COMP4321 Project Phase 1 Database Design

Group 8
WU, Yun-ju (ywubv@connect.ust.hk)
YOON, Hongseo (hyoon@connect.ust.hk)
SU, Heng (hsuab@connect.ust.hk)
DEL MUNDO, Gian Miguel Sero (gmsdelmundo@connect.ust.hk)

NOTE:

The database for this search engine project is created using **MapDB version 3.0.5**, not the JDBM library used in the course lab tutorials. MapDB is a successor to the original JDBM database library and hence this newer library was adopted to make use of the most stable and up-to-date features.

Overview:

The creation and the update of the database file is handled by the Indexer classes in the "com.findr.service.indexer" package. An interface "Indexer" is provided in Indexer.java, defining the basic operations that the Indexer class must be able to do, and the current implementation of the interface is the "MapDBIndexer" class. The indexer interface and the implementation was separated for the ease of code maintenance in possible changes or updates in the future for the indexer implementation.

As defined in the interface, indexers can:

- Open the database (.db) file from the project directory, load in all the databases in the file with "readDBFromDisk()" method.
- Get a collection of web pages to index from the crawler, and add or update the necessary indices with the "addWebpageEntries()" method.
- Make the change to the database file and close the file for completing the operation with the "commitAndClose()" method.
- Fetch all the information/attributes of a web page in the database, given its internal page ID with "getWebpage()" method.

MapDBIndexer:

MapDBIndexer is the current implementation of the Indexer used in this project, and it defines the specific designs of the database. The database (.db) file that the MapDBIndexer manipulates has a fixed name ("index.db"). The database loads the file, if it exists in the working directory, else a new db file will be created.

There are two long type numerical values, "pageID" and "wordID" kept by the MapDBIndexer. Both values are initialized to 0, and the values are incremented after every insertion to the databases. In case an existing database file is loaded, pageID and wordID are set to the size of the pageID_url database and the size of wordID_keyword database respectively. This is achieved through calling "counterEnable()" on the two databases upon initialization and using "sizeLong()" method to obtain the values.

MapDBIndexer creates db file that includes databases as shown below:

Type	Name
HTreeMap<String, Long>	keyword_wordID
HTreeMap<Long, String>	wordID_keyword
HTreeMap<Long, String>	pageID_url
HTreeMap<String, Long>	url_pageID
HTreeMap<Long, String>	pageID_title
HTreeMap<Long, Long>	pageID_size
HTreeMap<Long, Date>	pageID_lastmodified
HTreeMap<Long, String>	pageID_metaD
HTreeMap<Long, Integer>	pageID_tfmax
NavigableSet<Object[]>	content_inverted
NavigableSet<Object[]>	content_forward
NavigableSet<Object[]>	title_inverted
NavigableSet<Object[]>	title_forward
NavigableSet<Object[]>	parent_child

The web page attributes are separated and indexed into different databases as shown above, instead of a single database containing a mapping between the page ID to the Webpage object (containing all of them in one) to avoid unnecessary instantiation of Webpage object when not all of the web page information is needed during the indexing steps (details on indexing is explained in a latter section.

- keyword_wordID
    - This is the mapping table of an indexed keyword to its word ID, implemented with a hashmap. The key element is the keyword (String) and the value associated to it is the word ID (Long).

- wordID_keyword
    - This is the mapping table of the word ID of indexed keywords to the actual keyword in a hashmap in hashmap.The key element is the ID of the word (Long) and the value associated to the key is the actual keyword (String).

- pageID_url
    - This is the mapping table of the page ID of web pages indexed in the database and the URL of the web page in a hashmap.The key element is the page ID (Long) and the value is the URL of the page, represented in String form.

- url_pageID
    - This is the mapping table of the indexed web pages' URL (in String) to their page ID in a hashmap. The key element is the URL of the page (String) and the value is the page ID of the web page (Long).

- pageID_title
    - This is the mapping table of the page ID of indexed web pages to their page titles in a hashmap. The key element is the page ID of the web page (Long) and the associated value is the title of the page (String).

