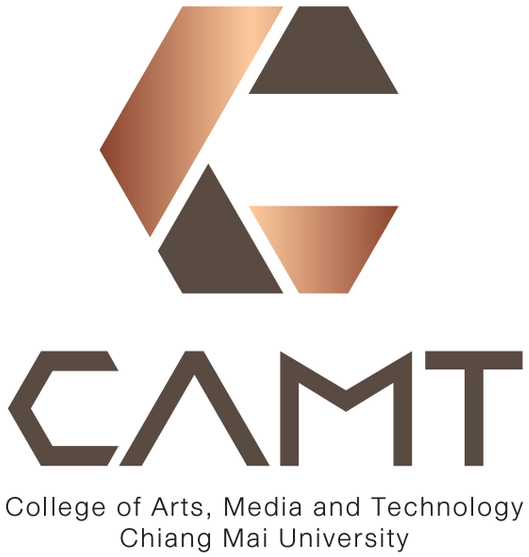
SE 481 Introduction to Information Retrieval

Module #1 — Overview



Passakorn Phannachitta, D.Eng.

passakorn.p@cmu.ac.th

College of Arts, Media and Technology

Chiang Mai University, Chiangmai, Thailand

Information Retrieval (IR)

- **Information retrieval** is the science of searching for information in a document, searching for documents themselves, and also searching for the metadata that describes data, and for databases of texts, images or sounds.

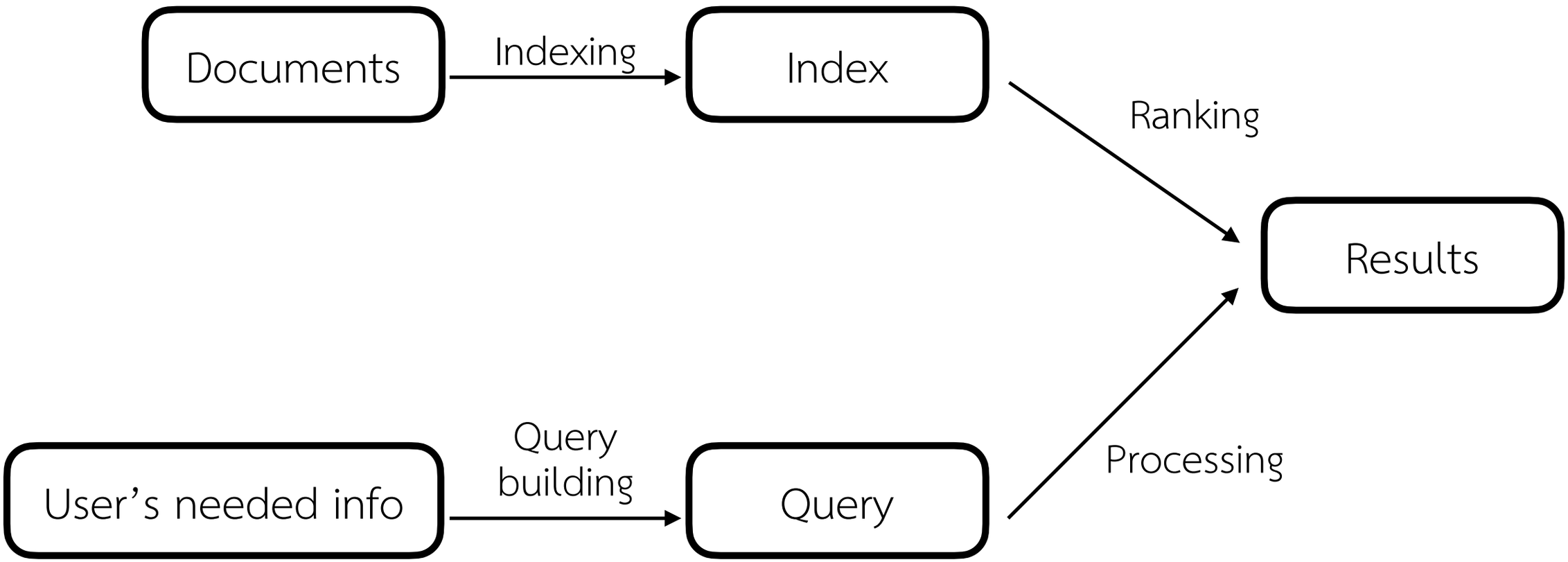
— Wikipedia

Shorter version

- IR is to connect between:
  - Users —> who want information
  - Documents —> provide information

