CS4320 Prelim Exam

October 28th, 2021

(60 minutes working time)

Name: _____ Cornell NETID: _____

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

Signature

Maximum number of points possible: 50. 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!

Note – you have 60 minutes working time for this exam, NOT 2 hours as on some other prelims.

Prelim Part A) SQL Queries. (20 points)

Consider the database schema created by the following SQL commands:

```
CREATE TABLE Airlines (aid integer PRIMARY KEY,
name text);

CREATE TABLE Airports (apid integer PRIMARY KEY,
name text);

CREATE TABLE Flights (fid integer PRIMARY KEY,
airline integer, start_airport integer,
end_airport integer, duration integer);

ALTER TABLE Flights ADD FOREIGN KEY (airline)
REFERENCES Airlines(aid);

ALTER TABLE Flights ADD FOREIGN KEY (start_airport)
REFERENCES Airports(apid);

ALTER TABLE Flights ADD FOREIGN KEY (end_airport)
REFERENCES Airports(apid);
```

The database stores information on flights (table `Flights`), on airlines (table `Airlines`), and on airports (table `Airports`). The following questions refer to this database.

A.1) Write an SQL query retrieving the average duration for all flights that start at Ithaca (airport with name 'Ithaca'). The query result contains one single column for the average duration and one single row. (4 points)

Select avg(F.duration) from Flights F join Airports A on (F.start_airport = A.apid) where A.name = 'Ithaca';

A.2) Write an SQL query retrieving for each airport the number of flights that end at this airport (this number may be zero). The query result contains two columns: the airport name and the associated count. The result may contain duplicate rows. (5 points)

Select AP.name, count(F.end_airport) from airports AP left outer join flights F on (AP.apid = F.end_airport) group by AP.apid, AP.name;

A.3) Write an SQL query retrieving all airlines that are not associated with any flights starting from airport 'Ithaca'. The query result must contain one single column with the airline name. The result may contain duplicate rows. (5 points)

Select name from airlines AL where not exists (select * from flights F join airports AP on (F.start_airport = AP.apid) where AP.name = 'Ithaca' and F.airline = AL.aid);

A.4) Write an SQL query retrieving the airport for which the number of flights starting at that airport is maximal (compared to all other airports). If there are multiple airports with the same number of flights, your query should return all of them. The query result has one column with the airport name. The result should not contain duplicate rows. (6 points)

Select distinct name from airports AP where not exists (select * from airports AP2 where (select count(*) from flights F where F.start_airport = AP.apid) < (select count(*) from flights F where F.start_airport = AP2.apid));

Prelim Part B) Indexing. (20 points)

We store 100,000 flights in the `Flights` table, defined as in Part A).

We build a B+ tree index on the `Flights` table, using the `airline` column as search key. The B+ tree has five levels (including root and leaf level). Also, we create a hash index on the `Flights` table, using `start_airport` as search key. This index uses static hashing. We assume that each hash bucket is associated with one single page containing references. No overflow pages are present.

Both indexes are unclustered and store references to data (not the data itself). All index data is stored on hard disk (not cached in main memory). We assume that all pages of each index are full (i.e., no space is wasted).

For the following questions, assume each page stores 1,000 bytes. Storing data for one integer field consumes four bytes. Storing a reference consumes 10 bytes. The `Flights` table contains data for 10 airlines and has 5,000 distinct values in the `start_airport` column. Assume data is uniformly distributed over different airlines and start airports.

B.1) We use the B+ tree index on the `airline` column to retrieve data on all flights associated with one specific airline. Calculate the cost for accessing the index and for retrieving the data. (6 points)

We read five pages to traverse the tree index from the root to a leaf node. Assuming uniform data, we expect 10,000 flights from the airline referenced by the query. The index stores references in the leaf pages and each reference consumes 10 bytes. Hence, having 1,000 bytes per page, we can store 100 references per page. We must read 10,000 / 100 = 100 leaf pages from the index (99 additional pages after reading the first leaf). As seen in class, we count one page access per reference to retrieve the associated data (i.e., 10,000 page reads). In total, we read 10,104 pages.

