# 基于图论模型的数据库原型

计算机 095 (10093701) 赵子川

**摘要：** 开题报告：基于图论模型的数据库编写，主要简述了数据库的各类模型，关系型数据库与非关系型数据库的比较，以及非关系型数据库中图论数据库的详细介绍。从实际需求，市场前景，应用案例, 与基本实现等方面详细说明了图论数据库的意义与价值

**关键词：** 图数据库，图数据模型，图处理，图搜索，并行计算，并发支持，分布式系统

# 1 研究背景

## 1.1 数据库的各类模型

现代数据库模型主要分为两大类，以兼容 SQL 语句为主的关系型数据库和非关系型数据库。

其中非关系型数据库又分为：以 Redis 为代表的键值数据库，以 MongoDB 为代表的文档数据库，以 VODB 为代表的对象模型数据库以及以 Neo4J 为代表的图数据库。

这些数据库模型各有其优、缺点，适用于各类不同的场合。

## 1.2 关系型数据库的局限性

在关系模型提出之后，关系型数据库由于其易用性、稳定性等因素，在当时的计算机界迅速发展起来，并一举拿下了数据库世界的占领地位。

然而，就如面向对象编程一样，在各类因素下，一种技术概念的迅速流行会导致这种技术本身被过度使用，甚至被宣称为“万能的”。事实上，现有的关系型数据库有其固有优势的同时，其缺点也是明显的，比如：

### 1.2.1 对数据结构化的要求

以 SQL 为代表的关系型数据库对数据的结构化有着非常强的要求。然而随着 Web2.0 时代的来临，互联网的信息形式开始变得多样，大量非结构化的数据的产生使得在相关环境下关系数据库的设计变得十分困难。

### 1.2.2 数据的扁平化

关系型数据库的数据是以表的形式存储的，在存储记录性数据的情况下是非常优秀、直观的选择。

然而，当不同数据之间开始出现关联的时候，关系型数据库必须对两张或以上的表作笛卡尔积，这一操作是十分消耗资源的。

在现在社交网络兴起的情况下，关系型数据库用来处理以人/人际关系为中心的数据无疑是并不适合的。

### 1.2.3 数据的低分区容忍性

现已证明，数据库系统遵循CAP原则，即高一致性（Consistency），高可用性（Availability），高分区容忍性（Partition Tolerance）三者不可同时满足。

在关系型数据库诞生的时代，数据的存储、处理往往由单一计算机完成，所以对于分区容忍性的牺牲是理所当然的。

然而现在分布式计算兴起，对于分区容忍性的要求提高，而关系型数据库在分布式计算中往往成为瓶颈。相反，大量要求高分区容忍性的应用场景中对一致性/可用性的要求并不那么高。

所以我们应当视实际情况而对数据库进行选择，而不是任何产品一概使用SQL作为数据库。对于适合关系型数据库作为数据库的产品，我们应当毫不犹豫地选择关系型数据库，而对于不适合关系模型数据库的产品，我们也应当考虑非关系型数据库的选用或关系型数据库和非关系型数据库的结合使用。

## 1.3 图数据库面临的现状

近年来，对于管理现实世界中固有的图状信息的需求使其重新回到相关领域。

事实上，随着大规模网络（如互联网，地理系统，运输系统，电话系统）的发展以及由于数据的自发聚集（如生物网络、社交网络）产生的各类网络，一大波全新的、面向图数据库的产品开始出现。

本项目将要实现的图数据库就是为了更好的解决图状信息的存储而产生的。它将更好的处理复杂的网络数据。它将是一个嵌入型内存图数据库，将在后阶段兼容SQL语句查询由此，一个数据库对现实世界的描述能力是衡量数据库的一个重要标准。

# 2 文献综述

图数据库主要需要解决的问题有：图结构的表示和存储、图检索/查询/处理算法的实现，I/O的实现与优化，大数据处理，高并发支持等。

## 2.1 图数据模型

图结构的表示和存储属于图数据库中最重要的环节之一，也是研究最为深入、最为广泛的一类学科。从数学意义上的图论，有Douglas.B.West著的Introduction to Graph Theory, Diestel著的Graph Theory，Gary Chartrand著的Introduction to Graph theory, 都广泛而深入地介绍了图论中的各类概念、问题与方案。

在图论的计算机科学实现方面，Swamy 著的 Graphs Networks and algorithms，对于图搜索，图处理，网络流等算法都做了深入的讨论，而著名书籍算法导论中，也对各类图算法及其数据结构进行了广泛的介绍。

对于图的数据结构，主要分为基于矩阵的表示方法和基于邻接表的表示方法，其中基于矩阵的表示方法在进行搜索遍历的时候有较好的性能但是消耗空间极大，而且难于处理超节点。而基于邻接表的各类数据结构有着较强的可伸缩性，空间冗余较小，是比较理想的一种表示方法。

## 2.2 并发支持

并发是当今各类数据处理系统（包括数据库）的重要特性，如何有效地处理并发也一直是一大难题。

目前的并发技术主要分为两大类，基于锁的并发机制和无锁并发机制。

Breshears 著有的 The Art of concurrency 对这两种算法均给予了一定的介绍。而 Goetz 的 Java Concurrency in Practice 也基于 Java 为各类并发机制给出了实现。

基于锁的并发机制一直是当今主流，然而锁自身是悲观的（即处理并发之前假设会产生冲突），由于悲观并发机制自身的排斥性特征，会导致许多不必要的操作浪费，甚至带来死锁等问题。

