

Name

CWID

Quiz 1

Sept 24th, 2018
Due Oct 2nd, 11:59pm

Quiz 1: CS525 - Advanced Database Organization

Please leave this empty! 1.1

1.2

1.3

1.4

Sum

Instructions

- **You have to hand in the assignment using your blackboard**
- **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 political party 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!**

party

pName	leaning	established	endowment
JDF	middle right	1950	6,000,000
LKE	middle left	1951	5,500,000
HHH	far right	2010	2,300,000
GGK	middle	1980	10,000,000

member

name	party	since
Peter	JDF	2014
Bob	JDF	2014
Alice	JDF	2011
Alice	GGK	1990

positions

name	pos	year	salary
Alice	Governor of Alaska	2017	100,000
Alice	Vice President	2018	200,000
Bob	Secretary of State	2016	150,000

donations

pName	year	donor	amount
JDF	2017	Walmart	20,000,000
HHH	2018	Koch	40,000,000

Hints:

- Attributes with black background are the primary key attributes of a relation
- Attribute party of relation member is a foreign keys to relation party.
- Attribute pName of relation donations is a foreign key to relation party.

Question 1.1.1 (2 Points)

Write an SQL query that returns for each party the total amount of donation dollars per year as a rolling sum, i.e., the output for 2016 will be the sum of all of the donations made in up to and including year 2016.

Question 1.1.2 (4 Points)

For each political leaning of parties (attribute `leaning`), return the names of the three donors which have donated the largest total donation (sum of donated dollars) to parties with this leaning.

Question 1.1.3 (4 Points)

Write a query that calculates the average increase of salaries over positions over the years. That is for year, you have to compute the average factor by which the average salaries of positions have increased (or decreased) compared to the previous year. For instance, assuming we have only two positions *A* and *B* and in 2014 the average salary for position *A* is 20000 and in 2015 it is 40000, then the average pay for this position has increased by a factor of 2.0.

Question 1.1.4 (5 Points)

Write an SQL query that returns positions (attribute `pos`) which have been held by some members of every party. That is, for each party we can find a member that has held the position.

Question 1.1.5 (2 Points)

Write an SQL Query that returns party members (their **name**) that have not have held any positions with a salary larger than \$200,000.

Question 1.1.6 (2 Points)

Write an SQL query that returns the names of donors that have donated more than 50,000,000 in total.

Question 1.1.7 (5 Points)

Write an SQL query that returns parties whose members have each hold at least one position.

Question 1.1.8 (4 Points)

Write an SQL query that returns pairs of persons (party members) that are members of the same set of parties, e.g., Peter and Alice are both members of JDF, but Alice is also a member of GGK which Bob is not. Thus, the pair (Alice,Bob) should not be returned. Ensure that a pair of users is only returned once (e.g., do not both return (Peter,Bob) and (Bob,Alice)).

Question 1.1.9 (4 Points)

Write a query that returns for each party and each year, the total amount of donations (dollar amount) the party has received in that year plus the two previous years. For instance, the result for a party in 2014 should include all donations made in 2012-2014 to that party.

Question 1.1.10 Optional Bonus Question (10 Bonus Points)

Let's prove that SQL with recursion and arithmetic expressions is turing complete. To do that we are going to write a simulator for turing machines as a recursive SQL query. Recall that a turing machine is a tuple $(Q, \Gamma, b, \Sigma, q_0, F, \delta)$ where Q is a set of states, Γ is tape alphabet, $b \in \Gamma$ is the "blank" symbol, $\Sigma \subseteq \Gamma$ is the input alphabet, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and $\delta : (Q - F) \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function. The state of a turing machine is stored on an infinite tape consisting of cells indexed by position. Each position of the tape stores a symbol from Γ . The initial state of the tape is the input to the turing machine. Only finitely many cells on the initial tape state may be non-blank (unequal to b). The turing machine starts in state q_0 at position 0 of the tape. In each step of its computation it reads the symbol under the current position of the tape and applies δ taking its current state and the symbol read from the tape as input. The result of δ is the new state for the next state, the symbol to be written to the tape at the current position, and the direction to move on the tape (if L then we update $pos = pos - 1$ and $pos = pos + 1$ otherwise). The computation of the turing machine stops once the computation reaches a final state q_f , i.e., a state in the set F .

