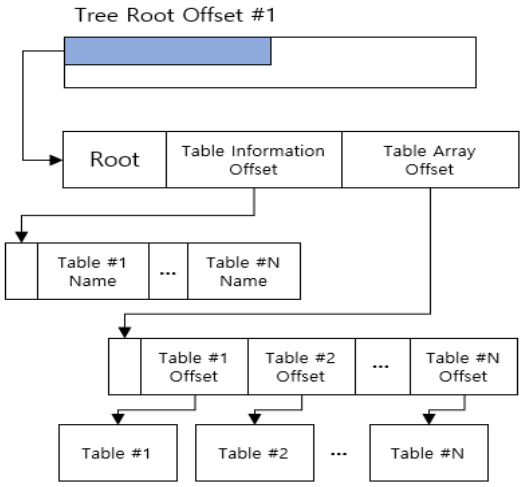

Fig. 4. Tree structure of Realm database

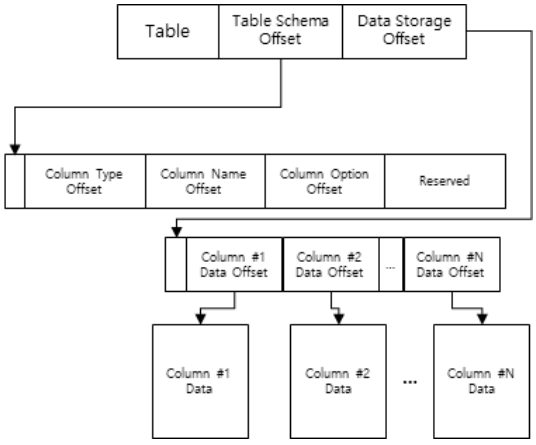

Fig. 5. Structure of Table Object

컬럼의 종류, 이름 그리고 기본키 설정과 같은 옵션 정보들이 저장되며, 데이터 스토리지 객체에는 테이블의 컬럼 순서에 따라 테이블에 저장된 컬럼 데이터 객체의 오프셋이 저장된다.

IV. Realm 데이터베이스의 삭제된 레코드 복구

4.1 Realm 데이터베이스의 레코드 삭제 방법

Realm은 COW를 사용하여 쓰기 동작을 수행하므로 일반적인 데이터베이스와는 다른 방식으로 레코드를 삭제한다. Realm의 레코드 삭제 과정은 원본 데이터 객체에 대한 복사본 생성, 복사본에서 레코드 삭제 그리고 테이블 객체의 데이터 스토리지 오프셋

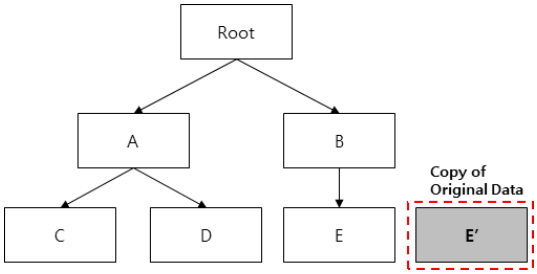

Fig. 6. First step of delete record in Realm Database

업데이트 세 단계로 구성된다. 먼저 Realm에서 사용자가 특정 레코드를 삭제하면 Fig. 6과 같이 원본 컬럼 데이터 객체에 대한 복사본 객체를 생성한다.

원본 데이터 객체에 대한 복사본 객체를 생성한 후에는 복사본 객체를 대상으로 레코드 삭제를 진행하며, Fig. 7과 같이 컬럼 데이터에서 삭제하고자 하는 레코드의 데이터를 테이블의 마지막 레코드의 데이터로 덮어씌워 레코드를 삭제한다. 삭제하고자 하는 레코드를 해당 레코드의 다음으로 덮어쓰는 방법은 최악의 경우 (테이블에 저장된 레코드 수 - 1) 만큼의 동작이 필요할 수 있지만, 마지막 레코드로 덮어씌워 삭제하는 방법은 한 번의 동작이 필요하므로 Realm은 레코드 삭제 과정에서 발생하는 동작을 최소화하기 위해 위와 같은 방식을 사용한다.

