深入理解SQL Server存储结构

陈畅亮

2015.4

DTCC

2015中国数据库技术大会

DATABASE TECHNOLOGY CONFERENCE CHINA 2015

大数据技术探索和价值发现

个人简介

- DBA
- SQL Server MVP
- 数据库/大数据/自动化运维
- **联系方式**
  - 微博：听风吹雨-ccl
  - 博客：听风吹雨

提纲

- 几个工具&几个概念(Page/RID)
- 堆表记录的存储结构
- char VS nchar VS nvarchar
- 非聚集索引的存储结构
- 聚集索引表的存储结构
- 行溢出存储结构
- LOB存储结构

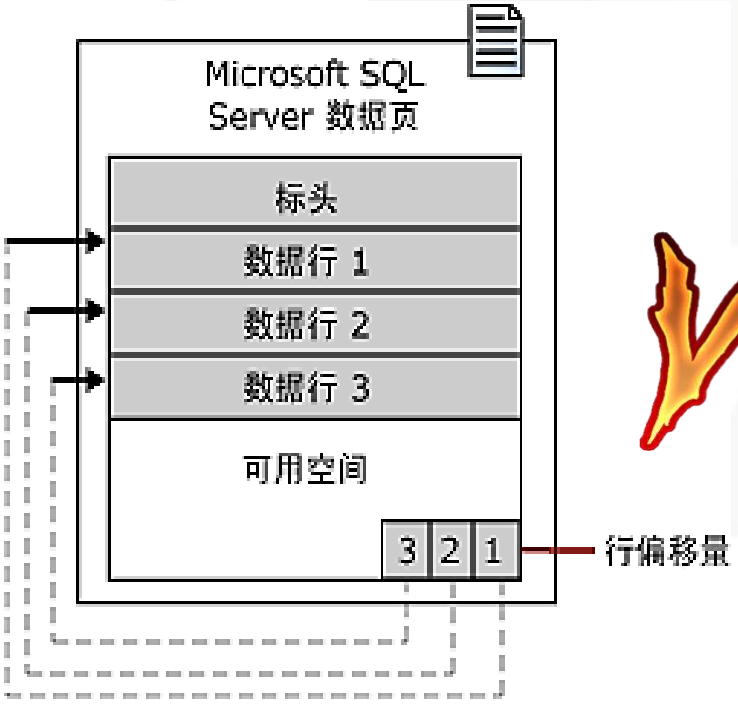
几个工具

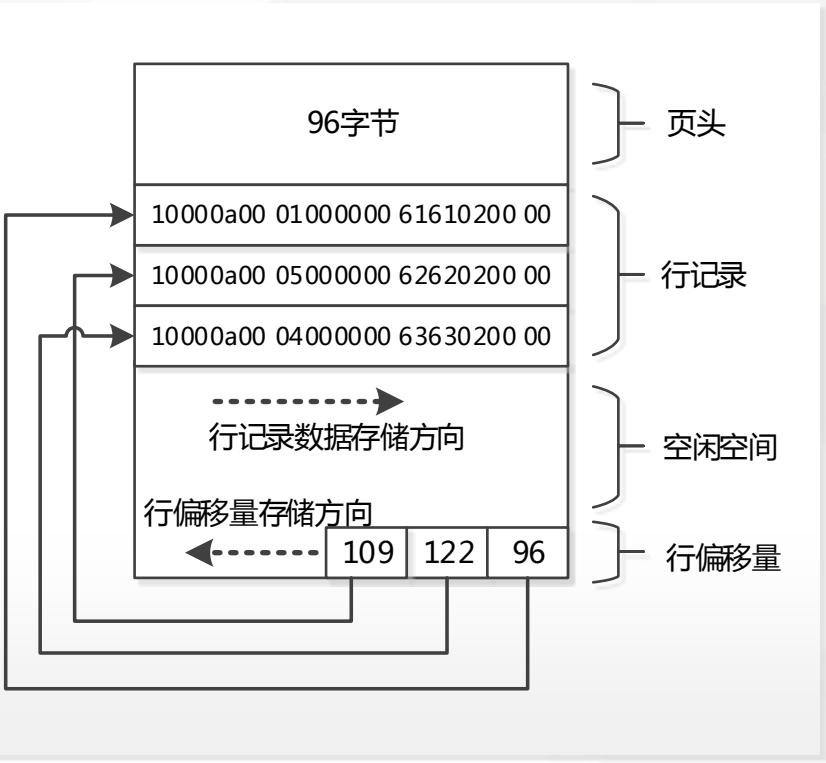
DBCC

Database Consloe Commands
数据库控制台命令

- DBCC HELP
- DBCC IND
- DBCC PAGE
- Winhex
- Internals Viewer for SQL Server

Page

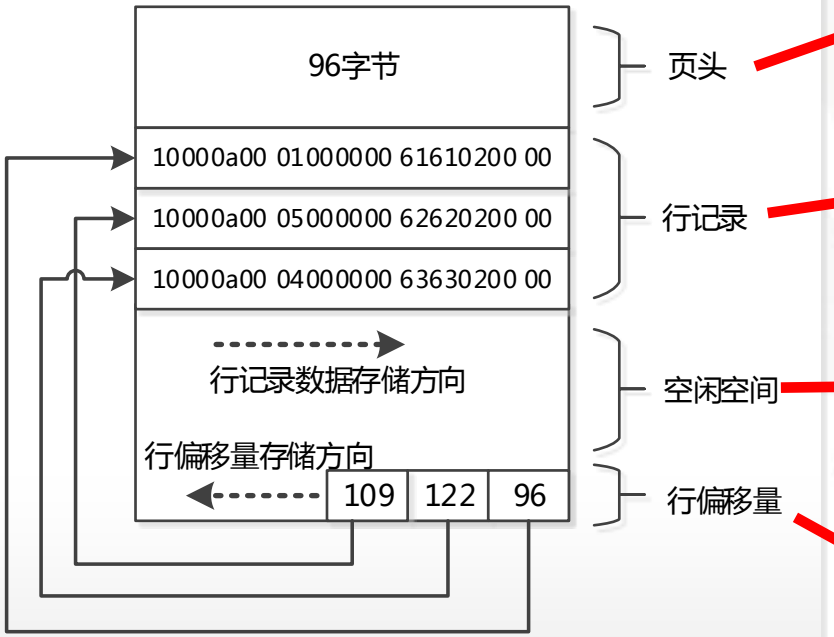




MSDN

个人理解

Page结构示意图



- 固定大小为96个字节
- 存储堆表或者索引数据
- 行记录与行偏移量公用的存储区域
- 从右往左称为槽（slot）

RID

- 什么是RID？
- RID:Row Identifier(行标识符)
- 十六进制RID = 页号 + 文件号 + 槽号
- 8个字节 = 4 + 2 + 2

- RID的存储结构？

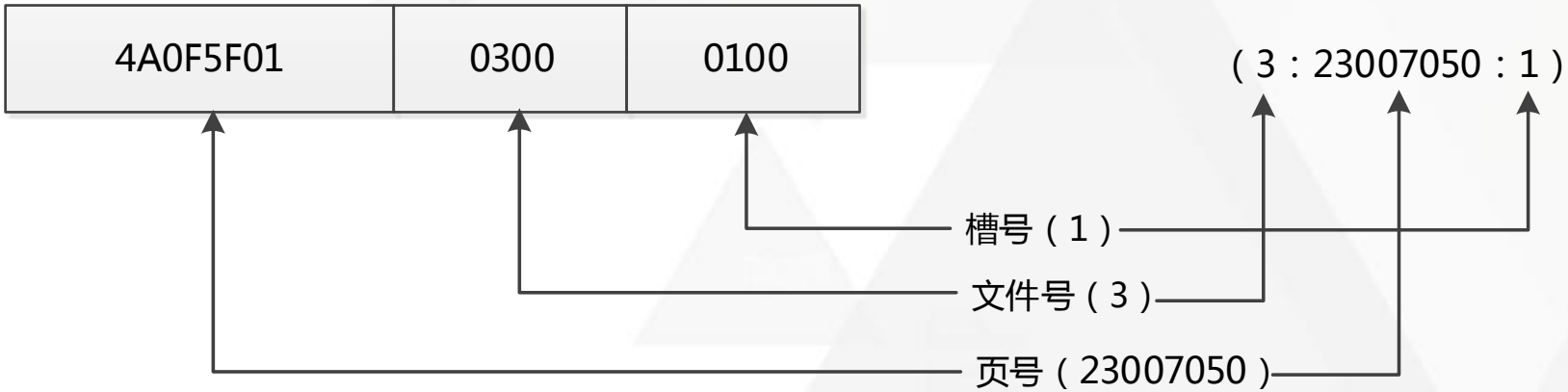
RID长什么样？

```
01 SELECT ID, %%physloc%% AS RID_16,
02   sys.fn_PhysLocFormatter(%%physloc%%) AS RID_10
03 FROM dbo.MyTableName
04 GO
```

	ID	RID_16	RID_10
1	996	0x4A0F5F0103000000	(3:23007050:0)
2	998	0x4A0F5F0103000100	(3:23007050:1)
3	1000	0xB3E4610103000000	(3:23192755:0)
4	1001	0xAAE4610103000000	(3:23192746:0)
5	1002	0xB4E4610103000000	(3:23192756:0)
6	1003	0xB4E4610103000100	(3:23192756:1)
7	1005	0xBAE4610103000000	(3:23192762:0)
8	1006	0xBAE4610103000100	(3:23192762:1)
9	1007	0xBAE4610103000200	(3:23192762:2)
10	1008	0xBAE4610103000300	(3:23192762:3)
11	1021	0xBAE4610103000400	(3:23192762:4)
12	1029	0xBAE4610103000500	(3:23192762:5)
13	1031	0xB7E4610103000000	(3:23192759:0)
14	1032	0xB7E4610103000100	(3:23192759:1)

