RAWDATA
Section 3

Information Retrieval

Henrik Bulskov & Troels Andreasen

Information Retrieval

- **Information Retrieval (IR)** is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).
  - These days we frequently think first of web search, but there are many other cases:
    - E-mail search
    - Searching your laptop
    - Corporate knowledge bases
    - Legal information retrieval

Information Retrieval Systems

- **Information retrieval systems** use a simpler data model than database systems
  - Information organized as a collection of documents
  - Documents are unstructured: no schema

- Information retrieval main approach
  - input, a query
    - a list of one or more keywords / keyword phrases
  - output, an answer:
    - relevant documents,
  - e.g., find documents containing the two words “data mining”

Information Retrieval Content

- This lecture
  - Information Retrieval basics
  - Information Retrieval in MySQL
  - Information Retrieval our approach
    - In assignment 5
    - In portfolio

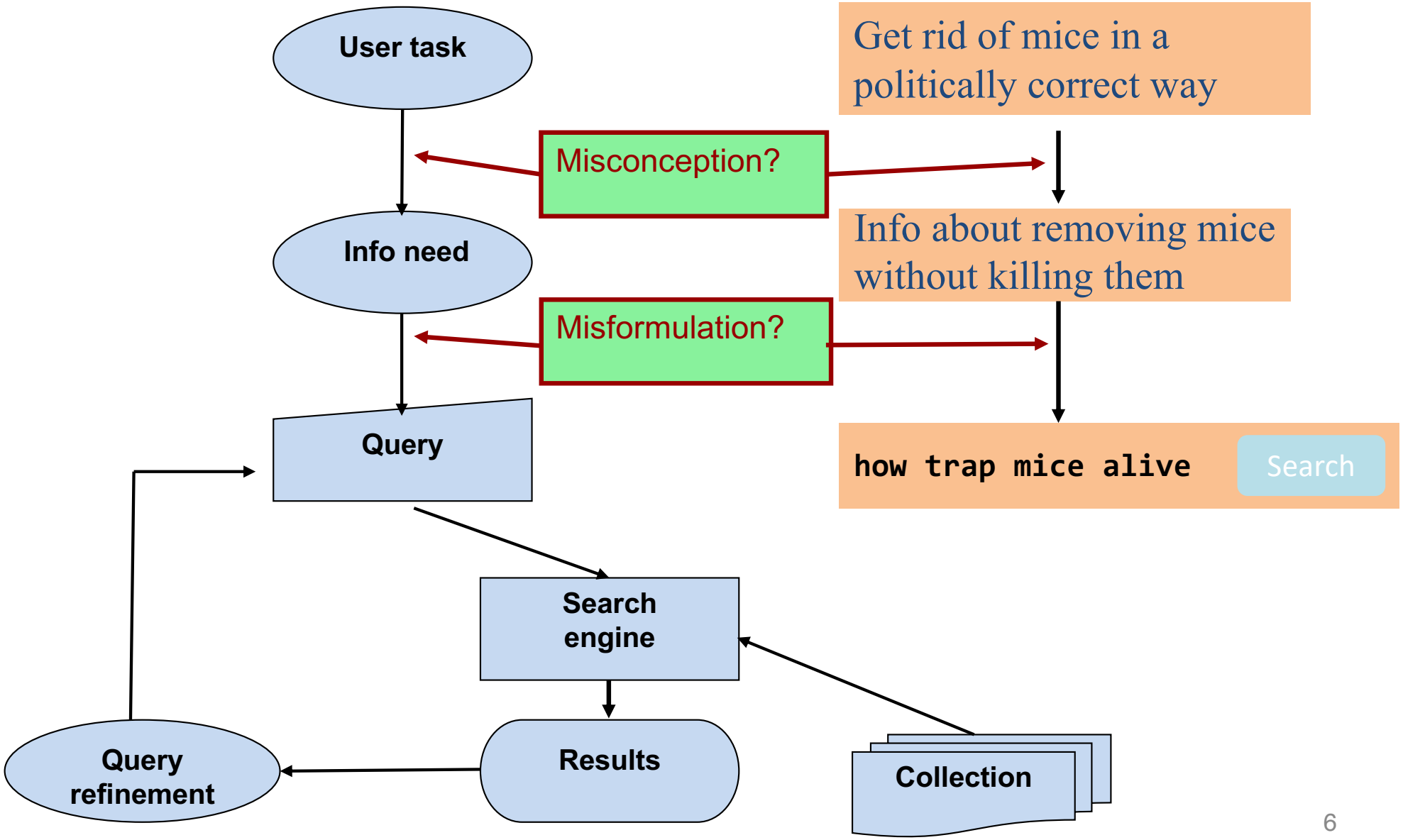
- Next lecture (Thursday)
  - Information Retrieval basics continued
  - Relevance ranking
  - Some advanced approaches

Basic assumptions of Information Retrieval

- Collection: A set of documents
  - Assume it is a static collection for now

- Goal: Retrieve documents with information that is relevant to the user's information need and helps the user complete a task

The classic search model



How good are the retrieved docs?

Important evaluation measures

- ***Precision*** :
  - Fraction of retrieved docs that are relevant (to the user's information need)
- ***Recall*** :
  - Fraction of relevant docs in collection that are retrieved
  - More precise definitions and measurements to follow

Unstructured data in 1620

- Which plays of Shakespeare contain the words ***Brutus*** *AND* ***Caesar*** but *NOT* ***Calpurnia***?
- One could `grep` all of Shakespeare's plays for ***Brutus*** and ***Caesar,*** then strip out lines containing ***Calpurnia***?
- Why is that not the answer?
  - Slow (for large corpora)
  - *NOT* ***Calpurnia*** is non-trivial
  - Other operations (e.g., find the word ***Romans*** near ***countrymen***) not feasible
  - Ranked retrieval (best documents to return) not feasible
    - Covered later

Term-document incidence matrices

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

Brutus *AND* ***Caesar*** *BUT NOT* ***Calpurnia***

1 if play contains word, 0 otherwise

Incidence vectors

- So we have a 0/1 vector for each term.
- To answer query: take bitwise *AND* of the vectors for ***Brutus, Caesar*** and ***Calpurnia*** (complemented)
  - 110100 AND 110111 AND NOT 010000 =
  - 110100 AND 110111 AND 101111 = 100100

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

Answers to query

- Antony and Cleopatra, Act III, Scene ii

Agrippa [Aside to DOMITIUS ENOBARBUS]: Why, Enobarbus,
When Antony found Julius ***Caesar*** dead,
He cried almost to roaring; and he wept
When at Philippi he found ***Brutus*** slain.

- Hamlet, Act III, Scene ii

Lord Polonius: I did enact Julius ***Caesar*** I was killed i' the Capitol; ***Brutus*** killed me.

Bigger collections

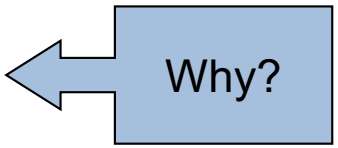
- Consider N = 1 million documents, each with about 1000 words.
- Say there are M = 500K *distinct* terms among these.

Can't build the matrix

- Consider N = 1 million documents, each with about 1000 words.
- Say there are M = 500K *distinct* terms among these.

- 500K x 1M matrix
  has half-a-trillion (500.000.000.000) 0's and 1's.

- But it has no more than
  one billion (1.000.000.000) 1's.
  - matrix is extremely sparse.



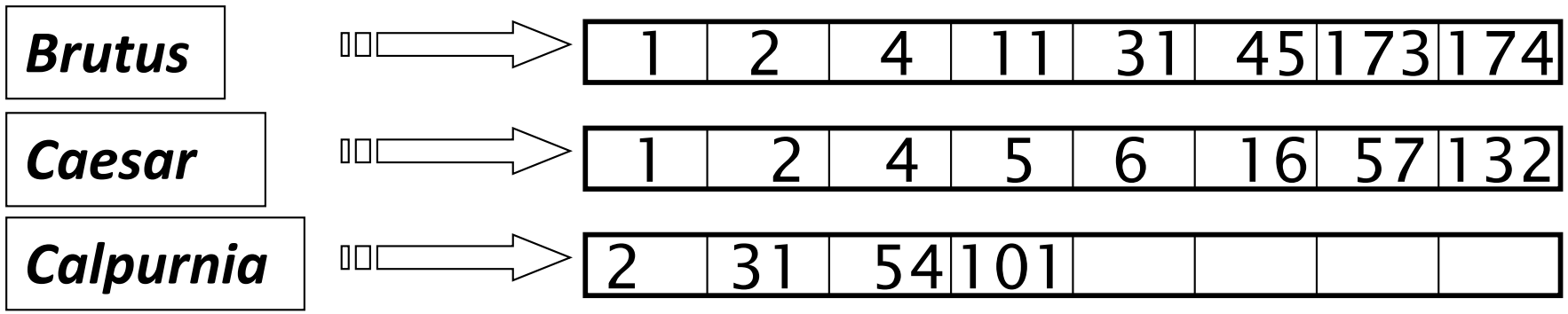
- What's a better representation?
  - We only record the 1 positions.

