

Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

**6.893 Database Systems: Fall 2004**

## Quiz 2

There are 14 questions and 12 pages in this quiz booklet. To receive credit for a question, answer it according to the instructions given. *You can receive partial credit on questions if you circle some of the correct answers, but we may subtract points for circling a wrong answer.* You have **80 minutes** to answer the questions.

**Write your name on this cover sheet AND at the bottom of each page of this booklet.**

Some questions may be harder than others. Attack them in the order that allows you to make the most progress. If you find a question ambiguous, be sure to write down any assumptions you make. Be neat. If we can't understand your answer, we can't give you credit!

**THIS IS AN OPEN BOOK, OPEN NOTES QUIZ.  
NO PHONES, NO LAPTOPS, NO PDAS, ETC.**

*Do not write in the boxes below*

1-3 (xx/24)	4-6 (xx/24)	7-9 (xx/22)	10-13 (xx/16)	14 (xx/14)	Total (xx/100)

**Name: Solutions**

## I Multiple Choice

1. [8 points]: The purpose of the two-phase commit protocol is to ensure that:  
(Circle ALL that apply)

- A. No additional sites in a distributed database start running actions as a part of a transaction  $t$  after one site has completed all of the actions it is executing as a part of  $t$  (i.e., the “growing” phase of  $t$  completely precedes the “shrinking” phase of  $t$ .)  
*The notion of a “growing” and “shrinking” phase are not part of two-phase commit.*
- B. Deadlock does not occur.  
*Two-phase commit plays no role in deadlock detection.*
- C. Transactions in a distributed database provide all-or-nothing atomicity, just like transactions in a single-site database.  
*This is true.*
- D. All sites in a distributed database commit at exactly the same instant.  
*This is not true. Though two phase commit will ensure that all sites eventually make the same decision to commit or abort, it cannot ensure that they all do this simultaneously. This is “two generals” or “Byzantine generals” problem.*
- E. If one site ABORTs while processing part of a distributed transaction  $t$ , all other sites processing part of  $t$  will also ABORT  $t$ .  
*This is true, as the statement is equivalent to C.*

2. [8 points]: The paper “Combining Systems and Databases: A Search Engine Perspective” discusses ways in which search engines and databases are alike. Which of the following **ARE NOT** similarities between search engines and database systems:

(Circle ALL that apply)

- A. Both perform a type of query planning and optimization.  
*This is true – search engines have a simple kind of query plan and perform several rewrite-like optimizations.*
- B. Both web search and database querying are highly parallelizable.  
*This is true – both are amenable to pipelined and partitioned parallelism.*
- C. Both need to be able to support ad-hoc read-write transactions from end-users.  
*This is false – search engines are not generally updated by end users.*
- D. Both probably use some form of two-phase locking.  
*This is false – since search engines aren’t updated, they probably don’t use two phase locking.*
- E. Both typically use a score metric to prioritize the order in which results are presented to a user.  
*This is false – database systems usually return all results to users, whereas search engines only return a subset prioritized by the score metric.*

3. [8 points]: The CONTROL system proposes that query processors should provide *online* result delivery rather than waiting until a query completes to provide *batched* answers. The following statements are true about the implementation of online query processing described in the CONTROL paper:

(Circle ALL that apply)

- A. Access methods optimized for online query processing do not perform as well as access methods optimized for batch query processing when considering the time-to-completion of the entire (non-approximate) query.

*This is true. Access methods in CONTROL are like clustered indices that produce results in a random order; using a traditional index that produces data in a particular order or that directly satisfies some selection or join predicate could substantially decrease total query completion time.*

- B. It is possible to implement the online query processing techniques in this paper without making modifications to database clients.

*This is false. Generally, implementing CONTROL algorithms requires changes to user interfaces, since applications must be able to deal with results that change over time.*

- C. The reorder operator usually has a lower performance overhead than the index stride method for result prioritization.

*This is true. The reorder operator generally performs sequential I/O, whereas the index stride method general performs random I/O.*

- D. The reorder operator guarantees that the distribution of delivered results will closely match the requested distribution (e.g., if the user sets the priority of group A to be ten times the priority of group B, approximately ten times as many A tuples will arrive as B tuples.)

*This is false. The reorder operator cannot guarantee that the generated results will closely match the requested distribution if the user requests a high priority for very rare results.*

- E. All of the above.

