Database Systems WS 2020/21
Prof. Dr.-Ing. Sebastian Michel
M.Sc. Angjela Davitkova / M.Sc. Nico Schäfer
Exercise 1: Handout 25.10.2021, Due 01.11.2021 16:00 CET
Lecture content: Videos up to #004

DBIS LAB TU KAISERSLAUTERN
https://dbis.cs.uni-kl.de

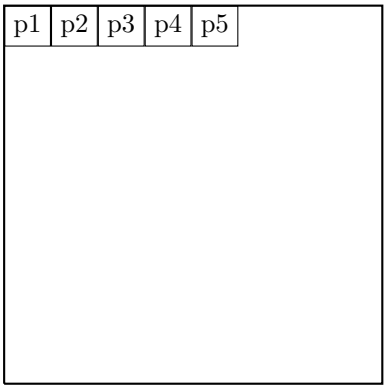
Question 1: Slotted Page (1 P.)

As discussed in the lecture, a page (aka. block) has a certain capacity in bytes, say 512, and is the smallest transfer unit of block storage devices (like a HDD). We look at each page as being simply a sequence of bytes, like it would be stored on disk. A slotted page is a page that stores a number of tuples of a relation. At the beginning of each slotted page, there are header information that tell where the tuples are located within the page. The page id together with the slot id is called TID, i.e., tuple identifier. To keep the implementation simple, in this exercise we assume that there are five slots. Each slot is a pointer that is pointing to the beginning of the tuple information. For the sake of this exercise, we only look at tuples of a specific type, namely, tuples from the TPC-H relation `Orders`.

Download from OLAT the `SlottedPageExercise.java` and the `orders_small.tbl` file. The Code already contains a simple class called FakeBlockStorage, which has methods to operate with pages in byte array form, simply storing and retrieving them, and if needed allocating new pages. As you can see, we have already implemented ways to create a byte array from an Order and vice versa. Your task is to implement the storing of a tuple, the retrieval of a tuple, and the calculation of unused bytes. In the Java code, we have marked with `//TODO` where you need to provide code. There is also a `main` method that reads |-separated lines from the given data file and stores them in the storage—some simple checks try to help you debugging the code. Additionally, we provide some utility methods to help you writing to the page's byte array, have a closer look, they should be very handy. As you can see there, each pointer consumes one `int`. The way to store the Orders tuples (in byte[] form of course) inside the page is up to you.

Submit the solution code file as .java and add the output of the execution to your solution PDF.

In the following illustration you see a depiction of a slotted page. There are the aforementioned five pointers to tuple offsets.



Question 2: Page Reference String (PRS) Analysis (1 P.)

a) Given the following PRS:

⟨ (T1,A) (T1,B) (T1,E) (T2,F) (T1,C) (T1,D) (T2,D) (T2,E) (T2,C) (T1,G) (T1,D) (T1,A) (T1,F) (T1,F) (T1,C) (T2,A) (T2,B) ⟩

Calculate the sequentiality and locality. Use the working-set-method for locality and calculate the average over both transactions. Assume a window size of $\tau = 3$.

b) Given the following stack depth distribution:

Stack depth	Frequency	Stack depth	Frequency
1	562	6	115
2	445	7	102
3	408	8	78
4	373	9	31
5	144	10	20

Answer the following questions:

i) How small can the buffer be, if we want no more than 8% of cache-misses?

ii) How many cache hits (in %) will there be, given a buffer size of 4?

iii) How many cache misses are avoided, if the buffer size is increased from 4 to 6?

Question 3: Replacement Strategies (1 P.)

Given the following sequence of page accesses:

D, B, A, C, B, B, C, A, D, D, C, B, C, A, C, B, A

a) Given a buffer size of 3, calculate the amount of cache misses for the strategies CLOCK, LRU-k with $k = 2$ and GCLOCK.
For GLOCK, assume an initial value $E_i = 2$ for every page and increment of $W_i = 1$ on reference.

b) Which replacements would happen for the given sequence if an optimal replacement strategy, which knows all future page accesses beforehand[1], was used?

c) Given a buffer size of 2: Find one example of a PRS for which FIFO has less cache **misses** than LRU, and one for which FIFO has less **hits** than LRU.

[1] Such a strategy (clairvoyant) can, of course, not be implemented, but it can be compared to other replacement strategies to benchmark their performance.

Question 4: Performance in PostgreSQL and pgAdmin (1 P.)

In this and the following exercise sheets we will use the relational database system *PostgreSQL*. PostgreSQL is available free of charge under an open source license, for example at `http://www.postgresql.org/` as installable packages or for Linux distributions via the common package managers (`apt-get, yum, pacman, ...`). Install PostgreSQL (version 10 or higher) in a way appropriate to your system.

pgAdmin is a graphical user interface for PostgreSQL, which simplifies its use and administration. Also install this program (`http://www.pgadmin.org/` or package manager).

Load the two data records linked in the `sql_dumps.txt` file in OLAT. Take the opportunity to refresh your SQL knowledge and make a few queries against the database. First, run `VACUUM ANALYZE`. This request has no result, however, it immediately creates statistics about the imported tables.

Indices and their effects

You can retrieve the query plan, which is created by the database by using "Explain" (German "Analysieren") in the query tool. The actual query results are not needed for this exercise. Use the TPC-H dataset for the following queries.

a) Which query plan is generated for the following query? Please describe the plan as detailed as possible. HINT: If you hover your mouse over an element in the query plan, a tool tip, with additional information, will be shown.

```
SELECT * FROM lineitem WHERE l_suppkey < 10
```

b) How will the plan change, if you create the following index?

```
CREATE INDEX index1 ON lineitem(l_suppkey);
```

c) By prefixing your query with `EXPLAIN (ANALYZE, BUFFERS)`, PostgreSQL will explain give additional information on DB buffer usage. How does the buffer usage change if you run the following query three times? Give the output row of `Buffers ...` each time.

```
EXPLAIN (ANALYZE, BUFFERS) SELECT * FROM orders WHERE o_totalprice < 100
```

d) PostgreSQL ships with an extension which allows "prewarming" a table, which means filling the buffer with relevant pages. Execute the following queries and again give the `Buffers ...` row. Explain what the output means and why there may still pages be read from the disk.

```
CREATE EXTENSION pg_prewarm;
SELECT pg_prewarm('orders');
EXPLAIN (ANALYZE, BUFFERS) SELECT * FROM orders WHERE o_totalprice < 100
```