МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Кафедра технологий программирования

ЕМЕЛЬЯНОВ

Дмитрий Викторович

АЛГОРИТМЫ И МОДЕЛИ ПОИСКА В СИСТЕМЕ ЗАКАЗОВ

Дипломная работа

Научный руководитель:

старший преподаватель

М.И. Давидовская

Допущена к защите

«__»____________ 2019 г.

Зав. кафедрой технологий программирования

Доктор технических наук, профессор А.Н. Курбацкий

Минск, 2019

Оглавление

Введение....8

Глава 1. Платформа ATG....10

1.1. Введение в платформу ATG....10

1.2. Инфраструктура Nucleus....11

1.3. ATG Repository....13

Глава 2. Oracle Endeca Commerce....19

2.1. Oracle Endeca Commerce....19

2.2. Структура Endeca Commerce....20

2.3. Endeca Information Transformation Layer (ITL)....20

2.4. Endeca MDEX Engine....22

2.5. Запросы Oracle Commerce Guided Search....23

2.6. Записи Endeca....25

2.7. Измерения Endeca....26

2.8. Описание Endeca Server....30

2.9. Endeca Server Data Model....32

Глава 3. Разработка методов поиска заказов....36

3.1. Алгоритмы поиска....36

3.2. Методы поиска заказов....36

3.3. Интеграция с Endeca в платформе ATG....37

3.4. Создание компонент индексации....38

3.5. Создание REST API....40

3.6. Метод поиска через Repository....41

3.7. Метод поиска через Endeca....41

3.8. Гибридный метод поиска..43

3.9. Результаты работы сервисов поиска..44

3.10. Применение методов в различных моделях......................................45

Заключение..48

Список использованных источников...50

ПРИЛОЖЕНИЕ...51

1.1. Листинг содержания конфигурационного файла соответствия..........51

1.2. Листинг кода класса OrderDroplet...51

1.3. Листинг кода класса OrderSearchParams..53

1.4. Листинг кода класса OrderRepositoryService..56

1.5. Листинг кода класса OrderEndecaService..59

1.6. Листинг кода класса DPDimensionValueCacheRefreshHandler............64

1.7. Листинг кода класса OrderHybridService..69

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ

ITL — Information Transformation Layer (Слой трансформации информации)

LDAP — Lightweight Directory Access Protocol (Облегчённый протокол доступа к каталогам)

JDO — Java Data Objects (Объекты данных Java)

EJB — Enterprise Java Beans (Спецификация разработки серверных компонентов, содержащих бизнес-логику)

JDBC — Java Database Connectivity (Соединение с базами данных на Java)

API — Application Programming Interface (Интерфейс прикладного программирования)

JTA — Java Transaction API (API транзакций Java)

ACL — Access Control List (Список контроля доступа)

РЕФЕРАТ

Дипломная работа, 72 стр., 8 рис., 2 таблицы, 5 источников.

Ключевые слова: ATG, Endeca, MDEX, поисковой механизм, индекс, атрибут, модель, сущность.

Объект исследования: объектом исследования является платформа ATG и поисковой механизм Endeca.

Цель работы: исследовать существующие алгоритмы и методы поиска заказов в системе электронной коммерции на основе платформы ATG, разработать методы поиска заказов, позволяющие улучшить эффективность системы.

Методы исследования: а) теоретические: изучение литературных источников по направлению исследования б) практические: обобщение работ в области оптимизации механизмов поиска, сравнительный анализ существующих решений, проектирование и разработка системы поиска заказов на основе Endeca и ATG.

Результаты работы: разработанные методы поиска заказов, сравнительная характеристика этих методов и модели, в которых их использование актуально.

Область применения: разработка программных продуктов в области электронной коммерции с использованием навигационного поиска.

РЕФЕРАТ

Дыпломная праца, 72 стар., 8 мал., 2 табліцы, 5 крыніц.

Ключавыя словы: ATG, Endeca, MDEX, пошукавы механізм, індэкс, атрыбут, мадэль, сутнасць.

Аб'ект даследавання: аб'ектам даследавання з'яўляецца платформа ATG і пошукавы механізм Endeca.

Мэта працы: даледаваць існуючыя алгарытмы і метады пошуку замоў у сістэме электроннай камерцыі на аснове платформы ATG.

Метады даследавання: а) тэарэтычныя: вывучэнне літаратурных крыніц па накірунку даследавання б) практычныя: абагульненне работ у галіне аптымізацыі механізмаў пошуку, параўнальны аналіз існуючых рашэнняў, праектаванне і распрацоўка сістэмы пошуку заказаў на аснове Endeca і ATG.

Вынікі працы: распрацаваныя метады пошуку заказаў, параўнальная характарыстыка гэтых метадаў і мадэлі, у якіх іх выкарыстанне актуальнае.

Вобласць прымянення: распрацоўка праграмных прадуктаў у галіне электроннай камерцыі з выкарыстаннем навігацыйнага пошуку.

ABSTRACT

Diploma, 72 p., 8 fig., 2 tables, 5 sources.

Keywords: ATG, Endeca, MDEX, search engine, index, attribute, model, entity.

Object of study: the object of study is the ATG platform and the Endeca search engine.

Objective: to investigate existing algorithms and methods for finding orders in the e-commerce system based on the ATG platform.

Research methods: a) theoretical: study of literary sources in the direction of research; b) practical: generalization of work in the field of search engine optimization, comparative analysis of existing solutions, design and development of an order search system based on Endeca and ATG.

Results: developed methods for finding orders, a comparative description of these methods and models in which their use is relevant.

Scope: development of software products in the field of e-commerce using navigation search.

ВВЕДЕНИЕ

При использовании различных систем, сайтов, прикладных программ важнейшей характеристикой для пользователей является время работы, время отклика этих систем на запросы пользователей. Важной составляющей многих систем также является поиск и навигация по структурным частям, составляющим этих систем.

В контексте сайтов в области электронной коммерции, ярким примером которой являются интернет-магазины, необходимо предоставить пользователю удобный и быстрый интерфейс по поиску и навигации по продуктам.

Зачастую нужды владельцев таких сайтов предполагают не только поиск по продуктам, но поиск по другим сущностям таких систем.

В данной работе рассматриваются платформа ATG и поисковой механизм Endeca. Их интеграция обеспечивают полноценный функционал для системы электронной коммерции и позволяет обеспечить поиск по продуктам. На основе этого, в данной работе проектируются и разрабатываются методы поиска по заказам с использованием данного поискового механизма, а также сравнение разработанных методов со стандартным поиском, представленным в платформе ATG.

В главе 1 описывается платформа ATG, ее возможности и составляющие части, основные механизмы и компоненты. Описана структура ATG. Описан уровень, обеспечивающий доступ к базе данных.

В главе 2 рассмотрены сущности и их соотношение в Endeca Commerce. Описана структура и взаимодействие внутренних компонентов поискового механизма Endeca Commerce. Также рассмотрен Endeca Server, способ, которым он хранит и оперирует сущностями Endeca.

В главе 3 описан процесс разработки методов поиска, их сравнение. Проведен анализ преимуществ и недостатков разработанных методов на

основе их сравнения. Построены модели, для которых данные методы актуальны.

В рамках работы проводится исследование принципа хранения данных в Endeca Server и возможность повышения скорость выборки данных по сравнению с базой данных. На основе полученных данных реализованы методы поиска, использующие как запросы к поисковому механизму Endeca, так и к базе данных.

Разработанные методы позволяют использовать их для ускорения поиска по заказам, что в свою очередь, ускорит работу с системой, в которой данные методы внедрены.

ГЛАВА 1. ПЛАТФОРМА ATG

1.1. Введение в платформу ATG

Oracle ATG Web Commerce — это технологическая платформа E-commerce, предлагаемая Oracle. Она обеспечивает полный стек программного обеспечения для создания приложений электронной коммерции. Некоторые из ключевых функций, доступных в этом стеке, включают:

- Commerce Reference Store: настраиваемое приложение для магазина электронной коммерции.
- Multi-Site Features: совместное использование ресурсов между несколькими сайтами электронной коммерции.
- Функции полного мерчендайзинга[1], используя приложение администрирования контента ATG.
- Модуль навигационного поиска с помощью интеграции с Endeca Search.

Oracle ATG Web Commerce предоставляет открытую серверную среду для создания и развертывания динамических, персонализированных приложений для проводных и других каналов связи, таких как электронные и беспроводные устройства. Приложения Oracle ATG Web Commerce реализуют модель разработки компонентов на основе JavaBeans и JSP. Разработчики собирают приложения из компонентов (на основе стандартных классов Oracle ATG Web Commerce или пользовательских классов Java), связывая их вместе с конфигурационными файлами в Nucleus (см. §1.2), открытой инфраструктуре объектов Oracle ATG Web Commerce. Дизайнеры страниц создают интерфейс для приложения из JSP, которые используют библиотеку тегов DSP Oracle ATG Web Commerce. Библиотека тегов DSP

1 Мерчанда́йзинг — часть процесса маркетинга, определяющая методику продажи товара в магазине. Призван определять набор продаваемых в розничном магазине товаров, способы выкладки товаров, снабжение их рекламными материалами, цены.

позволяет встраивать компоненты Nucleus в JSP и использовать эти компоненты для рендеринга динамического контента.

Продукты Oracle ATG Web Commerce упакованы в виде отдельных модулей приложений. Модуль приложения группирует файлы приложений и файлы конфигурации в дискретный пакет для развертывания. Модули приложений существуют в установке Oracle ATG Web Commerce как набор каталогов, определённых файлом манифеста. При сборке приложения эти модули анализируются для определения CLASSPATH, пути конфигурации и межмодульных зависимостей.

Модули приложений обеспечивают следующие основные функции:

- Компоненты приложения упакованы в простой, модульной структуре каталогов.
- Модули могут ссылаться на другие модули, от которых они зависят, и которые могут быть автоматически загружены.
- Правильный путь класса, путь конфигурации и основной класс Java для заданного набора модулей динамически вычисляются как для клиента, так и для сервера.
- Обновлённые модули автоматически загружаются с сервера на клиент.

1.2. Инфраструктура Nucleus

Nucleus — это компонентная модель Oracle ATG Web Commerce для создания приложений из JavaBeans. Nucleus позволяет собирать приложения через простые файлы конфигурации, которые определяют, какие компоненты используются приложением, какие параметры используются для инициализации этих компонентов, и как эти компоненты соединяются друг с другом.

Nucleus сам по себе не обеспечивает функций, специфичных для приложения. Компоненты JavaBean реализуют все функциональные возможности приложения. Nucleus — это механизм, который даёт этим компонентам «место для жизни», и способ для этих компонентов находить

друг друга [1].

Nucleus организует компоненты приложения в иерархию и присваивает имя каждому компоненту, основываясь на его позиции в иерархии. Например, компонент с именем /services/logs/FileLogger представляет собой компонент, называемый FileLogger, содержащий компонент контейнера, называемый журналами, который сам содержит компонент контейнера, называемый сервисами. Компонент услуг содержится в корневой компоненте иерархии, которая является Nucleus. Компоненты в иерархии могут ссылаться друг на друга по имени. Это включает в себя как абсолютные имена, такие как /services/logs/FileLogger, и относительные имена, такие как ../servers/HttpServer.

Nucleus также берет на себя задачу создания и инициализации компонентов. Приложение не обязательно должно содержать код, который создаёт компонент и добавляет его в пространство имён Nucleus. Вместо этого можем написать файл конфигурации, который указывает класс компонента и начальные значения свойств компонента. В первый раз, когда этот компонент ссылается на имя, Nucleus находит файл конфигурации компонента, создаёт компонент на основе значений в этом файле конфигурации и добавляет компонент в пространство имен Nucleus.

