Program High-Level Design

Platform

Our record-generating program is written in Ruby in two files -- `models.rb` and `record_generator.rb`. The dependencies for the program are listed in the `Gemfile`, and can easily be installed by running '`bundle install`' on a system configured for Rubygems and Bundler.

Overview

Data is generated with the help of the Faker gem, a popular Ruby library for generating pseudo-random, authentic-looking strings for names, addresses, zip codes, and company titles, as well as standard Lorem Ipsum strings. Those values that cannot be generated through Faker (such as the 20 character numeric IDs) are generated by custom functions that make use of Ruby's various randomization tools.

The program is divided into three main portions:

1. `models.rb` contains models for each individual relation. This is just a data object with the name of the relation.
2. `record_generator.rb` generates an array of models for each relation. For some relations (e.g. Customer, Library) this process is self-contained. Others (e.g. Loan, Item, Checkout) use data from a previous relation array as part of the generation process. This is to ensure consistency in the data (see Dependant Generation). Randomly-selected objects from a dependent array are used for dependent fields, such as picking out libraries for customers to access.
3. `record_generator.rb` then writes the arrays of data objects to a file in the form:

   ```
   insert into Relation (field1, field2, ...) values
   (field1Value, field2Value, ...),
   (field1Value, field2Value, ...)
   ```

   and so on. This file can then be run as a SQL script to insert the records into the database.

All data is stored in main memory as it is being generated. The generator functions run in O(n^2) time complexity and O(n) space complexity, which are more than sufficient for the numbers of records we plan on generating.

Dependent Generation

Some of our tables must have values that match up to values in other tables in order to have non-empty joins. For example, take the Accesses table. Each record in this table should have a customerID that is in the Customer table and a libraryName that is in the Library table. Otherwise, the table would make no sense. Thus, the generator function for Accesses takes as arguments the already-generated customers and libraries arrays. Then, for each record in Accesses, the function selects a random customerID from customers and a random

libraryName from libraries. The resulting array will join perfectly with the Customer and Library tables. Note that there can (and almost certainly will) be cases with dangling tuples -- for instance, the generator does not guarantee that every Customer will have an entry in the Accesses table. Likewise, some Employees are intentionally left without an associated library (representing employees that work for the system as a whole).

Additionally, the generation of some tables requires the inspection of others. For example, loans cannot randomly sample from books and libraries, because there is a good chance the book picked may not actually belong to the randomly chosen library. In such cases, the independent variable, such as the library, is randomly chosen; after this the dependant set, such as books, is then filtered so that only valid books can then be chosen.

The dependencies for the generator functions are as follows (there are no circular dependencies):

- Library depends on nothing
- Customer depends on nothing
- Item depends on Library
- Employee depends on Library
- Loan depends on Library and Item
- Accesses depends on Customer and Library
- Checkout depends on Customer, Library, and Item