数据库多表查询优化技术①

李文俊, 张爱林

(中国船舶重工集团公司第七一〇研究所 生产部, 宜昌 443003)

摘 要: 利用面向对象语言 C++作为开发语言, 采用了 ADO 数据库访问技术、Socket 网络通信技术进行了仓库信息管理系统的开发. 设计了一种改进混合蛙跳算法对关系数据库的多表查询进行优化, 使其通信费用和响应时间都显著降低. 该系统使用 C/S 架构, 具有登录权限管理, 数据库访问, 数据库操作, 数据库查询管理, 报表及打印等功能模块. 最后实验证明所设计的系统查询效率有显著提高, 能够准确完成各种查询管理任务, 提高了仓库管理效率并节约了成本, 具有较强的实际意义.

关键词: 混合蛙跳算法; 查询优化; 关系数据库; 赫夫曼编码; C++

Multiple-tables Query Optimization Technology for Database

LI Wen-Jun, ZHANG Ai-Lin

(Product Department, 710th Research Institute of China Shipbuilding Industry Corporation, Yichang 443003, China)

Abstract: This paper uses object-oriented programming language C++ as a development language. The work uses ADO database access technologies and socket network communication technology in warehouse management information system design. It uses Shufled Frog Leaping Algorithm on multiple-table query Optimization for relational database. It makes their communications and response time are significantly reduced. The system uses c/s framework with logon rights management, database access, database operations, database query management and function modules such as printing a report. At last, experiment proves system designed makes Query time improved significantly and it accurately complete all kinds of queries management tasks. The system improues the storage management efficiency and makes cost saued, it has a strong practical relevance.

Keywords: shufled frog leaping algorithm; query optimization; relational database; Huffman coding; C++

传统的人工仓库管理中, 仓库管理员需要花费大量的时间在物品的统计中, 手工操作、人工判断或记录分析信息导致工作效率低下, 成本高昂. 随着计算机技术和信息技术的高速发展, 数据库技术的出现和运用有效弥补了人工仓库管理的不足. 查询是数据库操作中使用频率最高的一种操作, 查询效率的优劣直接影响数据库系统的性能及市场前景. 随着数据库系统、硬件以及通讯技术的发展, 现代数据库存储的信息量越来越多, 面对庞大的信息, 数据库中关系表的个数和关系元组数等都会有很大的增长, 从而使查询优化更加复杂. 数据库系统中, 查询的开销主要是 CPU 和 I/O 运算时产生的代价. 在查询过程中, 如果统计概貌不完全或者连接顺序不正确, 这些因素都会造成查询效率的下降. 因此, 查询过程中查询序列的选择将是影响查询效率的主要因素, 也是查询优化的核心问题. 面对急剧增大的数据量, 传统的优化策略表现出很多不足, 一些算法很容易陷入局部最优或只针对特定的问题和用户等, 所以, 改进的优化算法应具备在较少的计划执行时间内, 找出全局的最优序列.

算法的优化在数据库查询优化的研究中占很大比重, 近些年来, 国内学者基于模糊集和可能性分布理论, 识别出 XML 文档中的多粒度数据模糊性, 提出了从模糊关系数据库到模糊 XML 模型的转化算法[1], 为全面构建模糊数据管理体系和通过模糊理论优化查

① 收稿时间:2013-11-02 收到修改稿时间:2013-12-09

询提供了理论基础. 针对大规模数据集列存储的方式, 严秋玲等提出适合于列存储的启发式查询优化机制[2], 有效减少候选计划的规模, 排除大量不可能生成最优计划的计划, 使得查询处理代价和执行时间大大减小. 为了提高关系数据库信息检索的效果, 张俊提出对象级别的关系数据库信息检索研究[3], 为关系数据库信息检索未来的研究提供了新的思路和方法. 关系数据库中空值存在不同的语义, 并且会影响模糊查询结果. 针对该问题, 陈逸菲等提出用标号来区分空值的语义, 并且在 EPTV 逻辑的基础上[4], 对关系运算和一些复杂的嵌套查询进行扩展. 针对数据库系统复杂的多连接查询问题, 可以运用贪婪算法[5]实现对数据库的查询优化, 缩短查询时间, 提高查询效率.