What to do?

- Build an inverted index instead …

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)

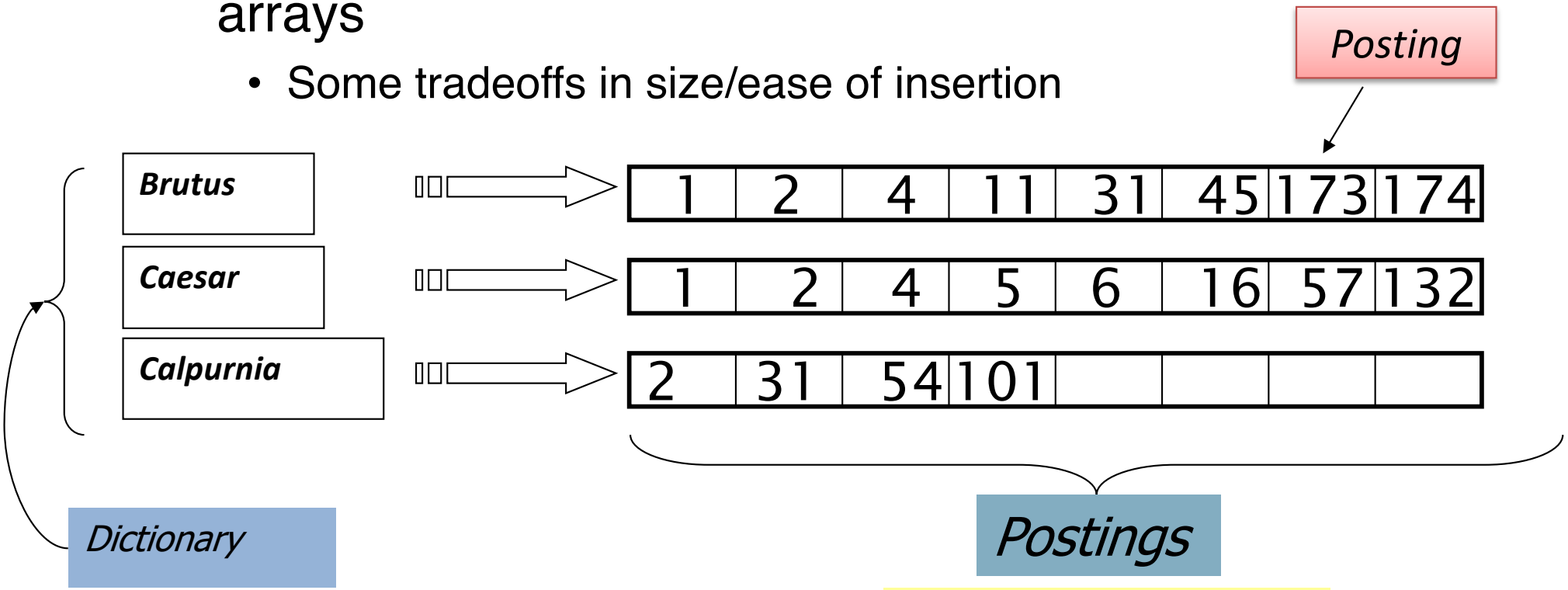


What happens if the word ***Caesar*** is added to document 14?

- Can we used fixed-size arrays for these "postings lists"?

Inverted index

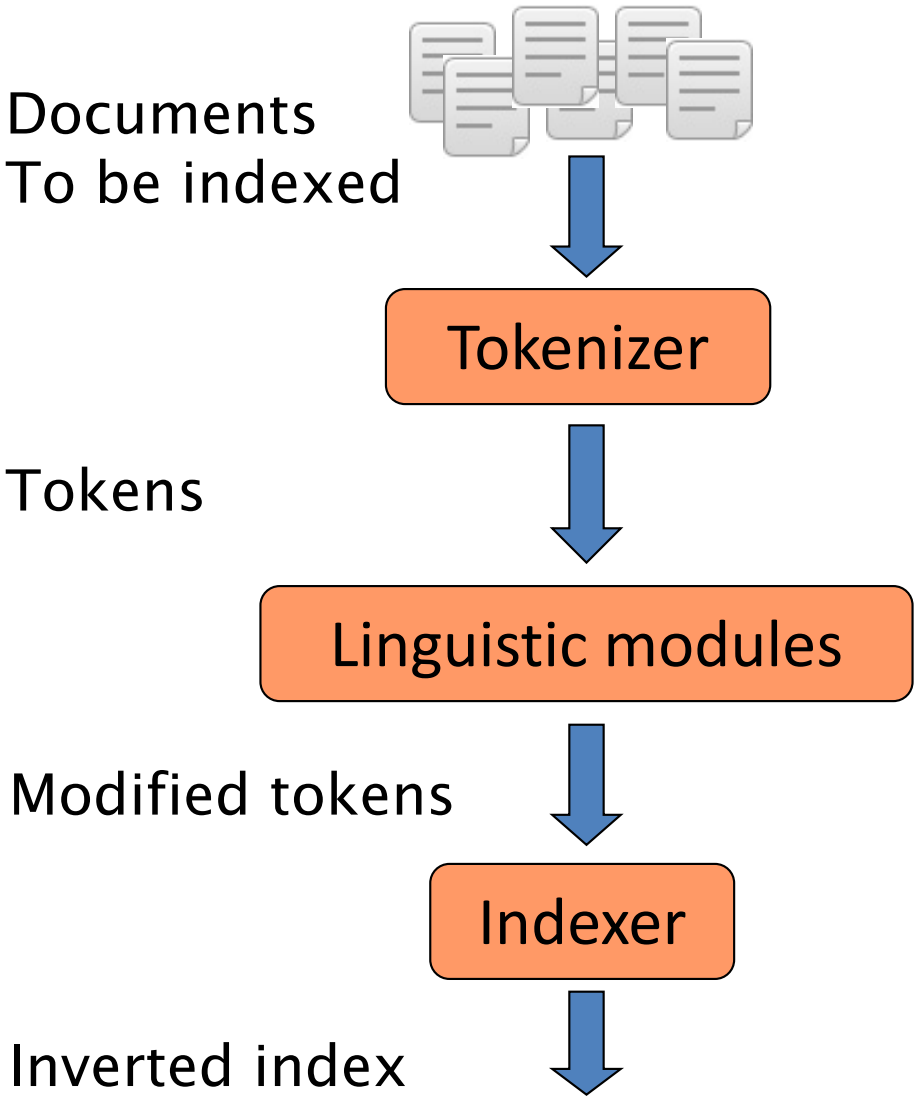
- We need variable-size postings lists
  - On disk, a continuous run of postings is normal and best
  - In memory, can use linked lists or variable length arrays
    - Some tradeoffs in size/ease of insertion