- pageID_size
    - This is the mapping table of the page ID of indexed web pages to their page sizes in a hashmap. The key element is the page ID of the web page (Long) and the associated value is the page size (Long).
- pageID_lastmodified
    - This is the mapping table of the page ID of indexed web pages to the last-modified-date of the pages in a hashmap. The key element is the page ID of the web page (Long) and the associated value is the last-modified-date of the pages, represented in Date type.

- pageID_metaD (*Not used in Phase 1)
    - This is the mapping table of the page ID of the indexed web pages to the meta-descriptions of the pages in a hashmap. The key element is the page ID of the web page (Long) and the associated value is the meta-description of the web page (String).

- pageID_tfmax (*Not used in Phase 1)
    - This is the mapping table of the page ID of the indexed web pages to the tfmax for the web pages (maximum frequency for all of its keywords), in a hashmap. The key element is the page ID of the web page (Long) and the value is the tfmax value (Integer).
    - The mapping table is checked for every keyword in a web page to be indexed and updated to keep the largest keyword frequency for the web page.

- content_inverted
    - This is the inverted index. Mapping the word ID of keywords to the page IDs of the web pages and the its keywords frequency (word ID -> page ID, frequency) Implemented with a treeset.
    - This design was adopted to implement a multi-map for the inverted index, associating multiple page ID and frequency values to a single word ID. The keys and values in mapDB databases are required to be immutable, and the multi-map implementation provides an efficient way to update the database

compared to other methods, such as repetitive modification of array of objects, which would require copying the old array to a new array of new size every update.

- Each entry in the content_inverted inverted index is an Object array (Object[ ]{ }), containing three elements, {word ID, page ID, frequency}. To insert a new entry to the inverted index, an Object array is constructed with the word ID, page ID, frequency and is added to the set. Since the set is sorted in the natural order of the elements (i.e. the Object array is sorted by the word ID in its increasing order, then in the order of the page ID, and frequency), retrieving all the entries for a given word ID can be achieved by taking a subset on this multi-map inverted index:

  e.g.
  Set<Object[]> entries = content_inverted.subSet(new Object[] {wordID}, new Object[] {wordID, null, null});
  - The first element, “new Object[ ] {wordID}” sets the lower bound on the elements to retrieve in the subset (the first Object array with the first element = wordID in the content_inverted)
  - The second element, “new Object[ ] {wordID, null, null}” sets the upper bound for the subset, which in this case means an entry with first element = word ID and maximum values for the rest of the two elements. This essentially returns all entries that has “wordID”, giving us the list of page IDs and the frequencies for a given word ID.

- content_forward
  - This is the forward index, keeping the mapping of the page ID of the web page to the word ID of all the keywords in that page and the frequency of the keywords, implemented with a treeset.
  - The object array set was designed for the multi-map implementation which maps multiple word IDs and frequency values to a single page ID.
  - The implementation details are identical to the inverted index, except the difference in the order of the objects in the array, in which the page ID comes first for the forward index, making it {page ID, word ID, frequency}.
  - Entry insertion and retrieval are done in the same way as the content_inverted inverted index.