以此同时, 国外相关领域的研究工作更多的是倾向于分布式数据库查询效率的提高. NICK ROUSSOPOULOS 设计的一种可扩展的数据库系统[6], DOKEROGLU T.等提出的一种新式启发式微粒群算法[7], WU Wei-hua 等提出的基于回溯算法的查询优化方法[8], TOP-K, K-means 等算法[9]-[12]的运用都大大提高了数据库的查询效率.

1 仓库信息管理系统的设计

1.1 系统的体系结构

仓库信息管理系统是基于 Microsoft Visual Studio 2005 平台开发的. 系统具有如下功能模块:

① 登录权限管理模块: 用户信息的管理, 系统安全的维护和系统日志的记录.

② 数据库访问模块: 主要负责数据库的连接.

③ 数据库操作模块: 可以对数据库进行初始化, 查询, 删除及更新.

④ 数据库查询管理模块: 各种查询条件的管理.

⑤ 查询优化模块: 利用混合蛙跳算法对查询效率进行优化, 找到最佳查询路径.

⑥ 报表及打印模块: 各种统计报表的生成及打印输出.

仓库信息管理体系结构框图如图 1 所示.

1.2 软件的层次结构

系统在软件实现上共有六个基本结构层:

① 表示层: 表示层是人机交互的通道.

② 任务适配层: 为表示层和底层提供统一通用的代码接口, 降低各层之间的代码耦合度.

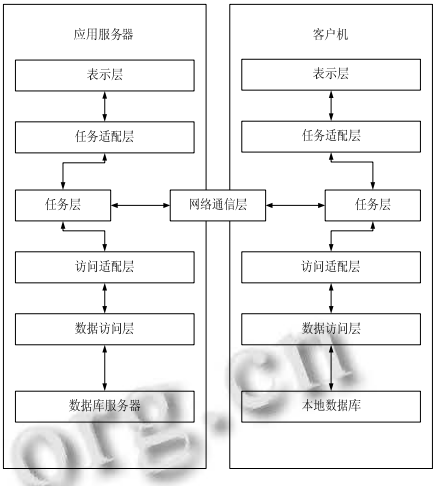

图 1 仓库信息管理体系结构框图

③ 网络通信层: 主要用来建立网络环境中不同计算机之间的连接.

④ 任务层: 处理各种各样的任务.

⑤ 访问适配层: 与任务适配层的功能相似, 起到隔离任务层和数据访问层的作用, 为任务层和数据访问层提供统一通用的代码接口, 提高了独立性, 降低了耦合度.

⑥ 数据访问层: 利用 ADO 技术提供对数据库的访问接口.

2 仓库信息管理系统的实现及其关键技术

2.1 数据库设计

系统软件开发时使用的数据库是以二维表的形式储存数据和表示数据结构的形式, 设计一种有效的关系型数据库表结构, 是系统数据库设计的核心. 数据库关系表之间的关系如图 2 所示:

2.2 基于混合蛙跳算法的查询优化关键模块功能的实现

2.2.1 混合蛙跳算法

Muzafar Eusuff 和 Kevin Lansey 提出的混合蛙跳算法(shuffled frog leaping algorithm , SFLA) 是一种基于群智能的亚启发式计算优化算法, 用于解决离散组合优化问题. 作为一种新型的仿生物学智能优化算法, SFLA 结合了基因算法(MA, memetic algorithm)和微粒群算法(PSO, particle swarm optimization)2 种算法的优点. 该算法具有概念简单, 调整的参数少, 计算速度快, 全局搜索寻优能力强, 易于实现的特点.

