

SHANGHAIUNIVERSITY

毕业设计（论文）

**UNDERGRADUATEPROJECT(THESIS)**

**题目:针对某领域的数据分析和存储**

|  |  |
| --- | --- |
| **学院** | **计算机工程与科学学院** |
| **专业** | **计算机科学与技术** |
| **学号** | **0** |
| **学生姓名** | **0** |
| **指导教师** | **0** |
| **起讫日期** | **2018.01.29 – 2018.06.08** |

目录

[摘要 III](#_Toc513199145)

[ABSTRACT IV](#_Toc513199146)

[第1章 绪论 1](#_Toc513199147)

[§1.1 数据分析和存储的背景及意义 1](#_Toc513199148)

[§1.2 数据领域现状及存在的问题 1](#_Toc513199149)

[§1.2.1 数据分析存在的问题 1](#_Toc513199150)

[§1.2.2 数据分析的难点 1](#_Toc513199151)

[§1.2.3 数据存储存在的问题 1](#_Toc513199152)

[§1.2.4 数据存储的难点 1](#_Toc513199153)

[§1.3 本文研究内容及目标 1](#_Toc513199154)

[§1.3.1 研究内容 1](#_Toc513199155)

[§1.3.2 研究目标 1](#_Toc513199156)

[§1.4 本文组织结构 1](#_Toc513199157)

[第2章 数据分析 2](#_Toc513199158)

[§2.1 数据分析概述 2](#_Toc513199159)

[§2.1.1 数据分析的基本概念 2](#_Toc513199160)

[§2.1.2 数据分析方法 2](#_Toc513199161)

[§2.2 针对不同类型数据的分析方法 2](#_Toc513199162)

[§2.2.1 Txt数据 2](#_Toc513199163)

[§2.2.2 Excel格式 2](#_Toc513199164)

[§2.2.3 Xml/json格式 2](#_Toc513199165)

[§2.3 本章小结 2](#_Toc513199166)

[第3章 数据存储 3](#_Toc513199167)

[§3.1 多种数据库概述 3](#_Toc513199168)

[§3.2 关系型数据库 3](#_Toc513199169)

[§3.2.1 MySQL 3](#_Toc513199170)

[§3.3 对象型数据库 3](#_Toc513199171)

[§3.3.1 Db4o 3](#_Toc513199172)

[§3.4 图数据库 3](#_Toc513199173)

[§3.4.1 neo4j 3](#_Toc513199174)

[§3.5 文档型数据库 3](#_Toc513199175)

[§3.5.1 mongodb 3](#_Toc513199176)

[§3.6 不同数据库的性能对比 3](#_Toc513199177)

[§3.7 系统中的数据结构与其表现 3](#_Toc513199178)

[§3.8 本章小结 3](#_Toc513199179)

[第4章 食物项目系统的设计与实现 4](#_Toc513199180)

[§4.1 系统整体设计 4](#_Toc513199181)

[§4.1.1 开发环境简介 4](#_Toc513199182)

[§4.1.2 系统整体架构 4](#_Toc513199183)

[§4.2 数据库设计 4](#_Toc513199184)

[§4.3 功能模块设计 4](#_Toc513199185)

[§4.3.1 数据可视化 4](#_Toc513199186)

[§4.4 本章小结 4](#_Toc513199187)

[第5章 总结与展望 5](#_Toc513199188)

[§5.1 本文总结 5](#_Toc513199189)

[§5.1.1 本文的主要工作 5](#_Toc513199190)

[§5.1.2 本文的主要创新点 5](#_Toc513199191)

[§5.2 展望 5](#_Toc513199192)

[致谢 6](#_Toc513199193)

[参考文献 7](#_Toc513199194)

针对某领域的数据分析和存储

# 摘要

XXX。

关键词：XXXX，XXX，XXX，XXX

XXXEnglish Title XXX

# ABSTRACT

Xxx

**Keywords:** xxx

# 绪论

本章主要描述了数据分析和存储的背景，意义。分析了相关课题的研究现状，提出了本文要研究的重点内容和目标。

## 数据分析和存储的背景及意义

现在社会对数据的看重程度与日俱增。最开始，我们只是存储当下的数据。再之后，我们意识到了数据背后的价值

## 数据领域现状及存在的问题

### 数据分析存在的问题

### 数据分析的难点

### 数据存储存在的问题

数据存储技术发展以来，

### 数据存储的难点

## 本文研究内容及目标

### 研究内容

### 研究目标

## 本文组织结构

# 数据分析

本章具体描述了数据分析：介绍了数据分析的基本概念，并引出针对不同类型数据的分析方法。

## 数据分析概述

### 数据分析的基本概念

### 数据分析方法

## 针对不同类型数据的分析方法

### Txt数据

### Excel格式

### Xml/json格式

## 本章小结

# 数据存储

本章是全文的重点章节，阐述和分析了数据存储方面的几种设计思路。首先简单介绍了关系型数据库，对象型数据库，图数据库，nosql数据库。随后分别举出例子，分析其特点和优劣，对比分析其性能。最后将系统数据存储的构建过程完整的进行了描述，并将部分系统功能运行的结果作了展示。