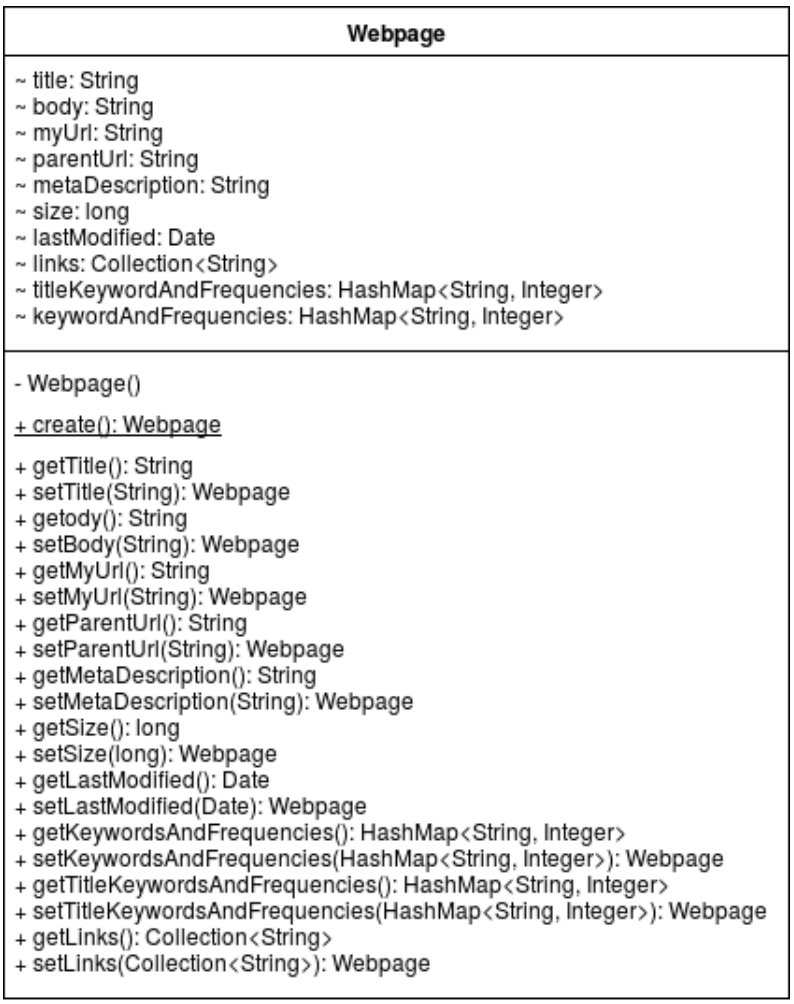
- title_inverted
  - This is the inverted index for the keywords in the title of web pages. The storing format and the design of the database are identical to that of the content_inverted. The only difference is that only keywords appearing in the are indexed.

- title_forward
  - This is the forward index for the keywords in the title of web pages. The storing format and the design of the database is identical to that of the content_forward. Same as title inverted, only keywords in the title are indexed.

- parent_child
    - This is the mapping table providing the mapping between a web page and its children web pages (pages corresponding to the links found on the parent web page).
    - An Object array treeset implementation is used as well to create a multi-map for a single page ID to multiple children pageIDs.
    - The array contains two Long objects in the order of {Parent_pageID, Child_pageID}

Indexing of Web Pages:

