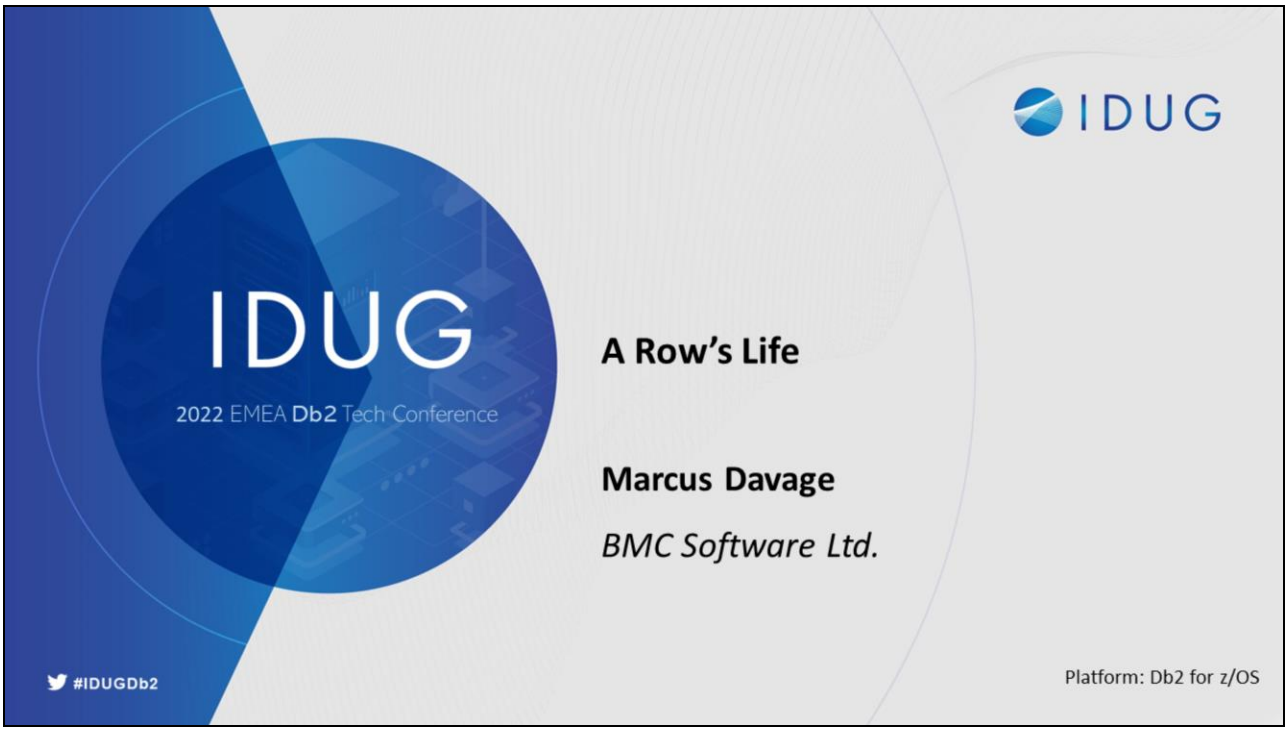


Originally produced and presented by Linda Ball (retired BMC Software) and subsequently modified by Ken McDonald, Jim Dee, Frank Rhodes and myself.

There are different ways to find them and different surroundings. And most DBAs find they need to understand these transformations thoroughly to manage performance and recovery and even security.

In this presentation, we take a lighthearted view of one row's life to emphasize some of the internals of DB2 on the z/OS platform.

Meet Rowdy

- We'll follow Rowdy, a row in a wordsmith's new data base as he
  - goes from raw data to row, gets inserted and indexed,
  - gets logged,
  - gets neighbours,
  - gets updated and accessed and makes some moves, and
  - gets affected by table version changes.
- We will see the definitions for Rowdy's table and other objects, his internal format and how they affect performance.
- And finally, the end will come…Rowdy gets deleted.
- We will discuss briefly how his life could have been different.

Rowdy's Table is Designed

Rowdy's DBA

- Wants the word to be a primary key.
- Might like to have a random word select and thinks an identity column might be a good idea.
- Needs multiple meanings and a quote.
- Needs notes about the origin of a word
- Will insert words, sometimes with minimal info.
- Will want to access multiple rows by letter of the alphabet.

Rowdy's Tablespace is Defined

```
CREATE DATABASE DBAWORDS;
CREATE TABLESPACE UTSPBG IN DBAWORDS
    USING STOGROUP SYSDEFLT PRIQTY -1 SECQTY -1
    FREEPAGE 1 PCTFREE 0 MAXPARTITIONS 1 SEGSIZE 4;
CREATE TABLESPACE UTSPBR IN DBAWORDS
    USING STOGROUP SYSDEFLT PRIQTY -1 SECQTY -1
    FREEPAGE 1 PCTFREE 0 NUMPARTS 1 SEGSIZE 4;
SET CURRENT APPLICATION COMPATIBILITY = 'V12R1M500' ;
CREATE TABLESPACE OLDPART IN DBAWORDS
    USING STOGROUP SYSDEFLT PRIQTY -1 SECQTY -1
    FREEPAGE 1 PCTFREE 0 NUMPARTS 1 SEGSIZE 0 ;
CREATE TABLESPACE OLDSEG IN DBAWORDS
    USING STOGROUP SYSDEFLT PRIQTY -1 SECQTY -1
    FREEPAGE 1 PCTFREE 0 SEGSIZE 4;
```

First of all, we create the database, then we create the table space to house the table and its data.

Db2 12's default table space type is a Universal Table Space (or UTS), it's segmented, and can be partitioned by growth, or partitioned by range.

You can still create old fashioned, non UTS tablespaces, segmented or partitioned, if you really, really want to, but you must issue the SET CURRENT APPLICATION COMPATIBILITY statement first.

This presentation will use a UTS PBG tablespace.

What's in a page?

- Page sets
  - File page sets, containing data entries
  - Index page sets, containing index entries
- Page sizes
  - 4KB, 8KB, 16KB, 32KB
- Page set Types
  - Partitioned, Non-partitioned
  - Segmented, Non-segmented
  - Compressed, Uncompressed
  - Universal Table Space
    - Segmented and Partitioned
    - Partitioned by Range / Growth

Db2 stores data in page sets.

• If the page set contains data records, it is called a *file page set*. A file page set is the physical (internal) representation of a table space. A file page set that contains LOB data is called a LOB page set.

• If it contains index entries, it is called an *index page set*. An index page set is the physical representation of an index space (index).

A page set is a collection of one or more data sets that are logically concatenated to form a linear addressing range.

Db2 data sets are defined as VSAM linear data sets (LDSs).

Each segments contains the same number of pages (in multiples of 4, from 4 to 64), are chained together, and provide performance and locking benefits.

The data sets in a page set contain pages that can be 4 KB, 8 KB, 16 KB, or 32 KB in size.

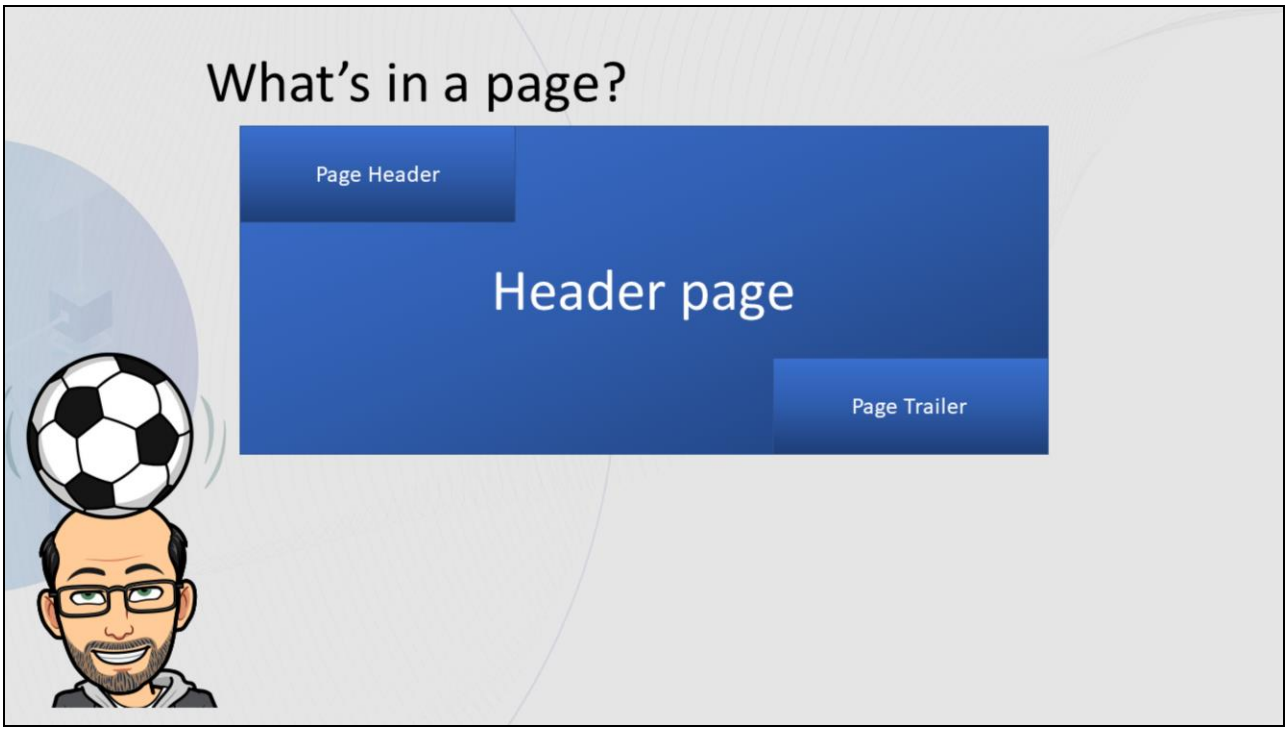
What's in a page?

- Data Page Types
  - Header pages
  - Space map pages
  - Data pages
  - System pages
- Index Page Types
  - Header pages
  - Space map pages
  - Non-leaf pages
  - Directory pages
  - Root pages
  - Leaf pages

A page set for a table space that has undergone ALTERs that resulted in changes to data type definitions also has *system pages*.

In a segmented table space, system pages are in dedicated system segments, with their own space map pages.

LOB and XML page sets have other types of pages, which we shan't go into in this presentation.



Header pages

Header pages of page sets have a 1-byte page trailer for 6-byte RBA and LRSN formats, and a 20-byte page trailer for 10-byte RBA and LRSN formats. The page header fields contain control information that Db2 uses.

Space map pages

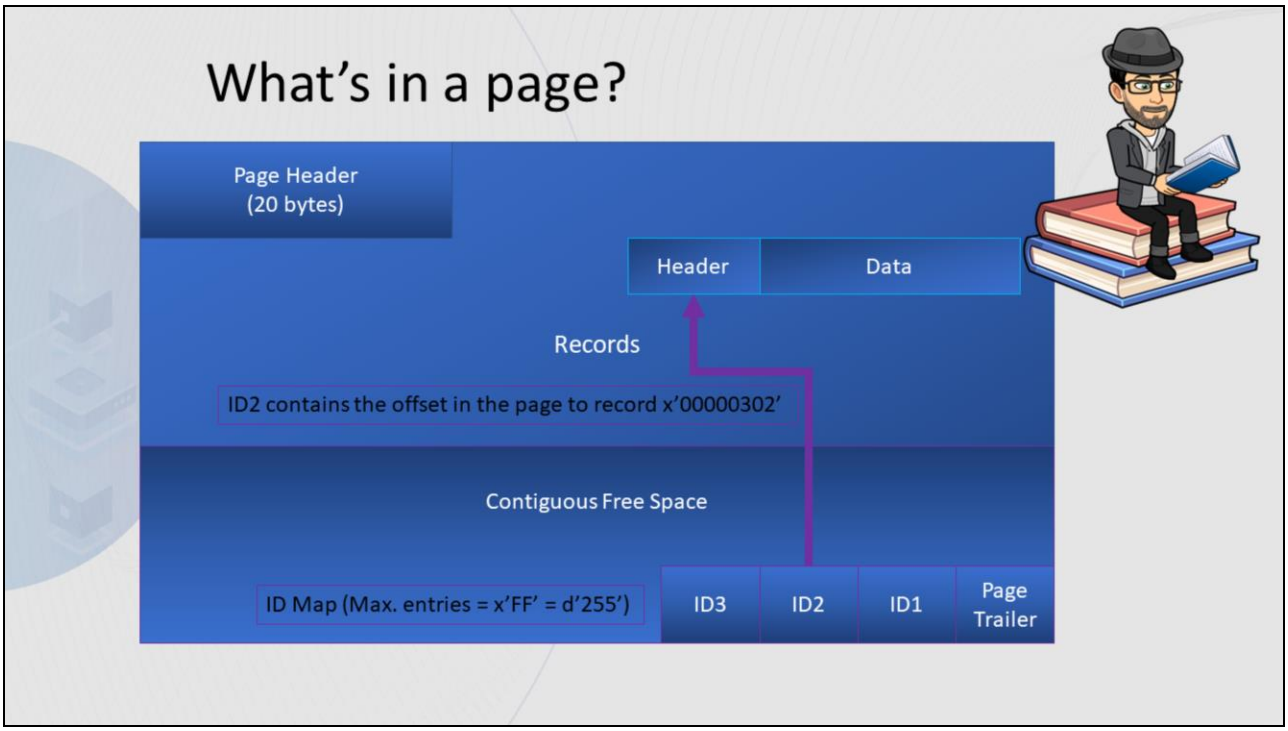
A space map page identifies the data pages that have enough free space for more data to be inserted.