## 多种数据库概述

在数据存储技术发展的过程中，我们经历了三个时代。第一代数据库系统主要指的是网状和层次数据库。他们分别以网状和分层的数据结构为基础，初步解决了数据的集中和共享问题。但它们逻辑独立性差，可移植性低，使用也非常繁琐。第二代数据库即为关系型数据库。也就是建立在关系模型基础上的[数据库](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93)，借助于集合代数等数学概念和方法来处理[数据](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE/33305)。第三代是新型数据库，主要指各种NoSQL数据库。以下我们将结合第二代和第三代数据库分析食物领域下的数据库设计。

## 关系型数据库

第一代数据库出现后，不到五年就进入了第二代。20世纪70年代即为关系型数据库理论研究和原型开发的时代。关系型数据库的一大进步就是引入了关系模型。关系模型的提出，是数据库发展史上具有划时代意义的事件。关系模型由关系数据结构、[关系操作](https://baike.baidu.com/item/%E5%85%B3%E7%B3%BB%E6%93%8D%E4%BD%9C)集合、关系完整性约束三部分组成。我们将数据规范化处理为二维表，并将数据间的联系以键的形式联系在二维表之间，数据操作则被处理为了关系表格的合并，连接，选择，投影等。

相比第一代数据库，关系型数据库结构更贴合逻辑世界，更容易理解。关系模型的存在使得数据结构简单清晰，数据独立性强。数据完整性保证了数据的一致性和可维护性，同时SQL语言方便了程序员操作数据库。这些优点都使得它风靡一时，虽然当后期业务场景越来越复杂，需求变更速度越来越快，关系型数据库渐渐体现出了它对复杂类型数据的无能为力和处理事务时的繁琐复杂，但依然有着极强的生命力。

### MySQL

MySQL是一个关系型数据库管理系统，由瑞典MySQL AB 公司开发，目前属于 Oracle 旗下产品。MySQL 是最流行的关系型数据库管理系统之一，因为它开源且社区支持强大，深受中小型网站和个人开发者的欢迎。

MySQL系统特性大致有：多语言，多平台支持；支持多线程，支持自我修复式集群；为多种编程语言提供了 API。如C、C++、Python、Java、Perl、PHP、Eiffel、Ruby,.NET和 Tcl 等。而且，既能够作为一个单独的应用程序应用在客户端服务器网络环境中，也能够作为一个库而嵌入到其他的软件中。

但MySQL近几年也在慢慢被放弃，主要原因可以归结为： MySQL 不如其它关系型数据库管理系统那样成熟； MySQL 的性能相比竞争对手还是太弱；MySQL目前是 Oracle 所有的，而不是社区驱动的。

这里我们把MySQL作为场景数据存储最基本的一种方法。之后我们会以MySQL为基础对比分析其他数据库的优劣。首先，食物领域的场景如下：。食物属性可以对应构建如图一数据结构的表。而测试员打分情况可构建为图2。可以看到，在建模的过程中我们只需要分析数据并以行列的形式保存即可。当需求发生变化，食物属性增加时，我们需要在相关的表中都新增字段，同时录入新的相关数据。哪怕不同食物增加的属性不一样，我们也需要把所有增加的属性都赋予到数据上，值得注意的是，这有可能产生稀疏表。

## 对象型数据库

虽然关系型数据库在存储管理方面性能强大，但由于以上提到的结构僵化等问题，80年代出现了面向对象数据库技术，其中面向对象数据库系统宣言是这个思想振聋发聩的宣告。其将面向对象性数据库管理系统特性归总为必备特性（复杂对象，对象标识，封装性，类型，继承性，重载，可扩充性，计算完备性，辅存管理，并发性，恢复，即时查询）和可选特性（多重继承，类型检查和推理，分布式，类型系统）。面向对象的数据库系统在逻辑上和物理上从面向记录上升为面向对象、面向可具有复杂结构的一个逻辑整体。允许用自然的方法，并结合数据抽象机制在结构和行为上对复杂对象建立模型，从而大幅度提高效率，降低用户使用复杂性。后来对象型数据库思想不断发展分化，数据库发展趋势大致如：面向对象数据库->对象关系型数据库->对象代理数据库。

