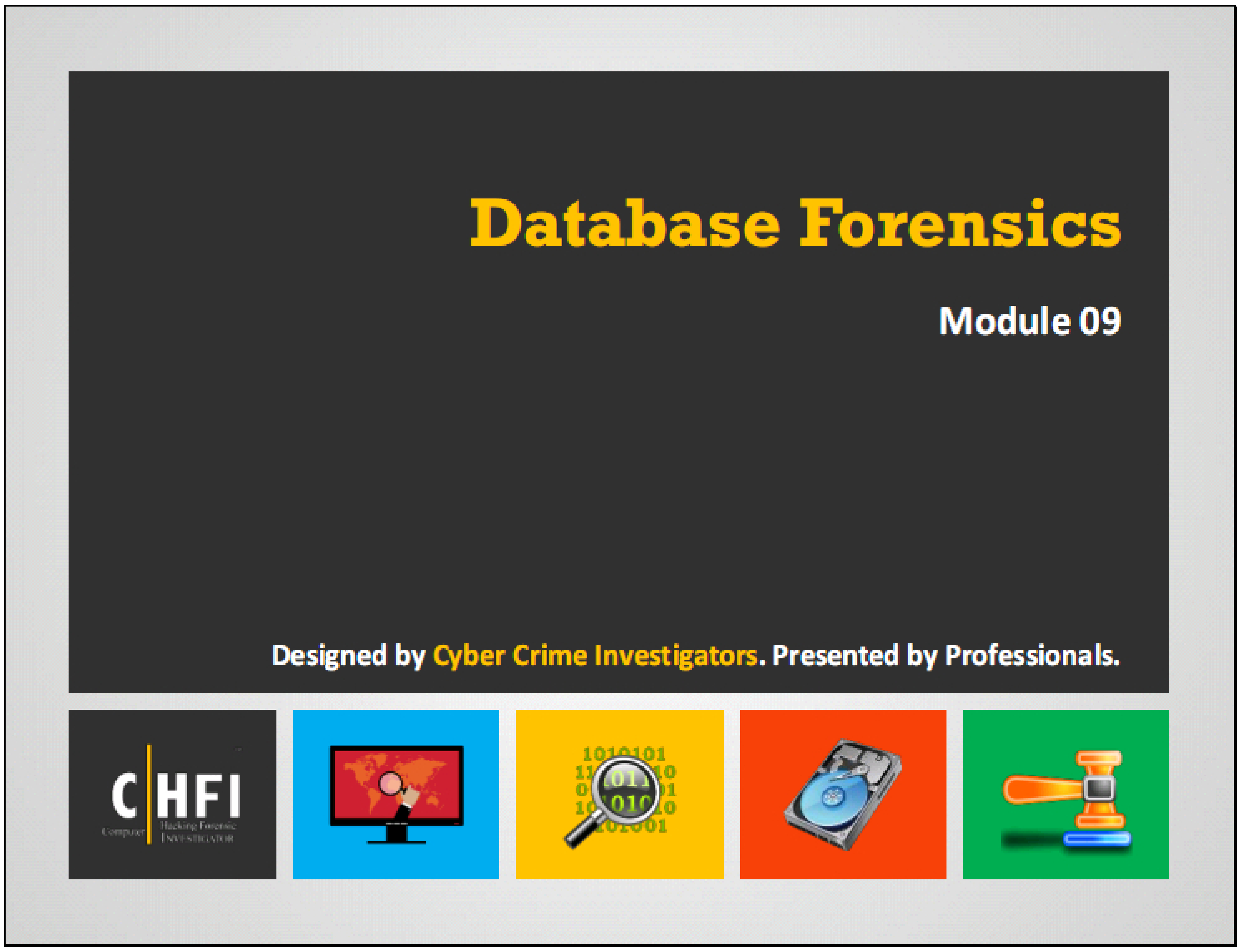
Database Forensics

Module 09



Database Forensics

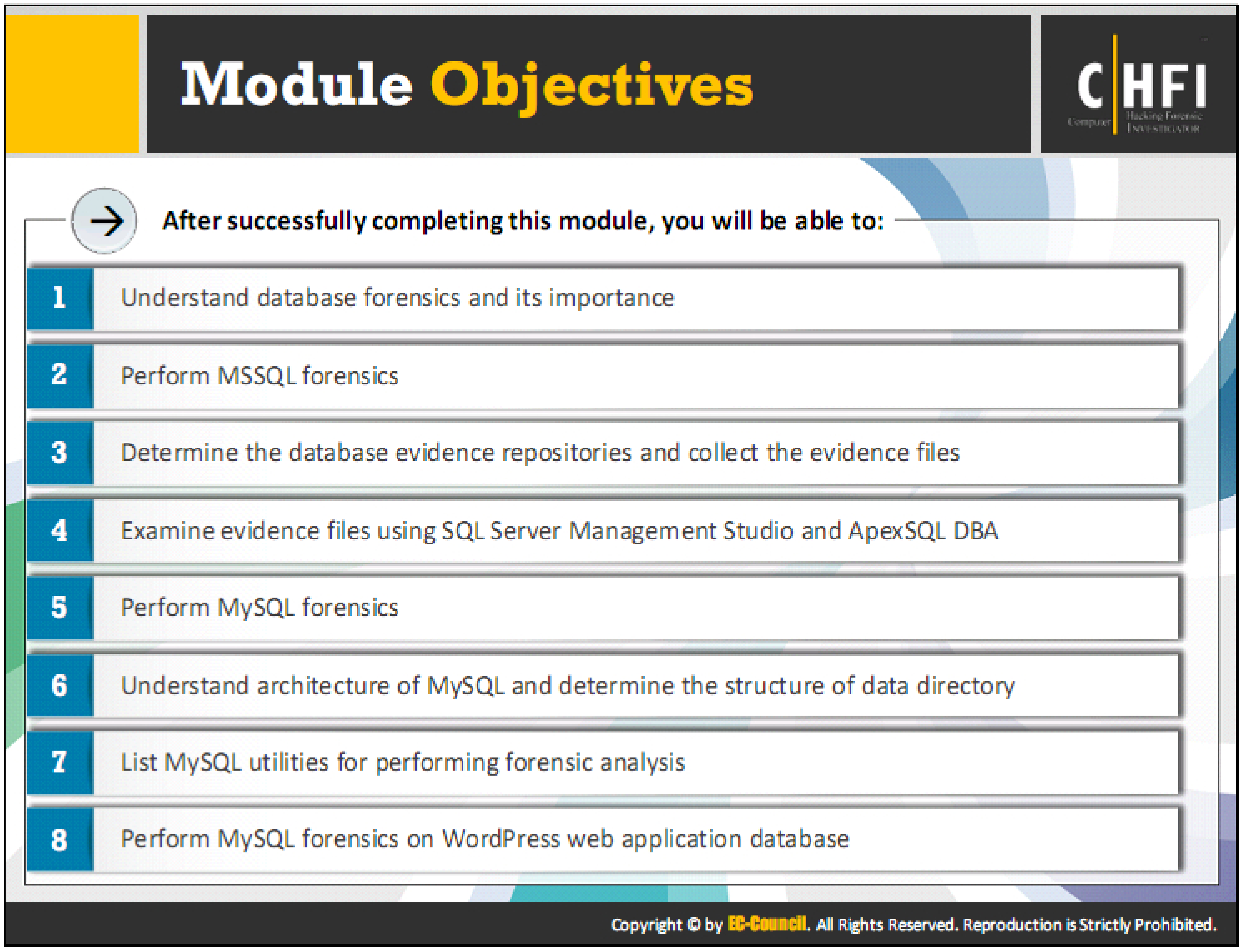
Module 09

Designed by Cyber Crime Investigators. Presented by Professionals.

Computer Hacking Forensic Investigator v9

Module 09: Database Forensics

Exam 312-49

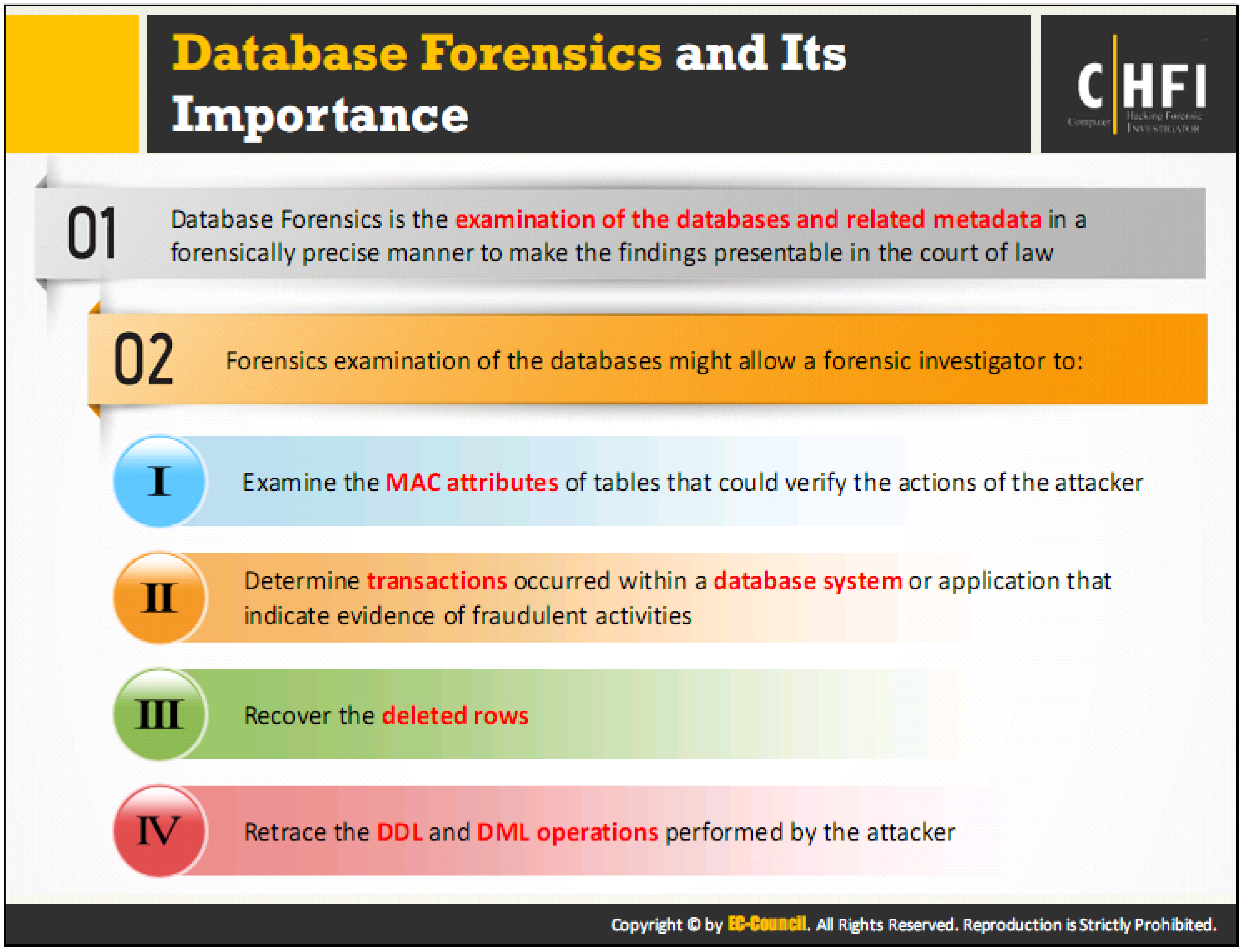


Module Objectives

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

Databases store the entire data pertaining to a web application and allow users to view, access, manage, and update the information. In some cases, either the databases or the web applications may contain vulnerabilities that allow attackers to manipulate the contents of the database. Therefore, a forensic investigator must have sound knowledge of the database servers, and their file systems. Additionally, the investigator should be able to examine their respective log files and find the cause of the attacks. This module discusses the file systems of MSSQL and MySQL servers. Furthermore, it explains the usage of various tools to examine the log files and find the fraudulent transactions.



Database Forensics and Its Importance

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

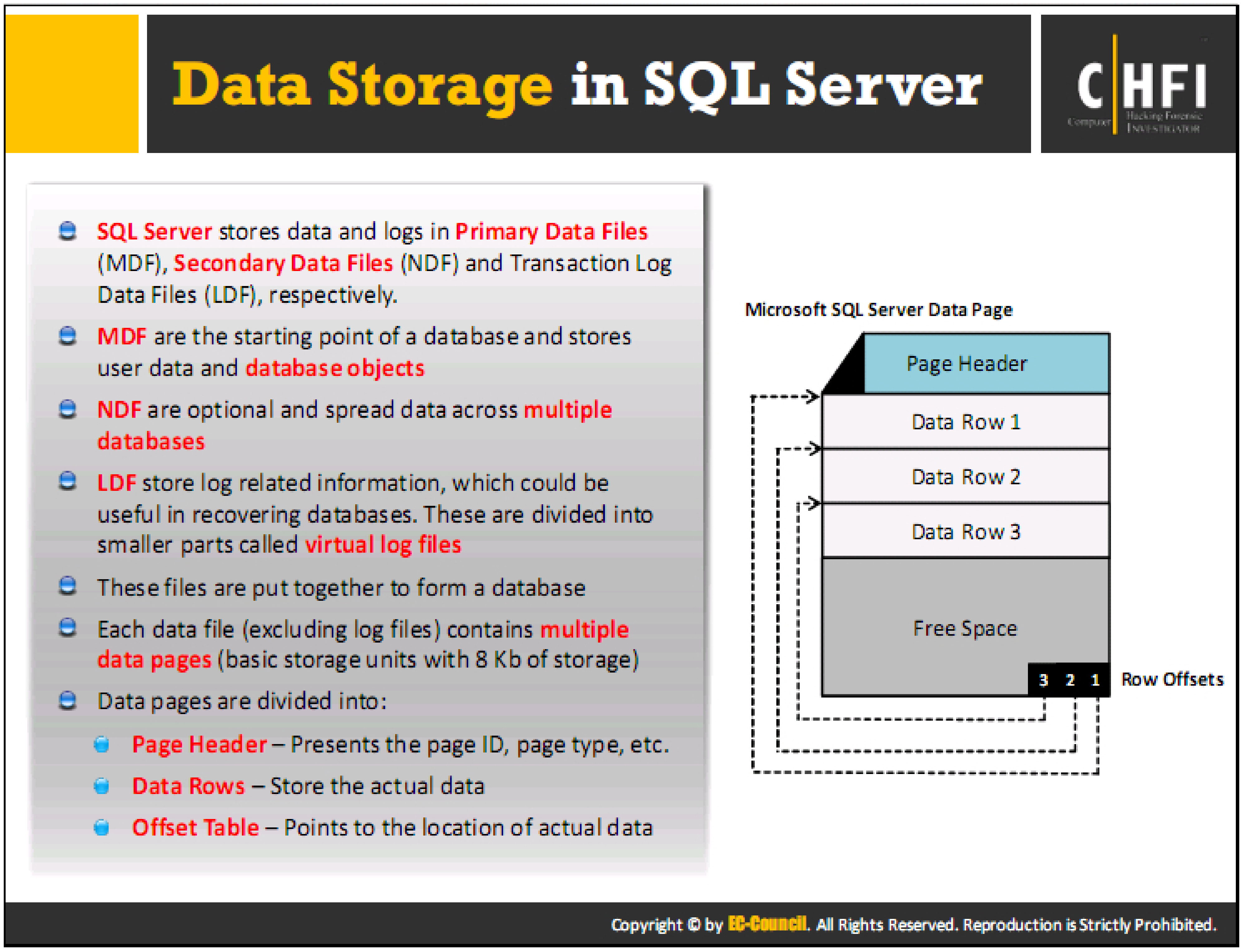
Currently, the majority of the applications use high-performance databases to manage the data. While, the organizations are implementing robust security mechanisms to protect the databases, hackers are introducing sophisticated ways to attack them, resulting in sensitive data exposure.

Database forensics deals with the examination of databases and its associated metadata. The process involved in database forensics is similar to the ones followed in computer forensics.

Databases act as the primary source of electronic evidence for every organization irrespective of its size and complexity. On the occurrence of an unexpected incident, a forensic examiner produces this evidence in the court of law, regardless the size of the databases. As a part of an investigation, the investigator may examine the time stamps to check and validate the activities carried out by the user/users on the database contents. They can also analyze the transactions in the Transaction Log Data Files (.ldf) to see if any user/users performed fraudulent activities on the database. A server hosting databases may contain cached information in its RAM. Forensic investigators may even examine this information using live analysis techniques.

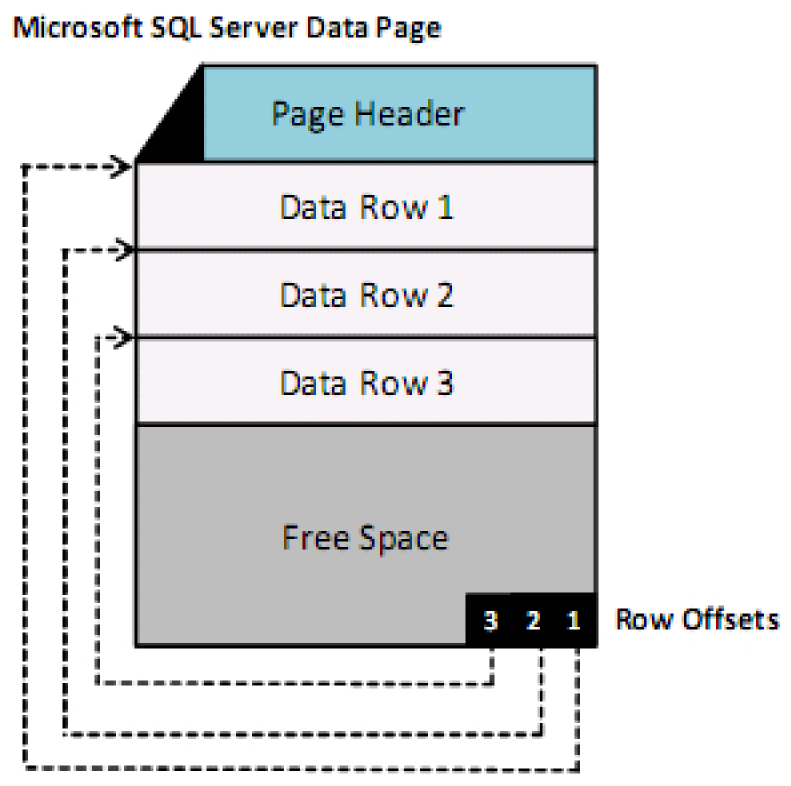
MSSQL Forensics

SQL server is a Relational Database Management System and is being widely adopted by various organizations to store data associated with the applications. This includes sensitive data related to the web application and users' accounts in the web application. MSSQL forensics take action when a security incident has occurred and detection and analysis of the malicious activities performed by criminals over the SQL database file are required. A forensic investigator needs to examine the Primary Database Files and Transaction Log Files for investigation purpose.



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



Data and Logs in SQL servers are stored in three different files:

- **Primary Data Files (MDF)**

  The primary data file is the starting point of a database and points to other files in the database. Every database has a primary data file. The primary data file stores all the data in the database objects (tables, schema, indexes, etc.). The file name extension for primary data files is .mdf.

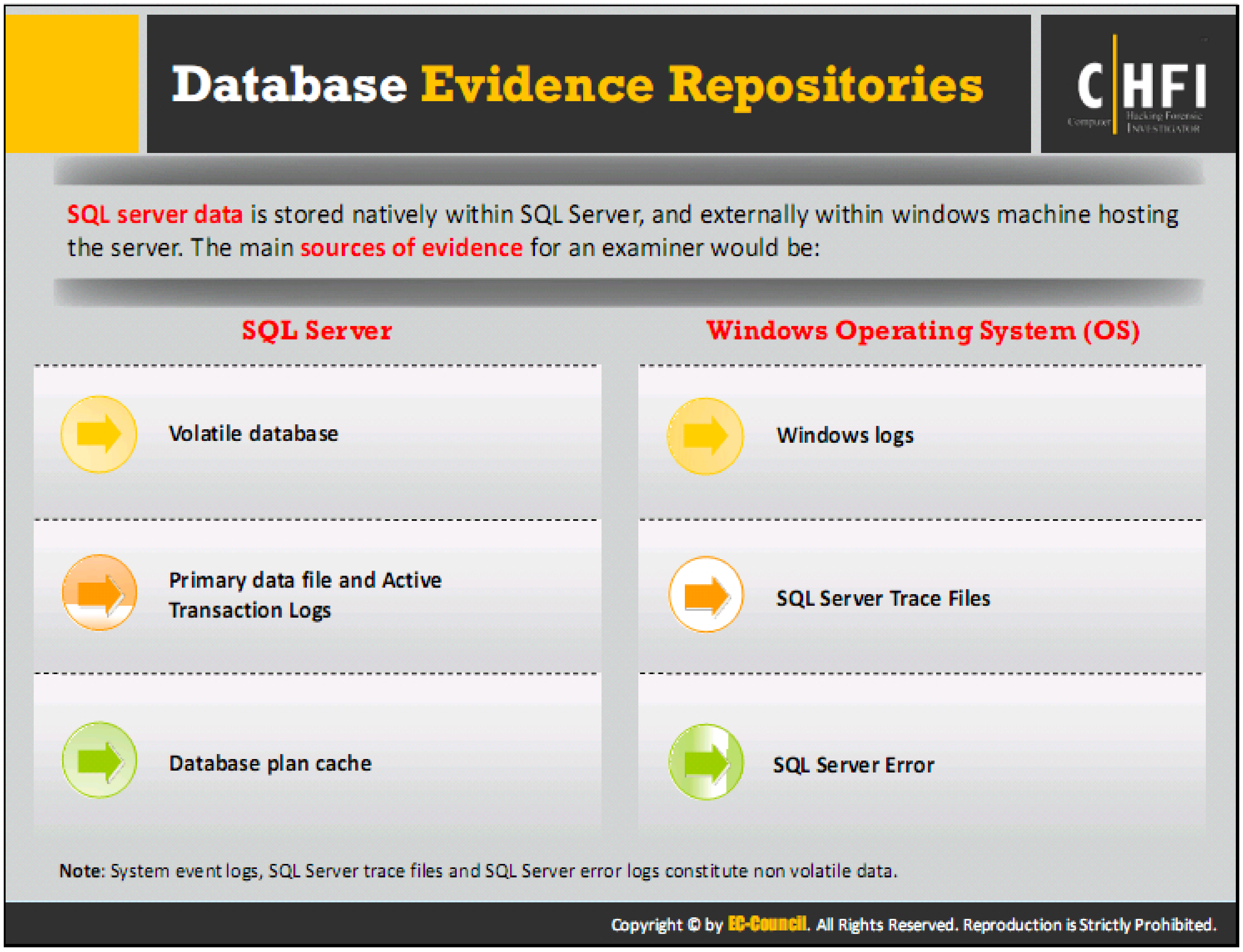
- **Secondary Data Files (NDF)**

  The secondary data files are optional. While a database contains only one primary data file, it can contain zero/single/multiple secondary data files. The Secondary data file can be stored on a hard disk, separate than the primary data file. The file name extension for secondary data files is .ndf.

- **Transaction LOG Data Files (LDF)**

  The transaction log files hold the entire log information associated with the database. The transaction log file helps a forensic investigator to examine the transactions occurred on a database, and even recover data deleted from the database. The file name extension for transaction log date files is .ldf and each file is divided into virtual log files.

These three files together constitute a database, and each data file contains multiple data pages, as discussed above.



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	Windows Operating System (OS)
Volatile database	Windows logs
Primary data file and Active Transaction Logs	SQL Server Trace Files
Database plan cache	SQL Server Error

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

Sources that provide the valuable information are at times overlooked by the investigators. For instance, in intellectual property cases, databases containing finance related data are the prime targets for attackers to damage databases. In such case, source code repositories, knowledge management systems, and document management systems may provide better insights to the investigator to a suspected breach. Thus, investigators will be able to help the defendants against invalid obligations.

The databases can be used for versioning and reviewing the document lifecycle. The extended metadata, like descriptions, keywords and comments may provide insights into the document's purpose. It discloses the information like who accessed and exposed the information and, where and when it was routed.

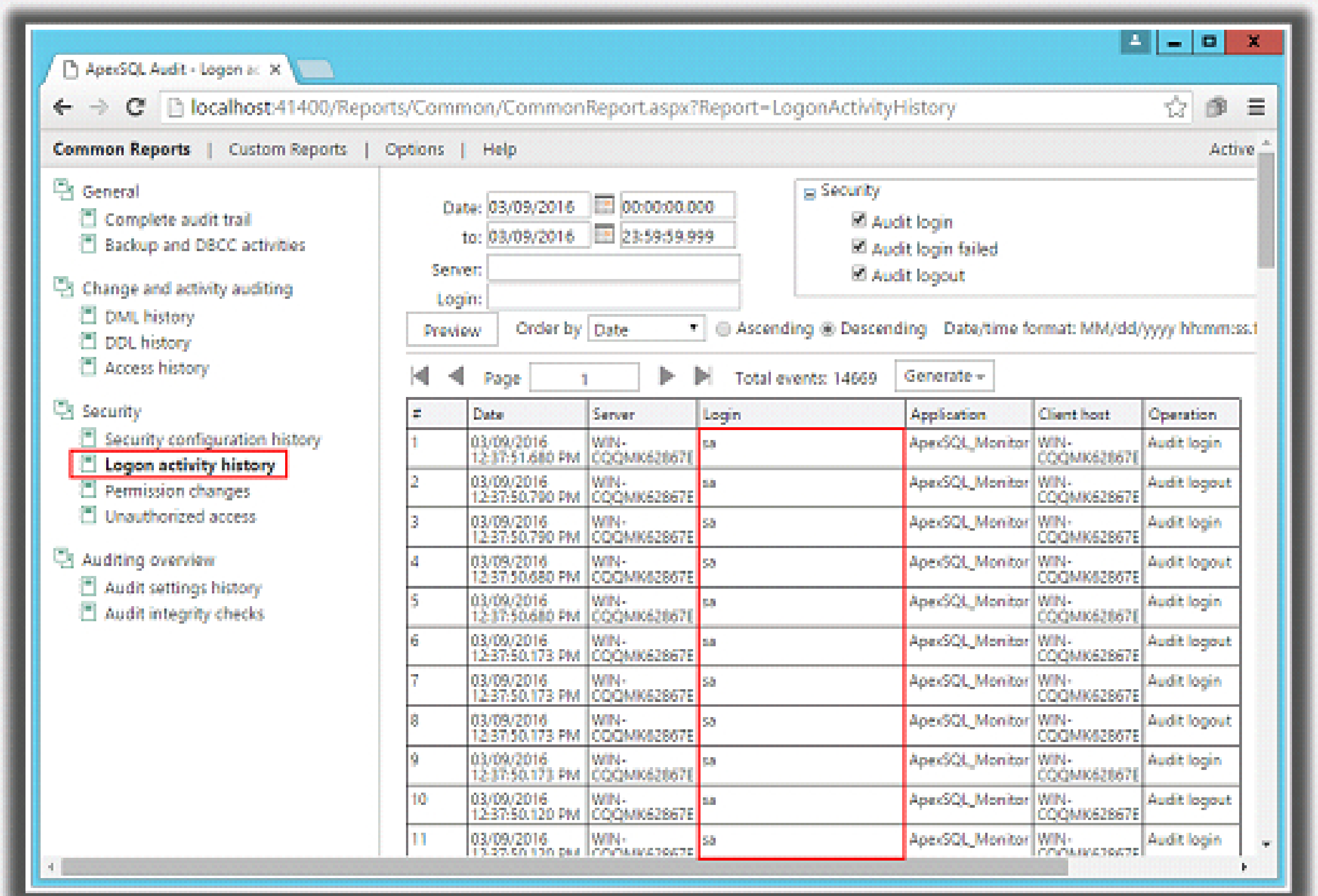
Location of Files to Restore the Evidence

Along with the Volatile database data, Windows logs and Database plan cache, investigators can examine the following files to have an insight of the activities occurred on the database:

- **Database & logs files:** \\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\*.MDF | *.LDF
- **Trace files:** \\Microsoft SQL Server\MSSQL11.MSSQLSERVER \MSSQL\ LOG\LOG_#.TRC
- **SQL Server error logs:** \\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\LOG\ERRORLOG

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

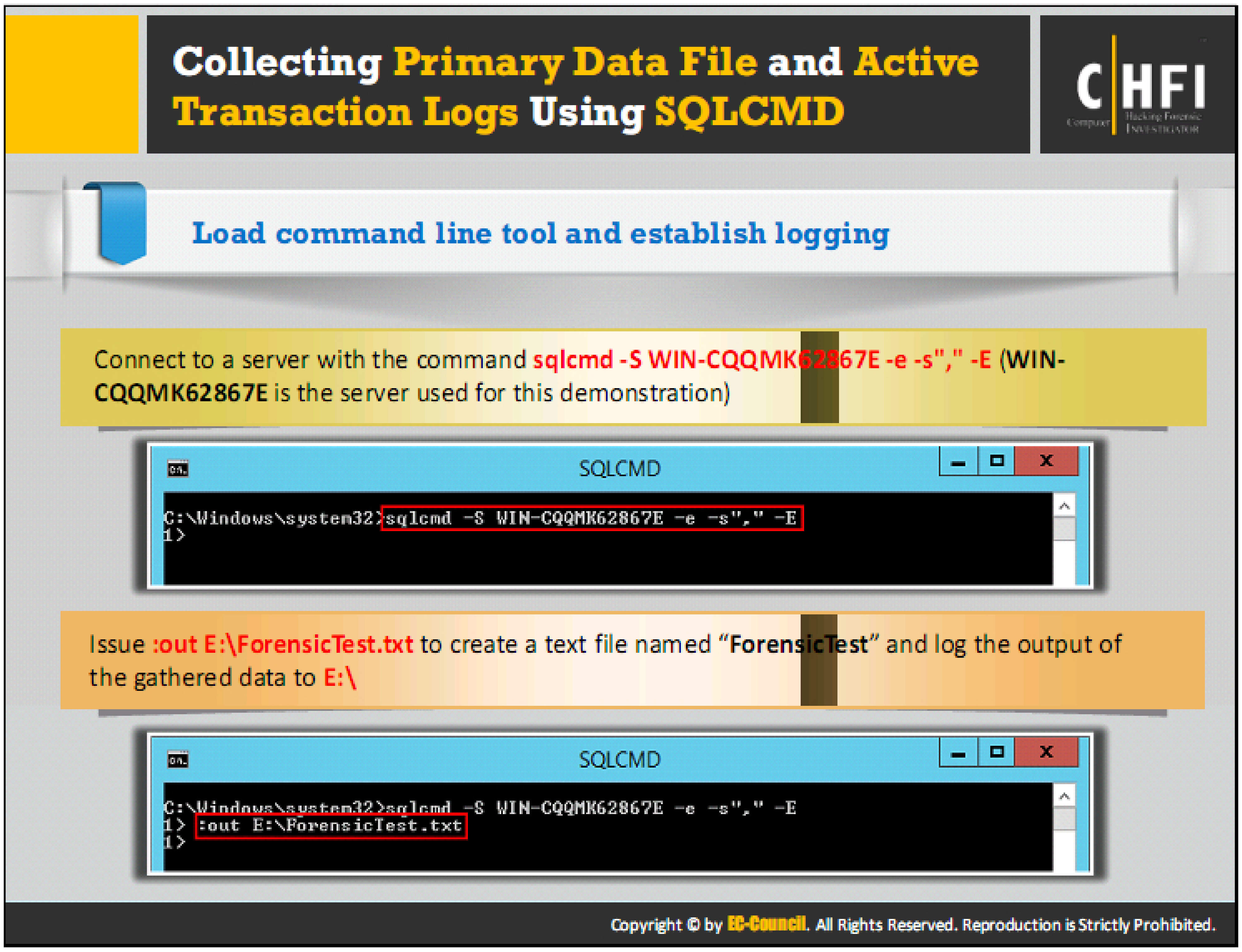


Date: 03/09/2016 00:00:00.000
to: 03/09/2016 23:59:59.999
Server:
Login:
Security: ☑ Audit login ☑ Audit login failed ☑ Audit logout
Order by: Date ○ Ascending ● Descending Date/time format: MM/dd/yyyy hh:mm:ss.f
Page 1 Total events: 14669

#	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	WIN-	sa	ApexSQL_Monitor	WIN-	Audit login

Volatile Database is a RAM-style memory, which usually loses all its contents on power cuts. Investigators can track the volatile database information like login sessions of an account and the transactions using ApexSQL DBA's ApexSQL Audit application.

By clicking on "Logon Activity history" in ApexSQL Audit application, the investigator can view the login history for a given date and time, as shown above.



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)

SQLCMD
```
C:\Windows\system32>sqlcmd -S WIN-CQQMK62867E -e -s"," -E
1>
```

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

SQLCMD
```
C:\Windows\system32>sqlcmd -S WIN-CQQMK62867E -e -s"," -E
1> :out E:\ForensicTest.txt
1>
```

The primary data file (mdf) and active transaction logs (ldf) play a key role in the forensic investigation. These files offer sufficient information to a forensic examiner for dealing with the investigation. A forensic examiner needs to know the location of mdf and ldf associated with a database, before proceeding with the investigation. The SQLCMD application helps an investigator to obtain the location of these files.

The SQLCMD application lets investigators load and establish a connection with the server.

To initialize connection with the server (WIN-CQQMK62867E), the following command is used in the application

```
sqlcmd -S WIN-CQQMK62867E -e -s"," -E
```

-e is used to echo input

-s is used for column separation

-E is used for trusted connection

The above command infers that we want to establish a trusted connection with the server **WIN-CQQMK62867E** and output the results of the forthcoming commands with the columns in the output separated by commas (,).

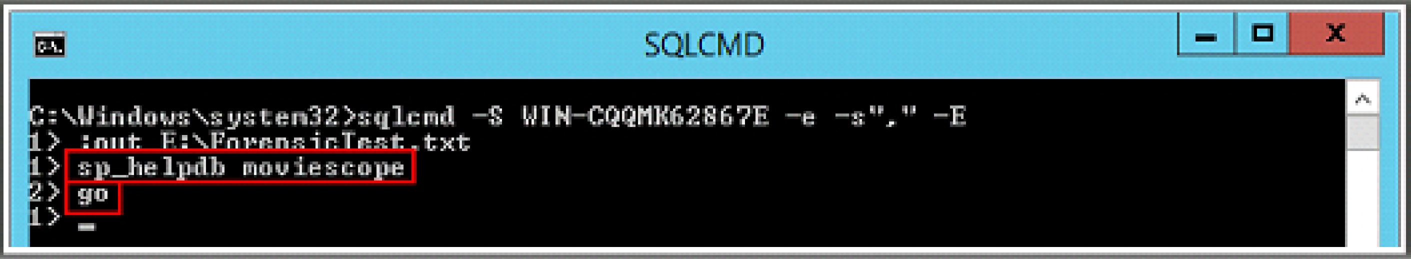
The following is to be issued in SQLCMD to create a new text file with name ForensicTest and save the output to E drive:

```
:out E:\ForensicTest.txt
```

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

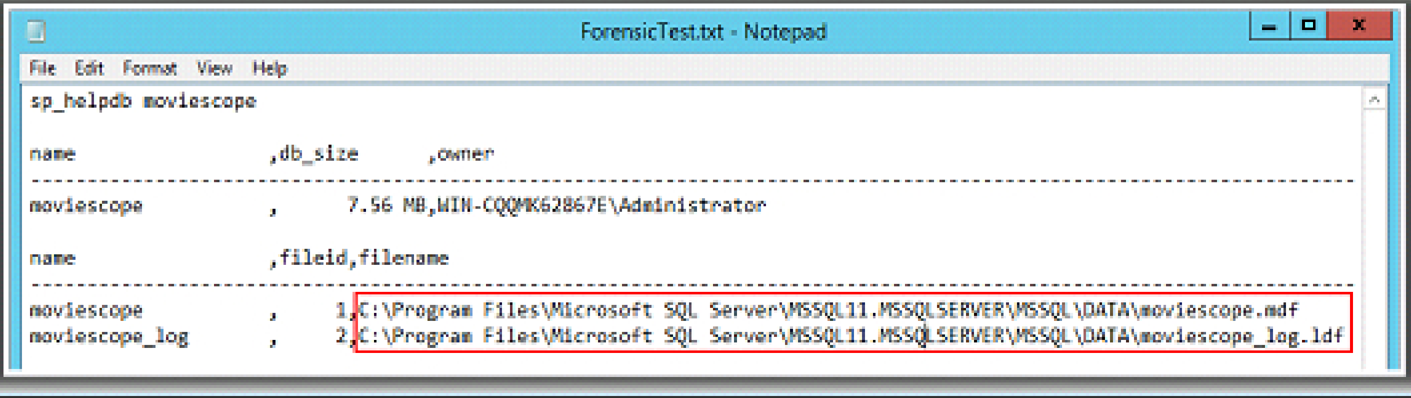
Collect the active transaction log

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



```
C:\Windows\system32>sqlcmd -S WIN-CQQMK62867E -e -s"," -E
1> :out E:\ForensicTest.txt
1> sp_helpdb moviescope
2> go
1> _
```

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



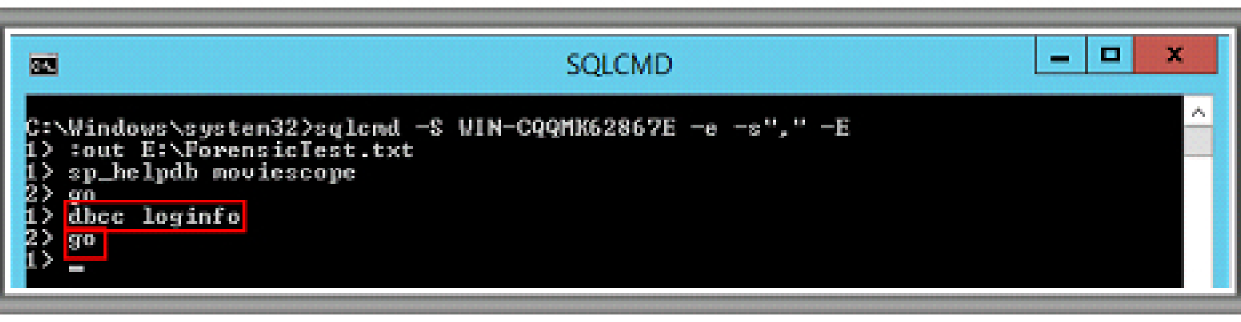
```
sp_helpdb moviescope

name                 ,db_size      ,owner
-------------------------------------------------------------------------------------------------------
moviescope           ,      7.56 MB,WIN-CQQMK62867E\Administrator

name                 ,fileid,filename
-------------------------------------------------------------------------------------------------------
moviescope           ,     1,C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\moviescope.mdf
moviescope_log       ,     2,C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\moviescope_log.ldf
```

The `sp_helpdb` command outputs the information related to the specified database. A forensic investigator can use this command to determine the location of the primary data file and transaction log file that is associated with a database.

