Let's Build a Simple Database

Writing a sqlite clone from scratch in C

Overview

View on GitHub (pull requests welcome)

Part 6 - The Cursor Abstraction

< Part 5 - Persistence to Disk

Part 7 - Introduction to the B-Tree >

This should be a shorter part than the last one. We're just going to refactor a bit to make it easier to start the B-Tree implementation.

We're going to add a `Cursor` object which represents a location in the table. Things you might want to do with cursors:

- Create a cursor at the beginning of the table
- Create a cursor at the end of the table
- Access the row the cursor is pointing to
- Advance the cursor to the next row

Those are the behaviors we're going to implement now. Later, we will also want to:

- Delete the row pointed to by a cursor
- Modify the row pointed to by a cursor
- Search a table for a given ID, and create a cursor pointing to the row with that ID

Without further ado, here's the `Cursor` type:

```
+typedef struct {
+  Table* table;
+  uint32_t row_num;
+  bool end_of_table;  // Indicates a position one past the last
```

```
+} Cursor;
```

Given our current table data structure, all you need to identify a location in a table is the row number.

A cursor also has a reference to the table it's part of (so our cursor functions can take just the cursor as a parameter).

Finally, it has a boolean called `end_of_table`. This is so we can represent a position past the end of the table (which is somewhere we may want to insert a row).

`table_start()` and `table_end()` create new cursors:

```
+Cursor* table_start(Table* table) {
+  Cursor* cursor = malloc(sizeof(Cursor));
+  cursor->table = table;
+  cursor->row_num = 0;
+  cursor->end_of_table = (table->num_rows == 0);
+
+  return cursor;
+}
+
+Cursor* table_end(Table* table) {
+  Cursor* cursor = malloc(sizeof(Cursor));
+  cursor->table = table;
+  cursor->row_num = table->num_rows;
+  cursor->end_of_table = true;
+
+  return cursor;
+}
```

Our `row_slot()` function will become `cursor_value()`, which returns a pointer to the position described by the cursor:

```
-void* row_slot(Table* table, uint32_t row_num) {
+void* cursor_value(Cursor* cursor) {
+  uint32_t row_num = cursor->row_num;
   uint32_t page_num = row_num / ROWS_PER_PAGE;
```

```
-  void* page = get_page(table->pager, page_num);
+  void* page = get_page(cursor->table->pager, page_num);
   uint32_t row_offset = row_num % ROWS_PER_PAGE;
   uint32_t byte_offset = row_offset * ROW_SIZE;
   return page + byte_offset;
 }
```

Advancing the cursor in our current table structure is as simple as incrementing the row number. This will be a bit more complicated in a B-tree.

```
+void cursor_advance(Cursor* cursor) {
+  cursor->row_num += 1;
+  if (cursor->row_num >= cursor->table->num_rows) {
+    cursor->end_of_table = true;
+  }
+}
```

Finally we can change our "virtual machine" methods to use the cursor abstraction. When inserting a row, we open a cursor at the end of table, write to that cursor location, then close the cursor.

```
   Row* row_to_insert = &(statement->row_to_insert);
+  Cursor* cursor = table_end(table);

-  serialize_row(row_to_insert, row_slot(table, table->num_rows)
+  serialize_row(row_to_insert, cursor_value(cursor));
   table->num_rows += 1;

+  free(cursor);
+
   return EXECUTE_SUCCESS;
 }
```

When selecting all rows in the table, we open a cursor at the start of the table, print the row, then advance the cursor to the next row. Repeat until we've reached the end of the table.

```
 ExecuteResult execute_select(Statement* statement, Table* table
```

```
+  Cursor* cursor = table_start(table);
+
   Row row;
-  for (uint32_t i = 0; i < table->num_rows; i++) {
-    deserialize_row(row_slot(table, i), &row);
+  while (!(cursor->end_of_table)) {
+    deserialize_row(cursor_value(cursor), &row);
     print_row(&row);
+    cursor_advance(cursor);
   }
+
+  free(cursor);
+
   return EXECUTE_SUCCESS;
 }
```

Alright, that's it! Like I said, this was a shorter refactor that should help us as we rewrite our table data structure into a B-Tree. `execute_select()` and `execute_insert()` can interact with the table entirely through the cursor without assuming anything about how the table is stored.

Here's the complete diff to this part:

```
@@ -78,6 +78,13 @@ struct {
 } Table;

+typedef struct {
+  Table* table;
+  uint32_t row_num;
+  bool end_of_table; // Indicates a position one past the last
+} Cursor;
+
 void print_row(Row* row) {
     printf("(%d, %s, %s)\n", row->id, row->username, row->emai
 }
@@ -126,12 +133,38 @@ void* get_page(Pager* pager, uint32_t page
     return pager->pages[page_num];
 }
```

```
-void* row_slot(Table* table, uint32_t row_num) {
-  uint32_t page_num = row_num / ROWS_PER_PAGE;
-  void *page = get_page(table->pager, page_num);
-  uint32_t row_offset = row_num % ROWS_PER_PAGE;
-  uint32_t byte_offset = row_offset * ROW_SIZE;
-  return page + byte_offset;
+Cursor* table_start(Table* table) {
+  Cursor* cursor = malloc(sizeof(Cursor));
+  cursor->table = table;
+  cursor->row_num = 0;
+  cursor->end_of_table = (table->num_rows == 0);
+
+  return cursor;
+}
+
+Cursor* table_end(Table* table) {
+  Cursor* cursor = malloc(sizeof(Cursor));
+  cursor->table = table;
+  cursor->row_num = table->num_rows;
+  cursor->end_of_table = true;
+
+  return cursor;
+}
+
+void* cursor_value(Cursor* cursor) {
+  uint32_t row_num = cursor->row_num;
+  uint32_t page_num = row_num / ROWS_PER_PAGE;
+  void *page = get_page(cursor->table->pager, page_num);
+  uint32_t row_offset = row_num % ROWS_PER_PAGE;
+  uint32_t byte_offset = row_offset * ROW_SIZE;
+  return page + byte_offset;
+}
+
+void cursor_advance(Cursor* cursor) {
+  cursor->row_num += 1;
+  if (cursor->row_num >= cursor->table->num_rows) {
+    cursor->end_of_table = true;
+  }
 }
```

```
 Pager* pager_open(const char* filename) {
@@ -327,19 +360,28 @@ ExecuteResult execute_insert(Statement* st
     }

   Row* row_to_insert = &(statement->row_to_insert);
+  Cursor* cursor = table_end(table);

-  serialize_row(row_to_insert, row_slot(table, table->num_rows)
+  serialize_row(row_to_insert, cursor_value(cursor));
   table->num_rows += 1;

+  free(cursor);
+
   return EXECUTE_SUCCESS;
 }

 ExecuteResult execute_select(Statement* statement, Table* table
+  Cursor* cursor = table_start(table);
+
   Row row;
-  for (uint32_t i = 0; i < table->num_rows; i++) {
-     deserialize_row(row_slot(table, i), &row);
+  while (!(cursor->end_of_table)) {
+     deserialize_row(cursor_value(cursor), &row);
      print_row(&row);
+     cursor_advance(cursor);
   }
+
+  free(cursor);
+
   return EXECUTE_SUCCESS;
 }
```

< Part 5 - Persistence to Disk

Part 7 - Introduction to the B-Tree >

This project is maintained by cstack

Hosted on GitHub Pages — Theme by orderedlist