복사본 객체를 대상으로 레코드 삭제가 완료되면 레코드가 저장된 테이블의 테이블 객체에서 데이터 스토리지에 저장된 원본 컬럼 데이터 객체의 오프셋을 복사본 객체의 오프셋으로 변경하여 레코드 삭제를 완료한다. 이 과정에서 원본 컬럼 데이터 객체는 삭제되지 않고 데이터베이스 파일 내부에 남아 있게 되며, 이를 그림으로 나타내면 Fig. 8과 같다.

Realm에서 레코드를 삭제하는 과정을 정리하면 Fig. 9와 같으며, 레코드 삭제 과정을 분석한 결과 레코드가 삭제되면 삭제되기 전의 원본 컬럼 데이터

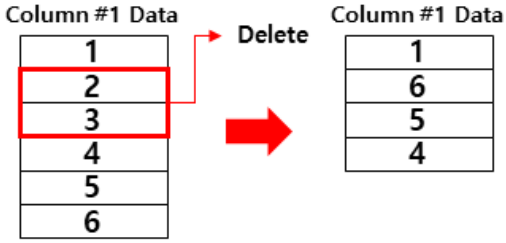

Fig. 7. Second step of delete record in Realm Database

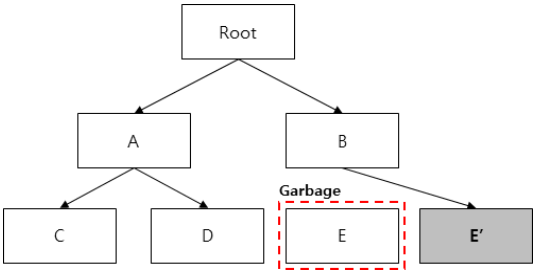

Fig. 8. Last step of delete record in Realm Database

객체가 Realm 데이터베이스 파일에 남아 있어 원본 컬럼 데이터 객체에 해당하는 영역이 다른 데이터로 덮어 쓰여지지 않았다면 원본 컬럼 데이터 객체를 추출하여 삭제된 레코드를 복구할 수 있다.

4.2 Realm 데이터베이스의 삭제된 레코드 복구 기법

Realm 데이터베이스 파일에서 삭제된 레코드를 복구하기 위해서는 데이터베이스 파일에서 삭제된 레코드를 파악하고 삭제된 레코드의 데이터가 저장된 컬럼 데이터 객체를 식별해야 한다. 이후 확인된 컬럼 데이터 객체에서 데이터를 추출하여 삭제된 레코드의 컬럼 단위 데이터를 복구할 수 있다. 그러나 Realm의 컬럼 데이터 객체에는 테이블에 대한 정보가 기록되지 않기 때문에 컬럼 데이터의 타입과 테이블 정보를 복구하는 과정이 필요하다. 테이블 정보는 정상 레코드의 컬럼 데이터 객체와 복구된 컬럼 데이터 객체를 비교하여 삭제된 레코드의 테이블 정보를 파악할 수 있다. 확인된 테이블 정보를 이용하여 복구된 컬럼 단위 데이터를 컬럼의 순서대로 정렬하여