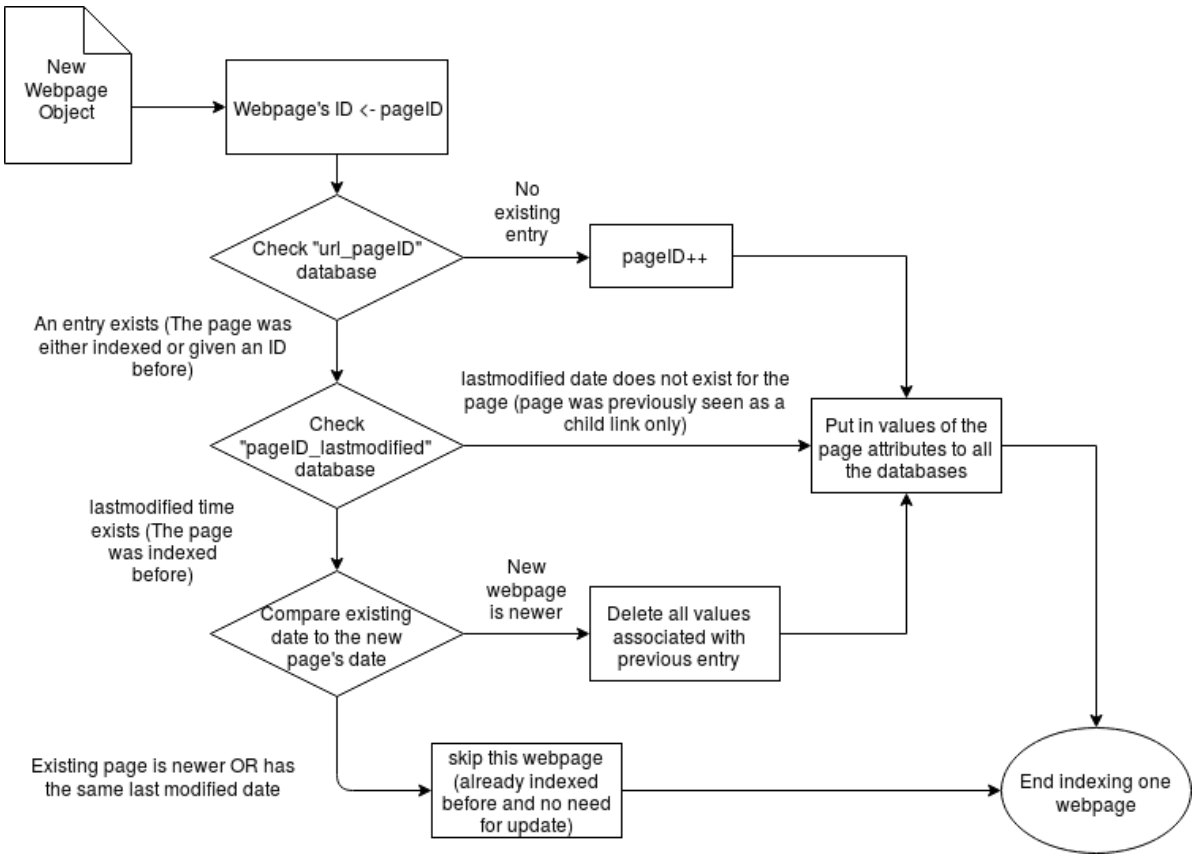
- In the Indexer, indexing of the web pages are done by calling the "addAllWebpageEntries" method, which takes a List of Webpage objects (the web pages to be indexed). The Webpage class can be found under the "com.findr.object" package. The webpage objects hold the web page information/attributes to be indexed in their fields. The "addAllWebpageEntries" calls the "addWebpageEntry" method, which takes in one Webpage object and indexes it, for all the Webpage objects in the list.
- The diagram below shows the Webpage class structure. Necessary information of a page is taken with the getter functions during indexing.



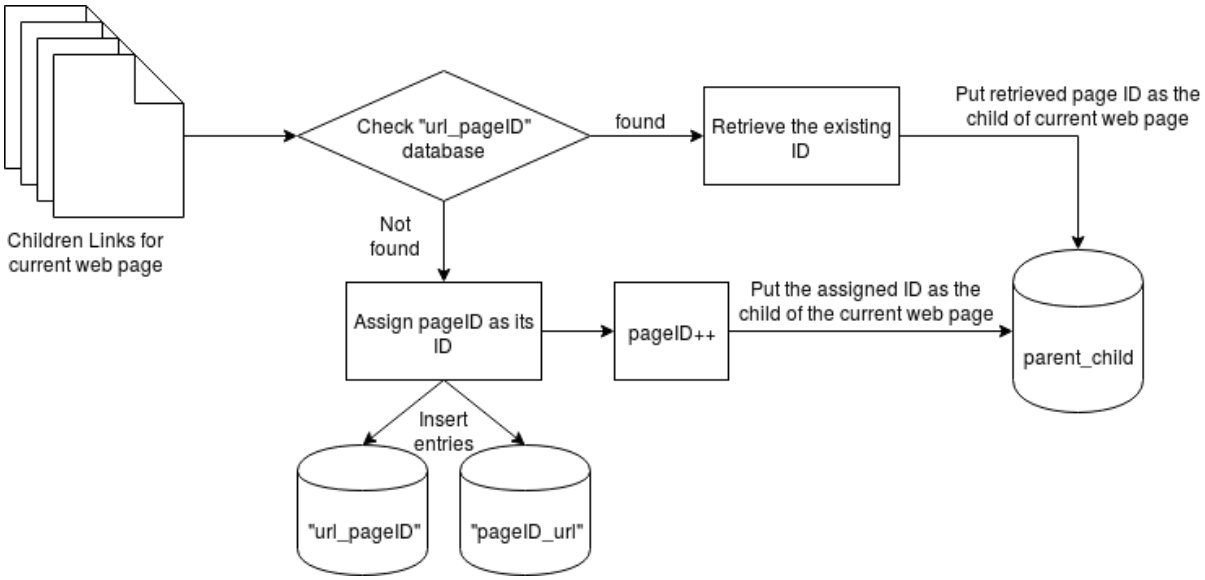
- In the indexing of a web page, there are three cases that the indexer needs to handle
    1. The web page is a completely new web page (has not been indexed before / assigned a page ID before)
    2. The web page has previously been indexed
    3. The web page was given a pageID before, but was not indexed

