**Inverted index**

From Wikipedia, the free encyclopedia

Jump to: [navigation](http://en.wikipedia.org/wiki/Inverted_index#mw-head), [search](http://en.wikipedia.org/wiki/Inverted_index#p-search)

In [computer science](http://en.wikipedia.org/wiki/Computer_science), an **inverted index** (also referred to as **postings file** or **inverted file**) is an [index data structure](http://en.wikipedia.org/wiki/Index_%28database%29) storing a mapping from content, such as words or numbers, to its locations in a [database file](http://en.wikipedia.org/wiki/Table_%28database%29), or in a document or a set of documents. The purpose of an inverted index is to allow fast [full text searches](http://en.wikipedia.org/wiki/Full_text_search), at a cost of increased processing when a document is added to the database. The inverted file may be the database file itself, rather than its [index](http://en.wikipedia.org/wiki/Index_%28database%29). It is the most popular data structure used in [document retrieval](http://en.wikipedia.org/wiki/Document_retrieval) systems,[[1]](http://en.wikipedia.org/wiki/Inverted_index#cite_note-0) used on a large scale for example in [search engines](http://en.wikipedia.org/wiki/Search_engine). Several significant general-purpose [mainframe](http://en.wikipedia.org/wiki/Mainframe_computer)-based [database management systems](http://en.wikipedia.org/wiki/Database_management_systems) have used inverted list architectures, including [ADABAS](http://en.wikipedia.org/wiki/ADABAS), [DATACOM/DB](http://en.wikipedia.org/wiki/DATACOM/DB), and [Model 204](http://en.wikipedia.org/wiki/Model_204).

There are two main variants of inverted indexes: A **record level inverted index** (or **inverted file index** or just **inverted file**) contains a list of references to documents for each word. A **word level inverted index** (or **full inverted index** or **inverted list**) additionally contains the positions of each word within a document.[[2]](http://en.wikipedia.org/wiki/Inverted_index#cite_note-isbn0-201-39829-X-p192-1) The latter form offers more functionality (like [phrase searches](http://en.wikipedia.org/wiki/Phrase_search)), but needs more time and space to be created.

|  |
| --- |
| **Contents**   * [1 Example](http://en.wikipedia.org/wiki/Inverted_index#Example) * [2 Applications](http://en.wikipedia.org/wiki/Inverted_index#Applications) * [3 See also](http://en.wikipedia.org/wiki/Inverted_index#See_also) * [4 Bibliography](http://en.wikipedia.org/wiki/Inverted_index#Bibliography) * [5 References](http://en.wikipedia.org/wiki/Inverted_index#References) * [6 External links](http://en.wikipedia.org/wiki/Inverted_index#External_links) |

**Example**

Given the texts T_0="it is what it is", T_1="what is it" and T_2="it is a banana", we have the following inverted file index (where the integers in the set notation brackets refer to the subscripts of the text symbols, T_0, T_1etc.):

"a": {2}

"banana": {2}

"is": {0, 1, 2}

"it": {0, 1, 2}

"what": {0, 1}

A term search for the terms "what", "is" and "it" would give the set \{0,1\} \cap \{0,1,2\} \cap \{0,1,2\} = \{0,1\}.

With the same texts, we get the following full inverted index, where the pairs are document numbers and local word numbers. Like the document numbers, local word numbers also begin with zero. So, "banana": {(2, 3)} means the word "banana" is in the third document (T_2), and it is the fourth word in that document (position 3).

"a": {(2, 2)}

"banana": {(2, 3)}

"is": {(0, 1), (0, 4), **(1, 1)**, (2, 1)}

"it": {(0, 0), (0, 3), **(1, 2)**, (2, 0)}

"what": {(0, 2), **(1, 0)**}

If we run a phrase search for "what is it" we get hits for all the words in both document 0 and 1. But the terms occur consecutively only in document 1.

Information retrival..

由于提前扫描、准备好了数据结构，所以后来查找数据时候就加快速度了，这是相当的自然，因为我们花了更多的时间在数据结构、更多精力在算法上，所以后来的查找速度快，容易是很合理的。