CARNEGIE MELLON UNIVERSITY
COMPUTER SCIENCE DEPARTMENT
15-445/645 – DATABASE SYSTEMS (FALL 2024)
PROF. ANDY PAVLO

Homework #2 (by William and Prashanth) – Solutions

Due: **Sunday September 22, 2024 @ 11:59pm**

IMPORTANT:

- Enter all of your answers into **Gradescope by 11:59pm on Sunday September 22, 2024.**
- **Plagiarism**: Homework may be discussed with other students, but all homework is to be completed **individually**.

For your information:

- Graded out of **100** points; **3** questions total
- Rough time estimate: ≈4-6 hours (1-1.5 hours for each question)

Revision : 2024/09/28 15:29

Question	Points	Score
Slotted Pages and Log-Structured	30	
Storage Models	35	
Database Compression	35	
Total:	100	

Question 1: Slotted Pages and Log-Structured . . . . [30 points]

Graded by:

(a) **[10 points]** Which problems are associated with the *slotted-page storage* in a database system? Select all that apply.

- ■ **Fragmentation**
- □ Write Amplification
- ■ **Increased Random Writes**
- ■ **Increased Random Reads**
- □ None of the above

Solution: The Slotted-Page Design often leads to fragmentation, as deletion of tuples can leave gaps in the pages, making them not fully utilized.

Since tuples can be stored across separate pages, it may increase the amount of random I/O that the DBMS has to incur when both reading data and when writing out dirty pages.

(b) **[10 points]** Which problems are associated with the *log-structured storage* in a database system? Select all that apply.

- □ Fragmentation
- ■ **Write Amplification**
- □ Increased Random Writes
- □ Increased Random Reads
- □ None of the above

Solution: Log-structure storage is particularly beneficial for write-intensive workloads, such as append-only data. But it incurs write amplification due to compaction.

Although a log-structured DBMS may have to read multiple pages to find a tuple, these will be sequential I/Os and <u>not</u> random reads.

(c) **[10 points]** You are asked to compare *log-structured storage* to *slotted-page storage* for a new system. Ignore any indexes and overhead from metadata. Select all true statements.

- □ Log-structured storage requires less disk space.
- □ Only log-structured storage supports variable length tuples.
- ■ **For an append-only workload, both achieve comparable performance.**
- □ After lots of insert/update/deletes, only log-structured benefits from maintenance.
- □ Log-structured storage is not suitable for systems with limited memory.
- □ None of the above are true.

Solution: In absence of indexes + metadata, then slotted-page for append-only workload becomes the log-structured storage architecture. Hence, this is the only correct statement.

After lots of inserts/updates/deletes, slotted-page may also benefit from maintenance to reclaim any empty space or compact partially empty pages.

Question 2: Storage Models . . . . [35 points]

Graded by:

Consider a database with a single table `E(event_id,event_name,host_id,duration number_of_events)`, where `event_id` is the *primary key*, and all attributes are the same fixed width. Suppose `E` has 10,000 tuples that fit into 100 pages. You should ignore any additional storage overhead for the table (e.g., page headers, tuple headers). Additionally, you should make the following assumptions:

- The DBMS does *not* have any additional meta-data.
- `E` does *not* have any indexes (including for primary key `event_id`).
- None of `E`'s pages are already in memory. The DBMS can store an infinite number of pages in memory.
- Content-wise, the tuples of `E` will always make each query run the longest possible and do the most page accesses.
- The tuples of `E` can be in any order (keep this in mind when computing *minimum* versus *maximum* number of pages that the DBMS will potentially have to read and think of all possible orderings)

(a) Consider the following query:

```
SELECT MAX(number_of_events) FROM E
    WHERE duration > 15445 AND host_id == 15645 ;
```

i. **[5 points]** Suppose the DBMS uses the decomposition storage model (DSM) with implicit offsets. How many pages will the DBMS potentially have to read from disk to answer this query?
Be sure to keep in mind the assumption about the contents of `E`.
☐ 1-40 ■ **41-60** ☐ 61-80 ☐ 81-100 ☐ $\geq$ 101 ☐ Not possible to determine

Solution: 60 pages. There are 20 pages per attribute. 20 pages to find `duration` and another 20 to find `host_id` for all tuples. In the worst-case scenario for `E`'s content, `number_of_events` for all tuples must be accessed as well. Hence, another 20 pages must be read.

ii. **[5 points]** Suppose the DBMS uses the N-ary storage model (NSM). How many pages will the DBMS potentially have to read from disk to answer this query?
Be sure to keep in mind the assumption about the contents of `E`.
☐ 1-40 ☐ 41-60 ☐ 61-80 ■ **81-100** ☐ $\geq$ 101 ☐ Not possible to determine

Solution: 100 pages. To find `duration` and `host_id` for all tuples, all pages must be accessed.

(b) Now consider the following query:

```
SELECT event_name, host_id FROM E
   WHERE event_id = 15445 OR event_id = 15645
```

i. Suppose the DBMS uses the decomposition storage model (DSM) with implicit offsets.

α) **[5 points]** What is the *minimum* number of pages that the DBMS will potentially have to read from disk to answer this query?
■ **1-3** □ 4-6 □ 7-9 □ 10-100 □ $\geq$ 101 □ Not possible to determine

Solution: 3 pages. Suppose all two primary keys appear on the first page. Since all attributes are of the same fixed width, each attribute of `event_id=15445` and `event_id=15645` will also appear on the same page. We'll thus need to read 1 page to find the two primary keys and read 2 pages to access `event_name`, `host_id` at their corresponding offsets.

