МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт кибербезопасности и цифровых технологий (ИКБ)
(наименование института, филиала)
Кафедра КБ-8 «Информационное противоборство»
(наименование кафедры)

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ НА ПРОИЗВОДСТВЕННУЮ ПРАКТИКУ

(указать вид практики: учебная / производственная)
Проектно-технологическая практика
(указать тип практики в соответствии с учебным планом)

Студенту 3 **курса учебной группы** ББББО-01-20
Шевченко Андрей Вадимович
(фамилия, имя и отчество)

Место и время практики: РТУ-МИРЭА
Должность на практике (при наличии): студент

1. СОДЕРЖАНИЕ ПРАКТИКИ:

1.1. Изучить: современные информационные технологии для поиска и обработки информации; нормативные правовые акты в области профессиональной деятельности.

1.2. Практически выполнить:провести исследование и работы в рамках проблематики ИБ социотехнических систем в соответствии с индивидуальным вариантом задания, в частности, выполнить работы по проектированию, разработке, установке, настройке, тестированию, сопровождению и обслуживанию программных и программно-аппаратных комплексов обеспечения безопасности социотехнических систем; обосновать полученные результаты и изложить их с соблюдением основных требований к оформлению научной продукции.

1.3. Ознакомиться: с методами работы с источниками информации, методами проектирования и создания концептов продуктов ИБ с использованием ЯП (Python); современными методами научных исследований и экспериментов.

2. ДОПОЛНИТЕЛЬНОЕ ЗАДАНИЕ: определить пути и перспективы дальнейшего развития полученных результатов.

3. ОРГАНИЗАЦИОННО-МЕТОДИЧЕСКИЕ УКАЗАНИЯ: перманентно рассматривать возможности повторного использования и интеграции разрабатываемых решений

Руководитель практики от кафедры
«08» февраля 2023 г. _____ К.К. Отраднов
Подпись *ФИО*

Задание получил:
«08» февраля 2023 г. _____ (А.В. Шевченко)
Подпись *ФИО*

СОГЛАСОВАНО:
Заведующий кафедрой:
«08» февраля 2023 г. _____ (В.Р. Григорьев)
Подпись *ФИО*

Москва 2023

Проведенные инструктажи:

Охрана труда: «08» февраля 2023 г.

Инструктирующий _____ (К.К. Отраднов, ст. преп.)
Подпись *Расшифровка, должность*

Инструктируемый _____ (А.В. Шевченко)
Подпись *Расшифровка*

Техника безопасности: «08» февраля 2023 г.

Инструктирующий _____ (К.К. Отраднов, ст. преп.)
Подпись *Расшифровка, должность*

Инструктируемый _____ (А.В. Шевченко)
Подпись *Расшифровка*

Пожарная безопасность: «08» февраля 2023 г.

Инструктирующий _____ (К.К. Отраднов, ст. преп.)
Подпись *Расшифровка, должность*

Инструктируемый _____ (А.В. Шевченко)
Подпись *Расшифровка*

_____ «08» февраля 2023 г.

Инструктирующий _____ (К.К. Отраднов, ст. преп.)
Подпись *Расшифровка, должность*

Инструктируемый _____ (А.В. Шевченко)
Подпись *Расшифровка*

С правилами внутреннего распорядка ознакомлен: «08» февраля 2023 г.

_____ (А.В. Шевченко)
Подпись *Расшифровка*

Москва 2023

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт кибербезопасности и цифровых технологий (ИКБ)

(наименование института, филиала)

Кафедра КБ-8 «Информационное противоборство»

(наименование кафедры)

ОТЧЁТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ

(указать вид практики: учебная / производственная)

Проектно-технологическая практика

(указать тип практики в соответствии с учебным планом)

Тема практики: «Проект обеспечения информационной безопасности бизнес процесса: деловая разведка»

приказ Университета о направлении на практику от «___» _____ 20 __ г. № _____

Отчет представлен к
рассмотрению:
Студент группы *ББББО-01-20* «__» _____ 20__ г. _____/А.В. Шевченко
(подпись и расшифровка подписи)

Отчет утвержден.
Допущен к защите:

Руководитель практики
от кафедры «__» _____ 20__ г. _____/ К.К. Отраднов
(подпись и расшифровка подписи)

Москва 2023

Москва 2023

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

РАБОЧИЙ ГРАФИК
ПРОВЕДЕНИЯ ПРОИЗВОДСТВЕННОЙ ПРАКТИКИ

студента Шевченко А.В 3 курса группы БББО-01-20 очной формы обучения, обучающегося по направлению подготовки 10.03.01 «Информационная безопасность», профиль «Организация и технологии защиты информации»

Неделя	Сроки выполнения	Этап	Отметка о выполнении
1-2	23.02.2023	Анализ предметной области задачи, составление литературного обзора по теме, постановка и анализ проблемы исследования	
3-4	09.03.2023	Первичное выявление функциональных и нефункциональных требований к автоматизированному средству решения задачи, составление обобщённых алгоритмов решения	
5-15	24.05.2023	Итерационная разработка и тестирование средства автоматизации решения задачи	
16	31.05.2023	Защита отчёта	

Руководитель практики от кафедры _____ /К.К. Отраднов/

Обучающийся _____ /А.В. Шевченко /

Согласовано:

Заведующий кафедрой _____ /В.Р. Григорьев, к.т.н./

Москва 2023

СОДЕРЖАНИЕ

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Бизнес-разведка – это анализ информации о внешнем окружении (потребителях, конкурентах и происходящих изменениях), позволяющий компании успешно развивать свой бизнес и добиваться превосходства над конкурентами, а также это сбор и обработка данных из разных источников для выработки управленческих решений с целью повышения конкурентоспособности коммерческой организации, проводимые в рамках закона и с соблюдением этических норм, а также структурного подразделения предприятия, выполняющего эти функции. [1]

В процессе сбора информации главным свойством информации является полнота и точность, которые оказывают сильное влияние на принятие верного управленческого решения.

Внешняя среда постоянно изменяется, оказывая влияние на результаты бизнеса.

Все сведения должны быть получены из доверительных открытых источников, это связано с тем, что из-за стремительного роста количество информации в сети Интернет возникает проблема достоверности информации.

В команде отдела деловой разведки должны быть:

1) Аналитики – сбор и анализ информации о рынке, конкурентах, тенденциях и других факторах.

2) Агенты – сбор информации на местах (например, в отелях, на выставках).

3) Юристы – анализ законодательства и правовой защиты компании.

4) Маркетологи – анализ потребительского спроса и трендов в индустрии.

5) Технические специалисты – анализ технологических инноваций и тенденций в отрасли.

6) Менеджеры – координация работы и принятие решений на основе собранной информации.

В процессе деловой разведки важным является анализ рынка, который включает в себя сбор данных о ситуации на рынке, интересах аудитории. [2]

Этапы анализа рынка:

– Сбор данных из доверительных источников.

– Анализ собранных сведений.

– Формирование выводов. [3]

1.1 Требования к разведывательной информации

Разведывательная информация имеет характеристики: качественные, количественные и ценностные.

К качественным характеристикам относится:

– Достоверность информации – информация должны быть приближенная к первоисточнику.

– Объективность информации – информация должна быть отражена в реальности, то есть она должна быть очищена от искажений.

– Однозначность – информация может вести к множеству последствий, поэтому нужно выбирать информацию с однозначными последствиями.

– Чистота информации – источники информации должны быть максимально приближены к месту появления информации.

К количественным характеристикам относится:

– Полнота информации – информация должна быть исчерпывающей и соответствовать целям разведки.

– Релевантность информации – информация должна быть максимальна приближенной к поставленной задаче разведки.

К ценностным характеристикам относится:

– Актуальность информации – информация должна быть своевременной и новой.

– Стоимость информации – сколько потребуется средств на получение информации.

1.2 Нормативно-правовая база

Правовая база является неотъемлемой частью деловой разведки и играет важную роль в обеспечении её законности. Законодательство регламентирует правила и порядок работы с информацией, а также определяет ответственность за незаконные действия.

Например, в ряде стран, включая Россию, существуют специальные законы о государственной, военной и экономической разведке, которые регулируют все стороны этой деятельности. Кроме того, ответственность за нарушение законодательства обязательна, если деловая разведка была проведена незаконным способом.

Таким образом, правовая база является главным фактором, который определяет границы законности в деятельности по деловой разведке. Правильно спроектированная и хорошо организованная работа в соответствии с правилами и законами помогает избежать проблем, связанных с законностью деятельности по деловой разведке, а также обеспечивает эффективность сбора и использования информации в организации.

Основные федеральные законы, которые ограничивают компании в сфере деловой разведки:

1) ФЗ от 27 декабря 1991 г. № 2124-1 (ред. От 29.12.2022) «О средствах массовой информации». [14]

2) ФЗ от 21 июля 1993 г. № 5485-1 (ред. от 5 января 2022 г.) «О государственной тайне». [15]

3) ФЗ от 27 июля 2006 г. № 149-ФЗ (ред. от 29 января 2022 г.) «Об информации, информационных технологиях и защите информации». [16]

4) ФЗ от 30 декабря 2004 г. № 218-ФЗ (ред. от 28 января 2022 г.) «О кредитных историях». [17]

5) ФЗ от 29 июля 2004 г. № 98-ФЗ (ред. от 14 июля 2018 г.) «О коммерческой тайне». [18]

1.3 Основные методы сбора информации

Методы деловой разведки бывают различными и зависят от конкретных целей и задач разведывательной деятельности. Самыми популярными являются:

– Сбор сведений из открытых источников. (СМИ, интернет, корпоративные издания и пресса, отраслевые библиотеки и т. п.)

– Мониторинг – наблюдение за активностью компании.

– Использование баз данных.

– Опросы и консультирование.

– Общение на специализированных форумах.

– Посещение выставок и конференций.

– Покупка образцов продукции конкурентов.

– Сбор информации под видом соискателя.

– Использование фальшивых вакансий.

– Использование связей, знакомств.

– Беседы со специалистами организации-конкурента.

– Переманивание ключевых специалистов. [4]

Важно отметить, что все методы деловой разведки должны применяться в соответствии с действующим законодательством, чтобы избежать нарушений прав конкурентов или других законных участников рынка.

1.4 OSINT-разведка

Открытый источниковый анализ (OSINT) — это методология, которая использует открытые источники, такие как социальные сети, интернет, открытые базы данных, а также различные источники общественной информации для получения информации о конкретном объекте (человек, компания, группа, организация, событие и т. д.). OSINT разведка является хорошим способом сбора информации в целях деловой разведки.

Методы OSINT могут варьироваться от простых поисков на открытых интернет-ресурсах до сложных аналитических методов, использующих машинное обучение и искусственный интеллект. Некоторыми известными сервисами могут служить Google, Facebook, LinkedIn, Twitter и другие социальные сети.

OSINT разведчики также могут использовать специальное программное обеспечение, такое как Palantir, Maltego, Spiderfoot, Echosec, Hunchly и другие, которые помогают автоматизировать процесс анализа и получения информации.

Результатом сбора информации является большое количество различных данных, которые затем должны быть оценены и отфильтрованы, чтобы получить актуальную и достоверную информацию.

С помощью OSINT можно:

– Получать максимально объективную и полезную информацию для принятия решений.

– Получать конкурентные преимущества для своей организации или ее продукта.

– Находить недостатки и уязвимости в собственной системе безопасности, защите конфиденциальных сведений о клиентах.

– Понимать психологические особенности, потребности, привычки представителей целевой аудитории.

В IT-индустрии и информационной безопасности OSINT помогает:

– Собирать информацию о конкурентах и искать конкурентные преимущества.

– Анализировать защищенность объекта, выявлять уязвимые точки системы безопасности.

– Находить информационные утечки.

– Выявлять возможные угрозы, их источники и направленность.

– Анализировать киберпреступления (кражи данных, взломы и т.д.). [19]

Одним из наиболее полезных инструментов для поиска информации о компании является поиск в новостных источниках. Этот метод дает возможность найти значимую информацию о компании, включая новые продукты, достижения, связанные с общественными отношениями, проблемы со здоровьем партнеров и сотрудников и другую актуальную информацию. Поиск в новостных источниках является важным инструментом для получения полного и точного понимания состояния и работы компании в настоящее время.

Парсинг новостей — это процесс автоматического сбора новостей и другой информации из новостных источников с использованием программных инструментов. Этот процесс позволяет собирать новости и другую информацию из различных источников и использовать ее для анализа текущей ситуации в определенной области или направлении.

Парсинг новостей может применяться в различных сферах, включая начальные этапы разведывательной работы, анализ ситуации в бизнесе, конкурентный анализ в сфере маркетинга, анализ социальных медиа и даже для исследования тенденций в правительственных вопросах.

Парсер новостей может быть полезным инструментом для деловой разведки. Парсинг новостей позволяет автоматически собирать информацию из различных источников, включая новости компании, информацию об инвестициях, уровень рентабельности, изменение владельцев, перестановки в руководстве, проблемы в производстве и другую актуальную информацию.

С помощью парсера данных можно автоматически отслеживать новостные публикации о конкурентах, следить за различными изменениями в компании и анализировать соответствующую информацию. Этот инструмент может также использоваться для сбора информации об инвестициях и финансовых показателях разных компаний, что позволяет сделать более обоснованный выбор в инвестировании денег в деловую среду.

1.5 База данных для отдела деловой разведки

База данных может быть полезным инструментом для выполнения деловой разведки, поскольку она содержит собранную информацию о компаниях, их продуктах, клиентах, промышленных трендах и других релевантных данных. С помощью базы данных можно исследовать отрасль, определить ключевых игроков, изучить конкурентных лидеров и помочь сформулировать более эффективные бизнес-стратегии. [8]

Несколько примеров практического использования баз данных в деловой разведке:

1) Изучение конкурентов: коммерческие базы данных могут помочь узнать, кто является конкурентами, какие продукты они предлагают, какими методами продвижения пользуются, и какие их сильные и слабые стороны. Информация о конкурентах может помочь определить свои преимущества и конкурентные преимущества, а также сформулировать более эффективные стратегии.

2) Исследование отрасли: базы данных могут содержать информацию о тенденциях в отрасли, предпочтениях и потребностях клиентов, официальных отчетах, экономической статистике и т. д. Это может помочь коммерческим фирмам сделать более интеллектуальное прогнозирование и решения.

3) Поиск новых рынков: базы данных могут помочь найти новые рынки или новые демографические группы, которые могут быть заинтересованы в продуктах и услугах.

Таким образом, база данных может стать ценным инструментом, который поможет найти ответы на вопросы, которые могут привести к успешным бизнес-решениям.

Концептуальный и логический этап проектирования базы данных.

1) Компания:

а) Название компании

б) Описание компании

в) Отрасль

г) Год основания

д) Количество сотрудников

е) Адрес: страна, город, адрес

ж) Доход, прибыль, активы, инвестиции, дивиденды, дата изменения

з) Продукты / услуги

и) Конкуренты

к) Новости / Публикации

л) Тенденции / Прогнозирование роста

2) Контакт:

а) Имя

б) Фамилия

в) Должность

г) Номер телефона, электронная почта

д) Компания

3) Страна:

а) Название страны

б) Население

в) ВВП, рост ВВП, инфляция, безработица, инвестиции

г) Политическая система, юридическая система, стабильность, отношения с другими странами

д) Отрасль

е) Дата изменения

4) Медиа:

а) Название медиа

б) Ссылка

1.6 Маркетинговые метрики для деловой разведки

Маркетинговые метрики в деловой разведке играют важную роль в оценке и улучшении эффективности программы деловой разведки. Они позволяют оценить результаты проведенной работы, выявить успехи и проблемы, определить, какие действия улучшают работу отдела деловой разведки и что нужно корректировать. Использование маркетинговых метрик в деловой разведке может помочь:

1) Определить цели: метрики могут помочь определить конкретные цели, которые надо достичь.

2) Оценить эффективность: метрики помогают оценить эффективность работы отдела деловой разведки, понять слабые и сильные стороны компаний.

3) Изменить стратегию: на основе метрик можно изменять стратегию деловой разведки и планировать будущие действия компании.

4) Принять решения: маркетинговые метрики могут помочь компании принимать более уверенные и обоснованные решения, основанные на данных и аналитике.

5) Гибко реагировать на изменения: маркетинговые метрики могут помочь компании реагировать на изменения в рыночных условиях и быстро реагировать на изменения в конкурентов.

Таким образом, маркетинговые метрики в деловой разведке играют важную роль в оценке и улучшении работы отдела деловой разведки, позволяют компании объективно оценивать эффективность своих действий и корректировать свое разведывательное поведение.