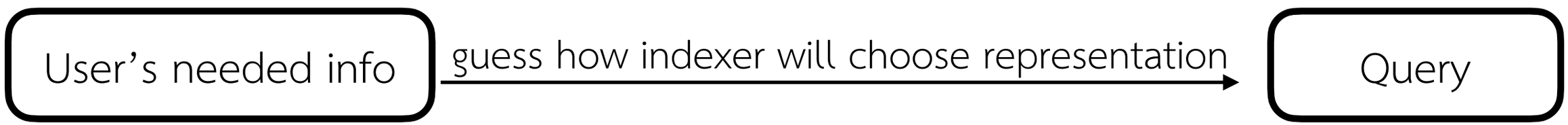
Users <> Documents

- IR



The connecting point

- IR



Another view

- IR is a search for similarity:
  - Documents and queries
  - Documents in collection
  - Users

Basic assumption of IR

- Information is stored in a set of documents
- An IR system retrieves documents with information that is **most relevant** to the **user's information need** and helps the user complete a **task**
- Data are born **unstructured**

Find an answer for any question

Which character has the most deaths to his/her name?

Asked 7 years, 6 months ago Active 2 years, 10 months ago Viewed 134k times

15

In the Dragon Ball series, we see tons of people dying and being revived by the dragon balls. This triggered my curiosity to know which character has the most deaths to his/her name in the whole Dragon Ball series. Who is it, and how many times did he/she die?

dragon-ball-series

Ref: https://anime.stackexchange.com/questions/8851/which-character-has-the-most-deaths-to-his-her-name

Ref: https://en.wikipedia.org/wiki/List_of_Dragon_Ball_characters

For a question sounded more SE

Posted by u/MossRock42 7 years ago

1

What is the best programming language for career development as a software engineer?

This is a question that I've been wondering for a while now. My current job is mostly doing C#/MVC Web Applications. But I am considering trying for a better job elsewhere. I see a lot of job postings asking for things like Ruby, Python, Java, NodeJS, etc. So if you had to pick one which one would it be?

14 Comments Award Share Save Hide Report 57% Upvoted

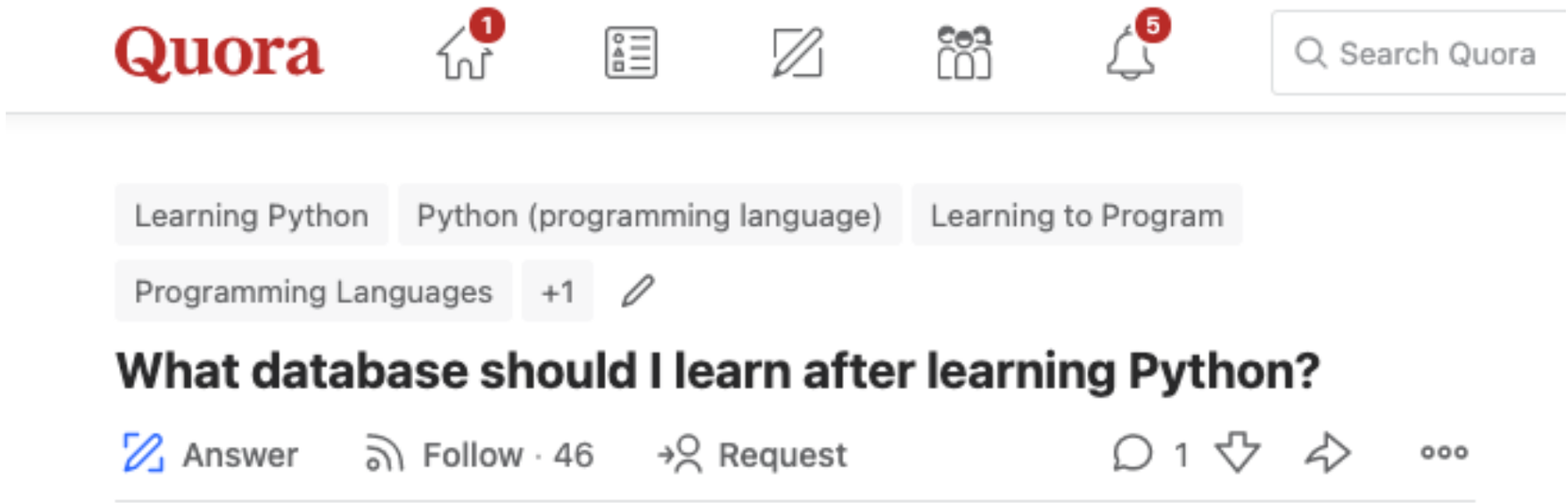
Ref: https://www.reddit.com/r/cscareerquestions/comments/2povgz/what_is_the_best_programming_language_for_career/

How to approach the problem

- Some web portal also have a summarized view for the query but if we want to make our own version, we shall be able to

  - **Crawl** several job search websites, e.g., Indeed and Glassdoor targeting programming-related carrier positions;
  - **List** all the thread-pair of (required programming language, salary);
  - **Summarize** the search results.

