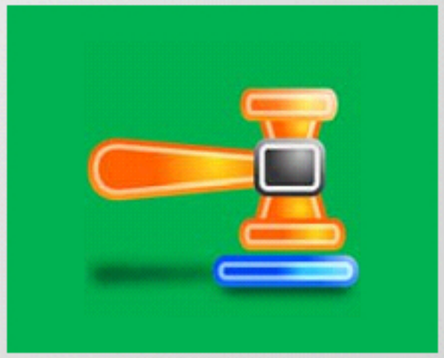
Database Forensics

Module 09

Designed by Cyber Crime Investigators. Presented by Professionals.



Module Objectives

CHFI Computer Hacking Forensic Investigator

After successfully completing this module, you will be able to:

1. Understand database forensics and its importance
2. Perform MSSQL forensics
3. Determine the database evidence repositories and collect the evidence files
4. Examine evidence files using SQL Server Management Studio and ApexSQL DBA
5. Perform MySQL forensics
6. Understand architecture of MySQL and determine the structure of data directory
7. List MySQL utilities for performing forensic analysis
8. Perform MySQL forensics on WordPress web application database

Database Forensics and Its Importance

C|HFI Computer Hacking Forensic Investigator

01 Database Forensics is the **examination of the databases and related metadata** in a forensically precise manner to make the findings presentable in the court of law

02 Forensics examination of the databases might allow a forensic investigator to:

I Examine the **MAC attributes** of tables that could verify the actions of the attacker

II Determine **transactions** occurred within a **database system** or application that indicate evidence of fraudulent activities

III Recover the **deleted rows**

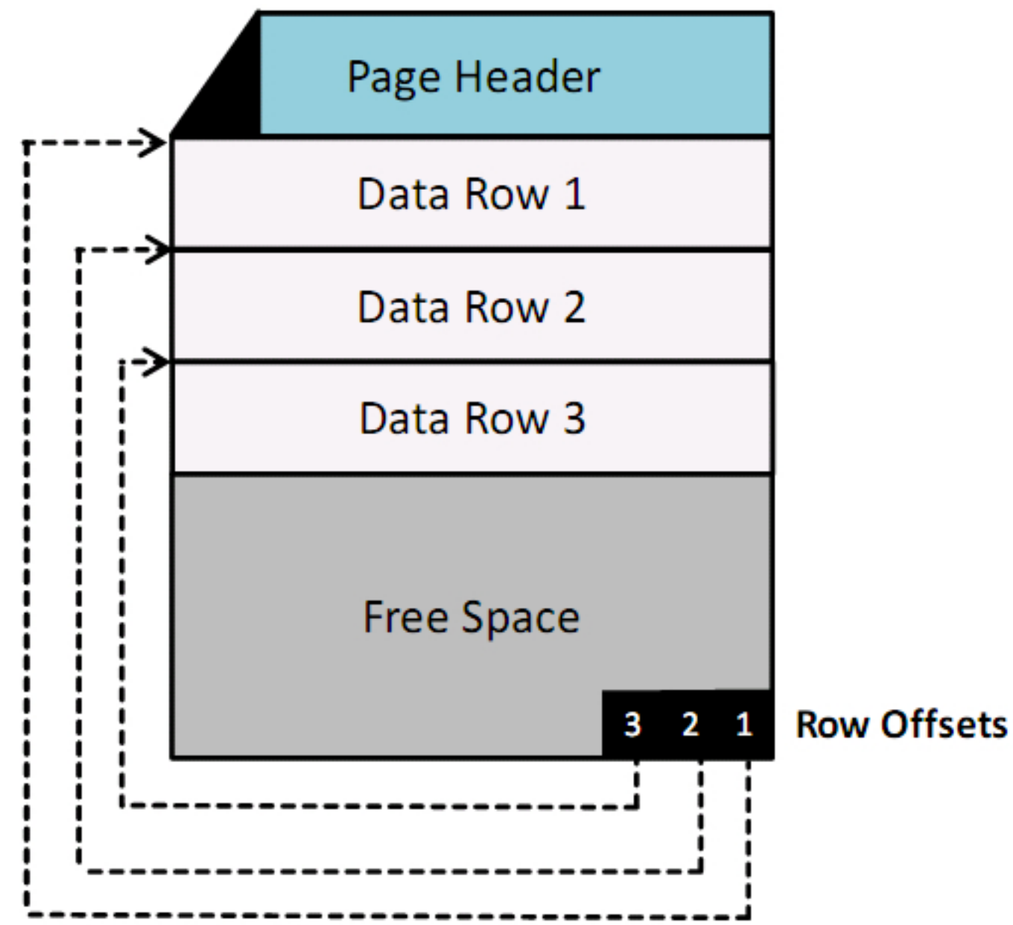
IV Retrace the **DDL** and **DML operations** performed by the attacker

MSSQL Forensics

Data Storage in SQL Server

- **SQL Server** stores data and logs in **Primary Data Files** (MDF), **Secondary Data Files** (NDF) and Transaction Log Data Files (LDF), respectively.
- **MDF** are the starting point of a database and stores user data and **database objects**
- **NDF** are optional and spread data across **multiple databases**
- **LDF** store log related information, which could be useful in recovering databases. These are divided into smaller parts called **virtual log files**
- These files are put together to form a database
- Each data file (excluding log files) contains **multiple data pages** (basic storage units with 8 Kb of storage)
- Data pages are divided into:
  - **Page Header** – Presents the page ID, page type, etc.
  - **Data Rows** – Store the actual data
  - **Offset Table** – Points to the location of actual data

Microsoft SQL Server Data Page



Database Evidence Repositories

SQL server data is stored natively within SQL Server, and externally within windows machine hosting the server. The main sources of evidence for an examiner would be:

SQL Server

- Volatile database
- Primary data file and Active Transaction Logs
- Database plan cache

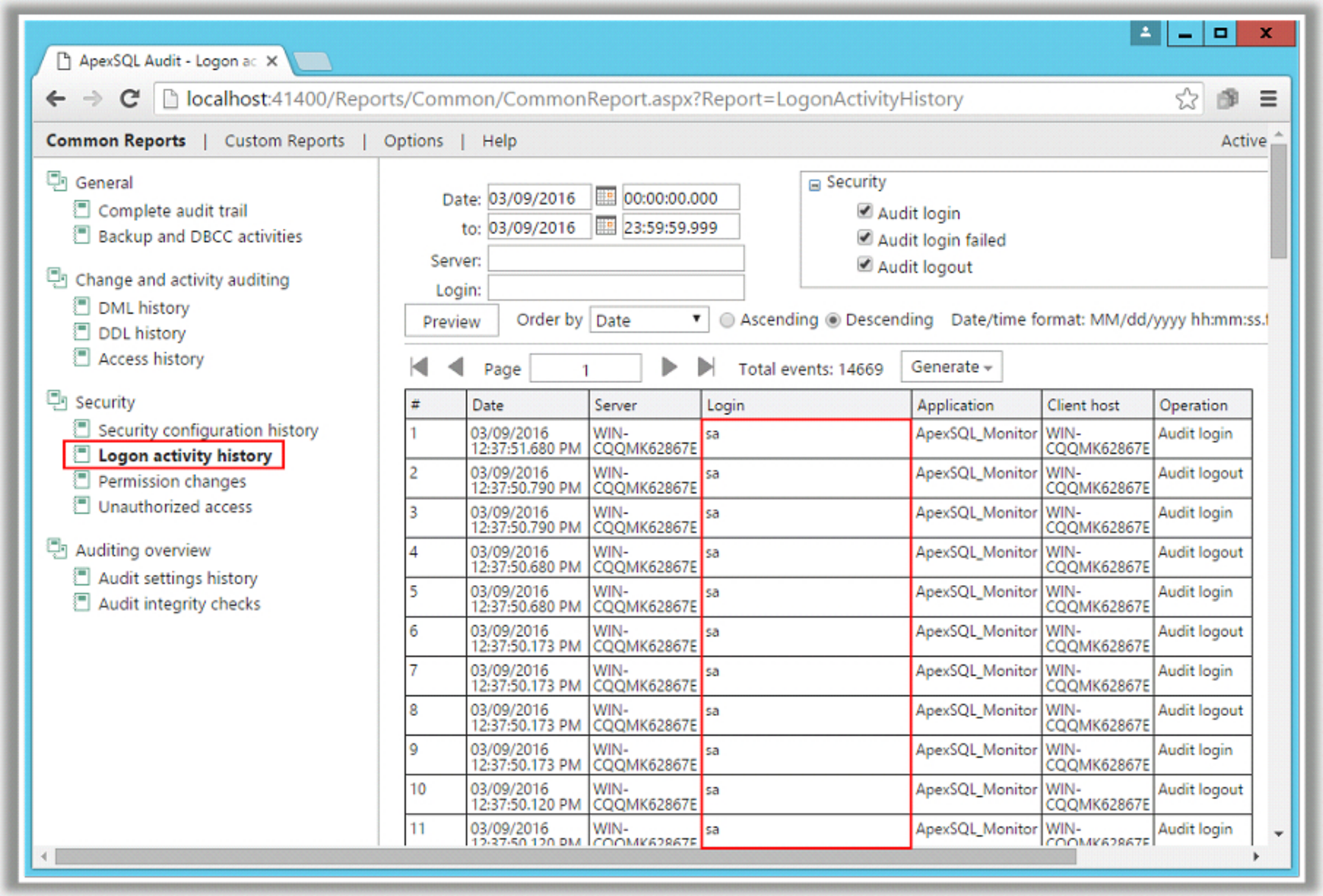
Windows Operating System (OS)

- Windows logs
- SQL Server Trace Files
- SQL Server Error

Note: System event logs, SQL Server trace files and SQL Server error logs constitute non volatile data.

Collecting Volatile Database Data

- Gather volatile database information such as **users' login sessions**, **user transactions**, etc.
- Use **ApexSQL DBA's ApexSQL audit** application to track the login history

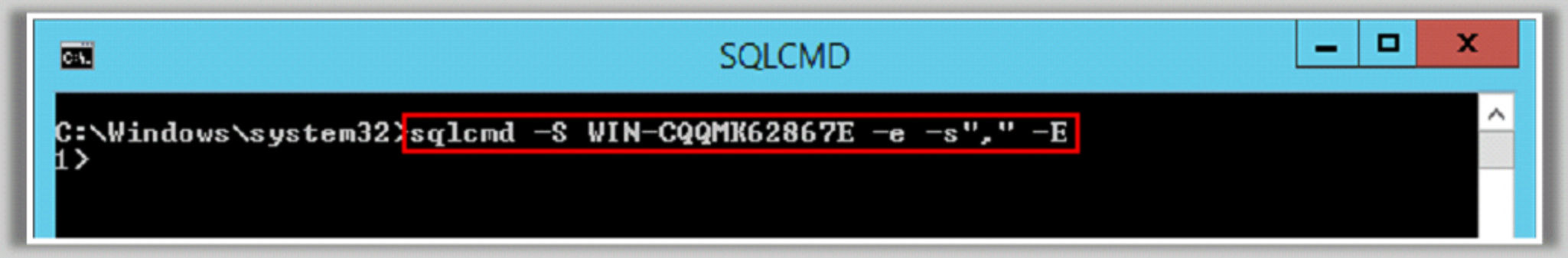


#	Date	Server	Login	Application	Client host	Operation
1	03/09/2016 12:37:51.680 PM	WIN-CQQMK62867E	sa	ApexSQL_Monitor	WIN-CQQMK62867E	Audit login
2	03/09/2016 12:37:50.790 PM	WIN-CQQMK62867E	sa	ApexSQL_Monitor	WIN-CQQMK62867E	Audit logout
3	03/09/2016 12:37:50.790 PM	WIN-CQQMK62867E	sa	ApexSQL_Monitor	WIN-CQQMK62867E	Audit login
4	03/09/2016 12:37:50.680 PM	WIN-CQQMK62867E	sa	ApexSQL_Monitor	WIN-CQQMK62867E	Audit logout
5	03/09/2016 12:37:50.680 PM	WIN-CQQMK62867E	sa	ApexSQL_Monitor	WIN-CQQMK62867E	Audit login
6	03/09/2016 12:37:50.173 PM	WIN-CQQMK62867E	sa	ApexSQL_Monitor	WIN-CQQMK62867E	Audit logout
7	03/09/2016 12:37:50.173 PM	WIN-CQQMK62867E	sa	ApexSQL_Monitor	WIN-CQQMK62867E	Audit login
8	03/09/2016 12:37:50.173 PM	WIN-CQQMK62867E	sa	ApexSQL_Monitor	WIN-CQQMK62867E	Audit logout
9	03/09/2016 12:37:50.173 PM	WIN-CQQMK62867E	sa	ApexSQL_Monitor	WIN-CQQMK62867E	Audit login
10	03/09/2016 12:37:50.120 PM	WIN-CQQMK62867E	sa	ApexSQL_Monitor	WIN-CQQMK62867E	Audit logout
11	03/09/2016 12:37:50.120 PM	WIN-CQQMK62867E	sa	ApexSQL_Monitor	WIN-CQQMK62867E	Audit login

Collecting Primary Data File and Active Transaction Logs Using SQLCMD

Load command line tool and establish logging

Connect to a server with the command **sqlcmd -S WIN-CQQMK62867E -e -s"," -E** (**WIN-CQQMK62867E** is the server used for this demonstration)

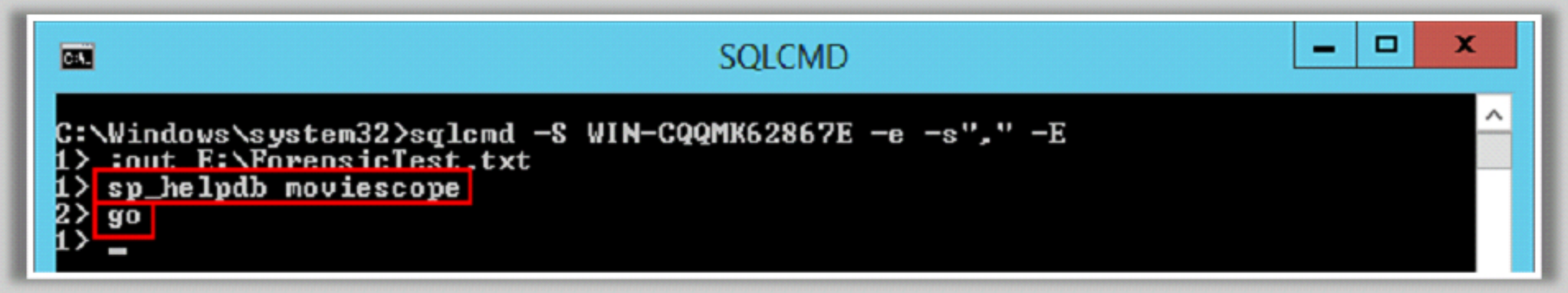


Issue **:out E:\ForensicTest.txt** to create a text file named "**ForensicTest**" and log the output of the gathered data to **E:**

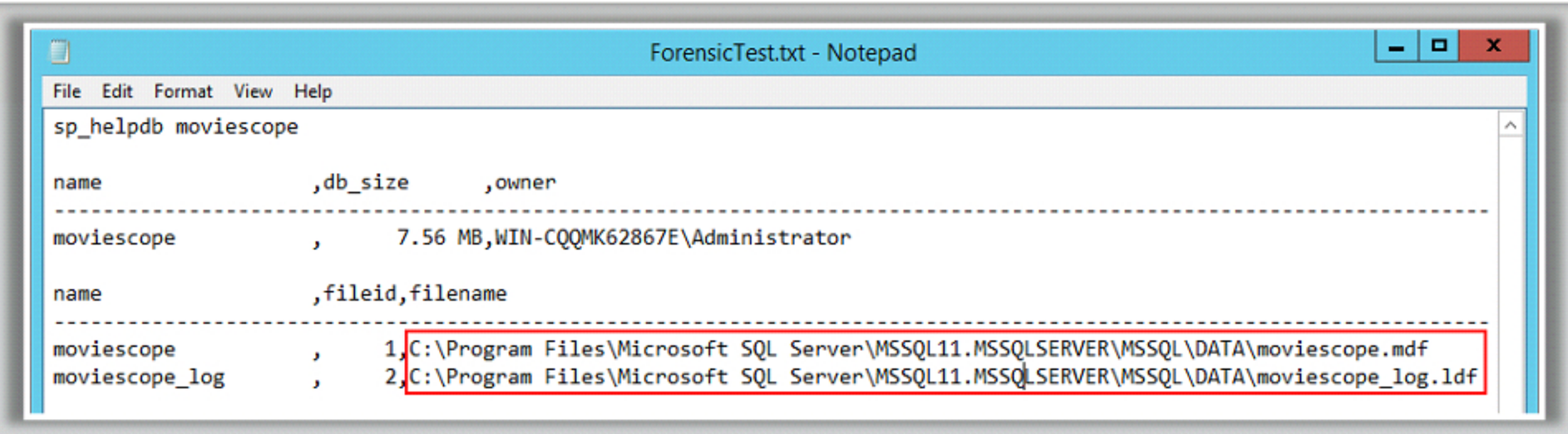
Collecting Primary Data File and Active Transaction Logs Using SQLCMD (Cont'd)

Collect the active transaction log

Issue the commands **sp_helpdb moviescope** and go to determine the locations of the transaction log files associated with moviescope database



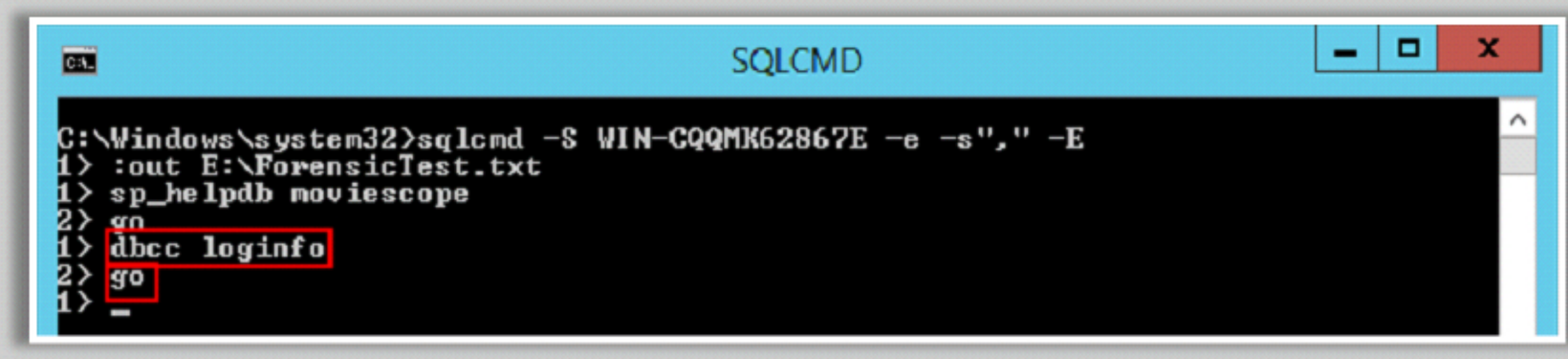
The result will be recorded in **E:** drive in the respective file (**ForensicTest.txt**) as shown in the following screenshot:



Collecting Primary Data File and Active Transaction Logs Using SQLCMD (Cont'd)

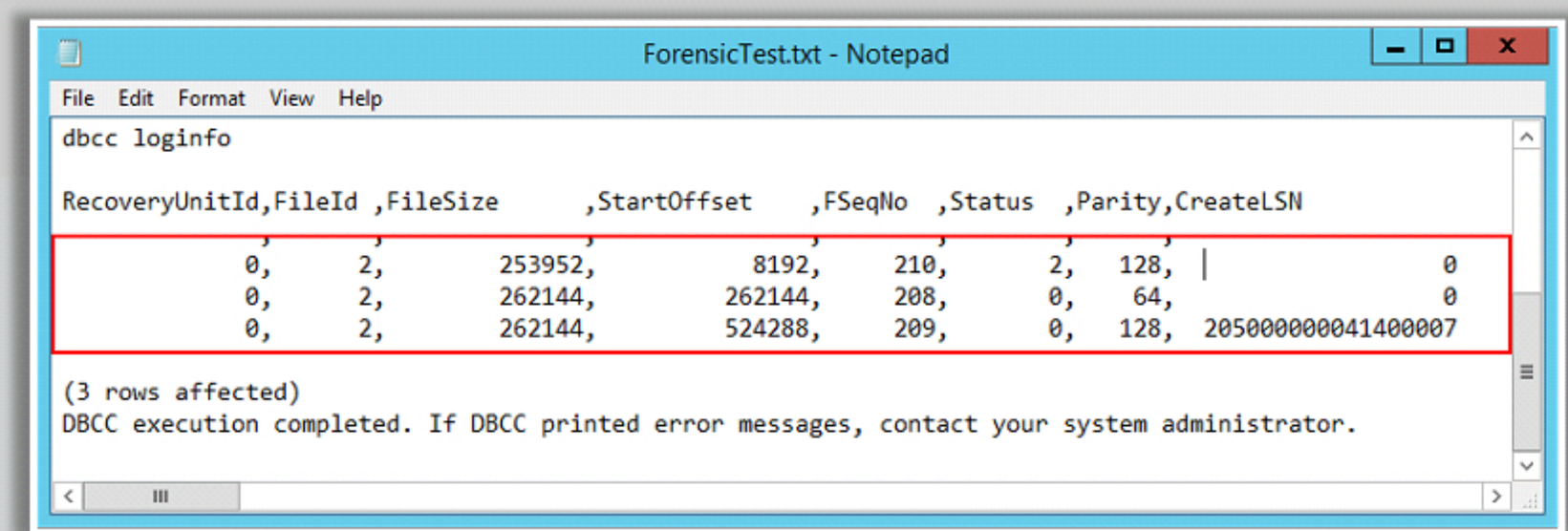
Collect the active transaction log (Cont'd)

Issue the commands **dbcc loginfo** and **go** to gather the VLF allocations for the **moviescope** database



