Project Step 4: CRUD Functionalities

liam beckman

02 august 2018

URL

`http://flip3.engr.oregonstate.edu:12345/`

Feedback by the peer reviewers and graders

Project Step 3 Draft version: DML + HTML:

Your DMQ looks good and syntactically correct. The website also looks good and contains all the required functionalities. Though one small thing that I would suggest changing is the Browse and Update pages. These two pages are exactly the same, so I would suggest either creating an update button in the Browse page, or changing the Browse page so that the user cannot edit the entries.

Project Step 3 Final Version: DML + HTML

the website is missing the feature to show authors/languages/etc

Actions based on the feedback

- Merged Browse and Update page, so that the user can update in the Browse page.
- Added authors, languages, operating systems, and sources page on website.

Upgrades to the Draft version

I have changed the following items in my submission:

- Merged Browse and Update page, so that the user can update in the Browse page.
- Updated Entity-Relationship diagram and schema by adding source and operating system entities.
- Reorganized entities to make a more logical relationship.
- Added authors, languages, operating systems, and sources page on website.

a) Outline

My project will be a simple organizational database for previous computer science projects and technical scripts. Oftentimes, I encounter a task or project that looks or sounds familiar to a previous project in some regard, but I have difficulty finding the source. Putting previously encountered projects into a relational database will allow me to keep track of what language it was written in, what its purpose is, original authors and contributors, etc. Overall, this database will serve as a rudimentary search engine for code and projects that I have encountered, and want to keep accessible in the future.

I expect this will achieve the following goals:

1. Faster recall for established projects, allowing for easier retrieval from a relational database.
2. Greater aid in creating new projects.
3. More effective dissemination of code through sharing of pertinent information.

b) Database Outline, in Words

Entities

program → the code itself, and the most important entity.

- id: an auto-incrementing primary key for the program (INT).
- name: the name of the program (VARCHAR).
- purpose: a small description that gives a large overview of the project in question, and what its goals are (VARCHAR).
- url: this will be a homepage that users can read about the program (e.g. `https://magit.vc/`). Not to be confused with the program's source, which is the place to download or clone the program (e.g. `https://github.com/magit/magit`) (VARCHAR).
- version: this will simply be a property which can be used to test whether the project is maintained currently, or is no longer in development. This will be a VARCHAR instead of DECIMAL to be more flexible with versioning formats (e.g. be able to use semantic versioning) (VARCHAR).

- license: what license was used for the code, and whether it is an open-source project or a proprietary project. This will let the user expect what kinds of restrictions are placed on the software (VARCHAR).

author → a list of people or organizations that have developed or maintained the project (many of the projects I have in mind will come from open source initiatives, or my own school projects).

- id: an auto-incriminating primary key for the author (INT).
- name: the name/username of the authors and contributors (VARCHAR).
- website: the website of the authors and contributors (VARCHAR).
- FOREIGN KEY: references a table of authors/contributors. The Author table will have a foreign key reference to the main Program table.

language → the primary language(s) used in the program (e.g. C++, Python, Rust, etc.).

- id: an auto-incriminating primary key for the language (INT).
- name: the name of the language or language family (VARCHAR).
- website: the website of the program one can visit for more information (e.g. `https://www.rust-lang.org/en-US/`) (VARCHAR).
- FOREIGN KEY: references a table of languages. The Language table will have a foreign key reference to the main Program table.

operating system (os) → the operating system(s) that the program can run on.

- id: an auto-incriminating primary key for the operating system (INT).
- name: the name of the operating system (VARCHAR).
- url: the url to the website of the operating system that users can get more information at (e.g. `https://www.kernel.org/`) (VARCHAR).
- FOREIGN KEY: references a table of operating systems. The Operating System table will have a foreign key reference to the main Program table.

source (src) → the location that the program is hosted at, and which can be used to download the program. This encompasses primary download mirrors.

- id: an auto-incrementing primary key for the source (INT).
- url: the URL to the code, either the source code itself, or some compressed format (e.g. zip, tar.gz). Not to be confused with the program's homepage (VARCHAR).
- type: the type of download (e.g. git, SVN, compressed download, etc.) (VARCHAR).
- FOREIGN KEY: references a table of sources. The Source table will have a foreign key reference to the main Program table.

Relationships

Program - Author

- *Many program can be written by many authors.* So programs and authors have a Many-to-Many relationship.

- *Authors can exist without programs.* So authors have partial participation with programs.
- *A program can not exist without authors.* So programs have total participation with authors.