Next question

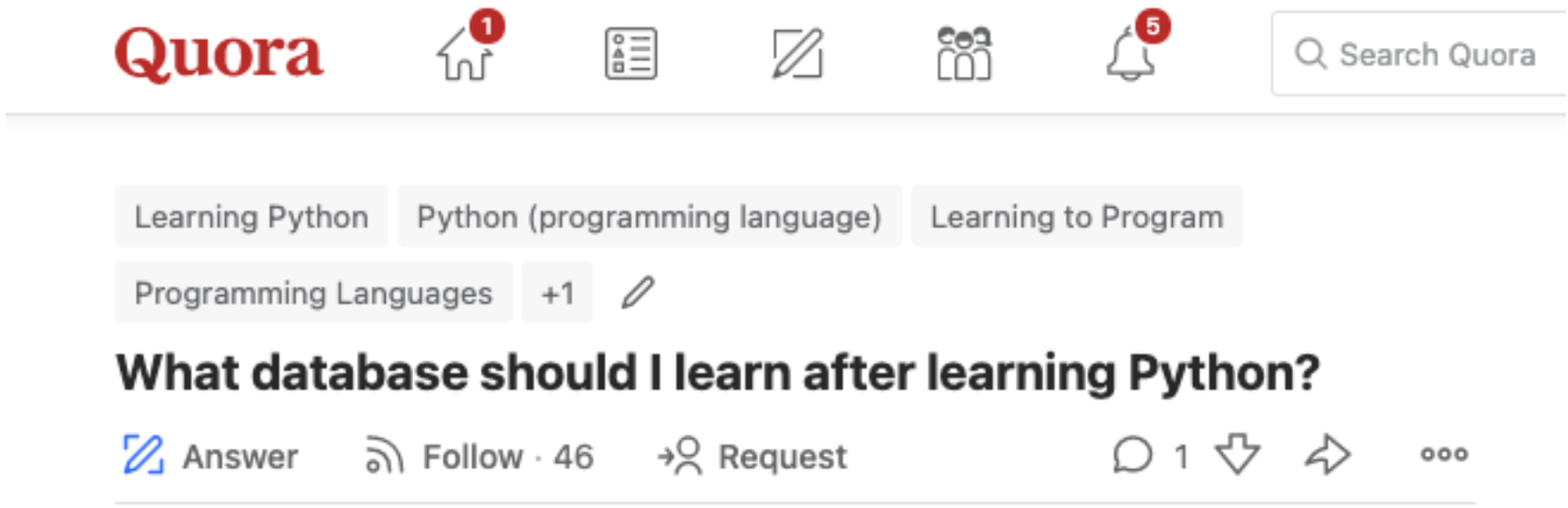


How to approach the problem

- Using similar approach to the earlier question, we may have to
  - **Crawl** several job search websites
  - **List** all the thread-pair of (python, a name of a database system)
  - **Summarize** the search results.

- If we do all these at the time we query.
  - Super slow
  - Question can not be easily modified and instantly get the answer.
  - Etc

A simple walkthrough



- Let’s play with a data snapshot provided in Kaggle https://www.kaggle.com/jobspikr/software-developer-job-listings-usa
  - It is 10,000 records of job posting entires from Indeed.com

A simple walkthrough

Data Explorer

41.51 MB

software_developer_united_states_1971_20...

software_developer_united_sta...

Detail | Compact | Column | 3 of 21 columns

job_title	company_...	job_descri...
Sr. Software Developer	Aerotek	The chosen Sr. Software Developer will be part of a larger engineering team developing software for ...
C# Lead Software Developer	3coast	Position: C# Lead Software Developer Location: Middletown, NJ Compensation: $90,000 - $110,0...
Senior Software Developer	s.com	Senior Software Developer Hoboken, NJ Starts as 9-12 month contract - possibility to extend Required...

Q: What DB should I learn after python?

- **Sub question:** How many positions required **MySql** and **Python**
- Create word list for each job entry

```
01  def get_and_clean_data():
02      data = pd.read_csv('src/resource/software_developer_united_states_1971_20191023_1.csv')
03      description = data['job_description']
04      cleaned_description = description.apply(lambda s: s.translate(str.maketrans('', '',
string.punctuation + u'\xa0')))
05      cleaned_description = cleaned_description.apply(lambda s: s.lower())
06      cleaned_description = cleaned_description.apply(lambda s:
s.translate(str.maketrans(string.whitespace, ' '*len(string.whitespace), '')))
07      cleaned_description = cleaned_description.drop_duplicates()
08      return cleaned_description
09
10  def simple_tokenize(data):
11      cleaned_description = data.apply(lambda s: [x.strip() for x in s.split()])
12      return cleaned_description
13
14  def parse_job_description():
15      cleaned_description = get_and_clean_data()
16      cleaned_description = simple_tokenize(cleaned_description)
17      return cleaned_description
```

Q: What DB should I learn after python?