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

Collect the active transaction log (Cont'd)

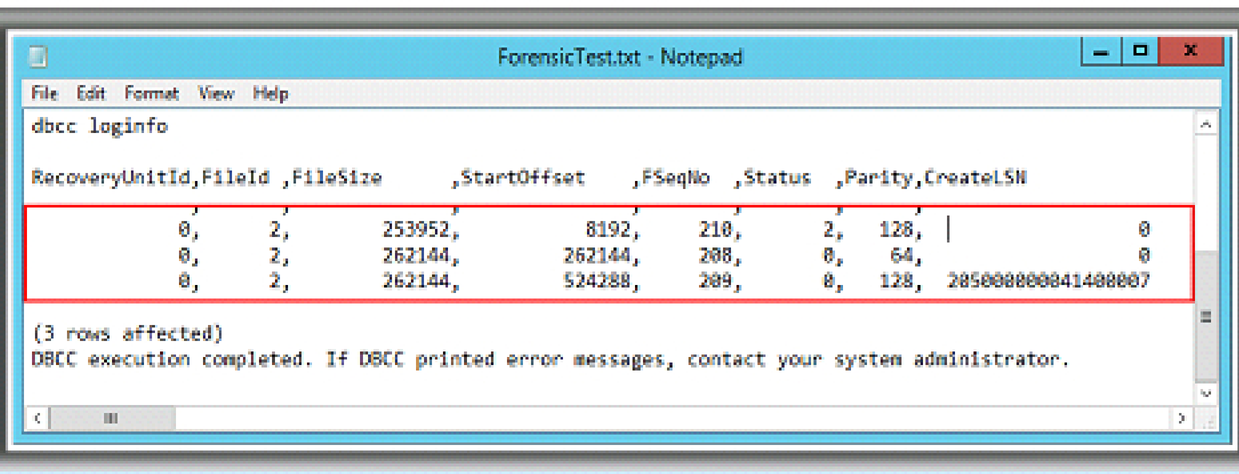
Issue the commands dbcc loginfo and go to gather the VLF allocations for the **moviescope** database



```
C:\Windows\system32>sqlcmd -S WIN-CQQMK62867E -e -s"," -E
1> :out E:\ForensicTest.txt
1> sp_helpdb moviescope
2> go
1> dbcc loginfo
2> go
1> _
```

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



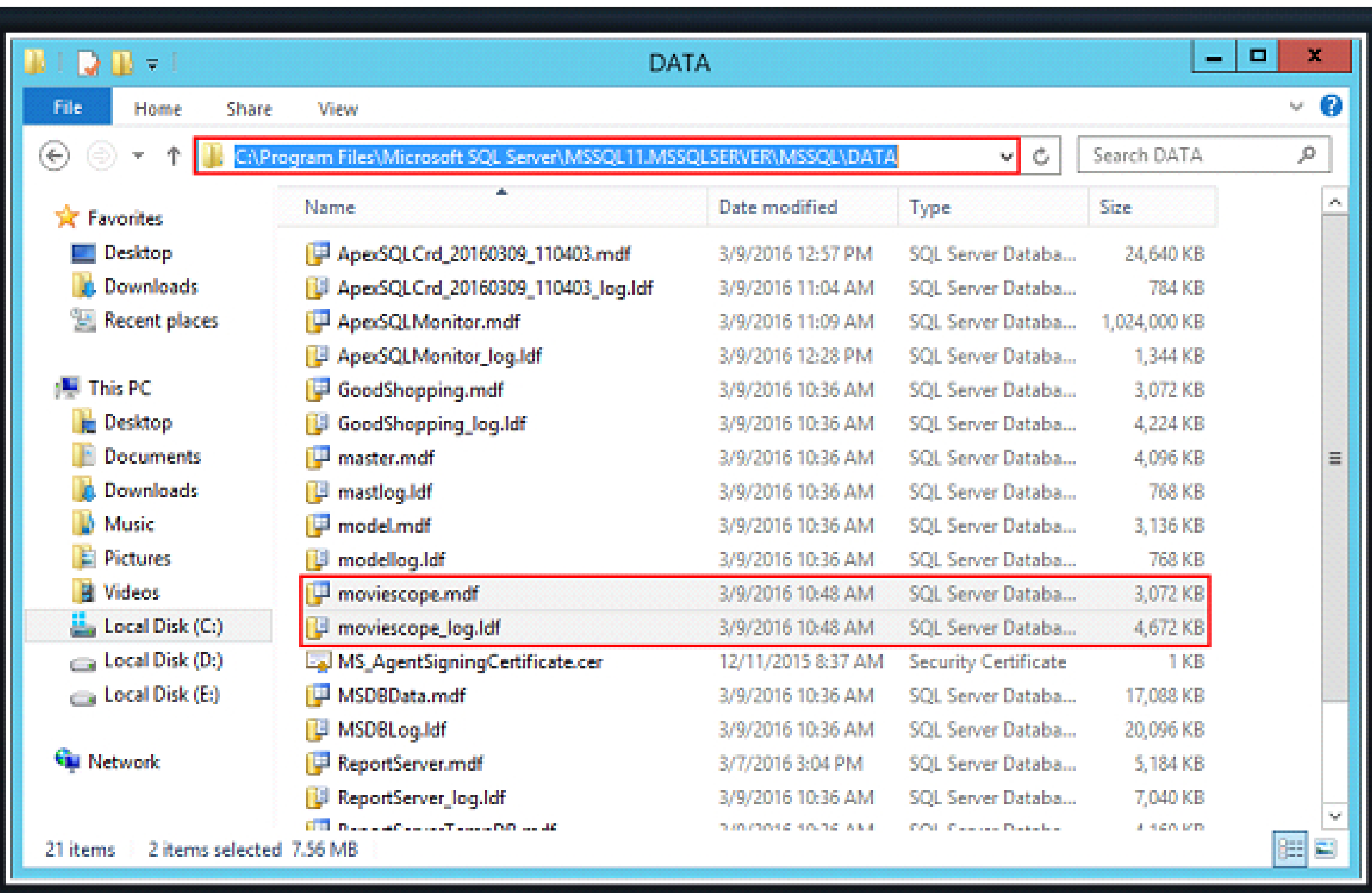
```
dbcc loginfo

RecoveryUnitId,FileId ,FileSize      ,StartOffset    ,FSeqNo  ,Status  ,Parity,CreateLSN
             0,      2,       253952,           8192,     210,       2,   128,                   0
             0,      2,       262144,         262144,     208,       0,    64,                   0
             0,      2,       262144,         524288,     209,       0,   128,  20500000041400007

(3 rows affected)
DBCC execution completed. If DBCC printed error messages, contact your system administrator.
```

Transaction Log Files store log related information, which could be useful in recovering databases. It is divided into smaller parts called virtual log files.

The moviescope database files are stored in the VLF allocations. These allocations can be traced using the following commands in SQLCMD application.

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



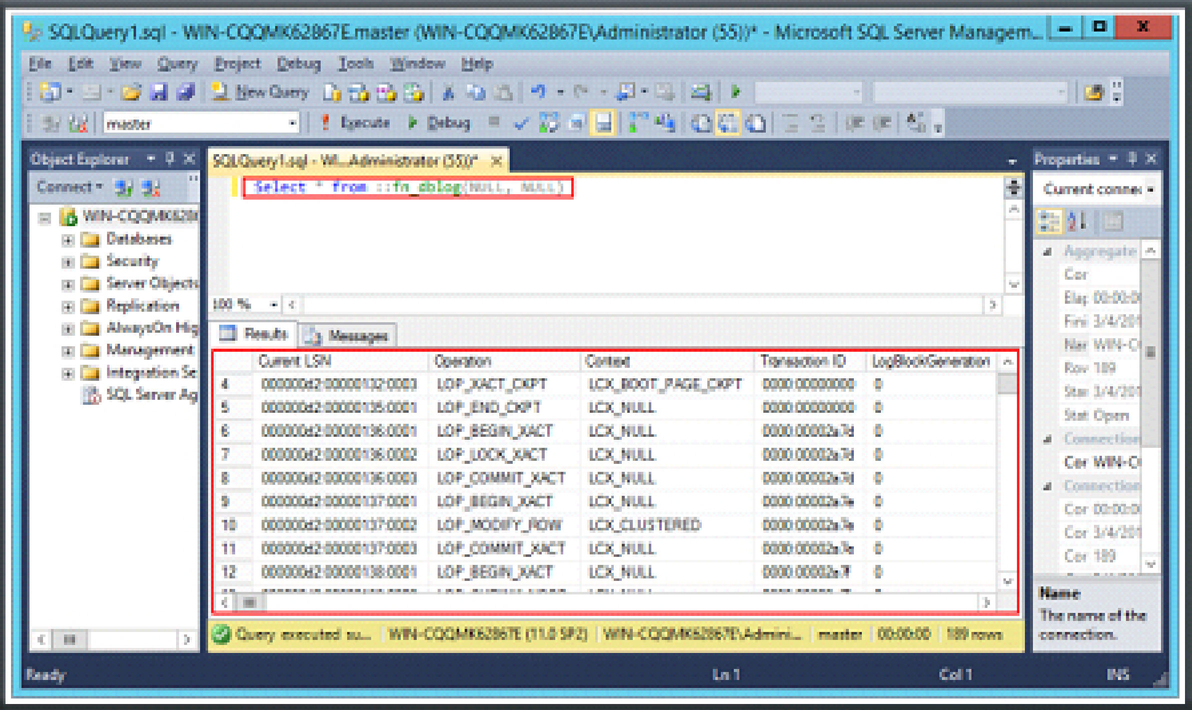
C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\DATA

Name	Date modified	Type	Size
ApexSQLCrd_20160309_110403.mdf	3/9/2016 12:57 PM	SQL Server Databa...	24,640 KB
ApexSQLCrd_20160309_110403_log.ldf	3/9/2016 11:04 AM	SQL Server Databa...	784 KB
ApexSQLMonitor.mdf	3/9/2016 11:09 AM	SQL Server Databa...	1,024,000 KB
ApexSQLMonitor_log.ldf	3/9/2016 12:28 PM	SQL Server Databa...	1,344 KB
GoodShopping.mdf	3/9/2016 10:36 AM	SQL Server Databa...	3,072 KB
GoodShopping_log.ldf	3/9/2016 10:36 AM	SQL Server Databa...	4,224 KB
master.mdf	3/9/2016 10:36 AM	SQL Server Databa...	4,096 KB
mastlog.ldf	3/9/2016 10:36 AM	SQL Server Databa...	768 KB
model.mdf	3/9/2016 10:36 AM	SQL Server Databa...	3,136 KB
modellog.ldf	3/9/2016 10:36 AM	SQL Server Databa...	768 KB
moviescope.mdf	3/9/2016 10:48 AM	SQL Server Databa...	3,072 KB
moviescope_log.ldf	3/9/2016 10:48 AM	SQL Server Databa...	4,672 KB
MS_AgentSigningCertificate.cer	12/11/2015 8:37 AM	Security Certificate	1 KB
MSDBData.mdf	3/9/2016 10:36 AM	SQL Server Databa...	17,088 KB
MSDBLog.ldf	3/9/2016 10:36 AM	SQL Server Databa...	20,096 KB
ReportServer.mdf	3/7/2016 3:04 PM	SQL Server Databa...	5,184 KB
ReportServer_log.ldf	3/9/2016 10:36 AM	SQL Server Databa...	7,040 KB

21 items 2 items selected 7.56 MB

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


SQLQuery1.sql - WIN-CQQMK62867E.master (WIN-CQQMK62867E\Administrator (55))* - Microsoft SQL Server Managem...
Object Explorer
Results
Messages
Properties

```
Select * from ::fn_dblog(NULL, NULL)
```

	Current LSN	Operation	Context	Transaction ID	LogBlockGeneration
4	000000d2:00000132:0003	LOP_XACT_CKPT	LCX_BOOT_PAGE_CKPT	0000:00000000	0
5	000000d2:00000135:0001	LOP_END_CKPT	LCX_NULL	0000:00000000	0
6	000000d2:00000136:0001	LOP_BEGIN_XACT	LCX_NULL	0000:00002a7d	0
7	000000d2:00000136:0002	LOP_LOCK_XACT	LCX_NULL	0000:00002a7d	0
8	000000d2:00000136:0003	LOP_COMMIT_XACT	LCX_NULL	0000:00002a7d	0
9	000000d2:00000137:0001	LOP_BEGIN_XACT	LCX_NULL	0000:00002a7e	0
10	000000d2:00000137:0002	LOP_MODIFY_ROW	LCX_CLUSTERED	0000:00002a7e	0
11	000000d2:00000137:0003	LOP_COMMIT_XACT	LCX_NULL	0000:00002a7e	0
12	000000d2:00000138:0001	LOP_BEGIN_XACT	LCX_NULL	0000:00002a7f	0

```
Query executed su... | WIN-CQQMK62867E (11.0 SP2) | WIN-CQQMK62867E\Admini... | master | 00:00:00 | 189 rows
```

As we know that the transaction logs store all the DML operations, along with some of the DDL operations, a forensic investigator can examine these transaction logs to see the transactions performed on the databases. However, since the logs are not in human readable format, it will be difficult for anyone without the knowledge of SQL to examine the log records.

Forensic investigators use undocumented functions like fn_dblog () and fn_dump_dblog () to view the transaction logs.

The function fn_dblog() accepts two parameters

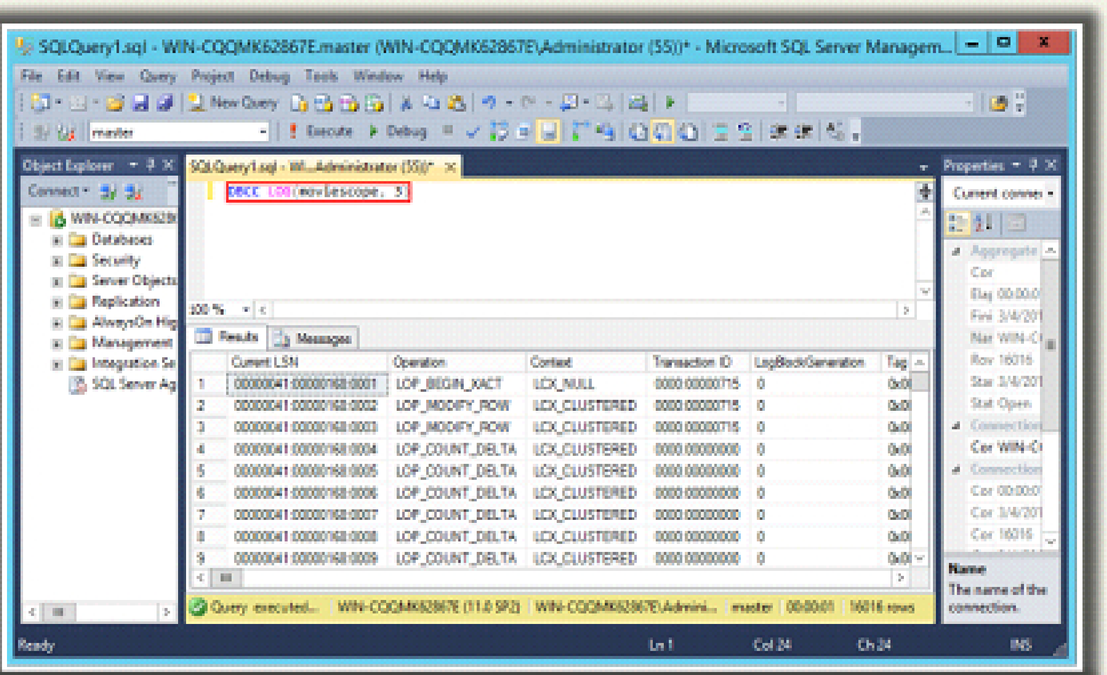
- The starting Log Sequence Number(LSN) or NULL(returs everything from the start of the log)
- The ending Log Sequence Number(LSN) or NULL(returs everything to the end of the log)

Note: This function should not run against an active database instance.

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.



```
DBCC LOG(moviescope, 3)
```

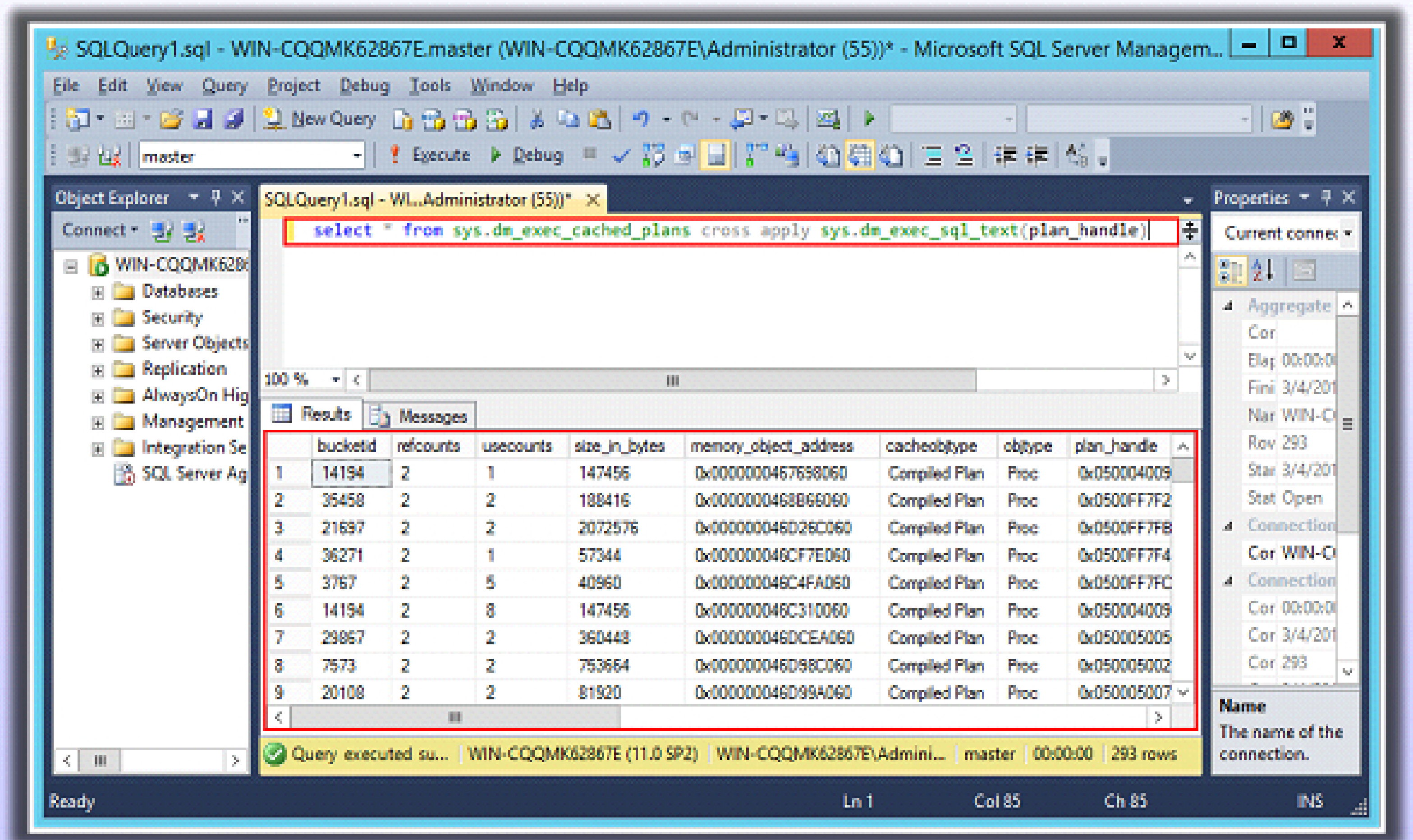
```
   Current LSN                Operation          Context         Transaction ID   LogBlockGeneration  Tag
1  00000041:00000168:0001     LOP_BEGIN_XACT     LCX_NULL        0000:00000715    0                   0x0
2  00000041:00000168:0002     LOP_MODIFY_ROW     LCX_CLUSTERED   0000:00000715    0                   0x0
3  00000041:00000168:0003     LOP_MODIFY_ROW     LCX_CLUSTERED   0000:00000715    0                   0x0
4  00000041:00000168:0004     LOP_COUNT_DELTA    LCX_CLUSTERED   0000:00000000    0                   0x0
5  00000041:00000168:0005     LOP_COUNT_DELTA    LCX_CLUSTERED   0000:00000000    0                   0x0
6  00000041:00000168:0006     LOP_COUNT_DELTA    LCX_CLUSTERED   0000:00000000    0                   0x0
7  00000041:00000168:0007     LOP_COUNT_DELTA    LCX_CLUSTERED   0000:00000000    0                   0x0
8  00000041:00000168:0008     LOP_COUNT_DELTA    LCX_CLUSTERED   0000:00000000    0                   0x0
9  00000041:00000168:0009     LOP_COUNT_DELTA    LCX_CLUSTERED   0000:00000000    0                   0x0

Query executed...   WIN-CQQMK62867E (11.0 SP2)   WIN-CQQMK62867E\Admini...   master   00:00:01   16016 rows
```

Database Consistency Checker (DBCC) commands may give the investigator valuable insight into what is happening within the Server system. The DBCC LOG command allows investigators to view and retrieve the active transaction log files for a specific database. Following are the other DBCC commands that allow the investigator to obtain additional information related to the specified database.

- **DBCC DBTABLE**: Returns the structure of the selected database table
- **DBCC DBINFO**: Returns information related to the database metadata
- **DBCC PROCBUF**: Returns the contents of the SQL Server Procedure Buffer. The buffer contains SQL Server cached executable statements such as stored procedures and SQL queries.
- **DBCC BUFFER**: Returns the buffer headers and pages from SQL Server's buffer cache, where SQL Server stores results.
- **DBCC SHOWFILESTATS**: Returns information related to the space occupied by the data files in the active database.
- **DBCC PAGE**: Returns the data page structure of the selected database

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



```
select * from sys.dm_exec_cached_plans cross apply sys.dm_exec_sql_text(plan_handle)
```

	bucketid	refcounts	usecounts	size_in_bytes	memory_object_address	cacheobjtype	objtype	plan_handle
1	14194	2	1	147456	0x0000000467698060	Compiled Plan	Proc	0x050004009
2	35458	2	2	188416	0x000000046BB66060	Compiled Plan	Proc	0x0500FF7F2
3	21697	2	2	2072576	0x000000046D26C060	Compiled Plan	Proc	0x0500FF7FB
4	36271	2	1	57344	0x000000046CF7E060	Compiled Plan	Proc	0x0500FF7F4
5	3767	2	5	40960	0x000000046C4FA060	Compiled Plan	Proc	0x0500FF7FC
6	14194	2	8	147456	0x000000046C310060	Compiled Plan	Proc	0x050004009
7	29867	2	2	360448	0x000000046DCEA060	Compiled Plan	Proc	0x050005005
8	7573	2	2	753664	0x000000046D98C060	Compiled Plan	Proc	0x050005002
9	20108	2	2	81920	0x000000046D99A060	Compiled Plan	Proc	0x050005007

Query executed su... | WIN-CQQMK62867E (11.0 SP2) | WIN-CQQMK62867E\Admini... | master | 00:00:00 | 293 rows

SQLQuery1.sql - WIN-CQQMK62867E.master (WIN-CQQMK62867E\Administrator (55))* - Microsoft SQL Server Managem...
File Edit View Query Project Debug Tools Window Help
New Query
master
Execute
Debug
Object Explorer
Connect
WIN-CQQMK628
Databases
Security
Server Objects
Replication
AlwaysOn Hig
Management
Integration Se
SQL Server Ag
SQLQuery1.sql - WI...Administrator (55))*
100 %
Results
Messages
Properties
Current connec
Name
The name of the connection.
Ready
Ln 1
Col 85
Ch 85
INS

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

To collect database plan cache, the following query is used in the application:

```
select * from sys.dm_exec_cached_plans cross apply
sys.dm_exec_sql_text(plan_handle)
```

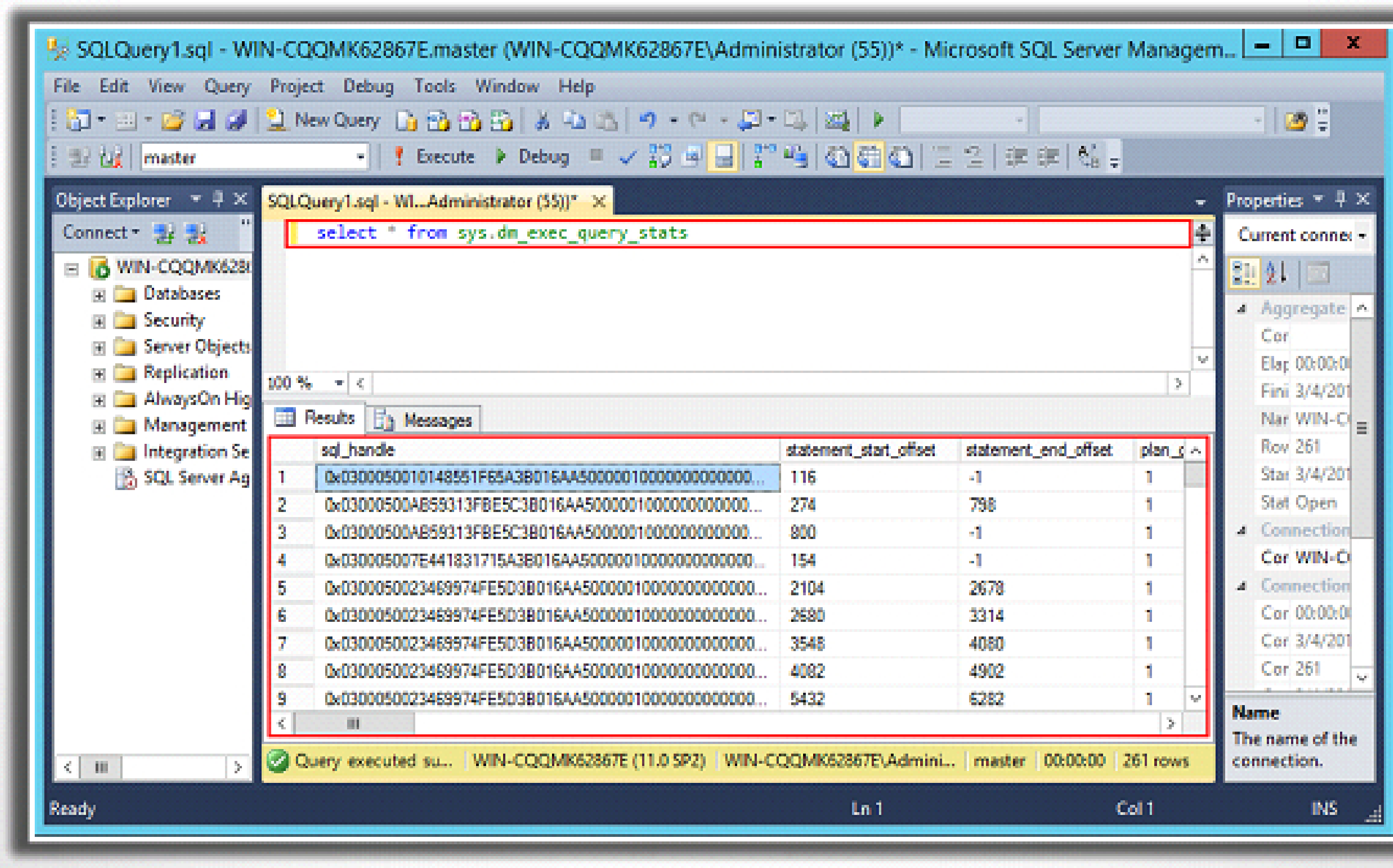
Issuing **sys.dm_exec_cached_plans** in the syntax returns a row for each query plan that the SQL server had cached to speed up the query execution. This dynamic management view will help users to find cached query plans, cached query text, the amount of memory taken by cached plans, and the reuse count of the cached plans.

The command retrieves the SQL text of all cached entries. Note that the `plan_handle` argument in the syntax uniquely identifies a query plan for a batch that server had cached or is currently executing.

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



```
select * from sys.dm_exec_query_stats
```

	sql_handle	statement_start_offset	statement_end_offset	plan_
1	0x030005001014B551F65A3B016AA50000010000000000000...	116	-1	1
2	0x03000500AB59313FBE5C3B016AA50000010000000000000...	274	798	1
3	0x03000500AB59313FBE5C3B016AA50000010000000000000...	800	-1	1
4	0x030005007E441831715A3B016AA50000010000000000000...	154	-1	1
5	0x0300050023469974FE5D3B016AA50000010000000000000...	2104	2678	1
6	0x0300050023469974FE5D3B016AA50000010000000000000...	2680	3314	1
7	0x0300050023469974FE5D3B016AA50000010000000000000...	3548	4080	1
8	0x0300050023469974FE5D3B016AA50000010000000000000...	4082	4902	1
9	0x0300050023469974FE5D3B016AA50000010000000000000...	5432	6282	1

Query executed su... | WIN-CQQMK62867E (11.0 SP2) | WIN-CQQMK62867E\Admini... | master | 00:00:00 | 261 rows

To collect additional plan cache specifics from the database, like viewing the aggregate performance statistics, the following query is used.

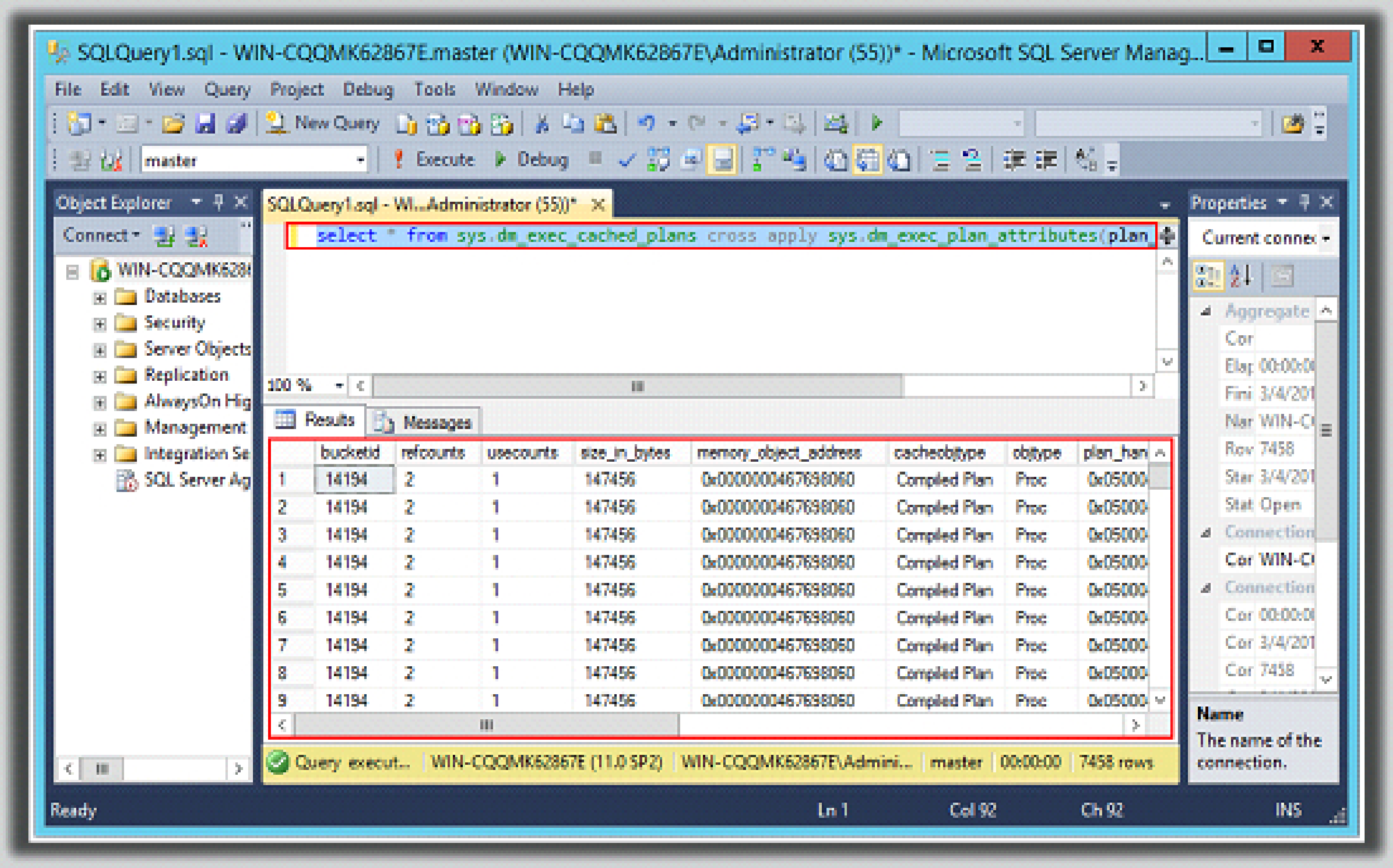
`select * from sys.dm_exec_query_stats`

The result contains one row per query statement within the cached plan, and the lifetime of the rows is tied to the plan itself. When a plan is removed from the cache, the corresponding rows are eliminated from this view.

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



```
select * from sys.dm_exec_cached_plans cross apply sys.dm_exec_plan_attributes(plan
```

	bucketid	refcounts	usecounts	size_in_bytes	memory_object_address	cacheobjtype	objtype	plan_han
1	14194	2	1	147456	0x0000000467698060	Compiled Plan	Proc	0x05000
2	14194	2	1	147456	0x0000000467698060	Compiled Plan	Proc	0x05000
3	14194	2	1	147456	0x0000000467698060	Compiled Plan	Proc	0x05000
4	14194	2	1	147456	0x0000000467698060	Compiled Plan	Proc	0x05000
5	14194	2	1	147456	0x0000000467698060	Compiled Plan	Proc	0x05000
6	14194	2	1	147456	0x0000000467698060	Compiled Plan	Proc	0x05000
7	14194	2	1	147456	0x0000000467698060	Compiled Plan	Proc	0x05000
8	14194	2	1	147456	0x0000000467698060	Compiled Plan	Proc	0x05000
9	14194	2	1	147456	0x0000000467698060	Compiled Plan	Proc	0x05000

Query execut... | WIN-CQQMK62867E (11.0 SP2) | WIN-CQQMK62867E\Admini... | master | 00:00:00 | 7458 rows

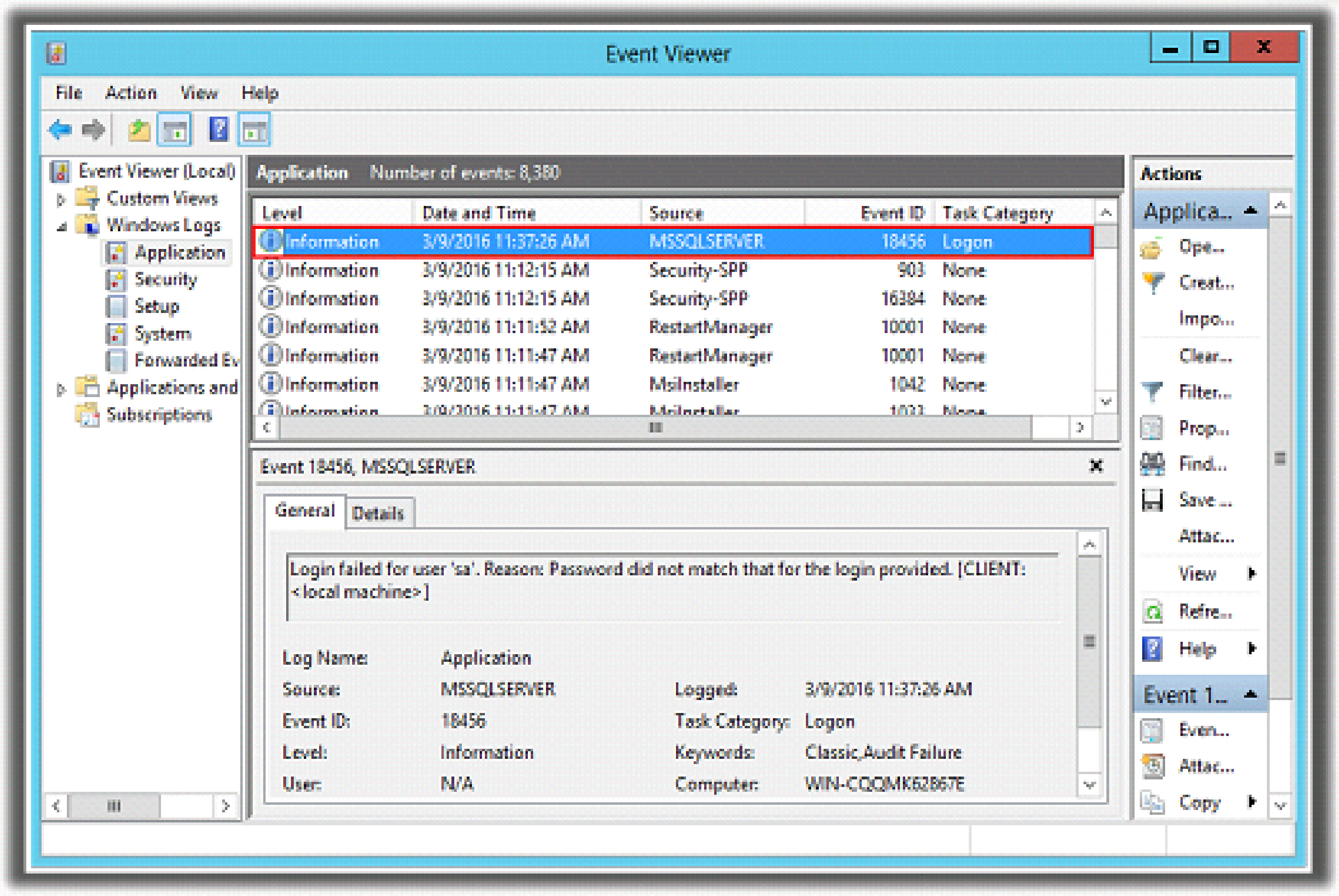
To view one row per plan attribute for the plan specified by the plan handle, the following query is used.