Each space map in a page set covers a specific range of pages. The size of the range is computed based on the type of page set (segmented file, non-segmented file, partitioned file, LOB, or index), the page size, and whether the page set has the MEMBER CLUSTER attribute.

There are six corresponding space map page formats: segmented, non-segmented, partitioned, LOB high-level, LOB low level, and index.

Non-segmented and partitioned file page set space map

pages are almost identical.



File page set data pages

The file page set data page includes several parts.

Contents of a data page

The data page includes the following basic parts:

1. Data Page Header

2. Record

• Records (defined either by the user or by Db2, which can be part of the user's data or part of the Db2 catalog or directory)

• Overflow records and pointer records

• Large and small holes

3. Contiguous free space

4. ID map and page trailer.

Data page header

Every file page set data page has a 20-byte page header, and appendage if an appendage is present, that contains control information that Db2 uses.

Records

Records are stored following the data page header.

Records that are stored in pages represent the rows of a table in a table space. The first 6 bytes of all records contain control information.

This portion of the record is called the record header or prefix. The header is followed by user data.

Rowdy's Table is Defined

```
CREATE TABLE DBA.WORDS
  (WORD          CHAR(30) NOT NULL PRIMARY KEY
  ,SP_TYPE       CHAR(3)
  ,MEANING       VARCHAR(100)
  ,MEANING2      VARCHAR(100)
  ,MEANING3      VARCHAR(100)
  ,CODENUM       SMALLINT
            GENERATED BY DEFAULT AS IDENTITY (CACHE 50,CYCLE)
  ,QUOTE_DATE    DATE
  ,QUOTE         VARCHAR(500)
  ,ORIGIN_NOTES VARCHAR(250)) IN DBAWORDS.OLDSEG;

CREATE UNIQUE INDEX DBA.WORDINDX ON DBA.WORDS (WORD)
  USING STOGROUP SYSDEFLT PRIQTY -1 SECQTY -1 CLUSTER;

CREATE INDEX DBA.CODEINDX ON DBA.WORDS (CODENUM)
  USING STOGROUP SYSDEFLT PRIQTY -1 SECQTY -1;
```

Rowdy Climbs into the Data Base

```
INSERT INTO DBA.WORDS
  (WORD, SP_TYPE, MEANING,
  MEANING2, MEANING3, QUOTE_DATE, QUOTE,
  ORIGIN_NOTES)
VALUES
  ('rowdy', 'N', 'A rough and disorderly person',
  NULL, NULL, NULL, NULL,
  'origin unknown');
```

For this example, ‘rowdy’ is the first row inserted into this empty table. As such, he will be the first row on page 6 (after header page 0 and spacemap page 1).

Rowdy Climbs into the Data Base

The Header Page

```
PARTITION: # 0001
PAGE: # 00000000
HEADER PAGE:  PGCOMB='00'X  PGBIGRBA='0000000032F96FD671BE'X  PGNUM='00000000'X  PGFLAGS='38'X
              HPGOBID='98550002'X  HPGHPREF='000000B4'X  HPGCATRL='00'X  HPGREL='Q'  HPGZLD='.'
              HPGCATV='00'X  HPGTORBA='000000000000'X  HPGTSTMP='20220929123612320650'X
              HPGSSNM='DEJM'  HPGFOID='0001'X  HPGPGSZ='1000'X  HPGSGSZ='0004'X  HPGPARTN='0001'X
.
.
               HPGBIGMASSDELETETIMESTAMP='00000000000000000000'X  FOEND='42'X
```

For this example, ‘rowdy’ is the first row inserted into this empty table. As such, he will be the first row on page 6 (after header page 0 and spacemap page 1).

Rowdy Climbs into the Data Base

The Segmented Spacemap Page

```
PAGE: # 00000001
SEGMENTED SPACEMAP PAGE:  PGCOMB='00'X  PGBIGRBA='0000000032F96FD4B000'X  PGNUM='00000001'X
                          PGFLAGS='30'X  SEGNUM='01AA'X  SEGFREE='01A8'X  SEGENT='0003'X
                          SEGSIZE='0004'X  SEGLENT='00000005'X  SEGBFMEM='00'X  FOEND='42'X
FIRST PART OF SEGMENTED SPACE MAP:
SEG 0001   000000000001E0 F000
SEG 0002   000000000009C0 3000

SECOND PART OF SEGMENTED SPACE MAP:
RELPG  00       20       40       60       80       A0       C0       E0
0000   0000
```

For this example, ‘rowdy’ is the first row inserted into this empty table. As such, he will be the first row on page 6 (after header page 0 and spacemap page 1).

Rowdy Climbs into the Data Base

The System Page

```
PAGE: # 00000002
SYSTEM PAGE:  PGCOMB='10'X  PGBIGRBA='0000000032F96FD427B6'X  PGNUM='00000002'X  PGFLAGS='28'X
              PGFREE=3376  PGFREE='0D30'X  PGFREEP=698  PGFREEP='02BA'X  PGHOLE1='0000'X
              PGMAXID='01'X  PGNANCH=1
SPOH:  SPOPREV='00000000'X  SPOBKT='01'X  SPOFLAGS='00'X
PGBIGTAIL:  PGPRETAIL='00000000000000000000000032F96FD427B6'X  PGIDFREE='00'X  PGEND='52'X
ID-MAP FOLLOWS:
  1 0024
```

For this example, ‘rowdy’ is the first row inserted into this empty table. As such, he will be the first row on page 6 (after header page 0 and spacemap page 1).

Rowdy Climbs into the Data Base

The Index Page

```
***  BEGINNING OF PAGE NUMBER 00000003 ***
0000 10000000 00000000 0000037C 00000000   00000000 000A40C8 00010F88 00620000
0020 0000001E 00000002 01060001 00010000   00000000 00000000 00010001 08809996
0040 A684A840 40404040 40404040 40404040   40404040 40404040 40404040 40000000
0060 06010000 00000000 00000000 00000000   00000000 00000000 00000000 00000000
.... LINES ARE ALL ZERO.
0FE0 00000000 00000000 0000003E 00000000   00000000 00000000 32F96EF9 90000052
```

Rowdy's data in this index (on WORD) includes his WORD value (rowdy or X'9996A684A8' followed by blanks) *and* the value used to locate his page and hence his row data: X'00000601', or Page 6, Row 1.

This is the WORDINDX index entry.

Index pages identify the rows by representing them in two-byte entries beginning (for row 1, Rowdy's ID) 20 bytes from the end of page (it used to be two).

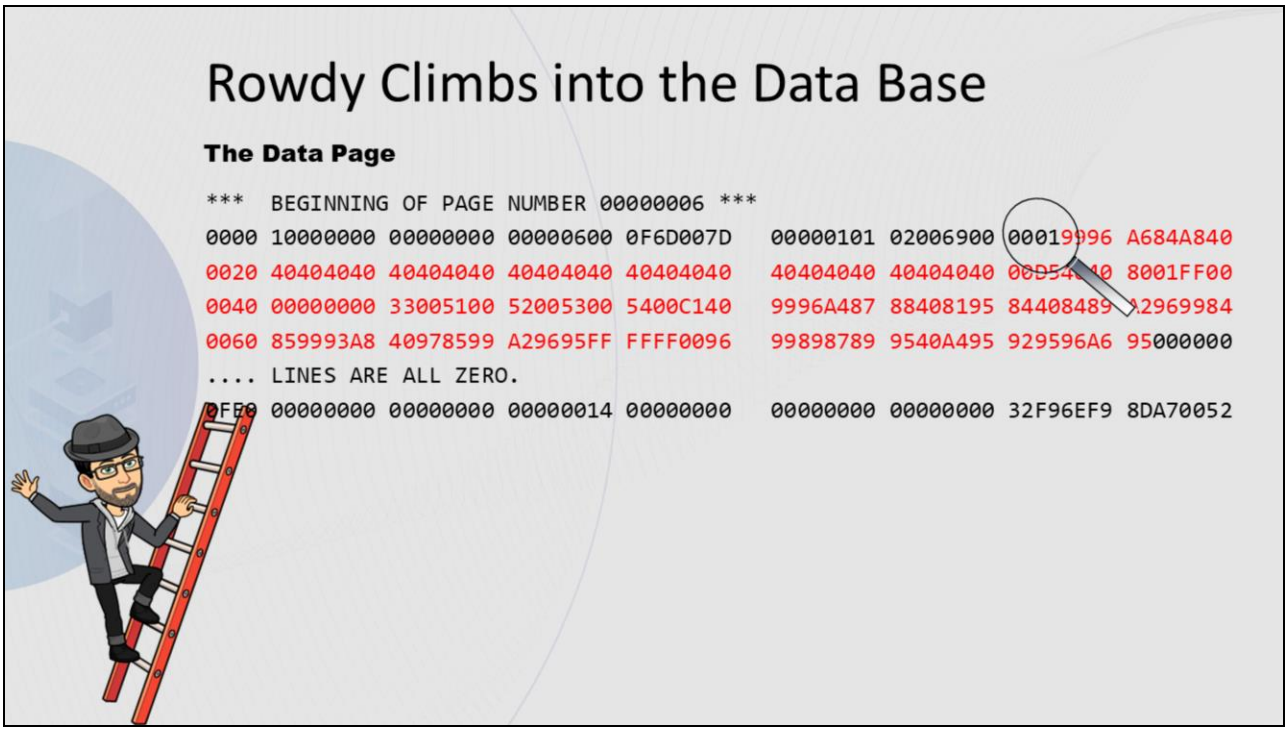
<click>

The index entry for Rowdy is at x'3E'.

<click>

Which is located here.

Rowdy's data in this index (on WORD) includes his WORD value (rowdy or X'9996A684A8' followed by blanks) *and* the value used to locate his page and hence his row data: X'00000601', or Page 6, Row 1.



Data pages identify the rows by representing them in two-byte entries beginning (for row 1, Rowdy's ID) 20 bytes from the end of page (it used to be two).

The offset where the data is located is in this entry.

The first x'14' bytes of all tablespace pages is the page header.

The last byte of the page is the parity byte.

We'll talk about the map entry at the end of the page in a bit more depth later.

<click>

But, 'rowdy' being ROWID 1 is at offset '14'

The entire row is highlighted here… but, we'll actually look at row layout a little bit more towards the end of the presentation.