Существует множество маркетинговых метрик, которые можно использовать для измерения конкурентоспособности компании.

1.6.1 Метрика – Средний размер компании в отрасли

Средний размер компании в отрасли показывает среднее количество работников или размер компании в данной отрасли. Это значение может быть рассчитано как среднее арифметическое количества работников или размера компаний в данной отрасли.

По формуле 1.6.1.1 можно рассчитать средний размер компании в отрасли.

$$\text{СрРК} = \frac{\sum \text{РК}}{\text{кК}} \qquad (1.6.1.1)$$

где СрРК – средний размер компании (ед.);

РК – размер компании (ед.);

кК – количество компаний (ед.).

Для расчёта этой метрики необходимо учитывать все компании в отрасли, включая малые, средние и крупные компании.

Польза от использования этой метрики заключается в том, что она позволяет оценить структуру и состояние отрасли. Например, если средний размер компаний в отрасли невелик, это может говорить о том, что отрасль еще не насыщена большими игроками или она находится в процессе развития. Если же средний размер компаний в отрасли высокий, это может указывать на

высокую консолидацию рынка или наличие нескольких доминирующих игроков.

Кроме того, данная метрика может быть полезна для сравнения отдельных компаний в отрасли. Если размер компании значительно выше или ниже, чем средний размер в отрасли, это может означать, что компания имеет особенности в своей структуре и управлении или находится на стадии роста или спада.

1.6.2 Метрика – Коэффициент рентабельности

Коэффициент рентабельности показывает, сколько прибыли остается у компании после вычета всех расходов. Это показатель, который говорит о том, насколько эффективно компания использует свои ресурсы и осуществляет свою деятельность.

По формуле 1.6.2.1 можно рассчитать коэффициент рентабельности.

$$\text{коэффР} = \frac{\text{П}}{\text{Выр}} * 100\% \qquad (1.6.2.1)$$

где коэффР – коэффициент рентабельности (процент);

П – чистая прибыль (руб.);

Выр – выручка (руб.).

Коэффициент рентабельности может рассчитываться как для всего бизнеса в целом, так и для каждого продукта, услуги или проекта отдельно.

Польза от использования этой метрики заключается в том, что она может помочь руководству компании принимать решения об улучшении работы бизнеса, определять наиболее прибыльные направления деятельности и проводить анализ конкурентов.

Например, если коэффициент рентабельности у компании слишком низкий, это может указывать на то, что расходы компании недостаточно контролируются или, что она не вырабатывает достаточно прибыли по своей основной деятельности.

Таким образом, коэффициент рентабельности является важным инструментом для определения эффективности работы компании и позволяет ее руководству принимать обоснованные решения и осуществлять контроль за финансовым состоянием бизнеса.

1.6.3 Метрика – Доля компаний с положительной прибылью

Доля компаний с положительной прибылью показывает, какая часть компаний в определенной отрасли заработала прибыль за определенный период времени. Эта метрика позволяет оценить, насколько успешным была отрасль в целом, а также сравнить производительность компаний в ней.

По формуле 1.6.3.1 можно рассчитать долю компаний с положительной прибылью.

$$дК_{П>0} = \frac{кК_{П>0}}{кК} \qquad (1.6.3.1)$$

где $дК_{П>0}$ – доля компаний с положительной прибылью (ед.);

$кК_{П>0}$ – количество компаний с положительной прибылью (ед.);

кК – общее количество компаний (ед.).

Польза от использования этой метрики заключается в том, что она может помочь предпринимателям, инвесторам и аналитикам оценить финансовое состояние отрасли и принять решения об инвестировании в этой отрасли.

Например, если доля компаний с положительной прибылью высока, это может указывать на то, что отрасль в целом процветает и имеет высокую

эффективность производства. Если же доля компаний с положительной прибылью низкая, это может свидетельствовать о проблемах в отрасли, таких как недостаточно высокие цены на продукцию, неэффективное использование ресурсов и других факторах.

Кроме того, данная метрика может использоваться для сравнения компаний между собой в одной отрасли. Например, если доля компаний с положительной прибылью для одной компании выше, чем для другой, это может указывать на более высокую эффективность управления этой компанией или на наличие у нее более конкурентоспособной продукции.

Таким образом, метрика доли компаний с положительной прибылью является важным инструментом для анализа финансового состояния отрасли и компаний в ней, а также для принятия обоснованных решений, связанных с инвестированием и развитием бизнеса.

1.6.4 Метрика – Отношение средневзвешенного роста

Отношение средневзвешенного роста (Growth Rate Weighted Ratio) позволяет оценить отношение роста средневзвешенных значений между периодами времени. Эта метрика позволяет оценить, как изменялась средневзвешенная мера роста за определенный период времени, отражая продвижение вперед в отрасли или компании.

По формуле 1.6.4.1 можно рассчитать отношение средневзвешенного роста.

$$GRWR = \frac{\sum(\frac{\text{Выр}_t - \text{Выр}_{t-1}}{\text{Выр}_{t-1}} * \text{Акт})}{\sum \text{Акт}} - 1 \qquad (1.6.4.1)$$

где $GRWR$ – отношение средневзвешенного роста (ед.);

Выр_t – выручка компании в момент времени t (руб.);

Выр_{t-1} – выручка компании в момент времени t-1 (руб.);

Акт – активы компании (руб.).

Growth Rate Weighted Ratio очень полезна при оценке прогресса в стабильной отрасли. Эта метрика позволяет оценить, насколько эффективно компании в этой отрасли развиваются, и какова скорость роста отрасли в целом. Если результаты этой метрики положительны, то это может свидетельствовать о том, что рынок в отрасли сильно растет в течение периода, а если результаты отрицательны, то это может указывать на спад в отрасли.

В целом, метрика отношения средневзвешенного роста может быть полезна при прогнозировании роста отрасли и анализе успешности компании в ней. Кроме того, она может помочь компаниям принимать решения об инвестировании в развитие бизнеса и планировании дальнейших действий.

1.6.5 Метрика – Растущая прибыль компаний во времени

Растущая прибыль компаний во времени (Growing Profitability Over Time) используется для измерения тенденции увеличения прибыльности компании в течение времени. Эта метрика позволяет оценить, как компания растет с течением времени и оценить ее финансовое положение на периоды.

По формуле 1.6.5.1 можно рассчитать растущую прибыль компании во времени.

$$GPOT = \frac{\text{Выр}_t - \text{Выр}_{t-1}}{\text{Выр}_{t-1}} * 100\% \qquad (1.6.5.1)$$

где $GPOT$ – растущая прибыль компании во времени (проц.);

Выр_t – выручка компании в момент времени t (руб.);

Выр_{t-1} – выручка компании в момент времени t-1 (руб.).

Данный показатель может быть рассчитан для отдельных компаний и отраслей в целом.

Польза от использования этой метрики заключается в том, что она позволяет оценить финансовое здоровье компании и ее возможности для роста и развития в будущем. Если метрика растущая прибыль компаний во времени высока, это означает, что компания постепенно увеличивает свою прибыльность, что является положительным сигналом для ее долгосрочного.

1.6.6 Метрика – Риска компании

Риск компании — это индикатор, который измеряет стабильность и надежность компании в отношении ее финансового состояния. Риск компании может зависеть от различных факторов, таких как уровень задолженности, изменения цен на управляемые активы, конкурентная среда и т.д.

По формуле 1.6.6.1 можно рассчитать риск компании.

$$\text{Риск} = \frac{\text{Акт} - \text{Инв}}{\text{Акт}} * 100\% \qquad (1.6.6.1)$$

где Риск – риск компании (проц.);

Акт – активы компании (руб.);

Инв – инвестиции компании (руб.).

Польза метрики риска компании заключается в том, что она позволяет инвесторам принимать обоснованные решения по выбору инвестиционных проектов и планированию уровня риска в своих инвестиционных портфелях. Она может использоваться как внутренний инструмент для оценки риска компании менеджментом, так и внешний инструмент для оценки рисков компании потенциальными инвесторами и кредиторами.

Отметим, что метрика риска компании не является гарантией финансовой стабильности компании и ее успешности в будущем.

1.6.7 Метрика – Доля компании, инвестируемой в научно-исследовательскую деятельность

Доля компании, инвестируемой в научно-исследовательскую деятельность, отражает долю финансовых ресурсов, выделенных компанией на научно-исследовательские и развивающие проекты.

По формуле 1.6.7.1 можно рассчитать долю компании, инвестируемой в научно-исследовательскую деятельность.

$$\text{Нд} = \frac{\text{Инв}}{\text{Выр}} \qquad (1.6.7.1)$$

где Нд – научная доля компании (ед.);

Инв – инвестиции компании (руб.);

Выр – выручка компании (руб.).

Польза метрики доли компании, инвестируемой в научно-исследовательскую деятельность, заключается в возможности оценить степень инновационной активности и потенциала компании для будущего роста и развития. Такая метрика может помочь инвесторам изучить потенциал компании для инвестиций в долгосрочной перспективе, а также помочь компаниям контролировать и анализировать свои расходы на исследования и развитие.

Отметим, что метрика доли компании, инвестируемой в научно-исследовательскую деятельность, не является единственным показателем, который следует учитывать при принятии инвестиционных решений. Она

должна рассматриваться в контексте других индикаторов, таких как прибыльность, рыночная доля и т.д.

1.6.8 Метрика – Коэффициент эффективности использования имущества

Коэффициент эффективности использования имущества — это индикатор, который показывает, как компания использует свое имущество для генерации прибыли. Эта метрика помогает определить, насколько эффективно используются активы компании и какие изменения могут потребоваться для увеличения ее прибыльности.

По формуле 1.6.8.1 можно рассчитать коэффициент эффективности использования имущества.

$$\text{коэффЭИИ} = \frac{\text{Выр}}{\text{Акт}} \qquad (1.6.8.1)$$

где коэффЭИИ – коэффициент эффективности использования имущества (ед.);

Выр – выручка компании (руб.);

Акт – активы компании (руб.).

Польза метрики коэффициента эффективности использования имущества заключается в том, что она позволяет оценить эффективность использования ресурсов и принимать правильные решения по управлению своими активами. Это может помочь компаниям увеличивать доход, уменьшать издержки и создавать более стабильную финансовую базу для будущих инвестиций.

Отметим, что метрика коэффициента эффективности использования имущества должна рассматриваться с учетом других финансовых параметров компании, а также ее отраслевой специфики. Кроме того, она не должна

рассматриваться в отрыве от других факторов, влияющих на финансовое состояние компании.

2 СТРУКТУРА И ФУНКЦИИ СЛУЖБЫ ДЕЛОВОЙ РАЗВЕДКИ С УЧЁТОМ НЕОБХОДИМОСТИ КОМПЛЕКСНОЙ ИНФОРМАЦИОННОЙ ЗАЩИТЫ СОЦИОТЕХНИЧЕСКИХ ПОДСИСТЕМ ОРГАНИЗАЦИИ

Функциональные требования — это требования, которые описывают функциональность программного продукта и его возможности. Они определяют, какие действия может выполнять система и как она должна реагировать на определенные входные данные. [10]

Для отдела деловой разведки функциональные требования зависят от компании, общими можно выделить:

1) Система должна иметь возможность собирать информацию из различных источников, включая открытые источники, социальные сети, базы данных компаний и другие места.

2) Система должна предоставлять пользователю возможность указывать параметры поиска, чтобы получать только необходимую информацию.

3) Система должна предоставить пользователю возможность сохранить или удалить данные из базы данных.

Система состоит из множества инструментов OSINT-разведки, управление базой данных, работой с SQL-запросами, в результате которых получаются метрики для анализа компаний.

Нефункциональные требования системы деловой разведки — это требования, которые не связаны напрямую с функциональностью системы, но являются важными для обеспечения качества ее работы и удовлетворения потребностей пользователей. [10] Ниже приведены основные категории нефункциональных требований, которые важны для системы деловой разведки.

1) Производительность: система должна обеспечивать быстродействие и корректность выполнения функций даже в условиях большой нагрузки,

количество пользователей системы должно быть распределено по ролям, позволяющим оптимальный доступ к данным.

2) Надежность: система должна быть надежной и стабильной, обеспечивая работу при критических ситуациях, сохранение и защиту всех данных пользователей.

3) Безопасность: система должна обеспечивать конфиденциальность и безопасность хранимой информации, иметь защиту от взломов, вирусов, DDOS атак.

4) Масштабируемость: система должна обеспечивать возможность масштабирования при росте количества пользователей и объемах хранимой информации.

Все перечисленные нефункциональные требования являются важными, так как влияют на работу системы с точки зрения ее качества, скорости и безопасности. Их учет при разработке системы деловой разведки позволит обеспечить ее эффективную работу в долгосрочной перспективе.

2.1 Функциональные требования

Карточки пользовательских историй — это краткие описания того, что необходимо создать в программном продукте для того, чтобы удовлетворить потребности пользователей. Они используются командами разработки программного обеспечения для описания требований к продукту с учетом потребностей пользователей. Карточки пользовательских историй содержат информацию об имени пользователя, описании задачи, целях создания новой функции и способе проверки готовности продукта. Использование карточек пользовательских историй помогает командам разработки улучшить понимание клиентских потребностей и сократить время на разработку продукта. [10]

Преимущества использования карточек пользовательских историй:

– Улучшение понимания пользовательских требований и потребностей;

– Сокращение времени на разработку и управление проектом;

– Снижение риска неправильного понимания требований;

– Улучшение коммуникации между командами разработки, тестирования, продукта и заказчиком;

– Повышение удовлетворенности клиентов за счет своевременной и правильной поставки функционала, в котором они действительно нуждаются.

В процессе анализа предметной области были выделены несколько ролей и функций, которые выполняет сотрудник. На основе их была реализована карточка пользовательских историй (рисунок 2.1.1).

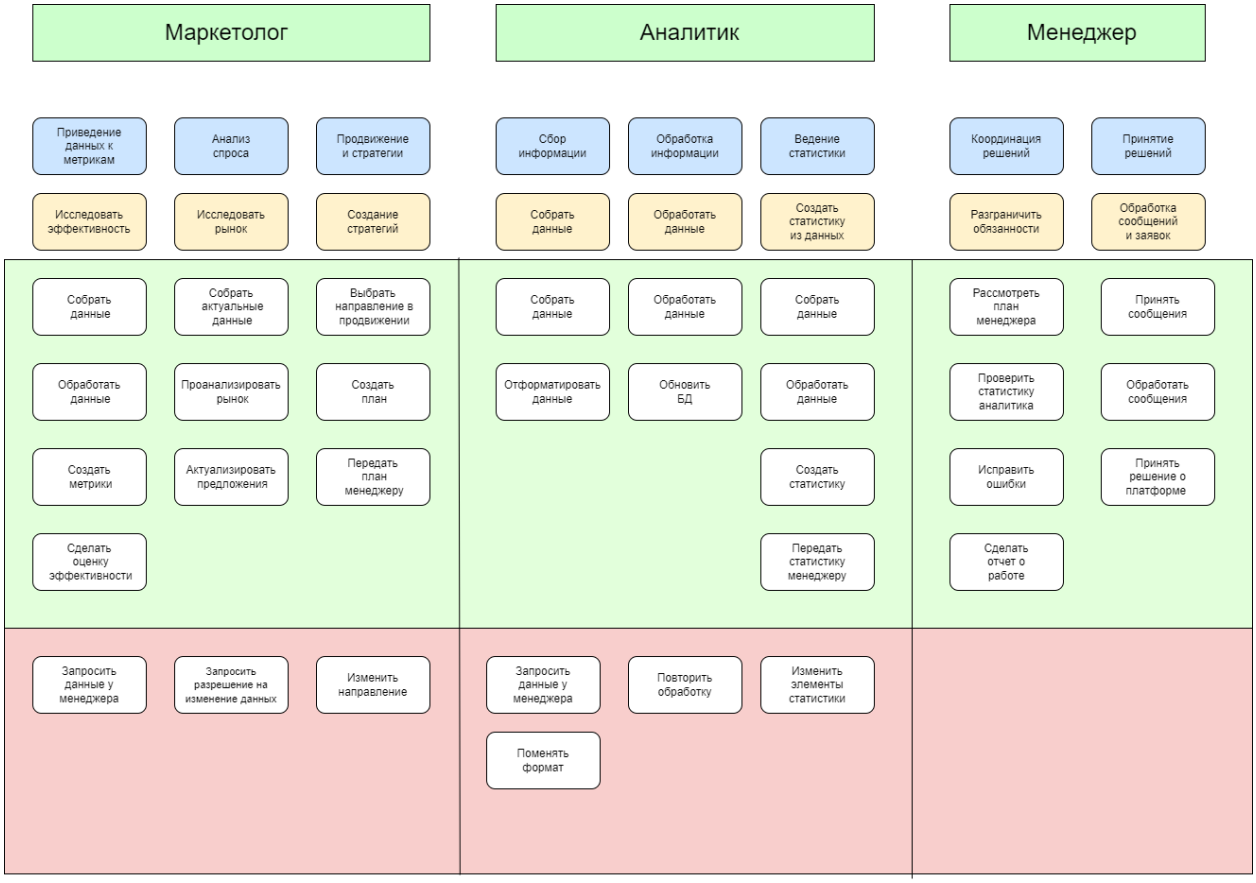

Рисунок 2.1.1 – Карточка пользовательских историй.

