Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИУ «ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ»

КАФЕДРА ИУ7 «ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ»

КУРСОВАЯ РАБОТА

НА ТЕМУ:

Разработка базы данных для хранения и обработки оценок трудовой эффективности сотрудников группы компаний

Студент	**ИУ7-64Б**		**Бугаков**
	(группа)	(подпись, дата)	(И.О. Фамилия)
Руководитель курсового проекта			**Гончаров С.С.**
		(подпись, дата)	(И.О. Фамилия)
Консультант			
		(подпись, дата)	(И.О. Фамилия)

2025 г.

СОДЕРЖАНИЕ

РЕФЕРАТ

Расчетно-пояснительная записка 82 с., 27 рис., 17 таблиц, 32 источника, 1 приложение.

Ключевые слова: базы данных, реляционные базы данных, постреляционные базы данных, объектно-реляционная модель хранения данных, эффективность персонала, вовлеченность персонала, компетенции персонала.

Цель работы: разработка базы данных для хранения и обработки оценок трудовой эффективности сотрудников группы компаний.

В данной работе рассматриваются аспекты проектирования и реализации базы данных и приложения для учета трудовой эффективности сотрудников группы команий. В качестве СУБД выбрана PostgreSQL, которая была интегрирована с приложением, реализованным на языке C# с использованем платформы .NET 9.0.

В результате выполненной работы, были созданы база данных и приложение реализующее интерфейс доступа к ней.

ВВЕДЕНИЕ

К числу проблем оценки эффективности труда персонала относятся формальность оценки профессиональной деятельности, сложность оценки в случаях, когда результаты являются результатами совместной с кем-то деятельности, либо, когда между каким-то видом деятельности и его результатами проходит достаточно большой промежуток времени. Также к трудностям такой оценки можно отнести недопонимание получаемых коэффициентов и показателей эффективности деятельности персонала и их влияния на стимулирование труда.

Существующие методы оценки результативности трудовой деятельности персонала в настоящее время далеки от совершенства. Их влияние на показатели результативности имеет двоякий и не всегда положительный характер, а мотивация сотрудников не всегда учитывается [22].

Цель работы — разработка базы данных для хранения и обработки оценок трудовой эффективности сотрудников группы компаний.

Для достижения поставленной цели необходимо решить следующие задачи:

1) провести анализ предметной области, выделить основные сущности и связи между ними, сформулировать требования и ограничения к разрабатываемой базе данных;

2) спроектировать сущности базы данных и определить ограничения целостности данных, спроектировать ролевую модель на уровне базы данных;

3) выбрать средства реализации базы данных и приложения;

4) реализовать сущности базы данных и ограничения целостности. Описать методы тестирования и разработать тестовые случаи для проверки корректности работы функционала;

5) провести исследование производительности базы данных при увеличении объема данных и количестве одновременно выполняемых запросов, а также с использованием кеширования и без него.

1 Аналитическая часть

В данной части приводится анализ области методов оценки трудовой эффективности сотрудников, оценка существующих решений. На основе анализа формулируются требования к базе данных и приложению, выделяются сущности и пользователи базы данных, создаются схемы данных, в том числе диаграмма вариантов использования и диаграмма сущность-связь в нотации Чена.

Также на основе свойств данных, которые будут храниться в системе, выбирается наиболее подходящая модель базы данных. Приводится сравнительный анализ различных методов хранения данных, на основе которого выбирается наилучший.

1.1 Анализ предметной области

1.1.1 Методы оценки трудовой эффективности

Существуют различные методы оценки трудовой эффективности [22]:

— количественные — числовые значения, показывающие эффективность сотрудника;

— качественные — описание и соответствующий анализ деятельности работника по его характеристике;

— комбинированные — сочетание цифровых значений с дополняющими описаниями.

В дальнейшем рассматриваются количественные методы или те, которые в конечном итоге могут быть преобразованы к таковым.

1.1.2 Эффективность работника

Говоря об оценке трудовой эффективности сотрудников, необходимо определить понятие эффективности. Эффективность — относительный показатель, определяемый как отношение между достигнутыми результатами и затраченными ресурсами [21].

Среди затрачиваемых ресурсов принято выделять следующие [31]:

— временные;

— материальные;

— трудовые.

Приведенные выше ресурсы можно оценить теоретически. Например, для расчета теоретических затрат применяются подходы сетевого планирования, такие как диаграммы Ганта, метод оценки и анализа проекта (PERT), метод критического пути (CPM) [24].

Для определения достигнутых результатов на сегодняшний день наиболее популярным является метод оценки по целям. Суть метода заключается в установлении работнику на задан-

ный период определенных профессиональных целей, достижение которых позволяет добиться конечной цели организации. В конце периода результаты деятельности сравниваются с установленными в плане показателями [27].

Для оценки эффективности кроме определения также могут применяться специальные подходы, например, подход с использованием ключевых показателей эффективности (KPI) [23].

1.1.3 Вовлеченность работника

Помимо непосредственно эффективности в смысле данного выше определения, необходимо также учитывать вовлеченность сотрудника в рабочий процесс, его мотивацию. Вовлеченность сотрудников —- это подход, позволяющий добиться того, чтобы каждый сотрудник искренне заботился о своей работе, о компании, в которой работает и о ее клиентах, подход, помогающий добиться того, чтобы сотрудник полностью посвящал себя работе, прилагал все усилия в этом отношении. Это выражается в том, что сотрудник проявляет проактивность и энтузиазм по поводу работы и берет на себя полную ответственность [29].

Для оценки вовлеченности сотрудников, необходимо сформулировать критерии по которым она будет производиться. Согласно [29] вовлеченный сотрудник должен обладать следующими качествами:

— поглощен работой — «на работе время летит быстро»;
— поддерживает концентрацию в течение длительного времени;
— чувствует сильную эмоциональную связь с компанией;
— относится к работе с энтузиазмом и страстью;
— расширяет сферу своей ответственности, гибкий, не ограничивается описанием должностных обязанностей;
— адаптируется к изменениям;
— стремится развивать рабочие навыки;
— не нуждается в напоминаниях и приказах;
— делает все вовремя;
— настойчив;
— проявляет инициативу;
— ориентирован на достижение цели;
— добросовестный;
— ответственный и обязательный;
— предан работе.

Оценивая, насколько развиты перечисленные качества у отдельно взятого сотрудника, можно оценить его вовлеченность.

1.1.4 Компетенция работника

Наконец, необходимо правильно оценивать компетенции работника. Независимо от цели исследования и используемого метода, оценка компетенции должна проводиться с учетом ряда принципов [30]:

— объективность — для характеристики работника, результатов его деятельности должна использоваться достоверная информация с учетом периода работы человека и показателей эффективности его работы;

— гласность — работник должен быть полностью ознакомлен с применяемой для оценки методикой и непосредственно с результатами оценки;

— оперативность — оценка должна проводиться своевременно, оперативно и систематически;

— демократизм — в процедуре оценивания должны принимать участие члены коллектива: коллеги, подчиненные;

— единство — работники, занимающие однородные должности должны оцениваться по единым требованиям;

— оптимальность — процедура оценки должна быть четкой, доступной и простой для понимания;

— результативность — в соответствии с результатами оценки в обязательном порядке должны приниматься и реализовываться соответствующие управленческие решения.

Наиболее популярные методы оценки компетенций работников [30]:

1) психологическое тестирование. Метод позволяет определить профессиональную компетентность на основе проявления определенных индивидуальных свойств личности. Использование метода эффективно при отборе кадров, ротации персонала и формировании кадрового резерва;

2) биографический метод. Суть метода заключается в получении данных о компетентности кандидата на основе автобиографии, документов об образовании, повышении квалификации, характеристик и рекомендаций;

3) метод структурированного интервью. Метод представляет собой беседу в режиме «вопрос-ответ» с кандидатом с целью получения дополнительной информации;

4) метод анкетирования. Оценка с помощью метода анкетирования проводится на основе обобщения и анализа ответов работников по определенным формам;

5) метод наблюдения. Оценка с помощью метода наблюдения проводится на основе увиденных и зафиксированных действий работника в процессе выполнения им должностных обязанностей;

6) экспертный метод. Оценка на основе экспертного метода проводится на основе сбора и анализа информации о соответствии уровня компетенций работника конкретной должности.

Возможно использование отдельных методов или нескольких сразу, по результатам которых будет сформирована итоговая оценка компетенций работника.

1.2 Анализ существующих решений

В настоящее время существует множество вариантов программ для оценки трудовой эффективности сотрудников, которые помогают оценивать как отдельные аспекты, например вовлеченность или компетенции, так и общую эффективность на основе заданных критериев.

Ниже представлены программы, предоставляющие возможность оценить вовлеченность сотрудников:

— Happy Jod — веб-приложение, позволяющее проводить оценку вовлеченности и лояльности сотрудников и расчитывать метрики eNPS (вероятность рекомендации компании сотрудником своим знакомым) и mNPS (удовлетворенность сотрудниками своим руководителем) на основе опросов [6];

— Jinn — веб-приложение, позволяющее формировать планы развития сотрудников, оценивать их вовлеченность и лояльность на основе опросов и встреч 1–1. Кроме того, позволяет создавать собственные опросы, например для оценки компетенций сотрудников [9].

Среди программ оценки компетенций сотрудников можно выделить:

— iSpring — веб-платформа с набором инструментов для создания собственных тестов или заданий на решение кейсов на основе готовых шаблонов или с нуля, в зависимости от требований компании, загружать собственные обучающие материалы. Предоставляет возможность построения аналитики по результатам выполнения задач. Задания могут быть персонализированными или коллективными для группы сотрудников [8];

— StartExam — веб-платформа, предоставляющая возможность создавать с помощью конструктора или загружать готовые тесты для оценки компетенций персонала [17];

— Proaction — веб-платформа, предоставляющая возможность создавать тесты для оценки компетенций персонала с помощью конструктора [13].

Для сравнения представленных выше систем оценки сотрудников, выделены следующие критерии:

— возможность создания тестовых заданий;

— возможность создания заданий на решение кейсов;

— возможность оценки вовлеченности сотрудника;

— возможность расчета NPS метрик.

В таблице 1.1 приведено сравнение с использованием сформулированных критериев.

Таблица 1.1 — Сравнение существующих решений в области оценки вовлеченности и компетенций персонала

Система	Создание тестовых заданий	Создание кейсов	Оценка вовлеченности	Рассчет NPS метрик
Happy Job	-	-	+	+
Jinn	+	-	+	+
iSpring	+	+	+	-
StartExam	+	-	-	-
Proaction	+	-	-	-

Для оценки эффективности могут быть использованы следующие программы:

— Redmine — веб-приложение для управления проектами. Позволяет отслеживать фактические устанавливать теоретические временные затраты сотрудника на выполнение задач, создавать и отслеживать записи об ошибках. Имеет встроенную поддержку диаграмм Ганта [15];

— Weeek — SaaS-сервис, сочетающий возможности использовать облачный сервис и хранить данные на локальном сервере, для управления проектами. В данной системе объединены таск-трекер, база знаний, CRM и аналитика прогресса по сотрудникам или проектам [20];

— ToolKeeper — веб-приложение для учета товарно-материальных ценностей. Позволяет определить какое оборудование используется сотрудником в течении какого периода. Может использоваться для учета материальных затрат сотрудника при работе над проектом [19].

Все вышеперечисленные решения заточены на решение определенных задач связанных с оценкой эффективности сотрудников. В результате данная работа отличается от существующих решений возможностью учета оценок всех перечисленных аспектов трудовой эффективности в единой системе для использования группой компаний. При этом исходные данные для проведения оценивания могут быть получены от отдельных компонентов представленных уже существующими программными продуктами.

1.3 Требования к базе данных и приложению

База данных должна предоставлять возможность хранения и обработки:

— оценок работников группы компаний по критериям непосредственно эффективности, вовлеченности и компетенции;

— информации о сотрудниках;

— информации о компаниях группы;

— должностях и позициях занимаемых сотрудниками и историях их изменения.

В базе данных должны быть предусмотрены роли и группы пользователей для реализации

системы безопасности и соответствующих ограничений доступа к данным.

Наиболее часто используемые запросы, например, поиск непосредственных подчиненных позиции или поиск текущих подчиненных сотрудника, должны быть выделены в виде хранимых процедур. Также необходимо реализовать триггеры обеспечивающие целостность и согласованность данных.

Приложение должно предоставлять интерфейс для просмотра информации о сотрудниках, компаниях, истории изменения позиций, должностей и оценок персонала, текущем дереве подчинения сотрудников в компании; возможность выставлять оценки сотрудникам их непосредственными руководителями. Также для учетной записи администратора должны быть предусмотрены необходимые интерфейсы добавления и редактирования позиций, должностей, компаний и сотрудников.

1.4 Анализ архитектур баз данных

По типу нагрузки базы данных разделяются на [28]:

— OLAP (Online Analytical Processing) — обработка аналитики в реальном времени;

— OLTP (Online Transaction Processing) — обработка транзакций в реальном времени.

Сравнение OLAP и OLTP приведено в таблице 1.2 [28].

Таблица 1.2 — Сравнение OLAP и OLTP

Показатель	Обработка транзакций	Обработка аналитики
Интенсивность операций	От десятков до тысяч операций в секунду	От единиц до сотен операций в секунду
Время выполнения операции	От десятков миллисекунд до секунд	От секунд до часов
Количество обрабатываемых операцией объектов	От единиц до тысяч	От десятков тысяч до миллиардов
Соотношение записи и чтения	От $100:1$ в пользу чтения до $100:1$ в пользу записи	От $1000:1$ до $10^6:1$ в пользу чтения
Использование колонок при выполнении записи	Все	Небольшое подмножество
Сложность запросов	Низкая	Высокая
Разнообразие запросов	Низкое (типовые запросы)	Высокое (специализированные запросы

По модели хранения базы данных разделяют на:

— дореляционные — системы, основанные на инвертированных списках, а также сетевой и иерархической моделях данных. В таких системах невозможен контроль целостности на уровне базы данных, что противоречит принципам организации данных [26];

— реляционные — системы, в которых данные рассматриваются пользователем в виде связанных между собой таблиц. Такая система позволяет накладывать ограничения

целостности на уровне базы данных, что обеспечивает неизбыточность и соблюдение логической согласованности данных [26];

— постреляционные — системы, представляющие собой расширения реляционной модели, главным допущением которых является отсутствие требования атомарности полей [32]. К таким системам относятся, например, объектно-ориентированные базы данных [26].

Проектируемая база данных должна поддерживать ограничения целостности на уровне базы данных, но при этом содержит составные типы. В связи с этим была выбрана объектно-реляционная модель хранения данных.

1.5 Формализация данных, хранимых в БД

Исходя из требований к базе данных и анализа предметной области, были выделены следующие категории данных:

— сотрудник;

— образование;

— должность;

— позиция;

— компания;

— история оценок сотрудника;

— история позиций сотрудника;

— история должностей сотрудника;

В таблице 1.3 представлены категории данных и хранимые о них сведения.

Таблица 1.3 — Категории данных и сведения

Категория	Хранимые сведения
Сотрудник	ФИО, телефон, рабочий email, дата рождения, возраст, фото, характеристика рабочих обязанностей
Образование	Дата начала обучения, дата окончания обучения, учебное заведение, уровень образования, направление подготовки
Должность	Название, оклад
Позиция	Название
Компания	Название, дата регистрации, телефон, email, ИНН, ОГРН, КПП, юридический адрес
История оценок	Оценка эффективности, оценка вовлеченности, оценка компетенций, автор оценки, позиция оцениваемого, дата оценивания
История позиций	Дата вступления на позицию, дата ухода с позиции
История должностей	Дата вступления в должность, дата ухода с должности

На рисунке 1.1 представлена диаграмма сущность-связь в нотации Чена.

