信息检索技术
清华同方光盘协办

编者按：清华同方光盘股份有限责任公司为发展我国“信息检索技术”，在理论和实践上推动网络信息检索技术的发展与应用，以进一步加快图书情报技术网络化发展进程愿与本刊合作，协办本栏目的工作，为此编辑部代表广大读者对清华同方光盘股份有限责任公司支持我国图书情报领域计算机信息检索技术发展的举措，表示衷心的感谢！

全文检索系统中动态索引技术的研究与实现

向桂林 刘锦华

（中国科学院文献情报中心 北京 100080）（山东胜利石油管理局孤岛采油厂 东营 257200）

【摘要】 分析了传统全文检索系统中静态索引技术的实现，讨论了静态索引技术的优缺点；然后提出来一种动态索引技术，阐述了动态索引技术的原理，并在两个数据库开发平台上给出了动态索引技术的实现。

【关键词】 全文检索 静态索引 动态索引 倒排文档 【分类号】 G354.4

Research and Completion on Dynamic Index Technology in Full Text Retrieval

Xiang Guilin

(*Library of Chinese Academy of Science, Beijing 100080, China*)

Liu Jinhua

(*Gudao Oil Extraction Factory of Shengli Petroleum Bureau, Dongying 257200, China*)

【Abstract】 The paper analyses the completion of static index technology in traditional full text retrieval system, discusses its merits and fault, then puts forwards a kind of dynamic index technology, elucidates its theory, completes it based on two database development kits.

【Keywords】 Full text retrieval Static index technology Dynamic index technology Inverted file

1 检索技术的发展

1.1 前方一致检索方式

在传统关系型数据库中，字段检索是按“前方一致”的检索方式。比如“毛主席在井冈山的生活”（书名），用户如果用“井冈山”这个词去查询，就找不到这条书目。因为在关系型数据库中，它采用的是平铺键值后的定长格式索引，比如键长定义为10字节，则“毛主席在井冈山的生活”在索引时，变成了“毛主席在井”这种样子；如果键长定义为24字节，则“毛主席在井冈山的生活”在索引时，变成了“毛主席在井冈山的生活BBBB”，其中B表示空格。显然，这种存储、索引方式是不能满足人们的需求的。后来，为了解决这一问题，又提出了关键词检索。

1.2 关键词检索方式

从“毛主席在井冈山的生活”这一书名中，抽取出“毛主席”，“井冈山”，“生活”三个词，形成了如下关键词索引如表1：

表1 关键词索引

关键词	出现信息（记录号）
毛主席	1，……
井冈山	1，……
生活	1，……

这样一来，用户使用关键词检索时，输入“井冈山”，就会找到这一条书目记录了。这是不是就把用户的需求满足了？不是。我们知道，除了“毛主席在井冈山的生活”之外，还有“朱德在井冈山的生活”，“贺子珍在井冈山的生活”等，用户想通过用“井冈山的生活”这个短语查询，来了解究竟有多少条书目记录是讨论关于某人在井冈山的生活，上述的关键词检索方法就不灵了，因为“井冈山的生活”这个词不是关键字。因此，有人提出布尔检索，比如“井冈山”*“生活”就能查出所有的含有“井冈山”和“生活”的记录。但是，这是针对有文献检索知识的专业人员，或者经过培训的检索用户，广大的一般用户（比如互连网上的检索用户），并不知道布尔检索。因此布尔检索还是谈不上方便用户。又怎么办呢？研究检索技术的专家，提出了全文检索技术。

收稿日期：2002－10－10

1.3 全文检索方式

从“毛主席在井冈山的生活”这一书名中，把每个字都进行索引(或者叫倒排)，认为每个字都很重要，都是关键词，形成了如下的全文索引数据(见表2)：

表2 全文索引

字	出现信息(元组={记录号,偏移,……})
毛	{1,0},……
主	{1,2},……
席	{1,4},……
在	{1,6},……
井	{1,8},……
冈	{1,10},……
山	{1,12},……
的	{1,14},……
生	{1,16},……
活	{1,18},……