Program - Language

- *A program can not exist without languages.* So programs have total participation with languages.
- *Languages can exist without programs.* So languages have partial participation with programs.
- *Many program can be written in many languages.* So programs and languages have a Many-to-Many relationship.

Program - Operating System

- *A program can exist without operating systems.* So programs have partial participation with operating systems. In this scenario, we define programs such that they do not have to run on an operating system to be a program. This may be the case for pseudocode, or other "theoretical" programs.
- *Operating systems can not exist without programs.* So operating systems have partial participation with programs. Slightly pedantic, but we will define operating systems as a collection of programs. In this case, an operating system can not exist without a program.
- *Many program can be written in many operating systems.* So programs and operating systems have a Many-to-Many relationship.

Program - Source

- *A program can exist without sources.* So programs have partial participation with sources. A program may not be hosted anywhere.
- *Sources can exist without programs.* So sources have partial participation with programs. An empty repository can exist.
- *One program can be hosted at one source.* So programs and sources have a One-to-One relationship. For the sake of simplicity, we will say that every program has a "primary" source at which it is hosted, even though mirrors and multiple repositories may be used. So instead of listing all sources a program can be downloaded from or read about, we will say it has a single source.

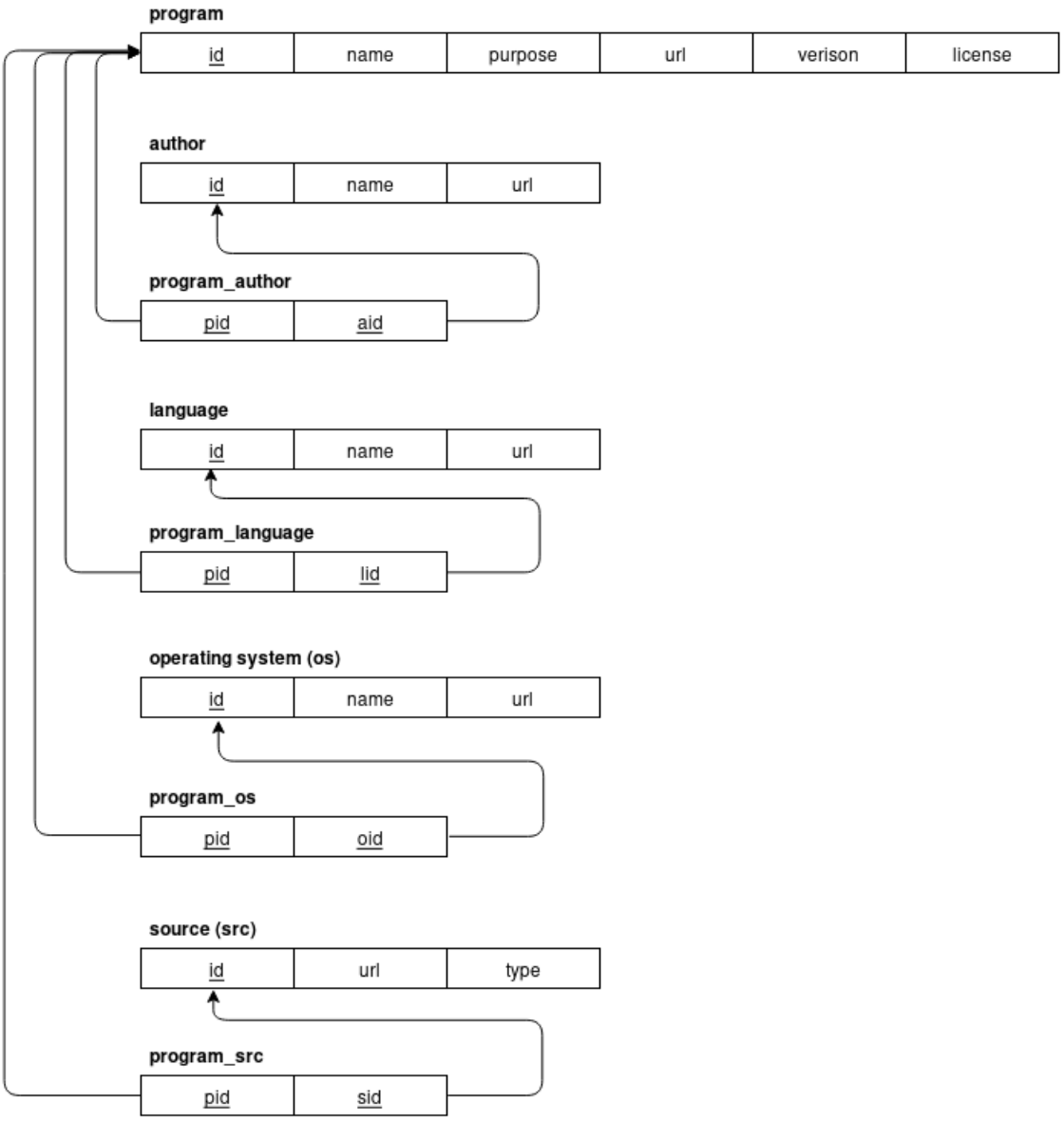

Figure 1: Schema

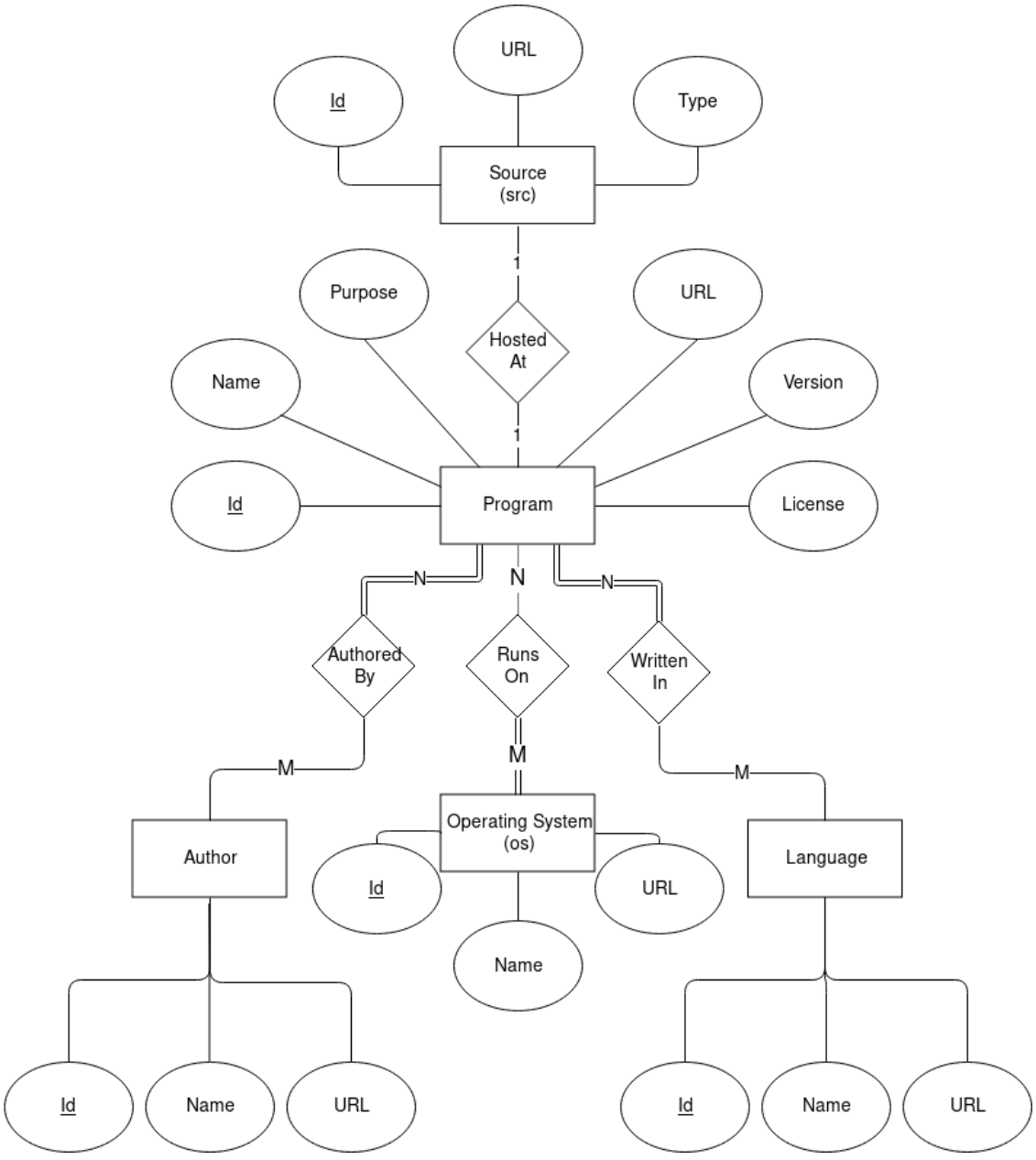

Figure 2: Entity-Relationship Diagram