Nucleus обеспечивает простой путь для написания новых компонентов. Любой объект Java с пустым конструктором может выступать в качестве компонента в Nucleus, поэтому писать новый компонент Nucleus так же просто, как писать класс Java. Соблюдая стандарты JavaBeans для определения полей и методов, класс Java может использовать механизм автоматического создания и настройки Nucleus. Используя различные интерфейсы, компонент Nucleus также может воспользоваться возможностями Nucleus и уведомлениями.

1.3. ATG Repository

Доступ к данным — большая часть большинства интернет-

приложений. ATG Data Anywhere Architecture ™ обеспечивает единое представление контента и данных для бизнеса для организаций и их клиентов. Ядром архитектуры ATG Data Anywhere является API репозитория. С помощью API репозитория можно использовать один подход к доступу к разрозненным типам данных, включая базы данных SQL, каталоги LDAP, системы управления контентом и файловые системы.

В ATG Data Anywhere логика приложения, созданная разработчиками, использует тот же подход для взаимодействия с данными независимо от источника этих данных. Одним из самых сильных аспектов этой архитектуры является то, что источник данных скрыт за абстракцией репозитория ATG. Легко перейти от реляционного источника данных к каталогу LDAP, так как ни одна из прикладных логик не нуждается в изменении. После получения данных из источника данных он преобразуется в объектно-ориентированное представление. Манипулирование данными можно выполнить с помощью простых методов getPropertyValue и setPropertyValue. API-интерфейс репозитория тесно связан с API-интерфейсами ATG, поэтому можно извлекать элементы из репозитория на основе различных правил таргетинга[2], а также получать определенные идентифицированные элементы [1].

На рисунке 1 представлен обзор архитектуры ATG Data Anywhere на высоком уровне.

2 Таргетинг – механизм выделения части данных из набора согласно заданным критериям.

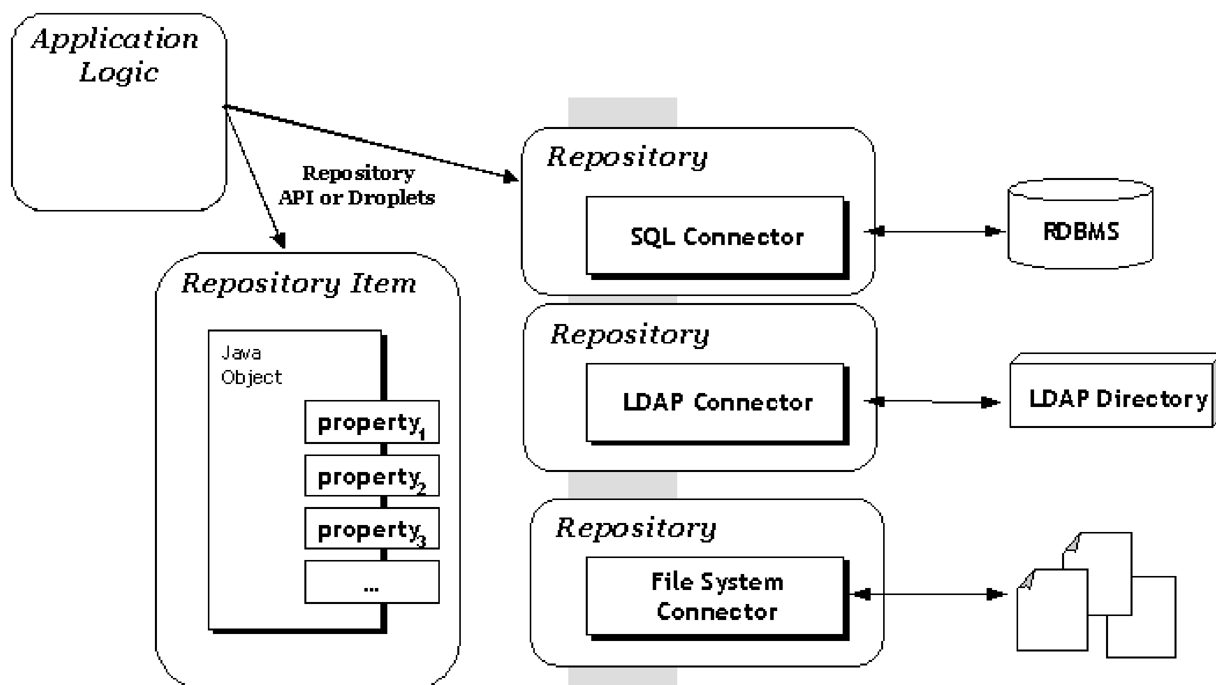

Рисунок 1. Архитектура ATG Repository

Архитектура ATG Data Anywhere предлагает несколько преимуществ по сравнению со стандартными методами доступа к данным, такими как Java Data Objects (JDO), Enterprise JavaBeans (EJB) и Java Database Connectivity (JDBC) [1]. Среди различий:

- Независимость источника данных.

Архитектура ATG Data Anywhere обеспечивает доступ к реляционным системам управления базами данных, каталогам LDAP и файловым системам с использованием тех же интерфейсов. Это изолирует разработчиков приложений от изменений схемы и механизма хранения. Данные могут даже перемещаться из реляционной базы данных в каталог LDAP без необходимости перекодировки. Объекты данных Java поддерживают независимость источника данных, но поставщики могут обеспечить реализацию LDAP.

- Меньше строк кода Java.

Меньше кода приводит к более быстрому выходу на рынок и снижению затрат на обслуживание. Стойкие типы данных, созданные с помощью Anywhere ATG, описываются в XML-файле без необходимости

использования кода Java.

- Единое представление обо всех взаимодействиях клиентов.

Единое представление данных о клиентах (собранное веб-приложениями, приложениями call-центра и системами ERP) может быть предоставлено без копирования данных в центральный источник данных. Это унифицированное представление данных о клиентах приводит к последовательной работе с клиентами.

- Максимальная производительность.

Интеллектуальное кэширование объектов данных обеспечивает отличную производительность и своевременные и точные результаты. Стандарты JDO и EJB основаны на реализации кэширования поставщиков, которые могут быть недоступны.

- Упрощенный транзакционный контроль.

Ключом к общей производительности системы является минимизация влияния транзакций при сохранении целостности ваших данных. В дополнение к полной поддержке API Java Transaction API (JTA), ATG Data Anywhere позволяет разработчикам программного обеспечения контролировать объем транзакций с теми транзакционными режимами, которые необходимы.

- Детализированный контроль.

Можно контролировать, кто имеет доступ к каким данным в типе данных, объекте данных, вплоть до отдельного свойства с помощью списков контроля доступа (ACL).

- Интеграция с наборами продуктов доступа ATG.

Приложения персонализации ATG, сценарии, коммерция, портал и приложения для администрирования контента используют репозитории для доступа к данным. Команда разработчиков может свободно использовать EJB по технологии ATG, но самый простой способ использовать инвестиции в технологии ATG — это следовать примеру, установленному наборами

решений. Набор решений ATG удовлетворяет всем требованиям доступа к данным с репозиториями.

API-интерфейс репозитория ATG (atg.repository. *) является основой постоянного хранения объектов, профилирования пользователей и таргетинга контента в продуктах ATG. Репозиторий — это уровень доступа к данным, который определяет общее представление хранилища данных. Разработчики приложений используют это общее представление для доступа к данным с использованием только таких интерфейсов, как репозиторий и объект репозитория. Хранилища получают доступ к базовому устройству хранения данных через коннектор, который переводит запрос в любые вызовы, необходимые для доступа к этому конкретному хранилищу данных. Коннекторы для реляционных баз данных и каталогов LDAP предоставляются в готовом виде. Коннекторы используют открытый, опубликованный интерфейс, поэтому при необходимости могут быть добавлены дополнительные настраиваемые коннекторы [1].

Разработчики используют репозитории для создания, запроса, изменения и удаления элементов репозитория. Элемент репозитория похож на JavaBean, но его свойства определяются динамически во время выполнения. С точки зрения разработчика доступные свойства в определённом элементе репозитория зависят от типа элемента, с которым они работают. Один элемент может представлять профиль пользователя (имя, адрес, номер телефона), а другой может представлять метаданные, связанные с новостной статьей (автор, ключевые слова, резюме).

Цель системы интерфейса репозитория — обеспечить единую перспективу доступа к данным. Например, разработчики могут использовать правила таргетинга с тем же синтаксисом, чтобы найти людей или контент.

Приложения, которые используют только интерфейсы репозитория для доступа к данным, могут взаимодействовать с любым количеством внутренних хранилищ данных только через конфигурацию. Разработчикам

не нужно писать один интерфейс или класс Java, чтобы добавить новый постоянный тип данных в приложение.

Каждый репозиторий подключается к одному хранилищу данных, но несколько хранилищ могут сосуществовать в продуктах ATG, где различные приложения и подсистемы используют разные репозитории или используют один и тот же репозиторий. Например, система безопасности может быть направлена на сохранение своего списка имен пользователей и паролей в базе данных SQL, указывая систему безопасности в репозитории SQL. Позже систему безопасности можно изменить, чтобы использовать каталог LDAP, перенастроив ее, чтобы она указывала на репозиторий LDAP. Какие репозитории планируется использовать, зависит от потребностей доступа к данным приложения, включая возможное требование доступа к данным в устаревшем хранилище данных.

Платформа ATG включает следующие модели для репозиториев:

- **Репозитории SQL** используют коннектор универсального SQL-адаптера ATG (GSA) для сопоставления ATG и данных в базе данных SQL. Можно использовать репозиторий SQL для доступа к контенту, профилям пользователей, информации о безопасности приложений и т. д.
- **Репозиторий профилей SQL**, включенный в модуль персонализации ATG, использует коннектор Generic SQL Adapter для сопоставления пользовательских данных, которые содержатся в базе данных SQL.
- **Репозитории LDAP** используют коннектор LDAP ATG для доступа к пользовательским данным в каталоге LDAP.
- **Композитные репозитории** позволяют использовать несколько хранилищ данных в качестве источников для одного репозитория.
- **Версионные репозитории** расширяют репозитории SQL и используются в администрировании контента ATG.

ВЫВОДЫ

В главе 1 описана платформа ATG, ее возможности и составляющие части, основные механизмы и компоненты. Описан механизм Nucleus, возможности и преимущества уровня репозитория.

ГЛАВА 2. ORACLE ENDECA COMMERCE

2.1. Oracle Endeca Commerce

Oracle Endeca Commerce — это поисковая система для электронной коммерции, которая предоставляет персонализированные возможности навигационного поиска, помогает пользователям найти именно тот продукт, который они ищут. Поиск в любом приложении электронной коммерции очень важен, поскольку 60% пользователей покидают сайт после неудачного поиска.

Oracle Endeca Commerce предлагает следующие возможности поиска Endeca:

- MDEX Engine — высокопроизводительная поисковая система без сохранения состояния.
- Моделирование данных или логическая организация данных сделаны простыми. Они спроектированы как «загружай и работай», с небольшим предварительным моделированием данных и дизайном, оптимизированным для быстрого обнаружения данных из разрозненных и неравномерных источников данных.
- Масштабируемая многосайтовая архитектура, которая использует контент и данные из любого источника и не ограничивается каталогом продуктов.
- Поисковая система Endeca имеет встроенные возможности SEO, поиска и управляемой навигации, которые позволяют клиентам легко найти сайт и то, что он ищет на сайте.
- С пользовательскими сегментами Endeca, персонализированным поисковым опытом, который специально адаптирован к интересам человека на основе последних действий, может быть реализован профиль поведения.

2.2. Структура Endeca Commerce

Oracle Endeca Commerce состоит из трех основных компонентов. Компоненты представлены на рисунке 2.