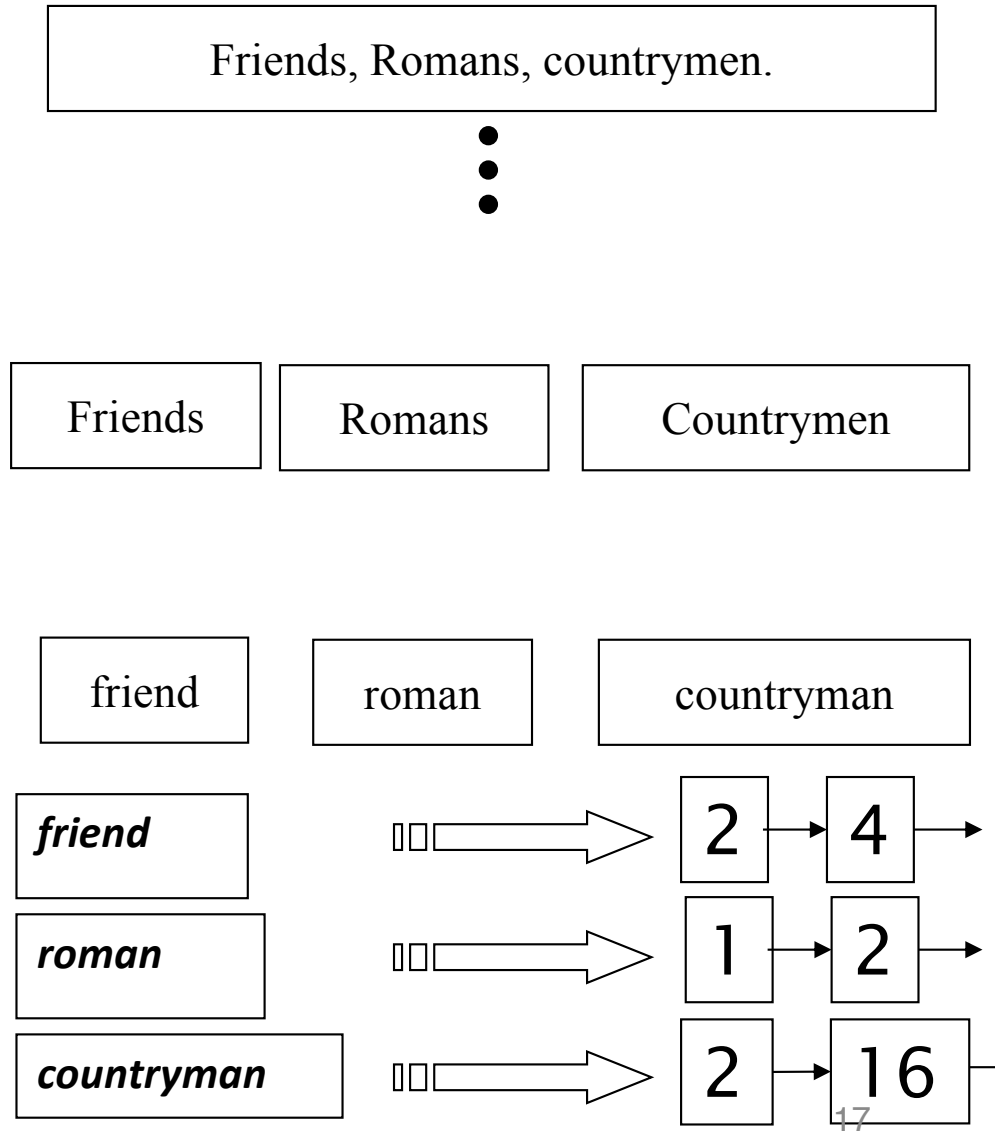


Sorted by docID

Why?

Inverted index construction





Indexer steps: Token sequence

- Sequence of (Modified token, Document ID) pairs.

Doc 1

I did enact Julius Caesar I was killed i' the Capitol; Brutus killed me.

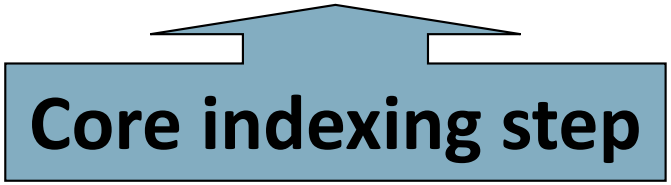
Doc 2

So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious

Term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Indexer steps: Sort

- Sort by terms
  - And then docID



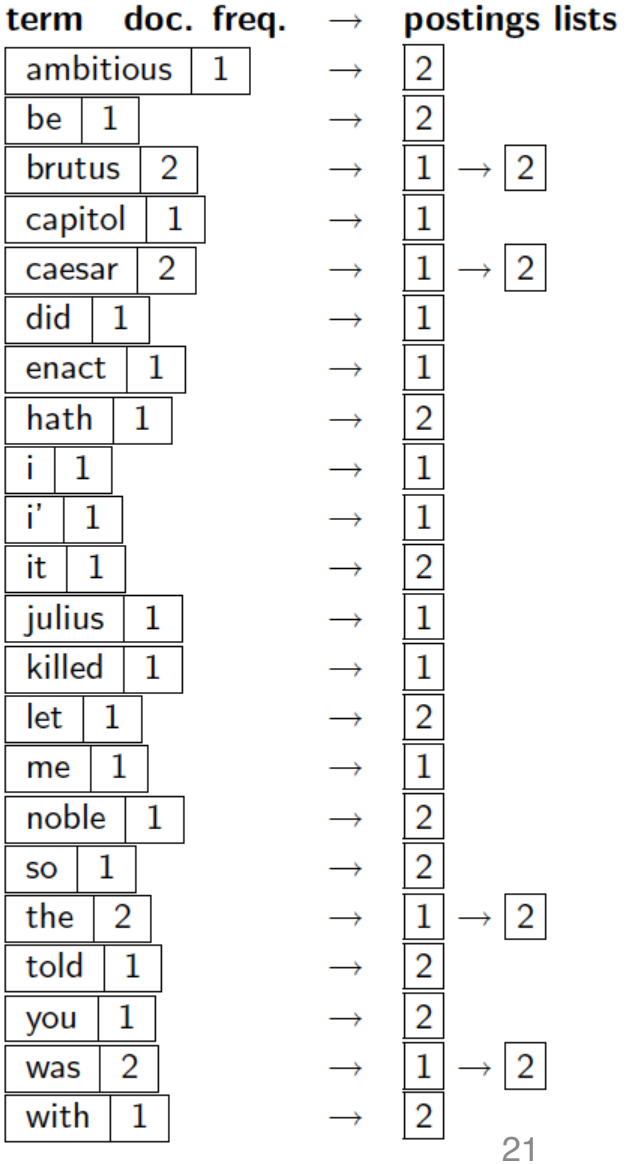
Term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

Indexer steps: Dictionary & Postings

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

Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2



Why frequency?
Will discuss later.

The index we just built

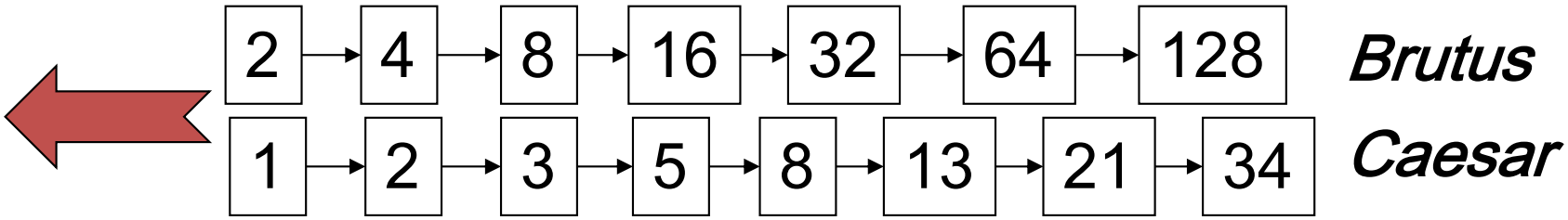
- How do we process a query?
  - Later - what kinds of queries can we process?