*False, since B and D are false.*

4. [8 points]: Finding a document or set of documents that satisfy an XML query is often more computationally complex than finding a set of records that satisfy a relational query because:

(Circle ALL that apply)

A. XML documents are often built “schema last”.

*False. Though XML documents are built “schema last”, this does not increase the computational complexity of query processing.*

B. Idrefs, long paths, and path expressions with wildcards are more complex to process than the flat tuples and boolean predicates in relational systems.

*True – these constructs make query processing over XML documents more difficult.*

C. XML query languages like XQuery are very complicated, making them hard to optimize.

*True – XQuery in particular is very complicated and less amenable to traditional relational optimizations than a language like SQL.*

D. XML must be able to pass through firewalls.

*Though it may be true that “XML can pass through firewalls” this has no bearing on the ability of query processors to process it.*

E. All of the above.

*Not true, since A and D are false.*

**5. [8 points]:** The NiagaraCQ paper proposes a way to simultaneously execute multiple queries which share selection and / or join predicates. Suppose you are given a set of queries with identical signatures that consist of a join and a selection. Assume that Niagara groups selections and shares joins as aggressively as possible and that individual selection predicates are low-cost. When the techniques proposed in the paper are applied to this set of queries, the following statements are true:

**(Circle ALL that apply)**

**A.** It is always better to push a grouped selection before a (possibly shared) join.

*False. If the selection has a high selectivity, it may be better to perform the shared join first, rather than computing a number of separate joins after the shared selection.*

**B.** It is always better to push a shared join before a (possibly grouped) selection.

*False. If the selection has a very low selectivity, it may be better to filter the results before performing any joins.*

**C.** Pushing a shared join before a (possibly grouped) selection is a good idea when the selectivity of the join is low.

*True. If very few of results pass through the join, and the grouped selection has high selectivity, then it will be better to perform the join first rather than performing it many times at the output of the grouped selection.*

**D.** Pushing a grouped selection before a (possibly shared) join will never perform worse than running all of the queries independently.

*This answer was discounted. The paper seems to suggest this is true, but the wording of the question is such that there are certain situations – such as when the selectivity of the filter is 1 for all queries – were you could imagine that independent queries with the join ordered first would outperform the grouped approach.*

**E.** Pushing a shared join before a (possibly grouped) selection will never perform worse than running all of the queries independently.

*False. The additional work of performing the join before the results are filtered can clearly make this a more expensive strategy than running the queries independently.*

**6. [8 points]:** The following are likely to contribute significantly to the runtime *overhead* of an eddy, assuming the original eddy specification described in the paper “Eddies: Continuously Adaptive Query Processing.” Assume that the selectivity of all joins is  $< 1$ .

(Circle ALL that apply)

- A.** Consultation of the routing policy on every tuple after each operator to decide where it should be routed next.  
*True. The major overhead in eddies is due to consultation of the routing policy.*
- B.** Use of ‘backpressure’ to measure the cost of operators.  
*False. Backpressure is a mechanism for measuring the cost of an operator, and incurs no overhead.*
- C.** Use of pipelined join algorithms.  
*False. Generally speaking, pipelined join algorithms are no less efficient than traditional join algorithms.*
- D.** Use of the river workflow system.  
*False. There is no reason to believe that the river system – which is based on the same workflow idea as many other database architectures – is a source of overhead.*
- E.** The use of a join algorithm with many *moments of symmetry*.  
*False. Common join algorithms with many moments of symmetry are quite efficient.*

## II Short Answers

7. [6 points]: List three situations in which adaptive query processing is beneficial. Limit your answer to one sentence per situation.

1. *In situations where statistics are inaccurate or unavailable – e.g., query processing over a remotely stored web site.*
2. *In situations where statistics change over the lifetime of the query – either because the query is long running (e.g., continuous) or stats are volatile.*
3. *In situations with variable or unpredictable resources e.g., query processing on a cluster of shared machines.*

8. [4 points]: Describe three queries or classes of queries that a streaming or continuous query processor can answer that a traditional database could not. You may provide specific example queries (e.g., “compute the average salary of all employees”) or general application domains (e.g., “OLTP-style workloads”). Limit your answer to 1 sentence per query.

1. *Network monitoring queries – e.g., “alert me whenever the request queue on my server exceeds a certain length”.*
2. *Stock price monitoring queries – e.g., “sell AAPL when it falls below \$50.”*
3. *Sensor network monitoring queries – e.g., “report the average humidity in room 938 every 10 seconds”.*

9. [12 points]: The eddies paper describes a simple routing policy based on the idea of *tickets* and *backpressure*. This policy cannot account for some of the same optimization issues as a conventional cost-based optimizer equipped with good statistics (e.g., multi-dimensional histograms.) Describe, in one or two sentences each, two types of queries which the ticket-based routing scheme would have trouble executing efficiently relative to a cost-based optimizer. Also include your reasoning for why the ticket-based scheme would perform poorly.