混合蛙跳算法的原理是: 混合蛙跳算法的核心部分是局部更新和全局更新. 局部更新中, 子蛙群根据

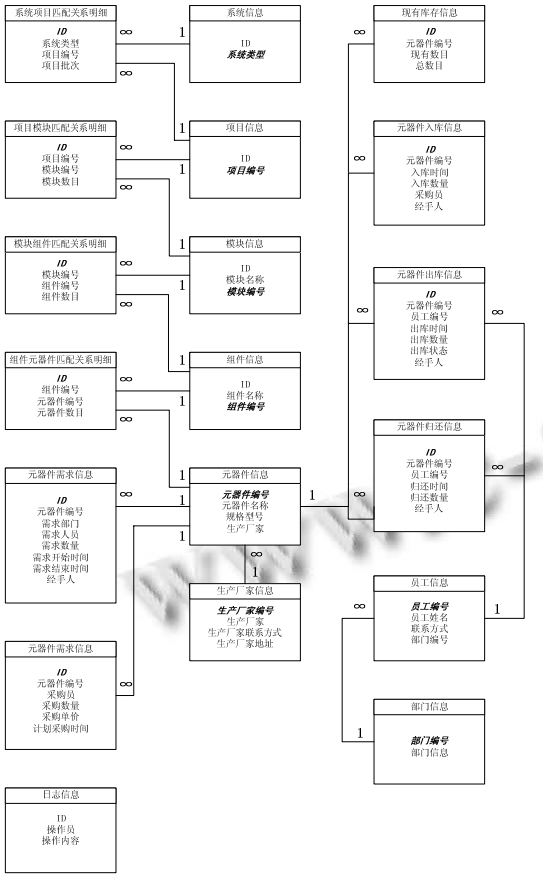

图 2　数据库关系图

特定更新策略进行局部更新, 优化局部个体; 全局更新中, 将所有子群青蛙混合, 混合的同时进行全局更新操作, 优化全局. 局部更新和全局更新一直持续交替进行到满足预先定义的收敛条件后, 更新结束, 输出结果.

初始化:　随机产生 F 个青蛙个体, 每个青蛙个体代表问题的一个可行解, 计算每个青蛙个体的适应度, 同时根据适应度大小, 将青蛙个体从大到小排序. 将适应度最大的青蛙个体设置为全局最优解.

划分子群: 将排序完成的青蛙个体按照一定的划分规则划分为若干个青蛙子群.

局部更新: 对于每个青蛙子群, 找到适应度最大的青蛙个体和适应度最小的青蛙个体, 同时设置局部最优解为适应度最大的青蛙个体; 局部最差解设置为适应度最小的青蛙个体. 对于局部最差解按照一定的更新规则进行更新操作, 同时将更新后得到的青蛙个体代替原来的局部最差解.

全局更新: 所有的青蛙子群进行若干次局部更新后, 将所有青蛙个体混合, 再根据青蛙个体的适应度由大到小排序, 将适应度最大的青蛙个体与全局最优解比较, 若干该青蛙个体优于全局最优解, 则替换全局最优解.

判定阶段: 判断是否满足算法收敛条件: a.在最近的 n 次全局更新之后, 最好解没明显改进或者;b.系统设置的更新次数已经达到. 以上 2 个收敛条件满足之一就可, 若满足, 输出更新后的全局最好解作为最优解; 否则, 返回到局部更新阶段.

2.2.2 算法模型

考虑一个关系数据库, 包含多张数据关系表, 多表查询优化问题, 实际上就是一个组合优化问题. 优化的目标就是根据查询连接树, 迅速找到一个查询执行代价最小的查询路径. 假设:

① 所有查询连接数用左深树表示.

② 所有权值两两相异.

③ 大部分数据关系表为被查询率低, 权值小的感应节点.

④ 每个关系表都被设置一个权重系数(取值范围 0 到 1), 由感应节点被查询频率和重要性决定. 例如, 在仓库信息管理系统中, 元器件表的权值比员工表的权值大.