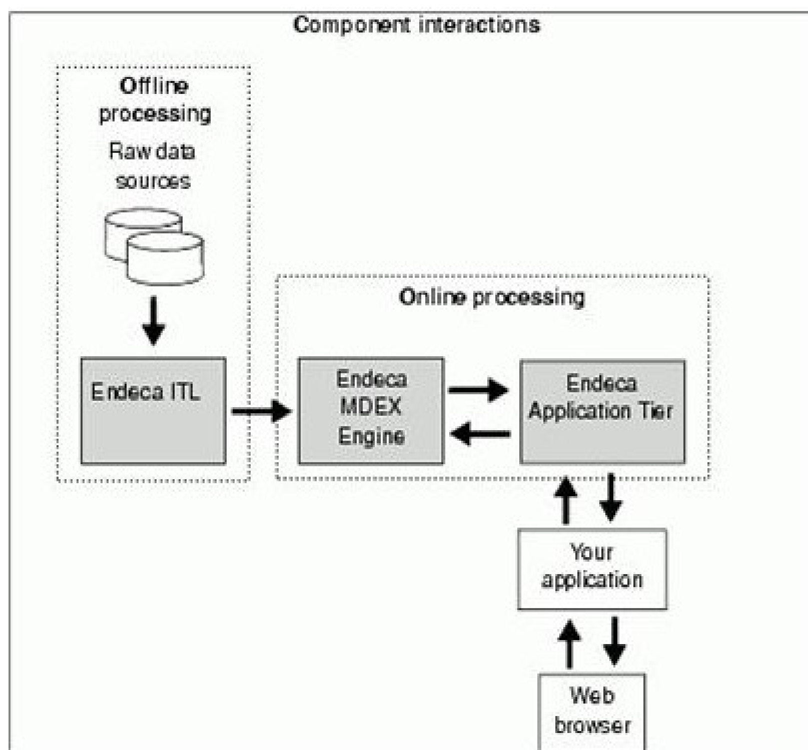

Рисунок 2. Структура Endeca Commerce

Эти компоненты:

1. Endeca Information Transformation Layer (ITL).
2. Endeca MDEX Engine.
3. Endeca Application Tier.

2.3. Endeca Information Transformation Layer (ITL)

Считывает исходные данные и обрабатывает их в виде набора индексов Oracle Endeca MDEX Engine.

ITL состоит из Content Acquisition System и Data Foundry (который включает в себя программы для работы с данными, такие как Forge).

Content Acquisition System включает в себя Endeca Web Crawler и Endeca CAS Server, а также большой набор упакованных адаптеров.

Эти компоненты сканируют неструктурированные источники контента и принимают структурированные данные. Источники контента включают в себя реляционные базы данных, файловые серверы, системы управления контентом и корпоративные системы, такие как Enterprise Resource Planning (ERP) и Master Data Management (MDM).

Упакованные адаптеры используются в самых распространенных системах, включая JDBC и ODBC. Content Adapter Development Kit (CADK) позволяет разработчикам создавать собственные адаптеры и манипуляторы Java.

Endeca Data Foundry собирает информацию и преобразует ее в записи Endeca и индексы MDEX Engine.

На этапе обработки данных Data Foundry:

- Импортирует исходные данные.
- Отмечает их значениями измерения, используемыми для навигации, и свойствами Endeca, используемыми для отображения.
- Хранит помеченные данные вместе с характеристиками значения измерения и любыми правилами конфигурации в виде записей Endeca, готовых для индексации.
- Индексирует записи Endeca, созданные на этапе обработки данных, и создает набор индексов в формате Endeca MDEX Engine.

Компонент Data Foundry состоит из двух основных программ, таких как Forge и Dgidx.

- Forge — это программа обработки данных, которая преобразует исходные данные в стандартизированные, маркированные записи Endeca.
- Dgidx — это программа индексирования, которая читает помеченные записи Endeca, подготовленные Forge, и создает собственные индексы

для Endeca MDEX Engine.

На рисунке 3 представлены процессы ITL и их взаимодействие.

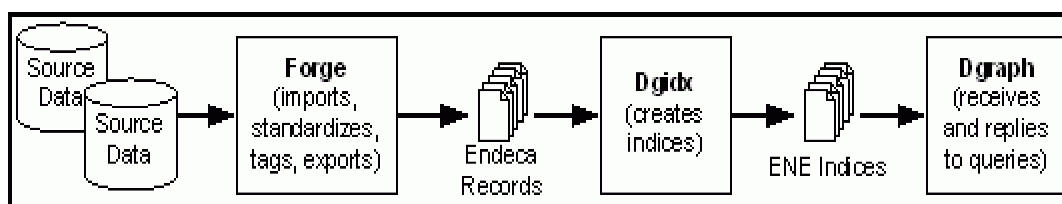

Рисунок 3. Процессы Information Transformation Layer

2.4. Endeca MDEX Engine

Endeca MDEX Engine — это механизм индексирования и запросов, который является основой Endeca. Механизм MDEX использует собственные структуры данных и алгоритмы, которые позволяют ему в режиме реального времени отвечать на запросы клиентов. Механизм MDEX хранит индексы, которые были созданы на уровне Endeca Information Transformation Layer (ITL). После сохранения индексов механизм MDEX получает запросы клиентов через уровень приложений, запрашивает индексы и затем возвращает результаты [2]. Схема взаимодействия с MDEX Endine представлена на рисунке 4.

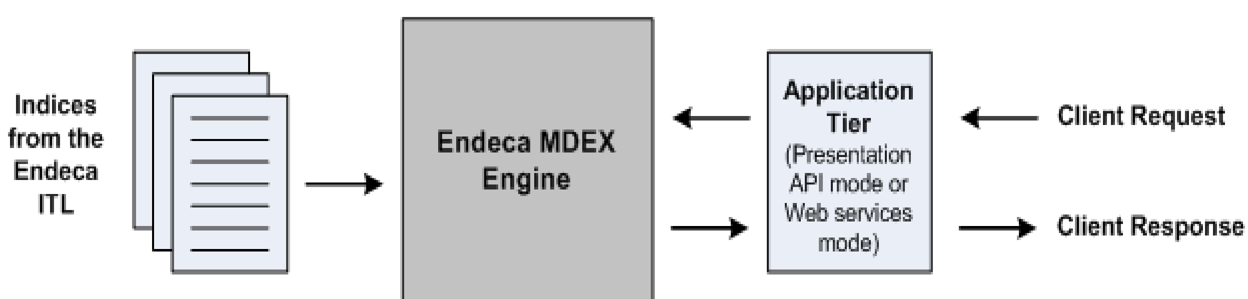

Рисунок 4. Схема взаимодействия MDEX Engine

MDEX Engine разработан так, что он не имеет состояния. Этот дизайн требует, чтобы полный запрос отправлялся в MDEX Engine для каждого запроса. Конструкция MDEX Engine без учета состояния позволяет добавлять серверы MDEX Engine для балансировки нагрузки и резервирования.

Поскольку MDEX Engine не имеет состояния, любая реплика MDEX Engine на одном сервере может отвечать на запросы независимо от реплики на других серверах MDEX Engine.

Следовательно, добавление реплик модулей MDEX на дополнительные серверы обеспечивает избыточность и улучшает время ответа на запрос. То есть, если какой-либо конкретный сервер выходит из строя, реплика MDEX Engine обеспечивает избыточность, позволяя другим серверам в реализации продолжать отвечать на запросы. Кроме того, общее время отклика улучшается за счет использования балансировщиков нагрузки для распределения запросов к реплике MDEX Engine на любом из дополнительных серверов.

Механизм MDEX, который является ядром Oracle Endeca Commerce, состоит из индексатора (Dgidx), Dgraph и Agraph. Хотя индексатор (также известный как Dgidx) устанавливается как часть пакета движка MDEX, он фактически является частью процесса ITL.

2.5. Запросы Oracle Commerce Guided Search

Все результаты запроса, возвращаемые из MDEX Engine, содержат два типа информации:

1. Соответствующие результаты для запроса (например, набор записей или отдельная запись).

2. Вспомогательная информация для построения последующих запросов.

Последующая информация запроса позволяет пользователям уточнять или расширять свой запрос и, соответственно, результаты их запросов. Метод, который MDEX Engine использует для вычисления этой информации, исключает недопустимые последующие запросы (тупики). Исключение путей навигации, которые никуда не ведут, и предоставление соответствующих вариантов выбора следующего шага — две из основных функций, которые отличают решения управляемого поиска от других типов поисковых

реализаций [2].

Oracle Commerce Guided Search использует два типа запросов, навигационные запросы и запросы поиска по ключевым словам:

- Навигационные запросы возвращают набор записей, основанных на характеристиках записи, определенных приложением (например, тип вина или регион в интернет-магазине вина), а также любую последующую информацию о запросах.

- Запросы поиска по ключевому слову возвращают набор записей или измерений на основе пользовательского ключевого слова плюс любые последующие запросы.

Навигационные запросы и поисковые запросы, по ключевым словам, дополняют друг друга. Фактически, поисковый запрос по ключевым словам является специализированной формой навигационного запроса, а структуры данных для результатов двух запросов идентичны: набор записей и последующая информация запроса.

Пользователи могут выполнять комбинацию навигационных запросов и запросов поиска по ключевым словам, чтобы перейти к желаемой записи, установленной таким образом, которая наиболее соответствует их запросу. Например, пользователи могут выполнить запрос поиска по ключевым словам для извлечения набора записей, а затем использовать следующий навигационный запрос для уточнения этого набора записей. Применение запросов в обратном порядке так же используется.

Данные в приложении «Guided Search» имеют как физическую структуру, так и логическую структуру, которая поддерживает запросы MDEX Engine.

Чтобы понять разницу между физической структурой и логической структурой, рассмотрим реляционную базу данных. Реляционная база данных имеет набор таблиц, каждый из которых содержит свои собственные

данные. Существуют связи между таблицами, которые позволяют создавать логические записи из данных, распределенных по нескольким таблицам. Физическая структура базы данных представляет собой набор отдельных таблиц, а ее логическая структура представляет собой набор записей, данные которых извлекаются из этих таблиц.

Реализация управляемого поиска также накладывается на физическую структуру и логическую структуру. Records (записи), Dimensions (измерения) и Properties (свойства) описывают эти структуры и то, как MDEX Engine использует их для ответа на запросы.

2.6. Записи Endeca

Записи Endeca содержат данные, которые пользователи просматривают и по которым выполняют поиск.

Записи Endeca основаны на традиционных записях в исходной базе данных. Записи исходной базы данных обычно содержат информацию, такую как записи клиентов в приложении CRM, взаимные фонды в оценщике фонда или сведения о бутылках вина в магазине и др.

Записи исходной базы данных хранят эту информацию в одной или нескольких парах ключ / значение, известных как свойства. Эта информация становится доступной для приложения при преобразовании записей исходной базы данных в записи Endeca. Чтобы преобразовать записи исходной базы данных в записи Endeca, необходимо сопоставить свойства исходной записи свойствам записей Endeca.

Таким образом, свойства записи Endeca соответствуют свойствам записей исходной базы данных. Подобно свойствам записи источника, свойства Endeca являются парами ключ / значение.

На рисунке 5 показаны пары ключ / значение в простой записи Endeca [2].

Record example

Record ID: 0001
Name: House White
Wine Type: Chardonnay
Vineyard: Sonoma Vineyards
Year: 1996
Price: $45.00
Rating: 96
Description: Intense, with complex earthy pear, fig, melon, citrus and hazelnut flavors that are remarkably elegant and sophisticated.

Рисунок 5. Endeca record

Одна запись Endeca может соответствовать любому количеству записей источников. Например, предположим, что четыре разные записи источника относятся к одной книге в разных форматах: в твердом переплете, в мягкой обложке, большой печати и аудио. Можно настроить приложение «Guided Search», чтобы объединить информацию из этих четырех исходных записей в одну запись Endeca.