1. *The Eddy ticket based routing scheme does not deal with correlations between operators properly; e.g., if two selections A and B have approximately the same selectivity, but all 90% of tuples that pass A also pass B, an Eddy will not properly model this. Traditional query optimizers/processors can deal with correlations via multi-dimensional histograms.*
2. *The ticket based routing scheme doesn't properly order join operators with selectivity > 1, since such operators would receive negative or 0 tickets.*

### III Aurora Operators

You are asked to determine the behavior of several queries in the Aurora database system. These queries use sensors and smart badges on employees to detect conditions and activity inside a building. You are given two streams:

The first stream, Stream A, consists of sensor data representing the current light and temperature in a number of rooms in a building. The schema for Stream A is:

```
Stream A:(ts timestamp, light int, temperature int, room char);
```

The second stream, Stream B, consists of a stream of building occupancy information, generated by the smart badges as employees move throughout the building. The schema for Stream B is:

```
Stream B:(ts timestamp, int emp_id, room char);
```

Suppose you are given the following data stream fragments.

Stream A	Stream B
1:00 PM, 105 lux, 22°C, a	1:02 PM, 1, a
1:14 PM, 110 lux, 23°C, b	1:04 PM, 2, b
1:24 PM, 120 lux, 24°C, a	1:06 PM, 2, a
1:34 PM, 117 lux, 21°C, c	1:11 PM, 1, a
1:23 PM, 103 lux, 35°C, b	1:24 PM, 3, b
1:31 PM, 97 lux, 33°C, a	1:27 PM, 2, c
1:31 PM, 45 lux, 21°C, c	1:31 PM, 1, c
1:34 PM, 22 lux, 22°C, a	1:34 PM, 2, a
1:42 PM, 3 lux, 23°C, b	1:39 PM, 2, b
1:21 PM, 111 lux, 26°C, b	1:41 PM, 1, b
	1:43 PM, 3, c

For each of the questions below, show the resulting output that the given Aurora operators would produce with the specified orderings and windowing arguments. Use the description of the Aurora operators given in Section 5 of the paper “Aurora: a new model and architecture for data stream management”. Assume that the current time is 1:45 PM, and that queries begin running at 1:00 PM.

For example, if the operator was `Filter(light < 100 lux)(A)`, the result would be:

```
1:30 PM, 97 lux, 33°C, a
1:30 PM, 45 lux, 21°C, c
1:35 PM, 22 lux, 22°C, a
1:42 PM, 3 lux, 23°C, b
```

For your reference, the last page of this exam contains a copy of the streams, which you may detach.

**10. [4 points]:**

BSort(Assuming Order(On ts, Slack 2))(A)

1:00 PM, 105 lux, 22°C, a  
1:14 PM, 110 lux, 23°C, b  
1:23 PM, 103 lux, 35°C, b  
1:24 PM, 120 lux, 24°C, a  
1:34 PM, 117 lux, 21°C, c  
1:31 PM, 97 lux, 33°C, a  
1:31 PM, 45 lux, 21°C, c  
1:21 PM, 111 lux, 26°C, b  
1:34 PM, 22 lux, 22°C, a  
1:42 PM, 3 lux, 23°C, b

**11. [4 points]:**

BSort(Assuming Order(On ts, Slack 2, Group By emp\_id))(B)

1:02 PM, 1, a  
1:04 PM, 2, b  
1:06 PM, 2, a  
1:11 PM, 1, a  
1:24 PM, 3, b  
1:27 PM, 2, c  
1:31 PM, 1, c  
1:34 PM, 2, a  
1:39 PM, 2, b  
1:41 PM, 1, b  
1:43 PM, 3, c

**12. [4 points]:**

Aggregate(MAX(temp), COUNT(temp))

Assuming Order(On ts, Slack 2),

Size 10 mins,

Advance 5 mins)

(A)

(Please provide both the maximum and count for each time.)

