关系数据库关键字检索系统的设计与实现

学　　院　计算机科学与技术

专　　业　计算机科学与技术

年　　级　2005

姓　　名　朱晓伟

指导教师　张坤龙

2009 年　**06** 月　**15** 日

摘　要

SQL 语言是检索关系数据库的传统方式。要想利用 SQL 语言查询数据库，用户不但需要事先掌握 SQL 语言的语法知识，还需要了解关系数据库的关系模式，因而比较复杂。关键字搜索则只需要用户输入关键词，它易于使用，所以近年来成为了一个研究热点。

论文设计和实现了一个基于关键词的关系数据库检索系统。利用该系统，用户只需输入几个关键词，就可以表达自己的检索需求。在用户输入关键词后，系统将自动地匹配所有的可能的结果，并按相关性的大小顺序将结果返回给用户。

在计算检索结果时，系统先采用启发式的方法找出所有可能与用户输入的关键词相关的元组连接模式，然后将属于该元组连接模式的复合元组搜索出来，最后将复合元组其转换为二维表的形式向用户展示。

关键词： 关系数据库；关键词；信息检索

ABSTRACT

People traditionally use Structured Query Language to search the data in relational databases. Before building the correct SQL statements, users need not only to master the grammars of SQL, but also to have basic knowledge about the schema of relation database, so it is rather complicated. Keyword search only requires the user to enter keywords, it is easy to use, so in recent years has become a research hotspot.

We will design and implement the system, by which users can search the relation database using keywords. Only inputting several keywords, can users find the wanted results. After users input several keywords to searching relation database, searching system will automatically match the relevant data and return these data in the order of relevance.

Search results in the calculation, the system first heuristic find schemas which are relevant to users' keywords. Then the system will return the results which belong to the selected schema and have relevance to users' keywords. Finally, convert it to two-dimensional form to the user display.

Key words：Relation Database；Keywords；Information Retrieval

目　录

第一章　绪论

1.1 背景介绍

从数据库技术诞生到现在，在不到半个世纪的时间里，形成了坚实的理论基础、成熟的商业产品和广泛的应用领域，给人们的日常工作与学习带来了许许多多的便利。但是随着数据库地发展与应用，也带来了一系列的问题与矛盾。例如在数据库中，SQL 语言这类专业性结构化语言也不方便为广大非计算机专业用户所使用与掌握。

这是因为首先用户不得不先掌握关系数据库的基础知识了以及 SQL 语言的语法规则等基础知识；其次在构造 SQL 语句之前用户必须事先了解关系数据库的模式；最后使用 SQL 语言构造查询，是复杂的并且容易出错的。因此 SQL 这类专业性结构化查询语言日益不能满足我们准确高效查询检索数据库中结构化数据的需要[1]。

我们都知道数据分为非结构化数据与结构化数据两大类。其中，非结构化数据是指文本文档等数据。而非结构化数据地查询与检索主要是使用关键字搜索地方法，例如百度等搜索引擎所使用的检索方法。这种查询结果不精确不完全而且需要根据相关性对查询结果进行排序。而结构化数据则是指数据库中的各种数据等。而针对结构化数据我们则采用复杂的结构化查询语言例如 SQL 语言以及 XQuery 语言等。查询结果是精确的完全的而且所有查询结果都被同样对待从而不需要排序。但是我们发现结构化查询语言不但比较复杂难以掌握并且在使用时容易出错。这就迫使我们去寻找一种更简洁的更高效的查询方法去检索查询结构化数据。例如使用关键字搜索方法检索查询结构化数据[2]。

关键词检索系统由于易用性而被广大用户所喜爱。因为在这种方式下，用户不需要任何的专业背景和技巧，只需输入几个关键词就可以完成查询，得到需要的查询结果。所以基于关键词的关系数据库检索方法，将能给用户提供极大的方便，更为用户所喜爱接受。

1.2 研究意义

使用关键字搜索这种检索方法去查询结构化数据能给我们的学习与工作带来许多便利。这种检索具有相当大的研究意义：

首先，广大的非计算机专业用户不再需要使用复杂地结构化查询语言。这样就不需要复杂地语言输入与操作，从而更加不易出错。只需要通过输入几个关键字便能由系统得出查询结果。这样既提高了工作效率又降低了出错率，从而方便了用户的操作与使用。

其次，从功能的上来说，数据库地关键词搜索是完成信息发现的任务。以前在数据库设计时将关系进行了规范化处理，这样原来作为一个整体的信息就可能

被分散到多个元组中。这样就造成了诸多不便与错误。而对数据库进行关键词搜索就能够发现和还原出原始的信息。这样就不会造成信息地丢失，从而保证了数据的完整性与准确性。

再次，这种检索方法也很好地避免了我们遇到不同地结构化数据就需要不同的结构化查询语言，减少了我们的工作量。例如关系数据要使用 SQL 语言，XML 数据要使用 XQuery 语言。所以数据库关键字搜索系统明显具有更好地数据兼容性，避免了我们需要使用大量地不同结构化查询语言[3]。

最后，随着数据库的广泛使用，就产生了一系列结构化数据需要使用关键字搜索查询方法进行处理。例如，公司的售后服务系统需要记录顾客反映的问题和公司解决问题的过程，并在下次遇到一个新问题时能够自动识别出该问题是否曾经出现过。目前互联网上有 80%的数据是存放在关系数据中。这些数据无法像普通 web 文档一样通过 web 搜索引擎进行检索。那么为这些存在于互联网上的大量关系数据提供类似于 web 搜索引擎式的检索方法就十分必要的。这类生活实际中新的问题的解决就需要我们设计出数据库关键字搜索系统来满足我们新的实际需要。

因此，使用关键字搜索这种方法来给我们查询结构化数据不但带来许多便利而且很好地避免了结构化查询语言的各种不足之处。从而体现出了这种新技术新方法的强大潜力以及广阔的发展运用前景。

1.3 相关工作

随着Hulgeri 等人在2001年时综述了与数据库关键词搜索系统有关的研究工作。数据库关键词搜索系统的研究与开发便逐渐取得了一系列研究成果。

目前主要出现了BANKS系统使用图来表示数据库其中图包含模式图与数据图，然后使用反向扩展搜索的启发式算法来寻找信息结点，接着使用嵌套表格的方式显示答案树，最后使用有向数据图进行关键字搜索[3]。

而DBXplorer系统则使用无向模式图来进行关键字搜索。关键词搜索主要包含三大步骤分别是搜索符号表，根据模式图计算连接树，对于每一棵计算出的连接树。最后由模式图计算生成连接树得出最终结果。

还有DISCOVER系统拥有两个版本，其中第一个版本与DBXplorer系统比较类似。除了根据元组集图计算候选网时是使用基于宽度优先遍历算法，而不用做元组集图的裁剪。但是第二个版本就有了很大的改变，主要体现在关键字搜索是使用OR语义而不同于以前的版本都是使用AND语义的。这样就不需要查询结果必须包含所有的关键词。同时还可以使用数据库系统的全文搜索功能，如Oracle数据库系统等[4]。

此外还有ObjectRank系统它首先将一个查询结果定义成为数据图中的一个结点。其次，它使用Google搜索引擎的PageRank算法来对查询结果进行计分。对

于用户提交的查询，ObjectRank系统的查询处理模块将根据分数索引计算出查询结果中一个结点的最终得分[5]。

以上这些所有研究工作都希望能够达到相同的目标：用户既不需要学习SQL等结构化查询语言，也不需要详细了解数据库的模式，他们只需要输入一组关键词，系统就能够自动列出所有相关的查询结果。

1.4 本文所作的工作

本文对关系数据库关键词检索系统的查询模型进行了研究。给出了关系数据库关键词检索的相关基础知识，并完成了系统体系架构与工作流程。最后很好的实现了一个系统的生成。本文所作的工作主要是以下方面：

1，熟悉并掌握数据库的相关知识；阅读相关的中外文献，了解所需的相关信息；进行 C#语言的学习以及 SQL Server 2003 与 Visual Studio 2008 等软件程序的安装与使用。

2，给出了数据库关键字检索的相关基本概念、查询模型和常用算法。

3，给出了数据库关键字检索系统的体系架构和工作流程。

4，将数据库关键字检索系统实现并给出相关的开发环境、基本架构、核心算法、实现技术和实现结果。

1.5 论文结构

本文的内容分为五章。第一章，介绍了关系数据库关键词搜索系统的研究背景、研究意义、相关工作和本文所做的工作。第二章，介绍了数据库关键词检索的基本概念、查询模型和常用算法。第三章，系统的体系结构和工作流程进行了详细论述。第四章，描述了系统实现的开发环境、基本架构、核心算法实现、实现技术和实现结果。第五章，对论文进行总结，并对未来工作进行展望。

第二章　数据库关键字检索

2.1 基本概念

2.1.1 关系数据库的定义及其实例

在一个给定的应用领域中，所有实体及实体之间联系的关系的集合构成一个关系数据库[6]。例如文献系统可以使用关系数据库来进行管理。下面的图2-1给出了一个文献数据库的模式。在这个文献数据库中，Author表用来书写作者的姓名和电子邮件地址，Writes表用来书写作者与论文间的写作联系，Paper 表用来书写论文的标题和发表年份，Cites表用来登记论文间的引用联系。在图2－1中，表的码使用粗斜体标识，表间的一条连线表示一个“外码→主码”联系并且箭头指向主码。

