SQL Server Database Corruptions
Part 1 – Database Structure

Like Chennai SQL Server User Group
Blog: http://sql-articles.com/category/cssug/

Ramkumar Gopal

Living For SQL Server

www.livingforsqlserver.com

Like https://www.facebook.com/#!/groups/livingforsqlserver/

Chennai SQL Server User Group (CSSUG) Meet – 26 Oct 2013

Agenda

- Database Files - Overview
- Pages And Extents
- System Pages
  - Header, PFS, GAM, SGAM, BCM, DCM, IAM etc
- User Pages
  - Data and Index Pages
- Partitions and Allocation Units
- DBCC Commands
  - DBCC EXTENTINFO
  - DBCC IND
  - DBCC PAGE
  - DBCC WRITEPAGE
- HexEditor

Database Files - Overview

A database can have three types of files

- Primary data file (.MDF)
  Every database has one primary data file and is the starting point of database.
  All user tables are created in PRIMARY file group if no other secondary files are present
- Secondary data file (.NDF) –
  A database can have optionally zero or more Secondary data files,
  created for better performance and to manage data efficiently.
- Transaction Log file (.LDF) –
  A database by default will have one log file, used to record changes made in tables.
  This will be backed up and used later in case of data loss.

Pages and Extents

Page

- Is the fundamental unit of data storage in SQL Server.
- Once database is created, data space in primary and secondary files are logically divided into 8k blocks called Pages.
- Page numbers are referred along with data file number (Ex: PAGE 1:456)

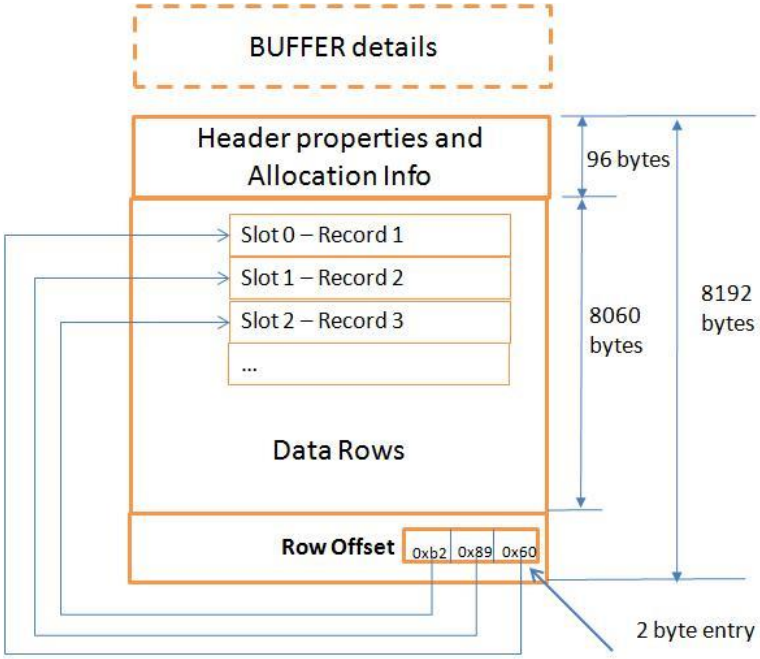
<u>Types of pages:</u>

System Page types

1. Header page
2. GAM page (Global allocation map)
3. SGAM page (Shared global allocation map)
4. IAM page (Index allocation map)
5. PFS page (Page free space)
6. BCM page (Bulk change map)
7. DCM page (Differential change map)
8. **Metadata Pages**

