分布式数据库管理系统

实验报告

王　川 P0916013
李　宁 P0916029
程向力 P0916012

2010-1-17

目　　录

1 需求分析.....1
1.1 客户端功能需求.....1
1.2 服务器功能需求.....2
2 系统结构.....2
3 全局数据字典.....3
3.1 GDD的节点结构.....4
3.2 GDD的整体结构.....5
3.3 GDD的更新与维护.....5
4 查询树.....6
4.1 查询分析.....6
4.2 查询树节点结构.....7
4.3 查询树结构.....7
5 网络.....8
5.1 协议的简介.....8
5.2 Client和Control Site之间的通信.....9
5.3 站点之间的通信.....9
6 查询处理.....10
6.1 查询过程简介.....10
6.2 通信量的统计.....11
7 全局查询处理.....11
7.1 查询树的生成.....11
7.2 查询树剪枝.....12
7.3 查询树优化.....13
7.4 生成执行方案.....13
7.5 执行查询.....14
8 界面.....14
8.1 系统初始化.....14
8.2 系统查询.....14
8.3 查询树显示.....15
8.4 GDD显示.....16
9 分工.....16
10 总结.....17

1 需求分析

1.1 客户端功能需求

图 1 显示了客户端需要提供初始化系统、导入数据、查询数据、显示数据和相应查询树等功能。具体来说，我们需要实现以下一些基本功能：（1）、 初始化系统：系统将会根据脚本创建 GDD 表，使系统能够在该 GDD 表的基础上完成每个站点的初始化；（2）、 导入数据：系统初始化后，可以把数据以文件的形式插入到相应的表中；（3）、 查询数据；（4）、 显示数据和查询树：根据输入的查询语句，显示查得的数据结果和查询树。

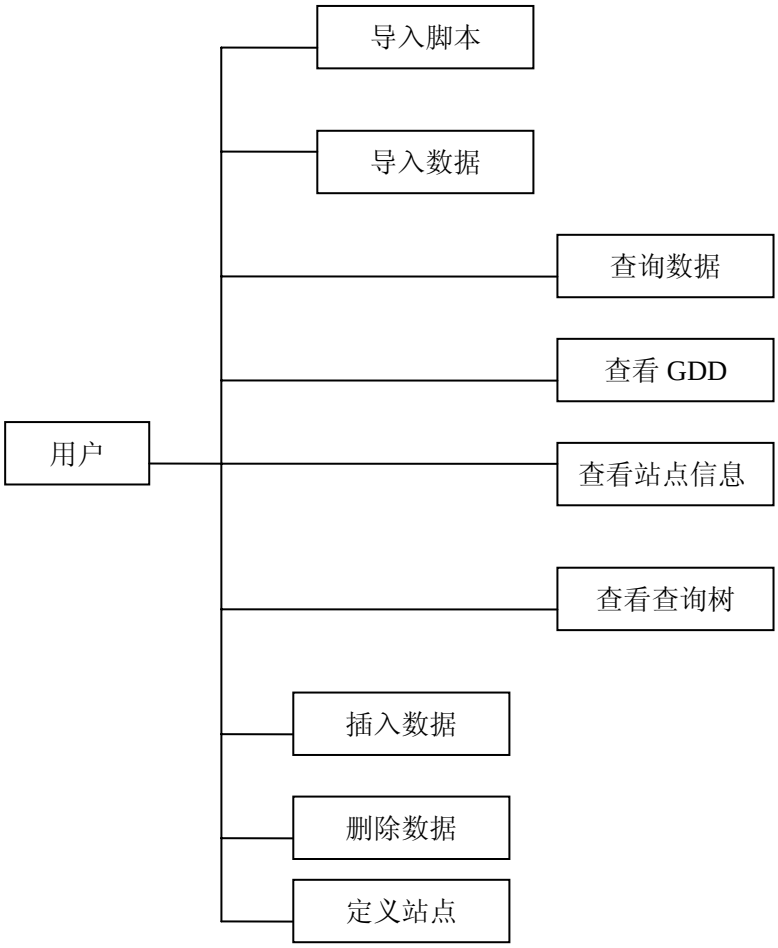

图 1 分布式数据库管理系统客户端业务流程图

1.2 服务器功能需求

图 2 显示了服务器提供的功能：（1）、 解析命令：服务器会在定义的主控站点上解析命令(导入脚本、查询、insert、delete)；（3）、传输命令：控制站点分发命令；（4）、GDD 更新；（5）、返回结果：各站点把查询结果返回给主控站点，服务器把查得的结果(数据和查询树)返回给客户端。