Rowdy Climbs into the Data Base

The Data Page (formatted)

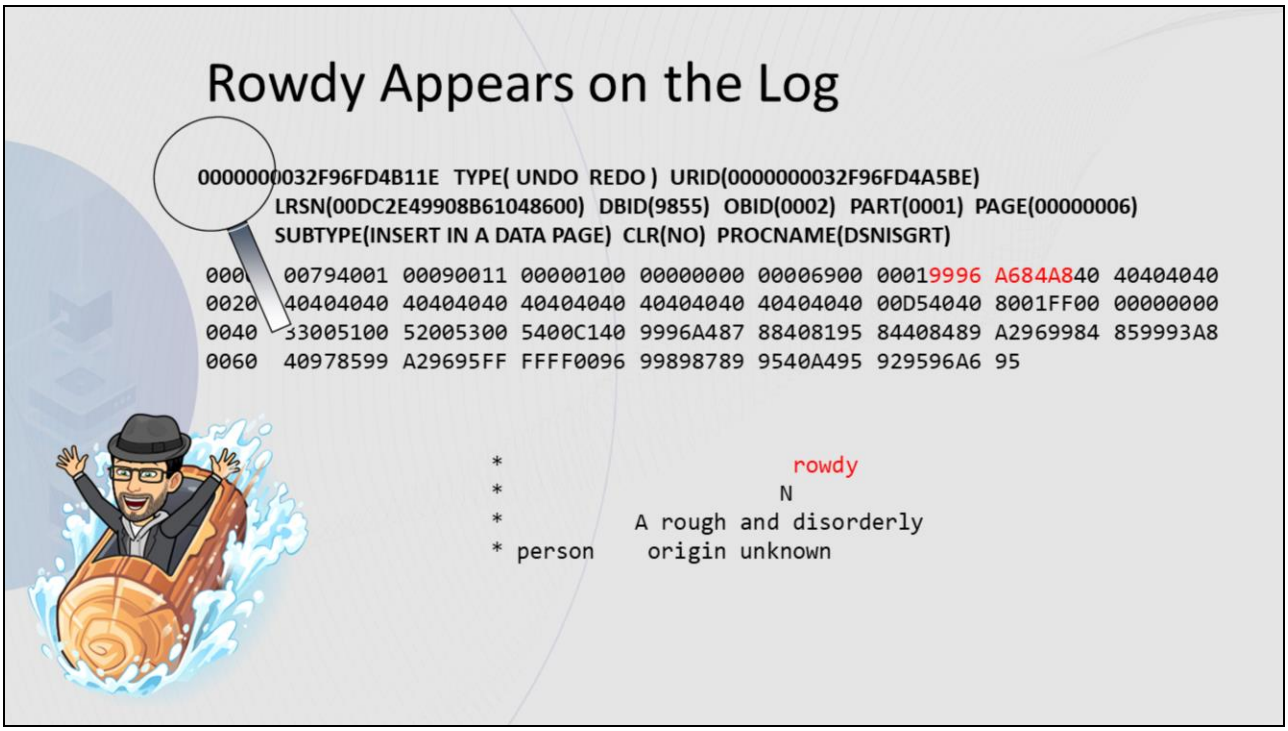
```
DATA PAGE:  PGCOMB='10'X  PGBIGRBA='0000000032F96FD4B11E'X  PGNUM='00000006'X  PGFLAGS='00'X
            PGFREE=3949  PGFREE='0F6D'X  PGFREEP=125  PGFREEP='007D'X  PGHOLE1='0000'X
            PGMAXID='01'X  PGNANCH=1
PGBIGTAIL:  PGPRETAIL='00000000000000000000000032F96FD4B11E'X  PGIDFREE='00'X  PGEND='52'X
ID-MAP FOLLOWS:
01  0014

RECORD:  XOFFSET='0014'X  PGSFLAGS='02'X  PGSLTH=105  PGSLTH='0069'X  PGSOBD='0000'X  PGSBID='01'X
9996A684 A8404040 40404040 40404040 40404040 40404040 40404040 404000D5   rowdy                          .N
40408001 FF000000 00003300 51005200 53005400 C1409996 A4878840 81958440     ..................A rough and
8489A296 99848599 93A84097 8599A296 95FFFFFF 00969989 87899540 A4959295   disorderly person....origin unkn
96A695                                                                     own
```

Here is the formatted version of the page. As you can see, the offset is x'14'.

<click>

The log RBA of the inserted row is here.

<click>



This is the INSERT log record for the tablespace data page. There are also log records associated with the index updates as well.

There have been previous NA and EU IDUG presentations on deciphering the log.

DSN1LOGP – It could save your job one day – covers syntax and examples of using DSN1LOGP to find logged information.

BITs and Pieces of the DB2 Log – A geek level presentation using the DSNMACS(DSNDQJ00) macro to map various log records and their content.

1. This is the RBA of the INSERT. This (or the LRSN in data sharing) is also the PGLOGRBA of the page being updated. The PGLOGRBA reflects the last activity (INSERT, UPDATE, DELETE, PAGE COMPACTION, etc.) against the page.

The PGLOGRBA is used by copy and recovery utilities as well as the various log tools.

This is the row header… 6 bytes which contain a byte of flags, a halfword length, the halfword OBID, and the last byte is usually the ROWID. The ROWID can be used to find the corresponding PGMAP entry at the bottom of the page to locate the offset into the page where the row resides.

Now, row VERSIONING introduced in V8.1 could impact this and the last byte could reflect the VERSION of the row instead of the ROWID. A bit in the flags (first byte of the row header) indicate what this byte represents.

The first 8 bytes prior to the row header here is the DM Segment Header… byte 0003 is also (and so far always) the ROWID. This is used by the recovery and log tools to find the PGMAP entry.

How did Rowdy get there?

```
032F96FD0BDB3 TYPE(REDO) DBID(9855) OBID(0000) SUBTYPE(EXCLUSIVE LOCK)
032F96FD15B07 TYPE(REDO) DBID(9855) OBID(0002) SUBTYPE(EXCLUSIVE LOCK)
032F96FD16559 TYPE(REDO) DBID(9855) OBID(0002) SUBTYPE(EXCLUSIVE LOCK)
032F96FD25358 TYPE(REDO) DBID(9855) OBID(0002) PART(0001) PAGE(00000000) SUBTYPE(RE/FORMAT,MODIFY HEADER/SPACEMAP/ROOT PAGE)
032F96FD262F2 TYPE(REDO) DBID(9855) OBID(0002) PART(0001) PAGE(00000001) SUBTYPE(RE/FORMAT,MODIFY HEADER/SPACEMAP/ROOT PAGE)
032F96FD2A68C TYPE(REDO) DBID(9855) OBID(0004) PART(0001) PAGE(00000000) SUBTYPE(RE/FORMAT,MODIFY HEADER/SPACEMAP/ROOT PAGE)
032F96FD2B626 TYPE(REDO) DBID(9855) OBID(0004) PART(0001) PAGE(00000001) SUBTYPE(RE/FORMAT,MODIFY HEADER/SPACEMAP/ROOT PAGE)
032F96FD2EC0A TYPE(REDO) DBID(9855) OBID(0006) PART(0001) PAGE(00000000) SUBTYPE(RE/FORMAT,MODIFY HEADER/SPACEMAP/ROOT PAGE)
032F96FD2FBA4 TYPE(REDO) DBID(9855) OBID(0006) PART(0001) PAGE(00000001) SUBTYPE(RE/FORMAT,MODIFY HEADER/SPACEMAP/ROOT PAGE)
032F96FD3361A TYPE(REDO) DBID(9855) OBID(0008) PAGE(00000000) SUBTYPE(RE/FORMAT,MODIFY HEADER/SPACEMAP/ROOT PAGE)
032F96FD345B4 TYPE(REDO) DBID(9855) OBID(0008) PAGE(00000001) SUBTYPE(RE/FORMAT,MODIFY HEADER/SPACEMAP/ROOT PAGE)
032F96FD3AF63 TYPE(REDO) DBID(9855) OBID(0002) SUBTYPE(EXCLUSIVE LOCK)
032F96FD3B520 TYPE(UNDO REDO) DBID(9855) OBID(0002) PART(0001) PAGE(00000000) SUBTYPE(RE/FORMAT,MODIFY HEADER/SPACEMAP/
                                                                                                    ROOT PAGE)
032F96FD3EE13 TYPE(REDO) DBID(9855) OBID(000B) PAGE(00000000) SUBTYPE(RE/FORMAT,MODIFY HEADER/SPACEMAP/ROOT PAGE)
032F96FD3F067 TYPE(REDO) DBID(9855) OBID(000B) PAGE(00000001) SUBTYPE(RE/FORMAT,MODIFY HEADER/SPACEMAP/ROOT PAGE)
032F96FD400FC TYPE(REDO) DBID(9855) OBID(000B) PAGE(00000002) SUBTYPE(RE/FORMAT,MODIFY HEADER/SPACEMAP/ROOT PAGE)
```

These are all the records from the log of the operations performed on the WORDS database and UTSPBG tablespace.

Here, the tablespace, the table and the indexes are being created, and their pages formatted.

How did Rowdy get there?

```
032F96FD450FC TYPE(REDO) DBID(9855) OBID(000D) PAGE(00000000) SUBTYPE(RE/FORMAT,MODIFY HEADER/SPACEMAP/ROOT PAGE)
032F96FD45323 TYPE(REDO) DBID(9855) OBID(000D) PAGE(00000001) SUBTYPE(RE/FORMAT,MODIFY HEADER/SPACEMAP/ROOT PAGE)
032F96FD463B8 TYPE(REDO) DBID(9855) OBID(000D) PAGE(00000002) SUBTYPE(RE/FORMAT,MODIFY HEADER/SPACEMAP/ROOT PAGE)
032F96FD4647B TYPE(REDO) DBID(9855) OBID(000D) PAGE(00000003) SUBTYPE(RE/FORMAT,MODIFY HEADER/SPACEMAP/ROOT PAGE)
032F96FD4654C TYPE(REDO) DBID(9855) OBID(000D) PAGE(00000004) SUBTYPE(RE/FORMAT,MODIFY HEADER/SPACEMAP/ROOT PAGE)
032F96FD4A720 TYPE(UNDO) DBID(9855) OBID(000B) PART(0001) PAGE(00000000) SUBTYPE(NOOP LOG RECORD)
032F96FD4AC59 TYPE(UNDO REDO) DBID(9855) OBID(0002) PART(0001) PAGE(00000001) SUBTYPE(SEGMENT ALLOCATION/DEALLOCATION)
032F96FD4ACE2 TYPE(UNDO REDO) DBID(9855) OBID(0002) PART(0001) PAGE(00000000) SUBTYPE(RE/FORMAT,MODIFY HEADER/SPACEMAP/
                                                                                                    ROOT PAGE)
032F96FD4AEC2 TYPE(UNDO REDO) DBID(9855) OBID(0002) PART(0001) PAGE(00000001) SUBTYPE(UPDATE SPACE MAP)
032F96FD4AF42 TYPE(REDO) DBID(9855) OBID(0002) PART(0001) PAGE(00000001) SUBTYPE(UPDATE SPACE MAP)
032F96FD4B000 TYPE(REDO) DBID(9855) OBID(0002) PART(0001) PAGE(00000001) SUBTYPE(CURRENT LAST ENTRY IN SPACE MAP PAGE)
032F96FD4B078 TYPE(REDO) DBID(9855) OBID(0002) PART(0001) PAGE(00000006) SUBTYPE(FORMAT PAGE OR MODIFY SPACE MAP)
032F96FD4B11E TYPE(UNDO REDO) DBID(9855) OBID(0002) PART(0001) PAGE(00000006) SUBTYPE(INSERT IN A DATA PAGE)
032F96FD4B307 TYPE(UNDO REDO) DBID(9855) OBID(000B) PART(0001) PAGE(00000003) SUBTYPE(TYPE 2 INDEX UPDATE)
032F96FD4B3E1 TYPE(UNDO) DBID(9855) OBID(000D) PART(0001) PAGE(00000000) SUBTYPE(NOOP LOG RECORD)
032F96FD4B990 TYPE(UNDO REDO) DBID(9855) OBID(000D) PART(0001) PAGE(00000003) SUBTYPE(TYPE 2 INDEX UPDATE)
```

Here, the table's pages are being formatted: the header page, the space map page, the root page.

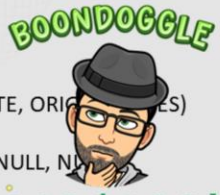
The segment is allocated and the space map page is updated.