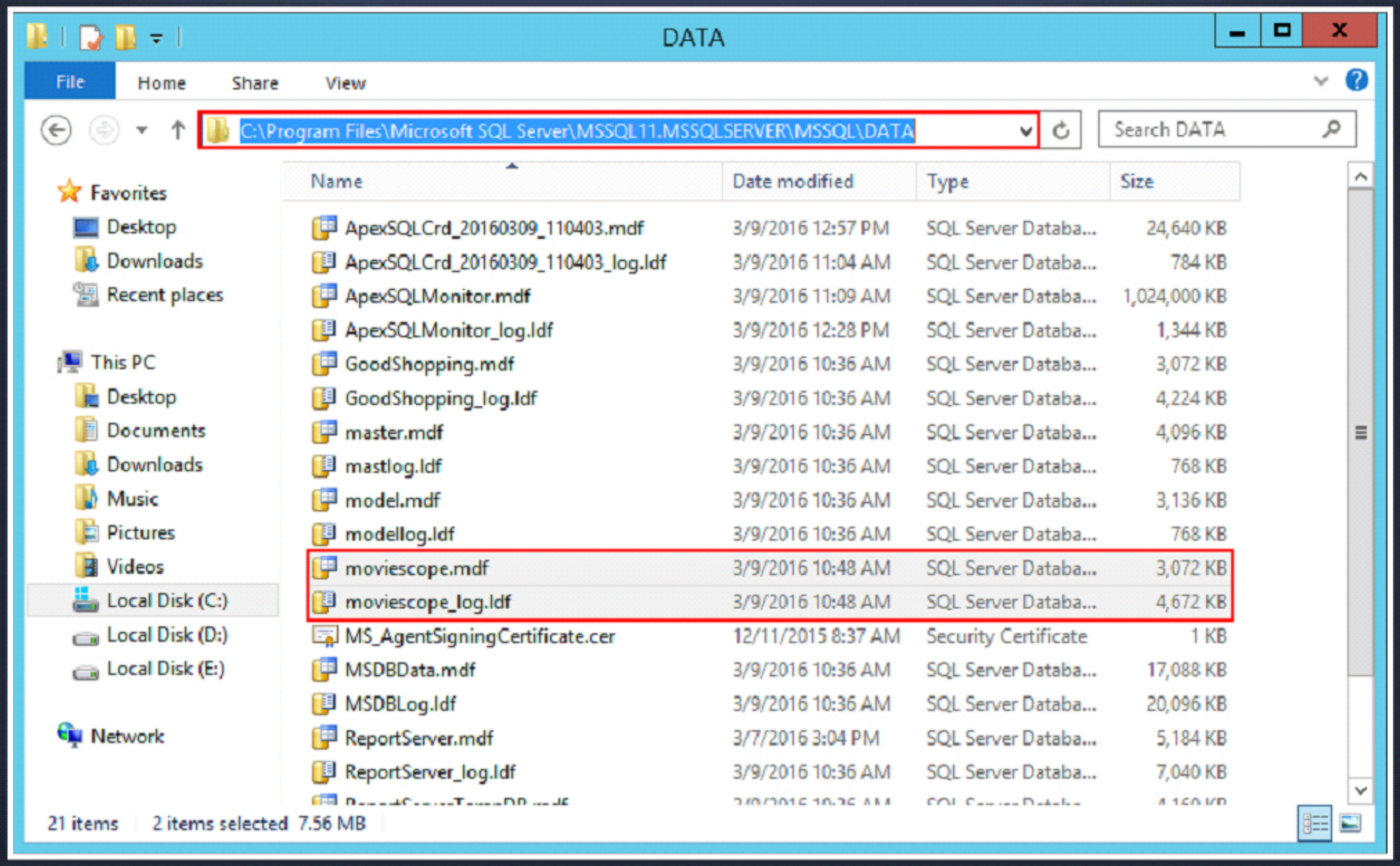
The result will be recorded in the respective file as shown in the following screenshot:

The status field displays the status of the file, where "2" represents an active file, while "0" represents a recoverable or unused file



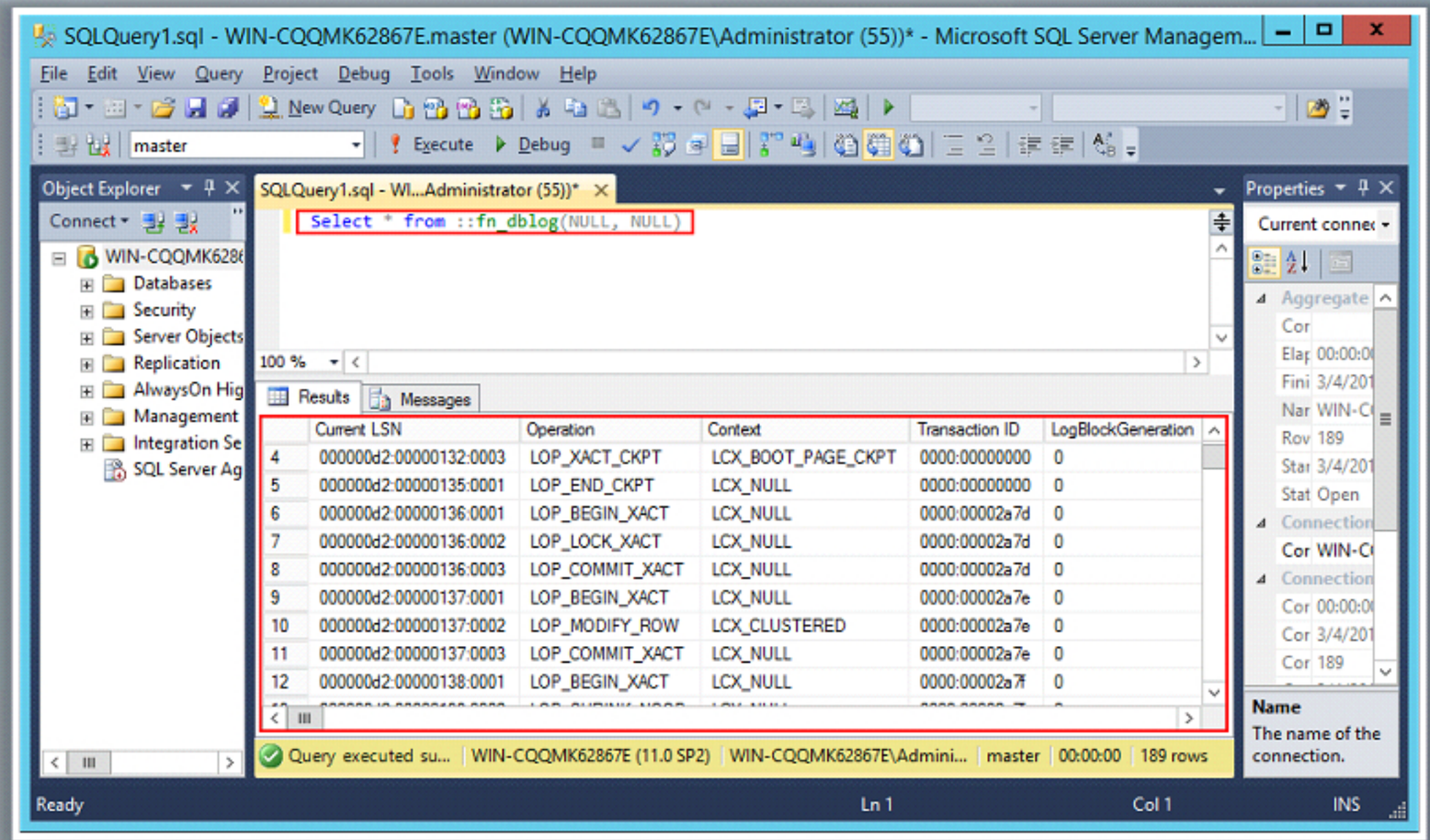
Collecting Primary Data File and Transaction Logs

- Collect the database files (.mdf) and log files (.ldf) from **C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\DATA**
- These files contain complete data (in **.mdf** files) and logs (in **.ldf** files) pertaining to the databases



Collecting Active Transaction Logs Using SQL Server Management Studio

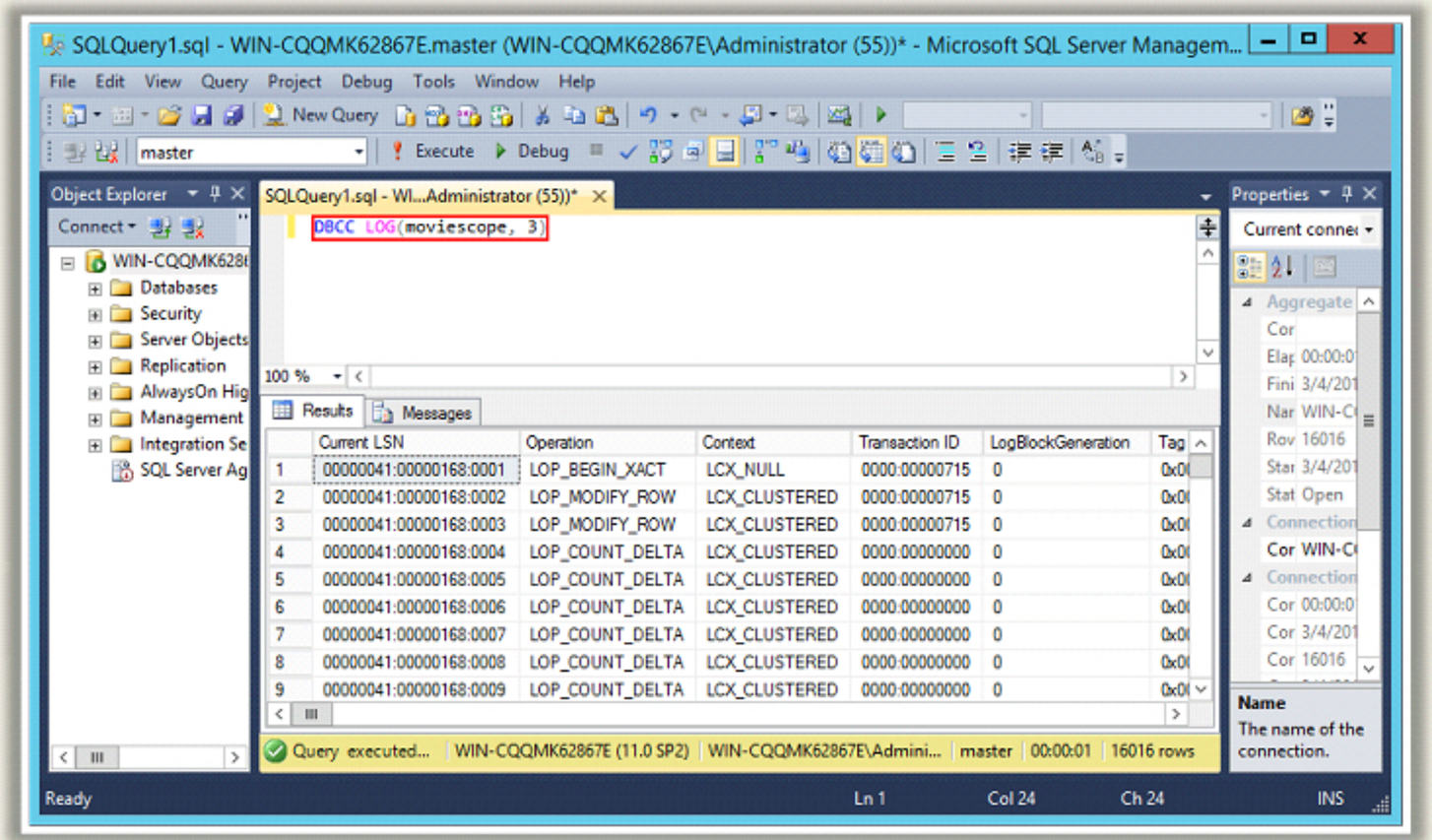
- The fn_dblog() function allows to retrieve the active portion of the transaction log file
- **fn_dblog ()** function filter transactions by:
  - Target database object
  - Specific columns
  - SPID and/or date/time range
- Issuing the query **Select * from ::fn_dblog(NULL, NULL)** displays the active portion of the transaction log file as shown in the screenshot
- Assigning NULL values imply that the start and end points for log sequence numbers (LSNs) are not specified



Collecting Active Transaction Logs Using SQL Server Management Studio (Cont'd)

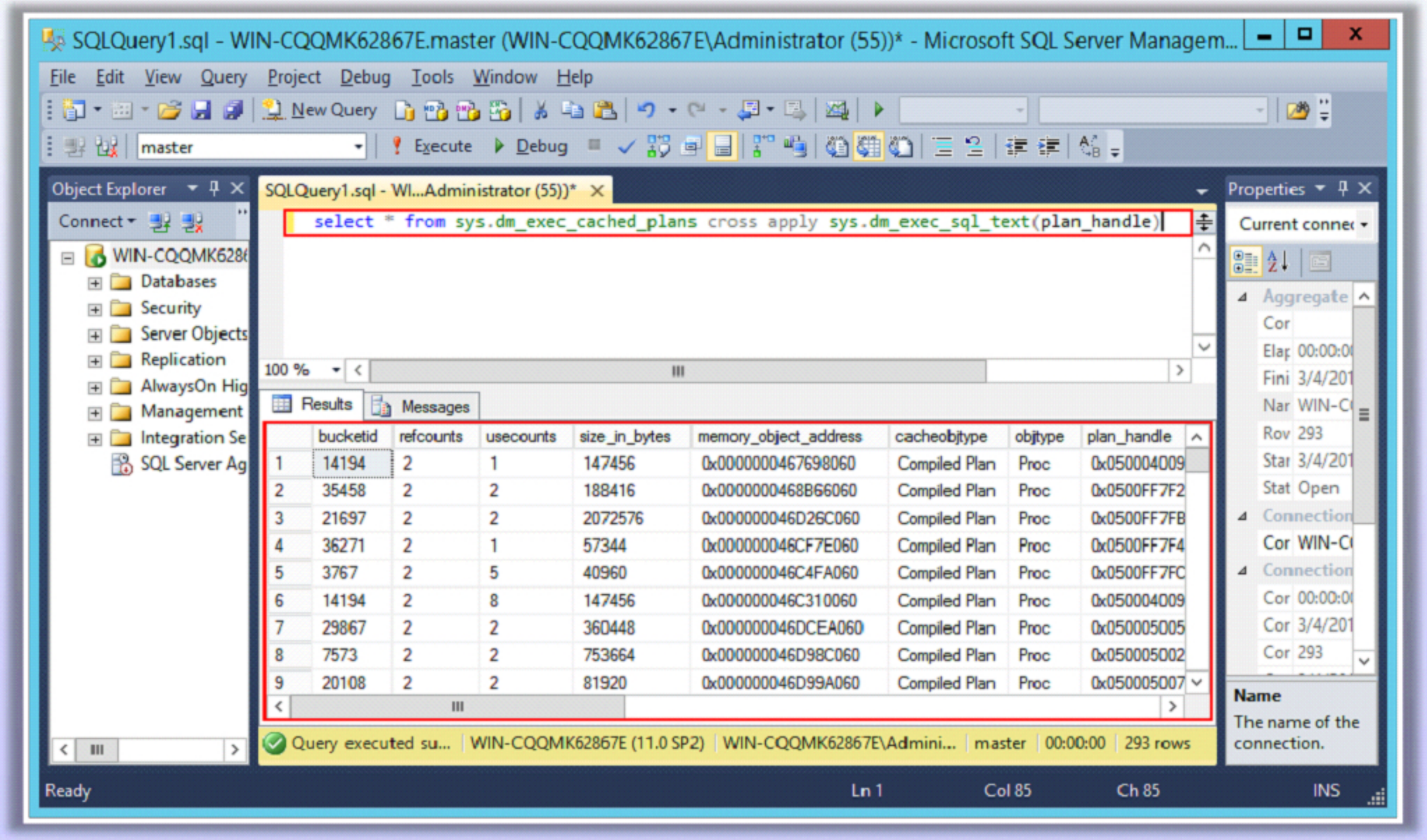
DBCC LOG

- The DBCC LOG command allows to retrieve the active transaction log files for the specified database.
- **Syntax**: `DBCC LOG(<databasename >, <output >)`
- The output parameter specifies the level of information a forensic examiner wants to retrieve
  - 0= minimal information of each operation such as the Current LSN, Operation, Transaction ID, etc.
  - 1 = slightly more info than 0, such as Flag Bits, Previous LSN, etc.
  - 2 = detailed information, including (AllocUnitId, page id, slot id, etc.)
  - 3 = full information about each operation
  - 4 = full information on each operation along with the hex dump of current transaction row
- Issue the query `DBCC LOG(moviescope, 3)` to view the transaction log file for moviescope database, with the detailed information for each operation.



Collecting Database Plan Cache

Issue the syntax **select * from sys.dm_exec_cached_plans cross apply sys.dm_exec_sql_text (plan_handle)** to retrieve SQL text of all cached entries

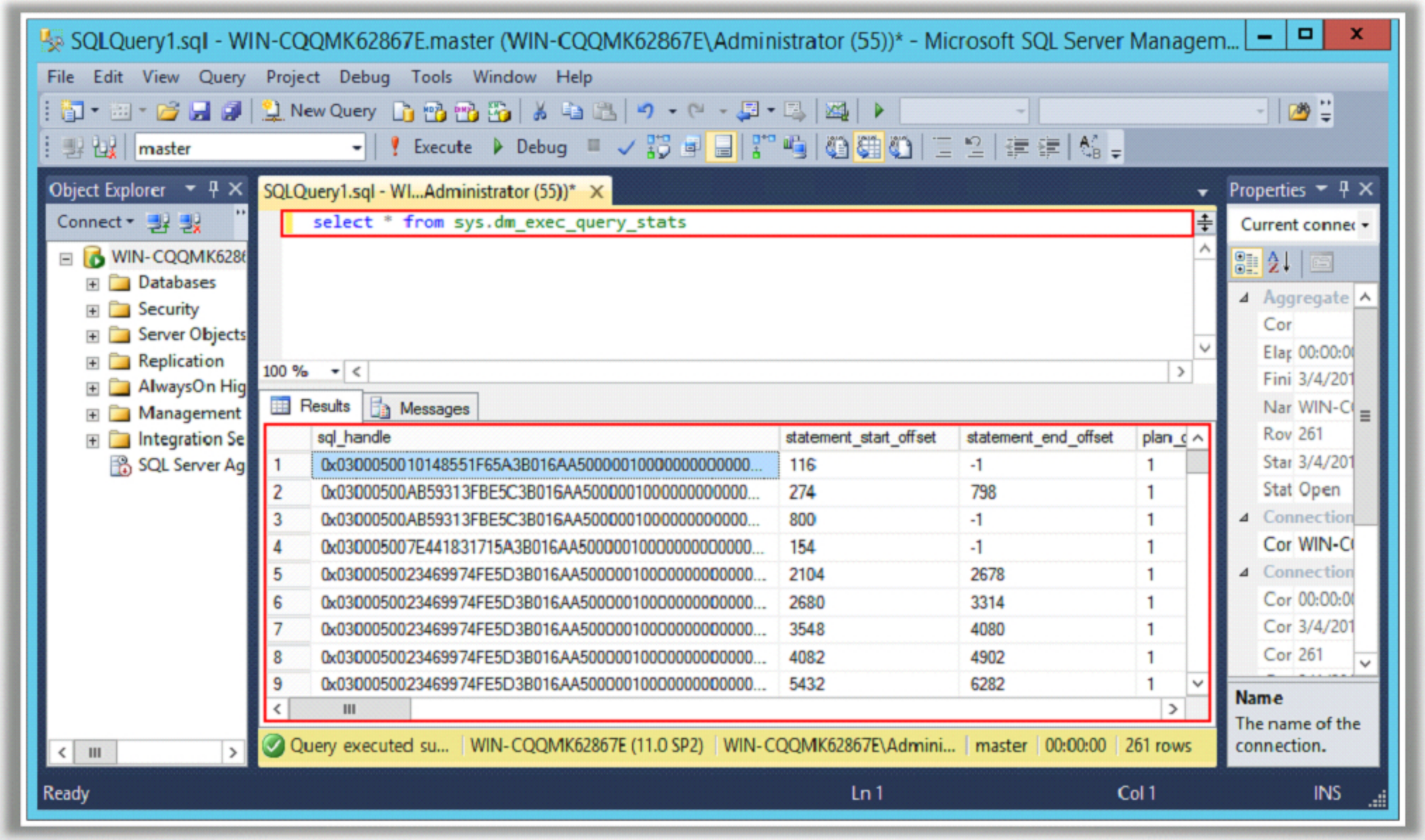


The `plan_handle` argument retrieves the compiled query plans from the SQLCP or the OBJCP cache stores

Collecting Database Plan Cache (Cont'd)

Collect additional plan cache specifics

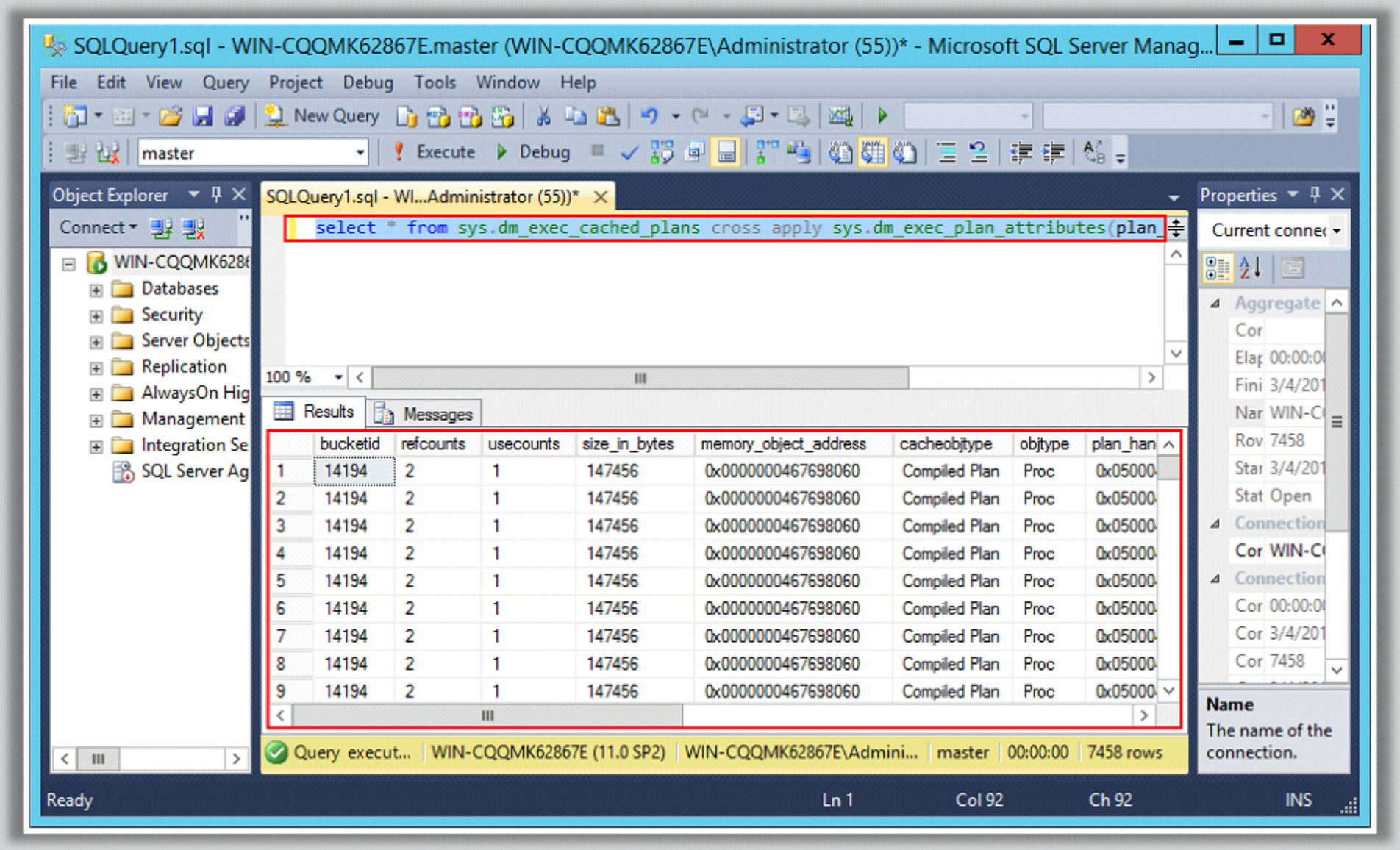
Issue the syntax **select * from sys.dm_exec_query_stats** to view the aggregate performance statistics for cached query plans. It displays only one row per query statement



Collecting Database Plan Cache (Cont'd)