Query processing: AND

- Consider processing the query:

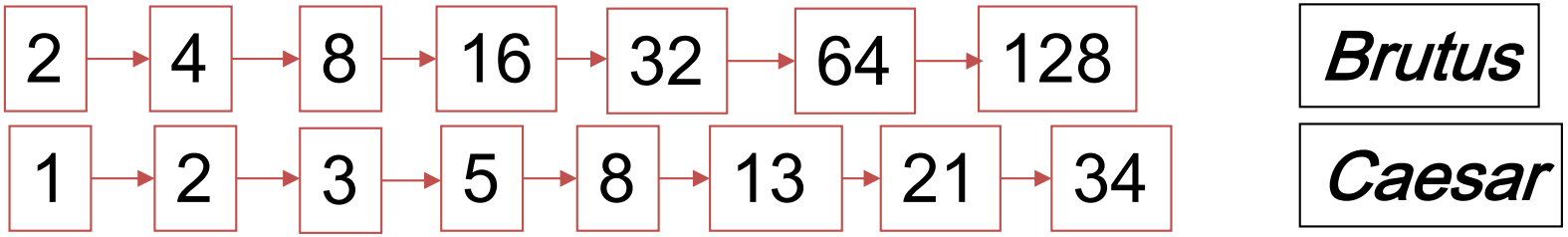
  Brutus *AND* ***Caesar***

  – Locate ***Brutus*** in the Dictionary;
    - Retrieve its postings.
  – Locate ***Caesar*** in the Dictionary;
    - Retrieve its postings.
  – "Merge" the two postings (intersect the document sets):



The merge

- Walk through the two postings simultaneously, in time linear in the total number of postings entries



If the list lengths are *x* and *y*, the merge takes O(*x*+*y*) operations.

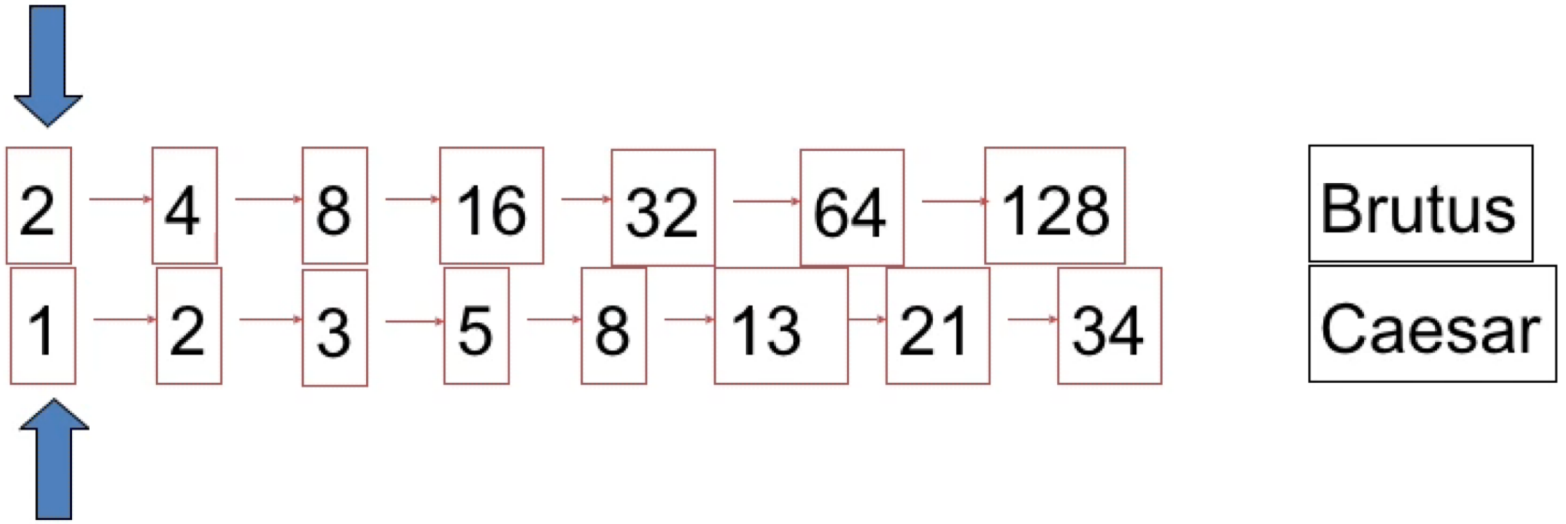
Crucial: we assume that postings sorted by docID.

Intersecting two postings lists (a "merge" algorithm)

```
INTERSECT(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then p1 ← next(p1)
 9                else p2 ← next(p2)
10  return answer
```

The merge

- Walk through the two postings simultaneously, in time linear in the total number of postings entries



answer = ⟨ ⟩

Boolean queries: Exact match

- The Boolean retrieval model is being able to ask a query that is a Boolean expression:
  - Boolean Queries are queries using *AND, OR* and *NOT* to join query terms
    - Views each document as a set of words
    - Is precise: document matches condition or not.
  - Perhaps the simplest model to build an IR system on
- Primary commercial retrieval tool for 3 decades.
- Many search systems you still use are Boolean:
  - Email, library catalog, Windows Search, MacOS Spotlight

Phrase queries

- We want to be able to answer queries such as ***"stanford university"*** – as a phrase

- Thus the sentence *"I went to university at Stanford"* is not a match.
  - The concept of phrase queries has proven easily understood by users; one of the few "advanced search" ideas that works
  - Many more queries are *implicit phrase queries*

- To support phrase queries, it no longer suffices to store only
  <*term* : *docs*> entries

A first attempt: Biword indexes

- Index every consecutive pair of terms in the text as a phrase
- For example the text “Friends, Romans, Countrymen” would generate the biwords
  - ***friends romans***
  - ***romans countrymen***
- Each of these biwords is now a dictionary term
- So, two-word phrase query-processing is now immediate.

Longer phrase queries

- Longer phrases can be processed by breaking them down
- ***stanford university palo alto*** can be broken into the Boolean query on biwords:

stanford university *AND* ***university palo*** *AND* ***palo alto***

- Without the docs, however, we cannot verify that the docs matching the above Boolean query do contain the phrase.

Issues for biword indexes

- False positives, as noted before
- Index may blowup due to bigger dictionary
  - Infeasible for more than biwords, big even for them
- Biword indexes
  - are not the standard solution
  - but can be part of a compound strategy

Solution 2: Positional indexes

- In the postings, store, for each ***term*** the position(s) in which tokens of it appear:

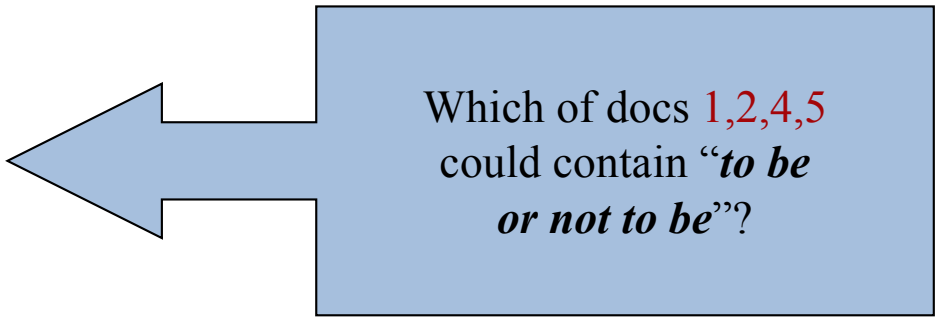
  <***term***, number of docs containing ***term***;
  doc1: position1, position2 … ;
  doc2: position1, position2 … ;
  etc.>

- Positional index example:

  <***be***: 993427;
  1: 7, 18, 33, 72, 86, 231;
  2: 3, 149;
  4: 17, 191, 291, 430, 434;
  5: 363, 367, …>

Positional index example

<*be*: 993427;
1: 7, 18, 33, 72, 86, 231;
2: 3, 149;
4: 17, 191, 291, 430, 434;
5: 363, 367, …>



- For phrase queries, we use a merge algorithm recursively at the document level
- But we now need to deal with more than just equality

Processing a phrase query

- Extract inverted index entries for each distinct term: ***to, be, or, not.***
- Merge their *doc:position* lists to enumerate all positions with "***to be or not to be***".
  - ***to****:*
    - *2*:1,17,74,222,551; *4*:8,16,190,429,433; *7*:13,23,191; ...
  - ***be****:*
    - *1*:17,25; *4*:17,191,291,430,434; *5*:14,19,101; ...
  - ***or***

    …
  - ***…***

- Same general method for proximity searches

Proximity queries

- Query: employment /3 place
  - here "/*k*" means "within *k* words of"

- Information need: Requirements for disabled people to be able to access a workplace.
  Query: disab! /p access! /s work-site work-place (employment /3 place)
  - "!" means is a wild-card (any trailing character sequence)
  - "/*p*" means "within same paragraph"
  - "/*s*" means "within same sentence"
  - " " (space) means disjunction (tightest binding)

- as implemented in e.g. **Westlaw**,
  - an (US) online legal service for lawyers and legal professionals
  - include documents on case law, state and federal statutes, newspaper and magazine articles, law journals, etc.