面向对象数据库主要是数据库与面向对象语言的结合。由以下几部分组成：对象模型（主要包括对象的特性，对象间的联系），对象描述语言：ODL（对象定义语言），OIF（对象交互格式），对象查询语言: OQL(对象查询语言。)，对象语言绑定（常见的是C++,JAVA。目的主要是实现ODL和ODMG缺乏的OML）。这里我们先提一下对象模型，有种说法是“面向对象数据库模式是类的集合”。面向对象的数据模型提供了一种类层次结构。在面向对象数据库模式中，一组类可以形成一个类层次。一个面向对象数据库可能有多个类层次。在一个类层次中，一个类继承其所有超类的全部属性、方法和消息。其次，对象语言绑定。这个特点出现的原因我认为主要是面向对象特点决定的，绑定可以方便的加入面向对象的功能，由用户灵活的实现定义和操作对象。但我认为引入语言绑定后，为了满足对多语言绑定的支持，还是牺牲了一些良好的特性。

接下来，我们在数据建模上与关系型进行对比分析。假设现在面对的对象是个复杂对象，它结构复杂，维度多样，且对象之间有较多组合继承关系。针对这种复杂对象建模时，1.关系数据库需要为所有数据在每种尺度下建立自己的表，访问时，通过不同尺度的查询条件，查询不同的表来返回信息。往往同种数据在不同尺寸下的表会有大量数据冗余，这加大了同种数据一致性维护难度。而如果通过集成，可以很好的降低这一难度。2.其次，我们无法在关系型数据库中实现对数据的计算，如添加c=a\*b，c属性必须也要建立一个字段，这是不必要的。对象型可以在表上定义一个计算的函数或者方法来实现这类查询，这是关系型所不能支持的。3.一些结构关系复杂的数据类型无法在关系型中储存，而是通过基础的数据类型去模拟记录，这往往导致数据库表单的设计过于复杂，而且需要大量的join来实现复杂数据的表达，而对象型为了满足多尺度的要求，可以对统一对象创建多个类并根据类层次结构来进行管理。

分析其组成并结合表现，我们可以得出结论，面向对象型数据库支持复杂对象，且建模方法自然，易理解。对象具有独立对象标识，使得查询速度极大加快。对象的封装性实现了良好的透明度，使用者不需知道操作的细节，加快了效率。因为面向对象数据库对复杂对象的表现优异，它广泛应用于：多媒体，工程数据，地理信息等复杂信息管理系统。但由于它没有原型数据库做基础，完全从零开始实现一个对象型数据库，虽然足够灵活，但也开发费时费力，开发者需要手动实现数据库功能，自主开发需要的扩展数据类型，同时还要考虑性能优化问题，而且与其他数据库的转接兼容也相对麻烦。所以这种思路虽然很强大，但受到的限制还是太多：缺乏数学模型的支持，复杂难用，成熟度低，最后没有撼动关系型的地位。

之后，人们在对象型数据库的基础上建立了对象关系型数据库。对象关系数据模型是从关系数据模型出发，指向对象思维，以关系数据库和sql为基础的扩展关系模型，具有扩展复杂数据类型和允许用户自定义函数功能。它具有面向对象特性，收通用性强的sql3标准支持。对象关系型数据库管理系统既满足了传统关系型数据库用户的要求，同时又能支持复杂数据管理需要的应用，具有强大的查询语言功能。关系数据库只能做到对复杂数据的存储和对简单数据的查询。而对象关系数据库的最大功能是能做到对复杂数据的查询且速度很快，具有可扩充性。对象关系型数据库的典型有oracle数据库。

### Db4o

面向对象数据库的概念提出后，对象数据库管理系统为面向对象编程语言增加了持久的概念。最早的商品化 ODBMS 出现在 1986 年，是 Servio 公司和 Ontos 公司推出的。后来，九十年代 Object Design、 Versant 、 Objectivity 、 Poet 、UniSQL等公司也加入了这个开拓行列。然而目前为止成熟的对象型数据库还是比较少，而且开源的更少。2004年，**db4o作为免费的开源ODBMS（纯面向对象数据库管理系统）发布**，2005年，db4o首先将Native Queries 作为面向对象的数据访问API 来实现，该API完全依赖于编程语言（Java / C＃）本身，由此它成为了100%原生数据库。目前国内相对比较流行的也是db4o,虽然很可惜这个项目已经在2014年被官方停止维护。