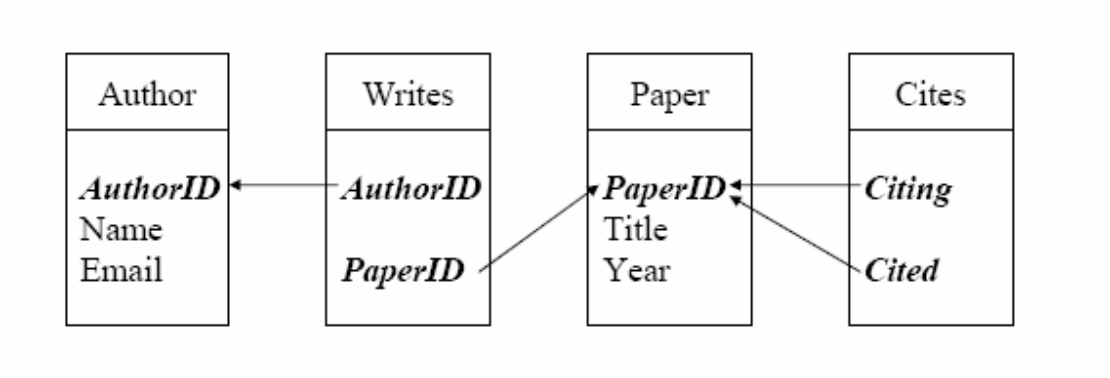

图2-1　文献数据库的模式

图2-2给出了上述文献数据库的一个实例。在这个实例中作者Vagelis Hristidis 和Yannis Papakonstantinou 合作写的论文“DISCOVER: Keyword Search in Relational Databases”被作者Vagelis Hristidis 、Luis Gravano 和Yannis Papakonstantinou 合作写的论文“Efficient IR-Style Keyword Search over Relational Databases”所引用。

因此如果用户想知道作者Hristidis和作者Gravano之间具有什么样的联系。就可以通过图2-2得到，其中比较简单的关系有两个作者合作写过的论文：Hristidis写的论文被Gravano写的论文所引用、Gravano写的论文被Hristidis写的论文所引用、曾和Hristidis合作写过论文的作者也和Gravano合作写了一篇论文等等。都可以通过图2-2清晰的表达出来，因此这上面详细地描述了各个实体与实体之间的联系。

Author

AuthorID	Name	Email
133701	Vagelis Hristidis	vagelis@cs.ucsd.edu
5875	Luis Gravano	gravano@cs.columbia.edu
3003	Yannis Papakonstantinou	yannis@cs.ucsd.edu

Writes

AuthorID	PaperID
133701	207959
3003	207959
133701	207806
5875	207806
3003	207806

Paper

PaperID	Title	Year
207959	DISCOVER: Keyword Search in Relational Databases	2002
207806	Efficient IR-Style Keyword Search over Relational Databases	2003

Cites

Citing	Cited
207806	207959

图2-2 文献数据库的实例

因此通过以上实例我们能够很清楚地通过图表详细明白数据库中实体与实体之间的各种关系。

2.1.2 查询结果的定义

对于一个关系数据库关键词检索系统来说，首先要做的就是对查询结果的定义。因为对查询结果的定义不同，系统的应用情境也就不同。而查询结果的定义主要是指一个查询结果的构成过程。关系数据库关键词检索系统的查询过程，是将信息重现和匹配的过程；是将由于关系数据库的规范化被打碎分散到各个元组中的信息重新拼接并与用户关键词进行匹配的过程。其中重现的过程是将各个元

组之间所有可能的联系再次连接的过程。而匹配则是考察用户关键词与重现后的信息单元是否相关的过程。而且因为查询结果的定义会影响系统的体系结构，所以对于不同定义的查询结果记分方法也不相同。同样对于用户来说，查询结果的含义越容易理解越好。

2.1.3 查询结果的排序

与 web 搜索引擎类似，关系数据库关键词检索系统也需要考察用户关键词与查询结果之间的相关程度并对查询结果进行排序。通常是对查询结果给予一个分数，这个分数反映它和查询的相关性，结果是分数值越大那么与查询的相关性就越高。而查询结果的排序就是将查询结果的分数值从大到小进行排序。所以一个查询结果记分的方法就显得相当重要了。因为关系数据库关键词检索系统的查询结果是最小元组连接树，它的相关性排序需要考虑各个元组之间的联系程度。那么组成元组连接树的各个元组之间的联系越紧密，就越符合用户的查询需求。反之，如果组成元组连接树的各个元组之间的联系越薄弱，就越不符合用户查询需求。这个关系数据库关键词检索系统就是按照元组连接树所包含元组的数目对结果进行排序。其中元组连接树所包含的节点越少，那么元组间的联系就越紧密，其在查询结果中的排序也就越紧密。

2.1.4 数据模型

数据模型就是数据库关键字检索系统是使用什么数据类型表达数据库。对于大多数的数据库关键字检索系统来说，都是使用图表来表示数据库。其中数据库的图可以是有向的也可以是无向的，主要包括模式图与数据图两种。例如 BANKS 系统便是通过有向的数据图来检索信息数据。在数据库关键字检索系统中，使用结点表示元组同时使用边表示元组之间的“外码→主码”联系。此外，web 与 XML 也可以使用图来表示，非常方便简洁[7]。

2.1.5 系统的体系结构

关系数据库关键词检索的体系结构一般分为预处理和查询处理两个部分。其中预处理模块是将数据库中的关系模式作为输入，输出各个元组的全文索引。而查询处理模块则是将用户输入的关键词集合为输入，输出与用户关键词相关的最小元组连接树列表。在关系数据库关键词检索系统的具体实现中，通常先由预处理模块为关系数据库中的元组生成所需要的全文索引，并将关系模式保存在系统数据库中。而查询处理模块则分为五个子模块：基础元组集产生模块、独立元组集计算模块、元组集连接图构建模块、候选元组集连接树产生模块和最小元组连接树产生模块[8]。

2.2 查询模型

查询模型主要包括模式图、数据图、元组连接树和元组集连接图等。

2.2.1 模式图与数据图

模式图与数据图生成的相关概念如下：

1，在R(A1,A2,…,An)中，X，Y是{A1,A2,…,An}的子集。若对于任意一个可能的关系r，r中不可能存在两个元组在X上的属性值相等，而在Y上的属性值不等，则称Y函数依赖于X。

2，在关系模式R中，如果Y函数依赖于X，并且对于X的任何一个真子集X’，Y 函数依赖于X’都不成立，则称Y完全函数依赖于X。

3，在R(A1,A2,…,An)中，K是{A1,A2,…,An}的子集，若{A1,A2,…,An}完全函数依赖于K则称K为R的候选码。若候选码多于一个，则选定其中一个为主码。

4，关系模式R中属性或属性组X并非R的候选码，但X是另一个关系模式的候选码，则称X是R的外码[9]。

5，标号图是一个四元组G=(V,E,Σ,L)，其中：V是结点的集合；E是边的集合；Σ是标号的集合；L是标记函数，L : V∪E→Σ。

6，若标号图G=(V,E,Σ,L)和G’=(V’,E’,Σ’,L’)是同构的，则存在一一映射I : V∪E→V’∪E’满足：（1） I(v)=v’，其中v∈V，v’∈V’。(2) I(e)=e’，其中e∈E, e’∈E’，e=(v1,v2), e’=(I(v1), I(v2))。（3）如果I(x)=x’，那么L(x)= L’(x’)[10]。

7，若无向标号图G是关系数据库的模式图，则存在这样的一个一一映射: 关系数据库中的关系Ri对应于G中的结点vi，并且当关系数据库中的两个关系Ri和Rj间有联系Ri→Rj时，G中有对应的边(vi, vj)。

8，若无向标号图G是关系数据库的数据图，则存在这样的一个一一映射: 关系数据库中的元组ti对应于G中的结点vi，并且当关系数据库中的两个元组ti和tj间有联系ti→tj时，G中有对应的边(vi, vj)。

按照如下方式构造文献数据库如图2-3所示的模式图G=(V,E, Σ,L)：

（1）V={v1, v2, v3, v4}；

（2）E={e1, e2, e3, e4}，其中e1=(v2, v1)，e2=(v2, v3)，e3=(v4, v3)，e4=(v4, v3)；

（3）Σ={A, W, P, C, 1, 2, 3, 4}；

（4）L(v1)=A，L(v2)=W，L(v3)=P，L(v4)=C，L(e1)=1，L(e2)=2，L(e3)=3，L(e4)=4[11]。

其中使用A, W, P, C表示文献数据库的四个关系Author，Writes，Paper，Cites，使用数字1,2,3,4表示文献数据库的四个“外码→主码”联系分别是Writes.AuthorID→Author.AuthorID ， Writes.PaperID→Paper.PaperID ，

Cites.Citing→Paper.PaperID，Cites.Cited →Paper.PaperID。构造的模式图结果如图2-3所示，构造的数据图结果如图2-4所示。