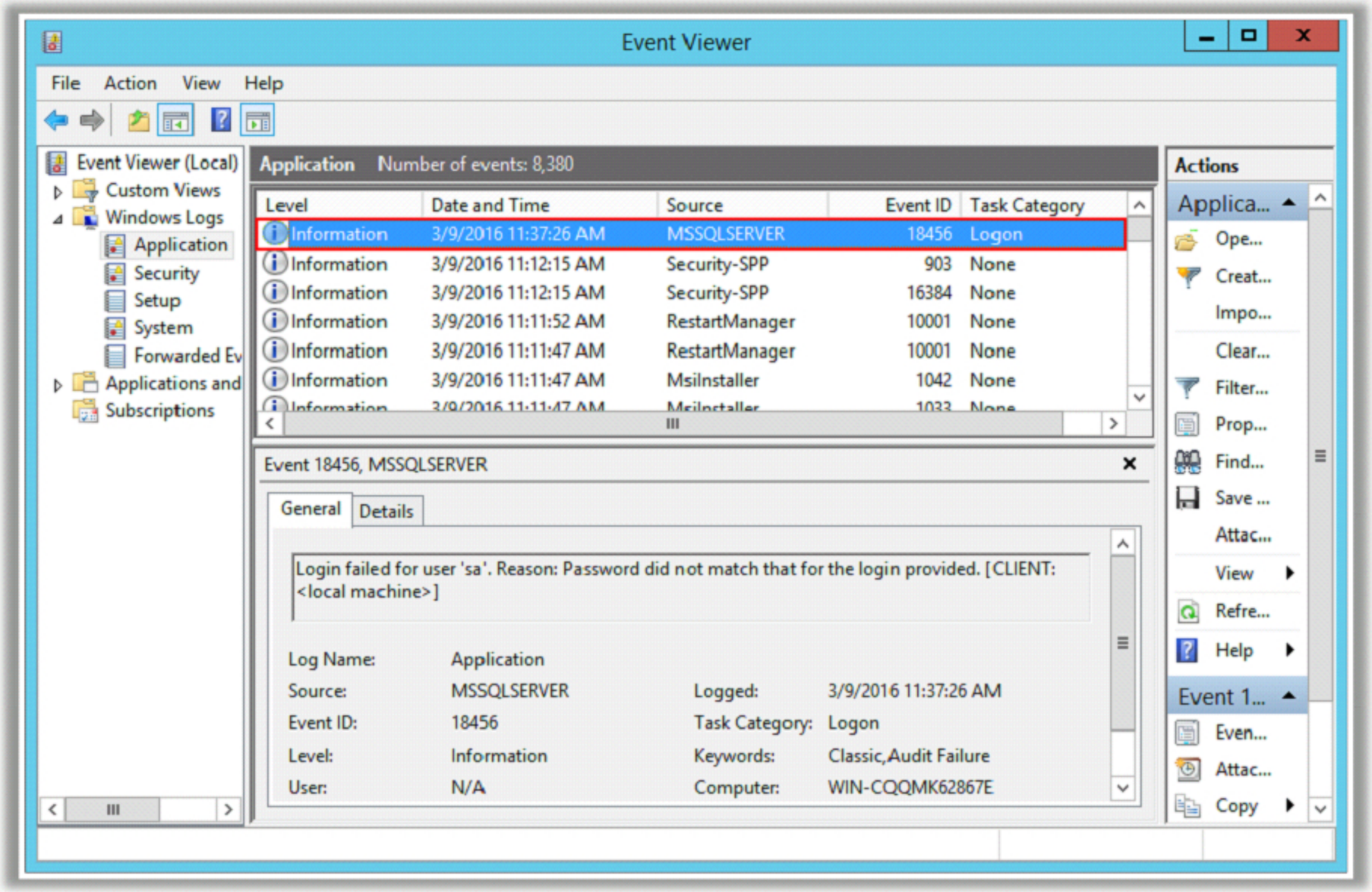
Collect additional plan cache specifics

Issue the syntax **select * from sys.dm_exec_cached_plans** cross apply **sys.dm_exec_plan_attributes(plan_handle)** to view one row per plan attribute for the plan specified by the plan handle



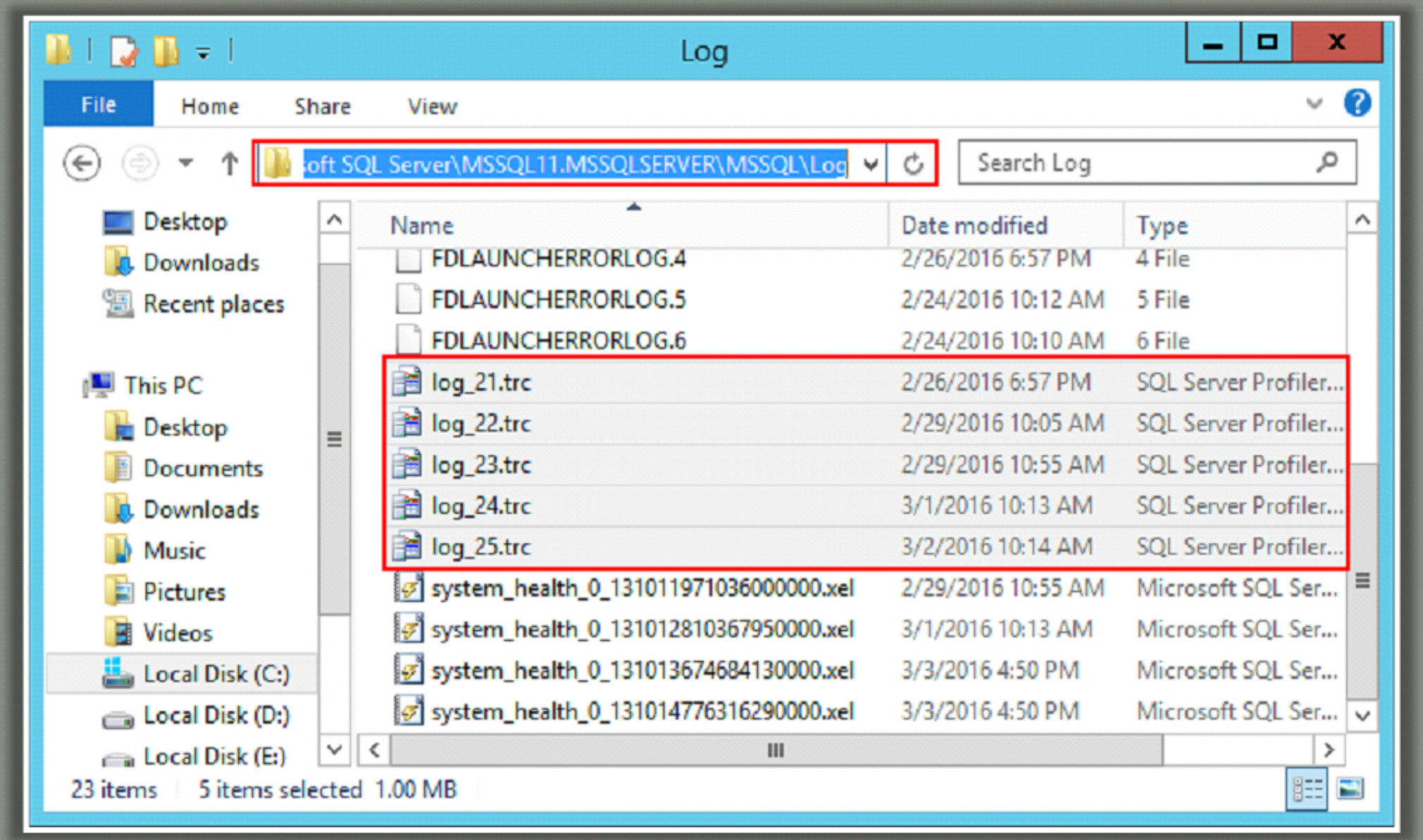
Collecting Windows Logs

- Windows Logs store the logon events performed on the **SQL Server**. Launch Event Viewer, expand **Windows Logs** node and view various **Windows event logs**



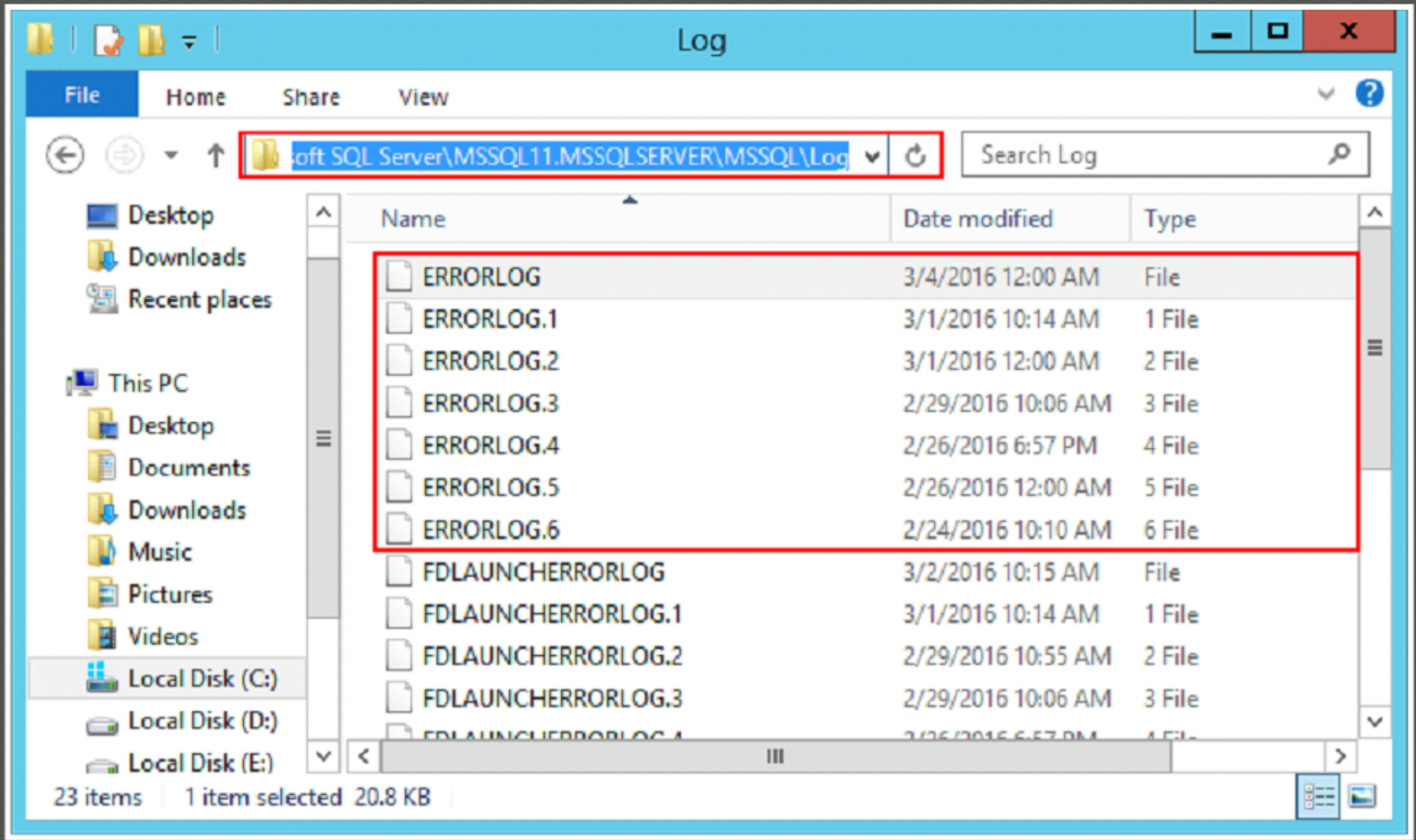
Collecting SQL Server Trace Files

- To collect the trace files (**.trc**) navigate to **C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\LOG**
- The trace files contain the events occurred on a SQL server and the host databases



Collecting SQL Server Error Logs

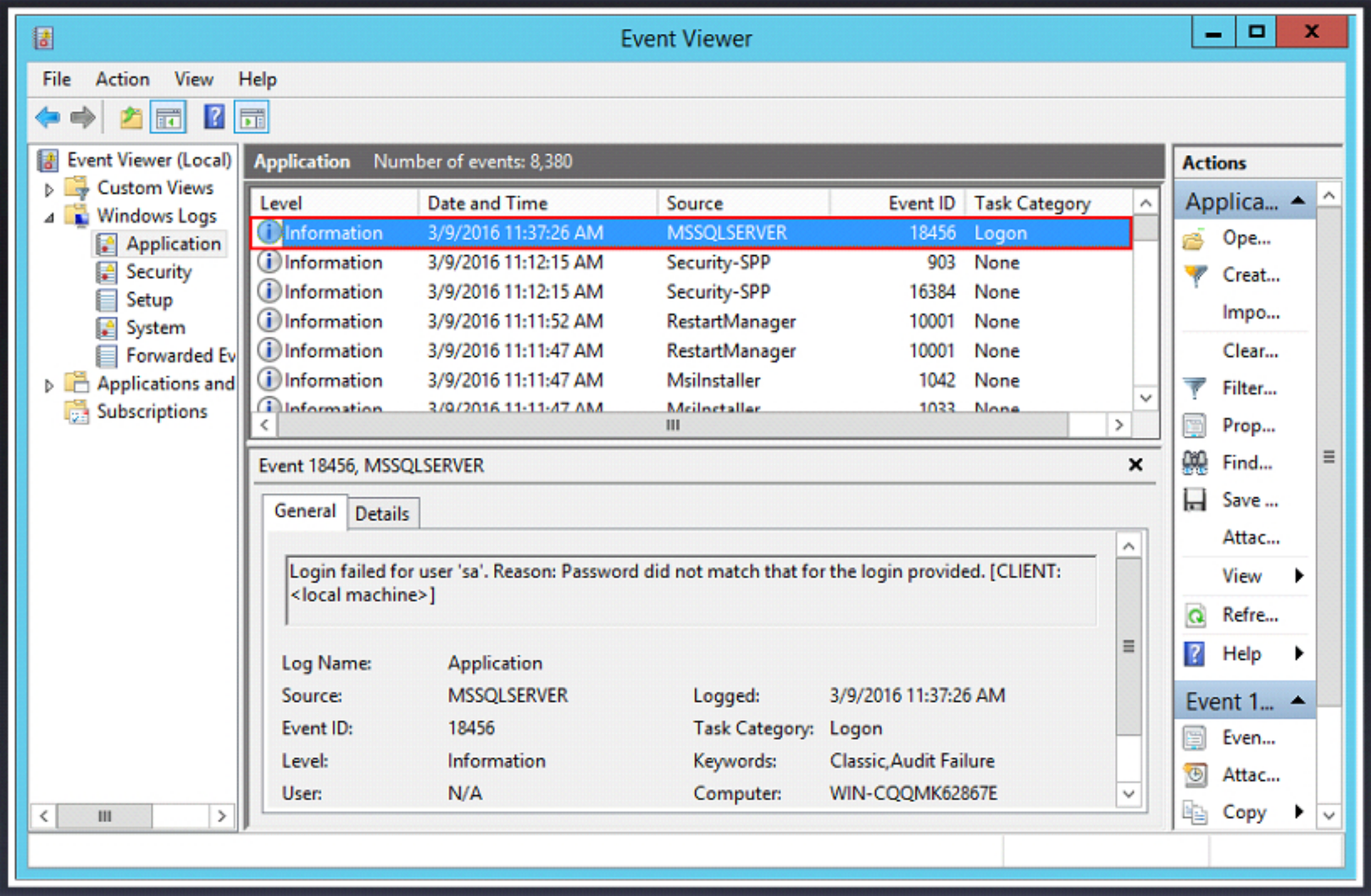
- To collect the SQL Server error logs navigate to **C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\LOG**
- The SQL Server error logs contain user defined events and specific system events



Database Forensics Using SQL Server Management Studio

Step 1: Examine Windows Logs

Examine the Windows Logs to obtain information related to SQL Server authentication, startup and shutdown instances, and the IP addresses of client connections



It is observed that an event associated with the server login and pertaining to MSSQL Server is recorded. Now, the error log need to be examined to find out any successful login event.

Database Forensics Using SQL Server Management Studio (Cont'd)

CHFI Computer Hacking Forensic Investigator

Step 2: Examine Error Logs

- Navigate to **C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\LOG** and open **ERRORLOG** file with Notepad
- Examine the log file to see the record of user defined events (such as user logins)

C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\Log\ERRORLOG - Notepad++ [Administrator]

File Edit Search View Encoding Language Settings Macro Run Plugins Window ?

ERRORLOG

```
13 2016-03-09 10:41:35.74 Server      Registry startup parameters:
14      -d C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\master.mdf
15      -e C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\Log\ERRORLOG
16      -l C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\mastlog.ldf
17 2016-03-09 10:41:35.74 Server      Command Line Startup Parameters:
18      -s "MSSQLSERVER"
19 2016-03-09 10:41:35.96 Server      SQL Server detected 1 sockets with 4 cores per socket and 4 logical proce
20 2016-03-09 10:41:35.96 Server      SQL Server is starting at normal priority base (=7). This is an informati
21 2016-03-09 10:41:35.96 Server      Detected 16280 MB of RAM. This is an informational message; no user actio
22 2016-03-09 10:41:35.96 Server      Using conventional memory in the memory manager.
23 2016-03-09 10:41:36.03 Server      This instance of SQL Server last reported using a process ID of 1716 at 3
24 2016-03-09 10:41:36.03 Server      Node configuration: node 0: CPU mask: 0x000000000000000f:0 Active CPU mas
25 2016-03-09 10:41:36.04 Server      Using dynamic lock allocation.  Initial allocation of 2500 Lock blocks an
26 2016-03-09 10:41:36.05 Server      Software Usage Metrics is disabled.
27 2016-03-09 10:41:36.05 spid4s      Starting up database 'master'.
28 2016-03-09 10:41:36.14 Server      CLR version v4.0.30319 loaded.
29 2016-03-09 10:41:36.19 Server      Common language runtime (CLR) functionality initialized using CLR version
30 2016-03-09 10:41:36.20 spid4s      Resource governor reconfiguration succeeded.
31 2016-03-09 10:41:36.20 spid4s      SQL Server Audit is starting the audits. This is an informational message
32 2016-03-09 10:41:36.20 spid4s      SQL Server Audit has started the audits. This is an informational message
33 2016-03-09 10:41:36.22 spid4s      SQL Trace ID 1 was started by login "sa".
```

Normal text file | length : 15516 lines : 127 | Ln : 33 Col : 75 Sel : 2 | 0 | Dos\Windows | UCS-2 LE BOM | INS

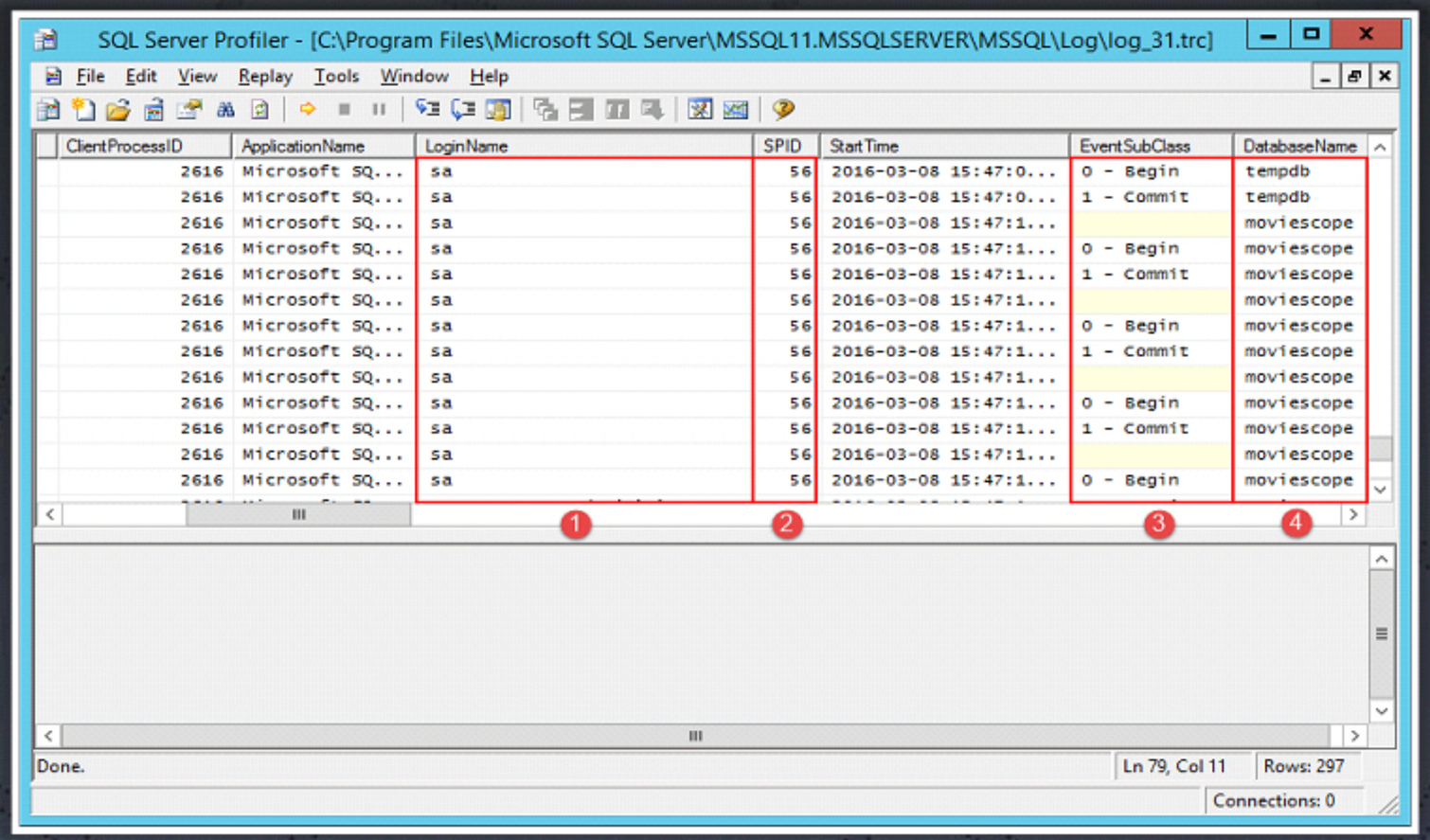
Here, it is evident that there is a successful login instance recorded on the name of a user **sa**. Now, the trace file can be viewed to examine the SQL Server based events associated with this user.

Database Forensics Using SQL Server Management Studio (Cont'd)

Step 3: Examine Trace Files

Navigate to **C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\LOG** and double-click **log_n.trc** file (where n is the last number in the sequence). The trace file opens in a SQL Server Profiler

Examine the file to identify any suspicious activity



SQL Server Profiler - [C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\Log\log_31.trc]

ClientProcessID	ApplicationName	LoginName	SPID	StartTime	EventSubClass	DatabaseName
2616	Microsoft SQ...	sa	56	2016-03-08 15:47:0...	0 - Begin	tempdb
2616	Microsoft SQ...	sa	56	2016-03-08 15:47:0...	1 - Commit	tempdb
2616	Microsoft SQ...	sa	56	2016-03-08 15:47:1...		moviescope
2616	Microsoft SQ...	sa	56	2016-03-08 15:47:1...	0 - Begin	moviescope
2616	Microsoft SQ...	sa	56	2016-03-08 15:47:1...	1 - Commit	moviescope
2616	Microsoft SQ...	sa	56	2016-03-08 15:47:1...		moviescope
2616	Microsoft SQ...	sa	56	2016-03-08 15:47:1...	0 - Begin	moviescope
2616	Microsoft SQ...	sa	56	2016-03-08 15:47:1...	1 - Commit	moviescope
2616	Microsoft SQ...	sa	56	2016-03-08 15:47:1...		moviescope
2616	Microsoft SQ...	sa	56	2016-03-08 15:47:1...	0 - Begin	moviescope
2616	Microsoft SQ...	sa	56	2016-03-08 15:47:1...	1 - Commit	moviescope
2616	Microsoft SQ...	sa	56	2016-03-08 15:47:1...		moviescope
2616	Microsoft SQ...	sa	56	2016-03-08 15:47:1...	0 - Begin	moviescope

By examining the file, some user based events observed on moviescope database. Make a note of the SPID and the start time of the instance.

