CARNEGIE MELLON UNIVERSITY
COMPUTER SCIENCE DEPARTMENT
15-415 DATABASE APPLICATIONS

MIDTERM EXAMINATION ANSWER KEY
SPRING 2003

- 80 minutes duration (1:30-2:50pm)
- Fill out your information on both this page and on the blue book cover.
- Read each question *carefully* and ask if there is something you do not understand.
- Start from the easy questions and work your way to the harder ones.
- All *paper* aids (i.e., books and notes) are allowed (no laptops/PDAs).
- Be brief and explain your answers where you are explicitly asked to do so.
- Please write clearly and be neat. Hard-to-read answers will not be considered.
- You should have **4 non-empty pages, including this one.**

LAST NAME	
first name	
andrew login	

Question	Points
1. ER Diagrams	/10
2. Relational Query Languages	/40
3. Indexing I	/4
4. Indexing II	/8
5. Indexing III	/8
6. External Sorting	/15
7. Disks and Files	/15
Total	/100

1. E-R diagrams [10]

Consider a video rental store, which keeps track of videos. Each video has a unique product-id, a title and a date of production. For each video, there are one or more copies ('tapes'), each with a unique tape-id number (unique across all tapes of all videos). We also want to keep track of the customers: for each one, we want to record a unique customer-id, credit-card number, name, and address. For each rental, we record the customer, the tape, the date it was rented, the date it is due, and the date it was actually returned.

(a) Give the correct E-R diagram for the above setting. You may choose to list the attributes separately, to un-clutter your diagram. [4]

(b) List all **candidate keys** for every strong entity. [2]

(c) List the **weak entities**, if any, and their partial keys. [2]

(d) List the **cardinalities** for every relationship. [2]

Answer:

Strong entities: VIDEO(p-id, title, date)
CUSTOMER(c-id, name, address, credit-card-number)
TAPE(tape-id)
Relationships: HAS-COPY: 1-to-N, from VIDEO to TAPE
BORROWS: N-to-M, from CUSTOMER to TAPE;
attributes: date-out, date-due, date-returned
partial key: date-out
TAPE(tape-id)
Strong entities: VIDEO(p-id, title, date)
CUSTOMER(c-id, name, address, credit-card-number)
TAPE(tape-id)

Also acceptable: to turn BORROWS into a weak entity:
RENTAL(date-out, date-due, date-returned)
depending on either CUSTOMER or TAPE (or both)
Then, there should be 1-to-N relationships
from CUSTOMER to RENTAL and from TAPE to RENTAL.
Also acceptable: to group CUSTOMER-BORROWS-TAPE with aggregation.

2. Relational Query Languages [40]

Consider the following relational schema, that keeps track of customers, orders, products and preferences:

CUSTOMER (cid, cname, address)
PRODUCT (pid, pname, unit_price)
ORDER (cid, pid, quantity, date)
DESIRES (cid, pid)

Each customer has a unique *cid*; each product has a unique *pid*; a customer may order the same product multiple times. *DESIRES* shows which customer desires what products (either he/she has ordered them, or not).

Express the following queries, in **all** of the three query languages:

- Relational Algebra (RA)
- Relational **Tuple** Calculus (RTC)
- SQL

The queries are:

(a) Find the *pid*'s of the products that cost 10 or more dollars. [2,2,1]

RA: $\Pi_{pid}\ (\sigma_{unit\text{-}price}=10(product))$
RTC: $\{P|\exists Q\ (Q\in Product \wedge Q.pid = P.pid \wedge Q.unit\_price \geq 10)\}$
SQL:
```
select pid
from products
where unit-price ≥ 10
```

(b) Find the *names* of the products that the customer with *cid*='C123' has ordered. [2,2,1]