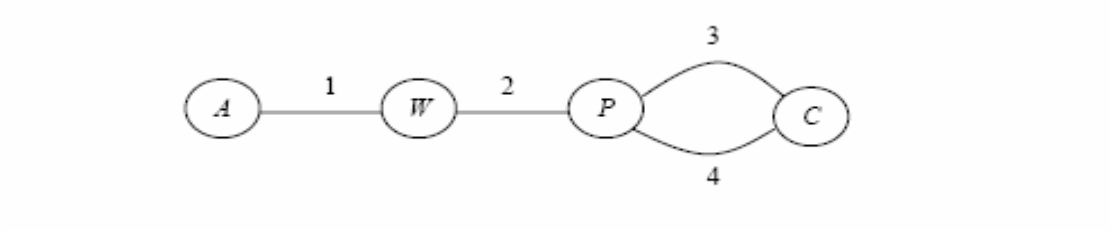

图 2-3 带标号的文献数据库模式图

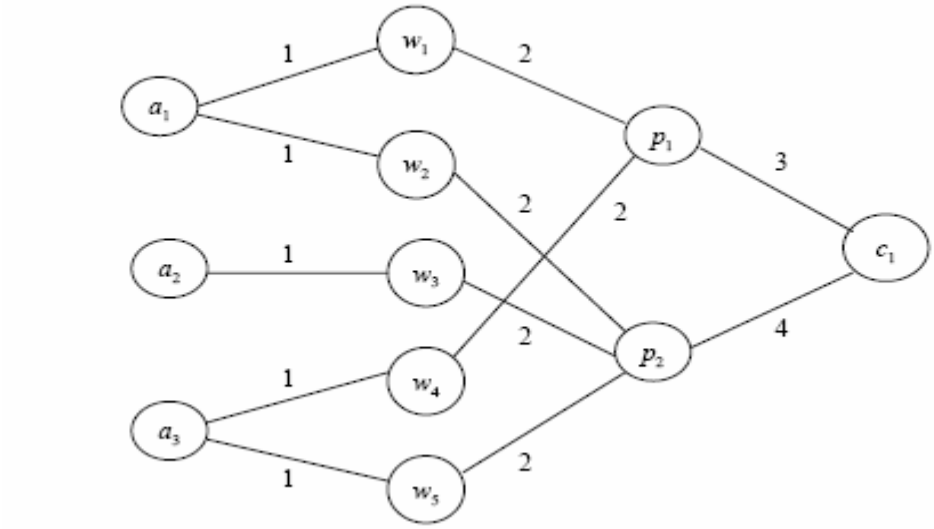

图 2-4 带标号的文献数据库的数据图

2.2.2 元组连接图和元组集连接图

元组连接图可以表示为一个二元组 TN={T，E}。其中 T 是关系数据库中所有元组的集合 $\{t_1,t_{2,}.....,t_m\}$；E 是关系数据库中元组间引用关系的集合 $\{<t_{i1},t_{j1}>,<t_{i2},t_{j2}>,.....,<t_{ip},t_{jp}>\}$，$<t_{ip},t_{jp}>$表示元组$t_{ip}$引用了另一个元组$t_{jp}$。图 2-5 展示了关系数据库 DBLP 的元组连接图的一个片段。其中图中的节点代表数据库中的元组，边代表元组间的“主键→外键”关系。节点标注中的字母部分代表该节点所代表的元组所属的关系，数字代表关系中的元组序号。图中的元组连接图片段所代表的语义是，作者 author1 和作者 auhtor2 合著了一篇论文 paper1，并且该篇论文引用了另外一篇论文 paper2。

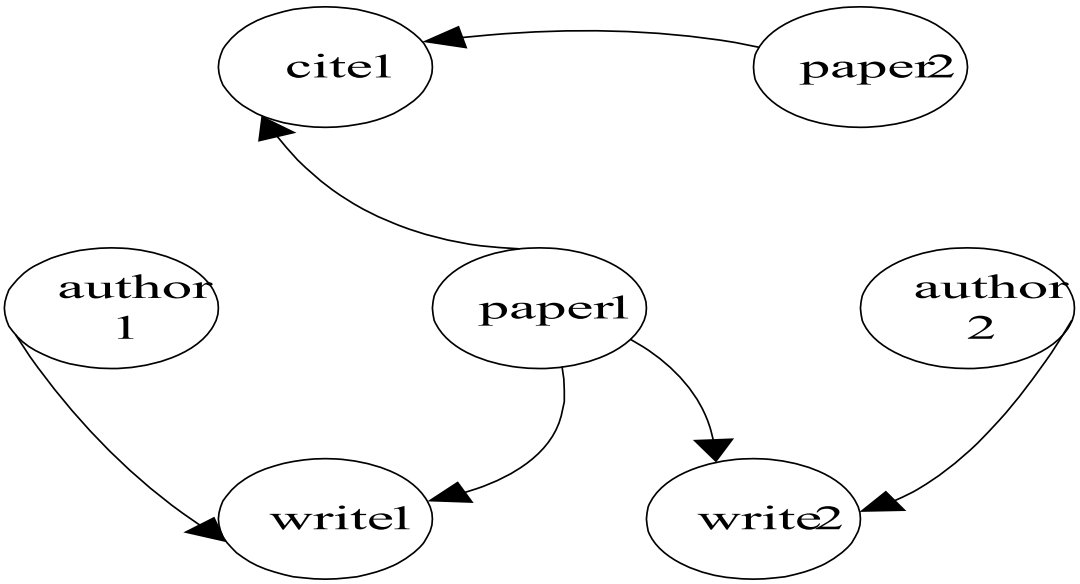

图 2-5 DBLP 数据库元组连接图片段

对于一个关键词集合 K 而言，关系数据库的元组集连接图 TSN 可以表示为一个二元组{T，E}。其中 T 是关系数据库中所有关系 R_i 相对于 K 的独立幂集中的非零独立元组集的集合。E 是 T 中元组集之间引用关系的集合。

对于关系数据库 $K = \{"smith","jim"\}$ 而言，关系数据的元组集连接树如图 2-6 所示。图中有 9 个节点，分别代表着关系数据库 DBLP 相对于关键词集合 K={“smith”，“jim”}的 9 个独立元组集。关系 author 相对于关键词集合 K 有 4 个元组集，关系 paper 相对于关键词集合 K 有 3 个元组集，关系 cites 和 writes 相对于关键词集合 K 则各有 1 个元组集。对于任意的两个元组集来说，如果两个元组集所属的关系之间存在引用关系，则两个元组集也存在引用关系[12]。

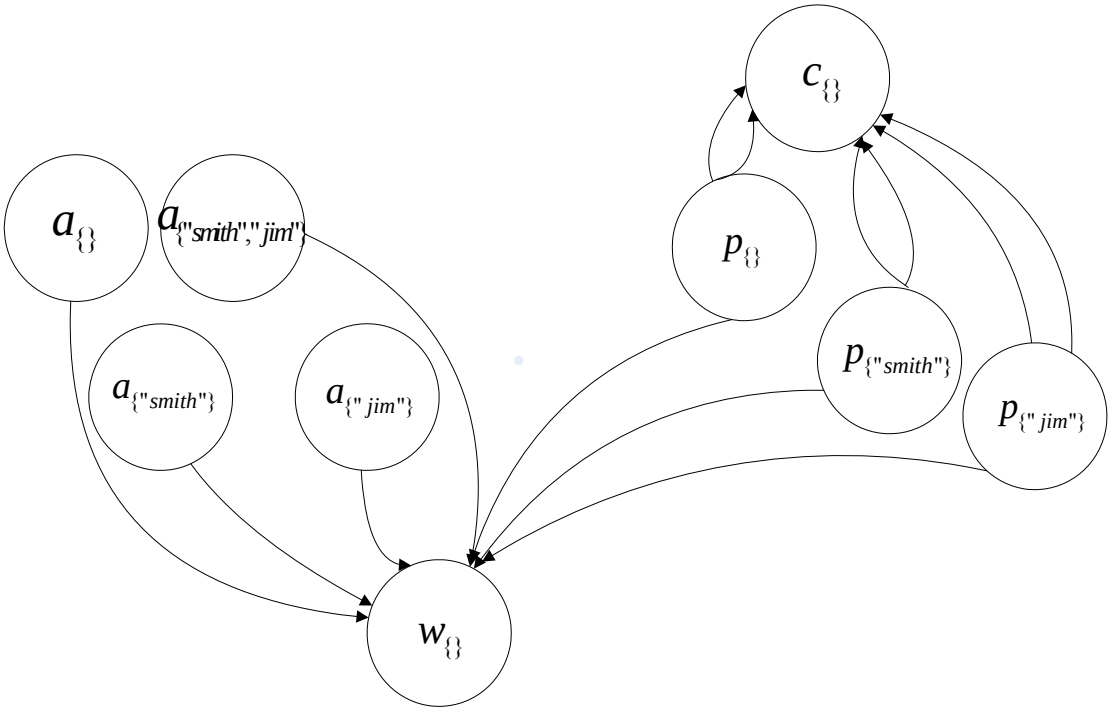

图 2-6 DBLP 数据库关于关键词查询{“smith”,”jim”}的元组集图

2.2.3 关键字查询

不同于以往的 SQL 语言，关系数据库关键词检索系统使用关键字检索方法来对关系数据库进行查询。因为传统的 SQL 语言尽管准确，但在使用过程中容易出现错误。例如查询具有**“**BANKS”信息的论文，那么用户可以输入 SQL 语句“select * from paper where title like ‘%BANKS%’”。但是如果是查找作者“smith”或“jim”所写的有关 BANKS 的论文，便需要输入 SQL 语句“select * from paper where title like ‘%BANKS%’and pid in select pid from writes where aid in select aid from author where authorid like‘%smith%’ or authorid like ‘%jim%’”。很明显随着查询情况的逐渐复杂，SQL 语句也变的越来越繁琐。对于用户来说，也变的容易出错。而使用数据库关键字检索系统自始至综只需要输入关键字**“**BANKS”即可[13]。

因此关系数据库关键字检索系统即使用户的查询不易出错，又使用户的操作非常简洁方便。关系数据库关键词检索系统的关键词查询，是一个由若干个关键词所组成的关键词集合。一个含有 n 个关键词的关键词查询可以形式化地表示为 $K=\{k_1,k_2,.....k_i,.....,k_n\}$。这是关键字查询的基本定义。

2.2.4 查询结果

查询结果的定义 ：一个关系模式图为 RS={R，E}的关系数据库关于关键词

查询 $K=\{k_1,k_2,.....k_i,.....,k_n\}$ 的查询结果，是关系数据库关于关键词查询 K 的最小元组连接树的列表。其中列表中的最小元组连接树按照其规模的大小升序排列。

而当我们使用关系数据库的元组连接图进行遍历时，我们会发现生成的元组连接树中，有的连接树会含有许多冗余信息。这些信息不但会占用系统的资源，而且会降低我们查询时的效率。因为我们查找我们需要的信息时，要小心的去分辨区分，不利于我们的查找。所以在关系数据库检索系统返回用户所查询的信息结果之前，我们最好去处掉这些冗余信息。