- In the first case, the webpage takes the current value of pageID variable as its pageID, increments the variable for the next indexing, then simply add all the web page data into the corresponding databases.
- In the second case, the pageID_lastmodified database is checked to retrieve the last-modified-time of the existing entry and the time is compared against that of the new web page that is about to be indexed. If the new webpage is newer than the existing one, then all the data associated with the previous entry is removed and the new web page is indexed with the pageID of the previous entry. If the current working web page has older or the same modified time, then the web page does not need to be indexed and it skipped.
- The third case could happen when the current web page to be indexed was seen by the indexer previously, while handling the children pages of another web pages. The web page would have been assigned a page ID when the indexer recorded the page as a child page of the previous page, so the indexing of the page should be done with the ID assigned to it.

The following flow-chart visualizes the indexing process in MapDBIndexer:

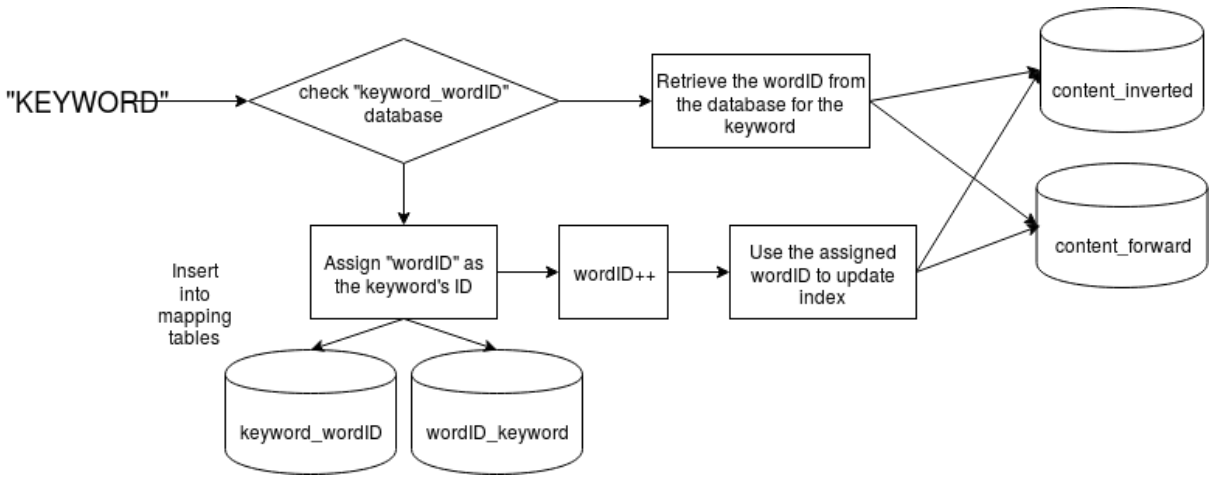


The child links and page ID assigning process is described in the diagram below:



The keywords are indexed and added to the databases in a similar way the children links are added to the databases. For each web page that is indexed, the collection of its keyword and frequency pair is retrieved from the Webpage object, then each of the keywords are checked with the “keyword_wordID” database for existing entries the ID is assigned to the keyword by either incrementing the wordID variable (if it is a new keyword) or by taking the word ID from the database (if the keyword appeared in other documents before and has been put into the mapping table).

The following diagram shows the keyword indexing process for a single keyword in a web page indexing:



The diagram only shows the indexing for content keywords, but the indexing of the title keywords are done in the same manner.