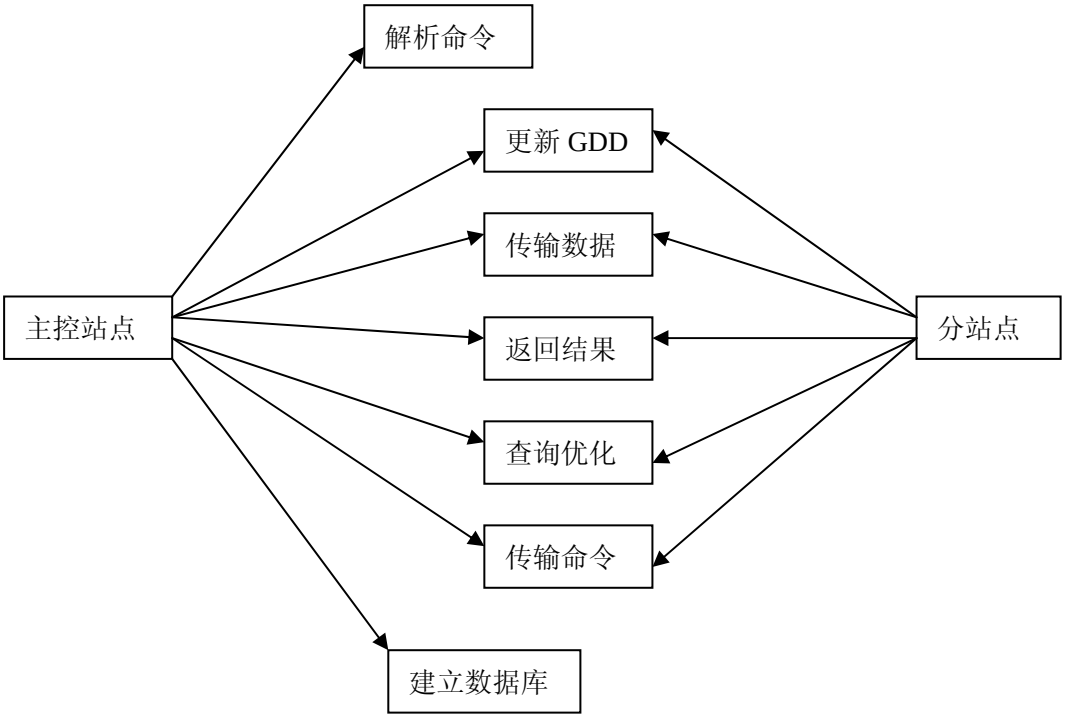

图 2 分布式数据库管理系统服务器业务流程图

2 系统结构

该系统是基于三层模型由用户、socket 网络通信和分布式数据库服务器组成的。图 3 显示了这一特殊结构的工作模式。

用户在客户端上可以进行一些操作，或发布命令；客户端会把用户输入的命令通过 socket 协议发送给服务器；在服务器的控制站点上先解析接收到的 command，

(1)接收到的是初始化脚本，控制站点会建立数据库，并根据解析的脚本初始化该数据库，生成 GDD 表，并把 GDD 表发送到每个分站点。

(2) 接收到的是操作命令，控制站点会根据 GDD 表的信息，把解析的命令

分发到相应的本地站点，在本地站点上完成各自的任务。在查询数据的过程中，由于需要做 union 操作，所以本地站点会把处理的中间结果发送到另外的本地站点上。最后本地站点把操作结果返回到控制站点上，由控制站点把最终结果返回到客户端，用户就能查看到相应的结果了。

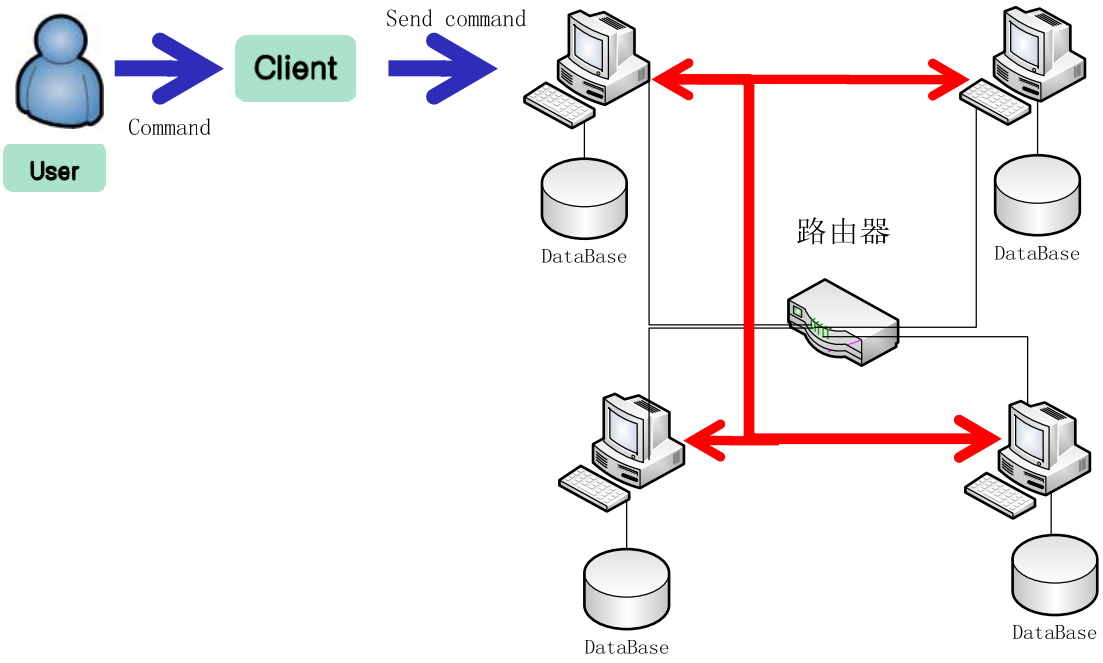

图3 分布式数据库管理系统结构图

3 全局数据字典

在传统数据库中，数据字典起到的描述系统中各类数据的作用，是进行详细的数据收集和数据分析所获得的主要成果。同样，我们在设计分布式数据库时，也需要这样一个数据字典，设计分布式系统时，这个数据字典我们称之为 Global Data Dictionary，简称为 GDD，在后面的叙述中我们将一直使用 GDD 来代替数据字典。