RID结构示例

十六进制的RID：0x4A0F5F0103000100



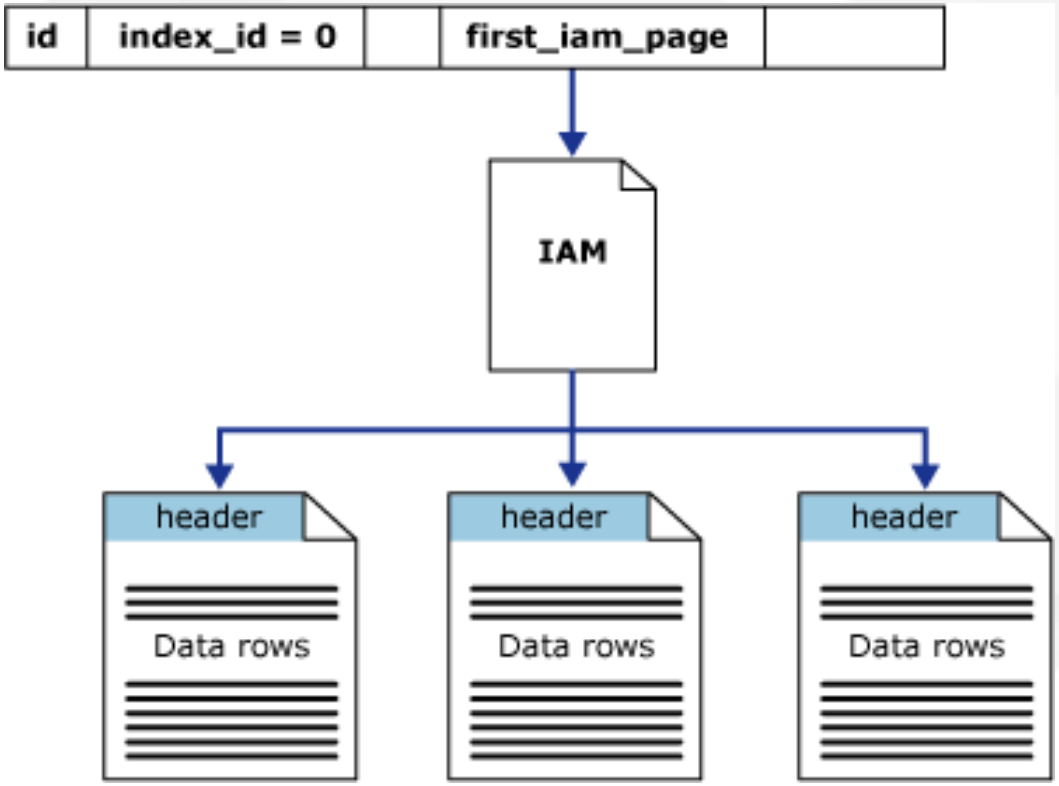
以1字节（byte）为一个单位进行逆序

RID的页号转换

8个字节的页号：0x4A0F5F01

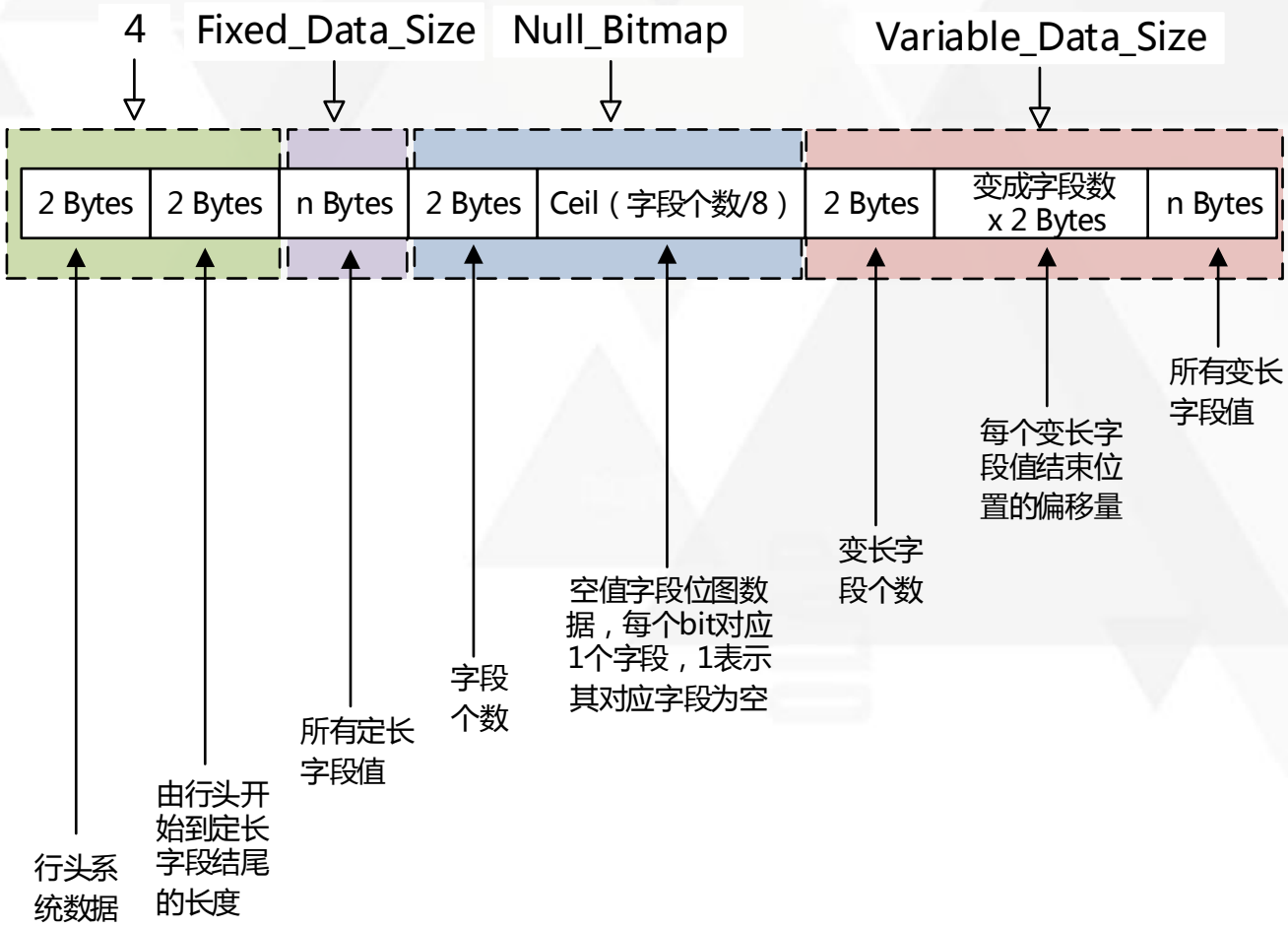
0x4A0F5F01 → 0x015F0F4A → 23007050

堆表记录的存储结构



MSDN

堆表记录的存储结构


4
Fixed_Data_Size
Null_Bitmap
Variable_Data_Size
2 Bytes
2 Bytes
n Bytes
2 Bytes
Ceil（字段个数/8）
2 Bytes
变成字段数
x 2 Bytes
n Bytes
行头系统数据
由行头开始到定长字段结尾的长度
所有定长字段值
字段个数
空值字段位图数据，每个bit对应1个字段，1表示其对应字段为空
变长字段个数
每个变长字段值结束位置的偏移量
所有变长字段值

char VS nchar VS nvarchar

char [(*n*)]

固定长度的非 Unicode 字符串数据，*n* 用于定义字符串长度，存储大小为 *n* 字节。

nchar [(n)]

固定长度的 Unicode 字符串数据，*n* 用于定义字符串长度，存储大小为 *n* 字节的两倍。

nvarchar [(n | max)]

可变长度的 Unicode 字符串数据。 n 用于定义字符串长度，存储大小（以字节为单位）是所输入数据实际长度的两倍 + 2 个字节。

```
01 USE master
02 GO
03 IF EXISTS(SELECT name FROM sys.databases WHERE name = 'Heap_Record')
04 DROP DATABASE Heap_Record
05 GO
06 CREATE DATABASE Heap_Record
07 GO
08
09 USE Heap_Record
10 GO
11 CREATE TABLE [dbo].[HeapPage_char](
12    [id] [int] IDENTITY(1,1) NOT NULL,
13    [names] [char](10) NULL
14 ) ON [PRIMARY]
15 GO
16
17 CREATE TABLE [dbo].[HeapPage_nchar](
18    [id] [int] IDENTITY(1,1) NOT NULL,
19    [names] [nchar](10) NULL
20 ) ON [PRIMARY]
21 GO
22
23 CREATE TABLE [dbo].[HeapPage_nvarchar](
24    [id] [int] IDENTITY(1,1) NOT NULL,
25    [names] [nvarchar](10) NULL
26 ) ON [PRIMARY]
27 GO
28
29 INSERT INTO [HeapPage_char](names) values('XX')
30 GO
31 INSERT INTO [HeapPage_char](names) values('XXXX')
32 GO 2
33
34 INSERT INTO [HeapPage_nchar](names) values('XX')
35 GO
36 INSERT INTO [HeapPage_nchar](names) values('XXXX')
37 GO 2
38
39 INSERT INTO [HeapPage_nvarchar](names) values('XX')
40 GO
41 INSERT INTO [HeapPage_nvarchar](names) values('XXXX')
42 GO 2
```

堆表行记录结构(char)

```
DBCC IND(Heap_Record,HeapPage_char,-1)
```

结果 消息

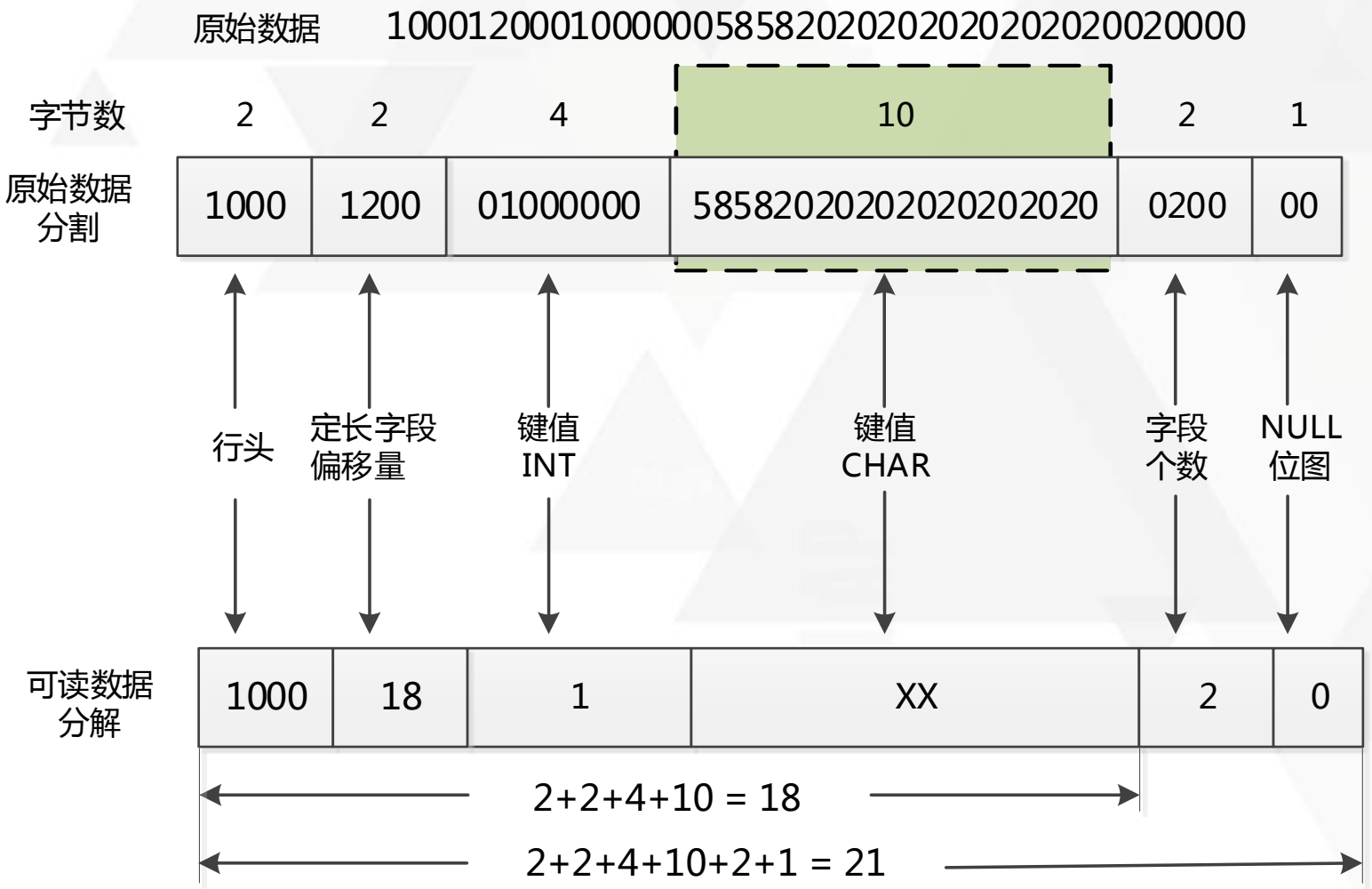
	PageFID	PagePID	IAMFID	IAMPID	ObjectID	IndexID	PartitionNumber	PartitionID	iam_chain_type	PageType
1	1	89	NULL	NULL	2105058535	0	1	72057594038779904	In-row data	10
2	1	80	1	89	2105058535	0	1	72057594038779904	In-row data	1

```
01 DBCC TRACEON(3604)
02 DBCC PAGE(Heap_Record,1,80,1)
```

