Lab 8: NoSQL and Query Evaluation

CS44 Database Systems, Spring 2015
Due by 11:59pm on Sunday, May 3

The primary objectives of this lab are to reinforce course objectives:

- Obtain experience with a non-relational database (MongoDB)
- Identify differences between relational and NoSQL database systems
- Identify algorithms for evaluating relational operators
- Analyze the performance of these various algorithms

You may work with one other person on this lab. There are two distinct components of this lab; each section below will detail the deliverables. Briefly, the files you will need to manually submit include:

- `history` – a history of your shell session
- `lab8.pdf` – the solution to the problem set below

The other deliverable is your database from the tutorial below; I will automatically obtain this. Place all files in your `~/cs44/labs/8/` directory and use `handin44` to electronically submit the lab. Be sure you indicate your partner using the `p` option during the handin process and also be writing all names on the PDF document. Your solution is due 11:59pm on **Sunday, May 3, 2015**.

MongoDB tutorial

Overview

In this part of the lab, you will complete a simple tutorial to illustrate usage of a popular NoSQL database system – MongoDB. The goal of this module is to complete the tutorial for *understanding*, not to exactly replicate the coding examples. This component will be graded only for completion – the files you hand in will verify that you completed the tutorial. So, to be clear, allocate your time to maximize understanding of the differences between MongoDB and a typical RDBMS like SQLite.

Getting Started

Access the tutorial via the following URL: `http://openmymind.net/mongodb.pdf` *Note: there are printed copies of this tutorial left behind by Kyle after his talk. I have noticed a few typos but it is almost exactly the same.*

To get started, run:

```
$ mongo licorice.cs.swarthmore.edu
MongoDB shell version: 3.0.2
connecting to: licorice.cs.swarthmore.edu/test
```

The program `mongo` invokes an interactive shell environment. The argument specifies to connect to a MongoDB server on `licorice.cs.swarthmore.edu`. *This is essential.* Forgetting this will cause a local instance to run, which will be difficult to find later.

Enter the following as the first command in the shell,

```
> use userID
```

Replace **userID** with your login ID (e.g., asas or asas_tnas if you prefer to include both partners ids). This is essential to create a unique space for you to work in – omitting this will cause conflicts with other users.

MongoDB has already been installed for you, so you can go ahead and start with **Chapter 1**. Feel free to skim/skip the following (i.e., you do not need to enter the commands for these sections, but should still read):

- Chapter 2 on Updates.
- Chapter 7 – everything after explaining queries

If you cannot finish the tutorial in one sitting, run the exact same setup as above – `use` will load the existing database from your previous session.

Deliverables

- The database resulting from your tutorial. *This is automatically generated if you properly connected to* `licorice` *and used a database named using your user id.*
- `history`. This is a shell history of user usage of `mongo`. To submit this history, copy the file to your lab 8 directory:

  ```
  $ cp ~/.dbshell ~/cs44/labs/8/history
  ```

Problem Set

1. **MongoDB tutorial**

   Answer these questions about NoSQL and MongoDB. Most of these can be found in the tutorial above.

   (a) What is the corresponding *relational* concept for each of these terms (e.g., "schema" or "table"). Not any important differences:
   - collection
   - document
   - field
   - index

   (b) In MongoDB, how do you define a selection operation ? projection?

   (c) How does mongodb store data - that is, what is the name of the file format of documents?

   Relational DBs define many-to-one relationships by defining distinct relationship tables (e.g., an *Enrolee* table encodes the many-to-many relationship between *Courses* and *Students*). How does MongoDB use arrays/embedded docs to handle many-to-one relationships?

2. **Duplicate elimination**

Consider the following query:

```sql
SELECT DISTINCT E.title, E.ename FROM Executives E
```

You have the following data stored in the database about the table `Executives`:

- The schema:

  $Executives(ename, title, dname, address)$

- All fields are strings and use the exact same amount of space
- The relation contains 10,000 pages
- There are 10 buffer pages
- *ename* is unique (this simplifies analysis by making the number of results predictable)

Using the optimized version of sort-based duplicate elimination (the initial pass creates sorted runs of tuples containing only *ename* and *title*; subsequent passes simultaneously merge runs while eliminating duplicates), answer the questions below:

(a) How many sorted runs are produced in the initial pass? How big is each run? What is the total I/O cost of this phase?

(b) How many merging passes are required to fully sort/eliminate duplicates? What is the I/O cost of each of these passes? You may assume the amount of duplicates is negligible.

(c) How many I/Os are needed if we instead used a clustered index on *title*? How about for an unclustered index? Assume in both cases that we are using a secondary B+-tree index, each with about 2500 leaf pages.

(d) Would an unclustered index on $< title, ename >$ be superior to sorting for eliminating duplicates? Explain your answer.

(e) Would your above analysis change if our query changed to:

```sql
SELECT E.title, E.ename FROM Executives E
```

That is, `DISTINCT` is no longer used. Be brief in your response.

3. **Join Algorithms**

Consider the join $R \bowtie_{R.a=S.b} S$ with the following information:

- Relation R contains 10,000 tuples with 10 tuples per page
- Relation S contains 2,000 tuples with 10 tuples per page
- Attribute b is the primary key of S
- Both tables are organized as heap files, with no indices available
- Your buffer is 52 pages

Answer the questions below, defining the precise page I/O cost of the algorithm and ignoring the size of the output in your analysis:

(a) What is the cost of the join if we use a page-nested loop join?

(b) What is the cost of a block-nested loop join?

(c) What is the cost of a sort-merge join (use the optimized version from lecture)?

(d) What is the cost of a hash join?

(e) For each of the previous three algorithms (block-nested loop, sort-merge, hash), calculate the smallest value of B buffer pages for which your cost *does not change*. That is, how small can we make the buffer and still have the same I/O cost? Your answer may vary for each algorithm.

Deliverables

Place a PDF document with your solution in your Lab 8 directory.