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:

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

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

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

Using **EXPLAIN** on Q_2

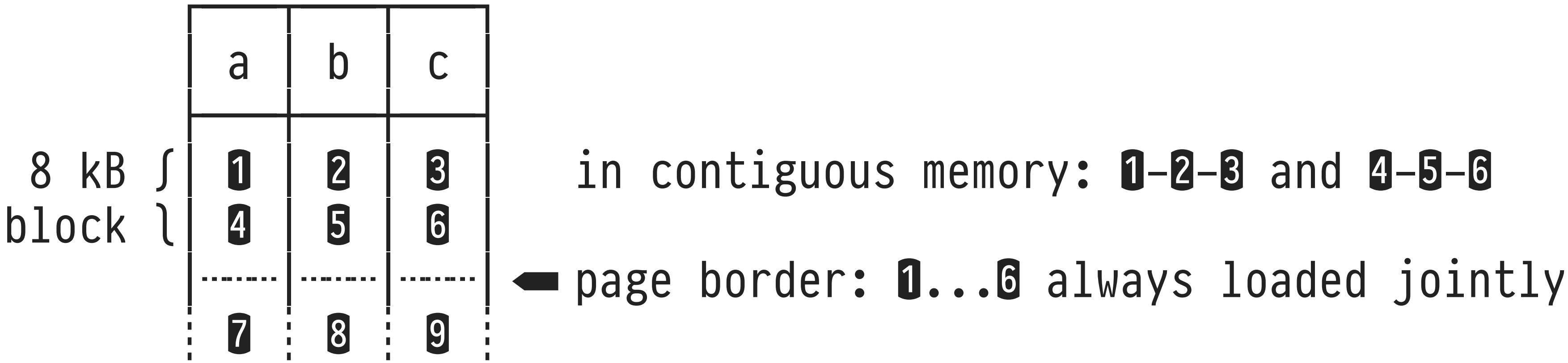
```
EXPLAIN VERBOSE
  SELECT t.*
  FROM   ternary AS t;
```

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.

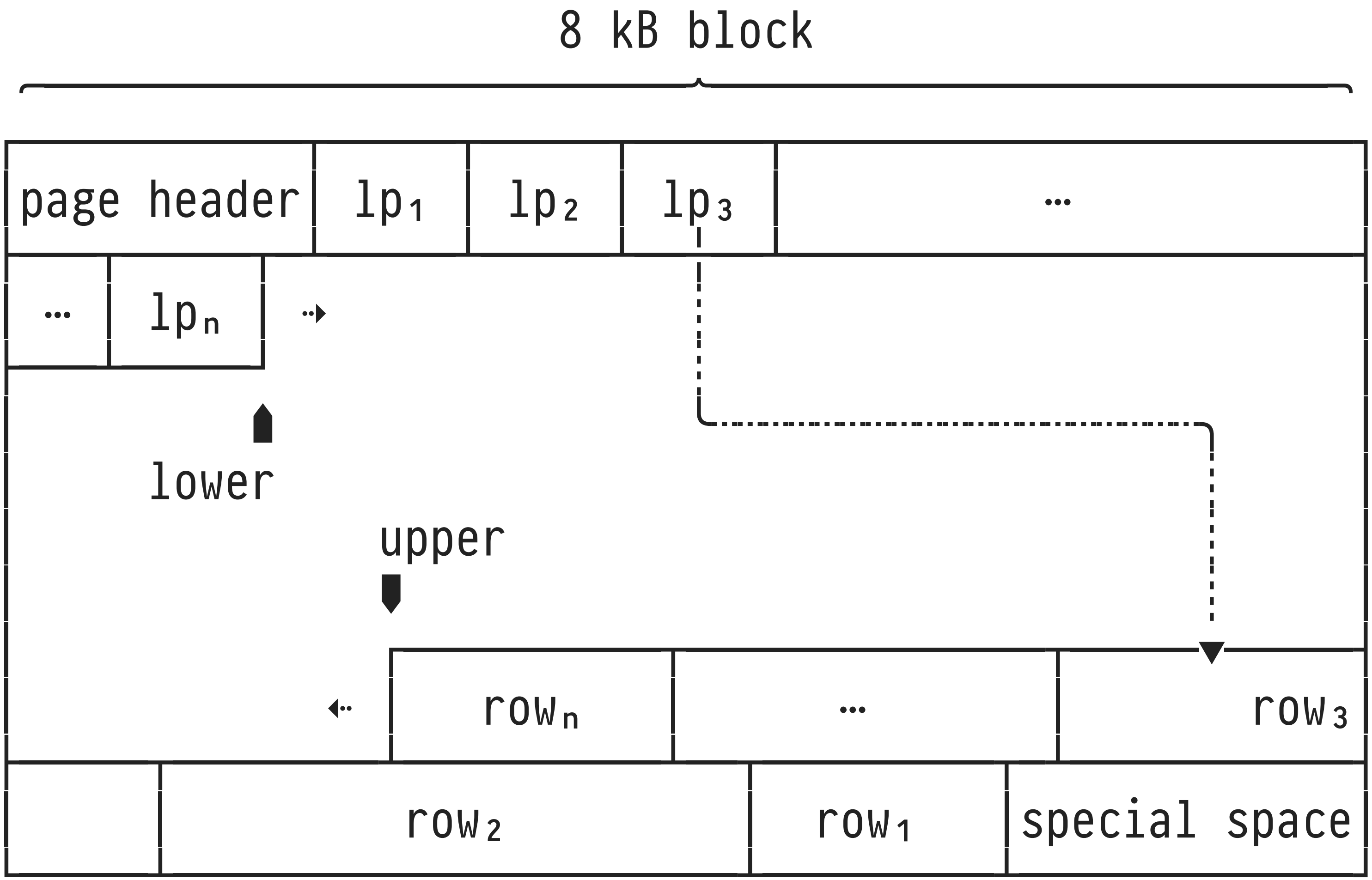
PostgreSQL: Row Storage

- 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

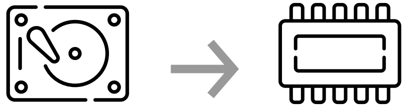
PostgreSQL comes with extension pageinspect that provides an "X-ray for heap file pages":

```
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›));
```

2 ¦ Q_2 — Querying a Wider Table



Recall SQL probe Q_2:

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

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

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

▥ column storage

a	b	c
❶	❹	❼
❷	❺	❽
❸	❻	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 equi-⋈ 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.