`select * from sys.dm_exec_cached_plans cross apply sys.dm_exec_plan_attributes(plan_handle)`

It is to be noted that `plan_handle` in the syntax uniquely identifies a query plan for a batch that has executed and whose plan resides in the plan cache.

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**



Application Number of events: 8,380

Level	Date and Time	Source	Event ID	Task Category
Information	3/9/2016 11:37:26 AM	MSSQLSERVER	18456	Logon
Information	3/9/2016 11:12:15 AM	Security-SPP	903	None
Information	3/9/2016 11:12:15 AM	Security-SPP	16384	None
Information	3/9/2016 11:11:52 AM	RestartManager	10001	None
Information	3/9/2016 11:11:47 AM	RestartManager	10001	None
Information	3/9/2016 11:11:47 AM	MsiInstaller	1042	None

Event 18456, MSSQLSERVER

Login failed for user 'sa'. Reason: Password did not match that for the login provided. [CLIENT: <local machine>]

Log Name:	Application		
Source:	MSSQLSERVER	Logged:	3/9/2016 11:37:26 AM
Event ID:	18456	Task Category:	Logon
Level:	Information	Keywords:	Classic,Audit Failure
User:	N/A	Computer:	WIN-CQQMK62867E

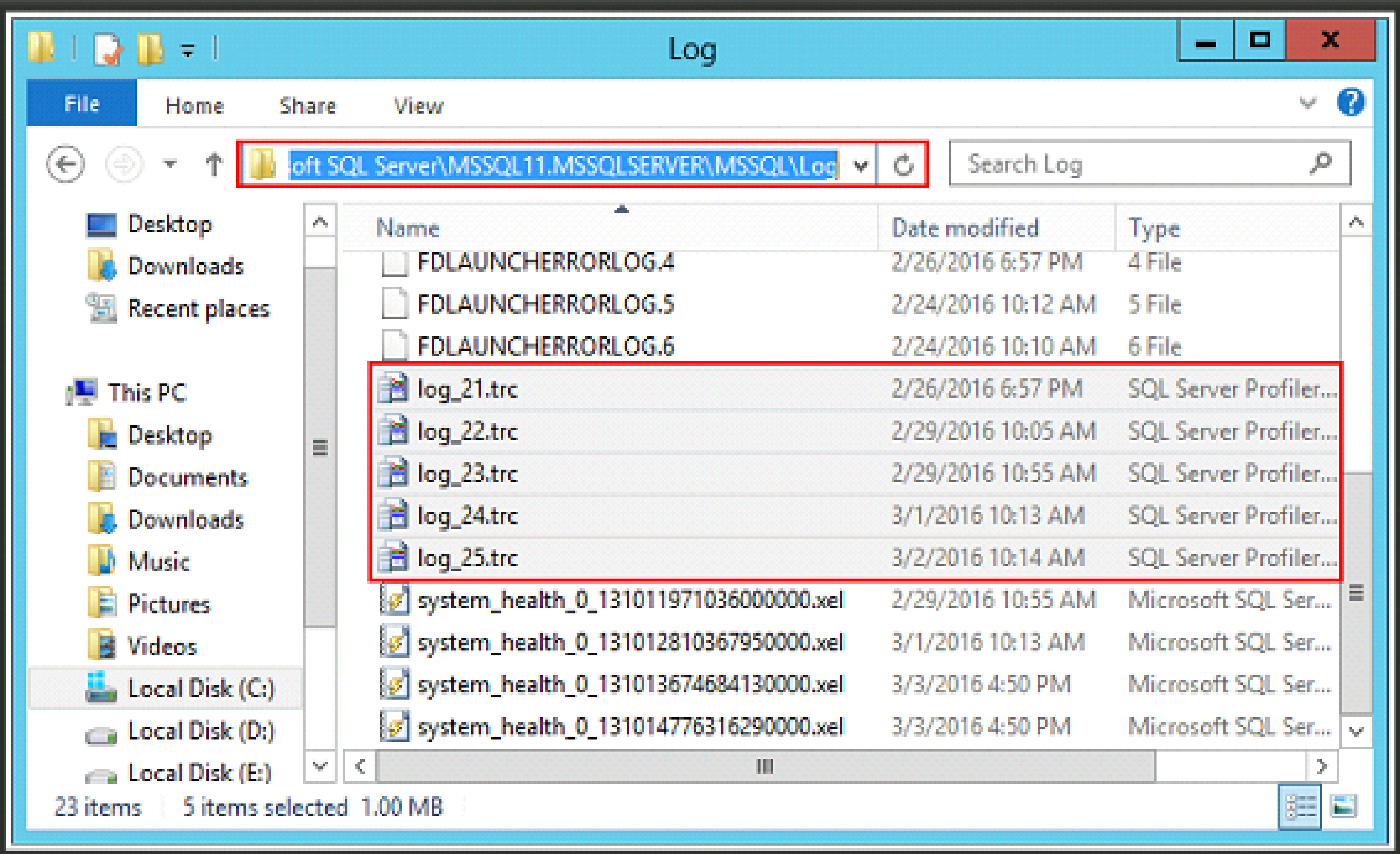
Windows event logs are simple text files in XML format (EVTX) used by Windows Vista and later versions. Windows holds different types of logs including Administrative, Operational, Analytic, Debug, application, etc.

The Event Viewer in the Windows operating system (OS) allows the user to view the event logs on a local or a remote machine. Launch Event Viewer, expand Windows Logs node and select the type of logs (i.e., logs pertaining to the Application, Security, Setup, System, or Forwarded Events) need to be viewed.

In the forensic point of view, the event log files play a vital role, as these event logs track all the "significant events" on any computer. Any program that runs on the computer posts a notification in the event log, and simultaneously posts a notification before it ends. Events which include system access, operating system jerk, driver or any hardware issue, etc., are saved in the event logs. Investigators can use this data to trace out the attackers.

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

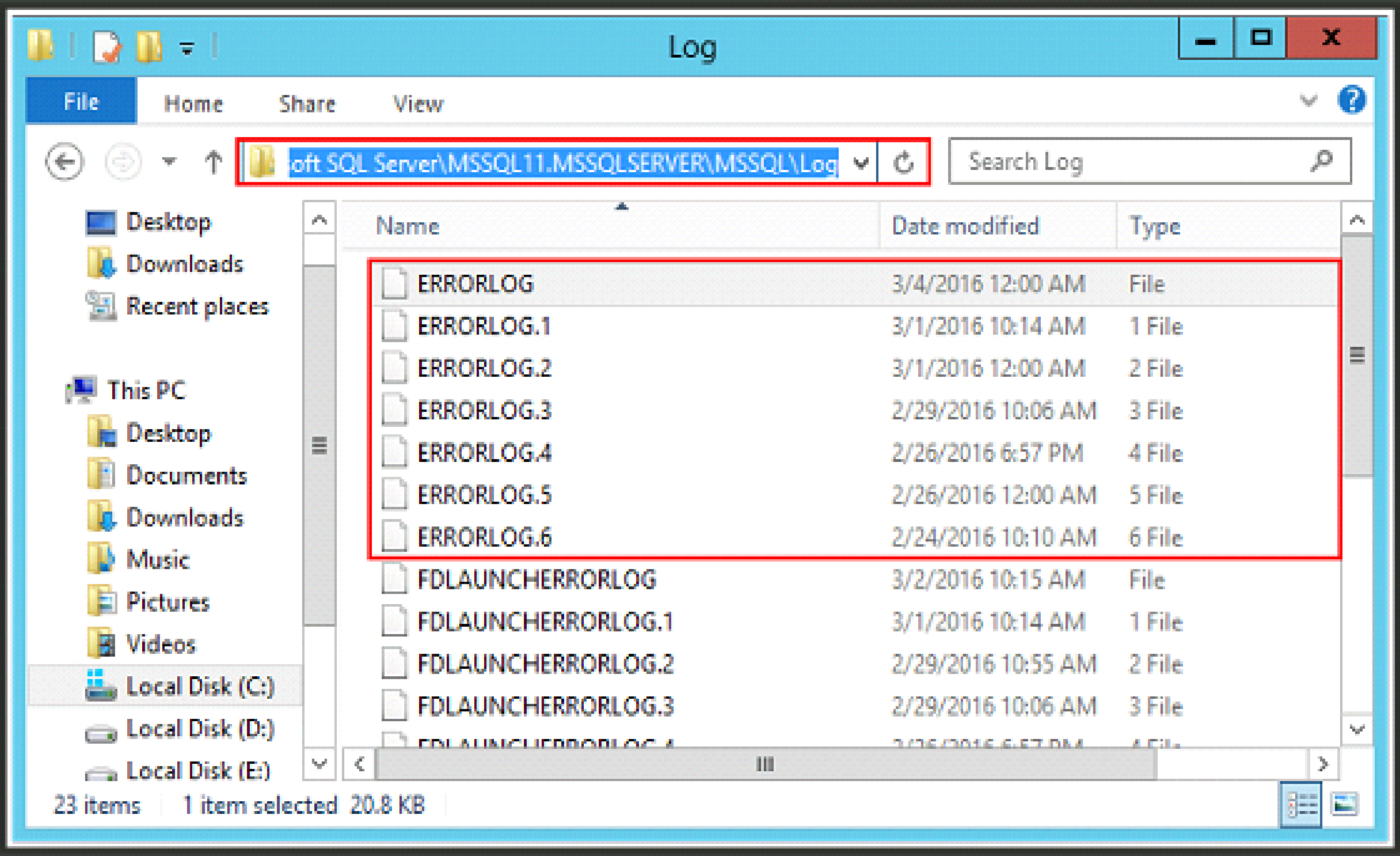


Name	Date modified	Type
FDLAUNCHERRORLOG.4	2/26/2016 6:57 PM	4 File
FDLAUNCHERRORLOG.5	2/24/2016 10:12 AM	5 File
FDLAUNCHERRORLOG.6	2/24/2016 10:10 AM	6 File
log_21.trc	2/26/2016 6:57 PM	SQL Server Profiler...
log_22.trc	2/29/2016 10:05 AM	SQL Server Profiler...
log_23.trc	2/29/2016 10:55 AM	SQL Server Profiler...
log_24.trc	3/1/2016 10:13 AM	SQL Server Profiler...
log_25.trc	3/2/2016 10:14 AM	SQL Server Profiler...
system_health_0_131011971036000000.xel	2/29/2016 10:55 AM	Microsoft SQL Ser...
system_health_0_131012810367950000.xel	3/1/2016 10:13 AM	Microsoft SQL Ser...
system_health_0_131013674684130000.xel	3/3/2016 4:50 PM	Microsoft SQL Ser...
system_health_0_131014776316290000.xel	3/3/2016 4:50 PM	Microsoft SQL Ser...

23 items 5 items selected 1.00 MB

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



Name	Date modified	Type
ERRORLOG	3/4/2016 12:00 AM	File
ERRORLOG.1	3/1/2016 10:14 AM	1 File
ERRORLOG.2	3/1/2016 12:00 AM	2 File
ERRORLOG.3	2/29/2016 10:06 AM	3 File
ERRORLOG.4	2/26/2016 6:57 PM	4 File
ERRORLOG.5	2/26/2016 12:00 AM	5 File
ERRORLOG.6	2/24/2016 10:10 AM	6 File
FDLAUNCHERRORLOG	3/2/2016 10:15 AM	File
FDLAUNCHERRORLOG.1	3/1/2016 10:14 AM	1 File
FDLAUNCHERRORLOG.2	2/29/2016 10:55 AM	2 File
FDLAUNCHERRORLOG.3	2/29/2016 10:06 AM	3 File

23 items 1 item selected 20.8 KB

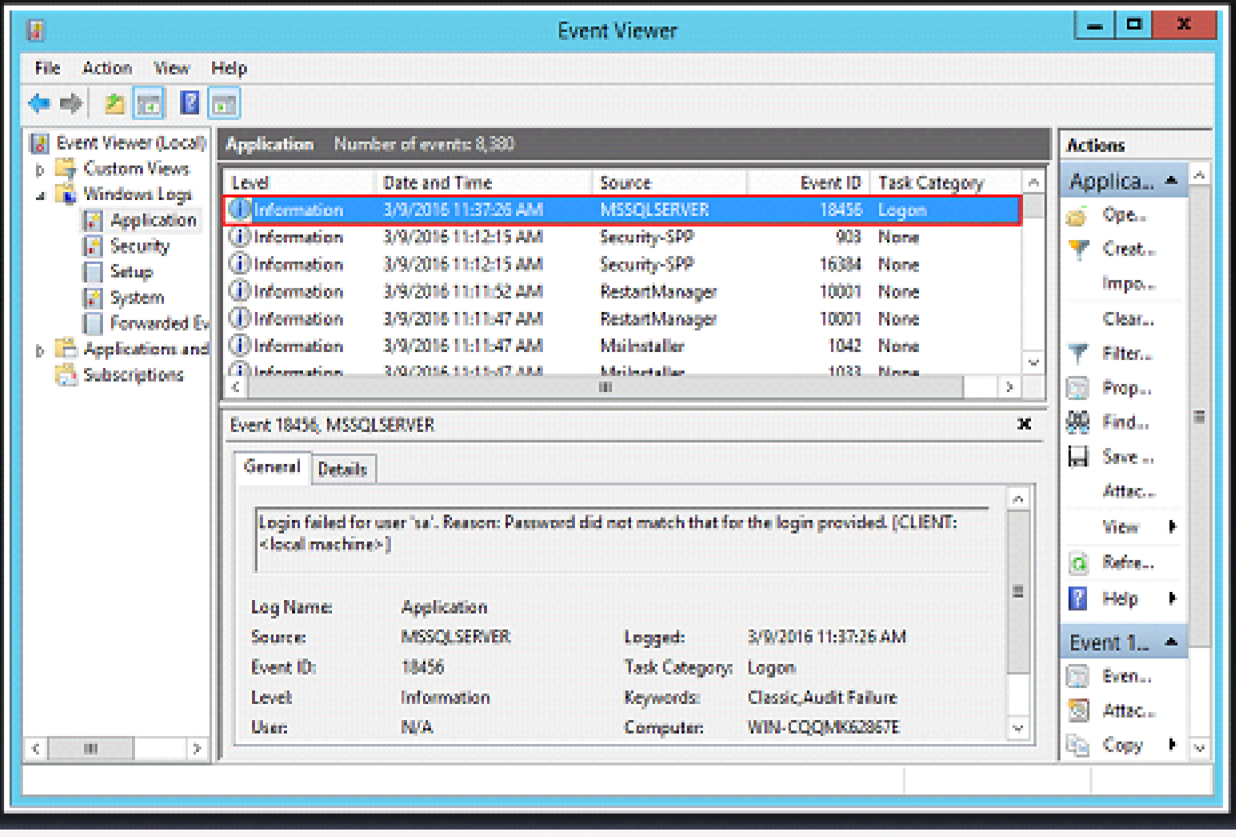
As discussed above, Trace files record all the events occurred on the SQL Server and databases present in it, while SQL Server error logs record user-defined events and specific system events. The error logs also contain the IP Address of SQL Server client connections. A new error log file is created every time a new SQL Server instance occurs.

Forensic investigators may use SQL Server Profiler to view the trace files, and SQL Server Management Studio or any text editor to view the error logs. Both the files act as a very important evidence for the forensic examiner while conducting an investigation on the SQL Server.

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



Level	Date and Time	Source	Event ID	Task Category
Information	3/9/2016 11:37:26 AM	MSSQLSERVER	18456	Logon
Information	3/9/2016 11:12:15 AM	Security-SPP	903	None
Information	3/9/2016 11:12:15 AM	Security-SPP	16384	None
Information	3/9/2016 11:11:52 AM	RestartManager	10001	None
Information	3/9/2016 11:11:47 AM	RestartManager	10001	None
Information	3/9/2016 11:11:47 AM	MsiInstaller	1042	None

Event 18456, MSSQLSERVER

Login failed for user 'sa'. Reason: Password did not match that for the login provided. [CLIENT: <local machine>]

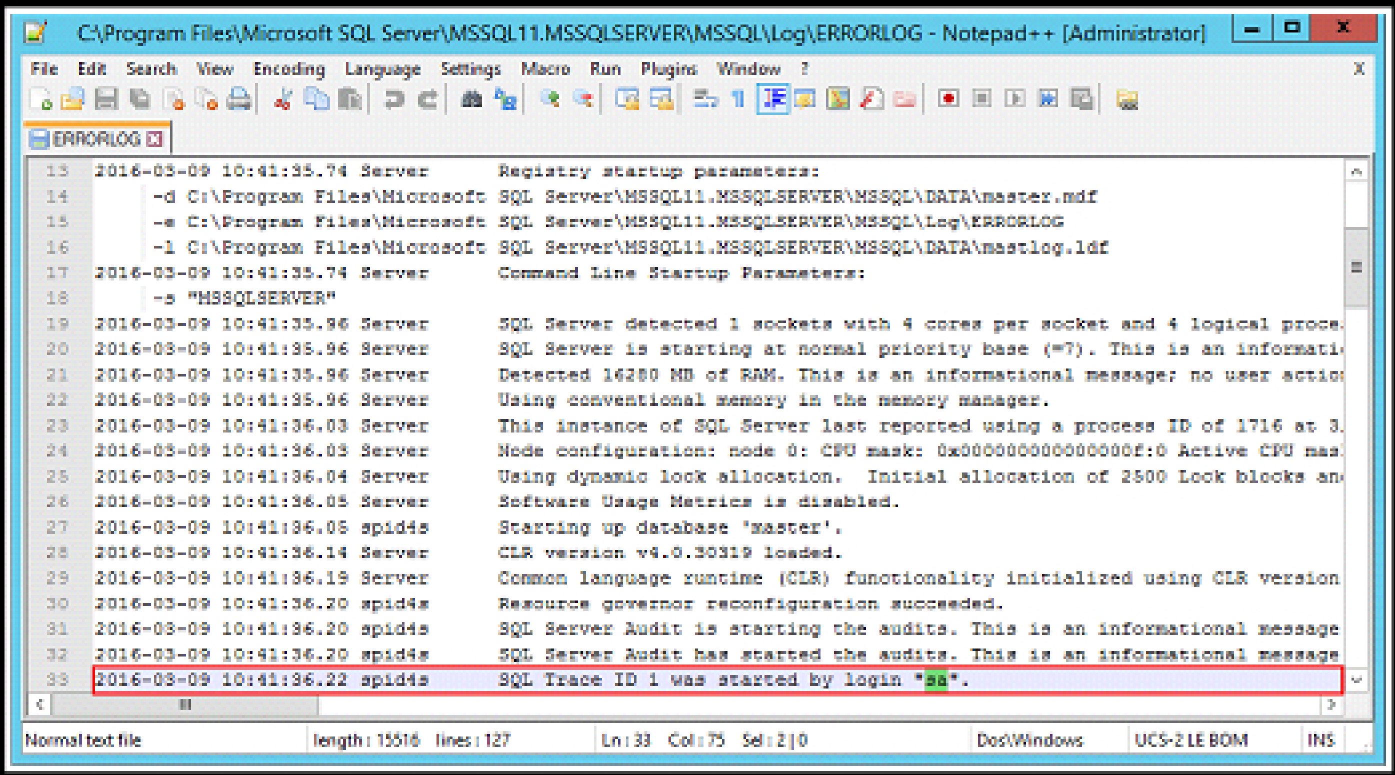
Log Name:	Application		
Source:	MSSQLSERVER	Logged:	3/9/2016 11:37:26 AM
Event ID:	18456	Task Category:	Logon
Level:	Information	Keywords:	Classic,Audit Failure
User:	N/A	Computer:	WIN-CQQMK62867E

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)

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]

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

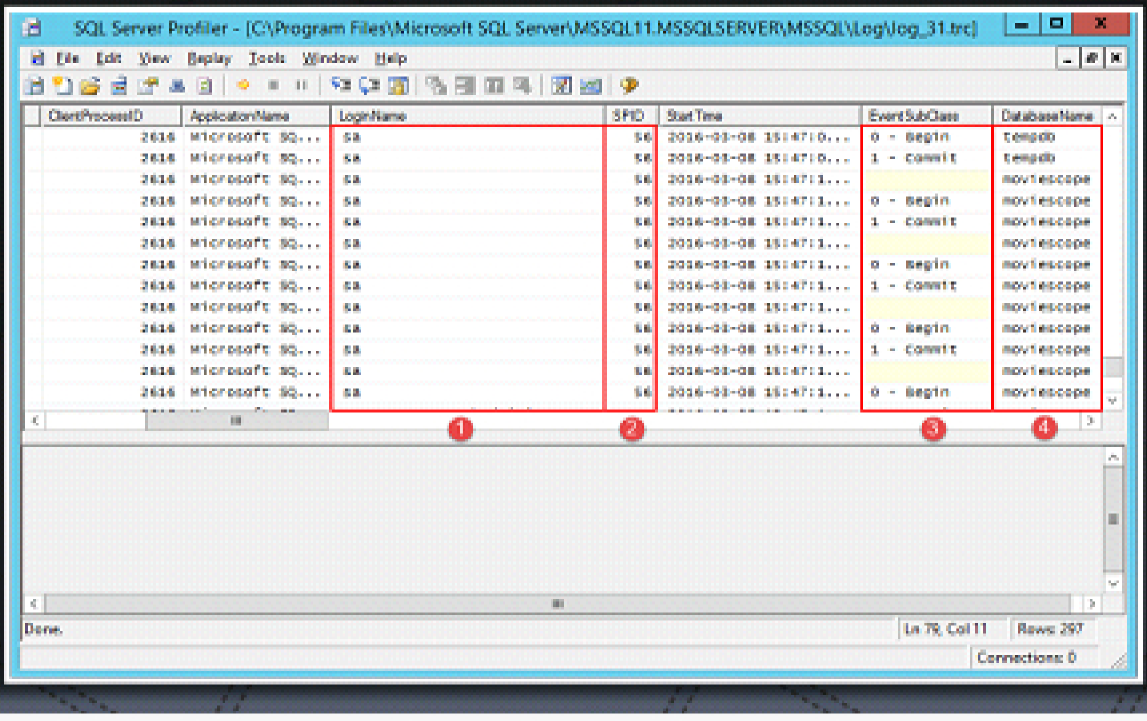
Here, it is evident that there is a successful login instance recorded on the name of a user sa. Now, the trace file can be viewed to examine the SQL Server based events associated with this user.

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

Done.                                                  Ln 79, Col 11   Rows: 297
                                                       Connections: 0
```

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

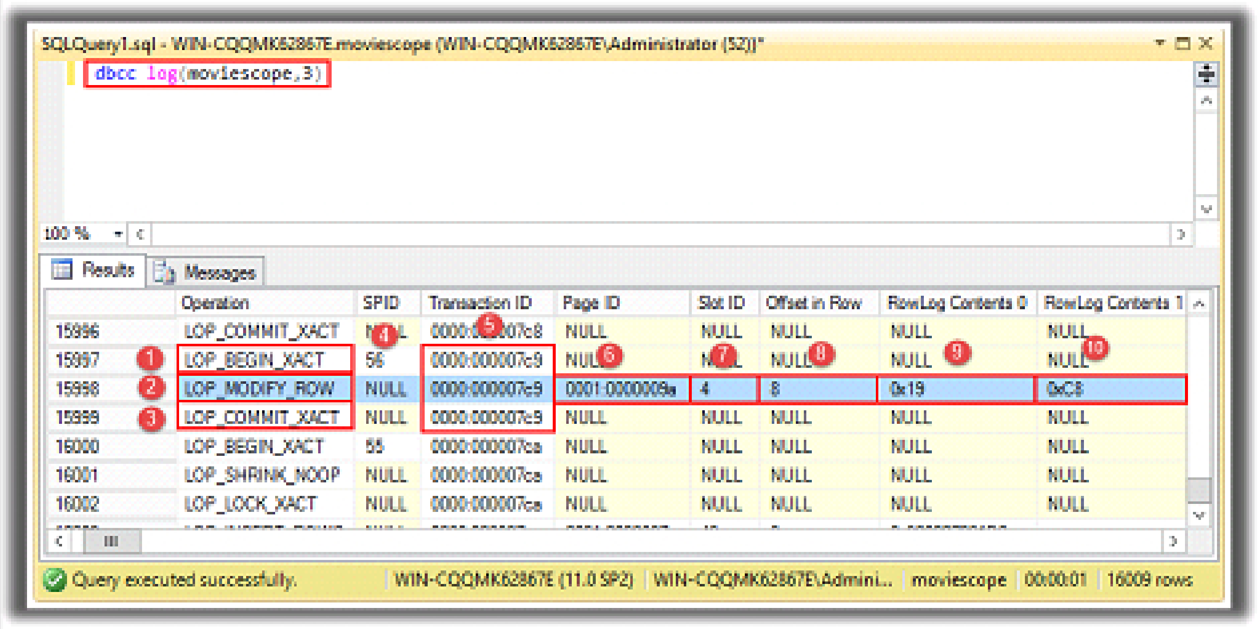
6 - Data Page Identifier for rows containing the updated record

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



```
SQLQuery1.sql - WIN-CQQMK62867E.moviescope (WIN-CQQMK62867E\Administrator (52))*
dbcc log(moviescope,3)
```

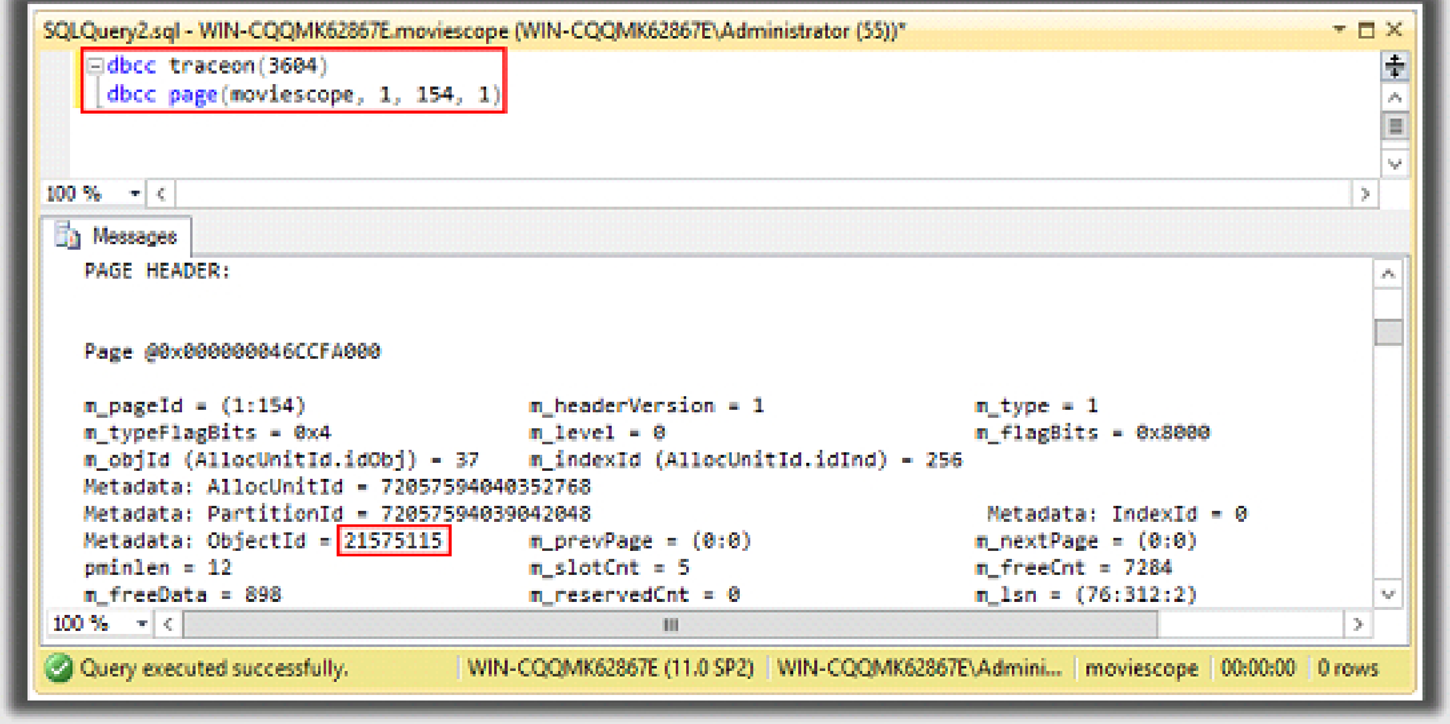
```
       Operation          SPID   Transaction ID   Page ID          Slot ID  Offset in Row  RowLog Contents 0  RowLog Contents 1
15996  LOP_COMMIT_XACT    NULL   0000:000007c8    NULL             NULL     NULL           NULL               NULL
15997  LOP_BEGIN_XACT     56     0000:000007c9    NULL             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

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

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)
```

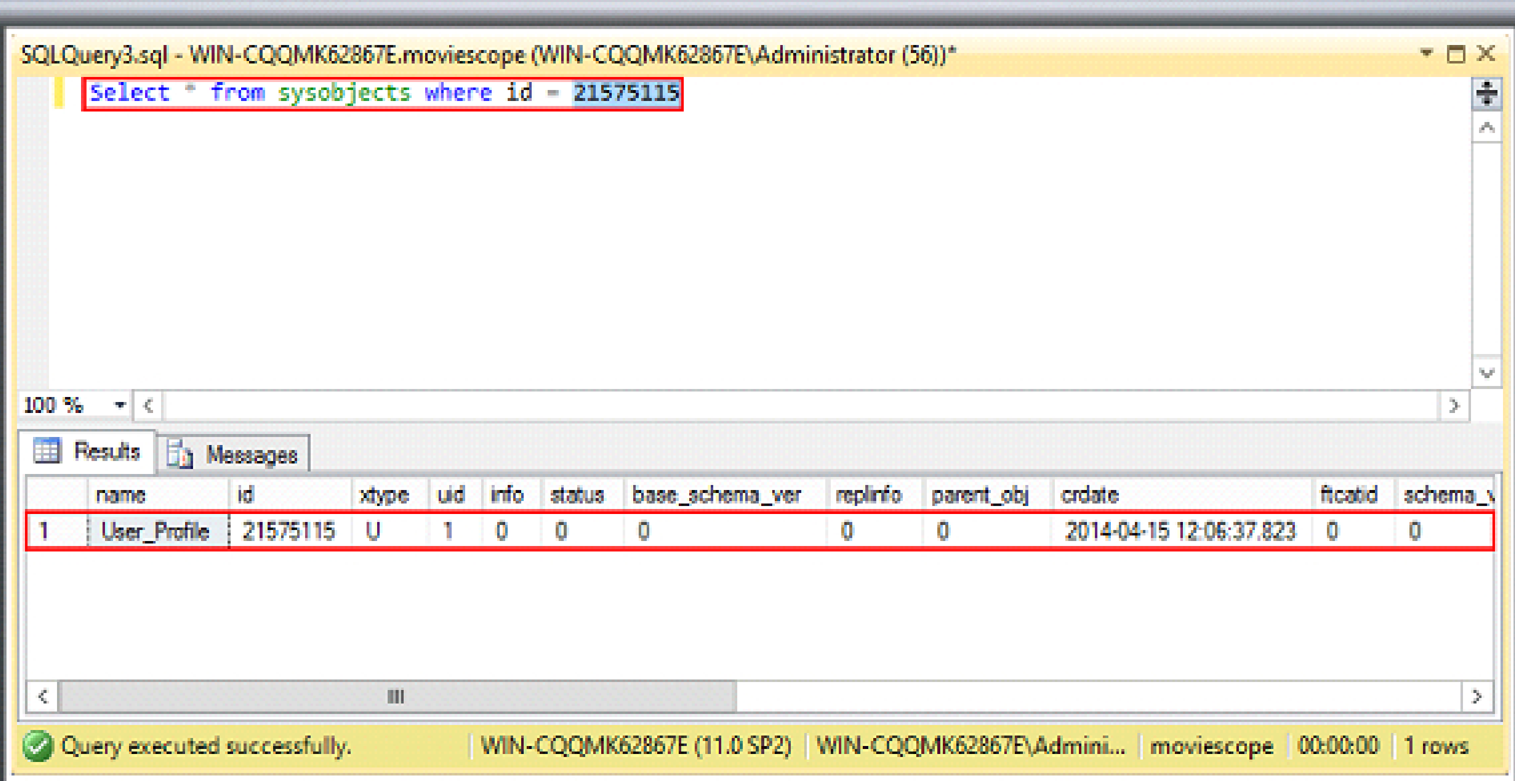
```
PAGE HEADER:


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)
```

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
```

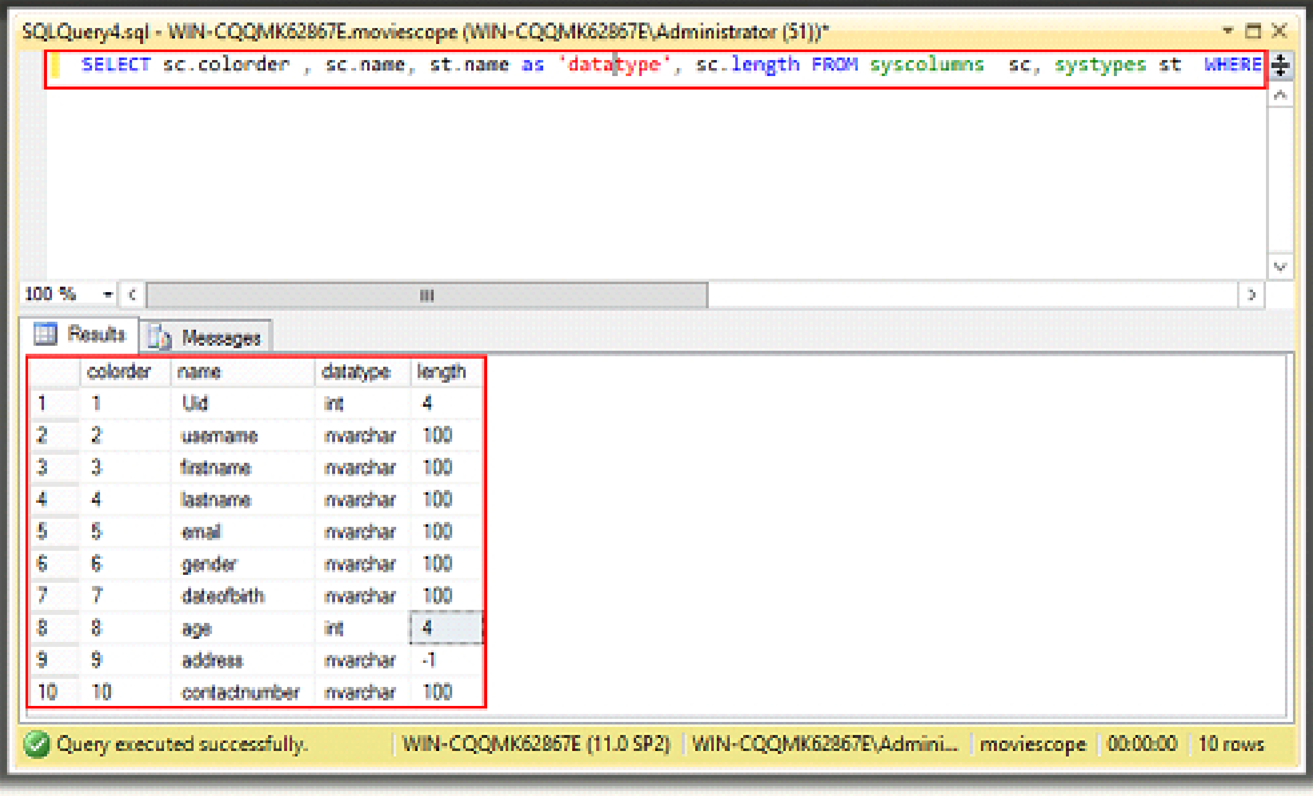
	name	id	xtype	uid	info	status	base_schema_ver	replinfo	parent_obj	crdate	ftcatid	schema_v
1	User_Profile	21575115	U	1	0	0	0	0	0	2014-04-15 12:06:37.823	0	0

