EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

DB2
Sommersemester 2022
Prof. Torsten Grust, Denis Hirn
WSI — Database Systems Research Group

Assignment 3 (13.05.2022)

Handin until: Friday, 20.05.2022, 09:00

1. [15 Points] **Decomposition and MonetDB**

   `decompositon.sql` contains the following table:

   t

a	b	c	d
6	'a'	3	'f'
7	'b'	4	'g'
8	'c'	3	'f'
9	'd'	4	'g'
0	'e'	3	'f'

   Load the table into MonetDB using:

   ```
   mclient -l sql <dbname> <path/to/decomposition.sql>
   ```

   (a) Given table `t(a,b,c,d)`, find any non-trivial non-key *Functional Dependencies (FDs)*[1].

   (b) Based on the FDs in (a), perform a BCNF split to create BCNF tables `t1` and `t2` from `t`.

   (c) Split table `t1` as well as `t2` into multiple *two-column* (binary) SQL tables whose head column carries an `id` and whose tail column carries the values of column `a` (or `b`, `c`, `d`).
   Hint: At this stage, you should end up with a total of five such two-column tables.

   (d) Formulate a MonetDB SQL query that shows that you can faithfully reconstruct `t` using all binary tables created in (c).

   (e) Based on the query in (d), explain briefly why MonetDB decides to *not* pursue normalization into BCNF form (but instead directly translates the original table `t` into binary tables).

[1] A refresher on FDs and BCNF can be found here: https://github.com/DBatUTuebingen/db1-ws2122/blob/master/slides/db1-10.pdf

2. [15 Points] Page Layout in *PostgreSQL*

   Use file `documents.sql` to create table `documents(title CHAR(4), doc TEXT)`. Rows in `documents` are — due to type `TEXT` of column `doc` — of variable length and may therefore grow and exceed the free page space on `UPDATE`. How does *PostgreSQL* handle that?

   Use the *PostgreSQL* extension `pageinspect`[2] to observe *PostgreSQL*'s behavior. **Hand in all queries you used** and describe your findings **briefly**. Proceed as follows:

   (a) Use function

   ```
   heap_page_item_attrs(get_raw_page('documents', ⟨page⟩), 'documents')
   ```

   to inspect the organization of rows on all pages of table `documents`. Next to each row's header information (`lp`: *slotno*, `lp_off`: row pointer, `lp_len`: *row size*, `t_ctid`: row version-chain pointer[3]), the function extracts the raw data of all attributes in an array `t_attrs` of type `bytea[]`.

   Use function `convert(str BYTEA)→TEXT` provided in `documents.sql` to convert this attribute's data to `TEXT`.

   (b) Determine the *RID* (*page*, *slotno*) and *row size* of the row containing `'doc1'` as well as the free space left on its page.

   (c) Perform an `UPDATE` on `'doc1'` doubling the size of its `doc` column, therefore exceeding the free page space. Does the *RID* still point to the same row? How are the pages and the physical location of the row data reorganized?

   (d) Perform an `UPDATE` on `'doc2'` growing its *row size* to more than 8 kB. The new row cannot fit into any page — even an empty one. How does *PostgreSQL* cope with that?

      i. How does the *row size* of the new row compare to the size of the inserted `doc`-value?

      ii. Read "*Out-of-line, on-disk TOAST storage*"[4] of the *PostgreSQL* documentation. Explain in your own words, how "*sliced bread*"[5] relates to *PostgreSQL* in terms of our current problem.

      iii. Search the system catalog table `pg_class`[6] to find the `relname` of the TOAST table associated with table `documents`.

      iv. Query table `pg_toast.`⟨*TOAST_relname*⟩ to determine, how many *chunks* (rows of the toast table) have been created to store your new `doc`-value.

[2] https://www.postgresql.org/docs/current/pageinspect.html
[3] See Slide 08 of Chapter 5 "Row Updates"
[4] https://www.postgresql.org/docs/current/storage-toast.html#STORAGE-TOAST-ONDISK
[5] https://en.wikipedia.org/wiki/Toast
[6] https://www.postgresql.org/docs/current/catalog-pg-class.html