Database Forensics Using SQL Server Management Studio (Cont'd)

Step 4: Examine Active Transaction Logs

Launch SQL Server Management Studio and connect to the SQL Server. Execute the command **dbcc log(moviescope, 3)** in the query window to view the transaction log file for moviescope database, with detailed information for each operation. Here, an event can be observed (SPID: **56** and Transaction ID: **0000:000007c9**) with a modified row.

1 - Indicates the beginning of a transaction

2 - Indicates the type of the transaction performed

3 - Indicates the end of a transaction

4 - SPID: Indicate the current user process ID

5 - Unique transaction identifier

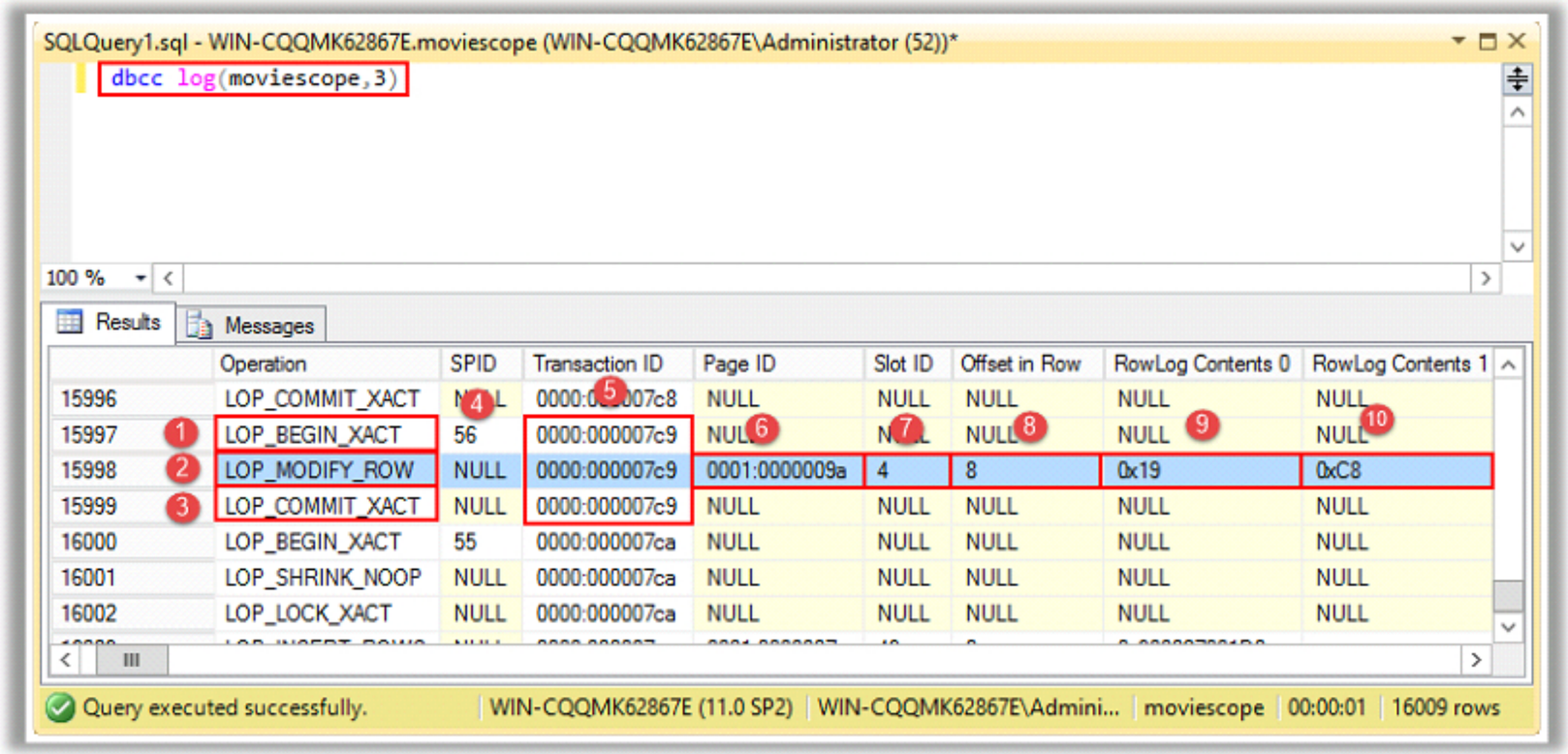
6 - Data Page Identifier for rows containing the updated record

7 - On data page row location of record

8 - In row data offset of modification

9 - Value of the row before modification

10 - Value of the row after modification



	Operation	SPID	Transaction ID	Page ID	Slot ID	Offset in Row	RowLog Contents 0	RowLog Contents 1
15996	LOP_COMMIT_XACT	NULL	0000:000007c8	NULL	NULL	NULL	NULL	NULL
15997	LOP_BEGIN_XACT	56	0000:000007c9	NULL	NULL	NULL	NULL	NULL
15998	LOP_MODIFY_ROW	NULL	0000:000007c9	0001:0000009a	4	8	0x19	0xC8
15999	LOP_COMMIT_XACT	NULL	0000:000007c9	NULL	NULL	NULL	NULL	NULL
16000	LOP_BEGIN_XACT	55	0000:000007ca	NULL	NULL	NULL	NULL	NULL
16001	LOP_SHRINK_NOOP	NULL	0000:000007ca	NULL	NULL	NULL	NULL	NULL
16002	LOP_LOCK_XACT	NULL	0000:000007ca	NULL	NULL	NULL	NULL	NULL

Convert hexadecimal value of the **page ID** to decimal, and locate the page containing the updated record. Here, the calculated decimal value of the page ID is **154. (0000009a)16 = (154)10**

Database Forensics Using SQL Server Management Studio (Cont'd)

Step 5: Examine Data Page

- Now, we inspect the modified data pages to find the object ID where the data has been modified
- Execute the commands: **dbcc traceon(3604)dbcc page(moviescope,1,154,1)** to view the 154th data page on the query window
- The PAGE HEADER contains information regarding the data page such as the type of page, partition ID, object ID, etc. Note down the Object ID

SQLQuery2.sql - WIN-CQQMK62867E.moviescope (WIN-CQQMK62867E\Administrator (55))*

```
dbcc traceon(3604)
dbcc page(moviescope, 1, 154, 1)
```

100 %

Messages

```
PAGE HEADER:


Page @0x000000046CCFA000

m_pageId = (1:154)                  m_headerVersion = 1                 m_type = 1
m_typeFlagBits = 0x4                m_level = 0                         m_flagBits = 0x8000
m_objId (AllocUnitId.idObj) = 37    m_indexId (AllocUnitId.idInd) = 256
Metadata: AllocUnitId = 72057594040352768
Metadata: PartitionId = 72057594039042048                               Metadata: IndexId = 0
Metadata: ObjectId = 21575115       m_prevPage = (0:0)                  m_nextPage = (0:0)
pminlen = 12                        m_slotCnt = 5                       m_freeCnt = 7284
m_freeData = 898                    m_reservedCnt = 0                   m_lsn = (76:312:2)
```

100 %

Query executed successfully. | WIN-CQQMK62867E (11.0 SP2) | WIN-CQQMK62867E\Admini... | moviescope | 00:00:00 | 0 rows

Database Forensics Using SQL Server Management Studio (Cont'd)

Step 6: View the Object

- Next, we use the object ID to find the name of the object/table in moviescope database, whose data was modified
- Execute the command **Select * from sysobjects where id = 21575115**
- The object **User_Profile** has been modified. Next, we use the same object ID to gather the object schema

SQLQuery3.sql - WIN-CQQMK62867E.moviescope (WIN-CQQMK62867E\Administrator (56))*

```
Select * from sysobjects where id = 21575115
```

100 %

Results | Messages

	name	id	xtype	uid	info	status	base_schema_ver	replinfo	parent_obj	crdate	ftcatid	schema_v
1	User_Profile	21575115	U	1	0	0	0	0	0	2014-04-15 12:06:37.823	0	0

Query executed successfully. | WIN-CQQMK62867E (11.0 SP2) | WIN-CQQMK62867E\Admini... | moviescope | 00:00:00 | 1 rows

Database Forensics Using SQL Server Management Studio (Cont'd)

Step 7: Gather the Object Schema

- Next, using the **object ID**, the object schema (table) associated with the **User_Profile object** is collected
- Execute the command: **SELECT sc.colorder , sc.name, st.name as 'datatype', sc.length FROM syscolumns sc, systypes st WHERE sc.xusertype = st.xusertype and sc.id = 21575115 ORDER BY colorder**
- By issuing the above command, the object schema is obtained. One of the entries in the table is subjected to modification

SQLQuery4.sql - WIN-CQQMK62867E.moviescope (WIN-CQQMK62867E\Administrator (51))*

```
SELECT sc.colorder , sc.name, st.name as 'datatype', sc.length FROM syscolumns  sc, systypes st  WHERE
```

100 %

Results | Messages

	colorder	name	datatype	length
1	1	Uid	int	4
2	2	username	nvarchar	100
3	3	firstname	nvarchar	100
4	4	lastname	nvarchar	100
5	5	email	nvarchar	100
6	6	gender	nvarchar	100
7	7	dateofbirth	nvarchar	100
8	8	age	int	4
9	9	address	nvarchar	-1
10	10	contactnumber	nvarchar	100

Query executed successfully. | WIN-CQQMK62867E (11.0 SP2) | WIN-CQQMK62867E\Admini... | moviescope | 00:00:00 | 10 rows

Database Forensics Using SQL Server Management Studio (Cont'd)

Step 8: View the Modified Record

- As we have seen in step 4, the page ID is 154 and slot ID is 4. Therefore, issue the commands
  dbcc trace (3604)
  dbcc page(moviescope,1,154,1)
- To view the data page **154**. Scroll down to **Slot no. 4 (data row no. 4)**

SQLQuery7.sql - WIN-CQQMK62867E.moviescope (WIN-CQQMK62867E\Administrator (51))*

```
dbcc page(moviescope, 1, 154, 1)
```

100 %

Messages

```
Slot 4, Offset 0x2e8, Length 154, DumpStyle BYTE

Record Type = PRIMARY_RECORD        Record Attributes =  NULL_BITMAP VARIABLE_COLUMNS
Record Size = 154
Memory Dump @0x000000001249A2E8

000000000000000:   30000c00 05000000 c8000000 0a000000 08002800  0.......È.........(.
000000000000014:   2e003600 4c005400 68007e00 9a006c00 65006500  ..6.L.T.h.~..l.e.e.
000000000000028:   6c006500 65006200 72006500 74006c00 65006500  l.e.e.b.r.e.t.l.e.e.
00000000000003C:   40006100 62006300 2e006300 6f006d00 6d006100  @.a.b.c...c.o.m.m.a.
000000000000050:   6c006500 30003900 2d003000 38002d00 31003900  l.e.0.9.-.0.8.-.1.9.
000000000000064:   38003800 41006c00 62007500 71007500 61007200  8.8.A.l.b.u.q.u.a.r.
000000000000078:   71007500 65003100 2d003200 30003200 2d003500  q.u.e.1.-.2.0.2.-.5.
00000000000008C:   30003600 2d003300 36003900 3100               0.6.-.3.6.9.1.
```

This is the data row in which transaction has occurred

100 %

Query executed successfully. | WIN-CQQMK62867E (11.0 SP2) | WIN-CQQMK62867E\Admini... | moviescope | 00:00:00 | 0 rows

Database Forensics Using SQL Server Management Studio (Cont'd)

Step 9: Identify the Data Type

Using **slot ID 4** and **row offset 8**, which were obtained previously from the transaction log, the specific point within the data row was identified in which the transaction began

SQLQuery7.sql - WIN-CQQMK62867E.moviescope (WIN-CQQMK62867E\Administrator (51))*

```
dbcc page(moviescope, 1, 154, 1)
```

100 %

Messages

```
Slot 4, Offset 0x2e8, Length 154, DumpStyle BYTE

Record Type = PRIMARY_RECORD        Record Attributes =  NULL_BITMAP VARIABLE_COLUMNS
Record Size = 154
Memory Dump @0x000000001249A2E8

0000000000000000:   30000c00 05000000 c8000000 0a000000 08002800  0.......È.........(.
0000000000000014:   2e003600 4c005400 68007e00 9a006c00 65006500  ..6.L.T.h.~..l.e.e.
0000000000000028:   6c006500 65006200 72006500 74006c00 65006500  l.e.e.b.r.e.t.l.e.e.
000000000000003C:   40006100 62006300 2e006300 6f006d00 6d006100  @.a.b.c...c.o.m.m.a.
0000000000000050:   6c006500 30003900 2d003000 38002d00 31003900  l.e.0.9.-.0.8.-.1.9.
0000000000000064:   38003800 41006c00 62007500 71007500 61007200  8.8.A.l.b.u.q.u.a.r.
0000000000000078:   71007500 65003100 2d003200 30003200 2d003500  q.u.e.1.-.2.0.2.-.5.
000000000000008C:   30003600 2d003300 36003900 3100               0.6.-.3.6.9.1.
```

100 %

Query executed successfully. | WIN-CQQMK62867E (11.0 SP2) | WIN-CQQMK62867E\Admini... | moviescope | 00:00:00 | 0 rows

Using the table schema obtained earlier, the data type within this row offset is the **age** column which contains a 4-byte **int** data type.

Database Forensics Using SQL Server Management Studio (Cont'd)

Step 10: Compare the Row Logs

- Note down the **hex values** of **RowLog Contents 0** and **RowLog Contents 1** and convert them to their equivalent decimal values
- Thus, it is evident that the age entity has tampered from **25 to 200**, which was successfully determined using forensic investigation

RowLog Contents 0	RowLog Contents 1
0x19	0xc8
Decimal Value	Decimal Value
25	200

SQLQuery7.sql - WIN-CQQMK62867E.moviescope (WIN-CQQMK62867E\Administrator (51))*

```
dbcc log(moviescope, 3)
```

100 %

Results | Messages

	opyVerionInfo Source Slot Count	RowLog Contents 0	RowLog Contents 1	RowLog Contents 2	RowLog Contents 3
15995	IULL	0x20010022007F...	0x0002002400CA...	0x163CCB3549010B0000000000000004...	0x0101000C0000...
15996	IULL	NULL	NULL	NULL	NULL
15997	IULL	NULL	NULL	NULL	NULL
15998	IULL	0x19	0xC8		0x0101000C0000...
15999	IULL	NULL	NULL	NULL	NULL
16000	IULL	NULL	NULL	NULL	NULL
16001	IULL	NULL	NULL	NULL	NULL
16002	IULL	NULL	NULL	NULL	NULL
16003	IULL	0x300027001D00...		0x0101000C00003600000000000020400...	NULL
16004	IULL	0x361D00000011...		0x0101000C00003600000000000020400...	NULL

Query executed successfully. | WIN-CQQMK62867E (11.0 SP2) | WIN-CQQMK62867E\Admini... | moviescope | 00:00:01 | 16009 rows

Database Forensics Using ApexSQL DBA

Step 1: Collecting Volatile Database Data

- Gather volatile database information such as Users' login sessions, user transactions, etc.
- Use **ApexSQL DBA's ApexSQL Audit** application to track the login history
- A login event has observed with the username **anonymous** from the Client host (WIN-CQQMK628662)

ApexSQL Audit - Logon a

localhost:41400/Reports/Common/CommonReport.aspx?Report=LogonActivityHistory

Common Reports | Custom Reports | Options | Help

General
- Complete audit trail
- Backup and DBCC activities

Change and activity auditing
- DML history
- DDL history
- Access history

Security
- Security configuration history
- Logon activity history
- Permission changes
- Unauthorized access

Auditing overview
- Audit settings history
- Audit integrity checks

Date: 03/11/2016 00:00:00.000
to: 03/11/2016 23:59:59.999
Server:
Login:

Security
- Audit login
- Audit login failed
- Audit logout

Preview | Order by Date | Ascending | Descending | Date/time format: MM/dd/yyyy hh:mm

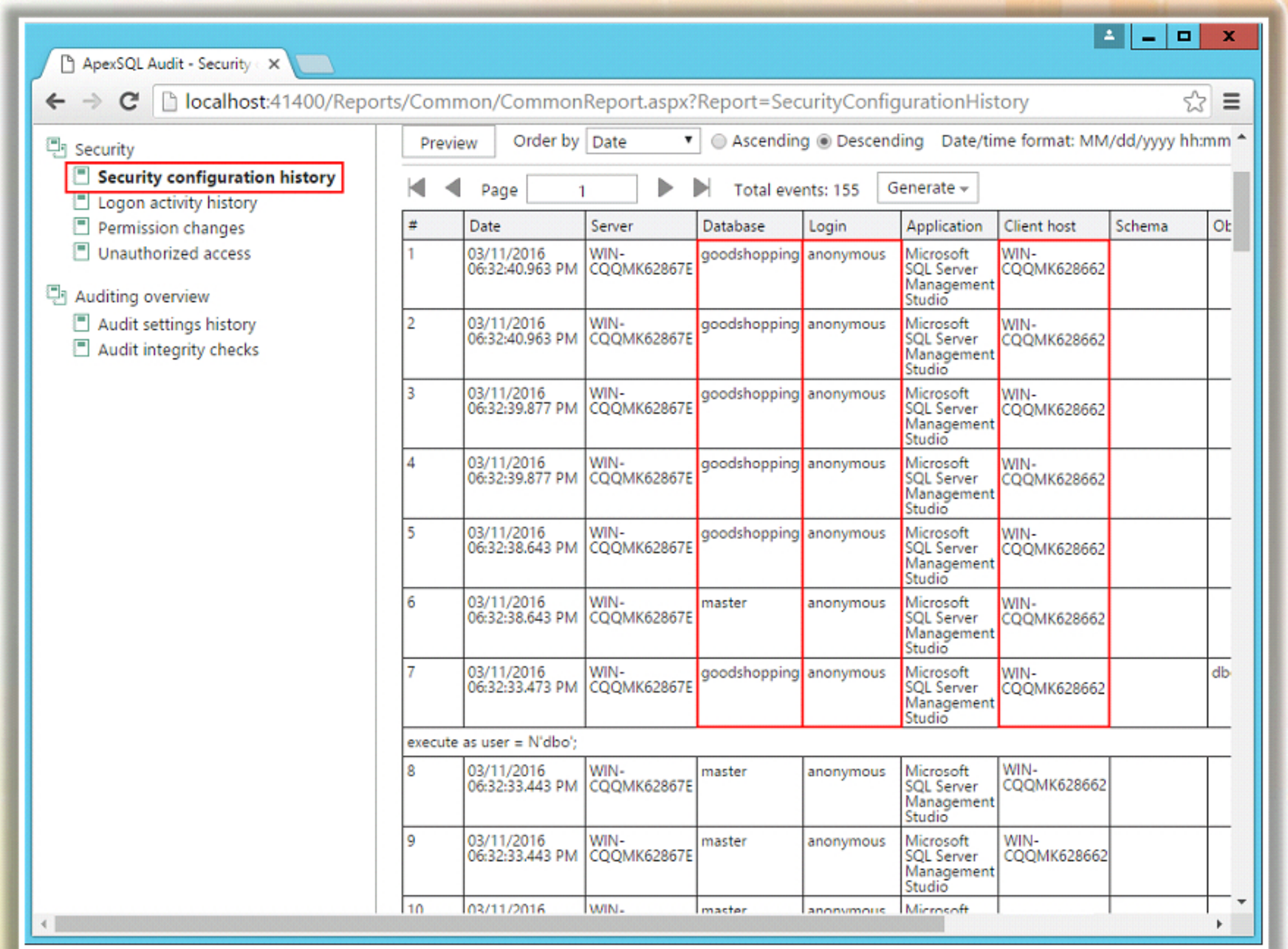
Page 13 | Total events: 470946 | Generate

#	Date	Server	Login	Application	Client host	Operation
1271	03/11/2016 06:32:26.183 PM	WIN-CQQMK62867E	anonymous	Microsoft SQL Server Management Studio	WIN-CQQMK628662	Audit logou
1272	03/11/2016 06:32:26.183 PM	WIN-CQQMK62867E	anonymous	Microsoft SQL Server Management Studio	WIN-CQQMK628662	Audit login
1273	03/11/2016 06:32:26.147 PM	WIN-CQQMK62867E	anonymous	Microsoft SQL Server Management Studio	WIN-CQQMK628662	Audit logou
1274	03/11/2016 06:32:26.147 PM	WIN-CQQMK62867E	anonymous	Microsoft SQL Server Management Studio	WIN-CQQMK628662	Audit login
1275	03/11/2016 06:32:25.740 PM	WIN-CQQMK62867E	anonymous	Microsoft SQL Server Management Studio	WIN-CQQMK628662	Audit logou
1276	03/11/2016 06:32:25.740 PM	WIN-CQQMK62867E	anonymous	Microsoft SQL Server Management Studio	WIN-CQQMK628662	Audit login
1277	03/11/2016 06:32:25.620 PM	WIN-CQQMK62867E	anonymous	Microsoft SQL Server Management Studio	WIN-CQQMK628662	Audit logou

Database Forensics Using ApexSQL DBA (Cont'd)

Step 1: Collecting Volatile Database Data (Cont'd)

- Click on **Security configuration history** to view the security related entry modifications
- It is observed that the transactions occurred on **goodshopping** database

