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

# Exam 1

**March 20th, 2017**

# CS525 - Midterm Exam Solutions

*Please leave this empty!*

| 1 | 2 | 3 | 4 | Sum |
|---|---|---|---|---|
| | | | | |

# Instructions

- Things that you are **not** allowed to use
  - Personal notes
  - Textbook
  - Printed lecture notes
  - Phone
- The exam is **75** 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 4 parts in this exam
  1. SQL (32)
  2. Relational Algebra (26)
  3. Index Structures (24)
  4. I/O Estimation (18)

## Part 1 SQL (Total: 32 Points)

Consider the following database schema and instance storing information about cities, points of interest (POI) and user reviews of points of interest. The $x$ and $y$ coordinates are geographical locations of POI. For cities we record the geographical extend of the city as a rectangle represented as two points: the lower left corner (`xLow`,`yLow`) and the upper right corner (`xHigh`,`yHigh`).

### POI

| pName | x | y | category |
|---|---|---|---|
| Cloud Gate | 1010 | 915 | Sculpture |
| Pizza Joe | 1014 | 915 | Restaurant |
| Fields Museum | 1016 | 916 | Museum |
| Moma | 1340 | 830 | Museum |

### city

| cName | xLow | xHigh | yLow | yHigh |
|---|---|---|---|---|
| Chicago | 950 | 1060 | 900 | 940 |
| New York | 1250 | 1400 | 790 | 850 |

### reviews

| poi | user | rating | review |
|---|---|---|---|
| Could Gate | Alice | 4 | looks cool |
| Pizza Joe | Peter | 1 | seriously overrated |
| Pizza Joe | Alice | 3 | my personal favourite |

**Hints:**

- When writing queries do only take the schema into account and **not** the example data given here. That is your queries should return correct results for all potential instances of this schema.
- Attributes with black background form the primary key of an relation. For example, **pName** is the primary key of relation **POI** (points of interest).
- The attribute **poi** of relation **reviews** is a foreign key to relation **POI**.

## Question 1.1 (6 Points)

Write an SQL query that returns the names of all restaurants (POIs with `category` equal to “Restaurant”) that are in “Chicago” their coordinate (`x`,`y`) lies within the bounds of the rectangle stored for Chicago.

## Solution

```
SELECT pName
FROM POI p, city c
WHERE x BETWEEN xLow AND xHigh
      AND y BETWEEN yLow AND yHigh
      AND cName = 'Chicago'
      AND category = 'Restaurant';
```

## Question 1.2 (10 Points)

Write an SQL query that returns the POI category with the highest number of POIs relative to its extend. That is normalize the number of POIs within each city by the area of the rectangle (`xLow`,`yLow`,`xHigh`,`yHigh`).

### Solution

```
WITH poiCity AS (
  SELECT cName, (xHigh − xLow) ∗ (yHigh − yLow) AS area
  FROM POI p, city c
       x BETWEEN xLow AND xHigh
       AND y BETWEEN yLow AND yHigh
),

poiNum AS (
  SELECT cName, count(∗) / area AS relPOI
  FROM poiCity
)

SELECT cName
FROM poiNum p
WHERE relPOI = (SELECT max(relPOI) FROM poiNum);
```

## Question 1.3 (7 Points)

Write an SQL query that returns the number of POI categories with an average rating of POIs in this category that is higher than 4.

### Solution

```
SELECT count(*) AS numC
FROM
    (SELECT category
    FROM POI p, reviews r
    WHERE pName = poi
    GROUP BY category
    HAVING AVG(rating) > 4) AS avgR
```

## Question 1.4 (9 Points)

Find the closest point of interest (POI) to geographical location (x=500, y=600) that belongs to category 'Restaurant' and where at least 25% of ratings for this POI are higher than 3. Use euclidian distance to compute distance $d(p_1, p_2) = \sqrt{(p_1.x - p_2.x)^2 + (p_1.y - p_2.y)^2}$. You can assume that the database system supports a function sqrt that computes the square root of a number.

### Solution

```
WITH AS rest (
  SELECT pName, x, y
  FROM POI p JOIN reviews r ON (pName = poi)
  WHERE category = 'Restaurant'
  GROUP BY pName, x, y
  HAVING count(1) AS totalCount / sum(CASE WHEN rating > 3 THEN 1 ELSE 0) <= 4.0
),

SELECT city, sqrt((x - 500) * (x - 500) + (y - 600) * (y - 600)) AS dist
FROM rest
ORDER BY dist
LIMIT 1;
```

## Part 2 Relational Algebra (Total: 26 Points)

### Question 2.1 Relational Algebra (10 Points)

Write an relational algebra expression over the schema from the SQL part that returns the name of the user which has written the most reviews. Use the **bag semantics** version of relational algebra.

