Meta-DB Database Requirements

Authors: Tian Liu, Sergiy Kolodyazhnyy, Jericha Bradley

Course: CS3810

Instructor: Prof. Lakhani

Contents:

1. Introduction

1.1 Purpose and Objectives

The goals of this application:

- to provide a convenient way for users to search for files based on particular metadata
- to provide a convenient way for users to show metadata of specific files
- to provide a more powerful alternative to the standard locatedb on Linux/Unix systems
- reduce complexity of searching for files based on their features for end users

The application aims at desktop use, however the application suits well for server environment as well. In such environments as file storage/archiving or file sharing service the application provides means to locate particular file on a more fine-grained sets of requirements. In the image/audio processing environment where multiple files and encoding types frequently have to coexist, the application can provide means of better control over indexing the processed data. The application can be used as standalone command as well as in scripting applications, where information written to stdout stream may be redirected to other applications via standard unix pipelines or temporary files.

2. Architectural Requirements

2.1 Overview of the Database Architecture

The application implementation should perform 4 core functionalities:

- load file and corresponding metadata information
- vacuum (i.e. remove) the file entires out of the database whose path no longer exists on the filesystem. While the file itself may exist on the filesystem, its path makes it no longer a valid entry
- update the database - records of files which have been changed (based on shasum) should be updated and newly added files to the filesystem should added to the database
- perform user-defined search queries based on minimal set of criteria:
    - search file by file major type and single metadata criteria
    - search file by minor type and single metadata criteria
    - display metadata of specific file.

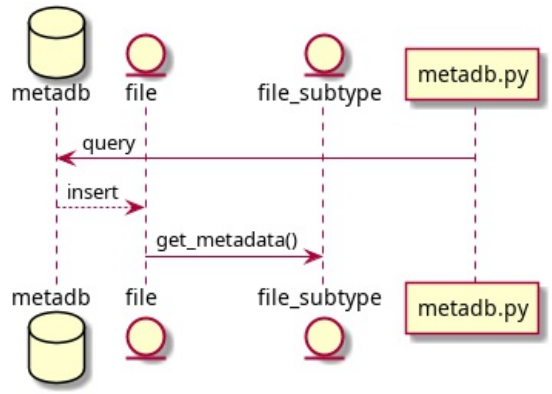
In order to speed up search queries, the database takes advantage of indexes. In order to maximize performance, indexes generally should be created on columns with high cardinality (i.e. uniqueness of data). In case of the `file` table highest cardinality can be achieved either via shasum or path. While searching based on shasum is rare for average desktop users, path string would be more frequent and logical. Within each subtype highest cardinality is again achieved via path of the file. Indexes can help even in low cardinality cases, such as when we only have very few Image records with minor type `png`. For these reasons, the database will have the following indexes:

- `file_path` index
- `subtype_minor_type` index for each subtype
- `metadata` index for each subtype

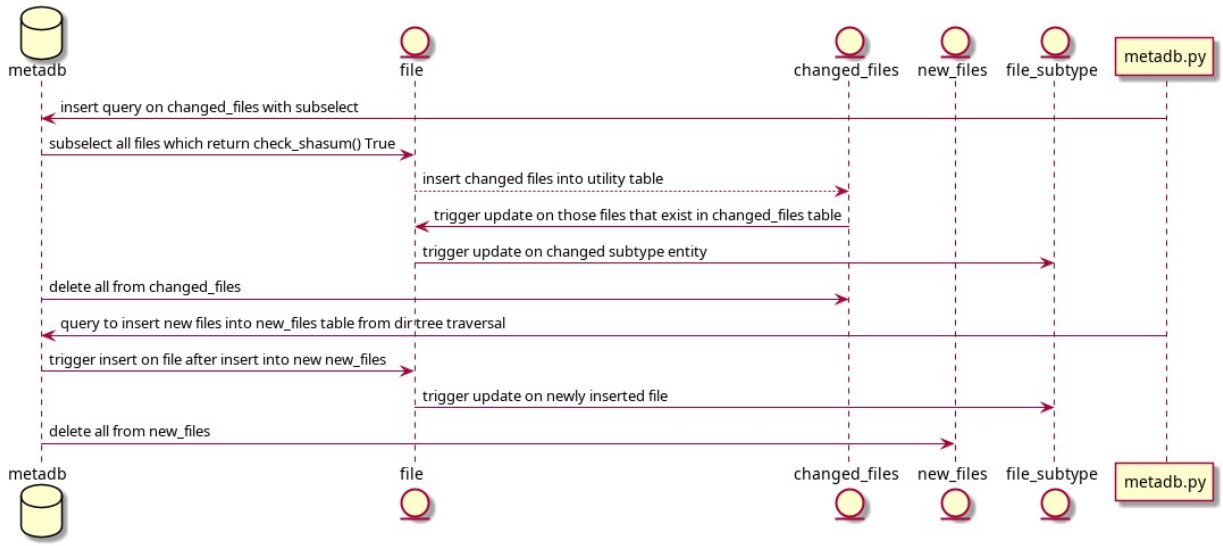
Modeling via supertypes and subtype is used to model the relationship between files and their metadata. This is further described in section 3.