针对这种索引，用户输入“井冈山的生活”，还无法直接查找，检索系统要经过一个匹配的过程。首先把短语中的每个字的出现信息取出来，然后再匹配。匹配是从查询短语的第一个字开始的，首先是第一个字与第二个字的位置信息匹配，如果匹配后结果集合不为空，再把结果集合与第三个字的位置信息比较，以此类推。直到结果集合为空，或者所有字都匹配完成。比如查询短语为“井冈山的生活”，“井”{1,8}、“冈”{1,10}、“山”{1,12}，“井”与“冈”字的记录号相同，且偏移值相差为2，说明“井”、“冈”二字是同在某记录中相邻位置出现，然后把匹配结果集{1,10}与第三个字“山”的出现信息{1,12}比较，发现“冈”与“山”字也是同记录中相邻出现，以此类推，最后发现“井冈山的生活”在记录1中出现。利用全文检索技术，用户就可以用自然语言在数据库中查询，不用记忆、学习各种检索语言，诸如布尔检索、位置检索、相邻检索等。推而广之，如果把整篇文档作了全文索引，用户输入“毛主席在井冈山”这个短语，就会找到这个短语在整个文档中的所有出现，这种检索方式在法律法规全文、产品技术标准全文检索中有重要意义。

全文检索系统本身比较复杂，包括索引构造技术、字索引技术、词索引技术、词干抽取技术、分词技术、索引压缩技术、字匹配技术、人机接口技术等等。本文只涉及索引构造技术中的静态索引技术与动态索引技术。所谓的静态索引，就是假设被索引的原文基本不变化，或者更新的周期很长，一旦被索引的原文更新，所有的索引文件都要重新做一遍，也有人称之为索引重装技术。所谓的动态索引技术，就是假定被索引的原文是频繁变化的，一旦被索引的原文更新，索引程序只是对新添加或修改后的原文进行索引，以前的索引(即对应原文未被改变的那部分索引)不必重做，也有人称这种索引方式为增量式索引。下面分别讨论静态索引技术和动态索引技术。

2 静态索引技术

静态索引技术早在20世纪70年代初就在Dialog系统中实现了。在静态索引技术中，整个数据库由4个文档组成[1]。如图1所示

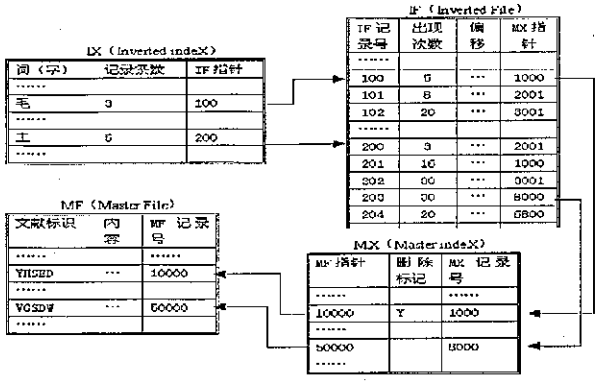

图1 静态索引技术意图

主文档(Master File)，存储与打印格式相似的文件正文，包括文件标识(格式、长度)以及每一段的格式长度、段代码和段内容。数据库中的全部文档按顺序文档方式存放。

主索引(Master index)，每条记录由指针、文献号、删除标志、文献密级和格式化数据组成。其中，指针指向主文档，格式化数据来自主文档中相应的格式化字段。

倒排文档(Inverted File)，与一般的倒排文档相似，但对于每个词在文件中出现的情况提供更详细的信息。每个记录由控制串及若干个出现值构成。控制串项的内容有某词的文献数量、总词频及出现位置数据。这些位置信息使得系统能在倒排检索阶段就能支持各种位置检索，检索过程中不必查询主文档便能满足检索者的各种检索运算要求。