### Solution

$$\begin{aligned} rewPerUser &= {}_{user}\alpha_{a \leftarrow count(1)}(reviews) \\ q &= \pi_{user}(rewPerUser \ltimes_{a=b} \alpha_{b \leftarrow max(a)}(rewPerUser)) \end{aligned}$$

## Question 2.2 Relational Algebra (10 Points)

Write an relational algebra expression over the schema from the SQL part that returns the distance between all pairs of POIs in New York. Ensure that each pair is only returned once. For example, return only one of `(Cloud Gate, Pizza Joe)` and `(Pizza Joe, Cloud Gate)`. Also do not pair a point with itself, e.g., `(Cloud Gate, Cloud Gate)`. Similar to the SQL case assume the existence of a function *sqrt* for computing the square root of a number. Use the **bag semantics** version of relational algebra.

## Solution

$$\begin{aligned} nyPoi &= \pi_{pName,x,y}(POI \bowtie_{xLow \leq x \wedge x \leq xHigh \wedge yLow \leq y \wedge y \leq yHigh} \sigma_{pName='NewYork'}(city)) \\ q &= \pi_{pName,oName,sqrt((x1-x2)\cdot(x1-x2)+(y1-y2)\cdot(y1-y2))}(\rho_{oName,x2,y2}(nyPoi) \bowtie_{pName<oName} \rho_{oName,x1,y1}(nyPoi) \end{aligned}$$

## Question 2.3 Relational Algebra (6 Points)

Write an relational algebra expression over the schema from the SQL part that returns for each POI the highest and lowest rating. Use the **bag semantics** version of relational algebra.

## Solution

$$q = {}_{poi}\alpha_{min(rating),max(rating)}(reviews)$$

## Part 3 Index Structures (Total: 24 Points)

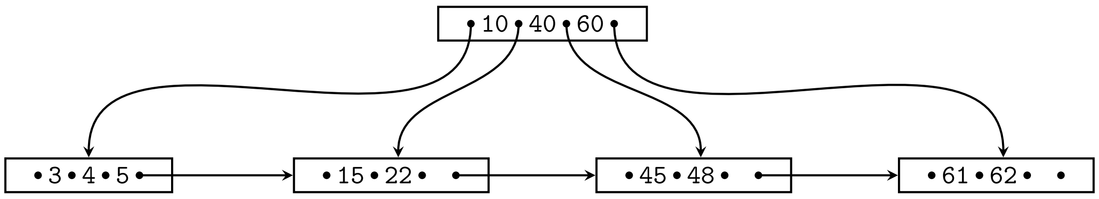
### Question 3.1 B+-tree Operations (24 Points)

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

**insert(88),insert(99),insert(6),delete(3)**

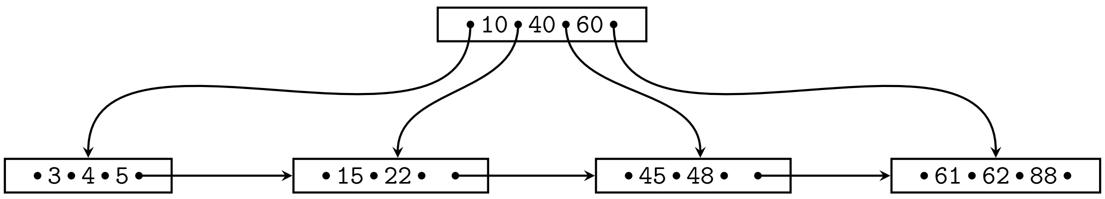
When splitting or merging nodes follow these conventions:

- **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.

