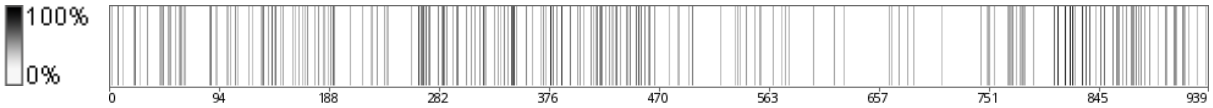
PaperPass旗舰版检测报告
简明打印版

比对结果(相似度):

总　体：14% (总体相似度是指本地库、互联网的综合对比结果)
本地库：14% (本地库相似度是指论文与学术期刊、学位论文、会议论文、图书数据库的对比结果)
期刊库： 7% (期刊库相似度是指论文与学术期刊库的比对结果)
学位库：11% (学位库相似度是指论文与学位论文库的比对结果)
会议库： 2% (会议库相似度是指论文与会议论文库的比对结果)
图书库： 6% (图书库相似度是指论文与图书库的比对结果)
互联网： 1% (互联网相似度是指论文与互联网资源的比对结果)

报告编号：5CB0940A8B2037TMW
检测版本：旗舰版
论文题目：移动套餐智能检测系统的设计与实现
论文作者：阿里
论文字数：25628字符(不计空格)
段落个数：604
句子个数：939 句
提交时间：2019-4-12 21:35:06
比对范围：学术期刊、学位论文、会议论文、书籍数据 、互联网资源
查询真伪：http://www.paperpass.com/check

句子相似度分布图:



本地库相似资源列表(学术期刊、学位论文、会议论文、书籍数据):

1.相似度： 1% 篇名：《软件测试方法和技术》
来源：书籍数据 清华大学出版社 2005-7-1

互联网相似资源列表：

1.相似度： 1% 标题：《基于Python的机器学习实战：Apriori ...》
https://www.cnblogs.com/90zeng/p/apriori.html
2.相似度： 1% 标题：《Apriori算法的python实现：通过限制候...》
https://blog.csdn.net/qq_35882901/article/details/80172124
3.相似度： 1% 标题：《第五次毕业设计任务书(4.2--4.9)》
http://www.mamicode.com/info-detail-1755344.html
4.相似度： 1% 标题：《关联规则的挖掘问题——机器学习 - Horace...》
https://blog.csdn.net/horacehe16/article/details/80085007?utm_source=blogxgwz6
5.相似度： 1% 标题：《关联规则的挖掘问题——机器学习 - Horace...》
https://blog.csdn.net/horacehe16/article/details/80085007
6.相似度： 1% 标题：《Apriori的Python实现 - 难 ...》
https://blog.csdn.net/qq644262163/article/details/74857006

全文简明报告:

为了满足不同的用户群体的需求，中国移动的套餐业务更新频繁，每个月都会推出很多个性化的套餐服务，当前，中国移动创建套餐的方式是手动编辑上百行SQL语句之后插入到数据库中，一旦SQL语句中出现错误，将会导致重大损失，而人工检测SQL语句十分依赖经验丰富的员工，且很难发现SQL语句中的所有错误，{46%：本文针对该问题提出了一种基于数据挖掘的自动化检测SQL语句的解决方案。}

本文通过协同过滤算法和关联规则挖掘算法分析历史套餐的SQL语句，对创建新套餐的SQL语句是否存在错误进行预测。其中利用协同过滤算法分析套餐与创建该套餐的SQL语句中涉及到的数据库表之间的关系，对可能遗漏或者多余的数据库表进行预警。为了能够检测数据库表内具体数值是否出错，本文利用关联规则挖掘算法对SQL语句进行关联分析，得到相应的频繁项集和置信度关系，并根据获得的频繁项集和置信度关系对新创建套餐的SQL语句中该数据库表的关键属性是否有错进行检测。

{67%：本文介绍了SQL自动化测试系统的开发过程。}本系统使用Python开发，通过web应用的形式实现，前端使用Bootstrap，后端使用Django进行开发。

1 概述

1.1 论文选题研究的背景及意义

{42%：随着近年来移动通信技术不断发展和智能手机普及度的不断提高，移动端通信业务的用户数量和通信流量大大增加，移动通信的市场竞争非常激烈。}为了扩大市场份额和更好的服务用户，中国移动的套餐业务更新频繁，每个月新上线的套餐数量都可以达到20左右，春节前后和暑假期间属于套餐更新的高峰，数量甚至可以超过50个。

为了满足不同的用户群体的需求，中国移动推出了很多个性化的套餐服务，现有的套餐编辑页面操作复杂，每当创建新套餐时都需要对空白界面进行编辑，需要创建套餐的员工填写上百个属性参数，而当新业务出现，涉及到了编辑页面上未包含的属性时，则需要对数据库直接操作进行手动添加。由于很多套餐之间存在相似关系，很多情况下新创建的套餐与数据库中已有的历史套餐可能只有很小的区别，只需要修改几个数值就可以得到新套餐，如果使用编辑页面进行操作，就相当于套餐间相同或相似的地方没有得到复用，反而需要创建套餐的员工从零开始配置，这样繁琐的操作导致了创建套餐的员工放弃使用图形化界面，{49%：反而直接使用手动编辑SQL语句进行数据库操作的方式来创建套餐。}{68%：如果使用手动编辑SQL语句来操作数据库，}创建套餐的员工就可以找相似的历史套餐，{42%：使用历史套餐的SQL语句并更改其中的部分参数再插入到数据库中，}这样的创建方式虽然一定程度上降低了操作的复杂度，但是也存在对操作人员技术要求高、风险大、缺乏有效的验证手段等问题。

目前中国移动为了保证套餐属性修改的完善性，只能通过人工检测的方法对待插入数据库中SQL语句进行检索和识别，{40%：但是由于创建一个套餐会涉及大量的数据库表，而每个数据库表中又包含大量的属性，}这就使得创建套餐的SQL语句非常复杂，其中包含的具体数值的数量十分庞大，如果只通过人工进行套餐属性关联性检索会耗费大量的人力物力，而且人工检测很难发现SQL语句中全部的错误，导致了业务质量难以保证，造成了用户流失和经济损失的潜在风险。

总的来说，由于移动套餐复杂多变且经常需要添加之前套餐中未出现过的业务的特殊属性，中国移动创建新套餐难以使用图形化编辑页面进行操作，现阶段只能通过SQL语句直接对数据库进行操作，而这样的创建套餐的方式存在对套餐配置人员技术要求高、风险大、操作

复杂的问题。 {53%：针对这个问题，本文提出了一种基于数据挖掘的SQL语句检测自动化的方法。} 通过本文所描述的方法，可以对创建套餐的 SQL语句进行自动检测，并对套餐中可能存在的错误进行预警， 将这些可能存在的错误展示给创建套餐的员工，员工根据这些错误的不同级别对 SQL语句进行修改后再插入到数据库中。 检测的过程不需要人工参与，可以降低人工成本和创建套餐的复杂度，提高中国移动套餐的业务质量。

本文所描述的方法对 SQL语句进行处理和分析，从中挖掘有效的数据，并利用协同过滤算法和关联规则挖掘算法对此进行分析， 推断出 SQL中可能存在的错误。 本论文选题目的在于充分利用现有的数据挖掘技术，对 SQL语句进行自动化处理分析， 自动检测出SQL语句中可能存在的错误，改变人工检测 SQL语句费时费力的现状。

1.2 国内外现状分析

虽然软件测试行业目前在国内仍处于起步阶段，但是随着国家政策的支持和软件行业的蓬勃发展，软件测试也正在成长为一个新兴的行业。 {42%：目前国内越来越多的互联网公司开始对软件测试逐渐重视了起来，软件测试的地位也得到不断提高。}

{50%：软件测试主要分为传统的手工测试和自动化测试两种，对于软件行业较为发达的西方国家，} {41%：传统的手工测试的主导地位已逐渐被自动化软件测试所取代，而由于起步较晚，} {60%：所以目前国内的软件测试仍然以传统的手工测试为主。} 中国移动创建套餐复杂度高和风险大一定程度上反映了目前国内大多数互联网公司的现状： {45%：软件测试的自动化程度低，而人工进行软件测试又有难度高且易出错的问题。}