倒排索引(Inverted index)，是系统利用“停用词表”进行自动抽词后生成的文档。每个记录由5项内容组成，即倒排文档指针、总文献量、总词频、同义词指针和词本身。

静态索引技术到现在都还很流行，笔者查阅了Ricardo Baeza－Yates在1999年出版的专著Modern Information Retrieval[2]，Charles T. Meadow在2000年出版的专著Text Information Rerrieval Systems[3]，两本专著在讲解全文倒排技术时，都只讲了静态索引技术。

静态索引技术有它的优点，就是把索引分小。如果把IX和IF放在一起，索引显得很大，内存中放不下索引数据，把IX和IF分开后，IX(有人称IX为数据库的词典)就基本可以存放在计算机的内存中，从而加快了查询速度；其次，静态索引技术与被索引原文的更新周期是相适应的，比如Dialog系统，他的更新周期以周、半月来计算，况且他的机器比较高档，每隔一周、半月之后，重新装入一次索引，也不觉得是一个负担。像Dialog这样联机检索系统，追求的是检索响应时间，每天都有几万人次访问，他的查询速度必须很快才行。但是，随着互联网的迅速发展，数字资源建设的突飞猛进，仅仅依赖于静态索引技术，还是很不够的。所以，笔者提出了动态索引技术。

3 动态索引技术

静态索引模型缺乏动态性主要是索引文件(IF)是连续

的，这样，在要对新加入的文件集添加索引时，必须要打乱原来的索引，导致所有索引需重新排序。这主要是数组的缺点。笔者根据数据结构中的链接表思想，提出了索引分块技术。

基本思想是：把索引分割成字节数相等的若干部分。如图2所示(X代表部分已填写了数据，— 代表该部分尚未填写数据)

……	XXXXXXXXX	XXX……	XXXXXXXX…

图2 索引分块技术示意

在索引创建过程中，每一个字符都有一条索引块链，并在一个数据结构中记录着每个字符的链首块的块号。这样，如新扫描到一个字符，根据该字符索引块链找到未填满的索引块，加入位置信息。如未找到(该字符第一次出现或刚刚填满一个索引块)，则分配给一个索引块，并将该索引块加入到该字符索引块链中。例如上例新增字符2，则修改索引如图3。

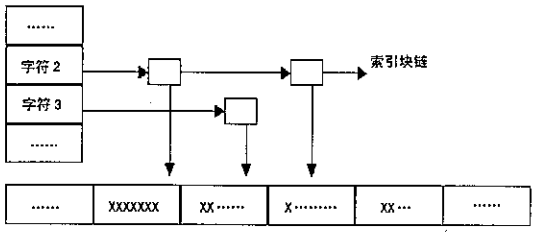

图3 新增字符2后的索引

从上面结构中可看出，对于检索来说，过程基本不变，只是增加字符索引块链，以便能从中找到与字符相对应的索引块。而在索引动态维护方面，如果是删除文件，这只要将相应记录清空、如果是增加文件，则如上所述，可在未填满的索引块里增加位置信息或新分配索引块。

由此可见，索引分块技术在不影响其它性能的情况下，较好的解决了索引的动态性需求，并且在索引建立过程中只需一次扫描文件集，因而提高了索引的建立速度。

4 动态索引技术的实现

实现动态索引技术主要是实现变长记录的存取。实现变长记录的存取，有三种办法，一是自己编制程序，也就是实现一个变长文件管理系统。如果仅仅是为了验证一个算法，这样做很好；二是借用别人已经开发好了的变长记录管理程序，读者可以参见《Turbo C 高级程序员手册》[4]。这本书上讲述了如何管理一个幻灯片的例子(幻灯片上的内容是可变长的，该书为这个例子定做了一套变长记录存取函数)；三是基于一个数据库开发平台来实现。笔者手中做的工作，是要开发一个功能完善的全文检索系统，所以笔者采用的是数据库开发平台。常用的数据库开发平台有Ctree[5]和Berkeley DB[6]。前者是由美国Faircom公司历经20余年开发出来的软件包，在银行业、航空业有着广泛的应用，Ctree是一个商品化的数据库开发软件包；后者是由加洲大学伯克利分校研制的免费的数据库开发平台，著名的MySql[7]关系型数据库和著名的免费全文检索系统 Mifluz[8]都是在 Berkeley DB 上开发出来的。目前，Berkeley DB 由 Sleepycat Software 公司维护。