GDD 的作用主要有两个方面，一方面，在进行分布式数据部署时，需要将数据部署的详细情况记录在 GDD 中。另一方面，在进行分布式处理数据时，需要知道数据在那个站点等详细的信息，GDD 为操作提供优化方案和操作流程。

在设计数据库时，我们将 GDD 的部分放到了 Server 端，采用这样的设计是经过和同学讨论得出的，放在 Server 端一是可以减轻 Client 端的压力，二是在更

新 GDD 时候省去了和 Client 端交互的传输量。Client 端将输入的数据和脚本直接传送给 Server 端，Server 进行处理之后生成 GDD 存储在本地，并与其他存储节点进行交互不断动态维护 GDD 信息。

在设计 GDD 时，我们认为将其设计成链式结构有利于 GDD 的扩展，例如存储节点在交互的过程中可能会出现 GDD 发生变化的时候，因为链式结构可以动态的编辑信息。并且我们的数据表需要进行分片，这样一个表就形成了一个从上至下的多层结构，如果设计成其他结构比较浪费空间，链式结构在访问分片信息也比较快。下面将比较具体的介绍一下 GDD 的结构。

3.1 GDD 的节点结构

因为 GDD 中需要存储网络中本节点和其他存储节点的网络信息，并且还应当包括本地的数据分片信息。因此在设计 GDD 时我们将这些信息都放在 GDD 的节点中，通过一个指针数组指向本数据表的分片。下图是使用 C++实现的节点代码：

```
typedef struct _SITEINFO {
    CString ip;
    UINT    port;
    CString name;
    BOOL    is_self;
}SiteInfo;
```

图 4 站点数据结构描述

上述代码描述了节点的信息，包括 IP，端口等信息，is_self 定义此节点是否为 Control site，Server 端通过查询这个值来判断自己是否为 Control site。

```
typedef struct _TABLE {
    CHAR table_name[MAX_TABLENAME_LEN];//定义表名
    CHAR tuple_name[MAX_TUPLE_OF_TABLE][MAX_TUPLE_NAME_LEN];
    UINT num_of_tuple;
    int  is_fragement; /*是否对表做了分片：0-不拆分，1-水平，2--垂直*/
    CHAR info[MAX_INFO_LEN];
    SiteInfo *whichSite;
}Table;
```

图 5 表结构描述

定义 Table 结构，这个结构对应表结构并添加了网络信息，tuple_name 定义

了表结构内的列信息。

3.2 GDD 的整体结构

我们定义的 GDD 结构可以支持水平分片，垂直分片。通过设置不同的 Table 结构中 is_fragment 的值来支持不同的分片。

```
typedef struct _TABLENODE {
    UINT  num_of_table;
    Table table[NUM_TABLES_OF_TABLENODE];
    /*table[i] is fragement, then the node_next[i] is not null*/
    struct _TABLENODE *node_next[NUM_TABLES_OF_TABLENODE];//可能会有多个分片
}TableNode;
```

图6 表节点描述

上面代码定义了 GDD 是如何进行组织，node_next 指针指向后面分片，每个分片也是使用了 Table 结构，因为数据分片实际上和数据表的地位相同。如果出现了分片，则在 Table 中设置 is_fragment，并在 node_next 指向的数据分片中定义分片信息。下图是一个简单的分片实例的空间结构：

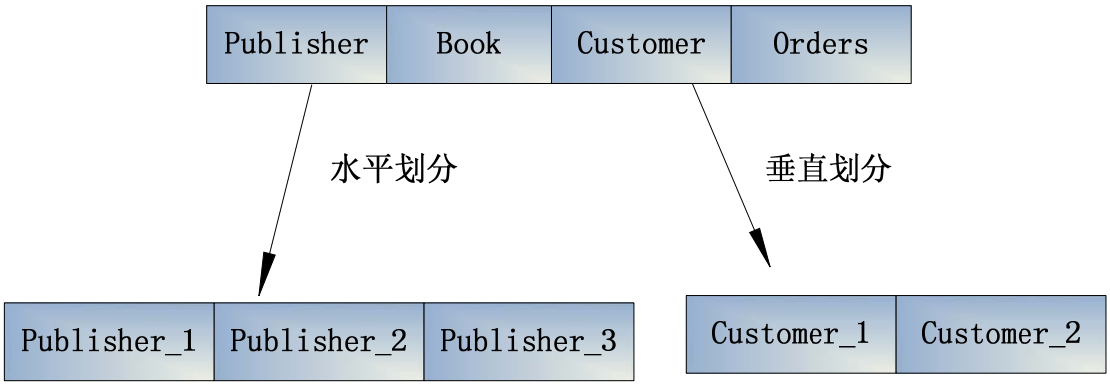

图7 GDD 结构图

3.3 GDD 的更新与维护

GDD 中数据需要在初始化阶段进行建表，然后根据脚本信息将 GDD 信息发送给存储节点。实际上存储节点应不停的进行互相交互，以确定是否是否有存储节点发生了宕机或程序崩溃的问题，动态的维护 GDD 表，但是我们在设计时刚开始没有考虑这种情况，因此我们的程序在有数据操作时才对 GDD 表进行更新，