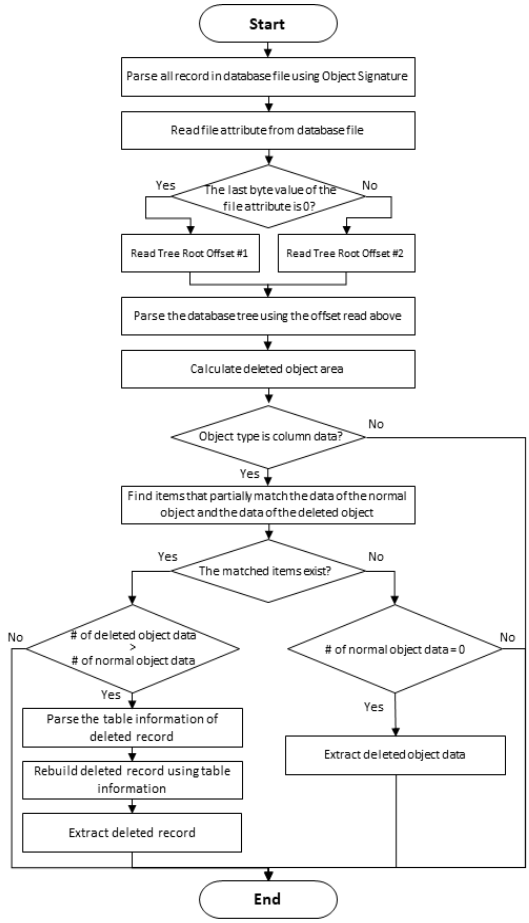

Fig. 10. The process of recovering deleted record

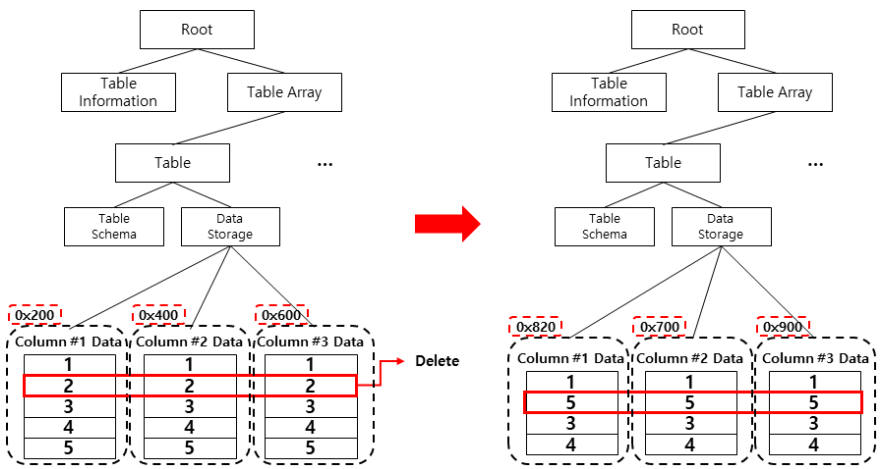

Fig. 9. The process of deleting records in Realm Database

Realm 데이터베이스에서 삭제된 레코드 복구를 완료한다. 삭제된 레코드를 복구하는 과정을 순서도로 나타내면 Fig. 10과 같다.

4.2.1 데이터베이스 파일 내 삭제된 레코드 파악

Realm 데이터베이스 파일에서 삭제된 레코드를 식별하기 위해서는 먼저 데이터베이스 파일의 모든 객체에서 정상 객체와 삭제된 객체를 식별해야 한다. 데이터베이스 파일에 저장된 모든 객체에 대한 정보는 객체의 시그니처인 'AAAA'를 이용하여 데이터베이스 파일을 대상으로 전수조사하여 파악한다. 그리고 정상 객체에 대한 정보는 데이터베이스 구조를 저장하는 트리를 분석하여 획득해야 하며, 이때 파일 속성의 하위 1바이트 값을 참조하여 분석 대상 트리의 트리 루트 오프셋을 확인한 후 분석해야 한다. 데이터베이스 파일에 저장된 모든 객체와 정상 객체에 대한 정보를 획득한 후에는 Fig. 11과 같이 데이터베이스 파일의 전체 객체 목록에서 정상 객체를 제외한 나머지 객체들을 삭제된 객체로 판단한다.