Information Flow within the application is described in the following diagrams:

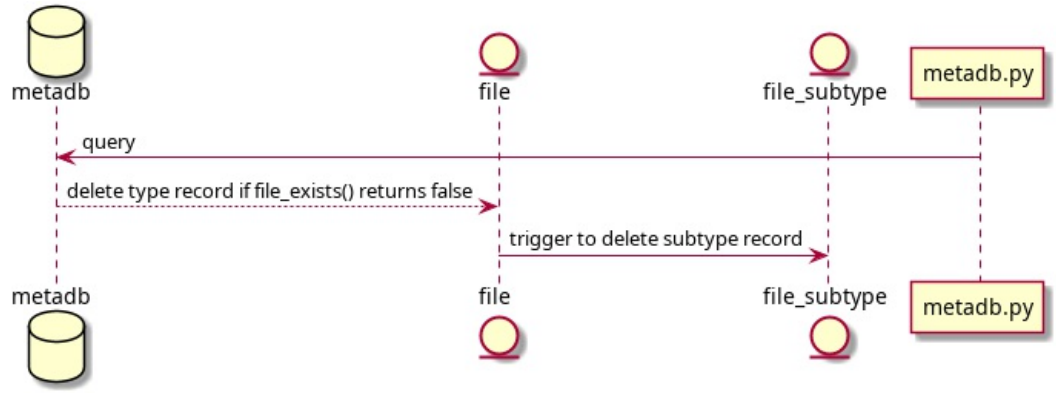
- `load` functionality:



- `update` functionality:



- `vacuum` functionality:



2.2 Software Dependencies

Primary software used within this application

- SQLite 3
- Python 3

Since the SQLite is intended for data aggregation and incapable of implementing system functions needed for the data discovery, such as traversing the directory tree for each user directory or discovering metadata for particular file, the bulk of the job is done by Python 3 modules. In particular, the following modules are necessary:

- `PIL` for image metadata discovery
- `os` for directory tree traversal and basic OS interfacing
- `audioread` for audiofile metadata discovery
- `subprocess` to take advantage of external commands
- `OrderedDict` to create json string with consistent order of items for performance benefits

The application is intended to work within Linux environment therefore user should have a proper Linux distribution, preferably with GNU set of utilities available. Since the application interfaces with the user via command-line a proper terminal emulator or standard Linux console is required.

2.3 Hardware Dependencies

Database does not particularly depend on specific hardware. Average desktop computer is sufficient. For best performance a hard drive with fast access time or solid state drive would be recommended. SQLite database format is cross-platform which also implies it does not depend on particular filesystem. Thus, Linux systems with any available filesystem such as ext4 (most common), or btrfs, or xfs are completely acceptable. the database does not store any filesystem-specific metadata.

2.4 Limitations and Constraints

The choice of the database software for this project influences the constraints on how SQL code will be structured. For instance, SQLite does not support conditional triggers. However, it is possible to implement conditional trigger via multiple triggers that fire when single specific condition is met. For instance, where in MySQL we could do

```
CREATE TRIGGER record_type AFTER INSERT ON file
FOR EACH ROW BEGIN
    IF (New.filetype = 'text') THEN
        INSERT INTO text_file (path) VALUES ( New.path );
    ELSE IF (New.filetype = 'image') THEN
        INSERT INTO image_file (path) VALUES  (New.path);
    END IF;
END;
```

in SQLite we will have to split this into two triggers as so

```
    CREATE TRIGGER record_text AFTER INSERT ON file
    WHEN New.filetype = 'text'
    BEGIN
        INSERT INTO text_file   (path) VALUES (New.path);
    END;

    CREATE TRIGGER record_image AFTER INSERT ON file
    WHEN New.filetype = 'text'
    BEGIN
        INSERT INTO image_file (path) VALUES (New.path);
    END;
```

Additionally, SQLite does not have stored procedures, which is a conscious choice on SQLite developer's side. The core idea is to offset functional and procedural capabilities onto the application's language. For that purpose, SQLite allows creating user-defined functions in the host language and connecting them with the database from within the application code. In particular, we will need to create several Python functions to be used from within the database itself, such as `check_shasum()`, `file_exists()`, and metadata-specific functions such as `get_exif()` to return exif information for jpeg files.

