**基于Neo4j的BIM数据库系统研究**

1. 绪论
   1. 研究背景及意义

BIM产业的发展、BIM数据的增长、数据共享的重要性等等 仍然存在数据查询效率地的问题

* 1. 国内外研究现状

关系数据库、BIMServer（BerkeleyDb）、内存数据库redis、Hbase；

空间关系数据处理的不足，引出图数据库

* 1. 本文贡献

围绕neo4j内核模型研究与改进、bim数据的存储与查询来写；

深入研究neo4j内核代码，阐述了neo4j的内核模型和缓存模型；

提出cache模型并实现；

改进neo4j路径查询并实现；

1.4 组织结构

第二章 相关技术基础

2.1 IFC相关技术

2.1.1 IFC标准

2.1.2 IFC文件格式

3.1 IFC数据模型概述

3.1.1 IFC属性数据模型

3.1.2 IFC关系模型 直接引用、间接引用

2.3 图数据库技术

2.3.1 图的概念

2.3.3 Neo4j概述

第三章 基于neo4j的BIM数据库设计

3.3 数据库架构

3.3.1 IFC解析过程

3.3.2 IFC存储过程

3.3.3IFC查询过程

第四章 基于Neo4j的缓存技术研究

4.1 neo4j内核模型

4.2 neo4j缓存模型

4.3 改进的缓存算法

4.3 特定路径查询算法

第五章 系统实验

5.1 实验环境

5.1 缓存算法试验

5.1.1 数据集

5.1.2 查询测试

5.2 特定路径查询算法试验

5.2.1 数据集

5.2.2 查询测试

5.3 系统可用性测试

5.3.1 访问界面

5.3.2 访问接口

第六章 总结与展望

# 参考文献

[1] Jorgensen K A,Skauge J,Christiansson P, Use of IFC Model Servers:Aalborg University and Aarhus School of Architecture, 2008

[2] 张洋，基于BIM的建筑工程信息集成与管理研究，清华大学，2009

[3] Sciences N I O B National Building Information Modeling Standard Verion1-Part1:Overview,Principles,and Methodologies,2011-9-10 ed 2007

[4] Fu C;Aouad G;Lee A IFC model viewer to support nD model application[外文期刊] 2006(2)

[5] Mell P;Grance T,The NIST Definition of Clouding Computing: National Institute of Standards and Technology

[6] Faraj I;Alshawi M;Aouad G.An Industry foundation classes Web-based collaborative construction computer environment:WISPER 2000(1)

[7] 陆宁，马智亮，利用面向对象数据库与关系数据库管理IFC数据的比较，清华大学学报（自然科学版），2012

[8] Kang H;Lee G,Development of an Object-Relational IFC Server 2009

[9] 刘强，张建平，胡振中，基于键-值缓存的IFC模型Web应用技术，清华大学学报（自然科学版），2016

[10] 余芳强，张建平，刘强，赵文忠，基于云计算的半结构化BIM数据库研究，土木建筑工程信息技术，2013(6)

[11] 张建平，BIM技术的研究与应用，2011

[12] 周雅莉，胡珉，喻钢，基于IFC标准的隧道工程信息传递研究，土木建筑工程信息技术，2015(12)

[13] 佐佐木达也，NoSql数据库入门，2012

[14] 岳莉，基于Lucene的全文检索系统的研究与应用，西安电子科技大学，2010

[15] 徐财应，基于Lucene的搜索引擎技术的研究与改进[D].中国科学技术大学，2014

[16] 陈亚杰,王锋,邓辉,刘应波, ElasticSearch分布式搜索引擎在天文大数据检索中的应用研究,天文学报，2016(3).

[17] Rafa l K, Marek R. ElasticSearch可扩展的开源弹性搜索解决方案. 时金桥, 柳厅文, 徐菲, 等译.电子工业出版社, 2015: 25

[18] 龙慧芬，移动社交网络中的数据库应用[J].山西青年管理干部学院学报，2013,26(3):106-108

[19] 王余蓝，图形数据库Neo4j的内嵌式应用研究[J].现代电子技术，2013,35(22):36-38

[20] 王余蓝，图形数据库Neo4j与关系数据库的比较研究[J].现代电子技术，2012,35(20):77-79.

[21] Huang H,Dong Z. Research on architecture and query performance based on distributed graph database Neo4j[C]. Consumer Electronics,Communications and Networks(CECNet),2013 3rd International Conference on .IEEE,2013,533-536

[22] Robin Hecht，Stefan Jablonski. NoSql evaluation：A use case oriented survey [A]. Proceedings of 2011 International Conference on Cloud and Service Computing [C]. 2011.336-341

[23] Sacco G M. Inverted index and inverted list process for storing and retriving information, U.S.Patent 8,738,631[P].2014-5-27.