在 Ctree 中，实现变长记录操作，要用到四个函数 AddVRecord，ReReadVRecord，ReWriteVRecord 和 DeleteVRecord。这四个函数的说明如下(见表3)：

表3 Ctree 中支持变长记录操作的函数

函数声明	COUNT AddVRecord(datno, recptr, bufsiz) COUNT datno; pVOID recptr; VRLEN bufsiz;
函数描述	AddVRecord 把记录按字节追加到数据文件中，datno 给出数据文件号，recptr 指向数据缓冲区，bufsiz 给出缓冲区的长度。数据追加完成后，自动更新与数据相依赖的索引。在多用户环境中，自动起用记录锁，保证追加的一致性。追加完成后，追加记录变成当前记录。
函数声明	COUNT ReReadVRecord(datno, recptr, bufsiz) COUNT datno; pVOID recptr; VRLEN bufsiz;
函数描述	ReReadVRecord 把当前可变长记录读入缓冲区中，数据文件号由datno 给出，缓冲区由recptr 指向。如果缓冲区的容量比记录的实际长度小，则返回一个错误。为了防止缓冲区溢出，可以先使用 VRecordLength 函数来返回变长记录的实际长度。在使用该函数时，程序员应保证缓冲区的容量足以装下变长记录。
函数声明	COUNT ReWriteVRecord(datno, recptr, varlen) COUNT datno; pVOID recptr; VRLEN varlen;
函数描述	ReWriteVRecord 允许对当前可变长记录进行更新，数据文件由datno 给出，缓冲区由recptr 指向，缓冲区的容量由varlen 给出。该函数假定缓冲区中的记录是原始版本，当该函数成功完成后，依赖于该数据记录的各种索引被相应自动更新，如果被更新后的记录长度比原来记录占用更多的磁盘空间，则为更新后的记录分配新的存储块。存储块的大小由数据库模式参数给出。
函数声明	COUNT DeleteVRecord(datno) COUNT datno;
函数描述	DeleteVRecord 删除当前的可变长记录，数据文件号由 datno 给出。如果数据记录被成功删除，则数据记录对应的索引被自动删除。注意的是：数据文件中的数据记录被删除后，但当前记录仍是被删除的那条记录，请用其它移动记录指针的函数获得一条当前数据记录。在多用户环境中，该函数自动起用记录锁，保证数据删除的一致性。数据记录删除后，数据块并没有归还给操作系统，只是做了一个删除标记。如果想把磁盘空间归还给操作系统，请用 CompactIFile 函数。

在 Berkeley DB 中，实现变长记录操作，只能采用 Recno 方式来访问数据库，用到的函数为DB—>put、DB—>get、DB—>del，函数的说明如下(见表4)：

上面讲的是变长记录存取的实现，在索引的逻辑结构设计时，笔者采用了如下的数据结构：

```
struct POS {   /* 某个字符在某条记录中的出现信息(偏移) */
    ULONG   recID;
```

```
    ULONG   cur_num;
    ULONG   *cur_pos;
};
struct POSINF {   /* 某个字符在数据库中的出现(记录号) */
    UCOUNT  chID;
    ULONG   rec_num;
    struct POS *rec_pos;
};
char   *chIDX[CHTOTAL];     /* 字符的位置信息打包数组 */
```