Имеются 3 роли при работе с системой: Менеджер, Аналитик, Менеджер.

Маркетолог может провести анализ данных и составить отчет.

Аналитик собирает данные и проводит предобработку, нормализуя и избавляясь от нулевых данных, также аналитик проводит статистических анализ данных.

Менеджер запрашивает информацию от Маркетологов и Аналитиков, а также проводит анализ отчетов, в результате чего предоставляет лучшее решение поставленной задачи.

2.2 Нефункциональные требования

Нефункциональные требования описывают характеристики системы, в каких условиях и как именно должны быть реализованы функциональные требования, чтобы достичь поставленной разведывательной задачи.

2.2.1 Производительность

Система должна обеспечивать быстродействие и корректность выполнения функций даже в условиях большой нагрузки. Для этого система должна иметь достаточную вычислительную мощность и использовать оптимальные алгоритмы для обработки и хранения данных. Также важно предусмотреть распределение пользователей системы по ролям, позволяющим оптимальный доступ к данным.

Главная цель производительности: обеспечение быстродействия, каждый пользователь должен получать доступ к необходимой информации быстро и без задержек.

Задача разработчика: оптимизировать архитектуру и код системы для обеспечения быстродействия.

2.2.2 Надежность

Система должна быть надежной и стабильной, обеспечивая работу при критических ситуациях. Это означает, что система должна быть защищена от сбоев, иметь систему резервного копирования данных и быстро восстанавливаться при непредвиденных сбоях.

Главная цель надежности: обеспечение продолжительного и корректного функционирования системы, чтобы обеспечивать непрерывную доступность к ресурсам и предотвращение потенциальных сбоев, с целью уменьшить риски для пользователей.

Задача разработчика: разработать механизмы автоматического восстановления системы после сбоев, создать тестовые сценарии и проверить работу системы на устойчивость.

2.2.3 Безопасность

Система должна обеспечивать конфиденциальность и безопасность хранимой информации. Это означает, что система должна иметь защиту от взломов, вирусов, DDOS атак, а также обеспечивать проверку прав доступа пользователей к информации и возможность контроля за доступом к данным.

Главная цель безопасности: обеспечение защиты конфиденциальных данных и бизнес-информации, передаваемой или хранящейся в системе, с целью предотвращения несанкционированного доступа.

Задача разработчика: обеспечить использование шифрования, аутентификацию и авторизацию пользователей, а также защиту хранилищ данных.

2.2.4 Масштабируемость

Система должна обеспечивать возможность масштабирования при росте количества пользователей и объемах хранимой информации. Это означает, что система должна быть готова к росту числа пользователей и увеличению объема хранимых данных, например, путем использования облачных технологий.

Главная цель масштабируемости: гибкость системы с возможностью расширения изначального объема функционала или функций для обеспечения работоспособности при росте количества пользователей.

Задача разработчика: разработать гибкую архитектуру для легкого масштабирования системы в соответствии с ростом количества пользователей.

3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО КОМПЛЕКСА ДЛЯ ФУНКЦИОНИРОВАНИЯ ОТДЕЛА ОРГАНИЗАЦИИ ДЛЯ ЭФФЕКТИВНОЙ И БЕЗОПАСНОЙ РЕАЛИЗАЦИИ ДЕЛОВОЙ РАЗВЕДКИ

Реализация деловой разведки является важным этапом в деятельности организации, позволяющим получить необходимую информацию о конкурентах, рынке, новых технологиях и прочих факторах для принятия управленческих решений и обеспечения безопасности. Однако для эффективности и безопасности такой деятельности необходимо создать программный комплекс для функционирования отдела организации, который обеспечит сбор, анализ и хранение информации, мониторинг деятельности конкурентов, а также обеспечит конфиденциальность и безопасность хранения и обработки информации.

3.1 Состав программного обеспечения

Состав программного обеспечения имеет важное значение для деловой разведки. В основе бизнес-процесса лежит сбор информации из новостных источников, хранение данных и аналитический анализ данных.

3.1.1 Программное обеспечения для сбора информации

Парсер новостных источников может быть полезным инструментов для выполнения деловой разведки, поскольку помогает мониторить последние события, связанные с компанией. С помощью парсера новостей можно исследовать новые тренды компаний и ситуации, которые влияют на конкурента.

Основной функцией парсера - проведение поиска новостей по новостным источникам, которым доверяет компания.

Данные собранные парсером можно будет использовать для сохранения данных в базе данных и дальнейшего исследования.

3.1.2 Реализация хранения данных деловой разведки

База данных может быть полезным инструментом для выполнения деловой разведки, поскольку она содержит собранную информацию о компаниях, их продуктах, клиентах, промышленных трендах и других релевантных данных. С помощью базы данных можно исследовать отрасль, определить ключевых игроков, изучить конкурентных лидеров и помочь сформулировать более эффективные бизнес-стратегии.

В процессе даталогического проектирования были выделены спецификации атрибутов всех таблиц (Таблицы 3.1.2.1 – 3.1.2.4). Для наглядности и выделения взаимоотношений между таблицами спроектирована ER-диаграмма (Рисунок 3.1.2.1).

Таблица 3.1.2.1 - Спецификация атрибутов сущности Company.

Атрибут	Наименование	Тип данных
Название компании	Company_name	Text
Описание компании	Company_description	Text
Отрасль	Industry	Text
Год основания	Year_established	Text
Количество сотрудников	Number_of_employees	Integer
Страна	Location_country	Text
Город	Location_city	Text
Адрес	Address	Text
Выручка	Revenue	Integer

Продолжение таблицы 3.1.2.1

Атрибут	Наименование	Тип данных
Прибыль	Profit	Integer
Активы	Assets	Integer
Инвестиции	Investments	Integer
Дивиденды	Dividends	Integer
Дата изменений	Modification_date	Date
Продукт / услуга	Products_services	Text
Конкуренты	Competitors	text
Новости	New_publications	Text
Тенденция	Growth_trends_forecasting	text

Таблица 3.1.2.2 - Спецификация атрибутов сущности Contact.

Атрибут	Наименование	Тип данных
Имя	First_name	Text
Фамилия	Last_name	Text
Должность	Position	Text
Компания	Company_name	Text
Номер телефона	Phone_number	Text
Майл	email	Text

Таблица 3.1.2.3 - Спецификация атрибутов сущности Country.

Атрибут	Наименование	Тип данных
Название страны	Country_name	Text
Численность населения	Population	Text
ВВП	GDP	Numeric(9,4)
Рост ВВП	GDP_growth	Numeric(9,4)
Инфляция	Inflation	Numeric(9,4)
Безработица	Unemployment	Numeric(9,4)
Инвестиции	Investments	Numeric(9,4)
Политическая система	Political_system	Text
Юридическая система	Legal_system	Text
Стабильность	Stability	Text

Продолжение таблицы 3.1.2

Атрибут	Наименование	Тип данных
Отношения с другими странами	Relation_with_other_countries	Text
Индустрия	Industry	Text
Дата изменений	Modification_date	Date

Таблица 3.1.2.4 - Спецификация атрибутов сущности Media.

Атрибут	Наименование	Тип данных
Название медиа-источника	Media-name	Text
Ссылка	url	Text

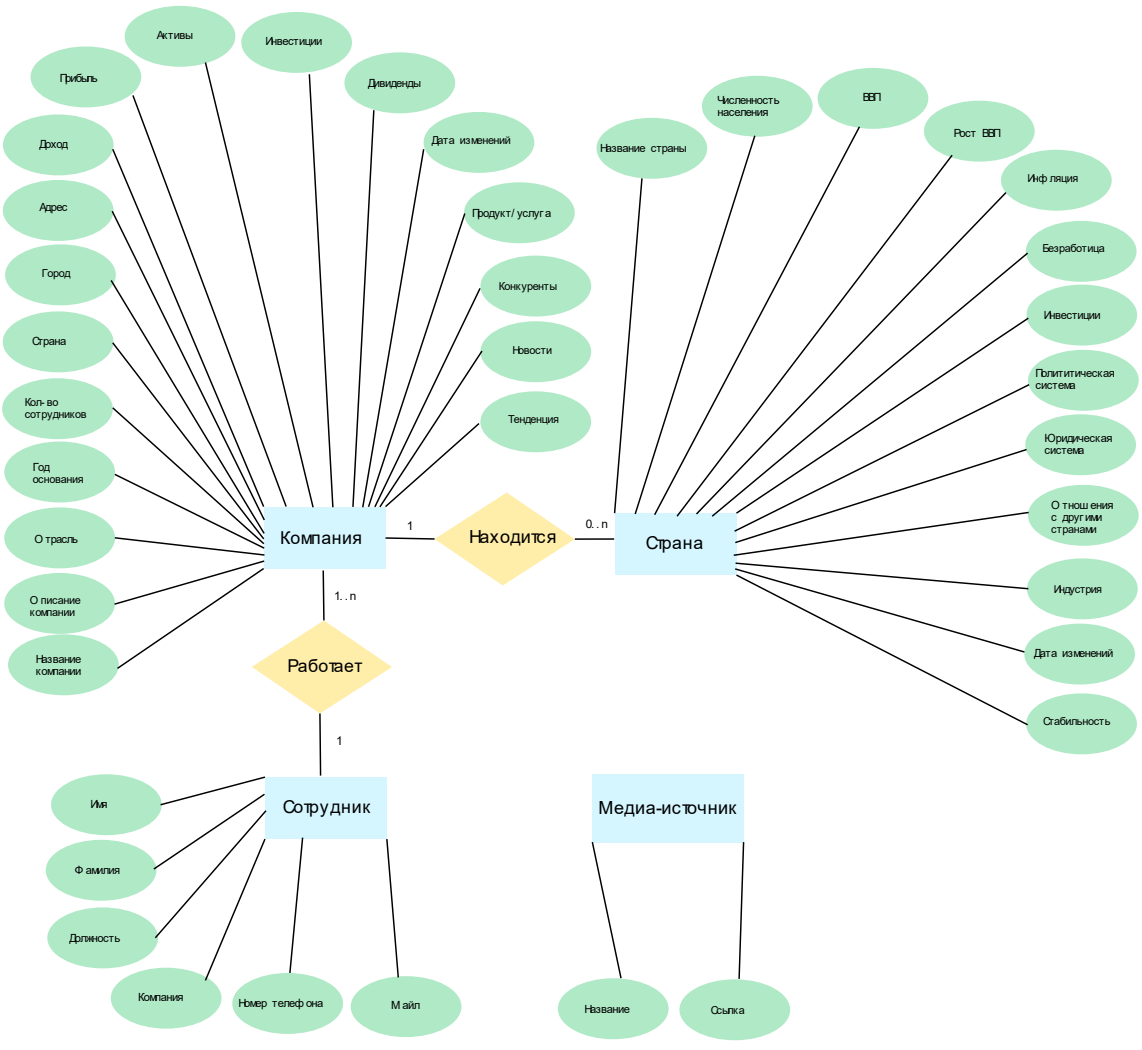

Рисунок 3.1.2.1 – ER-диаграммы базы данных

3.1.3 Анализ данных для деловой разведки с помощью SQL

Средний размер компаний в отрасли — это метрика, которая может быть использована для оценки типичного размера компаний в определенной отрасли. Она рассчитывается путем вычисления среднего числа сотрудников в компании в каждой отрасли (листинг 3.1.3.1).

Листинг 3.1.3.1 - Запрос среднего размера компании в отрасли

```
SELECT industry, AVG(number_of_employees) as avg_company_size
FROM company
GROUP BY industry;
```

Коэффициент рентабельности — это показатель, который выражает отношение прибыли к выручке за определенный период времени. Эта метрика отображает эффективность управления компанией в генерации прибыли из реализации продуктов или услуг (листинг 3.1.3.2).

Листинг 3.1.3.2 - Запрос коэффициента рентабельности

```
SELECT company_name,
(CASE WHEN revenue <> 0 THEN profit::numeric/revenue ELSE 0 END) * 100 AS profit_margin
FROM company_finance
ORDER BY profit_margin DESC;
```

Доля компаний с положительной прибылью — это метрика, которая вычисляет процент компаний, которые получают положительную чистую прибыль от продажи своей продукции или услуг. Этот показатель показывает, насколько успешно ведется бизнес в отрасли (листинг 3.1.3.3).

Листинг 3.1.3.3 - Запрос доли компаний с положительной прибылью

```
SELECT COUNT(*) / (SELECT COUNT(*) FROM company) * 100 AS percentage_profitable
FROM company_finance
WHERE profit > 0;
```

Отношение средневзвешенного роста — это метрика, которая вычисляет среднегодовой прирост выручки и делит его на общее значение активов

компании, что позволяет оценить рост компании в контексте используемых активов. Она используется для сравнения роста разных компаний в отрасли друг с другом (листинг 3.1.3.4).

Листинг 3.1.3.4 - Запрос отношений средневзвешенного роста

```
WITH revenue_growth AS (
  SELECT company_name, (revenue-lag(revenue) OVER (PARTITION BY company_name ORDER BY
modification_date))/lag(revenue) OVER (PARTITION BY company_name ORDER BY modification_date) AS
revenue_growth
  FROM company_finance
), avg_weighted_growth AS (
SELECT industry, SUM(revenue_growth*assets)/SUM(assets) AS avg_weighted_growth
  FROM revenue_growth
  JOIN company ON revenue_growth.company_name = company.company_name
  GROUP BY industry
)
SELECT industry, avg_weighted_growth
FROM avg_weighted_growth ORDER BY avg_weighted_growth DESC;
```

Растущая прибыль компании во времени — это метрика, которая позволяет измерять ежегодный процентный рост прибыли компании в течение определенного периода времени. Она используется для определения темпов роста прибыли, что позволяет оценить финансовое здоровье и стабильность компании (листинг 3.1.3.5).

Листинг 3.1.3.5 - Запрос растущей прибыли компаний во времени

```
WITH yearly_profit AS (
  SELECT company_name, EXTRACT(YEAR FROM modification_date) AS year, profit
  FROM company_finance
), yearly_growth AS (
  SELECT company_name, year, profit, profit/lag(profit) OVER (PARTITION BY company_name ORDER BY
year) - 1 AS yearly_growth
  FROM yearly_profit
), avg_yearly_growth AS (
  SELECT industry, AVG(yearly_growth) as avg_yearly_growth
  FROM yearly_growth
  JOIN company ON yearly_growth.company_name = company.company_name
  GROUP BY industry
)
SELECT industry, avg_yearly_growth
FROM avg_yearly_growth
ORDER BY avg_yearly_growth DESC;
```

Риск компании — это метрика, позволяющая оценить риск, связанный с финансовой устойчивостью компании. Эта метрика вычисляет отношение

вложения в компанию к ее активам, что дает понимание о том, насколько компания может себе позволить рисковать своими деньгами (листинг 3.1.3.6).

Листинг 3.1.3.6 - Запрос риска компаний

```
SELECT company_name, (assets - investments) / assets * 100 AS risk_index
FROM company_finance
ORDER BY risk_index;
```

Доля компаний, которые инвестируют в научно-исследовательскую деятельность — это метрика, которая позволяет оценить долю компаний, которые инвестируют в исследования и технологические разработки. Это может быть показателем осведомленности и готовности компании идти в ногу с инновационными технологиями (листинг 3.1.3.7).

Листинг 3.1.3.7 - Запрос доли компании, инвестируемой в научно-исследовательскую деятельность

```
SELECT COUNT(*) / (SELECT COUNT(*) FROM company_finanse) * 100 AS
percentage_companies_investing_in_rd
FROM company_finance
WHERE investments > 0;
```

Коэффициент эффективности использования имущества — это метрика, которая оценивает эффективность управления компании в увеличении выручки за единицу вложенных активов. Она показывает, насколько успешно компания использует свои активы для генерации дохода (листинг 3.1.3.8).

