Biblio

- Dan Crankshaw
- Cain Lu
- Paul O'Neill
- Jiefeng Zhai

Agenda

- Overview
- Database Code
- Typical Workflow
- Demo
- Searching Code

Motivation

- Strange File Names
- Music libraries, so why not text-based libraries?
- Non-hierarchical relationships
- Organize and view in one application

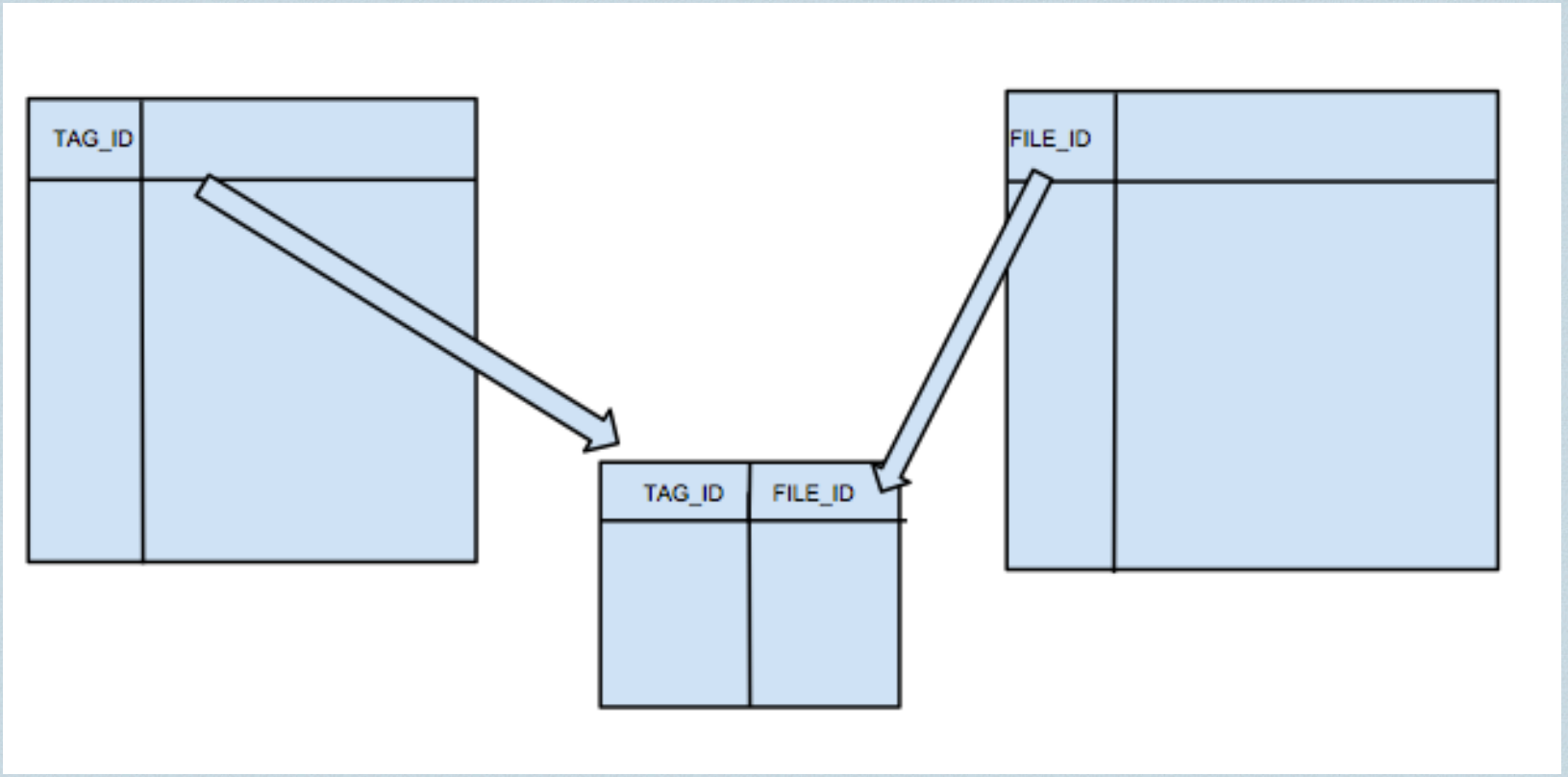
Overview

- Organize and read text-based files
- Organization centers around tags
- Tags are non-hierarchical (think directed graph not file system tree)
- Files and tags have a many-to-many relationship