RA: $\Pi pname\ (\sigma_{e\text{-}id} = \text{'C123'}\ (PRODUCT\ \omega\varpi\ ORDER))$
RTC: $\{X \mid \exists P(P\in PRODUCT \wedge P.pname=X.pname \wedge \exists O\ (O\in ORDER \wedge O.cid='C123' \wedge O.pid = P.pid))\}$
SQL:
```
select pname
from PRODUCT, ORDER
where PRODUCT.pid = ORDER.pid and ORDER.cid ='C123'
```

(c) Find the *pid*'s of products that are more expensive than the product with *pid*='P001'. [5,5,5]

RA: $\Pi_{PRODUCT.pid}\ [\rho_{P1}(\sigma_{pid='P001'}(\ PRODUCT))\ \omega\varpi_{\ \theta}\ \ PRODUCT]$
where $\theta = P_1.unit\text{-}price < PRODUCT.unit\text{-}price$
RTC: $\{P|\exists P_1(P_1 \in PRODUCT \wedge P.pid = P_1.pid \wedge \exists P_2\ (P_2 \in PRODUCT \wedge P_{1.unit\text{-}price} > P_{2.unit\text{-}price} \wedge P_{2\cdot pid}= \text{'P001'}))\}$
SQL:
```
select  P1.pid
from PRODUCT P1, PRODUCT P2
where P2.pid ='P0-01' and P1.unit-price . P2.unit-price
```

(d) List all the *cid's* for customers that have already ordered everything they desire. [5,5,5]

RA: Π_{cid} (DESIRES)- Π_{cid} (DESIRES – $\Pi_{cid,pid}$ (ORDER))
RTC: unsafe, but acceptable:
$\{D|\forall D_1\ (D_1 \in DESIRES \wedge D_1.cid = D.cid) => \exists O\ (O\in ORDER \wedge O.cid = D_1.cid \wedge O.pid = D_1.pid)\}$
safe:
$\{C|\exists C_1\ (C_1 \in CUSTOMER \wedge C_1.cid –c.cid \wedge \forall D\ (D\in DESIRES \wedge D.cid= C_1.cid\ ) => \exists O\ (O \in ORDER\ \wedge O.cid = D.cid \wedge O.pid = C_1.pid))\}$

```
SQL: (select cid
      From DESIRES )
      Except
         (select cid
         from ((select cid, pid
         from DESIRES )
         except
            (select cid,pid
            from ORDER )))
   or,
   select Customer.cid
   from customer
   where not exists
      (select DESIRES.cid
      from DESIRES
      where DESIRES.cid = CUSTOMER.cid and DESIRES.pid NOT IN
         (select ORDER.pid
         from ORDER
         where ORDER.cid = CUSTOMER.cid))
```

3. Indexing I [4]

For each of the following statements, state whether it is true or false.

____**F**____B^+-trees are always faster than sequential scanning on range queries. [1]

____**T**____A hash-based index can answer range queries efficiently (e.g., queries with range predicates, e.g., "$100 \leq ssn \leq 200$"). [1]

____**T**____A hash-based index can answer exact-match queries efficiently (queries with exact-match predicates, e.g, "ssn = 123"). [1]

__**T/F**__For exact-match queries, a hash-based index is guaranteed to be faster than a B^+-tree index. [1]

This last one is tricky, because it depends on the implementation of the hash index. The statement is True if the implementation is very good (a binary search on the corted bucket contents), otherwise it is false.

4. Indexing II [8]

Consider a clustering B^+-tree index on the primary key *ssn* of table *EMP(ssn, ...)*. The table spans *D* disk pages. The index uses the Alternative 1 for data entries (actual data records on leaves). Let *h* be the height of the tree.

(a) What is the minimum number of disk accesses required for an exact-match search query (eg., *ssn = 123*)? [1]
h+1. also acceptable: *h*, for unsuccessful search.

(b) What is the maximum? [1]
$h+1$.

(c) What is the minimum number of disk accesses (reads + writes) for inserting a new record? [1]
$h+1$ reads plus1 write.

