Name

CWID

Quiz
1

September 22th, 2015
Due September 29th, 11:59pm

Quiz 1: CS525 - Advanced Database Organization

Results

Please leave this empty! 1.1 1.2 1.3 1.4 Sum

Instructions

- **You have to hand in the assignment using your bitbucket account**
- **This is an individual and not a group assignment**
- 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 quiz
    1. SQL
    2. Relational Algebra
    3. Index Structures
    4. Result Size Estimation

Part 1.1 SQL (Total: 31 + 10 bonus points Points)

Consider the following desaster event database schema and example instance. The example data should not be used to formulate queries. SQL statements that you write should return the correct result for every possible instance of the schema!

city

name	state	population
Chicago	IL	6,500,200
Honolulu	HI	300,000
Madison	WI	400,000

event

city	year	eventType	deaths
Honolulu	1982	volcano	10
Chicago	1871	fire	300
Chicago	1915	ship wreck	844

predictions

city	eventType	deaths
Chicago	ice age	5,000,000
Honolulu	volcano	150
Madison	riot	1000

measures

type	cost	effectPerc	worksFor
police	$5,000	95	riot
firemen	$3,000	80	fire
firemen	$3,000	10	volcano
atomic heater	$100,000,000	70	ice age

Hints:

- Attributes with black background are the primary key attributes of a relation
- The attribute *city* of relation *event* and *predictions* are both foreign keys to attribute *name* of relation *city*.

Question 1.1.1 (2 Points)

Write a query that returns cities with predicted disasters (relation `predictions`) which have not been the site of disasters in the past (relation `event`). Return each such city only once.

Solution

```
SELECT city FROM predictions
EXCEPT
SELECT city FROM events;
```

or

```
SELECT name
FROM city
WHERE name IN (SELECT city FROM predictions)
      AND name NOT IN (SELECT city FROM event);
```

Question 1.1.2 (2 Points)

Write an SQL query that returns a rolling sum of the total number of deaths per city by year, i.e., in the example database there were two disasters in Chicago in 1871 and 1915. Thus, the output for Chicago would be `(1871,300)` and `(1915,1144)`.

Solution

```
SELECT city, year, sum(deaths) OVER (PARTITION BY city ORDER BY year)
FROM event;
```

Alternatively, a join can be used to combine an event tuple with all event tuples for that city with an earlier year.

Question 1.1.3 (3 Points)

Return a list of cities ordered by safety. The safety of a city is the number of deaths in the past and predicted disasters divided by the number of residents, e.g., if a city has 100 total deaths and 1000 inhabitants then its safety is 1/10. Note that lower safety values are better.

Solution

```
SELECT * FROM
(SELECT city, t.ttlDeaths / population AS safety
FROM (SELECT sum(deaths) AS ttlDeaths, city, population
     FROM
          (SELECT city, deaths FROM event)
          UNION ALL
          (SELECT city, deaths FROM predictions) d,
          city c
     WHERE c.name = d.city
     GROUP BY city, population
     ) numD
) safe
ORDER BY safety
```

Question 1.1.4 (4 Points)

Write an SQL query that returns events (all attributes from the event table) such that there were no other events in the same city for at least 100 years.

Solution

```
SELECT *
FROM event e
WHERE year + 100 < ALL (SELECT e2.year
                        FROM event e2
                        WHERE e.year < e2.year AND e.city = e2.city);
```

of course one can also use not exists

Question 1.1.5 (3 Points)

Write an SQL query that returns the names of cities where all types of disasters existing in the database have taken place (e.g., in the example instance the disaster types are `volcano`, `fire`, `ship wreck`, `ice age`, and `riot`).

Solution

```
WITH des AS (SELECT eventType FROM event
             UNION ALL
             SELECT eventType FROM predictions)

SELECT name
FROM city c1
WHERE c1.name NOT IN (SELECT c2.name
                      FROM city c2, des d
                      WHERE NOT EXISTS (SELECT *
                                        FROM event e
                                        WHERE e.eventType = d.eventType
                                              AND e.city = c2.name);
```

of course a double `NOT EXISTS` would work too!

Question 1.1.6 (3 Points)

Write an SQL query that returns the population of cities in the future assuming that all predicted disasters will occur and that the city population will not change otherwise.