데이터베이스 파일에서 정상 객체와 삭제된 객체를 파악한 후에 실제 레코드의 데이터를 저장하고 있는 컬럼 데이터 객체를 분류하여 데이터를 추출한다. 컬럼 데이터 객체에서 추출한 데이터는 삭제된 레코드를 파악하는 데 사용되며, 정상 객체와 삭제된 객체의 데이터를 비교하여 삭제된 레코드를 파악한다.

비교 결과에서 내용 일부가 일치하는 정상-삭제된 객체 쌍이 존재하는 경우에는 삭제된 객체보다 정상 객체에 더 많은 데이터가 저장되어 있으면 테이블에 새로운 레코드가 삽입된 것으로, 정상 객체보다 삭제된 객체에 더 많은 데이터가 저장되어 있으면 해당 데이터에 해당하는 레코드가 테이블에서 삭제된 것으로 판단할 수 있다(Case 1). 그리고 일치하는 정상-삭제된 객체 쌍이 없는 경우에는 정상 객체에만 데이터가 존재하면 테이블의 모든 레코드가 새로 추가된 것으로, 삭제된 객체에만 데이터가 존재하면 테이블의 모든 레코드가 동시에 삭제된 것으로 판단할 수 있다(Case 2).

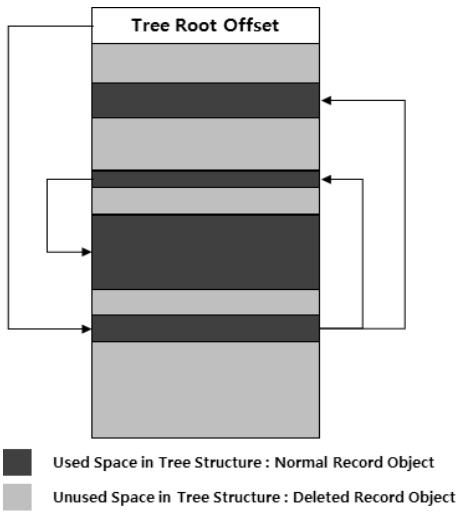

Fig. 11. Classify normal object and Deleted Object

4.2.2 삭제된 레코드 분석 및 데이터 복구

데이터베이스 파일에서 삭제된 레코드를 식별한 후에는 삭제된 레코드의 테이블 정보를 확인해야 한다. Realm은 컬럼 데이터 객체에 테이블 정보를 저장하지 않으므로 삭제된 레코드의 테이블 정보를 획득하는 과정이 필요하다. 삭제된 레코드의 테이블 정보는 4.2.1에서 정상 객체와 삭제된 객체의 데이터를 비교하는 과정에서 획득할 수 있다. Case 1에 해당하는 삭제된 레코드의 경우 내용 일부가 일치하는 정상 객체와 동일한 테이블 정보를 가지는 것으로 판단할 수 있으며, 삭제된 레코드의 테이블 정보를 확인한 후에는 테이블 정보를 이용하여 복구된 레코드를 컬럼의 순서대로 정렬함으로써 Realm 데이터베이스 파일에서 삭제된 레코드를 복구할 수 있다. 그러나 Case 2에 해당하는 삭제된 레코드의 경우에는 비교할 수 있는 정상 객체가 존재하지 않아 테이블 정보를 확인할 수 없어 삭제된 레코드의 데이터를 객체 단위로 추출만 가능하다.

