COMPSCI732

Indexes and storage organisation in the eXist NXD

eXist

eXist is an open source NXD

eXist is best suited for applications dealing with small to large collections of XML documents that are updated occasionally

from http://exist-db.org/

Index and data organisation

eXist uses four index files at the core of the XML storage backend:
- dom.dbx collects DOM nodes and associates unique node identifiers to the actual nodes
- collections.dbx manages the collection hierarchy
- elements.dbx indexes elements and attributes
- words.dbx keeps track of word occurrences and is used by the fulltext search extensions

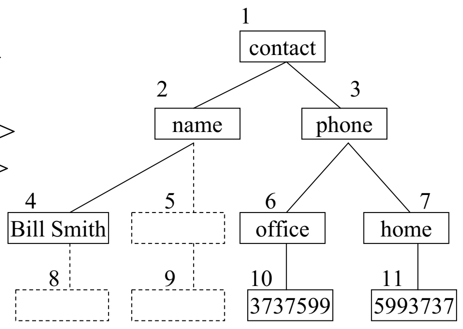
All indexes are based on B+-trees.

From http://exist-db.org/webdb.pdf
eXist: An Open Source Native XML Database

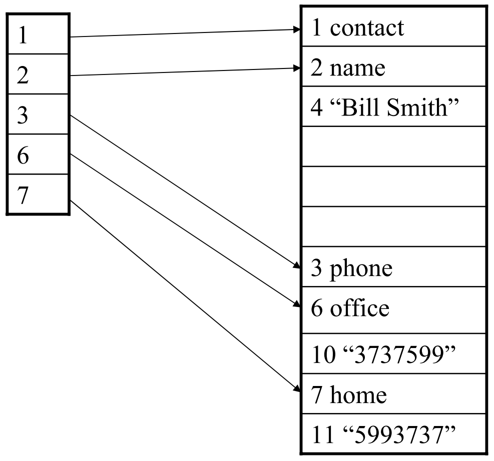
dom.dbx

Represents the central component of eXists native storage structure.

```
<contact>
  <name>Bill Smith</name>
  <phone>
    <office>3737599</office>
    <home>5993737</home>
  </phone>
</contact>
```

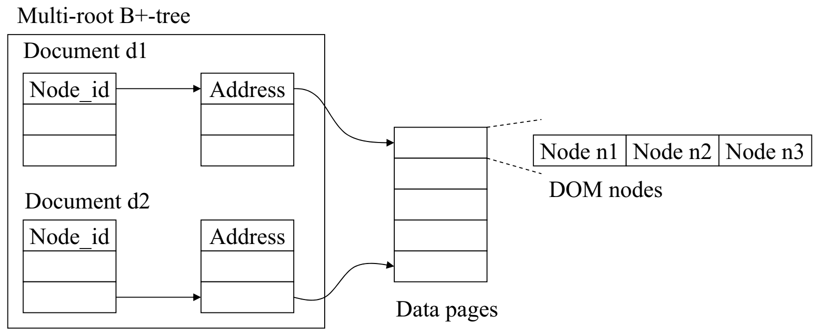


dom.dbx continued



Used to find ancestor/descendant, parent/child, sibling.

dom.dbx generally



Only top-level elements are indexed in the B+-tree.
Lower level nodes are just written to the data pages without adding a key to the B+-tree.
The cases where direct access to these nodes is required is very rare.

Collections

Documents can be divided into collections. From a users point of view, this is like storing files in a file system. e.g. one collection might contain documents describing the initial design of a computer system, another might contain documents in the detailed design, and yet another might contain the manuals for the system.

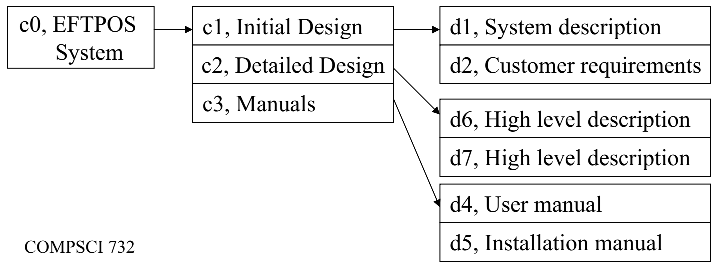
The collections can be arranged in a collection hierarchy.

Arbitrary documents may be mixed within the same collection.

Users usually query entire collections or even several collections at once. This is an assumption underlying the design of the indexes in eXist.

collections.dbx

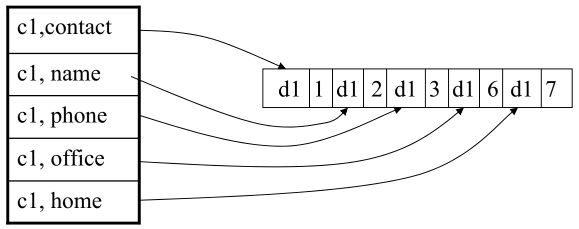
Manages the collection hierarchy and maps collection names to collection objects.
For performance reasons, document descriptions are stored with the collection object they belong to.
Improves queries that ask for all sections in c0 that have ‘XML’ in their title.



elements.dbx and words.dbx

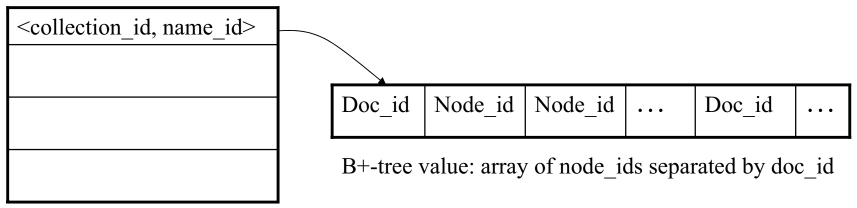
elements.dbx and words.dbx are organised by collection and not by document. E.g., all occurrences of a “section” element are stored as a single index entry in the elements index. This helps to keep the number of inner B+-tree pages small and yields a better performance for queries on entire collections. This is based on experience with previous versions which showed that creating an index entry for every single document in a collection leads to decreased performance for collections containing a larger number (>1000) of small documents (<50 KB).

elements.dbx



Find all the documents in collection c that have an element “phone”.

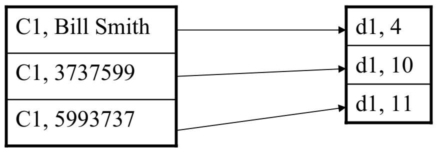
element.dbx generally



B+-tree value: array of node_ids separated by doc_id

B+-tree keys

words.dbx



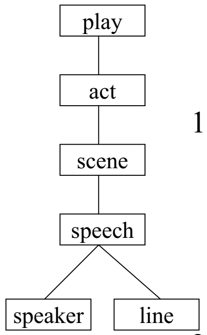
Find all elements that contain the keyword “3737599”.
By default, eXist indexes all text nodes and attribute values.
It is possible to exclude distinct parts of documents or switch full-text indexing off completely.

Indexes are used to…

- Access distinct nodes by their node ids
- Retrieve a list of node ids for a given node name
- Retrieve a list of node ids for text or attribute nodes containing a specified keyword

Example of how indexes used

Used a publicly available collection of Shakespeare plays.



/play//speech[speaker = 'Hamlet']

1 Find play//speech
    1.1 find root element play for all documents using elements.dbx
    1.2 find speech elements using elements.dbx
    1.3 compare <doc_id, node_id>s
    1.4 result is set of descendants <doc_id, node_id>
2 Find speaker using element.dbx and for each descendant do comparison with result above
3 Output speech node where speaker = 'Hamlet'