CS4320 Final Exam

December 15th, 2019

(120 minutes working time)

Name: ________________________________ Cornell NETID: _________

I understand and will adhere to the Cornell Code of Academic Integrity.

--

Signature

Maximum number of points possible: 95. This exam counts for 35% of your overall grade. Questions vary in difficulty. Do not get stuck on one question.

In all problems, whenever you think a problem is underspecified, make assumptions and clearly state them.

Good luck!

You have 120 minutes working time for this exam.

Part A) SQL Queries. (20 points)

Consider the database schema created by the following SQL commands:

```
CREATE TABLE Sailors (sid integer PRIMARY KEY,
sname varchar(20), rating integer, age real);

CREATE TABLE Boats (bid integer PRIMARY KEY,
bname varchar(20), color varchar(20));

CREATE TABLE Reserves(sid integer, bid integer,
day date, PRIMARY KEY (sid, bid, day),
FOREIGN KEY (sid) references Sailors (sid),
FOREIGN KEY (bid) references Boats (bid));
```

The database stores information on sailors (table `Sailors`), on boats (table `Boats`), and reservations for boats by sailors (table `Reserves`). Duplicates in the query result are okay (e.g., if the question asks to retrieve sailor names, it is okay if the query returns the same name multiple times). You may assume that the database contains no `NULL` values and that no table is empty.

A.1) Write an SQL query retrieving pairs of boats and sailors (i.e., `bid` and `sid`) that satisfy all of the following conditions: the sailor made a reservation for the boat, the sailor never made a reservation for a different boat, and the boat was never reserved by another sailor. (5 points)

Select R.Sid, R.bid from Reserves R where not exists (select * from Reserves R2 where R2.sid = R.sid and r2.bid <> R.bid) and not exists (select * from Reserves R3 where r3.bid = R.bid and R3.sid <> R.sid);

A.2) Write an SQL query retrieving sailors (attribute `sid`) who made two *consecutive* reservations for the same boat (i.e., the sailor did not make any reservation for different boats between the two reservations for the same boat). Note that the `day` attribute in `Reserves` stores the reservation date. You may assume that sailors reserve at most one boat per day.
(5 points)

Select Sid from Reserves R where exists (select * from Reserves R2 where R2.sid = R.sid and R2.bid = R.bid and R2.date>R.date and not exists (select * from Reserves R3 where R3.sid=R.sid and R3.bid=R.bid and R.date<R3.date and R3.date<R2.date));

A.3) Write an SQL query calculating for each boat color the average number of reservations that a boat of this color gets (considering all reservations and boats in the database). The query result has two columns, the color (attribute `color`) and the average number or reservations.
(5 points)

Select B.color, avg(count) from Boats B, (Select B.bid as bid, count(R.sid) as count from Boats B left outer join Reserves R on (R.bid=B.bid) group by B.bid) as Temp where B.bid = Temp.bid group by B.color;

A.4) Write an SQL query retrieving for each sailor the boat that the sailor reserved most often. The query result has two columns, the sailor ID (`sid`) and the boat ID (`bid`) such that the boat appears in most reservations of that sailor. Sailors without reservations must not appear in the query result. You can assume that there is only one boat with the maximal number of reservations for each sailor.
(5 points)

Select T.sid, T.bid from (Select Sid, bid, count(*) as count from Reserves R group by Sid, bid) as T where not exists (Select * from (Select Sid, bid, count(*) as count from Reserves R group by Sid, bid) as T2 where T2.count > T.count and T2.sid = T.sid);

Part B) Processing Cost. (15 points)

Note: we do not consider the cost of writing out the final result in the following.

B.1) We perform a block nested loops join between relations R and S with R as outer relation. R consumes x pages on disk and S consumes 2*x pages on disk (where x is an integer). We use 12 buffer pages (total, including input and output buffers) for the join. The join cost is 30 page reads. How large is x? Justify by showing your calculations.
(7.5 points)

After subtracting one page for the input buffer for the inner operand and one output buffer, we can store blocks of size 10 pages. The join cost is x + ceil(x/10)*2*x. Assume x<=10 which means that we have one block. Then the join cost simplifies to x+2*x. Setting x=10 leads to a cost of 30 page reads. Using a smaller x leads to smaller cost while using a larger x leads to larger cost. Hence, x=10 is the unique solution.