4.3 Realm의 삭제된 레코드 복구 기법 성능 평가

4.3.1 실험 시나리오

본 논문에서 제시한 삭제된 레코드 복구 기법의 성능을 평가하기 위해 사용자 행위를 고려하여 실험용 데이터베이스 파일을 만들고 앞서 제시한 삭제된 레코드 복구 기법을 적용하여 결과를 살펴본다. 실험 대상으로 사용하는 데이터베이스 파일은 레코드가 덮어쓰여지는 상황을 가정하여 레코드가 수정(삽입, 삭제)되는 과정을 반복하여 생성하였다. 실험용 데이터베이스 파일의 레코드 수정은 Realm 라이브러리를

Table 5. Result of Recovery Test

Operation	(Step 1) Initial State	(Step 2) Delete 25 Records	(Step 3) Insert 50 Records	(Step 4) Delete 25 Records	(Step 5) Insert 50 Records
Normal	100	75	125	100	150
Deleted	-	25	25	50	50
Recovered	-	25	25	25	25
Recovery Rate	-	100%	100%	50%	50%

이용한 코드를 작성하여 진행하였고, 데이터베이스에 저장된 레코드는 Realm 개발진에서 제공하는 뷰어를 사용하여 확인하였다[9]. 실험용 데이터베이스 파일을 생성하기 위해 먼저 데이터베이스에 네 개의 컬럼(문자열, 정수, 실수, 문자열)으로 구성된 하나의 테이블을 생성하고 임의의 값을 가진 레코드 100개를 삽입하였다(Step 1). 그리고 테이블에 저장된 레코드의 가장 앞에서부터 연속적으로 25개의 레코드를 삭제하고(Step 2), 테이블의 마지막에 새로운 레코드 50개를 삽입하였다(Step 3). 이후에 삭제된 레코드 영역을 덮어쓰기 위해 앞서 레코드를 삭제(Step 4)하고 삽입(Step 5)하는 과정을 다시 수행하여 대상 데이터베이스 파일을 생성하였다. 이와 같은 과정을 통해 생성된 데이터베이스 파일은 총 200개의 레코드가 삽입되었으며 150개의 정상 레코드와 50개의 삭제된 레코드로 구분할 수 있다. 실험에서는 동작에 따른 복구율을 비교하기 위해 단계별로 데이터베이스 파일을 생성하여 삭제된 레코드 복구 기법을 적용하였다.

4.3.2 실험 결과

4.3.1에서 생성한 실험용 데이터베이스 파일을 대상으로 본 논문에서 제시하는 삭제된 레코드 복구 기법을 적용한 결과는 Table 5와 같다. Step 2의 경우 삭제된 레코드 영역이 덮어 쓰여지지 않았기 때문에 삭제된 모든 레코드에 대한 복구가 가능하였으며, Step 3에서는 Step 2에서 삭제된 레코드 영역이 새로 삽입한 레코드로 덮어 쓰일 것으로 예상하였지만 덮어 쓰여지지 않아 삭제된 모든 레코드에 대한 복구가 가능하였다. 그리고 Step 4와 5에서는 Step 2에서 삭제된 레코드 영역이 새로운 레코드로 덮어 쓰였고, Step 4에서 삭제된 레코드 영역은 덮어 쓰여지지 않아 Step 2에서 삭제된 레코드를 제외한 나머지 레코드는 복구 가능하였다(평균 복구율 75%). 실험을 통해 삭제된 레코드 중 덮어 쓰여지지 않은 레코드의 경우 복구가 가능한 것을 확인하였으며, Realm은 모바일 애플리케이션에서 사용되는 점을 고려하였을 때 삭제된 데이터가 덮어 쓰여지는 경우는 거의 없으므로 삭제된 레코드 중 덮어 쓰여지지 않은 레코드를 복구하는 기법에 관한 연구는 의미가 있다. 실제로 Realm 데이터베이스 파일을 모바일 기기에서 수집하여 삭제된 레코드 복구 기법을 적용한 경우 사건 조사에 필요한 삭제된 사용자 데이터를 수집한 사례가 몇 차례 있었다.