堆表行记录结构(char)

```
01 DATA:
02 Slot 0, Offset 0x60, Length 21, DumpStyle BYTE
03 Record Type = PRIMARY_RECORD        Record Attributes =  NULL_BITMAP
   Record Size = 21
04 Memory Dump @0x6523C060
05 00000000:   10001200 01000000 58582020 20202020 ?.......XX
06 00000010:   20200200 00???????????????????????? ...
07
08 Slot 1, Offset 0x75, Length 21, DumpStyle BYTE
09 Record Type = PRIMARY_RECORD        Record Attributes =  NULL_BITMAP
   Record Size = 21
10 Memory Dump @0x6523C075
11 00000000:   10001200 02000000 58585858 20202020 ?.......XXXX
12 00000010:   20200200 00???????????????????????? ...
13
14 Slot 2, Offset 0x8a, Length 21, DumpStyle BYTE
15 Record Type = PRIMARY_RECORD        Record Attributes =  NULL_BITMAP
   Record Size = 21
16 Memory Dump @0x6523C08A
17 00000000:   10001200 03000000 58585858 20202020 ?.......XXXX
18 00000010:   20200200 00???????????????????????? ...
19
20 OFFSET TABLE:
21 Row - Offset
22 2 (0x2) - 138 (0x8a)
23 1 (0x1) - 117 (0x75)
24 0 (0x0) - 96 (0x60)
```

堆表行记录结构(char)



堆表行记录结构(nchar)

```
DBCC IND(Heap_Record,HeapPage_nchar,-1)
```


结果 消息

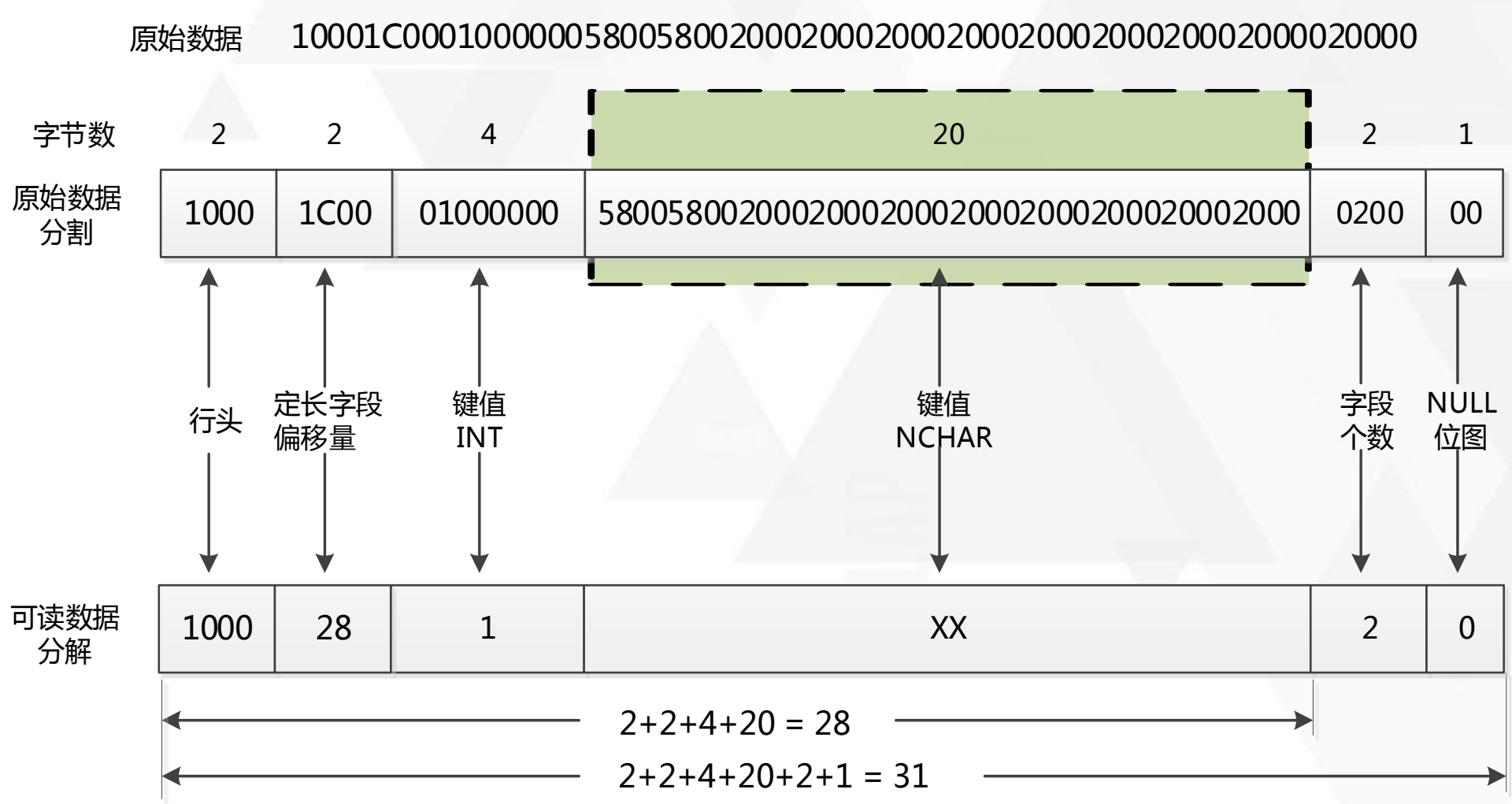
	PageFID	PagePID	IAMFID	IAMPID	ObjectID	IndexID	PartitionNumber	PartitionID	iam_chain_type	PageType
1	1	93	NULL	NULL	2121058592	0	1	72057594038845440	In-row data	10
2	1	90	1	93	2121058592	0	1	72057594038845440	In-row data	1


```
01 DBCC TRACEON(3604)
02 DBCC PAGE(Heap_Record,1,90,1)
```

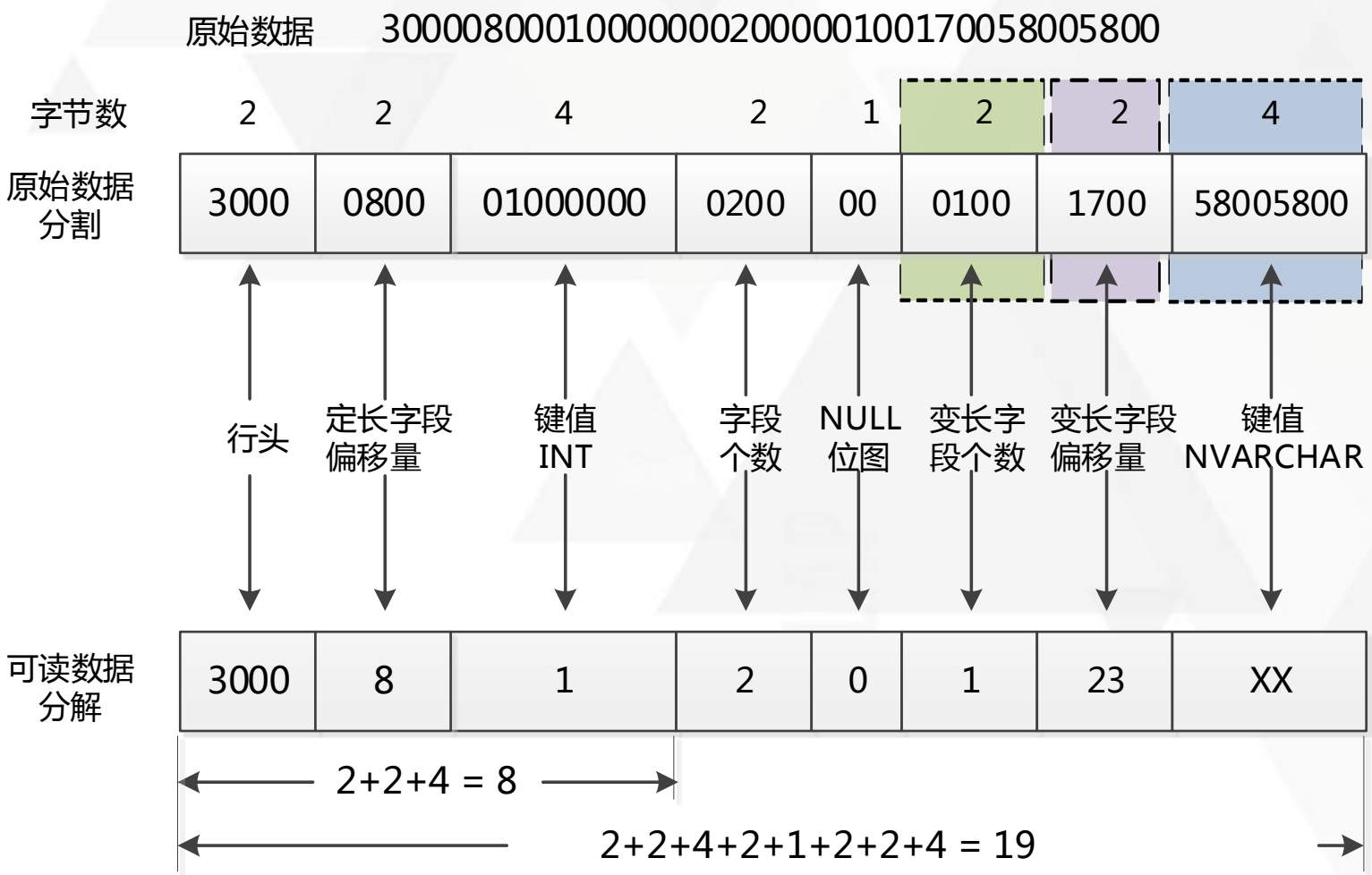
堆表行记录结构(nchar)

```
01 DATA:
02 Slot 0, Offset 0x60, Length 31, DumpStyle BYTE
03 Record Type = PRIMARY_RECORD        Record Attributes =  NULL_BITMAP
   Record Size = 31
04 Memory Dump @0x61ADC060
05 00000000:   10001c00 01000000 58005800 20002000 ?.......X.X. . .
06 00000010:   20002000 20002000 20002000 020000???? . . . . . ....
07
08 Slot 1, Offset 0x7f, Length 31, DumpStyle BYTE
09 Record Type = PRIMARY_RECORD        Record Attributes =  NULL_BITMAP
   Record Size = 31
10 Memory Dump @0x61ADC07F
11 00000000:   10001c00 02000000 58005800 58005800 ?.......X.X.X.X.
12 00000010:   20002000 20002000 20002000 020000???? . . . . . ....
13
14 Slot 2, Offset 0x9e, Length 31, DumpStyle BYTE
15 Record Type = PRIMARY_RECORD        Record Attributes =  NULL_BITMAP
   Record Size = 31
16 Memory Dump @0x61ADC09E
17 00000000:   10001c00 03000000 58005800 58005800 ?.......X.X.X.X.
18 00000010:   20002000 20002000 20002000 020000???? . . . . . ....
19
20 OFFSET TABLE:
21 Row - Offset
22 2 (0x2) - 158 (0x9e)
23 1 (0x1) - 127 (0x7f)
24 0 (0x0) - 96 (0x60)
```

堆表行记录结构(nchar)



堆表行记录结构(nvarchar)

