Name

CWID

Exam
1

Oct 21th, 2020

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
- You will have another **15** minutes for uploading the exam
- There are 4 parts in this exam
  1. SQL
  2. Relational Algebra
  3. Index Structures
  4. I/O Estimation

Part 1 SQL (Total: 32 Points)

Consider the following database schema and instance storing information about orders issued by customers. Orders consist of one or more lineitems — each of which records a particular item that was order with a given quantity. For instance, in the example database the order with `oid` 1 consists of two line time: 2 TVs and 1 coffee table.

customer

ssn	name	creditcard	accbalance
111-111-1111	Aisha	Visa	300.5
444-111-4444	Venkata	Mastercard	200
777-777-8888	Yi	Visa	0

order

oid	date	cust
1	2020-10-18	111-111-1111
2	2020-10-20	111-111-1111
3	2020-10-20	777-777-8888

lineitem

oid	linenumber	item	quantity
1	1	1	2
1	2	3	1
2	1	1	5

item

iid	desc	price
1	TV	400
2	bookshelf	120
3	coffee table	250

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, **ssn** is the primary key of relation **customer**.
- The attribute **cust** of relation **orders** is a foreign key to relation **customer**.
- The attribute **oid** of relation **lineitem** is a foreign key to relation **orders**.
- The attribute **item** of relation **lineitem** is a foreign key to relation **items**.

Question 1.1 (6 Points)

Write an SQL query that computes the total cost of each order (return two columns: `oid` and the total cost). The cost of a lineitem is computed as the price of the ordered item multiplied by the quantity. The total cost of an order is the sum of the cost of its lineitems. For instance, for the order with `oid` 1 from the example database, the total cost is:

$$400 \cdot 2 + 250 \cdot 1 = 1,050$$

Solution

```sql
SELECT oid, sum(price * quantity) AS totalcost
FROM lineitem l, items i
WHERE l.item = i.iid
GROUP BY oid;
```

Question 1.2 (8 Points)

Write a query that returns the ssn and name of customers which have not ordered any items that cost more than $300 (the cost of an item is stored in its `price` attribute). Make sure that each such customer is only returned once.

Solution

```sql
SELECT ssn, name
FROM customers c
WHERE ssn NOT IN (SELECT cust
                  FROM order o, lineitems l, items i
                  WHERE o.oid = l.oid
                        AND l.item = i.item
                        AND i.price > 300);
```

Question 1.3 (9 Points)

Write an SQL query that returns for each creditcard type (attribute `creditcard` of relation `customer`) and the item that was ordered the most with this creditcard type. Note that the quantity of lineitems should be taken into account for calculating the number of times an item was ordered with a particular creditcard type. For instance, in the example database, item 1 (a TV) was ordered seven time (the first lineitem of order 1 is item TV with quantity 2 and the first lineitem of order 2 is item TV with quantity 5).

Solution

```
WITH timesordered AS (
    SELECT iid, creditcard, sum(quantity) AS numordered
    FROM customer c, order o, lineitems l, items i
    WHERE c.ssn = o.cust AND o.oid = l.oid AND l.item = i.iid
    GROUP BY iid, creditcard
    ),
maxordered AS (
    SELECT max(numordered) AS maxordered, creditcard
    FROM timesordered
    GROUP BY creditcard
    )
SELECT creditcard, iid
FROM timesordered t, maxordered m
WHERE t.numordered = m.maxordered
      AND t.creditcard = m.creditcard
```

Question 1.4 (9 Points)

Write a SQL query that returns the name of the customer with the greatest total order cost. The total order cost is the sum of the cost of all orders from this customer. The cost of an order is computed as explained in Question 1.1.

Solution

```sql
WITH custcost AS (
    SELECT name, sum(price * quantity) AS ttlcost
    FROM customer c, order o, lineitem l, item i
    WHERE c.ssn = o.cust AND o.oid = l.oid AND l.item = i.iid
    ),
maxcust AS (
    SELECT max(ttlcost) AS maxcost
    FROM custcost
    )
SELECT name
FROM custcost WHERE ttlcost = (SELECT maxcost FROM maxcust)
```

Part 2 Relational Algebra (Total: 26 Points)

Question 2.1 Relational Algebra (8 Points)

Write a relational algebra expression over the schema from the SQL part that returns for the number of orders which contain both a TV and a bookshelf. Use the **bag semantics** version of relational algebra.

Solution

$$
\begin{aligned}
linewithitem &= \pi_{oid,desc}(lineitem \bowtie_{item=iid} item) \\
tvandtable &= \delta(\pi_{oid}(\sigma_{desc=TV}(linewithitem)) \bowtie \pi_{oid}(\sigma_{desc=TV}(linewithitem))) \\
q &= \gamma_{count(*)}(tvandtable)
\end{aligned}
$$

Question 2.2 Relational Algebra (6 Points)

Write a relational algebra expression over the schema from the SQL part that returns items which have not been ordered yet (there is no order with a lineitem with that item). Use the **bag semantics** version of relational algebra.