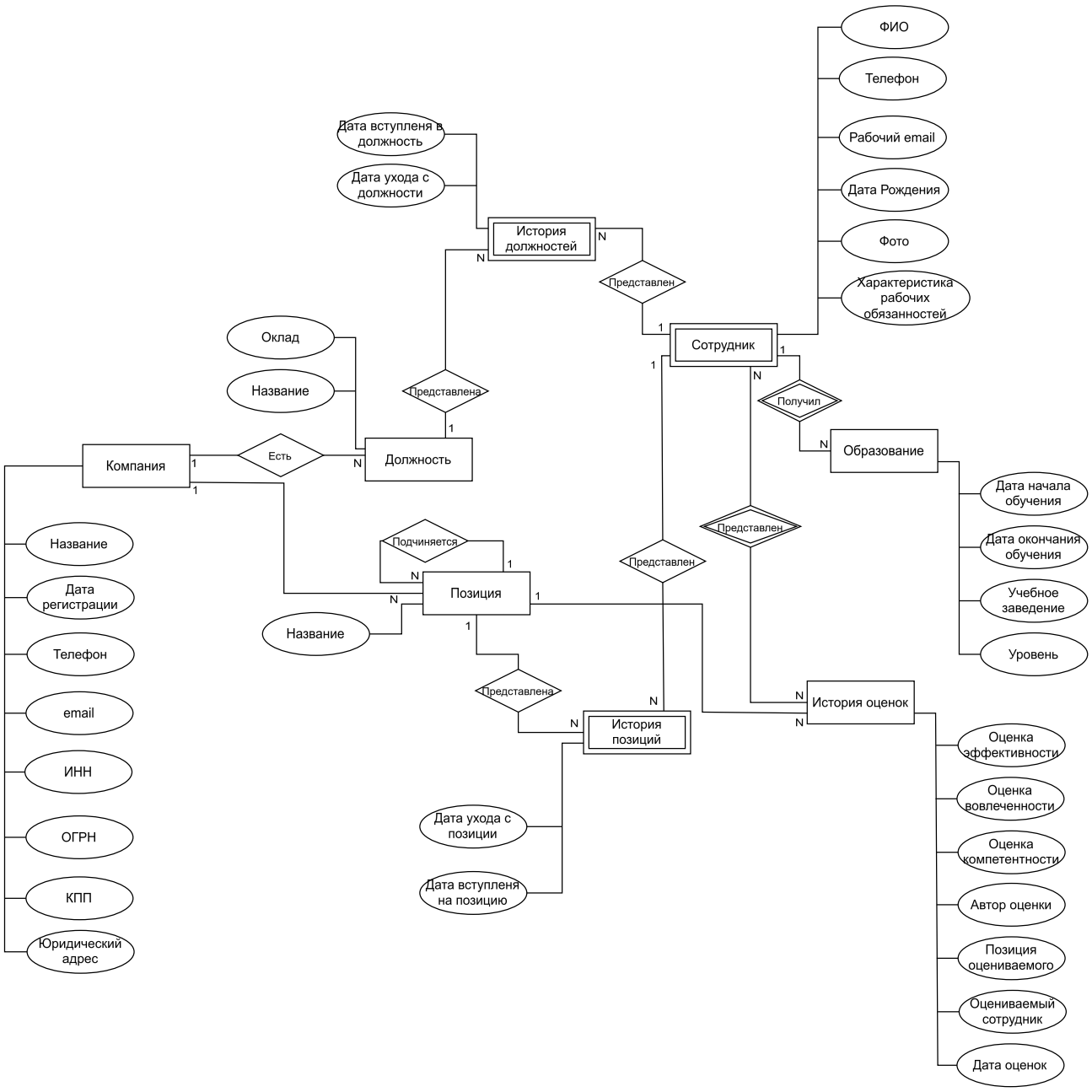

Рисунок 1.1 — Диграмма сущность-связь

1.6 Формализация сценариев использования

Среди пользователей базы данных были выделены три категории пользователей:

— администратор — осуществляет контроль и управление работой системы, добавляет и редактирует информацию о пользователях, компаниях, должностях и позициях;

— сотрудник — может просматривать информацию о сотрудниках на той же позиции и всех своих подчиненных, выставлять оценки непосредственным подчиненным. Дополнительно имеет доступ к информации о компаниях, позициях и должностях, без права редактирования. Также может просматривать историю изменения позиций и должностей своих подчиненных и сотрудников на том же уровне позиции;

— гость — неавторизованный пользователь, может просматривать информацию о позициях, должностях и компаниях, без возможности редактирования.

На рисунке 1.2 представлена диаграмма прецедентов приложения.

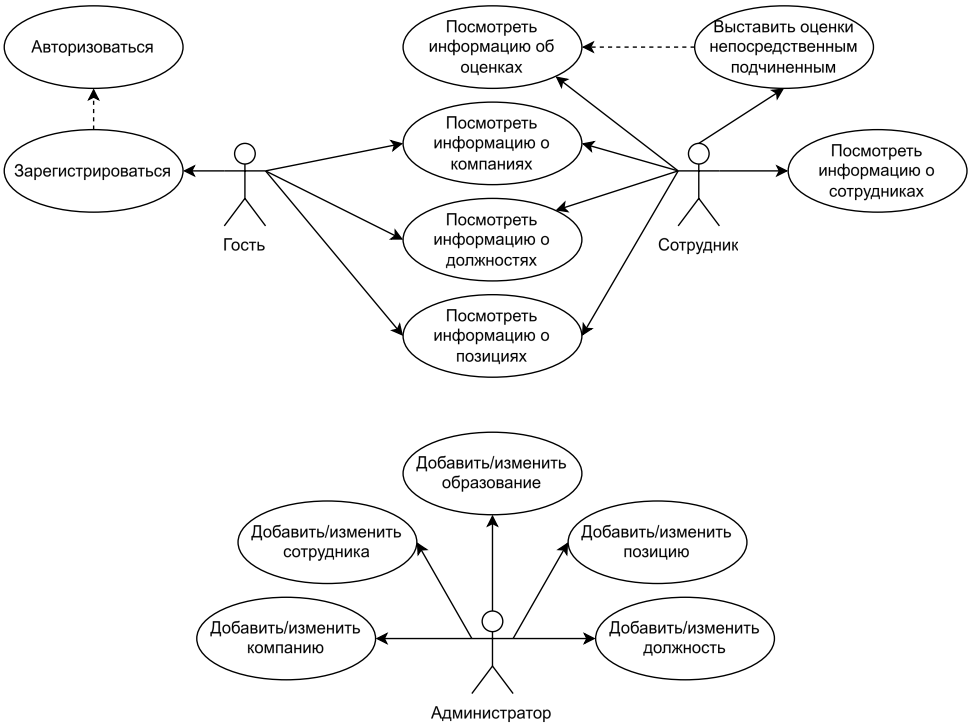

Рисунок 1.2 — Диаграмма прецедентов

Вывод

В данном разделе был проведен анализ предметной области оценки трудовой эффективности сотрудников и сравнительный анализ существующих решений. Были сформулированы требования к базе данных и приложению, формализованы данные, которые будут храниться в БД. Для хранения данных была выбрана постреляционная модель хранения данных, представлена диаграмма сущность-связь в нотации Чена. Описаны пользователи базы данных и приведена диаграмма прецедентов.

2 Конструкторская часть

2.1 Описание сущностей базы данных

В соответствии с таблицей 1.3, рисунком 1.1 и алгоритмом получения реляционный схемы из диаграммы сущность-связь, простые сущности преобразуются в следующие таблицы:

— company — таблица компаний в группы;

— post — таблица должностей, представленных в компания группы;

— position — таблица позиций сотрудника;

— employee — таблица сотрудников;

— post_story — таблица историй изменений должностей сотрудников;

— position_story — таблица историй изменений позиций сотрудников;

— education — таблица пройденных сотрудниками учебных курсов, в том числе получение среднего общего и высшего образования, курсов повышения профессиональной квалификации, курсов профессиональной переподготовки и т. п.;

— score_story — таблица оценок сотрудников за все время их работы в компаниях группы.

В таблицах 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8 представлена информация о структуре столбцов перечисленных таблиц, указаны их ограничения.

Таблица 2.1 — Описание столбцов таблицы employee

Столбец	Ограничения	Значение
id	primary key	Идентификатор сотрудника
full_name	not null	ФИО сотрудника
phone	check	Номер телефона
email	not null, check	Адрес рабочей электронной почты
birth_date	not null, check	Дата рождения
photo		Путь к фотографии в файловом хранилище
duties		Характеристика рабочих обязанностей в формате json

Таблица 2.2 — Описание столбцов таблицы education

Столбец	Ограничения	Значение
id	primary key	Идентификатор записи
employee_id	foreign key	Идентификатор сотрудника
institution	not null	Название учебного заведения
education_level	not null	Уровень полученного образования
study_field	not null	Направление подготовки
start_date	not null, check	Дата начала обучения
end_date	not null, check	Дата окончания обучения

Таблица 2.3 — Описание столбцов таблицы post

Столбец	Ограничения	Значение
id	primary key	Идентификатор должности
title	not null	Название
salary	not null, check	Оклад
company_id	foreign key	Идентификатор компании, в которой введена должность

Таблица 2.4 — Описание столбцов таблицы position

Столбец	Ограничения	Значение
id	primary key	Идентификатор позиции
parent_id	foreign key	Идентификатор непосредственного начальника
title	not null	Название
company_id	foreign key	Идентификатор компании позиции

Таблица 2.5 — Описание столбцов таблицы company

Столбец	Ограничения	Значение
id	primary key	Идентификатор компании
title	not null	Название
registration_date	not null, check	Дата регистрации
phone	check	Номер телефона
email	not null, check	Адрес рабочей электронной почты
inn	not null, check	Индивидуальный номер налогоплательщика компании
kpp	not null check	Код причины постановки на учет
ogrn	not null, check	Основной государственный регистрационный номер
address	not null	Юридический адрес компании

Таблица 2.6 — Описание столбцов таблицы post_history

Столбец	Ограничения	Значение
post_id	foreign key	Идентификатор должности
employee_id	foreign key	Идентификатор сотрудника
start_date	not null, check	Дата вступления в должность
end_date	check	Дата ухода с должности

Таблица 2.7 — Описание столбцов таблицы position_history

Столбец	Ограничения	Значение
position_id	foreign key	Идентификатор позиции
employee_id	foreign key	Идентификатор сотрудника
start_date	not null, check	Дата вступления на позицию
end_date	check	Дата ухода с позиции

Таблица 2.8 — Описание столбцов таблицы score_story

Столбец	Ограничения	Значение
id	primary key	Идентификатор оценки
employee_id	foreign key	Идентификатор сотрудника, которому выставлена оценка
author_id	foreign key	Идентификатор сотрудника, который выставил оценку
position_id	foreign key	Позиция оцениваемого
created_at	not null, default	Дата и время создания оценки
efficiency_score	check	Оценка эффективности сотрудника
engagement_score	check	Оценка вовлеченности сотрудника
competency_score	check	Оценка компетенций сотрудника

При удалении позиции, должности или компании, необходимо сделать так, чтобы история работы пользователя валидной. Для этого данные не будут удаляться на прямую, а будут введены специальные флаги в таблицах, указывающие, что данная позиция, должность или компания уже были удалены. Редактироваие удаленных сущностей будет запрещено. На рисунке 2.1 приведена диаграмма базы данных.

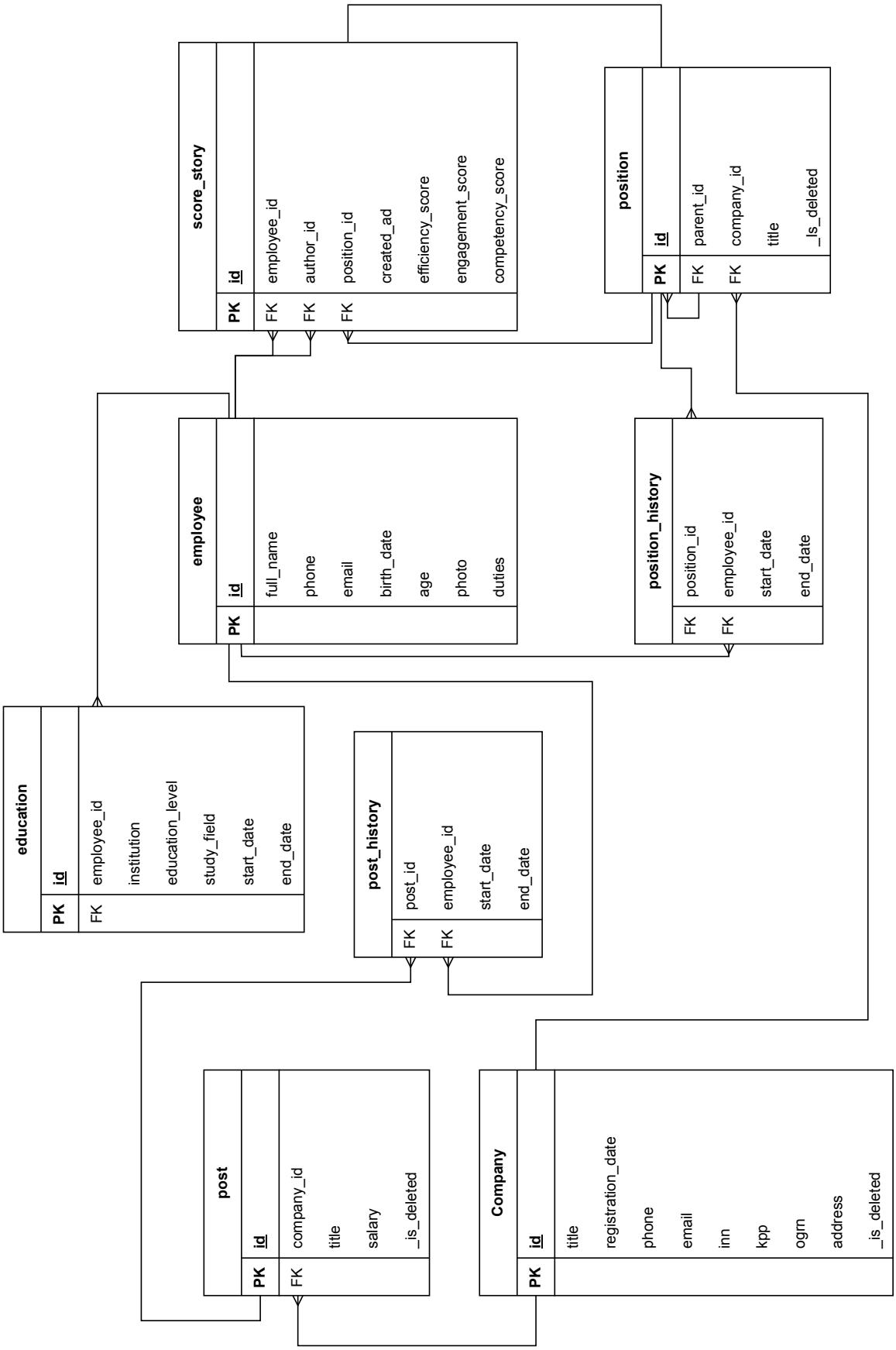

Рисунок 2.1 — Диаграмма базы данных

2.2 Ролевая модель

Гость имеет права на выполнение только SELECT запросов в таблицах company, post и position.

Сотрудник имеет права на выполнение SELECT запросов в таблицах:

— company без ограничений;

— post без ограничений;

— position без ограничений;

— post_hitsory при условии, что позиция просматриваемого сотрудника ниже позиции просматривающего или на том же уровне, в той же компании;

— position_hitsory при условии, что позиция просматриваемого сотрудника ниже позиции просматривающего или на том же уровне, в той же компании;

— employee при условии, что позиция просматриваемого сотрудника ниже позиции просматривающего или на том же уровне, в той же компании;

— score_history при условии, что позиция просматриваемого сотрудника ниже позиции просматривающего или на том же уровне, в той же компании;

— education при условии, что записи относятся к самому сотруднику, его подчиненным или сотрудникам на том же уровне в той же компании.