Finally, page 6 is allocated, the space map page is updated, the row is inserted and the index updated.

Rowdy's Neighbourhood Gets Crowded

r o y a l
9996A88193

```
INSERT INTO MVSMJD.WORDS
    (WORD, SP_TYPE, MEANING, MEANING2, MEANING3, QUOTE_DATE, QUOTE, ORIGIN_NOTES)
    VALUES ('royal', 'A', 'Of or pertaining to a king, queen, or other monarch',
     'Superior in size or quality', NULL, NULL, NULL,
     'Middle English roial from Old French from Latin regalis' );
```




```
INSERT INTO MVSMJD.WORDS
    (WORD, SP_TYPE, MEANING, MEANING2, MEANING3, QUOTE_DATE, QUOTE, ORI     ES)
    VALUES ('boondoggle', 'N',
     'Unnecessary, Wasteful, and often counterproductive work', NULL, NULL, NULL, N
     'Coined by R.H. Link, 20th-century American scoutmaster' );
```

b o o n d o g g l e
8296965849687879385

...

r o v e r
9996A58599

```
INSERT INTO MVSMJD.WORDS
    (WORD, SP_TYPE, MEANING, MEANING2, MEANING3, QUOTE_DATE, QUOTE, ORIGIN_NOTES)
    VALUES ('rover', 'N', 'One who roves; wanderer; nomad', 'A pirate vessel',
     'Archery. A mark selected by chance', NULL, NULL, 'Origin unknown' );
```

After many more INSERTs, the page fills up.

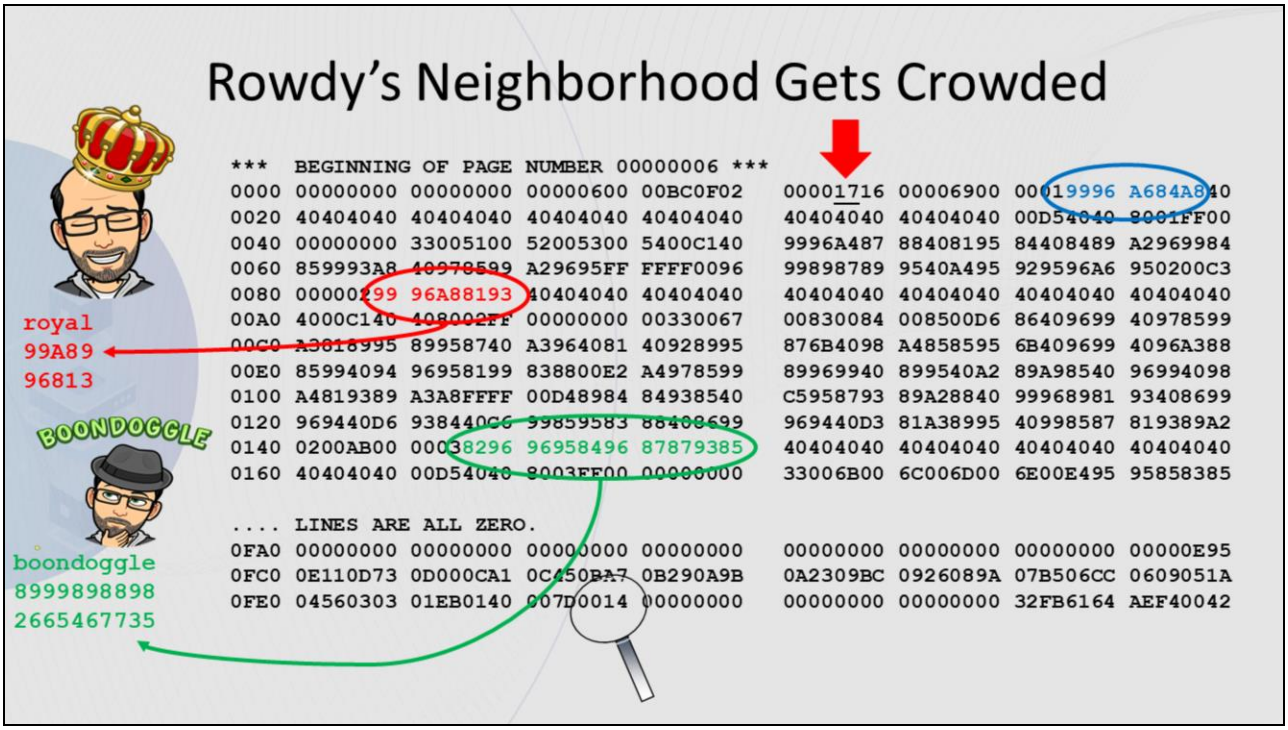
Rowdy's Neighborhood Gets Crowded

The Data Page (formatted)

```
DATA PAGE:  PGCOMB='00'X  PGBIGRBA='0000000032FB6164AEF4'X  PGNUM='00000006'X  PGFLAGS='00'X
            PGFREE=188  PGFREE='00BC'X  PGFREEP=3842  PGFREEP='0F02'X  PGHOLE1='0000'X
            PGMAXID='17'X  PGNANCH=22
PGBIGTAIL:  PGPRETAIL='0000000000000000000000032FB6164AEF4'X  PGIDFREE='00'X  PGEND='42'X
ID-MAP FOLLOWS:
01  0014 007D 0140 01EB 0303 0456 051A 0609
09  06CC 07B5 089A 0926 09BC 0A23 0A9B 0B29
11  0BA7 0C[illegible] 0CA1 0D00 0D73 0E11 0E95

RECORD:  XOFFSET='0014'X  PGSFLAGS='00'X  PGSLTH=105  PGSLTH='0069'X  PGSOBD='0000'X  PGSBID='01'X
9996A684 A8404040 40404040 40404040 40404040 40404040 40404040 404000D5   rowdy                          .N
40408001 FF000000 00003300 51005200 53005400 C1409996 A4878840 81958440     ..................A rough and
8489A296 99848599 93A84097 8599A296 95FFFFFF 00969989 87899540 A4959295   disorderly person....origin unkn
96A695                                                                     own
```

Look at the space map page now! See how it has grown!

The number of rows in the ID-MAP matches the PGMAXID value of x'17' or decimal 23.



The PGMAXID is underlined at offset x'12' in the 'Header' portion of the page. This page contains potentially x'17' or 23 rows. (There could be holes due to overflows or deletes.)

This byte is the architectural limit as to why the maximum number of rows on a single page is 255.

Here's Rowdy.

Here's Royal.

Here's Boondoggle.

The PGMAP is at the bottom of the page. Each halfword entry is the offset to that ROWID.

Rowdy gets a Makeover

```
UPDATE DBA.WORDS
SET
QUOTE_DATE = CURRENT_DATE,
QUOTE =  'There was a rowdy in the IDUG crowd.
WHERE WORD = 'rowdy';
```

An update to ‘rowdy’… this will increase his length and move him to a different location on the page but will not yet cause an overflow.

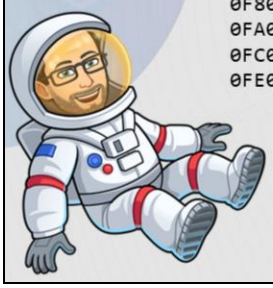
Rowdy finds some Space

This was the end of the page...

```
0FE0 00000000 00000000 00000014 00000000   00000000 00000000 32F96EF9 8DA70052
```

And now...

```
0F00 A6950200 8D000001 9996A684 A8404040   40404040 40404040 40404040 40404040
0F20 40404040 404000D5 40408001 00202209   30003300 51005200 53007800 C1409996
0F40 A4878840 81958440 8489A296 99848599   93A84097 8599A296 95FFFF00 E3888599
0F60 8540A681 A2408140 9996A684 A8408995   40A38885 40C9C4E4 C7408399 96A6844B
0F80 00969989 87899540 A4959295 96A69500   00000000 00000000 00000000 00000000
0FA0 00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000E95
0FC0 0E110D73 0D000CA1 0C450BA7 0B290A9B   0A2309BC 0926089A 07B506CC 0609051A
0FE0 04560303 01EB0140 007D0F02 00000000   00000000 00000000 32FB6FE0 547A0052
```



Note that ROWID 01's location changed from 0014 to 0EF3. Even more interesting that the page was squeezed to consolidate free space to allow for this row to fit into its 'home' page. Note that ROWID 02 is now at location 0014. This indicates that there is no implied order to the ROWs on a given page… row 1 does not have to be the first row… the PGMAP allows us to find the correct offset for the corresponding rows.

PAGE COMPACTION log records were introduced via into Versions 7 and 8 of DB2 via APAR PK19182. These records reflect the fact that a squeeze occurred.

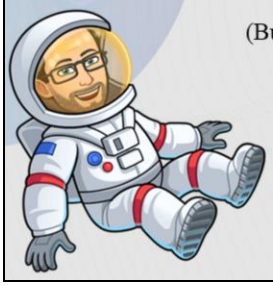
Rowdy finds some Space

Take a look at the top of the page…

```
***  BEGINNING OF PAGE NUMBER 00000006 ***
0000 10000000 00000000 00000600 00980F8F   00141701 80006900 00019996 A684A840
0020 40404040 40404040 40404040 40404040   40404040 40404040 00D54040 8001FF00
0040 00000000 33005100 52005300 5400C140   9996A487 88408195 84408489 A2969984
0060 859993A8 40978599 A29695FF FFFF0096   99898789 9540A495 929596A6 950000C3
```

Rowdy's still there!

(But inaccessible)



```
*............q............rowdy *
*                          .N  ....*
*..............A rough and disord*
*erly person....origin unknown..C*
```

Rowdy's original row data is still in its original position in the page – offset x'14', but as the row pointer has changed to x'0F02', this old data is inaccessible.

New Columns for Rowdy's Table

```
ALTER TABLE DBA.WORDS ADD QUOTE_SOURCE VARCHAR(100);
ALTER TABLE DBA.WORDS ADD NUM_REFERENCES SMALLINT;
```

```
PARTITION: # 0001     PAGE: # 00000001              PAGE: # 00000003
PAGE: # 00000000      SEGMENTED SPACEMAP PAGE:      SYSTEM PAGE:
HEADER PAGE:

DSN1985I ZERO PAGES ENCOUNTERED. FIRST PAGE =  00000004, LAST PAGE =  00000005
```

```
PAGE: # 00000006
DATA PAGE:
ID-MAP FOLLOWS:
01  0014 001A 0110 0116 0261 0267 035E 0364
09  0C61 0D7D 0565 063F 071A 07DA 07E0 0E95
11  0962 0968 0A1D 0ACA 0AD0 0AD6 0BB3
```

One thing to note here… In DB2 Version 8, this would not version the row. But, with Reordered Row Format in DB2 Version 9, the addition of a fixed length column to a row which has variable columns will cause versioning.

Once a page update (insert/update/delete) is performed, a system page is added to the page set.