2.7. Измерения Endeca

Измерения — это логические категории, которые позволяют организовать записи Endeca в структуры, которые клиенты могут перемещать, чтобы найти информацию о продуктах или услугах, которые они могут захотеть приобрести.

Измерения — это иерархия значений измерения. Измерения в целом обычно соответствует общей категории продуктов или услуг. Значения измерения содержат все более конкретную информацию о продуктах и услугах. Чем точнее значение измерения, тем ниже они находятся в иерархии [2].

Наибольшее значение измерения в измерениях называется корень измерений. Корень измерений служит его именем.

Каждое значение измерения может иметь одно или несколько значений дочернего измерения; значение измерения с дочерними значениями измерения называется родительское значение измерения. Дочернее значение

измерения может иметь только одно родительское значение измерения. Значения измерения, являющиеся дочерними элементами одного и того же родителя, известны как соседние значения измерения. Соседние значения измерения не могут быть одинаковыми. Однако значения измерения, которые не являются соседними, могут быть одинаковыми даже в одном измерении [2].

Значения измерения, у которых нет дочерних элементов, называется лист значения измерения. Лист значения измерения обычно содержит информацию о конкретных продуктах и услугах. Например, значение измерения может представлять диапазон цен, а лист значения измерения — его дочерние элементы — могут представлять собой отдельные продукты, цены которых попадают в этот диапазон. На рисунке 6 представлена иерархия значений измерения.

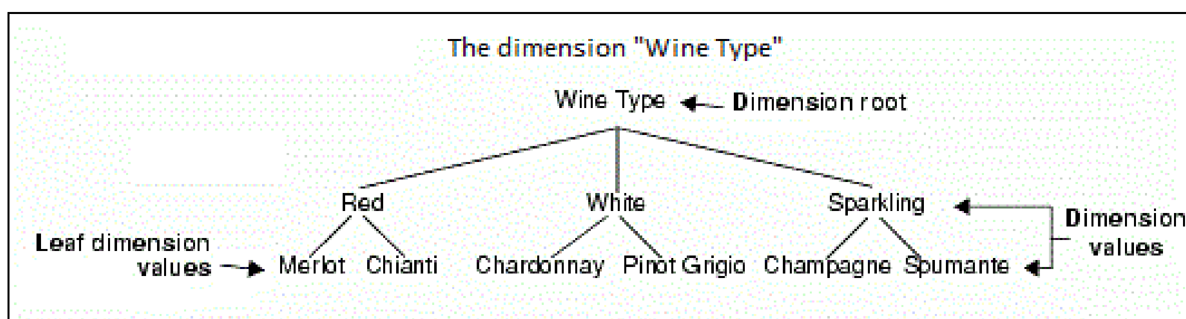

Рисунок 6. Структура значений измерения

Записи могут быть организованы в иерархии с возможностью поиска, помечая их значениями измерения. Записи обычно помечены листами значений измерения, но могут быть помечены значениями нелистового значения измерения для специальных целей.

Пометка записи значением измерения делает следующие вещи:

- Определяется местоположение записи в соответствующем значении измерения. В приведенном ниже примере записи Endeca для бутылок A и B помечены значением измерения Red в измерении тип вина, в то время как записи Endeca для бутылок C и D отмечены значением измерения White и т. д.

- Запись идентифицируется как действительный результат, когда его значение измерения выбрано в навигационном запросе.

На рисунке 5 навигационный запрос по значению измерения Red создает набор результатов, содержащий бутылки A и B [2]. На рисунке 7 представлена структура измерений и соответствующие им результаты запроса.

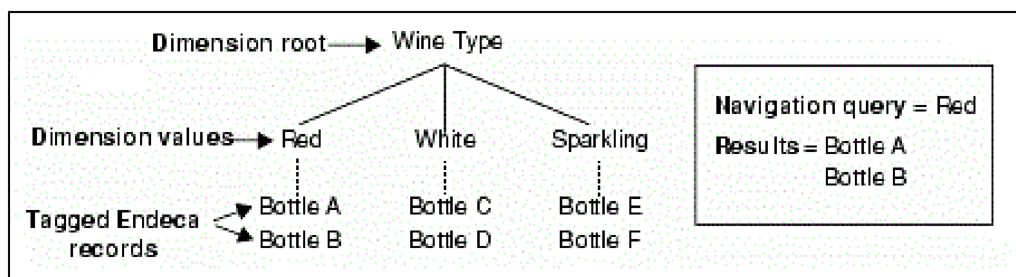

Рисунок 7. Пример структуры значений измерения

Можно ассоциировать навигационный запрос как «вернуть все записи, которые организованы в соответствии с значением измерения xxx в значении измерении yyy». Или можно думать об этом как «вернуть все записи, которые были помечены значением измерения xxx в значении измерения yyy». В любом случае результат будет таким же.

Запись может быть помечена любым количеством значений измерения из любого числа значений измерения. Это позволяет пользователям перемещаться по записям с помощью любого значения измерения или комбинации значений измерения, которые они выбирают.

Прежде чем преобразовать запись источника в запись Endeca, необходимо сделать одну из трех вещей для каждого из свойств исходной записи:

1. Сопоставить свойство источника с измерением Endeca.
2. Сопоставить свойство источника с значением измерения.
3. Сопоставьте свойство источника с свойством записи Endeca.

Значения свойств Endeca — это описательная информация, которая

может быть отображена с записью, когда пользователь обратился к записи, через управляемый поиск или навигацию. Свойства Endeca не могут использоваться для классификации записей или для перехода к ним.

Если исходное свойство не содержит информации, подходящей для навигации, поиска или отображения, можно указать, чтобы реализация управляемого поиска игнорировала ее во время процесса преобразования данных.

Когда все свойства исходной записи были сопоставлены, запись источника может быть преобразована в запись Endeca, состоящую из свойств Endeca и значений измерения Endeca.

Свойства и измерения Endeca:

- Обычно генерируются из свойств исходной записи, с которыми они были сопоставлены.
- Состоит из пар ключ / значение (имя свойства / значение свойства, имя измерения / значение измерения).
- Можно искать и отображать.

Хотя свойства Endeca и значения измерения Endeca обладают многими характеристиками, для навигации по руководству могут использоваться только значения измерения. MDEX Engine генерирует индексы на основе сопоставлений, и только индексы для значений измерения могут использоваться для навигации.

Таким образом, когда сопоставляется исходное свойство со свойством Endeca или значением измерения Endeca, необходимо рассмотреть, подходит ли значение этого свойства для использования в качестве навигационного термина. Если подходит, необходимо сопоставить его со значением измерения Endeca. Если не подходит, сопоставить его со свойством Endeca. Например, свойство источника, значение которого является «Тип вина», полезно для навигации, поскольку оно позволяет идентифицировать набор

вин в соответствии с типом: красный, белый, искрящийся и т. д. Свойства источника, содержащие этот тип информации, должны быть сопоставлены со значением измерения Endeca.

С другой стороны, свойство, значение которого является длинным описанием вина, не подходит для использования в качестве навигационного термина и по этой причине должно быть сопоставлено со свойством Endeca, а не со значением измерения Endeca. Например, конечные пользователи не запрашивают вина, которые являются «интенсивными, со сложными ароматами грушевых, фиговых, дыни, цитрусовых и фундука, которые необычайно элегантны и изощренны». Этот тип информации полезен для отображения, когда пользователь перешел к записи, а свойства источника, содержащие этот тип информации, должны отображаться в свойствах Endeca.

В свойствах Endeca часто содержится более конкретная информация о записи, чем в значении измерения. Например, значение измерения диапазона цен полезно для навигации — дайте мне все бутылки вина стоимостью от 10 до 20 долларов США, но это точная цена каждой бутылки, которую пользователь хочет увидеть при просмотре отдельных записей. Обычная реализация для этого типа приложений использует значение измерения диапазона цен для навигации и свойство Price, которое отображается, когда запись бутылки будет отображена.

2.8. Описание Endeca Server

Еще одно название, под которым фигурирует MDEX Engine (или название которое использовалось ранее), — это Endeca Server.

Oracle Endeca Server — это гибридный поисково-аналитический механизм, который организует сложные и разнообразные данные из разнородных источников. В основе Endeca Server модель данных, похожая на NoSQL, и архитектура в памяти Endeca Server создают чрезвычайно гибкую среду для обработки сложных комбинаций данных, устраняя необходимость в сложном предварительном моделировании данных и предлагая предельную

производительность при масштабировании [3].

Сервер Oracle Endeca предназначен для размещения нескольких доменов данных Endeca. Сервер Oracle Endeca поддерживает индекс записей для домена данных в памяти, принимает запросы, выполняет их по сохраненному индексу и возвращает результаты [3].

Домен данных — это логическая единица данных и метаданных, управляемая сервером Endeca. Благодаря своим интерфейсам сервер Endeca позволяет загружать данные, конфигурировать и запрашивать предметную область. Домен данных может наложить порядок на подмножества своих данных через коллекции. Домен данных — это самая большая единица данных, к которой сервер Endeca позволяет создавать запросы. Он представляет собой дискретный набор данных и включает в себя индексированные записи данных и системные записи. Приложения, желающие сопоставлять, объединять или отображать данные из нескольких доменов данных, должны делать это сами.

Dgraph — это имя процесса, созданного на сервере Oracle Endeca для каждой области данных. Каждый процесс Dgraph обрабатывает запросы, сделанные к предметной области. Процесс Dgraph сервера Oracle Endeca использует запатентованные структуры данных и алгоритмы, которые позволяют ему в реальном времени отвечать на запросы клиентов. Процесс Dgraph хранит индекс, созданный после приема исходных данных. После сохранения индекса он получает запросы клиентов через уровень приложений, запрашивает файлы данных и возвращает результаты через сервер Oracle Endeca. Связь между сервером Endeca и процессом Dgraph по умолчанию безопасна [3].

После создания домена данных нужно всего лишь использовать имя домена данных для управления им. Не нужно знать, на каком порту запущены процессы Dgraph для домена данных, поскольку сервер Endeca отслеживает эту информацию. Именная ссылка на домены данных

значительно упрощает их включение и отключение, а также выполнение других операций по управлению доменом данных.

Сервер Endeca включает в себя набор команд, доступных через командную оболочку Endeca, с помощью которых создаются и управляются домены данных. При желании можно использовать веб-сервисы сервера Endeca для этой цели.

Сервер Oracle Endeca предназначен для работы без сохранения состояния. Эта схема требует отправки полного запроса для каждого запроса для каждого домена данных Endeca, размещенного на сервере Endeca. Конструкция без сохранения состояния облегчает добавление нескольких экземпляров Oracle Endeca Server для балансировки нагрузки и избыточности — любой экземпляр кластера Oracle Endeca Server, на котором размещается домен данных, может отвечать на запросы независимо от других экземпляров, используя общий индекс домена данных.

Следовательно, для каждой области данных Endeca настройка дополнительных процессов Dgraph в качестве узлов в кластере области данных повышает доступность обработки запросов для области данных. Если узел в кластере домена данных выходит из строя, по крайней мере один из Dgraphs, запущенных в кластере, продолжает отвечать на запросы.

2.9. Endeca Server Data Model

Oracle Endeca Server использует уникальную гибкую модель данных, которая уменьшает необходимость в предварительном моделировании, обеспечение интеграции разнообразных и изменяющихся данных при поддержке широкого, непредсказуемого поиска, исследования и анализ потребностей бизнес-пользователей.