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
```

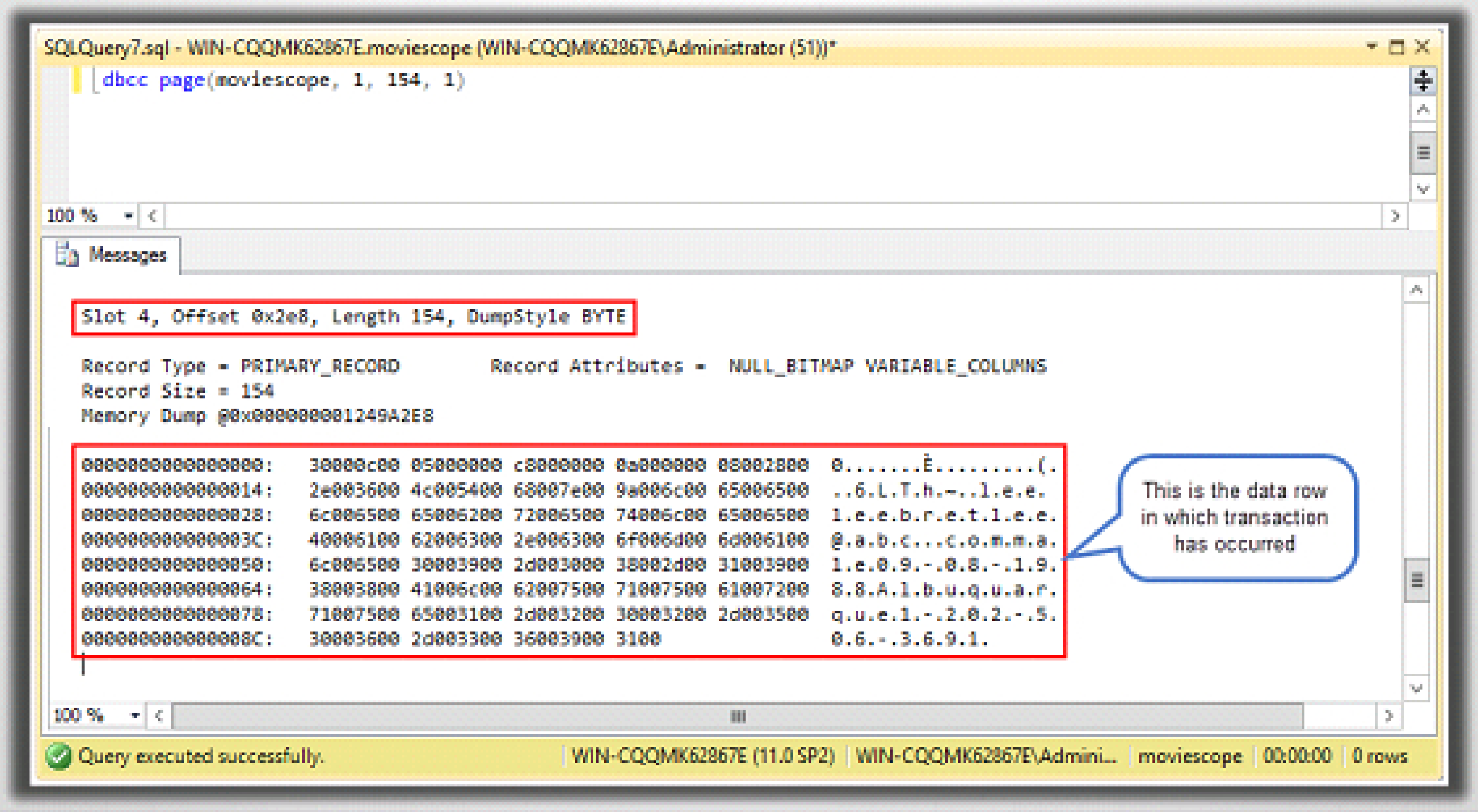
	colorder	name	datatype	length
1	1	Uid	int	4
2	2	username	nvarchar	100
3	3	firstname	nvarchar	100
4	4	lastname	nvarchar	100
5	5	email	nvarchar	100
6	6	gender	nvarchar	100
7	7	dateofbirth	nvarchar	100
8	8	age	int	4
9	9	address	nvarchar	-1
10	10	contactnumber	nvarchar	100

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)
```

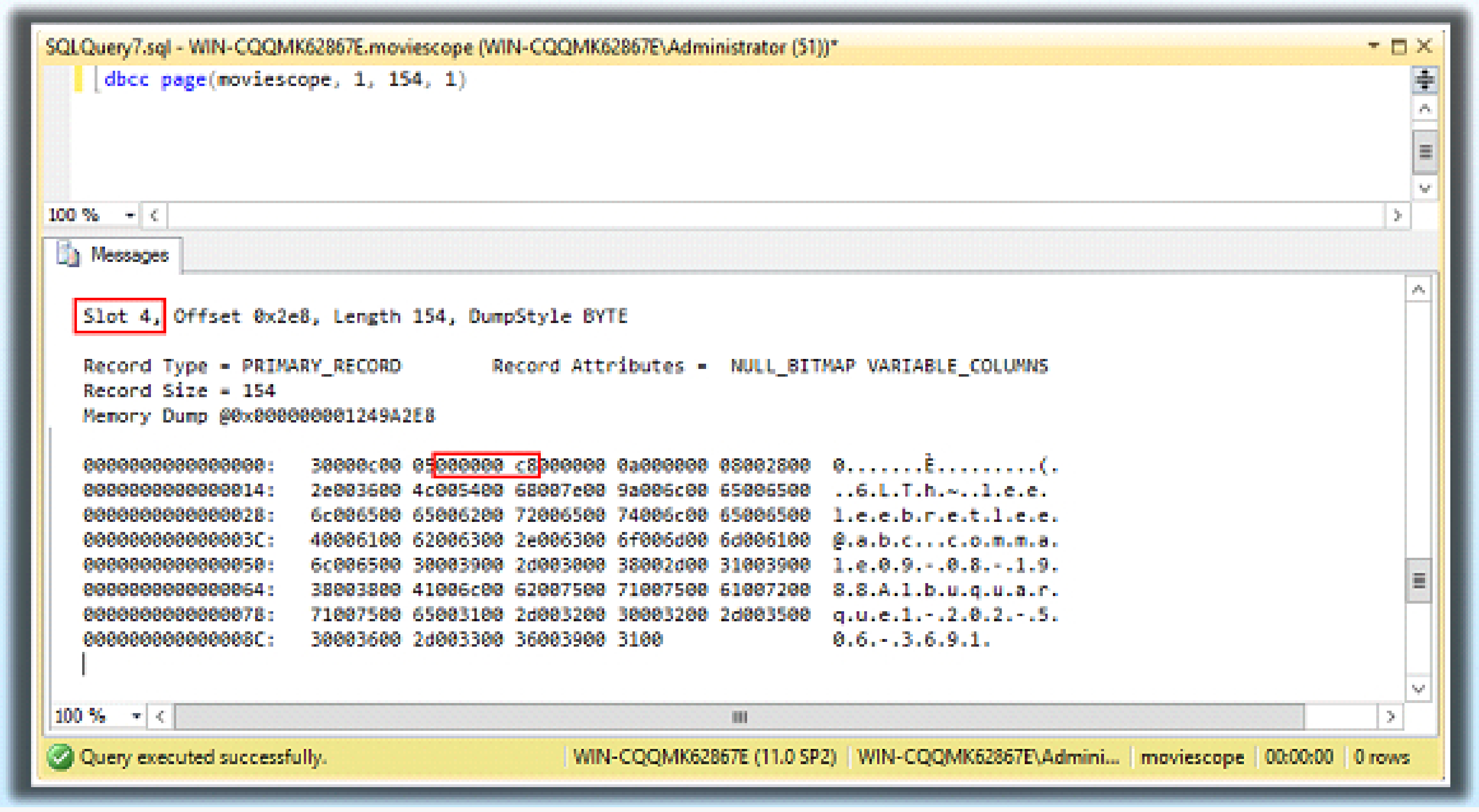
```
Slot 4, Offset 0x2e8, Length 154, DumpStyle BYTE

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.
```

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)
```

```
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.
```

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

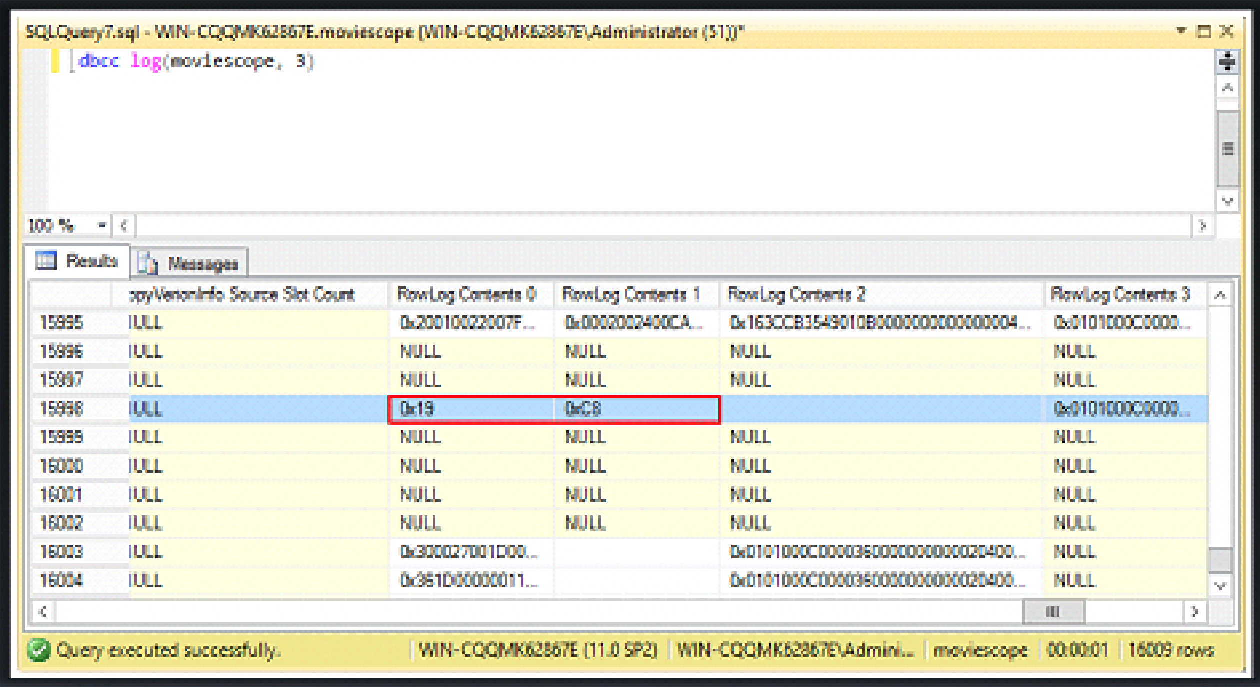
Using the table schema obtained earlier, the data type within this row offset is the age column which contains a 4-byte int data type.

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)
```

	opyVerionInfo Source Slot Count	RowLog Contents 0	RowLog Contents 1	RowLog Contents 2	RowLog Contents 3
15995	IULL	0x20010022007F...	0x0002002400CA...	0x163CCB354901080000000000000004...	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...		0x0101000C000036000000000020400...	NULL
16004	IULL	0x361D00000011...		0x0101000C000036000000000020400...	NULL

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

Windows Event Viewer records all the events that occur on a system. In conjunction with the system logs, the application also records the MSSQL logs at an instance of a login attempt failure, or a SQL Server initiation/shutdown.

Therefore, examining the windows event logs help forensic investigators to examine the logs and determine any false login records on the event viewer.

SQL Server Management Studio (SSMS):

Source: *https://msdn.microsoft.com*

SQL Server Management Studio (SSMS) is an integrated environment for accessing, configuring, managing, administering, and developing all components of SQL Server and Azure SQL Database. SSMS combines a group of graphical tools with script editors to provide access to SQL Server to developers and administrators of all skill levels.

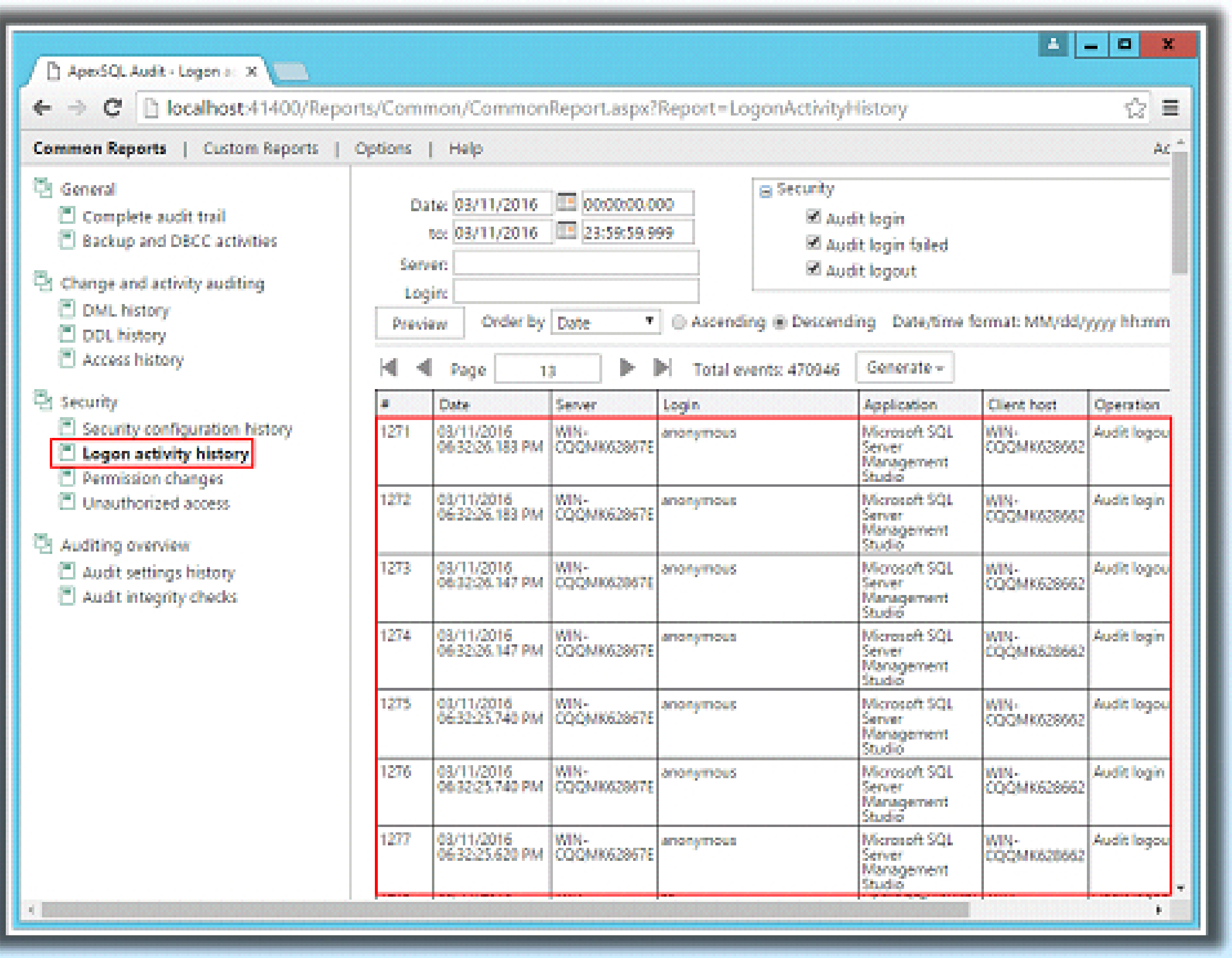
Forensic investigators need to have good knowledge of how to use various functions (such as dbcc log, fn_dblog, etc.) in the SSMS to view and analyze the logs in plain text format.

As discussed in the above slides, both Windows Event Viewer and SQL Server Management Studio help a forensic examiner in investigating the SQL Server databases. Along with these applications, forensic investigators use some other database management and monitoring tools such as ApexSQL DBA, SQLite Database Browser, Adminer, etc. to perform a forensic investigation on SQL Server databases.

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)



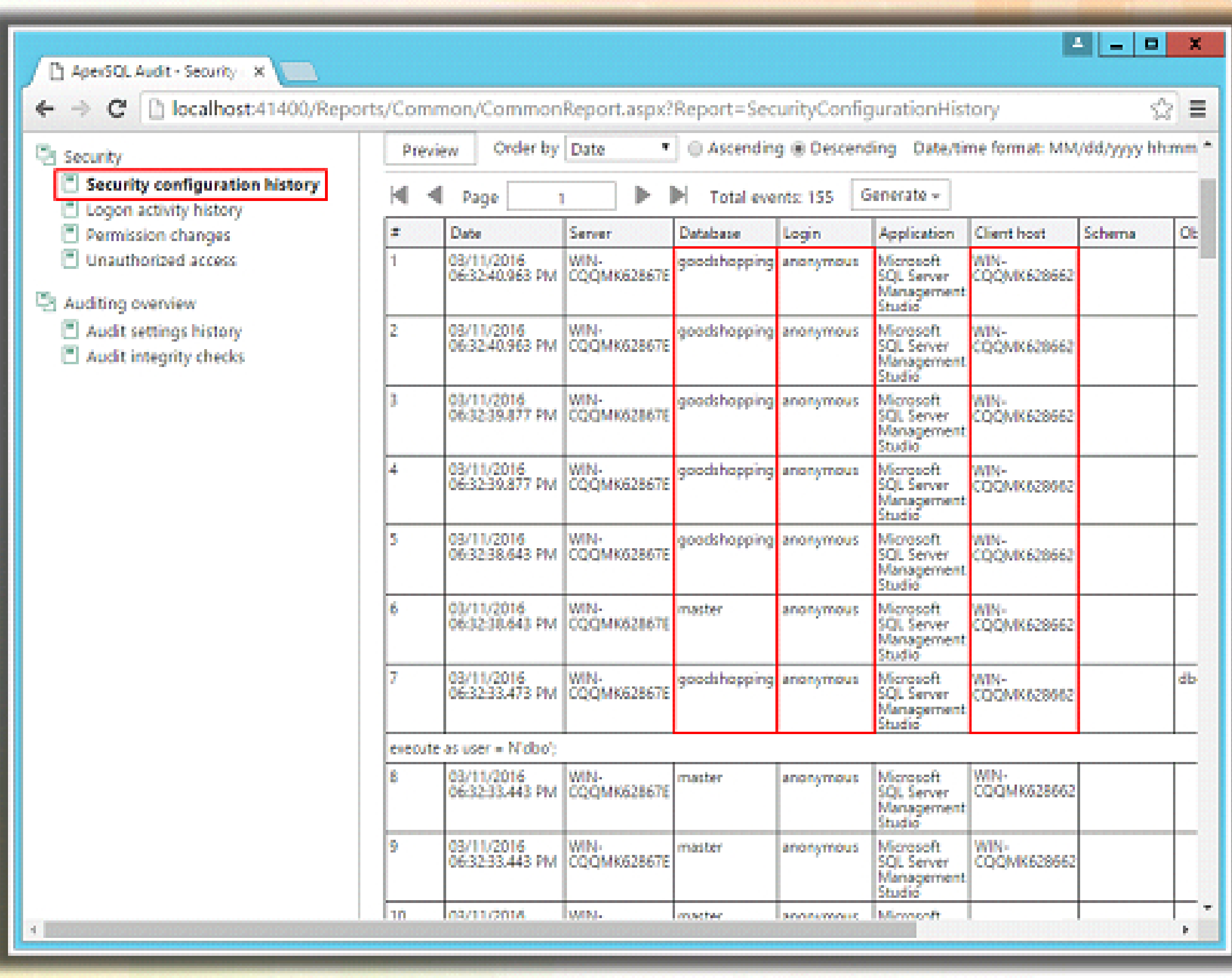
Date: 03/11/2016 00:00:00.000
to: 03/11/2016 23:59:59.999
Server:
Login:
Security: [✓] Audit login [✓] Audit login failed [✓] Audit logout
Order by: Date   Ascending [✓ Descending]   Date/time format: MM/dd/yyyy hh:mm
Page 13   Total events: 470946

#	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



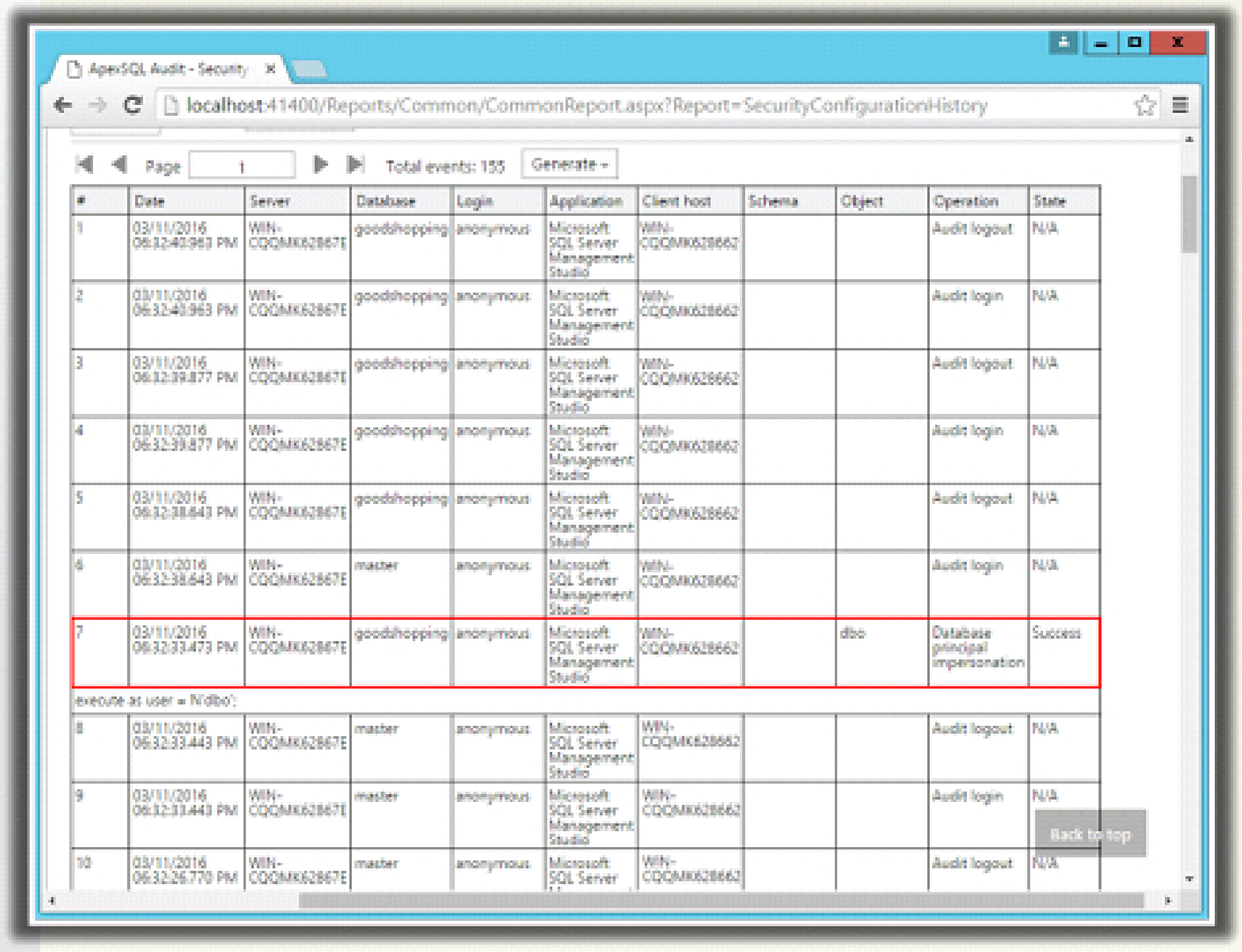
Order by: Date   Ascending [✓ Descending]   Date/time format: MM/dd/yyyy hh:mm
Page 1   Total events: 155

#	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


ApexSQL Audit - Security

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

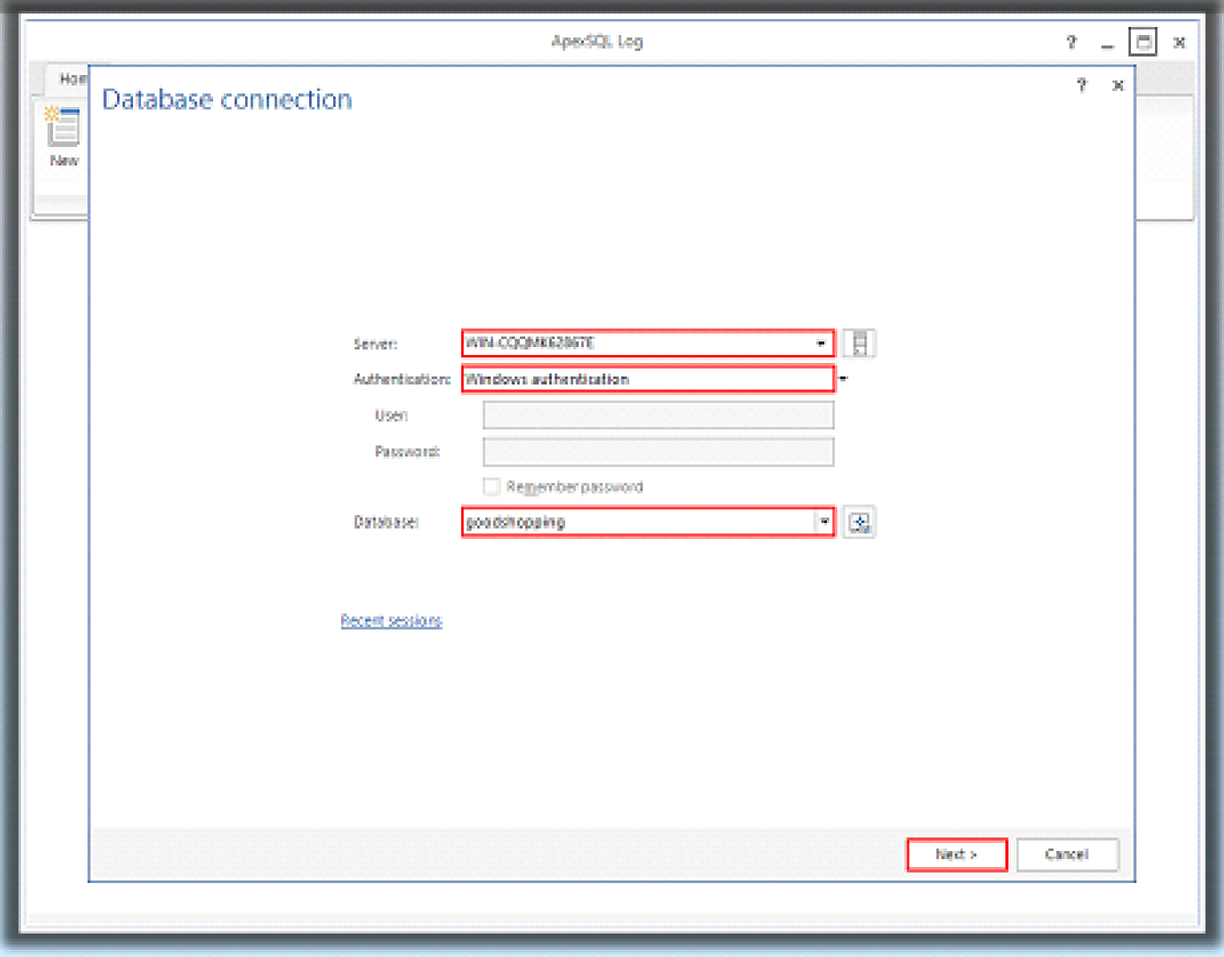
Page 1 Total events: 155 Generate

#	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

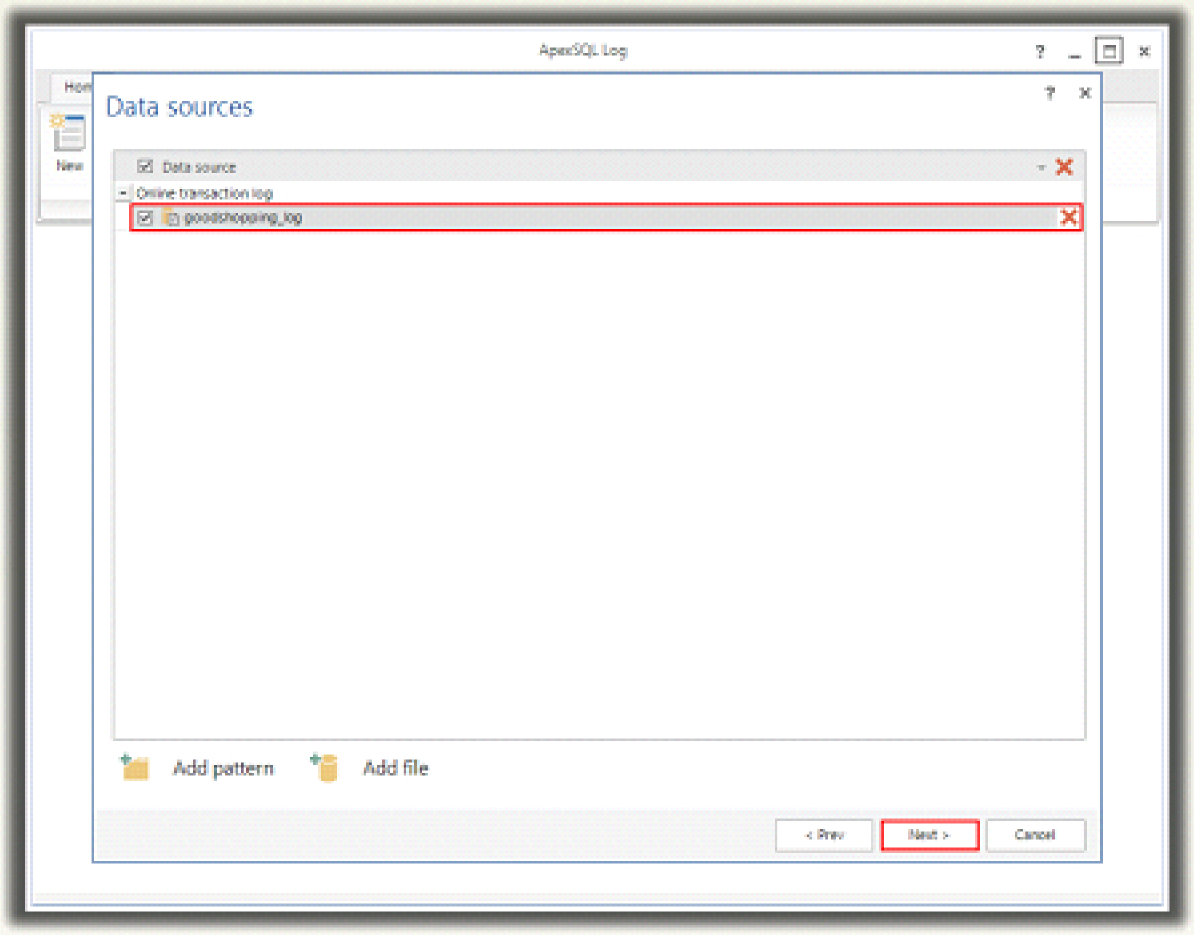


Server: WIN-CQQMK62867E
Authentication: Windows authentication
User:
Password:
Remember password
Database: goodshopping

Database Forensics Using ApexSQL DBA (Cont'd)

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

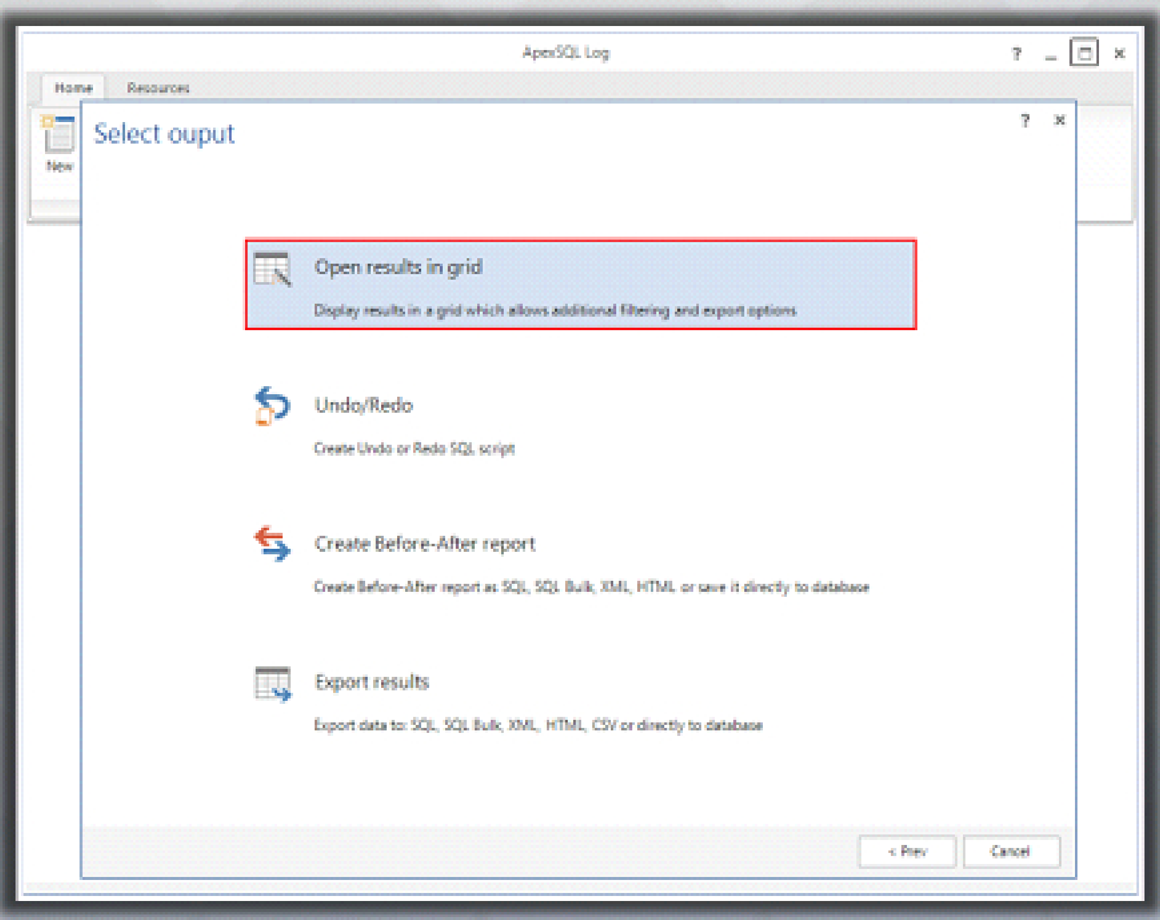
Select the goodshopping log file in the Data Sources section



Database Forensics Using ApexSQL DBA (Cont'd)

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

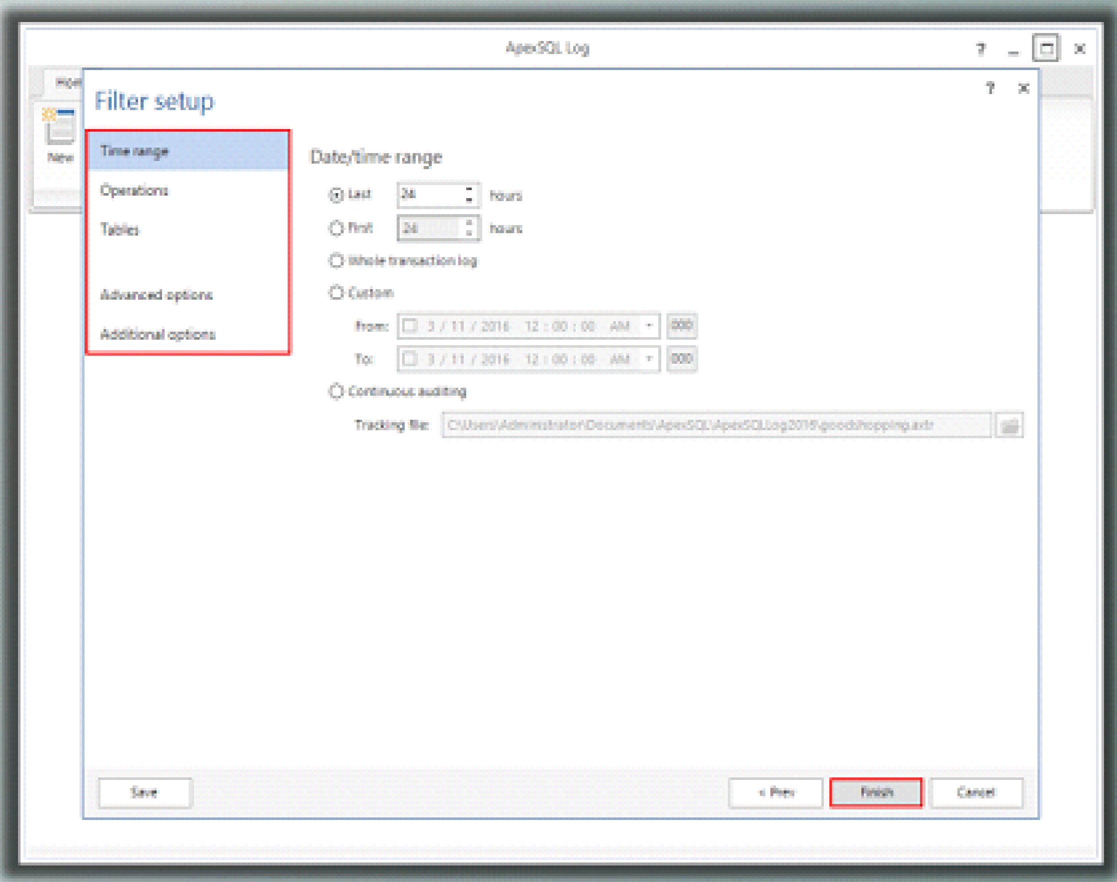
Select Open results in grid option in the Select output section


ApexSQL Log
Home
Resources
New
Select ouput
Open results in grid
Display results in a grid which allows additional filtering and export options
Undo/Redo
Create Undo or Redo SQL script
Create Before-After report
Create Before-After report as SQL, SQL Bulk, XML, HTML or save it directly to database
Export results
Export data to: SQL, SQL Bulk, XML, HTML, CSV or directly to database
< Prev
Cancel

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**


ApexSQL Log

Filter setup

- Time range
- Operations
- Tables
- Advanced options
- Additional options

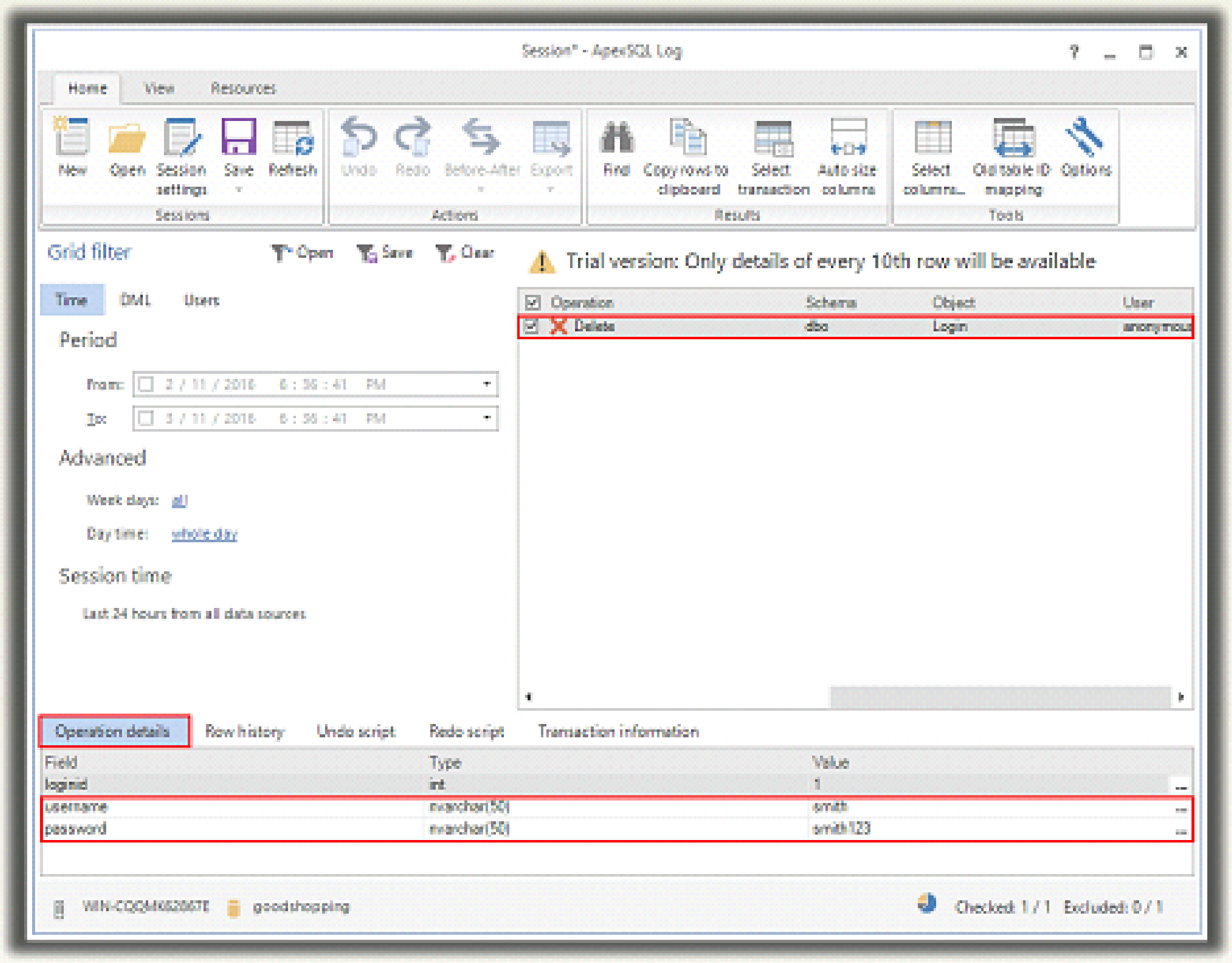
Date/time range

- [✓ Last] 24 hours
- First 24 hours
- Whole transaction log
- Custom
  - From: 3 / 11 / 2016 12 : 00 : 00 AM 000
  - To: 3 / 11 / 2016 12 : 00 : 00 AM 000
- Continuous auditing
  - Tracking file: C:\Users\Administrator\Documents\ApexSQL\ApexSQLLog2016\goodshopping.axtr

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.