SQL语句检测与传统的软件测试不同，传统的软件测试考虑的问题主要是分支覆盖率以及行覆盖率等代码覆盖问题， 而 SQL语句的检测考虑的则是每一条记录中的数值是否存在错误，因此不能用常规的测试思路去处理， {48%：本文通过基于数据挖掘的协同过滤算法和关联规则挖掘算法来对 SQL语句进行检测。} {45%：对于数据挖掘技术，目前国外有较多的研究机构和组织，关于数据挖掘技术的刊物和网站也有很多，数据挖掘技术已经得到了广泛的应用。} {51%：而国内对于数据挖掘的研究起步稍晚，专门从事与数据挖掘的人主要分布在大学和极少数的大型互联网公司中，缺乏有关数据挖掘的实际应用。} 协同过滤算法目前主要应用于个性化推荐系统中，无论在国内还是在国外都得到了广泛的使用，然而在软件测试领域协同过滤的应用并不多。 由于可以用来发现大数据内隐藏的关键信息，关联规则挖掘算法的应用领域十分广泛，涵盖了国内外的商业及医疗等多个领域。

1.3 本章小节及论文结构

{46%：本章首先介绍了写作本文的原因和背景，并分析了软件测试和数据挖掘技术在国内外的发展现状，} 也对本文将实现什么样的系统进行了简单的描述。

第二章中介绍了实现本文所述的系统需要使用到的相关技术及算法，对本文所述的系统进行可行性分析。

{44%：第三章对系统进行分析，从系统需求、系统用例以及系统活动图三个方面进行分析。} {55%：其中系统需求分析从功能性需求分析和给功能性需求分析两个方面入手。}

{47%：第四章对系统进行了设计，从总体结构到每个功能模块进行设计，并设计了系统将要用到的数据库。}

第五章是系统的具体实现，首先介绍了系统整体框架的搭建，之后结合部分核心代码对系统功能模块的具体实现进行介绍。

{46%：第五章是系统测试，本文中列举了多个测试用例，并结合测试用例的操作和结果对本系统进行分析。}

{44%：第七章是总结章，对本文所述的系统实现情况进行总结，分析系统现有的不足，并对如何解决这些问题和不足进行展望。}

2 相关技术简介及可行性分析

2.1 相关技术简介

2.1.1 Django

Django当前的主流的开源Web框架之一，Django框架与SpringBoot框架类似，只不过SpringBoot使用Java进行开发而Django使用Python开发。 Django采用MVC架构，即模型Model，视图View和控制器Controller。 作为一个开源的框架，Django最显著的优点就是有强大的社区和文档支持。 Django还有强大的数据库功能，使用 Python的类对应数据库中的表，即数据模型不依赖于特定的数据库， 实现了数据库和数据模型的解耦。 Django的设计目的是可以简便快捷的开发web应用程序，重视代码重用，组件可以以插件的形式服务于整个系统。 同时，Django还有许多功能强大的第三方插件，Django的可扩展性也支持用户开发自己的工具包。 Django可以使用Redis等缓存系统，网页的加载速度得到了保证，因此使用Django开发的web系统可以一定程度上保证用户体验。

与 SpringBoot需要 Shiro框架才能进行用户权限认证不同， Django本身集成了用户的认证功能，默认的认证系统会在系统创建的 auth_ user表中读取数据并与需要认证的数据进行认证， 开发者可以在用户认证完成后调用 login方法将该用户的登录信息记录在系统中。 当接口添加@login_required注解时，则只有登录的用户才可以访问，否则就会被重定向到指定的url中。 Django的认证系统还为开发者提供了重写功能，当 Django默认的认证系统满足不了实际的应用场景时， 可以重写认证的方法，或者修改默认的用户信息的数据库表进行个性化开发。

2.1.2 MVC架构

图2.1 MVC模式架构

MVC架构是当前主流的 Web系统架构，其中 M是指 Model，即数据库模型，{51%：与数据库中的表相对应，可以将对数据库的增、删、改、查等命令封装成函数，}{69%：将对数据库表的操作转化为对类的操作；} V是指View，即显示的视图，在Django项目中与template中的网页相对应； C是指Controller，即控制器，一般用来响应接口并进行数据处理。 MVC是一种常用的设计模式，可以将接口响应层、视图层、模型层、逻辑层和服务层分离，降低项目的耦合度。 {42%： MVC模式的架构如图2.1所示，当用户访问接口时，系统会通过 Controller层响应请求，} {42%：对请求进行处理，如果需要访问数据库，则通过 Controller层对 Model层进行操作，实现对数据库的操作，} 写入数据或读取数据后将反馈信息显示在 View层上反馈给用户。

当用户的请求实现比较复杂时， Controller层的业务会非常复杂，为了降低Controller层的耦合度，在 Web系统开发中， 一般除了 Controller层、 Model层和 View层以外，还会设置逻辑层和服务层。 逻辑层和服务层在 Controller层和Model层之间，当用户调用接口时， Controller层会对数据进行简单处理， 并调用相应的逻辑层方法，逻辑层对数据进一步处理，而服务层是各个方法的具体实现， 与 Model层进行交互，用来被逻辑层调用。

2.1.3 Nginx

Nginx是一款与Apache类似的轻量级的开源反向代理服务器，由于其轻量级且开源的特性，Nginx被广泛应用于Web应用程序中。 {43%：反向代理与正向代理的思路相反，正向代理是指用户想要访问一个指定的 URL时不是直接访问，而是通过代理服务器去访问这个已知的地址，} {44%：而反向代理是指用户只知道代理服务器的地址，而实际上访问哪个服务器对于访问者来说是透明的，} 它代理的是服务端，适用于服务器集群分布式部署的场景，反向代理更重要的特点是可以隐藏服务器的信息， {45%：过滤掉无效的请求，保证内网环境的安全。}

Nginx使用REST架构风格，基于http协议提供服务。 REST风格是指将http请求封装为创建、删除、更新、查询四种方法，简化浏览器与服务器端的交互。 与Apache不同的是，Nginx是一个轻量级的服务器，体积小，占用内存小。 {57%：Nginx还是一个跨平台的服务器，可以运行在Windows、Linux及Mac等操作系统上。}

Nginx可以将 http的请求进行转发，通过监听指定的端口，每当有请求发送到这个端口时， Nginx就可以根据该请求相应的进行转发， {42%：由于80端口是 http请求的默认端口，所以一般将 Nginx的监听端口配置为80端口。} Nginx可以根据配置同时监听多个端口，为多个服务进行代理。

2.1.4 uWSGI

uWSGI是一个与Nginx类似的Web服务器，实现了WSGI、uwsgi、http等协议，其中WSGI是一种通信协议，uwsgi是一种用来定义传输信息类型的线路协议。 uWSGI服务器一般与Django项目的部署同时出现，之所以要使用 uWSGI服务器，是因为 Python并不可以快速的处理所有的请求， Nginx接收到动态请求时会将请求转发给 uWSGI服务器， uWSGI收到后对请求进行处理，处理成 web应用可以接受的形式， 再调用对应的方法。

2.1.5 MySQL

MySQL是一款开源的、常用的SQL数据库服务器。 {46%：是学生以及中小型互联网企业常用的关系型数据库。} MySQL最重要的特性是其开源性，与Oracle数据库需要支付高额的费用不同，只要遵守开源协议，就可以免费使用MySQL。 MySQL还有简单易用等特点，Oracle数据库复杂的配置让很多开发者望而却步，而 MySQL是一款轻量级的关系型数据库服务器， 只需要简单的安装配置即可使用。 {41%：MySQL还是一款跨平台的数据库服务器，支持多种编程语言，可以在多种环境下开发，可以运行在Windows、Linux、Mac等多种操作系统上。}

2.1.6 协同过滤算法

图2.2 协同过滤推荐商品的步骤

协同过滤的主要功能是预测和推荐，是一个常见的推荐算法，推荐商品的步骤如图2.2所示。 协同过滤主要基于用户的历史操作或者其他用户的相似操作，用以挖掘用户的喜好偏向，并预测该用户可能喜好的商品进行推荐。 预测推荐过程主要有两种推荐思路，一种思路是根据与该用户喜好偏向类似的其他用户的喜好来进行推荐， 例如，当某用户购买了部分商品，而另一个其他用户购买的商品与这个用户的商品非常类似，大多数都是相同的，{41%：就可以向这个用户推荐他没有购买但是与他相似的用户已经购买的商品；} {50%：另一种是向用户推荐与其喜好的商品类似的其他商品，这两种方式分别被称为基于用户的协同过滤和基于物品的协同过滤。}

{65%：协同过滤算法的实现可以分为以下几个步骤}

首先是收集数据。 协同过滤算法需要从大量的数据中挖掘出有用的信息，除了需要指定用户的信息之外，还需要大量其他用户的信息。 这些信息根据应用场景的变化而变化，例如当应用场景是影院向用户推荐电影时，就应该收集用户看过的电影或者评论过的电影等信息； {49%：而当应用场景是向用户推荐商品时，就应该收集用户购买过的商品等信息。} 本节中以下的步骤将以向用户推荐商品为应用场景展开。