Db4o 提供两种运行模式，分别是本地模式和服务器模式。本地模式是指直接在程序里打开 db4o 数据库文件进行操作，而服务器模式则是客户端通过 IP 、端口来访问服务器。一般来说，javeEE更适用于服务器模式（这里假设api语言用的是java），嵌入式开发更适用于本地模式。它有几大特点：一：轻量级。实现该数据库只需要下载400k左右的jar包或者dll文件，并将其引入到系统中。这个特点也让它非常适用于嵌入式系统。二：原生。因为其为原生数据库，开发者不需下载管理工具，不需学习新的数据库语言，可以直接用编程语言管理数据库。三：多平台支持。db4o 支持从 Java 1.1 到 Java 5.0，此外还支持 .NET 、 CompactFramework 、 Mono 等 .NET 平台，也可以运行在 CDC 、 PersonalProfile 、 Symbian 、 Savaje 以及 Zaurus 这种支持反射的 J2ME 方言环境中，还可以运行在 CLDC 、 MIDP 、 RIM/Blackberry 、 Palm OS 这种不支持反射的 J2ME 环境中。四：性能优秀。下图为db4o 官方公布的基准测试数据，db4o 我们可以非常惊讶的发现在读写查删时，它比采用 Hibernate/MySQL 方案速度高出几到几十倍。



图2. db4o 官方基准测试数据

接下来我将使用db4o，结合我们的食品场景进行数据库设计。因为db4o是原生数据库，我们拟采用java进行数据库开发。首先，在设计类时，我们可以发现面向对象的思想给方便面复杂的属性管理提供了不少便利。我们可以设计两个父类：用户，方便面。接下来使用继承的思路，把人分为测试员（测试用的机器为了方便也可以放在这个子类里，作为人的特性被设置为null即可，主键则专门设置规则）和用户两个子类。方便面和人之间存在两种关系：一：用户—吃&评价是否喜欢🡪方便面，二：测试员从多个角度（时间，定性，定量，数据）--打分🡪方便面。类模型可以简单描述为：

|  |
| --- |
| 此处有图或者代码 |

建立类并将数据存入后，当我们需要新增食物属性时，我们可以直接在食物的类里定义新的属性和对应的方法。当我们需要查询食物信息时，我们有三种查询方式：QBE（Query by Example）、NQ（Native Queries）、SODA（Simple Object Database Access）。QBE 标准是由IBM提供的，是一种比较传统成熟的查询。SODA是 db4o 最底层的查询 API，虽然写起来较麻烦，但当遇到动态生成查询的时候 SODA 实在非常高效。db4o 更推荐使用 NQ 进行查询，NQ 方式支持原生语言，也就是说，我们可以直接使用java或者.NET来写数据库查询语句，这无疑非常方便。但我们也需要注意，NQ的本质是转换接口，数据库实现内部还是会设法把NQ转换为SODA。这种转换大多数情况下是成功的，但当有些查询表达式的流向图非常难于分析时，db4o 将不得不实例化一些持久对象来真实地运行 NQ 表达式。这个过程会影响性能。我在示例开发中使用的是最特色的NQ方法。但这里对类间的联系并没有很直接的体现。

### PostgreSQL

自从MySQL被Oracle收购以后，PostgreSQL逐渐成为开源关系型数据库的首选。从苹果即为MySQL 转向 PostgreSQL的代表厂家之一。PostgreSQL是一个自由的对象关系数据库管理系统。PostgreSQL支持大部分SQL标准,支持几乎所有 SQL[构件](https://baike.baidu.com/item/%E6%9E%84%E4%BB%B6)（包括[子查询](https://baike.baidu.com/item/%E5%AD%90%E6%9F%A5%E8%AF%A2)，事务和用户定义类型和函数）并且提供了许多其他现代特性：复杂查询、外键、触发器、视图、事务完整性、MVCC。同样，PostgreSQL 可以用许多方法扩展，比如， 通过增加新的数据类型、函数、操作符、聚集函数、索引方法、过程语言， 它可以获得非常广阔范围的开发语言绑定 （包括 C，C++，[Java](https://baike.baidu.com/item/Java/85979" \t "_blank)，perl，tcl，和 python）。并且，因为许可证的灵活，任何人都可以以任何目的免费使用、修改、和分发 PostgreSQL，不管是私用、商用、还是学术研究使用。

