DB 2

03 – Wide Table Storage

Summer 2018

Torsten Grust
Universität Tübingen, Germany

1 Q_2 — Querying a Wider Table

The next **SQL probe** Q_2 looks just Q_1. We query a wider table now, however:

```
SELECT t.*              -- * ≡ access all columns of row t
FROM   ternay AS t
```

> Retrieve all rows (in some arbitrary order) and all columns of table ternary, a three-column table created by SQL DDL statement

```
CREATE TABLE ternary (a int  NOT NULL,
                      b text NOT NULL, -- variable width
                      c float);        -- may be NULL
```

```
db2=# DROP TABLE IF EXISTS ternary;
db2=# CREATE TABLE ternary (a int NOT NULL, b text NOT NULL, c float);

db2=# INSERT INTO ternary(a, b, c)
        SELECT i,
               md5(i::text),
               log(i)
        FROM   generate_series(1, 1000, 1) AS i;

-- Q2: Retrieve all rows (in arbirary oder) and all columns of table ternary
db2=# SELECT t.*
      FROM   ternary AS t;

-- Equivalent to Q2
db2=# TABLE ternary;

db2=# EXPLAIN VERBOSE
      SELECT t.*
      FROM ternary AS t;
```

Using **EXPLAIN** on Q_2

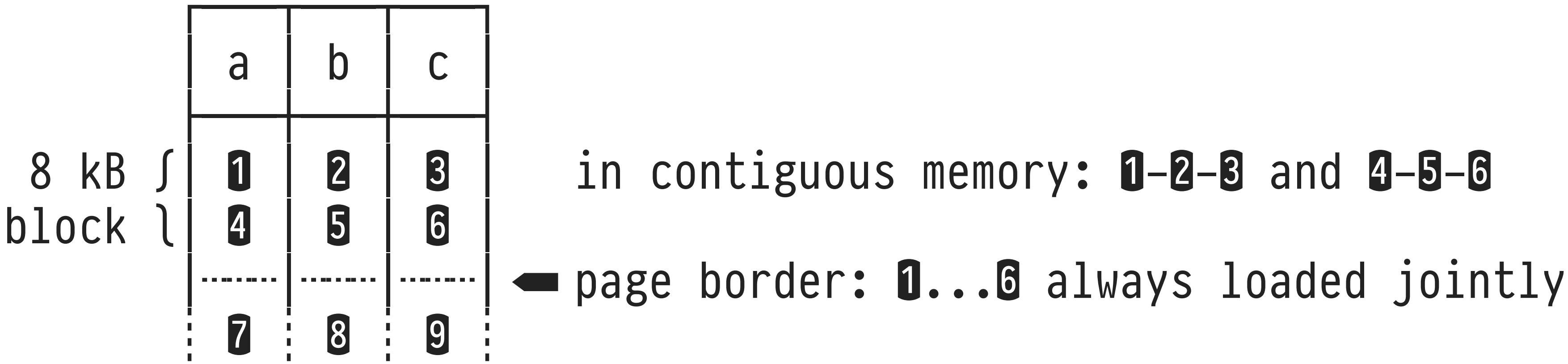
```
EXPLAIN VERBOSE
  SELECT t.*
  FROM   ternary AS t;
```

QUERY PLAN
Seq Scan on public.ternary t (cost=0.00..20.00 rows=1000 width=45)   Output: a, b, c

- Each row `t` carries multiple columns (`a`, `b`, `c`).
- **Seq Scan** scans wider rows now, *average* width: 45 bytes = 4 (int) + 33 (text) + 8 (float) bytes.
    - Column `b` of type text leads to **variable-width rows** in general.