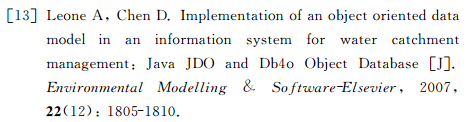
[24] See R, Karshoej J,Davis D. An Integrated Process for Delivering IFC Based Data Exchage[J]. 2012-09-16

[51] 张建平. BIM技术的研究与应用. 施工技术（资讯），2011，（02）

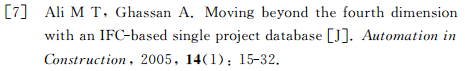
[52] 建筑产品BIM数据库的价值及发展前景. 李美华

[53] 基于IFCXML的建筑数据共享平台的研究与设计

[54]



[55]



【56】对象关系型数据库应用策略 李卓玲

[57]



[58] NoSQL数据库系统

**【59】2016\_A Natural-Language-Based Approach to Intelligent Data Retrieval and Representation for Cloud BIM**

【60】



黄忠东[95]等比较了基于关系数据库

和面向对象数据库实现的特点，如表6.6所示，认为关系数据在目前阶段

更加适用。

随着智慧城市的提出、建筑体量大型化趋势的到来，建筑数据有了长足的增长。2015年11月6日完工的广州周大福金融中心工程总高度530米，占地面积2.6万平方米，总建筑面积50.77万平方米，产生项目图纸5万多张，图纸变更3000余次 [1]。如此大体量的建筑数据为BIM数据库的设计提出了更高的要求。如何解决海量BIM数据的问题迫在眉睫。

当前，BIM数据库按数据库类型可分为关系型BIM数据库、面向对象型BIM数据库、关系-对象型BIM数据库和NoSql型数据库【1】。尤其是进入大数据时代以来，分布式技术如Hbase、Redis等被越来越多地应用在BIM数据库研究中，提高了海量BIM数据数据库的性能。然而，由于IFC数据模型面向对象的特殊性，现有的BIM数据库无法解决IFC对象的关系查询时多表连接可能产生的效率低下问题（这里应该有引用），急需一种新的基于IFC标准的BIM数据的存储和查询方案。

在当前，BIM数据库按数据库类型可分为关系型BIM数据库、面向对象型BIM数据库、关系-对象型BIM数据库和NoSql型数据库【1】。尤其是进入大数据时代以来，分布式技术如Hbase、Redis等被越来越多地应用在BIM数据库研究中，提高了海量BIM数据数据库的性能。然而，由于IFC数据模型面向对象的特殊性，现有的BIM数据库无法解决IFC对象的关系查询时多表连接可能产生的效率低下问题（这里应该有引用），急需一种新的基于IFC标准的BIM数据的存储和查询方案。

1. **关系型BIM数据库**

关系型BIM数据库,其主要采用关系型数据库存储BIM数据。一般根据IFC标准的逻辑大纲设计数据逻辑模式。譬如针对IFC中每个实体定义一个表，并根据该实体的每个属性定义一个表的字段。现有的关系型BIM数据库包括VTT Building and Transport And SECOM Co．开发的一个基于SQL Server的IFC数据库[1],清华大学张洋博士开发的“BIM信息集成平台”[2]，Cruz等开发的ACTIVe3D系统采用Oracle进行IFC数据存储[3][4][5]。由于IFC实体的具有大量可选属性，很多属性值为空(即稀疏性)，但关系数据库仍为其预留大量空间，从而导致存储空间的浪费。另外，关系数据库的扩展性较差，难以支持海量BIM数据的存储，所以关系型数据库已经不能满足BIM数据管理的需要。

1. **面向对象型BIM数据库**

鉴于IFC模型的面向对象特性，一些学者提出应用面向对象数据库存储BIM数据。Faraj利用面向对象数据库ObjectStore存储IFC数据[6]。陆宁设计和实现了基于面向对象数据库Versant Object Database 8的IFC数据库，并通过与关系数据库对比，指出采用面向对象型数据库来管理IFC数据比关系型数据库效率更高[7]。但鉴于面向对象数据库本身理论不完善、价格昂贵等问题，存在难以推广的问题，且其可扩展性较NoSql数据库差。