We encode the turing machine and its tape using the following schema:

- **states(q)** - this relation stores the states of the turing machine, i.e., it encodes Q
- **final(q)** - this relation stores F
- We denote the start state by the string "q0".
- **delta(qin, sin, qnew, sout, dir)** - This relation encodes δ . If $\delta((q, s)) = (q', s', d)$ then this would be encoded as a tuple (q, s, q', s', d) .
- **tape(pos, sym)** - This relation stores the symbol stored at each position of the initial configuration of the tape (the input). Positions that store the blank symbol are omitted from the relation.
- For simplicity assume that the blank symbol is the character 'b' and any symbol from Γ can be encoded as a single character

Write a single **recursive** SQL query that simulates the turing machine. The output of the query should be the final configuration of the tape once the machine reaches one of the final states from F (if the machine halts on this particular input). An example input database and result (tape state) is shown below.

states		final		delta				
q	q0	q	q2	qin	sin	qnew	sout	dir
	q1			q0	0	q1	1	R
	q2			q0	1	q2	1	R
				q0	b	q2	b	R
				q1	0	q1	1	R
				q1	1	q2	1	R
				q1	b	q2	b	R
tape		result						
pos	sym	pos	sym					
0	0	0	1					
1	0	1	1					
2	0	2	1					
3	1	3	1					

Part 1.2 Relational Algebra (Total: 29 Points)

Question 1.2.1 Relational Algebra (2 Points)

Write a relational (bag semantics) algebra expression over the schema from the SQL part (part 1) that returns the names of parties which are *far right*, were established before 1960, and have an endowment that is larger than 2,000,000.

Question 1.2.2 Relational Algebra (4 Points)

Write a relational (bag semantics) algebra expression over the schema from the SQL part (part 1) that returns parties that have gotten more than 3 donations.

Question 1.2.3 Relational Algebra (4 Points)

Write a relational (bag semantics) algebra expression over the schema from the SQL part (part 1) that returns parties that have no members and have not gotten any donations.

Question 1.2.4 SQL \rightarrow Relational Algebra (5 Points)

Translate the SQL query from Question 1.1.1 into relational algebra (bag semantics).

Question 1.2.5 SQL \rightarrow Relational Algebra (5 Points)

Translate the SQL query from question 1.1.2 into relational algebra (bag semantics).

Question 1.2.6 SQL \rightarrow Relational Algebra (5 Points)

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

Question 1.2.7 Equivalences (4 Points)

Consider the following relation schemas (primary key attributes are underlined):

$R(\underline{A}, B)$, $S(\underline{B}, C)$, $T(\underline{C}, D)$, $U(\underline{E}, F, G)$.

Furthermore, assume that $S.B$ is a foreign key to R and $T.C$ is a foreign key to S . Check equivalences that are correct under **bag semantics**. For example $R \bowtie R \equiv R$ should be checked, whereas $R \equiv S$ should not be checked.

- $R - (S - R) \equiv R - (R - S)$
- $R - (R - S) \equiv S - (S - R)$
- $G^{\alpha_{count(*)}}(U) \equiv G^{\alpha_{sum(c)}}(F, G^{\alpha_{count(*) \rightarrow c}}(U))$
- $\pi_C(S \bowtie R) \equiv \pi_C(S)$
- $R \triangleright S \equiv R - (\pi_{A,B}(R \bowtie S))$
- $\delta(\pi_A(R)) \equiv \pi_A(R)$
- $\delta(\pi_A(R \bowtie S)) \equiv \pi_A(R \bowtie S)$
- $\pi_A(R \bowtie S) \equiv \pi_A(R)$

