Name	CWID

Exam 1

March 13th, 2013

CS525 - Midterm Exam Solutions

Please leave this empty!	1		2		3		4		5		Sum	

Instructions

- Things that you are **not** allowed to use
  - Personal notes
  - Textbook
  - Printed lecture notes
- The quiz is **90** minutes long
- 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 5 parts in this quiz
  1. Disk Organization
  2. SQL
  3. Relational Algebra
  4. Index Structures
  5. Operator Implementations

Part 1 Disk Organization (Total: 14 Points)

Question 1.1 Record Layout and Identification (4 Points)

Check all of the following statements that are true.

- ❑ Given a TID of a record, the record can be loaded from disk with up to 3 disk assesses.
- ■ Using the TID approach, the identifier of a record does never change in its lifetime.
- ■ The space occupied by a tombstone is never reused.
- ❑ A fixed length representation requires the same amount of space as a variable length representation to represent a given record.
- ■ Without using a null-map we would have to reserve one value of each datatype to represent `NULL`.
- ❑ Using variable length records and pages with a header at the beginning of a page, it is a good idea to store records starting at the beginning page. E.g., the first record on a page will be placed right after the page header.

Question 1.2 Disk Access (3 Points)

Consider a disk with the following properties:

- Blocks size: 1KB
- Avg. seek time: 25ms
- RPM: 3000
- Transfer rate: 10 MB/sec

Compute the time to read 1000 blocks using random I/O (the 1000 blocks are randomly distributed over the disk) and sequential I/O (the 1000 blocks are in sequence). After locating a block on disk you can assume that sequential access is operating at transfer rate.

Solution

Avg. rotational delay: 50 revolutions per sec - so $20ms$ per revolution which means $10ms$.
Random I/O: $1000 * (25 + 10 + 0.1)ms = 1000 * 35.1ms = 35.1sec$
Sequential I/O: $25 + 10 + 1000 * (0.1)ms = 135ms$

Question 1.3 Page Replacement LRU (7 Points)

Consider a buffer pool with 6 pages using the LRU page replacement strategy. Initially the buffer pool is in the state shown below. We use the following notation $[page]_{fix}^{dirty}$ to denote the state of each buffer frame. *page* is the number of the page in the frame, fix is its fix count, and *dirty* is indicating with an asterix if the page is dirty. E.g., $[5]_2^*$ denotes that the frame stores page 5 with a fix count 2 and that the page is dirty. The current state was the result of executing the sequence of operations shown below. Here p stands for pin, u for unpin, and d for marking a page as dirty.

```
p(3),p(2),p(5),u(3),u(2),p(7),d(7),u(7),p(10),p(11),p(5),u(5)
```

$$[11]_1[2]_0[5]_1[7]_0^*[10]_1$$

Write down the state of the buffer pool after executing the following requests.

```
u(5),p(13),p(14),p(15),u(10),u(15),p(20)
```

Solution

$$[11]_1[13]_1[15]_0[14]_1[20]_1$$

Part 2 SQL (Total: 22 Points)

Consider the following database schema and instance:

movie

title	length	year
Citizen Kane	119	1941
Batman Begins	140	2005
American Psycho	102	2000

person

name
Orson Welles
Christopher Nolan
Agnes Moorehead
Christian Bale
Mary Harron

directedBy

person	movie
Orson Welles	Citizen Kane
Christopher Nolan	Batman Begins
Mary Harron	American Psycho

played

playedBy	character	movie
Agnes Moorehead	Mary Kane	Citizen Kane
Orson Welles	Kane	Citizen Kane
Christian Bale	Batman	Batman Begins
Christian Bale	Patrick Bateman	American Psycho

Hints:

- When writing queries do only take the schema into account and **not** the example data given here. That is you queries should return correct results for all potential instances of this schema.
- Underlined attribute form the primary key of an relation.
- The attribute *person* and *movie* of relation *directedBy* are foreign keys to the attributes *name* in relation *person* and *title* in relation *movie* respectively.
- The attribute *playedBy* and *movie* of relation *played* are foreign key to the attributes *name* in relation *person* and *title* in relation *movie* respectively.

Question 2.1 (6 Points)

Write an SQL statement that returns persons (the name) that are both directors and actors, i.e., have directed at least one movie and have played a character in at least one movie.