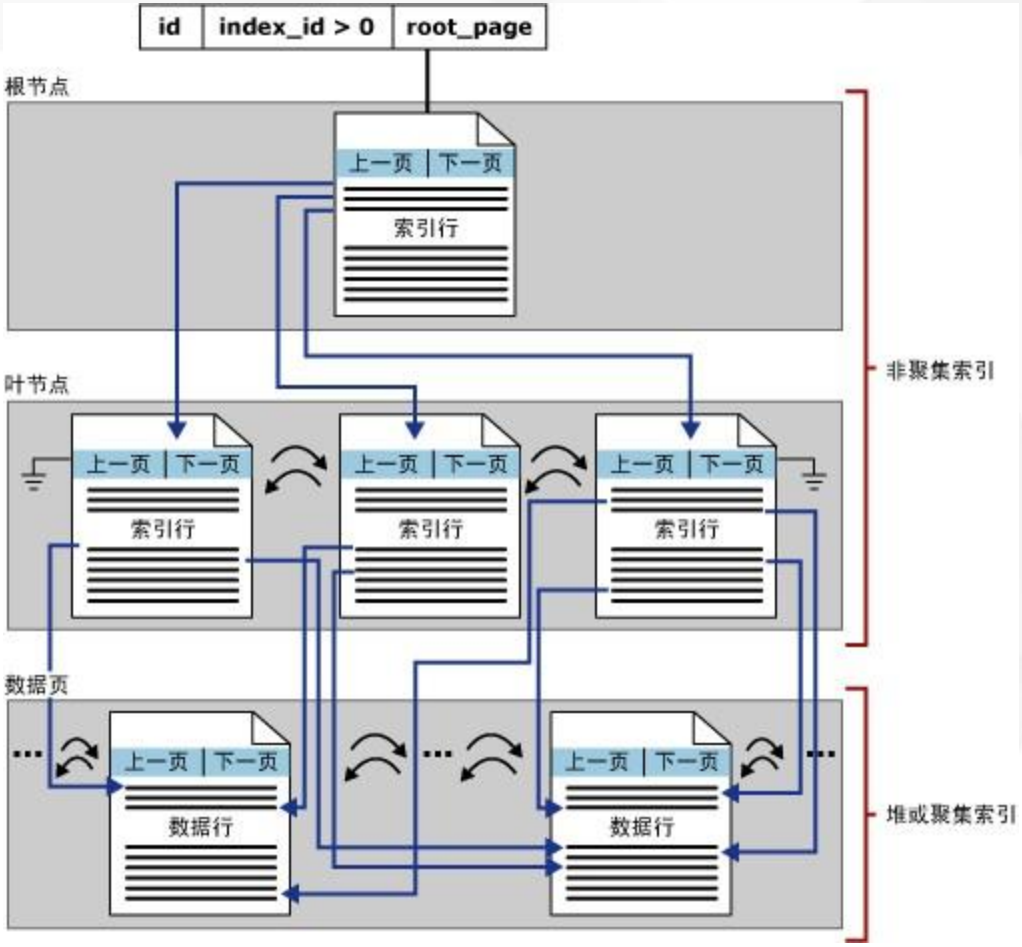


char VS nchar VS nvarchar

数据类型	字符编码	存储大小	“X”
char(*n=10*)	非 Unicode	固定10个字节	58
nchar(*n=10*)	Unicode	固定20个字节	5800
nvarchar(*n=10*)	Unicode	非固定字节	5800

数据类型	存储值“XX”	占用空间
char(*n=10*)	58582020202020202020	10
nchar(*n=10*)	58005800200020002000200020002000200020002000	20
nvarchar(*n=10*)	58005800	4

非聚集索引存储结构



MSDN

(堆表)非聚集索引存储结构

```
01 --Create Database
02 USE master
03 GO
04 IF EXISTS(SELECT name FROM sys.databases
     WHERE name = 'Heap_NonCluster')
05 DROP DATABASE Heap_NonCluster
06 GO
07 CREATE DATABASE Heap_NonCluster
08 GO
09
10 USE Heap_NonCluster
11 GO
12
13 --Heap
14 CREATE TABLE Page_NonCluster
15 (
16   a INT IDENTITY,
17   b CHAR(5) DEFAULT 'xxxxx',
18   c VARCHAR(50) DEFAULT 'ccccc'
19 )
20 GO
21
22 --Insert
23 INSERT INTO Page_NonCluster DEFAULT VALUES
24 GO 2000
25
26 --Index
27 CREATE NONCLUSTERED INDEX idx_ab ON Page_NonCluster(a,b)
```

(堆表)非聚集索引idx_ab(ind)

```
01 --idx_ab
02 DBCC IND (Heap_NonCluster, Page_NonCluster, 5)
03 GO
```

	PageFID	PagePID	IAMFID	IAMPID	IndexID	iam_chain_type	PageType	IndexLevel
1	1	418	NULL	NULL	5	In-row data	10	NULL
2	1	417	1	418	5	In-row data	2	0
3	1	419	1	418	5	In-row data	2	0
4	1	420	1	418	5	In-row data	2	1
5	1	421	1	418	5	In-row data	2	0
6	1	422	1	418	5	In-row data	2	0
7	1	423	1	418	5	In-row data	2	0
8	1	424	1	418	5	In-row data	2	0

(堆表)非聚集索引idx_ab(page)

```
01 DBCC TRACEON (3604)
02 GO
03 DBCC PAGE(Heap_NonCluster, 1, 420, 3)
04 GO
05 DBCC PAGE(Heap_NonCluster, 1, 420, 1)
06 GO
```

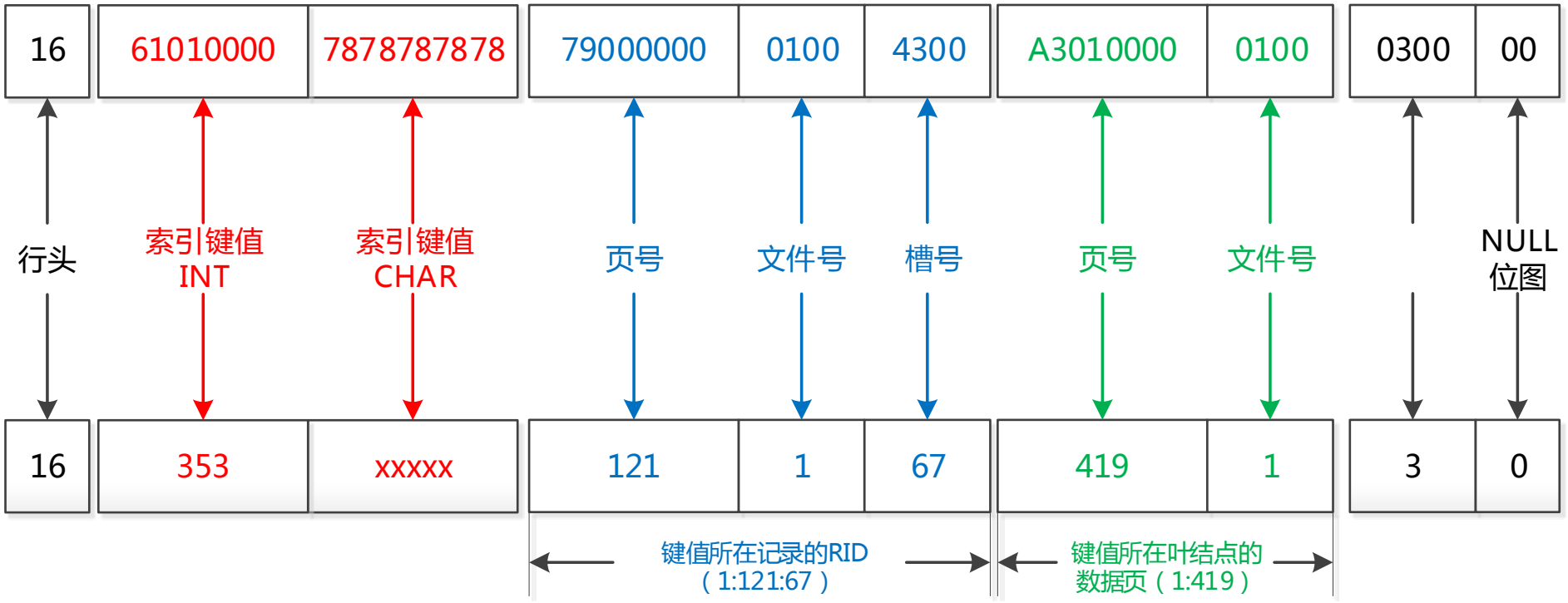
	FileId	PageId	Row	Level	ChildFileId	ChildPageId	a (key)	b (key)	HEAP RID (key)	KeyHashValue	Row Size
1	1	420	0	1	1	417	NULL	NULL	NULL	NULL	27
2	1	420	1	1	1	419	353	xxxxx	0x7900000001004300	NULL	27
3	1	420	2	1	1	421	705	xxxxx	0x7E00000001008600	NULL	27
4	1	420	3	1	1	422	1057	xxxxx	0x7F0000000100C900	NULL	27
5	1	420	4	1	1	423	1409	xxxxx	0x8E00000001000C01	NULL	27
6	1	420	5	1	1	424	1761	xxxxx	0x9100000001003200	NULL	27

(堆表)非聚集索引idx_ab(Root data)

```
01 DATA:
02 Slot 0, Offset 0x60, Length 27, DumpStyle BYTE
03
04 Record Type = INDEX_RECORD
   Record Attributes =  NULL_BITMAP
   Record Size = 27
05 Memory Dump @0x000000001004A060
06 0000000000000000:
    16010000 00787878 78787700 00000100 0000a101
07 .....xxxxxw.........
08 0000000000000014:   00000100 030000           .......
09
10 Slot 1, Offset 0x7b, Length 27, DumpStyle BYTE
11 Record Type = INDEX_RECORD
   Record Attributes =  NULL_BITMAP
   Record Size = 27
12 Memory Dump @0x000000001004A07B
13 0000000000000000:
    16610100 00787878 78787900 00000100 4300a301
14 .a...xxxxxy.....C...
15 0000000000000014:   00000100 030000           .......
```

(堆表)非聚集索引idx_ab(Root)