## 图数据库

### 概述

作为NoSQL的一员大将，图数据库近几年被广泛的讨论，其使用者也逐渐增多。不得不说在研究过它的思想后，我也爱上了这种形象自然的建模思路和设计方法。

首先，让我们想想什么是图？图就是节点和联系的集合，其中比较主流的图模型主要是**属性图**，资源描述框架（RDF）三元图和超图。拿最流行的属性图举例，它主要有两个特性。一：包含节点和联系，节点有自己的属性。二：联系有名字和方向，也可以有属性。这个模型在我看来是如此的通用，以至于我认为生活中任何东西都可以用图来建模表示。既然它是一个符合人们思维习惯的建模思路，那它必然也可以抽象出来用于计算机世界里的数据存储。而我们要讨论的图数据库就是建立在这样的基础上的。图数据库使用节点，边/关系(relationship)和属性。节点代表实体，关系代表实体间的联系，属性是节点的相关信息。引用维基百科对图数据库的定义“In [computing](https://en.wikipedia.org/wiki/Computing), a graph database (GDB) is a [database](https://en.wikipedia.org/wiki/Database) that uses [graph structures](https://en.wikipedia.org/wiki/Graph_(data_structure)) for [semantic queries](https://en.wikipedia.org/wiki/Semantic_query) with [nodes](https://en.wikipedia.org/wiki/Node_(graph_theory)), [edges](https://en.wikipedia.org/wiki/Edge_(graph_theory)) and properties to represent and store data. A key concept of the system is the graph (or edge or relationship), which directly relates data items in the store.”

知道了什么是图数据库后，我们就可以对比分析图数据库建模的方法和特点。传统的关系建模流程大致是：草图，er图，映射成表。在过程中需要把数据间的联系以逻辑的形式存储下来。同时应该使用范式，约束等技术整理模型结构，减少数据库中数据冗余，增进数据。也就是规范化。如此，建模过程看上去已经完成了，但实际项目中，需求总是变化的，所以实际生活中常常会继续使用**反规范化技术**。也就是说，在实现关系型解决方案的过程中，为了适应关系模型，我们对物理模型强加了很多变化，如为了获得查询性能，人为制造重复数据和重复结构。这些变化使得概念模型和数据真实的物理布局之间产生了差异。这种概念上的失调，使得将业务需求的变化转化为底层的稳固的关系结构变得非常复杂和繁琐。同时，需求的大量变更，使得**迁移**花销巨大和风险极高。总之，在实际的项目过程中，我们需要经历：设计-规范化-反规范化的过程。

而图建模就去掉了使用复杂数据管理代码来规范化和反规范化数据的这一步骤。我们可以很轻松的将复杂的层次结构抽象出来，又因为其建模思路和人的思维习惯相符，整个过程非常的易理解而又高保真。虽然在建模初期，常见一个陷阱是错误的将本应是节点的数据用联系+联系的属性表示出来，反而丢失了节点本身的内容。但这种错误随着对图的熟悉会越来越少，这是一个设计上容易遇到的问题，但错误并不在图数据库。同样值得注意的一点是：不要小看图的天然可扩展性。相比关系型需要不断确认逻辑设计，图数据库在修改实体和联系上方便了太多，例如，当我们需要新增实体并添加于已有实体间的关系时，我们不需要修改本来实体的内容，只需要新增节点然后添加一条指向已有节点的边即可。

这样分析了图建模的方式后，我们可以总结出图数据库的两大优势： 1.**灵活性**，可扩展性高。不必要一开始就把整个系统的结构设计出来，可以在开发的过程中低成本的增删节点和联系。2.**敏捷性**。图数据库不需要**schema**,所以它缺少以schema为导向的数据管理机制。他的管理通常作用于编程方式，利用测试来驱动数据模型和查询，以及依靠图来断言业务规则。它提供了数据模型的平稳演化，数据迁移和反规范化很少会成为问题，新的事实和新的组件会成为新节点和新联系。而优化关键性能访问模式通常需要在两个原本有中介连接的节点间引入一个直接联系。适用于敏捷开发。

那么，图在性能方面的表现足够好吗？让我们继续分析模型建好后的操作。

关系模型使用数据中的信息关联数据。例如，当需要查找所有电话号码包含区号“311”的用户时，它通过遍历整个表，在电话号码字段中查找字符串“311”。这在大型表中是一个耗时的过程，因此关系数据库提供了数据库索引的概念，以提高查询速度。