Part 1.3 Index Structures (Total: 30 Points)

Assume that you have the following table:

Item		
SSN	name	age
1	Pete	53
2	Bob	57
44	John	20
43	Joe	18
45	Alice	19
42	Lily	99
88	Gertrud	60
89	Heinz	14

Question 1.3.1 Construction (12 Points)

Create a B+-tree for table *Item* on key *age* 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:

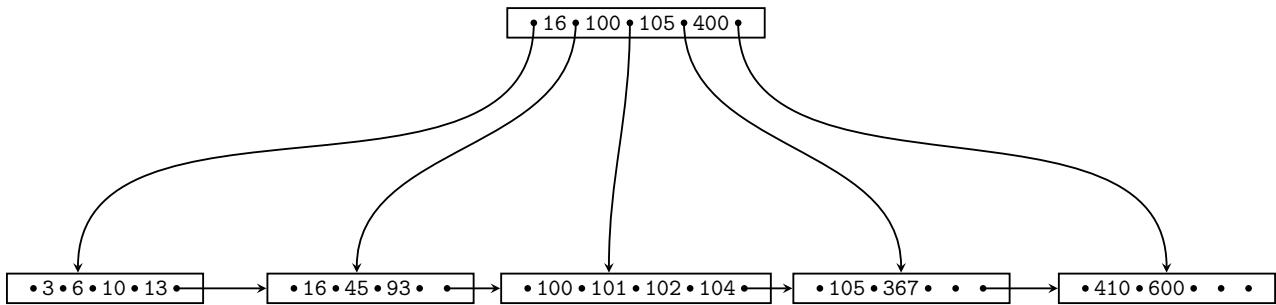
- **Leaf Split:** In case a leaf node needs to be split during insertion and n is even, the left node should get the extra key. E.g, if $n = 2$ and we insert a key 4 into a node $[1,5]$, then the resulting nodes should be $[1,4]$ and $[5]$. For odd values of n we can always evenly split the keys between 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 1.3.2 Operations (10 Points)

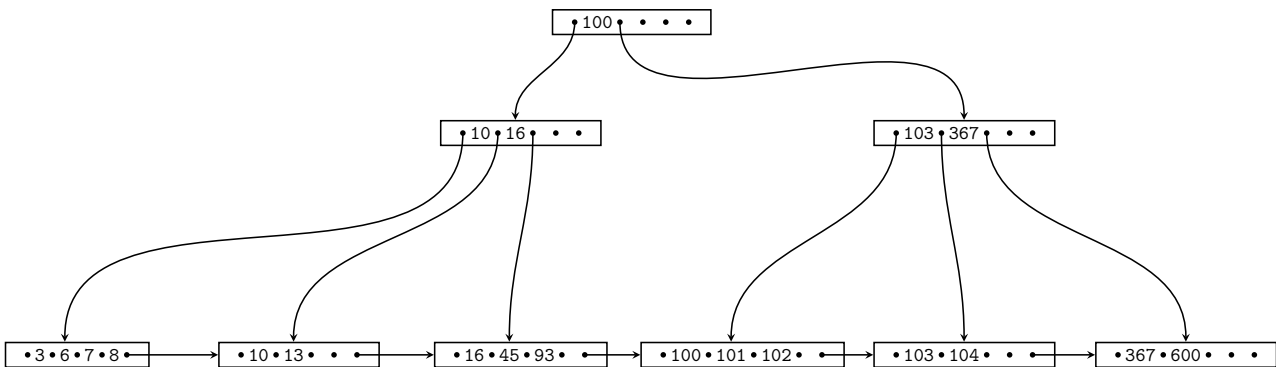
Given is the B+-tree shown below ($n = 4$). Execute the following operations and write down the resulting B+-tree after each operation:

insert(7), insert(8), insert(103), delete(105), delete(410), insert(1)