原始数据　166101000078787878787900000001004300A30100000100030000

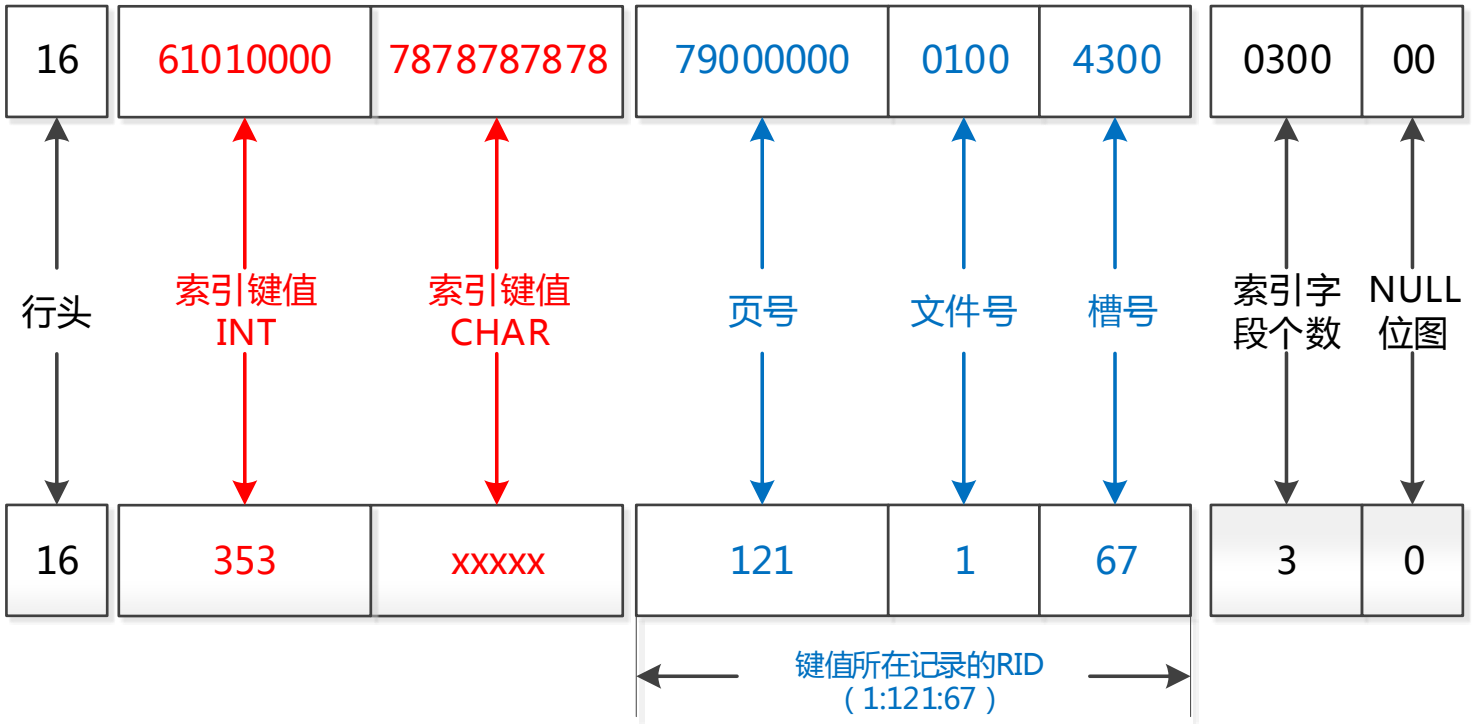


(堆表)非聚集索引idx_ab(Leaf data)

```
01 DATA:
02 Slot 0, Offset 0x60, Length 21, DumpStyle BYTE
03 Record Type = INDEX_RECORD         Record Attributes =
NULL_BITMAP    Record Size = 21
04 Memory Dump @0x000000001129A060
05 0000000000000000:
    16610100 00787878 78787900 00000100 43000300
06 .a...xxxxxy.....C...
07 0000000000000014:00              .
08
09 Slot 1, Offset 0x75, Length 21, DumpStyle BYTE
10 Record Type = INDEX_RECORD         Record Attributes =
NULL_BITMAP    Record Size = 21
11 Memory Dump @0x000000001129A075
12 0000000000000000:
    16620100 00787878 78787900 00000100 44000300
13 .b...xxxxxy.....D...
14 0000000000000014:   00
```

(堆表)非聚集索引idx_ab(Leaf)

原始数据　16610100007878787878790000000100430003000 0

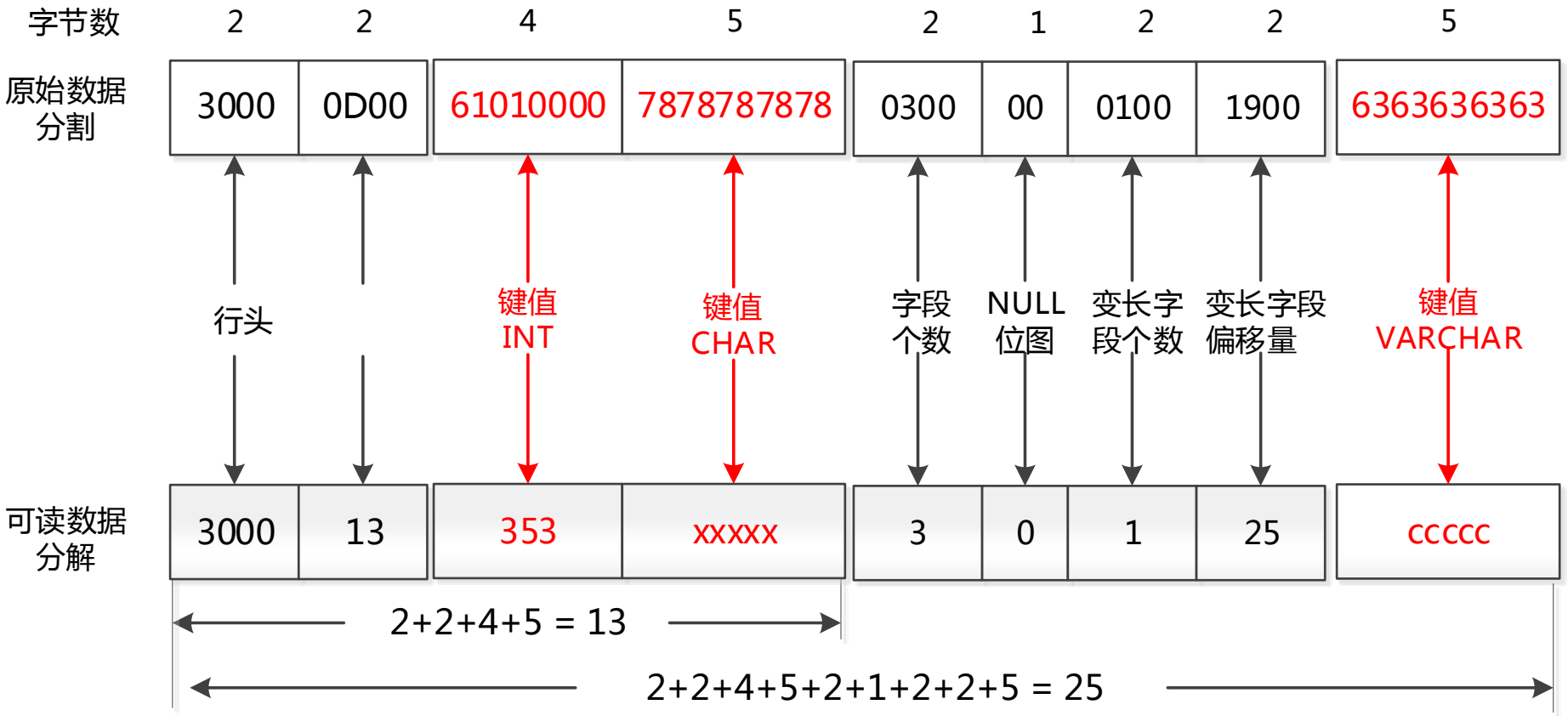


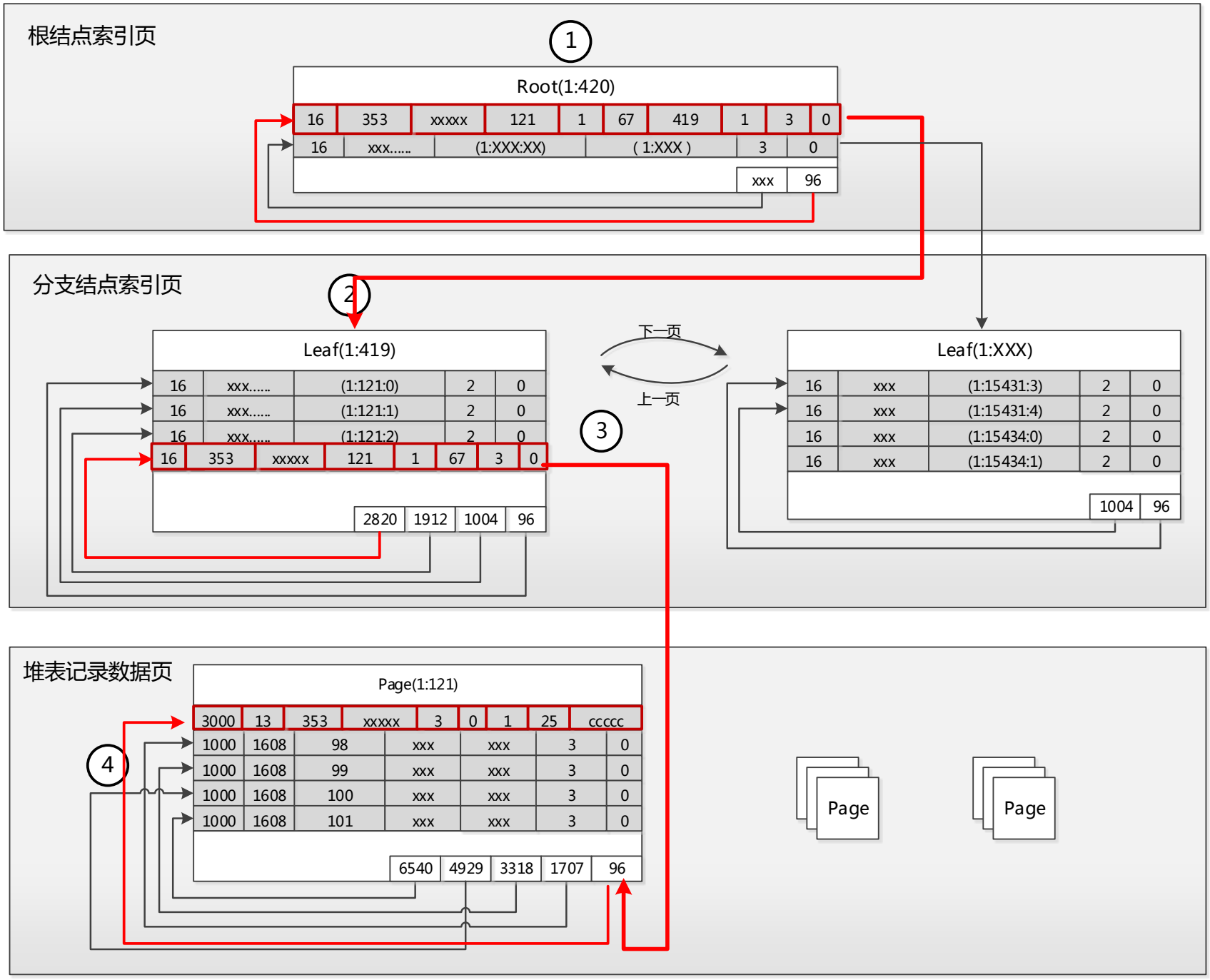
(堆表)非聚集索引idx_ab(row data)

```
01 DATA:
02 Slot 0, Offset 0x60, Length 25, DumpStyle BYTE
03 Record Type = PRIMARY_RECORD
   Record Attributes =  NULL_BITMAP VARIABLE_COLUMNS
04 Record Size = 25
05 Memory Dump @0x00000000122BA060
06 0000000000000000:
    30000d00 61010000 78787878 78030000 01001900
0.......xxxxx.......
07 0000000000000014:   63636363 63
ccccc
08
09 Slot 1, Offset 0x79, Length 25, DumpStyle BYTE
10 Record Type = PRIMARY_RECORD
   Record Attributes =  NULL_BITMAP VARIABLE_COLUMNS
11 Record Size = 25
12 Memory Dump @0x00000000122BA079
13 0000000000000000:
    30000d00 62010000 78787878 78030000 01001900
0.......xxxxx.......
14 0000000000000014:   63636363 63
ccccc
```