Mistakes were Made!

```
UPDATE DBA.WORDS
SET QUOTE_SOURCE =
'Jim Dee as quoted in a presentation to BMODUG',
MEANING2 = 'Full of rows, as a relational database';
 . . . .
SUCCESSFUL UPDATE   OF      470 ROW(S)
```

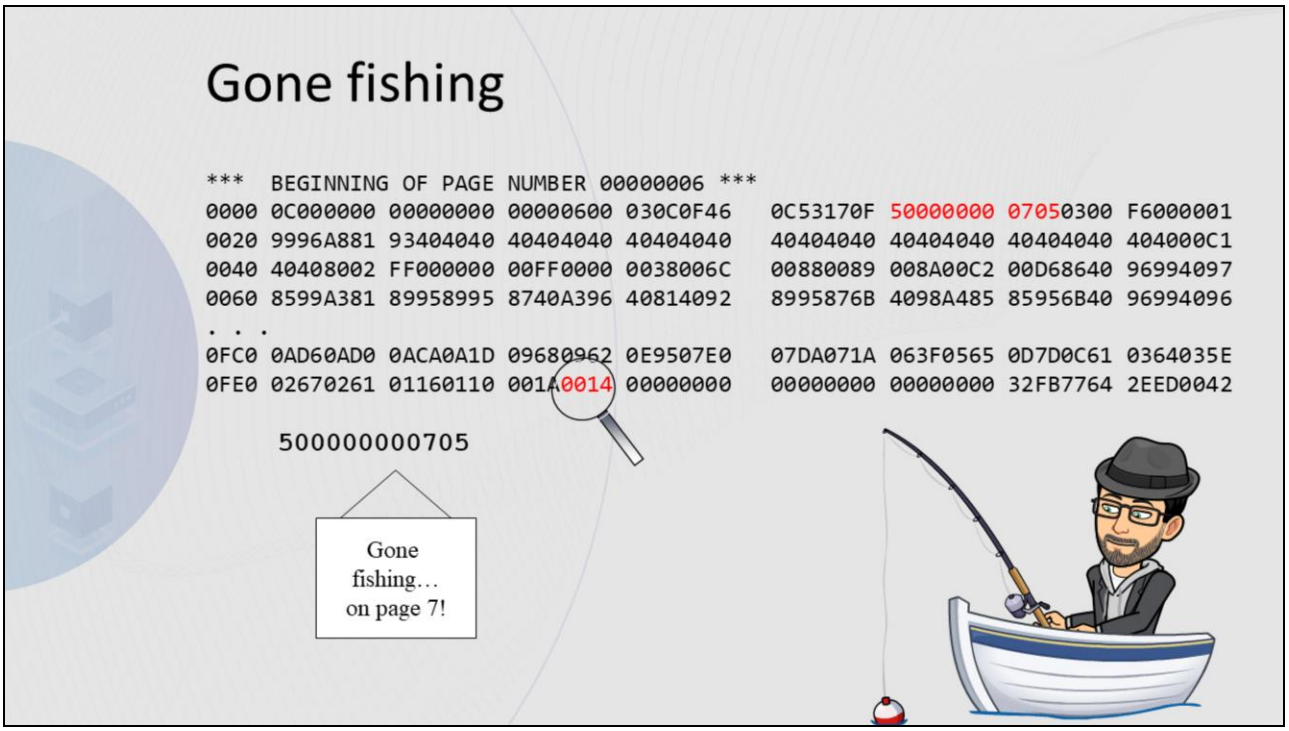
Oops… an unqualified UPDATE… impacted every row in our small database.

The DBA thought he would enter this quote source and a new meaning for Rowdy but forgot the WHERE clause.

He considered just setting the columns to NULL without a WHERE clause. But some words already entered had MEANING2.

So, he used his favorite log tool to generate updates to put things right.

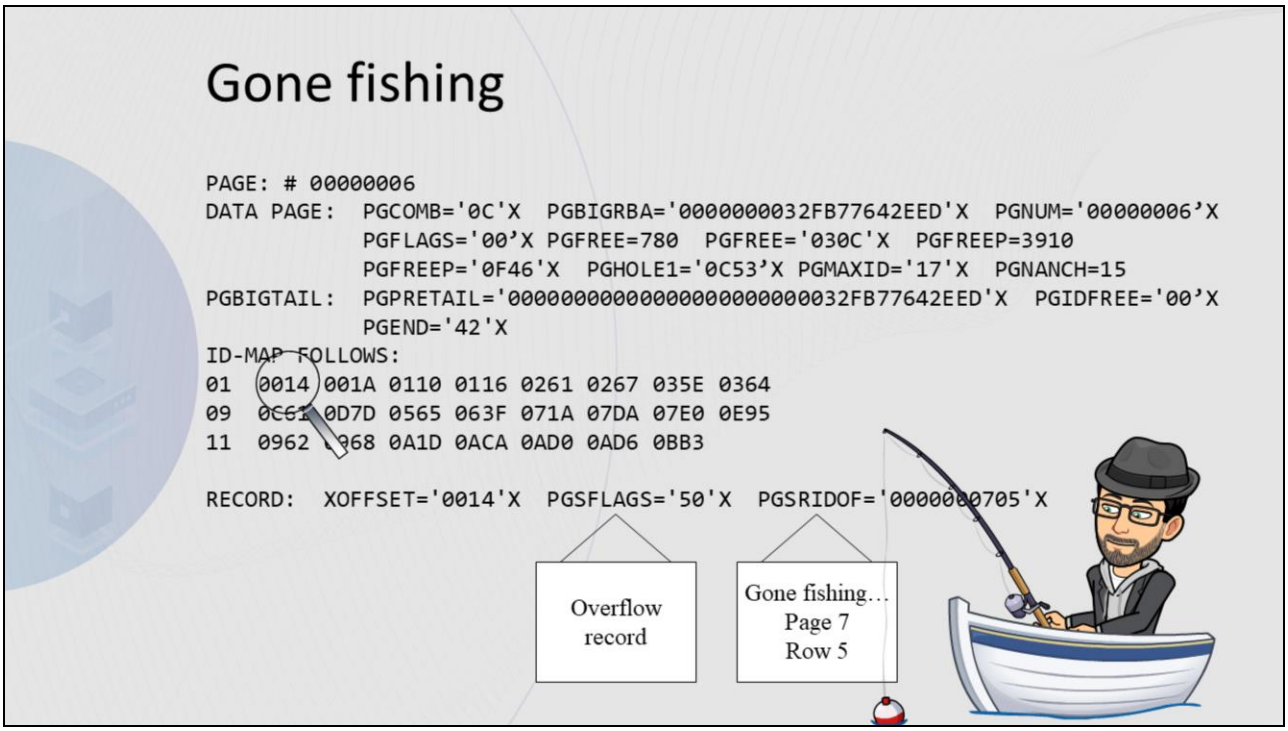
However, all this expanding of rows and contracting them again put internal formats in disarray (even though the unintentional changes are corrected) .



Remember discussing the row header earlier… Well, if the correct bits are on, instead of being a row header, it is now a POINTER record directing DB2 to the new location of the actual data.

The INDEX stays the same pointing to page 6, but when the page is read, DB2 recognizes the pointer and will read the reference page to get the row.

But, you can see that pointers will cause additional I/O activity to retrieve data.



Here's the formatted version of page 6.

Rowdy is still the first row in the ID map. Its offset, x'14' the overflow record bit set in PGSFLAGS, and the PGSRIDOFfset has the new offset of page 7, row 5.

Where is Rowdy Now??

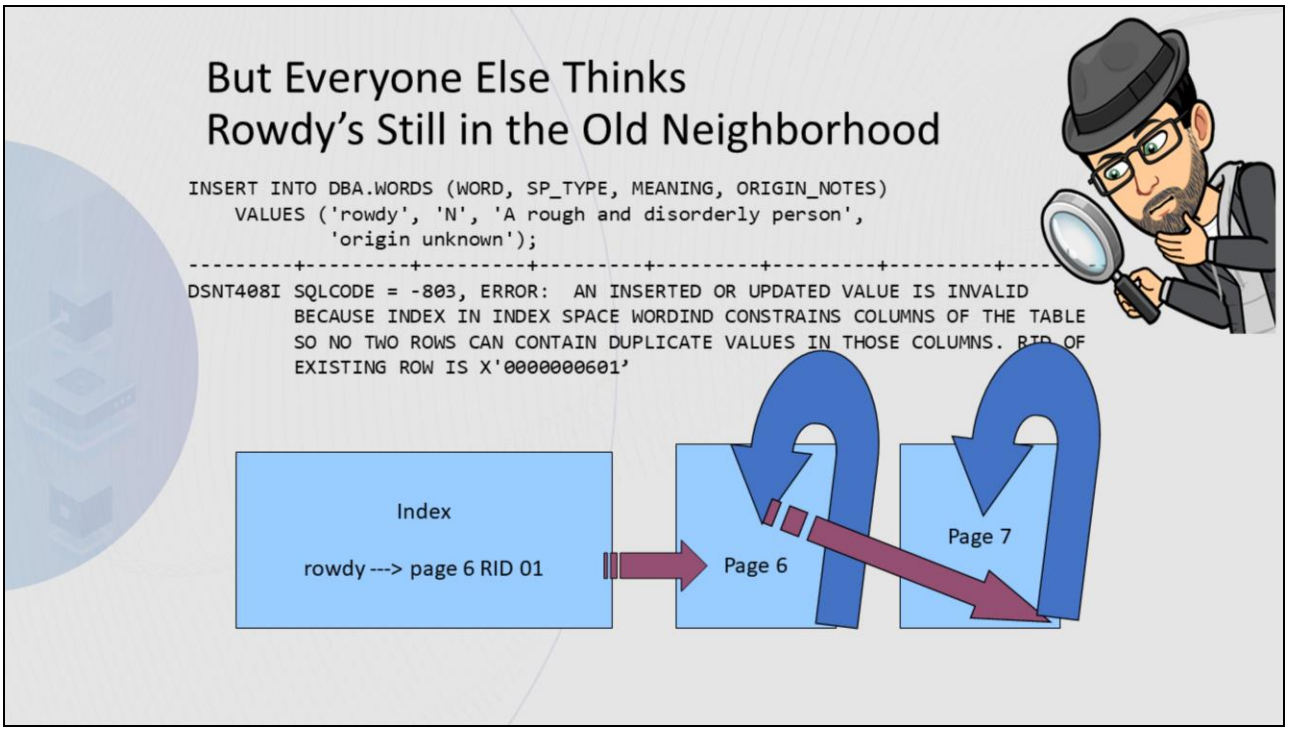
```
***  BEGINNING OF PAGE NUMBER 00000007 ***
0000 08000000 00000000 00000700 05A10D14   0B740C09 80005A00 00019796 97A49381
. . .                                                          rowdy
01E0 81948993 A8FFFF00 E2858540 9996A881   9321 00E6 0000 0199 96A6 84A8 4040 4040
0200 40404040 40404040 40404040 40404040   40404040 4000D540 40800100 20220930
0220 FF000000 38005600 7D007E00 A300B200   C1409996 A4878840 81958440 8489A296
0240 99848599 93A84097 8599A296 9500C6A4   93934096 86409996 A6A26B40 81A24081
. . .
0FC0 00000000 00000000 00000000 00000000   00000000 0B8E077F 06BA05D5 050903DB
0FE0 02D701F1 0AD709F2 091B0868 00000000   00000000 00000000 32FB7764 33840042
```

The pointer was to PAGE 7, ROWID 5. Using the 5th PGMAP offset we find "rowdy".

Where is Rowdy Now??

```
PAGE: # 00000007
DATA PAGE:
ID-MAP FOLLOWS:
01  0868 091B 09F2 0AD7 01F1 02D7 03DB 0509
09  05D5 06BA 077F 0B8E

RECORD:  XOFFSET='01F1'X
9996A684 A8404040 40404040 40404040 40404040 40404040 40404040 404000D5
40408001 00202209 30FF0000 00380056 007D007E 00A300B2 00C14099 96A48788
40819584 408489A2 96998485 9993A840 978599A2 969500C6 A4939340 96864099
96A6A26B 4081A240 81409985 9381A389 96958193 408481A3 818281A2 85FF00E3
88859985 40A681A2 40814099 96A684A8 40899540 A3888540 C9C4E4C7 40839996
A6844B00 96998987 899540A4 95929596 A69500D1 899440C4 85854081 A24098A4
96A38584 40899540 81409799 85A28595 A381A389 969540A3 9640C2D4 D6C4E4C7
```

And here it is again in formatted version



DB2 is smart about not allowing multiple pointers for a single row.

If a subsequent update would cause “rowdy” to overflow from page 6 to page 7…

DB2 deletes the page 6 reference, inserts the updated row into page 7, and updates the home page POINTER to page 7 instead of page 6.

The insert and delete are logged with a bit indicating that they were caused by an UPDATE statement.

If it can UNDERFLOW back to the home page, DB2 will do that as well.

Really smart!