Hibernate

```
@Entity
@Table(name = "TAG")
public class Tag implements Comparable<Tag> {

    @Id
    @GenericGenerator(name = "generator", strategy = "increment")
    @GeneratedValue(generator = "generator")
    @Column(name = "TAG_ID")
    private int id;

    @Column(name = "NAME", nullable = false)
    private String name;

    @ManyToMany(fetch = FetchType.EAGER)
    @JoinTable(
     name = "TAG_FILEMETADATA",
     joinColumns = @JoinColumn(
           name = "TAG_ID",
           referencedColumnName = "TAG_ID"),
    inverseJoinColumns = @JoinColumn(
           name = "FMETA_ID",
           referencedColumnName = "FMETA_ID"))
    private Set<FileMetadata> taggedFiles;
```

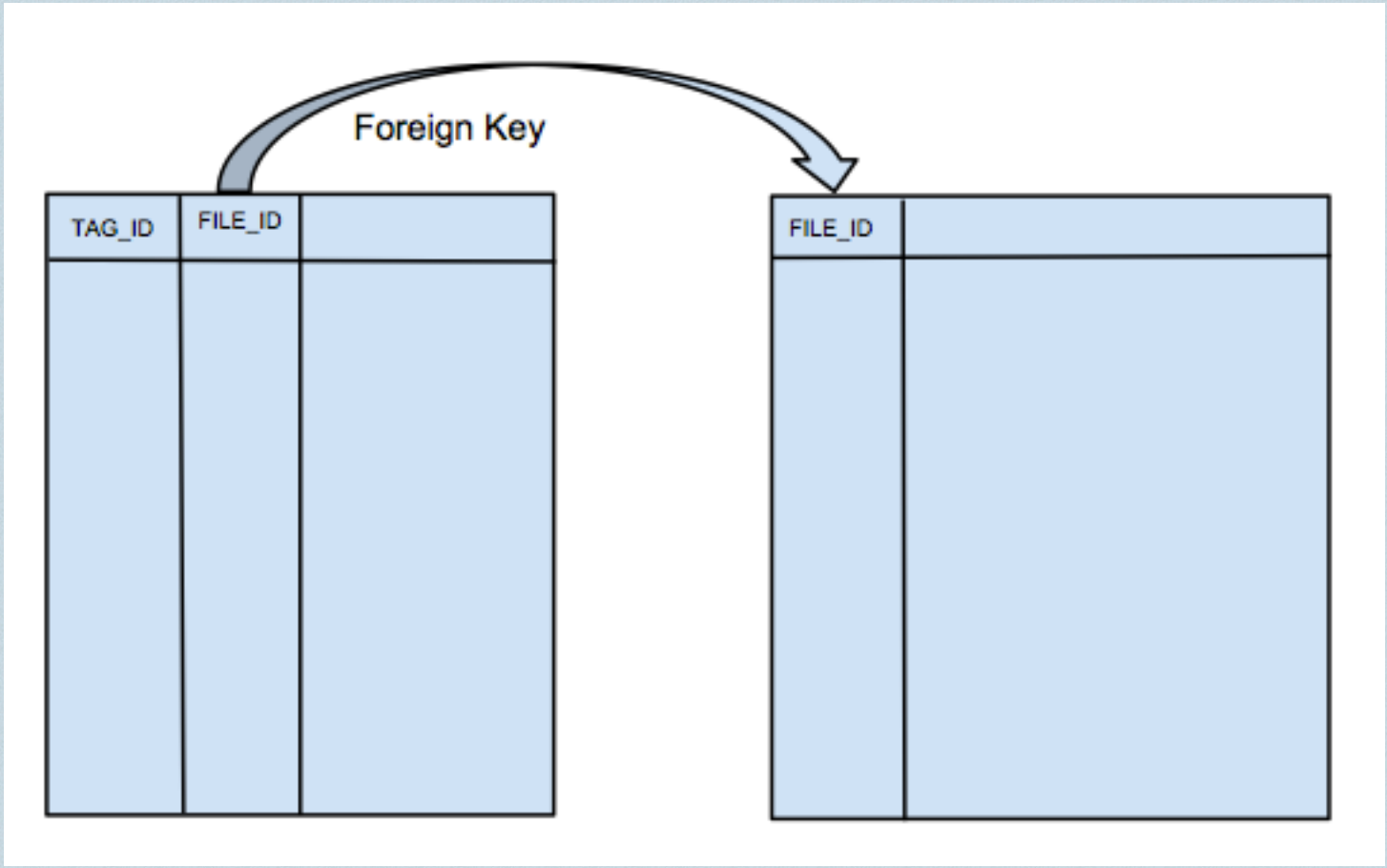
DB Schema



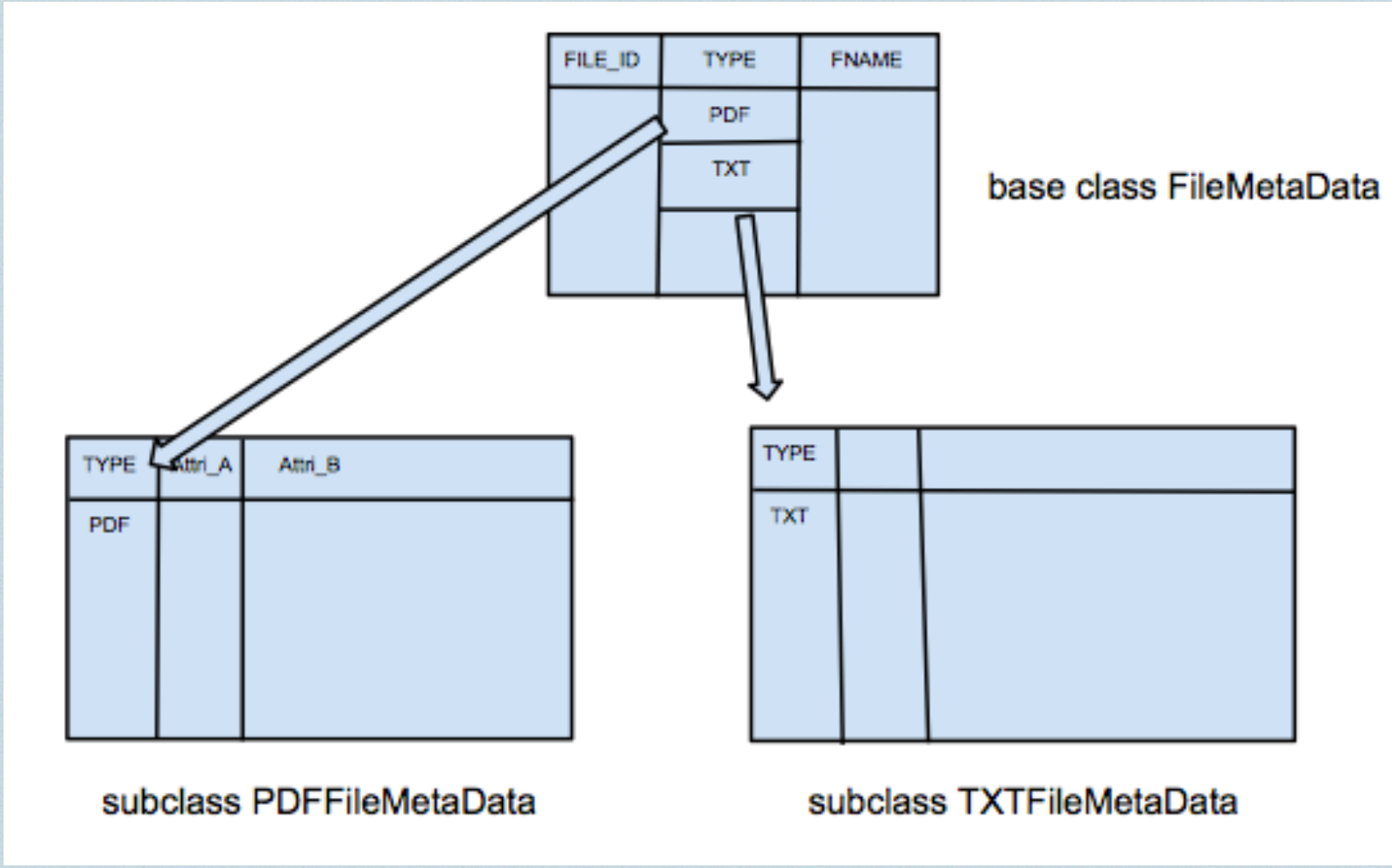
Hibernate

```
@Entity
@Table( name = "BOOKMARK" )
public class Bookmark {

     @ManyToOne(optional=false, fetch=FetchType.EAGER)
     @JoinColumn(name="FMETA_ID", nullable=false)
     private FileMetadata file;

     @OneToOne(optional=false, fetch=FetchType.EAGER)
     @JoinColumn(name="LOC_ID")
      private Location location;
```