Сотрудник имеет права на выполнение INSERT запросов в таблице score_history при условии, что сотрудник, которому выставляется оценка является непосредственным подчиненным.

Прочие запросы сотрудник выполнять не может. Для реализации таких ограничений доступа, будут использоваться row security policies [16]. Так как ограничения затрагивают таблицы на основе которых они будут вычисляться, необходимо ввести дополнительные таблицы. Для таблицы employee создадим таблицу employee_reduced, для position_history — position_history_reduced, для position — position_reduced, в последней будут поддерживаться только занимающие позицию на текущий момент сотрудники. Поля этих таблиц представлены в таблицах 2.9 2.10 2.11.

Таблица 2.9 — Описание столбцов таблицы employee_reduced

Столбец	Ограничения	Значение
id	foreign key	Идентификатор сотрудника
email	unique, not null, check	Адрес рабочей электронной почты

Таблица 2.10 — Описание столбцов таблицы position_reduced

Столбец	Ограничения	Значение
id	primary key	Идентификатор позиции
parent_id	foreign key	Идентификатор непосредственного начальника

Таблица 2.11 — Описание столбцов таблицы position_history_reduced

Столбец	Ограничения	Значение
position_id	foreign key	Идентификатор позиции
employee_id	foreign key	Идентификатор сотрудника

Для этих таблиц row security policies реализованы не будут, сотрудник не обладает правами прямого чтения или изменения этих таблиц, чтение будет осуществляться только из специальных функций, описанных ниже и используемых только для вычисления row security policies.

Администратор имеет права на выполнение любых запросов к любым таблицам, не нарушая при этом целостности данных.

2.3 Используемые триггеры

Для отслеживания целостности данных и предотвращения становления более не валидными в базе данных будут реализованы триггеры. Ниже описаны некоторые из них.

При удалении позиции, необходимо, чтобы подчиненные позиции остались в иерархии. В связи с этим необходимо реализовать триггер, выполняемый перед операцией DELETE в таблице position, назначающий руководителем подчиненных удаляемой позиции ее руководителя. При этом, поскольку в position_reduced поддерживаются только актуальные позиции, из этой таблицы запись необходимо удалить. На рисунке 2.2 представлена схема такого триггера.

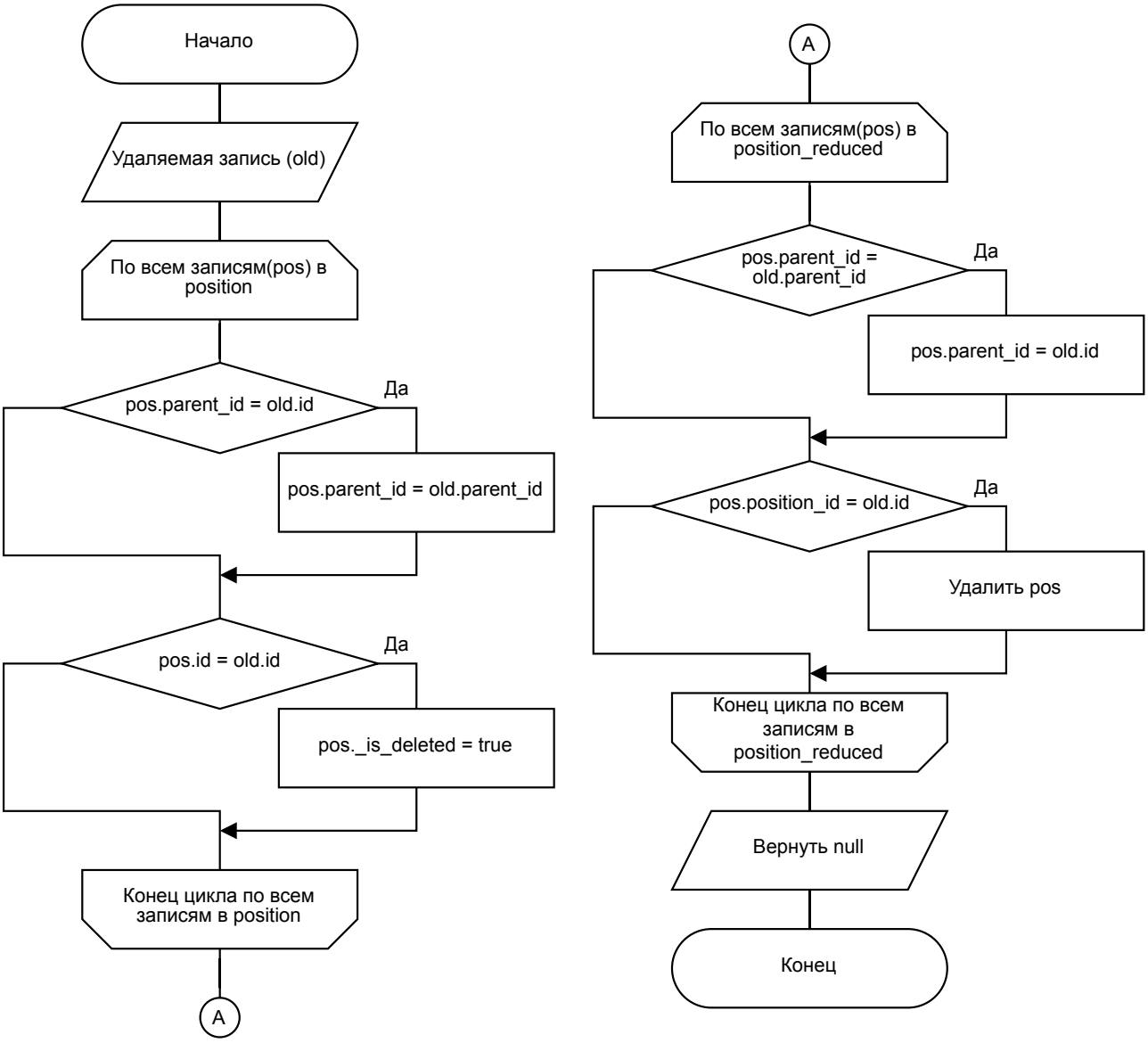

Рисунок 2.2 — Схема триггера before delete для таблицы position.

При обновлении непосредственного руководителя позиции, имеющей подчиненных, поведение пользователя не определено: пользователь может хотеть переместить только вершину, соответствующую обновляемой позиции в иерархии, или переместить все поддерево, корнем которого является вершина, соответствующая обновляемой позиции. Таким образом, необходимо реализовать триггер, который запретит использование операции UPDATE для таблицы position напрямую и будет указывать пользователю на необходимость использования специальных функций, которые будут описаны ниже. Схема триггер, выполняемого перед операцией UPDATE в таблице position и выполняющего необходимые проверки указаны на рисунке 2.3.

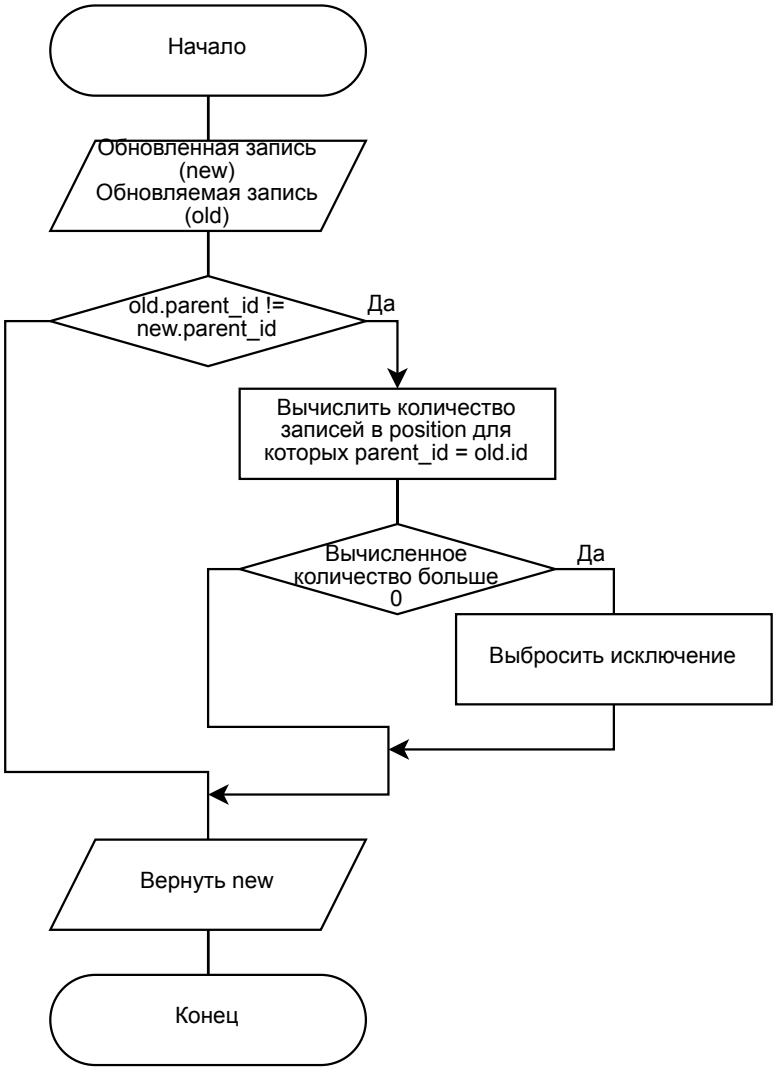

Рисунок 2.3 — Схема триггера before update для таблицы position.

Минимальный интервал между выставляемыми оценками сотруднику на одной и той же позиции должен составлять не менее одного месяца. Для проверки этого требования будет реализован триггер, выполняемый перед операциями INSERT и UPDATE в таблице score_story. На рисунке 2.4 представлена схема этого триггера.

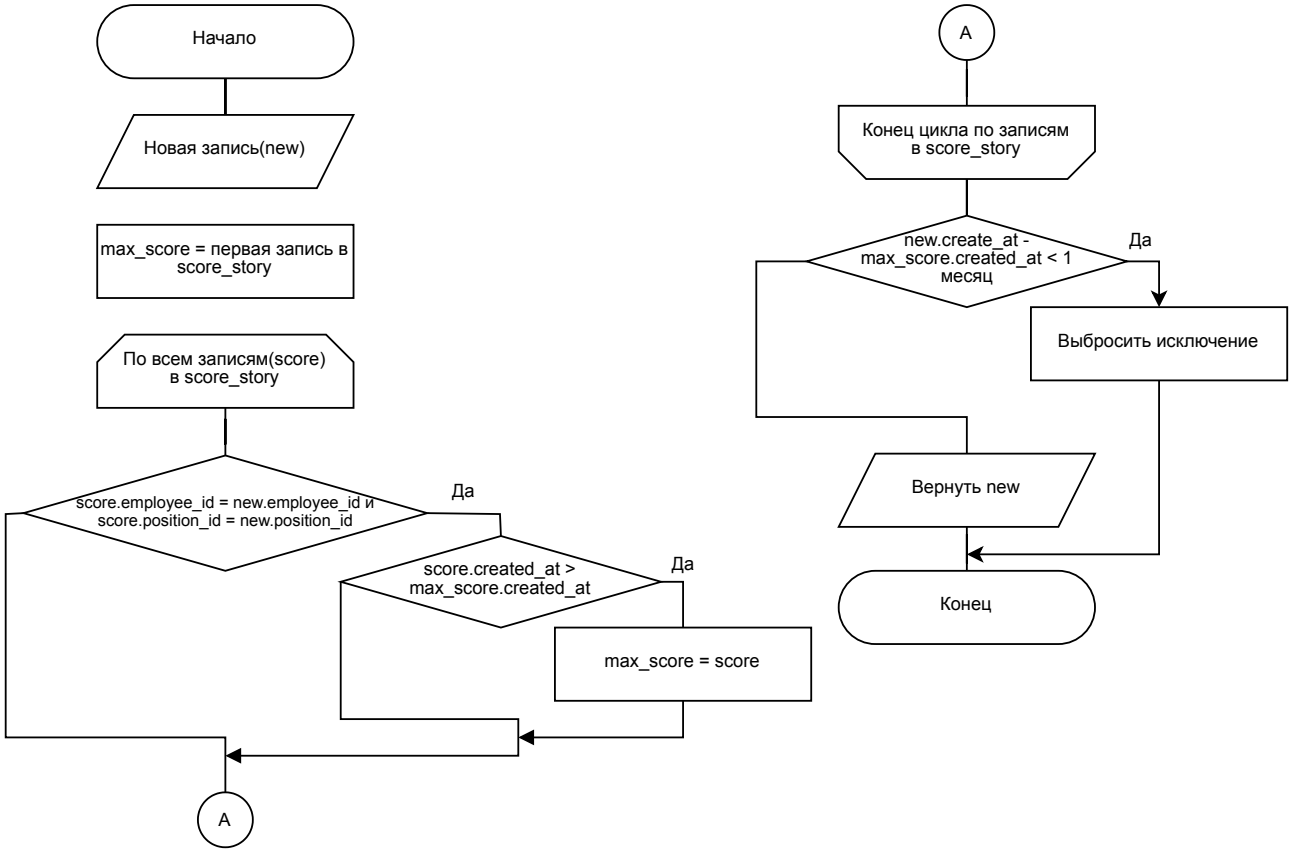

Рисунок 2.4 — Схема триггера before insert or update для таблицы score_story.

Поскольку компании не удаляются из таблицы напрямую, необходим триггер, который будет устанавливать соответствующий флаг. На рисунке 2.5 представлена схема триггера, вызываемого перед операцией DELETE для таблицы company.

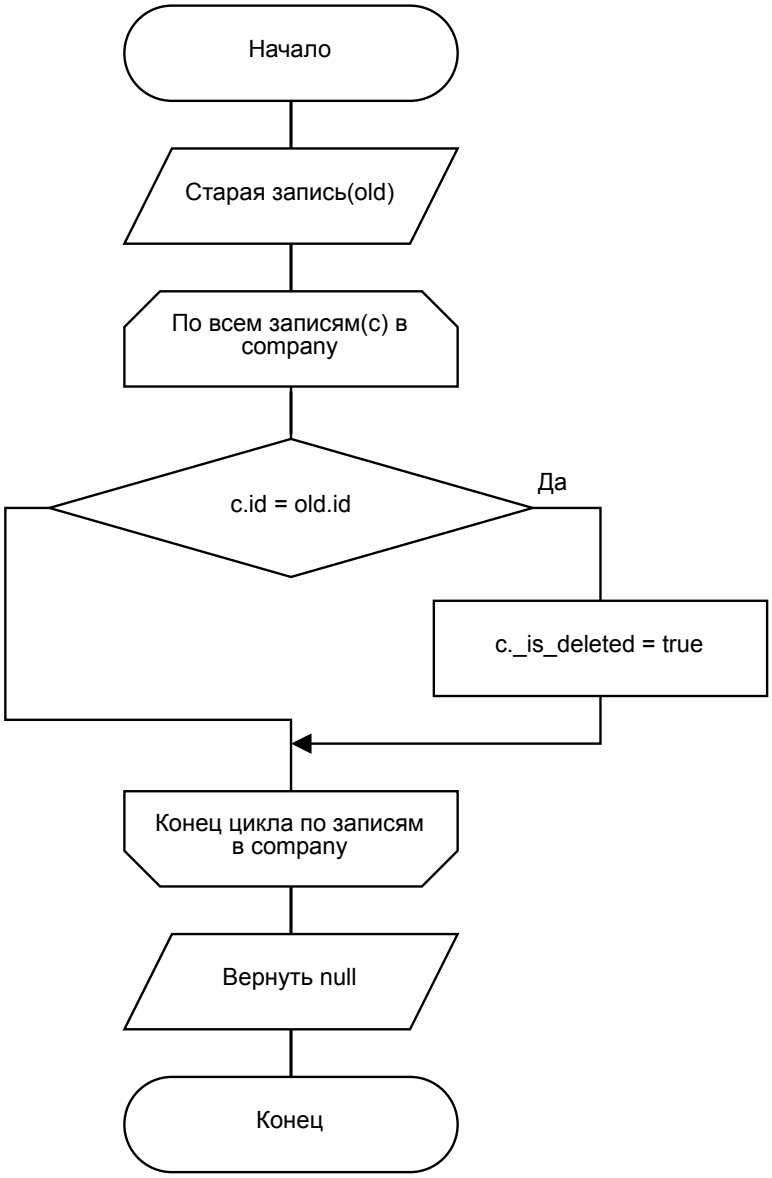

Рисунок 2.5 — Схема триггера before delete для таблицы company.

