

303 – RealmDB Record Recovery

Team Information

Team Name : LuckyVicky

Team Member : Eungchang Lee, Hyun Yi, Juho Heo, Dongkyu Lee

Email Address : dfc_luckyvicky@googlegroups.com

Instructions

Description The provided file is a RealmDB database.

Target	Hash (MD5)
demo.realm	f5913c7bbb49e040b6c4e223cb3ecdb3

Questions

1. Recover all deleted records of the target. (30 points)
2. Develop a tool to recover deleted records of the target. (120 points)
3. Develop a tool to recover the RealmDB. (150 points)

Teams must:

- Develop and document the step-by-step approach used to solve this problem to allow another examiner to replicate team actions and results.
- Specify all tools used in deriving the conclusion(s).

Tools used:

Name:	HxD	Publisher:	Maël Hörz
Version:	2.5.0.0		
URL:	https://mh-nexus.de/en/hxd/		

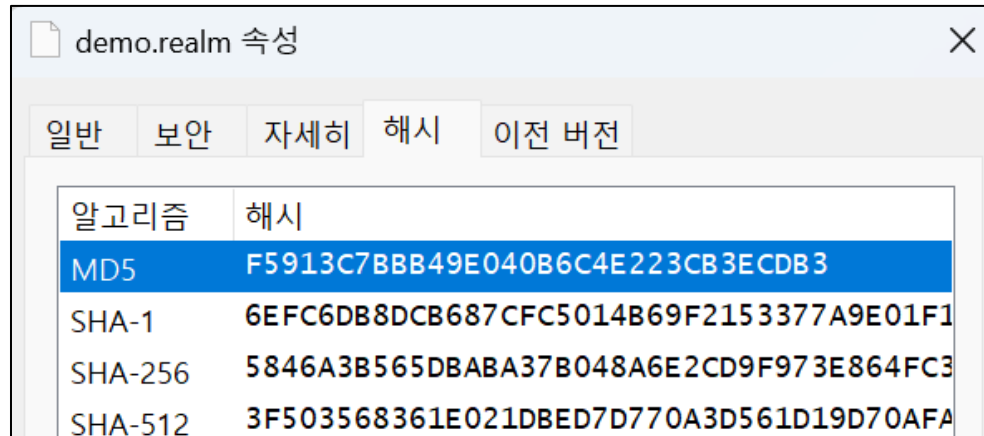
Name:	Python	Publisher:	Python Software Foundation
Version:	3.11.9		
URL:	https://www.python.org/		

Name:	Visual Studio	Publisher:	Microsoft
Version:	2022		
URL:	https://visualstudio.microsoft.com/ko/downloads/		

Name:	Visual Studio Code	Publisher:	Microsoft
Version:	1.93.1		
URL:	https://code.visualstudio.com/		

Name:	Realm Studio	Publisher:	Realm
Version:	3.10		
URL:	https://github.com/realm/realm-studio/releases		

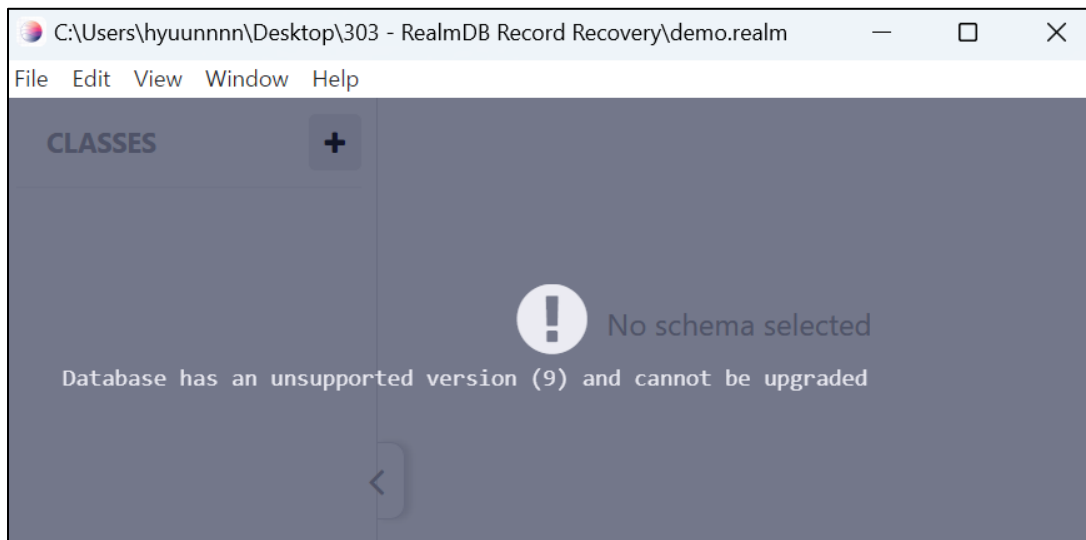
Step-by-step methodology:



알고리즘	해시
MD5	F5913C7BBB49E040B6C4E223CB3ECDB3
SHA-1	6EFC6DB8DCB687CFC5014B69F2153377A9E01F1
SHA-256	5846A3B565DBABA37B048A6E2CD9F973E864FC3
SHA-512	3F503568361E021DBED7D770A3D561D19D70AFA

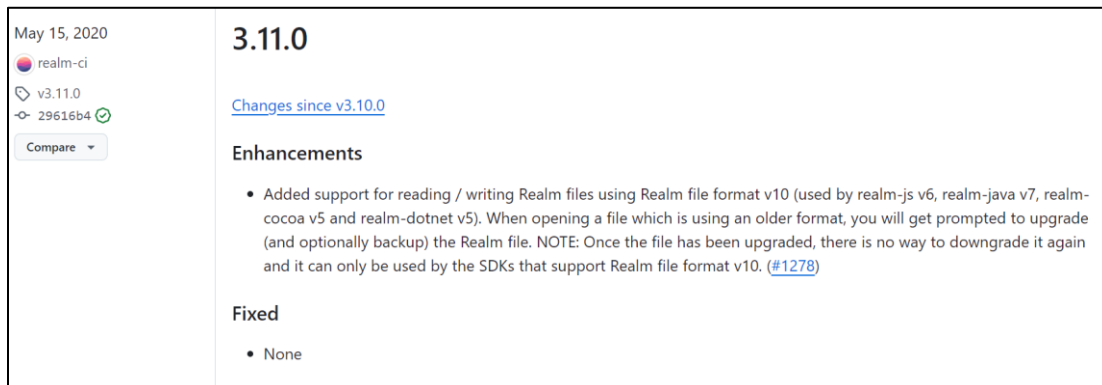
[그림 1] demo.realm 파일의 md5 해시 값 확인

문제로 주어진 demo.realm 파일의 MD5 해시 값이 일치함을 확인하였습니다.

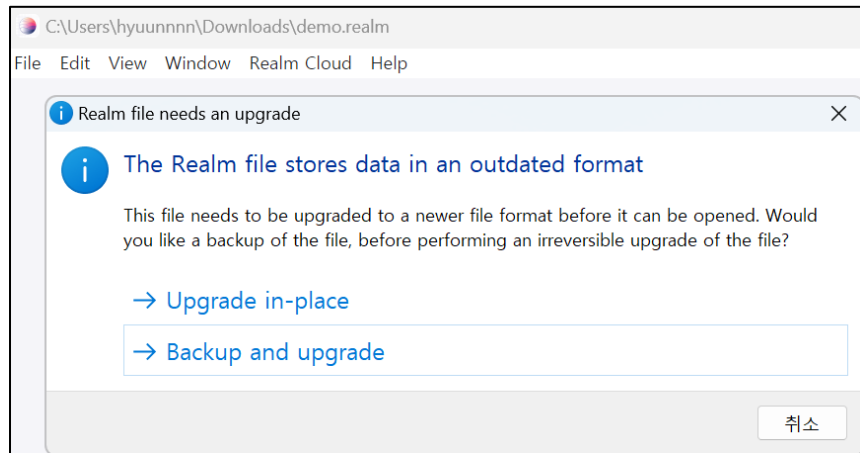


[그림 2] Realm Studio 15.2.1 버전에서 확인한 결과

24년 8월 31일 기준 최신 버전인 15.2.1 버전에서 문제 파일을 확인한 결과 [그림 2]와 같이 지원하지 않는 파일이라는 문구가 보입니다.