- **Sub question:** How many positions required **MySql** and **Python**

```
1   def count_python_mysql():
2       parsed_description = parse_job_description()
3       count_python = parsed_description.apply(lambda s: 'python' in s).sum()
4       count_mysql = parsed_description.apply(lambda s: 'mysql' in s).sum()
5       print('python: ' + str(count_python) + ' of ' + str(parsed_description.shape[0]))
6       print('mysql: ' + str(count_mysql) + ' of ' + str(parsed_description.shape[0]))
```

python: 1379 of 7540

mysql: 667 of 7540

Q: What db should I learn after python?

- What about other db

```
01  def parse_db():
02      html_doc = requests.get("https://db-engines.com/en/ranking").content
03      soup = BeautifulSoup(html_doc, 'html.parser')
04      db_table = soup.find("table", {"class": "dbi"})
05      all_db = [''.join(s.find('a').findAll(text=True,recursive=False)).strip() for s in
db_table.findAll("th", {"class": "pad-l"})]
06      all_db = list(dict.fromkeys(all_db))
07      db_list = all_db[:10]
08      db_list = [s.lower() for s in db_list]
09      db_list = [[x.strip() for x in s.split()] for s in db_list]
10      return db_list
```

```
>>>parse_db()
[['oracle'], ['mysql'], ['microsoft', 'sql', 'server'], ['postgresql'], ['mongodb'],
['redis'], ['ibm', 'db2'], ['elasticsearch'], ['sqlite'], ['cassandra']]
```

Q: Count of occurrences

```
1   raw = [None] * len(cleaned_db)
2   for i,db in enumerate(cleaned_db):
3       raw[i] = parsed_description.apply(lambda s: np.all([x in s for x in db])).sum()
4       print(' '.join(db) + ': ' + str(raw[i]) + ' of ' + str(parsed_description.shape[0]))
```

oracle: 1392 of 7583

mysql: 667 of 7583

microsoft sql server: 868 of 7583

postgresql: 261 of 7583

mongodb: 296 of 7583

db2: 130 of 7583

redis: 106 of 7583

elasticsearch: 161 of 7583

sqlite: 28 of 7583

microsoft access: 256 of 7583

Q: What db should I learn after python?

```
1   with_python = [None] * len(cleaned_db)
2   for i,db in enumerate(cleaned_db):
3       with_python[i] = parsed_description.apply(lambda s: np.all([x in s for x in db]) and 'python'
in s).sum()
4       print(' '.join(db) + ' + python: ' + str(with_python[i]) + ' of ' +
str(parsed_description.shape[0]))
```

oracle: 1392 of 7583

mysql: 667 of 7583

microsoft sql server: 868 of 7583

postgresql: 261 of 7583

mongodb: 296 of 7583

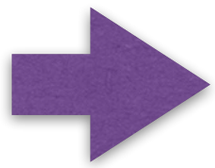
redis: 106 of 7583

ibm db2: 48 of 7583

elasticsearch: 161 of 7583

sqlite: 28 of 7583

cassandra: 142 of 7583



oracle + python: 243 of 7583

mysql + python: 207 of 7583

microsoft sql server + python: 51 of 7583

postgresql + python: 90 of 7583

mongodb + python: 111 of 7583

redis + python: 38 of 7583

ibm db2 + python: 12 of 7583

elasticsearch + python: 73 of 7583

sqlite + python: 7 of 7583

cassandra + python: 49 of 7583

Q: What db should I learn after python?

```
1   for i, db in enumerate(cleaned_db):
2       print(' '.join(db) + ' + python: ' + str(with_python[i]) + ' of ' + str(raw[i]) + ' (' +
str(np.around(with_python[i] / raw[i]*100,2)) + '%)')
```

oracle + python: 243 of 1392 (17.46%)

mysql + python: 207 of 667 (31.03%)

microsoft sql server + python: 51 of 868 (5.88%)

postgresql + python: 90 of 261 (34.48%)

mongodb + python: 111 of 296 (37.5%)

redis + python: 38 of 106 (35.85%)

ibm db2 + python: 12 of 48 (25.0%)

elasticsearch + python: 73 of 161 (45.34%)

sqlite + python: 7 of 28 (25.0%)

cassandra + python: 49 of 142 (34.51%)

Q: What db should I learn after python?

- Some insight

oracle + python: 243 of 1392 (17.46%)	raw #1
mysql + python: 207 of 667 (31.03%)	raw #2
microsoft sql server + python: 51 of 868 (5.88%)	
postgresql + python: 90 of 261 (34.48%)	
mongodb + python: 111 of 296 (37.5%)	raw #3, % #2
redis + python: 38 of 106 (35.85%)	% #3
ibm db2 + python: 12 of 48 (25.0%)	
elasticsearch + python: 73 of 161 (45.34%)	% #1
sqlite + python: 7 of 28 (25.0%)	
cassandra + python: 49 of 142 (34.51%)	

To enable a quick search for other queries

- E.g.,
  - What DB should I learn after python?
  - What DB should I learn after java?
  - Which DB is in demand alongside oracle?
  - What programing language is in demand alongside python?
  - ...