该模型中, 若关系表的权重值越大, 那么说明该关系表被查询的频率较高或者所含信息类型较重要, 那么构建搜索空间时, 它的查询优先级也就越大, 根据权重值大小来构建搜索空间的, 可以提高其实时性.

2.2.3 算法实现

首先对查询搜索空间中所有的感应节点按相应的权值进行赫夫曼编码, 关系数据库中的每个关系表都有对应的二进制赫夫曼码. 将码长相同的感应节点归为一组集合, 记为集合, 此处的下标 k 表示该集合中传感节点所对应的赫夫曼码码长. 根据查询 SQL 语句得到查询所关联的关系表, 根据关系表对应的赫夫曼码码长, 在相应的中筛选出需要的关系表. 所有关联的关系表组成搜索空间, 由 SQL 查询语句所定义的关系表之间的连接关系, 该搜索空间可以转换为二叉树(左深树).

(1)对该二叉树采用蛙跳算法求解代价最小解, 即:

(1.1)初始化, 在搜索空间内, 遍历二叉树的一序列为搜索空间的一可行解. 设定二叉树的可行解子集合(k=1……m)的数量 m 以及每个可行解集合内的可行解个数 n, 则每个搜索空间的所有可行解总数;

(1.2)随机生成若干个二叉树的 F 个可行解组成的初始群体, 并设二叉树可行解, 其中, , ……, 分别代表该二叉树可行解所依次经过的感应节点的编号;

(1.3)计算上述各个可行解的适应度, 该二叉树可行解的适应度等于该遍历序列代价的倒数. 对于二叉树内部节点 t, 若 l 和 r 是 t 的左右子节点代表的关系表, 即, c 是 l 和 r 的公共属性的集合, 是其中的一个公共属性, 和分别表示关系 l 和关系 r 的基数, 即元组的个数, v(a, t)表示关系 t 中属性 a 的不同取值的个数, 实际问题中, 连接关系的公共属性值往往是呈正态分布的, 设为有公共属性关系的概率值, . 公共属性越多则的值越小. 代价计算一般公式如下:

$$B(t)=\frac{n(l)\times n(r)}{\prod_{c_i\in c} max(v(c_i,l),v((c_i,r))}\omega_j \tag{1}$$

$$v(a,t)=\begin{cases} v(a,l), a\in l-r \\ v(a,r), a\in r-l \\ min(v(a,l),v(a,r)), a\in l且a\in r \end{cases} \tag{2}$$

适应度的公式如下:

$$f(q)=\frac{1}{B(t)}=\frac{\prod_{c_i\in c} max(v(c_i,l),v((c_i,r))}{n(l)\times n(r)\times\omega_j} \tag{3}$$

由公式(3)可以看出, 适应度函数值越高, 对应的代价就越小, 说明适应度越好, 查询效果也就越好.

(1.4)将 $f(q)$ 值大小进行降序排列, 并将 $f_{\max}(q)$ 对应的二叉树可行解记录为初始的全局最好解: $U_{GB}=[U_1,\ U_2,\ \cdots\cdots,\ U_n \mid f(q)=f_{\max}(q)]$; 同时, 按以下分配规则将上述各个可行解分配到 m 个子集合 Y_k 中, 每个子集合 Y_k 中包含 n 个二叉树可行解;

分配规则如下:

$$Y_k=\{U(p),f(q)\mid U(p)=U[k+m*(p-1)],f(q)=f[k+m*(p-1)]\} \tag{4}$$

(4)式中, p =1, 2, ……, n; k =1, 2, ……, m; $U(p)$ 表示子集合中的第 p 个可行解, $f(q)$ 表示第 p 个可行解的适应度;

(1.5)执行局部搜索, 对初始的全局最好解 U_{GB} 进行更新, 具体过程如下:

(1.5.1)在每个子集合 Y_k 内找到适应度最大的二叉树可行解即局部最优解 U_B, 以及适应度最小的二叉树可行解即局部最差解 U_W;

$$U_B=[U_1,\ U_2,\ \cdots\cdots,\ U_n \mid f^{Y_k}(q)=f^{Y_k}{}_{max}(q)] \tag{5}$$

$$U_W=[U_1,\ U_2,\ \cdots\cdots,\ U_n \mid f^{Y_k}(q)=f^{Y_k}{}_{min}(q)] \tag{6}$$

