| | |
|---|---|
| | |
| Name | CWID |

# Exam 1

## March 28, 2019, 01:50pm - 03:05pm

# CS525 - Spring 2019 - MidTerm Solutions

| *Please leave this empty!* | 1 | 2 | 3.1 | 3.2 | 3.3a | 3.3b | Sum |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

# Instructions

- Things that you are **not** allowed to use
  - Personal notes
  - Textbook
  - Printed lecture notes
  - Phone
- You are allowed to bring one page (both two sides can be used) of cheat sheet that must be turned in with the exam
- The exam is **75** minutes long
- For your convenience the number of points for each part and questions are shown in parenthesis.
- There are `3` parts in this exam (`85` points total)
  1. `Disk Organization` (10)
  2. `SQL` (13)
  3. `Index Structures` (62)

## Part 1 Disk Organization (Total: 10 Points)

### Question 1.1 Record Layout (10 Points)

Suppose a disk with an actual (formatted) capacity of 8 gigabytes ($2^{33}$ bytes). The disk has 16 surfaces and 1024 tracks. The disk rotates at 7200 rpm (rotations per minute). The average seek time is 9 ms. The block size is 8 KB.

1. What is the capacity (in bytes) of a single track?

   **Solution**

   I will accept either answers:

   Answer 1: If we assume the disk has 16 surfaces and 1024 tracks per surface:
   capacity (in bytes) of a single track = $\frac{8GB}{16*1024} = \frac{2^{33}}{2^4*2^{10}} = 2^{19}bytes = 0.5MB$

   Answer 2: If we assume the disk has 16 surfaces and 1024 tracks:
   capacity (in bytes) of a single track = $\frac{8GB}{1024} = \frac{2^{33}}{2^{10}} = 2^{23}bytes$

2. Suppose we are reading a file that occupies exactly one entire track. How long does it take to read the entire file sequentially?

   **Solution**

   Transfer time for one track = time for one rotation = $\frac{60s}{7200}$ = 8.3 ms/rotation
   Read time = average seek time + rotational delay + transfer time for track
   $= 9 + \frac{8.3}{2} + 8.3ms = 21.5ms$

## Part 2 SQL (Total: 13 Points)

You have been hired to work on a web site that maintains customer reviews of products. The main data is stored in the following tables:

- `Product`(<u>`pid`:**string**</u>, `pname`:**string**, `description`:**string**)
- `Reviewer`(<u>`rid`:**string**</u>, `rname`:**string**, `city`:**string**)
- `Review`(<u>`rid`:**string**, `pid`:**string**</u>,`rating`:**integer**, `comment`:**string**)

The tables contain the following information:

- `Product:` unique product id (`pid`), product name (`pname`), and product description. All strings.
- `Reviewer:` unique reviewer id (`rid`), and reviewer name (`rname`) and `city`, also all strings.
- `Review:` One entry for each individual review giving the reviewer and product ids, an integer `rating` in the range `0-5`, and the reviewer comment, which is a string.

Write the following queries in `SQL`. No duplicates should be printed in any of the answers.

### Question 2.1 (3 Points)

Return the ratings and comments from all reviews written by reviewers in Chicago for products named ‘ABCD’. The results should be sorted in descending order by rating.

### Solution

```
SELECT DISTINCT r.rating, r.comment
FROM Product p, Review r, Reviewer r1,
WHERE p.pname = ‘ABCD‘ AND r1.city = 'Chicago' AND p.pid = r.pid AND r1.rid = r.rid
ORDER BY r.rating DESC
```

### Question 2.2 (3 Points)

Return the number of reviewers in each distinct city. The results should list the city name and the number of reviewers in that city, and should be sorted alphabetically by city name.

### Solution

```
SELECT r.city, COUNT(r.rid)
FROM Reviewer r
GROUP BY r.city
ORDER BY r.city
```

## Question 2.3 (3 Points)

Return the names of all grumpy reviewers. A grumpy reviewer is one whose average review rating is less than or equal to 2. If someone is in the `Reviewer` table but has no reviews in the `Review` table, they should not be included in the result. The reviewer names in the result can be in any order.