DB Schema



DB Schema


FILE_ID
TYPE
FNAME
PDF
TXT
base class FileMetaData
TYPE
Attri_A
Attri_B
PDF
subclass PDFFileMetaData
TYPE
TXT
subclass TXTFileMetaData

Accessing the DB

```
SessionFactory sessionFactory = new
Configuration().configure().buildSessionFactory();
Session session = sessionFactory.getCurrentSession();

session.beginTransaction();

session.save(new Location());

session.getTransaction().commit();
```

Workflow

- User has a set of criteria (tags) in mind
- Searches for each tag
- Builds a set of tags
- Filters by selected tags to get the intersection of files tagged by those tags and their children

Important Classes

SearchManager

- Responsible for querying the database and doing the search
- All the “under the hood” work is here.
- Notifies other objects when searches are completed

SearchResultsListener

- Listens for the files matching a search

SearchTagsListener

- Listens for tags matching the search

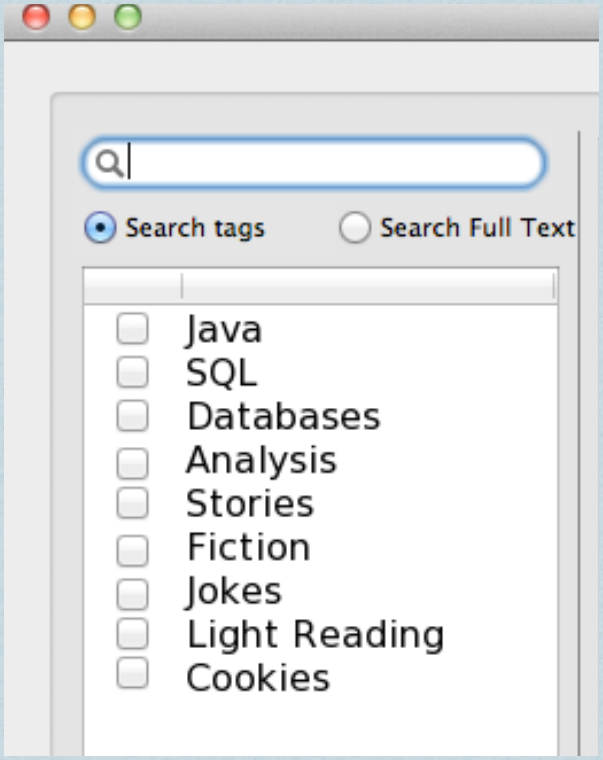
Important Classes

SearchPanel

- Responsible for getting user queries

TagTableModel

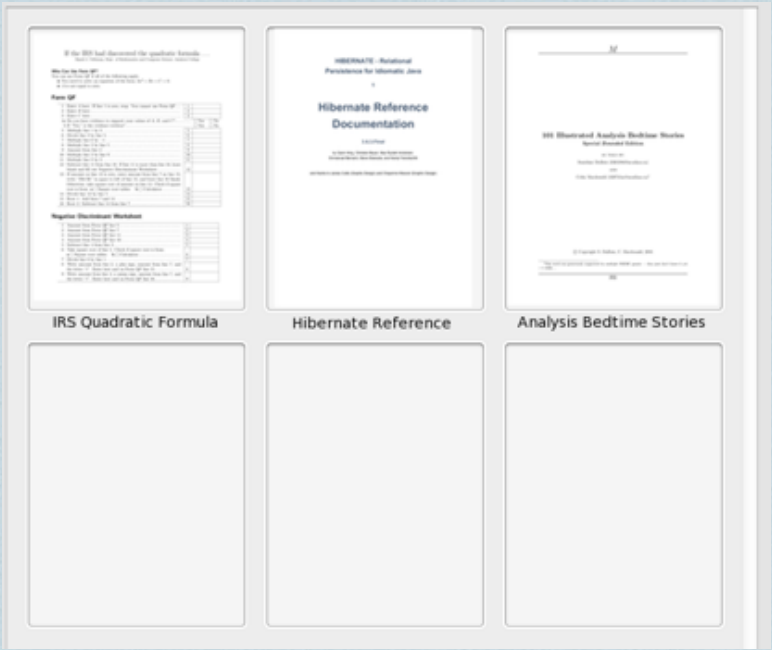
- Provides the JTable with information
- Keeps track of tags selected for filtering
- Implements SearchTagsListener



Important Classes

SearchResultsPreviewPanel

- Responsible for displaying the files matching the search
- Implements SearchResultsListener



Demo!