Solution

$$
\begin{aligned}
ordered &= \pi_{iid}(item \bowtie_{iid=item} lineitem) \\
q &= \pi_{iid}(item) - ordered
\end{aligned}
$$

Question 2.3 Relational Algebra (12 Points)

Write a relational algebra expression over the schema from the SQL part that returns customers (their `ssn`) whose account balance (attribute `accbalance`) is larger than the cost of each of the customers orders. For example, if a customer has an account balance of \$300 and has issued two orders with cost \$250 and \$50, then this customer should be returned because 300 > 250 and 300 > 50. Use the **bag semantics** version of relational algebra.

Solution

$$\begin{aligned}
bal &= \pi_{ssn,accbalance}(customer) \\
ocost &= \gamma_{oid;sum(lcost)\rightarrow ocost}(\pi_{oid,price\times quantity\rightarrow lcost}(lineitem \bowtie_{item=iit} item)) \\
maxorder &= \gamma_{ssn;max(ocost)\rightarrow maxcost}(ocost \bowtie_{oid=oid} orders) \\
q &= \pi_{ssn}(bal \bowtie_{ssn=ssn \wedge accbalance>maxcost} maxorder)
\end{aligned}$$

Part 3 Index Structures (Total: 24 Points)

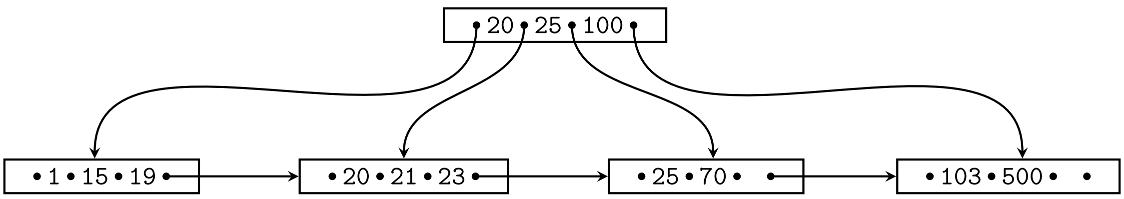
Question 3.1 B+-tree Operations (24 Points)

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

insert(2),insert(3),insert(4),delete(103)

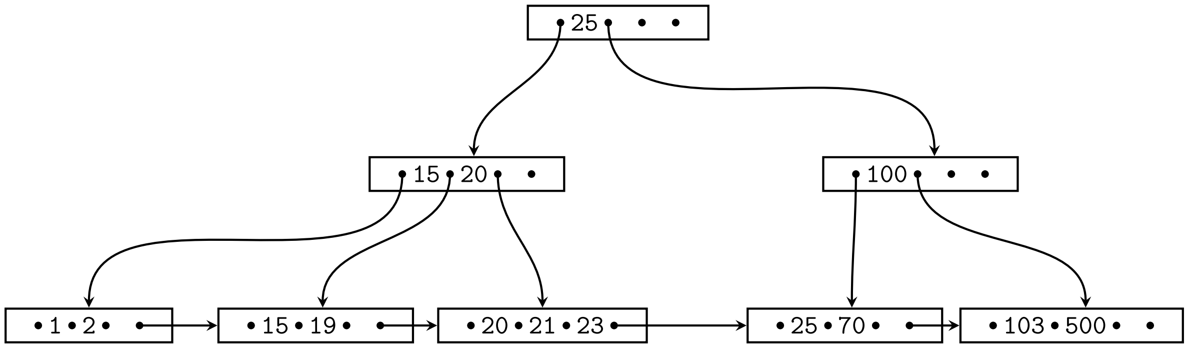
When splitting or merging nodes follow these conventions:

- **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.