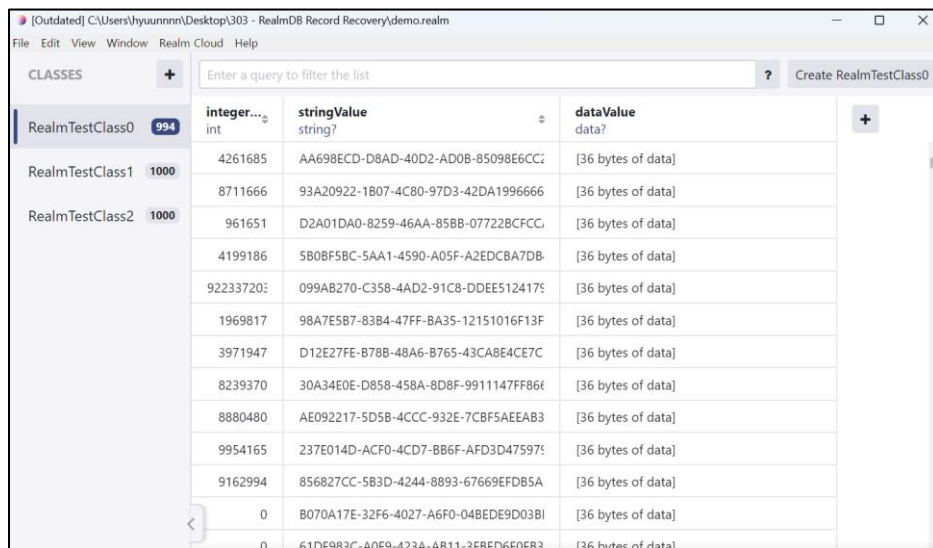


[그림 3] Realm Studio 3.11.0 버전의 릴리즈 노트



[그림 4] Realm Studio 3.11.0 버전에서 열린 결과

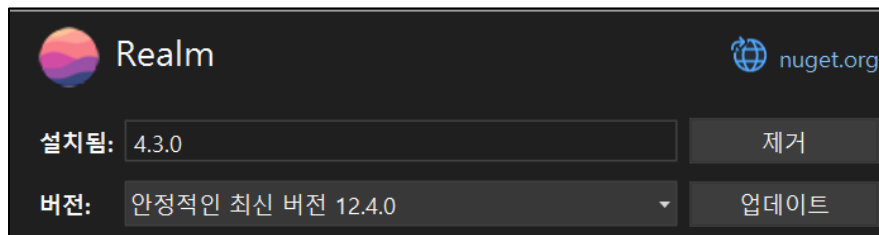
릴리즈 버전들을 전수조사한 결과 [그림 3]에서 볼 수 있는 릴리즈 노트를 통해 3.11부터는 Realm file format v10을 지원하며, 이전 버전의 포맷을 지원하는 마지막 버전인 3.10으로 열린 결과 [그림 5]와 같이 정상적으로 열리는 것을 확인할 수 있었습니다.



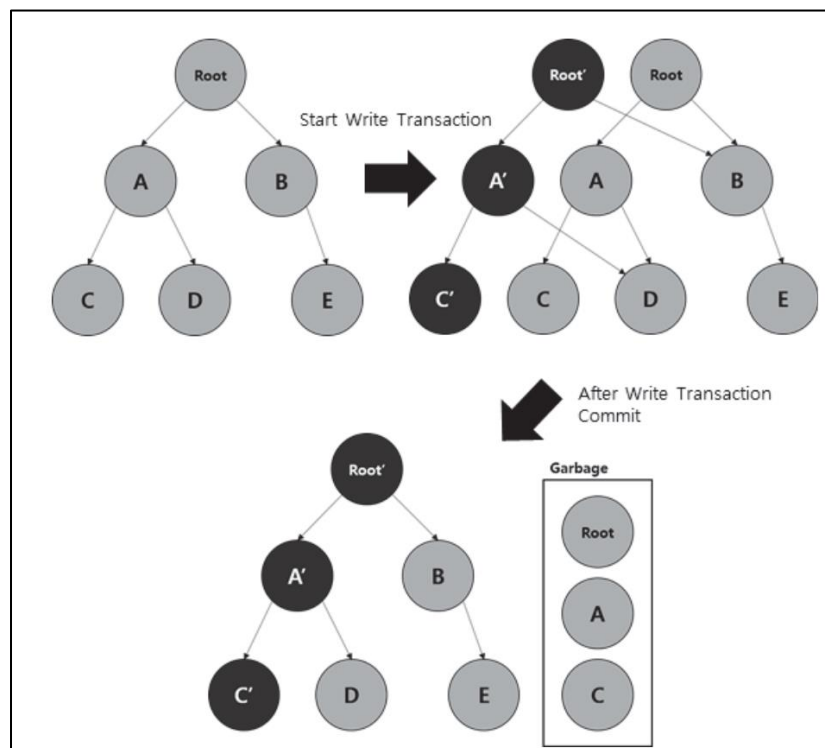
[그림 5] Realm Studio 3.10.0 버전에서 열린 결과

[그림 5]의 사진을 보면 RealmTestClass0, RealmTestClass1, RealmTestClass2 3개의 테이블이 존재하며, 각각 994, 1000, 1000개의 데이터를 확인할 수 있습니다. 또한 나머지 2개의 테이블에는 1000개의 데이터가 존재하지만 RealmTestClass0만 데이터가 994개 존재하는 것으로 보아 6개의 데이터가 삭제되었음을 예상할 수 있습니다.

삭제된 데이터를 복구하기 전에 Realm 데이터베이스의 구조를 이해하고 있어야 하며, 2개의 논문¹²을 참고하였습니다. 또한 테스트를 위한 DB 파일을 생성하기 위해 [그림 6]과 같이 구버전을 설치하였으며, Visual Studio의 “NuGet 패키지 관리” 기능을 통해 설치할 수 있습니다.



[그림 6] realm-dotnet 라이브러리 4.3.0 버전 설치



[그림 7] Copy On Write 방식의 동작 과정³

¹ 김준기 (2018). Realm 데이터베이스의 삭제된 레코드 복구 기법. 고려대학교 석사학위논문.

² Soram kim, Giyoon Kim, Sumin Shin, Byungchul Youn, Jian Song, Insoo Lee, Jongsung Kim (2022). Methods for recovering deleted data from the Realm database: Case study on Minitalk and Xabber. Forensic Science International: Digital Investigation Volume 40, March 2022, 301353.

³ 김준기 (2018). Realm 데이터베이스의 삭제된 레코드 복구 기법. 고려대학교 석사학위논문. 13p.

Realm 데이터베이스의 구조에 대해 설명하기 전에 Realm의 특징에 대해 간략히 짚고 넘어가겠습니다. Realm은 COW (Copy On Write) 방식을 사용하는데, 예를 들어 추가, 수정, 삭제 등의 이벤트가 발생했을 때 복사본에서 동작을 수행한 후 복사본의 데이터를 사용하는 방식입니다. 또한 행(row) 단위가 아닌 열(column) 단위로 데이터가 저장되며, 객체 지향 데이터 모델을 사용한다는 특징이 있습니다.

다음은 Realm 데이터베이스 구조에 대한 설명입니다.

	TreeRootOffset01	TreeRootOffset02	
00000000	80 E6 88 00 00 00 00 00	08 86 88 00 00 00 00 00	€æ^.....†^.....
00000010	54 2D 44 42 09 09 00 00	41 41 41 41 0E 00 00 05	T-DB...AAAA...
	File Signature	File Attribute	

[그림 8] Realm DB 구조

Realm은 원본과 복사본에 대한 두 개의 트리를 가지고 있습니다. 다음으로 File Signature 값에는 'T-DB'라는 문자가 존재하며, File Attribute의 하위 1바이트 값을 통해 Root Node를 선택하게 됩니다. 예를 들어 1바이트 값이 0인 경우 TreeRootOffset01을 참조하며, 1인 경우 TreeRootOffset02를 참조하는 방식입니다. [그림 8]은 0으로 설정되어 있기 때문에 TreeRootOffset01를 사용합니다.