相比之下，图形数据库直接存储记录之间的关系。同样是查找某用户的电话号码，图数据库使用户有直接指向电话记录的指针，而不是通过用户id查找用户表里电话号码字段的内容。也就是说，选择一个用户后，可以通过指针直接得到对应的电话号码，而不需要搜索用户表来查找匹配的记录。这可以消除昂贵的连接（join）操作。这种优势在进行深度较大的搜索时更加明显。当系统需要查找和用户a的电话号码前六位相同的其他用户最近在某地的购买的物品的生产地的最普遍的姓时，关系型需要多次搜索匹配，而图数据库只需要查找节点，顺着边继续进入下一个深度即可。从技术上讲，图数据库共用了O（log（ n））+ O（1）时间，即大致相对于数据大小的对数。而关系版本将是多个 O（log（ n））查找。由此我们可以得出结论：图数据库具有良好的查询**性能**。因为哪怕在数据集增大的情况下，查询也总是和图的一部分相关，因此，每个查询的执行时间只和满足查询条件的那部分遍历的图的大小而不是整个图的大小成正比。（使用免索引链接，图数据库可以将复杂的join查询转化为快速的图遍历，节点和关系只有在结果迭代器需要访问它们的时候才会被遍历并返回，对于大规模深度遍历而言，这极大地提高了性能。因此无论数据集有多大，都可以维持毫秒级的性能。）

总结以上的分析。图数据库适用于处理**半结构化**的，紧密**关联且变动性大的**的数据。现实中，图数据库典型使用的领域如[语义网和RDF](http://en.wikipedia.org/wiki/Semantic_Web)、[LinkedData](http://en.wikipedia.org/wiki/Linked_Data)、[GIS](http://resources.esri.com/help/9.3/arcgisengine/dotnet/e084da94-d4f7-4da7-86ed-7df684ff2144.htm)、[基因分析](http://code.google.com/p/pygr/)、[社交网络数据建模](http://blog.neo4j.org/2009/09/social-networks-in-database-using-graph.html)、[推荐算法](http://wiki.github.com/tinkerpop/gremlin/linkeddata-sail#recommendation)，地理空间（基于位置的服务），网络拓扑以及[其他领域](http://wiki.neo4j.org/content/Neo4j_In_The_Wild)。不适用更新全部或某个子集的实体。

### 图数据库实例之neo4j

在图数据库领域，现在比较流行的数据库有：[Neo4j](http://www.neo4j.org/) - 开源的Java属性图形模型。[AllegroGraph](http://www.franz.com/agraph/allegrograph/)，闭源，RDF-QuadStore。[Sones](http://www.sones.com/) - 闭源，关注于.NET。[Virtuoso](http://www.openlinksw.com/) - 闭源，关注于RDF。[HyergraphDB](http://www.kobrix.com/hgdb.jsp) - 开源的Java超图模型。其他的例如 [InfoGrid](http://infogrid.org/)、Filament、[FlockDB](http://github.com/twitter/flockdb)等。实际开发中，我使用的数据库是Neo4j。Neo4j是一款强健的，可伸缩的，用Java实现、完全兼容ACID的高性能图数据库。数据以一种针对图形网络进行过优化的格式保存在磁盘上。Neo4j的内核是一种极快的图形引擎，具有数据库产品期望的所有特性，如恢复、两阶段提交、符合XA等。自2003年起，Neo4j就已经被作为24/7的产品使用。Neo4j既可作为无需任何管理开销的内嵌数据库使用；也可以作为单独的服务器使用，在这种使用场景下，它提供了广泛使用的[REST](http://wiki.neo4j.org/content/Getting_Started_REST)接口，能够方便地集成到基于[PHP](http://github.com/onewheelgood/Neo4J-REST-PHP-API-client)、.NET和JavaScript的环境里。开发者可以通过[Java-API](http://api.neo4j.org/current/)直接与图形模型交互，随着开源的发展，目前它也支持[JRuby/Ruby](http://github.com/andreasronge/neo4j)、[Scala](http://github.com/ept/neo4j-scala-template)、[Python](http://components.neo4j.org/neo4j.py/)、[Clojure](http://github.com/JulianMorrison/neo4j-clojure)等其他语言。而它的这两个版本都可以做成集群，从而（使用主从复制）实现高可用性和水平读扩展，甚至我们可以将两个版本进行混合，在数据库级执行。

Neo4j的几个特点如：使用多种算法支持路径搜索，使用键值和关系进行索引，为读操作进行优化，支持事务（用 Java api），使用 Gremlin图形遍历语言，支持 Groovy脚本，支持在线备份，高级监控及高可靠性支持使用 AGPL/商业许可。