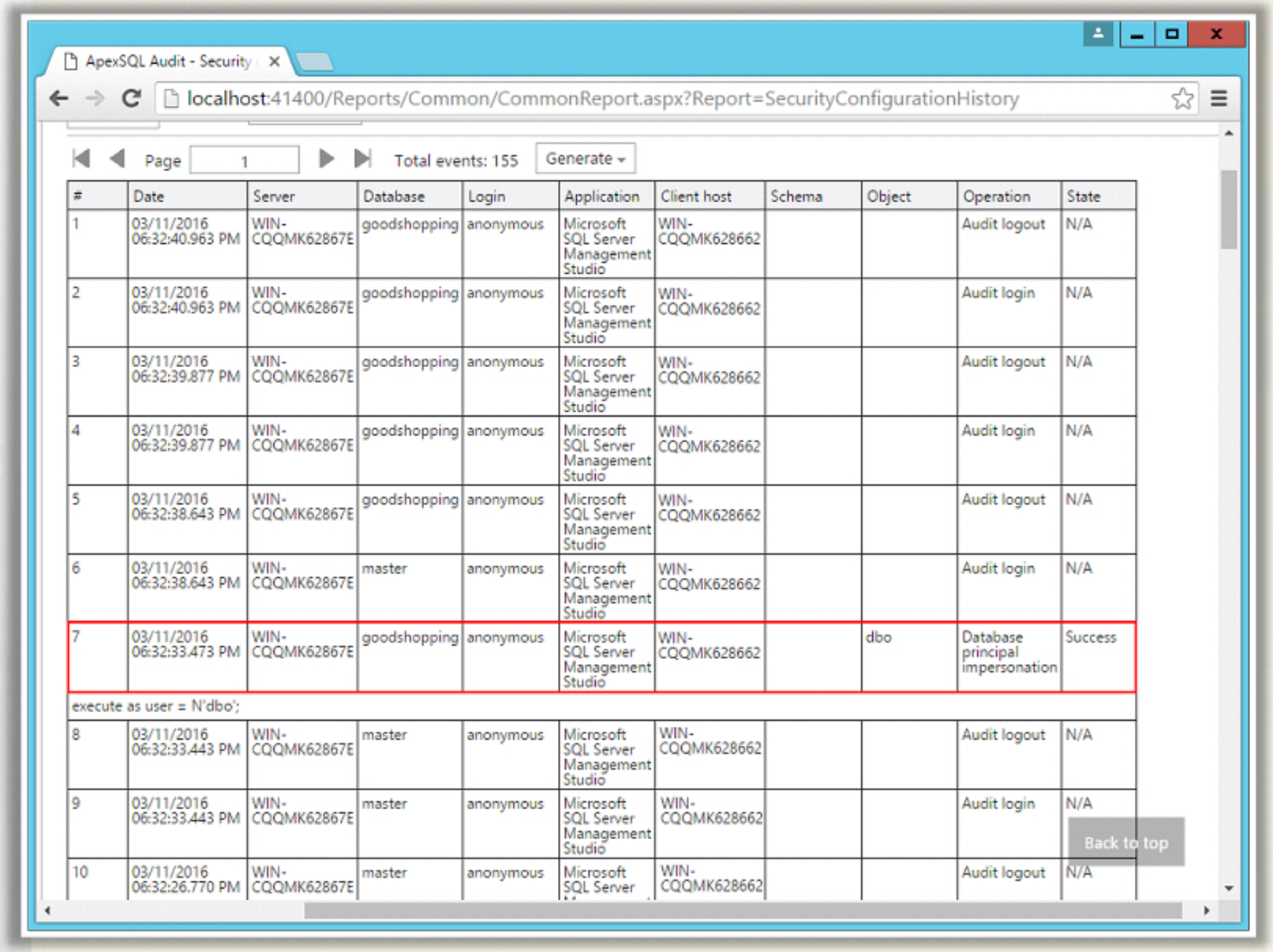


#	Date	Server	Database	Login	Application	Client host	Schema	Ob
1	03/11/2016 06:32:40.963 PM	WIN-CQQMK62867E	goodshopping	anonymous	Microsoft SQL Server Management Studio	WIN-CQQMK628662		
2	03/11/2016 06:32:40.963 PM	WIN-CQQMK62867E	goodshopping	anonymous	Microsoft SQL Server Management Studio	WIN-CQQMK628662		
3	03/11/2016 06:32:39.877 PM	WIN-CQQMK62867E	goodshopping	anonymous	Microsoft SQL Server Management Studio	WIN-CQQMK628662		
4	03/11/2016 06:32:39.877 PM	WIN-CQQMK62867E	goodshopping	anonymous	Microsoft SQL Server Management Studio	WIN-CQQMK628662		
5	03/11/2016 06:32:38.643 PM	WIN-CQQMK62867E	goodshopping	anonymous	Microsoft SQL Server Management Studio	WIN-CQQMK628662		
6	03/11/2016 06:32:38.643 PM	WIN-CQQMK62867E	master	anonymous	Microsoft SQL Server Management Studio	WIN-CQQMK628662		
7	03/11/2016 06:32:33.473 PM	WIN-CQQMK62867E	goodshopping	anonymous	Microsoft SQL Server Management Studio	WIN-CQQMK628662		db
execute as user = N'dbo';								
8	03/11/2016 06:32:33.443 PM	WIN-CQQMK62867E	master	anonymous	Microsoft SQL Server Management Studio	WIN-CQQMK628662		
9	03/11/2016 06:32:33.443 PM	WIN-CQQMK62867E	master	anonymous	Microsoft SQL Server Management Studio	WIN-CQQMK628662		
10	03/11/2016	WIN-	master	anonymous	Microsoft			

Database Forensics Using ApexSQL DBA (Cont'd)

Step 1: Collecting Volatile Database Data (Cont'd)

- Now, scroll to the right of the browser window to view all the operations performed on the database, along with the affected objects
- The result implies that a **database principal impersonation** has occurred on the object **dbo** inside the **goodshopping** database

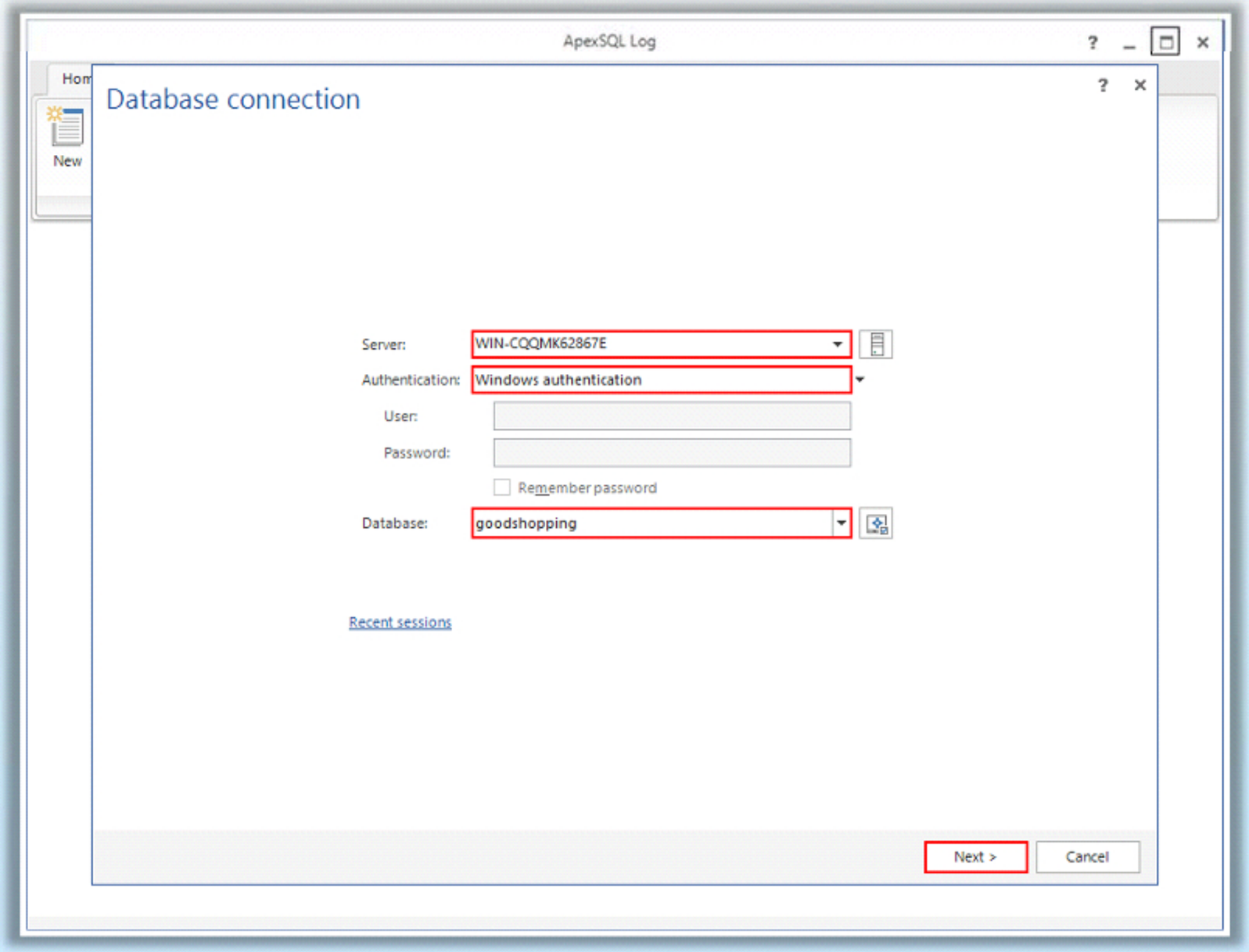


#	Date	Server	Database	Login	Application	Client host	Schema	Object	Operation	State
1	03/11/2016 06:32:40.963 PM	WIN-CQQMK62867E	goodshopping	anonymous	Microsoft SQL Server Management Studio	WIN-CQQMK628662			Audit logout	N/A
2	03/11/2016 06:32:40.963 PM	WIN-CQQMK62867E	goodshopping	anonymous	Microsoft SQL Server Management Studio	WIN-CQQMK628662			Audit login	N/A
3	03/11/2016 06:32:39.877 PM	WIN-CQQMK62867E	goodshopping	anonymous	Microsoft SQL Server Management Studio	WIN-CQQMK628662			Audit logout	N/A
4	03/11/2016 06:32:39.877 PM	WIN-CQQMK62867E	goodshopping	anonymous	Microsoft SQL Server Management Studio	WIN-CQQMK628662			Audit login	N/A
5	03/11/2016 06:32:38.643 PM	WIN-CQQMK62867E	goodshopping	anonymous	Microsoft SQL Server Management Studio	WIN-CQQMK628662			Audit logout	N/A
6	03/11/2016 06:32:38.643 PM	WIN-CQQMK62867E	master	anonymous	Microsoft SQL Server Management Studio	WIN-CQQMK628662			Audit login	N/A
7	03/11/2016 06:32:33.473 PM	WIN-CQQMK62867E	goodshopping	anonymous	Microsoft SQL Server Management Studio	WIN-CQQMK628662		dbo	Database principal impersonation	Success
execute as user = N'dbo';										
8	03/11/2016 06:32:33.443 PM	WIN-CQQMK62867E	master	anonymous	Microsoft SQL Server Management Studio	WIN-CQQMK628662			Audit logout	N/A
9	03/11/2016 06:32:33.443 PM	WIN-CQQMK62867E	master	anonymous	Microsoft SQL Server Management Studio	WIN-CQQMK628662			Audit login	N/A
10	03/11/2016 06:32:26.770 PM	WIN-CQQMK62867E	master	anonymous	Microsoft SQL Server	WIN-CQQMK628662			Audit logout	N/A

Database Forensics Using ApexSQL DBA (Cont'd)

Step 2: Examine the Database Transaction Log File

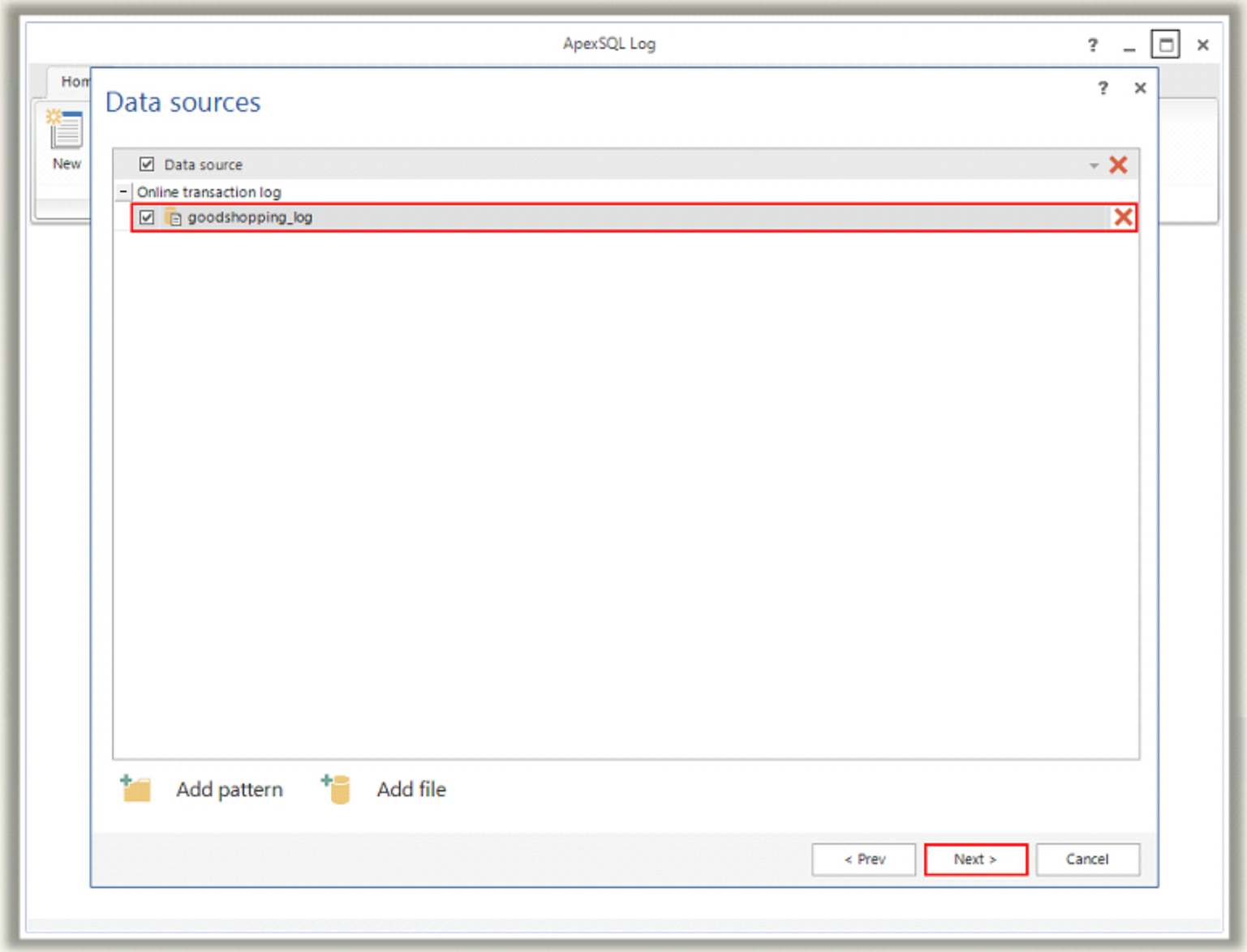
Launch **ApexSQL Log** and establish a database connection with **goodshopping** database



Database Forensics Using ApexSQL DBA (Cont'd)

Step 2: Examine the Database Transaction Log File (Cont'd)

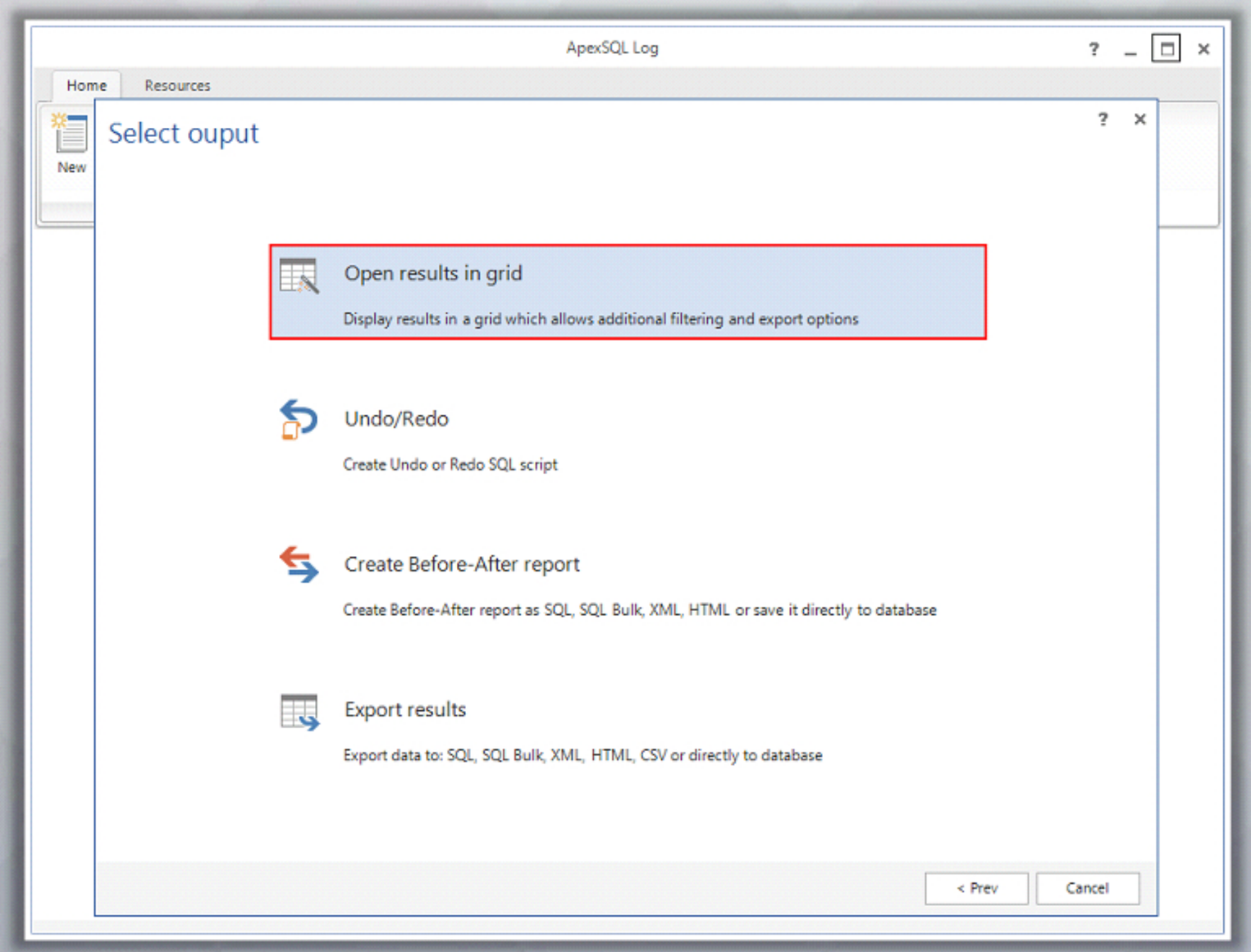
Select the goodshopping log file in the Data Sources section



Database Forensics Using ApexSQL DBA (Cont'd)

Step 2: Examine the Database Transaction Log File (Cont'd)

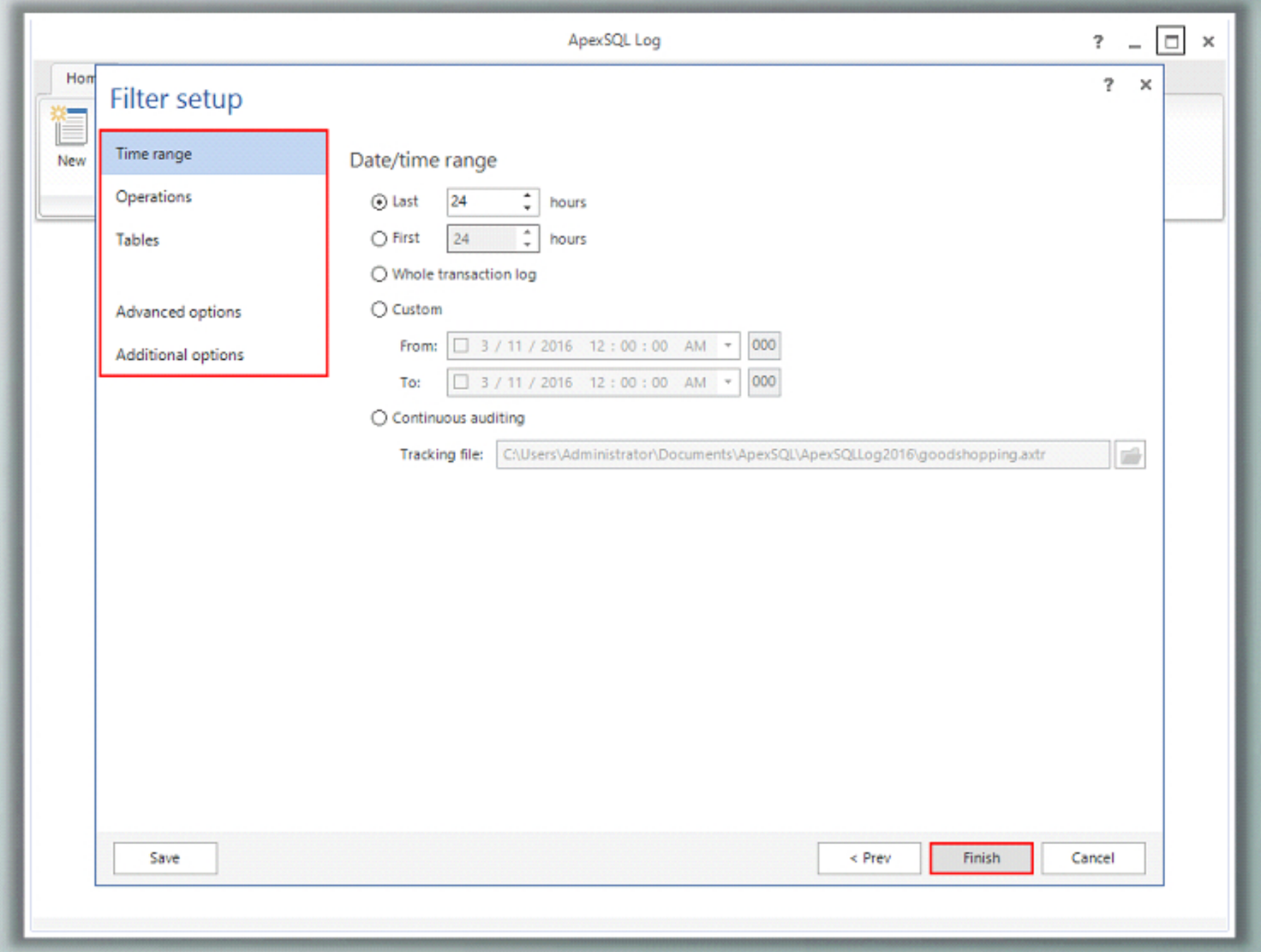
Select Open results in grid option in the **Select output** section



Database Forensics Using ApexSQL DBA (Cont'd)

Step 2: Examine the Database Transaction Log File (Cont'd)

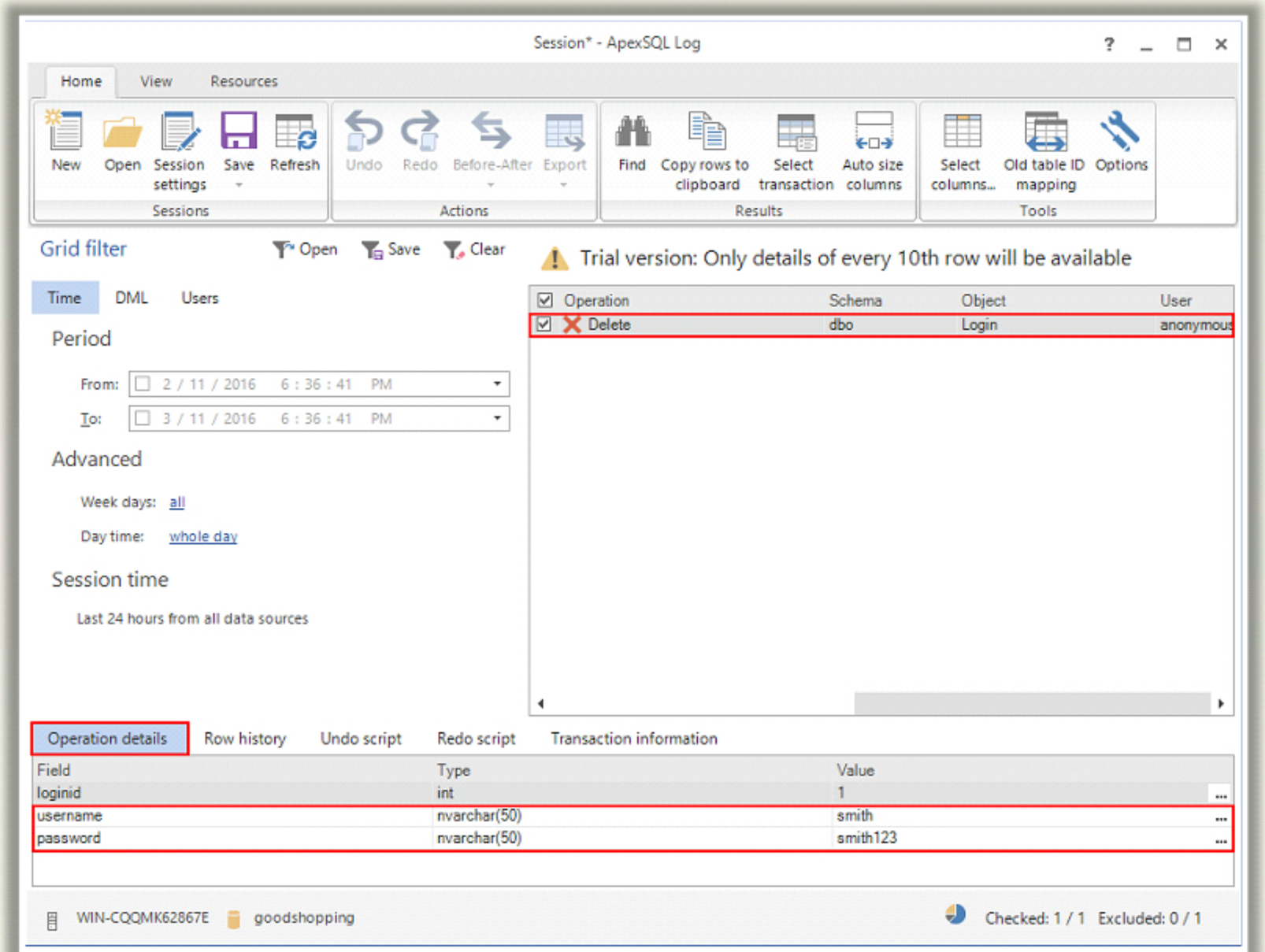
Configure the options in the Filter setup section and click **Finish**



Database Forensics Using ApexSQL DBA (Cont'd)

Step 2: Examine the Database Transaction Log File (Cont'd)

- The ApexSQL Log application examines the log file and displays the transactions occurred on the database
- A **delete** operation has been observed-on the login object by the **anonymous** user. The deleted entries (username: **smith** & password: **smith123**) can be observed under the **Operation details** tab.