Сервер Endeca организует данные в записи. Каждая запись представляет собой последовательность пар атрибут-значение. Например, запись с тремя парами атрибут-значение может быть:

[{ID, 1} {FirstName, Thomas} {Company, Oracle}]

Эта модель данных означает, что каждая запись может быть разной: им не нужно иметь одинаковые атрибуты или одинаковое количество пар атрибут-значение, и они могут даже иметь несколько значений для одного и того же атрибута. Так что в той же коллекции записей, также могут быть записи:

[{ID, 2} {Company, SAP} {Title, Sales Consultant} {Age, 45} {Comment, “Ich bin ein…”}]

[{ID, 3} {Hobby, Bowling} {Hobby, Tennis} {Company, Oracle}]

Понятно, что записи Endeca Server предлагают несколько технических преимуществ по сравнению со строками в реляционной базе данных. Например, Endeca Server естественным образом сжимает разреженные данные: если запись не имеет значения для атрибута, она просто никогда не ассоциируется с этим атрибутом. Если, наоборот, запись имеет несколько значений для атрибута, Endeca Server просто хранит их все, не дублируя остальную часть записи [5].

Встроенная поддержка неровных и уникальных записей означает, что Endeca Server может принимать данные без предварительного моделирования. Это снижает количество барьеров на пути к использованию, как для разработчиков, так и особенно для бизнес-пользователей: можно взять интересующие данные, поместите их на сервер Endeca, где они будут организованы для интегрированного поиска, анализа и навигации, и начать использовать их в считанные минуты. Если позже пользователь захочет получить данные из другого источника, это не составит никаких проблем — просто надо загрузить их, оставив старые записи такими, какие они есть.

Для каждого атрибута в данных Endeca Server хранит два индекса, в которых хранится каждая пара запись-значение. Прямой индекс сортируется по идентификатору записи; это позволяет быстро искать значения, связанные с определенными записями — полезно, когда пользователи детализировали

поиск и хотят видеть подробную информацию об определенных записях, например, в таблице результатов. Обратный индекс сортируется по значению атрибута; это оптимизирует поиск для случаев, когда пользователь хочет проанализировать распределение значений в данных, таких как агрегаты, фильтры диапазона и навигация. Каждая запись, вместо того, чтобы сохранять свои значения атрибута, указывает на соответствующую позицию в соответствующем индексе атрибута. В совокупности набор индексов, связанных с атрибутом, называется атрибутом модели [5].

Модели атрибутов отображаются в виртуальной памяти. Чтобы воспользоваться различными порядками сортировки, каждый индекс атрибута представлен структурой данных, подобный B-дереву, которая значительно ускоряет поиск записей и значений. Часто используемые сегменты столбцов кэшируются в физической памяти для ускорения обработки запросов. В этом отношении стратегия хранения Endeca Server предназначена для использования общей схемы использования обнаружения данных: пользователи часто имеют некоторое представление о том, что они ищут, и поэтому применяют ранние фильтры, такие как поиск по ключевым словам или пространственный / временной выбор, который значительно ограничивает приемлемый набор результатов, а затем изменяется шагами вперед и назад в этом подмножестве данных. Поддержание всех моделей атрибутов в виртуальной памяти позволяет Endeca Server обеспечить широту, необходимую для этих начальных фильтров, в то время как его стратегия кэширования позволяет интерактивные скорости во время фазы разведки назад и вперед. Решения строго в памяти обязательно ограничивают объем данных, доступных для этой начальной отправной точки. Кроме того, эта стратегия позволяет масштабируемое, итеративное расширение как анализа, так и данных [5].

ВЫВОДЫ

В главе 2 рассмотрен поисковой механизм Oracle Endeca Commerce, его

сущности и их соотношение. Также описан Endeca Server, способ, которым он хранит и оперирует сущностями Endeca.

ГЛАВА 3. РАЗРАБОТКА МЕТОДОВ ПОИСКА ЗАКАЗОВ

3.1. Алгоритмы поиска

Так как ATG Repository является уровнем абстракции, по сути являясь слоем над источником данных, однозначно выделить алгоритм поиска нельзя, ведь он зависит от выбранного источника данных — база данных (Oracle DB, MSSQL DB и тд), LDAP, файловая система. Например, если как источник данных выбрана база данных Oracle, реализация поиска в ней скрыта и прямого доступа к ней нет. Алгоритм поиска в MDEX Engine также скрыт. На основании неоднозначности алгоритма поиска через репозиторий и отсутствия доступа к алгоритму Endeca далее будут рассматриваться методы поиска, основанные на непосредственном использовании этих механизмов.

3.2. Методы поиска заказов

Стандартный метод поиска заказов в платформе Endeca использует механизм Repository для обращения к базе данных. Так как поисковой механизм Endeca ориентирован непосредственно на поиск, за счет него, а именно двух индексов в каждом атрибуте записи, можно уменьшить время поиска по заказам, если поместить их в него. Однако из-за необходимой индексации данных перед отправкой их в Endeca, возникает проблема с часто обновляемыми данными. На больших количествах данных процесс индексации может занимать недопустимое количество времени, что повлечет за собой нерелевантность данных в Endeca.

Если совместить два этих метода, можно организовать поиск по заказам по следующим правилам:

- Поиск осуществляется через Endeca.
- Одновременно с этим ведется поиск в базе данных, для тех заказов, у которых дата последнего изменения позже, чем дата последней индексации.
- Результаты обоих поисков «сливаются» в один. При этом

приоритет в выборке у двух заказов с одинаковым идентификатором имеет ордер из базы данных, так как он содержит релевантные данные.

Данный гибридный метод позволяет уменьшить время работы за счет поиска основной части заказов из Endeca, в то же время поддерживать релевантность данных (новые и обновленные заказы) за счет выборки из базы данных.

3.3. Интеграция с Endeca в платформе ATG

В платформе ATG изначально интеграция с Endeca обеспечивает поиск по продуктам. Основной Nucleous компонент для работы с индексацией продуктов — ProductCatalogSimpleIndexingAdmin. Из 6 основных компонент:

1. CategoryTreeService — осуществляет валидацию дерева каталога. Проверяет связи между категориями и продуктами. Очищает неправильные и пустые указатели. Подготавливает дерево к индексации.

2. SchemaExporter — определяет схему и мета свойства продуктов для дальнейшей работы с ними.

3. CategoryToDimensionOutputConfig — используется для генерации категорий в измерения для продуктов.

4. RepositoryTypeDimensionExporter — осуществляет дополнительные действия по генерации, связи и подготовки к индексации продуктов и категорий из репозиториев.

5. ProductCatalogOutputConfig — основной компонент, в котором указывается соответствие источника данных, то есть репозитория с продуктами, и записей Endeca.

6. EndecaScriptService — непосредственно компонент для импорта и запуска процесса индексации в Endeca.

На рисунке 8 представлены фазы индексации каталога. Доступно два действия:

1. Baseline Index — перестройка всего дерева каталога в индексированную структуру.
2. Partial Index — обновление только записей без перестройки дерева.

Соответственно Baseline более затратный в ресурсах. По сути включает в себя Partial Index.

Phase	Component	Records Sent	Records Failed	Status
PreIndexing				
	/atg/commerce/endeca/index/CategoryTreeService			PENDING
RepositoryExport				
	/atg/commerce/endeca/index/SchemaExporter	0	0	PENDING
	/atg/commerce/endeca/index/CategoryToDimensionOutputConfig	0	0	PENDING
	/atg/commerce/endeca/index/RepositoryTypeDimensionExporter	0	0	PENDING
	/atg/commerce/search/ProductCatalogOutputConfig	0	0	PENDING
EndecaIndexing				
	/atg/commerce/endeca/index/EndecaScriptService			PENDING
Actions:	Baseline Index Partial Index			

Рисунок 8. Фазы ProductCatalogSimpleIndexingAdmin

3.4. Создание компонент индексации

Так как структура каталога представляет из себя дерево, а заказы логически являются лишь записями, для обеспечения индексации заказов достаточно всего лишь 2:

1. OrderSchemeExporter — для генерации мета информации.
2. OrderOutputConfig — непосредственно для генерации записей и обеспечения соответствия.

Компонент OrderOutputConfig оперирует репозиторием и файлом описания соответствия между сущностью в репозитории и записью Endeca в формате XML. Файл состоит из описания записи — документа. Документ включает свойства, каждое из которых обладает своими атрибутами. В данном файле также задаются свойства которые будут являться значениями

измерения.

В файле задается соответствие исходных данных и свойств записи Endeca. Для разработки методов поиска по заказам и обеспечения широко используемых бизнес пользователем фильтров при использовании поиска были выбраны следующие поля:

- Id — идентификатор заказа. Необходим для поиска конкретного заказа при отображении его деталей.
- State — статус заказа. Наиболее частая операция при поиске заказов — отфильтровать заказы по статусу, например, заказы ожидающие подтверждения или заказы, доставленные покупателю. Данное свойство также обозначено как значение измерения, так как количество статусов ограничено, позволяет организовать заказы в категории.
- Email — электронная почта (логин или другой идентификатор покупателя / пользователя). Также обозначен как значение измерения для организации заказов по пользователям.
- ProfileId — идентификатор пользователя. Функции аналогичны Email.
- Description — описание заказа. Информация зачастую достаточная для пользователя, использующего поиск, без надобности переходить к деталям заказа. Используется для отображения в таблице результатов.
- submittedDate и lastModifiedDate — для организации поиска по временным промежуткам.

Так как в платформе ATG поддерживается поиск по промежуткам цен на основе каталога и дерева категорий при поиске по заказам ввиду отсутствия логической структуры дерева в исходных данных, а также для упрощения реализации данного поиска и снижения временных затрат при разработке продукта, свойства submittedDate и lastModifiedDate обозначены как обычные свойства, а не значения измерения.

Содержание конфигурационного файла в формате XML, задающего соответствие представлен в Приложении.

3.5. Создание REST API

Для доступа к поиску создаем REST API с помощью возможностей платформы ATG. Для создания конечных точек REST API достаточно создать компонент с классом atg.service.actor.ActorChainService и указать в конфигурационном файле сами точки и компоненты, обрабатывающие запросы.

Компонент, обрабатывающий запросы, должен являться наследником atg.servlet.DynamoServlet, который в свою очередь является наследником HttpServlet. Так как формат запроса для методов один и тот же достаточно создать один Java класс — OrderDroplet, задающий логику обработки запросов, сделать у него полем объект типа интерфейса OrderService. Этот сервис представляет один метод, возвращающий результаты поиска, в зависимости от входных параметров. Далее создаётся три компонента с этим классом, и каждому задается своя реализация сервиса, обеспечивающего конкретный метод поиска. Здесь наглядно можно увидеть гибкость предоставляемых механизмом Nucleus возможностей.

Таким образом для трех методов будет создано три компоненты и три сервиса. При этом сервис гибридного поиска будет использовать два других сервиса, для уменьшения количества кода, избегания дублирования кода и функционала.

Входными параметрами для сервисов являются:

- state — статус заказа для фильтрации;
- email — электронная почта (логин, идентификатор пользователя) пользователя для фильтрации;
- date — дата последнего изменения заказа для ограничени по временным рамкам;
- sort — поле по которому будет осуществляться сортировка;
- ascending — направление сортировки (по возрастанию / по убыванию);
- limit — ограничение количества заказов в результате поиска;

- offset — сдвиг в результате поиска относительно начала.

Параметры limit и offset необходимы для организации постраничного просмотра результатов на клиенте.

Код класса OrderDroplet и его зависимостей представлен в Приложении.

3.6. Метод поиска через Repository

Сервис поиска заказов через API Reposiory реализован в классе OrderRepositoryService. Данный сервис использует запросы на языке запросов к Repository RQL — Repository Query Language. Язык запросов RQL синтаксически схож с языком SQL.

Полный запрос к Repository будет иметь вид:

ALL AND state = <state> AND email = <email> AND (submittedDate > datetime(<date>) OR lastModifiedDate > datetime(<date>)) ORDER BY <sort_field> SORT <asc|desc> RANGE <offset>+<limit>

Соответственно если параметр не представлен в запросе он также будет отсутствовать. Параметры сортировки и постраничного поиска при отсутствии будут заменены значениями по умолчанию.