每次更新都由 Control Site 发送更新信息到所有的存储节点中，类似于网络的广播功能。

4 查询树

Query Tree 是一个树结构。通过查询语句生成这棵查询树，这棵查询树的结构应有利于查询优化并提供查询方案。生成查询树应先进行对查询语句的扫描，词法分析和语法分析。从查询语句中识别出语言符号，如 SQL 关键字，属性名和关系名等，进行语法检查和语法分析，判断查询语句是否符合 SQL 语法规则，在我们的程序中在 CQueryTree 中 ParseSQL 函数实现。

进行查询分析之后，需要根据数据字典对合法的查询语句进行语义检查，即检查语句中的数据库对象，如属性名，关系名，是否存在和是否有效。检查通过后便把 SQL 查询语句转换成等价的关系代数式。这个关系代数式就是我们的查询树。

每个查询都会有许多可供选择的执行策略和操作算法，查询优化就是选择一个高效执行查询处理策略。在这一章节中我们不会详细介绍查询优化，查询优化将在后面介绍。下面将详细介绍查询树的情况。

4.1 查询分析

我们是用 ParseSQL 函数来进行查询语句分析，这个函数在 CQueryTree.cpp 中具体实现，我们在设计 ParseSQL 时，主要在这个函数里进行如下几个操作：FindSelectCols，FindConditions，CreateGenericQueryTree，QueryTreeSave。

FindSelectCols 用来查询语句中的列名，FindConditions 用来查询语句中的条件，在这两个函数中查询结果要记录在数组中以方便我们在下一步进行使用。CreateGenericQueryTree 用来生成查询树，这里需要根据 GDD 的相关内容来生成查询树，最后一步查询树保存在文本文件中。查询列名和查询条件时需要进行词法检查和语法检查，只有符合条件的语句才能生成查询树。

4.2 查询树节点结构

查询树是一个树形结构，下面是主要的数据结构：

```
typedef struct _QUERYTREENODE {
    INT type;                //1--table; 2--operation
    Table *table;
    OPERATION operation;
    struct _QUERYTREENODE *pparent;
    struct _QUERYTREENODE *pchild[MAXCHILD];
}QueryTreeNode;
```

```
typedef struct _OPERATION {
    INT type;                //1--projection; 2--selection; 3--join; 4--union
    CHAR sql[NAME_BUFFER_LEN];
    CHAR whi[NAME_BUFFER_LEN];
    SiteInfo *site;          //在哪个站点执行
    CHAR recordsetName[NAME_BUFFER_LEN];
    CHAR fragment;//存储分片信息
}OPERATION;
```

图8 查询树节点定义

上述代码用来定义节点，节点分为两种类型，操作节点和表节点，所谓操作节点就是这个节点上有操作动作，表节点实际上指查询树上的叶子节点，这些叶子节点是真实的数据分片表，其他操作节点操作的数据都是临时表。在操作节点中有一个 OPERATION 的结构，OPERATION 结构如图 7，需要指出的是，在叶子节点中此结构不存储任何信息，只存储分片的表信息。

4.3 查询树结构

查询树结构时函数在进行 CreateGenericQueryTree 之后生成的查询树，生成的查询树的结构如下：

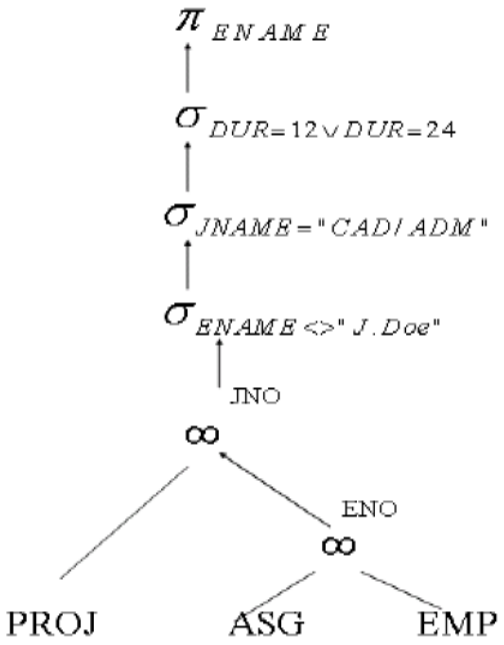

图9 查询树示意图

在这个查询树中没有分片信息，所以表下没有出现分片。生成这棵树首先应先定义一个根节点，根节点有自己的名字，我们在程序里定义的名字称为m_QueryTreeRoot。以方便在以后使用查询树生成查询方案时访问这棵树。除去叶子节点，所有的节点都是操作节点，都需要在 OPERTATION 结构里定义操作详细信息，生成的执行方案从最下层的操作开始不断的生成临时表，这样才能进行上层的操作。