Аналогичный триггер будет реализован для таблицы post. При этом предполагается, что при удалении должности, сотрудники перестают занимать эту должность. На рисунке 2.6 показана схема триггера, вызываемого перед операцией DELETE для таблицы post.

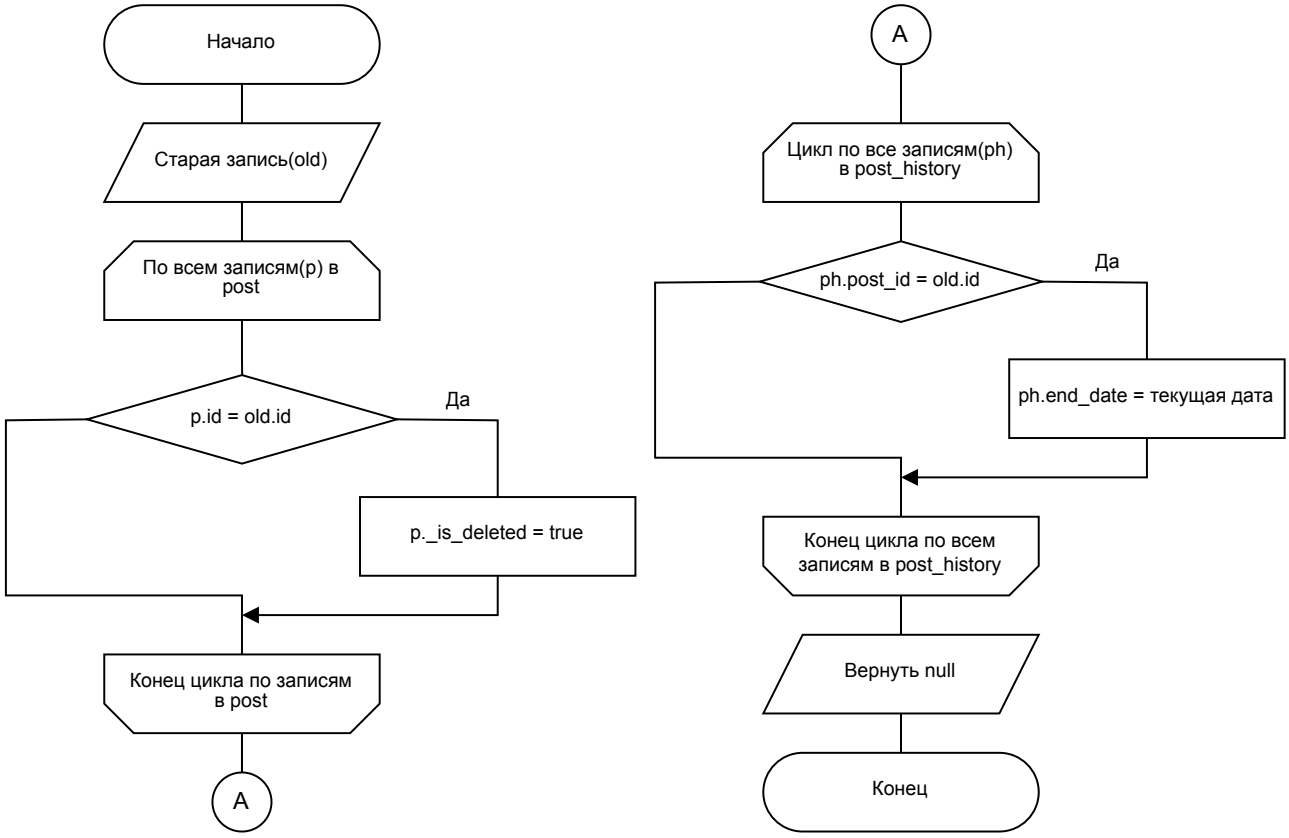

Рисунок 2.6 — Схема триггера before delete для таблицы post.

Аналогичный триггер для таблицы position, должен помимо снятия сотрудников с позиций в position_history, должен также удалять все записи об этой позиции в position_history_reduced. На рисунке 2.7, приведена схема триггера, выполняемого перед операцией DELETE в таблице position.

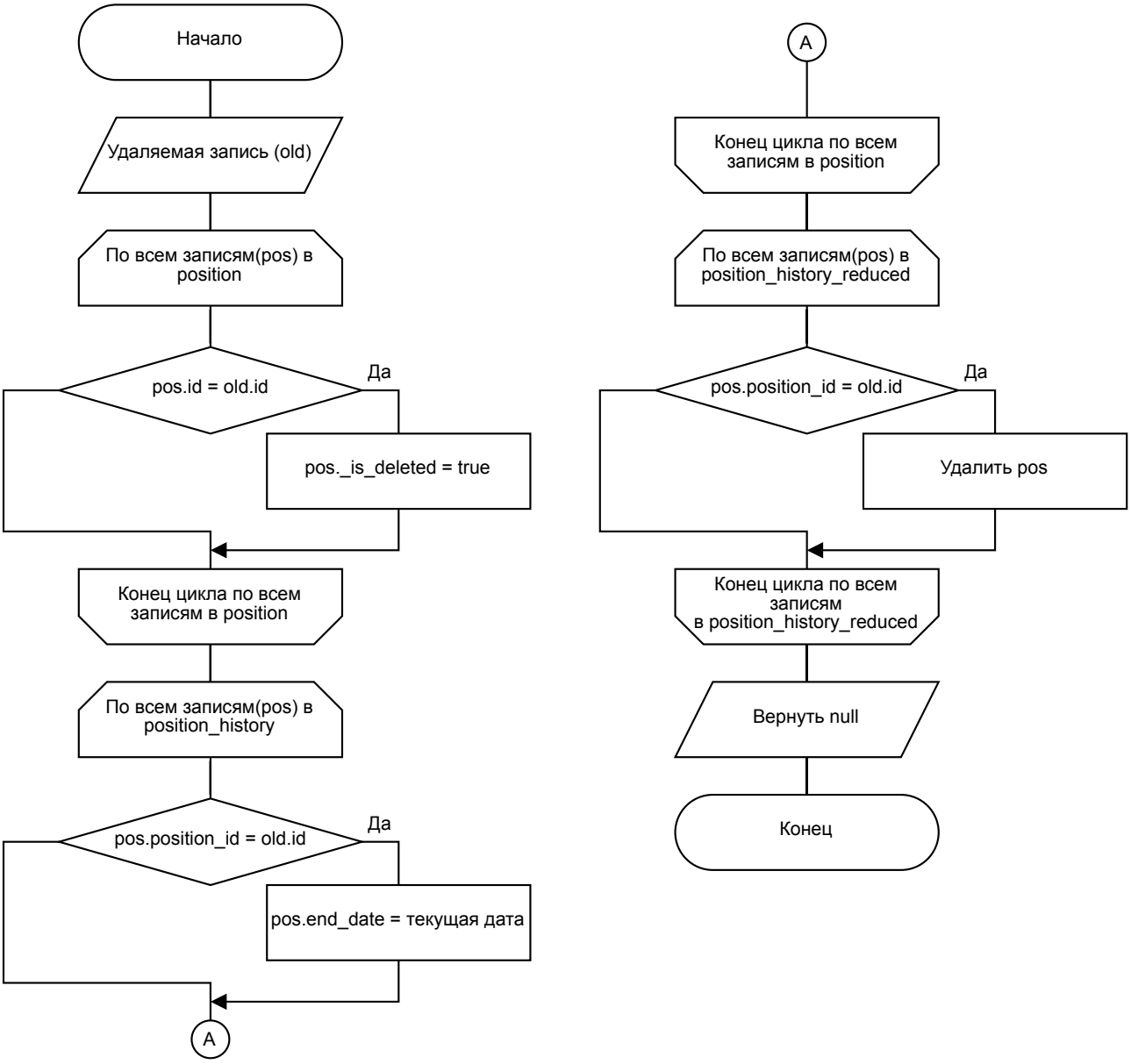

Рисунок 2.7 — Схема триггера before delete для таблицы position.

При обновлении данных в таблице position_history, необходимо также обновить данные в таблице position_history_reduced. При этом, если обновление снимает сотрудника с текущей позиции, необходимо удалить запись об этом в position_history_reduced. Схема триггера, реализующего такое поведение и вызываемого после операции UPDATE в таблице position_history, показана на рисунке 2.8.

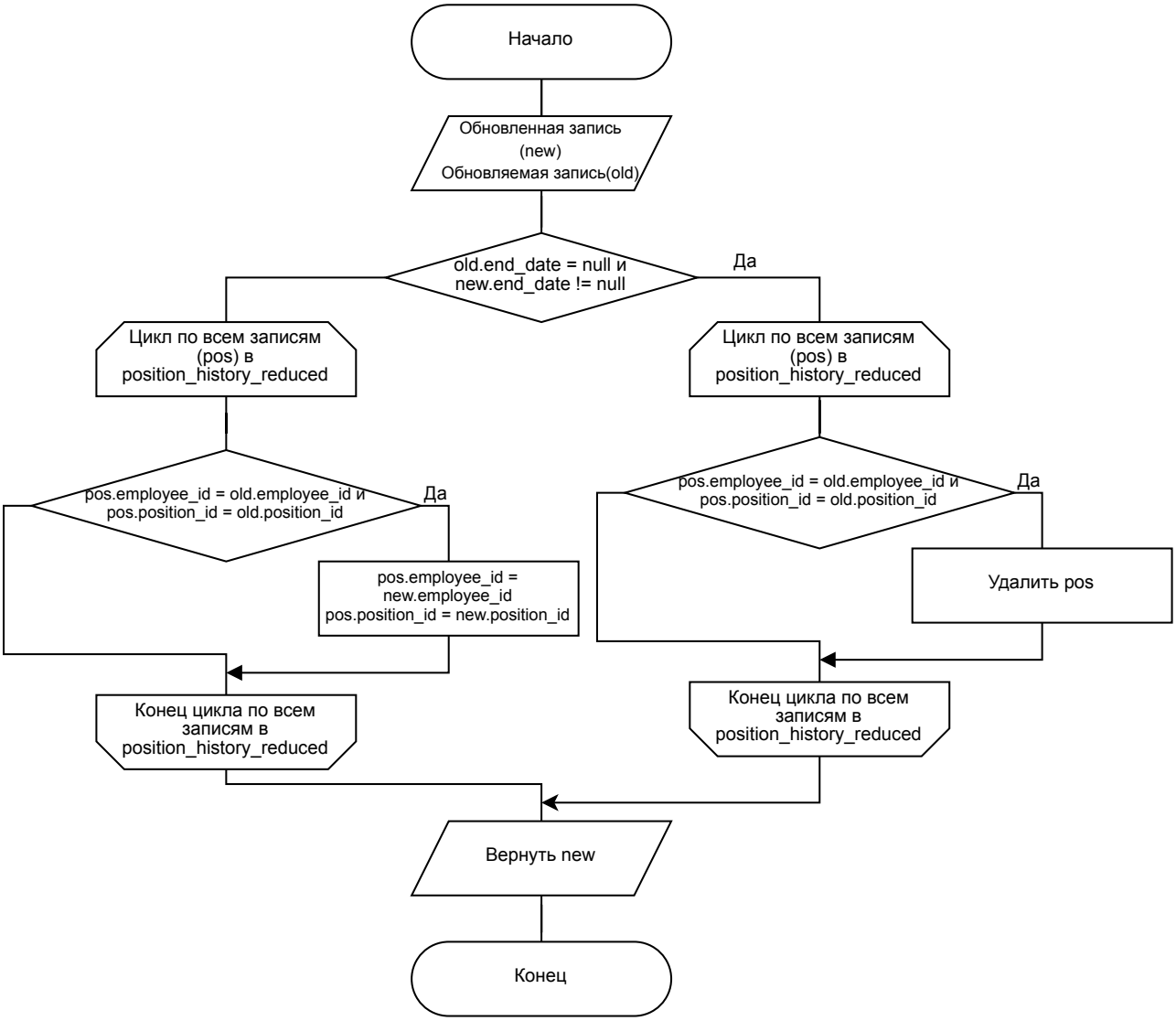

Рисунок 2.8 — Схема триггера after update для таблицы position_history.

Сотрудник может занимать единственную должность в каждой компании в каждый момент времени. Соответственно, в случае вставки данных в таблицу post_history, необходимо проверить, что сотрудник в настоящий момент не занимает никакой должности и, в случае если занятая им должность найдена, снять с нее сотрудника. На рисунке 2.9 показана схема триггера, вызываемого перед операцией INSERT в таблице post_history, для выполнения таких действий.

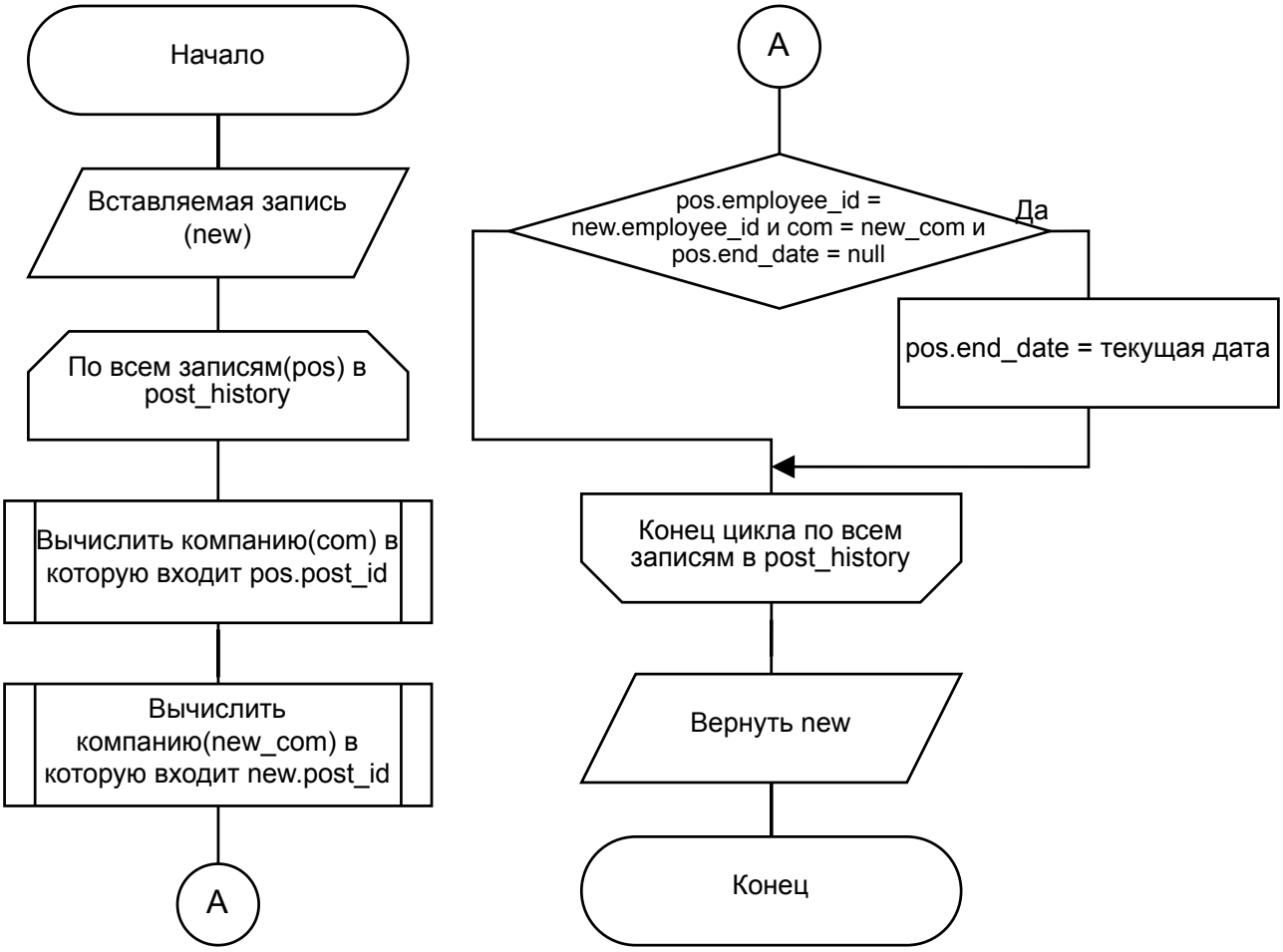

Рисунок 2.9 — Схема триггера before insert для таблицы post_history.

