Name

CWID

Exam
2

May 9th, 2014

CS525 - Final Exam
Solutions

Please leave this empty! 1 2 3 4 5 6 7

Sum

Instructions

- Things that you are **not** allowed to use
  - Personal notes
  - Textbook
  - Printed lecture notes
  - Phone
  - Calculator
- The exam is **120** minutes long
- Multiple choice questions are graded in the following way: You get points for correct answers and points subtracted for wrong answers. You do not have to answer every subquestion. 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. If you answer 3 questions correct, 1 incorrect, and do not answer 2, then you get $3 - 1 = 2$ points. ...
- For your convenience the number of points for each part and questions are shown in parenthesis.
- There are 7 parts in this exam (100 points total)
  1. Disk Organization and Buffering (12)
  2. SQL (20)
  3. Relational Algebra (9)
  4. Index Structures (20)
  5. I/O Estimation (15)
  6. Result Size Estimation (9)
  7. Schedules (15)

Part 1 Disk Organization and Buffering (Total: 12 Points)

Question 1.1 Page Replacement Clock (12 Points)

Consider a buffer pool with 3 pages using the **Clock** page replacement strategy. Initially the buffer pool is in the state shown below. We use the following notation ${}^{flag}[page]^{dirty}_{fix}$ to denote the state of each buffer frame. *page* is the number of the page in the frame, fix is its fix count, *dirty* is indicating with an Asterix that the page is dirty, and $flag$ is the reference bit used by the Clock algorithm. E.g., ${}^{1}[5]^{*}_{2}$ denotes that the frame stores page 5 with a fix count 2, that the page is dirty, and that the reference bit is set to 1. Recall that Clock uses a pointer S that points to the current page frame (the one to be checked for replacement next). The page frame S is pointing to is shown in bold. In your solution draw an arrow to the page frame that S is pointing to.

Current Buffer State

$$\mathbf{{}^{0}[10]^{*}_{0}}\ {}^{0}[5]_{0}\ {}^{1}[3]_{2}$$

Execute the following requests and write down state of the buffer pool after each request.

- p stands for pin
- u for unpin
- d for marking a page as dirty

```
u(3),p(1),p(2),u(3),u(1),p(17)
```

Solution

u(3)

$$\mathbf{^{0}[10]^{*}_{0}}\ {}^{0}[5]_{0}\ {}^{1}[3]_{1}$$

p(1)

$${}^{1}[1]_{1}\ \mathbf{^{0}[5]_{0}}\ {}^{1}[3]_{1}$$

p(2)

$${}^{1}[1]_{1}\ {}^{1}[2]_{1}\ \mathbf{^{1}[3]_{1}}$$

u(3)

$${}^{1}[1]_{1}\ {}^{1}[2]_{1}\ \mathbf{^{1}[3]_{0}}$$

u(1)

$${}^{1}[1]_{0}\ {}^{1}[2]_{1}\ \mathbf{^{1}[3]_{0}}$$

p(17)

$$\mathbf{^{0}[1]_{0}}\ {}^{0}[2]_{1}\ {}^{1}[17]_{1}$$

Part 2 SQL (Total: 20 Points)

Consider the following database schema and instance:

stock

itemId	suppId	quantity
1	1	9003
3	1	10
3	2	17

item

itemId	category	price
1	Houseware	15
2	Houseware	55
3	Entertainment	67

supplier

suppId	name	country
1	DistriAll	USA
2	UPSE	Canada
3	Alder	Mexico

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, **itemId** is the primary key of relation **item**.
- The attribute **itemId** of relation **stock** is a foreign key to the attribute **itemId** of relation **item**.
- The attribute **suppId** of relation **stock** is a foreign key to the attribute **suppId** of relation **supplier**.

Question 2.1 (3 Points)

Write an SQL query that returns the names and country of suppliers which have item 13 in stock with an available `quantity` of more than 1000 items. The query should return the results sorted by country in alphabetical order.

Solution

```
SELECT DISTINCT name, country
FROM stock s NATURAL JOIN supplier u
WHERE s.itemId = 13
ORDER BY country;
```

We also except without `DISTINCT`

Question 2.2 (3 Points)

Write an SQL statement that returns for each supplier the name and the total net worth of its stock (calculated using the price of items and the quantity of items in the stock).

Solution

Some students did assume that the supplier name is not necessary unique. In this case one has to group on `suppId`. We accept both solutions.

```
SELECT name, networth
FROM supplier s
     NATURAL JOIN
     (SELECT suppId, sum(quantity * price) AS networth
     FROM stock s NATURAL JOIN item i NATURAL JOIN supplier s
     GROUP BY suppId) net

SELECT name, sum(quantity * price) AS networth
FROM stock s NATURAL JOIN item i NATURAL JOIN supplier s
GROUP BY name
```

Question 2.3 (4 Points)

Write an SQL query that returns the total networth of suppliers per country (the total value of items computed as quantity times price they have in their stock) ordered by the total in descending order.