Data related page types

1. Data page
2. Index page



Pages and Extents

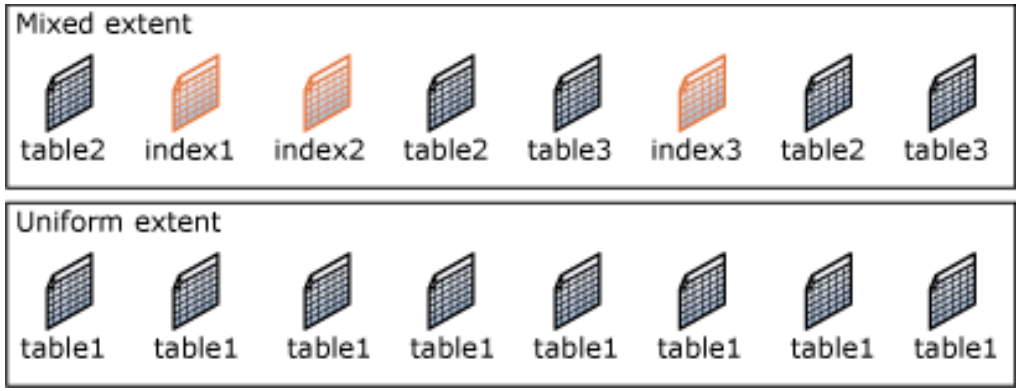
Extents

- Extents are collection of 8 pages.
- Pages in an extent can be owned by single table or many small tables.

There are two types of extents

1. Uniform extent – all 8 pages are belongs to one object
2. Mixed extent – 8 pages are shared by two or more tables

Fact is, when a table is created and a row is inserted, table gets 1 page in mixed extent.
When a table grows , table gets pages from uniform extent.
This logic is to manage pages (in other words space) effectively



System Pages

Page	Description
Header Page	Stores detals about data files like size and type etc.
GAM	Global Allocation Map. GAM is the 3rd page (1:2) in each data file. It's a bitmap with 64000 bits (8000 bytes) each bit tells whether the associated extent is uniform or not. With one GAM, Storage engine can identify upto 64000 uniform extents. Meaning one GAM will cover upto 4 GB data space. What if data file size is greater than 4GB? Simple. another GAM will cover next 4 GB portion
SGAM	Shared Global Allocation Map. SGAM is the 4th page (1:3) in each data file. It's a bitmap with 64000 bits (8000 bytes) each bit tells whether the associated extent is mixed extent or not. With one SGAM, Storage engine can identify upto 64000 mixed extents. Meaning one SGAM will cover upto 4 GB data space. What if data file size is greater than 4GB? Simple. another SGAM will cover next 4 GB portion
IAM	Index Allocation Map. One IAM page for each table to identify pages associated with a table (there can be many IAM for one table if table is partitioned and/or a table have row overflow pages/LOB data etc. im leaving them unexplained as this may confuse beginners). It's a bitmap with 64000 bits (8000 bytes) each bit tells whether the associated extent is allocated to a table or not. With one IAM, Storage engine can identify upto 64000 extents. Meaning one IAM will cover upto 4 GB data space. What if a table size is greater than 4GB? Simple. another IAM will cover next 4 GB portion.

System Pages

Page	Description
PFS	Page Free Space. PFS is used for storing space availability of each pages. One byte for each page. There will be one PFS for every 8000 pages.
DCM	Differential Change Map. DCM is a bitmap page helps differential backups to identify the extents changed since Full backup.
BCM	Bulk change map. Used to monitor extents affected by bulk operations. Used when database in bulk logged recovery mode.

Page Types

1 - data page.
2 - index page.
8 - GAM page.
9 - SGAM page.
10 - IAM page.
11 - PFS page.
13 - boot page.
15 - file header page.
16 - diff map page.
17 - ML map page

When a database is created, a data file will have pages like this,

Page	Header	PFS	GAM	SGAM	Unused	Unused	DCM	BCM	....	Data	Data
Page No.	1:0	1:1	1:2	1:3	1:4	1:5	1:6	1:7	....	1:15001	1:15002

http://www.sqlskills.com/blogs/paul/inside-the-storage-engine-gam-sgam-pfs-and-other-allocation-maps/

Partitions and Allocation Units

Partitions
When a table is created (Heap), by default all its pages are aligned in one partition.
Partition is a unit of data organization.

Whenever non clustered indexes are added, partitions are framed for each non clustered index
All index pages related to non clustered indexes are logically assigned to respective partitions.

Allocation Units
Based on data it holds, a partition can have 3 kind of page groups (in other term allocation units)
1. IN_ROW_DATA pages,
2. ROW_OVERFLOW_DATA pages and
3. LOB_DATA Pages.