Solution

```
SELECT name, GREATEST(population - allDeaths, 0) AS newPop
FROM city c, LATERAL (SELECT sum(deaths) AS allDeaths
                      FROM predictions p
                      WHERE p.city = c.name);
```

Alternatively, a join could be used or pure aggregation realizing that it is ok to group on population too.

Question 1.1.7 (4 Points)

Return the name of the city/cities with the most disasters.

Solution

```
SELECT city
FROM event
GROUP BY city
HAVING count(*) = (SELECT max(cnt)
                   FROM (SELECT count(*) AS cnt
                         FROM event
                         GROUP BY city) e);
```

Question 1.1.8 (4 Points)

Write an SQL query that returns the names of cities without any past (events) and future (predictions) disasters.

Solution

```
SELECT name
FROM city
WHERE name NOT IN (SELECT city FROM events UNION ALL SELECT city FROM predictions);
```

of course two `NOT IN` or `NOT EXISTS` would work too

Question 1.1.9 (6 Points)

Write a query that returns the most effective combination of up to 3 measures for future predicted disasters for which the combined cost is less than $1,000,000. Measures can only be applied to the disaster types indicated in the `measures` table (attribute `worksFor`). The effectiveness of a set of measures is the sum of the prevented deaths of the measure. The number of prevented deaths is computed as the number of deaths of a disaster multiplied by the effectPerc of the measure.

Hint: This is a relatively complex query. Recall that you can use `WITH` in SQL to define temporary views.

Solution

```
WITH applyM AS (
    (SELECT RANK() OVER (ORDER BY city,eventType) AS comID,
            deaths  / 100 * effectPerc AS eff,
            type AS measure,
            cost,
            city,
            eventType AS ev
    FROM predictions e JOIN measures m ON e.eventType =m.worksFor)
    UNION ALL
    (SELECT -1 AS comID, 0, NULL, 0, NULL, NULL
    FROM dual)
    UNION ALL
    (SELECT -2 AS comID, 0, NULL, 0, NULL, NULL
    FROM dual)
    ),

upToThree AS
    (SELECT m1.measure AS m1measure, m1.city AS m1city, m1.ev AS m1ev,
         m2.measure AS m2measure, m2.city AS m2city, m2.ev AS m2ev,
            m3.measure AS m3measure, m3.city AS m3city, m3.ev AS m3ev,
            m1.cost + m2.cost + m3.cost AS cost,
            m1.eff + m2.eff + m3.eff AS eff
    FROM applyM m1, applyM m2, applyM m3
    WHERE m1.comID > m2.comID AND m2.comID > m3.comID)

SELECT m1measure, m1city, m1ev,
       m2measure, m2city, m2ev,
       m3measure, m3city, m3ev,
       cost, eff
FROM (
     SELECT m1measure, m1city, m1ev,
            m2measure, m2city, m2ev,
            m3measure, m3city, m3ev,
            cost, eff
     FROM upToThree
     WHERE cost < 1000000
     ORDER BY eff DESC)
WHERE ROWNUM = 1;
```

Question 1.1.10 Optional Bonus Question (10 bonus points Points)

Write an interpreter of stack operations as a recursive SQL query. The initial stack state is stored in a relation `stack(pos,element)` that records which stack position stores which element (starting from position 0). The operations to be executed by your interpreter are stored in a relation `ops(seq,op,element)` where `seq` stores the order of operations, `op` is the type of operation, and `element` is the element which is used by the operation. You have to support the following operations: 1) `pop` removes the top element (at position 0) from the stack and 2) `push` pushes the element stored in attribute `element` onto the stack. The result of your query should be the new content of the `stack` relation. An example instance of the `stack` and `ops` relations are shown below.