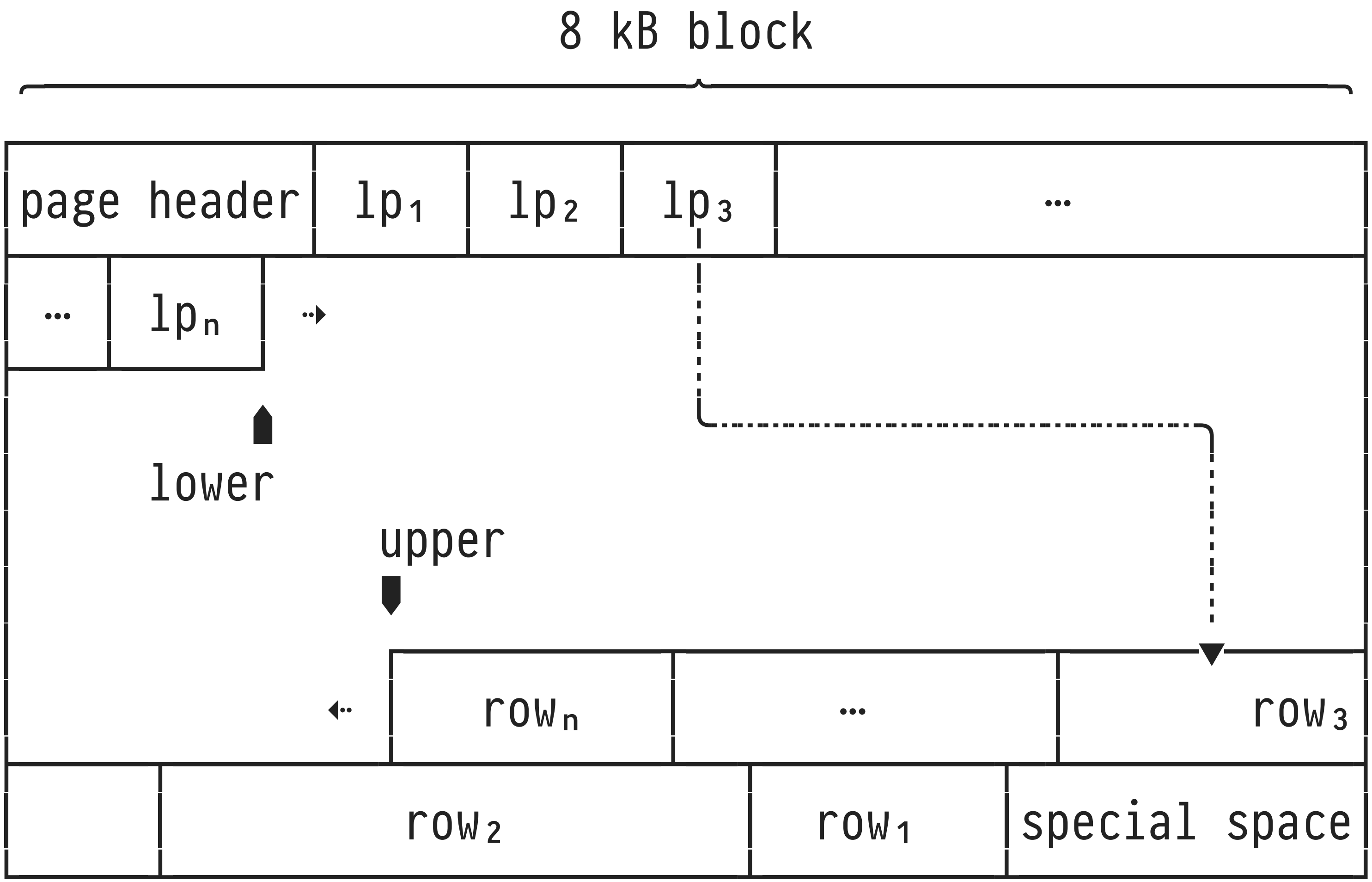
PostgreSQL: Row Storage

- PostgreSQL implements **row storage:** all columns of a row t are held in contiguous memory (≡ same heap file page):



- Loading one heap file page reads **all columns** of all contained rows (recall: block I/O), regardless of whether the query uses t.* or t.a to access the row.

The Innards of a Heap File Page



- lp_i: row pointer
- ⇢/⇠: direction of growth
- upper − lower: free page space
- row_i: payload

The Innards of a Heap File Page

Comments on the previous slide:

- **Page header** (24 bytes) carries page meta information.
- **Special space** is unused on regular table heap file pages (but used on index pages → later).
- Page is full if pointers **lower** and **upper** meet (row pointers and payload grow towards ⋯▸◂⋯ each other).
- **Row pointer** (or: line pointer, 4 byte) lp_i points to row_i, admits **variable-width rows** and **intra-page row relocation** (→ row updates).
- Internal structure of row payloads row_i addressed later.