Difference among IN_ROW_DATA, ROW_OVERFLOW_DATA and LOB_DATA Pages

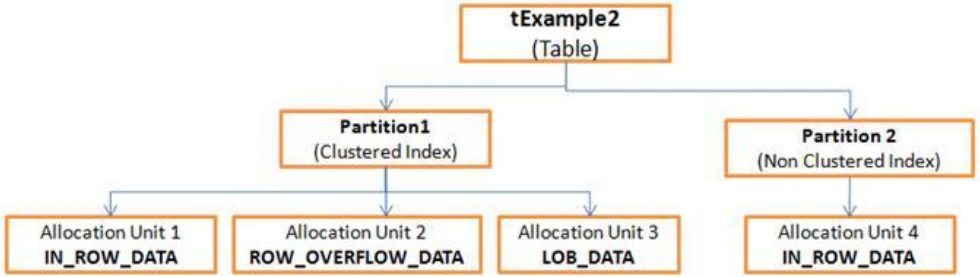
1. IN_ROW_DATA (default):
If a table is relatively simple in design. All records are stored in a pages referred as IN_ROW_DATA pages.
2. ROW_OVERFLOW_DATA:
holds data exceeds page maximum limit 8k.
3. LOB_DATA:
LOB data like varchar(max) is stored in special pages called LOB_DATA pages.
16 byte pointer in data page will be used to refer LOB_DATA page.

Extent Allocations - GAM/SGAM

GAM	SGAM	Any IAM	Comments
0	0	0	Mixed extent with all pages allocated
0	0	1	Dedicated extent (must be allocated to only a single IAM page)
0	1	0	Mixed extent with >= 1 unallocated page
0	1	1	Invalid state
1	0	0	Unallocated extent
1	0	1	Invalid state
1	1	0	Invalid state
1	1	1	Invalid state

Source: http://www.sqlskills.com/blogs/paul/inside-the-storage-engine-gam-sgam-pfs-and-other-allocation-maps/

Partitions and Allocation Units




```
SELECT so.name, so.object_id, sp.index_id, sp.partition_id, sp.hobt_id,
sa.allocation_unit_id, sa.type_desc, sa.total_pages
FROM  sys.objects so
            inner join sys.partitions sp on so.object_id = sp.object_id
            inner join sys.allocation_units sa on sa.container_id = sp.hobt_id
WHERE so.name = 'tExample2'
go
```

Object_id and Table name tExample2

2 Partition_id's are created.
One for clustered index another for NCI

3 Allocation_units are created for clustered index
1 Allocation_unit is created for non clustered index

name	object_id	index_id	partition_id	hobt_id	allocation_unit_id	type_desc	total_pages
tExample2	53575229	1	72057594038910976	72057594038910976	72057594043301888	IN_ROW_DATA	2
tExample2	53575229	1	72057594038910976	72057594038910976	72057594043367424	ROW_OVERFLOW_DATA	2
tExample2	53575229	1	72057594038910976	72057594038910976	72057594043432960	LOB_DATA	2
tExample2	53575229	2	72057594038976512	72057594038976512	72057594043498496	IN_ROW_DATA	2

Cluster and NonClustered Index Id's

Partitions and Allocation Units (cont.)

```
select so.name, so.object_id, sp.index_id, internals.total_pages, internals.used_pages,
internals.data_pages, first_iam_page, first_page, root_page
        from sys.objects so
        inner join sys.partitions sp on so.object_id = sp.object_id
        inner join sys.allocation_units sa on sa.container_id = sp.hobt_id
        inner join sys.system_internals_allocation_units internals
                on internals.container_id = sa.container_id
where so.object_id = object_id('tExample3')
go
```