V. 결 론

모바일 애플리케이션에서 Realm의 사용이 증가함에 따라 모바일 기기를 대상으로 디지털 포렌식 조사를 진행할 때 Realm 데이터베이스 파일을 수집할 가능성이 커졌다. Realm에는 사용자의 행위를 파악하고 모바일 기기의 동작 여부를 확인하는 데 도움이 될 수 있는 데이터가 주로 저장되기 때문에 Realm 데이터베이스 파일을 분석하는 것은 중요하다. 또한, 사용자가 의도적으로 데이터베이스에 저장된 데이터 삭제하는 등의 안티 포렌식 기법을 사용할 수 있으므로 삭제된 레코드를 복구하는 방법도 중요하다.

본 논문에서는 Realm 데이터베이스 파일을 분석한 결과를 바탕으로 정상 데이터를 추출하고 삭제된 데이터를 복구하는 기법을 제안하였다. 그리고 사용자 행위를 고려한 시나리오를 작성하여 실험용 데이터베이스 파일을 생성한 후 본 논문에서 제안한 삭제된 레코드 복구 기법을 적용하는 실험을 진행하였다. 실험 결과 데이터베이스 파일에서 삭제된 레코드의 약 75%가 복구되었다. 따라서 본 논문에서 제안한 삭제된 레코드 복구 기법은 디지털 포렌식 조사 과정에서 Realm 데이터베이스 파일의 삭제된 레코드를 복구하여 사건 조사에 필요한 정보를 찾을 수 있을 것으로 기대한다.

References

〔1〕 Realm Blog, "Introducing Realm", https://realm.io/blog/introducing-realm/, July. 2014

〔2〕 DB-Engines, "DB-Engines Ranking", https://db-engines.com/en/ranking, May. 2018

〔3〕 Vijay Kumar, Mobile Database Systems, Wiley, July. 2006

〔4〕 OBJECTBOX, "Mobile databases: SQLite and SQLite alternatives for Android and iOS", http://objectbox.io/sqlite-alternatives/, Jan. 2017

〔5〕 Roderic Geoffrey Galton Cattell, The Object Database Standard: ODMG 3.0 1st Ed,, Morgan Kaufmann Publishers, Feb. 2000

〔6〕 Realm Blog, "Detailed study of Realm threads", https://academy.realm.io/kr/posts/threading-deep-dive/, Apr. 2016

〔7〕 Realm, "Realm Document", https://realm.io/kr/docs/, Oct. 2017

〔8〕 Realm Blog, "Year in Review: 2015", https://realm.io/kr/blog/2015-in-review/, Jan. 2016

〔9〕 Realm, "Realm Studio", https://realm.io/products/realm-studio, July. 2014

〈저자소개〉

김 준 기 (Junki Kim) 정회원
2016년 2월: 경희대학교 전자정보대학 컴퓨터공학과 공학사
2016년 9월~현재: 고려대학교 정보보호대학원 석사과정
〈관심분야〉 디지털 포렌식, 역공학, 시스템 보안

한 재 혁 (Jaehyeok Han) 학생회원
2011년 2월: 서울시립대학교 수학과 졸업
2016년 2월: 고려대학교 정보보호대학원 정보보호학과 공학석사
2016년 3월~현재: 고려대학교 정보보호대학원 정보보호학과 박사과정
〈관심분야〉 디지털 포렌식, 파일시스템, 해쉬함수, 데이터 마이닝

최 종 현 (Jong-Hyun Choi) 학생회원
2012년 2월: 경희대학교 전자정보대학 컴퓨터공학과 공학사
2014년 8월: 고려대학교 정보보호대학원 정보보호학과 석사
2014년 9월~현재: 고려대학교 정보보호대학원 정보보호학과 박사과정
〈관심분야〉 디지털 포렌식

이 상 진 (Sangjin Lee) 종신회원
1989년 10월~1999년 2월: ETRI 선임 연구원
1999년 3월~2001년 8월: 고려대학교 자연과학대학 조교수
2001년 9월~현재: 고려대학교 정보보호대학원 교수
2008년 3월~현재: 고려대학교 디지털포렌식연구센터 센터장
〈관심분야〉 디지털 포렌식, 심층암호, 해쉬함수