Solution

```
SELECT DISTINCT name
FROM   person p, directedBy d, played l
WHERE  p.name = d.person AND p.name = l.playedBy
```

The access to relation person can actually be avoided here by executing the join on `person = playedBy`. Alternatively:

```
SELECT person AS name FROM directedBy
INTERSECT
SELECT playedBy FROM played
```

Question 2.2 (9 Points)

Write an SQL statement that returns the five directors (names) that directed the most movies.

Solution

```
SELECT person
FROM directedBy
GROUP BY person
HAVING count(*) >= ANY (SELECT count(*)
                        FROM directedBy
                        GROUP BY person
                        ORDER BY count(*) DESC
                        LIMIT 5)
ORDER BY count(*) DESC
LIMIT 5;
```

Alternatively, using applying a min to the subquery to get the 5th directory and join with aggregation on $<$.

Question 2.3 (7 Points)

Write an SQL query that returns the names of movie characters and persons where the name contains both the substring “Bat” and “man”.

Solution

```
SELECT name FROM person WHERE name LIKE '%Bat%' AND name LIKE '%man%'
UNION
SELECT character FROM playedBy WHERE character LIKE '%Bat%' AND name LIKE '%man%'
```

or

```
SELECT *
FROM
    (SELECT name FROM person
    UNION
    SELECT character FROM  playedBy) names
WHERE name LIKE '%Bat%' AND name LIKE '%man%'
```

Part 3 Relational Algebra (Total: 24 Points)

Question 3.1 Relational Algebra (10 Points)

Write an relational algebra expression over the schema from the SQL part (part 2) that returns all actors (their names) that played in the movie "Citizen Kane" and have acted in at least three movies (bag-semantics).

Solution

$$
\begin{aligned}
q &= kaneActors \cap moreThreeActors \\
kaneActors &= \delta(\pi_{playedBy}(\sigma_{movie='CitizenKane'}(played))) \\
moreThreeActors &= \pi_{playedBy}(\sigma_{count(*)>2}({}_{count(*)}\alpha_{playedBy}(\delta(\pi_{playedBy,movie}(played)))))
\end{aligned}
$$

Alternatively, using a join.

Question 3.2 SQL → Relational Algebra (8 Points)

Translate the following SQL query into relational algebra (bag semantics).

```
SELECT count(*), year
FROM movies m, directed d
WHERE m.movie = d.movie AND d.person = 'Orson␣Welles'
GROUP BY year
```

Solution

$$q = \pi_{count(*),year}({}_{count(*)}\alpha_{year}(\textbf{movies} \bowtie_{m.title=d.movie} \sigma_{person='OrsonWelles'}(\textbf{directed})))$$

Question 3.3 Equivalences (6 Points)

Consider the following relational schemata:

$R(A, B)$, $S(B, C)$, $T(C, D)$.

Check the equivalences that are correct (bag semantics). For example $R \equiv R$ should be checked, whereas $R \equiv S$ should not be checked.

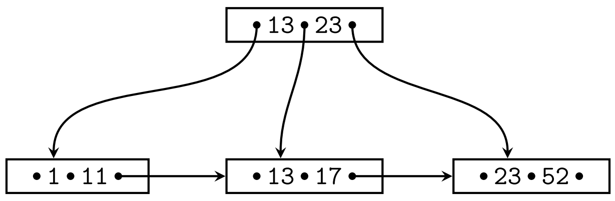
- [✓] $\delta(\sigma_{C_1}(R) \cap \sigma_{C_2}(R)) \equiv \delta(\sigma_{C_1 \wedge C_2}(R))$
- [✓] $\delta({}_{sum(A)}\alpha_B(R)) \equiv_{sum(A)} \alpha_B(R)$
- [✓] $R \bowtie (S \bowtie T) \equiv (T \bowtie R) \bowtie S$
- [ ] $R \bowtie R \equiv R$
- [✓] $\sigma_{B=3}(R) \bowtie S \equiv R \bowtie \sigma_{B=3}(S)$
- [ ] $R - \delta(R) \equiv \emptyset$

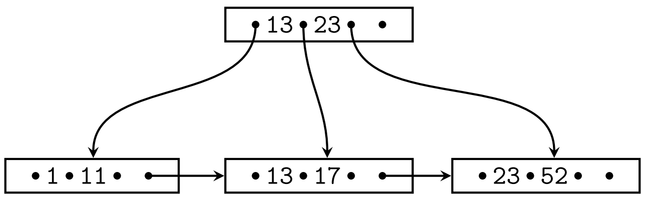
Part 4 Index Structures (Total: 24 Points)

Question 4.1 B+-tree Operations (14 Points)

Given is the B+-tree shown below (*n = 3* **or n = 2**). Execute the following operations and write down the resulting B+-tree:

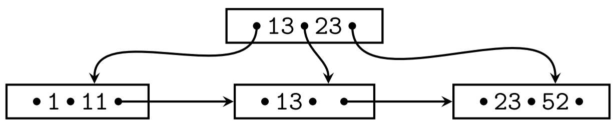
delete(17),insert(24),delete(23),delete(24)



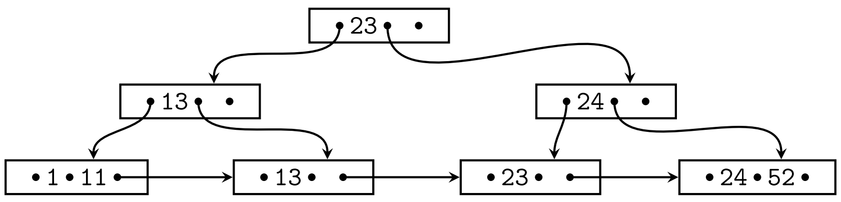


Solution

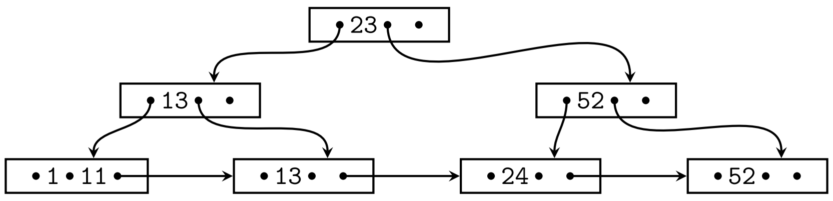
delete(17)



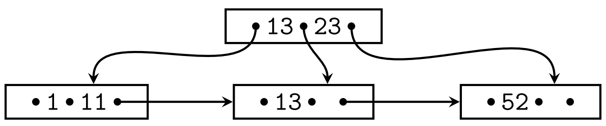
insert(24)



delete(23)

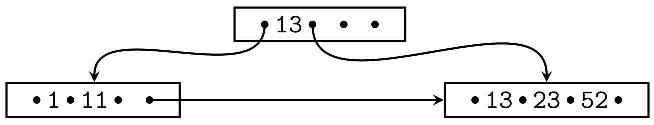


delete(24)

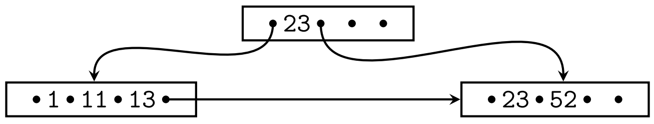


Solution

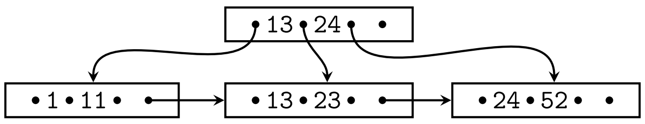
delete(17)



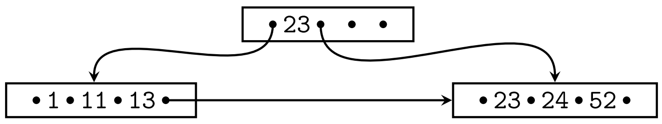
or



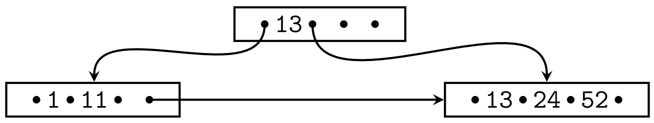
insert(24)



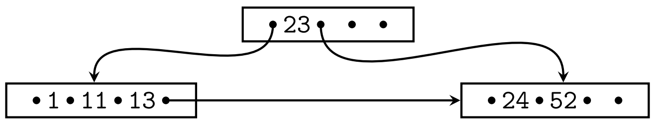
or



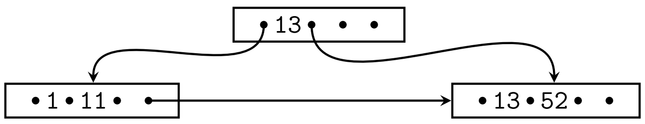
delete(23)



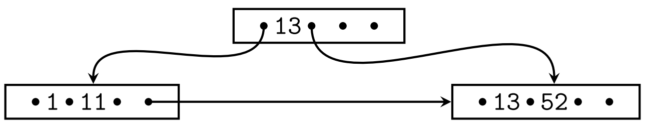
or



delete(24)