Index_id = 0 as this table is a heap

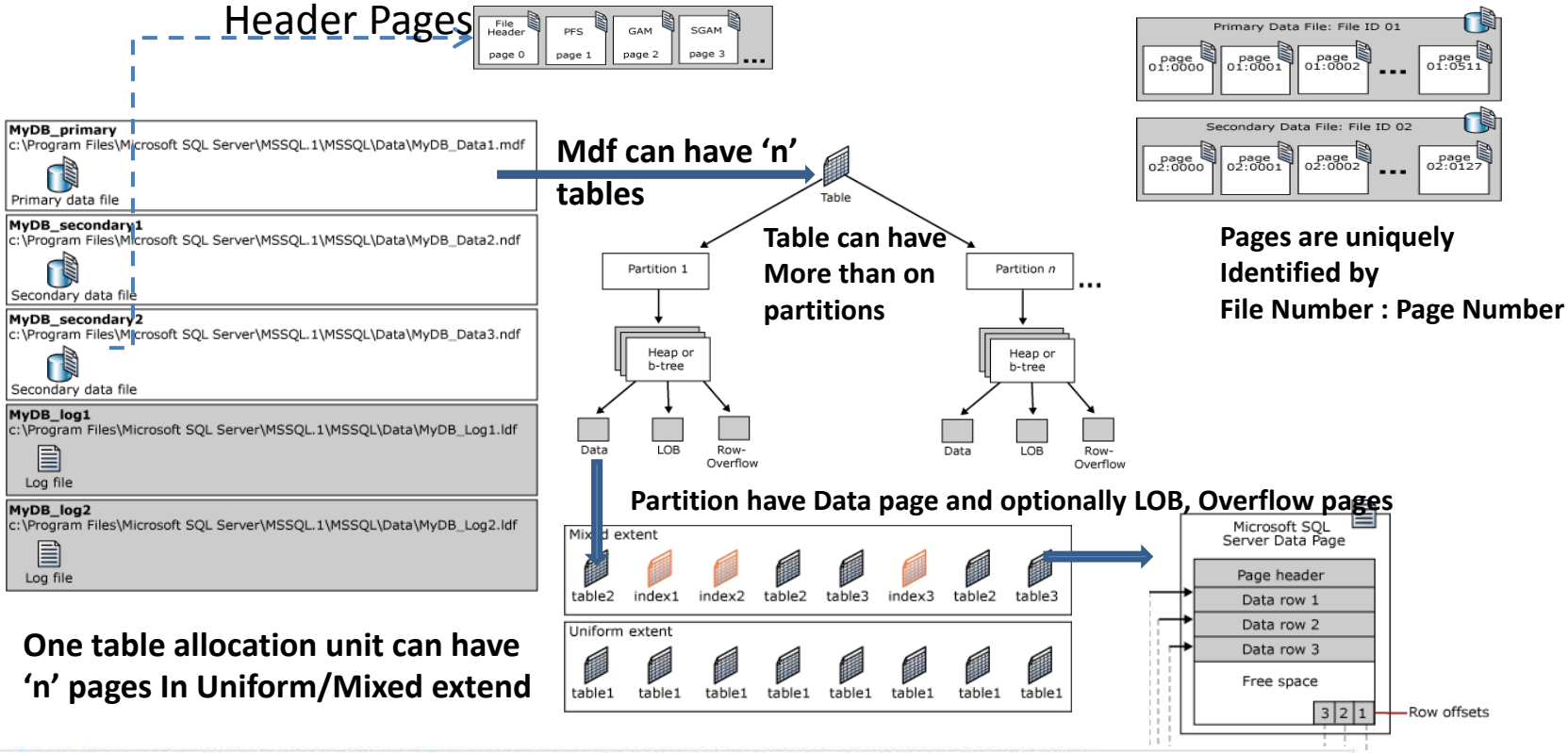
total pages allocated to heap is 7
1 for IAM page
6 for Data pages

Root_page reference is used for index structure. Not applicable for Heaps

name	object_id	index_id	total_pages	used_pages	data_pages	first_iam_page	first_page	root_page
tExample3	181575685	0	7	7	6	0xEB0000000100	0xEA0000000100	0x000000000000

Address of first IAM page and first data page in Hexadecimal

Database structure (high level)