In alphabetical order after the REORG based upon WORDINDX key sequence

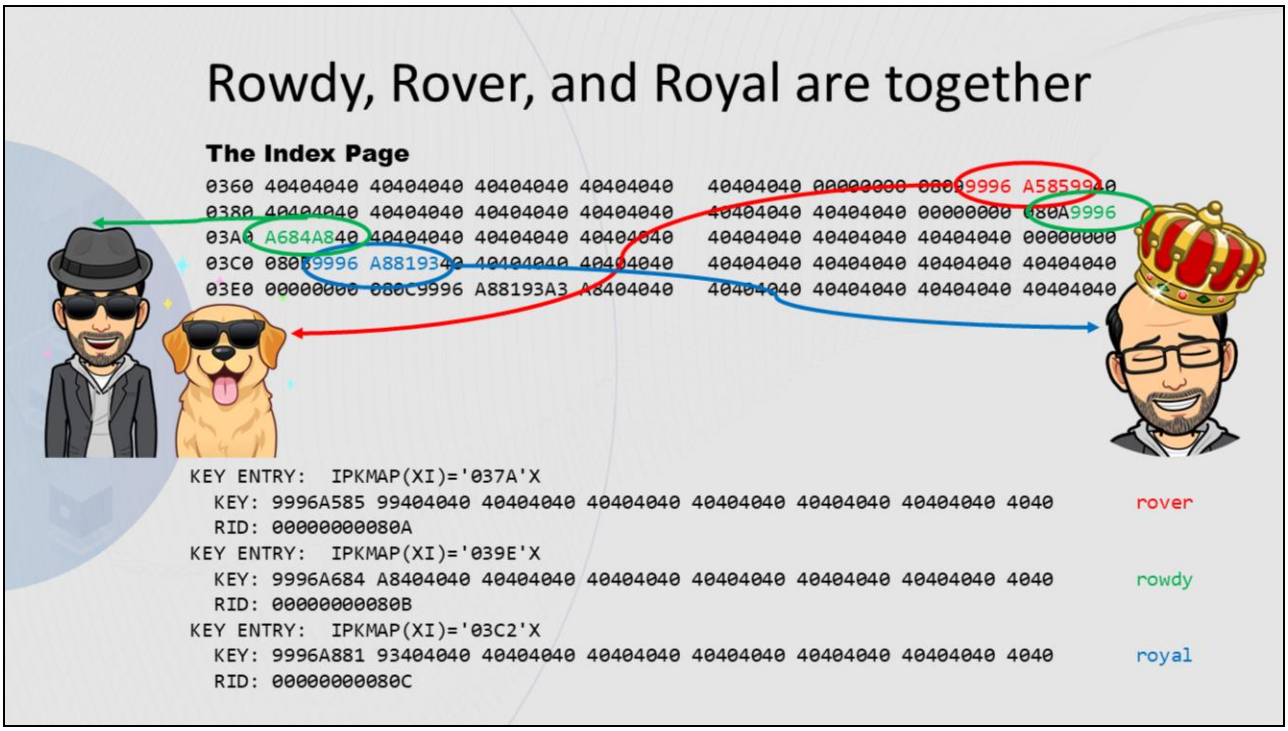
…

rover

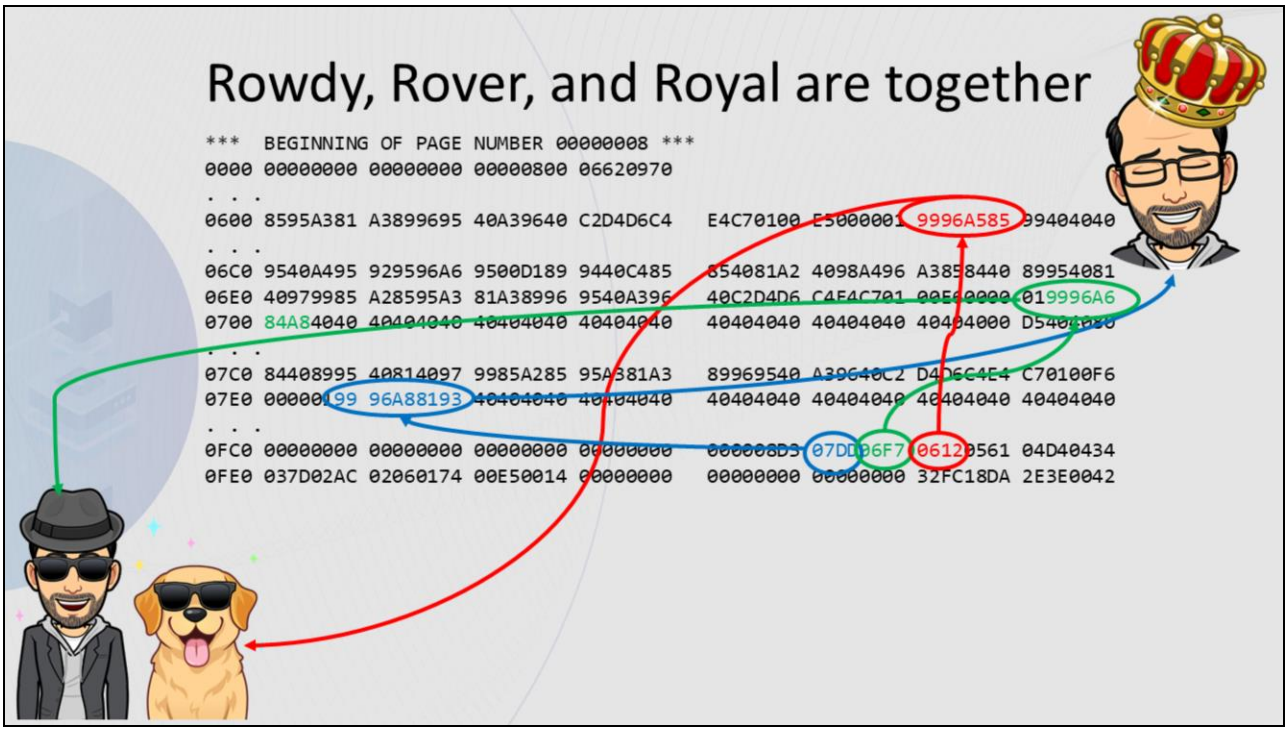
rowdy

royal

…



If you did a REORG LOG YES, each PAGE FORMAT would appear in the db2 log as well.



If you did a REORG LOG YES, each PAGE FORMAT would appear in the db2 log as well.

Not all available pages were used

```
***  BEGINNING OF PAGE NUMBER 00000007 ***
0000 00000000 00000000 00000700 0FD60014   00000100 00000000 00000000 00000000
.... LINES ARE ALL ZERO.
0FE0 00000000 00000000 00000000 00000000   00000000 00000000 32FC18DA 2D990142
```

Remember the FREEPAGE 1 option on the CREATE TABLESPACE? It finally takes effect! So page 7 was not used, and Rover, Rowdy, and Royal find themselves on page 8.

Changes in the FREEPAGE and PCTFREE values will increase the amount of free space on a freshly loaded or reorged tablespace to allow for growth in clustering order.

More Changes to Rowdy's Table

```
ALTER TABLE DBA.WORDS
    ALTER COLUMN MEANING        SET DATA TYPE VARCHAR(250)
    ALTER COLUMN MEANING2       SET DATA TYPE VARCHAR(250)
    ALTER COLUMN MEANING3       SET DATA TYPE VARCHAR(250)
    ALTER COLUMN NUM_REFERENCES SET DATA TYPE INTEGER;
```

After some experience with the WORDS table, the DBA decides that the meaning columns need to be longer.

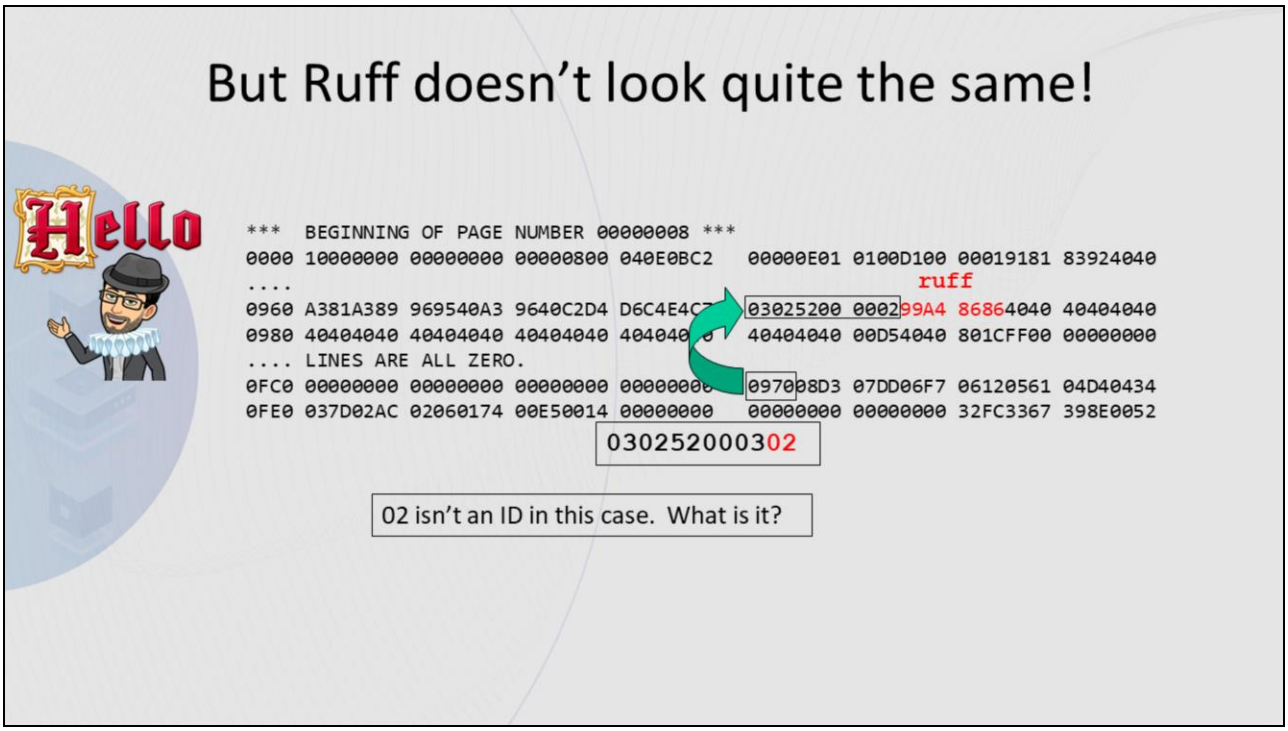
At the same time, interest is so high that the number of references column must be increased in size!

The ALTER VARCHAR does not cause versioning, but the change in the size of the NUM_REFERENCES column does.

Ruff joins the crowd


```
INSERT INTO DBA.WORDS (WORD, SP_TYPE, MEANING,
   MEANING2, MEANING3, ORIGIN_NOTES, NUM_REFERENCES)
   VALUES ('ruff', 'N',
'A stiffly starched, frilled or pleated circular collar of lace, ' ||
'muslin, or other fine fabric worn by men and women in the 16th '  ||
'and 17th centuries.',
'A distinctive, collar-like projection around the neck, as of '    ||
'feathers on a bird or of fur on a mammal.',
'Card Games.  The playing of a trump card when one cannot '        ||
'follow suit.',
'For senses one and two, short for ruffle (frill).  For sense '    ||
'three, Old French roffle, earlier ronfle, probably from Italian ' ||
'ronfa, perhaps alteration of trionfa, "triumph," trump card, '    ||
'from Latin triumphus.',
33072);
```

We insert RUFF after we did the versioning Alter.



This is an example of Versioning… an increase in size of a fixed length column (in this case from two byte SMALLINT to a four byte INTEGER) introduced a new version to map records.

The flag bit indicates that this is a versioned row and that what was originally the ROWID byte instead indicates the VERSION of the row.

And just a reminder that the log record still contains the real ROWID allowing for recovery and log utilities to function.

Actually, we could possibly gain more than just one page. When a space is VERSIONED, additional SYSTEM PAGES are created to store older versions of the DBD which contain the older version row formats to allow for later normalization to the current VERSION.

Information in the Header Page (page 0) point to and contain information about the SYSTEM pages.

The versioned data works!

```
SELECT WORD, MEANING, MEANING2, NUM_REFERENCES
    FROM DBA.WORDS
    WHERE WORD = 'rowdy' OR WORD = 'ruff';
```

```
  +---------+-----------------------------------------+----------------+
  |   WORD  |    MEANING                          ...| NUM_REFERENCES |
  +---------+-----------------------------------------+----------------+
1_| rowdy   | A rough and disorderly person       ...| ?              |
2_| ruff    | A stiffly starched, frilled or pleated...|          33072 |
  +---------+-----------------------------------------+----------------+
```

Versioning works, but...