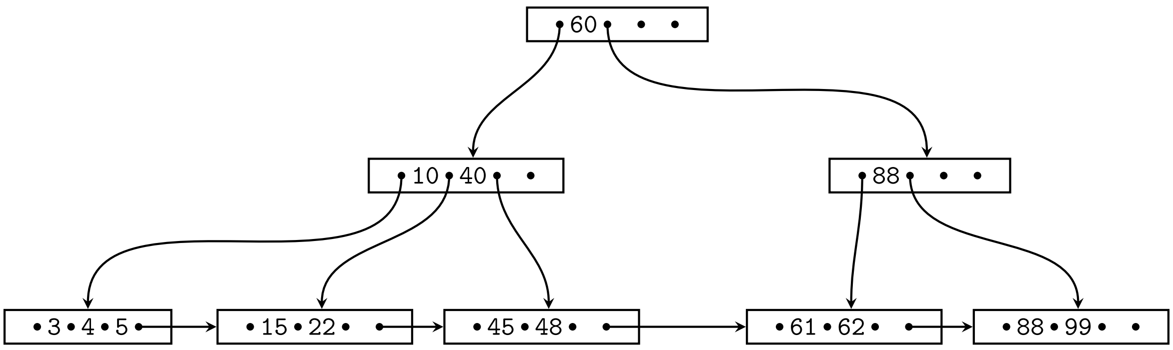


### Solution

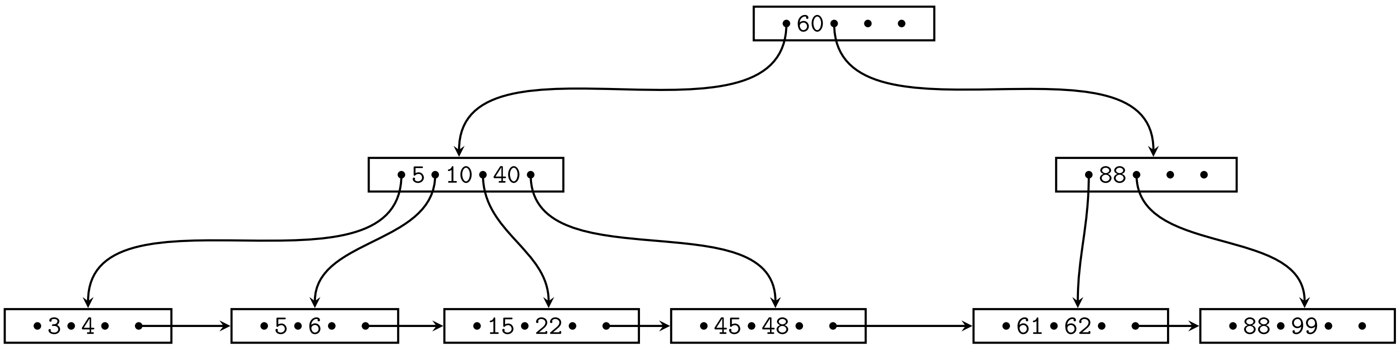
**insert(88)**



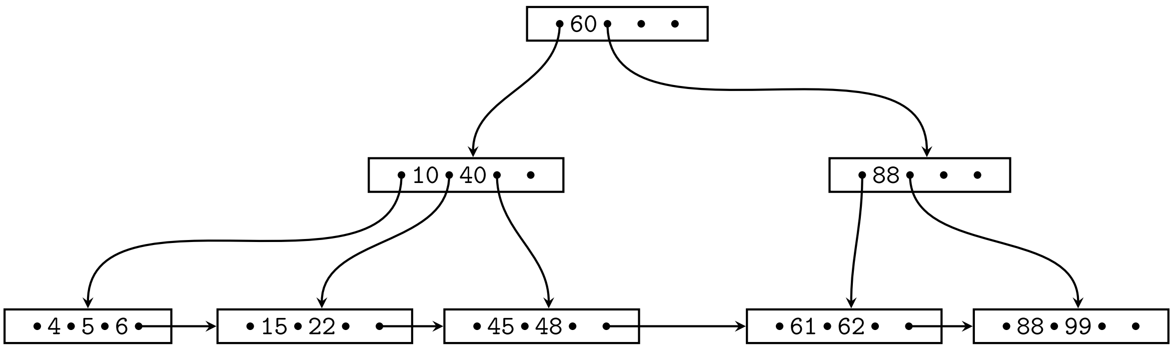
**insert(99)**



**insert(6)**



**delete(3)**



## Part 4 I/O Estimation (Total: 18 Points)

### Question 4.1 I/O Cost Estimation (12 = 4 + 4 + 4 Points)

Consider two relations $R$ and $S$ with $B(R) = 300$ and $B(S) = 20,000$. You have $M = 201$ memory pages available. Compute the 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 distributes keys evenly across buckets. Justify you result by showing the I/O cost estimation for each join method.

### Solution

**Block Nested-loop**:
Use smaller table $R$ as the inner. We only have two chunks of size 200. Thus, we get $2 \times (200 + B(S)) = 40{,}400$ I/Os.
**Merge-join**:
Relation $R$ can be sorted with one merge phase memory resulting in $2 * 2 * B(R) = 1,200$ I/Os. Relation $S$ requires one merge phase, merging 100 runs in the last phase: $2 \times 2 \times B(S) = 80,000$ I/Os. The last merge phase of relation $S$ can be combined with the last merge phase of $R$ ($3 + 100 = 103$ blocks of memory required). The merge join can be execute during these merge phases avoiding on read of relations $R$ and $S$. Without optimizations we get $5 * (B(R) + B(S)) = 101,500$. If we execute the merge-join during the last merge phases we get $3 * (B(R) + B(S)) = 60,900$.
**Hash-join**:
We need one partitioning phase for the partitions of Relation $R$ to fit into memory. Thus, the hash-join requires $3 * (B(R) + B(S)) = 60,900$ I/O.

### Question 4.2 External Sorting (6 Points)

Consider a relation $R$ with $B(R) = 64,000,000$. Assume that $M = 33$ memory pages are available for sorting. How many I/O operations are needed to sort this relation using no more than $M$ memory pages.

### Solution

External sorting requires $2 \times (1 + \lceil \log_{M-1}(\frac{B(R)}{M}) \rceil) \times B(R) = 2 \times 6 \times 64,000,000 = 728,000,000$ I/Os.