1. **对象关系型BIM数据库**

面向对象技术与传统关系型数据库技术相结合而形成的数据库系统，也可以说是一种扩展关系数据库，使它具有一定面向对象数据库特征。Kang等利用对象关系数据库Cubrid存储IFC数据[8]。但目前对象关系数据库尚不成熟，没有广泛使用的商业软件。

1. **键值型BIM数据库**

当关系型数据库在可扩展方面几乎已经达到极限时，键值对数据库开始流行起来，它为解决大数据的需求提供了解决方案。这是一种NoSQL（非关系型数据库）模型，其数据按照键值对的形式进行组织、索引和存储。

开源项目BIMServer采用键值对数据库Berkeley DB来进行数据的存储。Berkeley DB是一个开源的嵌入式数据库系统，提供了一系列直接访问数据库的函数，支持存储任意类型的键/值对，具有简单、小巧、可靠、高性能的特点。BIMServer赋予每一个IFC实体一个全局唯一的ID作为key，将IFC实体的所有字段值解析为字节数组作为value，将key/value对存储与Berkeley DB中，能够支持基于IFC实体ID的快速查询和应用。其局限在于键值对存储IFC实体信息的方式比较单一，不能够支持任何关系数据的存储，也不能支持基于字段的查询。

清华大学的刘强等为实现IFC模型在Web应用中的应用，使用分布式键值对存储系统Redis建立了IFC数据的高速缓存[9]。它将IFC文件以数据流方式写入Redis缓存，利用IFC文件的行编号进行索引，大幅提升了IFC文件的解析效率。同时结合内存中的EXPRESS字典使用JSON格式构建IFC对象模型，从而支持面向Web的IFC模型对象的传输和查询等应用。这种方式的局限类似于Berkeley DB，它只能支持基于IFC实体ID的查询。

还有mongo db

1. **列式数据库**

清华大学的余芳强等基于分布式的开源列式数据库Hbase来解决基于列值的查询问题和关系数据join问题。不同于关系数据库的按行存储数据的方式，Hbase以列为单位存储数据，使用列族（Family）和列名共同定义一个列，擅长以列为单位存取数据。Hbase支持动态添加列，从而支持半结构化的BIM模型设计，以一定的冗余避免了多表连接操作，极大地提高了信息提取速度。

但是其半结构化的数据库设计只关注了IFC类型的可交换实体和非可交换实体，存在一定的局限。

6）mongo db

本文在深入研究IFC标准中属性数据的基础上，结合ElasticSearch弹性搜索技术，提出了属性数据半结构化存储算法，同时为了减少冗余，本文基于对属性数据的估算引入冗余控制参数α，提出动态地合并属性数据到IFC实体数据的策略，从而在实现高效提取属性数据的同时，有效地控制了冗余。

其次，本文深入研究IFC标准中空间关系数据，结合Neo4j图数据库技术，实现IFC对象空间关系数据模型的存储。同时，提出了基于用户查询集的动态建立索引的算法，解决了单机存储空间存在限制时控制索引大小问题。

除此之外，本文进一步关注了一种特定的路径查询，并针对这种查询提出了基于BloomFilter的查询实现算法，相比于原生的Neo4j cypher查询，效率得到了明显提升。尤其是在空值查询时能够提前失败，给用户响应。

#### 存储过程

1. 数据模型转换

写清楚IFC的实体与节点的对应，属性与property的对应，空间关系的冗余存储等；

1. 存储过程实现1

写清楚neo4j存储的嵌入式api调用

#### 索引过程

1. 动态索引策略

写清楚日志索引过程

1. 索引实现

写清楚cypher调用

针对IFC实体的属性数据和空间关系数据进行处理。本文提出了属性数据的半结构化存储方式，将属性数据合并到IFC实体对象的数据中，以避免提取属性数据时join产生的效率低下的问题。同时提出了基于Neo4j的IFC实体的空间关系数据存储方式。下面针对属性数据、空间关系数据的处理进行分别介绍。