- Clearly, positional indexes can be used for such queries; biword indexes cannot.

IR vs. databases: Structured vs unstructured data

- Structured data tends to refer to information in "tables"

Employee	Manager	Salary
Smith	Jones	50000
Chang	Smith	60000
Ivy	Smith	50000

Typically allows numerical range and exact match (for text) queries, e.g.,

Salary < 60000 AND Manager = Smith.

Unstructured data

- Typically refers to free text

- Allows
  - Keyword queries including operators
  - More sophisticated “concept” queries e.g.,
    - find all web pages dealing with *drug abuse*

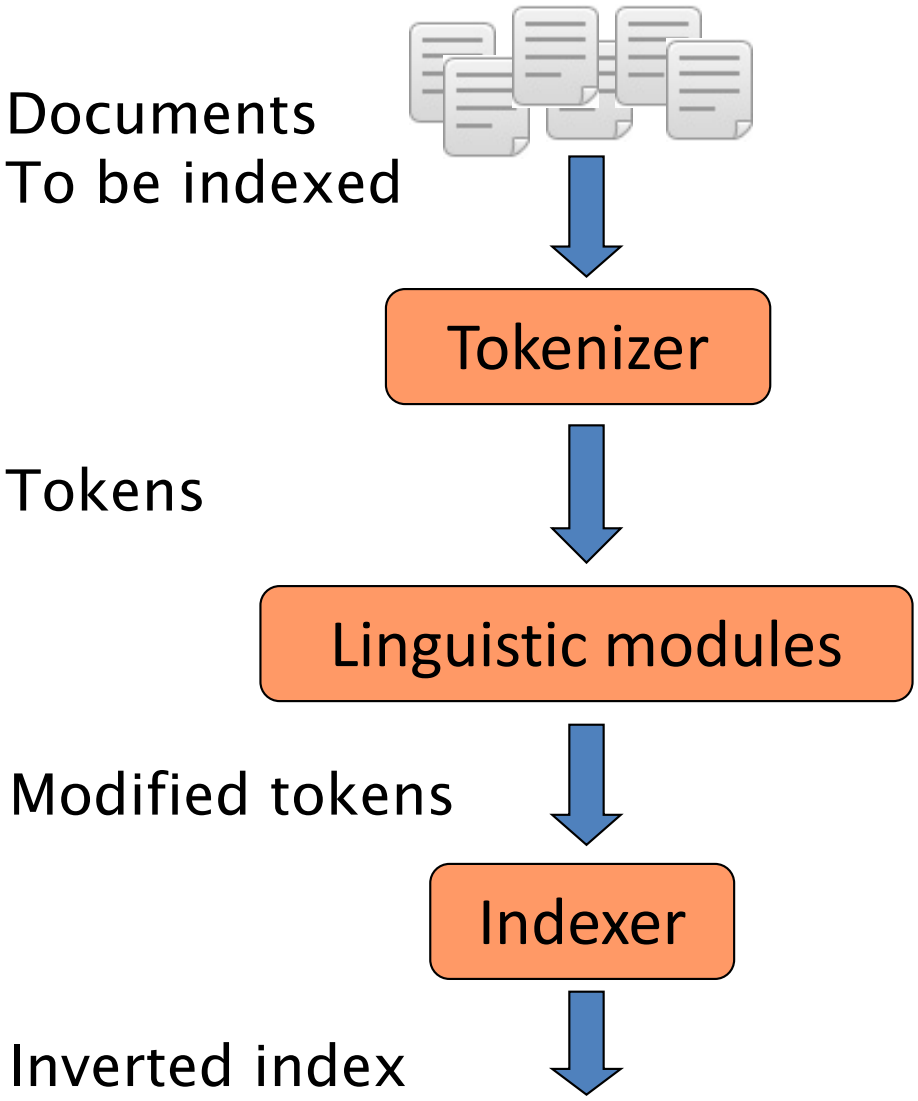
- Classic model for searching text documents

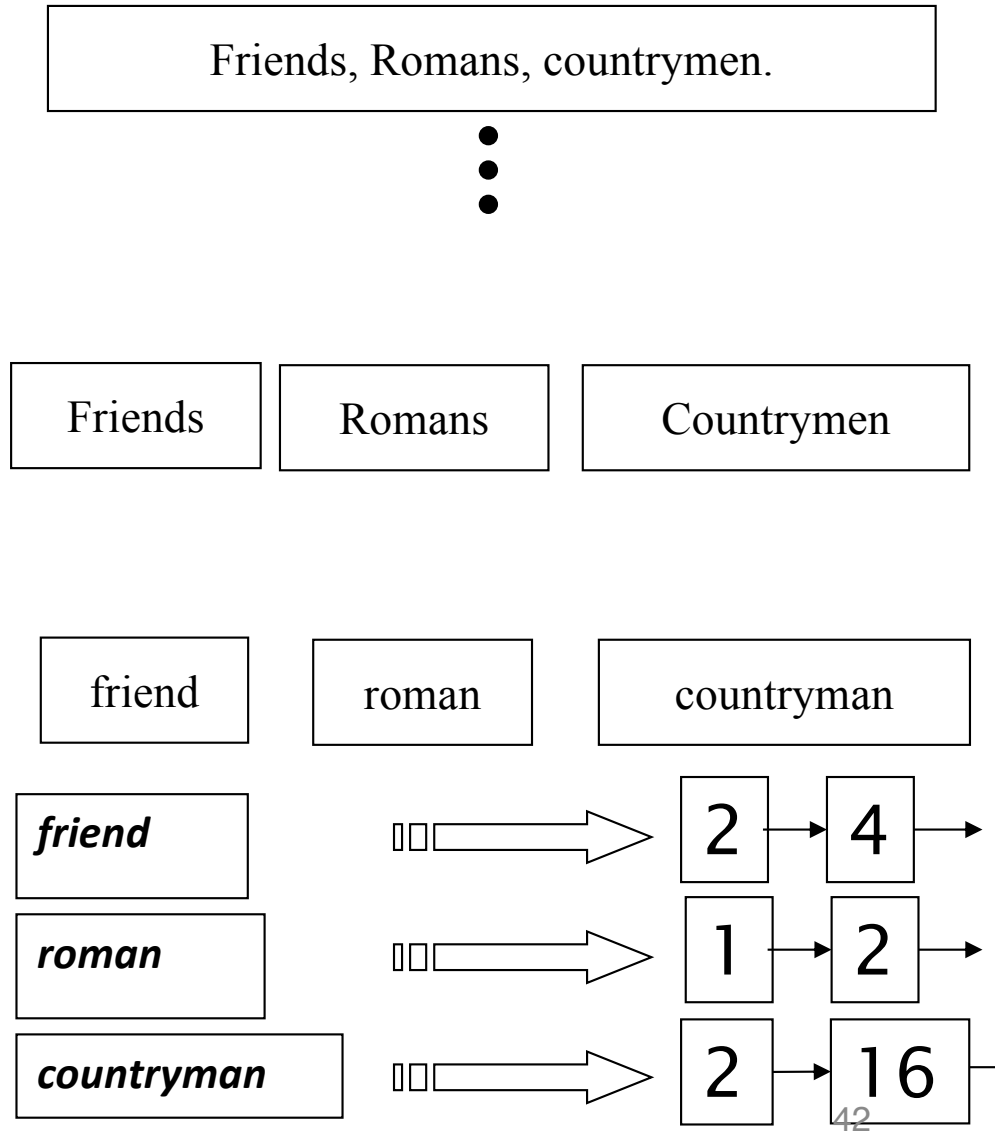
Semi-structured data

- In fact almost no data is “unstructured”
- E.g., this slide has distinctly identified zones such as the *Title* and *Bullets*
    - … to say nothing of linguistic structure

- Facilitates “semi-structured” search such as
  – *Title* contains data AND *Bullets* contain search

Inverted index construction (again)





Inverted index construction (again)

So, what are:

Documents?

Tokens?

Modified tokens?

Parsing a document

- What format is it in?
  - pdf/word/excel/html?
- What language is it in?
- What character set is in use?
  - (CP1252, UTF-8, …)

Often verified heuristically …

Complications: Format/language

- Documents being indexed can include multiple languages
  - French email with a German pdf attachment.
  - French email quote clauses from an English-language contract

- There are commercial and open source libraries that can handle a lot of this stuff

Complications: What is a document?

