**CS 122A: Introduction to Data Management – Winter 2017**

**Homework 7: Indexing for Performance  (100 points)**

***Solution Template***

1. To start down the path of exploring physical database designs (indexing) and query plans, start by checking the queries plans for each of the following queries without any indexes using the EXPLAIN function. Report the query plan after each query by taking snapshots and pasting them into your copy of this document.

1. SELECT \* FROM Bird WHERE Bird.first\_name = 'Elizabeth';
2. SELECT \* FROM Bird WHERE Bird.first\_name LIKE '%abeth';
3. SELECT \* FROM Bird WHERE Bird.first\_name LIKE 'Eliz%';
4. SELECT \* FROM Bird WHERE Bird.btag = 'laura43';
5. SELECT \* FROM Bird WHERE Bird.birthdate < '1961-1-1';

2. Now create indexes (which are B+ trees, under the hood of MySQL) on the Bird.first\_name attribute and Bird.birthdate attribute separately - i.e., create two indexes, one per attribute. Paste your CREATE INDEX statements below.

3. Re-explain the queries in Q1. Report on the query plan after each query, as before. Be sure to look carefully at each plan - think about what you are seeing there.

1. SELECT \* FROM Bird WHERE Bird.first\_name = 'Elizabeth';
2. SELECT \* FROM Bird WHERE Bird.first\_name LIKE '%abeth';
3. SELECT \* FROM Bird WHERE Bird.first\_name LIKE 'Eliz%';
4. SELECT \* FROM Bird WHERE Bird.btag = 'laura43';
5. SELECT \* FROM Bird WHERE Bird.birthdate < '1961-1-1';
6. Examine and compare the query plans for queries a-c above - in one brief sentence, what can you observe about B+ tree indexes and their usefulness for equality and different LIKE queries? (Enter your observation below.)
7. Examine and compare the query plans for queries d-e above - in one brief sentence, what can you observe about B+ tree indexes and their usefulness for equality and range queries? (Enter your observation below.)

4. It’s time to go one step further and explore the notion of a “composite Index”, which is an index that covers several fields together. **Before you start to play with composite indexes, you should drop your indexes in previous steps.**

1. Check the query plans of following queries. Report the query plan after each query.
   1. SELECT \* FROM Bird WHERE Bird.first\_name = 'Elizabeth' and Bird.last\_name = 'Smith';
   2. SELECT \* FROM Bird WHERE Bird.first\_name = 'Elizabeth';
   3. SELECT \* FROM Bird WHERE Bird.last\_name = 'Smith';
2. Create a composite index on the attributes last\_name and first\_name (in that order!) of the Bird table. Paste your CREATE INDEX statement below.
3. Re-explain these queries and report the query plan after each query.
   1. SELECT \* FROM Bird WHERE Bird.first\_name = 'Elizabeth' and Bird.last\_name = 'Smith';
   2. SELECT \* FROM Bird WHERE Bird.first\_name = 'Elizabeth';
   3. SELECT \* FROM Bird WHERE Bird.last\_name = 'Smith';
4. Examine and compare the query plans for this composite indexing example. As before, in one brief sentence, what can you observe about composite B+ tree indexes and their applicability to multi-attribute queries? (Enter your observation below.) If you notice anything “familiar”, based on things you saw earlier, feel free to mention that too. (:-))

**EXTRA CREDIT I** [10 points] If you would like to delve even further into the wild world of query plans and indexing, you could try dropping the composite index from Q4 part b (above) and instead creating two separate indexes, one per attribute, and examine the query plans for your queries under that alternative physical database design. If you wish to do so, your task (detailed below) will be to repeat steps Q4 b-d above but with those two indexes instead of the single composite index.

1. Drop the index that you created in Q4 using MySQL’s DROP INDEX statement. Paste your DROP INDEX statement below.
2. Create the two separate indexes. Paste your CREATE INDEX statements below.
3. Re-explain the queries in Q4 with the created indexes.
   1. SELECT \* FROM Bird WHERE Bird.first\_name = 'Elizabeth' and Bird.last\_name = 'Smith';
   2. SELECT \* FROM Bird WHERE Bird.first\_name = 'Elizabeth';
   3. SELECT \* FROM Bird WHERE Bird.last\_name = 'Smith';
4. Examine and compare the new query plans for these queries with the previous query plans from Q4; briefly summarize your observations below.

**EXTRA CREDIT II** [5 points] Going back to Q3 e, execute following query **with** the index that you created on birthdate:

SELECT \* FROM Bird WHERE Bird.birthdate > '1975-1-1';

Compare the query plan with what you get in Q3 e, and briefly explain your observation.