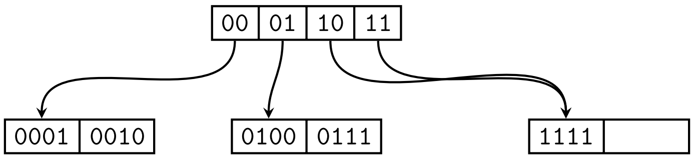


or



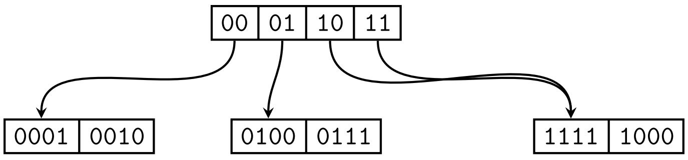
Question 4.2 Extensible Hashing (10 Points)

Consider the extensible Hash index shown below. Each page holds two keys. Write down the resulting index after inserting keys 3, 4, 5 with hash values $h(3) = 1000, h(4) = 1100, h(5) = 0000$.

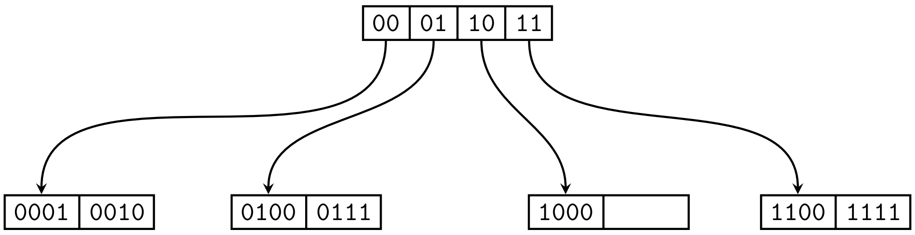


Solution

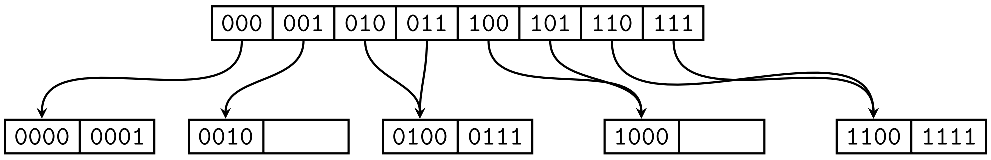
insert(3)



insert(4)



insert(5)



Part 5 Operator Implementations (Total: 16 Points)

Question 5.1 External Sorting (7 = 3.5 + 3.5 Points)

Assume we have $M = 101$ memory pages available for sorting and have to sort a relation R with $B(R) = 30,000$ pages and $N(R) = 600,000$ records.

1. Compute the number of I/O operations needed to sort R using external merge sort without using a min-heap during run generation.
2. Assume there is a B+-tree index on the sort attribute(s) with $K = 100$ keys in each leaf node and height 3. How many I/Os operations are needed to sort relation R if at the beginning of the sort none of the pages of the B+-tree are in memory.

Solution

1. $2 \cdot B(R) \cdot (1 + \lceil log_{M-1}(B(R)/M) \rceil) = 2 \cdot 30,000 \cdot (1 + 2) = 60,000 \cdot 3 = 180,000$ I/Os
2. B+-tree
   - **Clustered** $N(R)/K + HT(I) + B(R) = 600,000/100 + 3 + 30,000 = 36,003$ I/Os
   - **Unclustered** $N(R)/K + HT(I) + N(R) = 600,000/100 + 3 + 600,000 = 636,003$ I/Os

Question 5.2 Join I/O Cost Estimation (9 = 3+3+3 Points)

Consider two relations R and S with $B(R) = 3,000$ and $B(S) = 5,000$. You have $M = 201$ 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.

Solution

- **NL**: $(B(S) + (M-1)) \cdot \frac{B(R)}{M-1} = (5,000 + 200) \cdot \frac{3,000}{200} = 5,200 \cdot 15 = 78,000$ I/Os
- **MJ**: We can generate sorted runs of size 200 that means the number of sorted runs from both R and S is low enough to keep one page from each run in memory. Thus, we can execute the merge phase and join in one path. $3 \cdot (B(R) + B(S)) = 3 \cdot 8,000 = 24,000$ I/Os.
- **HJ**: $\sqrt{B(R)} < M$ and $\sqrt{R(S)} < M$. This means we only need one partition pass. $3 \cdot 8,000 = 24,000$ I/Os.