1. 属性数据处理

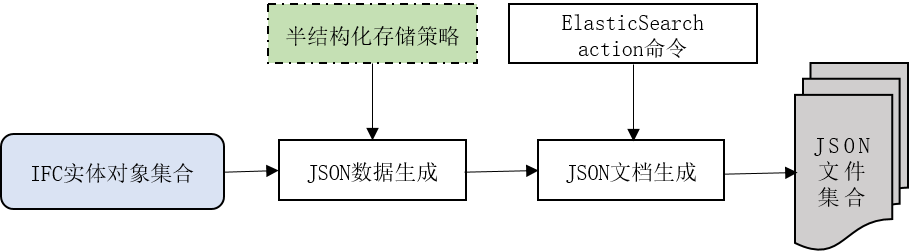
ElasticSearch支持的数据类型是JSON数据。属性数据处理的目的是结合半结构化存储策略，将属性数据合并到其所关联的IFC实体对象的数据中，映射为JSON数据，并结合ElasticSearch的action命令，将JSON数据嵌入到action命令中，形成JSON文档，为存储做准备。属性数据处理流程图如图3-3所示。由于IFC标准的属性类型丰富、结构复杂、类型嵌套、与IFC实体关联机制复杂等特点，在封装为JSON数据时，需要进行特殊的处理。对于属性数据模型和半结构化存储策略，将在第4章详细介绍。

图3-3 属性数据处理流程

1. 空间关系数据处理

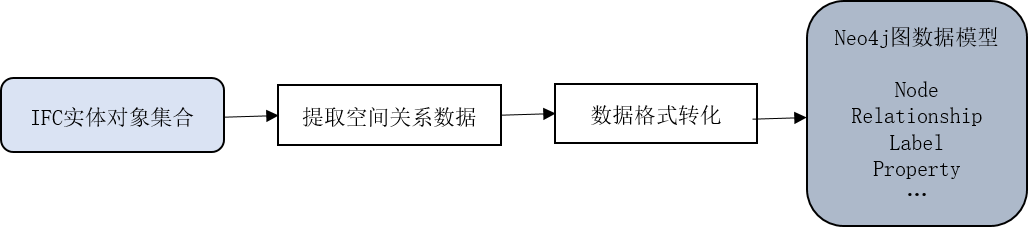
此模块的主要目的是提取IFC实体对象的空间关系数据，转换为Neo4j的节点（Node）、关系（Relationship）等数据类型。关于IFC空间关系数据模型将在第5章进行详细介绍。

图3-4 空间关系数据处理流程

### 查询访问模块

存储模块分为两部分：将属性数据索引到ElasticSearch集群中；将空间关系数据存储到Neo4j图数据库中。具体内容如下：

1. 基于ElasticSearch实现属性数据的半结构化存储。

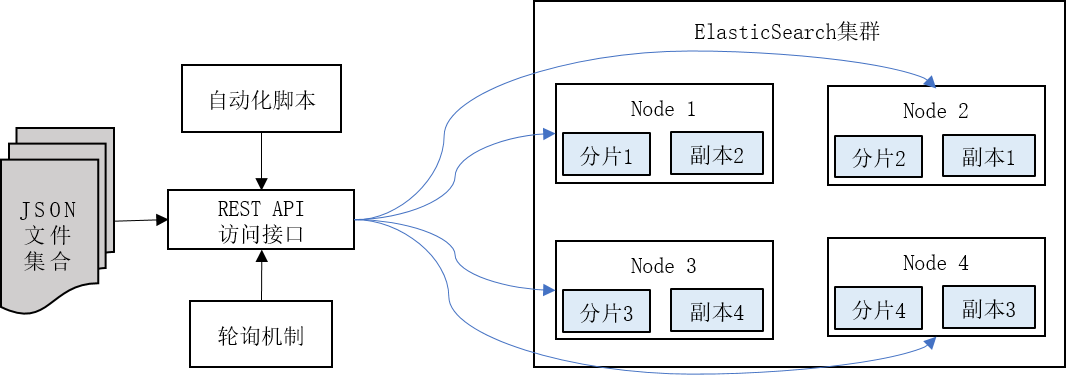
此模块的主要目的是实现批量索引JSON数据到ElasticSearch集群。ElasticSearch 提供了一套非常强大并且易操作的 REST API 用于集群的访问和索引的CRUD操作。本模块通过REST API，同时基于BULK批量导入机制，实现批量索引数据的目的。由于在数据处理模块，JSON数据被组织成以IFC类型名分类的JSON文件中，形成的JSON文件根据IFC文件中出现的不同的对象类型可能会有100多个到600多个。需要编写自动化脚本进行JSON文件的批量索引操作。整个批量索引请求都需要由接受到请求的节点加载到内存中，该节点解析元数据（\_index、\_type、\_id），然后分发给其它的节点的分片，进行操作。对于每个节点来讲，该请求越大，其它请求所能获得的内存就越少。为了减少批量索引时对单个节点造成的性能下降，在发送批量处理请求时采用轮询机制，将请求循环分发到集群中的各个节点，分散解析元数据的压力，也明显提高了索引的效率。