2.2.5 候选元组连接树

关系数据库关键词检索系统的查询结果是关系数据库关于关键词查询的最小元组连接树的有序列表。因此关系数据库关键词检索系统的查询过程，就是找出关系数据库关于关键词查询的所有最小元组连接树，并按照规模升序排列的过程。

一个关系数据库关于某个关键词查询 $K=\{k_1,k_2,.....k_i,.....,k_n\}$ 的元组集连接树 $TST=\{T',E'\}$ 是这个关系数据库关于这个关键词查询 K 的元组集连接图 $TSN=\{T.E\}$ 的子树。其中，$T'\subseteq T,E'\subseteq E$。而关系数据库关于关键词查询 $K=\{k_1,k_2,.....k_i,.....,k_n\}$ 的所有元组集连接树可以通过从元组集连接图 $TSN=\{T.E\}$ 的各个节点开始进行遍历的方法得到。

一个关系数据库关于某个关键词查询 $K=\{k_1,k_2,.....k_i,.....,k_n\}$ 的候选元组集连接树是关系数据库相对于 K 的元组集连接树。该元组连接树同时需满足以下两个条件：

1， 组成该树的节点所对应的关键词集合的并集为 K；

2， 如果去掉候选元组连接树中任意一个节点及与它相连接的边，则元组连接树不能满足条件 1 或者其他剩余的节点无法连接起来。

2.3 主要算法

这里的主要算法是候选元组集连接树产生算法和 dijkstra 算法。

2.3.1 候选元组集连接树产生算法

候选元组集连接树，是可能产生最小元组连接树的元组集连接模式。候选元组集连接树的产生，就是在元组集连接图上找出可能与用户关键词相关的最小连

接子树的过程。关系数据库关键词检索系统的候选元组集连接树的产生分为三个步骤：首先从用户关键词集合中随机取出一个关键词，并在元组集连接图中进行标记；然后以这些被标记的节点为起点，按照引用联系向周围进行扩展；最后对扩展形成的元组集连接树进行检查，如果其满足候选元组集连接树的要求，则输出。

2.3.2 dijkstra 算法

dijkstra 算法是很有代表性的最短路算法，其采用的是贪心法的算法策略。大概过程：创建两个表，OPEN, CLOSE。OPEN 表保存所有已生成而未考察的节点，CLOSED 表中记录已访问过的节点。算法过程：

1，访问路网中距离起始点最近且没有被检查过的点，把这个点放入 OPEN 组中等待检查。

2，从 OPEN 表中找出距起始点最近的点，找出这个点的所有子节点，把这个点放到 CLOSE 表中。

3，遍历考察这个点的子节点。求出这些子节点距起始点的距离值，放子节点到 OPEN 表中。

4，重复第 2 和第 3 步,直到 OPEN 表为空，或找到目标点。

第三章　系统体系结构和工作流程

3.1 体系结构

整个体系结构主要包括预处理模块和查询处理模块，以下是其详细介绍。

3.1.1 概述

关系数据库关键词检索系统总共有两部分构成：预处理组件和查询处理组件。整个系统的工作流程如图 3-1 所示。其中关系数据库中数据信息和模式信息作为输入，发送到预处理模块。然后预处理模块生成全文索引和其它的模式信息。它们和用户所写的关键字集合一起做为查询处理模块的输入，然后经过查询处理模块通过生成元组集、构建元组集图、进行遍历以生成候选元组集连接树和通过计算各个候选元组集连接树以生成最终的查询结果，最后将查询结果通过系统界面返回给用户。

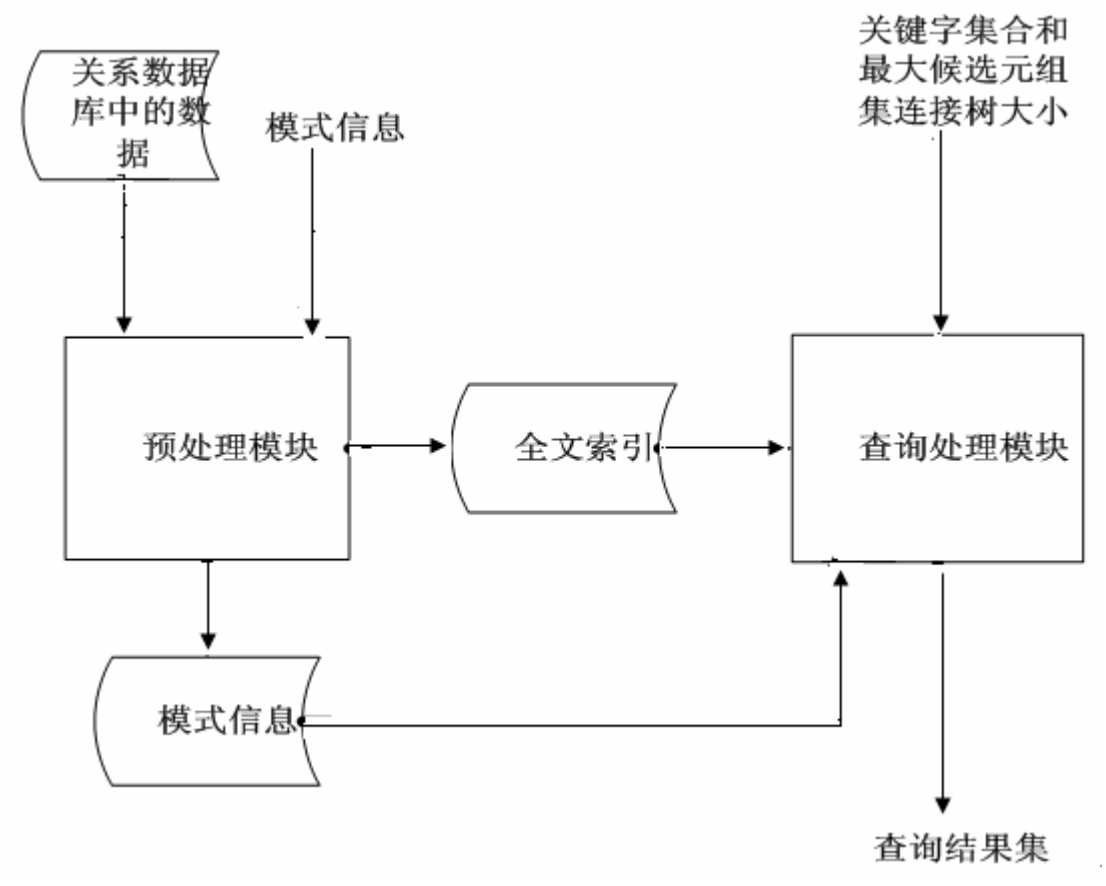

图 3-1 关系数据库关键字检索系统体系架构图

3.1.2 预处理过程

预处理模块首先对用户输入的模式信息进行正确性验证，然后将待发布的数

据库的模式信息存入已发布的关系数据库列表。然后根据待发布数据库的模式信息对数据库进行访问，并为关系数据库库中需要发布的文本类型的属性列生成所需要的全文索引。

其中模式信息包括待发布关系数据库的名称、关系数据库中的关系列表、各个关系之间的引用关系、各个关系的属性列名称和类型，以及是否需要全文索引等。预处理模块将这些信息存入一个系统数据库中保存，以备预处理后续阶段及查询处理过程中使用。后续的过程则分为三个步骤：首先预处理模块将为待发布的数据库在硬盘上建立一个全文目录。然后将根据之前输入的模式信息依次对各个关系进行访问，并遍历关系的每个属性。如果该属性列需要进行全文检索，则为其生成全文索引。最后待关系数据库中的所有关系均被访问以后，进行全文索引填充。

3.1.3 查询处理过程

查询处理过程主要是对用户的关键词查询进行处理，按照启发式的方法查找可能与用户输入的关键词集合相关的关系数据，并将之返回给用户。

在普通的文档查询中，查询系统一般只需通过查找倒排索引，并进行一系列的并、交或差等集合运算即可完成查询过程。但是关系数据库的查询过程则要复杂的多。这是因为关系数据库在设计之初，为了防止数据的冗余性，要进行规范化。规范化的结果，就是一个信息单元被分散存储在不同的关系表中。这就使得对于关系数据库的关键词检索，不但需要找出可能包含关键词集合的元组，还要探索各个元组集之间的关系。例如，对于关键词查询 K={“database”,“system”}而言，与其相关的元组不止包括在标题中包含有“database”和“system”的论文，还可能包括标题中含有“database”的论文引用了标题中含有“system”的论文或者标题中含有“database”的论文被标题中含有“system”的论文所引用等情况。在这个关系数据库关键字检索系统的实现中，是通过生成元组集连接图，并在连接图上进行遍历的方法来探索各个元组之间可能存在的关系的。

查询处理过程主要步骤是：

1，计算基础元组集。

2，计算独立元组集的集合。

3，构建元组集连接图。

4，生成候选元组集连接树集合。

5，对候选元组集连接树进行计算。

6，查询结果的显示。

3.2 工作流程

工作流程主要是预处理过程与查询处理过程两部分。

3.2.1 预处理过程

预处理过程主要是将待发布的关系数据库的模式信息保存下来，并为数据库中的各个元组的文本类型属性列生成全文索引。全文索引是一种基于关键词的倒排索引，是对数据库中文本型属性列进行分词、去停用词之后生成的。待发布数据库的模式信息，在发布时由用户手动输入并保存在系统数据库中。主要体现在以下方面：