(1.5.2)按以下公式对每个子集合 Y_k 中的局部最差解 P_W 进行更新:

$$U_q=U_W+S' \tag{7}$$

$$S'=min\{int[rand(U_B-U_W)],s_{max}\},U_B-U_W\geq 0 \tag{8}$$

$$S'=max\{int[rand(U_B-U_W)],-s_{max}\},U_B-U_W<0 \tag{9}$$

(7)式中, U_q 表示最差解更新后的解; S' 表示二叉树可行解的调整容限; s_{max} 表示二叉树可行解允许改变的最大容限, 0<rand<1 其值由系统随机生成;

如设=[1 3 5 4 2], =[2 1 5 3 4], 允许改变的最大步长 s_{max} =3, 若 rand=0.5, 则 $U_q(1)$ =1+min{int[0.5×(2-1)], 3}=1; $U_q(2)$ =3+max{int[0.5×(1-3)], -3}=2; 依此相同的操作完成更新策略后可得到一个新解 U_q =[1 2 5 4 3].

(1.6)在每个子集合都进行过一轮更新之后, 将各个子集合进行混合得到 $Y=Y_1\cup Y_2\cup\ldots\ldots Y_m$, 并将更新后的可行解按适应值大小进行降序排列, 并将适应度最大的可行解记录为更新后的全局最好解: $U_{GB}=[U_1,\ U_2,\ \cdots\cdots,\ U_n \mid f(q)=f_{max}(q)]$.

(1.7)判断是否满足算法收敛条件, 若满足, 输出更新后的全局最好解 U_{GB} 作为最优二叉树查询路径序列; 否则, 返回到步骤(1.4)再次更新全局最好解 U_{GB}.

上述步骤(1.7)所述算法收敛条件为: 在最近的 K 次全局更新后, 全局最好解没有得到明显的改进或算法预先定义的函数评价次数已经达到.

具体算法流程如图 3 所示.

算法预计达到的效果是: 1.该优化模型提出为各数据表设置权重值 ω_i, 权重值大小根据数据表被查询的频率以及重要性决定. 被查询频率及重要性高的数据表权重大, 形成搜索空间时, 缩短了筛选过程的时间. 2.利用赫夫曼编码分组, 进一步提高了建立搜索空间的效率, 减小了整个查询时间; 3.采用目前较新较成熟的蛙跳算法求.对二叉树 T, 求遍历路径的最优解, 再次有效的提高了查询效率.

3　系统的调试及实验结果

仓库信息管理系统界面如图 4 所示.

3.1 收敛性实验

查询过程中, 连接关系表个数的不同对算法的效率有很大影响. 因此, 通过改变连接关系表的个数,

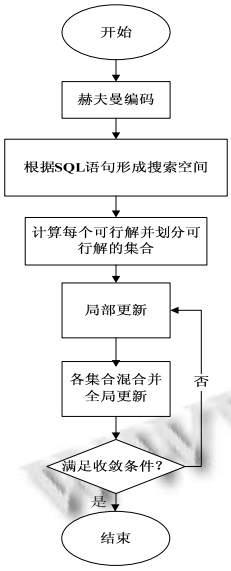

图 3　算法流程图

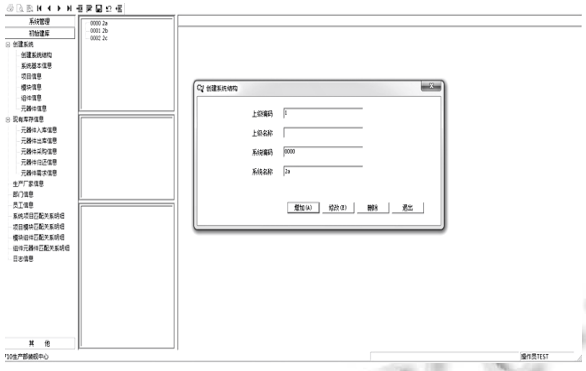

图 4　系统界面图

分别对混合蛙跳算法, 无穷算法, 遗传算法和贪婪算法进行如下实验测试. 在连接关系表数目 W 为 50, 100, 200, 500, 1000 时, 记录混合蛙跳算法 F、穷举算法 E、遗传算法 GA 和贪婪算法 G 的执行代价和收敛时间, 如表 1 所示.