	Object Signature	Table Information	Table Array	
0088E680	41 41 41 41 46 00 00 0A	18 00 00 00	D0 04 00 00	AAAAF.....Đ...
0088E690	01 00 20 01 A8 E3 88 00	F8 E4 88 00	48 E6 88 00	.. "ä^..øä^..Hæ^..
0088E6A0	0B 00 00 00 05 00 00 00	A8 05 00 00	01 00 00 00"

[그림 9] TreeRootOffset 구조

[그림 9]는 TreeRootOffset01 주소로 넘어간 사진이며, Object를 의미하는 시그니처 값인 AAAA와 Table Information, Table Array 주소 값이 존재합니다. AAAA 이후에 존재하는 4바이트 값은 추후에 설명할 예정이며, 해당 구간에서는 두 주소만 필요합니다.

	Object Signature	Flag	Count	
00000010	54 2D 44 42 09 09 00 00	41 41 41 41 0E	00 00 05	T-DB... AAAA...
00000020	70 6B 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00	pk.....
00000030	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00	1D.....
00000040	6D 65 74 61 64 61 74 61	00 00 00 00 00 00 00 00	00 00 00 00	metadata.....
00000050	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00	17.....
00000060	63 6C 61 73 73 5F 52 65	61 6C 6D 54 65 73 74 43		class_RealmTestC
00000070	6C 61 73 73 30 00 00 00	00 00 00 00 00 00 00 0A		lass0.....
00000080	63 6C 61 73 73 5F 52 65	61 6C 6D 54 65 73 74 43		class_RealmTestC
00000090	6C 61 73 73 31 00 00 00	00 00 00 00 00 00 00 0A		lass1.....
000000A0	63 6C 61 73 73 5F 52 65	61 6C 6D 54 65 73 74 43		class_RealmTestC
000000B0	6C 61 73 73 32 00 00 00	00 00 00 00 00 00 00 0A		lass2.....

[그림 10] Table Information 구조

Table Information은 현재 존재하는 테이블 이름을 확인할 수 있습니다.

Object Signature	Flag	Count	Table 01	Table 02	
000004D0	41 41 41 41	46 00 00 05	50 01 00 00	B0 01 00 00	AAAAF...P...°...
000004E0	50 02 00 00	C0 04 00 00	D0 F9 07 00	F0 2F 00 00	P...À...Đù..8/..
	Table 03	Table 04	Table 05		

[그림 11] Table Array 구조

Table Array에는 각 테이블의 Offset 주소를 확인할 수 있습니다. 또한 [그림 10]의 테이블 이름 개수와 [그림 11]의 테이블 offset 개수가 일치하는 것을 확인할 수 있습니다.

다음으로 AAAA 이후에 존재하는 4바이트 값의 의미에 대해 설명하겠습니다. 1바이트로 할당되어 있는 Flag 값을 통해 해당 Object에서 사용하는 크기를 알 수 있습니다.

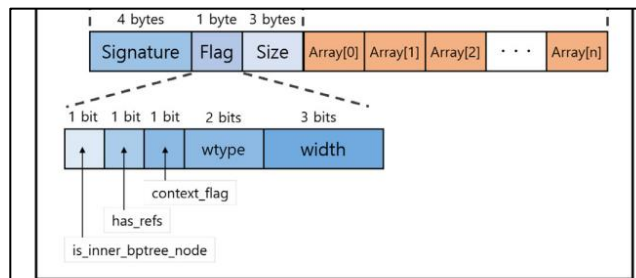


Fig. 2. Node structure.

Table 2

Maximum values of the representable widths for the bit width.

bit width	0	1	2	3	4	5	6	7
widths	0	1	2	4	8	16	32	64

Table 3

Calculation method of array data per wtype.

Value	Type	Size of array data (bytes)
0	bits	(width/8)*size
1	multiply	width*size
2	ignore	1*size

[그림 12] Flag 계산 방법⁴

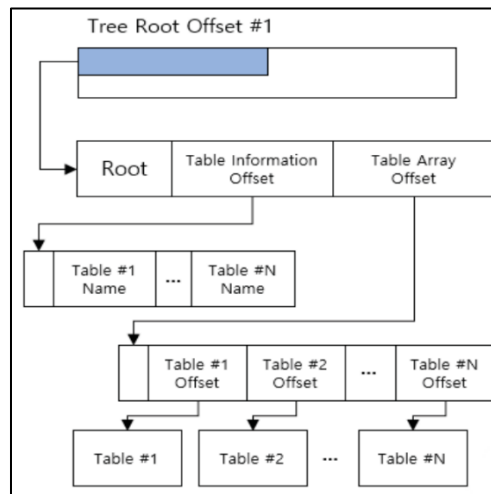
그러나 여러 테스트를 진행한 결과, Flag 값에 따라서 데이터의 타입이 매칭되는 것을 확인하였으며, 이는 컬럼 정보가 손상 또는 삭제되어 확인할 수 없는 경우에도 타입을 유추할 수 있습니다. 해당 결과는 [표 1]과 같습니다.

⁴ Soram kim, Giyeon Kim, Sumin Shin, Byungchul Youn, Jian Song, Insoo Lee, Jongsung Kim (2022). Methods for recovering deleted data from the Realm database: Case study on Minitalk and Xabber. Forensic Science International: Digital Investigation Volume 40, March 2022, 301353. 4p

Flag	Type
0x1	Boolean
0x4	Int8
0x5	Int16
0x6	Int32
0x7	Int64
0xB	Float
0xC	Double
0xD	String
0xE	String
0x11	String
0x45	Object (2 Bytes)
0x46	Object (4 Bytes)
0x65	Object (2 Bytes)
0x66	Object (4 Bytes)

[표 1] Flag 값에 따른 타입 매칭 결과

[표 1]의 방식으로 Flag에 따라서 데이터 타입이 매칭되며, 다음 3바이트 값은 길이 혹은 카운트로 사용됩니다.



[그림 13] TreeRootOffset에서 Table 객체에 접근하는 과정⁵

지금까지 TreeRootOffset에서 Table Information과 Table Array의 접근 과정을 시각화한 사진입니다.

⁵ 김준기 (2018). Realm 데이터베이스의 삭제된 레코드 복구 기법. 고려대학교 석사학위논문. 17p.

	Object Signature	Flag	Count	Table Schema Data Storage
00000250	41 41 41 41	45	00 00 02	28 02 38 02 00 00 00 00 AAAAE....(.8.....
00000260	41 41 41 41	05	00 00 08	B0 02 AD 02 53 00 80 01 AAAA....°...S.€.

[그림 14] Table 03 구조 – RealmTestClass0 테이블

[그림 14]는 [그림 11]의 테이블 Offset 중에서 3번째 테이블에 접근한 결과이며, 역시 이전에 설명했던 Object 구조와 동일함을 확인할 수 있습니다. 또한 [그림 11]은 Flag 값이 0x46이기 때문에 4바이트씩 사용했지만, [그림 14]는 0x45이며 2바이트씩 사용하는 것을 확인할 수 있습니다.

	Object Signature	Flag	Count	
00000220	D7 00 00 00 00 00 00 00	41 41 41 41	45	00 00 04 *.....AAAE...
00000230	C0 01 D0 01	08 02 18 02	41 41 41 41	46 00 00 04 À.Đ.....AAAF...
	Column Type	Column Name		

[그림 15] Table Schema 구조

Table Schema에서는 Column Type과 Column Name 정보를 확인할 수 있으며, Column Type 값의 의미는 논문⁶을 참고하였습니다. 또한 Column Name 다음에 해당하는 offset은 Column Option이나 해당 정보는 활용 가치가 없다고 판단하여 제외하였습니다.

Column Type	Type
0x0	Integer
0x1	Bool
0x2	String
0x4	Binary
0x5	Table
0x6	Mixed
0x7	OldDateTime
0x8	Timestamp
0x9	Float
0xA	Double
0xC	Link
0xD	LinkList

[표 2] Column Type 설명

⁶ 김준기 (2018). Realm 데이터베이스의 삭제된 레코드 복구 기법. 고려대학교 석사학위논문. 20p.