Листинг 3.1.3.8 - Запрос коэффициента эффективности использования имущества

```
SELECT company_name, revenue/assets AS asset_turnover_ratio
FROM company_finance
ORDER BY asset_turnover_ratio DESC;
```

3.2 Интегрирование программного обеспечения в единый программный комплекс

Для интегрирования программного обеспечения в единый программный комплекс использовался язык программирования Python. С этой целью было создано консольное приложение, которое совмещало в себе сбор информации для анализа, работу с базой данных и возможность отправлять запрос. В приложении А содержится код программного комплекса.

В консольном приложении идет проверка пользователя, с использованием ролей доступа в базе данных, в зависимости от роли дается доступ к определенным функциям.

Для работы с базой данных было принято решение создать функции, которые будут вызывать процедуры.

Была создана функция insert_company() и insert_company_finance(), которая запрашивает информацию о компании и вводит ее в базу данных, используя процедуру insert_company() и insert_company_finance() (листинг 3.2.1).

Листинг 3.2.1 – Код функции ввода данных о компании

```
def insert_company(cur):
    # Запрашиваем у пользователя данные о компании
    company_name = input("Введите название компании:")
    company_description = input("Введите описание компании:")
    industry = input("Введите отрасль:")
    year_established = input("Введите год основания:")
    number_of_employees = int(input("Введите количество сотрудников:"))
    location_country = input("Введите страну:")
    location_city = input("Введите город:")
    address = input("Введите адрес:")
    products_services = input("Введите продукт/услугу:")
    competitors = input("Введите конкурентов:")
    growth_trends_forecasting = input("Введите тенденцию:")

    try:
        # Вызываем хранимую процедуру "insert_company" и передаем ей параметры
        cur.execute('CALL public.insert_company(%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s);', (
            company_name,
            company_description,
            industry,
            year_established,
            number_of_employees,
            location_country,
            location_city,
```

```
            address,
            products_services,
            competitors,
            growth_trends_forecasting
        ))

        # Выводим сообщение об успешном добавлении компании
        print("Компания успешно добавлена в базу данных!")

    except Exception as e:
        # Выводим сообщение об ошибке в случае неудачной попытки добавить компанию в таблицу
        print("Ошибка ввода в таблицу:", e)

def insert_company_finance(cur):
    # Запрашиваем у пользователя данные о компании
    company_name = input("Введите название компании:")
    revenue = int(input("Введите доход:"))
    profit = int(input("Введите прибыль:"))
    assets = int(input("Введите автивы:"))
    investments = int(input("Введите инвестиции:"))
    dividends = int(input("Введите дивиденты:"))

    try:
        # Вызываем хранимую процедуру "insert_company_finance" и передаем ей параметры
        cur.execute('CALL public.insert_company_finance(%s, %s, %s, %s, %s, %s);', (
            company_name,
            revenue,
            profit,
            assets,
            investments,
            dividends
        ))

        # Выводим сообщение об успешном добавлении компании
        print("Компания успешно добавлена в базу данных!")

    except Exception as e:
        # Выводим сообщение об ошибке в случае неудачной попытки добавить компанию в таблицу
        print("Ошибка ввода в таблицу:", e)
```

Была создана функция insert_contact(), которая запрашивает информацию о человеке и вводит ее в базу данных, используя процедуру sp_insert_contact() (листинг 3.2.2).

Листинг 3.2.2 – Код функции ввода данных о человеке

```
def insert_contact(cur):
    # Запрашиваем у пользователя данные о контакте
    first_name = input("Введите имя:")
    last_name = input("Введите фамилию:")
    position = input("Введите должность:")
    company_name = input("Введите компанию:")
    phone_number = input("Введите номер телефона:")
    email = input("Введите электронную почту:")

    try:
        # Вызываем хранимую процедуру "sp_insert_contact" и передаем ей параметры
```

```
        cur.execute("CALL public.sp_insert_contact(%s, %s, %s, %s, %s, %s);", (first_name, last_name,
position,company_name, phone_number, email))
        # Выводим сообщение об успешном добавлении контакта
        print("Контакт успешно добавлен в базу данных!")
    except Exception as e:
        # Выводим сообщение об ошибке в случае неудачной попытки добавить контакт в таблицу
        print("Ошибка ввода в таблицу:", e)
```

Была создана функция insert_country() и insert_country_numeric(), которая запрашивает информацию о стране и вводит ее в базу данных, используя процедуру insert_country() и insert_country_numeric() (листинг 3.2.3).

Листинг 3.2.3 – Код функции ввода данных о стране

```
def insert_country(cur):
    # Запрашиваем у пользователя данные о стране
    country_name = input("Введите название страны:")
    population = int(input("Введите численность населения:"))
    political_system = input("Введите политическая система:")
    legal_system = input("Введите юридическая система:")
    stability = input("Введите стабильность:")
    relations_with_other_countries = input("Введите отношения с другими странами:")
    industry = input("Введите индустрию:")

    try:
        # Вызываем хранимую процедуру "insert_country" и передаем ей параметры
        cur.execute("CALL public.insert_country(%s, %s, %s, %s, %s, %s, %s)",
            ( country_name, population, political_system, legal_system, stability, relations_with_other_countries,
industry)
        )
        # Выводим сообщение об успешном добавлении страны
        print("Страна успешно добавлена в базу данных!")
        conn.commit() # применение изменений в базе данных
    except Exception as e:
        # Выводим сообщение об ошибке в случае неудачной попытки добавить страну в таблицу
        print("Ошибка ввода в таблицу:", e)

def insert_country_numeric(cur):
    # Запрашиваем у пользователя данные о стране
    country_name = input("Введите название страны:")
    gdp = float(input("Введите ВВП:"))
    gdp_growth = float(input("Введите рост ВВП:"))
    inflation = float(input("Введите инфляцию:"))
    unemployment = float(input("Введите безработицу:"))
    investments = float(input("Введите инвестиции:"))

    try:
        # Вызываем хранимую процедуру "insert_country_numeric" и передаем ей параметры
        cur.execute("CALL public.insert_country_numeric(%s, %s, %s, %s, %s, %s)",
            ( country_name, gdp, gdp_growth, inflation, unemployment, investments)
        )
        # Выводим сообщение об успешном добавлении страны
        print("Страна успешно добавлена в базу данных!")
        conn.commit() # применение изменений в базе данных
    except Exception as e:
```

```
        # Выводим сообщение об ошибке в случае неудачной попытки добавить страну в таблицу
        print("Ошибка ввода в таблицу:", e)
```

Была создана функция insert_media(), которая запрашивает информацию о медиа-источнике и вводит ее в базу данных, используя процедуру sp_insert_media() (листинг 3.2.4).

Листинг 3.2.4 – Код функции ввода данных о медиа-источнике

```
def insert_media(cur):
    # Запрашиваем у пользователя данные о медиа-источнике
    name = input("Введите название медиа-источника:")
    url = input("Введите ссылку:")

    try:
        # Вызываем хранимую процедуру "sp_insert_media" и передаем ей параметры
        cur.execute("CALL public.sp_insert_media(%s, %s)", (
            name,
            url
        ))
        # Выводим сообщение об успешном добавлении медиа-источника
        print("Медиа-источник успешно добавлена в базу данных!")
    except Exception as e:
        # Выводим сообщение об ошибке в случае неудачной попытки добавить медиа-источник в таблицу
        print("Ошибка ввода в таблицу:", e)
```

Была создана функция delete_company(), которая запрашивает информацию о компании и удаляет ее в базу данных, используя процедуру sp_delete_company() (листинг 3.2.5).

Листинг 3.2.5 – Код функции удаления данных о компании

```
def delete_company(cur):
    # Запрашиваем у пользователя название компании, которую нужно удалить
    company_name = input("Введите название компании:")

    try:
        # Вызываем хранимую процедуру "sp_delete_company" и передаем ей параметры
        cur.execute("CALL public.sp_delete_company(%s)", (company_name, ))
        # Проверяем, была ли удалена какая-либо запись из таблицы
        if cur.rowcount == 0:
            # Выводим сообщение, если данная компания не найдена в таблице
            print("Компания с таким названием не найдена!")
        else:
            # Если удаление прошло успешно, выводим соответствующее сообщение
            print("Компания успешно удалена из базы данных!")
    except Exception as e:
        # Выводим сообщение об ошибке в случае неудачной попытки удаления компании из таблицы
        print("Ошибка удаления данных в таблице:", e)
```

Была создана функция delete_contact(), которая запрашивает информацию о человеке и удаляет ее в базу данных, используя процедуру sp_delete_contact() (листинг 3.2.6).

Листинг 3.2.6 – Код функции удаления данных о человеке

```
def delete_contact(cur):
    # Запрашиваем у пользователя имя и фамилию контакта, которого нужно удалить
    first_name = input("Введите имя:")
    last_name = input("Введите фамилию:")

    try:
        # Вызываем хранимую процедуру "sp_delete_contact" и передаем ей параметры
        cur.execute("CALL public.sp_delete_contact(%s,%s)", (first_name, last_name, ))
        # Проверяем, была ли удалена какая-либо запись из таблицы
        if cur.rowcount == 0:
            # Выводим сообщение, если контакт не найден в таблице
            print("Контакт с таким именем и фамилией не найден!")
        else:
            # Если удаление прошло успешно, выводим соответствующее сообщение
            print("Контакт успешно удален из базы данных!")
    except Exception as e:
        # Выводим сообщение об ошибке в случае неудачной попытки удаления контакта из таблицы
        print("Ошибка удаления данных в таблице:", e)
```

Была создана функция delete_country(), которая запрашивает информацию о стране и удаляет ее в базу данных, используя процедуру sp_delete_country() (листинг 3.2.7).

Листинг 3.2.7 – Код функции удаления данных о стране

```
def delete_country(cur):
    # Запрашиваем у пользователя название страны, которую нужно удалить
    country_name = input("Введите название страны:")

    try:
        # Вызываем хранимую процедуру "sp_delete_country" и передаем ей параметры
        cur.execute("CALL public.sp_delete_country(%s)", (country_name, ))
        # Проверяем, была ли удалена какая-либо запись из таблицы
        if cur.rowcount == 0:
            # Выводим сообщение, если страна не найдена в таблице
            print("Страна с таким названием не найдена!")
        else:
            # Если удаление прошло успешно, выводим соответствующее сообщение
            print("Страна успешно удалена из базы данных!")
    except Exception as e:
        # Выводим сообщение об ошибке в случае неудачной попытки удаления страны из таблицы
        print("Ошибка удаления данных в таблице:", e)
```

Была создана функция delete_media(), которая запрашивает информацию о медиа-источнике и удаляет ее в базу данных, используя процедуру sp_delete_media() (листинг 3.2.8).

Листинг 3.2.8 – Код функции удаления данных о медиа-источнике

```
def delete_media(cur):
    # Запрашиваем у пользователя название медиа-ресурса, который нужно удалить
    name = input("Введите название медиа-источника:")

    try:
        # Вызываем хранимую процедуру "sp_delete_media" и передаем ей параметры
        cur.execute("CALL public.sp_delete_media(%s)", (name, ))
        # Проверяем, была ли удалена какая-либо запись из таблицы
        if cur.rowcount == 0:
            # Выводим сообщение, если медиа-источник не найден в таблице
            print("Медиа-источник с таким названием не найден!")
        else:
            # Если удаление прошло успешно, выводим соответствующее сообщение
            print("Медиа-источник успешно удален из базы данных!")
    except Exception as e:
        # Выводим сообщение об ошибке в случае неудачной попытки удаления медиа-ресурса из таблицы
        print("Ошибка удаления данных в таблице:", e)
```

С целью более удобного использования базы данных было принято решение создать функцию sql(), которая отправляет SQL-запрос в базу данных и выводит в консоли результат (листинг 3.2.9).

Листинг 3.2.9 – Код функции вызова SQL-запроса

```
def sql(cur):
    # Запрашиваем у пользователя SQL-запрос
    q = input("Введите SQL-запрос:")

    try:
        # Получаем текущие количество ролей в базе данных
        cur.execute("SELECT count(*) FROM pg_roles;")
        roles = cur.fetchone()[0]

        # Получаем текущее количество таблиц в базе данных
        cur.execute("SELECT count(*) FROM information_schema.tables WHERE table_schema = 'public';")
        tables = cur.fetchone()[0]

        # Выполняем SQL-запрос
        cur.execute(q)
        rows = cur.fetchall()

        # Проверяем изменение количества ролей в базе данных
        cur.execute("SELECT count(*) FROM pg_roles;")
        roles_count = cur.fetchone()[0]
        if roles_count > roles:
            cur.close()
            conn.close()
```

```
            raise Exception("Доступ временно заблокирован! Количество ролей в базе данных превысило допустимый предел.")

        # Проверяем изменение количества таблиц в базе данных
        cur.execute("SELECT count(*) FROM information_schema.tables WHERE table_schema = 'public';")
        tables_count = cur.fetchone()[0]
        if tables_count > tables:
            cur.close()
            conn.close()
            raise Exception("Доступ временно заблокирован! Количество таблиц в базе данных превысило допустимый предел.")

        # Выводим результаты запроса
        if cur.rowcount == 0:
            print("Нет данных, удовлетворяющих запросу!")
        else:
            for row in rows:
                print(row)

    except Exception as e:
        # Выводим сообщение об ошибке в случае неудачного выполнения запроса
        print("Ошибка выполнения SQL-запроса:", e)
        # Закрываем соединение с базой данных и завершаем работу программы
        cur.close()
        conn.close()
        exit()
```

Для получения данных были созданы функции вывода информации select_country(), select_company(), select_contact(), select_media(), которые вызывают SQL-запросы (листинг 3.2.10).

Листинг 3.2.10 – Код функции вызова SQL-запроса

```
def select_country(cur):
    # Запрашиваем у пользователя название страны
    name = input("Введите название страны:")

    # Формируем SQL-запрос для выборки информации о стране
    q = f"SELECT l.*, r.* FROM public.country AS l JOIN public.country_numeric AS r ON r.country_name = l.country_name WHERE l.country_name = '{name}';"

    # Выполняем SQL-запрос
    cur.execute(q)

    # Получаем результаты запроса
    res = cur.fetchall()

    # Выводим результаты запроса
    print(res)

def select_company(cur):
    # Запрашиваем у пользователя название компании
    name = input("Введите название компании:")

    # Формируем SQL-запрос для выборки информации о компании
    q = f"SELECT l.*, r.* FROM public.company AS l JOIN public.company_finance AS r ON r.company_name = l.company_name WHERE l.company_name = '{name}';"
```

```
    # Выполняем SQL-запрос
    cur.execute(q)

    # Получаем результаты запроса
    res = cur.fetchall()

    # Выводим результаты запроса
    print(res)

def select_contact(cur):
    # Запрашиваем у пользователя имя и фамилию контакта
    first_name = input("Введите имя:")
    last_name = input("Введите фамилию:")

    # Формируем SQL-запрос для выборки информации о контакте
    q = f"SELECT * FROM public.contact WHERE first_name = '{first_name}' AND last_name ='{last_name}'"

    # Выполняем SQL-запрос
    cur.execute(q)

    # Получаем результаты запроса
    res = cur.fetchall()

    # Выводим результаты запроса
    print(res)

def select_media(cur):
    # Запрашиваем у пользователя название медиа-источника
    name = input("Введите название медиа-источника:")

    # Формируем SQL-запрос для выборки информации о медиа-источнике
    q = f"SELECT * FROM public.media WHERE media_name = '{name}'"

    # Выполняем SQL-запрос
    cur.execute(q)

    # Получаем результаты запроса
    res = cur.fetchall()

    # Выводим результаты запроса
    print(res)
```

Разработанный подход позволяет значительно увеличить эффективность работы системы и упростить взаимодействие между ее компонентами.

3.3 Обеспечение защищенности социотехнической системы

В программном комплексе основной составляющей обеспечения безопасности является разграничение прав, также на основе этой меры

проходит аутентификация в консольном приложении, и резервное копирование базы данных и шифрование дампа.

3.3.1 Разграничение прав

Разграничение прав доступа по сотрудникам:

1) Руководитель отдела деловой разведки имеет полный доступ ко всей информации в базе данных. Он может редактировать, добавлять, удалять и просматривать все данные (листинг 3.3.1.1).

Листинг 3.3.1.1 – SQL-запрос создания роли “Менеджер”

```
CREATE ROLE manager LOGIN PASSWORD '3333';
GRANT ALL PRIVILEGES ON company,company_finance, country,country_numeric, contact TO manager;
```

2) Аналитик имеет доступ к таблицам, содержащим данные по рынку, конкурентам, контактам и финансам. Он может просматривать и анализировать эту информацию (листинг 3.3.1.2).