β) **[5 points]** What is the *maximum* number of pages that the DBMS will potentially have to read from disk to answer this query?
□ 1-20 ■ **21-40** □ 41-60 □ 61-80 □ 81-100 □ $\geq$ 101 □ Not possible to determine

Solution: 24 pages. There are 20 pages per attribute. In the worst case, we scan through all 20 pages to find the two primary keys. In the worst case, the two primary keys will be located on different pages. Since all attributes are of the same fixed width, each attribute of `event_id=15445` and `event_id=15645` will also appear on different pages. Hence we must read 2 pages to access each attribute at their corresponding offsets. Thus, we read 4 pages in total to access `event_name`, `host_id`.

ii. Suppose the DBMS uses the N-ary storage model (NSM).

α) **[5 points]** What is the *minimum* number of pages that the DBMS will potentially have to read from disk to answer this query?
■ **1** □ 2-3 □ 4-6 □ 7-9 □ 10-100 □ $\geq$ 101 □ Not possible to determine

Solution: We find the tuples of all two primary keys on the first page. No need to look in other pages since all attributes are stored together.

β) **[5 points]** What is the *maximum* number of pages that the DBMS will potentially have to read from disk to answer this query?
□ 1 □ 2-3 □ 4-6 □ 7-9 ■ **10-100** □ $\geq$ 101 □ Not possible to determine

Solution: 100 pages. At least one tuple with matching primary key is located on the last page. We must thus scan through every page.

(c) Finally consider the following query:

```
SELECT event_id FROM E
    WHERE duration = (SELECT MIN(duration) FROM E);
```

Suppose the DBMS uses the decomposition storage model (DSM) with implicit offsets.

i. **[5 points]** What is the *minimum* number of pages that the DBMS will potentially have to **read from disk** to answer this query?
☐ 1-20 ■ **21-40** ☐ 41-60 ☐ 61-80 ☐ 81-100 ☐ $\geq$ 101 ☐ Not possible to determine

Solution: 40 pages. 20 pages for the inner select, and 20 pages to get the `event_id` since the buffer pool will have the `duration` pages from the inner select. Remember content-wise the tuples make the queries always run for the longest time, you can only consider different orderings of the tuples.

Question 3: Database Compression............................[35 points]

Graded by:

(a) **[5 points]** Suppose that the DBMS has a `VARCHAR` column storing the following values:

```
[Museum of Art, Andy Warhol Museum, Museum of Natural
History, Children's Museum, Solders & Sailors]
```

Which of the following are valid encodings (`uint32`) for this column under dictionary compression as discussed in lecture that will support both point queries and range queries? Select **all** the valid encodings.

■ `[3,1,4,2,5]`
☐ `[10,20,30,40,50]`
☐ `[31,15,92,32,196]`
■ `[32,15,92,31,196]`
■ `[30,10,40,20,50]`

Solution: To support range queries, the DBMS must use an order-preserving encoding scheme. The values of the dictionary codes do not matter as long as they preserve the same ordering of the original data.

(b) **[15 points]** Suppose the DBMS wants to compresses a table `R(a)` using columnar compression. Which of the following compression schemes **will not benefit** from sorting the table before compressing column `a`? Select **all** that apply.
Hint: "Benefit" means that the efficacy of the compression scheme improves on sorted data. You should not make any assumptions about the column type or its distribution of values.

☐ Run-length Encoding
■ **Bit-packing Encoding**
■ **Mostly Encoding**
■ **Bitmap Encoding**
☐ Delta Encoding
■ **Dictionary Encoding**
☐ All of the above will not benefit.

Solution: Sorting only benefits Run-length encoding and Delta encoding for the below reasons. All other encodings do not benefit from sorting the table first.

- **Run-length Encoding:** Sorting improves the potential compression ratio for RLE because there could potentially be more consecutive values in a column.
- **Delta Encoding:** For numeric data types with a small range of values, the difference between consecutive values in the column after sorting could be smaller than the original value. Therefore, the compression ratio could improve.

Question 3 continues...

(c) **[15 points]** A colleague approaches with a list of true and false statements about run-length encoding, delta encoding, bitmap encoding, and dictionary encoding. The colleague wants your assistance in identifying the true statements. Select **all** that apply.

☐ Run-length Encoding is effective for compressing any integer column.

■ **Bitmap Encoding on high cardinality columns hurts inserts and updates.**

☐ Delta Encoding is good at compressing large text values.

■ **For *point lookup-only* workload, order-preserving dictionary encoding is unnecessary.**

■ **For a heavy update workload, dictionary performs better than delta encoding.**

☐ None of the above.

Solution:

- Run-length Encoding is effective for compressing any integer column: **F**. Run-length encoding is not effective for high cardinality values that vary significantly (e.g., primary key, stock ticker data).
- Bitmap Encoding on high cardinality columns hurts inserts and updates: **T**. On high cardinality columns with bitmap encoding, each insert/update would need to edit all the bitmaps which would generate extra writes.
- Delta Encoding is good at compressing large text values: **F**. Large text values in theory have large differences; hence delta encoding may not be an effective choice.
- For *point lookup-only* workload, order-preserving dictionary encoding is unnecessary: **T**. Since we are only looking up an exact value, we only require the dictionary's hash properties.
- For a heavy update workload, dictionary performs better than delta encoding: **T**. In the worst case, order-preserving dictionary encoding and delta encoding both require re-encoding the entire column. *However*, for non order-preserving dictionary encodings, you don't have to re-encode.

  In the case where the new value is different, delta encoding will have to re-encode, whereas order-preserving dictionary may not (i.e., dictionary already contains the target value or can assign an encoding without re-computing).