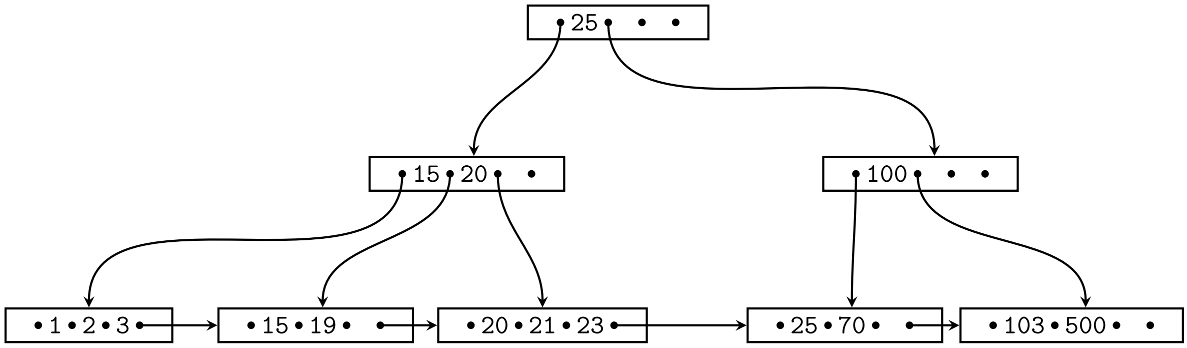


Solution

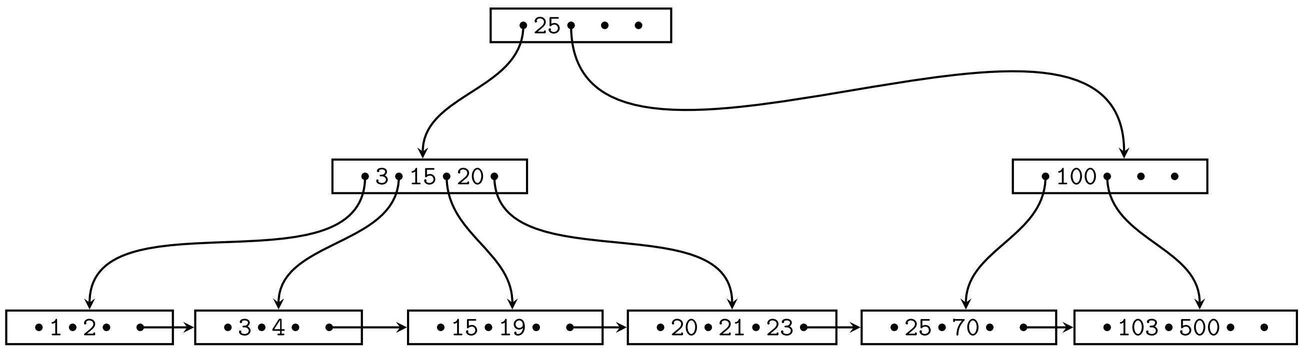
insert(2)



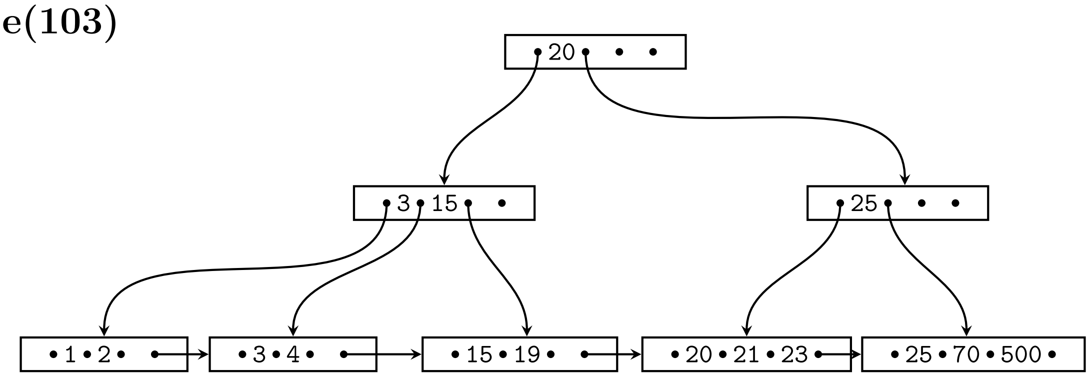
insert(3)



insert(4)



delete(103)



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 $B(R) = 100,000$ and $B(S) = 400,000$. You have $M = 101$ 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 outer. We get have $1,000$ chunks of size 100. Thus, we get $1,000 \times (100 + B(S)) = 400,100,000$ I/Os.
Merge-join:
Relation R can be sorted with two merge phases resulting in $3 * 2 * B(R) = 600,000$ I/Os merging 10 runs in the last phase. Relation S requires two merge phases, merging 40 runs in the last phase: $3 \times 2 \times B(S) = 2,400,000$ I/Os. The last merge phase of relation S can be combined with the last merge phase of R ($10 + 40 = 50 \leq 100$ 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 $7 * B(R) + 7 * B(S) = 3,500,000$. If we execute the merge-join during the last merge phases we get $5 * B(R) + 5 * B(S) = 2,500,000$.
Hash-join:
We need two partitioning phases for the partitions of relation R to fit into memory. Thus, the hash-join requires $5 * (B(R) + B(S)) = 2,500,000$ I/Os.

Question 4.2 External Sorting (6 Points)

Consider a relation R with $B(R) = 2,000,000$. Assume that $M = 401$ 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 $2 \times (1 + \lceil \log_{M-1}(\frac{B(R)}{M}) \rceil) \times B(R) = 2 \times 3 \times 2,000,000 = 12,000,000$ I/Os.