2.4 Используемые функции

Для упрощения доступа к данным и реализации row security policies в базе данных будут реализованы дополнительные пользовательские функции.

Так как к пользователь-сотрудник имеет доступ только к данным о своих подчиненных, необходимо реализовать функцию, вычисляющую текущих подчиненных сотрудника, по одному из его потенциальных ключей. Для вычисления иерархии подчинения позиций будет использоваться функция get_subordinates_by_id_rls, схема алгоритма которой представлена на рисунке 2.10. Эта функция принимает первичный ключ позиции руководителя и рекурсивно обходит подчиненных по иерархии представленной в таблице position_reduced. Поскольку пользователь-сотрудник не имеет прав на чтение из этой таблицы, а функция будет выполняться при вычислении его row security policies, функция будет отмечена ключевым словом security definer [4], что позволит выполняться ей от имени ее создателя-суперпользователя. Таким образом пользователь-сотрудник не сможет получить доступ к данным других своих сотрудников, кроме своих подчиненных, будучи авторизованным под своей учетной записью.

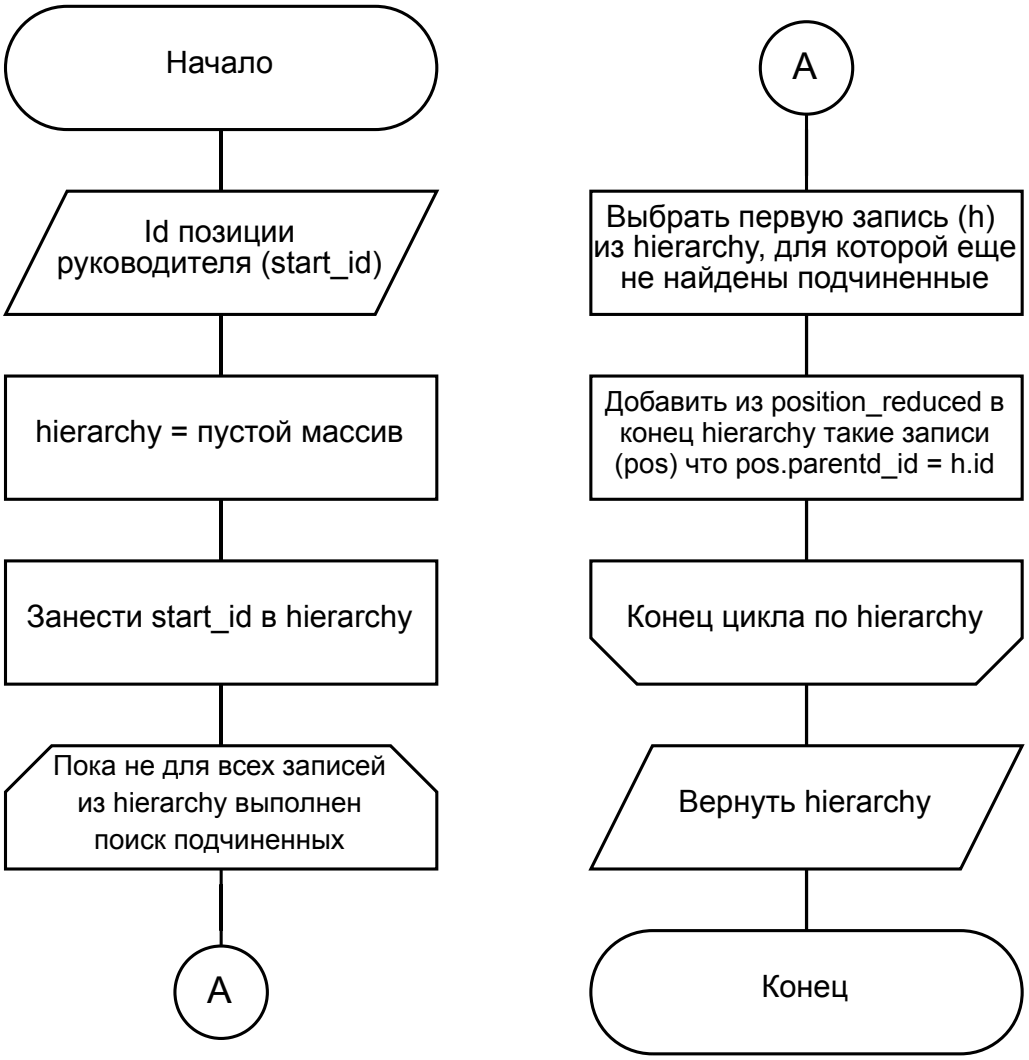

Рисунок 2.10 — Схема алгоритма поиска подчиненных позиций.

Для получения текущих подчиненных сотрудника будет реализована функция get_current_subordinates_id_by_employee_id_rls, схема алгоритма которой представлена на рисунке 2.11. Эта функция также будет отмечена ключевым словом security definer, чтобы использовать таблицы employee_reduced и position_history_reduced.

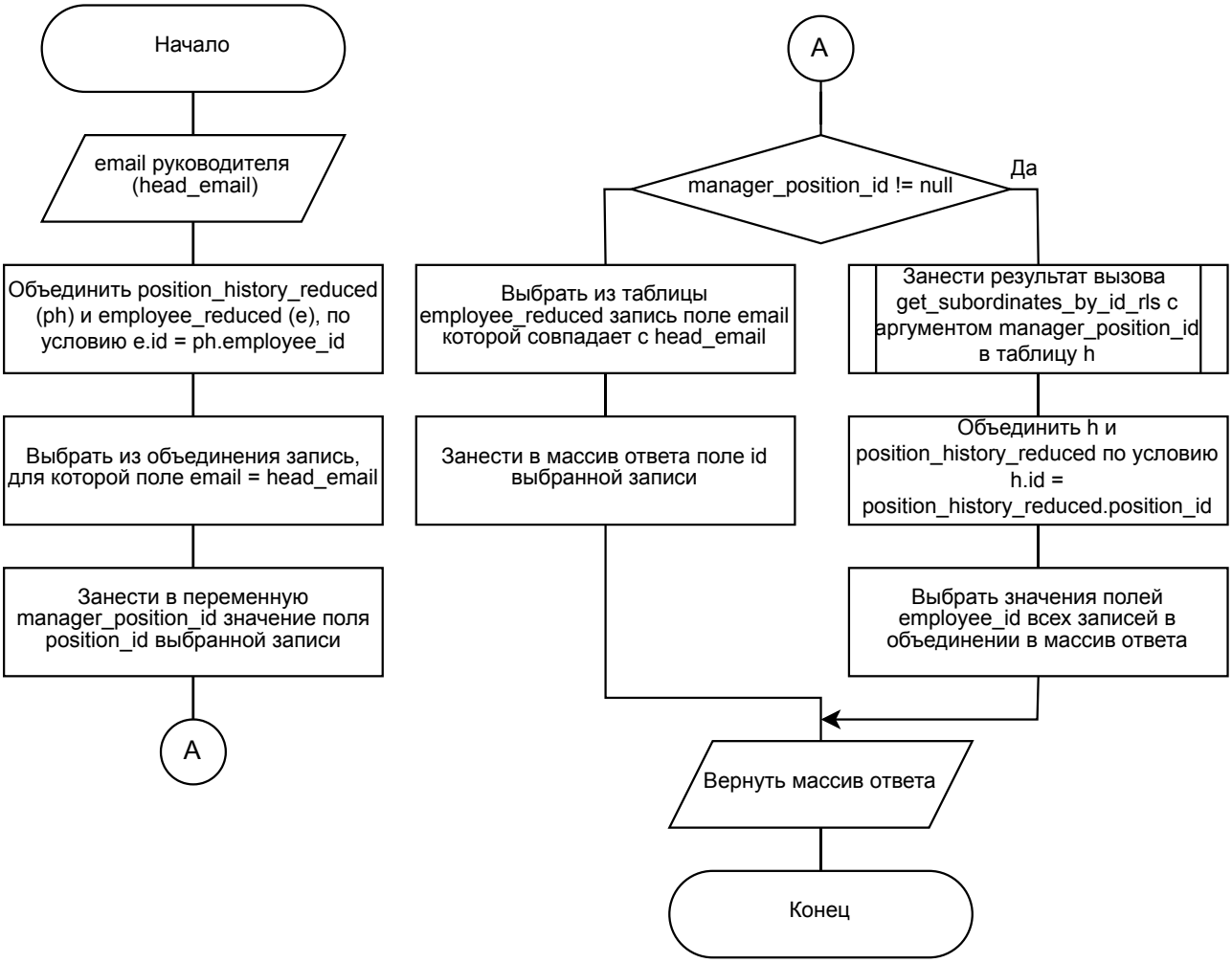

Рисунок 2.11 — Схема алгоритма поиска текущих подчиненных сотрудников.

Пользователь-сотрудник может использовать две функции, идентичные предыдущим, однако не отмеченные security definer и обращающиеся напрямую к таблицами employee, position, position_history. С использованием этих функций сотрудник может получать иерархии позиций или иерархии своих подчиненных, при этом он не сможет получить или изменить данные других сотрудников.

На рисунке 2.12 приведена схема алгоритма перемещения поддерева иерархии, корнем которой является обновляемая позиция, реализованного отдельной процедурой. Данная процедура использует параметр session_replication_role [3]: в начале выполнения функции параметр устанавливается в значение replica, что позволяет использовать операцию UPDATE к таблице position в данной сессии, в конце, значение возвращается к значению по умолчанию origin.

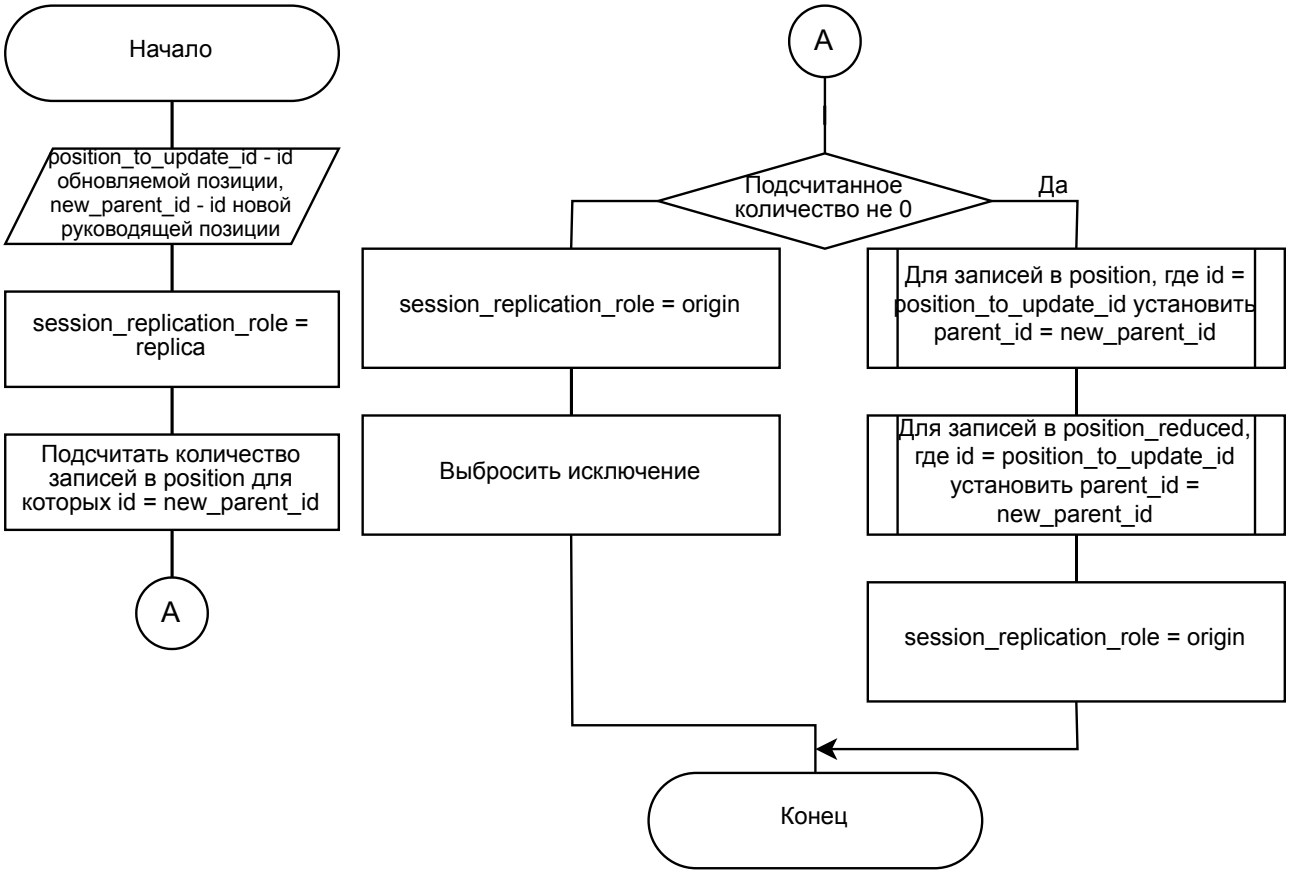

Рисунок 2.12 — Схема алгоритма перемещения поддерева иерархии.

На рисунке 2.13 приведена схема алгоритма перемещения вершины соответствующей обновляемой позиции, при этом подчиненные этой позиции становятся подчиненными руководителя этой позиции. Процедура использующая этот алгоритм, также изменяет значение параметра session_replication_role.