stack

pos	element
0	5
1	19
2	23

event

seq	op	element
0	pop	
1	push	13
2	push	35
3	push	37

Solution

```
WITH
   init AS (
       SELECT 0 AS iter, pos, elem
       FROM stack
       UNION ALL
       SELECT 0 AS iter, -1 AS pos, NULL AS elem
       FROM dual
   ),

   states(iter, pos, elem) AS (
       SELECT * FROM init
       UNION ALL
       (SELECT iter + 1 AS iter,
               CASE WHEN pos = -1 AND choice = 1 THEN -1
                    WHEN pos = -1 AND choice = 2 THEN 0
                    WHEN op = 'push' THEN pos + 1
                    ELSE pos - 1
               END AS pos,
               CASE WHEN pos < 0 AND choice = 2 AND op = 'push' THEN o.elem
                    ELSE states.elem
               END AS elem
        FROM  states
               JOIN
               ops o ON (iter = seq)
               JOIN
               (SELECT 1 AS choice
               FROM dual
               UNION ALL
               SELECT 2
               FROM dual) c ON (choice = 1 OR op = 'push')
        WHERE not (pos = 0 AND op = 'pop')
              AND (choice = 1 OR (pos = -1 AND op = 'push'))))

SELECT pos, elem
FROM states
WHERE iter = (SELECT max(iter) FROM states) AND pos >= 0
ORDER BY pos;
```

Part 1.2 Relational Algebra (Total: 29 Points)

Question 1.2.1 Relational Algebra (3 Points)

Write a relational algebra expression over the schema from the SQL part (part 1) that returns the number of deaths for all volcano and riot disasters.

Solution

$$\pi_{deaths}(\sigma_{eventType=volcano \vee eventType=riot}(\textbf{events}))$$

Question 1.2.2 Relational Algebra (4 Points)

Write a relational algebra expression over the schema from the SQL part (part 1) that returns the cities with more than 2 predicted disasters.

Solution

$$\pi_{city}(\sigma_{count(*)>2}(\ _{city}\alpha_{count(*)}(\textbf{predictions})))$$

Question 1.2.3 Relational Algebra (4 Points)

Write a relational algebra expression over the schema from the SQL part (part 1) that returns cities with volcano eruptions but without riots.

Solution

$$\pi_{city}(\sigma_{eventType=volcano}(\textbf{events})) - \pi_{city}(\sigma_{eventType=riot}(\textbf{events}))$$

Question 1.2.4 SQL → Relational Algebra (3 Points)

Translate the SQL query from Question 1.1.3 into relational algebra (bag semantics).

Solution

$$\begin{aligned}\textbf{comE} &= \pi_{city,deaths}(event) \cup \pi_{city,deaths}(\textbf{predictions})\\ \textbf{ttl} &= {}_{city,population}\alpha_{sum(deaths)}(\textbf{comE} \bowtie_{name=city} \textbf{city})\\ \textbf{q} &= \pi_{city,sum(deaths)/population \to safety}(\textbf{ttl})\end{aligned}$$

of course `ORDER BY` cannot be translated.

Question 1.2.5 SQL → Relational Algebra (5 Points)

Translate the SQL query from question 1.1.4 into relational algebra (bag semantics).

Solution

$$\begin{aligned}\textbf{hasSmallGap} &= (\rho_{e1}(\textbf{event}) \ltimes_{e1.year<e2.year \wedge e1.year+100 \geq e2.year \wedge e1.city=e2.city} \rho_{e2}(\textbf{event}))\\ \textbf{q} &= \textbf{event} - \textbf{hasSmallGap}\end{aligned}$$

Question 1.2.6 SQL → Relational Algebra (6 Points)

Translate the SQL query from question 1.1.5 into relational algebra (bag semantics).

Solution

$$\begin{aligned}
\textbf{types} &= \pi_{eventType}(\textbf{event}) \cup \pi_{eventType}(\textbf{predictions}) \\
\textbf{desCity} &= \pi_{name}(\textbf{city}) \times \textbf{types} \\
\textbf{notHasType} &= \pi_{name}(\textbf{desCity} - \pi_{city,eventType}(\textbf{events})) \\
\textbf{q} &= \pi_{name}(\textbf{city}) - \textbf{notHasType}
\end{aligned}$$

Question 1.2.7 Equivalences (4 Points)

Consider the following relation schemas:

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

Check equivalences that are correct under **set semantics**. For example $R \bowtie R \equiv R$ should be checked, whereas $R \equiv S$ should not be checked.

