Name

UIN

Quiz
2

Due November 24th, 2025

CS581 - Database Management Systems

Please leave this empty! 1 2 3 4 5 6 7

Sum

Instructions

- Multiple choice questions are graded in the following way: You get points for correct answers and points subtracted for wrong answers. The minimum points for each questions is **0**. For example, assume there is a multiple choice question with 6 answers - each may be correct or incorrect - and each answer gives 1 point. If you answer 3 questions correct and 3 incorrect you get 0 points. If you answer 4 questions correct and 2 incorrect you get 2 points. . . .
- For your convenience the number of points for each part and questions are shown in parenthesis.
- There are several parts in this homework:
  1. Disk Organization and Buffering
  2. Index Structures
  3. Result Size Estimations
  4. I/O Cost Estimation
  5. Schedules
  6. ARIES (Optional)
  7. Physical Optimization (Optional)

Part 1 Disk Organization and Buffer Management (Total: 14 Points)

Question 1.1 Page Replacement Clock (14 Points)

Consider a buffer pool with 4 pages using the **Clock** page replacement strategy. Initially the buffer pool is in the state shown below. We use the following notation ${}^{flag}[page]^{dirty}_{fix}$ to denote the state of each buffer frame. *page* is the number of the page in the frame, fix is its fix count, *dirty* is indicating with an Asterix that the page is dirty, and $flag$ is the reference bit used by the Clock algorithm. E.g., ${}^{1}[5]^{*}_{2}$ denotes that the frame stores page 5 with a fix count 2, that the page is dirty, and that the reference bit is set to 1. Recall that Clock uses a pointer S that points to the current page frame (the one to be checked for replacement next). The page frame S is pointing to is indicated by by $\downarrow$.

The solution should be provided in the file `q1_1_clock.txt`. In this file you need to write down the buffer pool state after each operation. In this file comments start with `--`. A buffer pool state starts with the keyword `POOL:` followed by the states of each page frame. The clock points is written as a prefix for the page frame it is pointing too as `->`. A frame's state ${}^{1}[17]^{*}_{0}$ is written as `[ 17, fix: 0, dirty: 1, clockflag: 1 ]`. For example, consider the example buffer state shown below:

Example Buffer State

$$\overset{\downarrow}{{}^{1}[17]_{1}} \quad , {}^{1}[1]_{0} \quad , {}^{1}[12]^{*}_{1} \quad , {}^{0}[16]^{*}_{0}$$

Corresponding Textual Representation

```
POOL:
->  [ 17, fix: 1, dirty: 0, clockflag: 1 ],
    [ 1, fix: 0, dirty: 0, clockflag: 1 ],
    [ 12, fix: 1, dirty: 1, clockflag: 1 ],
    [ 16, fix: 0, dirty: 1, clockflag: 0 ]
```

Current Buffer State

$$\overset{\downarrow}{{}^{0}[11]_{2}} \quad , {}^{1}[13]_{1} \quad , {}^{0}[16]^{*}_{0} \quad , {}^{1}[14]_{1}$$

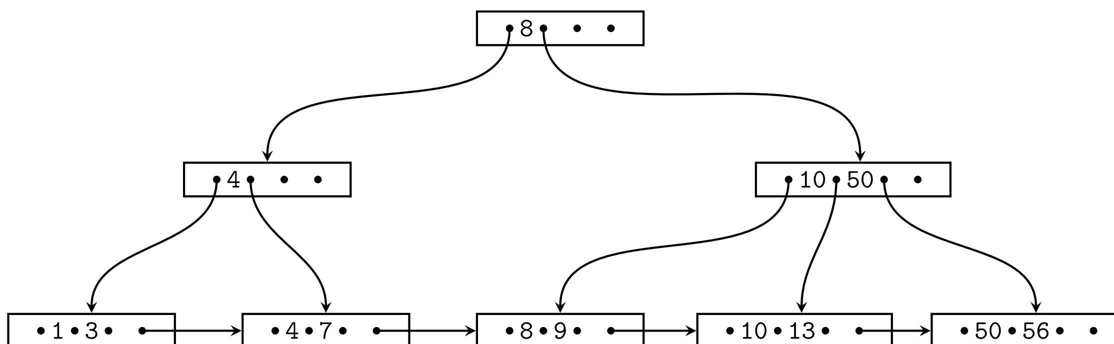
Execute the following requests and write down state of the buffer pool after each request.