表 1 各种算法的执行代价和收敛时间

W		50	100	200	500	1000
F	B	1600	2489	3950	3460	3780
	T	870	971	1115	1426	1928
E	B	2600	3680	4870	5192	5853
	T	300	780	2187	3892	8253
GA	B	1589	2134	3724	3190	3488
	T	560	980	1490	2200	3140
G	B	1728	2789	4580	3977	4955
	T	300	340	375	410	415

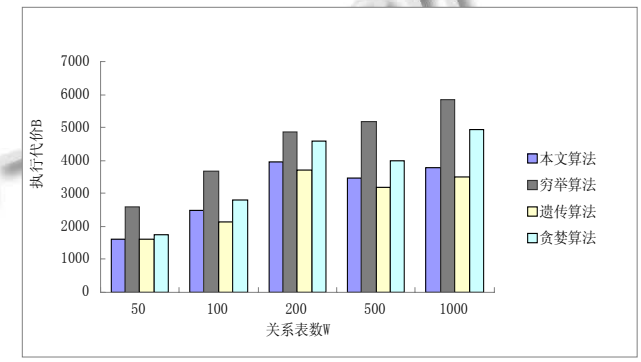

图 5　执行代价对比图

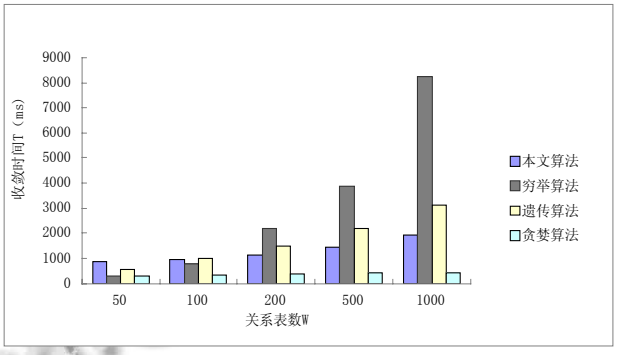

图 6　收敛时间对比图

由仿真实验可以看出, 在大规模的关系数据库中, 该优化方法相比传统的穷举法显著的提高了多表关系数据库的查询效率. 综合对比遗传算法和贪婪算法, 该优化方法可以在花费相对较小的计算代价和较少的时间得到近似全局最优序列.

3.2　f(q)与更新次数实验

可以看出, 当更新次数在 0~200 之间增加时, 函数的输出结果快速得到改善; 当更新次数大于 200 后, 改善效果变得缓慢, 曲线变得平滑; 当更新次数达到 1000 时, 得到的输出结果未比更新次数 200 时函数输出结果有明显改善. 而且过多的更新次数会导致数

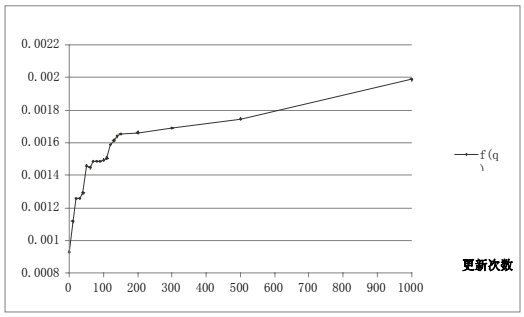

图 7 f(q)与更新次数关系图

据计算量加大, 程序运行时间过长, 从而导致网络的实时性降低, 为了能使网络的实时性达到最佳, 必须选择合适的更新次数.