- [x] $\sigma_{A=5}(R \bowtie_{B=C} T) \equiv \sigma_{A=5}(R) \bowtie_{B=C} T$
- [ ] $R \triangleright S \equiv S \ltimes R$
- [x] $R \bowtie (S \cup T) \equiv (R \bowtie S) \cup (R \bowtie \rho_{B \leftarrow C}(T))$
- [x] $\pi_A(R \rtimes S) \equiv \pi_A(S \ltimes R)$
- [ ] $R - (S - T) \equiv (R \cup T) - S$
- [x] $(R \cap S) \cup (R \cap T) \equiv R \cap (S \cup T)$
- [x] $\sigma_{A<3}(\sigma_{B<4}(R)) \equiv \sigma_{A<3}(R) \bowtie \sigma_{B<4}(R)$
- [x] $\delta(R) \equiv {}_{A,B}\alpha(R)$

Part 1.3 Index Structures (Total: 30 Points)

Assume that you have the following table:

Item

SSN	name	age
1	Pete	13
2	Bob	23
45	Alice	77
44	John	49
43	Joe	45
42	Lily	3
88	Gertrud	29
89	Heinz	14

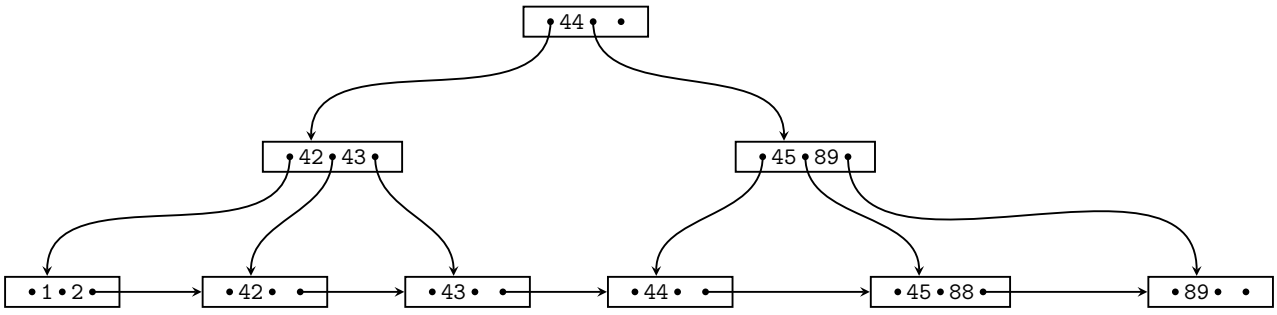
Question 1.3.1 Construction (12 Points)

Create a B+-tree for table *Item* on key *SSN* 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.

Solution

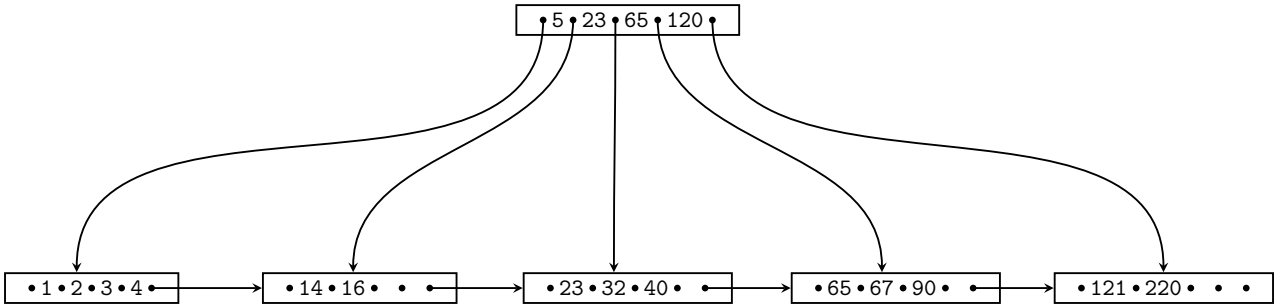


Question 1.3.2 Operations (10 Points)