	Object Signature	Flag	Count	Data
000001C0	41 41 41 41	03	00 00 04	20 E4 00 00 00 00 00 00 AAAA....ä.....
000001D0	41 41 41 41	0D	00 00 03	69 6E 74 65 67 65 72 56 AAAA....integerV

[그림 16] Column Type 구조

Column Type의 데이터는 binary로 변환한 후 count 만큼 shift 연산을 수행하여 최종적으로 [표 2]에 대입할 수 있는 값이 나오게 됩니다.

```
C:\Users\hyuunnn>python
Python 3.11.9 (tags/v3.11.9:de54cf5, Apr 2 2024, 10:12:12) [MSC v.1938 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> flag = int.from_bytes(b"\x20\xe4\x00\x00\x00\x00\x00\x00", "little")
>>> [hex((flag >> i) & 0xF) for i in range(0, flag.bit_length(), 4)]
['0x0', '0x2', '0x4', '0xe']
>>> |
```

[그림 17] Column Type 데이터 변환 결과

RealmTestClass0	integer...	stringValue	dataValue
994	int	string?	data?
RealmTestClass1	1000		
	4261685	AA698ECD-D8AD-40D2-AD0B-85098E6CC2	[36 bytes of data]
	8711666	93A20922-1B07-4C80-97D3-42DA1996666	[36 bytes of data]

[그림 18] RealmTestClass0 테이블 컬럼명 확인

해당 테이블의 Column Type은 Integer, String, Binary임을 확인할 수 있으며, [그림 18]과 동일함을 확인할 수 있습니다. 그러나 0xE 값을 추가로 확인할 수 있는데, 테스트 파일을 만들어서 확인한 결과 삭제 작업을 수행하지 않은 파일에서도 동일한 현상이 발생하여 넘어가는 것으로 진행하였습니다.

	Object Signature	Flag	Count	Data
000001D0	41 41 41 41	0D	00 00 03	69 6E 74 65 67 65 72 56 AAAA....integerV
000001E0	61 6C 75 65	00	00 00 03	73 74 72 69 6E 67 56 61 alue....stringVa
000001F0	6C 75 65 00	00	00 04	64 61 74 61 56 61 6C 75 lue....dataValu
00000200	65 00 00 00	00	00 06	41 41 41 41 04 00 00 04 e....AAAA....

[그림 19] Column Name 구조

[그림 19]를 보면 3개의 데이터와 [표 1]에서의 Object Type과 일치하는 것을 확인할 수 있습니다.

	Object Signature	Flag	Count	
00000230	C0 01 D0 01	08	02 18 02	41 41 41 41 46 00 00 04 À.Đ....AAAAF...
00000240	38 86 88 00	50	A5 88 00	E0 B4 88 00 70 C4 88 00 8t^..Pÿ^..ä^..pÄ^..
	Column 01	Column 02	Column 03	Column 04

[그림 20] Data Storage 구조

마지막으로 Data Storage에는 각 Column에 해당하는 데이터들의 Offset을 확인할 수 있으며, [그림 21]은 그 중에서 첫 번째 Column의 Offset에 접근하였습니다.

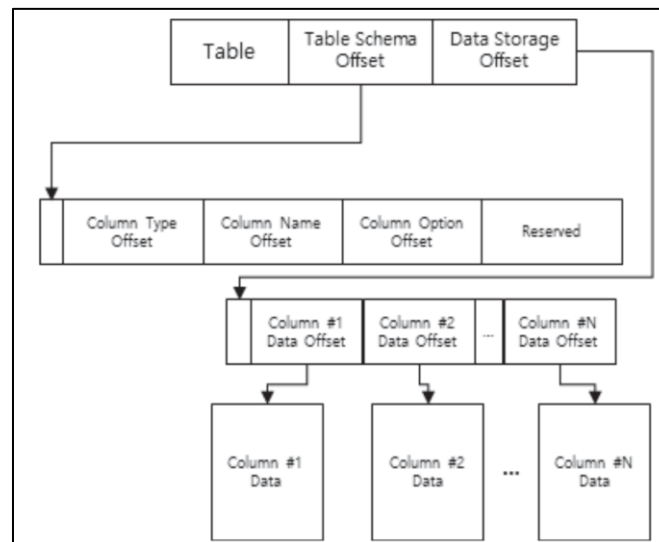
	Object Signature								Flag	Count	
00888630	50	FF	07	00	01	00	00	00	41 41 41 41	07 00 03 E2	Pÿ.....AAAA...â
00888640	35	07	41	00	00	00	00	00	F2 ED 84 00	00 00 00 00	5.A.....öi.....
00888650	73	AC	0E	00	00	00	00	00	12 13 40 00	00 00 00 00	5.....@.....
Values											

[그림 21] Column 01 구조 - integerValue

>>>			integer...
>>> 0x410735	RealmTestClass0	994	int
4261685			
>>> 0x84edf2	RealmTestClass1	1000	4261685
8711666			
>>> 0xeac73	RealmTestClass2	1000	8711666
961651			
>>> 0x401312			961651
4199186			
>>>			4199186

[그림 22] hex 값과 Realm Studio의 결과

해당 Column에는 994(0x3e2)개의 데이터가 존재하며, Object Type이 0x7이기 때문에 8바이트 단위로 존재합니다. 또한 [그림 22]의 결과처럼 동일한 값을 확인할 수 있었습니다.



[그림 23] Table 객체에서 Column 값에 접근하는 과정⁷

⁷ 김준기 (2018). Realm 데이터베이스의 삭제된 레코드 복구 기법. 고려대학교 석사학위논문. 19p.

```
[*] Table Information 일치
- [bytearray(b'__class_RealmTestClass0'), bytearray(b'__class_RealmTestClass1'), bytearray(b'__class_RealmTestClass2')]

[*] Table 개수 일치 - 3개

[*] 1번째 Table Schema 일치
- Table 1 Schema: [['Integer', 'String', 'Binary', 'Unknown'], [bytearray(b'integerValue'), bytearray(b'stringValue'), bytearray(b'dataValue')]]

[*] 1번째 Table Data Storage 불일치
- TreRootOffset01에 있으나 TreRootOffset02에 없는 데이터: []
- TreRootOffset02에 있으나 TreRootOffset01에 없는 데이터: [8535638, 9223372036854775807, 0]

- TreRootOffset01에 있으나 TreRootOffset02에 없는 데이터: []
- TreRootOffset02에 있으나 TreRootOffset01에 없는 데이터: [b'E9AB7C66-44F9-4C55-909D-0749548E043C', b'7FBDD08F-11B7-486A-9989-8BD3D7A1D742', b'65477902-952F-4439-96EF-8904E3D0E722']

- TreRootOffset01에 있으나 TreRootOffset02에 없는 데이터: []
- TreRootOffset02에 있으나 TreRootOffset01에 없는 데이터: [b'1E8F8FDE-C986-41E9-AD99-99EA8F428799', b'8FBA0FB3-1703-406A-9B35-AD8013548C55', b'D5140106-CE34-40B5-856B-D3457E8A8375']
```

구조를 따라가면서 데이터를 추출하는 코드를 작성한 후 시작 지점을 `TreeRootOffset01`과 `TreeRootOffset02`로 설정했을 때 데이터가 일치하는지 확인한 결과, **RealmTestClass0** 테이블에서 일치하지 않음을 확인할 수 있었습니다. 결국 삭제된 3개의 데이터를 확보할 수 있었습니다.

```
scan_unused_objects.txt U X
realm_recover > scan_unused_objects.txt
21 Offset: 0xc0f0, Type: 0x11, Count: 36, Object: b'F04CB00E-C2B5-4709-995C-B016CC8FCFFA'
22 Offset: 0xc120, Type: 0x11, Count: 36, Object: b'DC8AD53F-85B5-4C16-8398-A5C8273263FD'
23 Offset: 0x10000, Type: 0x7, Count: 1000, Object: [9056741, 6702896, 6088682, 4199186, 9223372036854775807, 1969817, 3971947, 8239370, 8880480, 9954165, 9162994
```

사용하지 않는 Object를 저장한 scan_unused_objects.txt 파일을 확인한 결과, 1000개의 데이터가 존재하는 것을 확인하였습니다.