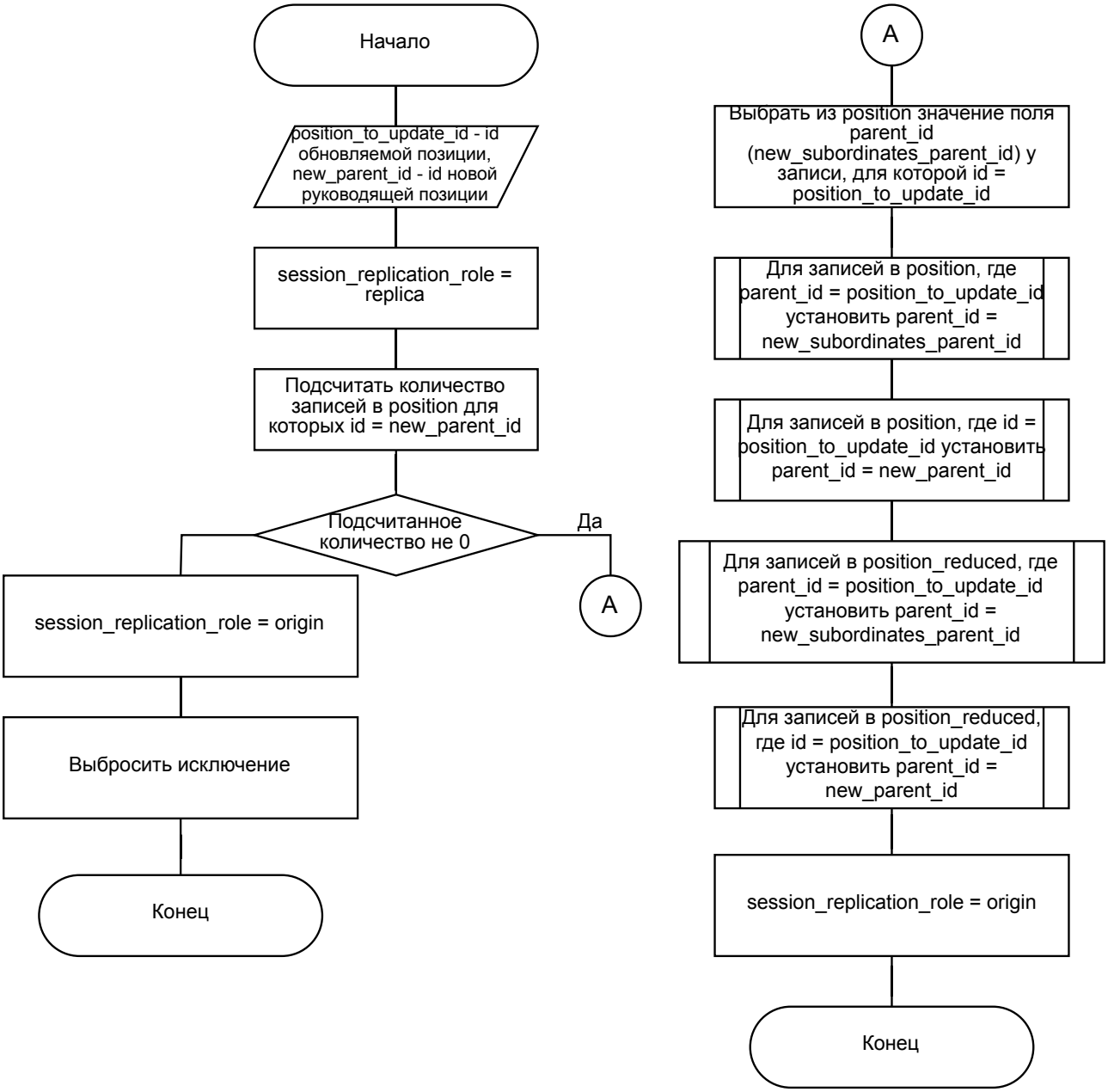

Рисунок 2.13 — Схема алгоритма перемещения одной вершины иерархии.

2.5 Диаграмма классов приложения

На рисунке 2.14 приведена диаграмма классов приложения, реализующего интерфейс доступа к базе данных, представленный в виде REST API. Для допустпа к данным используется паттерн репозиторий. Авторизация и аутентификация в приложении основываются на ролевой системе реализованной в базе данных.

Рисунок 2.14 — Диаграмма классов приложения

Вывод

В данном разделе были описаны все сущности базы данных и приведена диаграмма базы данных. Были спроектированы необходимые триггеры и хранимые процедуры, приведены соответствующие схемы алгоритмов. Также приведена диаграмма классов приложения реализующего необходимые методы REST API для предоставления доступа к базе данных.

3 Технологическая часть

3.1 Сравнение объектно-реляционных СУБД

Среди объектно-реляционных СУБД наиболее широко выделяются [25] Oracle Database [11], PostgreSQL [12] и DB2 [7].

Для сравнения СУБД были выбраны следующие критерии:

— возможность создания ролей;

— поддержка row level security;

— поддержка механизма sequrity definer или аналогов;

— поддержка работы с JSON полями;

— распространение по свободной лицензии.

В таблице 3.1 приведено сравнение вышеуказанных СУБД по сформулированным критериям.

Таблица 3.1 — Сравнение объектно-реляционных СУБД

	Oracle Database	PostgreSQL	DB2
Создание ролей	+	+	+
Row level security	-	+	+
Sequrity definer	+	+	-
Поддержка JSON	+	+	+
Свободное ПО	-	+	-

По результатам сравнения, в данной работе будет использоваться PostgreSQL, поскольку данная СУБД обеспечивает необходимый инструментарий и является распространяемой по свободной лицензии.

3.2 Выбор языка программирования

Для реализации приложения был выбран язык C# [2] в связи с тем, что он обладает следующими необходимыми качествами и инструментами:

— поддерживает парадигму объектно-ориентированного программирования, что позволяет упростить работу с сущностями, представленными в базе данных, путем определения соответствующих им классов и методов для работы с объектами этих классов;

— поддерживается платформами .NET [10] и ASP.NET [1], предоставляющими необходимый инструментарий для написания веб-приложений, а также библиотеки для работы с различными базами данных, в том числе ролями и пользователями БД.

— данный язык поддерживает ADO.NET Entity Framework [5], который предоставляет необходимый интерфейс для работы с сущностями базы данных, и возможность реали-

зации комплексных запросов к базе данных.

3.3 Создание таблиц базы данных

В листингах 3.1 3.2 3.3 3.4 3.5 3.6 3.7 3.8 представлено создание таблиц базы данных.

Листинг 3.1 — Создание таблиц employee_base и employee_reduced

```
create table if not exists employee_base
(
    id          uuid primary key default gen_random_uuid(),
    full_name   text                not null,
    phone       varchar(16) unique  not null check ( phone ~
↪   '^\+[0-9]{1,3}[0-9]{4,14}$' ),
    email       varchar(255) unique not null check ( email ~
↪   '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$' ),
    birth_date date                 not null check ( birth_date < CURRENT_DATE ),
    photo       text unique       default null,
    duties      jsonb             default null
);

create table if not exists employee_reduced
    (
        id          uuid references employee_base (id) on delete cascade,
        email       varchar(255) unique not null check ( email ~
↪   '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$' )
);
```

Листинг 3.2 — Создание таблицы education

```
create table if not exists education
(
    id              uuid primary key default gen_random_uuid(),
    employee_id     uuid references employee_base (id) on delete cascade,
    institution     text not null,
    education_level text not null check ( education_level in (
                                            'Высшее (бакалавриат)',
                                            'Высшее (магистратура)',
                                            'Высшее (специалитет)',
                                            'Среднее профессиональное (ПКР)',
                                            'Среднее профессиональное (ПССЗ)',
```

```
                                                'Программы переподготовки',
                                                'Курсы повышения квалификации'
        ) ),
    study_field     text not null,
    start_date      date not null check ( start_date < CURRENT_DATE ),
    end_date        date
);
```

Листинг 3.3 — Создание таблицы company

```
create table if not exists company
(
    id                uuid primary key             default gen_random_uuid(),
    title             text unique         not null,
    registration_date date                not null check ( registration_date <=
↪ CURRENT_DATE ),
    phone             varchar(16) unique  not null check ( phone ~
↪ '^\+[0-9]{1,3}[0-9]{4,14}$' ),
    email             varchar(255) unique not null check ( email ~
↪ '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$' ),
    inn               varchar(10) unique  not null check ( inn ~ '^[0-9]{10}$' ),
    kpp               varchar(9) unique   not null check ( kpp ~ '^[0-9]{9}$' ),
    ogrn              varchar(13) unique  not null check ( ogrn ~ '^[0-9]{13}$' ),
    address           text                not null,
    _is_deleted       bool                not null default false
);
```

Листинг 3.4 — Создание таблицы post

```
create table if not exists post
(
    id          uuid primary key        default gen_random_uuid(),
    title       text           not null,
    salary      numeric(10, 2) not null check ( salary > 0 ),
    company_id  uuid references company (id) on delete cascade,
    _is_deleted bool           not null default false
);
```

Листинг 3.5 — Создание таблиц position и position_reduced

```sql
create table if not exists position
(
    id          uuid primary key default gen_random_uuid(),
    parent_id   uuid references position (id),
    title       text not null,
    company_id  uuid references company (id) on delete cascade,
    _is_deleted bool not null    default false
);

create table if not exists position_reduced
(
  id uuid primary key,
  parent_id uuid references position_reduced(id)
);
```

Листинг 3.6 — Создание таблицы post_history

```sql
create table if not exists post_history
(
    post_id     uuid references post (id),
    employee_id uuid references employee_base (id) on delete cascade,
    start_date  date not null check ( start_date < CURRENT_DATE ),
    end_date    date check ( end_date <= CURRENT_DATE )
);
```

Листинг 3.7 — Создание таблиц position_history и position_history_reduced

```sql
create table if not exists position_history
(
    position_id uuid references position (id),
    employee_id uuid references employee_base (id) on delete cascade,
    start_date  date not null check ( start_date < CURRENT_DATE ),
    end_date    date check ( end_date <= current_date )
);

create table if not exists position_history_reduced(
    position_id uuid references position (id),
    employee_id uuid references employee_base(id) on delete cascade
);
```

Листинг 3.8 — Создание таблицы score_story

```
create table if not exists score_story
(
    id               uuid primary key     default gen_random_uuid(),
    employee_id      uuid references employee_base (id) on delete cascade,
    author_id        uuid        references employee_base (id) on delete set null,
    position_id      uuid references position (id),
    created_at       timestamptz not null default now(),
    efficiency_score int check ( efficiency_score > 0 and efficiency_score < 6 ),
    engagement_score int check ( engagement_score > 0 and engagement_score < 6 ),
    competency_score int check ( competency_score > 0 and competency_score < 6 )
);
```

3.4 Создание триггеров

Ниже приведены листинги создания триггеров:

— листинг 3.9 — триггер, запрещающий прямое обновление поля parent_id в таблице position;

— листинг 3.10 — триггер, проверяющий частоту выставления оценок;

— листинг 3.11 — триггер, обновляющий флаг _is_deleted и запрещающий прямое удаление записей в таблице company;

— листинг 3.12 — триггер, триггер, обновляющий флаг _is_deleted и запрещающий прямое удаление записей в таблице post;

— листинг 3.13 — триггер, обновляющий флаг _is_deleted и запрещающий прямое удаление записей в таблице position;

— листинг 3.14 — триггер, каскадом обновляющий данные в таблице position_history_reduced при обновлении данных в position_history;

— листинг 3.15 — триггер, устанавливающий текущую дату в поле end_date при вставке новой записи о работе сотрудника на новой должности в таблицу post_history.

Листинг 3.9 — Создание триггера protect_column_parent_id_trigger.

```
create or replace function protect_column_parent_id()
    returns trigger
    language plpgsql
as
$$
begin
```

```
7     if old.parent_id is distinct from new.parent_id and
8        (select count(*) from position where parent_id = old.id) > 0 then
9         raise exception 'Parent id cannot be changed directly for employees with
  ↪   subordinates, use the special functions:
  ↪   change_parent_id_with_subordinates(uuid, uuid),
  ↪   change_parent_id_without_subordinates(uuid, uuid)';
10    end if;
11    return new;
12 end;
13 $$;
14
15 create trigger protect_column_parent_id_trigger
16    before update
17    on position
18    for each row
19 execute function protect_column_parent_id();
```

Листинг 3.10 — Создание триггера check_scores_frequency_trigger.

```
1  create or replace function check_scores_frequency()
2     returns trigger
3     language plpgsql
4  as
5  $$
6  begin
7     if new.created_at - (select max(score_story.created_at)
8                          from score_story
9                          where employee_id = new.employee_id
10                           and position_id = new.position_id) < interval '1 month'
   ↪   then
11        raise exception 'Scores frequency must be at least 1 month';
12    end if;
13    return new;
14 end;
15 $$;
16
17 create trigger check_scores_frequency_trigger
18    before insert or update
19    on score_story
20    for each row
```

```
execute function check_scores_frequency();
```

Листинг 3.11 — Создание триггера soft_delete_company_trigger.

```
create or replace function soft_delete_company()
    returns trigger
    language plpgsql
as
$$
begin
    update company set _is_deleted = true where id = old.id;
    update post set _is_deleted= true where company_id = old.id;
    update position set _is_deleted = true where company_id = old.id;
    delete from position_reduced where id in (select id from position where
    company_id = old.id);
    delete from position_history_reduced where position_id in (select id from
    position where company_id = old.id);
    update post_history set end_date=current_date where post_id in (select id from
    post where company_id = old.id);
    update position_history
    set end_date=current_date
    where position_id in (select id from position where company_id = old.id);
    return null;
end;
$$;

create trigger soft_delete_company_trigger
    before delete
    on company
    for each row
execute function soft_delete_company();
```

Листинг 3.12 — Создание триггера soft_delete_post_trigger.

```
create or replace function soft_delete_post()
    returns trigger
    language plpgsql
as
$$
begin
```

```
    update post set _is_deleted = true where id = old.id;
    update post_history set end_date = CURRENT_DATE where post_id = old.id;
    return null;
end;
$$;

create trigger soft_delete_post_trigger
    before delete
    on post
    for each row
execute function soft_delete_post();
```

Листинг 3.13 — Создание триггера soft_delete_position_trigger.

```
create or replace function soft_delete_position()
    returns trigger
    language plpgsql
as
$$
begin
    if (select parent_id from position where id = old.id) is null then
        raise 'Unable to delete chief position';
    end if;
    set session_replication_role = replica;
    update position set parent_id = old.parent_id where parent_id = old.id;
    update position_reduced set parent_id = old.parent_id where parent_id =
↪   old.id;
    set session_replication_role = origin;
    update position set _is_deleted = true where id = old.id;
    update position_history set end_date = current_date where position_id = old.id
↪   and end_date is null;
    delete from position_history_reduced where position_id = old.id;
    return null;
end;
$$;

create trigger soft_delete_position_trigger
    before delete
    on position
    for each row
```

```
execute function soft_delete_position();
```

Листинг 3.14 — Создание триггера close_previous_position_update.

```
create or replace function close_previous_position_update() returns trigger
    language plpgsql as
$$
begin
    if (old.end_date is null and new.end_date is not null) then
        delete
        from position_history_reduced
        where employee_id = old.employee_id
          and position_id = old.position_id;
    else
        update position_history_reduced
        set employee_id = new.employee_id,
            position_id = new.position_id
        where employee_id = old.employee_id
          and position_id = old.position_id;
    end if;
    return new;
end;
$$;

create trigger close_previous_position_update
    after update
    on position_history
    for each row
execute function close_previous_position_update();
```

Листинг 3.15 — Создание триггера close_previous_post_insert.

```
create or replace function close_previous_post_insert() returns trigger
    language plpgsql as
$$
begin
    update post_history
    set end_date=CURRENT_DATE
    where employee_id = new.employee_id
      and end_date is null
```

```
      and (select company_id from post where post_id = post.id) =
          (select company_id from post where post.id = new.post_id);
    return new;
end;
$$;


create trigger close_previous_post_insert
    before insert
    on post_history
    for each row
execute function close_previous_post_insert();
```

3.5 Создание хранимых процедур и функций

Ниже приведены листинги хранимых процедур, описанных в конструкторском разделе:

— листинг 3.16 — функция поиска всех позиции находящихся ниже данной в иерархии для вычисления row security policies;

— листинг 3.17 — функция поиска всех текущих подчиненных сотрудников по email руководителя для вычисления row security policies;

— листинг 3.18 — функция реализующая перевод позиции в иерархии вместе с подчиненными;

— листинг 3.19 — функция реализующая перевод позиции в иерархии без подчиненных. Подчиненные переведенной позиции становятся подчиненными руководителя этой позиции.