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

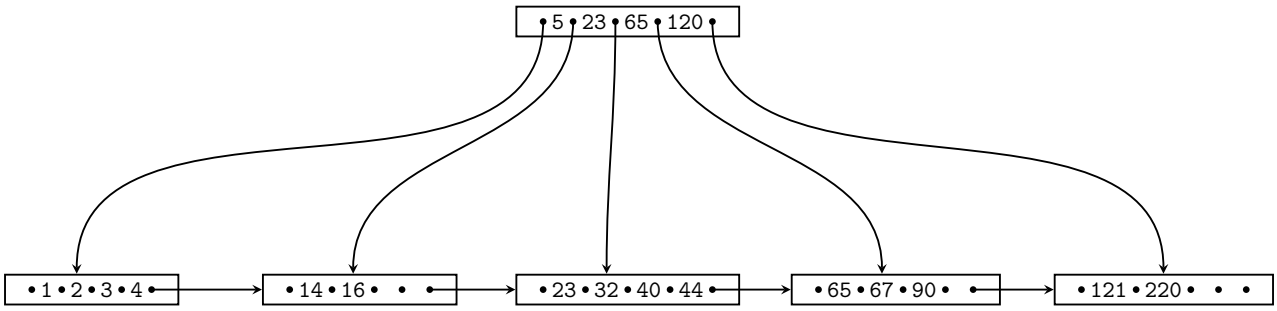
insert(44), insert(59), insert(5), delete(121), delete(220), insert(300)