Trial version: Only details of every 10th row will be available

Period
From: 3 / 11 / 2016 6 : 36 : 41 PM
To: 3 / 11 / 2016 6 : 36 : 41 PM

Advanced
Week days: all
Day time: whole day

Session time
Last 24 hours from all data sources

Operation	Schema	Object	User
Delete	dbo	Login	anonymous

Operation details

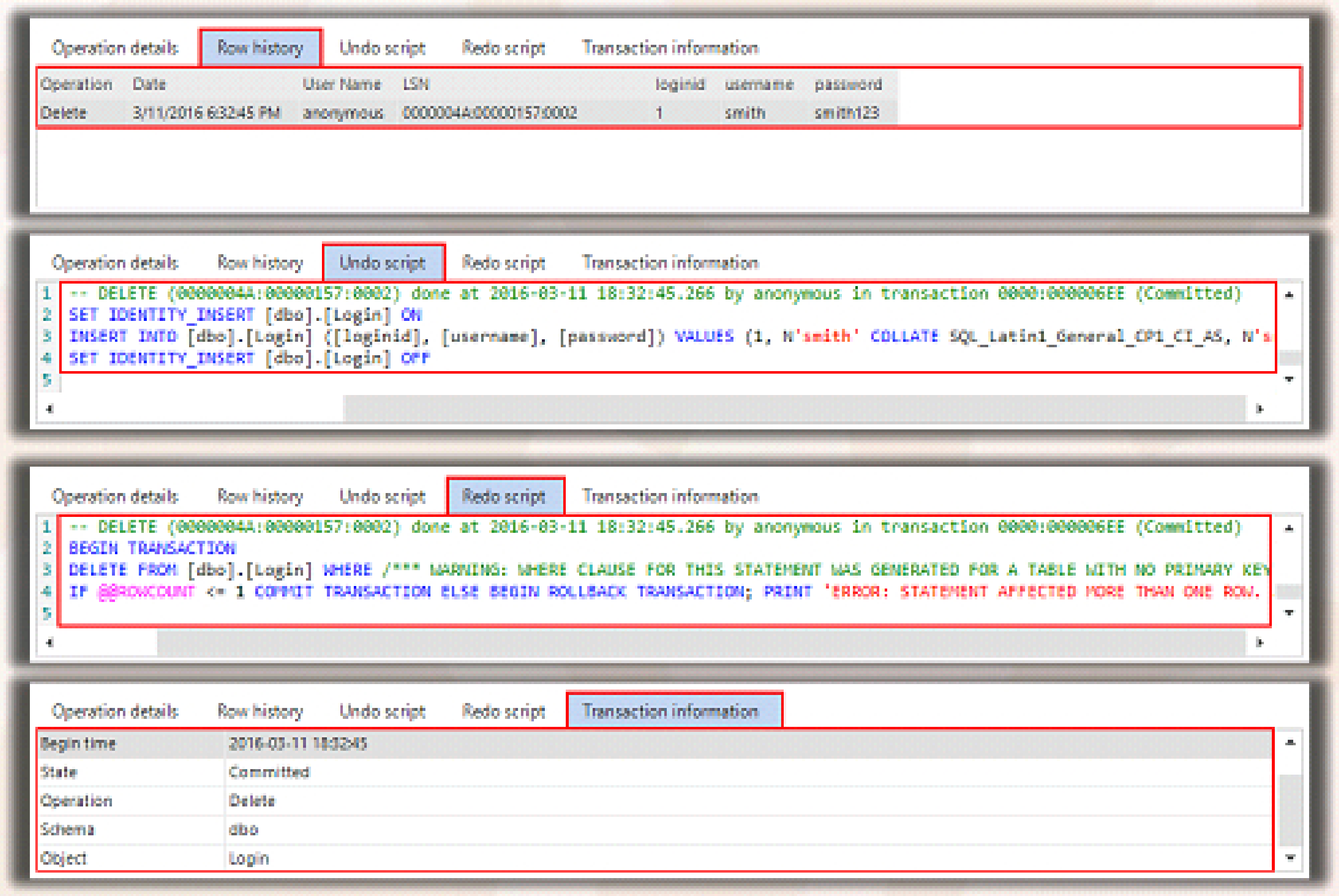
Field	Type	Value
loginid	int	1
username	nvarchar(50)	smith
password	nvarchar(50)	smith123

WIN-CQQMK62067E goodshopping Checked: 1 / 1 Excluded: 0 / 1

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

```
-- 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
```

```
-- 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.
```

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

ApexSQL DBA

Source: http://www.apexsql.com

ApexSQL Audit:

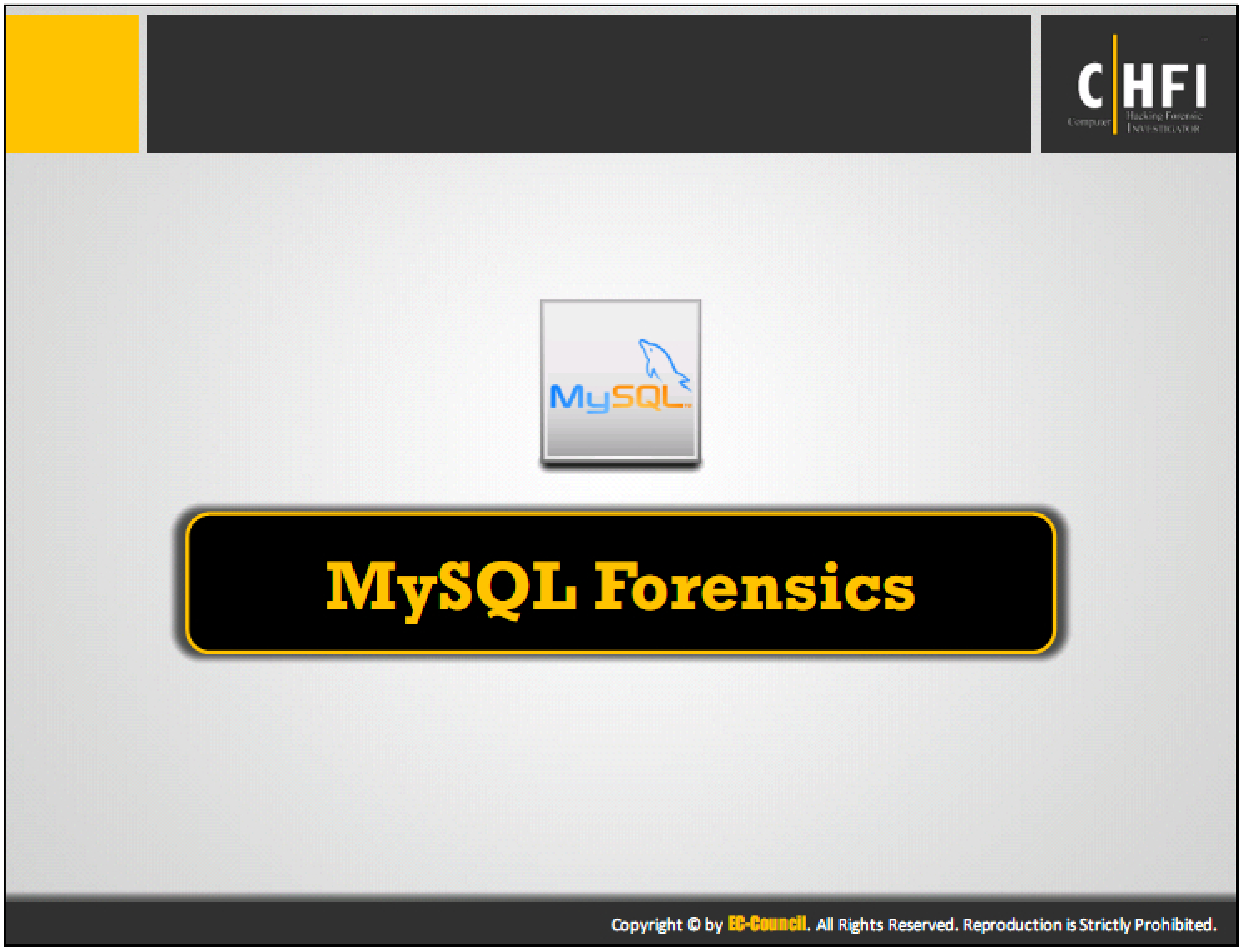
ApexSQL Audit is a SQL Server auditing tool, which provides auditing access, changes, and security on SQL Server instances, databases, and objects. It audits queries, DDL and DML operations, security events (authentication changes, permissions changes, and attempted logins), events on stored procedures and functions. ApexSQL Audit saves captured information in a centralized auditing repository and provides comprehensive reports.

Analyzing the volatile data with ApexSQL Audit helps forensic investigators gain insight on the login activities, the client connected to the server and the database on which the transactions occurred.

ApexSQL Log:

ApexSQL Log is an auditing and recovery tool for SQL Server databases that reads database transaction logs and audits, reverts or replays data and object changes affecting the database. It restores updated or missing data and objects, and captures information on the user, application, and host used to make each change.

Forensic investigators use SQL transaction log reader for forensic auditing and rollback of malicious or inadvertent database changes.



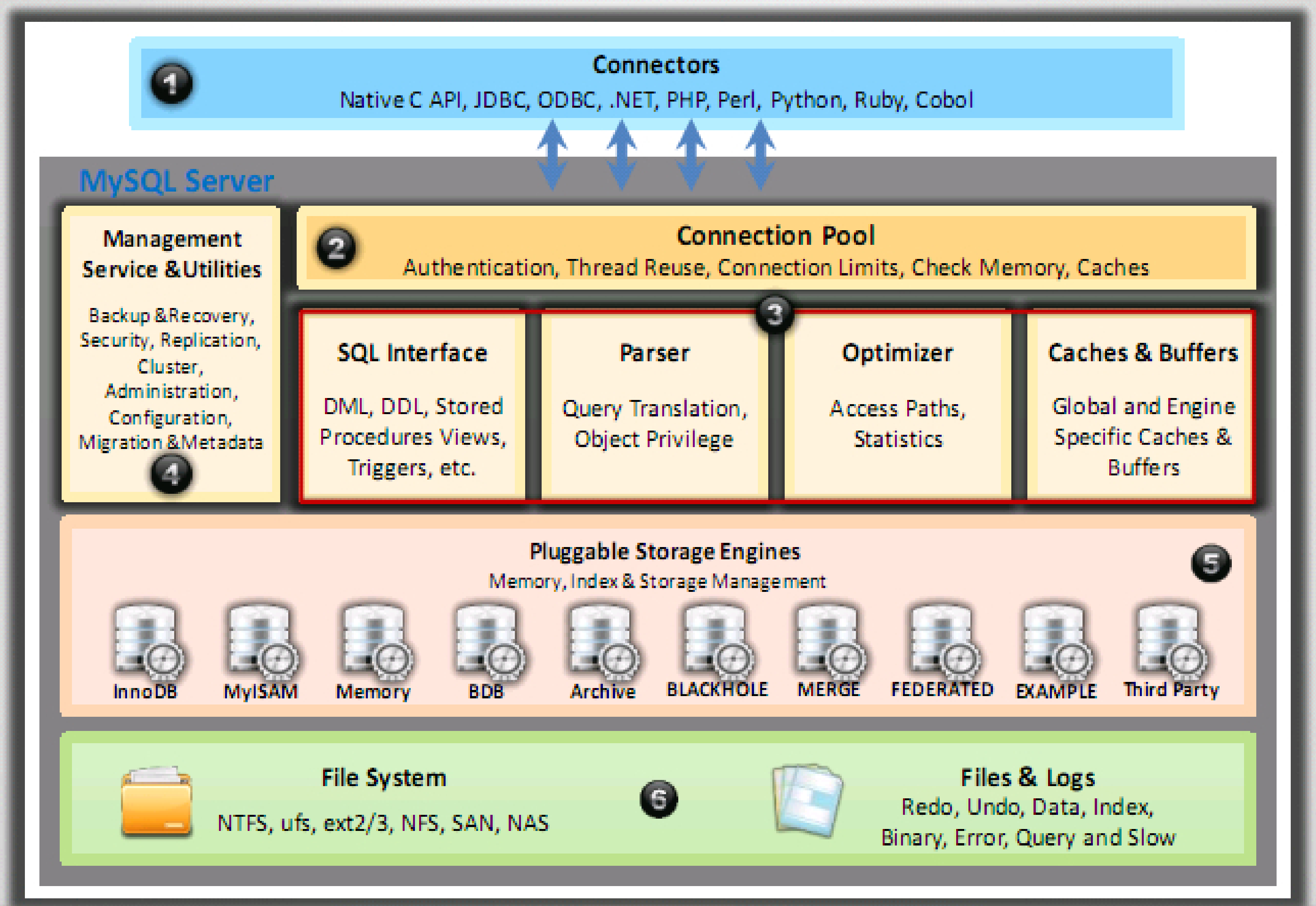
MySQL Forensics

MySQL database is one of the extensively used open source databases and freely available with unrestricted redistribution, providing users with full access to the source code. The database can contain different pluggable storage engines to suit the application. It supports transactions with the integration InnoDB or BDB storage engines for safer handling of parallel write operations required in enterprise environments.

As per the information security policies, administrators need to audit high-performance databases regularly to ensure the data integrity and data security. They should even be able to detect database manipulations.

Information auditing needs to be performed on regularly to find out if any part of the database is altered intentionally or accidentally by users at any point of time through bypassing auditing system. Such suspected behavior is to be inspected and analyzed by Database forensic investigators. The forensic approach for MySQL databases varies with the database engine used in the db server.

Internal Architecture of MySQL


1 Connectors
Native C API, JDBC, ODBC, .NET, PHP, Perl, Python, Ruby, Cobol
MySQL Server
Management Service & Utilities
Backup & Recovery, Security, Replication, Cluster, Administration, Configuration, Migration & Metadata
4
2 Connection Pool
Authentication, Thread Reuse, Connection Limits, Check Memory, Caches
3
SQL Interface
DML, DDL, Stored Procedures Views, Triggers, etc.
Parser
Query Translation, Object Privilege
Optimizer
Access Paths, Statistics
Caches & Buffers
Global and Engine Specific Caches & Buffers
Pluggable Storage Engines
Memory, Index & Storage Management
5
InnoDB
MyISAM
Memory
BDB
Archive
BLACKHOLE
MERGE
FEDERATED
EXAMPLE
Third Party
File System
NTFS, ufs, ext2/3, NFS, SAN, NAS
6
Files & Logs
Redo, Undo, Data, Index, Binary, Error, Query and Slow

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

The architecture of MySQL is based on a tiered architecture, which is the combination of subsystems and support components interacting with one another to read, analyze and execute the queries made to the database server, and return the results.

- 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 collectively form a key part of the database server.
  - The SQL Interface layer acts as an interface by accepting the SQL statements and delivers the result to the user. In spite of MySQL having the majority of the options that are not ANSI compliant, the layer supports the ANSI SQL standard. Whenever the client sends a request to the server through network communication pathways, the server creates a thread. Then the server analyzes the SQL command and stores the parts in an internal data structure.
  - Parser validates the SQL queries entered by a user. It constructs a query structure, which denotes the query statement (SQL) within the memory as a tree structure. This tree structure consists of a group of tables and field names referenced, join conditions, etc., and its main task is to check the accuracy of SQL statement. Once the SQL statement is validated, the Parser passes the query structure to the query processor and then, to the query optimizer.
  - The Query Optimizer's primary task is to validate the existence of tables, and the level of access to the application for a user. The Optimizer then returns the errors (if any) and the control returns to the thread manager or listener. The optimizer works on the SELECT-PROJECT-JOIN strategy that attempts to restructure the query.
  - A set of library methods designed to implement a specific query also control its execution and return the results from each of the execution methods using the network communication pathways library.
  - The query cache plays a vital role in query optimization and execution subsystem. It caches the query structure as well as the query results.
  - Caches and buffers ensure that commonly used data are made available in an efficient way. MySQL database server contains various types of caches, including table cache, key cache, privilege cache, Hostname Cache, Record Cache; and miscellaneous caches such as join buffer cache.

1. Storage engines are used to create, read, and update data within a database. Various Storage engines supported by MySQL are:
   a. **InnoDB (default storage engine in MySQL version >= 5.5.5)**
      - Is often used when you need to use transactions

- Supports traditional ACID (Atomicity, Consistency, Isolation, Durability) transactions and foreign key constraints
- Provides 64TB storage limit per table
- Offers MVCC/Snapshot reads
- Supports crash recovery
- Used in On-Line Transaction Processing (OLTP) systems

b. **MyISAM (default storage engine in MySQL version < 5.5.5)**

- Provides unlimited data storage
- Offers data compression technique
- Handles high-speed data loads
- Enables efficient storage

c. **Memory**

- It is an in-memory table that Implements a hashing mechanism for faster retrieval of commonly used data
- Offer validity for the data stored in memory only during the MySQL session

d. **BDB**

- Stands for Berkeley database
- Considered an alternative to InnoDB
- Supports additional transaction methods such as COMMIT and ROLLBACK

e. **ARCHIVE**

- Provides unlimited storage limit in a compressed format
- Well known for storing and retrieving huge volumes of seldom-accessed archival or historical data
- Supports automatic data compression
- Offers MVCC (Multiversion concurrency control)/Snapshot read
- Limits operations to INSERT and SELECT

f. **BLACKHOLE**

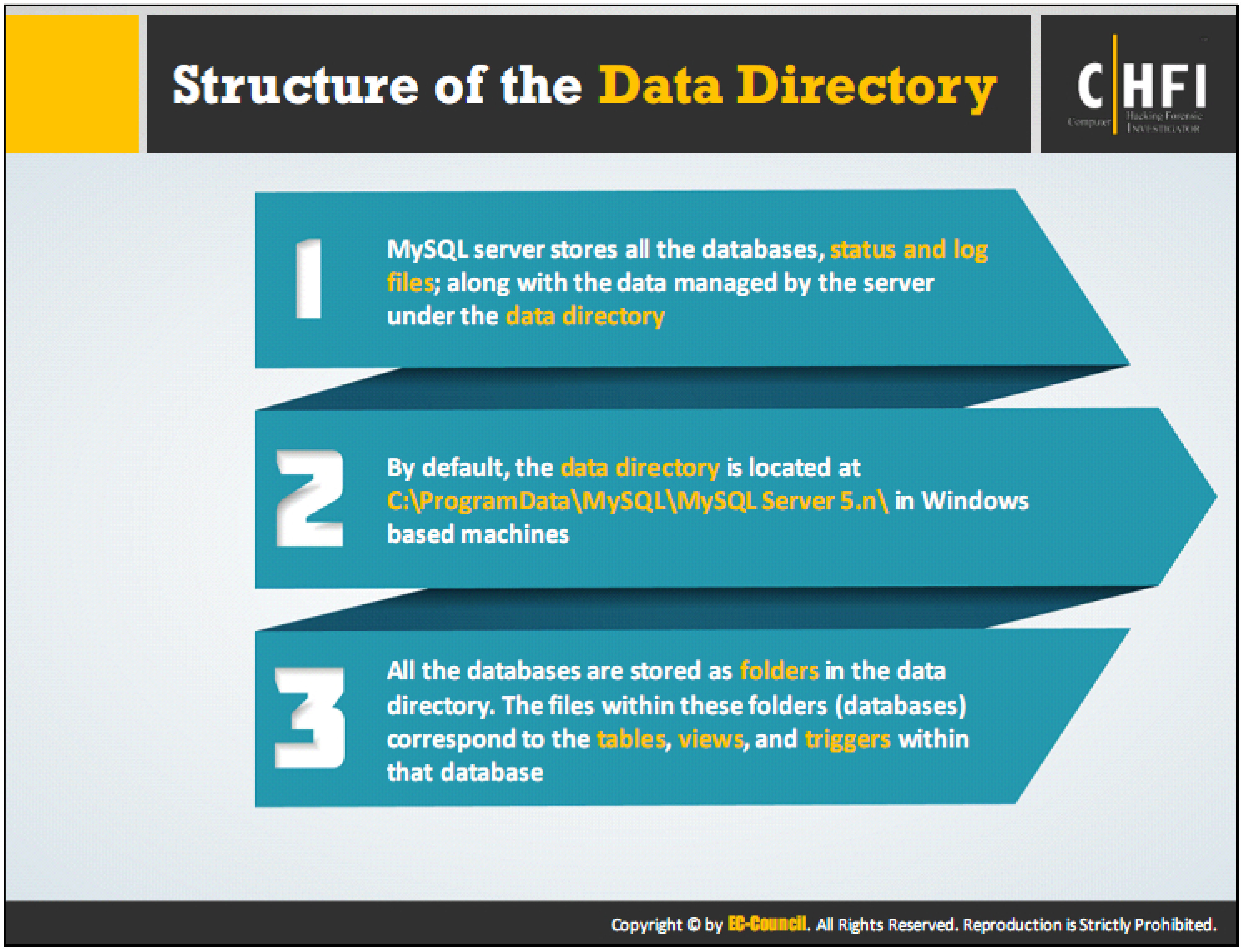
- Allows the system to write the data, however, that data is never saved
- Writes SQL statements to logs when binary logging is enabled
- Allows Database administrators to disable data injection in the database temporarily by swapping the table type

g. **MERGE**

- Built using a series of MyISAM tables (with tuple layout or schema)
- Ensures that all the tables reside on the same system
- Blocks the replace operation
- Performs search and sort options more quickly
- Allows users to access data using SELECT, UPDATE, INSERT, and DELETE operations
- Widely used in very large database (VLDB) applications like data warehousing

h. **FEDERATED**

- Creates a single table reference from one or more database systems
- Hence, similar to Merge, but allows data/tables linkage across database servers
- Enables data translation during storage and retrievals
- Supports all SQL operations
- Does not require middleware for remote data access
- Extensively used in distributed or data mart environments



Structure of the Data Directory

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

The data directory contains databases, tables and status files handled by the server. These data directories are organized in a tree-like structures by following hierarchical structure of the Unix or Windows file systems:

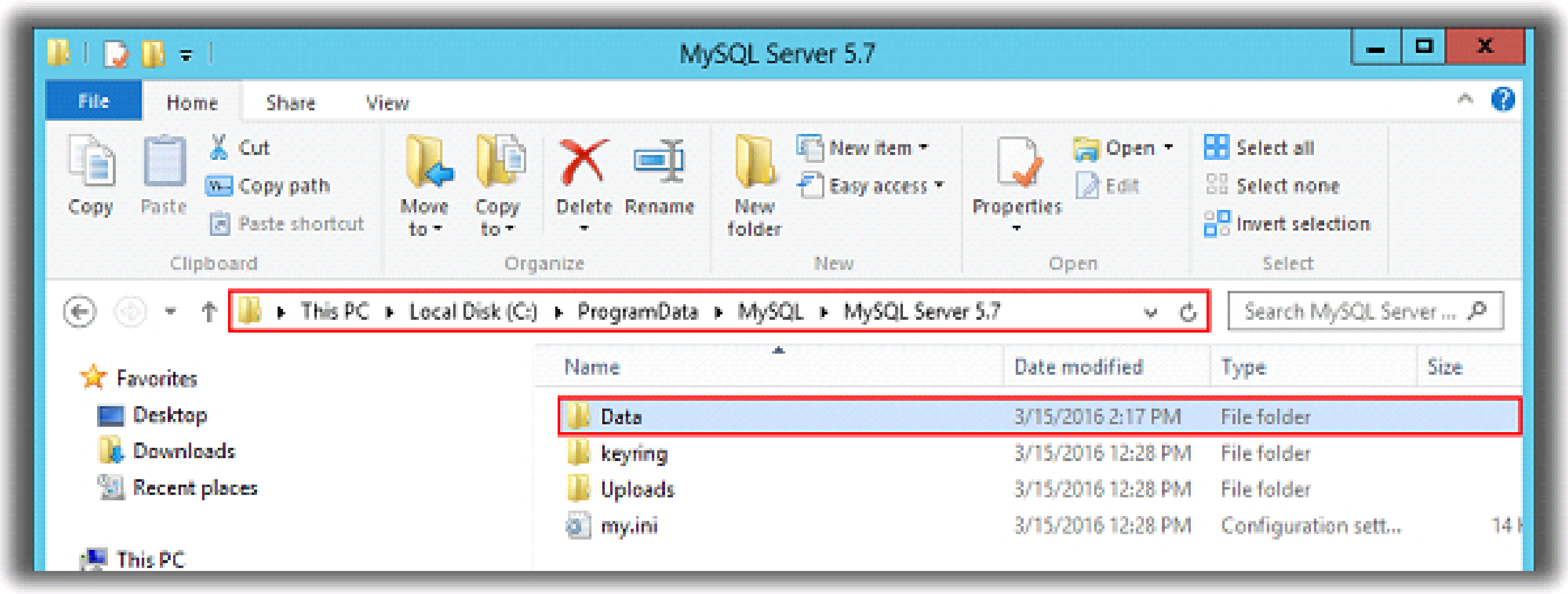
- Every database corresponds to a directory under the data directory
- The tables of a database, correspond to the files of database directory

The default path to the data directory is mentioned below for the windows based machines

C:\ProgramData\MySQL\MySQL Server 5.n

(or)

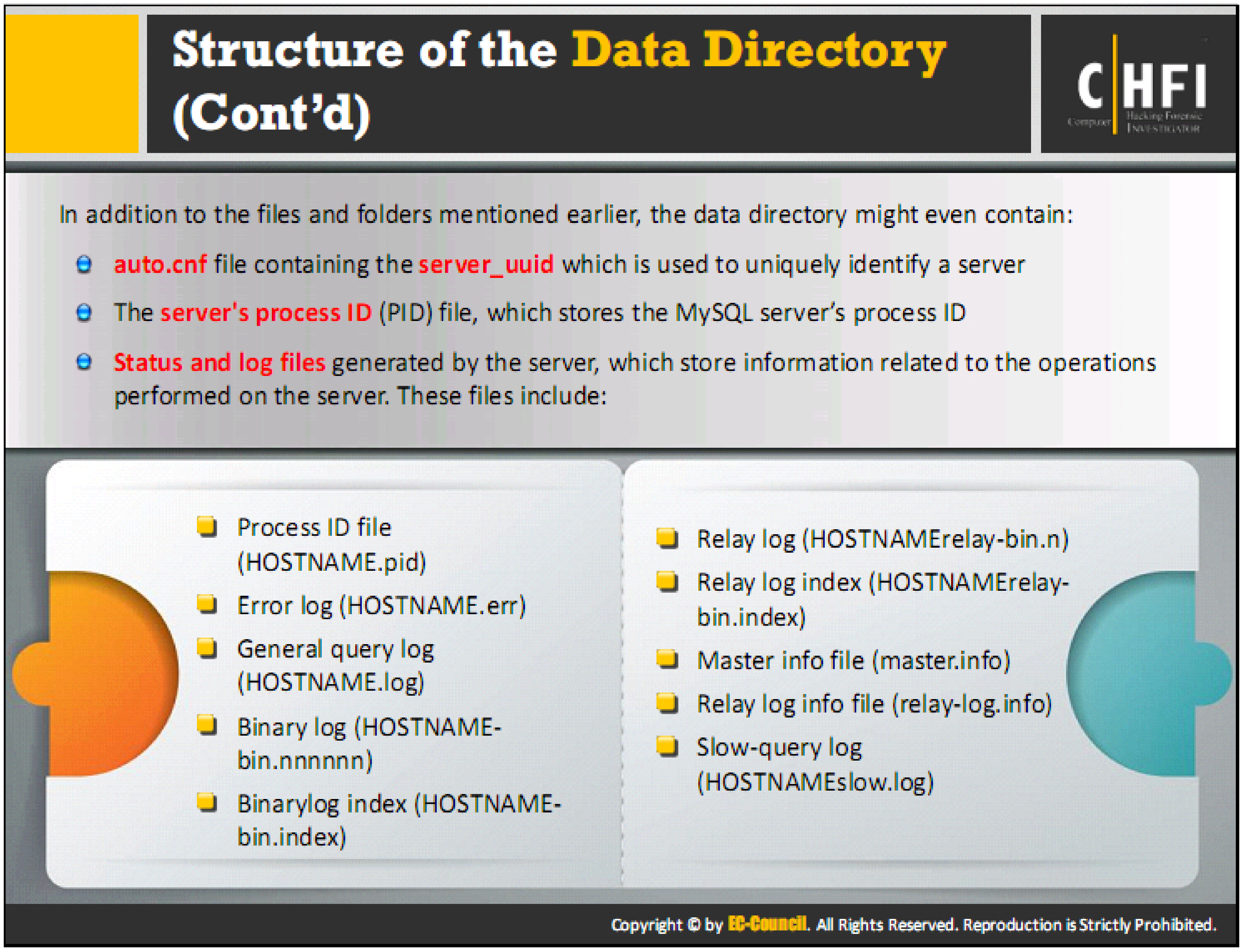
C:\mysql\data.



This PC ▸ Local Disk (C:) ▸ ProgramData ▸ MySQL ▸ MySQL Server 5.7

Name	Date modified	Type	Size
Data	3/15/2016 2:17 PM	File folder	
keyring	3/15/2016 12:28 PM	File folder	
Uploads	3/15/2016 12:28 PM	File folder	
my.ini	3/15/2016 12:28 PM	Configuration sett...	14

FIGURE 9.1: Data Directory in MySQL Server



Structure of the Data Directory (Cont'd)

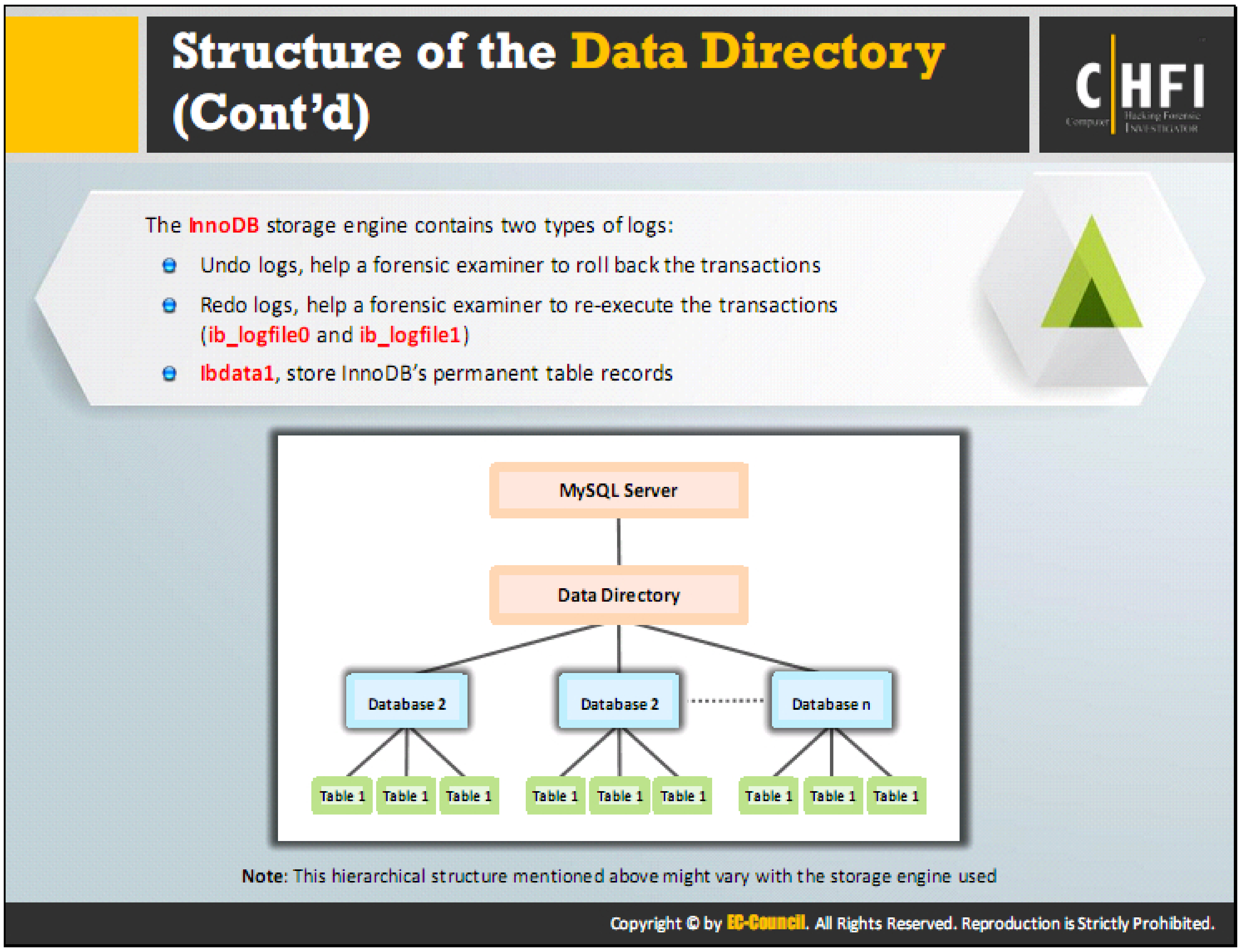
In addition to the files and folders mentioned earlier, the data directory might even contain:

- **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)

Status and log files stored in data directory include:

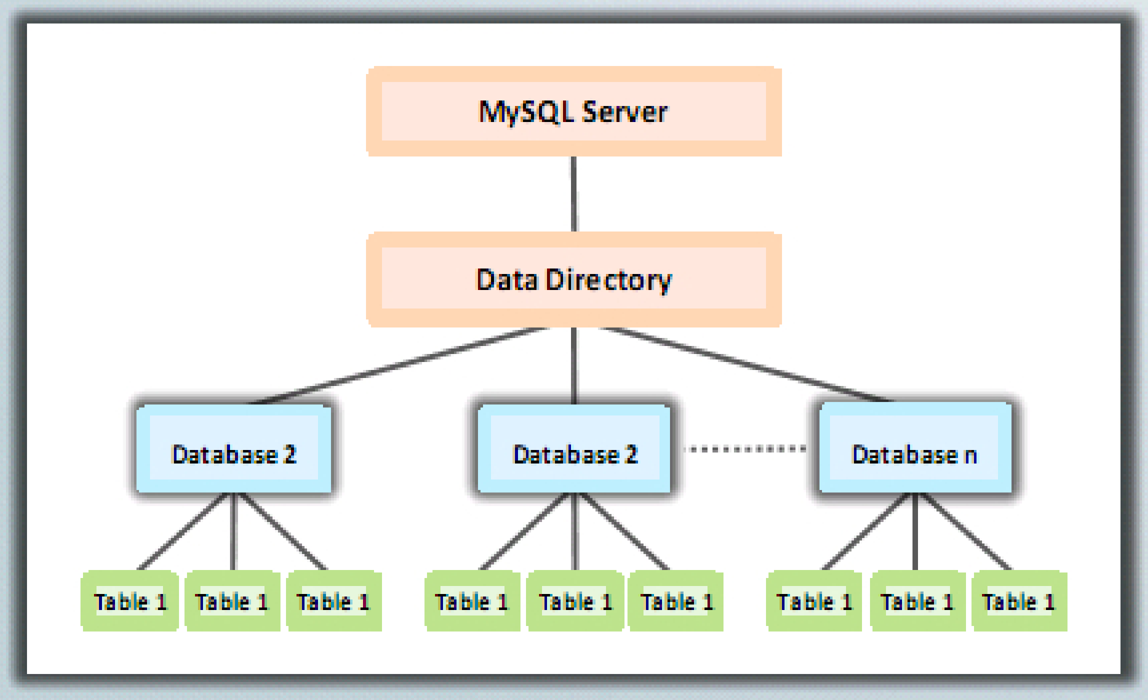
1. Process ID file (HOSTNAME.pid), contains the process ID created when the server starts
2. Error log (HOSTNAME.err), contains the information associated with the startup and shutdown events, and errors
3. General query log (HOSTNAME.log), logs the client connections and activities
4. Binary log (HOSTNAME-bin.nnnnnn), contains the events that describe the changes occurred in the database
5. Binary log index (HOSTNAME-bin.index), contains the list of all the binary log files currently available in the data directory
6. Relay log (HOSTNAMErelay-bin.n), contains the events that describe the changes occurred in the database
7. Relay log index (HOSTNAMErelay-bin.index), contains the list of all the relay log files currently available in the data directory
8. Master info file (master.info) created by a replication slave server, that contains the essential parameters used for connecting to the master slave
9. Relay log info file (relay-log.info) created by a replication slave server, that contains the status of relay log processing
10. Slow query log (HOSTNAMEslow.log), a text file that contains statements which take longer processing time



Structure of the Data Directory (Cont'd)

The InnoDB storage engine contains two types of logs:

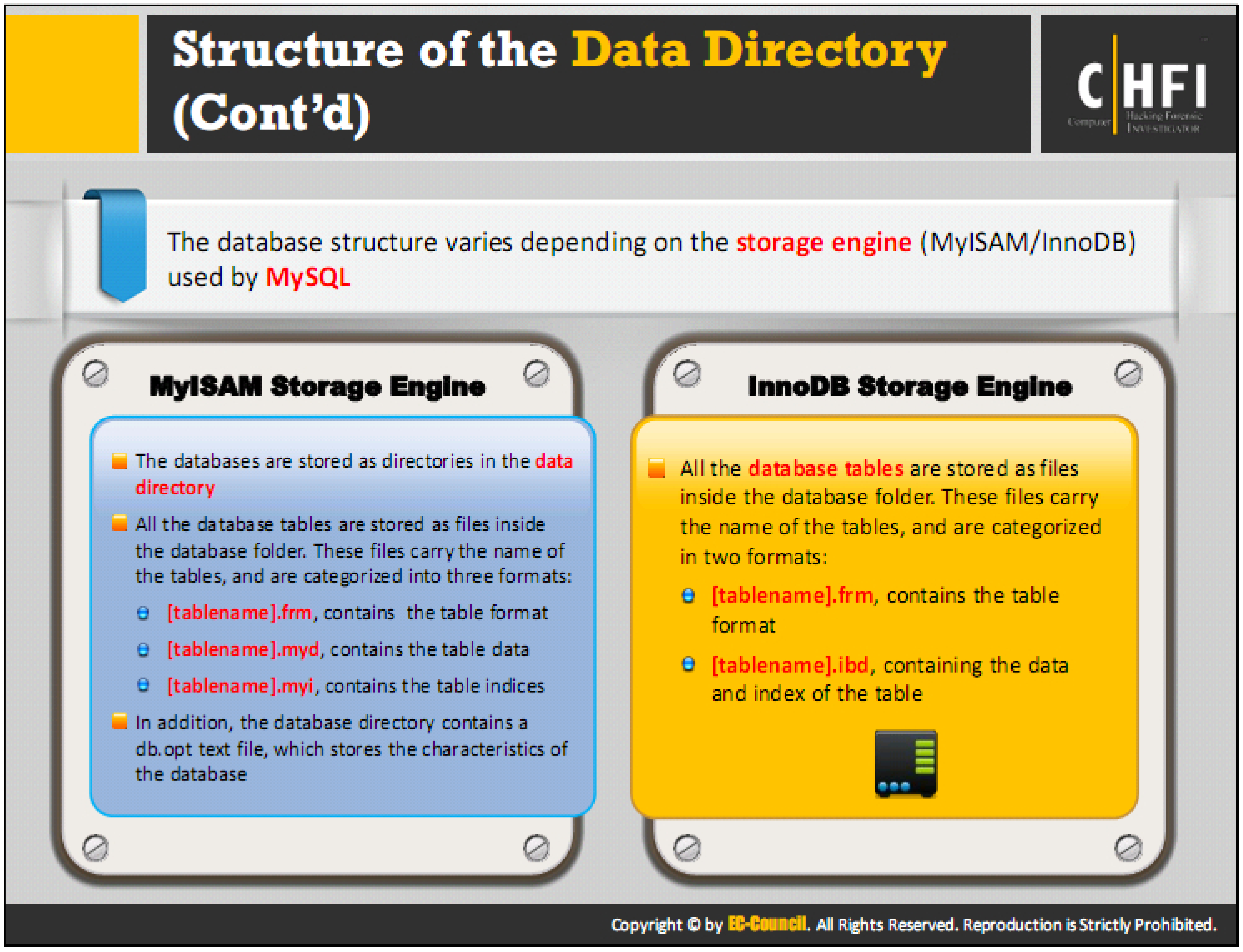
- Undo logs, help a forensic examiner to roll back the transactions
- Redo logs, help a forensic examiner to re-execute the transactions (ib_logfile0 and ib_logfile1)
- Ibdata1, store InnoDB's permanent table records



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

MySQL data directories contain all the databases managed by the servers. Their arrangement is shown in the screenshot below.