We return from our query "documents" but there are often interesting questions of grain size:

What is a unit document?

- A file?
- An email? (Perhaps one of many in a single mbox file)
  - What about an email with 5 attachments?
- A group of files (e.g., PPT or LaTeX split over HTML pages)

- A post from a question-answer blog

- A group of posts consisting of a question and the related answers

Tokenization

- Input: "***Friends, Romans and Countrymen***"
- Output: Tokens
  - ***Friends***
  - ***Romans***
  - ***Countrymen***
- A token is an instance of a sequence of characters
- Each such token is now a candidate for an index entry, after further processing
  - Described below
- But what are valid tokens to emit?

Tokenization

- Issues in tokenization:
  - ***Finland's capital*** →

    Finland AND ***s***? ***Finlands***? ***Finland's***?
  - ***Hewlett-Packard*** → ***Hewlett*** and ***Packard*** as two tokens?
    - ***state-of-the-art***: break up hyphenated sequence.
    - ***co-education***
    - ***lowercase***, ***lower-case***, ***lower case*** ?
  - ***San Francisco***: one token or two?
    - How do you decide it is one token?

Numbers

- ***3/20/91*** ***Mar. 12, 1991*** ***20/3/91***
- ***55 B.C.***
- ***B-52***
- ***(800) 234-2333***

- Numbers
  - Often have embedded spaces
  - Older IR systems may not index them at all

Tokenization: language issues

- French
  - ***L'ensemble*** → one token or two?
    - ***L*** ? ***L'*** ? ***Le*** ?
    - We want ***l'ensemble*** to match with ***un ensemble***

- German
  - In German noun compounds are not segmented
    - ***Lebensversicherungsgesellschaftsangestellter***
    - ('life insurance company employee')
  - German retrieval systems benefit greatly from a **compound splitter** module
    - Can give a 15% performance boost for German
  - Same issue for Danish

from tokens to terms …

Terms?

- Terms
  - The things indexed in an IR system

- Issues
  - Stopwords
  - Nomalization of terms
  - Case
  - “Similar words”: misspellings, synonyms,
  - Lemmatization
  - Stemming

Stopwords

- With a stopword list, you exclude from the dictionary entirely the commonest words. Intuition:
  - They have little semantic content: *the, a, and, to, be*
  - There are a lot of them: ~30% of postings for top 30 words
- But the trend is away from doing this:
  - Good compression techniques (IIR chapter 5) means the space for including stop words in a system is very small
  - Good query optimization techniques (IIR chapter 7) means you pay little at query time for including stopwords.
  - You need them for:
    - Phrase queries: “King of Denmark”
    - Various song titles, etc.: “Let it be”, “To be or not to be”
    - “Relational” queries: “flights to London”

Normalization to terms

- Normalization ≈ defining equivalence classes over terms

- We may need to "normalize" words in indexed text as well as query words into the same form
  - We want to match ***U.S.A.*** and ***USA***
- Result is terms: a term is a (normalized) word type, which is an entry in our IR system dictionary

- We most commonly implicitly define "equivalence classes" of terms by, e.g.,
  - deleting periods to form a term
    - ***U.S.A., USA*** → ***USA***
  - deleting hyphens to form a term
    - ***anti-discriminatory, antidiscriminatory*** → ***antidiscriminatory***

Normalization to terms

- Accents: e.g., French ***résumé*** vs. ***resume***.
- Umlauts: e.g., German: ***Tuebingen*** vs. ***Tübingen***
  - Should be equivalent
- Most important criterion:
  - How are your users likely to write their queries for these words?

- Even in languages that standardly have accents, users often may not type them
  - Often best to normalize to a de-accented term
    - ***Tuebingen, Tübingen, Tubingen*** → ***Tubingen***

- Crucial: Need to "normalize" indexed text as well as query terms identically

Case folding

- Reduce all letters to lower case
  - exception: upper case in mid-sentence?
    - e.g., General Motors
    - Fed vs. fed
    - SAIL vs. sail
  - Often best to lower case everything, since users will use lowercase regardless of 'correct' capitalization…

Normalization to terms

- An alternative to equivalence classing is to do **query expansion**
- An example of where this may be useful
  - Enter: ***window*** Search: ***window, windows***
  - Enter: ***windows*** Search: ***Windows, windows, window***
  - Enter: ***Windows*** Search: ***Windows***
- Potentially more powerful, but less efficient

Thesauri and soundex

- Do we handle **synonyms and homonyms**?
  - E.g., by hand-constructed equivalence classes
    - ***car*** = ***automobile*** ***color*** = ***colour***
  - We can rewrite to form equivalence-class terms
    - When the document contains ***automobile***, index it under ***car-automobile*** (and vice-versa)
  - Or we can expand a query
    - When the query contains ***automobile***, look under ***car*** as well

- What about spelling mistakes?
  - One approach is Soundex, which forms equivalence classes of words based on phonetic heuristics
    (Soundex is a phonetic algorithm for indexing names by sound, as pronounced in English)

Lemmatization

- Reduce inflectional/variant forms to base form
- E.g.,
  - *am, are, is* → *be*
  - *car, cars, car's, cars'* → *car*
- *the boy's cars are different colors* → *the boy car be different color*
- Lemmatization implies doing "proper" reduction to dictionary headword form

Stemming

- Reduce terms to their "roots" before indexing
- "Stemming" suggests crude affix chopping
  - language dependent
  - e.g., ***automate(s), automatic, automation*** all reduced to ***automat***.

for example compressed and compression are both accepted as equivalent to compress.

for exampl compress and compress ar both accept as equival to compress

Porter's algorithm

- Most common algorithm for stemming English
  - Results suggest it's at least as good as other stemming options

- Conventions + 5 phases of reductions
  - phases applied sequentially
  - each phase consists of a set of commands
  - sample convention: *Of the rules in a compound command, select the one that applies to the longest suffix.*

- Other stemmers exist:
  - Lovins stemmer
  - Paice/Husk stemmer
  - Snowball

Typical rules in Porter

- *sses* → *ss*
- *ies* → *i*
- *ational* → *ate*
- *tional* → *tion*

- Weight of word sensitive rules
- *(m>1) EMENT* →
  - *replacement* → *replac*
  - *cement* → *cement*

Does stemming help?

- English: very mixed results. Helps recall for some queries but harms precision on others
  - E.g., operative (dentistry) ⇒ oper

- Definitely useful for Spanish, German, Finnish, …
  - 30% performance gains for Finnish!

DB versus IR Systems

- Database (DB) systems
  - deal with structured data, with schemas that define the data organization
  - refined transactional updates, concurrency control or recovery

- Information Retrieval (IR) systems
  - don't normally deal with transactional updates, concurrency control or recovery
  - deal, however, with some querying issues not generally addressed by database systems:
    - Keyword search: Approximate searching by keywords
    - Ranking of retrieved answers by estimated degree of relevance
  - Ranking of documents on the basis of estimated relevance to a query is critical
    (ranking to be covered in more detail Thursday)

Information retrieval supported in MySQL

- On string columns (CHAR, VARCHAR, or TEXT) you can add a FULLTEXT index in MySQL

```
create table postes as select distinct id,title,body
        from stackoverflow_sample_universal.posts;
alter table postes ADD FULLTEXT (title);
alter table postes ADD FULLTEXT (body);
```

  - Can be specified in create table, alter table or create index statements (as for other indexes)
  - It is much faster to load your data into a table that has no FULLTEXT index and then create the index after that (as for other indexes)

Information retrieval supported in MySQL

- On columns with a FULLTEXT index you can do full-text search

```
SELECT title FROM postes
WHERE MATCH (title) AGAINST ("script in python");
```

- Full-text searching
  - is performed using MATCH() ... AGAINST syntax.
  - MATCH() takes a comma-separated list that names the columns to be searched.
  - AGAINST takes a string to search for, and an optional **modifier** that indicates what type of search to perform

title
Running a Python script (using an external module) through Android
Executing a python script from inittab not as root
is there a way to script in Python to change user passwords in Linux? if so, how?
Python - Should one start a new project directly in Python 3.x?
Long-running CGI script hangs
how do i refresh a php script every ten seconds?
Confused about a function returning a function in java script
Why does jQuery load twice in my GreaseMonkey Script
Exit code of a sourced shell script

Information retrieval supported in MySQL

- In full-text queries all rows have a relevance degree
- MATCH() ... AGAINST can be used in the SELECT clause to retrieve this degree:

```
SELECT title, MATCH (title) AGAINST ("script in python")
FROM postes
WHERE MATCH (title) AGAINST ("script in python");
```

and it may be used for ordering also:

```
SELECT title, MATCH (title) AGAINST ("script in python") deg
FROM postes
WHERE MATCH (title) AGAINST ("script in python")
ORDER BY deg DESC;
```

title	deg
Running a Python script (using an external module) through Android	12.774249076843...
Executing a python script from inittab not as root	12.774249076843...
is there a way to script in Python to change user passwords in Linux? if so, how?	12.774249076843...
Python - Should one start a new project directly in Python 3.x?	10.446834564208...
Long-running CGI script hangs	7.5508317947387...
how do i refresh a php script every ten seconds?	7.5508317947387...
Confused about a function returning a function in java script	7.5508317947387...