0x30 00 1400 01000000E10CF400EA9D00002A000000 0500 00 0200 28003700 446F6E27742050616E6963536861726520616E6420456E6A6F79

2 Bytes		2 Bytes		2 Bytes	Bytes = #Columns / 8	2 Bytes	Bytes = 2 per Column	
StatusBitsA	StatusBitsB	Fixed Bytes Length	Fixed Length Data	# of Columns	Null Byte Map	# of Variable Length Columns	Column Offset Array	Variable Length Data
0x30	0x00	0x1400	0x01000000	0x0500	0x00	0x0200	0x28003700	0x446F6E27742050616E6963
			0xE10CF400EA9D0000		00000000			0x536861726520616E6420456E6A6F79
			0x2A000000					

DBCC Commands

Syntax:

DBCC EXTENTINFO (Extents Allocations)

DBCC extentinfo [({'database_name'| dbid | 0} [,{'table_name' | table_id} [, {'index_name' | index_id | -1}]])]

DBCC IND (Page allocations of one database object)

DBCC ind ({ 'dbname' | dbid }, { 'objname' | objid }, { indid | 0 | -1 | -2 })

DBCC PAGE (Anatomy of a page)

DBCC page ({'dbname' | dbid}, filenum, pagenum [, printopt={0|1|2|3}])

DBCC WRITEPAGE (Write a byte in a page directly)

DBCC WRITEPAGE ({'dbname' | dbid}, fileid, pageid, offset, length {1-8}, data [, directORbufferpool])

More DBCC Commands and Syntax: http://ss64.com/sql/dbcc_undocumented.html

DBCC EXTENTINFO and DBCC IND

```
DBCC TRACEON(3604)
DBCC EXTENTINFO('LearningInternals', 'tExample4')
DBCC TRACEOFF(3604)
```

file_id	page_id	pg_alloc	ext_size	object_id	index_id	partition_	partition_id	iam_chain_type	pfs_bytes
1	241	1	1	197575742	0	1	72057594039566300	In-row data	0x6100000000000000

```
DBCC TRACEON(3604)
DBCC IND('LearningInternals', 'tExample3', -1)
DBCC TRACEOFF(3604)
```

IAM Page : 235 for the table tExample3

6 data pages are referring IAM page number 235

IAM page type : 10
data page type : 1

Index level for heap structure is 0

PageFID	PagePID	IAMFID	IAMPID	ObjectID	IndexID	PartitionNumber	PartitionID	iam_chain_type	PageType	Index_level	NextPageFID	NextPagePID	PrevPageFID	PrevPagePID
1	235	NULL	NULL	181575685	0	1	72057594039500800	In-row data	10	NULL	0	0	0	0
1	234	1	235	181575685	0	1	72057594039500800	In-row data	1	0	0	0	0	0
1	236	1	235	181575685	0	1	72057594039500800	In-row data	1	0	0	0	0	0
1	237	1	235	181575685	0	1	72057594039500800	In-row data	1	0	0	0	0	0
1	238	1	235	181575685	0	1	72057594039500800	In-row data	1	0	0	0	0	0
1	239	1	235	181575685	0	1	72057594039500800	In-row data	1	0	0	0	0	0
1	240	1	235	181575685	0	1	72057594039500800	In-row data	1	0	0	0	0	0

Previous and next page pointers are zero for heap. This means heap pages are not linked with each other

DBCC PAGE

```
BUFFER:
BUF @0x04274154

bpage = 0x0652E000                  bhash = 0x00000000                  bpageno = (1:241)
bdbid = 8                           breferences = 0                     bUse1 = 19558
bstat = 0x1c0010b                   blog = 0x21bb79bb                   bnext = 0x00000000
```

Buffer

```
PAGE HEADER:
Page @0x0652E000

m_pageId = (1:241)                  m_headerVersion = 1                 m_type = 1
m_typeFlagBits = 0x4                m_level = 0                         m_flagBits = 0x8000
m_objId (AllocUnitId.idObj) = 93    m_indexId (AllocUnitId.idInd) = 256
Metadata: AllocUnitId = 72057594044022784
Metadata: PartitionId = 72057594039566336                               Metadata: IndexId = 0
Metadata: ObjectId = 197575742      m_prevPage = (0:0)                  m_nextPage = (0:0)
pminlen = 18                        m_slotCnt = 1                       m_freeCnt = 8055
m_freeData = 135                    m_reservedCnt = 0                   m_lsn = (29:18:19)
m_xactReserved = 0                  m_xdesId = (0:0)                    m_ghostRecCnt = 0
m_tornBits = 0

Allocation Status
GAM (1:2) = ALLOCATED               SGAM (1:3) = ALLOCATED
PFS (1:1) = 0x61 MIXED_EXT ALLOCATED  50_PCT_FULL                       DIFF (1:6) = CHANGED
ML (1:7) = NOT MIN_LOGGED
```