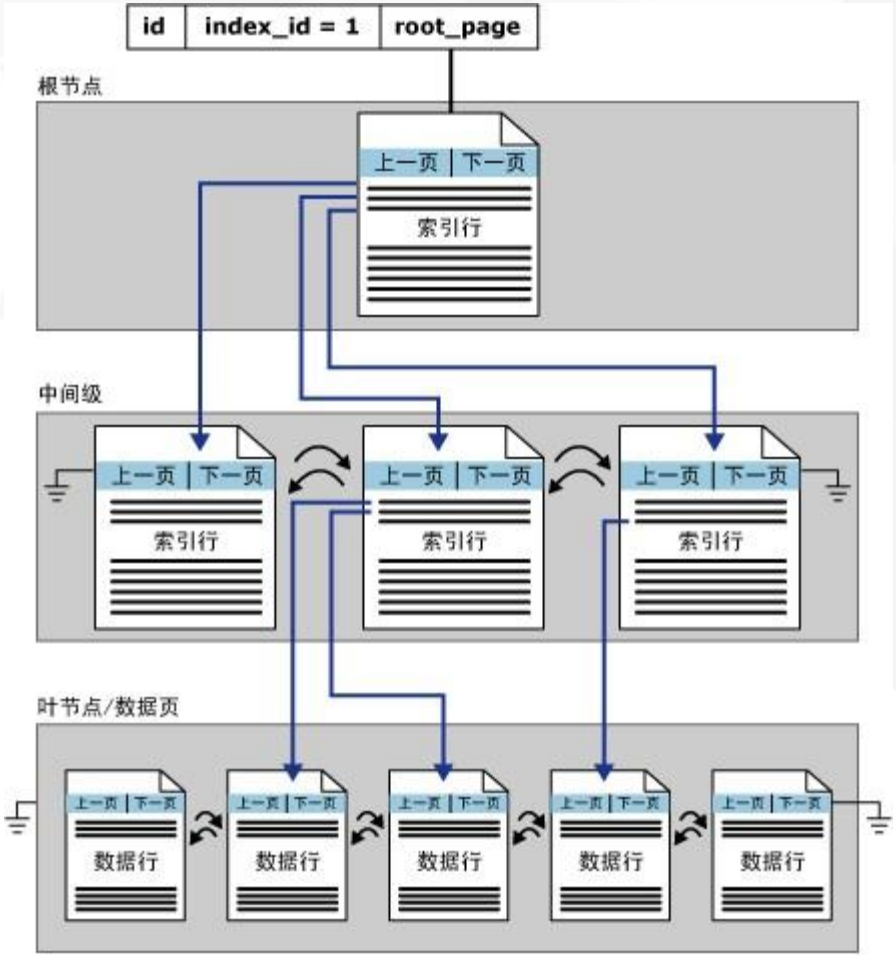
(堆表)非聚集索引idx_ab(row)

原始数据　30000D006101000078787878780300000100190063636363




根结点索引页
1
Root(1:420)
16 353 xxxxx 121 1 67 419 1 3 0
16 xxx..... (1:XXX:XX) (1:XXX) 3 0
xxx 96
分支结点索引页
2
Leaf(1:419)
16 xxx..... (1:121:0) 2 0
16 xxx..... (1:121:1) 2 0
16 xxx..... (1:121:2) 2 0
16 353 xxxxx 121 1 67 3 0
2820 1912 1004 96
下一页
上一页
Leaf(1:XXX)
16 xxx (1:15431:3) 2 0
16 xxx (1:15431:4) 2 0
16 xxx (1:15434:0) 2 0
16 xxx (1:15434:1) 2 0
1004 96
3
堆表记录数据页
Page(1:121)
3000 13 353 xxxxx 3 0 1 25 ccccc
1000 1608 98 xxx xxx 3 0
1000 1608 99 xxx xxx 3 0
1000 1608 100 xxx xxx 3 0
1000 1608 101 xxx xxx 3 0
6540 4929 3318 1707 96
4
Page
Page

聚集索引表的存储结构



MSDN

聚集索引表的存储结构

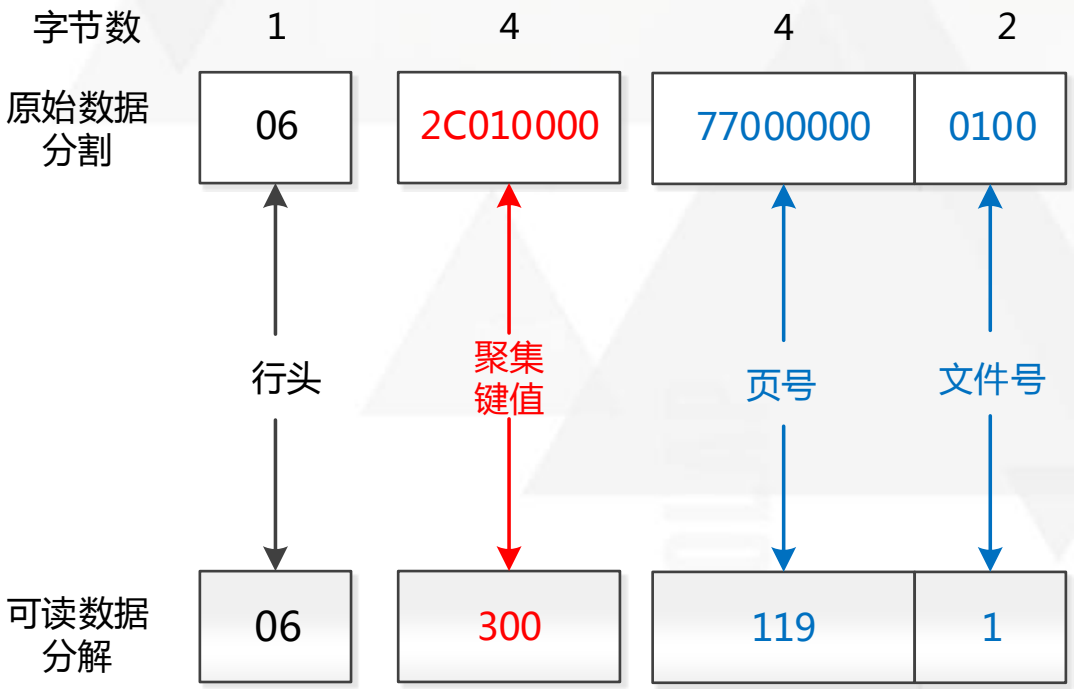
```
01 --Create Database
02 USE master
03 GO
04 IF EXISTS(SELECT name FROM sys.databases WHERE name =
'ClusterIndex')
05 DROP DATABASE ClusterIndex
06 GO
07 CREATE DATABASE ClusterIndex
08 GO
09
10 USE ClusterIndex
11 GO
12
13 --Create Table
14 CREATE TABLE [dbo].[Page_Cluster](
15  [a] [int] IDENTITY(1,1) NOT NULL,
16  [b] [char](5) DEFAULT 'xxxxx',
17  [c] [varchar](50) DEFAULT 'ccccc',
18 CONSTRAINT [PK_Page_Cluster] PRIMARY KEY CLUSTERED([id])
19 )
20 GO
21
22 --Insert
23 INSERT INTO Page_Cluster DEFAULT VALUES
24 GO 2000
```

聚集索引(Root data)

```
01 DATA:
02 Slot 0, Offset 0x60, Length 11, DumpStyle BYTE
03 Record Type = INDEX_RECORD
   Record Attributes =              Record Size = 11
04 Memory Dump @0x000000001197A060
05 0000000000000000:   0646bef3 fe5d0000 000100
.F...]......
06
07 Slot 1, Offset 0x6b, Length 11, DumpStyle BYTE
08 Record Type = INDEX_RECORD
   Record Attributes =              Record Size = 11
09 Memory Dump @0x000000001197A06B
10 0000000000000000:   062c0100 00770000 000100
.,...w......
```

聚集索引(Root)

原始数据　062C0100007700000000100

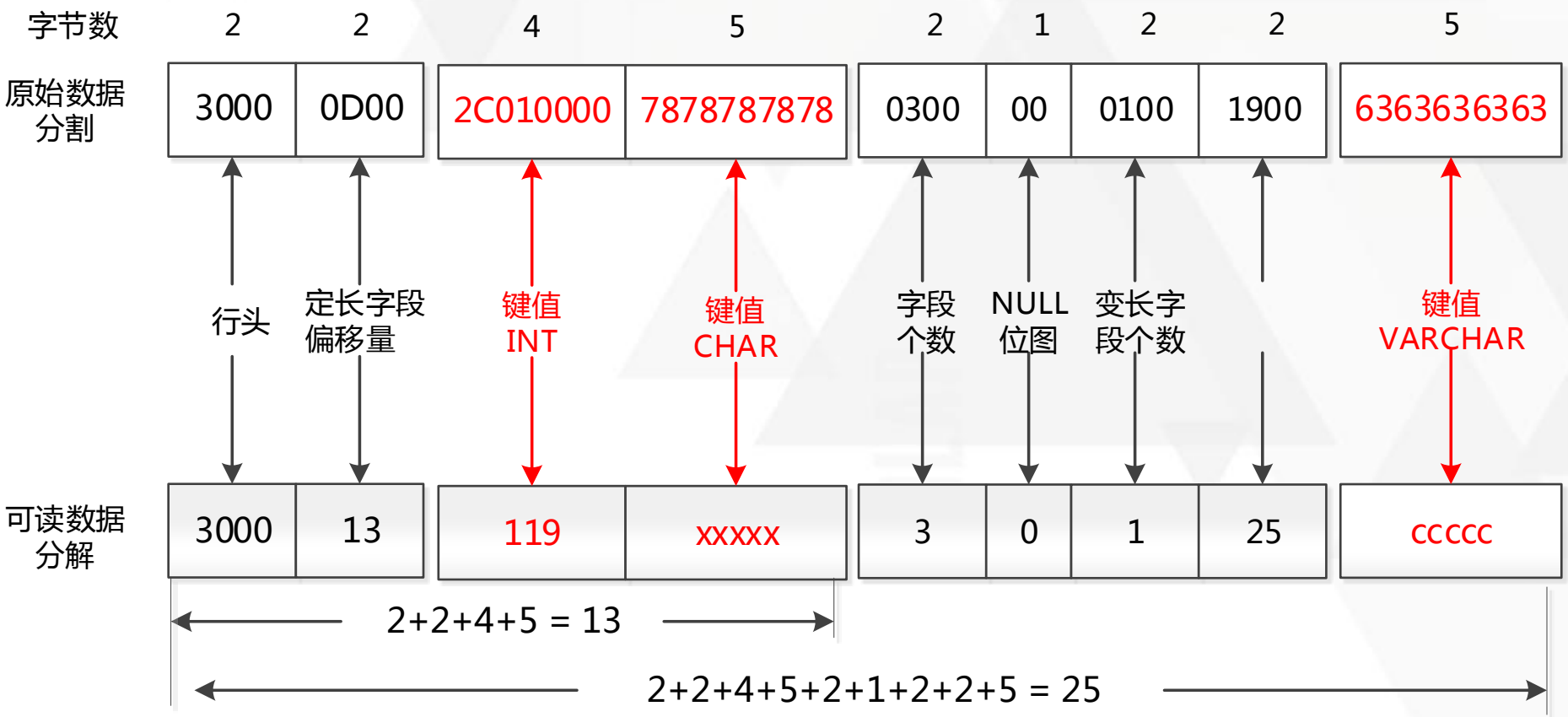


聚集索引(Leaf data)

```
01 DATA:
02 Slot 0, Offset 0x60, Length 25, DumpStyle BYTE
03 Record Type = PRIMARY_RECORD
   Record Attributes =  NULL_BITMAP VARIABLE_COLUMNS
04 Record Size = 25
05 Memory Dump @0x000000001197A060
06 0000000000000000:
   30000d00 2c010000 78787878 78030000 01001900
0..,...xxxxx.......
07 0000000000000014:   63636363 63
ccccc
08
09 Slot 1, Offset 0x79, Length 25, DumpStyle BYTE
10 Record Type = PRIMARY_RECORD       Record Attributes =
NULL_BITMAP VARIABLE_COLUMNS
11 Record Size = 25
12 Memory Dump @0x000000001197A079
13 0000000000000000:
   30000d00 2d010000 78787878 78030000 01001900
0...-...xxxxx.......
14 0000000000000014:   63636363 63
ccccc
```