其次是对获取的数据进行处理。 {74%：在计算用户相似度之前，需要对获取的数据进行处理。} 首先通过 Python编写的处理函数进行数据处理，整理出用户和商品的Python字典对象，其中用户作为 key值， {40%：该用户已购买的商品数组作为 value值，遍历所有相关用户的信息，得到 Python字典对象。}

{67%：然后是计算指定用户与其他用户之间的相似度。} 从Python字典中取出其他用户的信息，分别与指定用户计算相似度。 {70%：计算两个用户之间相似度的方法如下：} 遍历两个用户对应的商品数组，当某个商品出现在其中一个用户的已购买商品数组中，却没有出现在另一个用户的已购买数组中时， 这两个用户之间的距离加一，遍历所有的商品，计算出这两个用户的距离。 {42%：从定义中可以看出，用户之间的相似度与距离是成负相关的，取距离的倒数作为相似度，得到该指定用户与其他所有用户之间的相似度。}

最后是进行推荐。 {51%：在已经得到了该用户与其他用户之间的相似度的基础上，将其他用户按照相似度的大小进行排序。} 取出与用户相似度较高几个用户的信息，并将这几位用户的已购商品与指定用户的已购商品进行比较， 并向该用户推荐未出现在其已购商品数组中却出现在其他几位用户的已购商品数组中的商品。

协同过滤还可以根据用户对于之前的推荐的反馈信息来改变算法的推荐策略，进一步提高推荐的精度。 当用户数据比较少的时候，推荐信息可能会不太准确，但是在推荐的过程中，推荐策略可以得到不断调整，用户库也得到不断丰富，之后的推荐准确度会越来越高。

2.1.7 关联规则挖掘算法

2.1.7.1相关概念介绍

{45%：关联规则挖掘算法中会涉及支持度和置信度等概念，本节中将先介绍支持度等相关概念。}

支持度

{40%：支持度是指几个有关联关系的数据在数据集中同时出现的次数占总数据集的比例，公式为：}

$$Support(X\ Y) = P(X, Y) / P(I) = P(X \cap Y) / P(I) = Num(X \cap Y) / Num(I)$$

{49%：其中，I是总数据集，Num(X)表示X在总数据集中出现的次数，P（X）表示X出现的概率。}

最小支持度

最小支持度是算法开始前手动指定的一个阈值，一般情况下会根据算法的结果进行相应的调整。 {42%：在算法执行过程中会计算各项的支持度，判断各项是否为频繁项的标准就是

是否大于或等于最小支持度。}

置信度

{43%：置信度是指一个数据出现后另一个数据出现的概率，公式为：}

Confidence(X Y)= P(Y X)= P(X, Y)/ P(X) = P(X∩ Y)/ P(X)= Num(X∩ Y)/ Num(X)

{56%：其中，Num(X)表示X在总数据集中出现的次数，P（X）表示X出现的概率。}

最小置信度

最小置信度类似于最小支持度，是一个需要手动指定的阈值，并需要在算法执行过程中不断调整优化。 在得到频繁项集后需要对频繁项集进行处理得到置信度关系，当置信度低于最小置信度时，将不会将该条记录存放到结果中。

频繁项集

关联规则挖掘的过程中需要量化事件间的关联程度，常用的衡量标准是这些事件在数据集出现的次数。 {47%：频繁项集是指支持度大于或等于最小支持度的所有项的集合，其中最小支持度为指定的常数，取值在0到1之间。}

2.1.7.2算法基本步骤

图2.3 频繁集生成算法示例

{56%：关联分析可以从大量数据中挖掘出有价值的数据项之间的相互关系，常用于从数据库中关联分析出} “某个事件发生而导致其他事件发生” 这样的规则。 使用关联规则挖掘算法的目的是为了发现关联规则，通常会借助频繁项集，常用的评估频繁项集的标准是支持度。

{44%：以图2.3中所示的共9条数据为例，当使用关联规则挖掘这些数据项中元素的关联规则时，} {43%：首先需要指定最小支持度为0.2，计算每个数据在总数据集所占的比例作为该数据的支持度，} 可以看出图2.3中的数据共涉及 P1、 P2、 P3、 P4和 P5共五种数据， 这五种数据出现的次数如图2.3中 C1表所示，将它们出现的次数除以总数得到对应项的支持度， {41%：保留支持度不小于最小支持度的项，如图2.3中通过 C1得到计算并比较复杂度之后可以得到 L1中的结果，} 并对 L1中的结果进行连接得到如图2.3中 C2的新的项集，并分别计算各项在原始数据中出现的次数。 {54%：并将它们出现的次数除以总数，可以得到各项的支持度，保留支持度不小于最小支持度的项，} 从图2.3中的 L2可以看出，删除了 C2中的{ P1， {70%： P4}等支持度小于最小支持度的项。} {44%：对满足最小支持度的项进行连接操作，并计算连接操作之后所有项的支持度，保留所有满足最小支持度的项，} {52%：并重复以上操作，直至所有的项都不满足最小支持度为止。} 其中连接操作是指两个包含 n个元素，且前 n-1个元素都相同的项，将其中一个项的第 n个元素添加到另一个项中， 得到一个新的包含 n+1个元素的项。 记录下整个操作过程中所有满足最小支持度的项，分别计算各项的各种组合的置信度，例如项{ P1， P2}需要计算 P1=] P2和 P2=] P1的置信度。 得到了各项的置信度，也就得到了各项中元素之间的关联关系。

2.2 可行性分析

2.2.1 技术可行性分析

本文的系统实现遵循软件工程中的开发流程，服务器端通过开源框架 Django和Python语言进行开发， 使用 MVC模式，前端使用 Bootstrap框架和 JavaScript等技术进行开发，一定程度上实现了前后端分离。 数据库使用简单易用的轻量级框架MySQL，可以通过Django自带的引擎与Django框架共同工作。 由于本文中的系统业务不是特别复杂，所以项目的部署发布也比较简单，因此在技术上是可行的。

2.2.2 经济可行性分析

{46%：由于Django是一个轻量级的web应用框架，对于运行环境要求并不是很高。} 本文中的系统将会部署在腾讯云服务器上，通过学生认证后购买价格较低。 本文所使用的开发环境为PyCharm，通过学生认证后可以免费使用。 本文中所使用的其他的技术大多为开源项目，只要遵循开源协议就可以免费使用，因此本文所描述的系统在经济上是可行的。

2.2.3 操作可行性分析

本文中的系统涉及到的业务较少，系统的页面简单友好且操作性强，而且考虑到使用本系统的是中国移动的套餐配置员， 对计算机知识的了解较多，只需要简单的指导就可以对系统进行操作，使用系统的功能， 对 SQL文件进行测试，因此本系统在操作上是可行的。

2.2.4 其他分析

本文中的系统需要用户登录后才能进行操作，一切未登录的操作都会被重定向到登录界面。{41%：本文中的系统会对用户的密码进行加密后保存，数据库中不会保存用户的明文密码，}{45%：保护了用户的隐私，提高了系统的安全性，因此本系统在安全上是可行的。} 本文中所使用到的技术均为开源项目，所以在法律上是可行的。

3 系统分析

3.1 系统需求分析

3.1.1 系统功能需求分析

从实际应用需求考虑，本系统中主要有两种用户角色：

普通用户： {41%：普通用户可以注册和登录账号，但是刚注册的账号处于冻结状态，无法登录系统。} {45%：管理员用户可以激活普通用户的账号，激活后用户可以登录到系统中。} 登录成功之后普通用户可以上传需要测试的SQL文件到系统中，系统会根据上传的文件自动生成任务； 用户可以对自己创建的任务进行操作，当任务未执行时可以点击执行，当任务执行失败时可以点击重新执行， 当任务执行成功后，页面上的开始测试按钮会变成查看报告按钮，用户可以点击查看报告按钮跳转到相应的页面查看测试报告； {54%：用户可以在测试报告页面查看生成的测试报告；} 用户可以登出系统。

管理员用户： 与普通用户不同，管理员用户只能登录，无法注册，可以管理普通用户的账号，可以激活普通用户账号、冻结普通用户账号。 管理员用户没有测试SQL文件的功能。