- Indexing
  - Without indexing, it is impossible to process the queries for billion webpages x million users per seconds

Term-document incidence matrices

```
1   lang = [['java'],['python'],['c'],['kotlin'],['swift'],['rust'],['ruby'],['scala'],['julia'],
['lua']]
2   parsed_description = parse_job_description()
3   parsed_db = parse_db()
4   all_terms = lang + parsed_db
5   query_map = pd.DataFrame(parsed_description.apply(lambda s: [1 if np.all([d in s for d in db])
else 0 for db in all_terms]).values.tolist(), columns=[' '.join(d) for d in all_terms])
```

	java	python	c	kotlin	...	sqlite	microsoft access
0	0	0	1	0	...	0	0
1	0	0	1	0	...	0	0
2	0	0	0	0	...	0	0
3	0	0	0	0	...	0	0
4	0	0	1	0	...	0	0
	...	...	...	...	...	...	...
7578	1	1	0	0	...	0	0
7579	1	0	0	0	...	0	0
7580	0	0	1	0	...	0	0
7581	1	0	0	0	...	0	0
7582	1	0	0	0	...	0	0

- 1 = contain term, 0 not contain term.

Querying

- What DB should I learn after java?
- Which DB is in demand alongside oracle?
- What programing language is in demand alongside python?
  - Create a 0-1 vector using bitwise and/or
  - Start with something like:

```
query_map[query_map['java'] > 0].apply(lambda s: np.where(s==1)[0],
axis=1).apply(lambda s: list(query_map.columns[s]))
```

```
10      [java, python, c, oracle, mysql, mongodb]
11                                         [java]
12                           [java, swift, redis]
16                      [java, c, swift, oracle]
19                                 [java, python]
                           ...
7576                               [java, oracle]
7578                 [java, python, oracle, mysql]
7579                               [java, oracle]
7581                        [java, oracle, mysql]
7582                              [java, mongodb]
```

Issue

- Scalability — if #documents reaches 1 billion and each page contain 1000 words,
  - how large the 0/1 metrics would be?
  - How can we store them in the memory for searching?

- **Note:** 1 billion is the #websites in the world in September 2014

Some thoughts

- We need a better tokenizer.
- The 0/1 matrix is extremely sparse.
- We can better represent it.

Inverted index

- The same thing as:

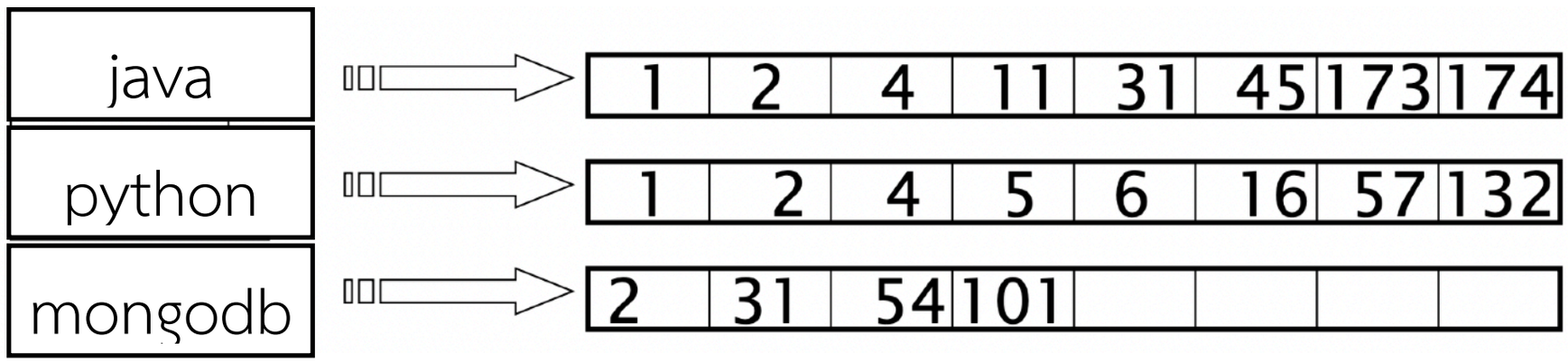
Index

Ref: https://www.userdesignillustrationandtypesetting.com/book-design/index.html

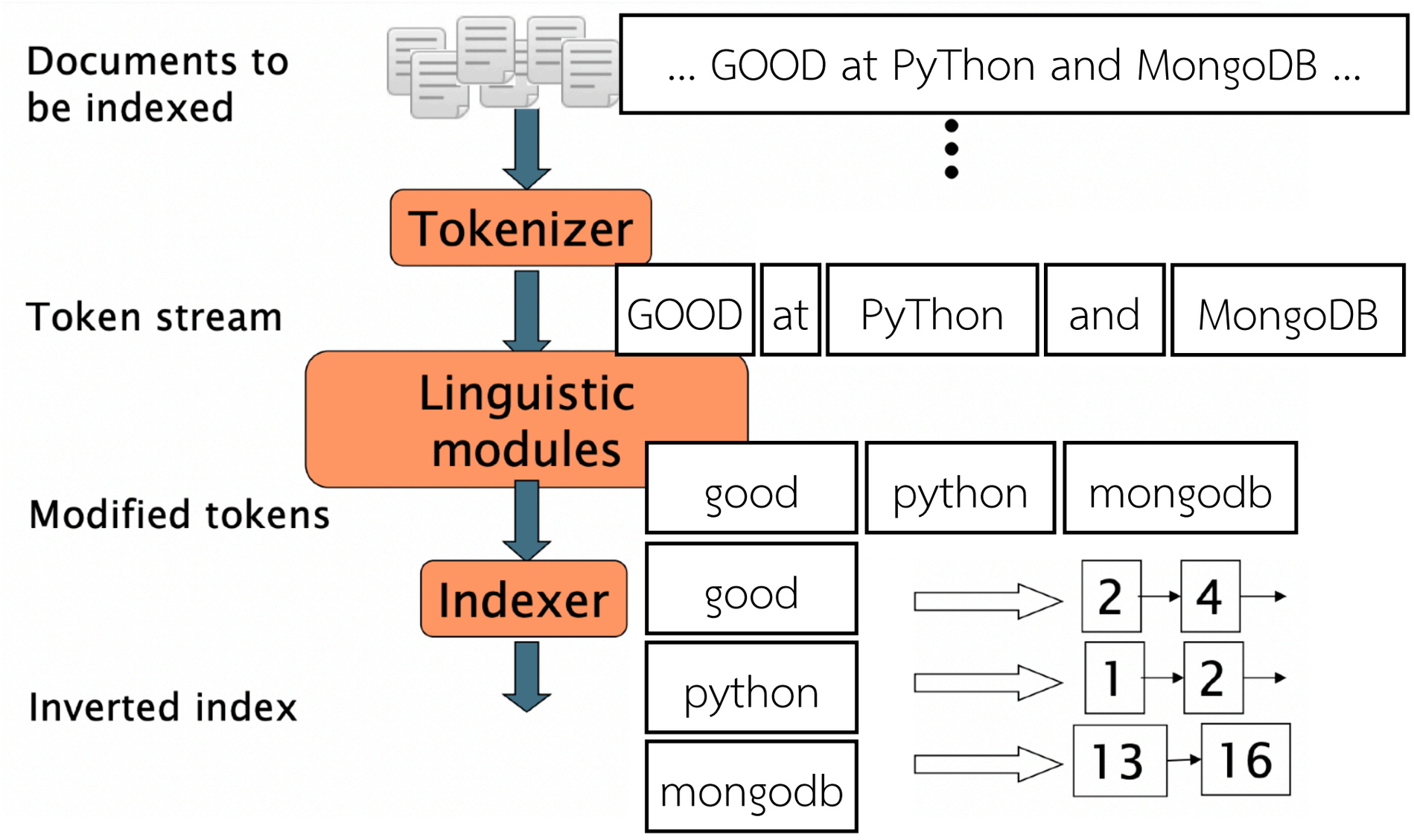
Inverted index

- For each term **t**, we store a list of all documents that contain **t**.
  - Identify each doc by a **docID**, a document serial number.