[그림 26] TreeRootOffset을 통해 접근하는 데이터를 추출하는 과정

[그림 24]에서 확인했듯이 사용 중인 TreeRootOffset을 통해 접근할 수 있는 994개의 정상 Object와 사용하지 않는 나머지 하나의 TreeRootOffset을 통해 접근 가능한 997개의 Object는 AAAA 시그니처를 기반으로 전수조사한 결과 파일인 scan_all_objects.txt에서 추출하였으며, 마지막으로 [그림 25]에서 추출한 1000개의 데이터를 비교하였습니다.

```
Offset: 0x10000, Type: 0x7, Count: 1000, Object: [9056741, 6702896, 6088682, 4199186, 9223372036854775807, 1969817, 3971947, 8239370, 8880480, 9954165, 9162994,
Offset: 0x882628, Type: 0x7, Count: 997, Object: [4261685, 8711666, 961651, 4199186, 9223372036854775807, 1969817, 3971947, 8239370, 8880480, 9954165, 9162994,
Offset: 0x888638, Type: 0x7, Count: 994, Object: [4261685, 8711666, 961651, 4199186, 9223372036854775807, 1969817, 3971947, 8239370, 8880480, 9954165, 9162994,
```

[그림 27] 추가로 찾은 integerValue에 수정된 데이터

```
, 7866083, 9223372036854775807, 5083841, 3771840, 0, 9223372036854775808, 6450167, 1390034, 8535638, 9223372036854775807, 0, 961651, 8711666, 4261685]
7866083, 9223372036854775807, 5083841, 3771840, 0, 9223372036854775808, 6450167, 1390034, 8535638, 9223372036854775807, 0]
7866083, 9223372036854775807, 5083841, 3771840, 0, 9223372036854775808, 6450167, 1390034]
```

[그림 28] 추가로 찾은 integerValue에 삭제된 데이터

3개의 리스트에 저장된 데이터들은 매우 유사했으며, 주변 데이터를 확인한 결과, 대부분의 데이터가 일치했지만 일부 삭제 및 수정된 것으로 보이는 데이터를 확보할 수 있었습니다.

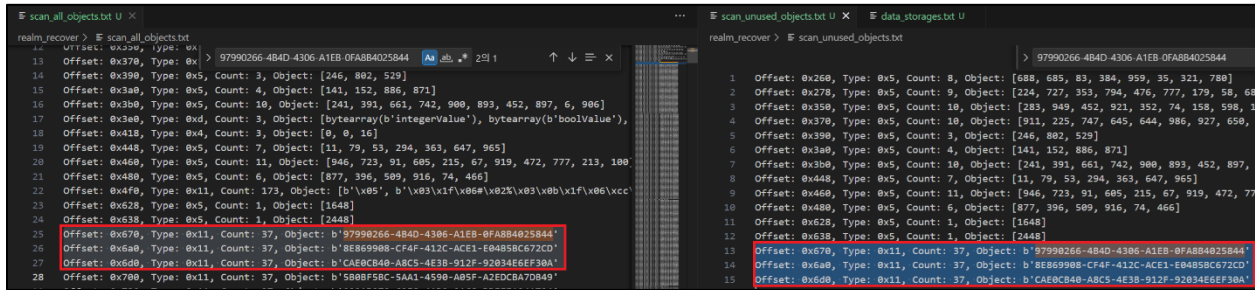
RealmTestClass0	integer...	stringValue
994	int	string?
RealmTestClass1	1000	4261685
RealmTestClass2	1000	8711666
		961651
		4199186
	922337203	099AB270-C358-4AD2-91C8-DDEE51241794
	1969817	98A7E5B7-83B4-47FF-BA35-12151016F13F
	3971947	D12E27FE-B78B-48A6-B765-43CA8E4CE7C
	8239370	30A34E0E-D858-458A-8D8F-9911147FF866

[그림 29] 추가로 찾은 stringValue에 수정된 데이터 - 1

RealmTestClass0	integer...	stringValue
994	int	string?
RealmTestClass1	1000	4261685
RealmTestClass2	1000	8711666
		961651
	4199186	5B0BF5BC-5AA1-4590-A05F-A2EDCBA7DB49
	922337203	099AB270-C358-4AD2-91C8-DDEE51241794

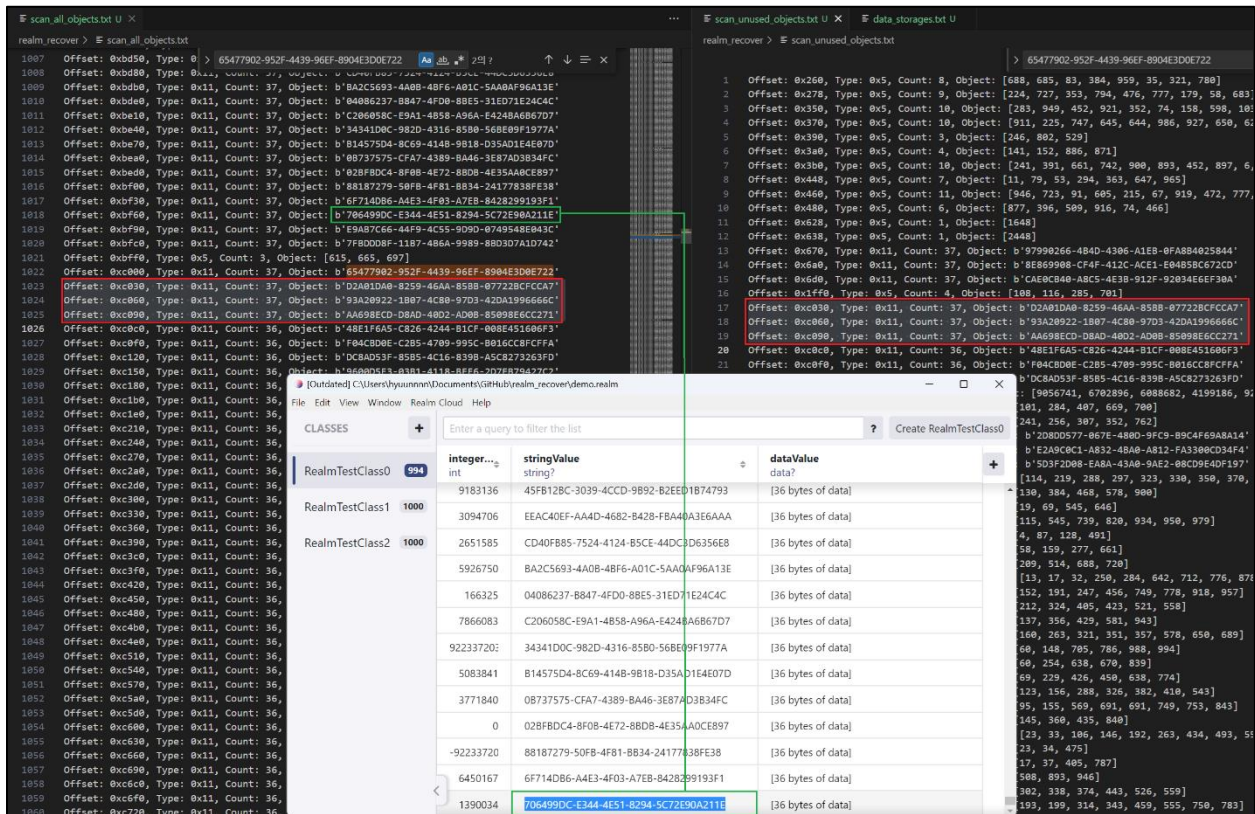
[그림 30] 추가로 찾은 stringValue에 수정된 데이터 - 2

[그림 30]에서 확인할 수 있는 3개의 데이터는 Realm Studio에서 보여지는 값임을 확인할 수 있습니다. 그러나 AAAA 시그니처를 기반으로 찾은 [그림 29]의 왼쪽 결과를 보면 1, 2, 3번째 값이 다른 것을 확인할 수 있습니다. 기존 데이터는 수정하지 않고, 다른 영역에 데이터를 추가한 후 정상 Object를 가리키는 Offset만 수정된 것으로 볼 수 있습니다.