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

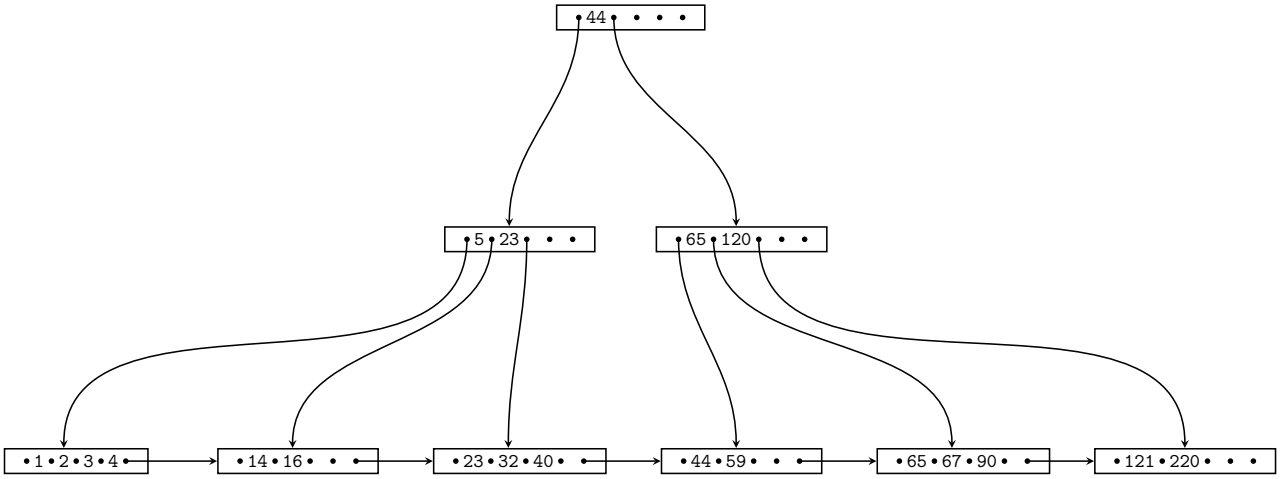


Solution

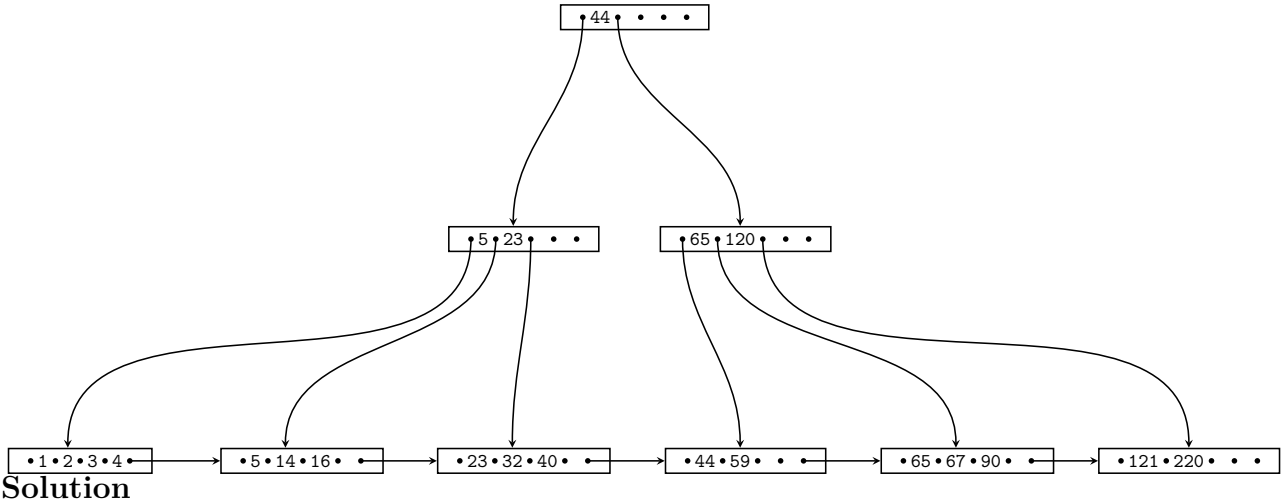
Insert 44



Insert 59

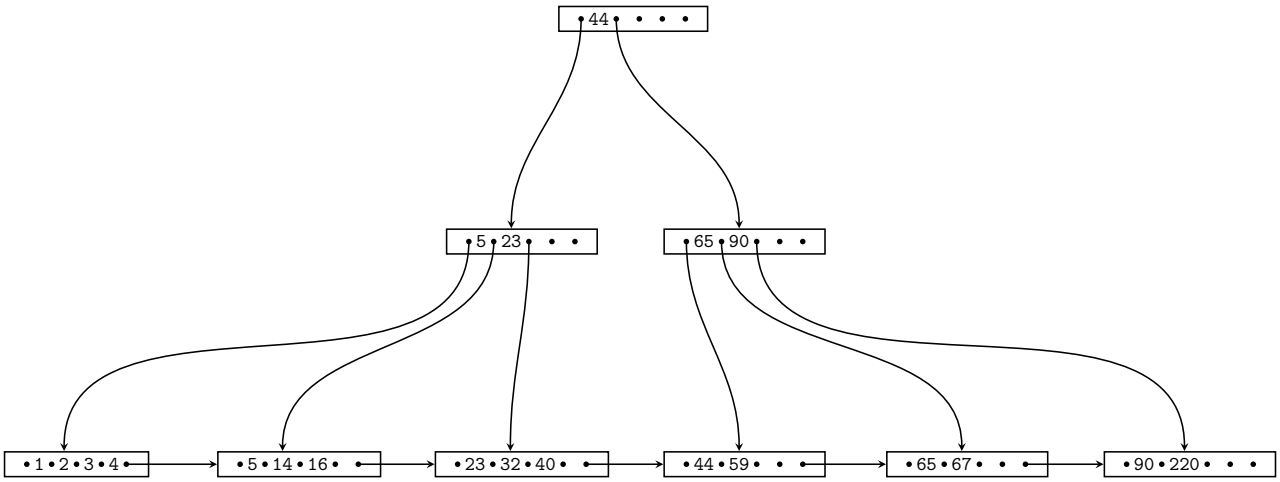


Insert 5

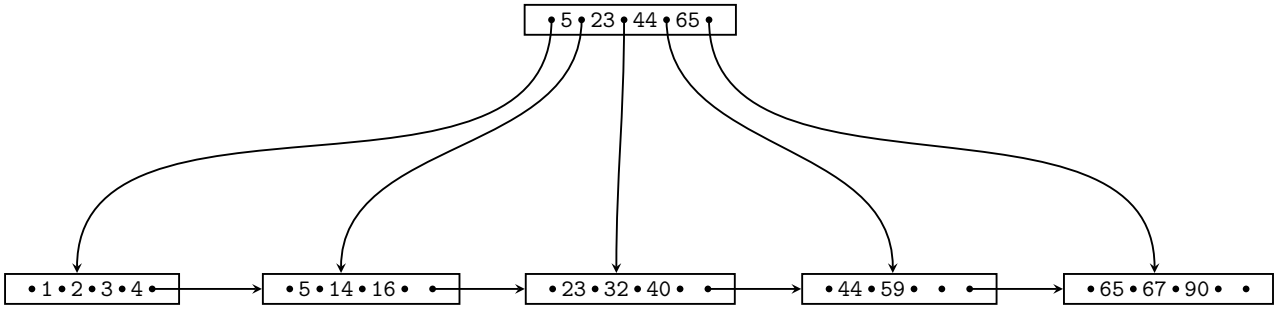


Solution

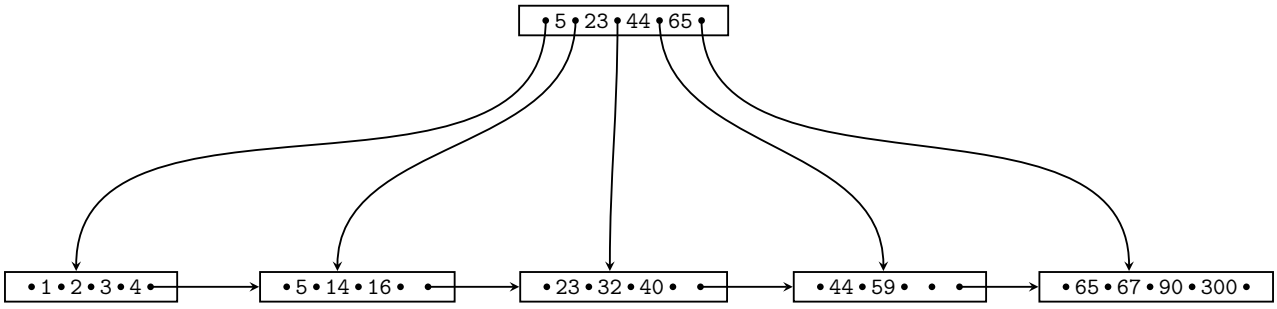
Delete 121



Delete 220 (Incorrectly showed Delete 200)



Insert 300



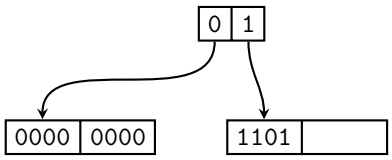
Question 1.3.3 Extensible Hashing (8 Points)