- Every database corresponds to a directory under the data directory
- Tables, views, and triggers of a database, correspond to the files of database directories
- The storage structures vary from the hierarchical implementation of databases.



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

Database structures alter as per the storage engines used by MySQL.

MyISAM Storage Engine

In a non-partitioned filesystem, MyISAM storage engine stores each MyISAM table in three files. The files contain names that start with the table name and will have an extension to indicate the file type. The file extensions are .frm, .MYI and .MYD. The .frm file stores the table format, the .MYI file stores the file index, while the .MYD file stores table data.

InnoDB Storage Engine

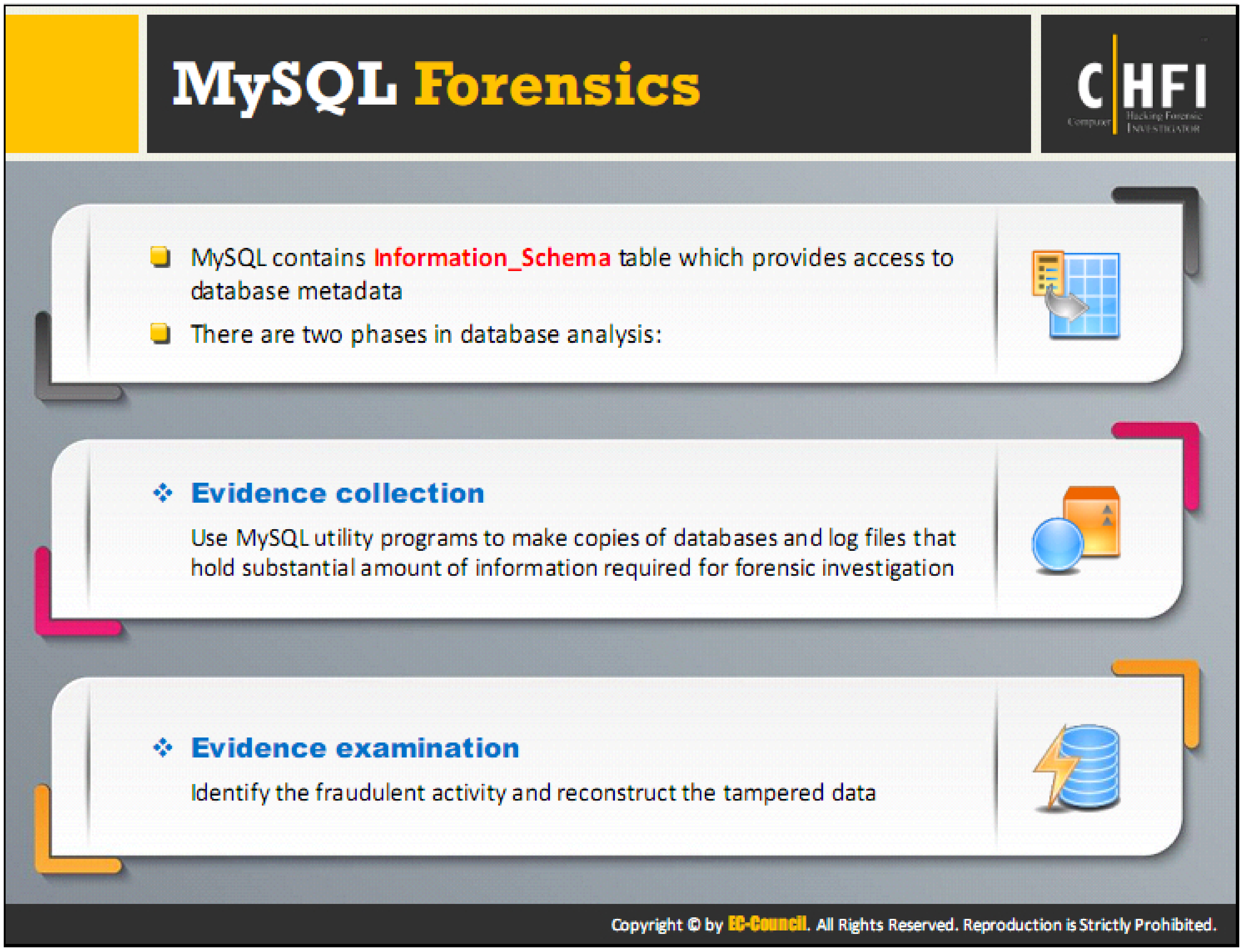
Table contents of InnoDB are represented by using tablespaces, and are of two types:

- **The shared tablespace**

  Users can configure InnoDB to use one tablespace file per table. In this shared tablespace, each InnoDB table contains two table-specific files in the database directory: The .frm file (as usual) and a .ibd file that contains the table's data and indexes.

- **Individual tablespaces**

  Users can configure InnoDB to use one tablespace file per table. In this individual tablespace, each InnoDB table has two table-specific files in the database directory: The .frm file (as usual) and a .ibd file that contains the table's data and indexes.



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

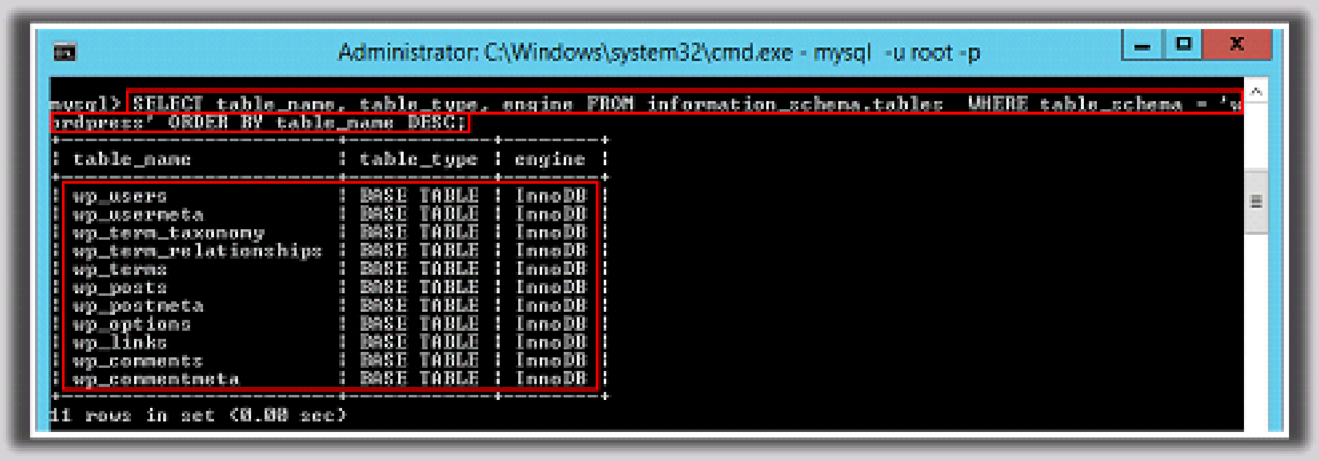
Identify the fraudulent activity and reconstruct the tampered data

MySQL is an open source relational database. Data entered in a MySQL database is duplicated and stored in multiple locations. Therefore, any users deleting data in the database either accidentally or intentionally will not completely delete the data. Forensic investigators can examine all the files containing a copy of the deleted data (in the data folder) and recover it.

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**

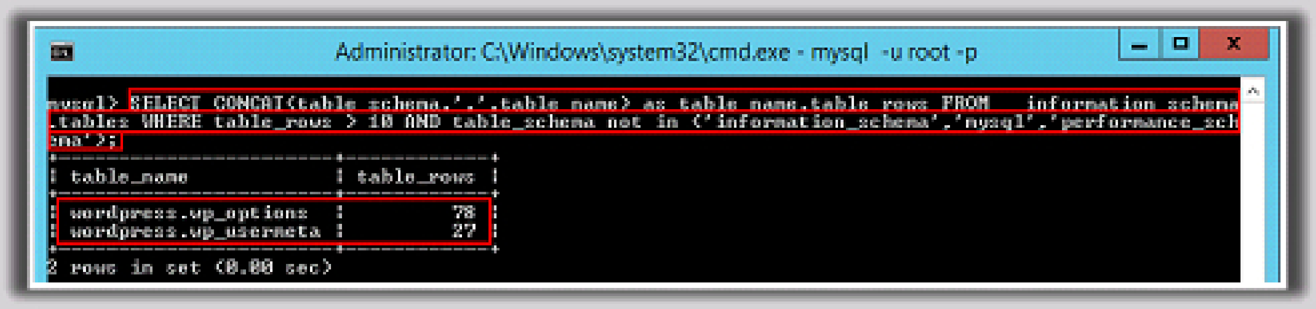
E.g. 1: Viewing a list of all the tables in "**wordpress**" database in reverse alphabetic order, displaying the table name, table type, and storage engine used for the database



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

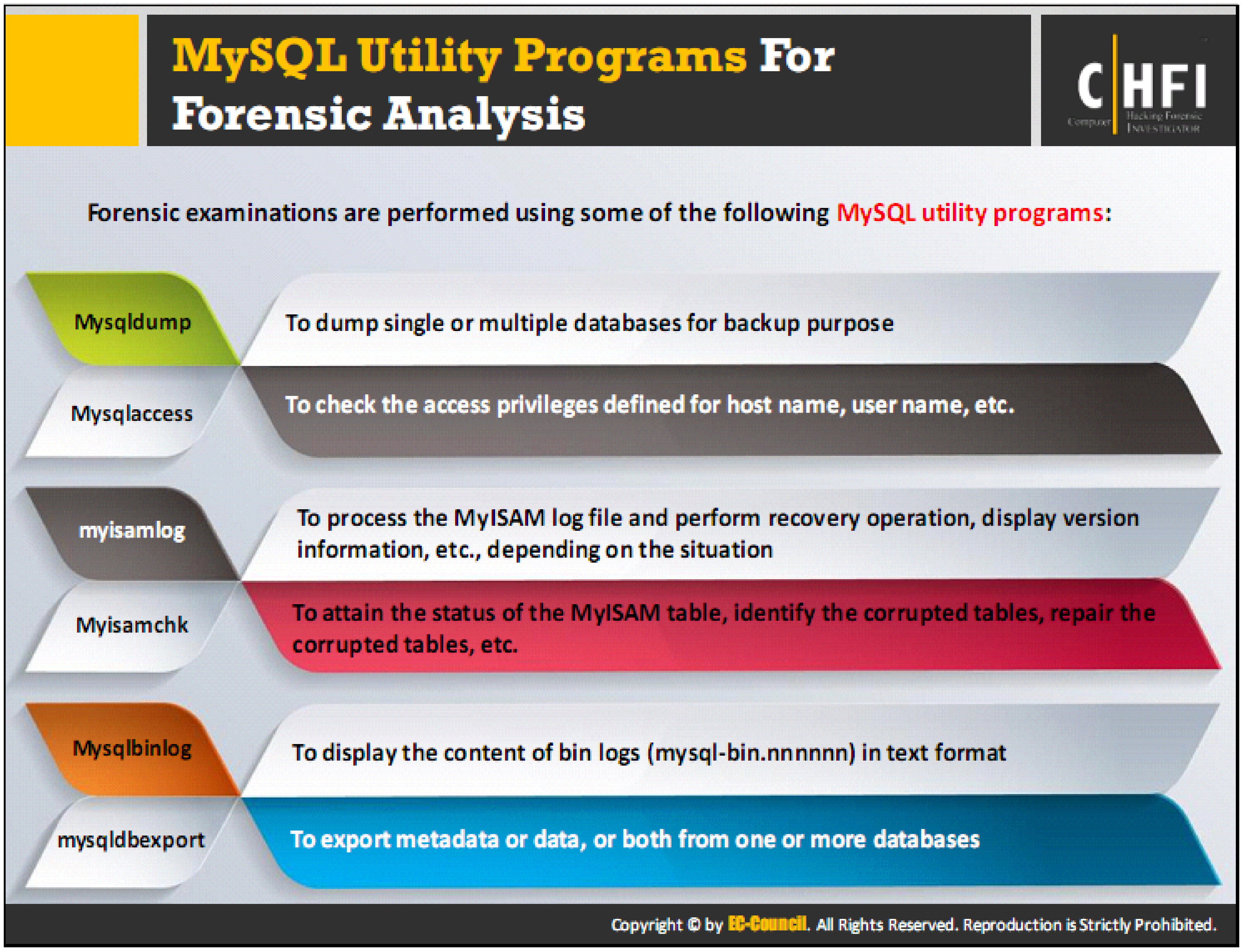
```
mysql> SELECT table_name, table_type, engine FROM information_schema.tables  WHERE table_schema = 'w
ordpress' ORDER BY table_name DESC;
+------------------------+------------+--------+
| table_name             | table_type | engine |
+------------------------+------------+--------+
| wp_users               | 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 (0.08 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

The following utility programs help investigators to perform the forensic examination.

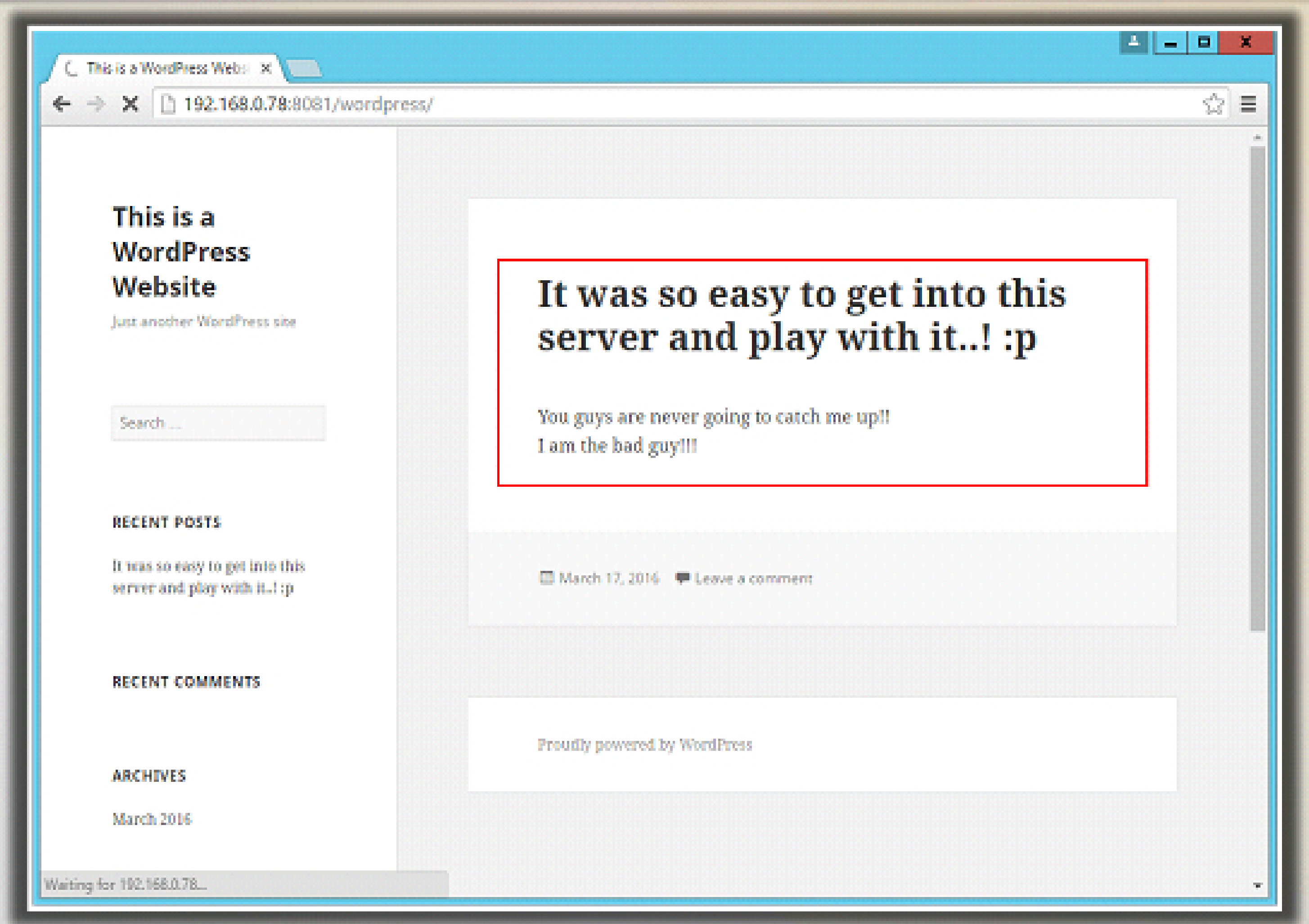
- **Mysqldump**
  - The utility allows you to dump a database or a collection of databases for backup purposes
  - It generates a .sql file with CREATE table, DROP table and INSERT into the SQL statement of the source database
  - It executes the .sql file on the destination database to restore the original database.
  - Syntax: `mysqldump [options] [db_name [tbl_name ...]]`
- **mysqlaccess**
  - Checks the access privileges defined for host name, user name, etc.
  - Validates access using the user, db, and host tables
  - Syntax: `mysqlaccess [host_name [user_name [db_name]]] [options]`
- **myisamlog**
  - Processes the contents of MyISAM log file and perform recovery operation, display version information, etc., depending on the situation
  - The default operations of the utility are update(-u) and recovery(-r)

  - Syntax: `myisamlog [options] [logfile-name [tbl_name] ...]`

- **myisamchk**
  - Views the status of the MyISAM table or checks, repairs, or optimizes them.
  - Syntax: `myisamchk [options] tbl_name ...`
- **mysqlbinlog**
  - Reads the binary log files directly and displays them in text format
  - Displays the content of bin logs (mysql-bin.nnnnnn) in text format
  - Syntax: `mysqlbinlog [options] log-file ...`
- **mysqldbexport**
  - Export metadata/data definitions
  - Produces output in a variety of formats by making data extraction easier and suitable for the external application
  - Syntax: `mysqldbexport --server=user:pass@host:port:socket db1, db2, db3`

Common Scenario for Reference

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


This is a WordPress Website
Just another WordPress site

RECENT POSTS
It was so easy to get into this server and play with it..! :p

RECENT COMMENTS

ARCHIVES
March 2016

It was so easy to get into this server and play with it..! :p

You guys are never going to catch me up!!
I am the bad guy!!!

March 17, 2016 Leave a comment

Proudly powered by WordPress

Administrators often manage WordPress websites' databases using MySQL and phpMyAdmin. Sometimes, they may not follow security best practices, such as updating the WordPress plugins, implementing strong passwords for MySQL logins, etc., leaving doors for attackers to gain access to the web application, or sometimes, the Relational Database Management Systems (RDBMS) as well.

Here, we are considering a WordPress website containing a suspicious post on the index page of the website. It is not evident whether someone gained unauthorized access to the WordPress web application, or gained access to the MySQL database.

Therefore, we have addressed two scenarios, wherein, the first scenario deals with identifying the attacker and collecting all the posts attacker made on the WordPress website, while the second scenario deals with recovering the deleted data.

MySQL Forensics for WordPress Website Database: Scenario 1

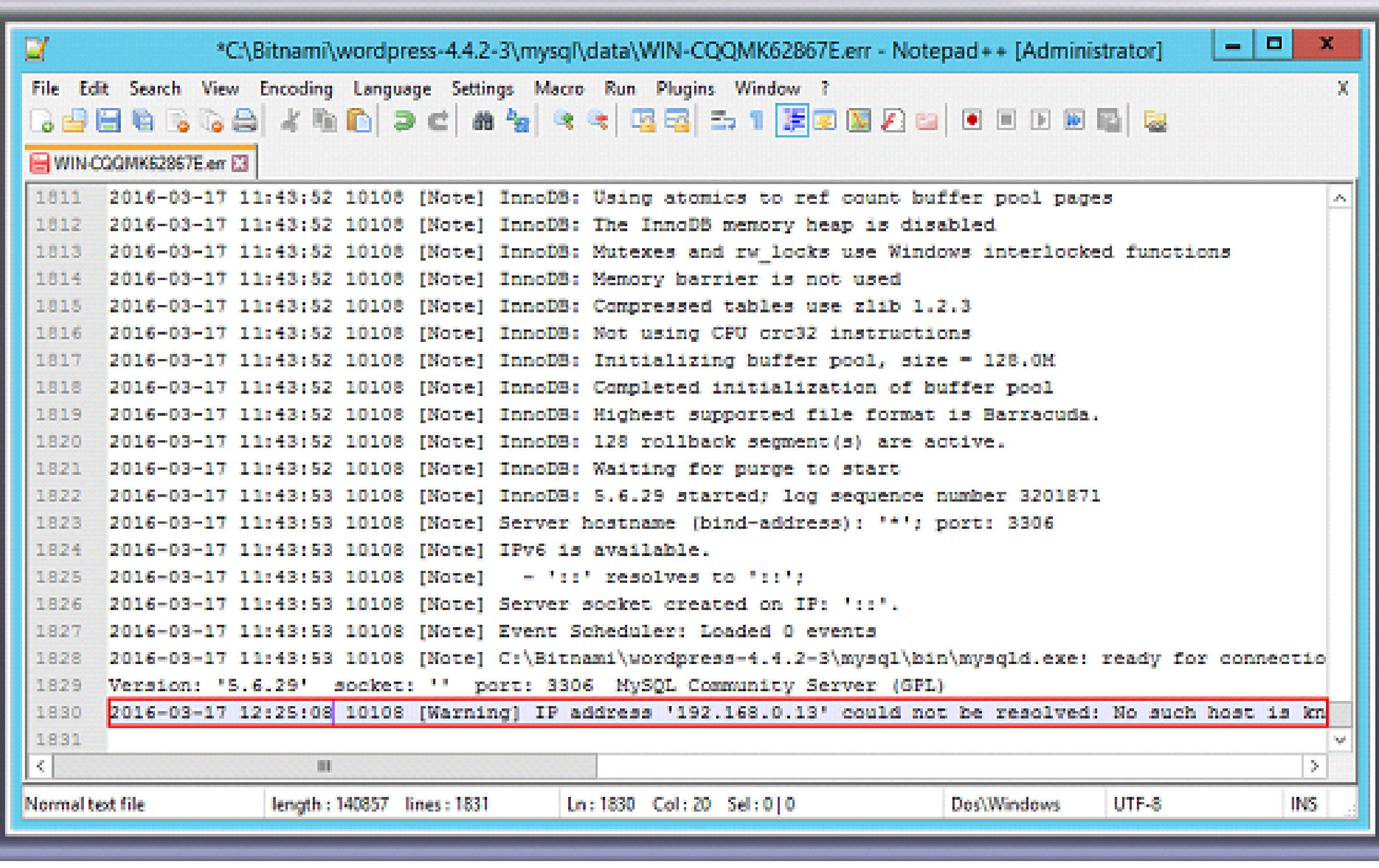
Identifying a malicious user and collecting all the posts made by him/her through command line client

Whenever a suspicious activity occurs on the MySQL database of a web application, the error log, and general query log files act as the main evidence for an investigator, to examine the transactions performed on a database.

In this scenario, first, we will be examining the error logs and later, we shall analyze the transactions occurred on the databases using 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

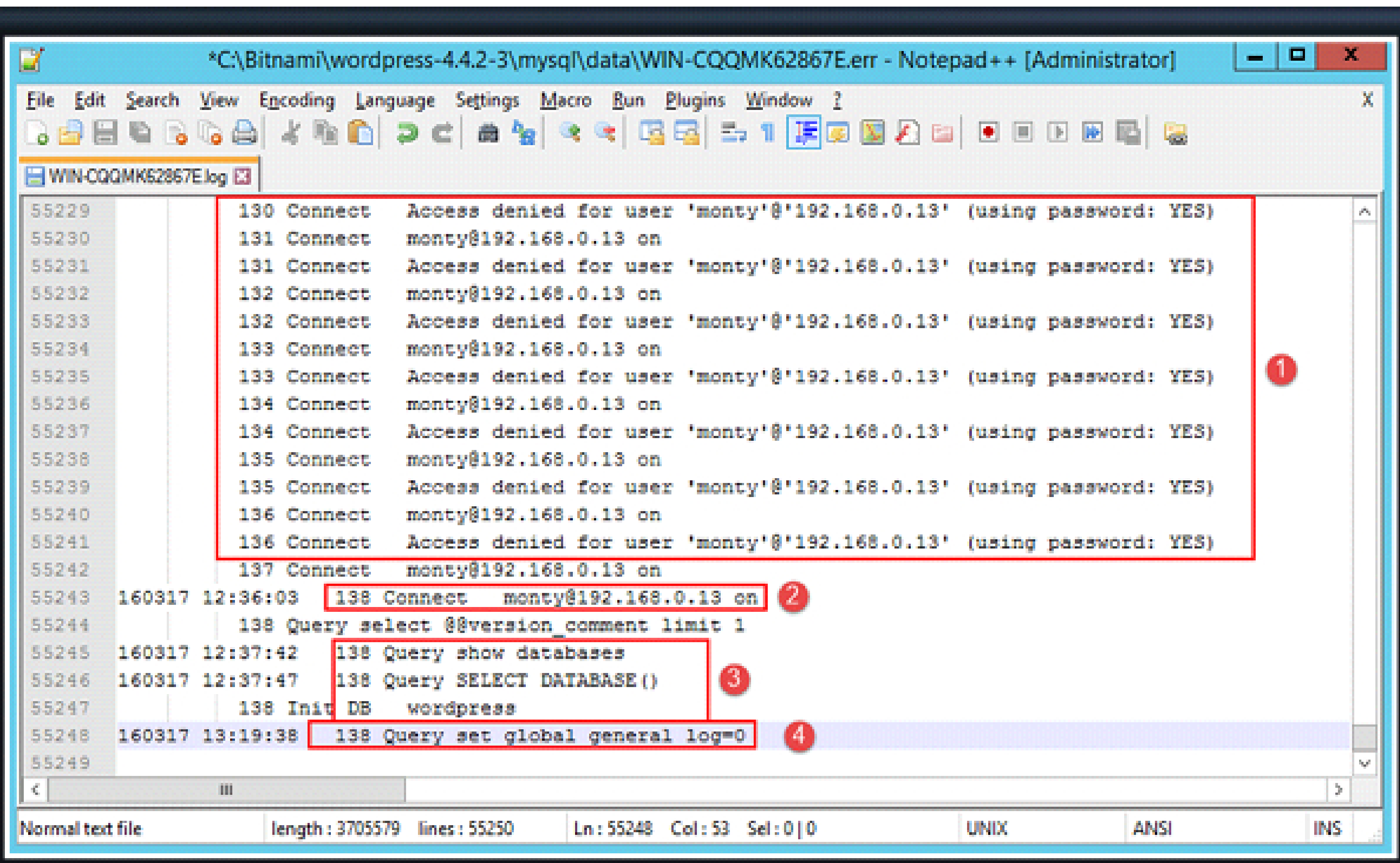
```
2016-03-17 11:43:52 10108 [Note] InnoDB: Using atomics to ref count buffer pool pages
2016-03-17 11:43:52 10108 [Note] InnoDB: The InnoDB memory heap is disabled
2016-03-17 11:43:52 10108 [Note] InnoDB: Mutexes and rw_locks use Windows interlocked functions
2016-03-17 11:43:52 10108 [Note] InnoDB: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

```
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
1
2
3
4

```
                    130 Connect   Access denied for user 'monty'@'192.168.0.13' (using password: YES)
                    131 Connect   monty@192.168.0.13 on
                    131 Connect   Access denied for user 'monty'@'192.168.0.13' (using password: YES)
                    132 Connect   monty@192.168.0.13 on
                    132 Connect   Access denied for user 'monty'@'192.168.0.13' (using password: YES)
                    133 Connect   monty@192.168.0.13 on
                    133 Connect   Access denied for user 'monty'@'192.168.0.13' (using password: YES)
                    134 Connect   monty@192.168.0.13 on
                    134 Connect   Access denied for user 'monty'@'192.168.0.13' (using password: YES)
                    135 Connect   monty@192.168.0.13 on
                    135 Connect   Access denied for user 'monty'@'192.168.0.13' (using password: YES)
                    136 Connect   monty@192.168.0.13 on
                    136 Connect   Access denied for user 'monty'@'192.168.0.13' (using password: YES)
                    137 Connect   monty@192.168.0.13 on
160317 12:36:03     138 Connect   monty@192.168.0.13 on
                    138 Query select @@version_comment limit 1
160317 12:37:42     138 Query show databases
160317 12:37:47     138 Query SELECT DATABASE()
                    138 Init DB   wordpress
160317 13:19:38     138 Query set global general log=0
```

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

Scenario 1: Analyze the General Log

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

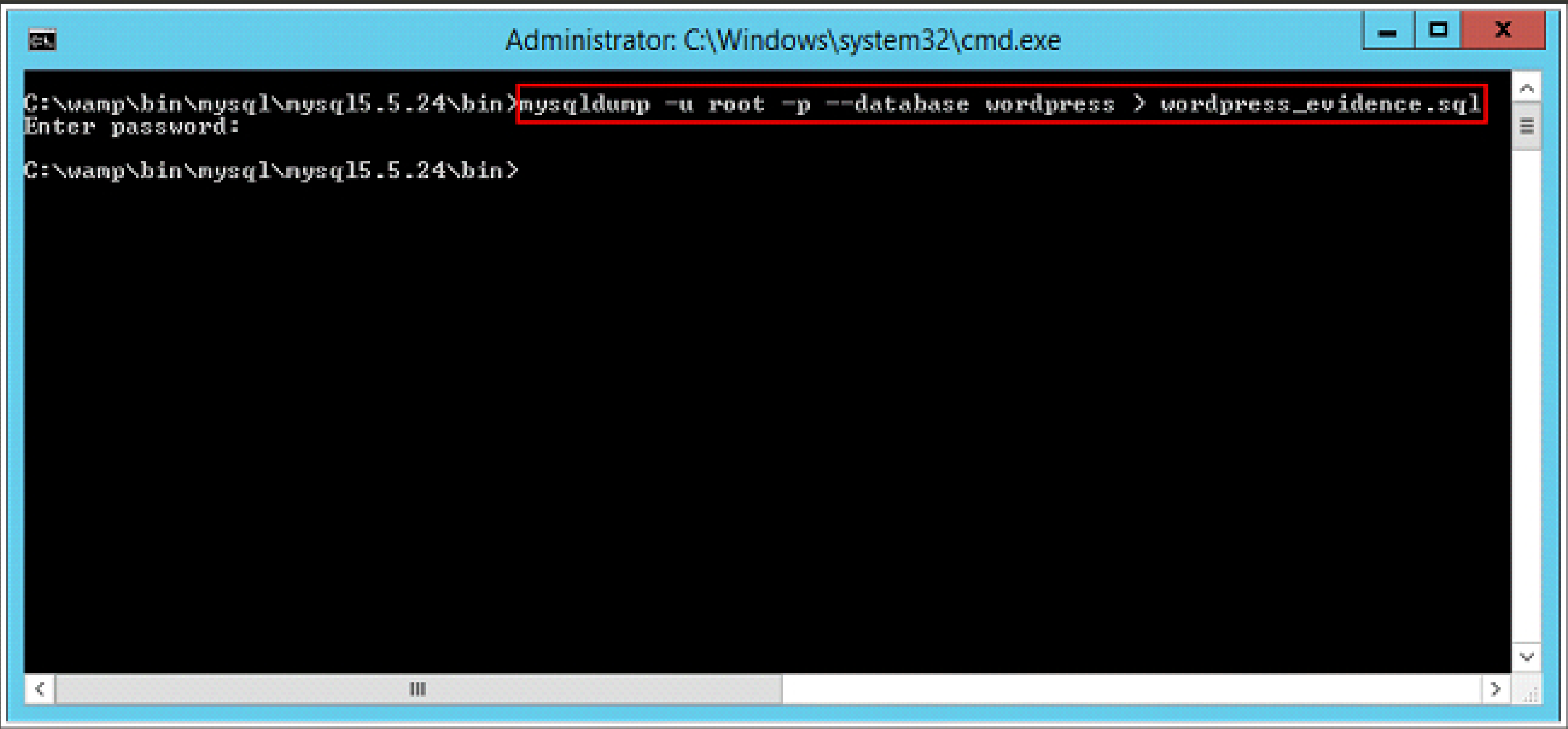
02 A connection was established from the attacker address 192.168.0.13

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

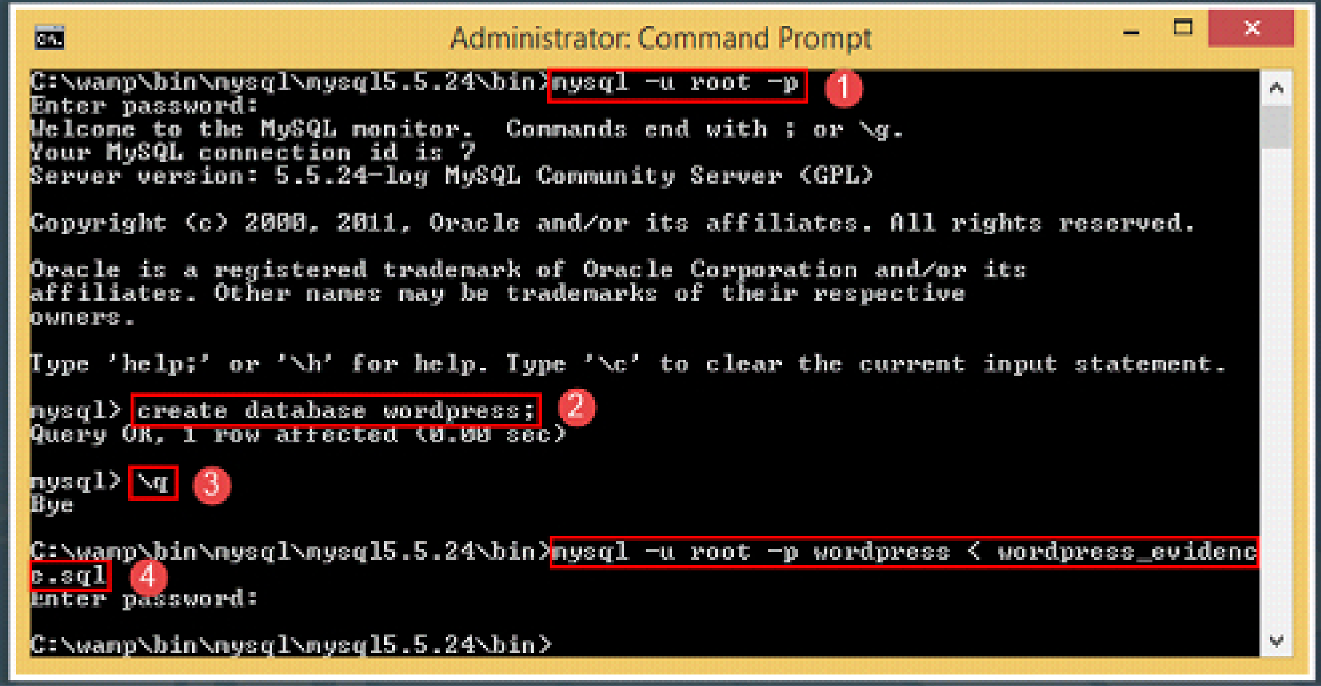