图3-5 批量索引数据到ElasticSearch流程

1. 基于Neo4j实现空间关系数据的存储。

此模块的主要目的是实现IFC空间关系的图数据到Neo4j的存储。本文采用Neo4j的嵌入式版本，自定义访问接口，可以实现图数据的批量存储。同时，引入动态建立索引的算法，其基本原理是结合用户查询热词以及索引空间限制，动态维护索引。动态建立索引算法将在第5章进行详细介绍。另外，本文在第5章针对一种的特定路径查询场景进行了介绍。在这种查询场景下，使用基于BloomFilter的查询算法，效率有明显地提升。

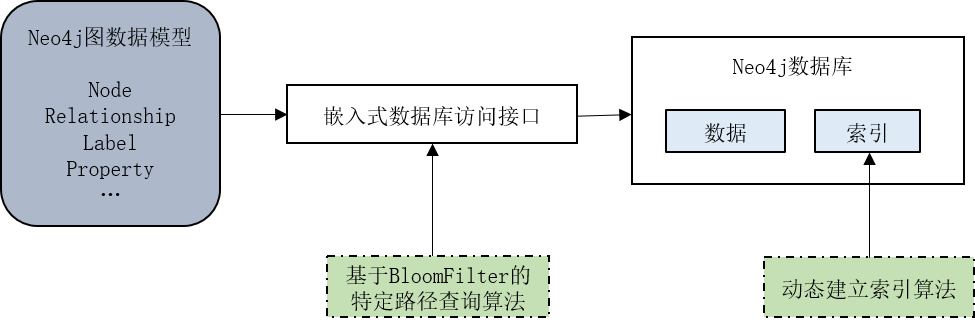


图3-6 Neo4j存储模块设计

在绪论中本文对IFC标准进行了介绍。IFC标准的核心层中不仅定义了IFC对象的位置、几何形状等，同时也定义了IFC对象之间的关系。这些关系表现为空间结构上的组成、包含、连接等多种关系，与IFC对象在物理世界中的关系一致。根据空间关系信息可以方便地进行建筑子模型的查询。例如，查询某栋建筑的二楼的所有建筑构件，或者查询某个房间内的所有建筑构件。目前还没有针对IFC实体的空间关系数据存储的研究。

本文在深入研究了IFC空间关系数据模型的基础上，提出了基于图数据库Neo4j来实现IFC对象的空间关系数据的存储，从而可以实现根据空间关系数据提取子模型。另外，本文基于BloomFilter技术提出了一种针对特定空间关系的查询优化，提高了查询效率，同时对于空值查询能够快速失败，提前给用户响应。

本章的内容组织是：首先介绍IFC实体的空间关系数据模型，然后介绍基于Neo4j的存储技术，最后介绍基于BloomFilter的特定路径查询算法。

当查询到来时，用户

先讲neo4j缓存层的初始化、配置、替换策略，再讲page cache的内存结构，以及在查询到来时如何进行响应。

再讲述一下它的问题，就是一访问之后依旧需要处理，速度慢，二需要将所有的数据都缓存进内存，占用空间大，一旦数据不再，需要进行页置换，消耗比较大。空间管理问题。

因此本文将增加面向图数据的查询缓存。其难点在于数据对象大小差别较大，当缓存空间有限时，需要采用合适的替换策略。采用分级LRU+Clock的置换策略。内存结构问题。（多线程访问问题）。

打开idea，找出内存结构以及内存LFU进行evict的实现。

农业搜索引擎倒排索引缓冲机制研究：

衡量替换策略优劣的指标

找到两级索引的优势。

关系DBMS的缓冲区自适应管理技术研究

其中**缓冲区替换算法和缓冲区空间分配**是传统缓冲区管理技术的核心。

2006面向对象数据库中内存的双缓冲管理机制 单缓冲区的局限性

1994面向对象数据库的缓冲区管理机制设计与实现 传统的页缓冲区的限制

2006关系DBMS的缓冲区自适应管理技术研究

一些数据库管理系统通过TP监视器来调整缓冲池相关配置

基于大对象的数据库缓冲区管理技术的研究与实现

每个缓冲区必须保证留有一些空闲空间，使得新的对象能够加入进来，如果超过允许的界限，需要进行缓冲区空间整理。

包括LRU（Least-Recently-Used）最近最少使用替换策略、MRU（Most-Recently-Used）最近最多使用替换策略、LFU（Least-Frequently-Used）最少使用频率替换策略、MFU（Most-Frequently-Used）最多使用频率替换策略、Clock时钟替换策略、RND（Random）随机替换策略以及这些替换策略的变种等。