Consider the extensible Hash index shown below that is the result of inserting values 2, 7, and 4. Each page holds two keys. Execute the following operations

`insert(1),insert(5),insert(6),insert(8),delete(2)`

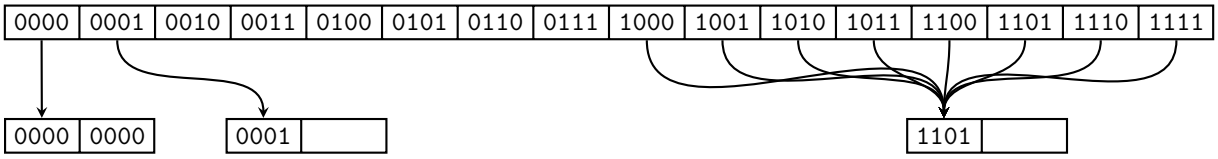
and write down the resulting index after each operation. Assume the hash function is defined as:

x	h(x)
0	1100
1	0001
2	0000
3	1010
4	1101
5	0111
6	1110
7	0000
8	1010

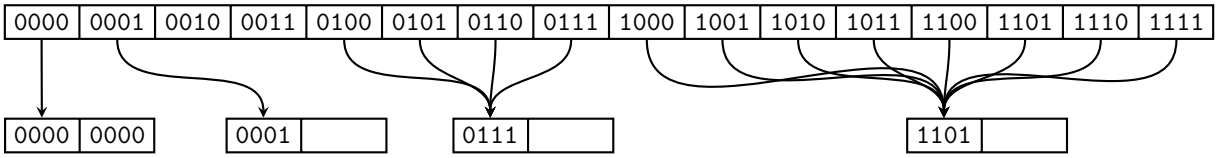


Solution

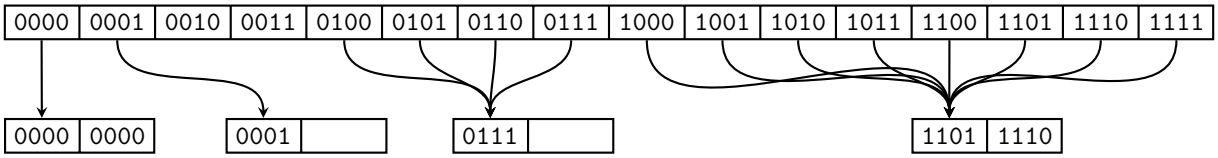
insert(1)