Листинг 3.3.1.2 – SQL-запрос создания роли “Аналитик”

```
CREATE ROLE analyst LOGIN PASSWORD ' 2222';
GRANT SELECT, INSERT, DELETE ON company,company_finance, country,country_numeric, contact TO
analyst;
```

3) Специалист по маркетингу имеет доступ к таблицам "Компания" и "Страна". Он может просматривать данные, относящиеся к продуктам и услугам, предлагаемым компаниями, а также использовать эту информацию для разработки маркетинговых стратегий (листинг 3.3.1.3).

Листинг 3.3.1.3 – SQL-запрос создания роли “Маркетолог”

```
CREATE ROLE marketer LOGIN PASSWORD '1111';
GRANT SELECT ON company,company_finance, country,country_numeric, media TO marketer;
```

Каждый сотрудник имеет уникальный логин и пароль для доступа к базе данных, для обеспечения безопасности и конфиденциальности информации.

3.3.2 Ограничение доступа к консольному приложению

С целью обеспечения безопасности в консольном приложении было создана функция, которая принимает имя пользователя и пароль, отправляет в базу данных, если подключение успешно, тогда доступ разрешен, иначе отключает от приложения. Это решает проблему доступа к консольному приложению и шифрование паролей, так как пароли хранятся в базе данных в хешированном виде (листинг 3.3.2.1).

Листинг 3.3.2.1 – Код функции проверки доступа

```
def entrance(user, password):
    try:
        # Устанавливаем соединение с базой данных PostgreSQL
        conn = psycopg2.connect(
            database="business_Intelligence",   # Указываем название базы данных
            user=user,   # Указываем имя пользователя, под которым производится подключение
            password=password,  # Указываем пароль пользователя
            host="localhost",   # Указываем хост (обычно это localhost)
            port="5432"   # Указываем порт (обычно это 5432)
        )

        # Создаем курсор для выполнения SQL-запросов
        cursor = conn.cursor()

        # Если подключение удалось, выводим сообщение об успешном подключении
        print("Подключение удалось")

        # Возвращаем объект курсора и объект соединения
        return cursor, conn

    except Exception as e:
        # Если возникла ошибка подключения, выводим сообщение об ошибке
        print("Ошибка подключения")
        print(f"Ошибка: {e}")

        # Возвращаем None для курсора и 0 для соединения, чтобы сигнализировать об ошибке
        return None, 0
```

3.3.3 Резервная копия базы данных и ее шифрование

В случае отказа системы в файловой системе должна храниться резервная копия базы данных, что позволяет восстановить в случае потери данных. Чтобы копия была защищена от кражи необходимо ее зашифровать, это обеспечить сохранность данных (листинг 3.3.3.1).

Листинг 3.3.3.1 – Запрос для создания дампа и его шифрования

```
pg_dump -U postgres -W -Fc business_Intelligence > “путь_к_папке\ business_Intelligence.dump”
#создание дампа
gpg –gen-key
#генерация ключей для шифрования
gpg –symetric “путь_до_дампа\business_Intelligence.dump”
#шифрование дампа и сохранение в той же папке
rm “путь_до_дампа\business_Intelligence.dump”
#удаление незашифрованного дампа
```

4 РАЗРАБОТКА РЕШЕНИЯ ДЛЯ АВТОМАТИЗАЦИИ ПРОЦЕДУР ОБЕСПЕЧЕНИЯ БЕЗОПАСНОСТИ ДЕЛОВОЙ РАЗВЕДКИ

Изменения в базе данных возможны только в при вызове функции sql(), при использовании которой может возникнуть проблема безопасности. Появляется возможность создать новую роль или таблицу, с этой целью в данную функцию была добавлена проверка на количество таблиц и ролей до вызова функции. Если в результате были изменения, то они отменяются и пользователя отключает от базы данных.

В листинге 3.2.9 предоставлена функция, которая позволяет вводить SQL-запрос. Если в запросе злоумышленник пытается создать новую таблицу или роль, в функция было добавлена проверка на изменение количества таблиц и ролей, в негативном случае пользователя отключает из консольного приложения (листинг 4.1). Изменять количество ролей и таблиц возможно при работе в самой базе данных с ролью “Менеджер”.

Листинг 4.1 – Фрагмент кода проверки на изменение количества ролей и таблиц.

```
# Получаем текущие количество ролей в базе данных
    cur.execute("SELECT count(*) FROM pg_roles;")
    roles = cur.fetchone()[0]

    # Получаем текущее количество таблиц в базе данных
    cur.execute("SELECT count(*) FROM information_schema.tables WHERE table_schema = 'public';")
    tables = cur.fetchone()[0]

    # Выполняем SQL-запрос
    cur.execute(q)
    rows = cur.fetchall()

    # Проверяем изменение количества ролей в базе данных
    cur.execute("SELECT count(*) FROM pg_roles;")
    roles_count = cur.fetchone()[0]
    if roles_count > roles:
        cur.close()
        conn.close()
        raise Exception("Доступ временно заблокирован! Количество ролей в базе данных превысило
допустимый предел.")

    # Проверяем изменение количества таблиц в базе данных
    cur.execute("SELECT count(*) FROM information_schema.tables WHERE table_schema = 'public';")
    tables_count = cur.fetchone()[0]
    if tables_count > tables:
        cur.close()
        conn.close()
```

raise Exception("Доступ временно заблокирован! Количество таблиц в базе данных превысило допустимый предел.")

5 РАЗВЕРТЫВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Для развертывания программного обеспечения необходимо реализовать сбор информации, хранение данных. При правильном построение программного комплекса можно проводить аналитику данных и строить гипотезы, которые могут повысить конкурентоспособность компании.

5.1 Реализация сбора информации

Для реализации были сбора информации были созданы парсеры в виде функций, который получали информацию из сайта и выводили новости.

Парсер новостного источника, исследуя новостную ленту, выводит текст, в котором было найдено ключевое слово, введенное сотрудником. Парсер работает на основе доверительных ссылка, хранящихся в базе данных (листинг 5.1.1).

Листинг 5.1.1 – Код парсера новостного источника.

```
def parser_news(words, URL):
    # Разделяем строку слов на список с помощью разделителя ","
    words = words.split(',')
    # Создаем пустой список для статей
    art = []
    # Итерируемся по каждому URL-адресу из списка URL
    for url in URL:
        # Указываем парсер для BeautifulSoup
        parser = 'html.parser'
        # Задаем заголовок для запроса
        headers = {"User-Agent": "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:89.0) Gecko/20100101 Firefox/89.0"}

        try:
            # Отправляем GET-запрос на URL-адрес
            resp = requests.get(url, headers=headers)
        except requests.exceptions.ConnectionError:
            # Если возникает ошибка соединения, выводим сообщение об ошибке и переходим к следующей итерации цикла
            print("Ошибка соединения с URL:", url)
            continue

        # Получаем кодировку ответа от сервера
        http_encoding = resp.encoding if 'charset' in resp.headers.get('content-type', '').lower() else None
        # Получаем кодировку HTML-страницы при помощи кодировщика
```

```
    html_encoding = EncodingDetector.find_declared_encoding(resp.content, is_html=True)
    # Выбираем кодировку, используя закодированную HTML-страницу или кодировку ответа от сервера
    encoding = html_encoding or http_encoding
    # Обрабатываем HTML-страницу при помощи BeautifulSoup
    soup = BeautifulSoup(resp.content, parser, from_encoding=encoding)

    # Ищем все внутренние ссылки на странице
    links = []
    for link in soup.find_all('a', href=True):
        # Пропускаем ссылки на JavaScript
        if "javascript" in link["href"]:
            continue
        links.append(link['href'])

    # Итерируемся по каждой ссылке на странице
    for link in links:
        try:
            # Загружаем статью из ссылки
            article = Article(link)
            # Загружаем статью с помощью методов .download() и .parse()
            article.download()
            article.parse()
            # Ищем каждое заданное слово в тексте статьи
            for word in words:
                if word in article.text:
                    # Если слово найдено, добавляем статью в список статей "art"
                    art.append(article.text)
        except:
            # Если возникает ошибка при загрузке статьи, переходим к следующей итерации цикла
            pass

# Возвращаем список статей, содержащих заданные слова
return art
```

Для получения доверительных медиа-источников была реализована функция, которая выводит список всех ссылок, которые будут использовать в парсере новостей (листинг 5.1.2).

Листинг 5.1.2 – Код функции вывода доверительных ссылок из базы данных

```
def get_media_urls(cursor):
    # Выполняем SQL-запрос для выборки всех URL-адресов из таблицы "media"
    cursor.execute("SELECT url FROM public.media")
    # Получаем все строки результата запроса в виде списка кортежей и извлекаем из них URL-адреса
    urls = [row[0] for row in cursor.fetchall()]
    # Возвращаем список URL-адресов
    return urls
```

5.2 Реализация БД на PostgresSQL

Для реализации хранения данных были созданы таблицы в базе данных. Одними из важных таблиц являются public.company и public.company_finance, которые хранят данные о компании (листинг 5.2.1). В таблице, содержащую основную информацию о компании изменения происходят редко, данная таблица принимает функцию справочника. В таблице, которая содержит информацию о финансовом положении компании, данные изменяются часто, она выполняет важную роль для создания и использования метрик оценки конкурентоспособности компаний.

Листинг 5.2.1 – SQL-запрос создания таблиц о компании

```
CREATE TABLE IF NOT EXISTS public.company_finance
(
    company_name text COLLATE pg_catalog."default" NOT NULL,
    revenue integer,
    profit integer,
    assets integer,
    investments integer,
    dividends integer,
    modification_date date NOT NULL,
    CONSTRAINT company_finance_pkey PRIMARY KEY (company_name, modification_date),
    CONSTRAINT company_finance_company_name_fkey FOREIGN KEY (company_name)
        REFERENCES public.company (company_name) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.company_finance
    OWNER to postgres;
-- Index: company_finance_modification_date_index

-- DROP INDEX IF EXISTS public.company_finance_modification_date_index;

CREATE INDEX IF NOT EXISTS company_finance_modification_date_index
    ON public.company_finance USING btree
    (modification_date ASC NULLS LAST)
    TABLESPACE pg_default;
CREATE TABLE IF NOT EXISTS public.company
(
    company_name text COLLATE pg_catalog."default" NOT NULL,
    company_description text COLLATE pg_catalog."default",
    industry text COLLATE pg_catalog."default",
    year_established text,
    number_of_employees integer,
    location_country text COLLATE pg_catalog."default",
    location_city text COLLATE pg_catalog."default",
    address text COLLATE pg_catalog."default",
    products_services text COLLATE pg_catalog."default",
    competitors text COLLATE pg_catalog."default",
    growth_trends_forecasting text COLLATE pg_catalog."default",
    modification_date date,
```

```
    CONSTRAINT companies_pkey PRIMARY KEY (company_name)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.company
    OWNER to postgres;
-- Index: companies_modification_date_index

-- DROP INDEX IF EXISTS public.companies_modification_date_index;

CREATE INDEX IF NOT EXISTS companies_modification_date_index
    ON public.company USING btree
    (modification_date ASC NULLS LAST)
    TABLESPACE pg_default;
```

Для оценки экономического положения стран были созданы таблицы: public.country и public.country_numeric (листинг 5.2.2). Они хранят справочные данные о стране, а также числовые данные. Таблицу с числовыми данными можно использовать для создания и использования метрик, которые помогут оценить ситуацию в стране.

Листинг 5.2.2 – SQL-запрос создания таблиц о стране

```
CREATE TABLE IF NOT EXISTS public.country
(
    country_name text COLLATE pg_catalog."default" NOT NULL,
    population text COLLATE pg_catalog."default",
    political_system text COLLATE pg_catalog."default",
    legal_system text COLLATE pg_catalog."default",
    stability text COLLATE pg_catalog."default",
    relations_with_other_countries text COLLATE pg_catalog."default",
    industry text COLLATE pg_catalog."default",
    CONSTRAINT country_text_pkey PRIMARY KEY (country_name)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.country
    OWNER to postgres;
CREATE TABLE IF NOT EXISTS public.country_numeric
(
    country_name text COLLATE pg_catalog."default" NOT NULL,
    "GDP" numeric(9,4),
    "GDP_growth" numeric(9,4),
    inflation numeric(9,4),
    unemployment numeric(9,4),
    investments numeric(9,4),
    modification_date date,
    CONSTRAINT country_numeric_pkey PRIMARY KEY (country_name),
    CONSTRAINT country_numeric_country_ FOREIGN KEY (country_name)
        REFERENCES public.country (country_name) MATCH SIMPLE
        ON UPDATE NO ACTION
```

```
    ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.country_numeric
  OWNER to postgres;
```

Для контакта с компаниями была создана таблица, которая хранит данные о сотрудниках компаний - public.contact (листинг 5.2.3). Данная таблица имеет только справочную функцию.

Листинг 5.2.3 – SQL-запрос создания таблицы о человеке

```
CREATE TABLE IF NOT EXISTS public.contact
(
  first_name text COLLATE pg_catalog."default" NOT NULL,
  last_name text COLLATE pg_catalog."default" NOT NULL,
  "position" text COLLATE pg_catalog."default" NOT NULL,
  company_name text COLLATE pg_catalog."default" NOT NULL,
  phone_number text COLLATE pg_catalog."default",
  email text COLLATE pg_catalog."default",
  CONSTRAINT contacts_pkey PRIMARY KEY (first_name, last_name, "position", company_name),
  CONSTRAINT contact_company FOREIGN KEY (company_name)
    REFERENCES public.company (company_name) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
    NOT VALID
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.contact
  OWNER to postgres;
```

Для использования парсера новостей необходима таблица с доверительными источниками - public.media (листинг 5.2.4). Данная таблица хранит название и ссылку на новостной источник.

Листинг 5.2.4 – SQL-запрос создания таблицы о медиа-источниках

```
CREATE TABLE IF NOT EXISTS public.media
(
  media_name text COLLATE pg_catalog."default" NOT NULL,
  url text COLLATE pg_catalog."default"
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.media
```

```
    OWNER to postgres;
```

В таблицах о компании и странах должно сохраняться дата последнего добавления данных, это необходимо для просмотра изменений данных во времени. С этой целью была создана триггерная функция для каждой из таблиц (листинг 5.2.5 – 5.2.6).

Листинг 5.2.5 – SQL-запрос создания функции ввода даты

```
CREATE OR REPLACE FUNCTION public.update_modification_date()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$
BEGIN
    NEW.modification_date = NOW();
    RETURN NEW;
END;
$BODY$;

ALTER FUNCTION public.update_modification_date()
    OWNER TO postgres;
```

Листинг 5.2.6 – SQL-запрос создания триггера ввода даты

```
CREATE TRIGGER tr_update_modification_date_country
    BEFORE INSERT OR UPDATE
    ON public.company
    FOR EACH ROW
    EXECUTE FUNCTION public.update_modification_date();
CREATE TRIGGER tr_update_modification_date_country
    BEFORE INSERT OR UPDATE
    ON public.company_finance
    FOR EACH ROW
    EXECUTE FUNCTION public.update_modification_date();
CREATE TRIGGER tr_update_modification_date_country
    BEFORE INSERT OR UPDATE
    ON public.country
            FOR EACH ROW

    EXECUTE FUNCTION public.update_modification_date();
CREATE TRIGGER tr_update_modification_date_country
    BEFORE INSERT OR UPDATE
    ON public.country_numeric
    FOR EACH ROW
    EXECUTE FUNCTION public.update_modification_date();
```

Для работы с базой данных были реализованы процедуры, которые помогают, не создавая запросы, вводить и удалять данные из таблицы. Процедуры представлены в приложении Б.

6 ПРИЁМОЧНОЕ ТЕСТИРОВАНИЕ

Программный комплекс выполняет ряд важных функций для деловой разведки:

– Сбор информации из доверительных новостных источников.

– Сохранение информации в базу данных.

– Удаление информации в базе данных.

– Отправления SQL-запроса в базу данных для исследования метрик.

Для работы программного комплекса нужно установить PostgresSQL 15 и Python 3.11.

При работе с Python нужно обновить библиотеки до последней версии, также убедиться, что установлены библиотеки: bs4, newspaper, requests, psycopg2.

Характеристики для работы с консольным приложением:

– Процессор: Intel core i5 и лучше, с частотой не менее 3Ггц.

– Оперативная память: не менее 4 Гб ОЗУ.

– Жесткий диск: не менее 64 Гб свободного места на диске.

– Видеокарта: с поддержкой DirectX 9.

– Клавиатура и мышь: стандартные, проводные или беспородные.

– Монитор: любой с разрешением не менее 1024x768.

– Операционная система: любая современная версия, обновленная до последних версий.

Тестирование системы со стороны аналитика, который является сотрудником компании N.