(d) What is the maximum? [3]
$3*(h+1)+1$: the worst case is a propagated split that creates a new root; then we have $h+1$ to find the place; 2 writes on the file; $2*h$ writes for all levels of the tree; 1 more write for the new root. Total $3*h+4$ disk accesses.

(e) What is the *minimum* number of disk accesses for printing all records in key order? [1]
$h+D$

(f) What is the *maximum*? [1]
$h+D$

5. Indexing III [8]

Consider a B^+-tree of order m=5 (i.e., at most 4 keys per non-root node). Consider the densest possible B^+-tree that can hold the keys 1, 2, ..., 15.

(a) (Yes/No) Is it possible for this tree to have 4 nodes? [2]
No.

(b) Give a drawing of it. [6]
Leaves: (1,2,3,4), (6,7,8,9), (11,12), (14,15); root (5,10,13). There are many more correct solutions.

6. External Sorting [15]

Consider a file with 20,000 pages, and you have three available pages in memory to use as sorting buffers. Answer the following questions, assuming you use the most general sorting algorithm to sort this file. Your answers should contain only numbers (and, for complex calculations, arithmetic operators) but *no variables* (e.g., $log_{17}(200*635)$ is an acceptable answer, but *N*300* is not).

Clarification given in class:

The "most general sorting algorithm" is the algorithm that, in Pass 0, uses all the available buffers to create the fewest, longest possible runs using quicksort.

(a) How many runs will you produce during the first pass (Pass 0)? [2]

In the first pass (Pass 0) $\lceil N/B \rceil$ runs are produced, where N is the number of file pages and B is the number of available buffer pages:

$\lceil 20000/3 \rceil = 6667$ sorted runs.

(b) How many passes will it take to sort the file completely? [3]

The number of passes required to sort the file completely, including the initial sorting pass, is $\lceil log_{B-1} N1 \rceil + 1$:, where $N1 = \lceil N/B \rceil$ is the number of runs produced by Pass 0:

$\lceil \log_2 6667 \rceil + 1 = 14$ passes.

(c) What is the total I/O cost of sorting the file? [4]

Since each file is read and written once per pass, the total number of page I/O operations for sorting the file is *2*N*#of passes*:

2*20000*14 = 560000 page I/Os.

(d) How many buffer pages do you need to sort the file completely in just *two* passes? [6]

In pass 0, $\lceil N/B \rceil$ runs are produced; in pass 1, we must be able to merge this many runs; i.e., $B-1 \geq \lceil N/B \rceil$. This implies that B must at least be large enough to satisfy $B*(B-1) \geq N$; this can be used to guess B, and the guess must be validated by checking the first inequality. Thus:

With 20000 pages in the file, B=142 satisfies both inequalities, B=141 does not, so we need 142 buffer pages.

7. Disks and Files [15]

Modern disk drives store more sectors on the outer tracks than on the inner tracks. Since the rotation speed is constant, the sequential data transfer rate is also higher on the outer tracks. The seek time and rotational delay are unchanged. Given this information, which are good strategies for placing files with the following kinds of access patterns? For instance, should we place the file on the inner tracks, on the outer tracks, on the middle tracks, or it does not matter? *Explain your answer.*

(a) Sequential scans of a large file (e.g., selection from a relation with no index). [5]

Place the file in the outer tracks. Sequential speed is most important, and outer tracks maximize it.

(b) Random accesses to a large file via an index (e.g., selection from a relation via an index). *Here, you must choose a strategy to store both the index and the file.* [5]

Place the file and index on the inner tracks. The DBMS will alternately access pages of the index and of the file, and thus the two should reside in close proximity to minimize seek times. By placing the file and the index on the inner tracks we also save valuable space on the faster(outer) tracks for other files that are accessed sequentially.

(c) Sequential scans of a small file. [5]

Place small files in the inner half of the disk. A scan of a small file is effectively random I/O because the cost is dominated by the cost of the initial seek to the beginning of the file.