Header

```
DATA:

Slot 0, Offset 0x60, Length 39, DumpStyle BYTE

Record Type = PRIMARY_RECORD        Record Attributes =  NULL_BITMAP VARIABLE_COLUMNS

00000000:   30001200 e9030000 44455054 3031d430 †0.......DEPT01.0
00000010:   00000500 e0020020 00270041 41414141 †....... .'.AAAAA
00000020:   4348454e 4e4149††††††††††††††††††††††CHENNAI
```

Data (Slots)

```
OFFSET TABLE:

Row - Offset
0 (0x0) - 96 (0x60)
```

Row Offset Array

DBCC PAGE

```
BUFFER:
BUF @0x04274154

bpage = 0x0652E000                  bhash = 0x00000000                  bpageno = (1:241)
bdbid = 8                           breferences = 0                     bUse1 = 19558
bstat = 0x1c0010b                   blog = 0x21bb79bb                   bnext = 0x00000000
```

Buffer

```
PAGE HEADER:
Page @0x0652E000

m_pageId = (1:241)                  m_headerVersion = 1                 m_type = 1
m_typeFlagBits = 0x4                m_level = 0                         m_flagBits = 0x8000
m_objId (AllocUnitId.idObj) = 93    m_indexId (AllocUnitId.idInd) = 256
Metadata: AllocUnitId = 72057594044022784
Metadata: PartitionId = 72057594039566336                               Metadata: IndexId = 0
Metadata: ObjectId = 197575742      m_prevPage = (0:0)                  m_nextPage = (0:0)
pminlen = 18                        m_slotCnt = 1                       m_freeCnt = 8055
m_freeData = 135                    m_reservedCnt = 0                   m_lsn = (29:18:19)
m_xactReserved = 0                  m_xdesId = (0:0)                    m_ghostRecCnt = 0
m_tornBits = 0

Allocation Status
GAM (1:2) = ALLOCATED               SGAM (1:3) = ALLOCATED
PFS (1:1) = 0x61 MIXED_EXT ALLOCATED  50_PCT_FULL                       DIFF (1:6) = CHANGED
ML (1:7) = NOT MIN_LOGGED
```

Header

```
DATA:

Slot 0, Offset 0x60, Length 39, DumpStyle BYTE

Record Type = PRIMARY_RECORD        Record Attributes =  NULL_BITMAP VARIABLE_COLUMNS

00000000:   30001200 e9030000 44455054 3031d430 †0.......DEPT01.0
00000010:   00000500 e0020020 00270041 41414141 †....... .'.AAAAA
00000020:   4348454e 4e4149††††††††††††††††††††††CHENNAI
```

Data (Slots)

```
OFFSET TABLE:

Row - Offset
0 (0x0) - 96 (0x60)
```

Row Offset Array

DBCC WRITEPAGE

```sql
CREATE TABLE [Test] (
[c1] INT IDENTITY,
[c2] CHAR (8000) DEFAULT 'a');
GO

INSERT INTO [Test] DEFAULT VALUES;
GO

DBCC IND (N'CorruptDB', N'Test', -1);
GO
ALTER DATABASE [CorruptDB] SET SINGLE_USER;
GO
DBCC WRITEPAGE (N'CorruptDB', 1, 153, 4000, 1, 0x45, 1);
GO

SELECT * FROM Test
```