Time	MAX(temp)	COUNT(temp)
1:00 - 1:10	22	1
1:05 - 1:15	23	1
1:10 - 1:20	23	1
1:15 - 1:25	35	2
1:20 - 1:30	35	2
1:25 - 1:35	33	4
1:30 - 1:40	33	4
1:35 - 1:45	23	1

**13. [4 points]:**

Join(A.room = B.room,

Size 10 mins,

Left Assuming Order(On ts, Slack 2),

Right Assuming Order(On ts, Slack 2))

(A, B)

Time(A)	Time(B)	Room
1:00	1:02	a
1:00	1:06	a
1:14	1:04	b
1:14	1:24	b
1:23	1:24	b
1:24	1:34	a
1:31	1:34	a
1:31	1:27	c
1:31	1:31	c
1:34	1:27	c
1:34	1:31	c
1:34	1:43	c
1:34	1:34	a
1:42	1:39	b
1:42	1:41	b

### IV AQuery

The AQuery system includes a number of optimizations designed to improve the performance of queries over ordered data. Suppose we have a table of orders, *O* and a table of products *P*, with the following schema:

```

Orders (oid int primary key, order_date timestamp, shipped boolean, ...);
Contents (oid int foreign key references Orders.oid,
            pid int foreign key references Products.pid)
            primary key (oid, pid);
Products (pid int primary key, name char(100));
    
```

Orders is stored sorted in order\_date order. Other tables are not explicitly ordered.

We want to run a query that finds the order that contains twinkies that is most in need of fulfillment. The order most in need of fulfillment is the one with the oldest order date that has not yet shipped. Using AQuery, we could write the following query:

```

SELECT first(1, Order.oid)
FROM Orders as O, Products as P, Contents as C
ASSUMING ORDER order_date
WHERE C.oid = O.oid
AND C.pid = P.pid
AND P.name = 'twinkie'
AND O.shipped = false;
    
```

Suppose the initial plan for this query looks as shown in Figure 1 below.

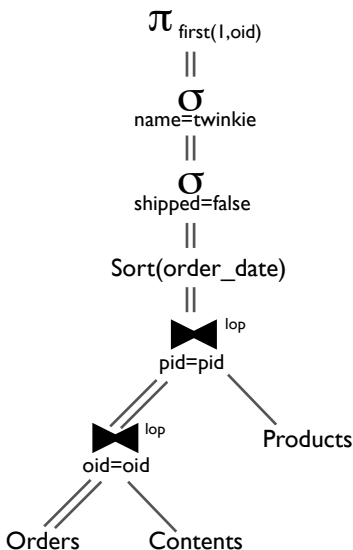


Figure 1: Unoptimized plan for 'most in need of fulfillment' query.

**14. [14 points]:** Show graphically, through a series of successive transforms, a possible optimized plan which would result after the AQuery optimizer was applied to this plan. You do not need to explicitly refer to the optimizations listed in Table 1 of the AQuery paper, though you may find them useful as a reference. You should consider both optimizations that a traditional optimizer might apply as well as those that are enabled by the additional ordering information present in this query.

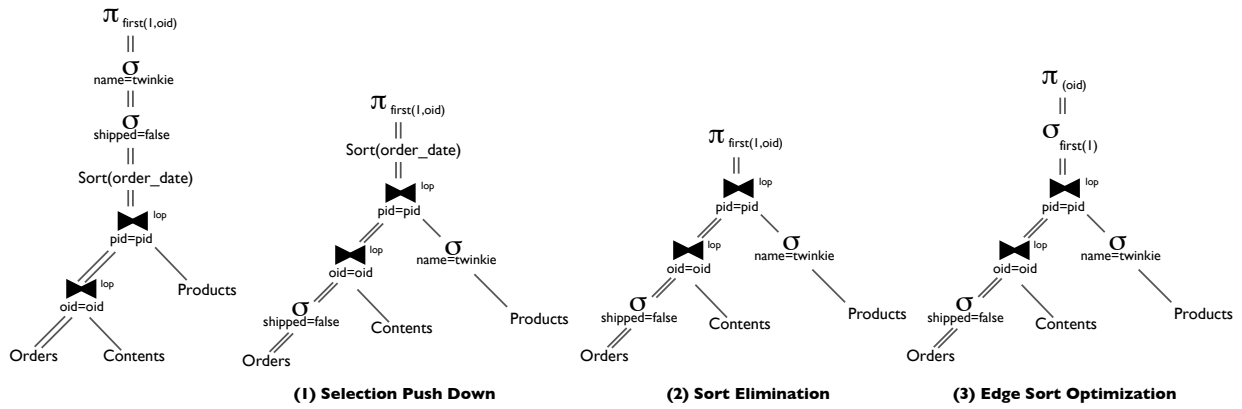


Figure 2: Solution.

End of Quiz 2