Solution

```
SELECT country, sum(quantity * price) AS ttlNetworth
FROM stock s NATURAL JOIN item i NATURAL JOIN supplier s
GROUP BY country
ORDER BY sum(quantity * price) DESC;
```

Question 2.4 (4 Points)

Write an SQL query that returns items (`itemId`) which are in the stock of at least two suppliers with a quantity over 1000.

Solution

```
SELECT itemId
FROM stock
WHERE quantity > 1000
GROUP BY itemId
HAVING count(*) > 1
```

Question 2.5 (6 Points)

Write an SQL query that returns the names of suppliers which have one item in stock with at least 10 times the quantity of another item they have in stock. For example, for the instance shown this query would return the name of supplier 1, because this supplier has an item (`itemId` 1) in stock that has more than 10 times the quantity of another item (`itemId` 3) from the same supplier.

Solution

For one supplier there may be many such combinations. However, it is sufficient to just compare the items with minimum and maximum quantity in the stock of this supplier.

```
SELECT name
FROM supplier s
     NATURAL JOIN
     (SELECT suppId, min(quantity) AS minQ, max(quantity) AS maxQ
     FROM stock
     GROUP BY suppId
     HAVING max(quantity) > 10 * min(quantity)) AS span
```

Part 3 Relational Algebra (Total: 9 Points)

Question 3.1 Relational Algebra (3 Points)

Write a relational algebra expression that returns the average price of items per category. Use the **set semantics** version of relational algebra.

Solution

$$_{category}\alpha_{avg(price)}(item)$$

Question 3.2 Relational Algebra (6 Points)

Write a relational algebra expression over the schema from the SQL part (part 2) that returns suppliers (their names) that do not have any item of category 'Houseware' in their stock. Use the **set semantics** version of relational algebra.

Solution

$$houseStock = \pi_{name}(\sigma_{category='Houseware'}(item) \bowtie stock \bowtie supplier)$$
$$q = \pi_{name}(supplier) - houseStock$$

Get the suppliers with 'Houseware' items in stock and then remove these suppliers from the list of suppliers.

Part 4 Index Structures (Total: 20 Points)

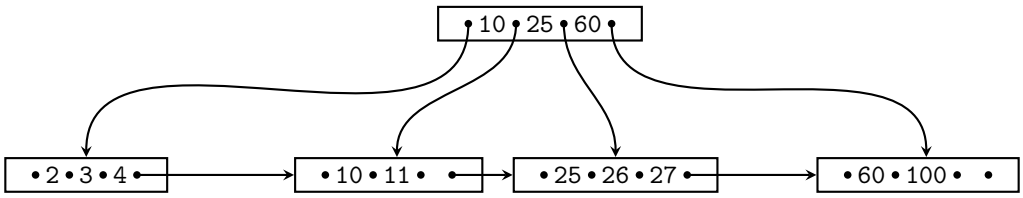
Question 4.1 B+-tree Operations (20 Points)

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

insert(102), delete(3), insert(62), delete(4), delete(25)

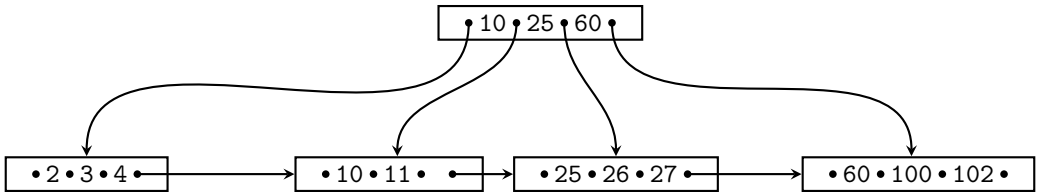
When splitting or merging nodes follow these conventions:

- **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.