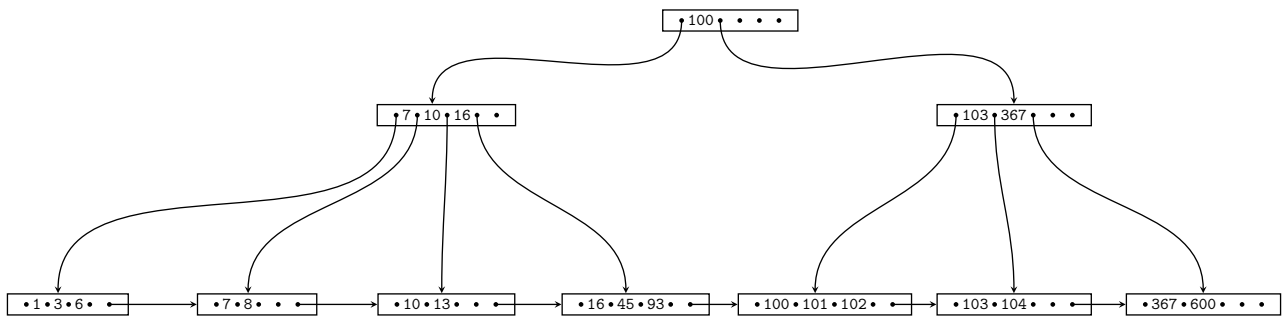
Use the conventions for splitting and merging introduced in the previous question.



Delete 410



Insert 1



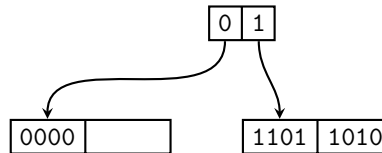
Question 1.3.3 Extensible Hashing (8 Points)

Consider the extensible Hash index shown below that is the result of inserting values 0, 1, and 2. Each page holds two keys. Execute the following operations

`insert(0)`, `insert(5)`, `insert(8)`, `insert(6)`

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

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



Part 1.4 Result Size Estimations (Total: 10 Points)

Consider a table `lecture` with attributes `title`, `campus`, `topic`, `roomSize`, a table `student` with `name`, `major`, `age`, and a table `attendsLecture` with attributes `student`, `lecture`, and `hoursAttended`. `attendsLecture.student` is a foreign key to `student`. Attribute `lecture` of relation `attendsService` is a foreign key to of relation `lecture`. Given are the following statistics:

$$\begin{array}{lll} T(\textit{lecture}) = 200 & T(\textit{student}) = 30,000 & T(\textit{attendsLecture}) = 600,000 \\ V(\textit{lecture}, \textit{title}) = 200 & V(\textit{student}, \textit{name}) = 30,000 & V(\textit{attendsLecture}, \textit{student}) = 25,000 \\ V(\textit{lecture}, \textit{campus}) = 3 & V(\textit{student}, \textit{major}) = 10 & V(\textit{attendsLecture}, \textit{lecture}) = 150 \\ V(\textit{lecture}, \textit{topic}) = 10 & V(\textit{student}, \textit{age}) = 30 & V(\textit{attendsLecture}, \textit{hoursAttended}) = 300 \\ V(\textit{lecture}, \textit{roomSize}) = 20 & & \end{array}$$

Question 1.4.1 Estimate Result Size (3 Points)

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

Question 1.4.2 Estimate Result Size (3 Points)

Estimate the number of result tuples for the query $q = \sigma_{\textit{hoursAttended} > 250}(\textit{attendsLecture})$ using the first assumption presented in class. The minimum and maximum values of attribute `hoursAttended` are 0 and 299.

Question 1.4.3 Estimate Result Size (4 Points)

Estimate the number of result tuples for the query q below using the first assumption presented in class.

$$q = (student \bowtie_{name=student} \sigma_{hoursAttended \neq 0}(attendsLecture) \bowtie_{lecture=title} \sigma_{campus=Mies}(lecture))$$