3.1.2 系统非功能需求分析

{49%：本系统的非功能需求分析将会从系统性能需求分析、系统开发环境以及系统生产环境三个方面进行分析。}

系统的性能分析需要考虑系统应该满足的容量约束或者时间约束，一般情况下包括系统的{43%：响应时间、系统能够支持的最大并发量、系统的可靠性和安全性等方面的分析}

{43%：系统的响应时间是指用户对系统进行操作时，从用户发出指令到系统做出回应并呈现给用户所经历的时间。} 除了文件测试功能以外，系统的其他功能需要保证在用户发出指令的5秒之内做出回应， 否则应该在日志中记录该功能调用超时的情况，方便以后的问题分析和性能优化。 系统的最大并发量是指系统能够在一段时间内能够容纳最多的在线用户数，影响该指标的最大因素是文件测试功能， 因为该功能执行时间较长，对资源的需求较高，但是考虑到系统的应用场景为中国移动创建套餐前检查 SQL文件， 由于同一时间内不会有多个套餐同时创建，所以不需要过多考虑最大并发量。 本系统将运算量较大的SQL测试功能放在了第三方云平台上，因此可靠性可以得到保证。 系统使用了Nginx服务进行反向代理，且对所有后端接口进行了权限控制，可以保证系统的安全运行。

本系统的开发环境如下： 操作系统为Windows 10； 开发平台为PyCharm； 数据库为MySQL5.7。 生产环境如下： 云服务： 腾讯云服务； 操作系统为Ubuntu 1604； 内存为2G； 数据库为MySQL5.7。

3.2 系统用例分析

3.2.1 系统总体用例分析

{44%：本系统的总体用例图如图3.1所示，主要包含用户管理模块、文件管理模块、任务管理模块以及测试报告管理模式共四个功能模块。}

图3.1 系统总体用例图

3.2.2 功能模块用例分析

本小节中将对上一小节所列出的功能模块的具体功能进行用例分析。

3.2.2.1用户管理模块

用户管理模块的参与者有普通用户的管理员，这两种角色都登录和登出系统，只有普通用户可以注册账号， {42%：而冻结用户和激活用户的功能只有管理员参与，普通用户无法参与。} {83%：用户管理模块的用例图如图3.2所示。}

图3.2 用户管理模块用例图

{60%：用户登录功能的用例描述如表3.1所示；} {62%：用户登出功能的用例描述如表3.2所示；} {71%：用户注册功能的用例描述如表3.3所示；} 冻结用户功能的用例描述如表3.4所示； {69%：激活用户功能的用例描述如表3.5所示。}

3.2.2.1文件管理模块

{48%：文件管理模块的参与者只有普通用户，普通用户可以上传 SQL文件，检测SQL文件，} {62%：读取 SQL文件等，文件管理模块的用例图如图3.3所示。}