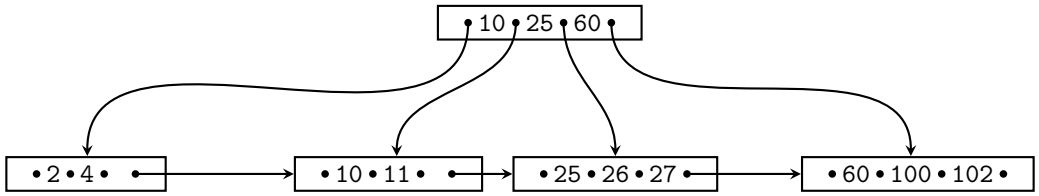


Solution

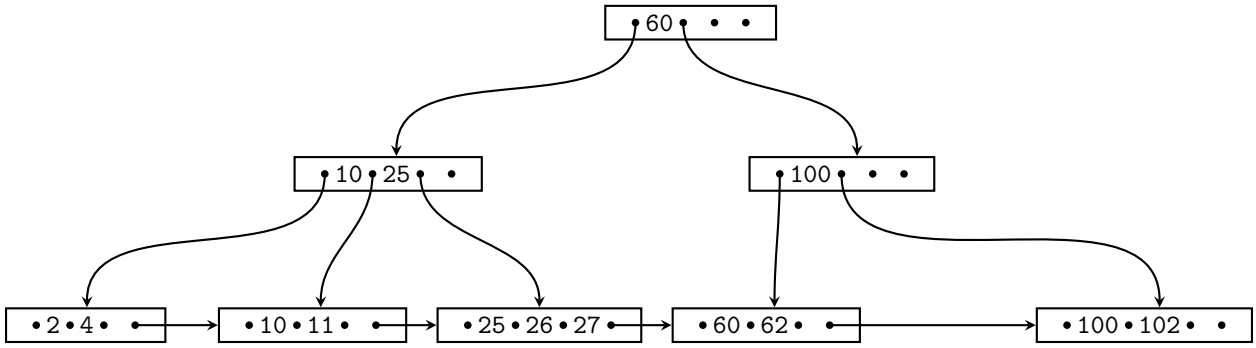
insert(102)



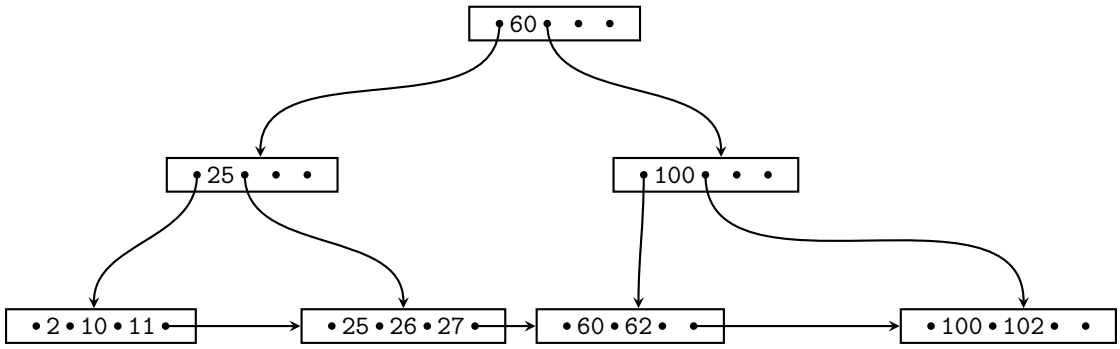
delete(3)



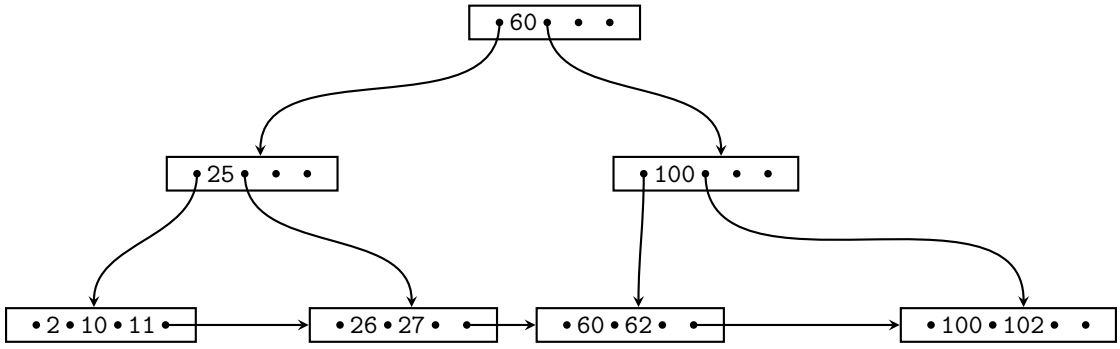
insert(62)



delete(4)



delete(25)



Part 5 I/O Cost Estimation (Total: 15 Points)

Question 5.1 External Sorting (3 Points)

You have $M = 31$ memory pages available and should sort a relation R with $B(R) = 90,000$ blocks. Estimate the number of I/Os necessary to sort R using the external merge sort algorithm introduced in class.

Solution

$$\begin{aligned} IO &= 2 \cdot B(R) \cdot (1 + \lceil log_{M-1}(\frac{B(R)}{M}) \rceil) \\ &= 2 \cdot 90,000 \cdot (1 + 3) \\ &= 720,000 \end{aligned}$$

Question 5.2 External Sorting (3 Points)

You have $M = 3$ memory pages available and should sort a relation R with $B(R) = 110$ blocks. Estimate the number of I/Os necessary to sort R using the external merge sort algorithm introduced in class.