Аналитику выдали логин analyst и пароль 2222 от системы. Сотрудник при входе в систему вводит данные (рисунок 6.1) и получается окно консольного приложения.

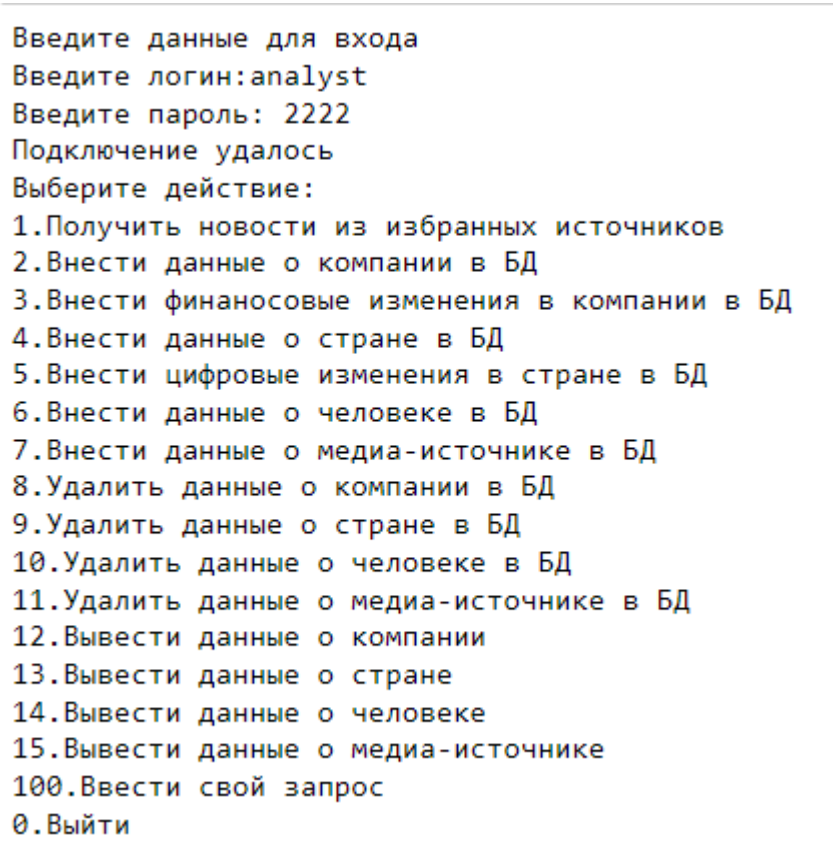

Рисунок 6.1 – Вход в систему.

Для анализа необходимо ввести первичные данные о компании конкуренте, например Роснефть. Необходимо вызвать действие 2 (рисунок 6.2).

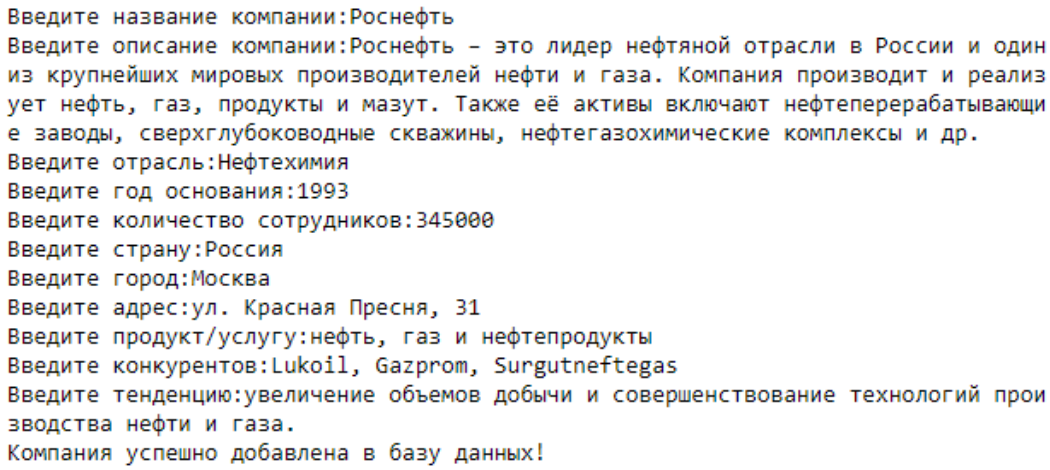

Рисунок 6.2 – Ввод данных о компании

Вызвав действие 3, система запросит ввести финансовую статистику компании (рисунок 6.3).

```
Введите название компании:Роснефть
Введите доход:1000000000
Введите прибыль:500000000
Введите автивы:1500000000
Введите инвестиции:200000000
Введите дивиденты:300000000
Компания успешно добавлена в базу данных!
```

Рисунок 6.3 – Ввода финансовых данных о компании

Вызвав действие 12, можно проверить успешность ввода данных (рисунок 6.4).

```
Введите название компании:Роснефть
[('Роснефть', 'Роснефть – это лидер нефтяной отрасли в России и один из крупней
ших мировых производителей нефти и газа. Компания производит и реализует нефть,
газ, продукты и мазут. Также её активы включают нефтеперерабатывающие заводы, с
верхглубоководные скважины, нефтегазохимические комплексы и др.', 'Нефтехимия',
345000, 'Россия', 'Москва', 'ул. Красная Пресня, 31', 'нефть, газ и нефтепродук
ты', 'Lukoil, Gazprom, Surgutneftegas', 'увеличение объемов добычи и совершенст
вование технологий производства нефти и газа.', datetime.date(2023, 5, 24), '19
93', 'Роснефть', 1000000000, 500000000, 1500000000, 200000000, 300000000, datet
ime.date(2023, 5, 24))]
```

Рисунок 6.4 – Вывод введенных данных

Для ввода данных о стране, необходимо выбрать действие 4 (рисунок 6.5).

```
Введите название страны:Россия
Введите численность населения:144500000
Введите политическая система:Федеральная политическая система
Введите юридическая система:Кодифицированное право
Введите стабильность:Стабильна
Введите отношения с другими странами:Сильные дипломатические связи
Введите индустрию:Нефтегазовая, транспортная, атомная
Страна успешно добавлена в базу данных!
```

Рисунок 6.5 – Ввод данные о стране

Для ввода численных данных о стране, которые можно будет в последствие исследовать, необходимо вызвать действие 5 (рисунок 6.6).

```
Введите название страны:Россия
Введите ВВП:1681000000000
Введите рост ВВП:2.3
Введите инфляцию:3.5
Введите безработицу:5.2
Введите инвестиции:338000000000
Страна успешно добавлена в базу данных!
```

Рисунок 6.6 – Ввод числовой статистике о стране

Для проверки успешности ввода данных о стране нужно вызвать действие 13 (рисунок 6.7).

```
Введите название страны:Россия
[('Россия', '144500000', 'Федеральная политическая система', 'Кодифицированное
право', 'Стабильна', 'Сильные дипломатические связи', 'Нефтегазовая, транспортн
ая, атомная', 'Россия', 1681000000000, Decimal('2.3000'), Decimal('3.5000'), De
cimal('5.2000'), 338000000000, datetime.date(2023, 5, 24))]
```

Рисунок 6.7 – Вывод данных о стране

Для сохранения справочной информации о человеке необходимо воспользоваться действием 6 (рисунок 6.8).

```
Введите имя:Игорь
Введите фамилию:Сечин
Введите должность:Директор
Введите компанию:Роснефть
Введите номер телефона:+7 (495) 777-01-04
Введите электронную почту:rosneft@rosneft.ru
Контакт успешно добавлен в базу данных!
```

Рисунок 6.8 – Ввод данных о человеке

Для проверки успешность ввода данных о человеке необходимо вызвать действие 14 (рисунок 6.9).

```
Введите имя:Игорь
Введите фамилию:Сечин
[('Игорь', 'Сечин', 'Директор', 'Роснефть', '+7 (495) 777-01-04', 'rosneft@rosn
eft.ru')]
```

Рисунок 6.9 – Вывод данных о человеке.

Для использования парсера новостных источников необходимо внести данные о доверительном источнике. Сотруднику необходимо вызвать действие 7 (рисунок 6.10).

```
Введите название медиа-источника:RBC Бизнес
Введите ссылку:https://www.rbc.ru/business/
Медиа-источник успешно добавлена в базу данных!
```

Рисунок 6.10 – Ввод данных о медиа источнике

Чтобы убедиться, что введенные данные сохранились необходимо вызвать действие 15 (рисунок 6.11).

```
Введите название медиа-источника:RBC Бизнес
[('RBC Бизнес', 'https://www.rbc.ru/business/')]
```

Рисунок 6.11 – Вывод данных о медиа-источнике

Основные данные были введены, теперь можно исследовать последние новости о Роснефти в доверительных источниках, воспользовавшись действием 1 (рисунок 6.12).

```
1
Введите ключевые слова:Роснефть
Выберите действие:
```

Рисунок 6.12 – Вывод последних новостей из доверительных источников

Можно заметить, что новостей о Роснефти нет на доверительном источнике, можно расширить количество ключевых слов (рисунок 6.13).

```
0.Выйти
1
Введите ключевые слова:Роснефть,газ,бизнес
Стимулирование бизнеса в целом и глубокой переработки в ключевых отраслях в час
тности – основные пути для переориентации экономики России на новые...
Расскажите, что нового у вашей компании Создавайте публикации регулярно, чтобы
чаще появляться в ленте

Компания «Красные крыши» – один из ведущих дистрибьюторов кровельных, облицовоч
ных материалов и комплектующих в России.

Предоставляем комплексные сервисные услуги и техническое сопровождение установл
енных систем связи и безопасности на весь период эксплуатации.

Получите представительство своей компании на РБК Ведите страницу своей компании
и создавайте публикации

Ведите медиа своего бизнеса на РБК

РБК Компании – это не только управление профилем бизнеса и представительство ко
мпании в деловом СМИ, но и PR-инструмент для создания публикаций компаний, брен
дов и корпораций.
```

Рисунок 6.13 – Вывод последних новостей с увеличенным количеством ключевых слов

Сотрудник имеет возможность ввести SQL-запрос для исследования метрик, например метрики коэффициента рентабельности, вызвав действие 100 (рисунок 6.14).

```
100
Введите SQL-запрос:SELECT company_name,  (CASE WHEN revenue <> 0 THEN profit::n
umeric/revenue ELSE 0 END) * 100 AS profit_margin  FROM company_finance  ORDER
BY profit_margin DESC;
('Роснефть', Decimal('50.0000000000000000000'))
```

Рисунок 6.14 – Ввод SQL-запроса для расчета коэффициента рентабельности

В случае, если необходимо удалить данные о компании, необходимо воспользоваться действиями 8-11 (рисунки 6.14-6.17).

```
8
Введите название компании:Роснефть
Компания успешно удалена из базы данных!
```

Рисунок 6.14 – Удаление данных о компании

```
9
Введите название страны:Россия
Страна успешно удалена из базы данных!
```

Рисунок 6.15 – Удаление данных о стране

```
10
Введите имя:Игорь
Введите фамилию:Сечин
Контакт успешно удален из базы данных!
```

Рисунок 6.16 – Удаление данных о человеке

```
11
Введите название медиа-источника:RBC Бизнес
Медиа-источник успешно удален из базы данных!
```

Рисунок 6.17 – Удаление данных о медиа-источнике

Тестирование системы со стороны менеджера не обязательно, так как менеджер имеет такие же возможности как аналитик. Тестирование со стороны маркетолога не обязательно, так как сотруднику разрешено только смотреть данные, т.е. действия вывода данных и получения новостей.

7 СПИСОК ЛИТЕРАТУРЫ

1) Ющук Е. Бизнес-разведка: законные методы и запрещенные приемы [Электронный ресурс] // Экономические преступления. – URL: https://www.auditit.ru/articles/finance/a106/189897.html (дата обращения 25.08.2023)

2) А.И. Доронин. Бизнес-разведка, 5-е издание – Москва – 2009 [Электронный ресурс] – URL: https://studfile.net/preview/5288609/page:2/ (дата обращения 25.08.2023)

3) Степанов Д. А., Крючков В. Н. Бизнес-разведка как информационная основа стратегического маркетинга // Вестник Омского университета. – Серия «Экономика». – № 3. – 2009. [Электронный ресурс] – URL: https://cyberleninka.ru/article/n/biznes-razvedka-kak-informatsionnaya-osnova-strategicheskogo-marketinga (дата обращения 25.08.2023)

4) РАЗВЕДЫВАТЕЛЬНЫЕ СВЕДЕНИЯ И ИНФОРМАЦИЯ- 2020-18_BIN_06-01_text.pdf [Электронный ресурс] – URL: https://online-edu.ranepa.ru/pluginfile.php/38731/mod_resource/content/5/2020-18_BIN_06-01_text.pdf (дата обращения 25.08.2023)

5) Евгений Ющук. Интернет-разведка. Руководство к действию [Электронный ресурс] – URL: https://portal.edu.asu.ru/pluginfile.php/148833/mod_resource/content/1/Ющук%20Интернет%20разветка.pdf (дата обращения 25.08.2023)

6) Навыки OSINT(интернет-разведки) в кибербезопасности [Электронный ресурс] – URL: https://proglib.io/p/navyki-osint-internet-razvedki-v-kiberbezopasnosti-2020-11-14 (дата обращения 25.08.2023)

7) Почему стоит научиться «парсить» сайты, или как написать свой первый парсер на Python / Хабр [Электронный ресурс] – URL: https://habr.com/ru/articles/568334/ (дата обращения 25.08.2023)

8) Разработка базы данных для современной организации [Электронный ресурс] – URL: https://cyberleninka.ru/article/n/razrabotka-bazy-dannyh-dlya-sovremennoy-organizatsii/viewer (дата обращения 25.08.2023)

9) 70+ метрик для интернет-маркетолога, которые помогут следить за эффективностью продвижения проекта / Хабр [Электронный ресурс] – URL: https://habr.com/ru/articles/675722/ (дата обращения 25.08.2023)

10) Функциональные и нефункциональные требования: полное руководство [Электронный ресурс] – URL: https://bestprogrammer.ru/izuchenie/funktsionalnye-i-nefunktsionalnye-trebovaniya-polnoe-rukovodstvo (дата обращения 25.08.2023)

11) Как писать функциональные требования / Хабр [Электронный ресурс] – URL: https://habr.com/ru/companies/retailrocket/articles/431572/ (дата обращения 25.08.2023)

12) Статья про НФТ [Электронный ресурс] – URL: https://schoolforanalyst.ru/aboutnonfuncreq (дата обращения 25.08.2023)

13) User Story Mapping как подход к проектированию / Хабр [Электронный ресурс] – URL: https://habr.com/ru/companies/akbarsdigital/articles/699950/ (дата обращения 25.08.2023)

14) Закон РФ "О средствах массовой информации" (СМИ) от 27.12.1991 N 2124-1 (последняя редакция) \ КонсультантПлюс [Электронный ресурс] – URL: https://www.consultant.ru/document/cons_doc_LAW_1511/ (дата обращения 25.08.2023)

15) Закон РФ "О государственной тайне" от 21.07.1993 N 5485-1 (последняя редакция) \ КонсультантПлюс [Электронный ресурс] – URL: https://www.consultant.ru/document/cons_doc_LAW_2481/ (дата обращения 25.08.2023)

16) Федеральный закон "Об информации, информационных технологиях и о защите информации" от 27.07.2006 N 149-ФЗ (последняя редакция) \ КонсультантПлюс [Электронный ресурс] – URL:

https://www.consultant.ru/document/cons_doc_LAW_61798/ (дата обращения 25.08.2023)

17) Федеральный закон "О кредитных историях" от 30.12.2004 N 218-ФЗ (последняя редакция) \ КонсультантПлюс [Электронный ресурс] – URL: https://www.consultant.ru/document/cons_doc_LAW_51043/ (дата обращения 25.08.2023)

18) Федеральный закон "О коммерческой тайне" от 29.07.2004 N 98-ФЗ (последняя редакция) \ КонсультантПлюс [Электронный ресурс] – URL: https://www.consultant.ru/document/cons_doc_LAW_48699/ (дата обращения 25.08.2023)

19) OSINT (Open Source INTelligence) - что это за инструменты поиска [Электронный ресурс] - URL: https://blog.skillfactory.ru/glossary/osint/ (дата обращения 26.05.2023)