- E.g.,

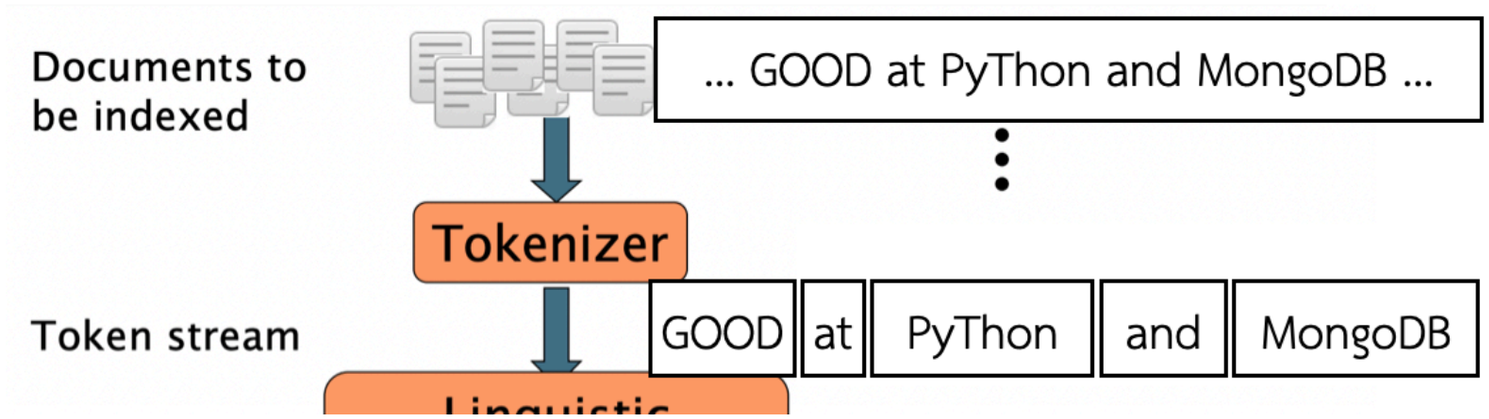


Inverted index construction



Each component

- Tokenization
  - Cut character sequence into word tokens



- Normalization
  - Map text and query term to same form,
    e.g., **S.Q.L.** and **sql** must refer to the same thing.

Each component

- Stemming
  - We may wish different forms of a root to match,
    e.g., **developed** and **developing** must refer to the same thing.

- Stop words
  - We may omit very common words (or not)
    e.g., **and, or, not, a, an ,the**

Each component

- Indexer steps: Token sequence
  - Sequence of (Modified token, Document ID) pairs.

Doc 1

The chosen software developer will be part of a larger engineering team developing software for medical devices

Doc 2

we are seeking a seasoned software developer with strong analytical and technical skills to join our public sector technology consulting team

Term	DocID
chosen	1
softwar	1
part	1
larger	1
engin	1
team	1
develop	1
medic	1
devic	1
seek	2
season	2
softwar	2
develop	2
strong	2
analyt	2
technic	2
skill	2
join	2
public	2
sector	2
technolog	2
consult	2
team	2

Each component

```
01  str1 = 'the chosen software developer will be part of a larger engineering team developing
software for medical devices.'
02  str2 = 'we are seeking a seasoned software developer with strong analytical and technical skills
to join our public sector technology consulting team.'
03
04  import nltk
05  nltk.download('stopwords')
06  nltk.download('punkt')
07  from nltk.corpus import stopwords
08  from nltk.tokenize import word_tokenize
09  from nltk.stem import PorterStemmer
10
11  tokened_str1 = word_tokenize(str1)
12  tokened_str2 = word_tokenize(str2)
13
14  tokened_str1 = [w for w in tokened_str1 if len(w) > 2]
15  tokened_str2 = [w for w in tokened_str2 if len(w) > 2]
16
17  no_sw_str1 = [word for word in tokened_str1 if not word in stopwords.words()]
18  no_sw_str2 = [word for word in tokened_str2 if not word in stopwords.words()]
19
20  ps = PorterStemmer()
21  stemmed_str1 = np.unique([ps.stem(w) for w in no_sw_str1])
22  stemmed_str2 = np.unique([ps.stem(w) for w in no_sw_str2])
23
24  full_list = np.sort(np.concatenate([stemmed_str1, stemmed_str2]))
```

Several components of the nltk library

Tokenize

Remove low frequency

Remove Stop words

Stemming

Each component

- Indexer steps: Sort
  - Sort by terms
    - At least conceptuall
      - And then docID

Core indexing step

Term	DocID
chosen	1
softwar	1
part	1
larger	1
engin	1
team	1
develop	1
medic	1
devic	1
seek	2
season	2
softwar	2
develop	2
strong	2
analyt	2
technic	2
skill	2
join	2
public	2
sector	2
technolog	2
consult	2
team	2

Term	DocID
analyt	2
chosen	1
consult	2
develop	2
develop	1
devic	1
engin	1
join	2
larger	1
medic	1
part	1
public	2
season	2
sector	2
seek	2
skill	2
softwar	2
softwar	1
strong	2
team	2
team	1
technic	2
technolog	2

Each component

- Indexer steps: Dictionary & Postings
  - Multiple term entries in a single document are merged.
  - Split into Dictionary and Postings
  - Doc. frequency information is added.