3.3 f(q) -关系实验

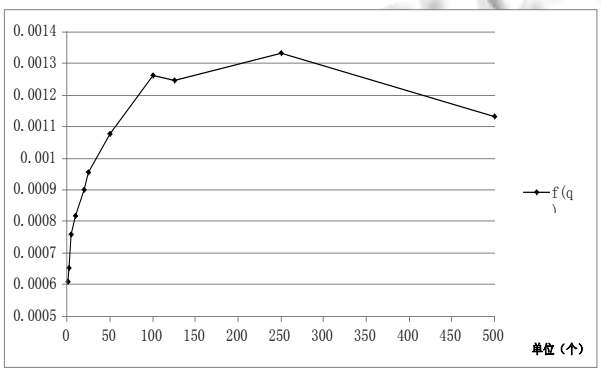

图 8 f(q)与关系图

可以看出在个数在 100 以内增长时, 适应度的输出结果明显得到了改善, 但是当个数在超过 100 后, 输出的结果改善效果不明显, 尤其是当个数达到 500 后, 适应度的输出结果甚至出现下降, 这有由于在总可行解不变的情况下, 盲目的增加子集个数导致的, 因此为了达到最优输出结果, 可行解总数与子集个数需具备一定的比例关系.

4 结 语

系统使用 C++语言开发, 基于 C/S 架构实现. 具有通用性高, 人机交互性好等优点. 主要的创新点在于针对多表关系数据库查询效率问题, 提出了一种基于赫夫曼编码和混合蛙跳算法相结合的查询算法优化方法. 根据实验结果, 可以看出在大规模的关系数据库中, 该优化方法有效的提高了多表关系数据库的查询效率. 小规模的关系数据库中由于搜索空间偏小, 该优化算法在提高查询效率方面与传统的穷举法相比优势不明显. 提高数据库查询效率, 不仅可以立足于本地集中式数据库, 更应该顺应时代的发展, 要着眼于分布式数据库, 通过提高查询并行处理能力和降低网络传输代价来增强数据查询效率.

参考文献

1 严丽,马宗民,刘健,张富.基于关系数据库映射的模糊数据 XML 建模.计算机学报, 2011,34(2):291–303.

2 严秋玲,孙莉,王梅,乐嘉锦,刘国华.列存储数据仓库中启发式查询优化机制.计算机学报,2011,34(10):2018–2026.

3 张俊,邵仁俊,曾一鸣.对象级别的关系数据库信息检索技术研究.计算机科学,2012,39(1):142–147.

4 陈逸菲,叶小岭,张颖超.关系数据库中基于 EPTV 的模糊查询.计算机工程,2010,36(4):25–27.

5 李志伟.基于贪婪策略的分布式数据库查询优化研究.计算机工程与设计,2010,31(17):3838–3840.

6 Roussopoulos N. MOCHA: A extensible database middleware system for distributed data sources. SIGMOD Conference, 2009, 8(12): 2130–2240.

7 Dokeroglu T, Tosun U，Cosar A. Particle swarm intelligence as a new heuristic for the optimization of distributed database queries. AICT/Tbilisi, 2012: 1–7.

8 Wu WH, Liou JG, Li YF. A query optimization algorithm of database based on layered backtracking. ICEOE/Dalian, 2011(3): 178–180.

9 Ilyas IF, Beskale G, Soliman MA. A survey of top-k query processing techniques in relational database systems. ACM Computing Surveys, 2008(4): 11.

10 Wang N, Wang NB. The realize the integration system and query optimization on the eterogeneous data source. Journal of Software, 2008, (2): 222–228.

11 Tsai D, Lai S. Independent component Analysis-Based background subtraction for indoor surveillance. IEEE Trans. on IP, 2009, 18(1): 158–167.

12 Rizman KZ. An efficient K-means clustering algorithm. Pattern Recognition Letters, 2008, (29): 1385–1391.