### Solution

```
SELECT rr.rname
FROM Reviewer rr, Review r
WHERE rr.rid = r.rid    -- Reviewers with no reviews are excluded from join here
GROUP BY r.rid, rr.rid, rr.rname
HAVING AVG(r.rating) <= 2
```

## Question 2.4 (4 Points)

Return the names and descriptions of all cool products. A “cool product” is one that has no reviews in the database with any rating less than `4`. A product must have at least one review in the database to be considered as a possible “cool product”. The results can be in any order.

### Solution

```
SELECT p.pname, p.description
FROM Product p, Review r
WHERE p.pid = r.pid
GROUP BY p.pid, p.pname, p.description
HAVING MIN(r.rating) >= 4
```

## Part 3 Index Structures (Total: 62 Points)

### Question 3.1 Conventional Indexes and and B$^+$-Trees (12 Points)

Suppose the usable space on a disk page (block) is `8,000` bytes for indexing. This is the space filled with key values and pointers. A key value and pointer pair is called an entry. The pointers in a leaf node point to tuples in a relation. Pointers in the upper levels point to index nodes at levels below. Each index node is mapped to a disk page (block) and all pointers are the same size: `16` bytes.

You are creating the following indices: index `i1` on `R(A)` and index `i2` on `R(A,B,C)` where attribute `A` is `4` bytes long, attribute `B` is `10` bytes long and attribute `C` is `8` bytes long.

First, compute the capacity of the nodes: maximum number of entries that can be stored in a node/disk page. Recall that each index node (internal or leaf) contains the same type of information and has the same capacity. You may have one extra pointer in each node, but we assume this is part of the header info that is not part of this computation. So you can safely disregard it. Assume each page contains about `75%` of maximum number of entries (the root node or one of the nodes in each level may contain fewer nodes).

Given that `Tuples(R)=500,000`, compute the size of indices `i1` and `i2` (number of nodes at each level). Show your work.

### Solution

Index $i_1$:
- Size of an entry = Size of Attribute A+ Pointer size = $4 + 16 = 20$.
- Maximum number of entries in each node = $\frac{8000}{20} = 400$.

  OR
- $4n + 16n \leq 8000 \Rightarrow$ Each node stores $n = 400$ entries max.

- If nodes contain about 75%, they will store $400 * 75\% = 300$ entries in general.
- Given a total of $500,000$ tuples:
  - Leaf level: $\lceil \frac{500,000}{300} \rceil = 1,667$ nodes
  - Next level: $\lceil \frac{1667}{300} = 6 \rceil$ nodes
  - Next level: $\lceil \frac{6}{300} = 1 \rceil$ node (root)

Index $i_2$:
- Size of an entry = Size of Attributes{A+B+C}+ Pointer size = $4 + 10 + 8 + 16 = 38$.
- Maximum number of entries in each node = $\frac{8000}{38} = 210$.

  OR
- $22n + 16n \leq 8000 \Rightarrow$ Each node stores $n = 210$ entries max.

- If nodes contain about 75%, they will store $210 * 75\% = 157$ entries in general.
- Given a total of $500,000$ tuples:
  - Leaf level: $\lceil \frac{500,000}{157} \rceil = 3,185$ nodes
  - Next level: $\lceil \frac{3,185}{157} = 20 \rceil$ nodes
  - Next level: $\lceil \frac{20}{157} = 1 \rceil$ node (root)

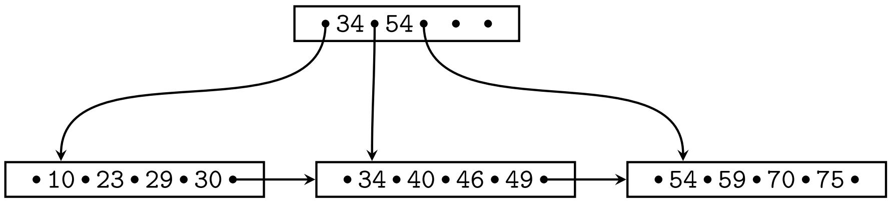
## Question 3.2 B$^+$ Operations (20 Points)

Consider a `B+`-tree whose nodes contain up to `4` keys. Execute the following operations and write down the resulting `B+`-tree after each step.

