

# doctag

Tagging and parsing tag queries in Python

---

Dan Turkel

June 10th, 2019

Background

A **tag** is a common form of metadata for organizing files (or **documents**).

Tags are a convenient supplement to a file system because they are not hierarchical like folders are.

Tags and documents have a **many-to-many** relationship: one document may have many tags, and one tag may be applied to many documents.

Tagging systems naturally motivate several additional actions beyond tagging and untagging:

Tagging systems naturally motivate several additional actions beyond tagging and untagging:

**show tags:** display all tags for a given  
document

**show docs:** display all documents with a  
given tag

Tagging systems naturally motivate several additional actions beyond tagging and untagging:

**show tags:** display all tags for a given document

**show docs:** display all documents with a given tag

**merge tags:** replace all instances of **old** tag with **new**

**delete tag:** remove all usages of a given tag

**clean doc:** remove all tags from a given doc

Tagging systems naturally motivate several additional actions beyond tagging and untagging:

**show tags:** display all tags for a given document

**show docs:** display all documents with a given tag

**merge tags:** replace all instances of **old** tag with **new**

**delete tag:** remove all usages of a given tag

**clean doc:** remove all tags from a given doc

**query docs:** display all documents matching a tag query

One way to represent a set of tagged documents is through relational tables.<sup>1</sup>

---

<sup>1</sup><http://howto.philippkeller.com/2005/04/24/Tags-Database-schemas/>



One way to represent a set of tagged documents is through relational tables.<sup>1</sup>

docs
doc_id
doc_name

tag_map
tag_map_id
tag_id
doc_id

tags
tag_id
tag_name

---

<sup>1</sup><http://howto.philippkeller.com/2005/04/24/Tags-Database-schemas/>

## Data Structures: Database (Example)

docs		tag_map			tags	
doc_id	doc_name	tag_map_id	tag_id	doc_id	tag_id	tag_name
1	movies.txt	1	1	1	1	list
2	books.txt	2	1	2	2	learning
3	school.txt	3	2	2		
		4	2	3		

We can tag/untag with **INSERT/DELETE**, delete tags or docs with **DELETE**, and show tags or docs **SELECT**. Merging tags can be done with a targeted **INSERT** followed by a **DELETE**, and building tag queries is simply a matter of building **WHERE** clauses for **SELECT** statements.

Another option for representing tagged documents is with an **index** and an **inverted index**.<sup>2</sup>

The index maps documents to tags, while the inverted index maps tags to documents.

Inverted indexes are used in NLP<sup>3</sup> and search<sup>4</sup> for quickly finding documents which contain specific user-defined content.

---

<sup>2</sup><https://stackoverflow.com/a/24993487>

<sup>3</sup><https://nlp.stanford.edu/IR-book/html/htmledition/a-first-take-at-building-an-inverted-index-1.html>

<sup>4</sup><https://www.elastic.co/guide/en/elasticsearch/guide/current/inverted-index.html>

## Data Structures: Inverted Index (Example)

index	
doc	tags
movies.txt	list
books.txt	list, learning
school.txt	learning

inverse index	
tag	docs
list	movies.txt, books.txt
learning	books.txt, school.txt

Tag and untag operations require writing to both indexes, but *show tags* and *show docs* operations become trivial. Deleting a tag is roughly the same process as in the tabular solution, and merging tags can be done by re-tagging in the index and unioning in the inverted index.

Querying has to be implemented through a series of intersections, unions, and negations.

doctag

**doctag** is a Python library for building index/inverted index tagging systems and performing actions on those systems.

The library includes a **TagIndex** class which stores the index and inverted index and implements methods for tagging and retrieval.

**ultrajson**<sup>5</sup> is used to (optionally) serialize and deserialize the **TagIndex** to disk *really fast*.

**boolean.py**<sup>6</sup> is used to parse arbitrarily complex tag queries, like:

“(list and learning) or (not work)”

---

<sup>5</sup><https://github.com/bastikr/boolean.py>

<sup>6</sup><https://github.com/esnme/ultrajson>

See `notebooks/features.ipynb`

See `notebooks/performance.ipynb`



Executing queries in doctag is a two-step proces.

First, the boolean.py library parses the query string and creates an expression object that we can interpret programmatically.

Then doctag recursively performs set operations as it traverses the expression.

doctag is on Github: <https://github.com/daturkel/doctag>