聚集索引(Leaf)

原始数据 30000D002C010000787878780300001001900 6363636363



堆表 VS 聚集索引表

堆表

- index_id的值为0
- 数据存储没有特定的顺序
- 扫描表或非聚集索引检索数据
- 数据页之间没有指针关联
- 非聚集索引保存RowID
- 不用额外的空间去存储聚集索引
- 不用额外维护聚集索引

聚集索引表

- index_id的值为1
- 数据存储基于聚集索引键顺序存储
- 聚集索引键或非聚集索引检索数据
- 数据页之间有指针链接(双向链表)
- 非聚集索引保存聚集键值(例如ID)
- 需要额外的空间存储聚集索引
- INSERT、UPDATE、DELETE操作时需要额外维护聚集索引

聚集索引 VS 非聚集索引

聚集索引

非聚集索引

- 一个表只能有一个聚集索引
- 聚集索引确定数据的物理顺序
- 叶结点就是数据结点
- index_id的值为1
- 找到索引就找到数据了
- 建议聚集索引值是唯一的

- 一个表能有999个非聚集索引(2008)
- 跟物理顺序没有直接关系
- 叶结点指向数据页
- index_id的值为>1
- 通过RID或者聚集键值找数据
- 建议数据选择性高就可以

行溢出提纲

- 什么是行溢出？
- 什么时候会发生行溢出？
- 行溢出的存储结构到底长什么样子？
- 行溢出带来什么问题？
- 怎么规避这些问题？

行溢出

什么是行溢出？

- In-row data
- Row-overflow data
- LOB data

什么时候会发生行溢出？

- 行溢出数据只会发生在变长字段上，变长列的长度不能超过标准变长列最大值8000个字节的限制
- 包括行头系统信息和所有定长列和变长系统信息的所有长度不能超过8060字节
- 变长列的实际长度一定要超过24个字节
- 变长列不能是聚集索引键的一部分

行溢出

```
01 USE master
02 GO
03 IF EXISTS(SELECT name FROM sys.databases WHERE name =
'Overflow')
04 DROP DATABASE [Overflow]
05 GO
06 CREATE DATABASE [Overflow]
07 GO
08
09 USE [Overflow]
10 GO
11 CREATE TABLE [HeapPage_Overflow] (
12  Id INT,
13  VarCol1 VARCHAR (6000),
14  VarCol2 VARCHAR (3000))
15 GO
16
17 INSERT INTO [HeapPage_Overflow]
18  VALUES (1,REPLICATE('a',6000),REPLICATE('b',3000))
19 GO
```

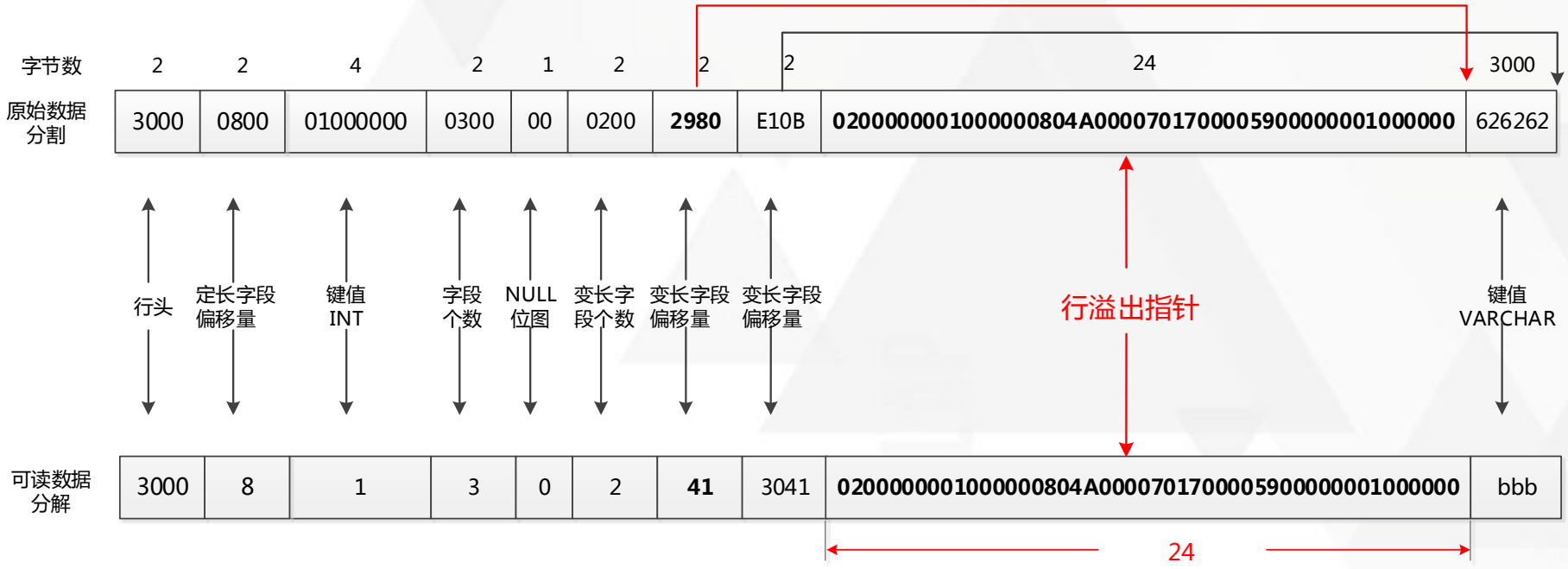
	PageFID	PagePID	IAMFID	IAMPID	ObjectID	IndexID	iam_chain_type	PageType	IndexLevel
1	1	94	NULL	NULL	53575229	0	In-row data	10	NULL
2	1	93	1	94	53575229	0	In-row data	1	0
3	1	90	NULL	NULL	53575229	0	Row-overflow data	10	NULL
4	1	89	1	90	53575229	0	Row-overflow data	3	0

行溢出(data)

```
01 DATA:
02 Slot 0, Offset 0x60, Length 3041, DumpStyle BYTE
03 Record Type = PRIMARY_RECORD        Record Attributes =
NULL_BITMAP VARIABLE_COLUMNS
04 Record Size = 3041
05 Memory Dump @0x000000001006A060
06
07 0000000000000000:
    30000800 01000000 03005802 002980e1 ?0.........X..)..
08 0000000000000010:
    0b020000 00010000 00804a00 00701700 ?.........J..p..
09 0000000000000020:
    00590000 00010000 00626262 62626262 ?.Y.......bbbbbbb
```

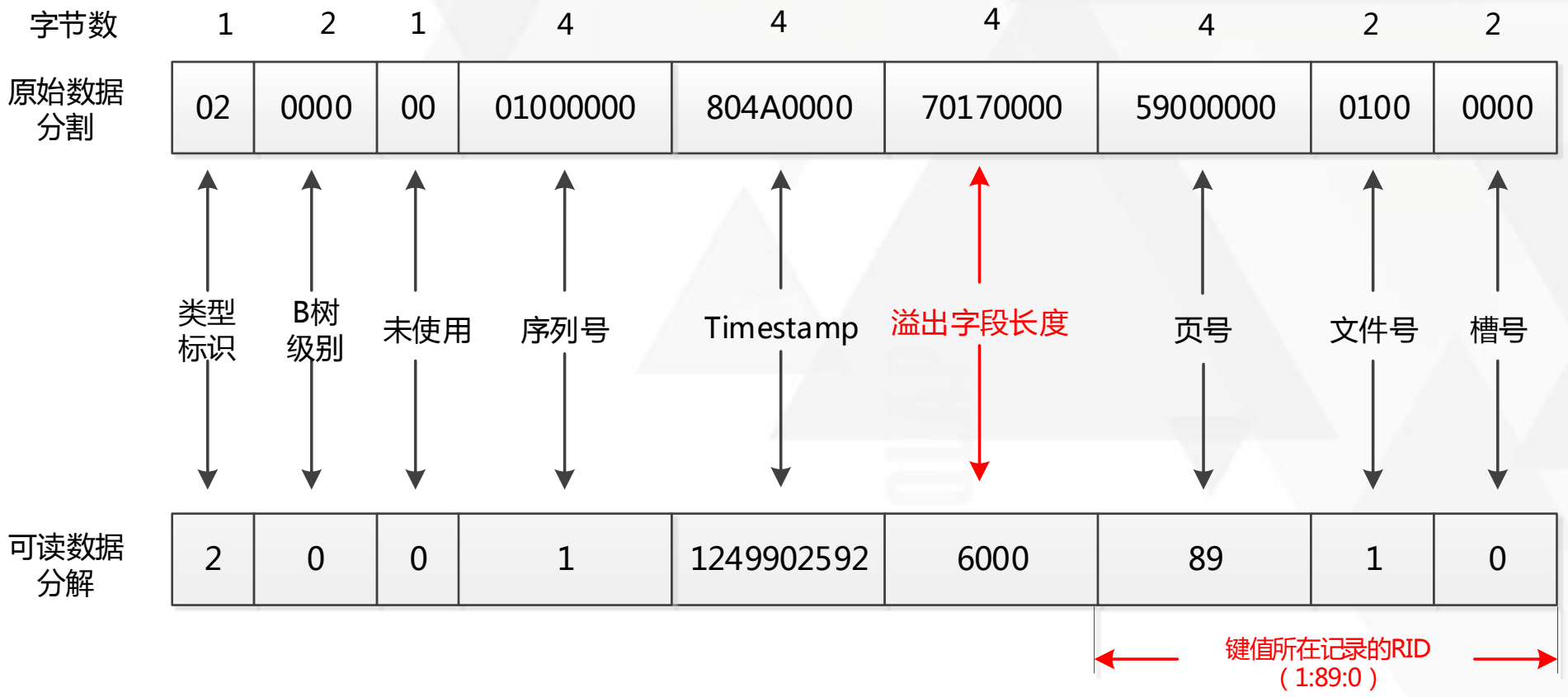
行溢出(行内数据)

原始数据 3000080001000000030058020029 80E10B020000000100000080 4A0000701700005900000001000000626262……