Information retrieval supported in MySQL

- There are three types of full-text searches:
- Specifyied by Modifiers
  - IN NATURAL LANGUAGE MODE
  - IN BOOLEAN MODE
  - IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION

```
SELECT title FROM postes WHERE MATCH (title)
AGAINST ("script in python" IN NATURAL LANGUAGE MODE);
```

```
SELECT title FROM postes WHERE MATCH (title)
AGAINST ("+script in +python -root" IN BOOLEAN MODE);
```

```
SELECT title FROM postes WHERE MATCH (title) AGAINST
("script in python" IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION);
```

Information retrieval supported in MySQL

- The three types of full-text searches:
- Natural language (IN NATURAL LANGUAGE MODE)
  - A natural language search interprets the search string as a phrase in natural human language (a phrase in free text). There are no special operators, with the exception of double quote (") characters.
- Boolean (IN BOOLEAN MODE)
  - A boolean search interprets the search string using the rules of a special query language. The string contains the words to search for and operators that specify requirements.
- Query expansion (IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION)
  - A modification of a natural language search:
    - a natural language search is performed
    - words from the most relevant rows returned by the search are added to the search string
    - the search is done again
    - query returns the rows from the second search.

Information retrieval supported in MySQL

- Example boolean query

```
SELECT title, MATCH (title)
AGAINST ("+script in +python" IN BOOLEAN MODE) deg
FROM postes
WHERE MATCH (title) AGAINST ("+script in +python" IN BOOLEAN MODE)
ORDER BY deg DESC;
```

title	deg
Running a Python script (using an external module) through Android	12.774249076843...
Executing a python script from inittab not as root	12.774249076843...
is there a way to script in Python to change user passwords in Linux? if so, how?	12.774249076843...

```
SELECT title, MATCH (title)
AGAINST ("+script in +python -root" IN BOOLEAN MODE) deg
FROM postes
WHERE MATCH (title) AGAINST ("+script in +python -root" IN BOOLEAN
MODE)
ORDER BY deg DESC;
```

title	deg
Running a Python script (using an external module) through Android	12.774249076843...
is there a way to script in Python to change user passwords in Linux? if so, how?	12.774249076843...

Our RAWDATA Information Retrieval approach

- We consider Information Retrieval starting from theory and solutions presented in the literature (especially IIR and DSC)

- We do, however, consider (and practice with) implementation in tables in our DBMS (MySQL)
  - Thus some of the algorithmic details in the literature are less relevant for our purpose

- We don't go into further details with MySQL full-text retrieval
  - MySQL full-text retrieval is black-box and
  - We need an open system where we can manipulate with the postings – especially when we move on to retrieve list-of-word-based answers (rather than list-of-posts)

Portfolio – Extended IR functionality

- Introducing IR-functionality in the portfolio project
  - a second iteration redesign
  - to simplify, a resource, in the form of a table **words** can be found on Moodle
- the **words** table is an inverted index and provides
  - given a word, all occurrences of the word can be found, by looking up the word in the **words** table
  - a combined index for posts and comments (may be separated, if preferred)
    all 3 combinations of (**tablename**, **what**) are:

tablename	what
posts	title
posts	body
comments	text

  - positional indexing by the **sen** and **idx** columns, which specifies the sentence number and the word position number (in the sentence) respectively

id	tablename	what	sen	idx	word	pos	lemma
14075810	posts	body	1	1	Using	VBG	use
14075810	posts	body	1	2	a	DT	a
14075810	posts	body	1	3	regular	JJ	regular
14075810	posts	body	1	4	expression	NN	expres…
14075810	posts	body	1	5	that	W…	that
14075810	posts	body	1	6	recognizes	VBZ	recogn…
14075810	posts	body	1	7	email	NN	email

Portfolio – Extended IR functionality

- the **words** table, in addition, provides
  - a lemmatization
    - in the column **lemma** and
    - a specification of the word category (part-of-speech) in the column **pos** (part-of-speech specified by Penn Tree Bank tags)

Alphabetical list of part-of-speech tags used in the Penn Treebank Project:

Number	Tag	Description
1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential *there*
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunction
7.	JJ	Adjective
8.	JJR	Adjective, comparative
9.	JJS	Adjective, superlative
10.	LS	List item marker
11.	MD	Modal
12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular

id	tablename	what	sen	idx	word	pos	lemma
14075810	posts	body	1	1	Using	VBG	use
14075810	posts	body	1	2	a	DT	a
14075810	posts	body	1	3	regular	JJ	regular
14075810	posts	body	1	4	expression	NN	expres…
14075810	posts	body	1	5	that	W…	that
14075810	posts	body	1	6	recognizes	VBZ	recogn…
14075810	posts	body	1	7	email	NN	email
14075810	posts	body	1	8	addresses	NNS	address

Portfolio – Extended IR functionality

- Some requirements
  - Introduce a multiword **querying** procedure that takes **one or more keywords** as arguments and returns posts that match these. Develop and use your own inverted index. Can be derived from the table **words.** (thus: **don't use** the **like**-operator, **don't use MySQL Full-text** search).
  - Introduce **ranked retrieval**: at least one simple and one more advanced ranking principle to provide **ranked lists of posts**
  - Provide **ranked lists of words** as query answer. (Preparing word clouds as data visualization in the SOVA front-end)

- Plus some more challenges (a few mandatory to be revealed and many optional to select from)
  - …
  - morphology similarity search,
  - word proximity based search
  - Relevance feedback
  - Query expansion, term browsing, term network visualization
  - Concept-based querying (involving also taxonomy, ontology, …)
  - ...

Your own ideas, if some should pop up, are obviously also welcome

Preparing the data

- Now, the words table combines information about
  - inverted index
    - on posts (using title and body) as well as
    - on comments (using text)
    - with a very broad definition of “word” (also single character)
    - not excluding so-called stop-words
  - details about actual positions of words in sentences
  - linguistic knowledge
    - part-of-speech tags (pos)
    - lemmas

id	tablename	what	sen	idx	word	pos	lemma
14075810	posts	body	1	1	Using	VBG	use
14075810	posts	body	1	2	a	DT	a
14075810	posts	body	1	3	regular	JJ	regular
14075810	posts	body	1	4	expression	NN	expres…
14075810	posts	body	1	5	that	W…	that
14075810	posts	body	1	6	recognizes	VBZ	recogn…
14075810	posts	body	1	7	email	NN	email

Preparing the data

- a lot of details …
  - You are not required to use everything …
  - but encouraged to consider and discuss alternatives

id	tablename	what	sen	idx	word	pos	lemma
14075810	posts	body	1	1	Using	VBG	use
14075810	posts	body	1	2	a	DT	a
14075810	posts	body	1	3	regular	JJ	regular
14075810	posts	body	1	4	expression	NN	expres…
14075810	posts	body	1	5	that	W…	that
14075810	posts	body	1	6	recognizes	VBZ	recogn…
14075810	posts	body	1	7	email	NN	email

Preparing the data

- Example of minimalistic preparation
  - consider only title and body in **posts**
  - ignore linguistic knowledge and positions

id	tablename	what	sen	idx	word	pos	lemma
14075810	posts	body	1	1	Using	VBG	use
14075810	posts	body	1	2	a	DT	a
14075810	posts	body	1	3	regular	JJ	regular
14075810	posts	body	1	4	expression	NN	expres…
14075810	posts	body	1	5	that	W…	that
14075810	posts	body	1	6	recognizes	VBZ	recogn…
14075810	posts	body	1	7	email	NN	email

Preparing the data

- Example of minimalistic preparation
  - consider only title and body in posts
  - ignore linguistic knowledge and positions
  - here introduced in a new inverted index table called **wi**

```
drop table if exists wi;
create table wi as
select id,word from words
where word regexp '^[A-Za-z][A-Za-z0-9_]{1,}$'
and tablename = 'posts' and (what='title' or what='body')
group by id,word;
```

- What is this?