B.2) We perform an index nested loops join between relations R and S with R as outer relation. Each tuple of R consumes 10 bytes and R has 5,000 tuples. Each page stores 1,000 bytes. We have an index for S on the join column and enough main memory to execute the join (the precise amount does not matter nor does the size of S). How expensive can each index access be (measured by the number of page reads per access, assuming that this number is an integer) such that the total join cost is at most 10,000 page reads? Justify by showing your calculations.
(5 points)

For R, each page stores 100 tuples. Hence, R consumes 5,000/100 = 50 pages in total. The join cost is therefore 50 + 5,000 * x page reads where x is the number of page reads per index access. If 50+5,000 * x<=10,000 and x is integer then x can be one at most.

B.3) We use a hard disk for which page writes take twice as long as page reads. Propose a new cost formula for the hash join for this scenario. The formula should depend on parameters m and n, representing the number of pages in the two input relations. Assume that enough main memory is available to partition data in one single pass. Justify your formula in one to two sentences.
(2.5 points)

The formula (m+n)*4 (as opposed to (m+n)*3) integrates the fact that writing out data after the first partitioning pass is twice as expensive as reading data.

Part C) Database Design and Normalization. (20 points)

C.1) Consider the set of functional dependencies F={A→C, D→E, A→D, B→A}. Relation R has attributes A, B, C, D, and E. Which of those five attributes are keys for R? Justify by calculating the attribute closure.
(5 points)

The attribute closure of A is ACDE. A is therefore no key.
The attribute closure of B is BACDE, it is therefore a key.
The attribute closure of C is C, it is no key.
The attribute closure of D is DE, it is no key.
The attribute closure of E is E, it is no key.

C.2) Consider the set of functional dependencies F={A→B, A→C}. Relation R has attributes A, B, and C. Which decompositions of R (into two relations such that exactly one attribute appears in both) are lossless join decompositions? Justify for each decomposition in one or two sentences.
(5 points)

Assume attribute A appears in both relations then we decompose into relations (AB) and (AC) or into (ABC), (A). A is a key in all those relations (as ABC is its attribute closure), hence the intersection of attributes contains a key for at least one decomposed relation. Hence, all decompositions are lossless join decompositions.

Assume attribute B appears in both relations then we decompose into relations (BA), (BC) or (BCA), B. The attribute closure of B is only B so only the second possibility is a lossless join decomposition (since the intersection of attributes, B, is a key for relation with attribute B).

Assume attribute C appears in both relations then we decompose into relations (CA), (CB) or (CAB), C. The attribute closure of C is only C so only the second possibility is a lossless join decomposition (since the intersection of attributes, C, is a key for relation with attribute C).

C.3) Consider the database created by the following SQL commands:

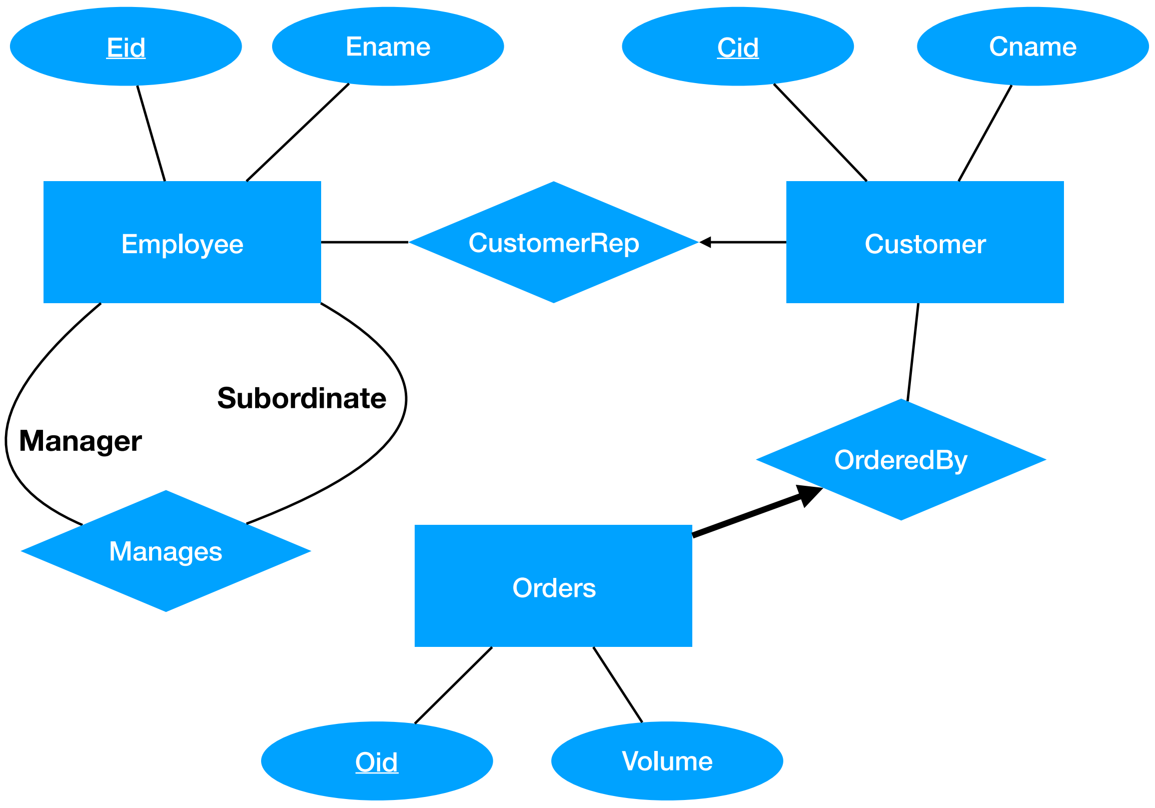
```
CREATE TABLE Employee (eid integer PRIMARY KEY,
ename varchar(20));

CREATE TABLE Customer (cid integer PRIMARY KEY,
cname varchar(20), customerRepresentative integer,
FOREIGN KEY (customerRepresentative) references
Employee(eid));

CREATE TABLE Orders (oid integer PRIMARY KEY,
volume integer, orderedBy integer NOT NULL,
FOREIGN KEY (orderedBy) references Customer(cid));

CREATE TABLE Manages(managerID integer, subordinateID
integer, PRIMARY KEY (managerID, subordinateID),
FOREIGN KEY (managerID) references Employee (eid),
FOREIGN KEY (subordinateID) references Employee (eid));
```

Draw an entity relationship diagram which translates into the SQL commands above when using the techniques seen in class. Make sure that all SQL constraints are represented in the diagram. Multiple valid solutions are possible.
(10 points)


Eid
Ename
Cid
Cname
Employee
CustomerRep
Customer
Subordinate
Manager
OrderedBy
Manages
Orders
Oid
Volume

Part D) Concurrency Control. (20 points)

In the following, we ask you to write out schedules with certain properties. Use the notation seen in the lecture (i.e., WT(A) means transaction number T writes object A, RT(A) means transaction T reads object A, CT means transaction T commits, AT means transaction T aborts).

D.1) Propose a schedule involving two transactions that is not recoverable but conflict-serializable. Justify why it is non-recoverable in one to two sentences. Justify why it is conflict-serializable by drawing the conflict graph.
(5 points)

W1(A) R2(A) W2(B) C2 A1.

This schedule is non-recoverable since transaction 2 commits after reading data from an uncommitted transaction (which aborts here). Nevertheless, the conflict graph has no cycles and only one node for T2 (aborted transactions are not taken into account) which means the schedule is conflict serializable.

D.2) Propose a schedule involving two transactions that avoids cascading aborts but is not strict. Justify in one to two sentences why it avoids cascading aborts and why it is not strict.
(5 points)

W1(A) W2(A) C2 C1.

The schedule avoids cascading aborts since no transaction reads uncommitted data. The schedule is not strict since uncommitted data is overridden (transaction 2 overrides the uncommitted version of A, written by transaction 1).