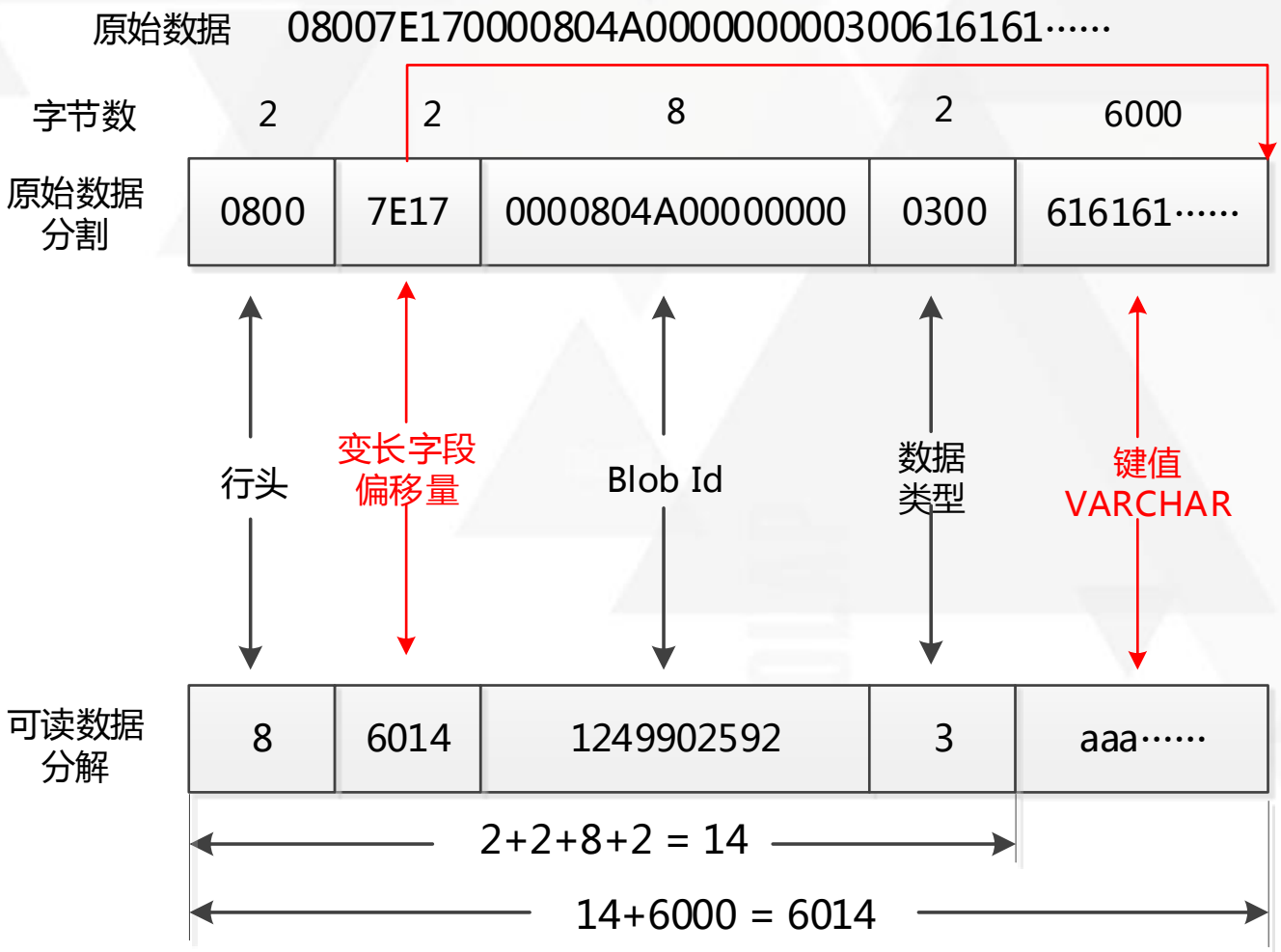


行溢出(指针)

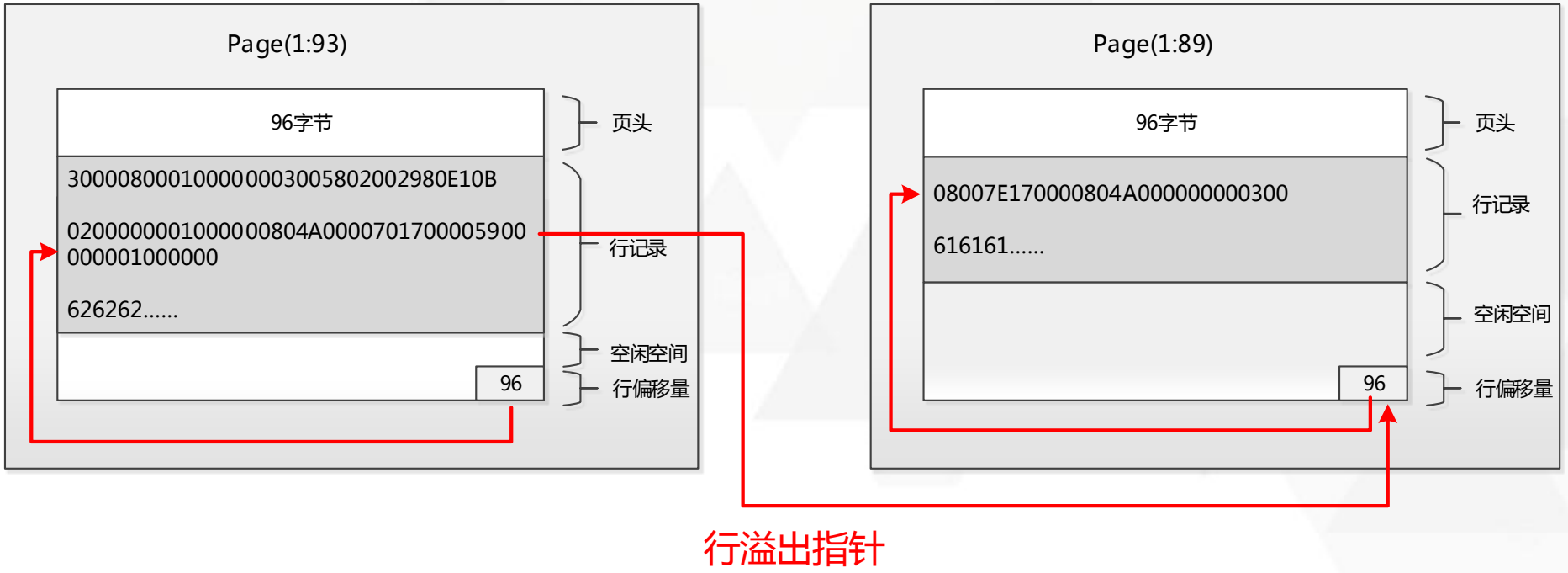
原始数据 0200000001000000804A0000701700005900000001000000



行溢出(溢出数据)


原始数据
08007E170000804A0000000000300616161……
字节数
2
2
8
2
6000
原始数据
分割
0800
7E17
0000804A00000000
0300
616161……
行头
变长字段
偏移量
Blob Id
数据
类型
键值
VARCHAR
可读数据
分解
8
6014
1249902592
3
aaa……
2+2+8+2 = 14
14+6000 = 6014

行溢出(逻辑图)


Page(1:93)
96字节
页头
3000080001000000030058020029 80E10B
020000000100000804A0000701700005900
000001000000
626262……
行记录
空闲空间
96
行偏移量
Page(1:89)
96字节
页头
08007E170000804A000000000300
616161……
行记录
空闲空间
96
行偏移量
行溢出指针

行溢出

- 行溢出带来什么问题？

- 增加随机IO
- 增大内存中Page
- 增大行记录占用的空间

- 怎么规避这些问题？（行溢出占用比例较大）

- 使用大对象存储数据，所有列值在其它页集中存储
- 分开存储，垂直切分表

LOB

- text、ntext和image

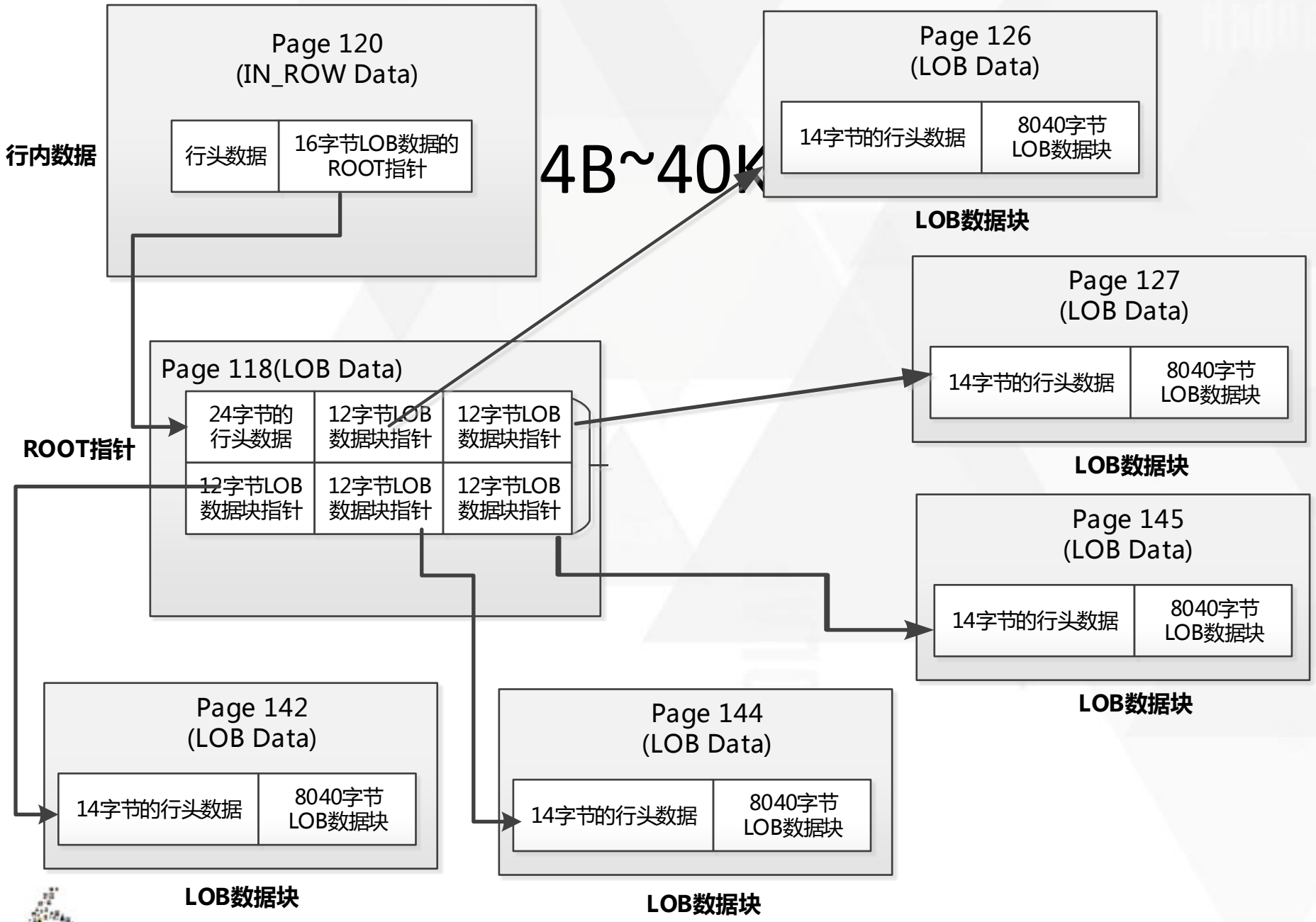
- 默认情况下，LOB数据不存储在行内记录
- 6个字节的指针指向LOB数据块
- ROOT结构，由ROOT结构包含的指针指向真实的LOB数据块
- text in row

- varchar(MAX)、nvarchar(MAX)和varbinary(MAX)

- 字段值 < 8000个字节
- 字段值 > 8000个字节
- large value types out of row

LOB

- <64B
- 64B~40KB
- 40KB~16MB
- >16MB
- text in row=500


Page 120
(IN_ROW Data)
行内数据
行头数据
16字节LOB数据的ROOT指针
4B~40K
Page 126
(LOB Data)
14字节的行头数据
8040字节LOB数据块
LOB数据块
Page 127
(LOB Data)
14字节的行头数据
8040字节LOB数据块
LOB数据块
Page 118(LOB Data)
ROOT指针
24字节的行头数据
12字节LOB数据块指针
12字节LOB数据块指针
12字节LOB数据块指针
12字节LOB数据块指针
12字节LOB数据块指针
Page 145
(LOB Data)
14字节的行头数据
8040字节LOB数据块
LOB数据块
Page 142
(LOB Data)
14字节的行头数据
8040字节LOB数据块
LOB数据块
Page 144
(LOB Data)
14字节的行头数据
8040字节LOB数据块
LOB数据块

更多信息请参考《SQL Server性能调优实战》