  `where word regexp '^[A-Za-z][A-Za-z0-9_]{1,}$'`
  - considering only words of length 2 or more of the form
    - first character – a letter,
    - following characters – alphanumeric

Using regular expressions

- MySQL pattern matching – two approaches
  - Simple matching with the **LIKE operator** (with '_' and '%')
  - More advanced regular expressions match with the **REGEXP operator**

Pattern	What the pattern matches
^	Beginning of string
$	End of string
.	Any single character
[...]	Any character listed between the square brackets
[^...]	Any character not listed between the square brackets
p1|p2|p3	Alternation; matches any of the patterns p1, p2, or p3
*	Zero or more instances of preceding element
+	One or more instances of preceding element
{n}	n instances of preceding element
{m,n}	m through n instances of preceding element

Using regular expressions, examples

- Query to find all the words starting with **'st'** –
  - `SELECT word FROM words WHERE word REGEXP '^st';`
- Query to find all the words ending with **'ok'** –
  - `SELECT word FROM words WHERE word REGEXP 'ok$';`
- Query to find all the words , which contain **'mar'** –
  - `SELECT word FROM words WHERE word REGEXP 'mar';`
- Query to find all the words starting with a vowel and ending with **'ok'** –
  - `SELECT word FROM words WHERE word REGEXP '^[aeiou].*ok$';`

Queries

- **Exact match queries**
  - Input: a set of keywords
  - Output: the list of posts that contain all the given keywords
  - This can e.g. be done by intersecting the postings lists (derivable from the words table
  - However, intersection is not supported in MySQL, so an alternative formulation is needed

- Ranking
  - Is an issue to be further elaborated
  - here's a simple ranked approach:

- **Best-match queries**
  - Input: a set of N keywords
  - Output: the list of posts that contain one or more of the given keywords
    - In decreasing order wrt. the number of contained keywords

Ranking and ordering

- Example: best-match and 3 keywords;

```
drop procedure if exists bestmatch3;
delimiter //
create procedure bestmatch3
       (in w1 varchar(100),in w2 varchar(100),in w3 varchar(100))
begin
  select id, sum(score) rank, body from posts,
    (select distinct id, 1 score from wi where word = w1
    union all
    select distinct id, 1 score from wi where word = w2
    union all
    select distinct id, 1 score from wi where word = w3) t
  where id=postid group by id order by rank desc limit 15;
end //
delimiter ;
CALL bestmatch3('using','regions','blocks');
```

id	rank	body
9063	3	My approach is similar to a few others here, using regions to organize code blocks into constructors, proper
21541902	2	While a lot of answers show why you'd use a <code>yield</code> to create a generator, there are more use
12316	2	I didn't start to really appreciate the "using" blocks until recently. They make things so much more tidy :)</p
367662	2	Pointer

Dereferencing a <code>NULL</code> pointer
Derefe |
653104	2	The worst abuses (and I'm guilty of doing this occasionally) is using the preprocessor as some sort of data
225330	2	I'm writing a tool to report information about .NET applications deployed across environments and regions w
13842742	2	I think these answers address your side-by-side part of your question but does not explain the "same imag
26849599	2	An alternative to packed bitmaps and wheels - but equally efficient in certain contexts - is storing the differe
444995	2	First, I think you already considered using an ORM vs. rolling your own. I won't go into this one.
&#

Ranking and ordering

- Example: best-match and 3 keywords;

```
drop procedure if exists bestmatch3;
delimiter //
create procedure bestmatch3
       (in w1 varchar(100),in w2 varchar(100),in w3 varchar(100))
begin
  select id, sum(score) rank, body from posts,
    (select distinct id, 1 score from wi where word = w1
    union all
    select distinct id, 1 score from wi where word = w2
    union all
    select distinct id, 1 score from wi where word = w3) t
  where id=postid group by id order by rank desc limit 15;
end //
delimiter ;
CALL bestmatch3('using','regions','blocks');
```

- Trick to provide the solution (for best-match with 3 keywords)
  - use an individual subquery for each keyword,
  - add a score attribute with the value 1
  - combine the individual subqueries with “union all”,
  - group by postid’s and aggregate a “sum” over the score-attribute for the groups
  - voila … the resulting aggregate is the rank according to best-match

Dynamic procedure - Problem

- Challenges
  - Allow for any number of keywords in the query and
  - provide in the answer all posts that match at least one keyword

- If you aim for a stored procedure in MySQL you need it to be dynamic
  - should take any number of keywords as arguments

- Any number of arguments requires
  - ideally, the support of passing a list / array of parameters
    - but collection-type parameters are not allowed in MySQL
  - a less elegant solution could be by means of overloading (create bestmatch-procedure in several version with 1, 2, 3, etc. parameters)
    - but overloading is not an option in MySQL
  - however, a single parameter can be passed: a string (varchar) with a comma-separated list of words ...

Dynamic procedure - Towards a solution

- A rewrite procedure that decomposes a comma-separated list of words. Can be applied in (or modified into) a dynamic procedure

```
drop procedure if exists rewrite;
delimiter //
CREATE PROCEDURE rewrite(param VARCHAR(1000))
BEGIN
  /* Rewrites a comma-separated 'n-argument' input string parameter and
     compose a new dummy string as output with a new substrings in
     replacement for the beginning, every "in-between" (comma) and the end*/
  SET @s = 'BEGINNING-';
  SET @s = concat(@s,replace(param,',','-INBETWEEN-'));
  SET @s = concat(@s,'-THEEND');
  SELECT @s;

end //
delimiter ;
CALL rewrite('one,two,three,four');
```

@s
BEGINNING-one-INBETWEEN-two-INBETWEEN-three-INBETWEEN-four-THEEND

Dynamic procedure - Towards a solution

```
drop procedure if exists rewrite;
delimiter //
CREATE PROCEDURE rewrite(param VARCHAR(1000))
BEGIN
  /* Rewrites a comma-separated 'n-argument' input string parameter and
     compose a new dummy string as output with a new substrings in
     replacement for the beginning, every "in-between" (comma) and the end*/
  SET @s = 'BEGINNING-';
  SET @s = concat(@s,replace(param,',','-INBETWEEN-'));
  SET @s = concat(@s,'-THEEND');
  SELECT @s;

end //
delimiter ;
CALL rewrite('one,two,three,four');
```

- Trick to provide the solution (for dynamic procedure)
  - simple use of a the string functions replace and concat
  - build a string by combining the input-words with appropriate string constants for beginning, in between and ending
  - in the example we build the output-string
    - 'BEGINNING-one-INBETWEEN-two-INBETWEEN-three-INBETWEEN-four-THEEND'
  - but you can obviously also build an SQL expression in a similar manner

Dynamic procedure - Towards a solution

- So what?
  - now we know how to build an SQL-expression as a string
  - but what if we want to execute this?

Dynamic procedure - Towards a solution

- Build an SQL-expression as a string an execute it:

```
drop procedure if exists find;
delimiter //
create procedure find(w1 varchar(100), w2 varchar(100))
begin
SET @s="select body from posts where postid in (select id from words where word = '";
 SET @s=concat(@s,w1);
 SET @s=concat(@s,"') and postid in (select id from words where word = '");
 SET @s=concat(@s,w2);
 SET @s=concat(@s,"');");
 PREPARE stmt FROM @s;
 EXECUTE stmt;
end //
delimiter ;
CALL find('sql','injection');
```

- trick here:
  - build an SQL expression in a string variable @s,
  - use PREPARE and EXECUTE statements to turn it into a statement and execute it