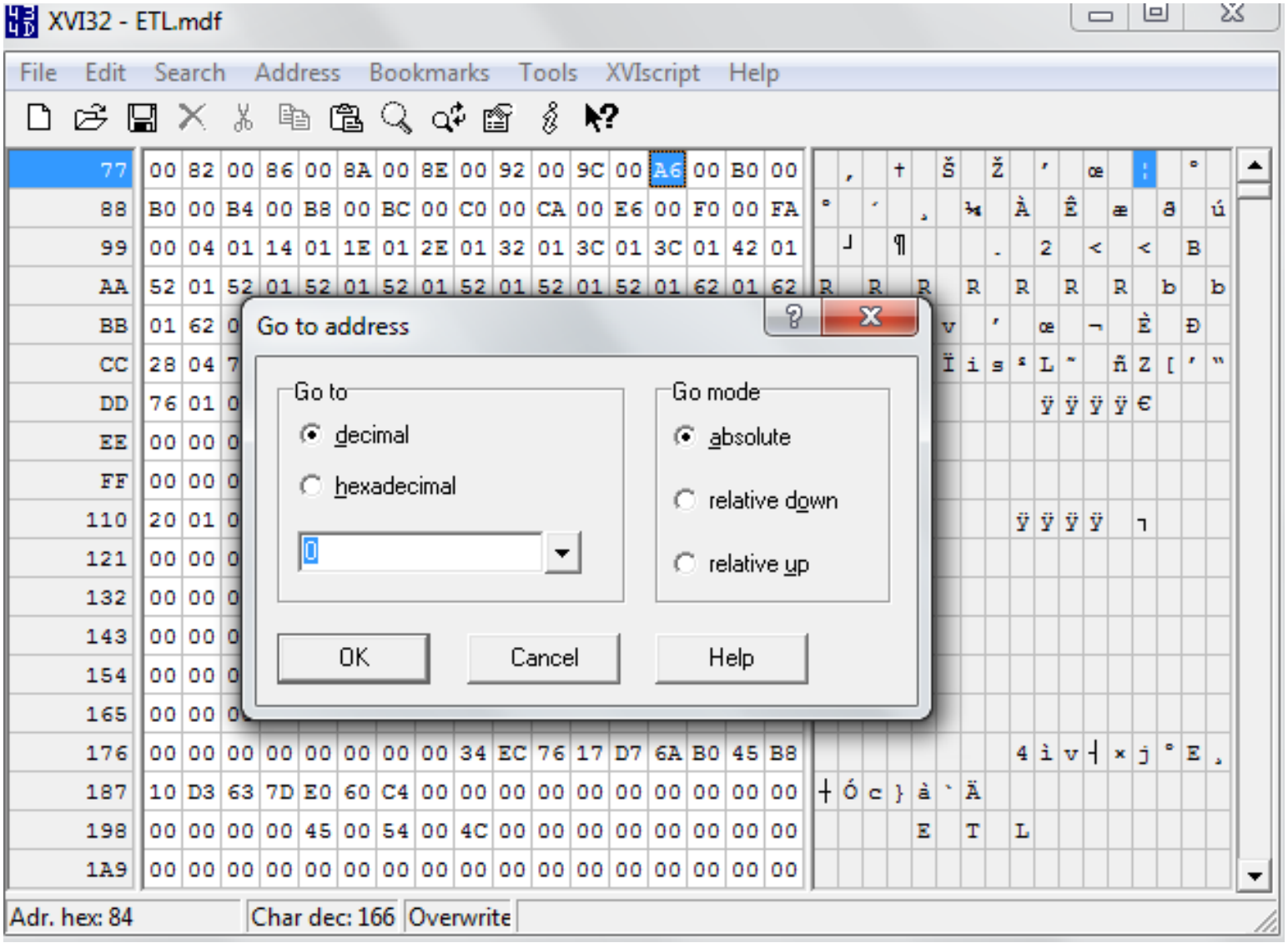
Results | Messages

```
Msg 824, Level 24, State 2, Line 1
SQL Server detected a logical consistency-based I/O error: incorrect checksum (expected: 0x41bfae55; actual: 0x41bfcb55)
```

Note:
Database PageVerify = CHECKSUM

Source: http://www.sqlskills.com/blogs/paul/dbcc-writepage/

Hex Editor (Ex: XVI32)



Download: http://www.chmaas.handshake.de/delphi/freeware/xvi32/xvi32.htm

What Next

DBCC CHECKDB Internals

Thanks