Alexander Taniguchi(Wiegman)
08-174510

Homework Week 5: Index and Algorithms in Databases

Q1) Take any large database, index it in SQLite and get the index value for a few randomly selected entries at ID [large-number].

I will use my database of many train stations in Japan and their associate byte-code value as stored on railway transport (SUICA, PASMO, etc.) cards throughout the country. This database is the driving force behind an Android mobile application I am developing, with re-use permission from the original project found at Abe Haruhiko's project 'SuicaPasmoReader.' The visual database application I use shows the table "StationCode" within this database has a total of 11727 entries (which may not be few hundred thousand but is still significantly large) and eight columns. Anyway, let's begin the work to index this database.

[Ah, but first I should figure out how to display Japanese character set in my command prompt!
I had to change the System Locale to Japanese, and then change the font set to MS Gothic and turn on UTF-8/Unicode character page. I think I can understand why people use Unix, this took 30 minutes to figure out from start to finish on my system.
]

```
sqlite> CREATE INDEX code_name ON StationCode (StationName);
sqlite> .indices StationCode
code_name
```

Now, we have created a simple index. The command is CREATE INDEX.

```
sqlite> select StationName from StationCode where _id=10000;
上小田井
sqlite> EXPLAIN QUERY PLAN SELECT StationName FROM StationCode WHERE _id = 10000;
0|0|0|SEARCH TABLE StationCode USING INTEGER PRIMARY KEY (rowid=?)
```

Here we use the SELECT command as usual to show me what is in the value at row 10000 as usual. However, we must verify it used an index for this. The second command, "EXPLAIN QUERY PLAN" does this by listing the method. We see that it uses a search index, instead of SCAN Table tags, so we know that the index is being used. For the fun of it, let's see a few other entries – Google's random number generator selected entries 9522, 7793, and 10751.

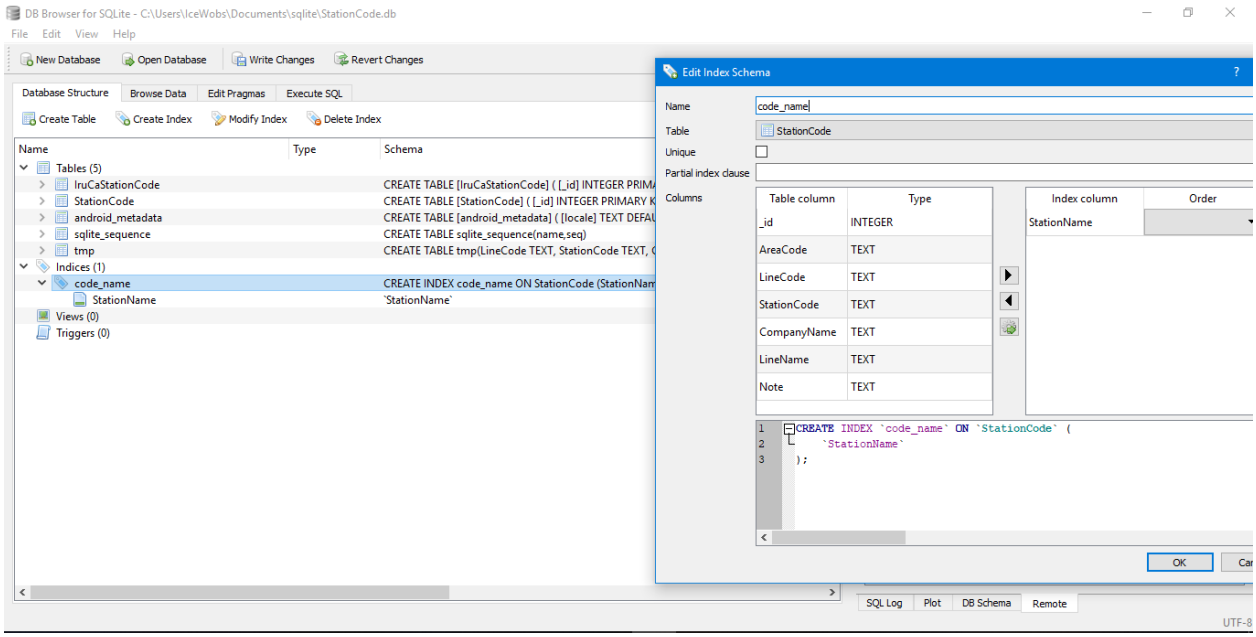
```
sqlite> select StationName from StationCode where _id=9522 AND _id=7793 AND _id=10751;
sqlite> select StationName from StationCode where _id=9522 OR _id=7793 OR _id=10751;
新習志野
新松田
公園東口
```