**insert(80), insert(24), insert(42), delete(29), delete(34), delete(30)**

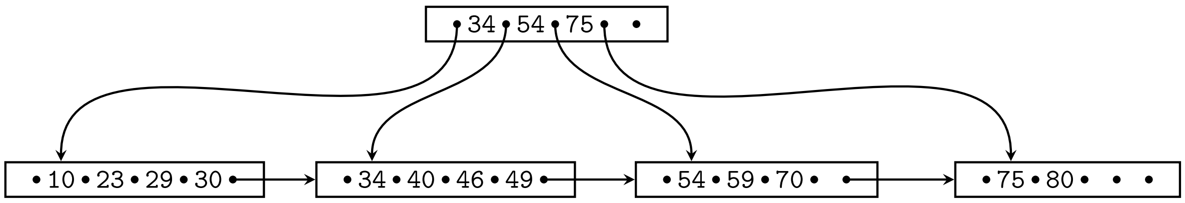
When splitting or merging nodes follow these conventions:

- **Leaf Split**: In case a leaf node needs to be split, the left node should get the extra key if the keys cannot be split evenly.
- **Non-Leaf Split**: In case a non-leaf node is split evenly, the "middle" value should be taken from the right node.
- **Node Underflow**: In case of a node underflow you should first try to redistribute and only if this fails merge. Both approaches should prefer the left sibling.