Листинг кода сервиса OrderRepositoryService представлен в Приложении.

3.7. Метод поиска через Endeca

Запросы к Endeca MDEX Engine осуществляются как HTTP GET к его развернутому приложению. В параметрах URL указываются параметры поиска.

Параметры необходимые для осуществления поиска по заказам:

- **N** — Уникальная комбинация идентификаторов значений измерения, которая определяет каждый объект навигации. Корневой объект навигации отображается, когда ноль является единственным значением в параметре [4].
- **Nrs** — Устанавливает выражение языка запросов Endeca для

запроса навигации. Это выражение не будет сконвертировано и будет направлено непосредственно к Endeca Server. Выражение будет действовать как фильтр для ограничения результатов запроса [4].

- **No —** смещение. Определяет начальный индекс для списка записей объекта навигации. Этот параметр позволяет пользователям просматривать длинный набор результатов напрямую или пошагово. Если смещение превышает количество элементов в списке записей объекта навигации, то возвращенный список записей будет пустым [4].
- **Ns** — Задает список свойств или измерений (ключей сортировки), по которым сортируются записи [4].
- **Nso —** Определяет порядок сортировки списка записей объекта навигации. Значение 0 указывает сортировку по возрастанию, которая используется по умолчанию, если параметр Nso отсутствует. Значение 1 указывает сортировку по убыванию [4].

Идентификатор значения измерения является числом и называется Navigation state. Данные идентификаторы генерируются при индексации и не меняются до следующей индексации, при этом однозначно определяют значение измерения. Для упрощения построения запроса к Endeca при первом запросе запрашиваются все значения измерения и их Navigation state. Эти значения сохраняются в компонент DPCacheTools в виде ассоциативного контейнера. Формат записи — ключ < имя значения измерения>+<значение измерения>, значение <navigation state>. Например при запросе к данному сервису с параметром state=INCOMPLETE из этого компонента можно получить navigation state по ключу order.state+ INCOMPLETE.

Таким образом запрос к Endeca будет выглядеть следующим образом:

https://<host>:<port>?N=<navigation state для state заказа>+<navigation state для

email>&Nrs=collection()/record[order.submittedDate > <date> or order.lastModifiedDate > <date>]&No=<offset>&Ns=<sortfield>&Nso=<1|0>

Данный запрос осуществляется с помощью классов ENEConnection, UrlENEQuery пакета com.endeca.navigation. Для установления лимита на количества записей используется метод setNavNumERecs класса UrlENEQuery.

Листинги кода класса сервиса и классов, используемых им представлены в Приложении.

3.8. Гибридный метод поиска

Сервис гибридного поиска реализован в классе OrderHybridService. Сервис гибридного поиска использует сервисы поиска через репозиторий и Endeca. При этом для сервиса репозитория добавляется параметр даты, которая является датой последнего процесса индексации. Таким образом из репозитория запрашиваются заказы, которые были обновлены или добавлены после индексации.

Для сохранения даты последней индексации расширяется компонент EndecaScriptService. Создается класс наследник, в котором переопределяется метод запуска скрипта индексации, после выполнения которого текущая дата сохраняется в сервис OrderHybridService. Сам сервис в свою очередь добавляется полем в компонент EndecaScriptService.

После получения результатов поиска из обоих сервисов они сливаются. Предварительно в сервис поиска по репозиторию передается множество, которое заполняется идентификаторами заказов. Проходя одновременно по списку заказов из репозитория и по списку заказов из Endeca, проверяется, если идентификатор заказа из Endeca содержится в множестве, заказ пропускается, так как есть более релевантная версия. Если не содержится проверяется надо ли вставить в текущую позицию списка заказов из репозитория на основании сортировки. Если вставка нужна, заказ и позиция сохраняется, позиция в списке заказов из Endeca сдвигается к следующему.

Если вставка не нужна, позиция в списке заказов из репозитория сдвигается к следующему.

После слияния двух списков результирующий список ограничивается в соответствии с параметром limit.

Листинг кода класса OrderHybridService представлен в Приложении.

3.9. Результаты работы сервисов поиска

Сравним сервис поиска по репозиторию и поиска через Endeca. В системе 19,799 заказов. Новых и обновленных заказов нет, то есть данные идентичны в репозитории (базе данных) и в Endeca. Для каждого набора параметров делается 10 запросов и берется среднее время выполнения запроса.

При этом на уровне репозитория установлен кэш в размере 1000 записей.

	Метод поиска из репозитория	Метод поиска из Endeca
Limit=20000	9102мс 19,799 записей	814мс 19,799 записей
Limit=250	21мс 250 записей	25мс 250 записей
Limit=700 State=INCOMPLETE	26мс 700 записей	30мс 700 записей
Limit=1100 State=INCOMPLETE	554мс 1100 записей	43мс 1100 записей
Limit=1500 State=INCOMPLETE	823мс 1500 записей	53мс 1500 записей

Таблица 1. Сравнение поиска по репозиторию и поиска через Endeca

Для тестирования всех методов эмулируется ситуация, когда в системе 20,099 заказов, из которых в Endeca находится 19,799, а новых заказов 300. Параметр days в данном случае обозначает — были изменены в последние n дней.

	Метод поиска из репозитория	Метод поиска из Endeca	Гибридный метод поиска
Limit=2000 email=<user>	1100мс 1918 записей	65мс 1618	72мс 1918 записей

		записей	
Limit=2000 email=<user> days=3	28мс 706 записей	22мс 406 записей	34мс 706 записей
Limit=4000 State=PENDING_APPROVAL days=4	1809мс 3778 записей	175мс 3718 записей	224мс 3778 записей

Таблица 2. Сравнение методов поиска

3.10. Применение методов в различных моделях

На основании результатов, полученных в пункте 3.6. можно выделить основные модели, в которых применимы разработанные методы.

Поиск по репозиторию оптимален при запросах, ограниченных числом, меньшим размера кэша. При этом увеличение кэша до количества всех заказов не рационален ввиду сложности поддержания актуального статуса кэша, постоянного увеличения количества заказов и затрат ресурсов. Данный метод можно использовать при создании постраничного поиска, где каждая страница получается отдельным запросом.

Однако, в последнее время актуален подход, при котором в клиент-серверных приложениях нагрузка постепенно переносится на сторону клиента. В такой модели постраничная организация результатов запроса создается непосредственно на стороне клиента, например, в браузере. Следовательно, клиентское приложение запрашивает сразу все результаты поиска одним запросом. В данной модели поиск по репозиторию будет показывать себе хуже, нежели гибридный поиск или поиск через Endeca.

Стоит отметить, что проблема с размером кэша не так остро проявляется в гибридном поиске. Так как при гибридном поиске из репозитория запрашиваются только обновленные, относительно даты последней индексации, заказы, то их количество не будет превышать некого предела. Например, если индексация будет проходить раз в день, то среднее количество заказов, созданных и измененных за этот промежуток времени можно вывести, используя статистику за некоторый период. Ограничив кэш

выведенным пределом, можно добиться минимальных временных затрат при запросе созданных и обновленных заказов.

Также стоит принимать во внимание что модели зависят от контекста их применения. Если ставится задача организовать поиск по заказам для пользователя системы B2C (бизнес-для-потребителя), где в роли покупателя выступает конечный потребитель и количество его заказов невелико, стоит использовать поиск из репозитория. Но система может быть B2B (бизнес-для- бизнеса) типа, где покупателем выступает организация с несколькими или большим количеством участников и количество заказов может достигать большого числа. При этом большая часть этих заказов будет уже закрыта, и не будет обновляться, но к ним все еще нужен доступ и поиск, например, если пользователь хочет найти ранее купленный товар и т. д. В этом случае можно использовать гибридный поиск.

Также возможная задача — организация поиска по заказам для сотрудников технической поддержки, службы доставки, службы возврата. Эти службы работают с заказами, которые уже обработаны, закрыты и также не будут обновляться, или будут, но не чаще некоторого периода, относительно которого можно задать периодичность индексации. В такой модели можно использовать поиск через Endeca.

ВЫВОДЫ

В главе 3 описан процесс реализации методов поиска по репозиторию, поиска через Endeca, гибридного поиска. Результаты работы разработанных методов проанализированы, на основании полученных данных сделаны выводы об актуальности применения методов в различных моделях.

ЗАКЛЮЧЕНИЕ

В работе рассмотрены платформа ATG, поисковой механизм Endeca, их интеграция и расширение возможностей поиска заказов на основе из интеграции. Были рассмотрены составляющие части и компоненты, внутренние процессы как платформы ATG, так и поискового механизма Endeca. Также были рассмотрены сущности, которыми оперируют эти процессы, взаимосвязь сущностей и их взаимосвязь с исходными данными. Также была изучена внутренняя структура и организация данных для обеспечения поиска, которая используется в системе, а также перемещение и реорганизация этих данных в данной системе.

В главе 1 рассмотрена платформа ATG, ее возможности и составляющие части, основные механизмы и компоненты. Описана структура ATG. Также исследованы структура и взаимодействие внутренних компонентов поискового механизма Endeca Commerce.

В главе 2 рассмотрены сущности и их соотношение в Endeca Commerce. Также изучены возможности Endeca Server, способ, которым он хранит и оперирует сущностями Endeca.

В главе 3 описан способ использования ATG и Endeca для организации поиска по сущностям, отличным от стандартных, в качестве примера использована сущность Заказ. Разработаны и исследованы метода поиска заказов, такие как поиск по репозиторию, поиск через поисковой механизм Endeca, гибридный поиск, использующий как поиск по репозиторию, так и поиск через поисковой механизм Endeca.

Разработанные методы и их анализ позволил выделить модели и сценарии, в которых каждый из методов позволяет добиться более высокой скорости работы, по сравнению с другими методами.

Поиск по репозиторию позволяет эффективно использовать его в

качестве организации модели постраничного поиска по часто обновляемым данным, например, при создании поиска по заказам по истории заказов пользователя, являющимся конечным покупателем. Гибридный поиск показывает себя наиболее эффективным в ситуациях, когда необходимо организовывать поиск по большому количеству заказов, где доля обновляемых заказов существенно меньше относительно всего количества заказов, в приложениях с нагрузкой на клиентскую часть. Поиск через поисковой механизм Endeca также эффективен на большом числе заказов в приложениях с нагрузкой на клиентскую часть, где промежуток обновления заказов сравнительно невелик для организации процесса индексации.

Разработанные методы позволяют внедрять их в приложения с небольшими затратами ресурсов, в том числе людских, при этом достигая гибкости и эффективности в разных моделях использования данных методов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация платформы ATG [Электронный ресурс]. — Режим доступа: https://www.oracle.com/technetwork/documentation/atgwebcommerce-393465.html. — Дата доступа: 21.02.2019.
2. Документация Endeca [Электронный ресурс]. — Режим доступа: https://docs.oracle.com/cd/E94707_01/index.html. — Дата доступа: 15.03.2019.
3. Документация Endeca Server [Электронный ресурс]. — Режим доступа: https://docs.oracle.com/cd/E40521_01/index.htm. — Дата доступа: 24.03.2019.
4. Endeca URL Parameter Reference [Электронный ресурс]. — Режим доступа: https://docs.oracle.com/cd/E29584_01/webhelp/mdex_basicDev/src/cbdv_urlparams_root.html. — Дата доступа: 04.05.2019.
5. Oracle Endeca Information Discovery: A Technical Overview [Электронный ресурс]. — Режим доступа: http://www.oracle.com/us/solutions/ent-performance-bi/oeid-tech-overview-1674380.pdf. — Дата доступа: 30.04.2019.