- p stands for pin
- u for unpin
- d for marking a page as dirty

`pin(12),unpin(12),unpin(13),pin(18),unpin(11),unpin(11),pin(8)`

Part 2 B-Trees Structures (Total: 18 Points)

This part is autograded. You have to provide the solutions for the two questions as files `q2_1_btree_inserts.txt` and `q2_2_btree_operations.txt`. In these files comments are lines that start with `--`. A btree starts with the keyword `BTREE`. Each node needs to be written on a separate line and consists of comma-separated list of keys. Key slots that are empty have to be written as `NULL`. The depth of a node in the tree is indicated by prefixing the node with a number of tabs (`\t`) equal to its depth. **Note that the depth of a node is solely determined by the number of tab characters that precede it, spaces are ignored!** The children of a node immediately follow that node. As an example, consider the B+-tree shown below.




```
BTREE
[8,NULL,NULL]
	[4,NULL,NULL]
		[1,3,NULL]
		[4,7,NULL]
	[10,50,NULL]
		[8,9,NULL]
		[10,13,NULL]
		[50,56,NULL]
```

Question 2.1 Construction (9 Points)

Assume that you have the following table:

Student

name	points	credits
Andrea Huggler	24	7.9
Kitty Hammoud	29	1.2
Aydan Blomstedt	3	5.8
Lola Platel	9	1.1
Lyle Dement	20	8.6
Tod Bangma	6	2.6
Scarlett Menconi	14	9.1
Merrill Norland	8	5.8
Floyd Mitera	16	0.8

Create a B+-tree for table *Student* over attribute *points* with $n = 2$ (up to two keys per node). You should start with an empty B+-tree and insert the keys in the order shown in the table above. Write down the resulting B+-tree after each step.

When splitting or merging nodes follow these conventions:

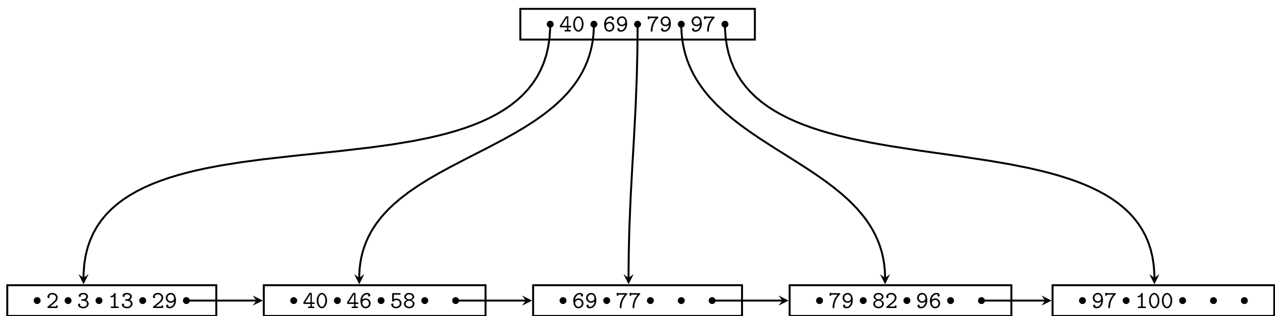
- **Leaf Split**: In case a leaf node needs to be split during insertion and n is even, the left node should get the extra key. E.g, if $n = 2$ and we insert a key 4 into a node [1,5], then the resulting nodes should be [1,4] and [5]. For odd values of n we can always evenly split the keys between the two nodes. In both cases the value inserted into the parent is the smallest value of the right node.
- **Non-Leaf Split**: In case a non-leaf node needs to be split and n is odd, we cannot split the node evenly (one of the new nodes will have one more key). In this case the "middle" value inserted into the parent should be taken from the right node. E.g., if $n = 3$ and we have to split a non-leaf node [1,3,4,5], the resulting nodes would be [1,3] and [5]. The value inserted into the parent would be 4.
- **Node Underflow**: In case of a node underflow you should first try to redistribute values from a sibling and only if this fails merge the node with one of its siblings. Both approaches should prefer the left sibling. E.g., if we can borrow values from both the left and right sibling, you should borrow from the left one.

Question 2.2 Operations (9 Points)

Given is the B+-tree shown below ($n = 4$). Execute the following operations and write down the resulting B+-tree after each operation:

insert(15),delete(79),delete(29),insert(72),insert(37),delete(15),delete(58),insert(93),delete(82)

Use the conventions for splitting and merging introduced in the previous question.