B.2) We use the hash index on the `start_airport` column to retrieve all entries for a specific start airport. Calculate the cost for accessing the index and for retrieving the data. (6 points)

To use the index, we calculate the hash value for the start airport we are interested in. Then, we follow the associated references and retrieve the corresponding data. There are 5,000 start airports in the data set and we assume uniform distribution. Hence, we expect that 100,000 / 5,000 = 20 flights satisfy the predicate. We count 20 page reads to follow all references. In addition, we need one page read to obtain the references. Therefore, we have a total of 21 page reads.

B.3) Calculate the cost of scanning (i.e., reading each page consecutively) the entire Flights table. (4 points)

The flights table has five columns of type integer. Each integer field consumes four bytes. Hence, we need to store 5 * 4 = 20 bytes per row. As each page stores 1,000 bytes, we can store 1,000 / 20 = 50 rows per page. We have 100,000 flights in total which requires 100,000 / 50 = 2,000 pages. Hence, a scan costs 2,000 page reads.

B.4) Consider the following two SQL queries on a table storing flights (here, we define `end_airport` as a text column):

- `SELECT * FROM Flights WHERE end_airport = 'Ithaca' and duration < 120`
- `SELECT * FROM Flights WHERE duration = 90`

Propose one index that could be used to answer both queries (according to the rules seen in class). Specify the index type and index search key. Justify in one to two sentences that the index is applicable. (4 points)

We create a B+ tree index with composite key (duration, end_airport). The set of restricted columns is a prefix of the composite index key for both queries. As the index is of type tree index, it supports inequality as well as equality predicates.

Prelim Part C) Join Operators. (10 points)

C.1) We join two tables (R and S) with a hash join. R contains 10,000 rows and S contains 50,000 rows. For both tables, we can store 100 rows per page. Calculate the cost of the hash join, using formulas seen in class. Show your calculations to justify your answer. (4 points)

We have 10,000 rows in R and can fit 100 rows per page. Hence, R consumes 10,000 / 100 = 100 pages. We have 50,000 rows in S which consumes 50,000 / 100 = 500 pages. The cost of a hash join is equivalent to three times the total number of input pages. Here, we have a cost of 3*(100 + 500) = 1,800 page I/Os.

C.2) We join the same tables as in C.1 (R and S) using a block nested loops join. We use a block size of 10 pages. S is the outer operand. Calculate the join cost, using the formulas seen in class. Show your calculations to justify your answer. (4 points)

As justified in C.1, R consumes 100 pages while S consumes 500 pages. We use S as outer operand with a block size of 10, hence we obtain 500 / 10 = 50 blocks. The join cost is given as the cost of reading the outer operand plus the cost of reading the inner operand 50 times (number of blocks). Hence, the cost is 500 + 50 * 100 = 5,500 pages.

C.3) Compared to the join described in C.2, how can we reduce the cost of joining R and S with a block nested loops join? Propose a change to the join described in C.2 that will likely reduce execution costs. Justify in one to two sentences. You do not need to calculate the precise cost after the change. (2 points)

Given fixed join operands, the block nested loops join becomes cheaper if the smaller table is used as outer operand. Hence, we can likely reduce join cost by using R (instead of S) as outer operand.

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: 4 points)		(Max: 20 points)
	A.2 (Max: 5 points)		
	A.3 (Max: 5 points)		
	A.4 (Max: 6 points)		
Part B	B.1 (Max: 6 points)		(Max: 20 points)
	B.2 (Max: 6 points)		
	B.3 (Max: 4 points)		
	B.4 (Max: 4 points)		
Part C	C.1 (Max: 4 points)		(Max: 10 points)
	C.2 (Max: 4 points)		
	C.3 (Max: 2 points)		
Total (Max: 50 points)			