5 网络

5.1 协议的简介

Client 和 Control Site 之间的通信以及 Site 之间的通信，通过 socket 来进行通信。它们之间的通信协议由字符串组成，采用同步的方式进行通信，每次需要向目的站点发送命令或传输文件的时候，先建立 socket 连接，然后发送相应的命令或传送文件。每个服务器开启一个主线程，监听端口，当有一个连接到来的时候，为该连接开启服务线程，当服务完毕的时候，关闭服务线程。用这种简单的方式实现服务器端的多线程。

5.2 Client 和 Control Site 之间的通信

Client 和 Control Site 之间的通信的主要内容有：（1）、定义控制站点；（2）导入数据；（3）导入脚本文件；（4）、insert 和 delete 指令；（5）、SQL 查询指令；（6）、获得 GDD；（7）、获得查询树；（8）、获得查询结果。涉及的命令及相关的含义如下：

“DEFINE”：定义控制站点，当一个站点接收到该命令后将自身标记为控制站点，同时 Client 端也记录下该站点为控制站点；

“SENDSCRIPT”：当控制站点接收到该命令，表示客户端在向自己发送脚本文件，作为控制站点，需要根据接收到的脚本生成 GDD，同时将该 GDD 发送到脚本中定义的其他站点；

“SENDINITDATA”：控制站点接收到该命令后，表示客户端在传递数据集，那么控制站点接收数据到自己的 Data 目录下面；

“SENDSQLCMD”：表示客户端需要进行 SQL 查询，控制站点对发送过来的 SQL 指令进行解析，并且执行，返回给客户端执行的结果；

“GETSQLRESULT”：获得查询的结果，控制站点将查询的结果发送给客户端；

“GETGDDTABLE”：获得全局数据字典；

“GETQUERYTREE”：获得查询树；

“GETTOTALDATATRANS”：客户端获得在查询过程中总的通信量；

“INSERT”：执行 insert 指令，根据分片的信息向表中插入数据，同时向客户端返回操作的结果；

“DELETE”：执行 delete 指令，根据分片的信息删除表中数据，同时向客户端返回操作的结果；

5.3 站点之间的通信

站点之间通信包含控制站点和其他服务器站点之间的通信以及非控制站点的服务器站点之间的通信，实现的功能有：（1）、控制站点发送脚本；（2）、控制站点发送分片好的数据；（3）、控制站点控制创建相应的表，并且将分片数据导入表；（3）、进行查询；（4）、中间结果的发送；（5）、统计各个站点的通信量等。涉及的命令及相关的含义如下：

“SENDDATA”：控制站点发送分片好的数据，接收到数据的站点将数据导入自己的表中；

“CREATETABLE”：根据命令创建相应的表，这个表可能是初始化的时候创建表，也有查询的时候用于存储中间结果的表；

“SENDRECORDSET”：将查询的结果集发送到对端；

“SENDRECORDSETTRANS”：接收到该命令的站点进行本地的 SQL 查询，同时将该结果转发到制定的站点上；

“LOADRECORDSET”：将中间结果集导入相应的表中；

“QUERYTABLE”：进行本地的 SQL 查询；

“GETDATATRANS”：将本地的通信量传递给控制站点；

“INSERTTABLE”：执行插入表的 SQL 指令；

“DELETETABLEDATA”：从指定的表中删除数据。

6 查询处理

6.1 查询过程简介

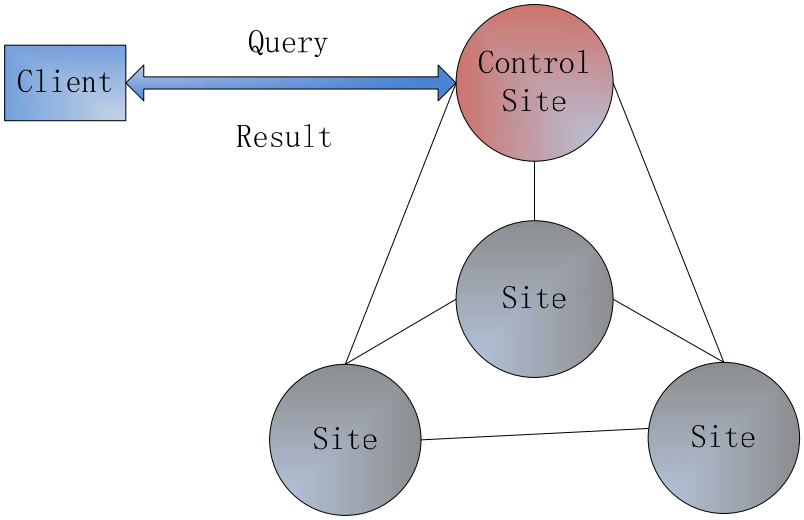

图 10 查询处理过程图

查询的过程比较简单，如图 10 所示。查询的时候只需要客户端向控制站点发送需要执行的 SQL 指令，控制站点在接收到客户端发来的 SQL 命令后，对 SQL 命令进行解析，创建查询树，并且做优化，生成执行方案。控制站点控制执行方

案的执行，在查询出最后的结果后，将结果发送回客户端，如果查询失败，控制站点需要传送失败的信息给客户端。客户端在接收到结果后，显示查询结果，同时打印查询时间，查询涉及的通信量。