Database Forensics Using ApexSQL DBA (Cont'd)

Step 2: Examine the Database Transaction Log File (Cont'd)

In the same way, examine the Row history, Undo script, Redo script and Transaction information sections to get a clear view of the transaction

Operation details | Row history | Undo script | Redo script | Transaction information

Operation	Date	User Name	LSN	loginid	username	password
Delete	3/11/2016 6:32:45 PM	anonymous	0000004A:00000157:0002	1	smith	smith123

Operation details | Row history | Undo script | Redo script | Transaction information

```
-- DELETE (0000004A:00000157:0002) done at 2016-03-11 18:32:45.266 by anonymous in transaction 0000:000006EE (Committed)
SET IDENTITY_INSERT [dbo].[Login] ON
INSERT INTO [dbo].[Login] ([loginid], [username], [password]) VALUES (1, N'smith' COLLATE SQL_Latin1_General_CP1_CI_AS, N's
SET IDENTITY_INSERT [dbo].[Login] OFF
```

Operation details | Row history | Undo script | Redo script | Transaction information

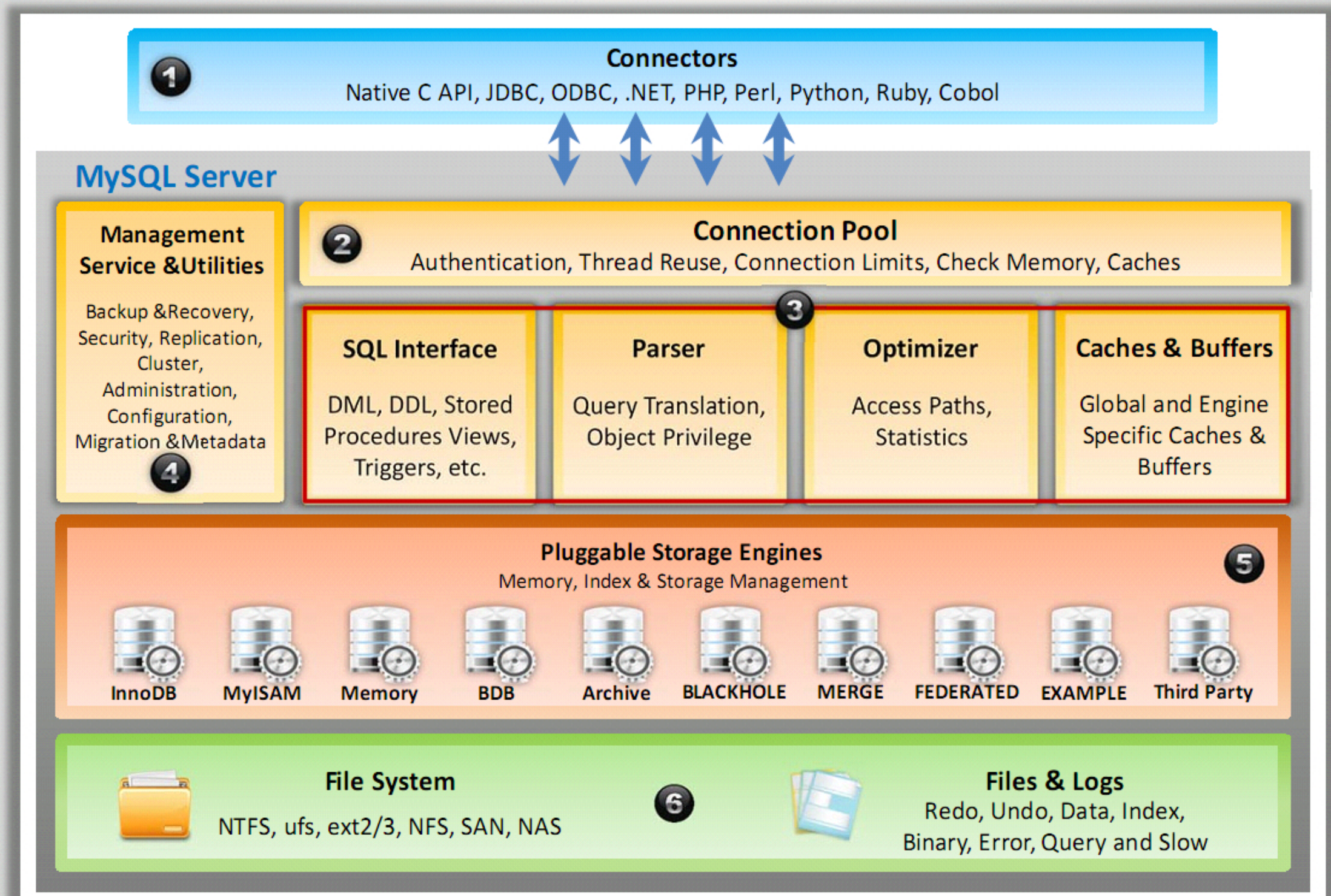
```
-- DELETE (0000004A:00000157:0002) done at 2016-03-11 18:32:45.266 by anonymous in transaction 0000:000006EE (Committed)
BEGIN TRANSACTION
DELETE FROM [dbo].[Login] WHERE /*** WARNING: WHERE CLAUSE FOR THIS STATEMENT WAS GENERATED FOR A TABLE WITH NO PRIMARY KEY
IF @@ROWCOUNT <= 1 COMMIT TRANSACTION ELSE BEGIN ROLLBACK TRANSACTION; PRINT 'ERROR: STATEMENT AFFECTED MORE THAN ONE ROW.
```

Operation details | Row history | Undo script | Redo script | Transaction information

Begin time	2016-03-11 18:32:45
State	Committed
Operation	Delete
Schema	dbo
Object	Login

MySQL Forensics

Internal Architecture of MySQL



Internal Architecture of MySQL (Cont'd)

MySQL is based on a **tiered architecture** containing subsystems and support components, which work together to respond to the queries made to the database server.

- The connectors act as a **medium** for the clients to connect to the SQL Server
- The **Connection Pool** handles all the **client connection** needs such as user authentication, memory checks, thread processing, caches, etc.
- These are the **ancillary tools** grouped by administration and enterprise services
- The SQL Interface, Parser, Optimizer, caches, and buffers together form the crucial part of the **database server**
  - **SQL Interface:** User interface that accepts **SQL syntax** and transmits results to the user
  - **Parser:** **Validates** the SQL statement syntax entered by the user
  - **Query Optimizer:** excludes **known-bad conditions** in the query prior to executing the join expression
  - **Query Execution:** It is handled by a set of specifically designed **library methods** and the results are returned by the network communication pathways library
  - **Query Cache:** caches the **query structure** and query results, unique to MySQL
  - **Cache and Buffers:** ensures availability of frequently used data in an effective way. Types of caches include Table Cache, Record Cache, Key Cache, Privilege Cache, Hostname Cache, and much more
- **Storage engines** are used to create, read and update data within a database. Various **Storage engines** supported by MySQL are:
  - InnoDB (default storage engine in MySQL version >= 5.5.5)
  - MyISAM (default storage engine in MySQL version < 5.5.5)
  - Memory
  - BDB
  - ARCHIVE
  - BLACKHOLE
  - MERGE
  - FEDERATED
  - EXAMPLE
  - Other third party storage engines
- This section stores all the **database files** and the logs that store all the transactions occurred within the databases

Structure of the Data Directory

C|HFI Computer Hacking Forensic Investigator

1. MySQL server stores all the databases, status and log files; along with the data managed by the server under the data directory

2. By default, the data directory is located at C:\ProgramData\MySQL\MySQL Server 5.n\ in Windows based machines

3. All the databases are stored as folders in the data directory. The files within these folders (databases) correspond to the tables, views, and triggers within that database

Structure of the Data Directory (Cont'd)

In addition to the files and folders mentioned earlier, the data directory might even contain:

- **auto.cnf** file containing the **server_uuid** which is used to uniquely identify a server
- The **server's process ID** (PID) file, which stores the MySQL server's process ID
- **Status and log files** generated by the server, which store information related to the operations performed on the server. These files include:

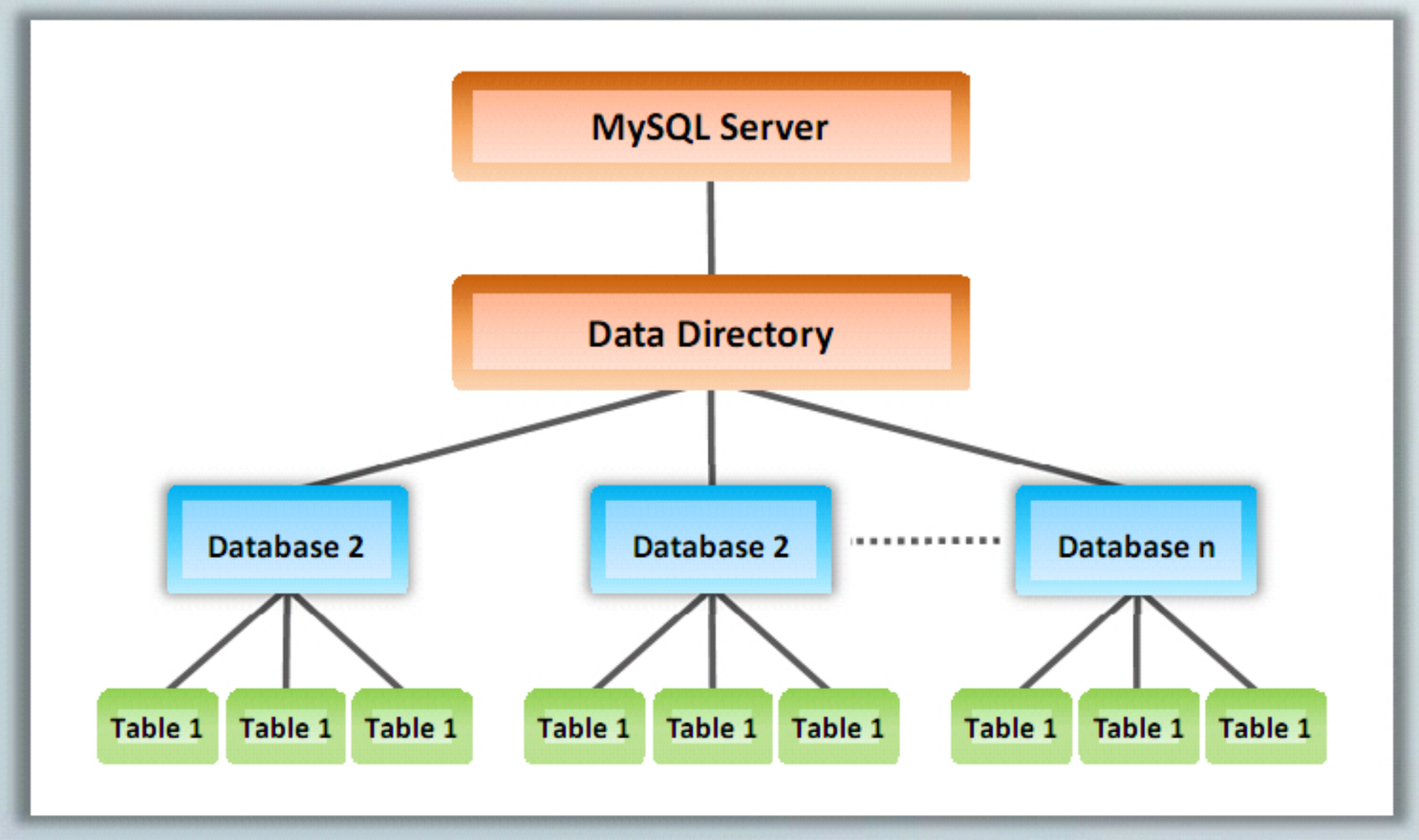
- Process ID file (HOSTNAME.pid)
- Error log (HOSTNAME.err)
- General query log (HOSTNAME.log)
- Binary log (HOSTNAME-bin.nnnnnn)
- Binarylog index (HOSTNAME-bin.index)

- Relay log (HOSTNAMErelay-bin.n)
- Relay log index (HOSTNAMErelay-bin.index)
- Master info file (master.info)
- Relay log info file (relay-log.info)
- Slow-query log (HOSTNAMEslow.log)

Structure of the Data Directory (Cont'd)

The **InnoDB** storage engine contains two types of logs:

- Undo logs, help a forensic examiner to roll back the transactions
- Redo logs, help a forensic examiner to re-execute the transactions (**ib_logfile0** and **ib_logfile1**)
- **Ibdata1**, store InnoDB's permanent table records



Note: This hierarchical structure mentioned above might vary with the storage engine used

Structure of the Data Directory (Cont'd)

The database structure varies depending on the **storage engine** (MyISAM/InnoDB) used by **MySQL**

MyISAM Storage Engine

- The databases are stored as directories in the **data directory**
- All the database tables are stored as files inside the database folder. These files carry the name of the tables, and are categorized into three formats:
  - **[tablename].frm**, contains the table format
  - **[tablename].myd**, contains the table data
  - **[tablename].myi**, contains the table indices
- In addition, the database directory contains a db.opt text file, which stores the characteristics of the database

InnoDB Storage Engine

- All the **database tables** are stored as files inside the database folder. These files carry the name of the tables, and are categorized in two formats:
  - **[tablename].frm**, contains the table format
  - **[tablename].ibd**, containing the data and index of the table

MySQL Forensics

- MySQL contains **Information_Schema** table which provides access to database metadata
- There are two phases in database analysis:

❖ **Evidence collection**

Use MySQL utility programs to make copies of databases and log files that hold substantial amount of information required for forensic investigation

❖ **Evidence examination**

Identify the fraudulent activity and reconstruct the tampered data

Viewing the Information Schema

- MySQL stores information related to all the databases, along with the **read-only tables**, in the Information Schema
- **Information** schema provides access to the **database metadata**

E.g. 1: Viewing a list of all the tables in "**wordpress**" database in reverse alphabetic order, displaying the table name, table type, and storage engine used for the database

Administrator: C:\Windows\system32\cmd.exe - mysql -u root -p

```
mysql> SELECT table_name, table_type, engine FROM information_schema.tables  WHERE table_schema = 'w
ordpress' ORDER BY table_name DESC;
+-----------------------+------------+--------+
| table_name            | table_type | engine |
+-----------------------+------------+--------+
| wp_users              | BASE TABLE | InnoDB |
| wp_usermeta           | BASE TABLE | InnoDB |
| wp_term_taxonomy      | BASE TABLE | InnoDB |
| wp_term_relationships | BASE TABLE | InnoDB |
| wp_terms              | BASE TABLE | InnoDB |
| wp_posts              | BASE TABLE | InnoDB |
| wp_postmeta           | BASE TABLE | InnoDB |
| wp_options            | BASE TABLE | InnoDB |
| wp_links              | BASE TABLE | InnoDB |
| wp_comments           | BASE TABLE | InnoDB |
| wp_commentmeta        | BASE TABLE | InnoDB |
+-----------------------+------------+--------+
11 rows in set (0.00 sec)
```

E.g. 2: Viewing all the tables containing more than 10 rows

Administrator: C:\Windows\system32\cmd.exe - mysql -u root -p

```
mysql> SELECT CONCAT(table_schema,'.',table_name) as table_name,table_rows FROM  information_schema
.tables WHERE table_rows > 10 AND table_schema not in ('information_schema','mysql','performance_sch
ema');
+----------------------+------------+
| table_name           | table_rows |
+----------------------+------------+
| wordpress.wp_options |         78 |
| wordpress.wp_usermeta |         27 |
+----------------------+------------+
2 rows in set (0.00 sec)
```

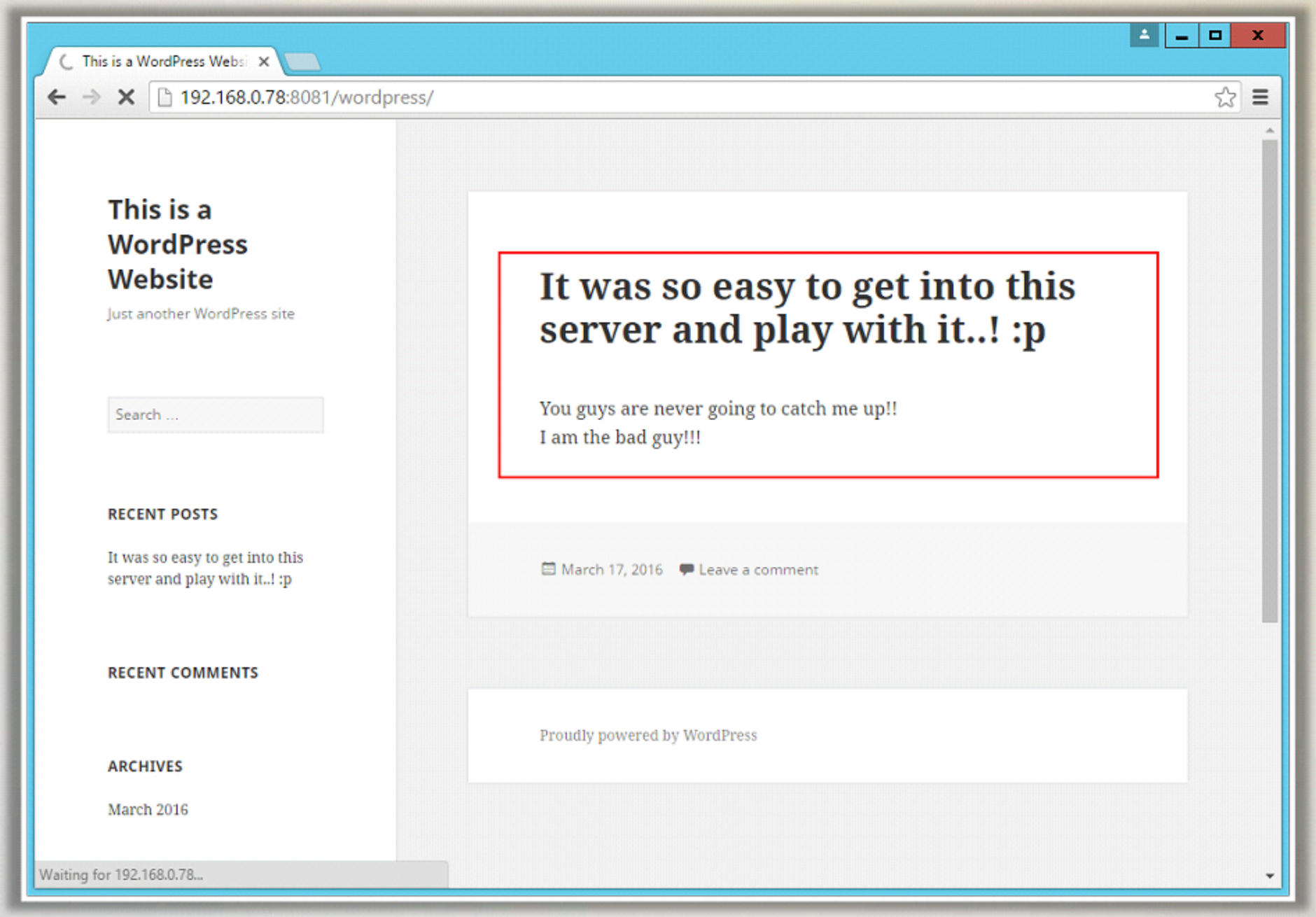
MySQL Utility Programs For Forensic Analysis

Forensic examinations are performed using some of the following MySQL utility programs:

Mysqldump	To dump single or multiple databases for backup purpose
Mysqlaccess	To check the access privileges defined for host name, user name, etc.
myisamlog	To process the MyISAM log file and perform recovery operation, display version information, etc., depending on the situation
Myisamchk	To attain the status of the MyISAM table, identify the corrupted tables, repair the corrupted tables, etc.
Mysqlbinlog	To display the content of bin logs (mysql-bin.nnnnnn) in text format
mysqldbexport	To export metadata or data, or both from one or more databases

Common Scenario for Reference

Consider a scenario where a post in the index page of **WordPress website** was appeared suspicious:



MySQL Forensics for WordPress Website Database: Scenario 1

Identifying a **malicious user** and collecting all the **posts** made by him/her through **command line client**

Scenario 1: Collect the Evidences

The primary task while performing a **forensic investigation** is to examine the error log files and check if there are any unintended **startup/shutdown** events, as well as **critical errors** while the server is running

*C:\Bitnami\wordpress-4.4.2-3\mysql\data\WIN-CQQMK62867E.err - Notepad++ [Administrator]

File Edit Search View Encoding Language Settings Macro Run Plugins Window ?