Part 3 Extensible Hashing (Total: 18 Points)

Question 3.1 Extensible Hashing (18 Points)

Solutions for this question have to be written in the provided text file `extensible_hashing.txt` and uploaded to `diderot`. The format of the file is a list of extensible hash tables (the state of the hash table after each of the operations. A hash table starts with the key word `EHASH` on a separate line. Buckets are written on separate lines starting with a list of directory entries that the buckets belongs to, followed by the depth of the bucket (e.g., `depth:1`), and finally the list of entries (hash values and keys) stored in the bucket (e.g., `[0000: 3, 0001: 5]`). Comments start with `--`. For example, the code below defines a hash table with directory depth of 1 (one bit) and two buckets, one with key 6 and 5 (hash `0101` and `0100`) and a second one with key 4 (hash `1100`). Note that for overflow buckets you list all overflow buckets for a bucket on the same line.

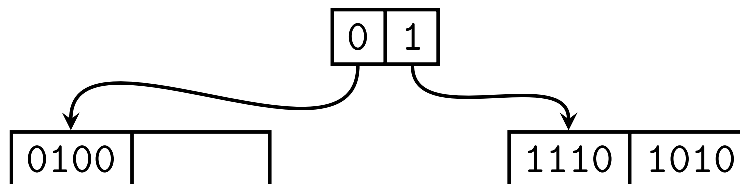
```
EHASH
[0] depth: 1 [0101:6, 0100: 5]
[1] depth: 1 [1100:4]
```

Consider the extensible Hash index shown below that is the result of inserting values 0,6,11. Each page holds two keys. Execute the following operations

`insert(5),insert(7),insert(9),insert(12),insert(4),insert(8),insert(2)`

and write down the resulting index after each operation. Assume the hash function is defined as:

x	h(x)
0	1110
1	1100
2	1100
3	0010
4	0000
5	1001
6	0100
7	0001
8	0011
9	1011
10	1001
11	1010
12	0001



Part 4 Result Size Estimations (Total: 20 Points)

Consider the city connection database shown below and the following statistics:

$$T(city) = 200 \qquad T(road) = 4000 \qquad T(train) = 3000$$
$$V(city, name) = 200 \qquad V(road, from) = 200 \qquad V(train, from) = 200$$
$$V(city, state) = 50 \qquad V(road, to) = 200 \qquad V(train, to) = 200$$
$$V(city, population) = 200 \qquad V(road, distance) = 2000 \qquad V(train, distance) = 600$$

The min and max values for some of the columns are:

$$min(city, population) = 100,000 \qquad max(city, population) = 3,000,000$$
$$min(road, distance) = 1 \qquad max(road, distance) = 530$$
$$min(train, distance) = 1 \qquad max(train, distance) = 1200$$

Question 4.1 Estimate Result Size (4 Points)

Estimate the number of result tuples for the query $q = \sigma_{state=IL \wedge population=200000}(city)$ using the first assumption presented in class (values used in queries are uniformly distributed within the active domain).

Question 4.2 Estimate Result Size (5 Points)

Estimate the number of result tuples for the query $q = \sigma_{from=Chicago \vee distance<100}(road)$ using the first assumption presented in class.

Question 4.3 Estimate Result Size (5 Points)

Estimate the number of result tuples for the query $q = \sigma_{(from=Chicago \vee (to=Chicago \wedge distance>800)}(train)$ using the first assumption presented in class.

Question 4.4 Estimate Result Size (6 Points)

Estimate the number of result tuples for the query

$q = \pi_{name,sto}(city \bowtie_{name=from} road \bowtie_{to=tname} \rho_{tcity,tstate,tpopulation}(city) \bowtie_{tname=sfrom} \rho_{sfrom,sto,sdistance}(road))$

using the first assumption presented in class.

Part 5 I/O Cost Estimation (Total: 20 Points)

Question 5.1 External Sorting (4 Points)

You have $M = 65$ memory pages available and should sort a relation R with $B(R) = 3,500,000,000$ blocks. Compute the number of I/Os necessary to sort R using the external merge sort algorithm introduced in class.

Question 5.2 External Sorting (4 Points)

You have $M = 5$ memory pages available and should sort a relation R with $B(R) = 4,500$ blocks. Compute the number of I/Os necessary to sort R using the external merge sort algorithm introduced in class.

Question 5.3 I/O Cost Estimation (6 = 2+2+2 Points)

Consider two relations R and S with $B(R) = 30,000$ and $B(S) = 1,200,000$. You have $M = 101$ memory pages available. Estimate the minimum number of I/O operations needed to join these two relations using **block-nested-loop join**, **merge-join** (the inputs are not sorted), and **hash-join**. You can assume that the hash function evenly distributes keys across buckets. Justify you result by showing the I/O cost estimation for each join method.