ПРИЛОЖЕНИЕ А

Код программного обеспечения.

```
from bs4 import BeautifulSoup  # для парсинга HTML и XML
from bs4.dammit import EncodingDetector  # для определения кодировки документа
from newspaper import Article  # для извлечения новостных статей
import requests  # для отправки HTTP-запросов
import psycopg2  # для работы с базами данных PostgreSQL

def entrance(user, password):
    try:
        # Устанавливаем соединение с базой данных PostgreSQL
        conn = psycopg2.connect(
            database="business_Intelligence",   # Указываем название базы данных
            user=user,   # Указываем имя пользователя, под которым производится подключение
            password=password,  # Указываем пароль пользователя
            host="localhost",   # Указываем хост (обычно это localhost)
            port="5432"   # Указываем порт (обычно это 5432)
        )

        # Создаем курсор для выполнения SQL-запросов
        cursor = conn.cursor()

        # Если подключение удалось, выводим сообщение об успешном подключении
        print("Подключение удалось")

        # Возвращаем объект курсора и объект соединения
        return cursor, conn

    except Exception as e:
        # Если возникла ошибка подключения, выводим сообщение об ошибке
        print("Ошибка подключения")
        print(f"Ошибка: {e}")

        # Возвращаем None для курсора и 0 для соединения, чтобы сигнализировать об ошибке
        return None, 0

def parser_news(words, URL):
    # Разделяем строку слов на список с помощью разделителя ","
    words = words.split(',')
    # Создаем пустой список для статей
    art = []
    # Итерируемся по каждому URL-адресу из списка URL
    for url in URL:
        # Указываем парсер для BeautifulSoup
        parser = 'html.parser'
        # Задаем заголовок для запроса
        headers = {"User-Agent": "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:89.0) Gecko/20100101 Firefox/89.0"}

        try:
            # Отправляем GET-запрос на URL-адрес
            resp = requests.get(url, headers=headers)
        except requests.exceptions.ConnectionError:
            # Если возникает ошибка соединения, выводим сообщение об ошибке и переходим к следующей
итерации цикла
            print("Ошибка соединения с URL:", url)
            continue

        # Получаем кодировку ответа от сервера
        http_encoding = resp.encoding if 'charset' in resp.headers.get('content-type', '').lower() else None
```

```
            # Получаем кодировку HTML-страницы при помощи кодировщика
            html_encoding = EncodingDetector.find_declared_encoding(resp.content, is_html=True)
            # Выбираем кодировку, используя закодированную HTML-страницу или кодировку ответа от сервера
            encoding = html_encoding or http_encoding
            # Обрабатываем HTML-страницу при помощи BeautifulSoup
            soup = BeautifulSoup(resp.content, parser, from_encoding=encoding)

            # Ищем все внутренние ссылки на странице
            links = []
            for link in soup.find_all('a', href=True):
                # Пропускаем ссылки на JavaScript
                if "javascript" in link["href"]:
                    continue
                links.append(link['href'])

            # Итерируемся по каждой ссылке на странице
            for link in links:
                try:
                    # Загружаем статью из ссылки
                    article = Article(link)
                    # Загружаем статью с помощью методов .download() и .parse()
                    article.download()
                    article.parse()
                    # Ищем каждое заданное слово в тексте статьи
                    for word in words:
                        if word in article.text:
                            # Если слово найдено, добавляем статью в список статей "art"
                            art.append(article.text)
                except:
                    # Если возникает ошибка при загрузке статьи, переходим к следующей итерации цикла
                    pass

    # Возвращаем список статей, содержащих заданные слова
    return art

def get_media_urls(cursor):
    # Выполняем SQL-запрос для выборки всех URL-адресов из таблицы "media"
    cursor.execute("SELECT url FROM public.media")
    # Получаем все строки результата запроса в виде списка кортежей и извлекаем из них URL-адреса
    urls = [row[0] for row in cursor.fetchall()]
    # Возвращаем список URL-адресов
    return urls

def insert_company(cur):
    # Запрашиваем у пользователя данные о компании
    company_name = input("Введите название компании:")
    company_description = input("Введите описание компании:")
    industry = input("Введите отрасль:")
    year_established = input("Введите год основания:")
    number_of_employees = int(input("Введите количество сотрудников:"))
    location_country = input("Введите страну:")
    location_city = input("Введите город:")
    address = input("Введите адрес:")
    products_services = input("Введите продукт/услугу:")
    competitors = input("Введите конкурентов:")
    growth_trends_forecasting = input("Введите тенденцию:")

    try:
        # Вызываем хранимую процедуру "insert_company" и передаем ей параметры
        cur.execute('CALL public.insert_company(%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s);', (
            company_name,
            company_description,
            industry,
```

```
            year_established,
            number_of_employees,
            location_country,
            location_city,
            address,
            products_services,
            competitors,
            growth_trends_forecasting
        ))

        # Выводим сообщение об успешном добавлении компании
        print("Компания успешно добавлена в базу данных!")

    except Exception as e:
        # Выводим сообщение об ошибке в случае неудачной попытки добавить компанию в таблицу
        print("Ошибка ввода в таблицу:", e)

def insert_company_finance(cur):
    # Запрашиваем у пользователя данные о компании
    company_name = input("Введите название компании:")
    revenue = int(input("Введите доход:"))
    profit = int(input("Введите прибыль:"))
    assets = int(input("Введите автивы:"))
    investments = int(input("Введите инвестиции:"))
    dividends = int(input("Введите дивиденты:"))

    try:
        # Вызываем хранимую процедуру "insert_company_finance" и передаем ей параметры
        cur.execute('CALL public.insert_company_finance(%s, %s, %s, %s, %s, %s);', (
            company_name,
            revenue,
            profit,
            assets,
            investments,
            dividends
        ))

        # Выводим сообщение об успешном добавлении компании
        print("Компания успешно добавлена в базу данных!")

    except Exception as e:
        # Выводим сообщение об ошибке в случае неудачной попытки добавить компанию в таблицу
        print("Ошибка ввода в таблицу:", e)

def insert_contact(cur):
    # Запрашиваем у пользователя данные о контакте
    first_name = input("Введите имя:")
    last_name = input("Введите фамилию:")
    position = input("Введите должность:")
    company_name = input("Введите компанию:")
    phone_number = input("Введите номер телефона:")
    email = input("Введите электронную почту:")

    try:
        # Вызываем хранимую процедуру "sp_insert_contact" и передаем ей параметры
        cur.execute("CALL public.sp_insert_contact(%s, %s, %s, %s, %s, %s);", (first_name, last_name,
position,company_name, phone_number, email))
        # Выводим сообщение об успешном добавлении контакта
        print("Контакт успешно добавлен в базу данных!")
    except Exception as e:
        # Выводим сообщение об ошибке в случае неудачной попытки добавить контакт в таблицу
        print("Ошибка ввода в таблицу:", e)
```

```
def insert_country(cur):
    # Запрашиваем у пользователя данные о стране
    country_name = input("Введите название страны:")
    population = int(input("Введите численность населения:"))
    political_system = input("Введите политическая система:")
    legal_system = input("Введите юридическая система:")
    stability = input("Введите стабильность:")
    relations_with_other_countries = input("Введите отношения с другими странами:")
    industry = input("Введите индустрию:")

    try:
        # Вызываем хранимую процедуру "insert_country" и передаем ей параметры
        cur.execute("CALL public.insert_country(%s, %s, %s, %s, %s, %s, %s)",
            ( country_name, population, political_system, legal_system, stability, relations_with_other_countries,
industry)
        )
        # Выводим сообщение об успешном добавлении страны
        print("Страна успешно добавлена в базу данных!")
        conn.commit() # применение изменений в базе данных
    except Exception as e:
        # Выводим сообщение об ошибке в случае неудачной попытки добавить страну в таблицу
        print("Ошибка ввода в таблицу:", e)

def insert_country_numeric(cur):
    # Запрашиваем у пользователя данные о стране
    country_name = input("Введите название страны:")
    gdp = float(input("Введите ВВП:"))
    gdp_growth = float(input("Введите рост ВВП:"))
    inflation = float(input("Введите инфляцию:"))
    unemployment = float(input("Введите безработицу:"))
    investments = float(input("Введите инвестиции:"))

    try:
        # Вызываем хранимую процедуру "insert_country_numeric" и передаем ей параметры
        cur.execute("CALL public.insert_country_numeric(%s, %s, %s, %s, %s, %s)",
            ( country_name, gdp, gdp_growth, inflation, unemployment, investments)
        )
        # Выводим сообщение об успешном добавлении страны
        print("Страна успешно добавлена в базу данных!")
        conn.commit() # применение изменений в базе данных
    except Exception as e:
        # Выводим сообщение об ошибке в случае неудачной попытки добавить страну в таблицу
        print("Ошибка ввода в таблицу:", e)

def insert_media(cur):
    # Запрашиваем у пользователя данные о медиа-источнике
    name = input("Введите название медиа-источника:")
    url = input("Введите ссылку:")

    try:
        # Вызываем хранимую процедуру "sp_insert_media" и передаем ей параметры
        cur.execute("CALL public.sp_insert_media(%s, %s)", (
            name,
            url
        ))
        # Выводим сообщение об успешном добавлении медиа-источника
        print("Медиа-источник успешно добавлена в базу данных!")
    except Exception as e:
        # Выводим сообщение об ошибке в случае неудачной попытки добавить медиа-источник в таблицу
        print("Ошибка ввода в таблицу:", e)

def delete_company(cur):
    # Запрашиваем у пользователя название компании, которую нужно удалить
```

```
    company_name = input("Введите название компании:")

    try:
        # Вызываем хранимую процедуру "sp_delete_company" и передаем ей параметры
        cur.execute("CALL public.sp_delete_company(%s)", (company_name, ))
        # Проверяем, была ли удалена какая-либо запись из таблицы
        if cur.rowcount == 0:
            # Выводим сообщение, если данная компания не найдена в таблице
            print("Компания с таким названием не найдена!")
        else:
            # Если удаление прошло успешно, выводим соответствующее сообщение
            print("Компания успешно удалена из базы данных!")
    except Exception as e:
        # Выводим сообщение об ошибке в случае неудачной попытки удаления компании из таблицы
        print("Ошибка удаления данных в таблице:", e)

def delete_contact(cur):
    # Запрашиваем у пользователя имя и фамилию контакта, которого нужно удалить
    first_name = input("Введите имя:")
    last_name = input("Введите фамилию:")

    try:
        # Вызываем хранимую процедуру "sp_delete_contact" и передаем ей параметры
        cur.execute("CALL public.sp_delete_contact(%s,%s)", (first_name, last_name, ))
        # Проверяем, была ли удалена какая-либо запись из таблицы
        if cur.rowcount == 0:
            # Выводим сообщение, если контакт не найден в таблице
            print("Контакт с таким именем и фамилией не найден!")
        else:
            # Если удаление прошло успешно, выводим соответствующее сообщение
            print("Контакт успешно удален из базы данных!")
    except Exception as e:
        # Выводим сообщение об ошибке в случае неудачной попытки удаления контакта из таблицы
        print("Ошибка удаления данных в таблице:", e)

def delete_country(cur):
    # Запрашиваем у пользователя название страны, которую нужно удалить
    country_name = input("Введите название страны:")

    try:
        # Вызываем хранимую процедуру "sp_delete_country" и передаем ей параметры
        cur.execute("CALL public.sp_delete_country(%s)", (country_name, ))
        # Проверяем, была ли удалена какая-либо запись из таблицы
        if cur.rowcount == 0:
            # Выводим сообщение, если страна не найдена в таблице
            print("Страна с таким названием не найдена!")
        else:
            # Если удаление прошло успешно, выводим соответствующее сообщение
            print("Страна успешно удалена из базы данных!")
    except Exception as e:
        # Выводим сообщение об ошибке в случае неудачной попытки удаления страны из таблицы
        print("Ошибка удаления данных в таблице:", e)

def delete_media(cur):
    # Запрашиваем у пользователя название медиа-ресурса, который нужно удалить
    name = input("Введите название медиа-источника:")

    try:
        # Вызываем хранимую процедуру "sp_delete_media" и передаем ей параметры
        cur.execute("CALL public.sp_delete_media(%s)", (name, ))
        # Проверяем, была ли удалена какая-либо запись из таблицы
        if cur.rowcount == 0:
            # Выводим сообщение, если медиа-источник не найден в таблице
```

```
            print("Медиа-источник с таким названием не найден!")
        else:
            # Если удаление прошло успешно, выводим соответствующее сообщение
            print("Медиа-источник успешно удален из базы данных!")
    except Exception as e:
        # Выводим сообщение об ошибке в случае неудачной попытки удаления медиа-ресурса из таблицы
        print("Ошибка удаления данных в таблице:", e)

def sql(cur):
    # Запрашиваем у пользователя SQL-запрос
    q = input("Введите SQL-запрос:")

    try:
        # Получаем текущие количество ролей в базе данных
        cur.execute("SELECT count(*) FROM pg_roles;")
        roles = cur.fetchone()[0]

        # Получаем текущее количество таблиц в базе данных
        cur.execute("SELECT count(*) FROM information_schema.tables WHERE table_schema = 'public';")
        tables = cur.fetchone()[0]

        # Выполняем SQL-запрос
        cur.execute(q)
        rows = cur.fetchall()

        # Проверяем изменение количества ролей в базе данных
        cur.execute("SELECT count(*) FROM pg_roles;")
        roles_count = cur.fetchone()[0]
        if roles_count > roles:
            cur.close()
            conn.close()
            raise Exception("Доступ временно заблокирован! Количество ролей в базе данных превысило
допустимый предел.")

        # Проверяем изменение количества таблиц в базе данных
        cur.execute("SELECT count(*) FROM information_schema.tables WHERE table_schema = 'public';")
        tables_count = cur.fetchone()[0]
        if tables_count > tables:
            cur.close()
            conn.close()
            raise Exception("Доступ временно заблокирован! Количество таблиц в базе данных превысило
допустимый предел.")

        # Выводим результаты запроса
        if cur.rowcount == 0:
            print("Нет данных, удовлетворяющих запросу!")
        else:
            for row in rows:
                print(row)

    except Exception as e:
        # Выводим сообщение об ошибке в случае неудачного выполнения запроса
        print("Ошибка выполнения SQL-запроса:", e)
        # Закрываем соединение с базой данных и завершаем работу программы
        cur.close()
        conn.close()
        exit()

def select_country(cur):
    # Запрашиваем у пользователя название страны
    name = input("Введите название страны:")

    # Формируем SQL-запрос для выборки информации о стране
```

```
    q = f"SELECT l.*, r.* FROM public.country AS l JOIN public.country_numeric AS r ON r.country_name =
l.country_name WHERE l.country_name = '{name}';"

    # Выполняем SQL-запрос
    cur.execute(q)

    # Получаем результаты запроса
    res = cur.fetchall()

    # Выводим результаты запроса
    print(res)

def select_company(cur):
    # Запрашиваем у пользователя название компании
    name = input("Введите название компании:")

    # Формируем SQL-запрос для выборки информации о компании
    q = f"SELECT l.*, r.* FROM public.company AS l JOIN public.company_finance AS r ON r.company_name =
l.company_name WHERE l.company_name = '{name}';"

    # Выполняем SQL-запрос
    cur.execute(q)

    # Получаем результаты запроса
    res = cur.fetchall()

    # Выводим результаты запроса
    print(res)

def select_contact(cur):
    # Запрашиваем у пользователя имя и фамилию контакта
    first_name = input("Введите имя:")
    last_name = input("Введите фамилию:")

    # Формируем SQL-запрос для выборки информации о контакте
    q = f"SELECT * FROM public.contact WHERE first_name = '{first_name}' AND last_name ='{last_name}'"

    # Выполняем SQL-запрос
    cur.execute(q)

    # Получаем результаты запроса
    res = cur.fetchall()

    # Выводим результаты запроса
    print(res)

def select_media(cur):
    # Запрашиваем у пользователя название медиа-источника
    name = input("Введите название медиа-источника:")

    # Формируем SQL-запрос для выборки информации о медиа-источнике
    q = f"SELECT * FROM public.media WHERE media_name = '{name}'"

    # Выполняем SQL-запрос
    cur.execute(q)

    # Получаем результаты запроса
    res = cur.fetchall()

    # Выводим результаты запроса
    print(res)

while True:
```

```
    # Флаг для контроля успешности выполнения входа в систему
    tr = True
    # Запрашиваем данные для входа
    print("Введите данные для входа")
    user = input("Введите логин:")
    password = input("Введите пароль:")

    # Выполняем вход в систему
    cursor, conn = entrance(user, password)
    # Если вход не выполнен, устанавливаем флаг в False
    if cursor == None:
        tr = False

    # Запускаем цикл для выполнения действий в выбранной роли пользователя
while True:
    # Флаг для контроля успешности выполнения входа в систему
    tr = True
    # Запрашиваем данные для входа
    print("Введите данные для входа")
    user = input("Введите логин:")
    password = input("Введите пароль:")

    # Выполняем вход в систему
    cursor, conn = entrance(user, password)
    # Если вход не выполнен, устанавливаем флаг в False
    if cursor == None:
        tr = False

    # Запускаем цикл для выполнения действий в выбранной роли пользователя
    while tr:
        if user == 'marketer':
            # Выводим меню для менеджера
            print("Выберите действие:")
            print("1.Получить новости из избранных источников")
            print("2.Вывести данные о компании")
            print("3.Вывести данные о стране")
            print("4.Вывести данные о медиа-источнике")
            print("100.Ввести свой запрос")

            # Ждем ввода действия
            inp = int(input())

            # Обрабатываем ввод пользователя
            if inp == 0:
                break
            elif inp == 1:
                news = parser_news(words=input("Введите ключевые слова:"),URL=get_media_urls(cursor=cursor))
                for n in news:
                    print(n)
            elif inp == 2:
                select_company(cur=cursor)
            elif inp == 3:
                select_country(cur=cursor)
            elif inp == 4:
                select_media(cur=cursor)
            elif inp == 100:
                sql(cur=cursor)

        if user == 'analyst' or user == 'manager' or user == 'postgres':
            # Выводим меню для аналитика или администратора
            print("Выберите действие:")
            print("1.Получить новости из избранных источников")
            print("2.Внести данные о компании в БД")
```

```
        print("3.Внести финаносовые изменения в компании в БД")
        print("4.Внести данные о стране в БД")
        print("5.Внести цифровые изменения в стране в БД")
        print("6.Внести данные о человеке в БД")
        print("7.Внести данные о медиа-источнике в БД")
        print("8.Удалить данные о компании в БД")
        print("9.Удалить данные о стране в БД")
        print("10.Удалить данные о человеке в БД")
        print("11.Удалить данные о медиа-источнике в БД")
        print("12.Вывести данные о компании")
        print("13.Вывести данные о стране")
        print("14.Вывести данные о человеке")
        print("15.Вывести данные о медиа-источнике")
        print("100.Ввести свой запрос")
        print("0.Выйти")

        # Ждем ввода действия
        inp = int(input())

        # Обрабатываем ввод пользователя
        if inp == 0:
            break
        elif inp == 1:
            news = parser_news(words=input("Введите ключевые слова:"),URL=get_media_urls(cursor=cursor))
            for n in news:
                print(n)
        elif inp == 2:
            insert_company(cur=cursor)
        elif inp == 3:
            insert_company_finance(cur=cursor)
        elif inp == 4:
            insert_country(cur=cursor)
        elif inp == 5:
            insert_country_numeric(cur=cursor)
        elif inp == 6:
            insert_contact(cur=cursor)
        elif inp == 7:
            insert_media(cur=cursor)
        elif inp == 8:
            delete_company(cur=cursor)
        elif inp == 9:
            delete_country(cur=cursor)
        elif inp == 10:
            delete_contact(cur=cursor)
        elif inp == 11:
            delete_media(cur=cursor)
        elif inp == 12:
            select_company(cur=cursor)
        elif inp == 13:
            select_country(cur=cursor)
        elif inp == 14:
            select_contact(cur=cursor)
        elif inp == 15:
            select_media(cur=cursor)
        elif inp == 100:
            sql(cur=cursor)
        else:
            print("Ошибка ввода")

    # Фиксируем изменения в БД
    conn.commit()

# Закрываем соединение и курсор
```

```
if cursor == None:
    break
elif inp == 0:
    cursor.close()
    conn.close()
    break
```