图3.3 文件管理模块用例图

上传文件功能的用例描述如表3.6所示； 检测文件功能的用例描述如表3.7所示；

读取文件功能如表3.8所示。

用　例　名　称用户登录

标识符A11

{60%：用例描述用户输入用户名和密码等信息进行登录}

参与者用户

状态通过用户认证

{51%：前置条件用户信息已在系统中注册，用户打开系统登录页面}

{74%：后置条件用户登录成功或登录失败}

{55%：基本操作流程1.用户打开系统登录页面；}

2.用户输入用户名和密码；

3.用户点击登录按钮

{55%：可选操作流程1.如果用户名或密码错误，则会提示用户错误信息}

假设用户已经注册且打开了登录界面

表3.1　用户登录功能用例描述

用　例　名　称用户登出

标识符A12

{47%：用例描述用户清除在系统的登录状态并退出本系统}

参与者用户

状态通过用户认证

{55%：前置条件用户信息已在系统中注册，用户以及登录到系统中}

{65%：后置条件用户登出成功或登出失败}

基本操作流程1. 用户点击登出按钮

可选操作流程1.如果出现网络错误或者系统错误则会提示用户登出失败

假设用户已经登录到了系统中

表3.2　用户登出功能用例描述

用　例　名　称用户注册

标识符A13

{53%：用例描述用户输入用户名和密码及注册类型等信息进行注册}

参与者用户

状态通过用户认证

前置条件用户打开本系统的注册页面

{52%：后置条件注册成为普通用户或者注册失败}

基本操作流程1.用户打开本系统的注册页面

2.用户输入用户名和密码

{53%：可选操作流程　1.如果注册失败，系统会提示用户失败信息}

假设用户已经打开了本系统的注册页面

表3.3　用户注册功能用例描述

用　例　名　称冻结用户信息

标识符A14

{55%：用例描述管理员可以冻结普通用户的账号，使其失效}

参与者管理员用户

状态通过用户认证

{76%：前置条件管理员用户已登录到系统中}

{42%：后置条件冻结成功则该普通用户无法登入到系统中}

{44%：基本操作流程1.管理员用户选择需要冻结的用户；}

2.点击操作列表中的冻结按钮

可选操作流程无

假设管理员已登录到系统中

表3.4　冻结用户信息功能用例描述

用　例　名　称激活用户信息

标识符A15

{49%：用例描述管理员可以激活普通用户的账号，使其生效}

参与者管理员用户

状态通过用户认证

{49%：前置条件管理员用户已登录到系统中，且该普通用户账户被冻结}

后置条件激活成功则该普通用户账户恢复正常使用

{40%：基本操作流程1.管理员用户选择要激活的用户；}

2.管理员点击操作列表中的激活按钮

可选操作流程无

{46%：假设管理员已登录到系统中，且普通用户的信息被冻结}

{60%：表3.5　激活用户信息功能用例描述}

用　例　名　称上传文件功能

标识符A21

{51%：用例描述普通用户可以上传需要检测的SQL文件}

参与者普通用户

状态通过用户认证

{75%：前置条件用户已经登录到本系统中}

{65%：后置条件文件上传成功或上传失败}

{84%：基本操作流程1.用户选择需要上传的文件。}

2.用户点击上传文件按钮。

{47%：可选操作流程文件上传成功或失败都会提示用户}

假设用户已经登录到了系统中

表3.6　上传文件功能用例描述

用　例　名　称检测文件功能

标识符A22

用例描述普通用户可以检测SQL文件

参与者普通用户

状态通过用户认证

{60%：前置条件用户已经登录到本系统中，且已经创建好了任务}

后置条件检测成功或检测失败

基本操作流程1.用户上传文件；

2.用户选择需要检测的文件；

3.用户点击开始检查按钮。

可选操作流程无

{44%：假设用户已经登录到了系统中且已经创建好了任务}

表3.7 检测功能用例描述

用 例 名 称读取文件功能

标识符A23

用例描述普通用户可以读取已经上传的文件

参与者普通用户

状态通过用户认证

{42%：前置条件用户已经登录到本系统中，且已经上传了文件，且用户进入了检测列表界面}

后置条件显示用户选取的文件

{54%：基本操作流程用户点击想要查看的文件}

可选操作流程无

{43%：假设用户已经登录到了系统中，且进入到了检测列表界面}

表3.8 读取文件功能用例描述

3.2.2.1任务管理模块

任务管理模块的参与者只有普通用户，用户可以创建新的任务，启动任务以及读取任务信息等功能， {100%：任务管理模块的用例图如图3.4所示。}

图3.4 任务管理模块用例图

{63%：创建新的任务功能的用例描述如表3.9所示。}

用 例 名 称创建新任务功能

标识符A31

{54%：用例描述普通用户可创建新的检测任务}

参与者普通用户

状态通过用户认证

{75%：前置条件用户已经登录到本系统中}

后置条件创建好用户指定的任务

{84%：基本操作流程1.用户选择需要检测的文件；}

2.用户点击上传按钮；

3.系统自动根据上传的文件创建任务。

可选操作流程无

假设用户已经登录到了系统中

表3.9 创建任务功能用例描述

启动任务功能的用例描述如表3.10所示； {66%：查看任务信息功能的用例描述如表3.11所示。}

用 例 名 称启动任务功能

标识符A32

用例描述普通用户启动自己创建的任务

参与者普通用户

状态通过用户认证

{51%：前置条件用户已经登录到系统中，且已经创建好了任务}

后置条件启动任务，开始检查文件

{55%：基本操作流程1.用户选择需要启动的任务；}

2.用户点击该任务操作列表中的“开始检测”按钮

可选操作流程无

{44%：假设用户已经登录到了系统中且已经创建好了任务}

表3.10 启动任务功能用例描述

用 例 名 称查看任务信息功能

标识符A33

{54%：用例描述普通用户可以查看自己创建的任务的信息}

参与者普通用户

状态通过用户认证

{57%：前置条件用户已经登录到本系统中且已经创建好了任务}

后置条件显示任务信息

{42%：基本操作流程1.用户点击主菜单中“ 检测列表 ”选项卡}

可选操作流程无

{44%：假设用户已经登录到了系统中且已经创建好了任务}

{66%：表3.11　查看任务信息功能用例描述}

3.2.2.1测试报告管理模块

{48%：测试报告管理模块的参与者只有普通用户，用户可以生成测试报告和查看测试报告，测试报告管理模块的用例图如图3，5所示。}

{69%：图3.5　测试报告管理模块用例图}

{54%：生成测试报告功能的用例描述如表3.12所示。}

用　例　名　称生成测试报告功能

标识符A41

{46%：用例描述普通用户根据需要检测的SQL文件生成测试报告}

参与者普通用户

状态通过用户认证

{51%：前置条件用户已经登录到本系统中且进入到了“ 测试列表 ”页面}

后置条件生成测试报告

{40%：基本操作流程1.用户选择需要检测的任务并点击“ 开始检测 ”按钮；}

{43%：2.系统会对文件进行检测并自动生成测试报告}

可选操作流程无

{46%：假设用户已经登录到了系统中进入到了“ 测试列表 ”页面}

表3.12　生成测试报告用例描述

{67%：查看测试报告的用例描述如表3.13所示。}

用　例　名　称查看测试报告功能

标识符A42

{47%：用例描述普通用户查看需要检测的SQL文件的测试报告}

参与者普通用户

状态通过用户认证

前置条件用户进入到了“ 测试列表 ”页面，且需要查看的测试报告已生成

后置条件显示测试报告详情

{54%：基本操作流程1.用户选择需要查看的任务}

{47%：2.用户点击该任务操作列表中的“ 查看报告 ”按钮}

可选操作流程无

{63%：表3.13　查看测试报告功能用例描述}

3.3　系统活动图

3.3.1　用户管理模块活动图

由于登入的活动较为简单，冻结用户与激活用户的活动类似，所以在本模块中选取了登录功能、注册功能以及激活用户的活动图进行展示

3.3.1.1登录功能活动图

{67%：用户登录功能活动图如图3.6所示。}

3.3.1.2注册功能活动图

{77%：用户注册功能活动图如图3.7所示。}

3.3.1.3激活用户功能活动图

{65%：激活用户功能活动图如图3.8所示。}

3.3.2　文件管理功能模块活动图

由于文件管理功能模块中的几个功能活动比较相似，本文中只展示上传文件功能的活动图。

3.3.2.1上传文件功能活动图

{60%：上传文件功能的活动图如图3.9所示。}

图3.6　登录功能活动图

图3.7　用户注册功能活动图

{49%：图3.8　管理员激活普通用户功能活动图}

{62%：图3.9　普通用户上传文件活动图}

3.3.3　任务管理功能模块活动图

由于任务管理功能模块中的几个功能活动比较相似，本文中只展示启动任务功能的活动图。

3.3.3.1启动任务功能活动图

{54%：普通用户启动任务活动图如图3.10所示。}

图3.10 普通用户启动任务活动图

3.3.4 测试报告管理功能模块活动图

3.3.4.1查看测试报告功能活动图

图3.11 查看测试报告功能活动图

4 系统设计

4.1 系统体系结构设计

本系统主要通过协同过滤算法和关联规则挖掘算法等数据挖掘方法对数据进行分析，使用者通过浏览器/服务器（B/S）模式与系统进行交互。 本系统将通过 Django搭建，用户通过交互界面输入 SQL，服务器端接收数据后调用使用 Python编写的服务对数据进行处理，再通过利用协同过滤算法和关联规则挖掘算法编写的服务进行分析，最后将分析结果生成报告，通过前端浏览器进行展示。 系统框架流程如图4.1所示，当有http请求访问服务器时，会被Nginx接收并处理，Nginx会判断该请求为静态请求还是动态请求。 {42%：静态请求即访问静态文件的请求，动态请求即需要服务器对输入的数据进行处理并返回相应数据的请求。} Nginx会将静态请求直接处理，而动态请求会被转发给uWSGI服务器进行处理，uWSGI服务器会对该请求进行相应的处理后再交给Django处理。

图4.1 系统框架流程

4.2 系统功能模块设计

4.2.1 用户管理模块

本系统的用户管理模块主要有注册账号、用户登录、用户登出、激活用户、冻结用户等功能，如图4.2所示。

图4.2 用户管理模块

4.2.2 文件管理模块

{42%：由于本系统的核心功能是对SQL文件进行扫描检测，所以需要对文件进行管理。} {46%：本系统的文件管理模块主要有上传文件、保存文件、读取文件内容等功能，如图4.3所示。}

图4.3 文件管理模块

4.2.3 任务管理模块

由于系统中是将需要检测的文件抽象到任务中，所以任务管理模块是本系统的核心模块，该模块主要包含创建任务、启动任务、获取任务信息等功能，而启动任务功能中包含 SQL文件数据处理、协同过滤算法调用、关联规则挖掘算法调用等功能， 如图4.4所示。

图4.4 任务管理模块

4.2.4 测试报告管理模块

{45%：本系统通过测试报告将SQL文件测试结果展示给用户，测试报告管理功能包括数据读取功能和测试报告生成功能，如图4.5所示。}

图4.5 测试报告管理模块

4.3 数据库设计

{64%：本系统的数据库设计将从以下几个方面考虑：} 首先需要设计用户表(user)，用户需要登录以后才可以进行相关操作，该表中记录用户的账号、密码、角色等信息。 User表的设计如表4.1所示。

属性名类型是否为空属性说明

idintNot nullPrimary_key用户id

namevarcharNot null用户名

passwordvarcharNot null用户密码

create_timevarcharNot null账号创建时间

modify_timevarcharNot null账号最后修改时间

roleintNot null用户角色

user_stateintNot null账号状态

表4.1 user表设计

另外，用户提交 SQL文件进行检测在系统中是以“任务”的形式存在的， 即每次提交都会在系统中新建一个“任务”，该任务记录本次 SQL检测的用户和文件信息以及任务的执行状态等内容， 所以需要设计任务表(task)。 task表的设计如表4.2所示

属性名类型是否为空属性说明

idintNot nullPrimary_key任务id

user_idintNot nullForeign_key用户id

file_namevarcharNot null文件名

create_timevarcharNot null任务创建时间

stateintNot null任务状态

表4.2 task表设计

检测结果将会以测试报告的形成呈现给用户，报告中包含 SQL文件中可能存在的错误，

因此本系统需要设计错误表(mistake)， {43%：包含错误位置和错误类型等信息，mistake表的设计如表4.3所示。}

属性名类型是否为空属性说明

idintNot nullPrimary_key错误id

task_idintNot nullForeign_key任务id

mistake_typevarcharNot null错误类型

mistake_gradevarcharNot null错误预警等级

mistake_detailvarcharNot null错误详情

find_timevarcharNot null错误检测时间

methodvarcharNot null错误处理方法

extendsvarcharNot null其他

表4.3 mistake表设计

5 系统实现

本章将对系统框架搭建的详细步骤以及核心功能的实现进行介绍。 {41%：本系统采用B/S模式进行搭建，通过Django、Nginx、uWSGI等实现，数据库则采用较为常用的MySQL。}

5.1 搭建系统框架

{47%：为了保证系统的运行效率及稳定性，本系统选择Linux作为开发环境，操作系统的版本为Ubuntu16.04。}

本系统虽然没有将前端和后端分成两个项目实现，但是前端所有的数据处理工作都是由JavaScript实现， {42%：极大的减少了服务器的压力，现了动态资源和静态资源分离，提高了性能和扩展性。}

图5.1 系统操作流程

在搭建系统之前需要考虑系统的操作流程，本系统的操作流程如图5.1所示，首先需要验证用户的身份， 用户的密码通过 Django自带的加密方式进行加密，提高系统的安全性，用户登录之后在主页中上传需要检测的文件， 系统接收用户上传的文件并对该文件创建任务，通过任务信息调用系统的扫描检测服务，最后将扫描结果保存到数据库并生成测试报告，{55%：用户可以通过浏览器在线查看测试报告。}

5.1.1 Django项目创建

在创建Django项目之前需要先安装Python环境，本系统所使用到的Python版本为Python3.6，安装完Python后将安装路径配置到系统环境变量中。 创建Django项目可以从其官网上下载对应版本的项目压缩包，或者使用pip命令进行安装，两种途径创建的结果并无太大区别。 项目创建完成后通过pip命令安装mysqlclient与numpy等模块，所有模块导入之