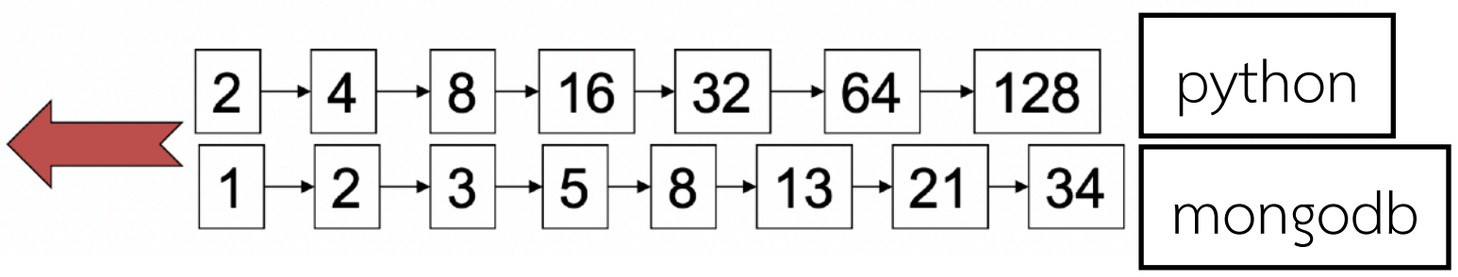
Why frequency?
Will discuss later.

Term	DocID
analyt	2
chosen	1
consult	2
develop	2
develop	1
devic	1
engin	1
join	2
larger	1
medic	1
part	1
public	2
season	2
sector	2
seek	2
skill	2
softwar	2
softwar	1
strong	2
team	2
team	1
technic	2
technolog	2

Term	doc count	posting lists
analyt	1	1
chosen	1	1
consult	1	1
develop	2	1 → 2
devic	1	1
engin	1	1
join	2	1 → 2
larger	1	1
medic	1	1
part	1	1
public	1	2
season	1	2
sector	1	2
seek	1	2
skill	1	2
softwar	2	1 → 2
strong	1	2
team	2	1 → 2
technic	1	2
technolog	1	2

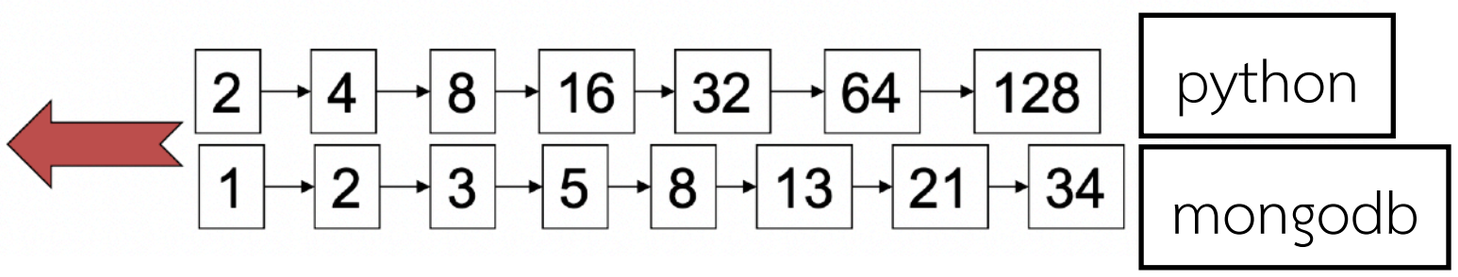
Querying

- AND
  - E.g., **Python** and **MongoDB**
    - Locate **python** in the **Dictionary** and get its **Posting list**;
    - Locate **mongodb** in the **Dictionary** and get its **Posting list**;
    - Merge the two **Posting lists, i.e., set intersection**.



Querying

- OR
  - E.g., **Python** and **MongoDB**
    - Locate **python** in the **Dictionary** and get its **Posting list**;
    - Locate **mongodb** in the **Dictionary** and get its **Posting list**;
    - Merge the two **Posting lists, i.e., set union**.



Query optimization example

- Process in order of increasing frequencies:
  - Start with smallest set, then keep cutting further.
  - E.g., (python OR ruby) AND (mysql or sqlite)

java: 3268 of 7583	oracle: 1392 of 7583
python: 1379 of 7583	mysql: **667** of 7583
c: 3434 of 7583	microsoft sql server: 868 of 7583
kotlin: 45 of 7583	postgresql: 261 of 7583
swift: 124 of 7583	mongodb: 296 of 7583
rust: 10 of 7583	db2: 130 of 7583
ruby: 346 of 7583	redis: 106 of 7583
scala: 108 of 7583	elasticsearch: 161 of 7583
julia: 2 of 7583	sqlite: **28** of 7583
lua: 20 of 7583	microsoft access: 256 of 7583

More general optimization

- Get document frequency's for all terms.
- Estimate the size of each OR by the sum of its document frequency's (conservative).
- Process in increasing order of OR sizes.

Assignment

- Show that with the approach demonstrated in the class you are able to answer the following questions:
  1. What DB should I learn after java?
  2. Which DB is in demand alongside oracle?
  3. What programing language is in demand alongside python?

- Create one question can be answered with the data we have played with and provide the answer to the question

Time for questions