LRFU

[30]

算法结合 LRU 和 LFU 算法，利用

一个评估函数选择使用两 者之一，但算法并没有实时动态调整策略。

本算法将LRU链表分成两部分，前半部分和后半部分，如果访问的

页面在前半部分，什么也不做，而不是像传统的LRU算法那样一定将页面移动

到链表头部。这样做是通过牺牲LRU的精确性来减少锁的操作次数，缓存读取

只需要一些原子操作，不需要加锁，大大减少了锁粒度。不过这样命中率可能

会降低。目前很多数据库都采用这样的策略。

1.Collection.sort()->sortedNodeList

/\*得到缓存收益率降序排列的节点集合\*/

2.rest←0,i←0,targetNodeList为空

3.for(;i<sortedNodeList.size();i++){

4. /\*按照收益率从高到低依次将节点加入targetNodeList\*/

5. node←sortedNodeList.get(i)

6. if(rest<node.size){

7. break;

8. }

9. targetNodeList.add(node)

10. rest-= node.size

11.}

12./\*计算最后一个能够加载入内存的节点\*/

13.node← sortedNodeList.get(i)

14. targetNodeList.add(node)



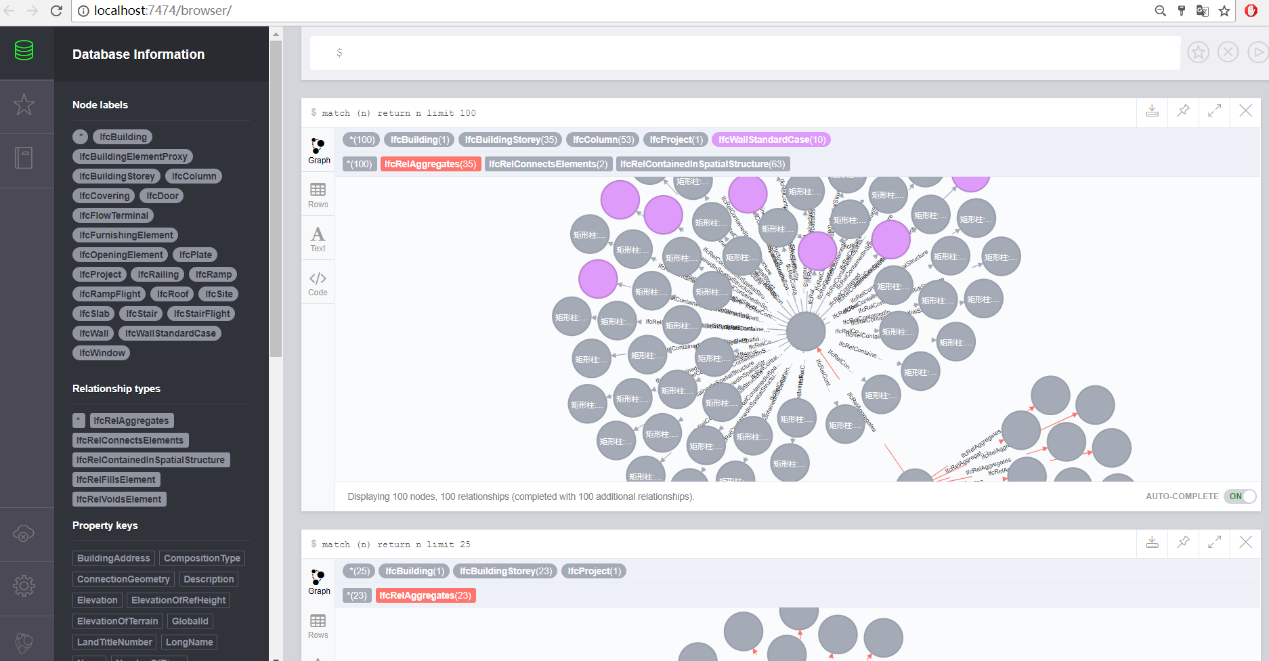
本实验使用BatchInsert方式批量索引节点和关系，经试验，插入的效率是平均每秒9708个Node，接近1万，是相当高效的。索引数据后，使用Neo4j社区版本的客户端连接数据库，并可以通过7474端口从浏览器访问数据库。其界面如图6-5：

图6-5

其中，左侧显示了节点的标签、关系的类型、属性等信息，右边可以使用Cypher语句进行查询。