The Innards of a Heap File Page

PostgreSQL comes with extension pageinspect that provides an "X-ray for heap file pages":

```
CREATE EXTENSION IF NOT EXISTS pageinspect;

-- inspect page header (first 24 bytes)
SELECT *
FROM   page_header(get_raw_page('ternary', ‹page›));

-- inspect row pointers (lp_i)
SELECT *
FROM   heap_page_items(get_raw_page('ternary', ‹page›));
```

1 Inspect page header:

```
db2=# CREATE EXTENSION IF NOT EXISTS pageinspect;

db2=# SELECT t.ctid, t.* FROM ternary AS t;
```

ctid	a	b	c
(0,1)	1	c4ca4238a0b923820dcc509a6f75849b	0
(0,2)	2	c81e728d9d4c2f636f067f89cc14862c	0.301029995663981
(0,3)	3	eccbc87e4b5ce2fe28308fd9f2a7baf3	0.477121254719662
(0,4)	4	a87ff679a2f3e71d9181a67b7542122c	0.602059991327962
(0,5)	5	e4da3b7fbbce2345d7772b0674a318d5	0.698970004336019
[...]			
(9,34)	997	ec5aa0b7846082a2415f0902f0da88f2	2.99869515831166
(9,35)	998	9ab0d88431732957a618d4a469a0d4c3	2.99913054128737
(9,36)	999	b706835de79a2b4e80506f582af3676a	2.99956548822598
(9,37)	1000	a9b7ba70783b617e9998dc4dd82eb3c5	3

37 rows on page 9

```
db2=# SELECT * FROM page_header(get_raw_page('ternary', 0));
```

lsn	checksum	flags	lower	upper	special	pagesize	version	prune_xid
1/D4073838	0	0	452	488	8192	8192	4	0

488 - 452 = 36 bytes of free space
(36 < 45 bytes row size: cannot fit more rows)

special @ page end
(⇒ no special space)

```
db2=# SELECT * FROM page_header(get_raw_page('ternary', 9));
```

lsn	checksum	flags	lower	upper	special	pagesize	version	prune_xid
1/D408A420	0	0	172	5528	8192	8192	4	0

5528 - 172 = 5356 bytes of free space

(172 - 24) bytes / 4 bytes = 37 row pointers ≡ 37 rows on page 9

page header

per row pointer

Cross-checking with the free space information for table ternary:

```
db2=# VACUUM ternary;
db2=# SELECT * FROM pg_freespace('ternary');
```

blkno	avail	
0	32	← approximates 36 free bytes in 32-byte units
1	32	
2	32	
3	32	
4	32	
5	32	
6	32	
7	32	
8	32	
9	5344	← approximates 5356 free bytes in 32-byte units

2 Inspect row pointers (on page 0):

```
db2=# SELECT lp, lp_off, lp_len, t_hoff, t_ctid, t_infomask :: bit(16), t_infomask2
      FROM   heap_page_items(get_raw_page('ternary', 0));
```

lp	lp_off	lp_len	t_hoff	t_ctid	t_infomask	t_infomask2
1	8120	72	24	(0,1)	0000100100000010	3
2	8048	72	24	(0,2)	0000100100000010	3
3	7976	72	24	(0,3)	0000100100000010	3
[...]						
106	560	72	24	(0,106)	0000100100000010	3
107	488	72	24	(0,107)	0000100100000010	3

Also see https://www.postgresql.org/docs/10/static/storage-page-layout.html.

- lp: number i of lp_i
- lp_off: location/offset of row_i on the page
- lp_len: width of tuple (includes row header of 24 bytes): 24 bytes header + 45 bytes payload = 69 bytes ≈ 72 bytes (rounded to 8 bytes)
- t_hoff: offset to row payload data (beyond header and NULL bitmap)
- t_ctid: row ID of this row or its next newer version (MVCC)
    - versions of the same row form a chain connected by their t_ctid fields, chain ends when t_ctid points to the current row
- t_infomask:
    - xxxxxxxxxxxxxxx0: does tuple have any NULL attribute values?
    - xxxxxxxxxxxxxx0x: does tuple have any variable-length attributes?
    - xxxxxxxxxxxxx0xx: have attributes of this tuple been stored externally?
    - xx0xxxxxxxxxxxxx: is this an UPDATEd version of the tuple?
- t_infomask2: number of attributes (lower 11 bits + additional flags)