ПРИЛОЖЕНИЕ

1.1. Листинг содержания конфигурационного файла соответствия

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE   item   SYSTEM   "http://www.atg.com/dtds/search/indexing-
dependency-schema.dtd">
<item item-descriptor-name="order" is-document="true">
  <title property-name="displayName"/>
  <properties>
   <property name="id" is-dimension="false" type="string"/>
   <property name="description" is-dimension="false" type="string"/>
   <property name="state" is-dimension="true" type="string"/>
   <property name="profileId" is-dimension="true" type="string"/>
   <property name="email" is-dimension="true" type="string"/>
   <property name="submittedDate" is-dimension="false" type="date"/>
    <property name="lastModifiedDate" is-dimension="false" type="date"/
>
  </properties>
</item>
```

1.2. Листинг кода класса OrderDroplet

```
public class OrderDroplet extends DynamoServlet {
  private static final String OUTPUT_OPARAM = "output";
  private static final String ORDERS_FIELD = "orders";
  private static final String TOTAL_FIELD = "total";
  private OrderService service;
  private int defaultLimit;
  private int defaultOffset;
  private String defaultSort;
  private boolean defaultAscending;
  @Override
        public    void    service(DynamoHttpServletRequest    pRequest,
DynamoHttpServletResponse    pResponse)    throws    ServletException,
IOException {
    OrderSearchParams params = new OrderSearchParams();
```

```
    params.setOffset(defaultOffset);
    params.setLimit(defaultLimit);
    params.setSort(OrderSearchParams.SortFields.valueOf(defaultSort));
    params.setAscending(defaultAscending);
    params.initializeFields(pRequest);
    List<Map<String, Object>> list = service.getOrders(params);
    pRequest.setParameter(ORDERS_FIELD, list);
    pRequest.setParameter(TOTAL_FIELD, list.size());
    pRequest.serviceParameter(OUTPUT_OPARAM, pRequest, pResponse);
  }
  public OrderService getService() {
    return service;
  }
  public void setService(OrderService service) {
    this.service = service;
  }
  public int getDefaultLimit() {
    return defaultLimit;
  }
  public void setDefaultLimit(int defaultLimit) {
    this.defaultLimit = defaultLimit;
  }
  public int getDefaultOffset() {
    return defaultOffset;
  }
  public void setDefaultOffset(int defaultOffset) {
    this.defaultOffset = defaultOffset;
  }
  public String getDefaultSort() {
    return defaultSort;
  }
  public void setDefaultSort(String defaultSort) {
    this.defaultSort = defaultSort;
  }
  public boolean isDefaultAscending() {
    return defaultAscending;
  }
  public void setDefaultAscending(boolean defaultAscending) {
```

```
    this.defaultAscending = defaultAscending;
  }
}
```

1.3. Листинг кода класса OrderSearchParams

```
public class OrderSearchParams {
    private int limit;
    private int offset;
    private String state;
    private String email;
    private Date date = null;

    private SortFields sort;
    private boolean ascending;

    public void initializeFields(final DynamoHttpServletRequest
pRequest) {
        state = pRequest.getParameter("state");
        email = pRequest.getParameter("email");
        if (StringUtils.isBlank(state)) {
            state = null;
        }
        if (StringUtils.isBlank(email)) {
            email = null;
        }
        int days = getDays(pRequest);
        if (days > 0) {
            Calendar calendar = Calendar.getInstance();
            calendar.add(Calendar.DATE, -1 * days);
            date = calendar.getTime();
        }
        sort = getSort(pRequest);
        ascending = getAscending(pRequest);
        limit = getLimit(pRequest);
        offset = getOffset(pRequest);
    }
```

```
    private int getLimit(final DynamoHttpServletRequest pRequest) {
        return getNumber(pRequest, "limit", limit);
    }

    private int getOffset(final DynamoHttpServletRequest pRequest) {
        return getNumber(pRequest, "offset", offset);
    }

    private int getDays(final DynamoHttpServletRequest pRequest) {
        return getNumber(pRequest, "days", -1);
    }

    private int getNumber(final DynamoHttpServletRequest pRequest,
String paramName, int defaultValue) {
        int result;
        final String field = pRequest.getParameter(paramName);
        if (null == field) {
            result = defaultValue;
        } else {
            try {
                result = Integer.parseInt(field);
            } catch (NumberFormatException e) {
                result = defaultValue;
            }
        }
        return result;
    }

    private SortFields getSort(final DynamoHttpServletRequest
pRequest) {
        String sSort = pRequest.getParameter("sort");
        SortFields sort = null;
        try {
            sort = SortFields.valueOf(sSort.toUpperCase());
        } catch (Exception e) {
            sort = SortFields.DATE;
        }
        return sort;
```

```java
    }

    private boolean getAscending(final DynamoHttpServletRequest
pRequest) {
        String sAscending = pRequest.getParameter("asc");
        return "1".equals(sAscending);
    }

    public int getLimit() {
        return limit;
    }

    public void setLimit(int limit) {
        this.limit = limit;
    }

    public int getOffset() {
        return offset;
    }

    public void setOffset(int offset) {
        this.offset = offset;
    }

    public String getState() {
        return state;
    }

    public void setState(String state) {
        this.state = state;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
```

```
    }

    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }

    public SortFields getSort() {
        return sort;
    }

    public void setSort(SortFields sort) {
        this.sort = sort;
    }

    public boolean isAscending() {
        return ascending;
    }

    public void setAscending(boolean ascending) {
        this.ascending = ascending;
    }

    public enum SortFields {
        STATE, EMAIl, ID, DATE
    }

}
```

1.4. Листинг кода класса OrderRepositoryService

```
public class OrderRepositoryService extends GenericService implements
OrderService {
    private Repository mRepository;

    public Repository getRepository() {
```

```
        return mRepository;
    }

    public void setRepository(Repository pRepository) {
        mRepository = pRepository;
    }

    private String buildQuery(OrderSearchParams params) {
        StringBuilder builder = new StringBuilder();
        builder.append("ALL");
        String state = params.getState();
        if (state != null) {
                                        builder.append("   AND   state
= \"").append(state).append("\"");
        }
        String email = params.getEmail();
        if (email != null) {
                                        builder.append("   AND   email
= \"").append(email).append("\"");
        }
        Date date = params.getDate();
        if (date != null) {
             SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss zzz");
            String sDate = format.format(date);
            builder.append(" AND (");
                                    builder.append("submittedDate   >
datetime(\"").append(sDate).append("\")");
            builder.append(" OR ");
                                  builder.append("lastModifiedDate   >
datetime(\"").append(sDate).append("\")");
            builder.append(")");
        }
        String sort = params.getSort().toString().toLowerCase();
        if ("date".equals(sort)) {
            sort = "lastModifiedDate";
        }
        builder.append(" ORDER BY ").append(sort);
```

```
        builder.append(" SORT ").append(params.isAscending() ? "ASC" :
"DESC");
        builder.append(" RANGE ?0+?1");
        return builder.toString();
    }

    public List<Map<String, Object>> getOrders(OrderSearchParams
params) {
        return getOrders(params,null);
    }

    public List<Map<String, Object>> getOrders(OrderSearchParams
params,Set<String> ids) {
        List<Map<String, Object>> list = new LinkedList<>();
        Repository repository = getRepository();
        try {
            RepositoryItemDescriptor descriptor =
repository.getItemDescriptor("order");
            RepositoryView view = descriptor.getRepositoryView();
            String query = buildQuery(params);
            vlogInfo("Repository query: " + query);
            RqlStatement statement =
RqlStatement.parseRqlStatement(query);
            Object[] repositoryParams = new Object[2];
            repositoryParams[0] = params.getOffset();
            repositoryParams[1] = params.getLimit();
            RepositoryItem[] items = statement.executeQuery(view,
repositoryParams);
            if (items != null) {
                for (RepositoryItem item : items) {
                    if(ids != null){
                        ids.add(item.getRepositoryId());
                    }
                    list.add(generateMap(item));
                }
            }
        } catch (RepositoryException e) {
            e.printStackTrace();
```

```
        }
        return list;
    }

    private Date getDate(Object time){
        Date date = null;
        if (time != null) {
            long lTime = ((Timestamp) time).getTime();
            date = new Date(lTime);
        }
        return date;
    }

    private Map<String, Object> generateMap(RepositoryItem item) {
        Map<String, Object> map = new HashMap<>();
        map.put("id", item.getRepositoryId());
        map.put("state", item.getPropertyValue("state"));
        map.put("profileId", item.getPropertyValue("profileId"));
        map.put("description", item.getPropertyValue("description"));
        map.put("email", item.getPropertyValue("email"));
                                                map.put("submittedDate",
getDate(item.getPropertyValue("submittedDate")));
                                              map.put("lastModifiedDate",
getDate(item.getPropertyValue("lastModifiedDate")));
        return map;
    }
}
```

1.5. Листинг кода класса OrderEndecaService

```
public class OrderEndecaService extends GenericService implements
OrderService {

    private DPCacheTools cacheTools;
    private Map<OrderSearchParams.SortFields, String> sortMap;
    private DPDimensionValueCacheRefreshHandler handler;

    public OrderEndecaService() {
        sortMap = new HashMap<>();
```

```
        sortMap.put(OrderSearchParams.SortFields.ID, "order.id");
                        sortMap.put(OrderSearchParams.SortFields.EMAIl,
"order.email");
                        sortMap.put(OrderSearchParams.SortFields.STATE,
"order.state");
                         sortMap.put(OrderSearchParams.SortFields.DATE,
"order.lastModifiedDate");
    }


    public static void main(String[] args) {

        System.out.println(new ArrayList<>().subList(0, 2).size());
    }

    private String buildQuery(OrderSearchParams params) {
        StringBuilder builder = new StringBuilder();
        boolean hasEmptyN = false;
        String sourceN = cacheTools.get("record.source+Orders");
        builder.append("N=").append(sourceN);
        String state = params.getState();
        if (state != null) {
            String stateN = cacheTools.get("order.state+" + state);
            if (StringUtils.isNotBlank(stateN)) {
                builder.append("+").append(stateN);
            } else {
                hasEmptyN = true;
            }
        }
        String email = params.getEmail();
        if (email != null) {
            String stateN = cacheTools.get("order.email+" + email);
            if (StringUtils.isNotBlank(stateN)) {
                builder.append("+").append(stateN);
            } else {
                hasEmptyN = true;
            }
        }
```

```
        Date date = params.getDate();
        if (date != null) {
            long lDate = date.getTime();
            builder.append("&Nrs=collection()/record[");
            builder.append("order.submittedDate > ").append(lDate);
            builder.append(" or ");
            builder.append("order.lastModifiedDate > ").append(lDate);
            builder.append("]");
        }
        builder.append("&No=").append(params.getOffset());
        builder.append("&Ns=").append(sortMap.get(params.getSort()));
        int ascending = params.isAscending() ? 0 : 1;
        builder.append("&Nso=").append(ascending);
        String result = null;
        if (!hasEmptyN) {
            result = builder.toString();
        }
        return result;
    }


        public  List<Map<String,  Object>>  getOrders(OrderSearchParams
params) {
        if (cacheTools.getMapSize() == 0) {
            try {
                handler.process();
            } catch (CartridgeHandlerException e) {
                 vlogError("Error fetching dimensions in handler {0}",
e);
            }
        }

        List<Map<String, Object>> list = new LinkedList<>();
             ENEConnection  nec  =  new  HttpENEConnection("localhost",
"15000");
        try {
            String query = buildQuery(params);
            if (query != null) {
```