6.2 通信量的统计

在进行查询的时候，控制站点将自己的一个用于统计通信量的变量置为零，同时给其它的站点发送命令，将变量置为零。在查询的过程中，如果某个站点有接收数据的操作，则将该变量加上传输的文件的大小。当查询完毕后，客户端向控制站点发送统计，每个站点将变量的值发送给控制站点，控制站点把所有的变量加上自己的通信量后即为查询过程中总的通信量。最后，控制站点将总的通信量发送给客户端显示即可。

7 全局查询处理

在控制站点接收到 SQL 指令后，为了简单处理，没有对 SQL 指令进行正确性检查。解析 SQL 命令，生成查询树。在生成查询树的过程中，不用关心表是否进行了分片，生成查询树后，对每个叶子节点进行分片的处理，然后进行剪枝的操作，剪枝后进行优化的操作，最后生成执行方案。

7.1 查询树的生成

关于查询树的结构在第四节的时候已经介绍过了，现再详细介绍下其生成的过程：

首先，分析 SQL 命令，获得查询的条件以及条件的数目；

其次，分析 SQL 命令，获得该查询条件涉及的表以及表的数目；

然后，为了优化的更加简单，对查询条件进行排序处理，JOIN 操作放在最后处理；

再然后，根据涉及的表，创建叶子节点，再根据查询的条件从叶子节点往上生成整棵查询树。

这样，生成的查询树与分片的信息是无关的。如果在 SQL 语句中某个表是

可以不需要的，那么需要删除创建的该叶子节点。生成查询树的同时，需要记录下叶子节点的信息，因为在生成执行方案的时候，需要从叶子节点往上生成执行方案，如果记录下叶子节点，就不需要再搜索查询树。

7.2 查询树剪枝

查询树的剪枝操作包含两种类型的剪枝操作：水平分片剪枝和垂直分片剪枝。

1、水平分片剪枝

如果 SQL 查询中涉及的表是水平划分的话，那么根据前面的生成查询树的规则，生成的查询树如图 11 所示。在做剪枝操作的时候，需要对每一个分片的表从他们的 UNION 节点出发直到查询树的根节点，按照分片的信息进行比对，如果在这过程中所有节点上的操作都和该分片的表不相关的话，那么将该分片的表从查询树中删除，只要有一个节点的操作和该表相关，则开始处理下一个分片的表。

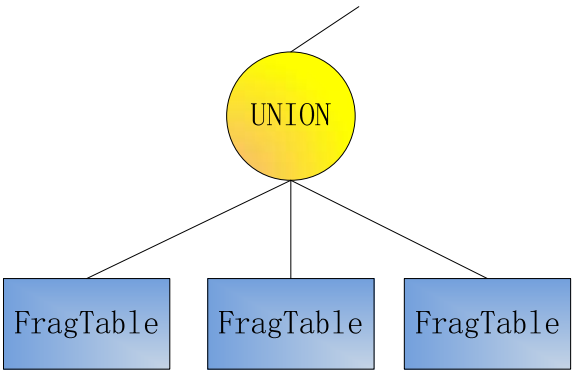

图 11 水平剪枝示意图

2、垂直分片剪枝

如果表是垂直分片的话，那么生成的查询树节点如图 12 所示。在做剪枝操作的时候，从 JOIN 节点出发一直到查询树的根节点，如果每个节点上的操作都不涉及到垂直分片的表列项，那么剪掉该分片，否则不能剪枝。

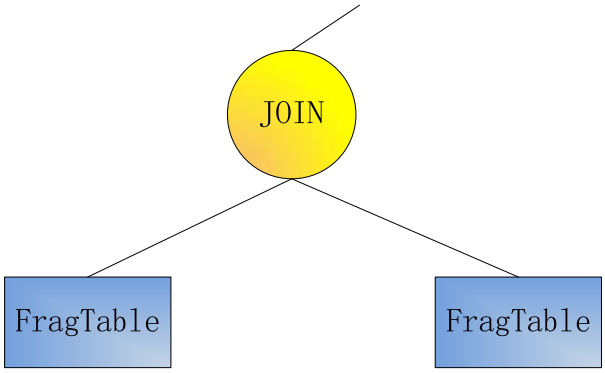

图 12 垂直剪枝示意图

7.3 查询树优化

查询树优化遵循的规则应该有：

1）、先做 selection，将 selection 下移；

2）、在做 join 前先做 project 操作；

3）、做 union 前先做 selection 操作。

在做查询优化的时候，做 selection 下移的时候，先采取对条件排序的办法，这样可以确保先处理的是 selection 操作。

Join 操作前做 project，需要判断在 join 操作节点上面的节点需要的列是哪些，然后做 project 选择出这些列。

Union 前先做 selection 操作，可以在生成好查询树后，从 Union 节点往上搜索，如果有关于分片的表的 selection 操作，那么把它们移到 union 前进行处理。

另外，为了加快查询的速度，需要建立表的索引，而我们的程序中没有创建索引的步骤，导致在做查询的时候速度比较慢，这个需要改进。

7.4 生成执行方案

执行方案的数据结构如下：

```
typedef struct defSCHEME {
    INT         type;       //该操作的类型
    CString     sql;        //需要执行的 SQL 命令
    SiteInfo    *site[6];   //该 SQL 指令涉及的站点
    CString     recordset;  //SQL 命令执行后形成的结果集
}SCheme;
```