1，正确性验证

这一部分主要是对数据库中输入的信息进行正确性验证，同时如果在这过程中出现错误的信息，则要对这信息进行提示。从而保证数据库中存取信息的正确性。

2，模式信息保存

这一部分主要是进行模式信息的保存。主要是将关系数据库中的模式信息保存在数据库中。模式信息主要是数据库中表的列值、主键信息、外键信息以及其它。

3，全文索引创建过程

（1）启动数据库的全文处理功能。

（2）为数据库 DBLP 建立一个全文索引目录。

（3）在全文目录 test 中，注册数据库 DBLP 中的 paper 表。

（4）指出表中需要全文索引的列名。

（5）为 paper 表创建全文索引。

（6）将表 paper 的 paperid 列和 title 列的倒排索引填充到全文目录 test 中。

4，预处理过程的示意图

关系数据库关键字检索系统的预处理过程分为三个子模块：正确性验证、模式信息保存和全文索引的建立。其中正确性验证模块，负责检查用户输入模式信息的正确性并对错误信息进行提示；模式信息保存模块，负责将通过正确性验证的待发布数据库的模式信息保存在系统数据库中；全文索引建立模块，负责用户数据库发布情况。其过程如图 3-2 所示。

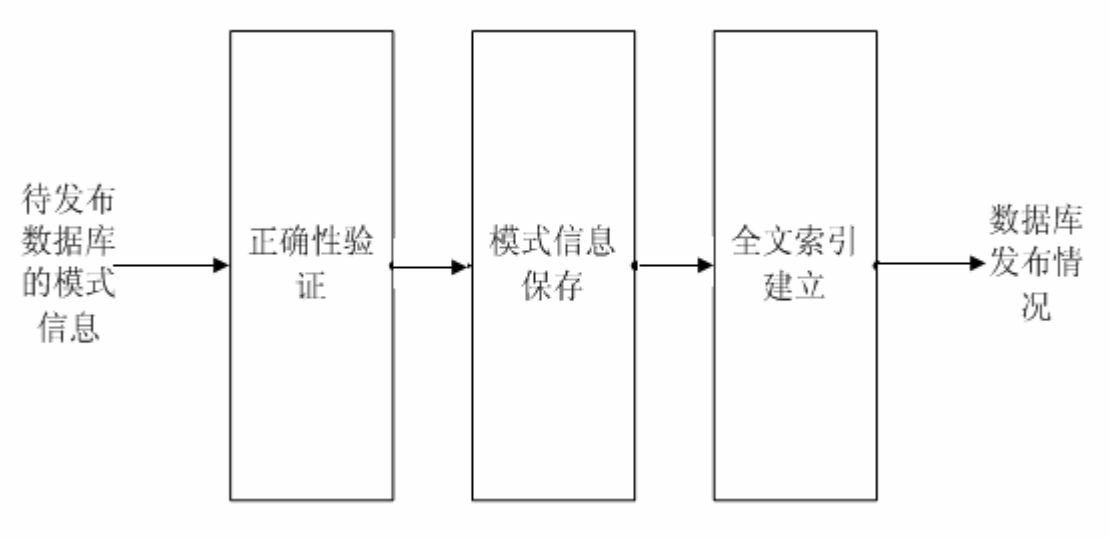

图 3-2 预处理过程

3.2.2 查询处理过程

关系数据库关键词检索系统的查询处理模块，以关键词集合 K 作为输入，输出与 K 相关的最小元组连接树列表。查询处理的过程分为三个步骤：独立元组集计算、候选元组集连接树的枚举和最小元组连接树计算。独立元组集的是通过查询全文索引，并进行集合的并、交和差计算得到的。然后，系统以非空的独立元组集作为节点，以元组集之间的引用关系为边生成元组集连接图，并通过候选元组集连接树的枚举算法生成候选元组集连接树。最后，将得到的候选元组集连接树转换成 SQL 语句，交给关系数据库管理系统执行生成最小元组连接树。整个查询处理过程如图 3-3 所示。

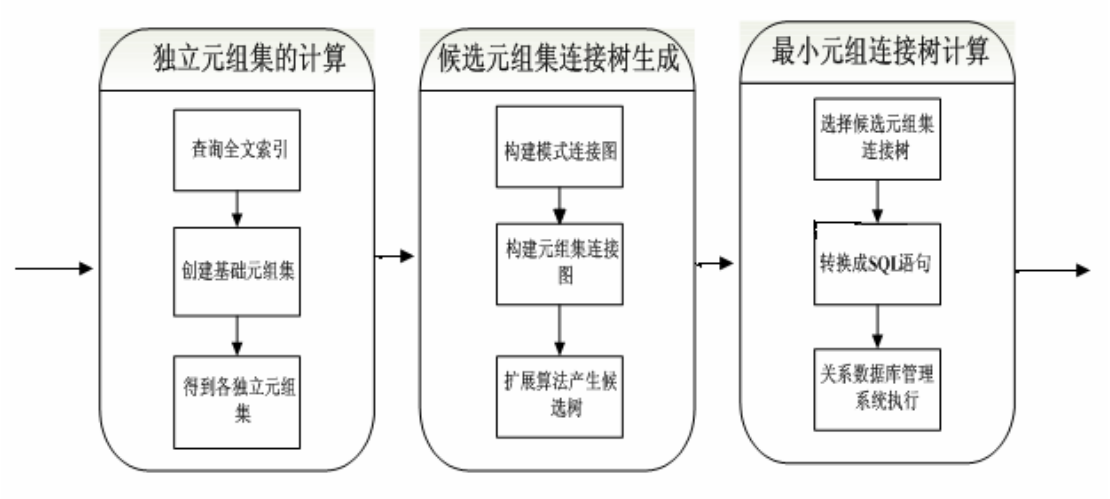

图 3-3 查询处理过程

主要体现为以下三方面：

1，创建基础元组集

一个关系 R 相对于一个关键词k_i的基础元组集是由 R 中包含关键词k_i的所有元组所组成的集合。对于一个由 n 个关键词所组成关键词查询$K=\{k_1,k_2,.....,k_i,.....k_n\}$而言，关系 R 相对于 K 中各个关键词的基础元组集的集合称之为 R 相对于 K 的基础元组集。可以形式化地表示为$\{R_{k_1},R_{k_2},.....,R_{k_n}\}$。一个关系数据库中所有关系$R_1,R_2,.....,R_m$相对于 K 的基础元组集的并集称之为数据库相对于 K 的基础元组集。

一个关系 R 相对于一个关键词查询$K=\{k_1,k_2,.....,k_i,.....k_n\}$的独立元组集，是关系 R 相对于 K 的幂集中的所有元素的独立元组集的集合。对于含有关键词个数为 n 的关键词查询 K 而言，关系 R 相对于 K 的独立元组集共包含2^n个元素。其中 R 相对于空集的独立元组集为自由元组集$R_{\{\}}$，它是 R 中所有元组所组成的集合。R 相对于 K 中所有元素的独立元组集称为充足元组集，它是 R 中包含 K 中所有关键词的元组所组成的集合。同时，它也是一棵大小为 0 的候选元组集连接树，其中的元组构成了与关键词查询 K 相关的最简单的查询结果。

独立元组集的计算可以分为两个步骤：

（1） 求得关键词查询$K=\{k_1,k_2,.....,k_i,.....k_n\}$的所有幂集。

（2） 求得 R 相对于幂集中的每一个元素的独立元组集。

2，生成候选元组集连接树

利用关系数据库的关系模式图创建其相对于 K 的元组集连接图的流程如下：

首先，依次读取关系数据库中关系列表$R_1,R_2,.....,R_n$中的关系 R 相对于关键词查询 K 的独立元组集的列表$\{R_{K1},R_{K2},R_{K3},.....,R_{K2^n}\}$，$Ki\in K$的幂集，$0<i<2^n+1$。对于独立元组集列表中的每一个独立元组集，如果它所包含的元组的个数不为零，则为其创建一个新的节点 N。然后，遍历图中现有的节点，如果发现与新节点存在引用关系的节点 N’，则将 N’填入新节点的邻接表中并且将 N 填入到 N’的邻接表的末尾。

而候选元组集连接树的概念关系则是数据库相对于关键词查询K的候选元组集连接树，是数据库相对于K的元组集连接图的子树。该子树 T 具有如下特征：

(1) T 的所有叶子节点所包含的关键词集合等于 K.。

(2) T 不存在冗余节点，即对于 T 中的任意一个节点 N，如果从 T 中删除 N. 则 T 或者不能包含所有的关键词，或者将不能将剩余的节点连接起来。

3，生成最小元组连接树

关系数据库关于用户关键词查询 $K=\{k_1,k_2,.....k_i,.....,k_n\}$ 的最小元组连接树是关系数据库的一棵包含关键词查询中的所有关键词并不包含冗余节点的元组连接树。

这里的冗余节点包含两层含义。一层是指最小元组连接树中不包含相同的元组；另一层的含义是，如果删除最小元组连接树中的任意一个节点 N，则元组连接树不能包含所有的查询关键词或者无法将剩余节点连接起来。

候选元组集连接树代表了具有相同结构的最小元组连接树的模式信息。对候选元组集连接树进行计算的结果，就是具有相应结构的最小元组连接树。而计算候选元组集连接树的过程，是将候选元组集连接树转换成相应的 SQL 语句，并交付关系数据库管理系统执行的过程。

第四章　系统实现

系统实现主要包括关系数据库关键字检索系统的技术路线、程序运行流程、程序具体的系统实现过程以及程序的运行结果.。

4.1 技术路线

参照传统的 web 搜索引擎模式，关系数据库关键字检索系统采用 B/S 架构实现。用户通过浏览器输入关键字，发出查询请求，服务程序接受关键字，运行查询算法，向用户返回排序的查询结果。服务器端处理程序是整个系统的核心，它需要连接数据库、获取数据库的数据图、运行核心查询算法并返回查询结果。作为一个关系数据库关键字检索系统的设计，本系统并没有做到可以针对任意的数据库进行运行(当前只对 ORACLE 可以运行)，但是把数据库连接操作作为一个接口显然是可取的方法，因此需要设置一个连接池。