Search For Tag

```
public void searchTags(String searchTerm) {

    if (searchTerm.contains(":"))
        searchCategory(searchTerm);
    else {
        Session session = sessionFactory.getCurrentSession();
        session.beginTransaction();

        //TODO cleanse the input, using sql parameters instead of string concatenation
        Criteria crit = session.createCriteria(Tag.class).add(
                Restrictions.like("name", "%" + searchTerm + "%"));
        @SuppressWarnings("unchecked")
        List<Tag> results = (List<Tag>) crit.list();
        session.getTransaction().commit();
        fireSearchTags(results);
    }
}
```

Search Tag in Category

```
private void searchCategory(String term) {
    List<Tag> results = null;
    // only search if colon appears exactly once in searchterm
    if (term.indexOf(":") == term.lastIndexOf(":")) {

        Set<Tag> potentialTags = new TreeSet<Tag>();
        results = new ArrayList<Tag>();

        String[] split = term.split(":");
        //we have already verified that a colon appears exactly once in the searchTerm, so we
        //know that String[] split will have exactly two items in it
        String category = split[0].trim();
        String tagName = split[1].trim();

        Session session = sessionFactory.getCurrentSession();
        session.beginTransaction();
        Criteria crit = session.createCriteria(Category.class).add(
                Restrictions.like("name", category + "%"));
        @SuppressWarnings("unchecked")
        List<Category> cats = (List<Category>) crit.list();
        session.getTransaction().commit();
        for (Category c : cats) {
            potentialTags.addAll(c.getTags());
        }
        for (Tag t : potentialTags) {
            if (t.getName().contains(tagName)) {
                results.add(t);
            }
        }
    }
    fireSearchTags(results);
}
```

Filter by Tags

```
public void filterByTags(Set<Tag> tags) {

    selectedFiles.clear();
    if (tags != null && !tags.isEmpty()) {
        List<Set<FileMetadata>> taggedFiles = new ArrayList<Set<FileMetadata>>();
        Set<FileMetadata> filteredFiles = new HashSet<FileMetadata>();
        for (Tag currentTag : tags) {

            Set<Tag> currentChildren = currentTag.getAllDescendants();
            Set<FileMetadata> currentTaggedFiles = new HashSet<FileMetadata>();
            currentTaggedFiles.addAll(currentTag.getTaggedFiles());
            for (Tag currentChild : currentChildren) {
                currentTaggedFiles.addAll(currentChild.getTaggedFiles());
            }
            taggedFiles.add(currentTaggedFiles);
        }
        // get the list of files matching the first tag or its descendants
        filteredFiles.addAll(taggedFiles.get(0));
        // take the intersection of those files with the files matching the
        // rest of the tags
        for (int i = 1; i < taggedFiles.size(); i++) {
            filteredFiles.retainAll(taggedFiles.get(i));
        }
        selectedFiles.addAll(filteredFiles);
        Collections.sort(selectedFiles, new Comparator<FileMetadata>() {
            @Override
            public int compare(FileMetadata a, FileMetadata b) {
                return a.getName().compareToIgnoreCase(b.getName());
            }
        });
    }
    fireSearchResult();
}
```

Full Text Search

```
public void searchText(String searchTerm) {
    /* ************************************************************************
     * We need to figure out a way (or if we even need to) normalize our
     * search results that doesn't automatically give higher precedence to
     * longer documents - Dan
     *
     * Also, eventually searching can be done in a separate thread(s)
     */
    List<ResultsPair> pairs = new ArrayList<ResultsPair>();
    for (FileMetadata file : selectedFiles) {
        int freq = 0;
        try {
            freq = file.searchText(searchTerm);
        } catch (Exception e) {
            e.printStackTrace();
            // TODO: maybe launch a dialog warning about a corrupted file -
            // Dan
        }
        if (freq != 0) {
            // remove all files with 0 occurrences
            pairs.add(new ResultsPair(freq, file));
        }
    }
    Collections.sort(pairs);
    List<FileMetadata> matchedFiles = new ArrayList<FileMetadata>();
    for (ResultsPair pair : pairs) {
        matchedFiles.add(pair.file);
    }
    selectedFiles = matchedFiles;

    fireSearchResult();
}
```

Finishing Touches

- Finish import support
- RSS Reader
- Expand to support EPUB
- Add notes to files