Листинг 3.16 — Функция поиска подчиненных позиций по первичному ключу руководящей позиции.

```
create or replace function get_subordinates_by_id_rls(start_id uuid)
    returns table
            (
                id uuid
            )
    security definer
AS
$$
begin
    return query
        with recursive hierarchy as (select p.id
```

```
12                                      from position_reduced as p
13                                      where p.id = start_id
14
15                                      union all
16
17                                      select p.id
18                                      from position_reduced p
19                                               join hierarchy as h on p.parent_id =
    ↪ h.id)
20          SELECT *
21          from hierarchy;
22  END;
23  $$ LANGUAGE plpgsql;
```

Листинг 3.17 — Функция поиска текущих подчиненных сотрудников по email руководящего сотрудника.

```
1   create or replace function
    ↪ get_current_subordinates_id_by_employee_id_rls(head_employee_email text)
2       returns table
3               (
4                   employee_id uuid
5               )
6       security definer
7   AS
8   $$
9   declare
10      manager_position_id uuid;
11  begin
12      select position_history_reduced.position_id
13      into manager_position_id
14      from employee_reduced
15               join position_history_reduced on employee_reduced.id =
    ↪ position_history_reduced.employee_id
16      where employee_reduced.email = head_employee_email;
17
18      if manager_position_id is null then
19          return query (select id from employee_reduced where email =
    ↪ head_employee_email);
20      end if;
21
```

```
22      return query (select ph.employee_id
23                    from (select position_history_reduced.employee_id,
    ↪   position_history_reduced.position_id
24                          from position_history_reduced) as ph
25                             join (select * from
    ↪   get_subordinates_by_id_rls(manager_position_id)) as h
26                                  on h.id = ph.position_id);
27  end;
28  $$ LANGUAGE plpgsql;
```

Листинг 3.18 — Функция для перемещения позиции с подчиненными в иерархии позиций.

```
1   create or replace function
    ↪   change_parent_id_with_subordinates(position_to_update_id uuid, new_parent_id
    ↪   uuid)
2       returns void
3       language plpgsql
4       security definer
5   as
6   $$
7   begin
8       set session_replication_role = replica;
9       if ((select count(*) from position where id = new_parent_id) = 0) then
10          set session_replication_role  = origin;
11          raise exception 'New parent id is not an employee';
12      else
13          update position set parent_id = new_parent_id where id =
    ↪   position_to_update_id;
14          update position_reduced set parent_id = new_parent_id where id =
    ↪   position_to_update_id;
15      end if;
16      set session_replication_role = origin;
17  end;
18  $$;
```

Листинг 3.19 — Функция для перемещения позиции с подчиненными в иерархии позиций.

```
1   create or replace function
    ↪   change_parent_id_without_subordinates(position_to_update_id uuid,
    ↪   new_parent_id uuid)
2       returns void
```

```
    language plpgsql
    security definer
as
$$
begin
    set session_replication_role = replica;
    if ((select count(*) from position where id = new_parent_id) = 0) then
        set session_replication_role = origin;
        raise exception 'New parent id is not an employee';
    else
        update position
        set parent_id = (select parent_id from position where id =
↪   position_to_update_id)
        where parent_id = position_to_update_id;
        update position set parent_id = new_parent_id where id =
↪   position_to_update_id;

        update position_reduced
        set parent_id = (select parent_id from position where id =
↪   position_to_update_id)
        where parent_id = position_to_update_id;
        update position_reduced set parent_id = new_parent_id where id =
↪   position_to_update_id;
    end if;
    set session_replication_role = origin;
end;
$$;
```

3.6 Создание ролевой модели

В соответствии с описанием в конструкторском разделе, в листинге 3.20 приведено создание ролевой модели.

Листинг 3.20 — Создание ролевой модели.

```
create role admin_user with
    login
    password 'admin_pass'
    superuser
    createdb
    createrole
```

```
    replication
    bypassrls;

create role guest_user with
    login
    password 'guest_pass';
grant select on company, post, position, public.users to guest_user;
grant execute on function get_subordinates_by_id to guest_user;

create role employee_user with
    login password 'employee_pass';
grant select on all tables in schema public to employee_user;
revoke select on employee_reduced, position_reduced, position_history_reduced from
↪  employee_user;
grant insert, update on score_story to employee_user;
grant execute on function get_subordinates_by_position to employee_user;
grant execute on function get_subordinates_by_id to employee_user;
grant execute on function get_current_subordinates_id_by_employee_id to
↪  employee_user;
grant execute on function get_current_subordinates_id_by_employee_id_rls to
↪  employee_user;
grant execute on function get_subordinates_by_id_rls to employee_user;

create policy employee_reading_policy on position_history
    for select
    to employee_user
    using (employee_id in(select * from
↪  get_current_subordinates_id_by_employee_id_rls((SELECT
↪  current_setting('app.current_user_name')))));

create policy employee_reading_policy on post_history
    for select
    to employee_user
    using (employee_id in(select * from
↪  get_current_subordinates_id_by_employee_id_rls((SELECT
↪  current_setting('app.current_user_name')))));

create policy employee_reading_policy on education
    for select
    to employee_user
```

```
    using (employee_id in(select * from
↪   get_current_subordinates_id_by_employee_id_rls((SELECT
↪   current_setting('app.current_user_name')))));

create policy score_reading_policy on score_story
    for select
    to employee_user
    using (employee_id in(select * from
↪   get_current_subordinates_id_by_employee_id_rls((SELECT
↪   current_setting('app.current_user_name')))));

create policy employee_reading_policy on employee_base
    for select
    to employee_user
    using (id in(select * from
↪   get_current_subordinates_id_by_employee_id_rls((SELECT
↪   current_setting('app.current_user_name')))));
```

3.7 Реализация доступа к данным в приложении

Для доступа к данным из приложения, для каждой сущности реализован собственный репозиторий, использующий Entity Framework Core. Также реализован общий контекст базы данных, представленный классом ProjectDbContext, осуществляющий сопоставление сущностей приложения с сущностями базы данных.

В листинге 3.21 приведен листинг метода репозитория сущности position, вызывающего функцию реализованную в базе данных.

Листинг 3.21 — Метод класса PositionRepository, использующий функцию change_parent_id_with_subordinates.

```
public async Task<BasePosition>
↪   UpdatePositionParentWithSubordinatesAsync(UpdatePosition position)
{
    try{
        var positionDb = await _context.PositionDb
            .FirstOrDefaultAsync(p => p.Id == position.Id && p.CompanyId ==
↪   position.CompanyId);
        if (positionDb is null)
        {
            _logger.LogWarning("Position with id {Id} not found for update",
↪   position.Id);
```

```
9               throw new PositionNotFoundException($"Position with id {position.Id}
    ↪ not found");
10          }
11          var existingPosition = await _context.PositionDb
12              .Where(p => p.Id != position.Id &&
13                          p.CompanyId == position.CompanyId &&
14                          p.Title == position.Title)
15              .FirstOrDefaultAsync();
16          if (existingPosition is not null)
17          {
18              _logger.LogWarning("Position with title {Title} already exists in
    ↪ company {CompanyId}", position.Title,
19                  position.CompanyId);
20              throw new PositionAlreadyExistsException(
21                  $"Position with title {position.Title} already exists in company
    ↪ {position.CompanyId}");
22          }
23          await _context.Database.ExecuteSqlAsync(
24              $"select change_parent_id_with_subordinates({position.Id},
    ↪ {position.ParentId})");
25          positionDb.Title = position.Title ?? positionDb.Title;
26          await _context.SaveChangesAsync();
27          _logger.LogInformation("Position with id {Id} was updated", position.Id);
28          return PositionConverter.Convert(positionDb)!;
29      }
30      catch (Exception e)
31      {
32          _logger.LogError(e, "Error occurred while updating position with id {Id}",
    ↪ position.Id);
33          throw;
34      }
35  }
```

В листинге 3.22 приведен листинг метода onModelCreating класса ProjectDbContext, в котором производится регистрация конфигураций сущностей, а также функций, реализованных в базе данных. В частности в листинге показана регистрация функций get_current_subordinates_-id_by_employee_id и get_subordinates_by_id

Листинг 3.22 — Метод onModelCreating и объявление функций реализованных в БД.

```
1  protected override void OnModelCreating(ModelBuilder modelBuilder){
2          base.OnModelCreating(modelBuilder);
```

```
        modelBuilder.HasDbFunction(
            typeof(ProjectDbContext)
                .GetMethod(nameof(GetCurrentSubordinatesIdByEmployeeId))!);
        modelBuilder.HasDbFunction(() =>
    ↪ GetCurrentSubordinatesIdByEmployeeId(Guid.Empty));
        modelBuilder.HasDbFunction(
            typeof(ProjectDbContext)
                .GetMethod(nameof(GetSubordinatesById))!);
        modelBuilder.HasDbFunction(()=> GetSubordinatesById(Guid.Empty))
        modelBuilder.ApplyConfiguration(new EmployeeDbConfiguration());
        modelBuilder.ApplyConfiguration(new CompanyDbConfiguration());
        modelBuilder.ApplyConfiguration(new EducationDbConfiguration());
        modelBuilder.ApplyConfiguration(new PostDbConfiguration());
        modelBuilder.ApplyConfiguration(new PositionDbConfiguration());
        modelBuilder.ApplyConfiguration(new PostHistoryDbConfiguration());
        modelBuilder.ApplyConfiguration(new ScoreDbConfiguration());
        modelBuilder.ApplyConfiguration(new PositionHistoryDbConfiguration());
        modelBuilder.ApplyConfiguration(new UserDbConfiguration());
    }
    [DbFunction(Name = "get_current_subordinates_id_by_employee_id", Schema =
    ↪ "public")]
    public IQueryable<PositionHierarchyWithEmployeeIdDb>
    ↪ GetCurrentSubordinatesIdByEmployeeId(Guid startId){
        return Set<PositionHierarchyWithEmployeeIdDb>()
            .FromSqlRaw("SELECT * FROM
    ↪ get_current_subordinates_id_by_employee_id({0})", startId);
    }
    [DbFunction(Name = "get_subordinates_by_id", Schema = "public")]
    public IQueryable<PositionHierarchyDb> GetSubordinatesById(Guid startId){
        return Set<PositionHierarchyDb>()
            .FromSqlRaw("SELECT * FROM get_subordinates_by_id({0})", startId);
    }
```

Функции объявленные в ProjectDbContext используются как методы этого класса, например, в классе PositionHistoryRepository для получения текущих подчиненных сотрудника. В листинге 3.23 приведен метод, использующий эти функции как методы класса.

Листинг 3.23 — Метод использующий функции БД, как методы класса ProjectDbContext.

```
public async Task<PositionHierarchyWithEmployeePage> GetCurrentSubordinatesAsync(
        Guid managerId,
        int pageNumber,
```

```
        int pageSize)
{
    try
    {
        _logger.LogInformation(
            "Getting current subordinates position history for manager
↪   {ManagerId}, page {PageNumber}, size {PageSize}",
            managerId, pageNumber, pageSize);

        var query = _context.GetCurrentSubordinatesIdByEmployeeId(managerId)
            .OrderBy(x => x.Level)
            .ThenBy(x => x.Title);
        var totalCount = await query.CountAsync();
        var totalPages = (int)Math.Ceiling(totalCount / (double)pageSize);

        var items = await query
            .Skip((pageNumber - 1) * pageSize)
            .Take(pageSize)
            .Select(x => PositionHierarchyWithEmployeeIdConverter.Convert(x)!)
            .ToListAsync();

        _logger.LogInformation(
            "Successfully retrieved {Count} current subordinates position history
↪   records for manager {ManagerId}",
            items.Count, managerId);

        return new PositionHierarchyWithEmployeePage(items, new Page(pageNumber,
↪   totalPages, totalCount));
    }
    catch (Exception ex)
    {
        _logger.LogError(ex,
            "Error getting current subordinates position history for manager
↪   {ManagerId}",
            managerId);
        throw;
    }
}
```

Для вычисления row security policies используется параметр конфигурации app.current_user_name, в котором записан email текущего пользователя, если последний принадлежит роли

employee_user. Для установки этого параметра при открытии соединения и начале сессии написан класс RlsInterceptor, представленный в листинге 3.24.

Листинг 3.24 — Класс, устанавливающий значение параметра конфигурации при открытии соединения.

```
public class RlsInterceptor : DbConnectionInterceptor
{
    private readonly string _userId;
    public RlsInterceptor(string userId)
    {
        _userId = userId;
    }
    public override async Task ConnectionOpenedAsync(
        DbConnection connection,
        ConnectionEndEventData eventData,
        CancellationToken cancellationToken = default)
    {
        await base.ConnectionOpenedAsync(connection, eventData,
↪   cancellationToken);
        await using var cmd = connection.CreateCommand();
        cmd.CommandText = $"SET app.current_user_name = '{_userId}';";
        await cmd.ExecuteNonQueryAsync(cancellationToken);
    }
}
```

3.8 Тестирование триггеров и функций

Ниже приведены тестовые случаи для перечисленных выше триггеров и процедур. Тест разделены на классы эквивалентности, для каждого из которых приведено по одному тесту.

3.8.1 Триггер protect_column_parent_id_trigger

1) У позиции нет подчиненных, поле parent_id изменяется.

Тестовый сценарий:

```
update position set parent_id = '0ae9589e-0004-4000-8000-000000000004' where
↪   id = '0ae9589e-0006-4000-8000-000000000006';
select id, parent_id from position where id =
↪   '0ae9589e-0006-4000-8000-000000000006';
```

Ожидаемые выходные данные:

id	parent_id
0ae9589e-0006-4000-8000-000000000006	0ae9589e-0004-4000-8000-000000000004

Полученные выходные данные:

id	parent_id
0ae9589e-0006-4000-8000-000000000006	0ae9589e-0004-4000-8000-000000000004

2) У позиции есть подчиненные, поле parent_id изменяется.

Тестовый сценарий:

```
update position set parent_id = '0ae9589e-0003-4000-8000-000000000003' where
↪  id = '0ae9589e-0005-4000-8000-000000000005';
```

Ожидаемые выходные данные:

[P0001] ERROR: Parent id cannot be changed directly for employees with subordinates, use the special functions: change_parent_id_with_subordinates(uuid, uuid), change_parent_id_-without_subordinates(uuid, uuid)

Полученные выходные данные:

[P0001] ERROR: Parent id cannot be changed directly for employees with subordinates, use the special functions: change_parent_id_with_subordinates(uuid, uuid), change_parent_id_-without_subordinates(uuid, uuid)

3) Поле parent_id не изменяется.

Тестовый сценарий:

```
update position set title = 'Старший специалист по безопасности 1' where id
↪  = '0ae9589e-0005-4000-8000-000000000005';
select title from position where id =
↪  '0ae9589e-0005-4000-8000-000000000005';
```

Ожидаемые выходные данные:

title
Старший специалист по безопасности 1

Полученные выходные данные:

title
Старший специалист по безопасности 1

3.8.2 Триггер check_scores_frequency_trigger

1) Вставка в таблицу score_story, с момента последней оценки прошло не меньше месяца.

