Let's Build a Simple Database

Writing a sqlite clone from scratch in C

Overview

View on GitHub (pull requests welcome)

Part 11 - Recursively Searching the B-Tree

< Part 10 - Splitting a Leaf Node

Part 12 - Scanning a Multi-Level B-Tree >

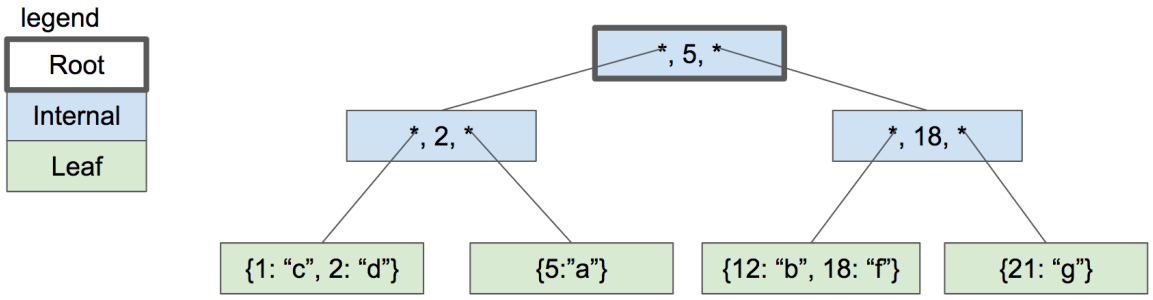
Last time we ended with an error inserting our 15th row:

```
db > insert 15 user15 person15@example.com
Need to implement searching an internal node
```

First, replace the code stub with a new function call.

```
   if (get_node_type(root_node) == NODE_LEAF) {
     return leaf_node_find(table, root_page_num, key);
   } else {
-    printf("Need to implement searching an internal node\n");
-    exit(EXIT_FAILURE);
+    return internal_node_find(table, root_page_num, key);
   }
 }
```

This function will perform binary search to find the child that should contain the given key. Remember that the key to the right of each child pointer is the maximum key contained by that child.



three-level btree

So our binary search compares the key to find and the key to the right of the child pointer:

```
+Cursor* internal_node_find(Table* table, uint32_t page_num, uir
+  void* node = get_page(table->pager, page_num);
+  uint32_t num_keys = *internal_node_num_keys(node);
+
+  /* Binary search to find index of child to search */
+  uint32_t min_index = 0;
+  uint32_t max_index = num_keys; /* there is one more child tha
+
+  while (min_index != max_index) {
+    uint32_t index = (min_index + max_index) / 2;
+    uint32_t key_to_right = *internal_node_key(node, index);
+    if (key_to_right >= key) {
+      max_index = index;
+    } else {
+      min_index = index + 1;
+    }
+  }
```

Also remember that the children of an internal node can be either leaf nodes or more internal nodes. After we find the correct child, call the appropriate search function on it:

```
+  uint32_t child_num = *internal_node_child(node, min_index);
+  void* child = get_page(table->pager, child_num);
+  switch (get_node_type(child)) {
+    case NODE_LEAF:
```

```
+      return leaf_node_find(table, child_num, key);
+    case NODE_INTERNAL:
+      return internal_node_find(table, child_num, key);
+  }
+}
```

Tests

Now inserting a key into a multi-node btree no longer results in an error. And we can update our test:

```
       "    - 12",
       "    - 13",
       "    - 14",
-      "db > Need to implement searching an internal node",
+      "db > Executed.",
+      "db > ",
     ])
   end
```

I also think it's time we revisit another test. The one that tries inserting 1400 rows. It still errors, but the error message is new. Right now, our tests don't handle it very well when the program crashes. If that happens, let's just use the output we've gotten so far:

```
     raw_output = nil
     IO.popen("./db test.db", "r+") do |pipe|
       commands.each do |command|
-        pipe.puts command
+        begin
+          pipe.puts command
+        rescue Errno::EPIPE
+          break
+        end
       end

       pipe.close_write
```

And that reveals that our 1400-row test outputs this error:

```
      end
      script << ".exit"
      result = run_script(script)
-     expect(result[-2]).to eq('db > Error: Table full.')
+     expect(result.last(2)).to match_array([
+       "db > Executed.",
+       "db > Need to implement updating parent after split",
+     ])
    end
```

Looks like that's next on our to-do list!

< Part 10 - Splitting a Leaf Node

Part 12 - Scanning a Multi-Level B-Tree >