Solution

$$\begin{aligned} IO &= 2 \cdot B(R) \cdot (1 + \lceil log_{M-1}(\frac{B(R)}{M}) \rceil) \\ &= 2 \cdot 110 \cdot (1 + 6) \\ &= 1,540 \end{aligned}$$

Question 5.3 I/O Cost Estimation (9 = 3 + 3 + 3 Points)

Consider two relations R and S with $B(R) = 800,000$ and $B(S) = 300,000$. You have $M = 1,001$ memory pages available. Compute the minimum 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 evenly distributes keys across buckets. Justify you result by showing the I/O cost estimation for each join method.

Solution

- **BNL**: S is the smaller relation.
  $\lceil \frac{B(S)}{M-1} \rceil \cdot [B(R) + min(B(S), (M-1))] = 300 \cdot [800,000 + 1000] = 240,300,000$ I/Os
- **MJ**: We can generate sorted runs of size $1,000$ that means the number of sorted runs from R and S (800 and 300) is not low enough to keep one page from each run in memory. We need 1 merge pass for both R and S the sort and cannot execute the last merge phase and join in one pass. $5 \cdot (B(R) + B(S)) = 5 \cdot (800,000 + 300,000) = 5,500,000$ I/Os.
- **HJ**: After one partition phase the size of the partitions for S (300 pages) is small enough to fit one partition into memory. Build in memory hash table of each partition of S (assume that it fits) and stream a partition of R once probing the hash table. $(2 + 1) \cdot (B(R) + B(S)) = 3,300,000$ I/Os.

We would HJ, because it has the lowest I/O requirements.

Part 6 Result Size Estimations (Total: 9 Points)

Consider the tables *stock* with attributes *itemId*, *suppId*, *quantity* and a table *item* with *itemId*, *category*, *price*. Attribute *itemId* of relation stock is a foreign key to attribute *itemId* of relation item. Given are the following statistics:

$$T(stock) = 60,000 \qquad T(item) = 1,000$$
$$V(stock, itemId) = 500 \qquad V(item, itemId) = 1,000$$
$$V(stock, suppId) = 50 \qquad V(item, category) = 20$$
$$V(stock, quantity) = 3,000 \qquad V(item, price) = 500$$

Question 6.1 Estimate Result Size (3 Points)

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

Solution

$$T(q) = \frac{T(item)}{V(item, category)} = \frac{1,000}{20} = 50$$

Question 6.2 Estimate Result Size (6 Points)

Estimate the number of result tuples for the query $q = \sigma_{quantity=100}(item \bowtie stock)$ using the first assumption presented in class.

Solution

We first need to estimate the number of expected join results and the number of distinct values in attribute *quantity* in the join result.

$$T(item \bowtie stock) = \frac{T(item) \cdot T(stock)}{max(V(item, itemId), V(stock, itemId))} = \frac{1,000 \cdot 60,000}{1,000} = 60,000$$

According to the assumption from class: $V(item \bowtie stock, quantity) = 3,000$). Thus,

$$T(q) = \frac{T(item \bowtie stock)}{V(item \bowtie stock, quantity)} = \frac{60,000}{3,000} = 20$$

Part 7 Schedules (Total: 15 Points)

Question 7.1 Schedule Classes (15 = 5 + 5 + 5 Points)

Indicate which of the following schedules belong to which class. Recall transaction operations are modelled as follows:

$w_1(A)$	transaction 1 wrote item A
$r_1(A)$	transaction 1 read item A
c_1	transaction 1 commits
a_1	transaction 1 aborts

$S_1 = r_2(A), r_1(B), w_1(B), w_3(B), w_2(A), r_4(E), w_4(E), r_1(C), c_1, c_3, r_2(C), w_2(C), w_2(B), r_4(A), c_2, c_4$
$S_2 = r_2(C), w_3(C), w_4(B), w_3(A), c_3, r_4(A), c_4, r_1(B), c_1, w_2(B), c_2$
$S_3 = w_2(A), w_1(A), w_3(B), c_3, w_2(B), w_2(B), c_2, w_1(B), c_1$

S_1 is recoverable	❑ no	■ yes
S_1 is cascade-less	■ no	❑ yes
S_1 is strict	■ no	❑ yes
S_1 is conflict-serializable	❑ no	■ yes
S_1 is 2PL	❑ no	■ yes

S_2 is recoverable	❑ no	■ yes
S_2 is cascade-less	❑ no	■ yes
S_2 is strict	❑ no	■ yes
S_2 is conflict-serializable	■ no	❑ yes
S_2 is 2PL	■ no	❑ yes

S_3 is recoverable	❑ no	■ yes
S_3 is cascade-less	❑ no	■ yes
S_3 is strict	■ no	❑ yes
S_3 is conflict-serializable	❑ no	■ yes
S_3 is 2PL	■ no	❑ yes