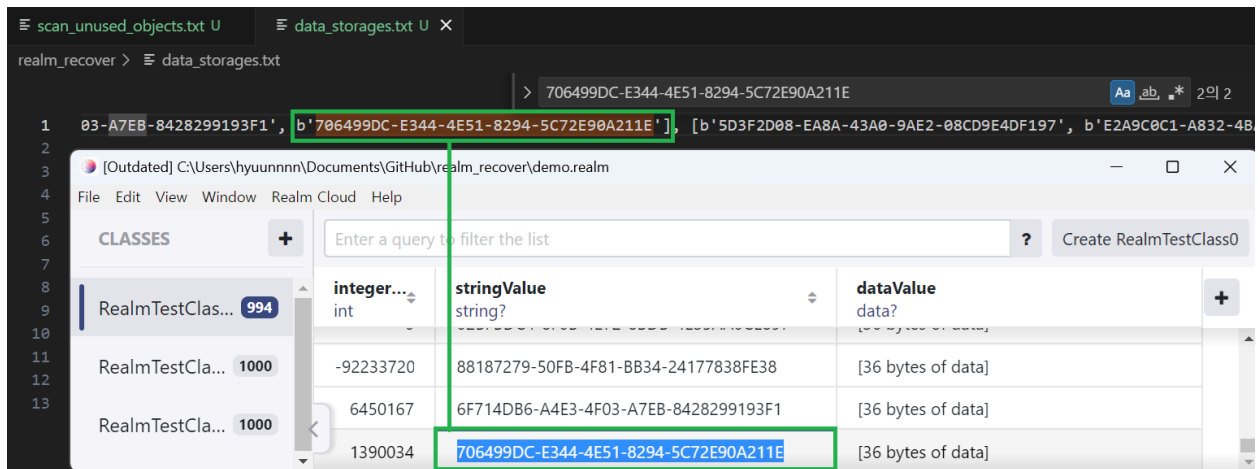
[그림 31] 3개의 데이터가 사용되지 않음을 검증하는 과정

또한 scan_unused_objects.txt 파일을 통해 동일한 Offset에 해당하는 Object가 사용하지 않음을 검증할 수 있었습니다.



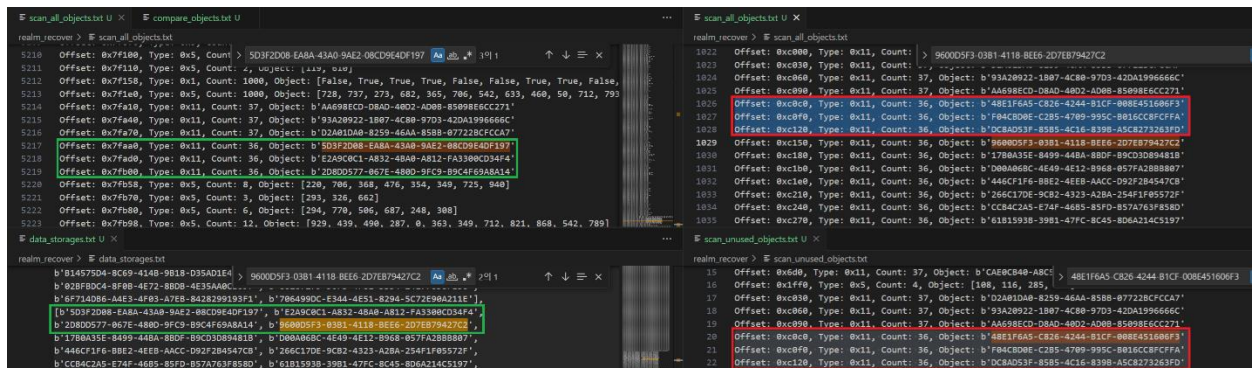
[그림 32] 추가로 찾은 stringValue에 삭제된 데이터

[그림 24]에서 stringValue에 삭제된 데이터 중 하나인 "65477902-952F-4439-96EF-8904E3D0E722" 값을 scan_all_objects.txt 파일에서 검색한 결과, [그림 32]와 같이 "706499DC-E344-4E51-8294-5C72E90A211E" 값 이후에 기존에 [그림 24]에서 확인한 삭제 데이터 3개와 추가로 빨간 박스에서 확인할 수 있듯이 3개의 데이터가 삭제되었음을 확인할 수 있었습니다. 또한, 분석가의 판단 하에 데이터 길이에 따른 특징에서 37과 36 count가 지나는 데이터가 다를음을 확인했으므로 "AA698ECD-D8AD-40D2-AD0B-85098E6CC271" 값 이후에는 다른 데이터임을 검증할 수 있었습니다.



[그림 33] dataValue에 저장된 데이터를 확인하는 과정

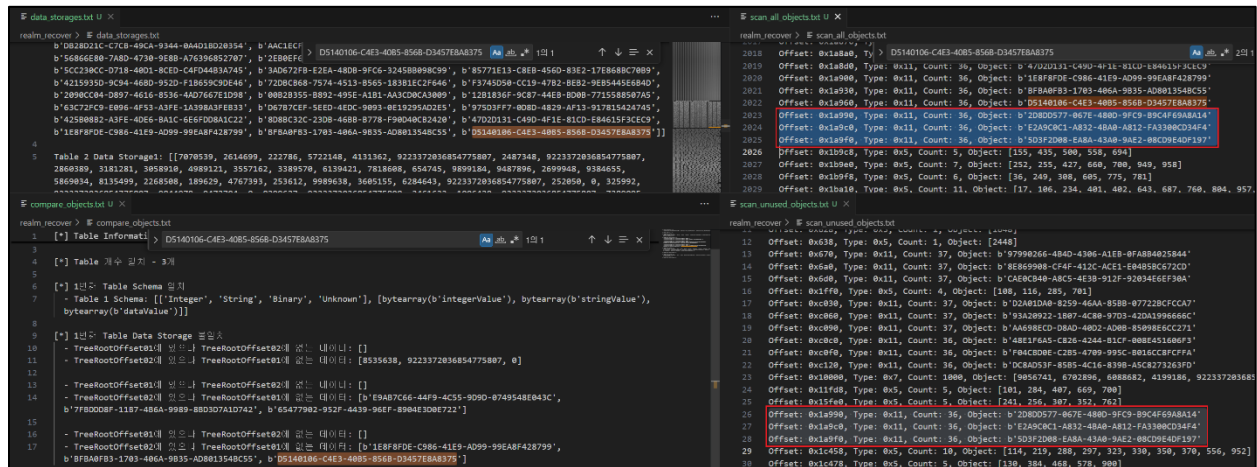
[그림 33]은 정상 데이터를 추출한 data_storages.txt 파일 결과이며, Realm Studio에서는 data의 길 이만 보여주고 있지만, 위 리스트 결과를 통해 dataValue의 1번째 값은 "5D3F2D08-EA8A-43A0-9AE2-08CD9E4DF197"임을 확인할 수 있습니다.



[그림 34] 추가로 찾은 dataValue에 수정된 데이터

또한 [그림 34]의 왼쪽 결과와 같이 data_storages.txt와 scan_all_objects.txt에서 "5D3F2D08-EA8A-43A0-9AE2-08CD9E4DF197"을 검색했을 때 최상단에 위치하는 3개의 데이터가 다른 영역에 존재함을 확인할 수 있습니다.

[그림 34]의 오른쪽 결과와 같이 다음으로 존재하는 "9600D5F3-03B1-4118-BEE6-2D7EB79427C2"을 scan_all_objects.txt, scan_unused_objects.txt에서 검색했을 때 사용하지 않는 3개의 데이터를 확보할 수 있었으며, [그림 30]과 동일한 패턴임을 확인할 수 있었습니다.



[그림 35] 추가로 찾은 dataValue에 삭제된 데이터

마지막으로 이전에 찾았던 데이터를 검색하여 해당 영역 주변에서 3개의 데이터를 추가로 찾을 수 있었습니다. 결과적으로 Realm Studio를 통해 확인한 **RealmTestClass0** 테이블의 데이터가 일부 삭제 되었음을 확인하였으며, 994개에서 삭제된 데이터 6개와 수정된 데이터 3개를 찾을 수 있었습니다.