选择使用.NET 和 ASP.NET、C#作为开发平台和语言，visual stdio 2008 作为开发环境。数据集的准备可以选择合适的方法去实现，因为这与系统实现没有紧密的联系，只是存在一个数据库管理系统的选择问题，原本计划选择 SQLSERVER 作为数据库管理系统，但是由于实现过程中发现 SQLSERVER 不能让用户获取数据记录的 ROWID，导致系统的实现出现阻碍，因而转向使用 ORACLE，它可以方便的获取数据记录的 ROWID，作为一个原型系统的实现，这样做是比较省力和可行的。

C#语言本身是一个面向对象的程序设计语言，但是由于本系统的实现采用的是面向过程的思想，所以大部分情况下选择使用公共类和静态方法，这样做也避免了在创建对象时占用更多的内存，在整个实现过程都需要考虑尽可能减少内存的使用，甚至有些用时间换空间的策略也是可取的，因为通常面对的都是大数据量的作业。

4.2 程序运行流程

关系数据库关键字检索系统的设计并没有采用面对对象设计的方法，因为相对于确定功能封装良好的类，按照系统的流程去设计和编写程序显得更加简便和实用。

经过分析确定程序的运行流程如下：

1，准备工作。准备测试数据集，选取了 DBLP 数据集作为程序测试数据集，但是从网上下载得到的是 XML 文档，因此需要将其范式化存入数据库中。

2，获取数据库模式信息，包含数据库中所有用户表的表名、字段和主外键信息(外键揭示了表间的引用关系)。

3，生成数据图(生成以元组为节点，元组间的外键关系为有向边的数据图，同时需要统计节点的入度和边的权重信息)。

4，获取用户请求，统计每个查询关键字对应的元组集。

5，运行核心搜索算法，迭代生成若干搜索结果有根树。

6，按序输出搜索结果树。

系统的流程图如图 4-1 所示。

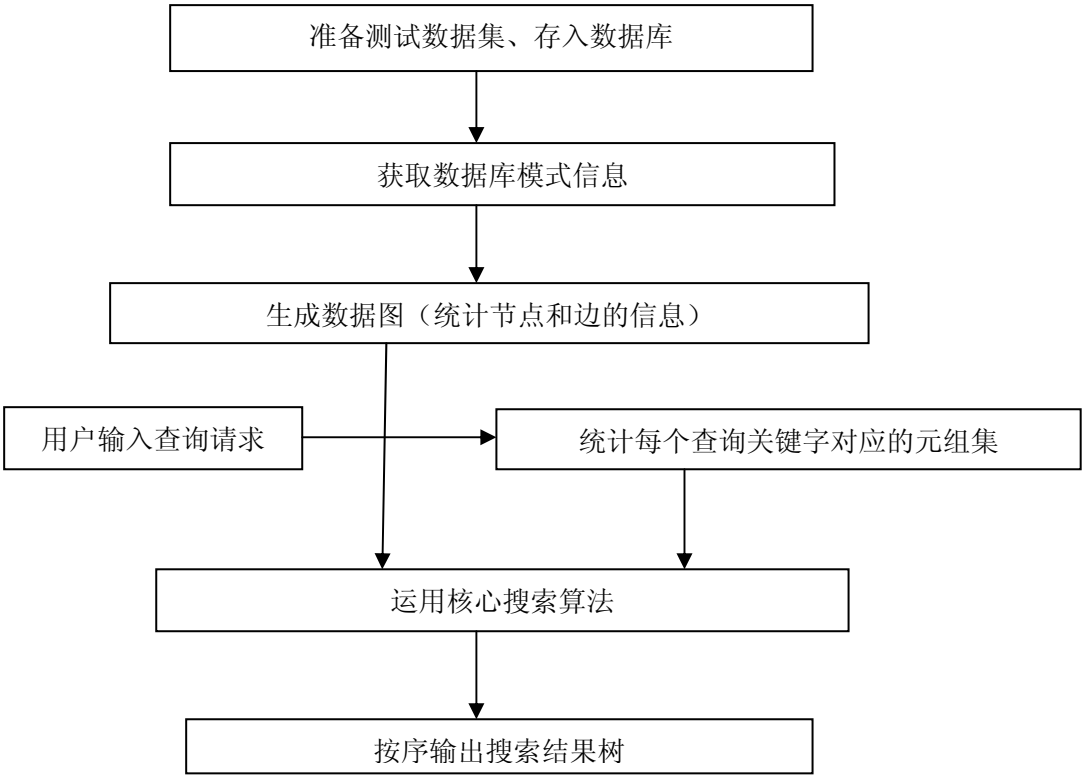

图 4-1 程序运行流程图

4.3 系统实现

关系数据库关键字检索系统的具体系统实现过程主要包括测试数据集的准备、数据库模式的获取、数据图的生成、查询关键字元组集的获取以及搜索算法。

4.3.1 测试数据集的准备

数据集的准备是系统实现的前期准备工作，如果已经存在测试数据库则可以省略这一步，它与整个系统的实现关系不大。

我选取的是 DBLP，它是一个计算机科学参考数目的网站，最初是一个数据库和逻辑编程(DataBase and Logic Programming)书目的网站，现在已经发展到包含几乎所有重要的会议和期刊的论文、专著、学位论文和 WWW 上的相关文章。从 DBLP 网站可以下载到 DBLP 数据集，它是一个 XML 文档，大约将近 500M(484M)。由 DBLP 数据集生成测试数据库主要的工作在于解析 XML 文档，对其中的信息进行合理的提取和划分，并存入到对应的数据库表中。我是用 JAVA

语言编写运用 SAX 解析器来进行 XML 文档的解析的，它的优点在于非常类似于一个流模型，只是在读取数据时检查数据，不需要一次性将整个 XML 文档导入内存中处理，这对于大型文档来说是个巨大的优点。SAX 解析器采用了基于事件的模型，它在解析 XML 文档的时候可以触发一系列的事件，当发现给定的 tag 的时候，它可以激活一个回调方法，告诉该方法制定的标签已经找到。它的缺点在于很难同时访问同一个文档中的多处不同数据，内存中只存在当前访问的数据[14]。

下面来看一下 DBLP 的 DTD 文件：

<!ELEMENT dblp (article—inproceedings—proceedings—book
—incollection—phdthesis—mastersthesis—www)*>

针对 DBLP XML 文档的特点，可以在遍历 XML 时使用栈结构保存每一个完整的参考书目的信息，然后再分析栈中的数据，提取需要的信息存储到相应的数据库关系表中。最后得到了四个关系表，分别为 authors、papers、writes、cites，同时关系之间存在引用关系。至此，测试数据集准备完毕。

在此需要指出的一个问题是，测试数据库是用来测试最终实现的系统的性能的。同时在系统的编码实现过程中需要在一个示例数据库上进行调试，测试数据库可以用来作为调试数据库，但是我选择使用一个数据量很小的 mybookbiz 数据库作为调试数据库，因为它较小，所以每一次调试运行花费的时间相对 DBLP 测试数据库来说要少很多，同时不会改变系统实现的原理，这样做是比较可取的。下图是 mybookbiz 数据库的一个模式图，如图 4-2 所示：

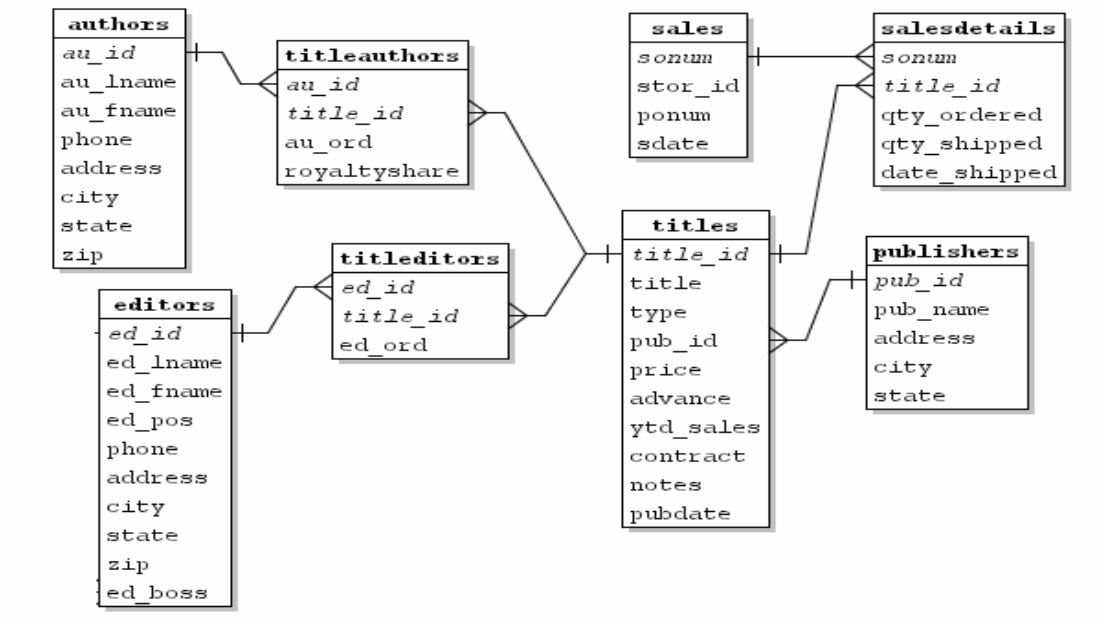

图 4-2 mybookbiz 数据库模式图

4.3.2 数据库模式的获取

系统对于数据库管理系统的唯一要求只是访问的权限，即用户名和密码，数据库不提供数据库模式或其他附加信息。所以程序的第一步就必须去获取数据库的模式，然后才能进行后续的工作。如何针对 SQL SERVER 数据库获取其所有用户表、各个表的列名和类型、以及主外键信息，以及建立合适的数据结构去存储这些模式信息是这一步的主要工作。程序中建立的数据结构如下：