后新建一个应用。 Django的一个项目中可以创建多个App，每个App都是相互独立的，可以移植的业务。

App创建完成后需要进行配置，首先需要配置静态文件存放的位置，其次是指定所使用的数据库为 MySQL， {47%：并配置数据库名称和密码等参数，最后对 MySQL数据库进行初始化操作，} 将需要使用到的数据表在 MySQL中生成。 本项目采用的 MVC架构，将控制层、服务层、模型层分开，在应用根目录下新建 service和 model等 Python包用来存放相应的 python文件， 当接收到前端发送的请求时会先通过 urls. py中的接口配置访问指定的 Controller函数， Controller函数对请求进行处理后调用 service层中的方法， 关联规则挖掘方法、协同过滤方法、数据处理方法等服务的函数都是在这一层实现的，当需要对数据库进行操作的时候， {50%： service层会通过 model层相应的类对数据库进行操作。}

{42%：虽然前后端分离有利于系统的并行开发和降低系统的耦合度，但是由于本系统的业务较少，} 前后端分离开发的必要性不是很高，因此没有选择前后端完全分离的设计模式。本系统的html页面放置在系统的template路径下，css及js等静态文件存放在系统是static路径下。 Django项目结构如图5.2所示。

图5.2 Django项目结构

5.1.2 uWSGI环境配置

uWSGI的配置较为简单，安装完 uWSGI之后，修改相应配置文件，使 uWSGI服务器监听指定端口， 当该端口接收到请求时，对该请求进行处理之后把请求转发到 Django所监听的端口。 设置服务类型为socket而非http，因为请求不是直接通过http方式访问uWSGI服务器的，而是通过Nginx服务器来访问的。

5.1.3 Nginx环境配置

{48%：安装完Nginx服务器之后通过修改Nginx自带的配置文件nginx.conf对Nginx服务器进行配置。} 由于使用者是通过浏览器访问服务器端的，所以 Nginx需要监听服务器的80端口，并将所有发送到 uWSGI服务器所监听的端口， 此处配置的端口号应与 uWSGI配置的端口号保持一致。 Nginx服务器中还应该配置静态文件的路径，当使用者发送静态请求，只访问静态文件时， Nginx服务器对该请求不做转发， 而是直接处理，只有当 Nginx服务器接收到动态请求时才会转发到 uWSGI中进行处理。

5.2 用户信息认证功能实现

Django项目会在数据库中创建默认的用户表，该表中虽然包含了很多信息，但是无法满足本系统的需要， {41%：所以本系统额外创建了用户表，用来保存用户的角色及状态等信息。} 用户信息的认证使用 Django默认的认证方法，而 Django默认的认证方法只会检索系统的数据库表， 为了让用户表内的信息得到认证，本系统的实现方法是将用户表内的信息实时同步到系统的表中， 示例代码如表5.1所示。

{46%：当用户发送登录请求时，系统会对用户请求中的用户名密码的参数进行处理，}并通过系统的 auth. authenticate(username= username， password= password)进行处理，当认证通过时， 该方法会返回该用户的相关信息，否则返回 None值。 {46%：如果认证通过，则会通过login(user)方法将用户登录到系统中。}

如果用户未经过系统认证并登录到系统中，则其访问除登录和注册以外的所有接口都会被

拦截并重定向到登录页面。 实现的方法是使用@login_required注解。

```
user_service.py

def create_user(username, password):

""" "

: param username: 用户名

: param password: 密码

: return:

""" "

curr_time = datetime.datetime.now().strftime(' %Y-%m-%d %H: %M:
%S' )

User.objects.create_user(username=username, password=password,
email=' null' )

models.user.objects.create(username=username,
password=hashers.make_password(password), create_time=curr_time,

modify_time=curr_time)
```

{49%：表5.1 同步用户信息到系统认证数据库表示例代码}

用户登录界面如图5.3所示。

图5.3 用户登录界面

5.3 文件处理功能实现

当用户选择文件时，系统会先检测文件类型，如果不满足. sql或者. txt的后缀要求，则用户无法选择该文件，代码实现如表5.2所示。

```
main.js

$(" #input-1a" ).fileinput({

… … /////

allowedFileExtensions: [' sql' , ' txt' ],

… … ////

})
```

{52%：表5.2 前端对用户上传文件类型进行检测}

用户发送上传文件请求后，系统接收文件并将其保存在指定位置，并通过该文件创建新的任务，上传文件页面如图5.4和图5.5所示，上传功能代码实现如表5.3所示，任务创建成功的页面如图5.6所示。

图5.4 上传文件页面

图5.5 上传文件成功页面

当任务启动后，系统会根据任务内容中保存的文件信息去读取文件，默认情况下，本系统会使用‘ UTF-8’的编码格式去读文件，当使用‘ UTF-8’无法读取文件时，系统会使用‘ GBK’的编码格式读取文件。 {43%：读取文件内容后通过正则匹配取出需要的格式化的数据，} 得到数据后再调用利用协同过滤算法实现的检测工具和利用关联规则挖掘算法实现的检测工具对 SQL语句进行扫描检测，以协同过滤方法为例，读出所有历史套餐中的所有数据库表名，正则匹配实现代码如表5.4所示。

file_controller.py

```
@login_required(login_url=’/’)
def upload_file(request):
files = request.FILES
file = None
if files:
for i in files:
file = files[i]
data = {}
if file:
file_map = file_service.upload_file(file)
data = file_map
user_id = user_service.get_user_by_name(request.user)[0][’id’]
task_service.create_task(user_id, file_map[’file_name’])
else:
data[’code’] = 2
data[’message’] = ’文件上传失败’
return HttpResponse(json.dumps(data))
```

表5.3 读取文件功能实现代码、

```
file_service.py

def  get_tables(sql_str):

res  =  re.findall(r" insert  into  (.+?  )  " ,  sql_str,  re.S)

if  res  is  None  or  len(res)  ==  0:

res  =  re.findall(r" INSERT  INTO  (.+?  )  " ,  sql_str,  re.S)

data  =  []

for  i  in  res:

if  i  not  in  data:

data.append(i)

return  data
```

表5.4 正则匹配数据库表名实现代码

图5.6 任务列表页面

5.4 SQL文件检测功能实现

SQL文件检测功能是本系统的核心功能，使用协同过滤算法和关联规则挖掘算法对SQL进行检测，并预测出可能存在的错误。

{46%：5.4.1 利用协同过滤算法检测数据库表错误}

由于待检测的 SQL会涉及到多表操作，所以检测的第一步应该是定位到哪些表的操作可能存在问题， {44%：之后再对可能存在错误的表进行进一步的分析。} 多个套餐的SQL和数据库表之间存在多对多的关系，协同过滤算法适合对这样的场景进行预测分析。 本方法中将一个套餐看作是一个用户，而套餐中涉及到的数据库表看作商品。 本文计算套餐间相似度使用的是较为简单的方法，即计算套餐间的距离，将距离的倒数作为相似度， 套餐间的距离即两个套餐中涉及到的不相同的数据库表的个数。 本文中将协同过滤算法通过python语言实现，并将其服务化，方便系统的调用。

图5.7 协同过滤算法应用过程

{52%：协同过滤算法在本系统中的应用过程如图5.7所示。} 首先从历史套餐中读取数据到系统中，通过 Python编写的处理函数进行数据处理，整理、格式化大量的历史套餐，将每个套餐的 ID作为 Python字典对象的 key值，套餐中所涉及到的数据库表名列表作为 value值。 遍历所有的历史套餐，从历史套餐中整理出一份Python字典对象。 由于本系统中通过协同过滤算法实现定位错误是以 MVC架构中服务的形式实现的，所以用户的输入是通过 http请求的方式进行， 相应的 Controller对用户的请求进行处理分析，并将通过协过滤算法实现的服务对传入的数据进行处理， 与处理历史套餐的过程类似，得到新建套餐的 ID和包含的数据库表的数组。

{48%：处理完新套餐数据和历史套餐数据之后，} 将历史套餐的数据逐一与新套餐的数

据比较，当一个数据库表出现在其中一个套餐中却没有出现在另一个套餐中时，这两个套餐的距离便加一，遍历两个套餐包含的所有数据库表后，就可以得到这两个套餐间的距离。考虑到套餐间距离与它们之间的相似度成反比，本系统将距离取倒数作为量化两个套餐间的相似度的标准。{47%：重复以上操作，得到新套餐与所有历史套餐间的相似度。}