1. Recover all deleted records of the target. (30 points)

integerValue	stringValue	dataValue
8535638	E9AB7C66-44F9-4C55-9D9D-0749548E043C	1E8F8FDE-C986-41E9-AD99-99EA8F428799
9223372036854775807	7FBDDD8F-11B7-4B6A-9989-8BD3D7A1D742	BFBA0FB3-1703-406A-9B35-AD801354BC55
0	65477902-952F-4439-96EF-8904E3D0E722	D5140106-C4E3-40B5-856B-D3457E8A8375
961651	D2A01DA0-8259-46AA-85BB-07722BCFCCA7	2D8DD577-067E-480D-9FC9-B9C4F69A8A14
8711666	93A20922-1B07-4C80-97D3-42DA1996666C	E2A9C0C1-A832-4BA0-A812-FA3300CD34F4
4261685	AA698ECD-D8AD-40D2-AD0B-85098E6CC271	5D3F2D08-EA8A-43A0-9AE2-08CD9E4DF197

[표 3] 삭제된 데이터 복구 결과

integerValue	index	stringValue	dataValue
9056741	1	97990266-4B4D-4306-A1EB-0FA8B4025844	48E1F6A5-C826-4244-B1CF-008E451606F3
6702896	2	8E869908-CF4F-412C-ACE1-E04B5BC672CD	F04CBD0E-C2B5-4709-995C-B016CC8FCFFA
6088682	3	CAE0CB40-A8C5-4E3B-912F-92034E6EF30A	DC8AD53F-85B5-4C16-839B-A5C8273263FD

[표 4] 수정된 데이터 복구 결과

2. Develop a tool to recover deleted records of the target. (120 points)

3. Develop a tool to recover the RealmDB. (150 points)

RealmDB를 범용적으로 복구하는 도구를 개발하였기 때문에 두 지문을 묶어 아래 개발한 도구의 동작 과정과 사용 방법 및 결과에 대해 설명하도록 하겠습니다. 논문에서 제안한 방법은 TreeRootOffset을 통해 추출할 수 있는 정상 데이터를 제외한 모든 데이터를 삭제된 데이터라고 가정한 후 AAAA 시그니처를 기반으로 전수조사를 진행하여 비교하는 방법입니다. 이때 정상적으로 추출한 영역의 주변 데이터와 전수조사한 데이터를 비교하여 어떤 테이블에 있던 데이터인지 유추할 수 있습니다.

```
C:\Users\hyuunnn\Documents\GitHub\realm_recover>dir
C 드라이브의 볼륨에는 이름이 없습니다.
볼륨 일련 번호: DA8C-5E04

C:\Users\hyuunnn\Documents\GitHub\realm_recover 디렉터리

2024-09-13 오후 10:25 <DIR> .
2024-09-10 오후 04:37 <DIR> ..
2024-08-02 오전 02:55      9,437,184 demo.realm
2024-09-13 오후 08:44      6,556 objects.py
2024-09-10 오후 04:38        93 README.md
2024-09-13 오후 08:44      6,978 realm_recover.py
2024-09-09 오전 01:22      1,697 util.py
                5개 파일          9,452,508 바이트
                2개 디렉터리 15,840,296,960 바이트 남음

C:\Users\hyuunnn\Documents\GitHub\realm_recover>python realm_recover.py -h
usage: realm_recover.py [-h] --file FILE

RealmDB Recovery Tool

options:
  -h, --help  show this help message and exit
  --file FILE filepath
```

[그림 36] 현재 폴더에 존재하는 파일 목록 및 realm_recover.py 실행 결과

현재 폴더에는 문제로 주어진 demo.realm 파일과 3개의 py 파일이 존재합니다. 그 중에서 realm_recover.py를 실행하면 되며, -h 옵션을 사용하면 상세한 옵션 설명을 출력해줍니다. 파이썬 버전은 3.11.9를 사용하고 있습니다.

```
C:\Users\hyuunnn\Documents\GitHub\realm_recover>python realm_recover.py --file demo.realm

C:\Users\hyuunnn\Documents\GitHub\realm_recover>dir
C 드라이브의 볼륨에는 이름이 없습니다.
볼륨 일련 번호: DA8C-5E04

C:\Users\hyuunnn\Documents\GitHub\realm_recover 디렉터리

2024-09-13 오후 10:25 <DIR> .
2024-09-10 오후 04:37 <DIR> ..
2024-09-13 오후 10:26      1,733 compare_objects.txt
2024-09-13 오후 10:26    1,032,660 data_storages.txt
2024-08-02 오전 02:55      9,437,184 demo.realm
2024-09-13 오후 10:26      6,497 objects.py
2024-09-10 오후 04:38        93 README.md
2024-09-13 오후 08:44      6,978 realm_recover.py
2024-09-13 오후 10:26    1,817,739 scan_all_objects.txt
2024-09-13 오후 10:26    1,133,360 scan_unused_objects.txt
2024-09-09 오전 01:22      1,697 util.py
2024-09-13 오후 10:26 <DIR> __pycache__
                9개 파일          13,437,941 바이트
                3개 디렉터리 15,833,882,624 바이트 남음
```

[그림 37] realm_recover.py 분석 결과

--file 옵션을 사용하여 realm 파일을 지정한 후 코드를 실행하면 TreeRootOffset01과 TreeRootOffset02의 비교 결과가 compare_objects.txt에 저장되며, 데이터의 경우 방대하기 때문에 data_storages.txt 파일로 따로 분리하여 저장하였습니다. 또한 AAAA 시그니처를 통해 전수조사한 Object는 scan_all_objects.txt, TreeRootOffset01, 02에서 시작하여 Offset에 접근하는 과정에서 사용하지 않았던 Object는 scan_unused_objects.txt에 저장하였습니다.

```
def scan_all_objects(self, used_offsets):
    EXCLUDED_TYPES = {0x45, 0x46, 0x65, 0x66}

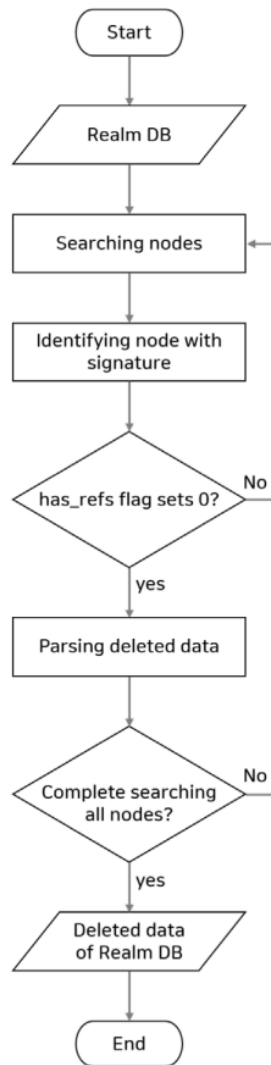
    offsets = self._scan_for_signature()
    all_objects = []
    unused_objects = []

    for offset in offsets:
        parser = ObjectParser(self._buf, offset)
        if parser.object_type in EXCLUDED_TYPES:
            continue

        try:
            obj = parser.parse_object()
            if parser.c > 0:
                all_objects.append((offset, parser.object_type, parser.c, obj))
                if offset not in used_offsets:
                    unused_objects.append((offset, parser.object_type, parser.c, obj))
        except ValueError as e:
            pass
```

[그림 38] scan_all_objects 코드

해당 코드를 보면 [표 1]에서 Object라고 명시했던 0x45, 0x46, 0x65, 0x66을 제외한 데이터만 가져오고 있는데, 해당 Object는 바이너리에 존재하는 또 다른 Offset에 접근하여 데이터를 가져오기 때문입니다. 이러한 내용은 [그림 39]에서 확인할 수 있습니다.



(b) Case 1-2

[그림 39] 논문에서 설명하는 Case 1-2 순서도⁸

0x45, 0x46, 0x65, 0x66은 2진법으로 변환했을 때 모두 [그림 12]에서 확인할 수 있는 has_refs 값이 1이며, 위 순서도를 보면 has_refs 값이 0인 경우에 데이터를 추출하는 것을 확인할 수 있습니다.

⁸ Soram kim, Giyoon Kim, Sumin Shin, Byungchul Youn, Jian Song, Insoo Lee, Jongsung Kim (2022). Methods for recovering deleted data from the Realm database: Case study on Minitalk and Xabber. Forensic Science International: Digital Investigation Volume 40, March 2022, 301353. 7p