WIN-CQQMK62867E.err

```
1811  2016-03-17 11:43:52 10108 [Note] InnoDB: Using atomics to ref count buffer pool pages
1812  2016-03-17 11:43:52 10108 [Note] InnoDB: The InnoDB memory heap is disabled
1813  2016-03-17 11:43:52 10108 [Note] InnoDB: Mutexes and rw_locks use Windows interlocked functions
1814  2016-03-17 11:43:52 10108 [Note] InnoDB: Memory barrier is not used
1815  2016-03-17 11:43:52 10108 [Note] InnoDB: Compressed tables use zlib 1.2.3
1816  2016-03-17 11:43:52 10108 [Note] InnoDB: Not using CPU crc32 instructions
1817  2016-03-17 11:43:52 10108 [Note] InnoDB: Initializing buffer pool, size = 128.0M
1818  2016-03-17 11:43:52 10108 [Note] InnoDB: Completed initialization of buffer pool
1819  2016-03-17 11:43:52 10108 [Note] InnoDB: Highest supported file format is Barracuda.
1820  2016-03-17 11:43:52 10108 [Note] InnoDB: 128 rollback segment(s) are active.
1821  2016-03-17 11:43:52 10108 [Note] InnoDB: Waiting for purge to start
1822  2016-03-17 11:43:53 10108 [Note] InnoDB: 5.6.29 started; log sequence number 3201871
1823  2016-03-17 11:43:53 10108 [Note] Server hostname (bind-address): '*'; port: 3306
1824  2016-03-17 11:43:53 10108 [Note] IPv6 is available.
1825  2016-03-17 11:43:53 10108 [Note]   - '::' resolves to '::';
1826  2016-03-17 11:43:53 10108 [Note] Server socket created on IP: '::'.
1827  2016-03-17 11:43:53 10108 [Note] Event Scheduler: Loaded 0 events
1828  2016-03-17 11:43:53 10108 [Note] C:\Bitnami\wordpress-4.4.2-3\mysql\bin\mysqld.exe: ready for connectio
1829  Version: '5.6.29'  socket: ''  port: 3306  MySQL Community Server (GPL)
1830  2016-03-17 12:25:08 10108 [Warning] IP address '192.168.0.13' could not be resolved: No such host is kn
1831
```

Normal text file | length : 140857 lines : 1831 | Ln : 1830 Col : 20 Sel : 0 | 0 | Dos\Windows | UTF-8 | INS

It is evident from the error log that a remote login attempt has occurred on the server

Scenario 1: Examine the Log Files

Since the **General Query log file** stores the client connections and activities performed on the server, examine the file to see if any **suspicious events** were recorded

*C:\Bitnami\wordpress-4.4.2-3\mysql\data\WIN-CQQMK62867E.err - Notepad++ [Administrator]

File Edit Search View Encoding Language Settings Macro Run Plugins Window ?

WIN-CQQMK62867E.log

```
55229          130 Connect   Access denied for user 'monty'@'192.168.0.13' (using password: YES)
55230          131 Connect   monty@192.168.0.13 on
55231          131 Connect   Access denied for user 'monty'@'192.168.0.13' (using password: YES)
55232          132 Connect   monty@192.168.0.13 on
55233          132 Connect   Access denied for user 'monty'@'192.168.0.13' (using password: YES)
55234          133 Connect   monty@192.168.0.13 on
55235          133 Connect   Access denied for user 'monty'@'192.168.0.13' (using password: YES)
55236          134 Connect   monty@192.168.0.13 on
55237          134 Connect   Access denied for user 'monty'@'192.168.0.13' (using password: YES)
55238          135 Connect   monty@192.168.0.13 on
55239          135 Connect   Access denied for user 'monty'@'192.168.0.13' (using password: YES)
55240          136 Connect   monty@192.168.0.13 on
55241          136 Connect   Access denied for user 'monty'@'192.168.0.13' (using password: YES)
55242          137 Connect   monty@192.168.0.13 on
55243  160317 12:36:03  138 Connect   monty@192.168.0.13 on
55244          138 Query select @@version_comment limit 1
55245  160317 12:37:42  138 Query show databases
55246  160317 12:37:47  138 Query SELECT DATABASE()
55247          138 Init DB   wordpress
55248  160317 13:19:38  138 Query set global general log=0
55249
```

Normal text file | length : 3705579 lines : 55250 | Ln : 55248 Col : 53 Sel : 0 | 0 | UNIX | ANSI | INS

Scenario 1: Analyze the General Log

C|HFI Computer Hacking Forensic Investigator

Brute-force Attack:

01

- It is evident that the MySQL Server was subjected to a brute-force attack from the IP address 192.168.0.13, in an attempt to gain database credentials
- The brute-force attack was successful and the attacker cracked the user credentials of a database user named Monty

02

A connection was established from the attacker address 192.168.0.13

03

The attacker viewed the databases in the server, and selected the database 'WordPress'

04

The attacker disabled the general query log, which makes it difficult to trace the transactions performed on the server

Scenario 1: Take a Backup of the Database

- Since a **malicious activity** is evident, a backup of the database has to be made for further **forensic investigation**
- Take a backup of the database using **mysqldump** command line utility

Administrator: C:\Windows\system32\cmd.exe

```
C:\wamp\bin\mysql\mysql5.5.24\bin>mysqldump -u root -p --database wordpress > wordpress_evidence.sql
Enter password:

C:\wamp\bin\mysql\mysql5.5.24\bin>
```

Scenario 1: Create an Evidence Database

- The database backup has been taken from the **affected machine**
- The next step would be to create a database in the **forensic examiner's machine** and dump the contents of the previously taken backup

- Log in to **mysql** server in the forensic examiner's machine
- Create a database with the same name as that in the **affected machine**
- Exit the **mysql terminal**
- Copy all the contents of the **dump file** to the newly created database

Administrator: Command Prompt

```
C:\wamp\bin\mysql\mysql5.5.24\bin>mysql -u root -p   1
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7
Server version: 5.5.24-log MySQL Community Server (GPL)

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> create database wordpress;   2
Query OK, 1 row affected (0.00 sec)

mysql> \q   3
Bye

C:\wamp\bin\mysql\mysql5.5.24\bin>mysql -u root -p wordpress < wordpress_evidenc
e.sql   4
Enter password:

C:\wamp\bin\mysql\mysql5.5.24\bin>
```

Scenario 1: Select the Database

- The next task would be to analyze the **affected database**
- Login to **mysql** (forensic examiner's machine) and select the **wordpress** database from the **command prompt**

Administrator: Command Prompt - mysql -u root -p

```
C:\wamp\bin\mysql\mysql5.5.24\bin>mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 5.5.24-log MySQL Community Server (GPL)

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| test               |
| wordpress          |
+--------------------+
5 rows in set (0.00 sec)

mysql> use wordpress;
Database changed
mysql> _
```

Scenario 1: View the Tables in the Database

1. View the tables in **wordpress database** and check if any tables are missing

2. In this scenario, all the tables appear to be present in the **database**

Administrator: Command Prompt - mysql -u root -p

```
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| test               |
| wordpress          |
+--------------------+
5 rows in set (0.00 sec)

mysql> use wordpress;
Database changed
mysql> show tables;
+-----------------------+
| Tables_in_wordpress   |
+-----------------------+
| wp_commentmeta        |
| wp_comments           |
| wp_links              |
| wp_options            |
| wp_postmeta           |
| wp_posts              |
| wp_term_relationships |
| wp_term_taxonomy      |
| wp_terms              |
| wp_usermeta           |
| wp_users              |
+-----------------------+
11 rows in set (0.00 sec)

mysql> _
```

Scenario 1: View the Users in the Database

The next task would be to analyze the **user accounts'** tables and see if there is/are any **unauthorized user accounts** in the database

Administrator: Command Prompt - mysql -u root -p

```
mysql> select * from wp_users;
+----+------------+------------------------------------+-----------------+-----------------+
| ID | user_login | user_pass                          | user_nicename   | user_email      |
+----+------------+------------------------------------+-----------------+-----------------+
|  1 | admin      | $P$BawZBPIuhkGfUaz9Nt3kBSdk4KsMTd. | admin           | dsada@gmail.com |
| 123 | bad_guy    | $P$Bo0JMj76Was0M/PMuOT8ym6zFcpaNN/ | anonymous_hacker | badguy@abc.com |
+----+------------+------------------------------------+-----------------+-----------------+
2 rows in set (0.00 sec)

mysql>
```

- In this process, a **user account** was found, with the login name **bad_guy**
- Make a note of the **user ID** and check all posts user has made on the **website**

Scenario 1: View Columns in the Table

I View the columns in the **wp_posts** table, to view the table structure

II Make a note of the **post_author** field corresponds to the posts made by the **malicious user**

Administrator: Command Prompt - mysql -u root -p

```
mysql> show columns in wp_posts;
+-----------------------+---------------------+------+-----+---------------------+----------------+
| Field                 | Type                | Null | Key | Default             | Extra          |
+-----------------------+---------------------+------+-----+---------------------+----------------+
| ID                    | bigint(20) unsigned | NO   | PRI | NULL                | auto_increment |
| post_author           | bigint(20) unsigned | NO   | MUL | 0                   |                |
| post_date             | datetime            | NO   |     | 0000-00-00 00:00:00 |                |
| post_date_gmt         | datetime            | NO   |     | 0000-00-00 00:00:00 |                |
| post_content          | longtext            | NO   |     | NULL                |                |
| post_title            | text                | NO   |     | NULL                |                |
| post_excerpt          | text                | NO   |     | NULL                |                |
| post_status           | varchar(20)         | NO   |     | publish             |                |
| comment_status        | varchar(20)         | NO   |     | open                |                |
| ping_status           | varchar(20)         | NO   |     | open                |                |
| post_password         | varchar(20)         | NO   |     |                     |                |
| post_name             | varchar(200)        | NO   | MUL |                     |                |
| to_ping               | text                | NO   |     | NULL                |                |
| pinged                | text                | NO   |     | NULL                |                |
| post_modified         | datetime            | NO   |     | 0000-00-00 00:00:00 |                |
| post_modified_gmt     | datetime            | NO   |     | 0000-00-00 00:00:00 |                |
| post_content_filtered | longtext            | NO   |     | NULL                |                |
| post_parent           | bigint(20) unsigned | NO   | MUL | 0                   |                |
| guid                  | varchar(255)        | NO   |     |                     |                |
| menu_order            | int(11)             | NO   |     | 0                   |                |
| post_type             | varchar(20)         | NO   | MUL | post                |                |
| post_mime_type        | varchar(100)        | NO   |     |                     |                |
| comment_count         | bigint(20)          | NO   |     | 0                   |                |
+-----------------------+---------------------+------+-----+---------------------+----------------+
23 rows in set (0.00 sec)
```

Scenario 1: Collect the Posts Made by the User

The next task would be to **dump** all the posts made by the user

Use the **post_author** and the user ID to retrieve all the posts made by the user

Administrator: Command Prompt - mysql -u root -p

```
| post_parent          | bigint(20) unsigned | NO   | MUL | 0                   |                |
| guid                 | varchar(255)        | NO   |     |                     |                |
| menu_order           | int(11)             | NO   |     | 0                   |                |
| post_type            | varchar(20)         | NO   | MUL | post                |                |
| post_mime_type       | varchar(100)        | NO   |     |                     |                |
| comment_count        | bigint(20)          | NO   |     | 0                   |                |
+----------------------+---------------------+------+-----+---------------------+----------------+
23 rows in set (0.00 sec)

mysql> select * from wp_posts
    -> where post_author = '123'
    -> into outfile 'evidence.txt';
Query OK, 4 rows affected (0.00 sec)

mysql>
```

Scenario 1: Examine the Posts Made by the User

Thus, the **posts** made by the malicious user are **attained**. These posts can be examined and used for further **investigation**.

C:\wamp\bin\mysql\mysql5.5.24\data\wordpress\evidence.txt - Notepad++

File Edit Search View Encoding Language Settings Macro Run Plugins Window ?

evidence.txt

```
4   123 2016-03-17 08:45:35 0000-00-00 00:00:00     Auto Draft      auto-draft  open    open
5   123 2016-03-17 08:46:18 2016-03-17 08:46:18 Welcome to WordPress. This is your first post. Edit or delete :
6   123 2016-03-17 08:48:44 2016-03-17 08:48:44 You guys are never going to catch me up!!
\
I am the bad guy!!! It was so easy to get into this server and play with it..! :p       publish open    open
7   123 2016-03-17 08:48:44 2016-03-17 08:48:44 You guys are never going to catch me up!!
\
I am the bad guy!!! It was so easy to get into this server and play with it..! :p       inherit open    open
```

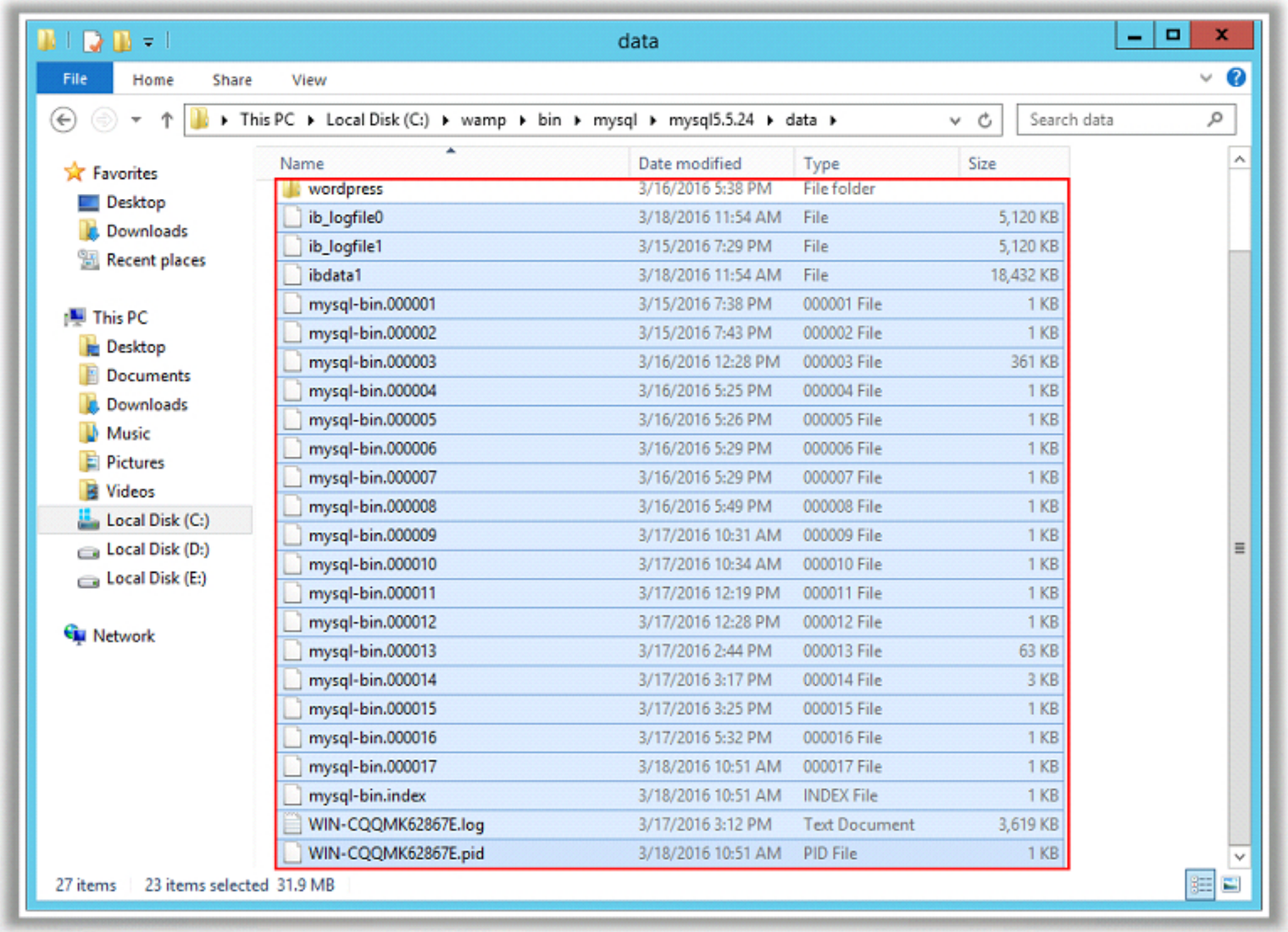
Normal text file | length : 1106 lines : 9 | Ln : 9 Col : 1 Sel : 0 | 0 | UNIX | UTF-8 w/o BOM | INS

MySQL Forensics for WordPress Website Database: Scenario 2

Tracking the Events Performed by the Malicious User (MyISAM Storage Engine) and Recovering the Deleted Data

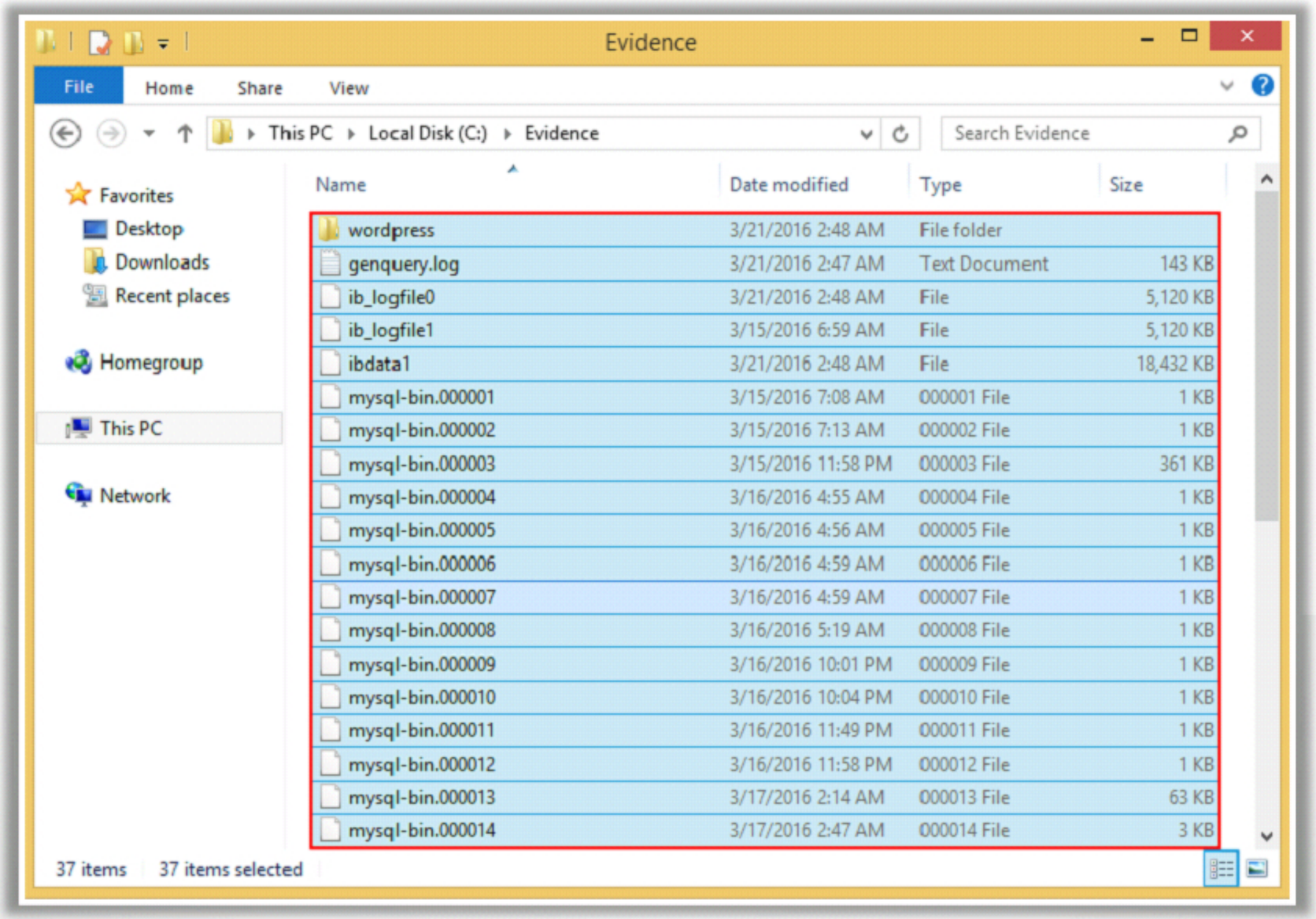
MySQL Forensics for WordPress Website Database: Scenario 2

- This scenario is a continuation of the first scenario
- Here, we shall analyze all the **log files** to trace the **activities performed** by the malicious user
- Therefore, the primary task here, is to collect all the logs and the **WordPress** folder (database) from the affected machine and dump them in the forensic examiner's machine



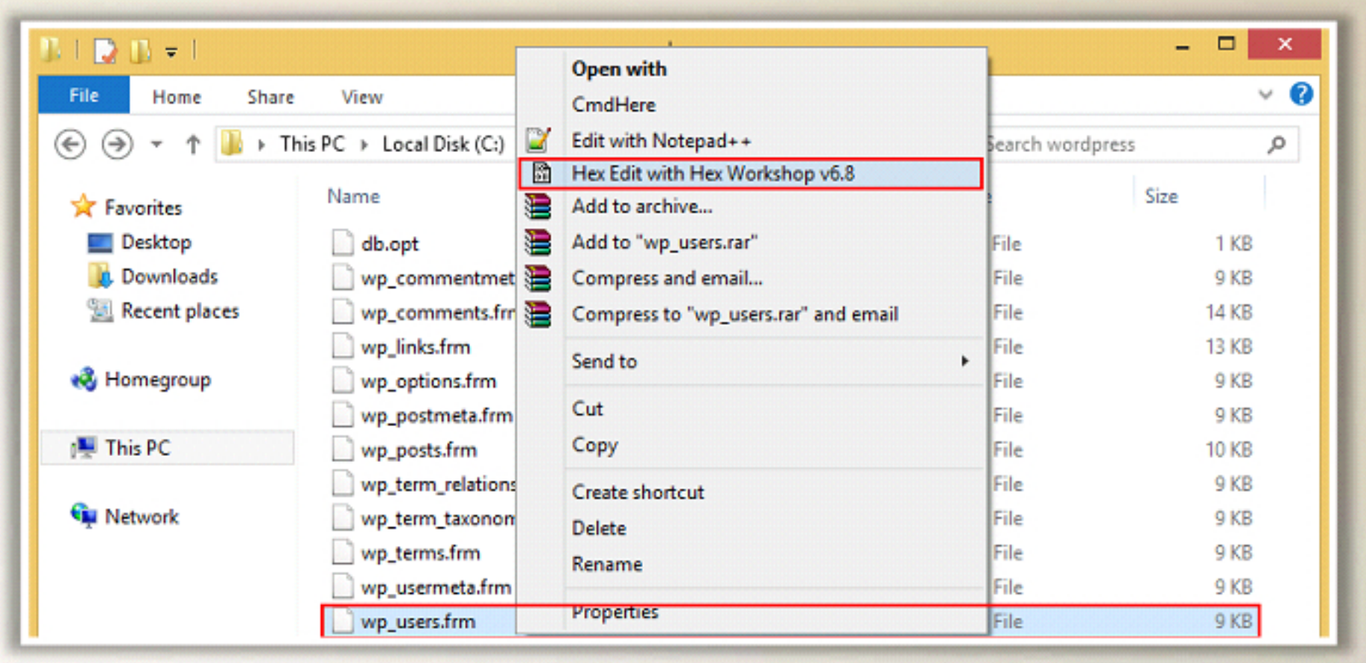
Scenario 2: Collect the Database and all the Logs

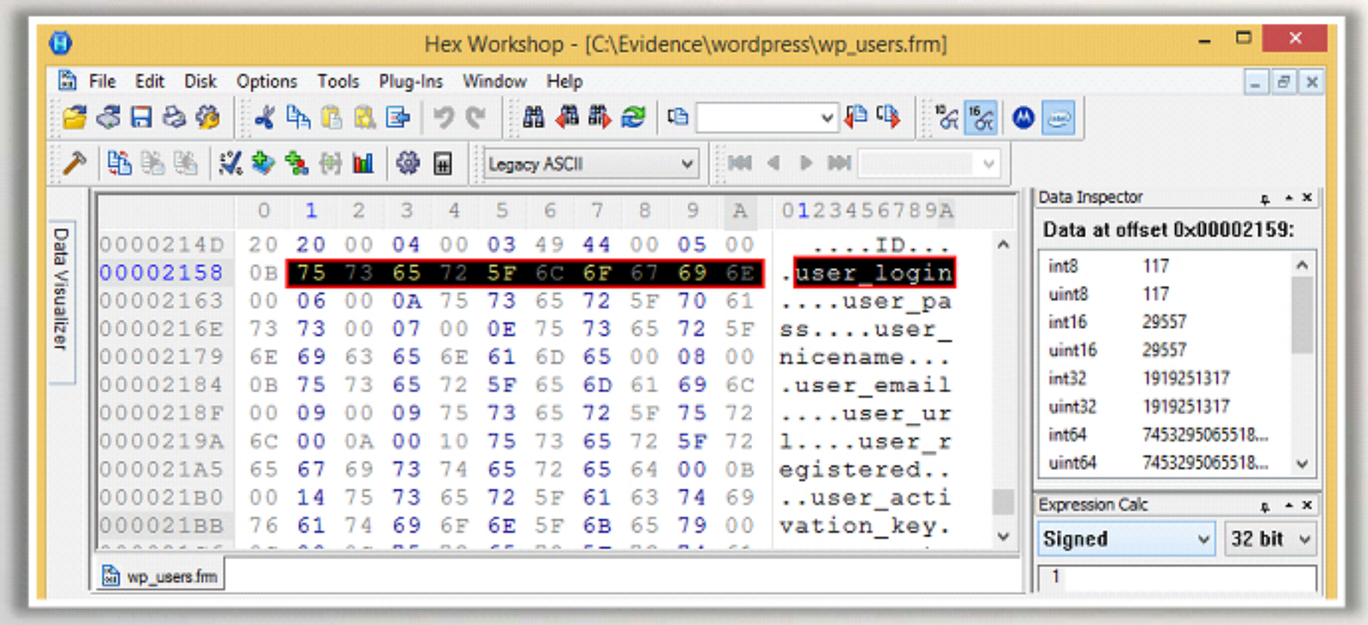
- Dump all the files in the forensic examiner's machine in a folder named **Evidence**



Scenario 2: Examine the .frm Files

- Analyzing the **.frm** files help a forensic examiner to understand the table format and the terms related to the table content.
- Since the malicious user created a user account for himself with the login name **bad_guy**, you may analyze the **wp_users.frm** file with a hex editor to view the column name (along with its hexadecimal equivalent) that contains a list of login names associated with the users.