D.3) Propose a schedule that has the following conflict graph:

$$\begin{tikzcd}
1 \arrow[r] & 2 \arrow[d] \\
3 \arrow[ur] \arrow[r] & 4
\end{tikzcd}$$

(5 points)

W1(A) R2(A) W3(B) R2(B) W3(C) R4(C) W2(D) R4(D).

D.4) Propose a schedule that could not have been generated when using any variant of two-phase locking. Justify in one to two sentences.
(5 points)

W1(A) R2(A) W1(A)

This schedule is not conflict-serializable (cycle via conflicts on A). As two-phase locking only generates conflict-serializable schedules, the schedule could not have been generated by 2PL.

Part E) Logging and Recovery. (20 points)

The ARIES algorithm maintains various data structures during execution:

- A dirty page table, containing entries of the form (pageID, recLSN) referring to a data page and the log entry at which it became dirty,
- A transaction table, containing entries of the form (transactionID, status, lastLSN) referring to a transaction, its status, and its last action,
- An "in-memory" pageLSN for each data page in main memory,
- An "on-disk" pageLSN for each data page stored on hard disk,
- the flushedLSN counter indicating up to which point the log has been written to disk.

For the following log entries, indicate precisely which data structures are changed and how. For changes to dirty page table or transaction table, indicate each field for newly inserted entries or indicate precisely which fields of existing entries change. Assume that data structures are only changed when necessary (e.g., flushedLSN is only updated when required by the rules of write-ahead logging). Assume that dirty page table and transaction table are initially empty, all pageLSN and flushedLSN counters are initialized to zero. Page steals happen only if explicitly mentioned.

After each entry on the following page, describe precisely how data structures are updated after the entry has been processed (e.g., write "in-memory pageLSN for page P10 changes to 40 and transaction T5 is inserted into the transaction table with status running and lastLSN 40").

10 T1 writes P3
in-memory pageLSN for P3 changes to 10 and transaction T1 is inserted with status running and lastLSN 10, P3 with recLSN 10 is inserted into the DPT.

20 T2 writes P2
in-memory pageLSN for P2 changes to 20 and transaction T2 is inserted with status running and lastLSN 20, P2 is inserted with recLSN 20 into the DPT.

30 T1 writes P5 - P2 gets written to disk due to page stealing
in-memory pageLSN for P5 changes to 30, T1 lastLSN is updated to 30, P2 is taken out of DPT, on-disk pageLSN for P2 changes to 20, P5 is inserted into DPT with recLSN 30, flushedLSN is updated to 20.
40 T3 writes P1
in-memory pageLSN for P1 changes to 40, T3 inserted into transaction table with status running and lastLSN 40, P1 inserted into DPT with recLSN 40.

50 T1 Commits
flushedLSN is updated to 50, status of T1 changed to committed in transaction table and lastLSN changed to 50

60 T1 End
T1 is taken out of transaction table

70 T3 writes P6
in-memory page LSN for P6 changes to 70, lastLSN for T3 is updated to 70, P6 is inserted into DPT with recLSN 70

80 T2 Aborts
Status of transaction T2 is updated to aborted, lastLSN set to 80

90 CLR T2 P2 (Undoing LSN 20)
P2 is inserted into DPT with recLSN 90, in-memory pageLSN for P2 updated to 90, lastLSN of T2 changed to 90

CS4320 Final Exam

This page will be used for grading your exam. Do not write anything on this page.

SECTION	QUESTION	SCORE	SECTION TOTAL
Part A	A.1 (Max: 5 points)		(Max: 20 points)
	A.2 (Max: 5 points)		
	A.3 (Max: 5 points)		
	A.4 (Max: 5 points)		
Part B	B.1 (Max: 7.5 points)		(Max: 15 points)
	B.2 (Max: 5 points)		
	B.3 (Max: 2.5 points)		
Part C	C.1 (Max: 5 points)		(Max: 20 points)
	C.2 (Max: 5 points)		
	C.3 (Max: 10 points)		
Part D	D.1 (Max: 5 points)		(Max: 20 points)
	D.2 (Max: 5 points)		
	D.3 (Max: 5 points)		
	D.4 (Max: 5 points)		
Part E	E (Max: 20 points)		(Max: 20 points)
Total (Max: 95 points)			