Additionally SQLite maximum number of rows in a table is 2^{64}, maximum database size of 140 terabytes at page size of 65536 bytes.

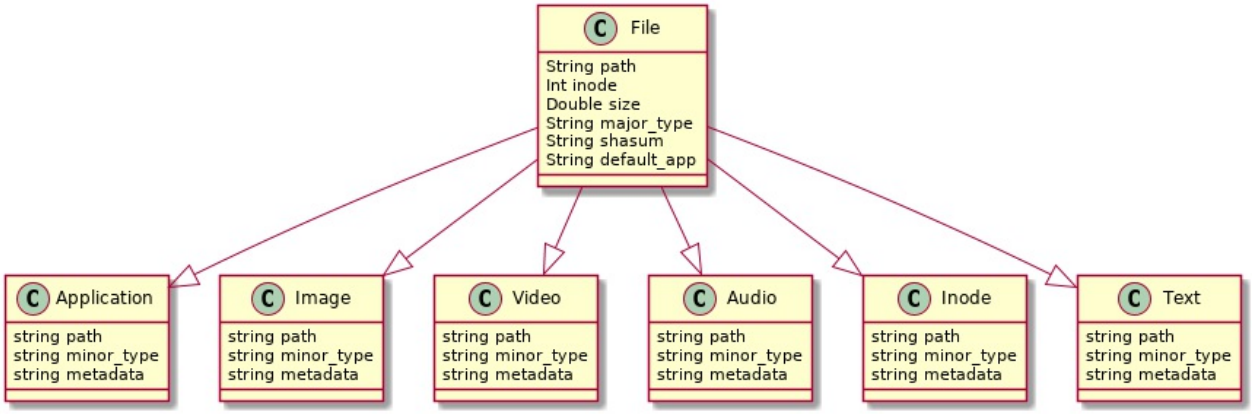
3. Database Model

3.1 Entities

The database models files and relation with their metadata via inheritance model. All objects that are discovered by the application via traversal of the user's directories are going to be files, but each file has particular 'is a' relationship with major filetype. Natural key to uniquely identify a file is pathname, which will be used as both primary key and foreign key to link entities.

The major supertype in `file` table contains common information for all files, such as full filesystem path to file, access and modification time, inode number. Subtype entity inherits pathname from the supertype. In order to reduce complexity, the metadata will be stored as json string. At basic level the application is meant to display full metadata. The advantage of json approach is that the command-line front end written in Python can convert json string to Python's native dictionary datastructure to provide more fine grained approach to metadata searching. This reduces complexity on the SQL side of the development.

Inheritance of files and filetype is further illustrated in the diagram below. To facilitate queries another entity default_application will be used.



3.2 Required Tables

For entity relationship:

- File, supertypes
- text
- applications
- image
- video
- audio
- inode
- default_application

Auxilary tables:

- changed_files
- new_files

4. Database-User Interfacing

The application is written as command-line application, which implies the application can interface with the user via text. The user can control which database queries are issued via command-line switches such as `-l`, `-u`, `-v`. For this reason a proper terminal emulator or default Linux virtual console is required. The application can also be part of other scripting solutions in order to provide more advanced functionality or be combined with other tools.

5. Future Considerations and Extending

- While the application's core aim is to provide metadata aggregation and lookup, the database is capable of additional functionality which may be potentially considered in future releases. As the demands of the users of the application grow, the users may require new sets of features. In particular, common complain of the desktop users within Linux ecosystem is the lack of tagging for files and searching based on the tags. The SQLite database is very much suitable for such purpose, and such feature may be added in future. Such feature would extend the database from supporting aggregation of metadata attached to the files themselves to providing capability of aggregating user-defined metadata.

- Further performance improvements could be achieved via leveraging Python's regex searching capabilities in parsing the json metadata strings.