```
                vlogInfo("Endeca query: " + query);
                ENEQuery nequery = new UrlENEQuery(query, "UTF-8");
                nequery.setNavNumERecs(params.getLimit());
                ENEQueryResults qr = nec.query(nequery);
                Navigation nav = qr.getNavigation();
                ERecList records = nav.getERecs();
                for (Object o : records) {
                    ERec record = (ERec) o;
                    list.add(generateMap(record));
                }
            }
        } catch (ENEQueryException e) {
            e.printStackTrace();
        }
        return list;
    }

    public Date getDate(Object time) {
        Date date = null;
        if (time != null) {
            date = new Date(Long.valueOf((String) time));
        }
        return date;
    }

    private Map<String, Object> generateMap(ERec record) {
        Map<String, Object> map = new HashMap<>();
        PropertyMap recordProperties = record.getProperties();
        List list = record.getDimValues();
        map.put("id", recordProperties.get("order.id"));
                                                        map.put("description",
recordProperties.get("order.description"));
                                                   map.put("lastModifiedDate",
getDate(recordProperties.get("order.lastModifiedDate")));
                                                      map.put("submittedDate",
getDate(recordProperties.get("order.submittedDate")));
        AssocDimLocationsList dimListValues = record.getDimValues();
        for (Object dimValue : dimListValues) {
```

```
            AssocDimLocations assocDimLocation = (AssocDimLocations) dimValue;
            if (assocDimLocation.size() > 0) {
                DimLocation location = (DimLocation) assocDimLocation.get(0);
                DimVal val = location.getDimValue();
                String dimName = val.getDimensionName();
                if (dimName != null && !dimName.equals("record.source")) {
                    map.put(dimName.replaceAll("order\\.", ""), val.getName());
                }

            }
        }
        return map;
    }

    public DPCacheTools getCacheTools() {
        return cacheTools;
    }

    public void setCacheTools(DPCacheTools cacheTools) {
        this.cacheTools = cacheTools;
    }



    public DPDimensionValueCacheRefreshHandler getHandler() {
        return handler;
    }

    public void setHandler(DPDimensionValueCacheRefreshHandler handler) {
        this.handler = handler;
    }
}
```

1.6. Листинг кода класса DPDimensionValueCacheRefreshHandler

```
public class DPDimensionValueCacheRefreshHandler extends
DimensionValueCacheRefreshHandler {

    private MdexRequest mMdexRequest;

    private RecordFilterBuilder[] mRecordFilterBuilders;
    private Set<String> dimensionNames;
    private DPCacheTools cacheTools;
    private DimensionSearchResultsConfig config;




    public void preprocess() throws CartridgeHandlerException {
                final NavigationState navigationState =
getFilteredNavigationState();
                SearchFilter searchFilter =
getDimensionSearchFilter(navigationState);
        if (null != searchFilter && null != config &&
config.isEnabled()) {
                final NavigationState dimNavState =
createDimensionSearchNavigationState(navigationState);
                setMdexRequestBroker((MdexRequestBroker)
NucleusResolverUtil.resolveName("/atg/endeca/assembler/cartridge/manag
er/MdexRequestBroker"));
                this.mMdexRequest =
createMdexRequest(dimNavState.getFilterState(), buildMdexQuery(config,
searchFilter));
        }
    }

        public List<DimensionSearchResultGroup> query(final
DimensionSearchResultsConfig pCartridgeConfig) throws
CartridgeHandlerException {
        List<DimensionSearchResultGroup> results = null;
        if (null != mMdexRequest) {
                final ENEQueryResults eneResults =
```

```
executeMdexRequest(mMdexRequest);

                           final  DimensionSearchResult  dimResults  =
eneResults.getDimensionSearch();
            if (null != dimResults) {
                results = dimResults.getResults();
            }
        }
        return results;
    }



    public void process() throws CartridgeHandlerException {
        preprocess();
        vlogDebug("DP Dimension process start");
                    List<DimensionSearchResultGroup>  allDimensions  =
query(config);
                                Map<String,    String>    newCache    =
getCacheTools().createEmptyCache();
        for (DimensionSearchResultGroup dimension : allDimensions) {
                                  DimVal   dimensionRoot   =   (DimVal)
dimension.getRoots().get(0);
            String dimensionName = dimensionRoot.getDimensionName();
            if (dimensionNames.contains(dimensionName)) {
                               Iterator<DimLocationList>  dimensions  =
dimension.iterator();
                while (dimensions.hasNext()) {
                                    DimLocationList  dimLocationList  =
dimensions.next();
                            cacheValue(dimLocationList, dimensionName,
newCache);
                }
            }
        }
    }

      private  void  cacheValue(final  DimLocationList  pDimLocationList,
final String pDimensionName, final Map<String, String> pNewCache) {
```

```
                    final   DimLocation   dimLocation   =   (DimLocation)
pDimLocationList.get(0);
          if (null == dimLocation || null == dimLocation.getDimValue())
{
        } else {
            final String key = getKey(dimLocation, pDimensionName);
            if (StringUtils.isNotBlank(key)) {
                                              final   String   dimvalId   =
String.valueOf(dimLocation.getDimValue().getId());
                pNewCache.put(key, dimvalId);
            }
        }
    }

    private String getKey(final DimLocation pDimLocation, final String
pDimensionName) {
        final String keyName = getRepositoryIdProperty();
                                    String     result     =     (String)
pDimLocation.getDimValue().getProperties().get(keyName);
        if (StringUtils.isBlank(result)) {
                                                    result    =    (String)
pDimLocation.getDimValue().getProperties().get("displayName");
        }
        if (StringUtils.isBlank(result)) {
            result = pDimLocation.getDimValue().getName();
        }
        result = pDimensionName + "+" + result;
        return result;
    }

    protected NavigationState getFilteredNavigationState() {
                        NavigationState   state   =   (NavigationState)
NucleusResolverUtil.resolveName("/atg/endeca/assembler/cartridge/manag
er/UnfilteredNavigationState");
        NavigationState navigationState = state.clearFilterState();
                          if    (mRecordFilterBuilders    !=    null    &&
mRecordFilterBuilders.length > 0) {
                            final   List<String>   recordFilters   =   new
```

```
ArrayList<String>();
                         for (final RecordFilterBuilder filter :
mRecordFilterBuilders) {
                recordFilters.add(filter.buildRecordFilter());
            }
                                                      navigationState    =
navigationState.updateRecordFilters(recordFilters);
        }
        return navigationState;
    }

    private void vlogDebug(String paramString, Object... paramVarArgs)
{
         AssemblerTools.getApplicationLogging().vlogDebug(paramString,
paramVarArgs);
    }

    public RecordFilterBuilder[] getRecordFilterBuilders() {
        return mRecordFilterBuilders;
    }

    /**
     * Sets the record filter builders.
     *
     * @param pRecordFilterBuilders the new record filter builders
     */
          public   void   setRecordFilterBuilders(RecordFilterBuilder[]
pRecordFilterBuilders) {
        mRecordFilterBuilders = pRecordFilterBuilders;
    }

        private  static  DimensionSearchMdexQuery  buildMdexQuery(final
DimensionSearchResultsConfig pItemConfig, final SearchFilter pSearch)
{
                         DimensionSearchMdexQuery   query   =   new
DimensionSearchMdexQuery();
                   if  (pItemConfig.getDimensionList()  !=  null  &&
pItemConfig.getDimensionList().size() > 0) {
```

```
            query.setDimensionValues(pItemConfig.getDimensionList());
        }
        query.setShowCountsEnabled(pItemConfig.isShowCountsEnabled());
                   if   (null   ==   pItemConfig.getRelRankStrategy()   ||
"".equals(pItemConfig.getRelRankStrategy())) {
            query.setRelRankStrategy(null);
        } else {
             query.setRelRankStrategy(pItemConfig.getRelRankStrategy())
;
        }
        if (null != pSearch) {
            query.setTerms(pSearch.getTerms());
        }
         query.setMaxDvalsPerDimension(pItemConfig.getMaxResultsPerDime
nsion());
        return query;
    }

    public Set<String> getDimensionNames() {
        return dimensionNames;
    }

    public void setDimensionNames(Set<String> dimensionNames) {
        this.dimensionNames = dimensionNames;
    }

    public DPCacheTools getCacheTools() {
        return cacheTools;
    }

    public void setCacheTools(DPCacheTools cacheTools) {
        this.cacheTools = cacheTools;
    }

    public DimensionSearchResultsConfig getConfig() {
        return config;
    }
```

```
    public void setConfig(DimensionSearchResultsConfig config) {
        this.config = config;
    }
}
```

1.7. Листинг кода класса OrderHybridService

```
public class OrderHybridService extends GenericService implements
OrderService {


    private OrderRepositoryService repositoryService;
    private OrderEndecaService endecaService;
    private Date lastDate;


    @Override
        public List<Map<String, Object>> getOrders(OrderSearchParams
params) {
                        List<Map<String, Object>> endecaOrders =
endecaService.getOrders(params);
        params.setDate(lastDate);
        Set<String> ids = new HashSet<>();
                     List<Map<String, Object>> repositoryOrders =
repositoryService.getOrders(params, ids);
        int limit = params.getLimit();
        if (repositoryOrders.size() > 0) {
            mergeOrders(params,ids,endecaOrders,repositoryOrders);
            if (repositoryOrders.size() > limit) {
                repositoryOrders = repositoryOrders.subList(0, limit);
            }
        } else {
            repositoryOrders = endecaOrders;
        }
        return repositoryOrders;
    }

        private void mergeOrders(OrderSearchParams params,Set<String>
ids,List<Map<String, Object>> endecaOrders,List<Map<String, Object>>
```

```
repositoryOrders){
        String sort = params.getSort().toString().toLowerCase();
        boolean ascending = params.isAscending();
        if ("date".equals(sort)) {
            sort = "lastModifiedDate";
        }
        int i = -1;
                        Iterator<Map<String,   Object>>   iterator   =
endecaOrders.iterator();
        Map<String, Object> endecaOrder = iterator.next();
        String id = (String) endecaOrder.get("id");
        Map<Integer, Map<String, Object>> positions = new HashMap<>();
        for (Map<String, Object> repositoryOrder : repositoryOrders) {
            i++;
            while (ids.contains(id)) {
                if (iterator.hasNext()) {
                    endecaOrder = iterator.next();
                    id = (String) endecaOrder.get("id");
                } else {
                    endecaOrder = null;
                    break;
                }
            }
            if (endecaOrder == null) {
                break;
            }
            Object rValue = repositoryOrder.get(sort);
            Object eValue = endecaOrder.get(sort);
                    boolean  needAdd  =  compare(rValue,  eValue,  sort,
ascending);
            if (needAdd) {
                positions.put(i, endecaOrder);
                if (iterator.hasNext()) {
                    endecaOrder = iterator.next();
                    id = (String) endecaOrder.get("id");
                } else {
                    break;
                }
```

```
            }
        }
        if (endecaOrder != null) {
            repositoryOrders.add(endecaOrder);
        }
        while (iterator.hasNext()) {
            repositoryOrders.add(iterator.next());
        }
        for (Integer key : positions.keySet()) {
            repositoryOrders.add(key, positions.get(key));
        }
    }

    private boolean compare(Object rValue, Object eValue, String sort,
boolean ascending) {
        int mul = ascending ? -1 : 1;
        int result;
        if ("lastModifiedDate".equals(sort)) {
            result = ((Date) eValue).compareTo(((Date) rValue));
        } else {
            result = ((String) eValue).compareTo(((String) rValue));
        }
        return result * mul > 0;
    }

    public OrderRepositoryService getRepositoryService() {
        return repositoryService;
    }

        public   void   setRepositoryService(OrderRepositoryService
repositoryService) {
        this.repositoryService = repositoryService;
    }

    public OrderEndecaService getEndecaService() {
        return endecaService;
    }
```

```
    public void setEndecaService(OrderEndecaService endecaService) {
        this.endecaService = endecaService;
    }

    public Date getLastDate() {
        return lastDate;
    }

    public void setLastDate(Date lastDate) {
        this.lastDate = lastDate;
    }

}
```