DBTable 类
table_name:string //关系表名
table _columns:ArrayList //关系表的列字段链表
primarykeys:ArrayList //关系表的主键链表
foreignkeys:ArrayList //关系表的外键链表
TableColumn 类
table _name:string //字段所在的关系表名
column_name:string //字名
column_type:string //字段类型
PrimaryKey 类
table_name:string //主键所在的表名
columnIndex:int //主键在该表中的字段序号
key_column:string //主键字段名
ForeignKey 类
key_column:string //外键字段名
columnIndex:int //外键在该表中的字段序号
depend_table:string //依赖的关系表的表名
depend_column:string //依赖的关系表的字段名

再定义一个 DBSchema 类，其中定义一个静态成员 dbtables:ArrayList 用来存储获取的关系表，定义一个静态方法 GetTables()来获取数据库的表、字段、主键、外键信息，并将创建的 DBTable 对象逐个加入到 dbtables 中。GetTables()方法的实现过程如下：首先通过"select table _name from user _tables "获取所有的用户表，然后根据获得的每个用户表名再去查找其对应的字段信息和主外键信息。

4.3.3 数据图的生成

1，元组的获取

生成数据图，第一步就是获取数据库的所有元组信息，考虑到 ORACLE 数据管理系统对每个数据记录有唯一的 ROWID(由对象 ID、文件 ID、块 ID、行 ID

组成)，所以只需要在内存中存储元组的 ROWID 即可，这样也使得将所有元组信息放入内存中成为可能。定义元组类的数据结构如下：

```
Tuple 类
RID:string   //此元组的 ROWID
refust:ArrayList   //此元组指向的元组 ROWID 列表
backlist:ArrayList   //此元组在后面的后扩展搜索算法
                     //(BESearch)中获得的反向链表
sList:ArrayList   //此元组包含的关键字列表
```

定义类 DBTuples，其中的成员变量如下：

```
DBTuples 类
tuples:ArrayList   //存储所有数据库元组的链表
rid2indexHashtable   //元组的唯一标识 RID 与其
                     //所在链表中的索引之间的哈希表
relations:ArrayList   //各个关系的统计数据链表
```

其中的哈希表 rid2index 存储从元组的 ROWID 到其在 tuples 链表中的下标的映射，可以方便在以后的程序通过 ROWID 去获取元组更多的信息，relations 链表的元素是 Relation 对象，它包含一个关系表的表名以及该关系表中的元组在 tuples 链表中起始和终止下标，以后可以用来在已知元组下标的情况下确定其所属的关系表[15]。

另外定义静态方法 GetTuples()用来获取元组信息。其执行过程如下：

```
GetTuples()方法
for(int i=0;i<dbtables.Count;i++)//对于数据库中的每一个关系
{
    初始化一个 Relation 对象，统计该关系的起始元组下标信息;
    判断该关系是否有引用关系;
    If(hasFK==false) //如果没有引用关系
    {
        获取该关系的所有元组的 ROWID;
        对于该关系的所有元组 ROWID;
        创建 Tuple 对象，加入到 tuples 中;
        更新 rid2index;
        更新 Relation 对象，将其加入 relations 链表中;
    }
    else
    {
```

获取该关系的所有元组的 ROWID 和所有外键 FKs;
对于该关系中的每一个元组创建 Tuple 对象;
对元组的每一个外键根据外键字段和值，在引用表中;
查找到引用元组，将引用元组的 ROWID 加入到 refList 当中;
将 Tuples 对象加入到 tuples 中;
更新 rid2index;
更新 Relation 对象，将其加入 relations 链表中;
}
}

2，数据图的生成

常见的图的存储方式有邻接表和邻接矩阵，而且核心搜索算法中会使用到最短路径算法，因此对于小数据集采用邻接矩阵的方式较好。但是在前面也曾提到过，关系数据库关键字检索系统面对的经常是一些大数据集作业，内存往往成为瓶颈，所以需要尽可能减少内存的消耗，因此选择邻接表的方式来存储数据图。前面获取元组的时候已经存储了元组对象链表，而且每个对象都包含了其引用的对象链表，所以 tuples 本身就已经是邻接表了，在这个邻接表中包含了所有的节点信息和节点之间的有向边关系，唯一缺少的是边的权重信息。关于边的权重信息的计算，涉及到关系数据库关键字检索系统中对数据库的建模思想。

关系数据库关键字检索系统中使用到的概念包括节点、节点权重、边(分为前向边和反向边)、边的权重以及关系之间的相似度的度量。数据库中的每一个元组都对应数据图中的一个节点，节点的权重正比于节点的入度，当元组 T1 中有一个外关键字引用元组 T2 时，存在一条从 u(代表元组 T1)指向 v(代表元组 T2)的有向边(u,v)，同时会产生一条从 v 到 u 的反向边，且其权重正比 u 所在的关系中指向节点 v 的节点数。

关系 R1、R2 之间的相似度 s(R1,R2)取决于从 R1 到 R2 的连接类型，如果从 R1 到 R2 没有连接，则 s(R1,R2)为无穷大。

边(u,v)的权重 w(u,v)的计算分为以下四种情况(假设 uv 代表存在从 u 到 v 的有向边、vu 代表存在从 v 到 u 的有向边，注意此处的边不包括反向边);

(1)uv==FALSE & vu==FALSE~w(u,v)=MAXVALUE

(2)uv==TRUE & vu==FALSE~w(u,v)=s(R(u),R(v))

(3)uv==FALSE& vu==TRUE~w(u,v)= IN_v (u)s(R(v),R(u))

(4)uv==TRUE& vu==TRUE~w(u,v)=min{s(R(u),R(v)), IN_v (u)s(R(v),R(u)) }

为了生成数据图，定义类 DBDataGraph，其中的成员变量包括如下：

DBDataGraph 类

relationMatrix:short[,]~//关系表之间的相似度矩阵

nodesArrayList~//节点的入度统计信息链表

DBDataGraph 类中成员函数的计算流程如下：

(1)计算关系之间的相似度函数 computeRelationMatrix()。关系之间的相似度取决于它们之间的依赖关系，默认情况下当从关系 R1 到 R2 存在引用关系时，s(R1,R2)值为 1.根据前面获得的模式信息，很容易通过遍历 dbtables 链表来得到任意 s(Ri,Rj)的取值。考虑到关系表的数目相对不会太大，所以采用矩阵的形式来存储关系表之间的相似度值。

(2)统计节点的入度信息函数 computeIndegrees()。节点的入度信息决定节点的权重，是最终结果树权重的一部分。在实现中另外使用 nodes 链表来存储节点的入度信息，nodes 链表和 tuples 链表中相同的下标对应同一个节点，不需要使用额外的对应数据结构。每个节点的入度信息包括入度总值和一个入度对象链表，每个入度对象包括关系所在的索引和入度值。computeIndegrees()函数使用一次遍历 tuples 链表的方法来统计所有节点的入度信息，具体做法是：对于遍历到的每一个元组，获取其所在的关系下标，再通过其引用元组链表获取其所有的引用元组，针对每一个引用元组更新它们因此元组带来的入度信息变化，这样遍历一遍之后，所有的节点入度信息统计完毕。

(3)计算节点之间的距离(即对应的有向边的权重)的函数 getDistance()。计算节点的距离可以通过一次遍历节点计算完毕，但是问题是需要将这些距离信息存入矩阵保存在内存中，前面已经提到过这样做对往往会因为内存的限制而出现问题。因此程序中对于节点之间距离的计算采用了以时间换空间的策略，不事先通过计算存储所有的节点之间的距离信息，而是给出计算任意两个节点之间的距离的函数，在需要的时候去调用函数来即时计算。计算函数的流程下：

```
getDistance(int i,int j)
//根据节点的下标来计算节点之间距离的函数
{
    if(i==j) return distance = 0;
    判断两个节点之间的有向边指向关系;
    根据节点的有向边指向关系，结合边的
    权重计算公式和已经得到的关系之间的
    相似度数据和节点的入度统计数据计算
    节点之间的距离 distance;
    return distance;
}
```

4.3.4 查询关键字元组集的获取

根据最后的搜索算法的需求，需要获取每个查询关键字对应的元组集。为了

获取这些查询关键字元组集，可以通过针对每一个关键字遍历一次所有的元组，找出所有包含此关键的元组，这样做的缺点是增加了遍历的次数。程序中采用的是一次遍历的方法，对每一个元组分析其包含的查询关键字信息，并更新对应的元组集信息。

另一个重要的问题是关键字的匹配问题，即如何判断在元组内容中是否存在关键字对象，程序中采用的较为简单的方法：将元组内容按某些特定符号划分为字符串数组，然后遍历此字符串数组，查看其中是否包含查询关键字。需要注意的是，若一个元组中包含多个查询关键字，则它会被加入到这多个关键字对应的不同的元组集中。

4.3.5 搜索算法

在介绍搜索算法之前需要先说明下其中用到的三个算法，分别是单源最短路径的 dijkstra 算法、堆排序算法、叉积算法。

dijkstra 算法：dijkstra 算法在计算图中的单源最短路径时比较常用，算法的原理在此不作赘述。需要指出的是因为程序中使用邻接表而不是邻接矩阵来存储图，所以在获取边的距离时与传统的直接读取矩阵元组值有区别，在程序的实现中需要去调用计算节点之间的距离(即对应的有向边的权重)的函数 getDistance() 来即时获取，算法的思想并未改变，但是增加了调用函数计算的开销，这也是为了节约内存的考虑。

堆排序算法：堆排序的思想也不再赘述。在程序中有小顶堆和大顶堆的排序类，类中都有两种方法，一是 HeapSort()用来对一组元素进行排序,一是 OutputTop() 用来输出堆顶元素，同时保存剩余的元素依然是一个堆。另外排序的元素对象 Element 包括索引和值两项，索引用来表示该元素在原始的数组或链表中的位置，值表示该元素的关键字值，也是排序的依据。