Using regular English grammar will fail for relatively obvious reasons. AND requires all three conditions to be true, while OR will require that each condition be non-exclusively (non-overlap) true. Regular logic tells us that OR is an optimal way to display all the station names.

```
sqlite> EXPLAIN QUERY PLAN select StationName from StationCode where _id=9522 OR _id=7793 OR _id=10751;
0|0|0|SEARCH TABLE StationCode USING INTEGER PRIMARY KEY (rowid=?)
0|0|0|EXECUTE LIST SUBQUERY 1
```

The regular verification shows that the index was used, as expected.

Figure 1



A quick look at the visual database program indeed shows that such ‘simple index’ was created, so it should have been available for use within my commands. This checks out as expected.

Q2) Database algorithmic search functions for the future - Quantum Search using "Grover Method."

As an introduction to algorithmic search, we must understand that at their core they allow for the solving of functions that lead to an element that we presume is marked as a 'target' value. At times, such an element does not exist, but I will ignore this case for the rest of this response. At the simplest form, we might have such a function that maps f: {0,1} to the coordinate {0,1}. To prove this true, a classical analysis first lists out for equations f(0)=0, f(0)=1, f(1)=0, and f(1)=1. The classic-style search would use two Boolean functions to link the four equations towards the target coordinate mapping - the first would be noting f(0)=f(1)=0 and the second would be f(0)=f(1)=1. We may see that this is elementary O(n) efficiency.

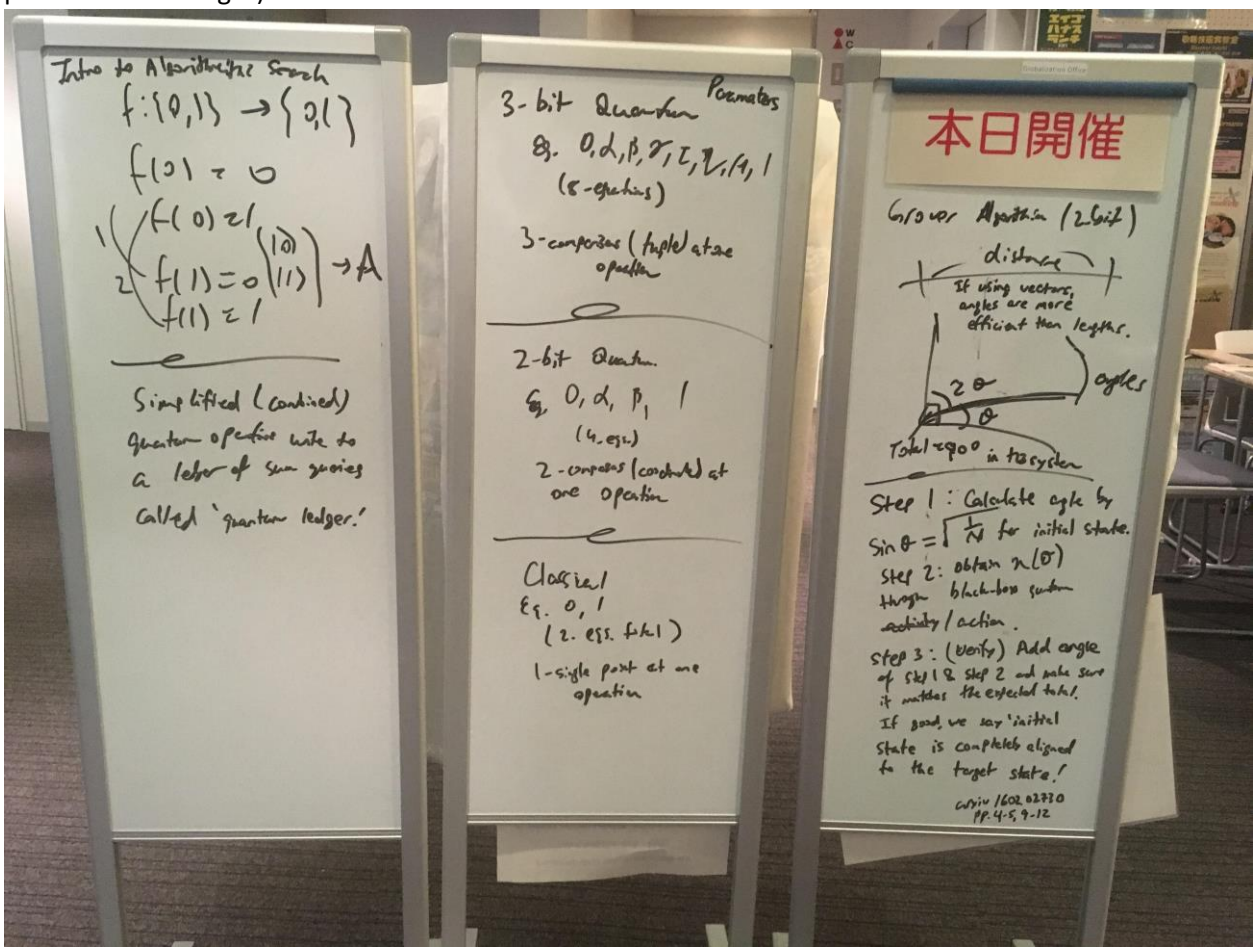
In quantum form, the mathematical structure of vectors allows us to organize that couple of equations into one single vector {f(0)=f(1)=0, f(0)=f(1)=1}. We may call this vector [A] for simplicity. For quantum operations, we combine (simplify) operations into a ledger of all sum queries, which conveniently is called a "quantum ledger." Our world has 1-bit classical calculations (equations 0 and 1) which allow us to use the relatively efficient, "binary search function" for example - where one single point is operated upon in each iteration to find (after some finite iterations and time) the target point. Quantum computers are currently (if only as a mind experiment) feasible in 2-bit and 3-bit form. 2-bit quantum can handle 0, α, β, and 1 (4 equations) with two comparisons (coordinate) operations possible in each iteration. 3-bit quantum has up to eight equations - 0, α, β, γ, τ, η, μ, and 1. At any one iteration, we may use a three comparison (tuple) state to complete search operations.