通过上面的计算得到了所有历史套餐与新建套餐间的相似度关系，将这些历史套餐按照相似度的大小进行排序。按照需求取出相似度最高的几个套餐的信息，如果需要推荐精度高一些，就多取几个套餐的信息，但是这样运算的复杂度就会高一点；反之则可以少取几个套餐的信息，减少运算的复杂度。如果只取一个历史套餐的信息，则可以比较取出的这个历史套餐的信息与新创建套餐的信息，则需要记录下新创建的套餐没有包含，而这个历史套餐却包含的数据库表。如果取了多个历史套餐的信息，则需要按照不同的需求使用不同的策略，可以取这几个历史套餐所包含的数据库表的交集或并集，然后再跟新建套餐所包含的数据库表比较，记录下新创建的套餐没有包含，而这些历史套餐的交集或并集却包含的数据库表，如果采用的策略是取并集，则SQL的检测结果覆盖面会比较广，会检测出更多的错误，自然也会包含更多的无效结果；如果采用的是交集策略，则SQL的检测结果会减少很多，无效结果也会减少很多，可是交集策略也会导致该检测出来的错误未被检测出来，其中无效结果是指某个数据库表实际上未发生错误，但是扫描结果却显示该数据库表出错。另外，还可以根据在相似套餐中出现的次数来记录预警级别，本系统中所使用的分类标准如下：当某个数据库表在9个或9个以上相似的套餐中出现却没有在待检测套餐中出现或者在新套餐中出现，或者在新套餐中出现却没有在2个以上的相似历史套餐中出现，则将该数据库表标记为高危错误；当某个数据库表在7个至9个相似的套餐中出现却没有在待检测的套餐中出现，或者在新套餐中出现却没有在5个以上的相似套餐中出现，则将该数据库表标记为中危错误；当某个数据库表在5个至7个相似的套餐中出现却没有在待检测的套餐中出现，或者在新套餐中出现，却没有在7个以上的相似套餐中出现，则将该数据库表标记为低危错误。

如图5.7所示，新创建的套餐的检测结果如果没有问题则会被加入到历史套餐中，用以丰富历史套餐库。当检测结果有问题时，则会将本次扫描检测发现的可能存在的错误存进数据库中，提交测试的用户点击查看报告时，本系统中生成报告的服务就会获取数据库中的错误信息，将其生成报告，以网页的形式展现给用户。{41%：协同过滤算法的核心实现代码如表5.5所示，计算相似的代码实现如表5.6所示}

```
collaborative_filtering_service.py

def start(package):

# 获取历史套餐数据

history_map = file_service.get_history_tables()

history = history_map[’data’]

sims = []# 计算历史套餐与新套餐的相似度

for his in history:

sim = get_similarity(package[’tables’], his[’tables’])

sims.append((his[’name’], sim))
```

```
#  将相似度从大到小排列
sims.sort(key=lambda  x:  x[1],  reverse=True)
#  获取相似度最高的10个套餐的数据库表
similar_tables_list  =  get_table_list(history,  sims[:  10])
#  计算相似度最高的十个套餐中未在新套餐中出现的表出现的次数
table_times  =  get_similar_tables_times(similar_tables_list)
#  计算结果
errors  =  get_error_tables(package['tables'],  table_times)
return  errors
```

表5.5 协同过滤算法实现核心代码

```
collaborative_filtering_service.py
def  get_similarity(new_tables,  old_tables):
distance  =  1
for  i  in  new_tables:
if  i  not  in  old_tables:
distance  +=  1
for  i  in  old_tables:
if  i  not  in  new_tables:
distance  +=  1
return  1  /  float(distance)
```

表5.6 协同过滤计算相似度实现代码

{50%：5.4.2 利用关联规则挖掘算法检测关键属性错误}

协同过滤算法本质上是通过分析与待处理的数据相似的一组数据，并对待处理数据的结果进行预测，其准确度无法达到百分之百。 为了提高检测的准确度，本系统采用关联规则挖掘算法配合协同过滤算法共同检测SQL中的错误。

本文中的关键属性是指套餐创建过程中可能设计到的某个数据库表中的属性。 一般来说，关键属性可以是每个数据库表中的外键，也可以是一些与数据库表有某些约束关系的属性。例如某个数据库表的功能是记录套餐间的关系，则其中的套餐ID和关系类型都可以看作关键属性。

{41%：与协同过滤算法在本系统中的应用类似，本系统中并没有用户和商品的概念，所以要在思路上进行转变。} {49%：与协同过滤算法不同，协同过滤算法检测的层次是套餐和数据库表之间，} {47%：而通过关联规则挖掘算法检测的层次是套餐涉及的数据库表和关键属性之间，} 基于关键属进行关联规则挖掘，将套餐内涉及到的一个数据库表视为一个事件，将每个数据库表中的关键属性视为交易的商品。 与利用协同过滤算法定位错误的数据库表相同，本文中利用关联规则挖掘算法检测关键属性错误的功能也会封装成服务，方便服务器的调用。

本系统并非是在新套餐创建时才进行历史套餐分析的，因为利用关联规则基于关键属性检测错误运算量很大， {44%：运算时间也很长，所以本系统设计将在闲时运算关联规则挖掘服务，并将分析得到的支持度和置信度保存在数据库中，} 当检测新套餐时可以直接从数据库中取数据进行比对，并将处理结果生成错误报告反馈给前端， 改善用户体验的同时，也可以提高系统效率。

图5.8 检测关键属性错误过程

检测关键属性错误的过程如图5.8所示，首先会读取历史套餐的数据， 分析历史套餐会涉及的数据库表中的关键属性，其中包括“商品间关系表”中的 “关联商品 ID”属性和“商品发布地市表”中的“城市 ID”属性等。 将每个套餐中的每个关键属性的具体的数据都整理出来，以关键属性为基础作为分类标准将这些数据分成不同的 Python字典对象，Python字典对象中的每个元素都是套餐 ID与该套餐中这个关键属性的所有具体数值组成的数组。

获得了 Python字典对象以后，对这些对象进行遍历分析，取出当前分析的 Python字典对象所有的 value值所包含的元素并去重， 各自作为一个数据项，这些数据项是原始数据。 {47%：计算这些项在数据集中出现的次数，并计算其支持度，将这些项的支持度与设定的最小支持度进行比较，} {50%：保留支持度大于或等于最小支持度的项，删除支持度小于最小支持度的项，因为包含该项的其他项的支持度必然会小于最小支持度。} {46%：将支持度大于或等于最小支持度的项添加到频繁项集中，并将这些项进行连接操作后进行下一轮的运算。} {45%：对连接操作后的项重新计算其支持度，并与最小支持度比较，小于最小支持度的项将被删除，} {44%：满足最小支持度的项会被添加到频繁项集中并进行连接操作，进入新的一轮运算。} {57%：重复以上操作，直至没有项的支持度大于或等于最小支持度。}

经过以上的操作可以得到套餐和关键属性的Python字典对的频繁项集。 对频繁项集中各项进行处理，将频繁项集中各项按照蕴含式的形式进行调整（即 p-] q的形式），{43%：每项都对应着不同的蕴含式，例如项{ A， B}可以生成 A-] B和 B-] A两个蕴含式，} 项{ A， B， C}可以生成 A-] BC、 AB-] C、 AC-] B等六个不同的蕴含式， 需要对这些不同的蕴含式分别计算置信度， {47%：将这些蕴含式的置信度与最小置信度进行比较，} {42%：满足最小置信度蕴含式将会被加入到最后结果集合中，} {52%：不满足最小置信度的蕴含式则会被弃置。} {45%：通过这一步的处理可以得到满足最小置信度的所有的蕴含式，这些蕴含式即是关联规则。}

{47%：需要利用关联规则挖掘算法检测套餐时，则根据数据库中的置信度信息，进行检测。} 例如“商品间关系表”中的“关联商品 ID”属性生成的置信度表保存了Confidence(“130000000653”=]“130000000654”)=1的信息， 而当待检测套餐“商品间关系表”中有“关联商品 ID”属性值有 “130000000653”的记录却没有“关联商品 ID”属性值有“130000000654”的记录， 则将该情况记录为关键属性错误，可以根据该蕴含式的置信度大小进行记录，本系统中使用的分类标准如下： 当置信度大于0.9时记录为高危

错误，置信度处于0.7至0.9之间时记录为中危错误，　置信度低于0.7则记录为低危错误，并生成错误报告反馈给前端。　{48%：关联规则挖掘算法实现核心代码如表5.7（a）（b）所示。}

```
apriori.py

def　start():

data　=　loadDataSet()

for　i　in　range(len(data)):

for　j　in　range(len(data[i])):

data[i][j]　=　str(data[i][j])

L,　supportData　=　apriori(data,　0.5)

rules　=　generateRules(L,　supportData,　minConf=0.3)

return　rules
```