图13 执行方案数据结构

执行方案的生成是通过从叶子节点开始扫描，当操作为选择或投影操作的时候，直接生成执行方案，如果是 union 或 join 则停，并且标记下这个节点，移向下一个叶子节点，往上走到该节点后再处理该节点。

最后形成的执行方案存放在一个数组中，执行的时候只需要到该数组中取得执行方案，再发送到执行站点上即可。

7.5 执行查询

控制站点将生成的执行方案发送到执行方案中的执行站点，然后等待执行站点返回结果，如果是在控制站点上执行，那么不需要传递命令，直接执行即可。

8 界面

8.1 系统初始化

定义 Control Site，导入数据，导入脚本，完成了初始化。

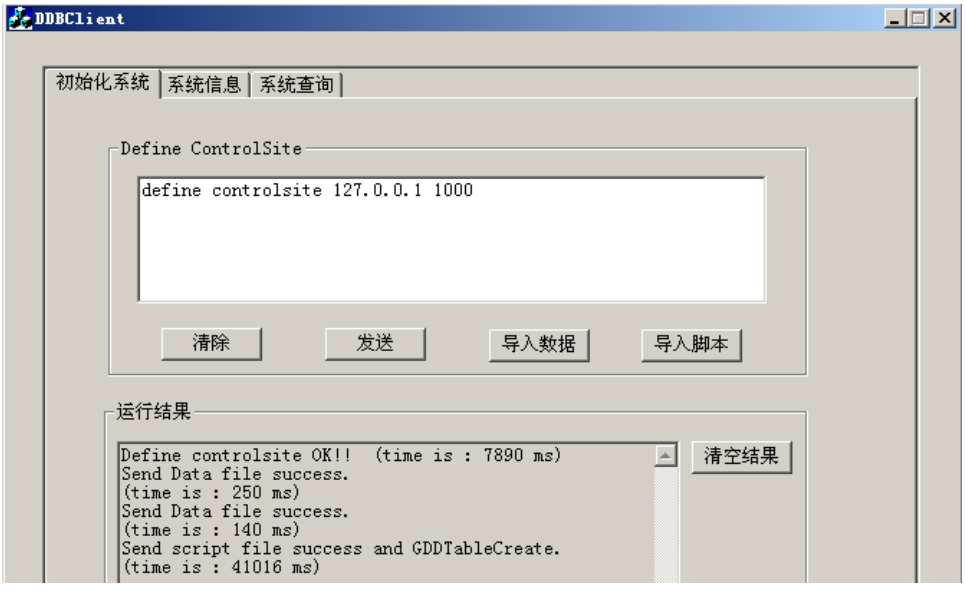

图14 初始化

8.2 系统查询

输入查询语句，点击执行(使用示例 select * from Customer)。

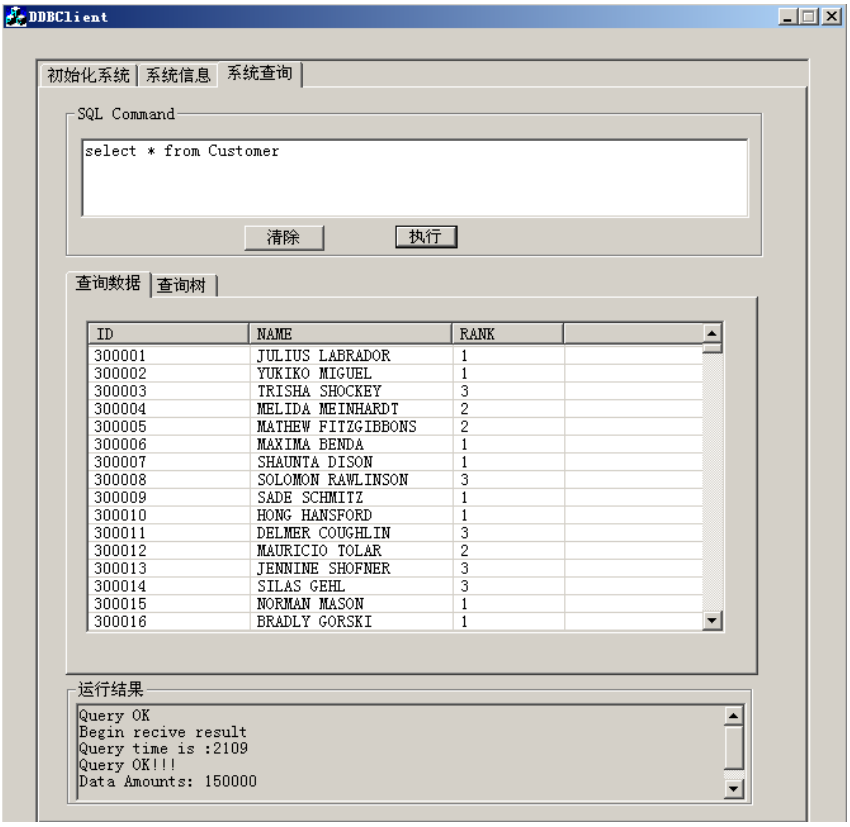

图15 系统查询

8.3 查询树显示

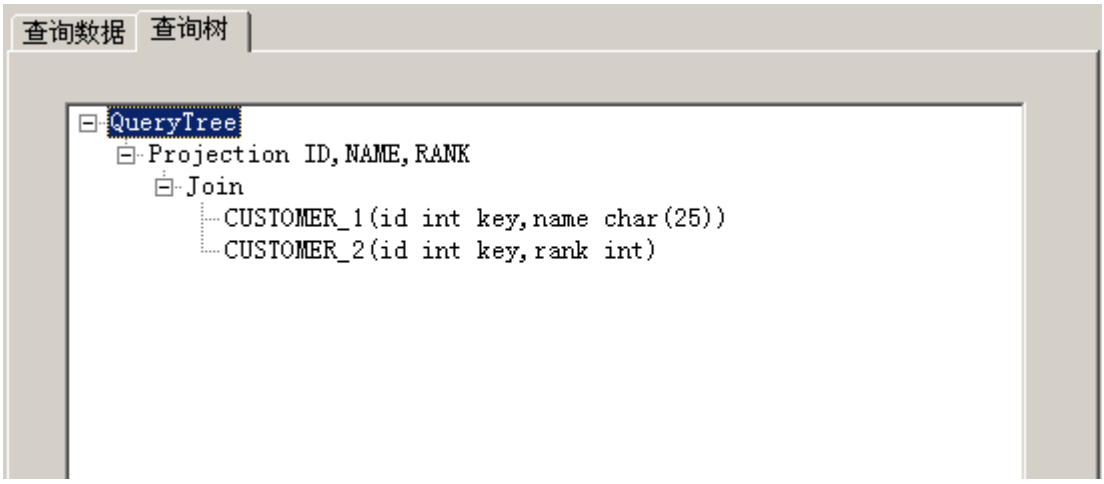

图16 查询树显示

8.4 GDD 显示