乐观的无锁机制，现在比较成熟的技术有 STM（Software Transactional Memory 软件事务性内存），通过在内存中事务机制的实现，基于线程安全的可回滚操作，为乐观的无锁并发机制提供了良好的解决，也是本项目处理并发时的首选技术。

在 STM 方面，由于 Clojure 编程语言对其提供了良好的支持，大量 Clojure 书籍，如 Halloway 的 Programming Clojure，Fogus 的 The Joy of Clojure 等都对 STM 的机制和实现做了深入的探讨。

# 3 技术路线

## 3.1 技术概述

该数据库使用有向超点图进行数据的存储与处理, 使用 Clojure 语言编写，属于嵌入式内存数据库的原型。着眼于网状数据的节点之间关联与联系，主要应用范围可能有：社交网络应用及社交网络分析，地图类应用后端开发，逻辑推断引擎等专家系统开发等。

### 3.1.1 编程语言的选用

首先，Clojure 是一种基于 JVM 实现的 Lisp 派生语言，同时具备 Java 的速度、稳定性、跨平台性以及 Lisp 的灵活性、高抽象性、代码与数据的统一性。

尤其是对于通过临接表存储的图而言，LISP(LISt Proccessing 表处理语言) 拥有其先天优势。

在 LISP 诞生初期，计算机科学刚刚起步不久，当时计算机的性能以及当时的优化技术很难支撑 LISP 这一高抽象层次语言的运行，加之 LISP 的书写格式并非时间线性的，这一点使得熟悉 ALGO 语系（如 C/C++/JAVA）语法的程序员难以接受。所以，一直以来 LISP 以及以其为代表的各类函数式编程语言一直处于非常小众的局面。

然而，近年来随着计算机科学的发展，越来越多的语言开始走向抽象化，LISP 的实用性也开始渐渐上升。加之面向数据流而非时间序列的函数式编程范式在处理并发时有其不可比拟的优势，以 LISp 为代表的各类函数式语言（包括 CLISP，Clojure 等 LISP 家族语言，Haskell，Erlang, Scala 等其他函数式语言），已经在并发系统上有着非常广泛的应用。

### 3.1.2 数据结构的表示

对于有向超点图，我们通过有向邻接表实现，对于超点，我们专门为邻接表设计了一种更加优化的数据结构，通过将邻接指针只想由红黑查找树实现的嵌套 Hash-map，我们实现了命名链接、链接超边等特性。同时对该 Hash-map 的特定键进行保留化，使得树状的超点嵌套成为可能。

我们使用 Clojure 内置的数据结构实现数据的持久化。作为一种代码即数据的语言，从数据在内存中的实现、数据通信以及在磁盘上的持久化，都可以使用统一的数据结构通过统一的方式（即 Clojure 代码）进行表达，而不需要额外的 XML、DAT、BSON 等持久化数据结构。

### 3.1.3 数据的存储

我们使用系统内置的文件系统进行节点的存储。使用文件树存储节点树，单个文件存储每个节点的属性。

这一举措的支撑理论在于：早期磁盘效率低下，文件系统也不完善，在文件的储存、查找性能方面有着大量的限制（如单个文件夹内文件数量的限制等），所以早期设计的数据库使用单一的文件（如.dat）实现存储。然而现在的文件系统已经做了足够的优化，目录树产生的冗余相对于数据本身而言微乎其微，同时将文件以目录树形式存储还避免了文件游标的定位过程，也更加方便了分布式系统中数据的切片。

同时，由于数据分散度较高，多个线程/进程对不同数据进行读写的时候不会由于读写同一文件造成文件锁冲突——这一点对 RAID（磁盘阵列）而言尤其重要。

### 3.1.4 数据 IO

为了提高数据库的性能，我们将数据库设置为基于延迟写入机制的内存数据库，在配置中设定内存使用上限之后，即可以尽可能的将数据在内存中进行操作，只有在内存使用超过限制时，才使用特定算法，将一些数据换出（即写入磁盘后在内存中释放）。

这样可以极大的提高数据库的性能，但会牺牲一部分的可靠性（如系统突然掉电的情况下，未写入磁盘的数据将丢失），为此，我们引入了日志系统、定时热备、异常捕获时抢写等

技术来使风险降到最低。

## 3.2 功能描述与基本流程

在主要开发周期内(原型),应当实现的功能有:图结构的内存表示与硬盘存储、节点及链接的基本查询、更改、图/子图查询与处理(如最短路径查询,最小生成树识别,网络流处理等)、基于图处理的逻辑推断引擎、图处理语言的实现(基于 LISP 表处理语言)、基本的 I/O 支持、对于超节点的支持、基于无锁乐观机制的多线程优化。

计划中的额外功能有:更加优化的高一致性保持、日志事物系统、数据库冷/热备份、对于超图的完全支持、对于 SQL 查询的兼容、大数据优化、高并发优化与分布式系统实现。

# 4 进度安排

| | | |
|---|---|---|
| 12/12 | 13/03 | 查阅文献,熟悉环境,完成文献翻译和开题报告 |
| 13/03 | 13/04 | 相关算法分析及设计 |
| 13/03 | 13/6 | 对主要开发周期的基本功能(原型)进行开发和编码,采用测试驱动开发将开发与测试相结合 |
| 13/05 | 13/06 | 撰写论文 |
| 13/06 | 13/06 | 参加答辩 |
| 13/06 | | 继续进行扩展功能的开发 |