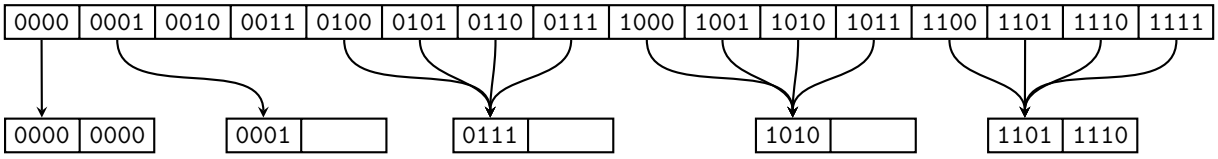
insert(5)



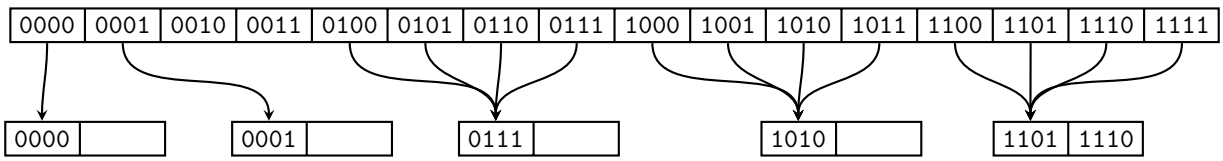
insert(6)



insert(8)



delete(2)



Part 1.4 Result Size Estimations (Total: 10 Points)

Consider a table *church* with attributes *name*, *city*, *confession*, *capacity*, a table *person* with *name*, *confession*, *age*, and a table *attendsService* with attributes *person* and *church*. *attendsService.person* is a foreign key to person *name*. Attribute *church* of relation attendsService is a foreign key to attribute *name* of relation church. Given are the following statistics:

$$T(church) = 50 \qquad T(person) = 300,000 \qquad T(attendsService) = 250,000$$
$$V(church, name) = 50 \qquad V(person, name) = 300,000 \qquad V(attendsService, person) = 250,000$$
$$V(church, city) = 30 \qquad V(person, confession) = 6 \qquad V(attendsService, church) = 45$$
$$V(church, confession) = 5 \qquad V(person, age) = 100$$
$$V(church, capacity) = 45$$

Question 1.4.1 Estimate Result Size (3 Points)

Estimate the number of result tuples for the query $q = \sigma_{confession=catholic}(church)$ using the first assumption presented in class (values used in queries are uniformly distributed within the active domain).

Solution

$$T(q) = \frac{T(church)}{V(church, confession)} = \frac{50}{5} = 10$$

Question 1.4.2 Estimate Result Size (3 Points)

Estimate the number of result tuples for the query $q = \sigma_{age>30}(person)$ using the first assumption presented in class. The minimum and maximum values of attribute *age* are 1 and 100.

Solution

$$T(q) = \frac{(max(age) - 30) \times T(person)}{max(age) - min(age) + 1} = \frac{(100 - 30) \times 300,000}{100 - 1)} = 210,000$$

Question 1.4.3 Estimate Result Size (4 Points)

Estimate the number of result tuples for the query $q = (person \bowtie_{name=person} attendsService \bowtie_{church=church.name} church$ using the first assumption presented in class.

Solution

$$T(q) = \frac{T(person) \times T(attendsService) \times T(church)}{max(V(person, name), V(attendsService, person)) \times max(V(attendsService, church), V(church, name))}$$

$$= \frac{300,000 \times 250,000 \times 50}{max(300,000, 250,000) \times max(45, 50)} = 250,000$$