```
C:\wamp\bin\mysql\mysql5.5.24\bin>mysqldump -u root -p --database wordpress > wordpress_evidence.sql
Enter password:

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

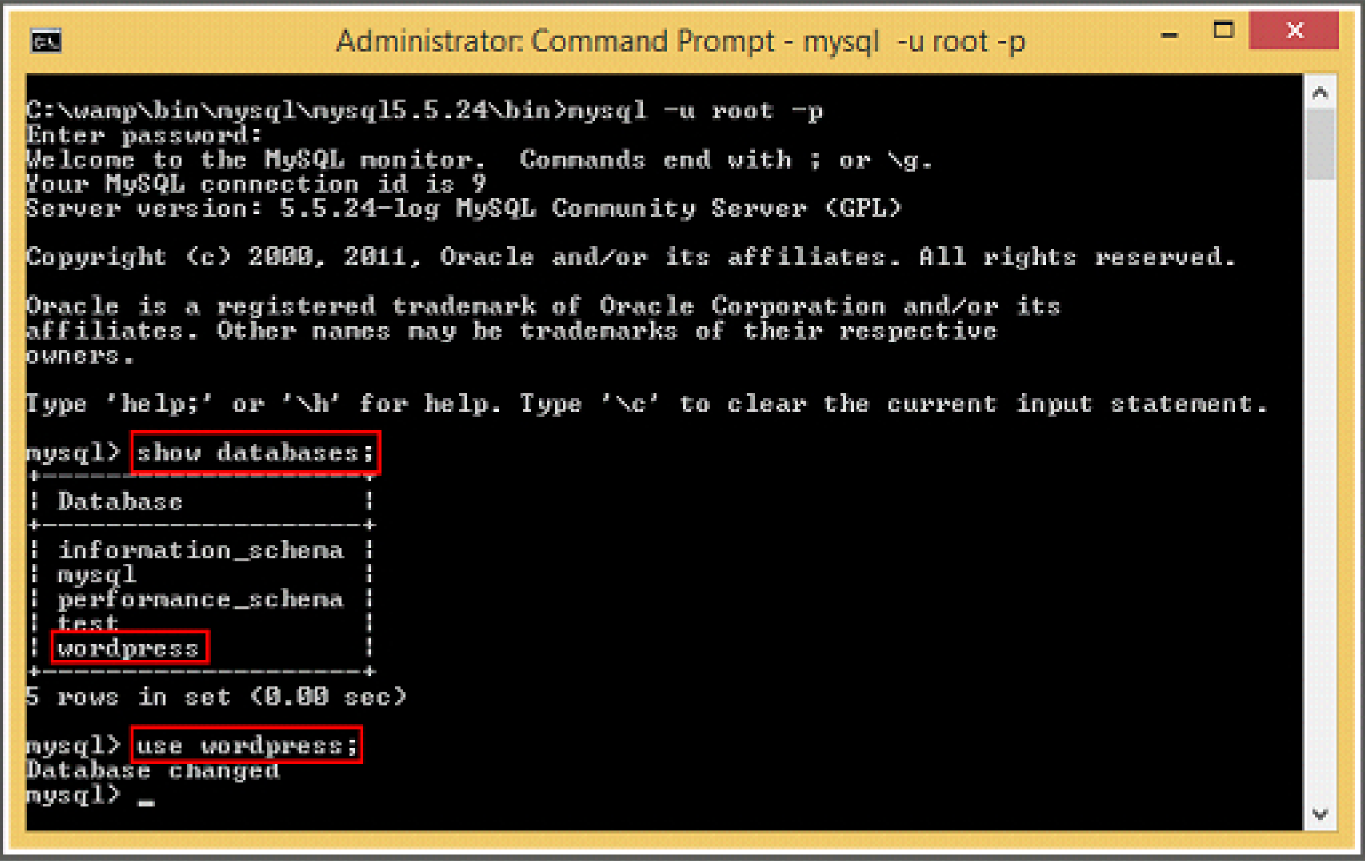
- Log in to mysql server in the forensic examiner's machine
- Create a database with the same name as that in the affected machine
- Exit the mysql terminal
- Copy all the contents of the dump file to the newly created database


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

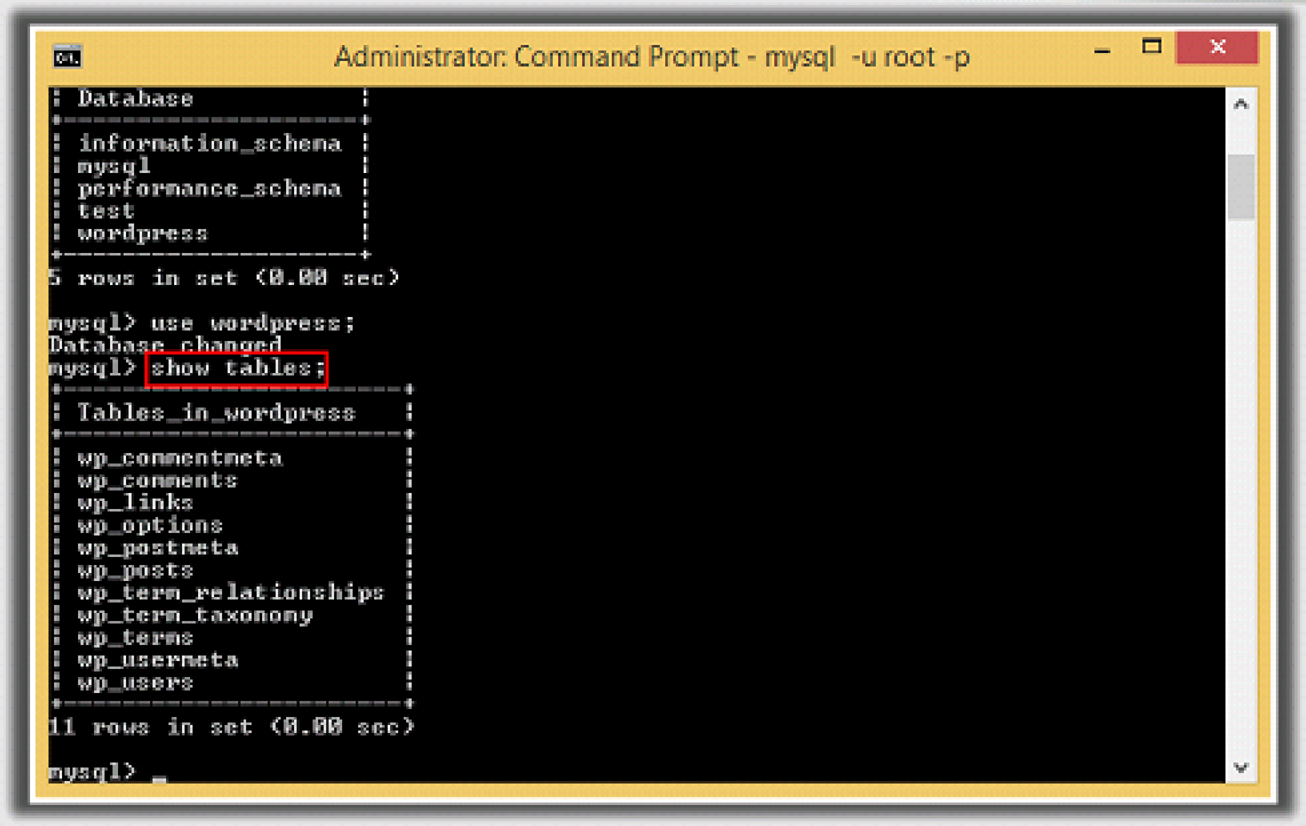

Administrator: Command Prompt - mysql -u root -p

```
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

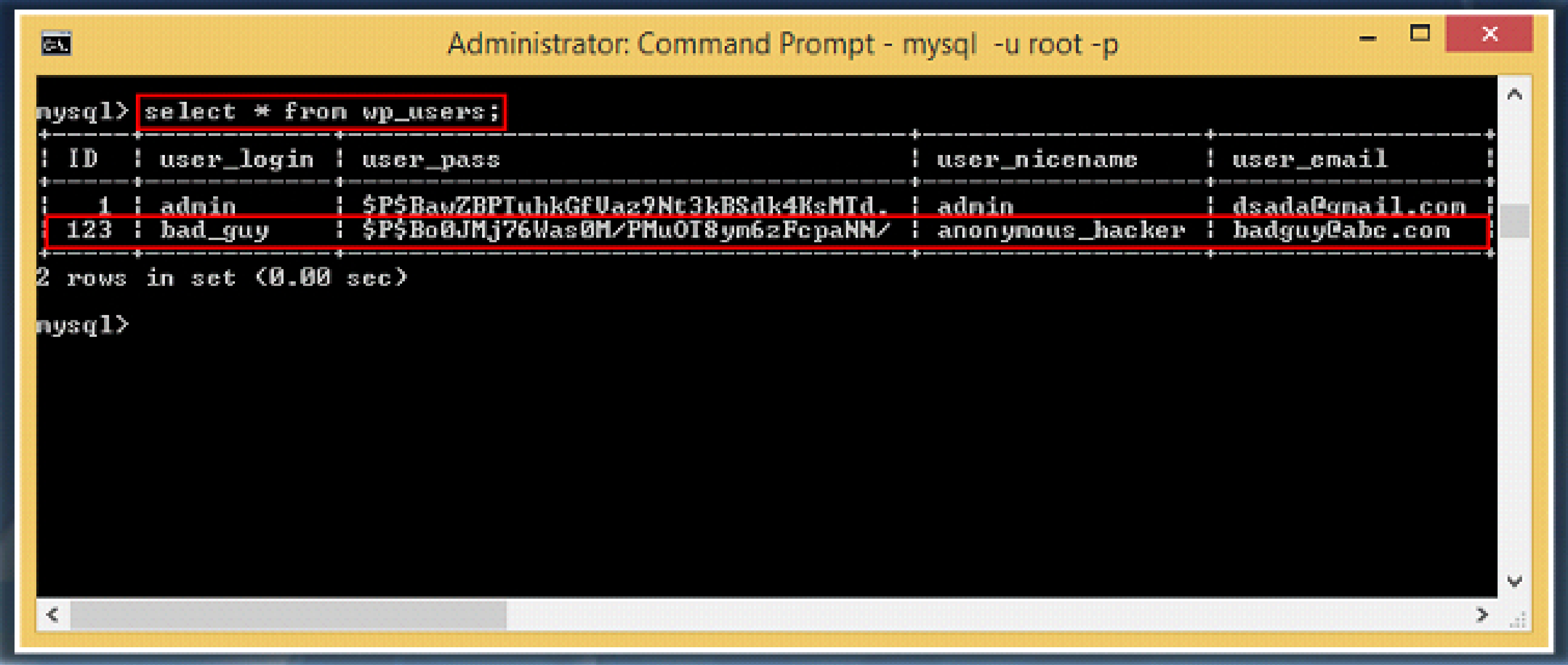
2 In this scenario, all the tables appear to be present in the database



```
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



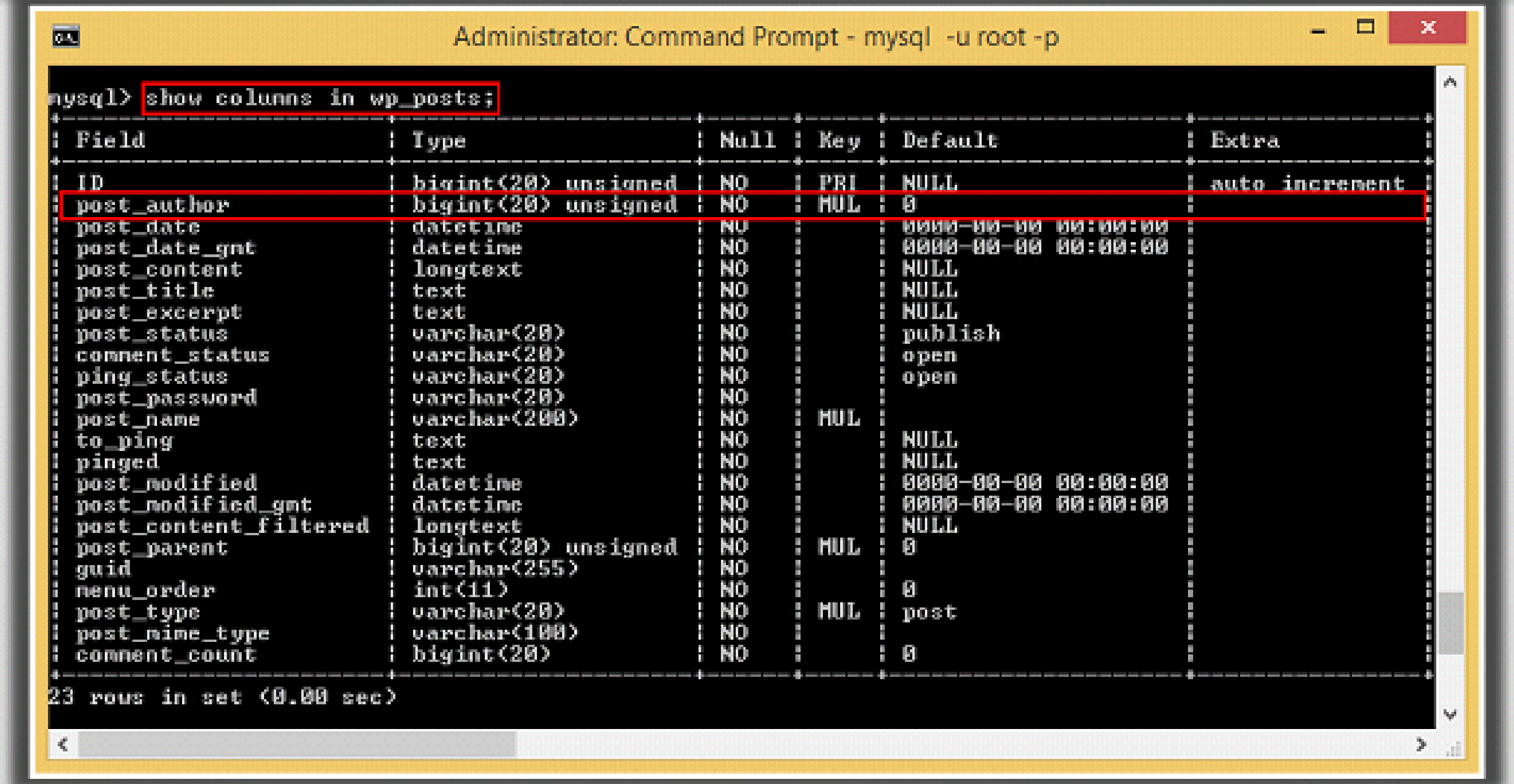
```
Administrator: Command Prompt - mysql  -u root -p
mysql> select * from wp_users;
+-----+------------+------------------------------------+------------------+-----------------+
| ID  | user_login | user_pass                          | user_nicename    | user_email      |
+-----+------------+------------------------------------+------------------+-----------------+
|   1 | admin      | $P$BawZBPTuhkGfVaz9Nt3kBSdk4KsMTd. | 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

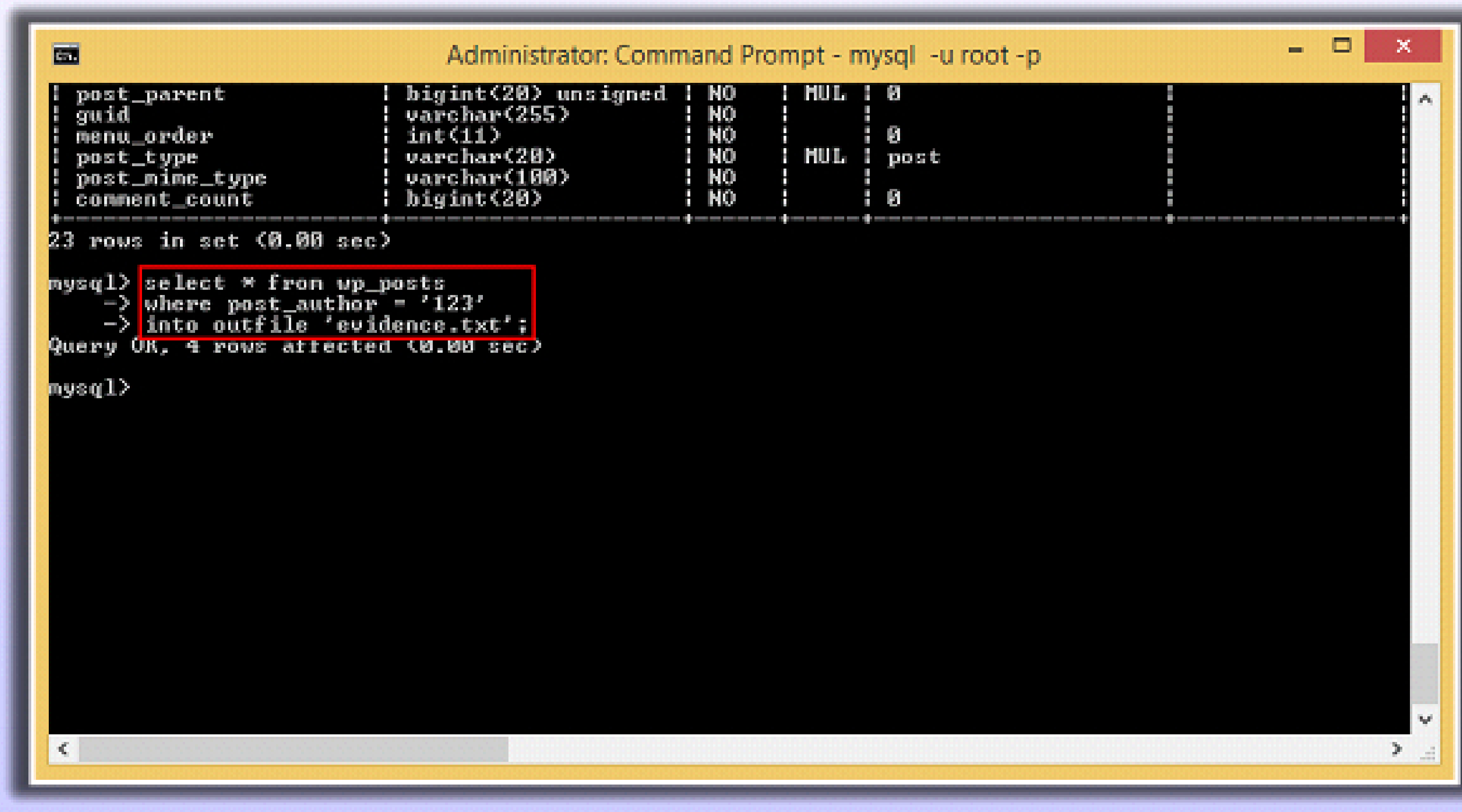

Administrator: Command Prompt - mysql -u root -p

```
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)
```

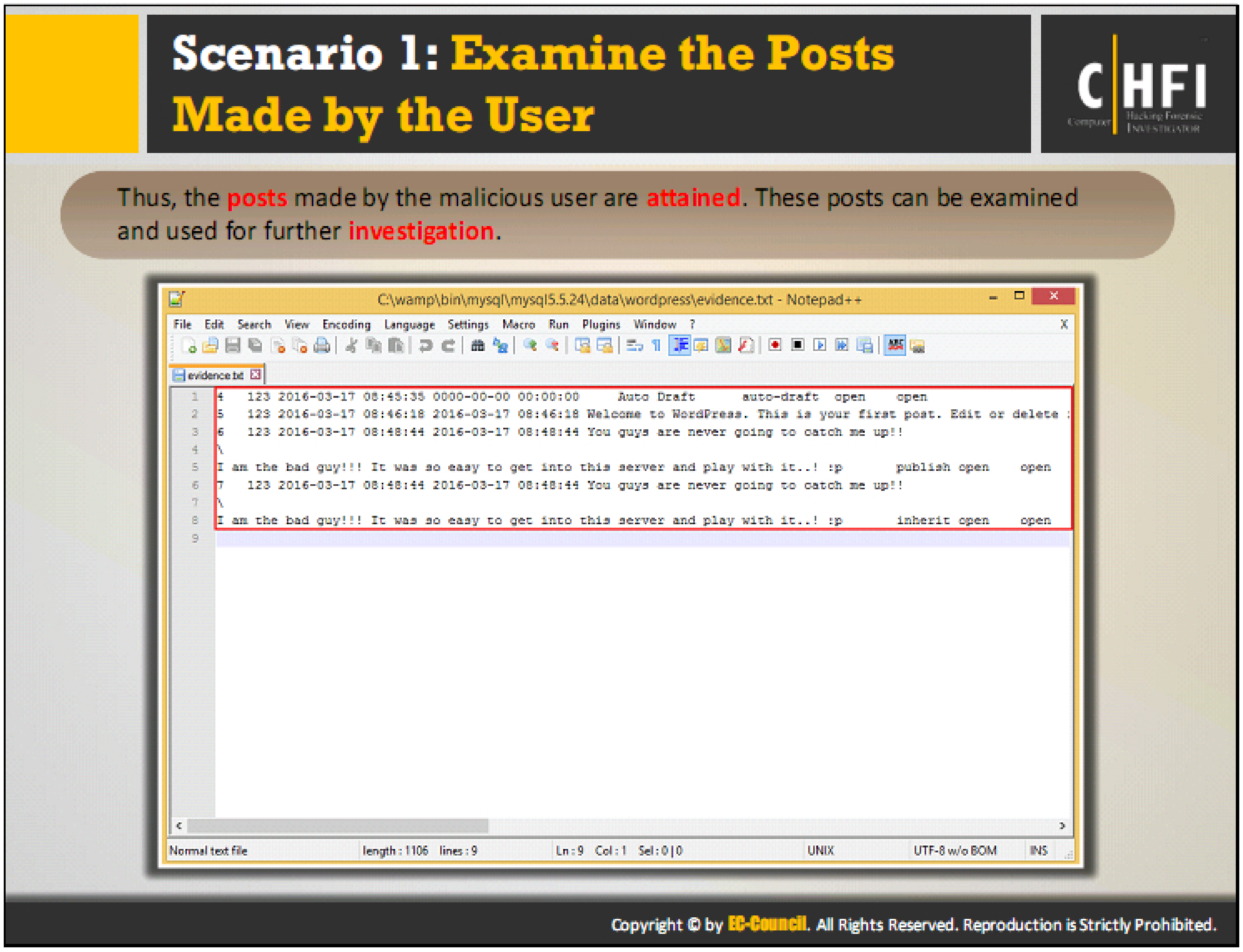
Scenario 1: Collect the Posts Made by the User

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
```

In this scenario, the attacker has performed brute-force attack on MySQL database and succeeded in cracking the user's credentials. Using the credentials, the attacker has logged into the database, created a user account, and then disabled the general query log. Turning off this log means that MySQL will not be able to record the transactions in this log file.

So, as a part of the forensic investigation on MySQL database, a command line client was used and a backup of the database was taken, as forensic investigations should not be performed on the affected machine. Once completed, a database was created on the forensic machine and dumped the database contents onto it. Later, the attacker's user ID was found and via using it the posts made by the attacker was found on the web application.

MySQL Forensics for WordPress Website Database: Scenario 2

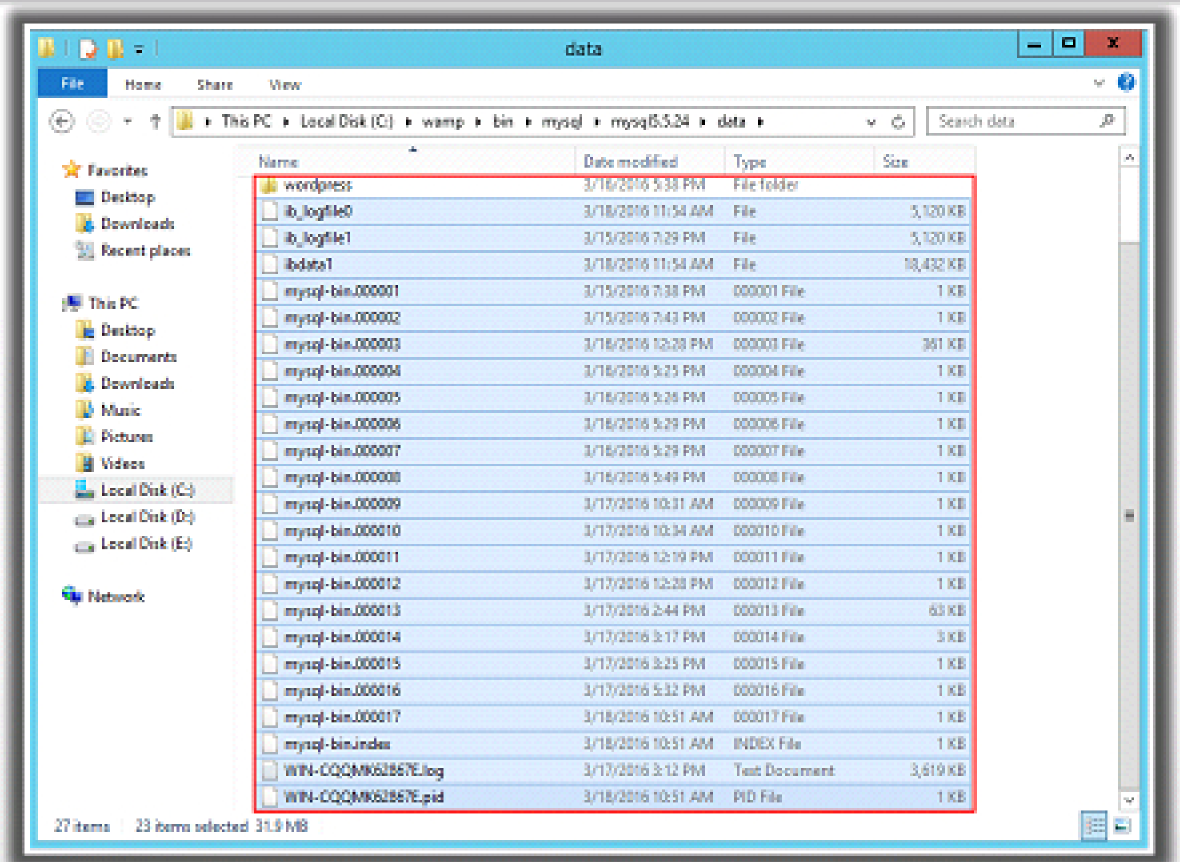
Tracking the Events Performed by the Malicious User (MyISAM Storage Engine) and Recovering the Deleted Data

As mentioned earlier, MySQL stores data and transactions in multiple files. MySQL stores all the transactions and data in the binary log files, and data alone in ibdata file.

Therefore, whenever someone deletes any important information intentionally or accidentally, forensic investigators examine the binary log files and ibdata file to view the transactions, as well as to recover the deleted information.

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

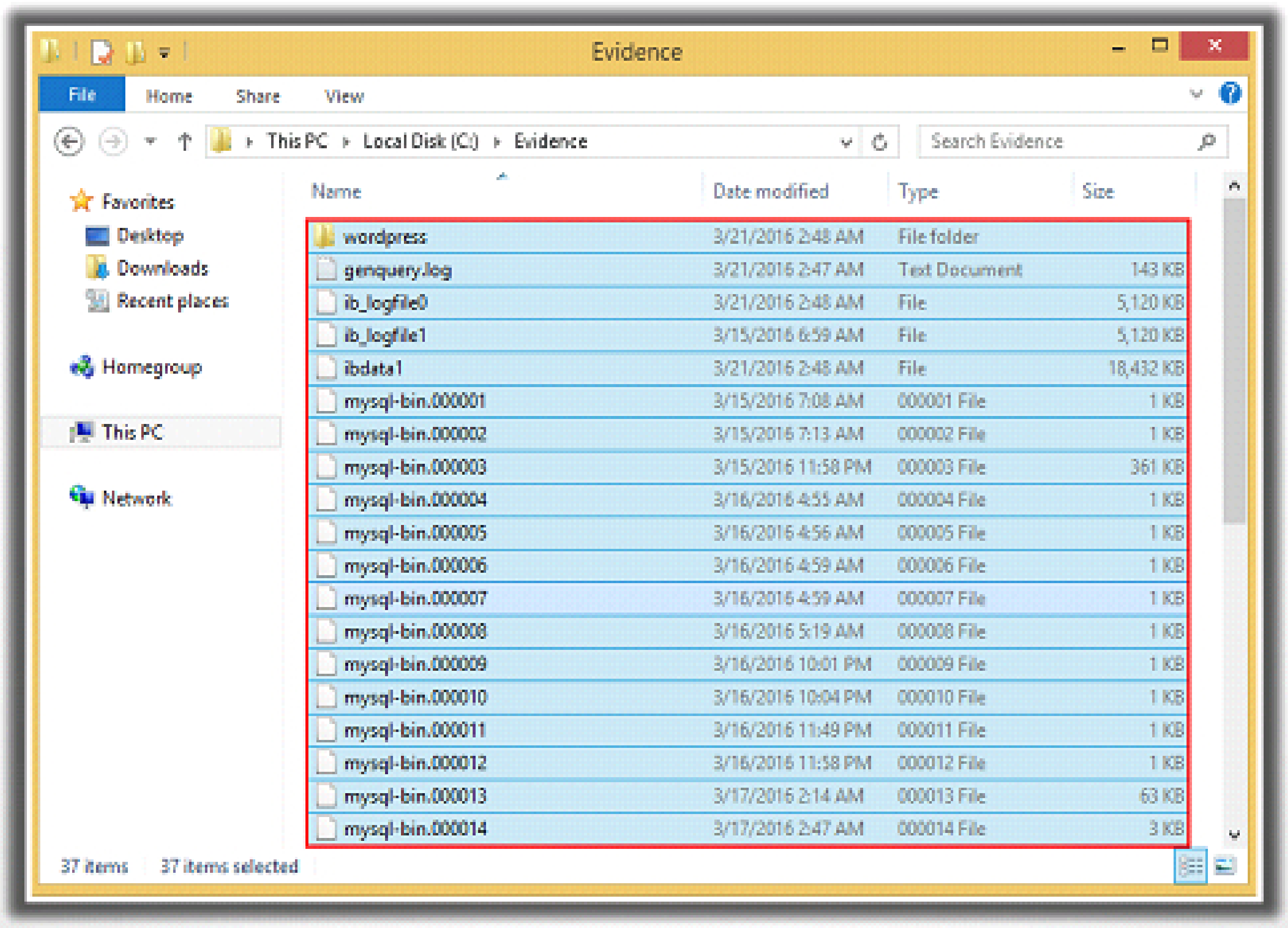


Name	Date modified	Type	Size
wordpress	3/16/2016 5:38 PM	File folder	
ib_logfile0	3/18/2016 11:54 AM	File	5,120 KB
ib_logfile1	3/15/2016 7:29 PM	File	5,120 KB
ibdata1	3/18/2016 11:54 AM	File	18,432 KB
mysql-bin.000001	3/15/2016 7:38 PM	000001 File	1 KB
mysql-bin.000002	3/15/2016 7:43 PM	000002 File	1 KB
mysql-bin.000003	3/16/2016 12:28 PM	000003 File	361 KB
mysql-bin.000004	3/16/2016 5:25 PM	000004 File	1 KB
mysql-bin.000005	3/16/2016 5:26 PM	000005 File	1 KB
mysql-bin.000006	3/16/2016 5:29 PM	000006 File	1 KB
mysql-bin.000007	3/16/2016 5:29 PM	000007 File	1 KB
mysql-bin.000008	3/16/2016 5:49 PM	000008 File	1 KB
mysql-bin.000009	3/17/2016 10:31 AM	000009 File	1 KB
mysql-bin.000010	3/17/2016 10:34 AM	000010 File	1 KB
mysql-bin.000011	3/17/2016 12:19 PM	000011 File	1 KB
mysql-bin.000012	3/17/2016 12:28 PM	000012 File	1 KB
mysql-bin.000013	3/17/2016 2:44 PM	000013 File	63 KB
mysql-bin.000014	3/17/2016 3:17 PM	000014 File	3 KB
mysql-bin.000015	3/17/2016 3:25 PM	000015 File	1 KB
mysql-bin.000016	3/17/2016 5:32 PM	000016 File	1 KB
mysql-bin.000017	3/18/2016 10:51 AM	000017 File	1 KB
mysql-bin.index	3/18/2016 10:51 AM	INDEX File	1 KB
WIN-CQQMK62867E.log	3/17/2016 3:12 PM	Text Document	3,619 KB
WIN-CQQMK62867E.pid	3/18/2016 10:51 AM	PID File	1 KB

27 items 23 items selected 31.9 MB

Scenario 2: Collect the Database and all the Logs

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

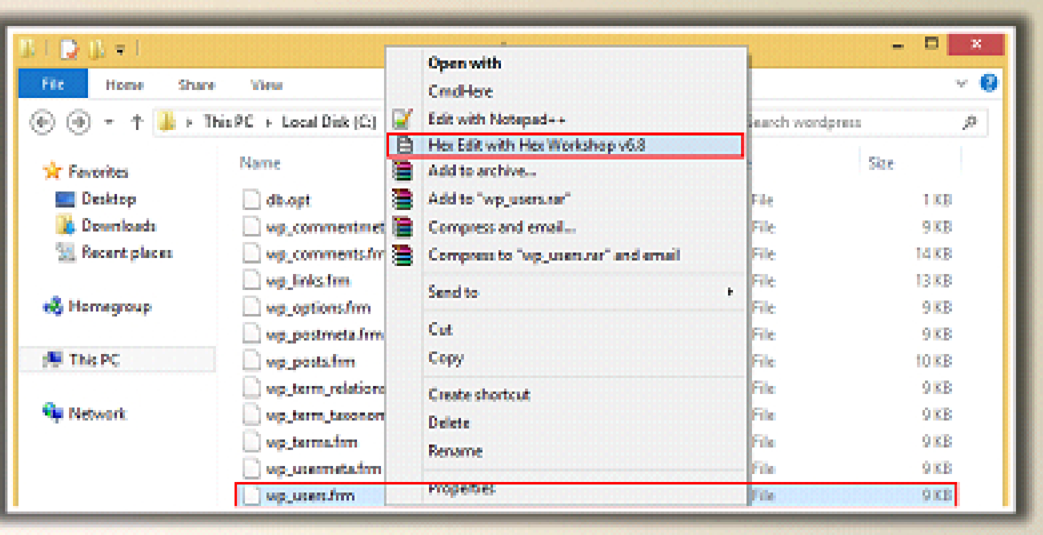


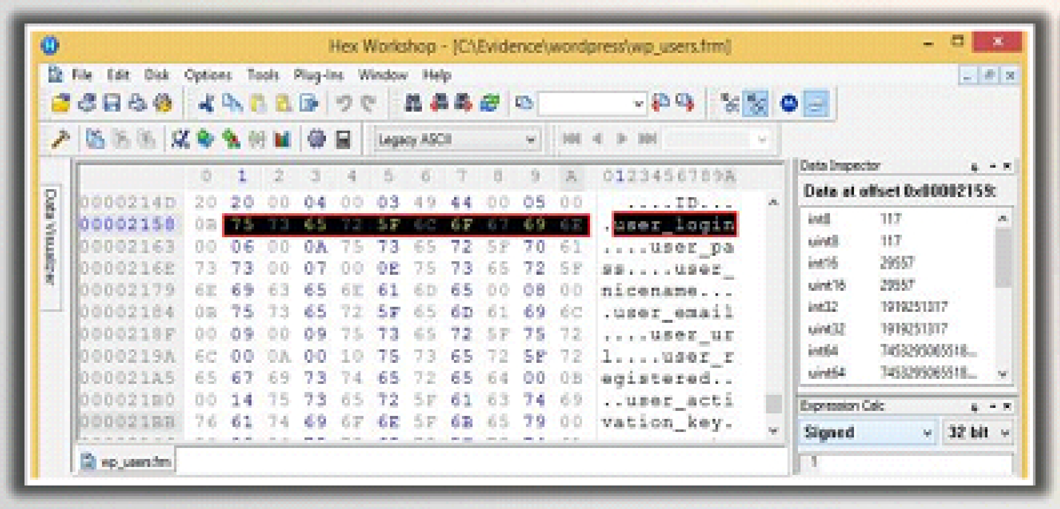
Name	Date modified	Type	Size
wordpress	3/21/2016 2:48 AM	File folder	
genquery.log	3/21/2016 2:47 AM	Text Document	143 KB
ib_logfile0	3/21/2016 2:48 AM	File	5,120 KB
ib_logfile1	3/15/2016 6:59 AM	File	5,120 KB
ibdata1	3/21/2016 2:48 AM	File	18,432 KB
mysql-bin.000001	3/15/2016 7:08 AM	000001 File	1 KB
mysql-bin.000002	3/15/2016 7:13 AM	000002 File	1 KB
mysql-bin.000003	3/15/2016 11:58 PM	000003 File	361 KB
mysql-bin.000004	3/16/2016 4:55 AM	000004 File	1 KB
mysql-bin.000005	3/16/2016 4:56 AM	000005 File	1 KB
mysql-bin.000006	3/16/2016 4:59 AM	000006 File	1 KB
mysql-bin.000007	3/16/2016 4:59 AM	000007 File	1 KB
mysql-bin.000008	3/16/2016 5:19 AM	000008 File	1 KB
mysql-bin.000009	3/16/2016 10:01 PM	000009 File	1 KB
mysql-bin.000010	3/16/2016 10:04 PM	000010 File	1 KB
mysql-bin.000011	3/16/2016 11:49 PM	000011 File	1 KB
mysql-bin.000012	3/16/2016 11:58 PM	000012 File	1 KB
mysql-bin.000013	3/17/2016 2:14 AM	000013 File	63 KB
mysql-bin.000014	3/17/2016 2:47 AM	000014 File	3 KB

37 items 37 items selected

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.