Question 5.4 I/O Cost Estimation (6 = 2+2+2 Points)

Consider two relations R and S with $B(R) = 150,000$ and $B(S) = 200,000$. You have $M = 65$ memory pages available. Compute the minimum number of I/O operations needed to join these two relations using **block-nested-loop join**, **merge-join** (the inputs are not sorted), and **hash-join**. You can assume that the hash function evenly distributes keys across buckets. Justify you result by showing the I/O cost estimation for each join method.

Part 6 Schedules (Total: 20 Points)

Multiple choice questions are autograded. The solutions for this question should be submitted in a file `q6_schedules.txt`. For your convenience the questions are already in the file, Just mark correct answers using `[X]` and keep wrong answers empty (`[ ]`).

```
[X] this is a yes answer
[ ] this is a wrong answer
```

Question 6.1 Schedule Classes (20 Points)

Indicate which of the following schedules belong to which class. Recall transaction operations are modeled as follows:

$w_1(A)$	transaction 1 wrote item A
$r_1(A)$	transaction 1 read item A
c_1	transaction 1 commits
a_1	transaction 1 aborts

$S_1 = w_1(C), r_4(D), w_4(B), w_4(D), r_4(A), w_2(A), w_4(E), r_1(B), w_2(A), w_4(E), r_1(B), w_1(A), w_2(B), c_2, w_3(D), c_3, c_4, c_1$
$S_2 = r_1(B), r_3(C), w_3(A), w_2(B), r_1(A), c_1, c_3, r_4(B), r_4(A), w_4(C), c_4, c_2$
$S_3 = w_4(B), w_3(D), r_2(C), w_1(A), w_2(A), c_2, r_4(D), c_4, w_3(C), c_3, w_1(B), c_1$
$S_4 = w_3(C), w_3(D), w_4(A), w_4(B), w_1(A), w_2(C), w_4(E), w_1(B), w_2(D), w_1(E), c_1, c_2, c_3, c_4$

❑ S_1 is recoverable

❑ S_1 is cascade-less

❑ S_1 is strict

❑ S_1 is conflict-serializable

❑ S_1 is 2PL

❑ S_2 is recoverable

❑ S_2 is cascade-less

❑ S_2 is strict

❑ S_2 is conflict-serializable

❑ S_2 is 2PL

❑ S_3 is recoverable

❑ S_3 is cascade-less

❑ S_3 is strict

❑ S_3 is conflict-serializable

❑ S_3 is 2PL

- ❏ S_4 is recoverable
- ❏ S_4 is cascade-less
- ❏ S_4 is strict
- ❏ S_4 is conflict-serializable
- ❏ S_4 is 2PL

l

Part 7 Optional: ARIES (Total: 10 Optional Points)

Question 7.1 Recovery (10 Points)

Consider the state of the log and pages on disk shown below. For simplicity we do not show the actual undo/redo actions for updates, but instead show only the affected page. Assume a crash occurred after the last log entry. Answer the following questions:

1. **Analysis**: Write down the result of the analysis phase (RedoLSN, Transaction Table, Dirty Page Table)
2. **Redo**: Which pages will be loaded from disk during redo? Which pages will be modified during redo?
3. **Undo**: Write down the additional log entries that will be written during undo.

The solution should be provided in a file q7_aries. In this file comments start with --. The results of the three phases are prefixed by ANALYSIS, REDO, and NEWLOG. An example is shown below.

ANALYSIS: For the results of the analysis phase, write down the redo LSN (RedoLSN: X). The transaction table is given as a comma-separated list of entries of the form <Tx, STATE, PrevLSN, UndoNxtLSN> (missing entries are represented as empty strings). The dirty page table is written as a comma-separated list of entries of the form <page, RecLSN>.

REDO: For the redo phase provide the pages to be loaded and the pages that will get modified as separate lists of the form [NUMBER, ...].

NEWLOG: The new log entries are a list of entries (space separated) of the form [LSN: lsn, Type: recordtype, TID: transac
As above missing entries should be written using whitespace.

As an example consider the following analysis result and the corresponding file content:

(Analysis):
RedoLSN: 2
Transaction Table: $< T_2, u, 6, - >, < T_3, u, 8, - >, < T_4, u, 12, - >$
Dirty Page Table: $< 1, 2 >, < 3, 5 >$

(Redo):
Pages 1 and 3 have to be loaded from disk.
Only page 1 will be modified based on redo info from log entries 2, 3, 6, 7, and 12.