❸ Cross-check in on-disk heap file for contents of the third row (lp = 3):

```
db2=# SELECT oid FROM pg_database WHERE datname = 'db2';
```

oid
71857

```
db2=# SELECT relfilenode FROM pg_class WHERE relname = 'ternary';
```

relfilenode
80247

```
db2=# show data_directory;
```

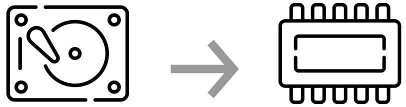
data_directory
/Users/grust/Library/Application Support/Postgres/var-10

```
$ cd '/Users/grust/Library/Application Support/Postgres/var-10/base/71857'
$ hexdump -C -n 72 -s 7976 80247
00001f28  58 31 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |X1..............|
00001f38  03 00 03 00 02 09 18 00  03 00 00 00 43 65 63 63  |............Cecc|
00001f48  62 63 38 37 65 34 62 35  63 65 32 66 65 32 38 33  |bc87e4b5ce2fe283|
00001f58  30 38 66 64 39 66 32 61  37 62 61 66 33 00 00 00  |08fd9f2a7baf3...|
00001f68  fd d5 4f 96 27 89 de 3f                           |..O.'..?|
00001f70
```

```
db2=# SELECT t.ctid, t.* FROM ternary AS t LIMIT 3;
```

ctid	a	b	c
(0,1)	1	c4ca4238a0b923820dcc509a6f75849b	0
(0,2)	2	c81e728d9d4c2f636f067f89cc14862c	0.301029995663981
(0,3)	3	eccbc87e4b5ce2fe28308fd9f2a7baf3	0.477121254719662

2 ¦ Q_2 — Querying a Wider Table



Recall SQL probe Q_2:

```
SELECT t.*            -- * ≡ access all columns of row t
FROM   ternay AS t
```

It is expected that the retrieval of all columns via `t.*` has consequences for a column-oriented DBMS. We need to **touch and synchronize multiple column vectors.**

Create and populate table ternary on MonetDB:

```
$ mclient -d scratch
Welcome to mclient, the MonetDB/SQL interactive terminal (Jul2017-SP4)
Database: MonetDB v11.27.13 (Jul2017-SP4), 'scratch'
Type \q to quit, \? for a list of available commands
auto commit mode: on
sql>DROP TABLE IF EXISTS ternary;
sql>CREATE TABLE ternary (a int NOT NULL, b text NOT NULL, c double);

sql>INSERT INTO ternary(a, b, c)
      SELECT value, md5(value), log(value)
      FROM   generate_series(1, 1001);

sql>SELECT t.* FROM ternary AS t LIMIT 5;
+------+----------------------------------+--------------------------+
| a    | b                                | c                        |
+======+==================================+==========================+
|    1 | c4ca4238a0b923820dcc509a6f75849b |                        0 |
|    2 | c81e728d9d4c2f636f067f89cc14862c |       0.6931471805599453 |
|    3 | eccbc87e4b5ce2fe28308fd9f2a7baf3 |       1.0986122886681098 |
|    4 | a87ff679a2f3e71d9181a67b7542122c |       1.3862943611198906 |
|    5 | e4da3b7fbbce2345d7772b0674a318d5 |       1.6094379124341003 |
+------+----------------------------------+--------------------------+
```

Using **EXPLAIN** on Q_2

MAL program for Q_2, shortened and formatted:

```
 ⋮
X_4               := sql.mvc();
C_5 :bat[:oid] := sql.tid(X_4, "sys", "ternary");
X_25:bat[:dbl] := sql.bind(X_4, "sys", "ternary", "c",…);
X_31:bat[:dbl] := algebra.projection(C_5, X_25);
X_18:bat[:str] := sql.bind(X_4, "sys", "ternary", "b",…);
X_24:bat[:str] := algebra.projection(C_5, X_18);
X_8 :bat[:int] := sql.bind(X_4, "sys", "ternary", "a",…);
X_17:bat[:int] := algebra.projection(C_5, X_8);
 ⋮
 <create schema of result table>
 ⋮
sql.resultSet(…, X_17, X_24, X_31);
```

N-ary vs Decomposed Storage Model (NSM/DSM)

MonetDB follows the **Decomposed Storage Model (DSM)** and represents *n*-ary tables using **full vertical fragmentation:**

a	b	c
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_3
a_4	b_4	c_4
a_5	b_5	c_5

NSM (*n*-ary table)

a:bat[:τ_1]

head	tail
0@0	a_1
1@0	a_2
2@0	a_3
3@0	a_4
4@0	a_5

b:bat[:τ_2]

head	tail
0@0	b_1
1@0	b_2
2@0	b_3
3@0	b_4
4@0	b_5

c:bat[:τ_3]

head	tail
0@0	c_1
1@0	c_2
2@0	c_3
3@0	c_4
4@0	c_5

"row" 2@0

DSM (*n* binary tables)

Turn Heads 90°

▤ row storage

a	b	c
❶	❷	❸
❹	❺	❻
❼	❽	❾

▥ column storage

a	b	c
❶	❹	❼
❷	❺	❽
❸	❻	❾

in contiguous memory:
❶-❷-❸, ❹-❺-❻, ❼-❽-❾

- Both types of DBMS exhibit strengths/weaknesses for different classes of workloads (→ OLTP vs. OLAP).