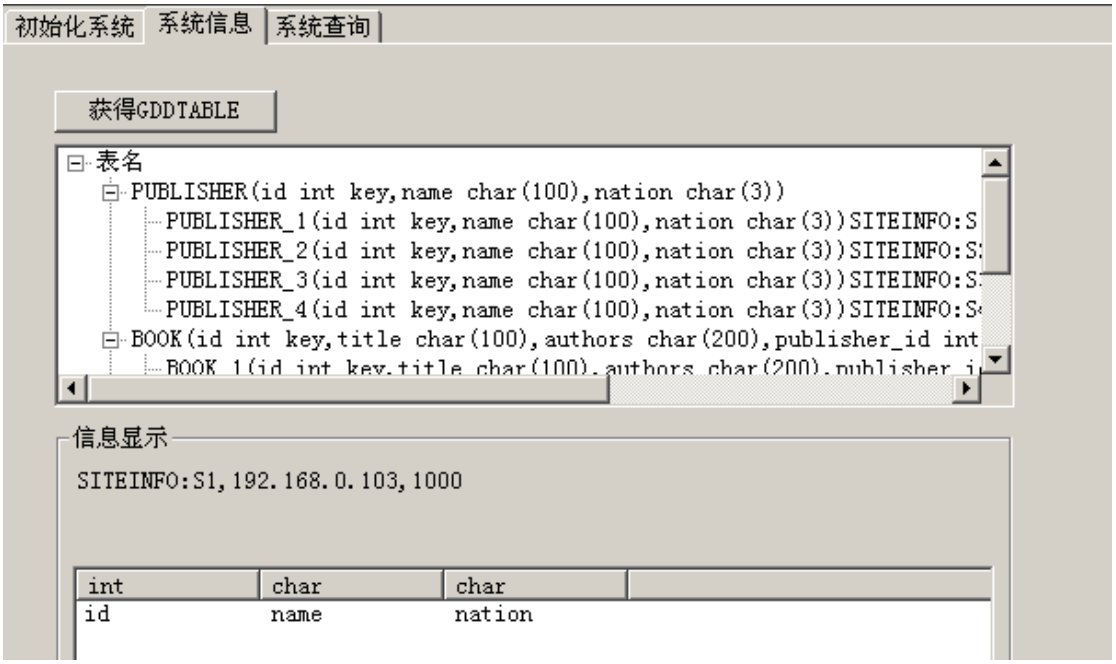

图17 全局数据字典显示

9 分工

一个简单的分工表格如下：

负责人	工作内容
王　川	（1）、系统设计 （2）、网络、GDD （3）、查询树及优化 （4）、主要的代码工作 （5）调试、文档
李　宁	（1）、查询树 （2）、调试 （3）、文档
程向力	（1）、界面 （2）、调试 （3）、文档

10 总结

总的来说，分布式数据库算是代码任务比较重的一门课。最后统计完成的工程，代码大概有 5000 行，虽然完成的水平一般，但是在这一学期写了这么多的代码，对自己也是一个不错的锻炼。我们这组由于是外校的，并且外校只有我们这一组选这门课，能够交流的比较少，导致完成的系统中有两个比较主要的问题：

1）、查询树在做优化的时候没有在 union 操作前加 selection 操作，导致了 union 操作的时间增加；

2）、做查询的时候没有做索引，导致在做查询的时候尤其是涉及到多个 join 操作的时候时间比较长，用 MYSQL 在本地不建索引做测试也需要很长的时间，导致这个问题是因为对数据库不熟悉。

非常感谢周老师和冯老师这学期的教导，虽然写代码写的很累，但是做完后想想锻炼了不少，同时也感谢范举助教这学期的指导。