- Rendering older versions to the current version requires more CPU
- Altering an indexed column forces REBUILD.
- Updating many rows in older version causes overflow.
- REORG as soon as possible.

The REORG will also remove the SYSTEM PAGES that are not necessary if all rows are at the current VERSION.

But, the row headers will remain with the ‘version’ number in the ROWID byte going forward.

Rowdy Gets Deleted

- Scraps of rowdy may still be in the database.
- Rowdy's row will be on the log.
- Rowdy is still in old image copies.

Time for Rowdy to go away.

…Or does he?

```
PAGE: # 00000008
DATA PAGE:
ID-MAP FOLLOWS:
01  0014 00E5 0174 0206 02AC 037D 0434 04D4
09  0561 0612 86F7 07DD 08D3 0970

PD-REC: XOFFSET='06F7'X  PGSFLAGS='03'X  PGSLTH=230  PGSLTH='00E6'X  PGSOBD='0000'X  PGSBID='01'X
9996A684 A8404040 40404040 40404040 40404040 40404040 40404040 404000D5   rowdy                          .N
40408001 00202209 30FF0000 00380056 007D007E 00A300B2 00C14099 96A48788     ..............'.=.t...A rough
40819584 408489A2 96998485 9993A840 978599A2 969500C6 A4939340 96864099    and disorderly person.Full of r
96A6A26B 4081A240 81409985 9381A389 96958193 408481A3 818281A2 85FF00E3   ows, as a relational database..T
88859985 40A681A2 40814099 96A684A8 40899540 A3888540 C9C4E4C7 40839996   here was a rowdy in the IDUG cro
A6844B00 96998987 899540A4 95929596 A69500D1 899440C4 85854081 A24098A4   wd..origin unknown.Jim Dee as qu
96A38584 40899540 81409799 85A28595 A381A389 969540A3 9640C2D4 D6C4E4C7   oted in a presentation to BMODUG
```

As the table space is UTS and PGIDFREE is 0, this is a pseudo-delete and the row is not turned into a hole and the PGMAP entry is not freed.

As you can see, the record is marked as Pseudo-Deleted, and the page map entry is still there, with the broken bit turned on.

Scraps of rowdy

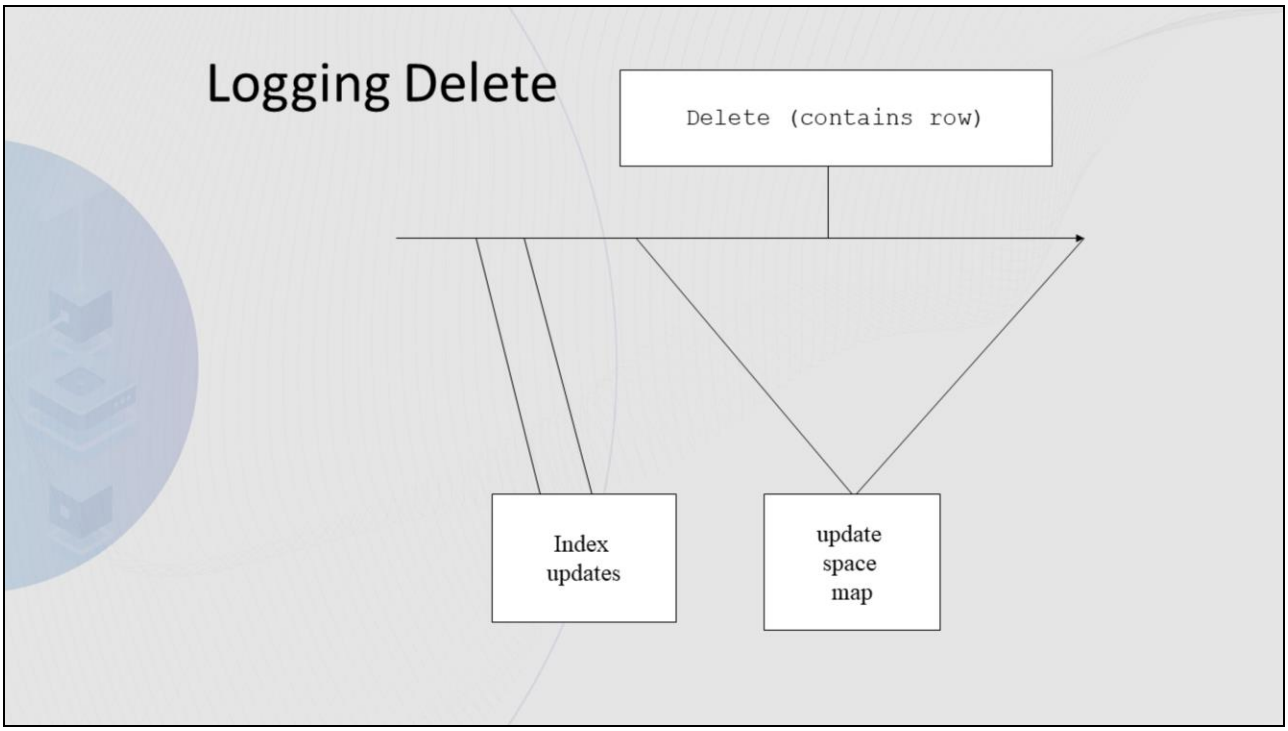

```
***  BEGINNING OF PAGE NUMBER 00000008 ***
0000 08000000 00000000 00000800 04F40BC2   00000E03 0100D100 00019181 83924040
. . . .
06E0 40979985 A28595A3 81A38996 9540A396   40C2D4D6 C4E4C703 00E60000 019996A6
0700 84A84040 40404040 40404040 40404040   40404040 40404040 40404000 D5404080
0720 01002022 0930FF00 00003800 56007D00   7E00A300 B200C140 9996A487 88408195
0740 84408489 A2969984 859993A8 40978599   A2969500 C6A49393 40968640 9996A6A2
0760 6B4081A2 40814099 859381A3 89969581   93408481 A3818281 A285FF00 E3888599
0780 8540A681 A2408140 9996A684 A8408995   40A38885 40C9C4E4 C7408399 96A6844B
07A0 00969989 87899540 A4959295 96A69500   D1899440 C4858540 81A24098 A496A385
07C0 84408995 40814097 9985A285 95A381A3   89969540 A39640C2 D4D6C4E4 C70100F6
.... .
.... LINES ARE ALL ZERO.
0FC0 00000000 00000000 00000000 00000000   097008D3 07DD86F7 06120561 04D40434
0FE0 037D02AC 02060174 00E50014 00000000   00000000 00000000 32FC3486 10D70042
```

9996A684A8

r o w d y

But the data will remain until the space is claimed by other DM activity (INSERT, UPDATE) or REORG occurs to reclaim the space.

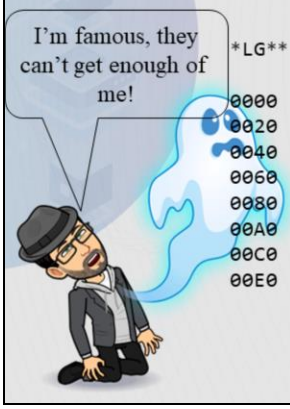
Old images of rowdy still exist on image copies prior to the delete as well as after as long as the space is not reclaimed.



In addition to the physically logged delete of the row, there is index and space map maintenance which occurs with a delete

Final Performance - The Delete

```
032FC348610D7  TYPE( UNDO  REDO )  URID(0000000032FC3486079D)
       LRSN(00DC2F590ADA3A2AA400)  DBID(9855)  OBID(0002)  PART(0001)  PAGE(00000008)
       SUBTYPE(DELETE IN A DATA PAGE)  CLR(NO)  PROCNAME(DSNIDILS)

*LRH* 0000016A 005E0009 0EA00000 00000000 00000000 32FC3486 079D0000 00000000
      00000000 32FC3486 0A6A5000 06000001 00000000 000032FC 34860A6A 000000DC
      2F590ADA 3A2AA400 00000000 00000000
*LG** 08985500 02000000 08000000 000032FC 3367398E 39408000 00010000 00000000
      00000000
0000  00F6200B 00090015 00000000 00000000 0100E600 00019996 A684A840 40404040
0020  40404040 40404040 40404040 40404040 40404040 00D54040 80010020 220930FF
0040  00000038 0056007D 007E00A3 00B200C1 409996A4 87884081 95844084 89A29699
0060  84859993 A8409785 99A29695 00C6A493 93409686 409996A6 A26B4081 A2408140
0080  99859381 A3899695 81934084 81A38182 81A285FF 00E38885 998540A6 81A24081
00A0  409996A6 84A84089 9540A388 8540C9C4 E4C74083 9996A684 4B009699 89878995
00C0  40A49592 9596A695 00D18994 40C48585 4081A240 98A496A3 85844089 95408140
00E0  979985A2 8595A381 A3899695 40A39640 C2D4D6C4 E4C7
```



Just like we looked at the DSN1LOGP print of the INSERT log record for "rowdy" earlier in the presentation, the DELETE is also logged and could be reversed using a log tool or manually deciphered if you are so inclined.

If this was an overflowed row, the home page pointer row delete would also be logged.

How Could Rowdy's Life be Different?

- Some we'll talk about a little
  - ALTER-ing of COLUMNs… either adding or increasing size
  - Row Versioning in DB2 V8.1 and above
  - Compression / Encryption?
  - Reordered Row Format in DB2 V9.1
  - LOB or XML COLUMNs (V6.1 for LOBs, V9.1 for XML)
  - What if I'm an ASCII or UNICODE table?
  - Extended RBA/LSRN in DB2 V11
- Others we won't
  - NON-PADDED INDEX
  - Compressed Index

Column changes

- When VARCHARs are expanded later or columns are added, rows grow and may move off the original page and can actually take an extra read page to get a single row. REORG or design changes can avoid this to some degree (avoiding VARCHAR, avoiding adding columns later or REORG after adding them).
- Versioning to increase column sizes has the same issues, plus conversion overhead during retrieval.
- Certain ALTERs which did not Version a row in V8 will now induce Versioning due to Reordered Row Format introduced in V9
- Example of Versioning discussed in the main flow of the presentation.

Change in space parameters

- FREEPAGE (and PCTFREE) are only honored on REORG and LOAD, so if you only INSERT rows then the order of the inserts, not the clustering key, will determine the order. But if you happen to insert in clustering order, REORG may actually cause MORE pages to be involved in a query for a section of rows.
- The setting of the FREE parameters is dependent upon the DML activity expected for each table…
  - What is the nature of INSERTs in relation to the clustering key
    - Random INSERTs? Need higher FREEs
    - Only increasing INSERTs with the space growing at the end? Lower FREEs
  - Do you expect UPDATEs which will significantly change row sizes?
  - DELETEs expected to create holes?
- Intelligent setting of FREEPAGE & PCTFREE reduce need to REORG

Compression

If Rowdy and his friends were compressed, then

- it would be harder to read the many images of the rows on logs, spaces and copies (which might provide a small measure of data security… but the data dictionary used for compression is contained within the same space or image copy)
- the variable nature of the compressed results might cause more overflow
- more rows would probably fit on a page (reducing the number of pages read for certain queries)

- Additional ENCRYPTION is possible
  - For image copies destined for offsite transport
  - DB2 has some native encryption capabilities
  - Hardware and Software (hardware independent) solutions are available

There are many effects when a space is defined with compression. One is that the data the rows leave on VSAM files and the logs is far less readable by you and me. There are advantages of squeezing more rows on a page which allows, for example, one letter of the alphabet into a buffer pool with fewer I/Os. The data will also take up less space on external media. Compressed rows are, by definition, variable even if no VARCHARs are defined. Compression only takes effect on REORG and LOAD REPLACE. The space contains a dictionary created by these utilities. Once compression is defined and REORG or LOAD REPLACE is executed, then inserted and updated rows will be compressed using the dictionary.

Index keys are not in a compressed format prior to DB2 Version 9. Version 9 does allow for index compression. It's much different than tablespace compression… it's done at the page level instead of the row or index entry level.