Positional BAT Joins

Reconstruction of the *n*-ary table requires *n* BATs that are **synchronized on their heads** (≡ identical cardinality).

- Conceptually: (*n*−1)-fold natural ⋈ on the head columns.
- Implemented: synchronized scan of the *n* tail columns:

head	tail
0@0	a_1
1@0	a_2
2@0	a_3

head	tail
0@0	b_1
1@0	b_2
2@0	b_3

head	tail
0@0	c_1
1@0	c_2
2@0	c_3

- See variadic MAL builtin `io.print(…,…)`, for example.

EXPLAIN Q_2 in mclient SQL REPL, then replay plan at mserver5 console:

```
sql>EXPLAIN SELECT t.* FROM ternary AS t;
+-----------------------------------------------------------------------------------------------------+
| mal                                                                                                 |
+=====================================================================================================+
| function user.s16_1():void;                                                                         |
|     X_1:void := querylog.define("explain select t.* from ternary as t;", "default_pipe", 41:int);  |
|     X_33 := bat.new(nil:str);                                                                       |
|     X_39 := bat.new(nil:int);                                                                       |
|     X_37 := bat.new(nil:int);                                                                       |
|     X_36 := bat.new(nil:str);                                                                       |
|     X_35 := bat.new(nil:str);                                                                       |
|     X_4 := sql.mvc();                                                                               |
|     C_5:bat[:oid] := sql.tid(X_4, "sys", "ternary");                                                |
|     X_25:bat[:dbl] := sql.bind(X_4, "sys", "ternary", "c", 0:int);                                  |
|     X_31 := algebra.projection(C_5, X_25);                                                          |
|     X_18:bat[:str] := sql.bind(X_4, "sys", "ternary", "b", 0:int);                                  |
|     X_24 := algebra.projection(C_5, X_18);                                                          |
|     X_8:bat[:int] := sql.bind(X_4, "sys", "ternary", "a", 0:int);                                   |
|     X_17 := algebra.projection(C_5, X_8);                                                           |
|     X_40 := bat.append(X_33, "sys.t");                                                              |
|     X_42 := bat.append(X_35, "a");                                                                  |
|     X_44 := bat.append(X_36, "int");                                                                |
|     X_46 := bat.append(X_37, 32:int);                                                               |
|     X_48 := bat.append(X_39, 0:int);                                                                |
|     X_50 := bat.append(X_40, "sys.t");                                                              |
|     X_51 := bat.append(X_42, "b");                                                                  |
|     X_53 := bat.append(X_44, "clob");                                                               |
|     X_55 := bat.append(X_46, 0:int);                                                                |
|     X_56 := bat.append(X_48, 0:int);                                                                |
|     X_57 := bat.append(X_50, "sys.t");                                                              |
|     X_58 := bat.append(X_51, "c");                                                                  |
|     X_60 := bat.append(X_53, "double");                                                             |
|     X_62 := bat.append(X_55, 53:int);                                                               |
|     X_64 := bat.append(X_56, 0:int);                                                                |
|     sql.resultSet(X_57, X_58, X_60, X_62, X_64, X_17, X_24, X_31);                                  |
| end user.s16_1;                                                                                     |
[...]
```

Replay plan at mserver5 console:

```
sql.init();
sql             := sql.mvc();

ternary:bat[:oid] := sql.tid(sql, "sys", "ternary");
c0      :bat[:dbl] := sql.bind(sql, "sys", "ternary", "c", 0:int);
c                 := algebra.projection(ternary, c0);
b0      :bat[:str] := sql.bind(sql, "sys", "ternary", "b", 0:int);
b                 := algebra.projection(ternary, b0);
a0      :bat[:int] := sql.bind(sql, "sys", "ternary", "a", 0:int);
a                 := algebra.projection(ternary, a0);

io.print(a,b,c)  ⬅  prints three-column table (+ leading void column)
```

sql.resultSet(...,a,b,c) creates ternary result table (of name sys.t, see row variable t) and adds column name and type information.