接下来我们尝试**将图数据库与场景结合起来**。在过程中继续分析其优劣。

根据拿到的资料，我们简单拿方便面多维度属性和定量描述举例。具体的场景如下：有品牌A,B,C生产了口味a,b,c,d等n种方便面，每种方便面有自己表层物品属性，如编号，生产日期，规格，风味等。厂商为了更好的分析食品口味，研究市场对口味的喜好，请了m名测试员从色泽，光滑性，复水性，韧性，黏性，耐泡性等多个角度进行打分。同时，在口味方面，厂商进行了细化研究，通过多种评价标准，如强度等级，麻度等级，斯科维尔指数，描述词，根据方便面在不同时间的表现进行评级，得出了味道的时间强度评价和时间频率记录。当食物投放市场后，厂商进行了反馈调查，拿到了o位用户针对n种方便面的喜恶数据。

分析以上场景，我认为大致可以分为三个模型。模型一：方便面的属性记录。这里主要包括表层属性，分数情况。模型二：测试员分析过程中的打分记录。这里数据模型简单，数据量大。需要注意的是如何把时间因素记录得当。模型三：用户的反馈记录。这里可以为后续的推荐功能，产量设计提供数据支持。

分析完数据模型后，我们可以结合图数据库的特点来看如何设计数据存储模型。在以上提到的食物领域，我认为源数据的主要特点是结构简单，数据量大，变动性高。那么，个人比较遗憾的发现，图数据库最大的优点——联系在这个场景里并没有很好的得到表现。因为食物之间并不会有很多的事件交集，显然无法像社交网络那样完美的体现图数据库中联系一等公民的地位。

那么，这就意味着图数据库不适合了吗？我认为也不是这样的。虽然事物间的联系不多，但从以上描述中可以清晰看到一个食物的属性是很多的。特别是在口味这里，作为食物最重要的特性，我们可以很明显的看到厂商在这里用了多种角度分析，在味觉粒度和时间粒度上都做了细化。也就是说，一个物品的属性之间是有联系的。这样无疑也可以让图数据库发挥特长。而且，在反馈分析这里，我认为图数据库是非常适合的。不同人对不同事物有不同的喜好。这个过程可以看出有3部分的联系。人和食物（是否喜爱），人和人（口味是否类似），食物和食物（是否是统一风格大类）值得注意的是，我们在介绍图数据库时提到过它擅长的领域，正如当时分析的，做推荐系统图数据库具有天然的优势，它带来的对性能的提高不容小觑。

同时，正如之前提到的，本情景下，需求变更快，源数据的变动性高，换句话说就是对数据模型的可扩展性有很高的要求。如果整体模型太过僵化，那么食物每增加一种评价方式，每发生一次某属性的突然增删，测试员每提供不完全统一的数据，数据库每涉及到迁移（无论大小），无疑都是一次痛苦的重新设计，系统的维护成本升高不说，万一由于操作者（开发者）经验问题，很可能导致冗余数据/丢失数据/模型臃肿等恶性情况的发生。这大大增大了项目的风险，也有很大概率降低系统的性能。

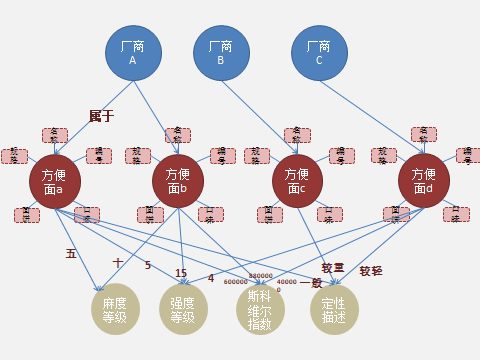
考虑到如上的可能性，图数据库瞬间变得非常亲切可人了。它天生良好的可扩展性无疑非常适合于需求变更快的敏捷开发模式。它没有那些复杂的，恼人的物理模型，概念模型转换。建立表结构是如此的简单，在原有表上新增属性和联系只需新增节点并画上线即可。实在是非常的高效方便。

再其次，在数据的查找方面，图数据库更有的可说。介于它的查询只发生在图的局部，不需要遍历整个库，而且由于它联系分明，在深层次的数据查询方面比传统的好上太多，传统关系型查询性能随深度增加是指数级减小的，而对于图数据库，这基本是线性的，当深度大于3时，其中的差距实在需要引起注意。

然而，虽然我这么喜爱这种“又聪明又懒惰”的图数据库建模方式，总体来说，它也还是有表现不佳的地方。这里主要体现在对时间流的掌控上，图的亮点在于实体间的联系，而时间是流式的，每个时间都有新的不同的数据，个人认为，这个变化的过程用图数据库来记录实在是不太合适的，图数据库表示时间一般是两种方法，时间轴数和链表。放在本例的场景下都不是很合适。