Remember that pages are limited by PGMAXID to x'FF' or 255 rows per page… this is regardless of the PAGESIZE – 4K, 8K, 16K, or 32K. If your 255 rows use less space than the PAGESIZE, you have unused space in your database.

Compression

'rowdy' as a non compressed row:

```
0000 1004DD1C 51C9D100 00002100 000D0FCD   00001201 0100C400 030198A7 83818140
...                                row header      r   w d y
04A0 40C2D4D6 C4E4C7FF 00000000 0300E800   03019996 A684A840 40404040 40404040
04C0 40404040 40404040 40404040 40404040   00D50040 001E00C1 409996A4 87884081
04E0 95844084 89A29699 84859993 A8409785   99A29685 002700C6 A4939340 96864099
0500 96A6A26B 4081A240 81409985 9381A389   96958193 408481A3 818281A2 850001FF
0520 81D60020 07112700 2500E388 85998540   A681A240 81409996 A684A840 899540A3
0540 888540C9 C4E4C740 839996A6 844B000F   00969989 87899540 A4959295 96A69500
0560 2E00D189 9440C485 854081A2 4098A496   A3858440 89954081 40979985 A28595A3
                                                    next row begins here
0580 81A38996 9540A396 40C2D4D6 C4E4C700   800007D0 01010300 03019996 A8819340
...
0FC0 9640C2D4 D6C4E4C7 FF000000 00000000   00000000 00000000 00000F09 0E450D81
0FE0 0CBD0BF9 0B350A71 09AD08E9 06970594   04AC03E8 03240260 019C00D8 001400D5
```

This was 'rowdy' on a Version 8 system after all of the test activity (DDL and SQL) and ultimately a REORG. He was living on page x'21'/d'33'.
There were x'12' or 18 rows on the page
With a Length of x'00E8'/d'232'
Version 01 (on a DB2 V8.1 system)
Character data is readable

This is 'rowdy' after I did an ALTER TABLESPACE COMPRESS YES and executed another REORG. Considering that the rows were already in the same order, all this REORG should have done was compress the data.

'rowdy' moved from page x'21'/d'33' to x'15'/d'21' or is now 11 pages earlier in the database.

And, the rows per page jumped from 18 to 225. So, you can see how this could reduce I/O.. But, the cost is cycles somewhere to decompress.

X'E1' or 225 rows on the page
Length of x'004A'/d'74' (158 bytes less than the non-compressed row)
Version 01 (on a DB2 V8.1 system)
Character and all data now mangled into 3 nibble dictionary index entries

Reordered Row Format

- BRF (Basic Row Format) only option prior to DB2 9.1
  - Columns appear in logical order (e.g. COL1, COL2, etc.)
  - Length attributes of VAR columns included with the column
- RRF (Reordered Row Format) introduced in Version 9.1 of DB2
  - All fixed columns appear in logical order at the beginning of the row
  - Followed by an array of offsets to the beginning of each VAR column (from beginning of data)
  - All VAR columns appear in logical order after the offset array
  - Length attribute not included, it must be calculated via the difference between its offset and the subsequent offset or the length of the row for the last column.
  - All newly created tables are in RRF
  - Migrated Spaces (Partitions) converted on first REORG or LOAD REPLACE (with caveats)

The variable length attributes includes ‘1’ for a NULL byte if the column is nullable. So a nullable VARCHAR(30) field which is the maximum length would actually have a length attribute of 31 or x’001F’… (whether as exists in BRF or calculated in RRF).

Numeric column types are encoded on DB2 pages to allow for correct sort ordering. These must be decoded to z/OS format for usability if you are deciphering a row.

Date, Time, and Timestamp columns are also in a packed decimal type format without a sign nibble. Again, that must be accounted for if directly processing a row.

The encoding methodology is discussed in various DB2 manuals.

Basic Row Format

- The column layout at the end of our test:

```
 WORD           CHAR(30) NOT NULL PRIMARY KEY
,SP_TYPE        CHAR(3)
,MEANING        VARCHAR(250)
,MEANING2       VARCHAR(250)
,MEANING3       VARCHAR(250)
,CODENUM        SMALLINT GENERATED BY DEFAULT AS IDENTITY
                    CACHE 50,CYCLE
,QUOTE_DATE     DATE
,QUOTE          VARCHAR(500)
,ORIGIN_NOTES   VARCHAR(250)
,QUOTE_SOURCE   VARCHAR(100)
,NUM_REFERENCES INTEGER
```

The fixed columns are in blue, bolded, and italicized above.

Basic Row Format

```
0000 1004DD1C 51C9D100 00002100 000D0FCD   00001201 0100C400 030198A7 83818140
...                             row header     r o  w d y
                                               WORD
04A0 40C2D4D6 C4E4C7FF 00000000 0300E800   03019996 A684A840 40404040 40404040
                                           N SP_TYP len N MEANING
04C0 40404040 40404040 40404040 40404040            001E00C1 409996A4 87884081
                                                    len N MEANING2
04E0 95844084 89A29699 84859993 A8409785   99A29695
                                                          (MEANING3) len N
0500                                                                  0001FF
 CODENUM N QUOTE_DT len  N QUOTE
0520     0020 071127
                                      len    N ORIGIN_NOTES                  len
0540                                  000F   00969989 87899540 A4959295 96A695
       N QUOTE_SOURCE
0560
                                        N    NUM_REFS next row begins here
0580                                    00   800007D0 01010300 03019996 A8819340
...
0FC0 9640C2D4 D6C4E4C7 FF000000 00000000   00000000 00000000 00000F09 0E450D81
0FE0 0CBD0BF9 0B350A71 09AD08E9 06970594   04AC03E8 03240260 019C00D8 001400D5
```

BASIC ROW FORMAT…

Logical column order. If you have a color copy, I alternated the color of the column values.

N = NULL BYTE. X'00' not NULL, x'FF' is NULL.

The COLUMN NAME appears at the beginning of the column data after its corresponding 'N'ULL byte.

(MEANING3) appears above and before it's physical location because it was NULL and I could barely fit it in.

Reordered Row Format

- How the columns look on the page:

```
 WORD            CHAR(30) NOT NULL PRIMARY KEY
,SP_TYPE         CHAR(3)
,CODENUM         SMALLINT GENERATED BY DEFAULT AS IDENTITY
                     CACHE 50,CYCLE
,QUOTE_DATE      DATE
,NUM_REFERENCES  INTEGER
,MEANING         VARCHAR(250)
,MEANING2        VARCHAR(250)
,MEANING3        VARCHAR(250)
,QUOTE           VARCHAR(500)
,ORIGIN_NOTES    VARCHAR(250)
,QUOTE_SOURCE    VARCHAR(100)
```

The fixed columns are in blue, bolded, and italicized above.

Reordered Row Format

```
0000 10007D2C 6E02FB00 00002400 000D0FCD   00001201 0100C400 030298A7 83818140
...                             row header     r o  w d y
                                               WORD
04A0 969540A3 9640C2D4 D6C4E4C7 03008800   03029996 A684A840 40404040 40404040
                                           N SP_TYP COD N QUOTE_DT N  NUM_REFS
04C0 40404040 40404040 40404040 40404040            81D6           00 800007D0
     off off  off off  off off  N MEANING1
04E0 003A     007F     00A5     00C14099   96A48788 40819584 408489A2 96998485
                           N MEANING2
0500 9993A840 978599A2 9695
                                  (MEANING3) N N QUOTE
0520                                         FF
                                                    N  ORIGIN_NOTES
0540                                                00 96998987 899540A4
                   N QUOTE_SOURCE
0560 95929596 A695
                                                 next row begins here
0580                                             01010300 03029996 A8819340
...
0FC0 89969540 A39640C2 D4D6C4E4 C7000000   00000000 00000000 00000F09 0E450D81
0FE0 0CBD0BF9 0B350A71 09AD08E9 06970594   04AC03E8 03240260 019C00D8 001400D5
```

Reorderd Row Format… It's the same length.

All fixed columns appear first in their COLNO order.

Each 'off' relates to each VAR which follows in their COLNO order. The offset is from the beginning of actual data, not from the row header.

VAR lengths are calculated by using the next offset… (next off – my off)

The last VAR column length is calculated by using the row header length as the terminating point (row header – off).

Adds efficiency… If you want to SELECT QUOTE_SOURCE, DB2 can index right to it by using the correct offset versus having to traverse the entire row.

LOB and XML Columns

- LOB and XML Column differences
  - No longer just the base TABLESPACE (partitions)
    - An additional space per LOB/XML column per part (LOB only) exists to hold the column data
    - Additional indexes for AUXILIARY parts too
    - Additional column (ROWID for LOB, DOCID for XML) in base table to reference
    - Base table has a reference for each LOB/XML column used to index into LOB/XML parts
  - Implicates more logging (even if LOG NO)
  - Implicates more I/O to fetch a single row
  - Utilities (e.g., COPY) need to consider all related spaces
  - High-level and Low-level space map pages

ASCII or UNICODE

- All examples so far have been on an EBCDIC table
- What if DBAWORDS was defined as ASCII or UNICODE?
  - Really, no physical difference… only character code points change
  - Basic Latin character set for UNICODE same as ASCII
  - Encoding scheme and CCSID used to store data would be different
  - CHAR and VARCHAR data would be in the appropriate CCSIDs
  - SORT sequence is different for EBCDIC than ASCII/UNICODE
    - EBCDIC – Lower case, Upper case, Numeric
    - UNICODE – Numeric, Upper case, Lower case
  - Are you as familiar with ASCII code points as you are with EBCDIC? e.g:
    - `c'a' = x'81' EBCDIC and x'61' ASCII`
    - `c'N' = x'D5' EBCDIC and x'4E' ASCII`
    - `c'1' = x'F1' EBCDIC and x'31' ASCII`
    - `c'rowdy' or x'9996A684A8' becomes x'726F776479'`

Extended RBA/LRSN

The Data Page

Before Extended RBA/LSRN

```
***  BEGINNING OF PAGE NUMBER 00000002 ***
0000 1002930A 51D05800 00000200 0F7F007D   00000101 02006900 03019996 A684A840
0020 40404040 40404040 40404040 40404040   40404040 40404040 00D54040 001E00C1
0040 409996A4 87884081 95844084 89A29699   84859993 A8409785 99A29695 0001FF00
0060 01FF8001 FF000000 000001FF 000F0096   99898789 9540A495 929596A6 95000000
.... LINES ARE ALL ZERO.
0FE0 00000000 00000000 00000000 00000000   00000000 00000000 00000000 001400D5
```

After Extended RBA/LSRN

```
0000 10000000 00000000 00000200 0F6D007D   00000101 02006900 0301D9D6 E6C4E840
0020 40404040 40404040 40404040 40404040   40404040 40404040 00D54040 8001FF00
0040 00000000 33005100 52005300 5400C140   D9D6E4C7 C840C1D5 C440C4C9 E2D6D9C4
0060 C5D9D3E8 40D7C5D9 E2D6D5FF FFFF00D6   D9C9C7C9 D540E4D5 D2D5D6E6 D5000000
.... LINES ARE ALL ZERO.
0FE0 00000000 00000000 00000014 00000000   00000000 00000000 05569570 97970052
```

Prior to Extended RBA/LSRN

Shows ROWID 1 starting after the last 2 bytes of the page. x0FFC

After you go to DB2 V11 and perform a get your object using Extended RBA.

- Create after you are DB2 V11
- Load REPLACE after at DB2 V11
- REORG after at DB2 V11

Shows ROWID 1 starting after the last 20 bytes of the page. xFEA0

You may also note that the data above is Basic Row format and below is Reordered Row Format

`Bye, Rowdy!

- We followed Rowdy
  - Building his home (CREATE TABLESPACE/TABLE)
  - His birth (INSERT INTO TABLE)
  - Growing up (UPDATE)
  - Making friends (other INSERTs)
  - Changing circumstances (ALTER TABLE)
  - His sad departure (DELETE)
  - His biography (LOG)
- We have seen how his life might have been
  - Row formats
  - Compression
- We learned a little about Db2 in the process (I hope)

IDUG

2022 EMEA Db2 Tech Conference

IDUG

Please fill out your session evaluation!