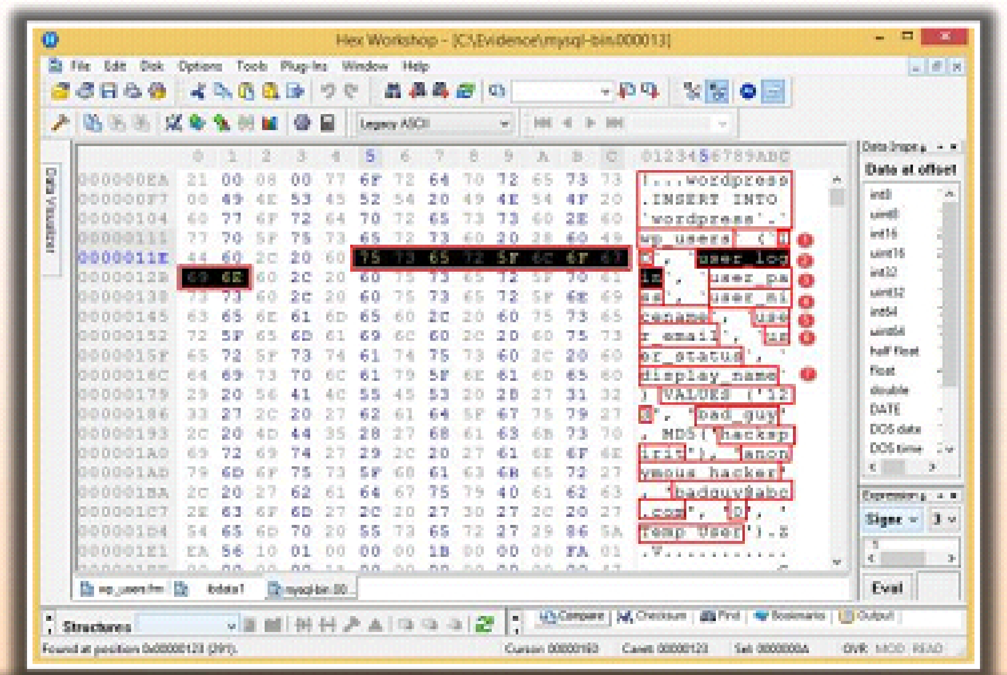

Hex Workshop - [C:\Evidence\wordpress\wp_users.frm]

```
          0  1  2  3  4  5  6  7  8  9  A   0123456789A
0000214D 20 20 00 04 00 03 49 44 00 05 00   .....ID...
00002158 0A 75 73 65 72 5F 6C 6F 67 69 6E  .user_login
00002163 00 06 00 0A 75 73 65 72 5F 70 61  ....user_pa
0000216E 73 73 00 07 00 0E 75 73 65 72 5F  ss....user_
00002179 6E 69 63 65 6E 61 6D 65 00 08 00  nicename...
00002184 0A 75 73 65 72 5F 65 6D 61 69 6C  .user_email
0000218F 00 09 00 09 75 73 65 72 5F 75 72  ....user_ur
0000219A 6C 00 0A 00 10 75 73 65 72 5F 72  l....user_r
000021A5 65 67 69 73 74 65 72 65 64 00 0B  egistered..
000021B0 00 14 75 73 65 72 5F 61 63 74 69  ..user_acti
000021BB 76 61 74 69 6F 6E 5F 6B 65 79 00  vation_key.
```

- 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.


Hex Workshop - [C:\Evidence\mysql-bin.000013]

```
          0  1  2  3  4  5  6  7  8  9  A  B  C   0123456789ABC
000000EA 21 00 08 00 77 6F 72 64 70 72 65 73 73  !...wordpress
000000F7 00 49 4E 53 45 52 54 20 49 4E 54 4F 20  .INSERT INTO 
00000104 60 77 6F 72 64 70 72 65 73 73 60 2E 60  `wordpress`.`
00000111 77 70 5F 75 73 65 72 73 60 20 28 60 49  wp_users` (`I
0000011E 44 60 2C 20 60 75 73 65 72 5F 6C 6F 67  D`, `user_log
0000012B 69 6E 60 2C 20 60 75 73 65 72 5F 70 61  in`, `user_pa
00000138 73 73 60 2C 20 60 75 73 65 72 5F 6E 69  ss`, `user_ni
00000145 63 65 6E 61 6D 65 60 2C 20 60 75 73 65  cename`, `use
00000152 72 5F 65 6D 61 69 6C 60 2C 20 60 75 73  r_email`, `us
0000015F 65 72 5F 73 74 61 74 75 73 60 2C 20 60  er_status`, `
0000016C 64 69 73 70 6C 61 79 5F 6E 61 6D 65 60  display_name`
00000179 29 20 56 41 4C 55 45 53 20 28 27 31 32  ) VALUES ('12
00000186 33 27 2C 20 27 62 61 64 5F 67 75 79 27  3', 'bad_guy'
00000193 2C 20 4D 44 35 28 27 68 61 63 6B 73 70  , MD5('hacksp
000001A0 69 72 69 74 27 29 2C 20 27 61 6E 6F 6E  irit'), 'anon
000001AD 79 6D 6F 75 73 5F 68 61 63 6B 65 72 27  ymous_hacker'
000001BA 2C 20 27 62 61 64 67 75 79 40 61 62 63  , 'badguy@abc
000001C7 2E 63 6F 6D 27 2C 20 27 30 27 2C 20 27  .com', '0', '
000001D4 54 65 6D 70 20 55 73 65 72 27 29 86 5A  Temp User').Z
000001E1 EA 56 10 01 00 00 00 1B 00 00 00 FA 01  .V...........
```

Found at position 0x00000123 (291).

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

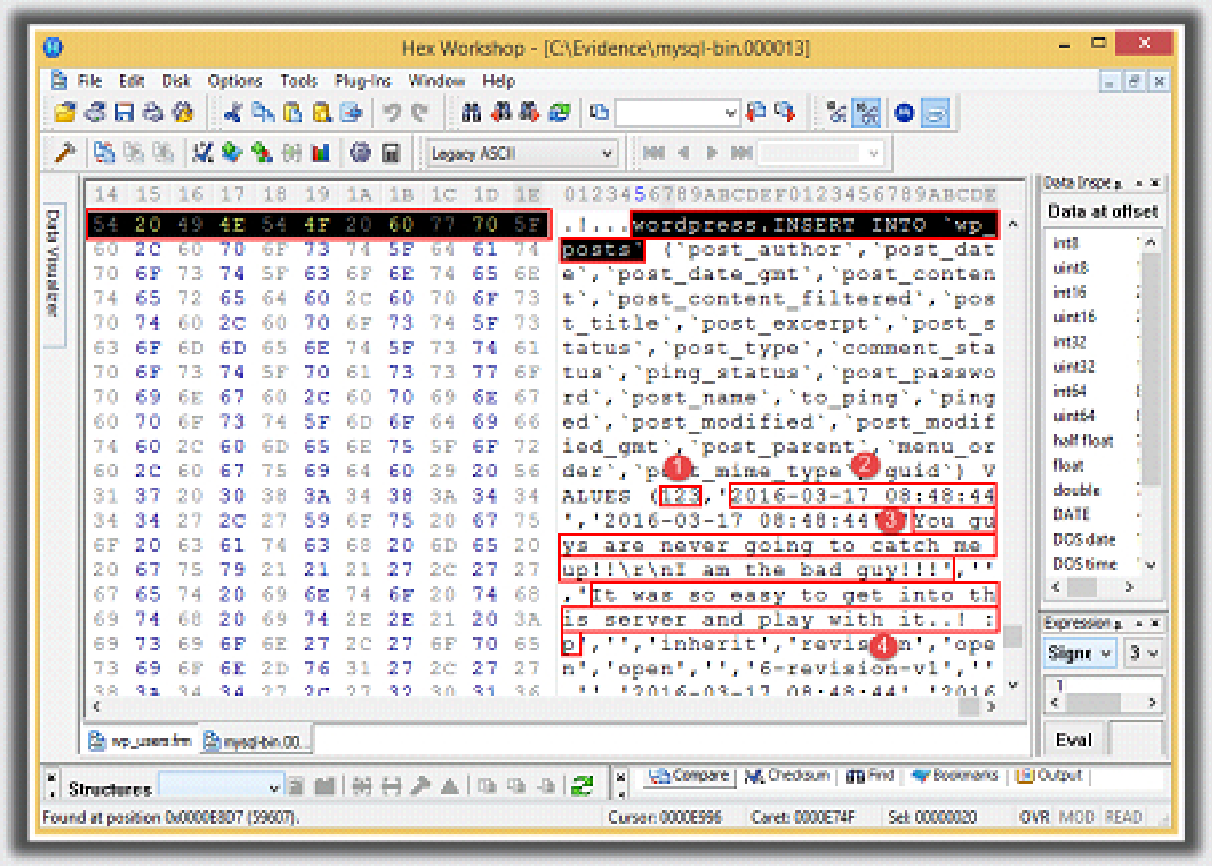
Note: Examine only those activities which correspond to the user ID/port_author ID 123

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.


Hex Workshop - [C:\Evidence\mysql-bin.000013]

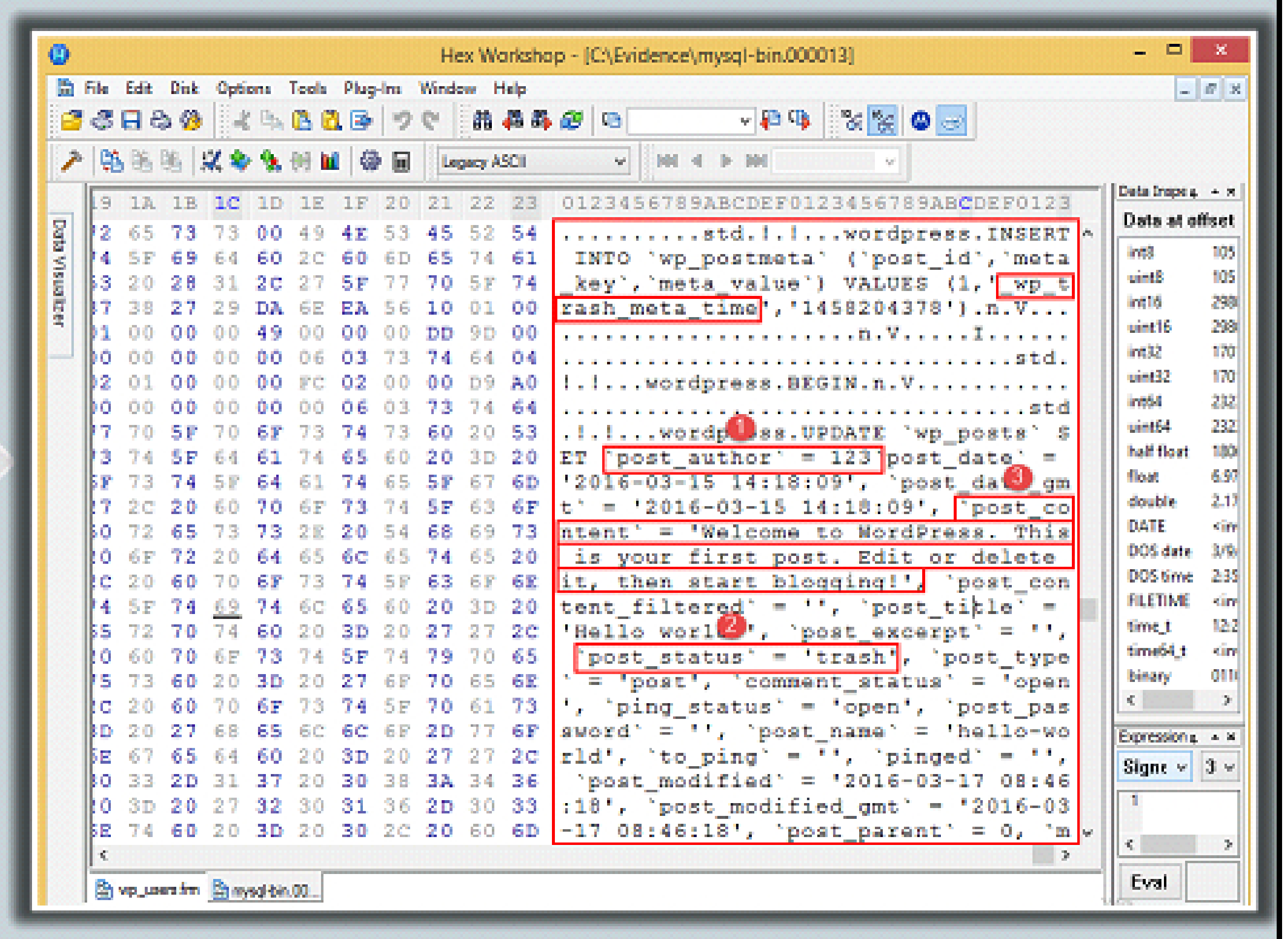
```
14 15 16 17 18 19 1A 1B 1C 1D 1E   0123456789ABCDEF0123456789ABCDE
54 20 49 4E 54 4F 20 60 77 70 5F   .!...wordpress.INSERT INTO `wp_
60 2C 60 70 6F 73 74 5F 64 61 74   posts` (`post_author`,`post_dat
70 6F 73 74 5F 63 6F 6E 74 65 6E   e`,`post_date_gmt`,`post_conten
74 65 72 65 64 60 2C 60 70 6F 73   t`,`post_content_filtered`,`pos
70 74 60 2C 60 70 6F 73 74 5F 73   t_title`,`post_excerpt`,`post_s
63 6F 6D 6D 65 6E 74 5F 73 74 61   tatus`,`post_type`,`comment_sta
70 6F 73 74 5F 70 61 73 73 77 6F   tus`,`ping_status`,`post_passwo
70 69 6E 67 60 2C 60 70 69 6E 67   rd`,`post_name`,`to_ping`,`ping
60 70 6F 73 74 5F 6D 6F 64 69 66   ed`,`post_modified`,`post_modif
74 60 2C 60 6D 65 6E 75 5F 6F 72   ied_gmt`,`post_parent`,`menu_or
60 2C 60 67 75 69 64 60 29 20 56   der`,`post_mime_type`,`guid`) V
31 37 20 30 38 3A 34 38 3A 34 34   ALUES (123,'2016-03-17 08:48:44
34 34 27 2C 27 59 6F 75 20 67 75   ','2016-03-17 08:48:44','You gu
6F 20 63 61 74 63 68 20 6D 65 20   ys are never going to catch me
20 67 75 79 21 21 21 27 2C 27 27   up!!\r\nI am the bad guy!!!',''
67 65 74 20 69 6E 74 6F 20 74 68   ,'It was so easy to get into th
69 74 68 20 69 74 2E 2E 21 20 3A   is server and play with it..! :
69 73 69 6F 6E 27 2C 27 6F 70 65   p','','inherit','revision','ope
73 69 6F 6E 2D 76 31 27 2C 27 27   n','open','','6-revision-v1',''
```

```
Found at position 0x0000E8D7 (59607).    Cursor: 0000E996    Caret: 0000E74F    Sel: 00000020    OVR  MOD  READ
```

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.


Hex Workshop - [C:\Evidence\mysql-bin.000013]

```
19 1A 1B 1C 1D 1E 1F 20 21 22 23   0123456789ABCDEF0123456789ABCDEF0123
2 65 73 73 00 49 4E 53 45 52 54    ...........std.!.!...wordpress.INSERT
4 5F 69 64 60 2C 60 6D 65 74 61     INTO `wp_postmeta` (`post_id`,`meta
3 20 28 31 2C 27 5F 77 70 5F 74    _key`,`meta_value`) VALUES (1,'_wp_t
7 38 27 29 DA 6E EA 56 10 01 00    rash_meta_time','1458204378').n.V...
1 00 00 00 49 00 00 00 DD 9D 00    ........................n.V.....I......
0 00 00 00 00 06 03 73 74 64 04    ...................................std.
2 01 00 00 00 FC 02 00 00 D9 A0    !.!...wordpress.BEGIN.n.V...........
0 00 00 00 00 00 06 03 73 74 64    ...................................std
7 70 5F 70 6F 73 74 73 60 20 53    .!.!...wordpress.UPDATE `wp_posts` S
3 74 5F 64 61 74 65 60 20 3D 20    ET `post_author` = 123`post_date` =
F 73 74 5F 64 61 74 65 5F 67 6D    '2016-03-15 14:18:09', `post_date_gm
7 2C 20 60 70 6F 73 74 5F 63 6F    t` = '2016-03-15 14:18:09', `post_co
0 72 65 73 73 2E 20 54 68 69 73    ntent` = 'Welcome to WordPress. This
0 6F 72 20 64 65 6C 65 74 65 20     is your first post. Edit or delete
C 20 60 70 6F 73 74 5F 63 6F 6E    it, then start blogging!', `post_con
4 5F 74 69 74 6C 65 60 20 3D 20    tent_filtered` = '', `post_title` =
5 72 70 74 60 20 3D 20 27 27 2C    'Hello world!', `post_excerpt` = '',
0 60 70 6F 73 74 5F 74 79 70 65     `post_status` = 'trash', `post_type
5 73 60 20 3D 20 27 6F 70 65 6E    ` = 'post', `comment_status` = 'open
C 20 60 70 6F 73 74 5F 70 61 73    ', `ping_status` = 'open', `post_pas
D 20 27 68 65 6C 6C 6F 2D 77 6F    sword` = '', `post_name` = 'hello-wo
E 67 65 64 60 20 3D 20 27 27 2C    rld', `to_ping` = '', `pinged` = '',
0 33 2D 31 37 20 30 38 3A 34 36     `post_modified` = '2016-03-17 08:46
0 3D 20 27 32 30 31 36 2D 30 33    :18', `post_modified_gmt` = '2016-03
E 74 60 20 3D 20 30 2C 20 60 6D    -17 08:46:18', `post_parent` = 0, `m
```

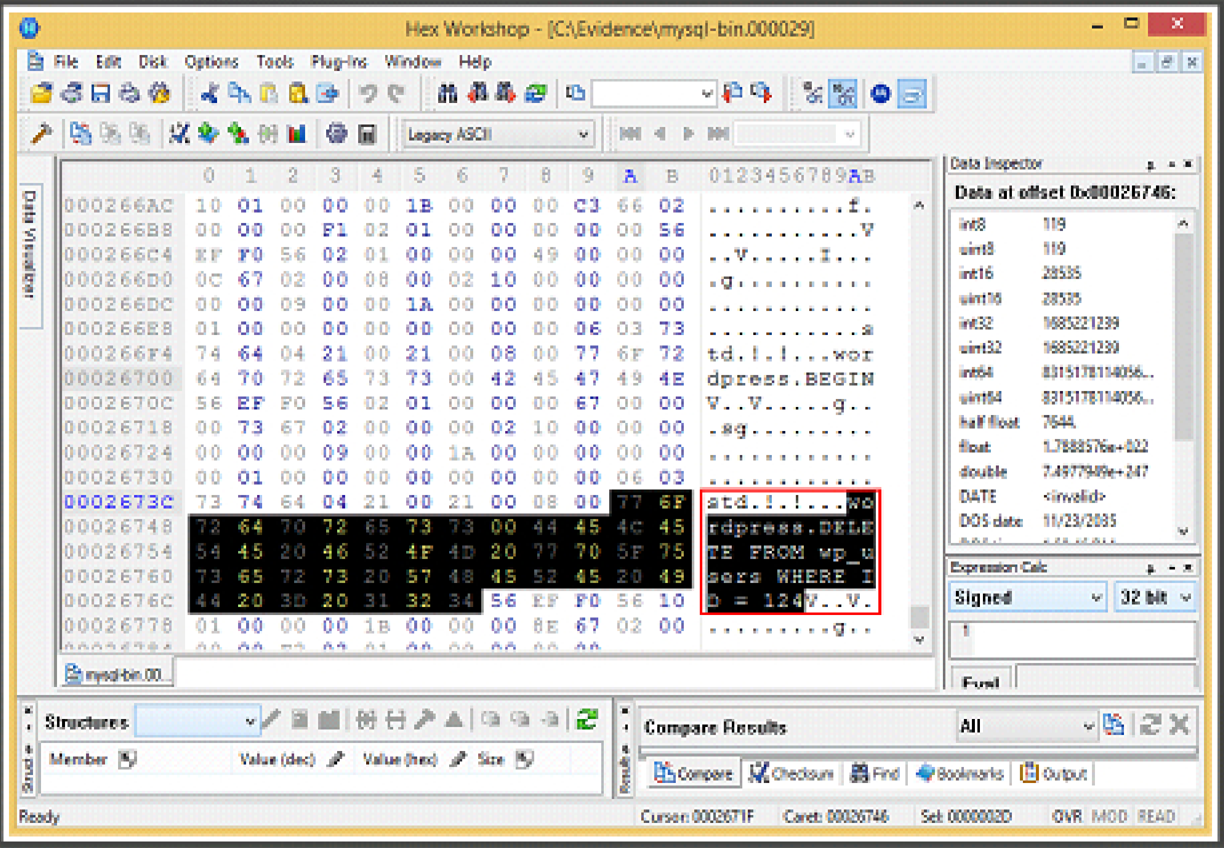
Scenario 2: Examine the Binary Logs (Cont'd)

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
Data Visualizer
Structures
Member Value (dec) Value (hex) Size
Compare Results
All
Compare Checksum Find Bookmarks Output
mysql-bin.00...
Expression Calc
Signed
32 bit
Eval

```
          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       1.788576e+022
double      7.4977949e+247
DATE        <invalid>
DOS date    11/23/2035
```

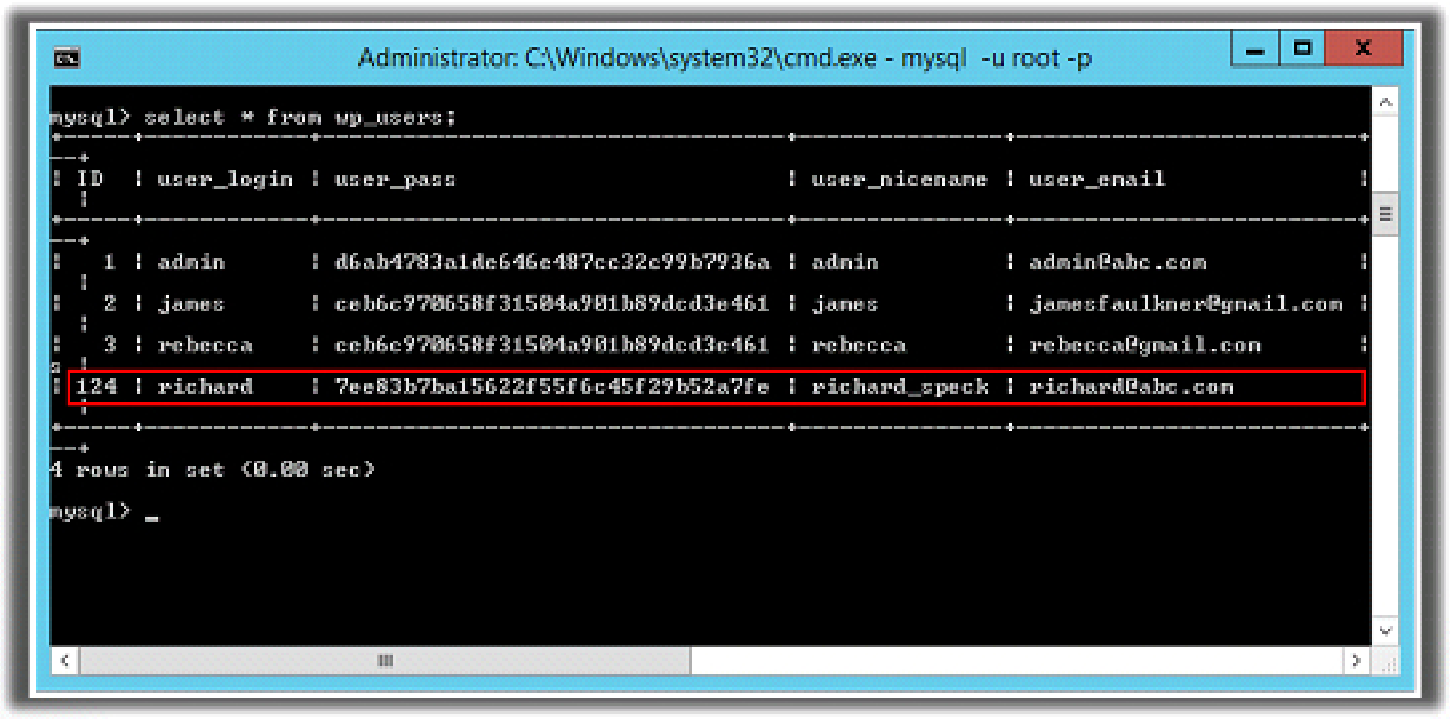
```
Ready    Cursor: 0002671F    Caret: 00026746    Sel: 0000002D    OVR MOD READ
```

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

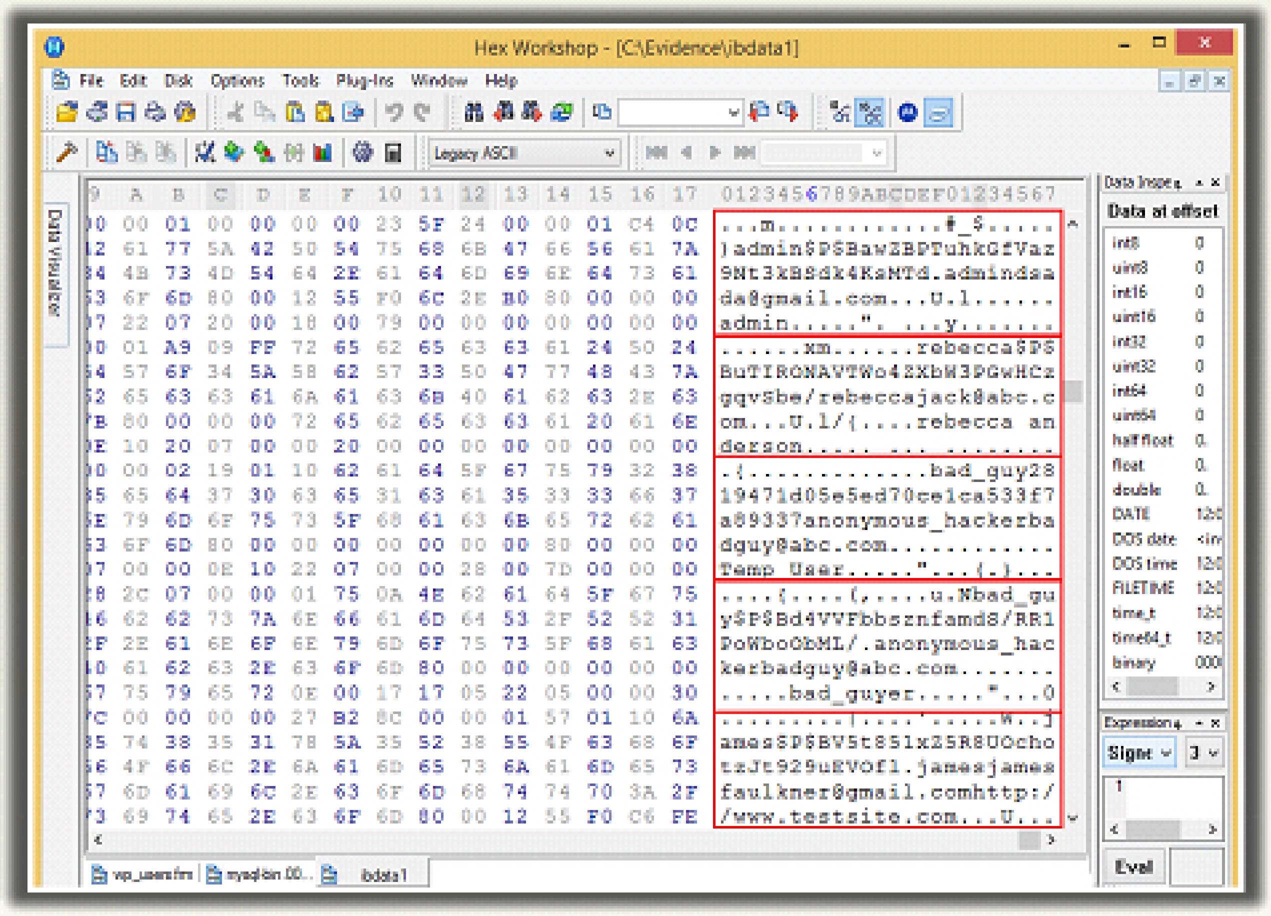

Administrator: C:\Windows\system32\cmd.exe - mysql -u root -p

```
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.

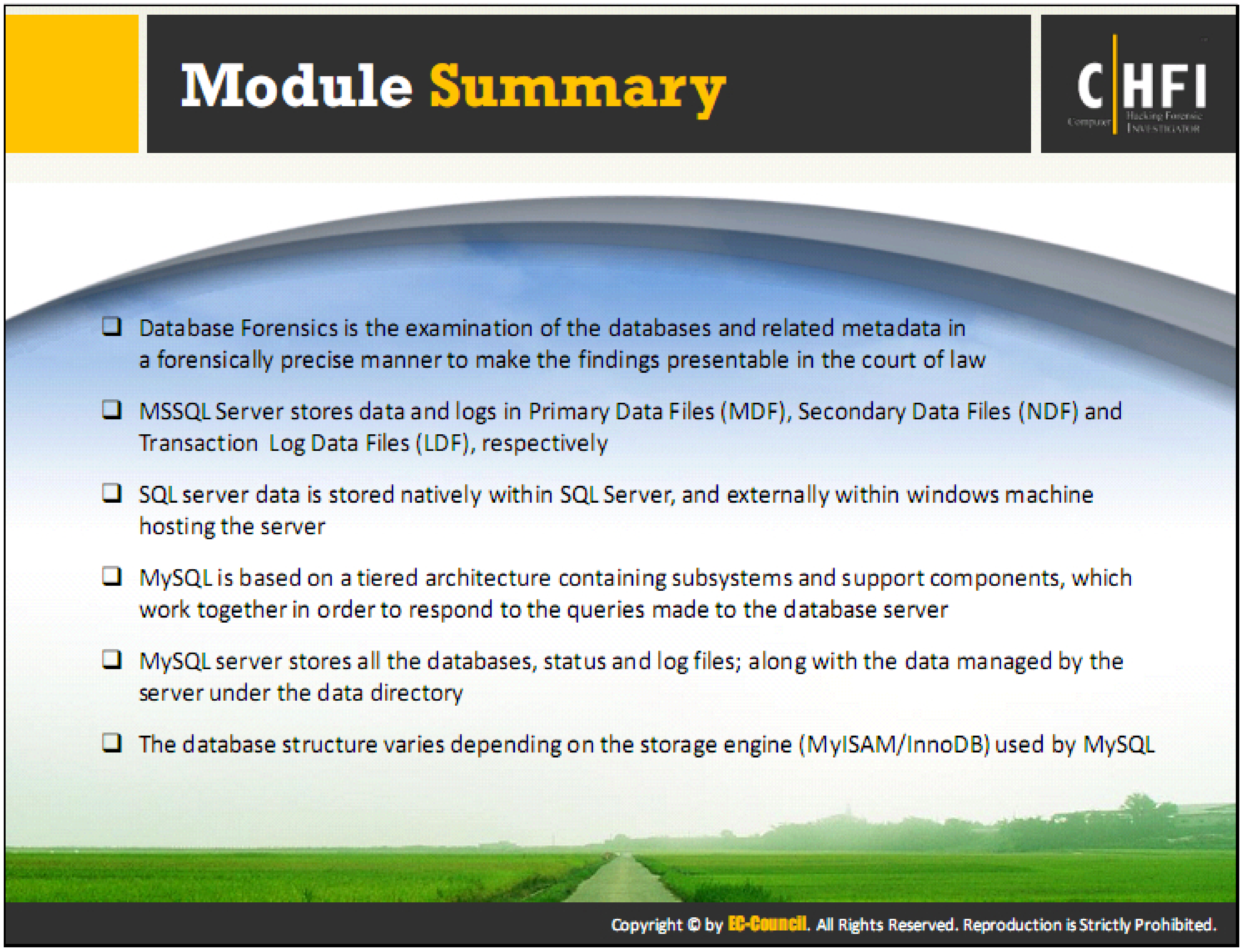

Hex Workshop - [C:\Evidence\ibdata1]

```
9  A  B  C  D  E  F  10 11 12 13 14 15 16 17  0123456789ABCDEF01234567
0  00 01 00 00 00 00 23 5F 24 00 00 01 C4 0C  ...m..............#_$.....
2  61 77 5A 42 50 54 75 68 6B 47 66 56 61 7A  )admin$P$BawZBPTuhkGfVaz
4  4B 73 4D 54 64 2E 61 64 6D 69 6E 64 73 61  9Nt3kBSdk4KsMTd.admindsa
3  6F 6D 80 00 12 55 F0 6C 2E B0 80 00 00 00  da@gmail.com...U.l......
7  22 07 20 00 18 00 79 00 00 00 00 00 00 00  admin.....". ...y.......
0  01 A9 09 FF 72 65 62 65 63 63 61 24 50 24  .......xm.......rebecca$P$
4  57 6F 34 5A 58 62 57 33 50 47 77 48 43 7A  BuTIRONAVTWo4ZXbW3PGwHCz
2  65 63 63 61 6A 61 63 6B 40 61 62 63 2E 63  gqvSbe/rebeccajack@abc.c
B  80 00 00 00 72 65 62 65 63 63 61 20 61 6E  om...U.l/{.....rebecca an
E  10 20 07 00 00 20 00 00 00 00 00 00 00 00  derson..... ... ........
0  00 02 19 01 10 62 61 64 5F 67 75 79 32 38  .{...............bad_guy28
5  65 64 37 30 63 65 31 63 61 35 33 33 66 37  19471d05e5ed70ce1ca533f7
E  79 6D 6F 75 73 5F 68 61 63 6B 65 72 62 61  a89337anonymous_hackerba
3  6F 6D 80 00 00 00 00 00 00 00 80 00 00 00  dguy@abc.com............
7  00 00 0E 10 22 07 00 00 28 00 7D 00 00 00  Temp User....."...{.}...
8  2C 07 00 00 01 75 0A 4E 62 61 64 5F 67 75  ....{.....(,.....u.Nbad_gu
6  62 62 73 7A 6E 66 61 6D 64 53 2F 52 52 31  y$P$Bd4VVFbbsznfamdS/RR1
F  2E 61 6E 6F 6E 79 6D 6F 75 73 5F 68 61 63  PoWboObML/.anonymous_hac
0  61 62 63 2E 63 6F 6D 80 00 00 00 00 00 00  kerbadguy@abc.com.......
7  75 79 65 72 0E 00 17 17 05 22 05 00 00 30  .....bad_guyer....."...0
C  00 00 00 00 27 B2 8C 00 00 01 57 01 10 6A  ..........|.....'.....W..j
5  74 38 35 31 78 5A 35 52 38 55 4F 63 68 6F  ames$P$BV5t851xZ5R8UOcho
E  4F 66 6C 2E 6A 61 6D 65 73 6A 61 6D 65 73  tzJt929uEVOfl.jamesjames
7  6D 61 69 6C 2E 63 6F 6D 68 74 74 70 3A 2F  faulkner@gmail.comhttp:/
3  69 74 65 2E 63 6F 6D 80 00 12 55 F0 C6 FE  /www.testsite.com...U...
```

In this scenario, the attacker has created a user account in the database and manipulated its contents.

Therefore, as a part of the forensic investigation on MySQL database, we have dumped all the files in the forensic examiner's machine in a folder named 'Evidence'. Once completed, we have examined .frm to understand the table structure. On obtaining the table structure, we opened the binary log files, and began to search for the text string "user_login". Using the string, the query executed by the attacker was found, to create a user account. Later, the posts made by the attacker was analyzed, and we were able to reveal that the attacker has deleted a post, and we recovered it. The binary file also recorded an event, which showed that a user has deleted the account. There is an ibdata file, which stores data permanently. Forensic investigators can refer this file as a part of a forensic investigation.

Thus, we examined the binary log files and determined the deleted posts and users.



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

In this module, we learn about the two types of database management systems, MSSQL and MySQL, and their underlying file structure as well as evidence repositories. Forensic investigators can implement the techniques discussed in this module and perform an investigation of the database management systems. They will be able to recover the deleted entries, find the transactions occurred on the databases (MSSQL and MySQL), and also take a backup of the database files (MySQL) and work on them.