또한 TreeRootOffset01, 02를 비교했던 방법처럼 테이블 매칭까지 진행하려고 하였으나, row 단위가 아닌 column 단위로 저장되기 때문에 AAAA 시그니처를 기반으로 전수조사를 진행했을 때 추출한 데이터가 어느 컬럼에 해당되는지 매칭할 수 없었습니다. 이와 관련된 내용은 논문⁹에서도 확인할 수 있습니다. 결국 수동 작업을 최소화하기 위해 도구를 사용하여 유의미한 데이터들을 제공받은 후 이를 수동으로 확인하여 삭제된 데이터를 복구해야 합니다.

```
[*] Table Information 일치
- [bytearray(b'class_RealmTestClass0'), bytearray(b'class_RealmTestClass1'), bytearray(b'class_RealmTestClass2')]

[*] Table 개수 일치 - 3개

[*] 1번째 Table Schema 일치
- Table 1 Schema: [['Integer', 'String', 'Binary', 'Unknown'], [bytearray(b'integerValue'), bytearray(b'stringValue'), bytearray(b'dataValue')]]

[*] 1번째 Table Data Storage 불일치
- TreeRootOffset01에 있으나 TreeRootOffset02에 없는 데이터: []
- TreeRootOffset02에 있으나 TreeRootOffset01에 없는 데이터: [8535638, 9223372036854775807, 0]

- TreeRootOffset01에 있으나 TreeRootOffset02에 없는 데이터: []
- TreeRootOffset02에 있으나 TreeRootOffset01에 없는 데이터: [b'E9AB7C66-44F9-4C55-9D9D-0749548E043C', b'7FBD0D8F-11B7-4B6A-9989-8BD3D7A1D742', b'65477982-952F-4439-96EF-8904E3D0E722']

- TreeRootOffset01에 있으나 TreeRootOffset02에 없는 데이터: []
- TreeRootOffset02에 있으나 TreeRootOffset01에 없는 데이터: [b'1E8F8FDE-C986-41E9-AD99-99EA8F428799', b'BFBA8FB3-1703-406A-9B35-AD8013548C55', b'D5140106-C4E3-40B5-8568-D3457E8A8375']

[*] 2번째 Table Schema 일치
- Table 2 Schema: [['Integer', 'Bool', 'Float', 'Double', 'String', 'Timestamp', 'Linklist', 'Unknown'], [bytearray(b'integerValue'), bytearray(b'boolValue'), bytearray(b'floatValue'), bytearray(b'doubleValue'), bytearray(b'stringValue'), bytearray(b'dateValue'), bytearray(b'arrayReference')]]

[*] 2번째 Table Data Storage 일치

[*] 3번째 Table Schema 일치
- Table 3 Schema: [['Integer', 'Bool', 'Link'], [bytearray(b'integerValue'), bytearray(b'boolValue'), bytearray(b'objectReference')]]

[*] 3번째 Table Data Storage 일치
```

[그림 40] compare_objects.txt 파일 내용

```
Table 1 Data Storage1: [[4261685, 8711666, 961651, 4199186, 9223372036854775807, 1969817, 3971947, 8239370, 8880480, 9954165, 9162994, 3699782, 6320186, 8243821, 8011812, 6109418, 3916298, 9650233, 2076182, 9093114, 7810149, 247112, 9983807, 3245481, 0, 9223372036854775807, 9223372036854775808, 0, 9223372036854775807, 1108364, 6746293, 4958739, 1260619, 887494, 4546819, 823900, 9223372036854775807, 4943122, 9223372036854775808, 9408718, 2218044, 0, 5416467, 5750089, 7489715, 8300540, 9699394, 6971533, 9580744, 9176257, 9054082, 5374088, 882, 9223372036854775808, 4488782, 0, 4655724, 5538232, 628636, 4736205, 0, 4983592, 4925972, 2053093, 616417, 1405001, 287123, 4075297, 0, 5522910, 676692, 7312571, 9223372036854775808, 0, 9223372036854775808, 4891047, 4832109, 3865702, 1427890, 1714694, 9223372036854775807, 2554361, 517823, 5183262, 1586846, 9223372036854775807, 8920823, 1550880, 6338246, 0, 1334814, 4198648, 6743587, 9223372036854775808, 9, 5447666, 9040412, 5594167, 2323868, 9311463, 0, 4496152, 6491064, 307502, 4469045, 4924171, 3404134, 4955370, 8551238, 8585153, 224546,
```

[그림 41] data_storages.txt 파일 내용 중 일부

```
1 Offset: 0x18, Type: 0xe, Count: 5, Object: [bytearray(b'pk'), bytearray(b'metadatas'), bytearray(b'class_RealmTestClass0'), bytearray(b'class_RealmTestClass1'), bytearray(b'class_RealmTestClass2')]
2 Offset: 0xd0, Type: 0xd, Count: 2, Object: [bytearray(b'pk_table'), bytearray(b'pk_property')]
3 Offset: 0xf8, Type: 0x1, Count: 2, Object: [True]
4 Offset: 0x120, Type: 0x6, Count: 0, Object: []
5 Offset: 0x168, Type: 0xc, Count: 1, Object: [1.35441148275183e-306]
6 Offset: 0x190, Type: 0x4, Count: 1, Object: [0]
7 Offset: 0x1d0, Type: 0xd, Count: 3, Object: [bytearray(b'integerValue'), bytearray(b'stringValue'), bytearray(b'dataValue')]
8 Offset: 0x208, Type: 0x4, Count: 4, Object: [0, 16, 16, 0]
9 Offset: 0x260, Type: 0x5, Count: 8, Object: [688, 685, 83, 384, 959, 35, 321, 780]
10 Offset: 0x278, Type: 0x5, Count: 9, Object: [224, 727, 353, 794, 476, 777, 179, 58, 683]
11 Offset: 0x2a8, Type: 0xd, Count: 7, Object: [bytearray(b'integerValue'), bytearray(b'boolValue'), bytearray(b'floatValue'), bytearray(b'doubleValue'), bytearray(b'stringValue'), bytearray(b'dateValue'), bytearray(b'arrayReference')]
```

[그림 42] scan_all_objects.txt 파일 내용 중 일부

⁹ Soram kim, Giyoon Kim, Sumin Shin, Byungchul Youn, Jian Song, Insoo Lee, Jongsung Kim (2022). Methods for recovering deleted data from the Realm database: Case study on Minitalk and Xabber. Forensic Science International: Digital Investigation Volume 40, March 2022, 301353. 9p

```
realm_recover > # scan_unused_objects.txt
13 Offset: 0x670, Type: 0x11, Count: 37, Object: b'97990266-484D-4306-A1EB-0FA8B4025844'
14 Offset: 0x6a0, Type: 0x11, Count: 37, Object: b'RE869908-CF4F-412C-ACE1-E04B5BC672CD'
15 Offset: 0x6d0, Type: 0x11, Count: 37, Object: b'CAEBC848-ABCS-4E38-912F-92034EE6F3BA'
16 Offset: 0x1f0, Type: 0x5, Count: 4, Object: [1a0, 116, 285, 701]
17 Offset: 0xc030, Type: 0x11, Count: 37, Object: b'D2A01D48-8159-46AA-8588-077228CFCCA7'
18 Offset: 0xc060, Type: 0x11, Count: 37, Object: b'93A20922-1807-4C80-97D3-42DA1996666C'
19 Offset: 0xc090, Type: 0x11, Count: 37, Object: b'AA698ECD-D8AD-40D2-AD08-85098E6CC271'
20 Offset: 0xc0c0, Type: 0x11, Count: 36, Object: b'48E1F6A5-C826-4244-B1CF-088E451606F3'
21 Offset: 0xc0f0, Type: 0x11, Count: 36, Object: b'F04C8D0E-C2B5-4709-995C-B016CC8FCFFA'
22 Offset: 0xc10, Type: 0x11, Count: 36, Object: b'DC84D53F-85B5-4C1E-8398-ASC8273D63FD'
23 Offset: 0x1000, Type: 0x7, Count: 1000, Object: [9056741, 6702896, 6088682, 4199186, 9223372036854775807, 1969817, 3971947, 8239370, 8880480, 9954165, 9162994, 0, 0, 711936, 9223372036854775808, 1307258, 3699782, 6320186,
24 Offset: 0x1fd, Type: 0x5, Count: 5, Object: [181, 284, 407, 669, 700]
25 Offset: 0x1fe, Type: 0x5, Count: 5, Object: [241, 236, 307, 352, 762]
26 Offset: 0x1990, Type: 0x11, Count: 36, Object: b'2080D577-067E-488D-9FC9-B9C4F69ABA14'
27 Offset: 0x19c0, Type: 0x11, Count: 36, Object: b'E2A9CB1-AB32-48A0-AB12-FA3308CD34F4'
28 Offset: 0x19f0, Type: 0x11, Count: 36, Object: b'5D3F2D88-EABA-43A0-9AE2-08CD964DF197'
```

[그림 43] scan_unused_objects.txt 파일 내용 중 일부

따라서, RealmDB를 복구하는 도구를 범용적으로 개발하였기 때문에, 2번 질문에 해당하는 삭제 및 수정된 데이터를 찾아 복구하는 과정을 위 결과 파일들을 활용하여 [그림 24]부터 [그림 35]에서 설명한 것으로 대체하였습니다