ПРИЛОЖЕНИЕ Б

Код SQL-запросов создания базы данных

```
CREATE OR REPLACE FUNCTION public.update_modification_date()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$
BEGIN
    NEW.modification_date = NOW();
    RETURN NEW;
END;
$BODY$;

ALTER FUNCTION public.update_modification_date()
    OWNER TO postgres;

CREATE TABLE IF NOT EXISTS public.company
(
    company_name text COLLATE pg_catalog."default" NOT NULL,
    company_description text COLLATE pg_catalog."default",
    industry text COLLATE pg_catalog."default",
    number_of_employees integer,
    location_country text COLLATE pg_catalog."default",
    location_city text COLLATE pg_catalog."default",
    address text COLLATE pg_catalog."default",
    products_services text COLLATE pg_catalog."default",
    competitors text COLLATE pg_catalog."default",
    growth_trends_forecasting text COLLATE pg_catalog."default",
    modification_date date,
    year_established text COLLATE pg_catalog."default",
    CONSTRAINT companies_pkey PRIMARY KEY (company_name)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.company
    OWNER to postgres;

GRANT DELETE, SELECT, INSERT ON TABLE public.company TO analyst;

GRANT ALL ON TABLE public.company TO manager;

GRANT SELECT ON TABLE public.company TO marketer;

GRANT ALL ON TABLE public.company TO postgres;
-- Index: companies_modification_date_index

-- DROP INDEX IF EXISTS public.companies_modification_date_index;

CREATE INDEX IF NOT EXISTS companies_modification_date_index
    ON public.company USING btree
    (modification_date ASC NULLS LAST)
    TABLESPACE pg_default;

-- Trigger: tr_update_modification_date_country

-- DROP TRIGGER IF EXISTS tr_update_modification_date_country ON public.company;
```

```
CREATE TRIGGER tr_update_modification_date_country
    BEFORE INSERT OR UPDATE
    ON public.company
    FOR EACH ROW
    EXECUTE FUNCTION public.update_modification_date();

CREATE TABLE IF NOT EXISTS public.company_finance
(
    company_name text COLLATE pg_catalog."default" NOT NULL,
    revenue bigint,
    profit bigint,
    assets bigint,
    investments bigint,
    dividends bigint,
    modification_date date NOT NULL,
    CONSTRAINT company_finance_pkey PRIMARY KEY (company_name, modification_date),
    CONSTRAINT company_finance_company_name_fkey FOREIGN KEY (company_name)
        REFERENCES public.company (company_name) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.company_finance
    OWNER to postgres;

GRANT DELETE, SELECT, INSERT ON TABLE public.company_finance TO analyst;

GRANT ALL ON TABLE public.company_finance TO manager;

GRANT SELECT ON TABLE public.company_finance TO marketer;

GRANT ALL ON TABLE public.company_finance TO postgres;
-- Index: company_finance_modification_date_index

-- DROP INDEX IF EXISTS public.company_finance_modification_date_index;

CREATE INDEX IF NOT EXISTS company_finance_modification_date_index
    ON public.company_finance USING btree
    (modification_date ASC NULLS LAST)
    TABLESPACE pg_default;

-- Trigger: tr_update_modification_date_country

-- DROP TRIGGER IF EXISTS tr_update_modification_date_country ON public.company_finance;

CREATE TRIGGER tr_update_modification_date_country
    BEFORE INSERT OR UPDATE
    ON public.company_finance
    FOR EACH ROW
    EXECUTE FUNCTION public.update_modification_date();

CREATE TABLE IF NOT EXISTS public.contact
(
    first_name text COLLATE pg_catalog."default" NOT NULL,
    last_name text COLLATE pg_catalog."default" NOT NULL,
    "position" text COLLATE pg_catalog."default" NOT NULL,
    phone_number text COLLATE pg_catalog."default",
    email text COLLATE pg_catalog."default",
    company_name text COLLATE pg_catalog."default",
    CONSTRAINT contact_company FOREIGN KEY (company_name)
        REFERENCES public.company (company_name) MATCH SIMPLE
```

```
        ON UPDATE NO ACTION
        ON DELETE SET NULL
        NOT VALID
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.contact
    OWNER to postgres;

GRANT INSERT, SELECT, DELETE ON TABLE public.contact TO analyst;

GRANT ALL ON TABLE public.contact TO manager;

GRANT ALL ON TABLE public.contact TO postgres;

CREATE TABLE IF NOT EXISTS public.country
(
    country_name text COLLATE pg_catalog."default" NOT NULL,
    population text COLLATE pg_catalog."default",
    political_system text COLLATE pg_catalog."default",
    legal_system text COLLATE pg_catalog."default",
    stability text COLLATE pg_catalog."default",
    relations_with_other_countries text COLLATE pg_catalog."default",
    industry text COLLATE pg_catalog."default",
    CONSTRAINT country_text_pkey PRIMARY KEY (country_name)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.country
    OWNER to postgres;

GRANT DELETE, SELECT, INSERT ON TABLE public.country TO analyst;

GRANT ALL ON TABLE public.country TO manager;

GRANT SELECT ON TABLE public.country TO marketer;

GRANT ALL ON TABLE public.country TO postgres;

-- Trigger: tr_update_modification_date_country

-- DROP TRIGGER IF EXISTS tr_update_modification_date_country ON public.country;

CREATE TRIGGER tr_update_modification_date_country
    BEFORE INSERT OR UPDATE
    ON public.country
    FOR EACH ROW
    EXECUTE FUNCTION public.update_modification_date();

CREATE TABLE IF NOT EXISTS public.country_numeric
(
    country_name text COLLATE pg_catalog."default" NOT NULL,
    gpg bigint,
    gpg_growth numeric(9,4),
    inflation numeric(9,4),
    unemployment numeric(9,4),
    investments bigint,
    modification_date date,
    CONSTRAINT country_numeric_pkey PRIMARY KEY (country_name),
    CONSTRAINT country_numeric_country_ FOREIGN KEY (country_name)
        REFERENCES public.country (country_name) MATCH SIMPLE
```

```
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.country_numeric
    OWNER to postgres;

GRANT DELETE, SELECT, INSERT ON TABLE public.country_numeric TO analyst;

GRANT ALL ON TABLE public.country_numeric TO manager;

GRANT SELECT ON TABLE public.country_numeric TO marketer;

GRANT ALL ON TABLE public.country_numeric TO postgres;

-- Trigger: tr_update_modification_date_country

-- DROP TRIGGER IF EXISTS tr_update_modification_date_country ON public.country_numeric;

CREATE TRIGGER tr_update_modification_date_country
    BEFORE INSERT OR UPDATE
    ON public.country_numeric
    FOR EACH ROW
    EXECUTE FUNCTION public.update_modification_date();

CREATE TABLE IF NOT EXISTS public.media
(
    media_name text COLLATE pg_catalog."default" NOT NULL,
    url text COLLATE pg_catalog."default"
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.media
    OWNER to postgres;

GRANT DELETE, SELECT, INSERT ON TABLE public.media TO analyst;

GRANT ALL ON TABLE public.media TO manager;

GRANT SELECT ON TABLE public.media TO marketer;

GRANT ALL ON TABLE public.media TO postgres;

CREATE OR REPLACE PROCEDURE public.insert_company(
            IN p_company_name text,
            IN p_company_description text,
            IN p_industry text,
            IN p_year_established text,
            IN p_number_of_employees integer,
            IN p_location_country text,
            IN p_location_city text,
            IN p_address text,
            IN p_products_services text,
            IN p_competitors text,
            IN p_growth_trends_forecasting text)
LANGUAGE 'plpgsql'
AS $BODY$
BEGIN
    INSERT INTO company (
```

```
        company_name,
        company_description,
        industry,
        year_established,
        number_of_employees,
        location_country,
        location_city,
        address,
        products_services,
        competitors,
        growth_trends_forecasting
    ) VALUES (
        p_company_name,
        p_company_description,
        p_industry,
        p_year_established,
        p_number_of_employees,
        p_location_country,
        p_location_city,
        p_address,
        p_products_services,
        p_competitors,
        p_growth_trends_forecasting
    );
END;
$BODY$;
ALTER PROCEDURE public.insert_company(text, text, text, text, integer, text, text, text, text, text, text)
    OWNER TO postgres;

CREATE OR REPLACE PROCEDURE public.insert_company_finance(
            IN p_company_name text,
            IN p_revenue numeric,
            IN p_profit numeric,
            IN p_assets numeric,
            IN p_investments numeric,
            IN p_dividends numeric)
LANGUAGE 'plpgsql'
AS $BODY$
BEGIN
    INSERT INTO company_finance (
        company_name,
        revenue,
        profit,
        assets,
        investments,
        dividends,
        modification_date
    ) VALUES (
        p_company_name,
        p_revenue,
        p_profit,
        p_assets,
        p_investments,
        p_dividends,
        NOW()
    );
END;
$BODY$;
ALTER PROCEDURE public.insert_company_finance(text, numeric, numeric, numeric, numeric, numeric)
    OWNER TO postgres;

CREATE OR REPLACE PROCEDURE public.insert_country(
            IN p_country_name text,
```

```
            IN p_population integer,
            IN p_political_system text,
            IN p_legal_system text,
            IN p_stability text,
            IN p_relations_with_other_countries text,
            IN p_industry text)
LANGUAGE 'plpgsql'
AS $BODY$
BEGIN
   INSERT INTO country (
      country_name,
      population,
      political_system,
      legal_system,
      stability,
      relations_with_other_countries,
      industry
   ) VALUES (
      p_country_name,
      p_population,
      p_political_system,
      p_legal_system,
      p_stability,
      p_relations_with_other_countries,
      p_industry
   );
END;
$BODY$;
ALTER PROCEDURE public.insert_country(text, integer, text, text, text, text, text)
   OWNER TO postgres;

CREATE OR REPLACE PROCEDURE public.insert_country_numeric(
            IN p_country_name text,
            IN p_gdp numeric,
            IN p_gdp_growth numeric,
            IN p_inflation numeric,
            IN p_unemployment numeric,
            IN p_investments numeric)
LANGUAGE 'plpgsql'
AS $BODY$
BEGIN
   INSERT INTO country_numeric (
      country_name,
      GDP,
      GDP_growth,
      inflation,
      unemployment,
      investments,
      modification_date
   ) VALUES (
      p_country_name,
      p_gdp,
      p_gdp_growth,
      p_inflation,
      p_unemployment,
      p_investments,
      NOW()
   );
END;
$BODY$;
ALTER PROCEDURE public.insert_country_numeric(text, numeric, numeric, numeric, numeric, numeric)
   OWNER TO postgres;
```

```
CREATE OR REPLACE PROCEDURE public.sp_insert_contact(
	IN p_first_name text,
	IN p_last_name text,
	IN p_position text,
	IN p_phone_number text,
	IN p_email text)
LANGUAGE 'plpgsql'
AS $BODY$
BEGIN
   INSERT INTO contact (
      first_name,
      last_name,
      position,
      phone_number,
      email
   ) VALUES (
      p_first_name,
      p_last_name,
      p_position,
      p_phone_number,
      p_email
   );
END;
$BODY$;
ALTER PROCEDURE public.sp_insert_contact(text, text, text, text, text)
   OWNER TO postgres;

CREATE OR REPLACE PROCEDURE public.sp_insert_media(
	IN p_name text,
	IN p_url text)
LANGUAGE 'plpgsql'
AS $BODY$
BEGIN
   INSERT INTO Media (
      media_name,
		url
   ) VALUES (
      p_name,
		p_url
   );
END;
$BODY$;
ALTER PROCEDURE public.sp_insert_media(text, text)
   OWNER TO postgres;

CREATE OR REPLACE PROCEDURE public.sp_delete_company(
	IN p_company_name text)
LANGUAGE 'plpgsql'
AS $BODY$
BEGIN
   DELETE FROM company_finance WHERE company_name = p_company_name;
   DELETE FROM company WHERE company_name = p_company_name;
END;
$BODY$;
ALTER PROCEDURE public.sp_delete_company(text)
   OWNER TO postgres;

CREATE OR REPLACE PROCEDURE public.sp_delete_contact(
	IN p_first_name text,
	IN p_last_name text)
LANGUAGE 'plpgsql'
AS $BODY$
BEGIN
```

```
   DELETE FROM contact WHERE first_name = p_first_name AND last_name = p_last_name ;
END;
$BODY$;
ALTER PROCEDURE public.sp_delete_contact(text, text)
   OWNER TO postgres;

CREATE OR REPLACE PROCEDURE public.sp_delete_country(
	IN p_country_name text)
LANGUAGE 'plpgsql'
AS $BODY$
BEGIN
   DELETE FROM country WHERE country_name = p_country_name;
	DELETE FROM country_numeric WHERE country_name = p_country_name;
END;
$BODY$;
ALTER PROCEDURE public.sp_delete_country(text)
   OWNER TO postgres;

CREATE OR REPLACE PROCEDURE public.sp_delete_media(
	IN p_name text)
LANGUAGE 'plpgsql'
AS $BODY$
BEGIN
   DELETE FROM Media WHERE media_name = p_name;
END;
$BODY$;
ALTER PROCEDURE public.sp_delete_media(text)
   OWNER TO postgres;
```