叉积算法：在程序中需要计算若干个(个数未知)数组或链表的叉积，我采用了分治算法来解决这个问题，即每次计算两个数组之间的叉积，具体的做法如下：在计算若干个数组的连乘叉积时，可以通过迭代使用此函数得到最终的叉积结果。

下面介绍程序中的核心搜索算法，称为反向扩展算法。算法的实现过程如下：

输入：

查询关键字点集:$(S_1,S_2,\cdots,S_n)$,$S=\cup S_i$;

IteratorHead;

OutputHeap

过程：

对于 S 中的每一个节点 n,

以 n 为源点创建一个单源最短路径迭代器，并将此迭代器加入到迭

代器堆中，迭代器堆按照迭代器输出的第一个节点的最短路径距离排成最小堆

```
While(IteratorHead.Count>0&&resultNum<NEEDED>
{
    Iterator=迭代器堆的堆顶元素;
    V=迭代器的堆顶元素;
    If(Iterator.Count>0)
    {
        将迭代器重新加入到迭代器堆中并按照迭代器输出的第一个节点的最短路径的距离重新排成最小堆。
    }
    if(v 尚未被任何迭代器访问到过)
        for i=1…n;创建 v.Li;
     CrossProduct=origin*[v.Lj](j=1…n,j！=i)
     //其中是迭代器 Iterator 的源点，任何 v.Lj 为空将使得 CrossProduct 为空
     将 origin 加入到 v.Li 中;
     对于叉积中的每一个元素
     {
         根据此元素来创建结果树;
         将结果树加入到 OutputHeap 中，排序 OutputHeap 为最大值;
     }
     if(OutputHeap.Count>N)输出堆顶结果树到输出缓冲中，在输出缓冲中再对结果按相似度进行排序
```

程序中建立 BESearch 类来实现此算法过程，类中定义如下：

```
BESearch 类
Iterators:ArrayList   //迭代器的链表
IteratorHeap:ArrayList   //最短路径迭代器堆
SearchResults:ArrayList   //查询结果链表
OutputHeap:ArrayList   //查询结果输出堆
FullCount=10:int   //输出堆的最大长度
ASPRESULT:ArrayList   //ASP 页面打印字符串链表
ASPCOUNT=20:int   //页面总共打印的结果数目
```

其中迭代器链表存储所有的 S 中的节点计算出的单源最短路径迭代器，IteratorHeap 堆中的元素则是 Element 对象，包括迭代器在 Iterators 链表中的下标

和该迭代器当前最短路径的最小值。SearchResults 存储的是产生的查询结果树链表，链表元素是 Result 对象，其中包括结果树的权重、根节点和叶子节点链表(即叉积结果)，OutputHeap 堆中的元素是 ResultRecord 对象，对象字段包括查询结果树在 SearchResults 链表中的索引和结果树的权重。在程序中进行的堆排序(迭代器堆是小顶堆，查询结果树记录堆是大顶堆)是都是通过对中间记录进行排序，然后通过对应的索引信息去获取真实的迭代器或结果树。

在实现中为了 ASP 页面的显示，所以将每一个查询结果转为了包含 HTML 标记的字符串，并将字符串存入了 ASPRESULT 链表中，这样就可以直接在页面打印出查询结果了。

4.4 程序运行结果

程序运行结果如图 4-3 所示程序运行界面：

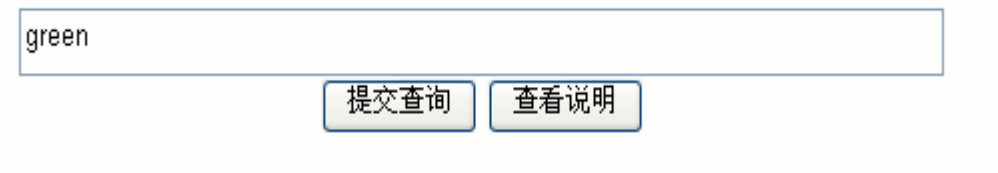

图 4-3 程序运行界面

下面是程序的查询运行结果，如图 4-4 所示：

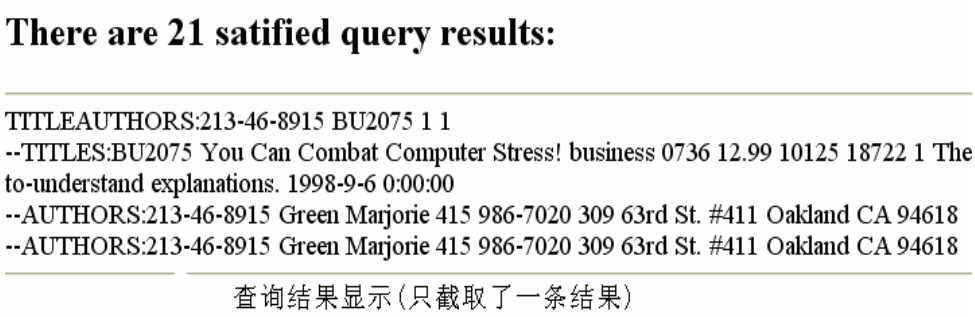

图 4-4 程序的查询运行结果

结果表明程序对关键字 green 的查询。

第五章　总结与展望

5.1 论文总结

本文设计和实现了一个关系数据库关键字检索系统。使用该系统，用户只需输入若干个关键词就能完成关系数据库的查询。系统将采用启发式的语义探索方式，返回所有可能与用户关键词相关的元组连接模式。在用户选择了与自己查询意愿一致的连接模式后，系统将属于该元组连接模式的复合元组搜索出来，并以二维表的形式向用户展示。

5.2 未来工作展望

关系数据库关键字检索系统仍存在很多的问题需要进一步的解决和完善：

一、关于图的存储问题，程序中是按照邻接表的方式来存储的，距离信息需要即时调用函数去计算，虽然这是结合内存限制考虑舍弃邻接矩阵方式的选择，但是需要进一步的考虑更好更有效的方式来存储图，例如图的压缩存储的问题。

二、并行的问题，在程序中有多处可以借助并行来提高程序的效率，例如多线程运行 dijkstra 算法等。在并行时还需要考虑数据冲突和加锁等细节问题。

三、查询结果树的同构判断和存储。当前的程序中没有进行同构树的判断，这是比较艰难的一个问题，需要以后继续的深入去做。另外在当前的实现中只存储了结果树的权重、根节点和叶子节点的信息，忽略了中间节点，这样做对权重的计算和结果的可视化展示都会有影响，也需要进一步的考虑。

四、数据库的可视化展示，在这次的实现中将这一部分未作考虑，但是此功能对于整个系统来说是很重要的，尤其可以增加用户友好性

五、系统的性能测试，这是系统实现之后的关键步骤，也往往是可以出成果或发现问题的地方，我会在接下来重点研究这一部分。

参考文献

[1]B. Aditya，Gaurav Bhalotia，Soumen Chakrabarti et al．BANKS: Browsing and Keyword Searching in Relational Databases[Z]．Bombay：Computer Science and Engg. Dept., I.I.T，2002．

[2]沃森, K. (Watson, Karli) ．C# 2005 数据库编程经典教程．北京：人民邮电出版社，2002．60－70．

[3]Gaurav Bhalotia，Arvind Hulgeri，Charuta Nakhe et al．Keyword Searching and Browsing in Databases using BANKS[Z]．Bombay：Computer Science and Engg. Dept., I.I.T，2001．

[4]Vagelis Hristidis，Yannis Papakonstantinou．DISCOVER: Keyword Search in Relational Databases[Z]．San Diego：University of California，2002．

[5]Arvind Hulgeri，Gaurav Bhalotia，Charuta Nakhe et al．Keyword Search in Databases[Z]．Bombay：Dept. of Computer Science and Engg., Indian Institute of Technology，2005．

[6]萨师煊，王珊．数据库系统概论．北京：高等教育出版社，2000．46－55．

[7]严蔚敏，吴伟民．数据结构（C 语言版). 北京：清华大学出版社，1997．30－40．

[8]刘启芬，顾韵华，吕静．数据库实用教程．北京：电子工业出版社，2009．33－55．

[9]李春葆．数据结构教程．北京：清华大学出版社，2009．25－40．

[10]左孝凌，李为鉴，刘永才．离散数学．上海：上海科学技术文献出版社, 1982．50－65．

[11]方世昌．离散数学．西安：西北电讯工程学院出版社，1985．60－70．

[12]赵致格．数据库系统与应用 SQL Server．北京：清华大学出版社，2005．40－120．

[13]陈明．数据库系统及应用 SQL Server 2000．北京：清华大学出版社，2007．30－80．

[14]王华杰．精通 C#数据库编程．北京：科学出版社，2003．20－50．

[15]普赖斯，J. (Price, Jason)．C#数据库编程从入门到精通．北京：电子工业出版社，2003．50－60．

致　　谢

在论文的完成之际，我要感谢张坤龙副教授对我的悉心指导。在这半年的学习中，张坤龙副教授即使在百忙中也要抽出时间给我的论文编写提出宝贵的意见，他的敬业精神是我学习的榜样。张老师不辞辛苦的谆谆教导是我不断奋进的动力，我要再次衷心的对他表示感谢。

同时感谢的我师兄吴宗远、陈伟飞，他们在我论文的完成过程中给了我热情的帮助，使我免走了很多的弯路。同时感谢我的室友吕恒、王鹏、赵盾、许强和谢晓龙，他们在我完成论文的繁忙之际，从生活上支持和关心我。

同时感谢我的家人，是他们的理解和支持使我能顺利完成大学四年的学习和生活，让我一次次鼓足勇气应对各种挑战。同时对那些在大学期间给我帮助的师长、亲人、朋友们一并表示由衷的感谢。