表 4 BerkeleyDB 中支持变长记录操作的函数

函数声明	Int DB->put(DB *db, DB_TXN *txnid, DBT *key, DBT *data, u_int32_t flags);
函数描述	DB->put 函数把键值/数据对存入数据库中。如果该键值已经存在,且数据库不允许重复键值,则替换原先的键值/数据对;如果该键值已经存在,且数据库允许重复键值,则把新的键值/数据对添加到已有的键值/数据对后面,如果数据库允许重复键值排序,则把新键值插入到合适的位置;如果数据库起用了事务保护操作,则该函数自动起用记录锁,保证添加数据记录的一致性。
函数声明	int DB->get(DB *db, DB_TXN *txnid, DBT *key, DBT *data, u_int32_t flags);
函数描述	DB->get 函数从数据库中检索出键值/数据对,data 结构指向返回的数据及相关的键值。如果数据库允许重复键值,则返回重复键值的第一条记录,要访问重复键值记录,请使用游标操作函数。
函数声明	int DB->del(DB *db, DB_TXN *txnid, DBT *key, u_int32_t flags);
函数描述	DB->del 函数从数据库中删除键值/数据对,与该键值/数据对相关的其它键值也一并被自动删除,在允许重复键值的情况下,所有重复键值都将被删除。

事实上,某个字符在多少条记录中出现,以及在某条记录中又出现了多少次都是可变的,这就是全文索引的难点之所在。笔者设计的两层结构,都是变长的。在Ctree 和 Berkeley DB 中,进行变长记录存入/取出时,要求有一个打包/解包的过程。上面的chIDX 就是一个打包/解包数组,它为每一个索引字符分配一块或多块内存,用来存放该字符的位置信息。为了加快索引的建立速度,可以采取动态内存分配技术:针对预先读入内存中的若干数据记录,扫描每一条记录,如果某个字被遇到,则分配 16K 或 8K 的内存,用来存放这个字的出现信息,当分配的内存空间被用完之后,再为这个字分配一块内存。这样一来,对于读入内存中的数据记录,它们对应的全文索引全部存放在内存中,从而大大地减少了写磁盘的次数,加快了对原文索引的速度。

在 Ctree 平台上,采用上述的逻辑结构和动态内存分配技术,笔者做的实验结果为:

(1) 在已有 96857 条记录的数据库中,追加10322 条记录,追加耗时 302 秒、索引耗时 819 秒;

(2) 往一个空库中追加 10322 条记录,追加耗时 290 秒、索引耗时 719 秒。除去机器本身的影响,基本上实现动态索引的目标。

5 结束语

本文讨论了静态索引技术与动态索引技术,并不是说一种技术比另一种技术更先进、更有用。这两种技术在不同的应用场合,有不同的功效。那些原文数据更新不是很频繁的场合,可以采取静态索引技术;对那些原文数据更新比较频繁的场合,比如网络爬行者抓取回来的网页、每日都要生产的书目数据,可以采用动态索引技术。采用动态索引技术或者静态索引技术都可以实现全文检索,这是方便用户的一种检索方法,这比前方一致检索,或者关键词检索,更加符合用户的使用习惯,让用户可以用自然语言与检索系统交互,这可能是人机界面设计者必须考虑的问题。

参考文献:

[1] 赖茂生,王延飞等. 计算机情报检索. 北京:北京大学出版社,1993:104—112

[2] Ricardo Baeza—Yates. Modern Information Retrieval. NewYork: ACM Press, 1999:191—198

[3] Charles T. Meadow. Text Information Retrieval Systems 2nd edition. San Diego:Academic Press, 2000:131—132

[4] 陈捍东. Turbo C 高级程序员编程指南. 北京:中国科学院希望高级电脑技术公司,1991:193~218

[5] http://www.faircom.com/support/ctpdoc/index.htm, 2002—09—29

[6] http://www.sleepycat.com/docs/index.html, 2002—09—29

[7] http://www.mysql.com/documentation/index.html, 2002—09—29

[8] http://www.gnu.org/software/mifluz/doc.en.html, 2002—09—29