One possible algorithm used for quantum search databases is Grover's Algorithm (typically completed in 2-bit quantum behavior). Again, classical computer's operations depend on distance to conduct search algorithms. Put in a form for visualizations; we may say that vector search operations make use of angles and black-box fuzzy logic to solve angular calculations of vectors. Indeed, this requires that there be a target state, preferably one where we know what the final condition looks

like in a translated 'quantum machine form').* Grover algorithm can operate at up to O[(2a)√(n)] (alternatively, O(n^(1/2a))-type) efficiency, where “a” is the number of quantum bits.

The general idea is as follows. First, calculate the initial state's angle between vectors by using the equation sin(θ)=√(1/n). The second step is to obtain n(θ) [n, an arbitrary positive integer constant] through some black-box quantum actions. These are equations dictated by differential equations and much higher math than necessary to explain for this course, so I omit detailed explanations. However, note there are some parallels to the ability to solve some matrix equations with m [elementary matrix] + n [eigenvector] from elementary linear algebra. (See the next step.) If needed, conduct some √(n) iteration, until one reaches the expected final state. Finally, there is an optional verification step at the end. Here, we can add the angles of all iterations of all actions (see note on linear algebra above) and make sure it matches the expected total. If it corrects (and indeed, matching an expected 'target state') then we can say it is "aligned" to such "target state." QED.

*The following ***figure 2*** may help elucidate my statement about distances and angles (esp. see the panel on the far right).



A mind-map of my thoughts to craft the answer written above. Written at the GO Tutors Office, Komaba. Many thanks to them for putting up with my takeover of their event boards.

Sources:

https://stackoverflow.com/questions/23986566/. “How to Properly Display Chinese Characters.”
https://medium.com/@JasonWyatt/squeezing-performance-from-sqlite-indexes-indexes-c4e175f3c346/. [Feinstein], “Squeezing Performance from SQLite: Indexes? Indexes!”
arXiv/1602.02730 [Giri, Korepin], "A Review of Quantum Search Algorithms," 2016