这里画出图数据库的模型设计：

模型一：



## 文档型数据库

### mongodb

## 不同数据库的性能对比

## 系统中的数据结构与其表现

## 本章小结

本章，我们介绍了数据库技术发展的后两代成果——关系型数据库，对象型数据库，对象关系型数据库及NoSQL数据库中的图数据库和文档数据库，分别以MySQL，db4o，postgresql，neo4j和mongodb为例。在每个数据库分类里，我们介绍了他们的来源，特征，优劣。并结合食物领域场景进行了简单的建模分析。对比后，我们可以看到：目前数据存储的方式百花齐放。大致可以将数据库类型分为关系型，对象型，键-值型，列型，文档性，图型。虽然不同的数据库成熟度不同，使用者量级不同，底层数据结构不同，但没有哪个数据库是最好的，只有哪个数据库在某种情况下是最合适的。在选择最终使用哪个数据库的时候，我们需要考虑的不只是数据库的类型，还要考虑数据的体量，场景的复杂性，可扩展性等等。例如当我们构建社交网络的场景时，似乎第一反应就是图数据库，毕竟它是那么完美的适应于这种场景。但我们同样需要考虑当数据量太多，我们是否需要加入“大数据”的实现，引入Rick或HBASE。所以我有一个想法，我认为数据管理的未来会是多持久并存模型，即一个项目中使用多种数据库。这样就可以最大化利用每种数据库类型的优势。但这个想法意味着不同数据库间必须要有兼容彼此的接口，可能只能等一个足够有话语权的组织制定规范后才会实现。

# 食物项目系统的设计与实现

本章主要介绍食物项目系统的设计与实现，包括系统整体设计、数据库设计以及系统内各功能模块的设计与界面展示。

## 系统整体设计

### 开发环境简介

本文设计的食物项目系统搭建于Ubuntu环境上，过程中主要使用的语言包括HTML,JavaScript，PHP，Python,Java，使用的数据库如第三章提到的，包括MySQL，postgresql，db4o,neo4j，MongoDB。

Ubuntu是最流行的开源Linux操作系统。它免费，安全，用户友好，可高级定制，有强大的社区支持。作为服务器，它相比Windows便宜，稳定，且性能更高。因此，在搭建食物项目系统时，我选用亚马逊云配置了Ubuntu环境，版本为Ubuntu 16.04.4，Linux版本为4.4.0-1052-aws (buildd@lgw01-amd64-031)。远程连接的工具是mobaxterm。

食物项目系统的前端和后台语言主要是HTML和PHP。虽然前端开发技术日新月异，但它不是本文讨论的重点，因此本系统我们采取最古老的HTML结合JavaScript。而PHP是一种语法简单的通用开源脚本语言。它广泛应用于web系统，因为它可以嵌入到HTML里，相比其他语言开发实用方便，并且，它与MySQL合作默契，几乎成为小项目个人开发者的标配。

数据库这里我采用多种存储方式，以MySQL为切入点，既可通过更改后台接口切换数据库的选用，也可将多个数据库结合起来使用。这里XX因为功能局限，只作为示范通过接口切换使用。而XX,XX,XX则在本系统中结合使用。

### 系统整体架构

系统整体分为三部分构建。

一：数据上传和分析。

二：数据存储。

三：数据操作。

## 数据分析

## 数据库设计

### Mysql设计

### Neo4j设计

## 功能模块设计

### 数据可视化

这里的数据可视化主要指数据操作，包括增删改查。四种操作对应的页面分别如:

## 本章小结

# 总结与展望

本章对全文的主要工作和创新点作了总结，并提出需要进一步研究和优化之处。

## 本文总结

本文通过对食物领域方便面数据的研究，探讨了不同格式不同内容的数据应该使用怎样的分析和建模方式进行存储，以方便对数据的操作，甚至从已知数据（信息）中推测出未知。

### 本文的主要工作

### 本文的主要创新点

## 展望

# 致谢

XXX。

# 参考文献

1. 林鸿飞,杨元生.用户兴趣模型的表示和更新机制.计算机研究与发展，200239(7):838-842.
2. 胡学联,潘金贵,李俊,张灵玲. 一个个性化的信息搜集Agent的设计与实现.软件报,2001,12(7):1074-1079.
3. FragoudisD.User Modeling in Information Discovery:An overview.InProceedings of Advanced Course on Artificial Intelligence,1999(ACAI99), July,1999, Greece.
4. Balabanovie M, Shoharn Y.Learning Information Retrieval Agents:Experiments with Auto-mated Web BrowsingIn:Proceedings of the AAAISpring Symposium Series on Information Gathering from Heterogeneous, Distributed Environments, March, 1995:13-18.
5. 附录：部分源程序清单