Let's Build a Simple Database

Writing a sqlite clone from scratch in C

Overview

View on GitHub (pull requests welcome)

Part 9 - Binary Search and Duplicate Keys

< Part 8 - B-Tree Leaf Node Format

Part 10 - Splitting a Leaf Node >

Last time we noted that we're still storing keys in unsorted order. We're going to fix that problem, plus detect and reject duplicate keys.

Right now, our `execute_insert()` function always chooses to insert at the end of the table. Instead, we should search the table for the correct place to insert, then insert there. If the key already exists there, return an error.

```
ExecuteResult execute_insert(Statement* statement, Table* table)
   void* node = get_page(table->pager, table->root_page_num);
-  if ((*leaf_node_num_cells(node) >= LEAF_NODE_MAX_CELLS)) {
+  uint32_t num_cells = (*leaf_node_num_cells(node));
+  if (num_cells >= LEAF_NODE_MAX_CELLS) {
     return EXECUTE_TABLE_FULL;
   }

   Row* row_to_insert = &(statement->row_to_insert);
-  Cursor* cursor = table_end(table);
+  uint32_t key_to_insert = row_to_insert->id;
+  Cursor* cursor = table_find(table, key_to_insert);
+
+  if (cursor->cell_num < num_cells) {
+    uint32_t key_at_index = *leaf_node_key(node, cursor->cell_r
+    if (key_at_index == key_to_insert) {
```

```
+      return EXECUTE_DUPLICATE_KEY;
+    }
+  }

   leaf_node_insert(cursor, row_to_insert->id, row_to_insert);
```

We don't need the `table_end()` function anymore.

```
-Cursor* table_end(Table* table) {
-  Cursor* cursor = malloc(sizeof(Cursor));
-  cursor->table = table;
-  cursor->page_num = table->root_page_num;
-
-  void* root_node = get_page(table->pager, table->root_page_num
-  uint32_t num_cells = *leaf_node_num_cells(root_node);
-  cursor->cell_num = num_cells;
-  cursor->end_of_table = true;
-
-  return cursor;
-}
```

We'll replace it with a method that searches the tree for a given key.

```
+/*
+Return the position of the given key.
+If the key is not present, return the position
+where it should be inserted
+*/
+Cursor* table_find(Table* table, uint32_t key) {
+  uint32_t root_page_num = table->root_page_num;
+  void* root_node = get_page(table->pager, root_page_num);
+
+  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);
+  }
+}
```

I'm stubbing out the branch for internal nodes because we haven't implemented internal nodes yet. We can search the leaf node with binary search.

```
+Cursor* leaf_node_find(Table* table, uint32_t page_num, uint32_
+  void* node = get_page(table->pager, page_num);
+  uint32_t num_cells = *leaf_node_num_cells(node);
+
+  Cursor* cursor = malloc(sizeof(Cursor));
+  cursor->table = table;
+  cursor->page_num = page_num;
+
+  // Binary search
+  uint32_t min_index = 0;
+  uint32_t one_past_max_index = num_cells;
+  while (one_past_max_index != min_index) {
+    uint32_t index = (min_index + one_past_max_index) / 2;
+    uint32_t key_at_index = *leaf_node_key(node, index);
+    if (key == key_at_index) {
+      cursor->cell_num = index;
+      return cursor;
+    }
+    if (key < key_at_index) {
```

```
+      one_past_max_index = index;
+    } else {
+      min_index = index + 1;
+    }
+  }
+
+  cursor->cell_num = min_index;
+  return cursor;
+}
```

This will either return

- the position of the key,
- the position of another key that we'll need to move if we want to insert the new key, or
- the position one past the last key

Since we're now checking node type, we need functions to get and set that value in a node.

```
+NodeType get_node_type(void* node) {
+  uint8_t value = *((uint8_t*)(node + NODE_TYPE_OFFSET));
+  return (NodeType)value;
+}
+
+void set_node_type(void* node, NodeType type) {
+  uint8_t value = type;
+  *((uint8_t*)(node + NODE_TYPE_OFFSET)) = value;
+}
```

We have to cast to `uint8_t` first to ensure it's serialized as a single byte.

We also need to initialize node type.

```
-void initialize_leaf_node(void* node) { *leaf_node_num_cells(nc
+void initialize_leaf_node(void* node) {
+  set_node_type(node, NODE_LEAF);
+  *leaf_node_num_cells(node) = 0;
+}
```

Lastly, we need to make and handle a new error code.

```
-enum ExecuteResult_t { EXECUTE_SUCCESS, EXECUTE_TABLE_FULL };
+enum ExecuteResult_t {
+  EXECUTE_SUCCESS,
+  EXECUTE_DUPLICATE_KEY,
+  EXECUTE_TABLE_FULL
+};
```

```
       case (EXECUTE_SUCCESS):
         printf("Executed.\n");
         break;
+      case (EXECUTE_DUPLICATE_KEY):
+        printf("Error: Duplicate key.\n");
+        break;
       case (EXECUTE_TABLE_FULL):
         printf("Error: Table full.\n");
         break;
```

With these changes, our test can change to check for sorted order:

```
       "db > Executed.",
       "db > Tree:",
       "leaf (size 3)",
-      "  - 0 : 3",
-      "  - 1 : 1",
-      "  - 2 : 2",
+      "  - 0 : 1",
+      "  - 1 : 2",
+      "  - 2 : 3",
       "db > "
     ])
   end
```

And we can add a new test for duplicate keys:

```
+  it 'prints an error message if there is a duplicate id' do
+    script = [
```

```
+      "insert 1 user1 person1@example.com",
+      "insert 1 user1 person1@example.com",
+      "select",
+      ".exit",
+    ]
+    result = run_script(script)
+    expect(result).to match_array([
+      "db > Executed.",
+      "db > Error: Duplicate key.",
+      "db > (1, user1, person1@example.com)",
+      "Executed.",
+      "db > ",
+    ])
+  end
```

That's it! Next up: implement splitting leaf nodes and creating internal nodes.

< Part 8 - B-Tree Leaf Node Format

Part 10 - Splitting a Leaf Node >