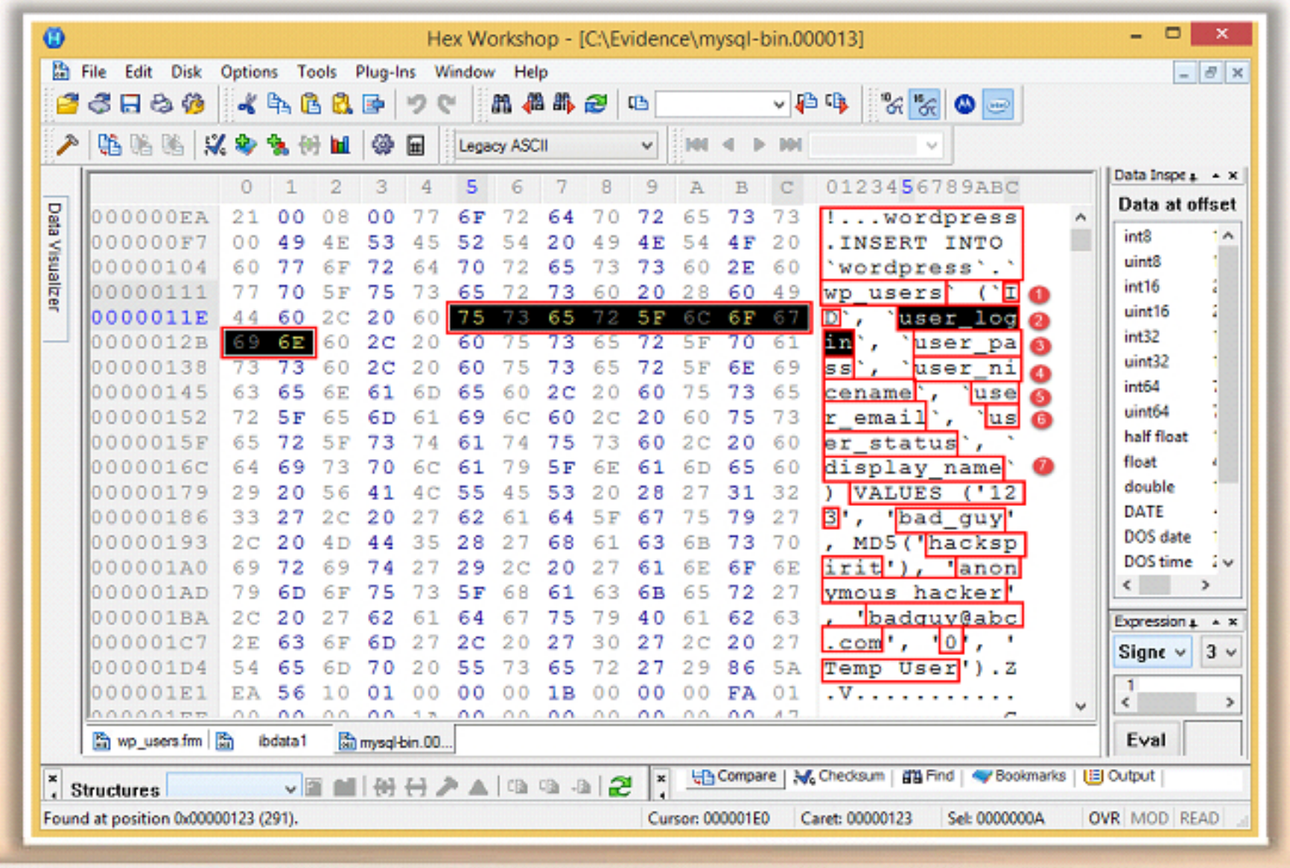




- It is observed that the login names are stored under the **user_login** column whose hexadecimal equivalent is **757365725F6C6F67696E**.
- Using this phrase, we shall first find for the attacker's login name "**bad_guy**" from the binary logs, and from there on, we shall trace the user activities performed by the malicious user.

Scenario 2: Examine the Binary Logs

- Binary logs allow a forensic examiner to trace all the events that occurred on the MySQL Server.
- Examine each binary log for the text string **user_login** or hex value **757365725F6C6F67696E**.



Detailed examination of the binary files found that one of the binary files recorded an event where a query is executed for creating a user account with the:

1. User ID – 123
2. Login name – bad_guy
3. Password – hackspirit
4. Nice name – anonymous_hacker
5. Email ID – badguy@abc.com
6. User status – 0
7. Display name – temp user

The next task would be to examine the binary logs and trace the operations performed by the forensic investigator

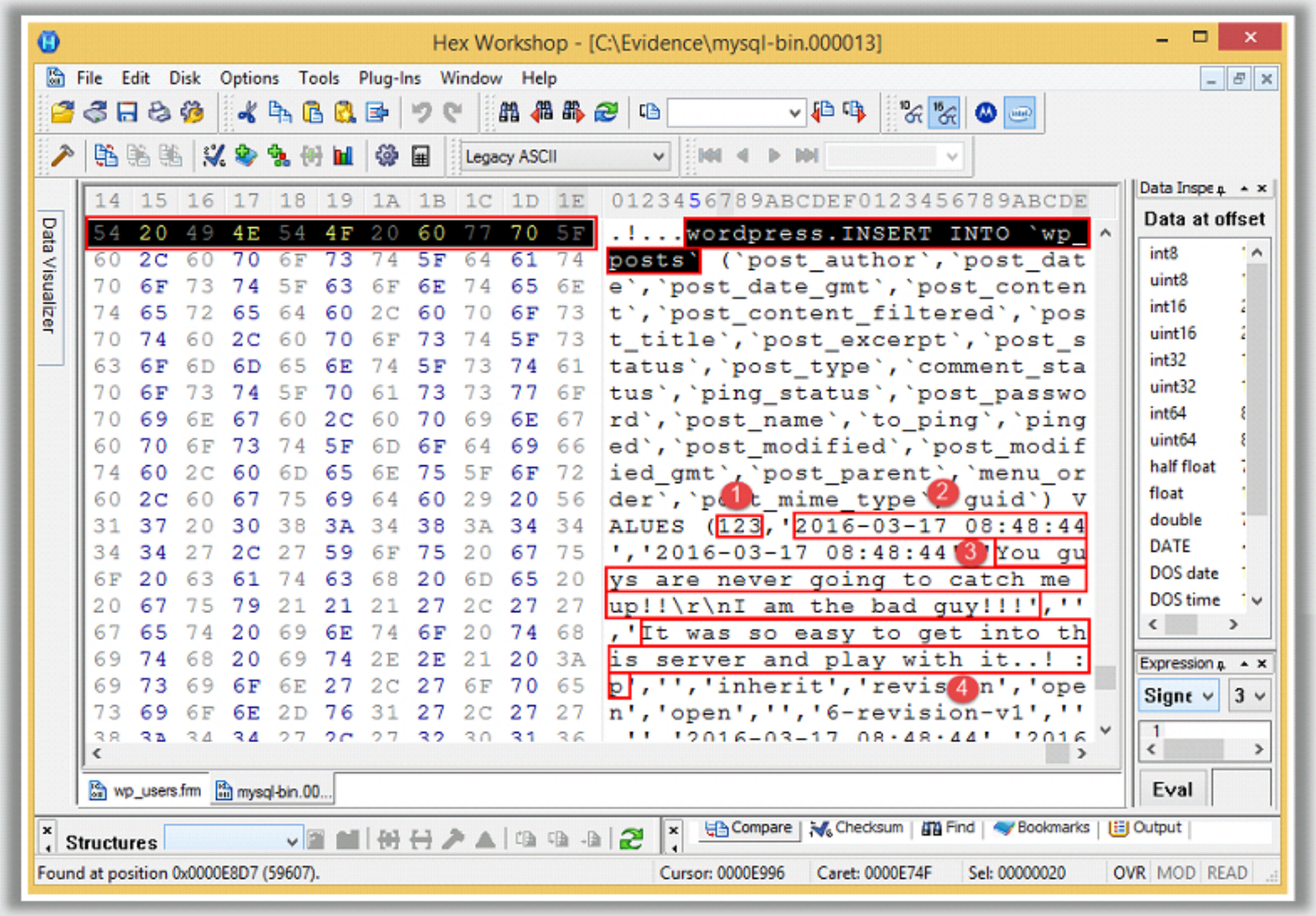
Note: Examine only those activities which correspond to the user ID/port_author ID **123**

Scenario 2: Examine the Binary Logs (Cont'd)

- Scroll down the binary logs one by one to see the **logs** corresponding to the **malicious user's actions**

It is observed that a post was made by the attacker (post_author id: **123**) on **17th march, 2016** at GMT **08:48:44**. The post title was: "It was so easy to get into this server and play with it..! :p" and the post content being "You guys are never going to catch me up!! I am the bad guy!!!"

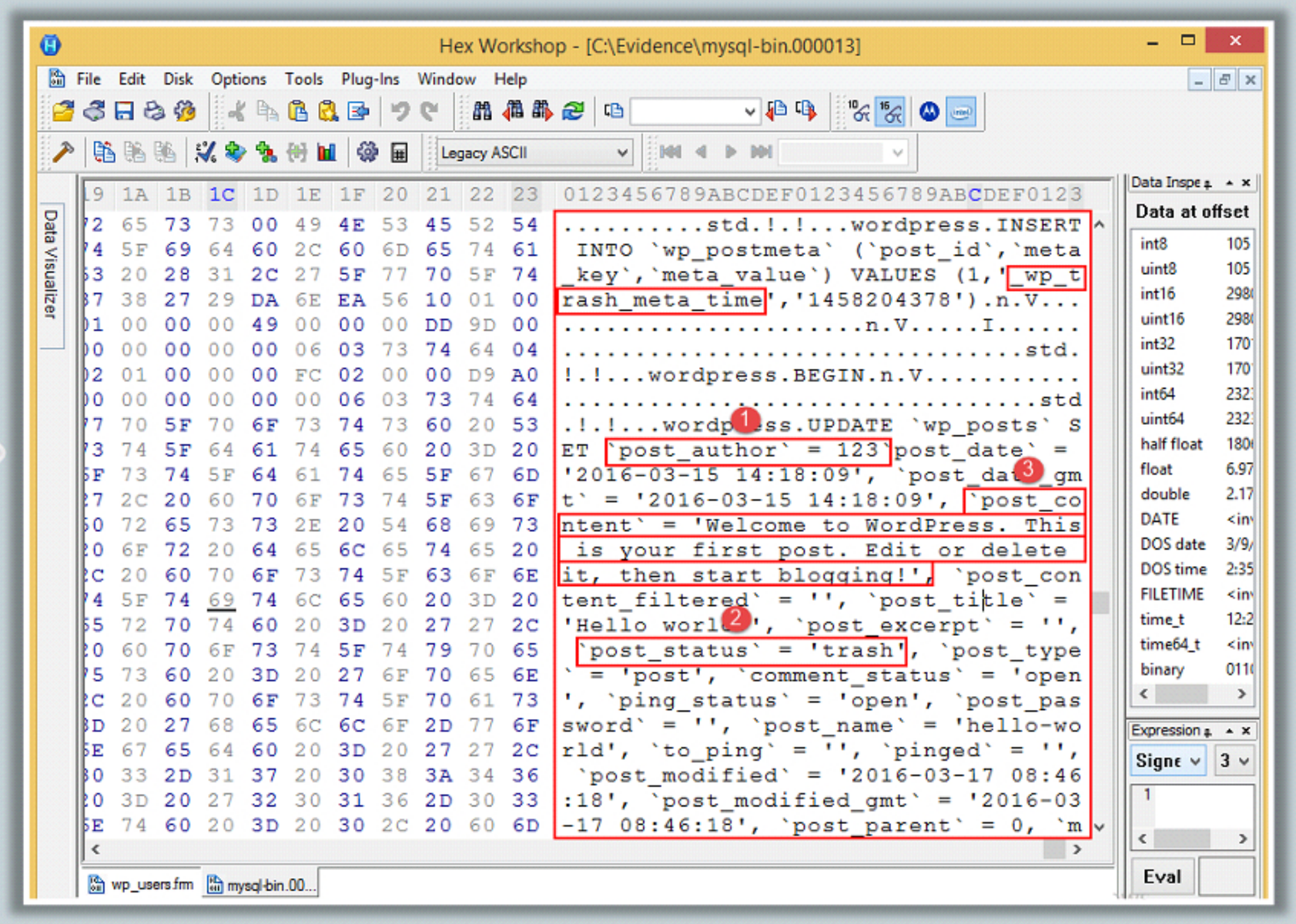
In the same way, you may search for all the actions performed by the attacker on the posts, by looking for **`post_author` = 123** in the hex editor.



Scenario 2: Examine the Binary Logs (Cont'd)

Data Recovery:

By looking for the query `post_author` = 123 in all the bin logs, you may come across various posts that were added, deleted and updated by the attacker.



Scenario 2: Examine the Binary Logs (Cont'd)

From the screenshot in the previous slide, it is observed that a post has been deleted (2) by the attacker whose ID is **123** (1). The post that was deleted (3) by him/her is

Post Title: "**Hello world**"

Post Content: "**Welcome to WordPress. This is your first post. Edit or delete it, then start blogging**"

Thus, the binary logs help in identifying the user transactions and recover the data that has been deleted.

Scenario 2: Examine the Binary Logs (Cont'd)

- Further scrolling down showed that a user account bearing the user ID **124** has been deleted

Hex Workshop - [C:\Evidence\mysql-bin.000029]

File Edit Disk Options Tools Plug-Ins Window Help

Legacy ASCII

	0	1	2	3	4	5	6	7	8	9	A	B	0123456789AB
000266AC	10	01	00	00	00	1B	00	00	00	C3	66	02	.....f.
000266B8	00	00	00	F1	02	01	00	00	00	00	00	56	V
000266C4	EF	F0	56	02	01	00	00	00	49	00	00	00	..V.....I...
000266D0	0C	67	02	00	08	00	02	10	00	00	00	00	.g.....
000266DC	00	00	09	00	00	1A	00	00	00	00	00	00	.....
000266E8	01	00	00	00	00	00	00	00	00	06	03	73	.....s
000266F4	74	64	04	21	00	21	00	08	00	77	6F	72	td.!.!...wor
00026700	64	70	72	65	73	73	00	42	45	47	49	4E	dpress.BEGIN
0002670C	56	EF	F0	56	02	01	00	00	00	67	00	00	V..V.....g..
00026718	00	73	67	02	00	00	00	02	10	00	00	00	.sg.....
00026724	00	00	00	09	00	00	1A	00	00	00	00	00	.....
00026730	00	01	00	00	00	00	00	00	00	00	06	03	.....
0002673C	73	74	64	04	21	00	21	00	08	00	77	6F	std.!.!...wo
00026748	72	64	70	72	65	73	73	00	44	45	4C	45	rdpress.DELE
00026754	54	45	20	46	52	4F	4D	20	77	70	5F	75	TE FROM wp_u
00026760	73	65	72	73	20	57	48	45	52	45	20	49	sers WHERE I
0002676C	44	20	3D	20	31	32	34	56	EF	F0	56	10	D = 124V..V.
00026778	01	00	00	00	1B	00	00	00	8E	67	02	00	.....g..

Data Inspector

Data at offset 0x00026746:

int8	119
uint8	119
int16	28535
uint16	28535
int32	1685221239
uint32	1685221239
int64	8315178114056...
uint64	8315178114056...
half float	7644.
float	1.7888576e+022
double	7.4977949e+247
DATE	<invalid>
DOS date	11/23/2035

Expression Calc

Signed 32 bit

Structures

Member Value (dec) Value (hex) Size

Compare Results All

Compare Checksum Find Bookmarks Output

Ready Cursor: 0002671F Caret: 00026746 Sel: 0000002D OVR MOD READ

Note: Since a database administrator in an organization stores regular backups of all the databases, analyzing these backups help in identifying the user who is associated with the ID **124**.

Scenario 2: Retrieve the Deleted User Account

1. Examine the **database backups** to view the all user accounts present in the database (before the malicious activity has occurred)

2. Examining an old backup revealed that the user ID **124** pertains to **Richard** user account; thus allowing you to successfully recover the deleted user account

Administrator: C:\Windows\system32\cmd.exe - mysql -u root -p

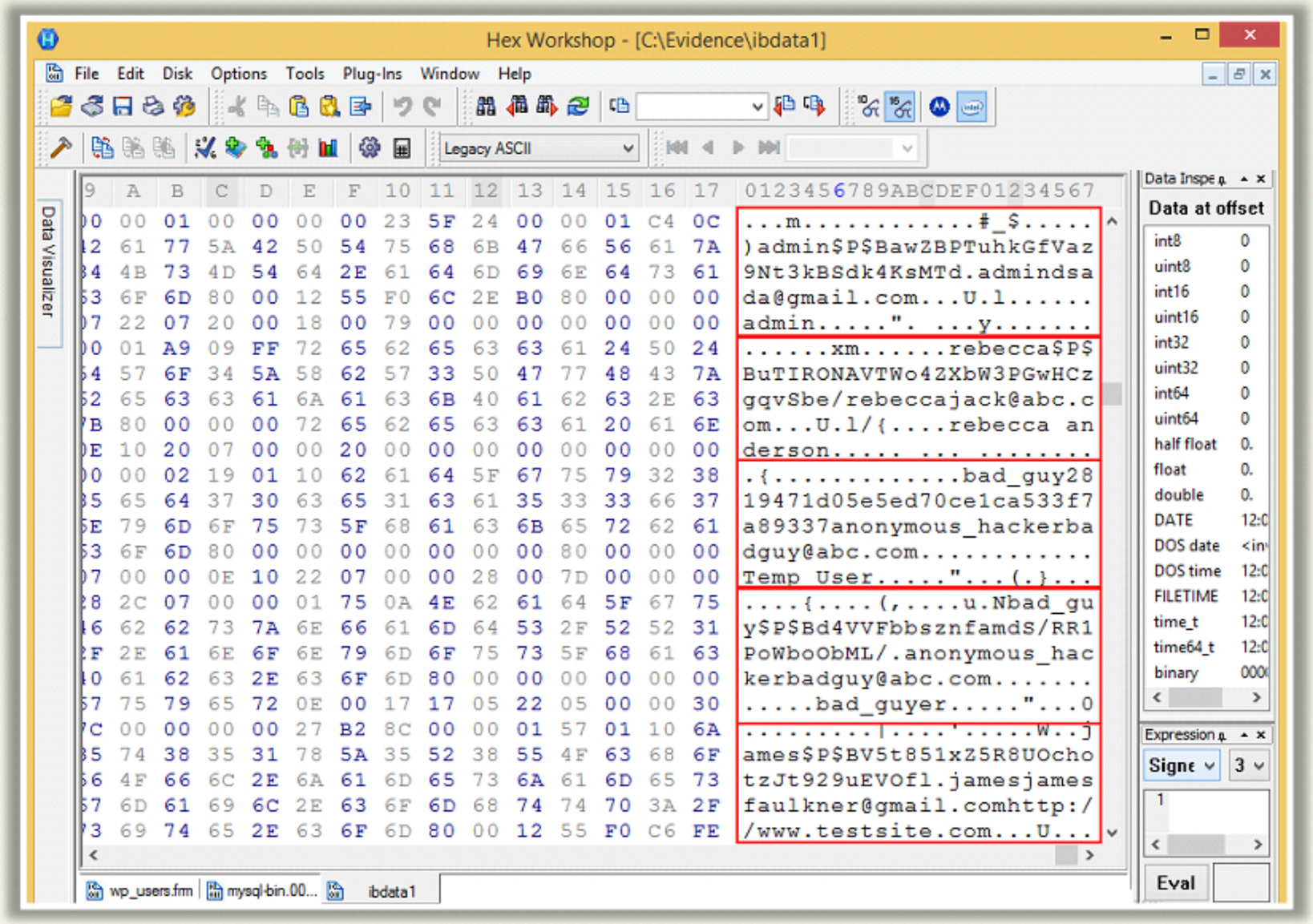
```
mysql> select * from wp_users;
+-----+------------+------------------------------------+---------------+---------------------+
| ID  | user_login | user_pass                          | user_nicename | user_email          |
+-----+------------+------------------------------------+---------------+---------------------+
|   1 | admin      | d6ab4783a1de646e487cc32c99b7936a   | admin         | admin@abc.com       |
|   2 | james      | ceb6c970658f31504a901b89dcd3e461   | james         | jamesfaulkner@gmail.com |
|   3 | rebecca    | ceb6c970658f31504a901b89dcd3e461   | rebecca       | rebecca@gmail.com   |
| 124 | richard    | 7ee83b7ba15622f55f6c45f29b52a7fe   | richard_speck | richard@abc.com     |
+-----+------------+------------------------------------+---------------+---------------------+
4 rows in set (0.00 sec)

mysql> _
```

Scenario 2: ibdata1 in Data Directory

Additional Information

The **ibdata1** file can be referred during a forensic investigation, as it stores the database data permanently, including the data that has been deleted.



Module Summary

- Database Forensics is the examination of the databases and related metadata in a forensically precise manner to make the findings presentable in the court of law
- MSSQL Server stores data and logs in Primary Data Files (MDF), Secondary Data Files (NDF) and Transaction Log Data Files (LDF), respectively
- SQL server data is stored natively within SQL Server, and externally within windows machine hosting the server
- MySQL is based on a tiered architecture containing subsystems and support components, which work together in order to respond to the queries made to the database server
- MySQL server stores all the databases, status and log files; along with the data managed by the server under the data directory
- The database structure varies depending on the storage engine (MyISAM/InnoDB) used by MySQL