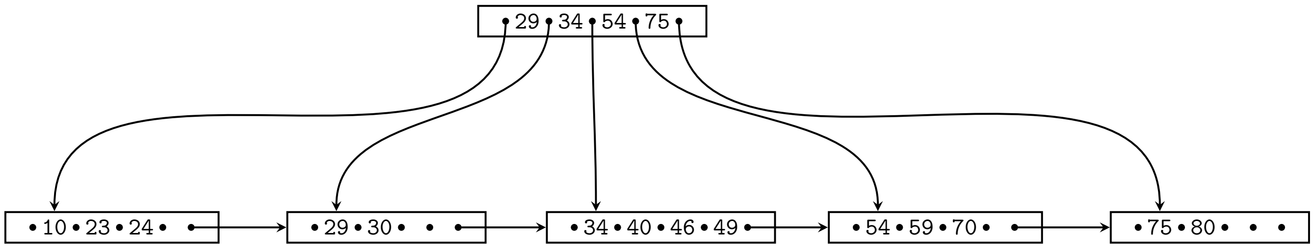


## Solution

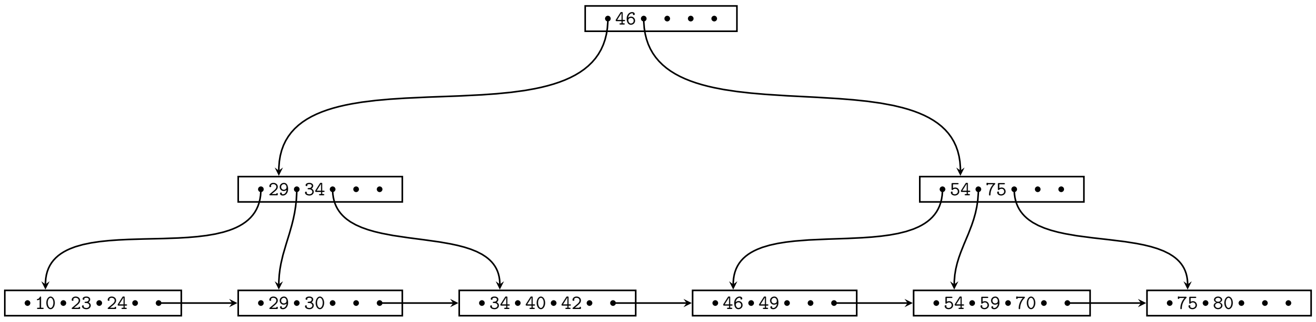
`Insert 80`



`Insert 24`

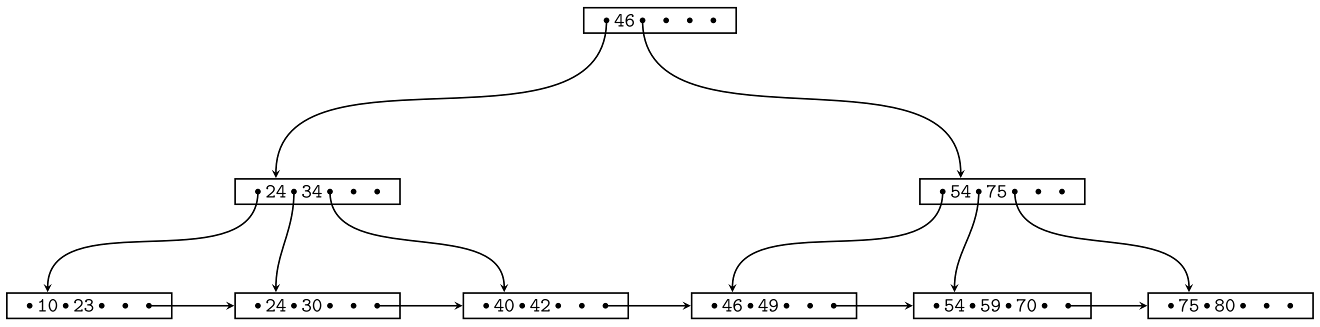


`Insert 42`

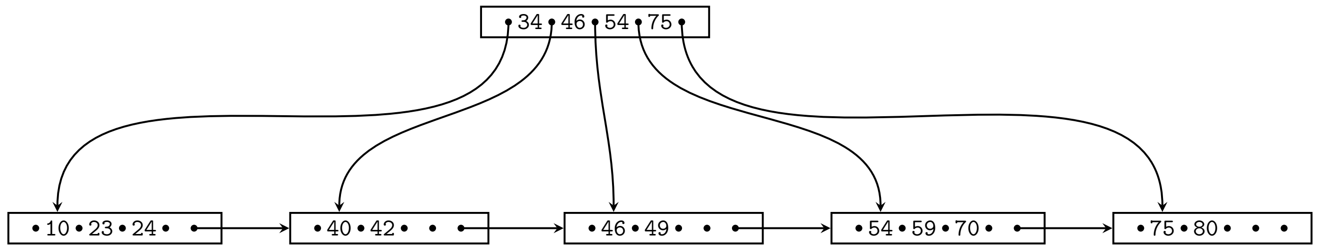


## Solution

Delete 29, then 34



Delete 30



## Question 3.3 Hash Table Operations (30 Points)

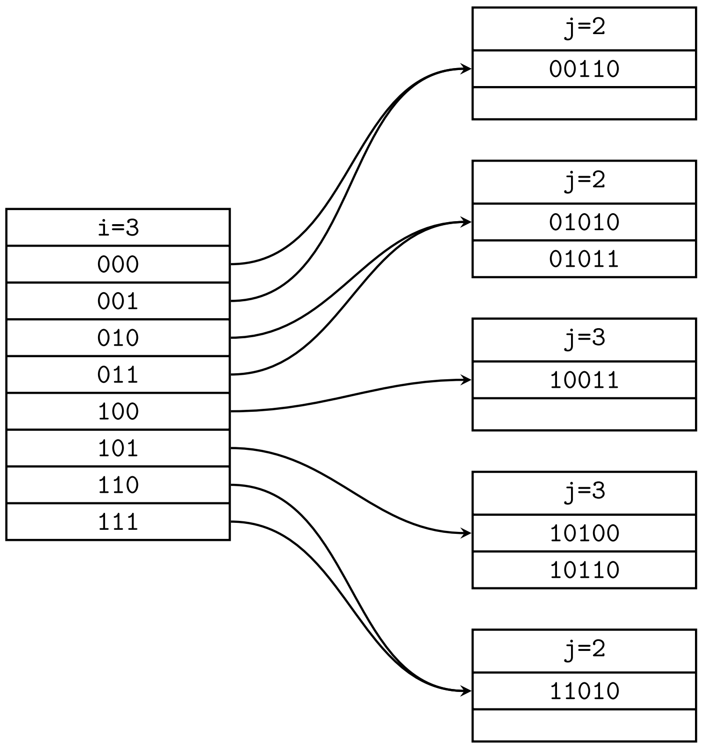
Make the following assumptions:

- A bucket can hold two keys and a pointer.
- The initial database `D` contains one object with key `10100`.

Six objects with the following keys are inserted to `D` in the following order:
`00110, 11010, 10011, 01010, 10110,` and `01011`.

a. Assume that an extensible hash table is used to index the database. Show the index structure after the insertions.

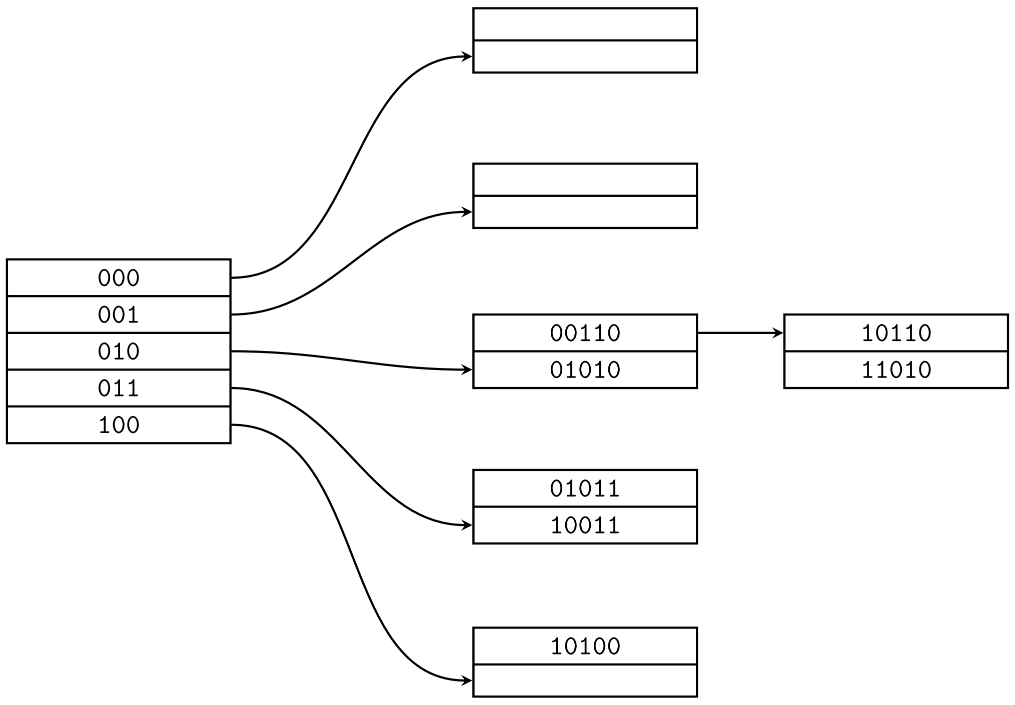
### Solution



b. Assume that a linear hash table is used to index the database with the restriction that at most 80% of the hash table can be full at any time. Show the index structure after the insertions.

## Solution

**i = 3, n = 5, r = 7**



**This page left blank intentionally. There are no more questions.**

**This page left blank intentionally. There are no more questions.**