表5.7（a）　关联规则挖掘功能对外接口的实现代码

通过以上两个方法对SQL文件进行检测，检测结构通过测试报告的形式呈现，如图5.9-5.10所示。

```
apriori.py

def　apriori(dataSet,　minSupport=0.5):

#　构建初始候选项集C1

C1　=　createC1(dataSet)

#　将dataSet集合化，以满足scanD的格式要求

D　=　list(map(set,　dataSet))
```

{69%：#　构建初始的频繁项集，即所有项集只有一个元素，L1是频繁项集，suppData是支持度表}

```
L1,　suppData　=　scanD(D,　C1,　minSupport)

L　=　[L1]
```

{100%：#　最初的L1中的每个项集含有一个元素，新生成的}

```
#　项集应该含有2个元素，所以　k=2

k　=　2
```

```
while len(L[k - 2]) ] 0:
Ck = aprioriGen(L[k - 2], k)
Lk, supK = scanD(D, Ck, minSupport)
{100%：# 将新的项集的支持度数据加入原来的总支持度字典中}
suppData.update(supK)
{100%：# 将符合最小支持度要求的项集加入L}
L. append(Lk)
{100%：# 新生成的项集中的元素个数应不断增加}
k += 1
{100%：# 返回所有满足条件的频繁项集的列表，和所有候选项集的支持度信息}
return L, suppData
```

{47%：表5.7（b） 获取初始数据的频繁项集和支持度信息}

图5.9 错误统计情况

图5.10 错误详情

6 系统测试

6.1 测试用例

6.1.1 用户登录功能测试用例设计

{80%：用户登录功能测试用例的设计如表6.1所示。}

用 例 名 称用户登录

标识符T1

{72%：测试目的1.测试用户是否能够正常登录；}

2.测试系统的拦截是否生效；

3.测试系统的认证是否有效。

{71%：测试操作1.使用正确的用户名和错误的密码；}

2.使用错误的用户名；

3.使用正确的用户名和密码

4.不登录系统而直接访问其他接口

测试结果1.测试操作1无法登录系统，说明可以对密码进行校验；

{40%：2.测试操作2无法登录系统，说明系统可以对用户名进行校验；}

{44%：3.测试操作3成功登录系统，说明系统可以对用户信息进行认证；}

4.测试操作4无法访问其他接口，说明系统的拦截生效。

表6.1 用户登录功能测试用例

6.1.2 用户登出功能测试用例设计

{65%：用户登出功能测试用例的设计如表6.2所示。}

用 例 名 称用户登出

标识符T2

{66%：测试目的1. 测试用户是否能正常登出}

2. 测试系统的认证是否有效

测试操作1.用户点击登出按钮后访问其他接口

2.用户不登出直接访问其他接口

测试结果测试操作1无法访问其他接口，测试操作2可以访问其他接口，说明用户可以正常登出本系统，系统认证有效。

表6.2 用户登出功能测试用例

6.1.3 用户注册功能测试用例设计

{62%：用户注册功能测试用例的设计如表6.3所示。}

用 例 名 称用户注册

标识符T3

{59%：测试目的测试系统的注册功能是否有效}

{52%：测试操作1.使用已注册过的用户名进行注册}

{56%：2.使用未注册过的用户名进行注册}

测试结果测试操作1注册失败，测试操作2注册成功，说明系统会在用户注册时数据库表，说明了注册功能有效。

表6.3 用户注册功能测试用例

{49%：6.1.4 管理员冻结普通用户功能测试用例设计}

{53%：管理员冻结普通用户功能测试用例设计如表6.4所示。}

用 例 名 称冻结用户

标识符T4

{52%：测试目的测试管理员用户冻结普通用户的功能是否有效}

测试操作1.状态不为冻结的普通用户登录系统

2.登录管理员账号点击冻结按钮将该用户的状态设置为冻结后，该用户再次登录系统

测试结果测试操作1中普通用户可以登录成功，测试操作2中用户无法登录，说明管理员冻结普通用户的功能有效。

{55%：表6.4 管理员冻结普通用户功能测试用例}

{55%：6.1.5 管理员激活普通用户功能测试用例设计}

{55%：管理员激活普通用户功能测试用例设计如表6.5所示。}

6.1.6 上传文件功能测试用例设计

{59%：上传文件功能测试用例设计如表6.6所示。}

6.1.7 读取文件功能测试用例设计

{51%：读取文件功能测试用例设计如表6.7所示。}

6.1.8 启动任务功能测试用例设计

{58%：启动任务功能测试用例设计如表6.8所示。}

用 例 名 称激活用户

标识符T5

{52%：测试目的测试管理员用户激活普通用户的功能是否有效}

测试操作1.状态为冻结的普通用户登录系统

2.登录管理员账号点击激活按钮将该用户的状态设置为激活后，该用户再次登录系统

测试结果测试操作1中的普通用户无法登录，测试操作2中的用户可以登录成功，说明管理员激活普通用户的功能有效。

{55%：表6.5 管理员激活普通用户功能测试用例}

用 例 名 称上传文件

标识符T6

测试目的1.测试是否可以上传文件

2.测试是否可以拦截其他类型的文件

测试操作1.上传test.sql文件

2.上传test.txt文件

3.上传test.png文件

{42%：测试结果1.测试操作1上传成功，说明系统可以上传正常的sql文件；}

2.测试操作2上传成功，说明系统可以上传正常的txt文件；

3.测试操作3上传失败，说明系统可以拦截其他类型的文件。

表6.6 上传文件功能测试用例

用 例 名 称读取文件

标识符T7

{56%：测试目的测试系统是否可以正常读取文件}

{48%：测试操作1.选择一个未测试的任务，点击读取文件}

2.选择一个已测试的任务，点击读取文件

测试结果测试操作1可以正常读取文件，测试操作2可以正常读取文件，说明系统可以正常读取文件。

表6.7 读取文件功能测试用例

用 例 名 称启动任务

标识符T8

{57%：测试目的测试系统是否能够正常启动任务}

{58%：测试操作选择一个未启动的任务，点击启动任务}

{47%：测试结果任务执行成功，并生成了测试报告，说明系统可以正常启动任务。}

表6.8 启动任务功能测试用例

6.1.9 查看测试报告功能测试用例设计

{55%：查看测试报告功能测试用例设计如表6.9所示。}

用 例 名 称查看测试报告

标识符T9

{57%：测试目的查看系统能否正常查看测试报告}

{52%：测试操作选择一个测试完成的任务，点击查看报告}

{44%：测试结果系统成功跳转到报告页面，说明系统可以正常查看测试报告。}

表6.9 查看测试报告功能测试用例

6.2 测试总结

{44%：6.1节中介绍了几个具有代表性的测试用例，其实本系统有很多测试用例，} 有部分测试用例找出了系统的漏洞并对系统的漏洞进行了修补。 但是由于时间和个人能力所限，系统测试用例的分支覆盖率和行覆盖率都没有达到太高， 所以会有一些隐藏的系统漏洞还未发现，代码也有很多丑陋的地方， 希望能在今后对这些文件进行处理，完善该系统。

7 总结与展望

本文通过协同过滤算法和关联规则挖掘算法来实现对中国移动套餐创建过程中的错误检测，即对SQL语句的错误检测。 {43%：为了提高错误检测的准确率，本文实现的系统中同时利用协同过滤算法和关联规则算法进行错误预测。}

本文所实现的系统通过Django框架搭建，系统使用B/S 模式设计，提高了系统的易用性，简化了系统的开发、维护和使用。 本文使用以上两种数据挖掘算法进行预测，利用协同过滤算法预测创建套餐的 SQL语句中是否遗漏或多余某些数据库表， 利用关联规则挖掘算法预测创建套餐的 SQL语句中的关键属性值是否存在错误。 两个算法都是通过Python语言实现。

虽然两个算法可以通过调整最小支持度和最小置信度等参数提高预测的准确性，但是两个算法本质上都是通过对历史数据的分析， 进而对结果进行预测，因此准确性无法达到百分之百。 {42%：利用关联规则挖掘算法分析 SQL语句中的关键属性时，以及使用协同过滤算法分析数据库表时，} 由于关键属性和套餐包含的数据库表较多，导致程序执行时间较长，所以下一步工作是对这两个算法进行优化， {41%：使用多线程或者分布式的方法并行处理加快处理速度。}