(New log entries):
Transactions T_2, T_3, and T_4 will be rolled back. The CLRs written during undoing these transactions' updates is shown below.

LSN	Type	TID	PrevLSN	UndoNxtLSN	Data
13	CLR	4	12	-	Page 1
14	CLR	2	6	-	Page 1

```
-- ********************************************************************************
-- ANALYSIS RESULT
-- ********************************************************************************

ANALYSIS
RedoLSN: 2
TransactionTable: <T2, u, 6, > <T3, u, 8, > <T4, u, 12, >
DirtyPageTable: <1,2> <3,5>

-- ********************************************************************************
-- REDO
-- ********************************************************************************

REDO
load pages: [ 1, 3 ]
modified pages: [ 2, 3, 6, 7, 12 ]

-- ********************************************************************************
-- NEW LOG ENTRIES GENERATED DURING RECOVERY
-- ********************************************************************************

NEWLOG
[LSN: 13, Type: CLR, TID: 4, PrevLSN: 12, UndoNxtLSN: , DataPage: 1 ]
[LSN: 14, Type: CLR, TID: 2, PrevLSN: 6, UndoNxtLSN: , DataPage: 1 ]
```

Log

LSN	Type	TID	PrevLSN	UndoNxtLSN	Data
1	begin	1	-	-	-
2	update	1	1	-	Page 1
3	update	1	2	-	Page 1
4	begin	2	-	-	-
5	update	3	-	-	Page 3
6	update	2	4	-	Page 1
7	update	3	5	-	Page 1
8	begin	4	-	-	-
9	begin_cp	-	-	-	-
10	commit	3	7	-	-
11	update	4	8	-	Page 4
12	update	4	11	-	Page 1

Disk

PageID	PageLSN
1	3
2	0
3	0
4	11
5	0

Part 8 Bonus: Physical Optimization (Total: 10 Bonus Points)

Consider the following relations $R(A, B)$, $S(C, D)$, $T(E, F)$ with $S = \frac{1}{10}$ (10 tuples fit on each page). The sizes and value distributions are:

$$T(R) = 10,000 \quad V(R, A) = 10,000 \quad V(R, B) = 50$$
$$T(S) = 50,000 \quad V(S, C) = 100 \quad V(S, D) = 25,000$$
$$T(T) = 30 \quad V(T, E) = 10 \quad V(T, F) = 3$$

Question 8.1 Greedy Join Enumeration (10 Points)

Use the greedy join enumeration algorithm to find the cheapest plan for the join $R \bowtie_{B=C} S \bowtie_{D=E} T$. Assume that **nested-loop** (not the block based version) is the only available join implementation with the left input being the "outer" (for each tuple from the outer we have to scan the whole inner relation). Furthermore, there are no indicies defined on any of the relations (that is you have to use **sequential scan** for each of the relations). As a cost model consider the **total number of I/O operations**. For example, if you join two relations with $5,000$ and $10,000$ tuples with $S = \frac{1}{10}$, where the $5,000$ tuple relation is the outer, then the cost would be $5,000,000$ (scan the inner 5000 times) + 500 to scan the other once. The total cost is then $5,000,500$ I/Os. Assume that the system supports pipelining for the outer input of a join. That is if you join the result of a join with a relation where the join result is the outer, then there is no I/O cost for scaning the outer. Also under these assumptions you never have to store join results to disk. Hint: You will have to estimate the size of intermediate results. Use the estimation based on the number of values and not the one based on the size of the domain. Use the assumption that the number of values in a join attribute of a join result is the minimum of the number of values in the join attribute of each input.

Write down the state (the current plans) after each iteration of the algorithm in file `q8_joinenum.txt`. In this file comments start with `--`. The state for an iteration starts with the keyword `ITERATION`. A plan consists of three parts: an algebra tree (separated by `:` from the next part), the expected number of result tuples (separated by `;` from the next part) and the expected total cost (I/Os) for the plan. An algebra tree (expression) uses the key word `JOIN` to indicate joins and uses parentheses to indicate the order of joins, e.g., `R JOIN (S JOIN T)` represents the plan which joins `R` with the result of joining `S` with `T`. An example result is shown below.

```
-----------------------------------------------------
-- FIRST ITERATION

ITERATION

R: 300; 30
S: 100; 10
T: 100; 10

-----------------------------------------------------
-- SECOND ITERATION

ITERATION

(R JOIN S): 3000; 300
T: 100; 10

-----------------------------------------------------
-- THIRD ITERATION

ITERATION
((R JOIN S) JOIN T): 30; 100543
```