Тестовый сценарий:

```
1 insert into score_story(employee_id, author_id, position_id, created_at,
  ↪ efficiency_score, engagement_score, competency_score) values
  ↪ ('4b5377fa-cb51-4954-8552-54acb626c108',
  ↪ 'ad5887f2-0bec-4c4a-b8e0-318fb303d99e',
  ↪ '0ae9589e-0004-4000-8000-000000000004', now() - interval '2 month', 5,
  ↪ 4, 5);
2 insert into score_story(employee_id, author_id, position_id, created_at,
  ↪ efficiency_score, engagement_score, competency_score) values
  ↪ ('4b5377fa-cb51-4954-8552-54acb626c108',
  ↪ 'ad5887f2-0bec-4c4a-b8e0-318fb303d99e',
  ↪ '0ae9589e-0004-4000-8000-000000000004', now() - interval '1 month', 5,
  ↪ 4, 5);
3 select count(*) from score_story where employee_id =
  ↪ '4b5377fa-cb51-4954-8552-54acb626c108' and position_id =
  ↪ '0ae9589e-0004-4000-8000-000000000004';
```

Ожидаемые выходные данные:

count
2

Полученные выходные данные:

count
2

2) Вставка в таблицу score_story, с момента последней оценки прошло меньше месяца.

Тестовый сценарий:

```
1 insert into score_story(employee_id, author_id, position_id, created_at,
  ↪ efficiency_score, engagement_score, competency_score) values
  ↪ ('4b5377fa-cb51-4954-8552-54acb626c108',
  ↪ 'ad5887f2-0bec-4c4a-b8e0-318fb303d99e',
  ↪ '0ae9589e-0004-4000-8000-000000000004', now() - interval '10 day', 5, 4,
  ↪ 5);
```

Ожидаемые выходные данные:

[P0001] ERROR: Scores frequency must be at least 1 month

Полученные выходные данные:

[P0001] ERROR: Scores frequency must be at least 1 month

3.8.3 Триггер soft_delete_company_trigger

В случае данного триггера класс эквивалентности один: удаление существующей компании.

Тестовый сценарий:

```
1 delete from company where id = '0ae9589e-b2e8-4d26-b1b6-9decdd7ab5af';
2 select id, _is_deleted from company where id =
  ↪ '0ae9589e-b2e8-4d26-b1b6-9decdd7ab5af';
3 select id, position._is_deleted from position where company_id =
  ↪ '0ae9589e-b2e8-4d26-b1b6-9decdd7ab5af';
4 select id, _is_deleted from post where company_id =
  ↪ '0ae9589e-b2e8-4d26-b1b6-9decdd7ab5af';
5 select count(*) from position_reduced where id in (select id from position where
  ↪ company_id='0ae9589e-b2e8-4d26-b1b6-9decdd7ab5af');
6 select count(*) from public.position_history_reduced where position_id in (select
  ↪ id from position where company_id='0ae9589e-b2e8-4d26-b1b6-9decdd7ab5af');
7 select count(*) from position_history where end_date is null and position_id in
  ↪ (select id from position where
  ↪ company_id='0ae9589e-b2e8-4d26-b1b6-9decdd7ab5af');
8 select count(*) from post_history where end_date is null and post_id in (select id
  ↪ from post where company_id='0ae9589e-b2e8-4d26-b1b6-9decdd7ab5af');
```

Ожидаемые выходные данные:

id	_is_deleted
0ae9589e-b2e8-4d26-b1b6-9decdd7ab5af	true

id	_is_deleted
0ae9589e-0001-4000-8000-000000000001	true
0ae9589e-0002-4000-8000-000000000002	true
0ae9589e-0003-4000-8000-000000000003	true
0ae9589e-0007-4000-8000-000000000007	true
0ae9589e-0008-4000-8000-000000000008	true
0ae9589e-0009-4000-8000-000000000009	true
0ae9589e-000a-4000-8000-00000000000a	true
0ae9589e-0004-4000-8000-000000000004	true
0ae9589e-0006-4000-8000-000000000006	true
0ae9589e-0005-4000-8000-000000000005	true

id	_is_deleted
6e49bd73-a683-4c1e-81a8-e2d3368b0d95	true
11e89ab9-d8ee-48d2-8c34-ebff24cdf4f5	true
5f18d34e-7bee-4f49-8c44-0eaca8bb1557	true
b992b65b-ca2f-487a-ac8a-cc0aa7737d63	true
96fc47f4-d8aa-4fb5-9089-0b7ee39bfe90	true
0cc0fc8a-4185-434f-b384-ddd509ce8539	true
3155a63f-5871-47f6-8c83-2811dc20405e	true
984035b8-a9a7-45f7-88a2-2f46fd4ec9a6	true

count
0

count
0

count
0

count
0

Полученные выходные данные:

id	_is_deleted
0ae9589e-b2e8-4d26-b1b6-9decdd7ab5af	true

id	_is_deleted
0ae9589e-0001-4000-8000-000000000001	true
0ae9589e-0002-4000-8000-000000000002	true
0ae9589e-0003-4000-8000-000000000003	true
0ae9589e-0007-4000-8000-000000000007	true
0ae9589e-0008-4000-8000-000000000008	true
0ae9589e-0009-4000-8000-000000000009	true
0ae9589e-000a-4000-8000-00000000000a	true
0ae9589e-0004-4000-8000-000000000004	true
0ae9589e-0006-4000-8000-000000000006	true
0ae9589e-0005-4000-8000-000000000005	true

id	_is_deleted
6e49bd73-a683-4c1e-81a8-e2d3368b0d95	true
11e89ab9-d8ee-48d2-8c34-ebff24cdf4f5	true
5f18d34e-7bee-4f49-8c44-0eaca8bb1557	true
b992b65b-ca2f-487a-ac8a-cc0aa7737d63	true
96fc47f4-d8aa-4fb5-9089-0b7ee39bfe90	true
0cc0fc8a-4185-434f-b384-ddd509ce8539	true
3155a63f-5871-47f6-8c83-2811dc20405e	true
984035b8-a9a7-45f7-88a2-2f46fd4ec9a6	true

count
0

count
0

count
0

count
0

3.8.4 Триггер soft_delete_post_trigger

В случае данного триггера класс эквивалентности также один: удаление существующей должности.

Тестовый сценарий:

```
1 delete from post where id = '6e49bd73-a683-4c1e-81a8-e2d3368b0d95';
2 select id, post._is_deleted from post where id =
  ↪ '6e49bd73-a683-4c1e-81a8-e2d3368b0d95';
3 select count(*) from post_history where
  ↪ post_id='6e49bd73-a683-4c1e-81a8-e2d3368b0d95' and end_date is null;
```

Ожидаемые выходные данные:

id	_is_deleted
6e49bd73-a683-4c1e-81a8-e2d3368b0d95	true

count
0

Полученные выходные данные:

id	_is_deleted
6e49bd73-a683-4c1e-81a8-e2d3368b0d95	true

count
0

3.8.5 Триггер soft_delete_position_trigger

1) Удаление не корневой позиции в иерархии.

Тестовый сценарий:

```
delete from position where id = '0ae9589e-0004-4000-8000-000000000004';
select id, _is_deleted from position where id
↪ ='0ae9589e-0004-4000-8000-000000000004';
select count(*) from position where parent_id =
↪ '0ae9589e-0004-4000-8000-000000000004';
select count(*) from position_reduced where parent_id =
↪ '0ae9589e-0004-4000-8000-000000000004';
select count(*) from position_history_reduced where position_id =
↪ '0ae9589e-0004-4000-8000-000000000004';
select count(*) from position_history where position_id =
↪ '0ae9589e-0004-4000-8000-000000000004' and end_date is null;
```

Ожидаемые выходные данные:

id	_is_deleted
0ae9589e-0004-4000-8000-000000000004	true

count
0

count
0

count
0

count
0

Полученные выходные данные:

id	_is_deleted
0ae9589e-0004-4000-8000-000000000004	true

count
0

count
0

count
0

count
0

2) Удаление корневой позиции в иерархии.

Тестовый сценарий

```
delete from position where id = '0ae9589e-0001-4000-8000-000000000001';
```

Ожидаемые выходные данные:

[P0001] ERROR: Unable to delete chief position

Полученные выходные данные:

[P0001] ERROR: Unable to delete chief position

3.8.6 Триггер close_previous_position_update

1) Обновление даты окончания пребывания на позиции.

Тестовый сценарий:

```
update position_history set end_date = '2025-06-07' where
↪ position_id='0ae9589e-0004-4000-8000-000000000004' and employee_id =
↪ '4b5377fa-cb51-4954-8552-54acb626c108';
select end_date from position_history where
↪ position_id='0ae9589e-0004-4000-8000-000000000004' and employee_id =
↪ '4b5377fa-cb51-4954-8552-54acb626c108';
select count(*) from position_history_reduced where position_id
↪ ='0ae9589e-0004-4000-8000-000000000004' and employee_id =
↪ '4b5377fa-cb51-4954-8552-54acb626c108';
```

Ожидаемые выходные данные:

end_date
2025-06-07

count
0

Полученные выходные данные:

end_date
2025-06-07

count
0

2) Изменение id сотрудника.

Тестовый сценарий:

```
1 update position_history set employee_id =
  ↪ 'bad8a5a0-ec08-412e-8f19-0d9e993d5651' where position_id =
  ↪ 'fa001e78-0008-4000-8000-000000000008' and employee_id =
  ↪ '33dea239-2610-49d9-8f33-748f62f04e74';
2 select count(*) from position_history_reduced where position_id =
  ↪ 'fa001e78-0008-4000-8000-000000000008' and employee_id =
  ↪ '33dea239-2610-49d9-8f33-748f62f04e74';
3 select count(*) from position_history_reduced where position_id =
  ↪ 'fa001e78-0008-4000-8000-000000000008' and employee_id =
  ↪ 'bad8a5a0-ec08-412e-8f19-0d9e993d5651';
```

Ожидаемые выходные данные:

count
0

count
1

Полученные выходные данные:

count
0

count
1

3.8.7 Триггер close_previous_post_insert

1) Вставка новой записи о пребывании на должности.

Тестовый сценарий:

```
1 select post_id, start_date, end_date from post_history where employee_id =
  ↪ '446a4190-0a9a-49db-b205-2099138c15ef';
2 insert into  post_history values ('6e49bd73-a683-4c1e-81a8-e2d3368b0d95',
  ↪ '446a4190-0a9a-49db-b205-2099138c15ef', current_date - interval '1
  ↪ days', null)
3 select post_id, start_date, end_date from post_history where employee_id =
  ↪ '446a4190-0a9a-49db-b205-2099138c15ef';
```

Ожидаемые выходные данные:

5f18d34e-7bee-4f49-8c44-0eaca8bb1557	2015-07-01	2018-06-30
b992b65b-ca2f-487a-ac8a-cc0aa7737d63	2018-07-01	2021-06-30
96fc47f4-d8aa-4fb5-9089-0b7ee39bfe90	2021-07-01	2023-06-30
0cc0fc8a-4185-434f-b384-ddd509ce8539	2023-07-01	null

5f18d34e-7bee-4f49-8c44-0eaca8bb1557	2015-07-01	2018-06-30
b992b65b-ca2f-487a-ac8a-cc0aa7737d63	2018-07-01	2021-06-30
96fc47f4-d8aa-4fb5-9089-0b7ee39bfe90	2021-07-01	2023-06-30
0cc0fc8a-4185-434f-b384-ddd509ce8539	2023-07-01	2025-09-16
6e49bd73-a683-4c1e-81a8-e2d3368b0d95	2025-09-16	null

Полученные выходные данные:

5f18d34e-7bee-4f49-8c44-0eaca8bb1557	2015-07-01	2018-06-30
b992b65b-ca2f-487a-ac8a-cc0aa7737d63	2018-07-01	2021-06-30
96fc47f4-d8aa-4fb5-9089-0b7ee39bfe90	2021-07-01	2023-06-30
0cc0fc8a-4185-434f-b384-ddd509ce8539	2023-07-01	null

5f18d34e-7bee-4f49-8c44-0eaca8bb1557	2015-07-01	2018-06-30
b992b65b-ca2f-487a-ac8a-cc0aa7737d63	2018-07-01	2021-06-30
96fc47f4-d8aa-4fb5-9089-0b7ee39bfe90	2021-07-01	2023-06-30
0cc0fc8a-4185-434f-b384-ddd509ce8539	2023-07-01	2025-09-16
6e49bd73-a683-4c1e-81a8-e2d3368b0d95	2025-09-16	null

3.8.8 Функция get_subordinates_by_id

1) Позиция является корнем в дереве иерархии позиций.

Тестовый сценарий:

```
select id, parent_id, level from
↪ get_subordinates_by_id('0ae9589e-0001-4000-8000-000000000001');
```

Ожидаемые выходные данные:

0ae9589e-0001-4000-8000-000000000001	null	0
0ae9589e-0002-4000-8000-000000000002	0ae9589e-0001-4000-8000-000000000001	1
0ae9589e-000a-4000-8000-00000000000a	0ae9589e-0001-4000-8000-000000000001	1
0ae9589e-0003-4000-8000-000000000003	0ae9589e-0002-4000-8000-000000000002	2
0ae9589e-0007-4000-8000-000000000007	0ae9589e-0003-4000-8000-000000000003	3
0ae9589e-0008-4000-8000-000000000008	0ae9589e-0003-4000-8000-000000000003	3
0ae9589e-0009-4000-8000-000000000009	0ae9589e-0003-4000-8000-000000000003	3
0ae9589e-0004-4000-8000-000000000004	0ae9589e-0003-4000-8000-000000000003	3
0ae9589e-0005-4000-8000-000000000005	0ae9589e-0003-4000-8000-000000000003	3
0ae9589e-0006-4000-8000-000000000006	0ae9589e-0005-4000-8000-000000000005	4

Полученные выходные данные:

0ae9589e-0001-4000-8000-000000000001	null	0
0ae9589e-0002-4000-8000-000000000002	0ae9589e-0001-4000-8000-000000000001	1
0ae9589e-000a-4000-8000-00000000000a	0ae9589e-0001-4000-8000-000000000001	1
0ae9589e-0003-4000-8000-000000000003	0ae9589e-0002-4000-8000-000000000002	2
0ae9589e-0007-4000-8000-000000000007	0ae9589e-0003-4000-8000-000000000003	3
0ae9589e-0008-4000-8000-000000000008	0ae9589e-0003-4000-8000-000000000003	3
0ae9589e-0009-4000-8000-000000000009	0ae9589e-0003-4000-8000-000000000003	3
0ae9589e-0004-4000-8000-000000000004	0ae9589e-0003-4000-8000-000000000003	3
0ae9589e-0005-4000-8000-000000000005	0ae9589e-0003-4000-8000-000000000003	3
0ae9589e-0006-4000-8000-000000000006	0ae9589e-0005-4000-8000-000000000005	4

2) Позиция является листом в дереве иерархии позиций.

Тестовый сценарий:

```
select id, parent_id, level from
↪ get_subordinates_by_id('0ae9589e-0006-4000-8000-000000000006');
```

Ожидаемые выходные данные:

0ae9589e-0006-4000-8000-000000000006	0ae9589e-0005-4000-8000-000000000005	0

Полученные выходные данные:

0ae9589e-0006-4000-8000-000000000006	0ae9589e-0005-4000-8000-000000000005	0

3) Позиция не является ни корнем ни листом в дереве иерархии позиций.

Тестовый сценарий:

```
select id, parent_id, level from
↪ get_subordinates_by_id('0ae9589e-0003-4000-8000-000000000003');
```

Ожидаемые выходные данные:

0ae9589e-0003-4000-8000-000000000003	0ae9589e-0002-4000-8000-000000000002	0
0ae9589e-0007-4000-8000-000000000007	0ae9589e-0003-4000-8000-000000000003	1
0ae9589e-0008-4000-8000-000000000008	0ae9589e-0003-4000-8000-000000000003	1
0ae9589e-0009-4000-8000-000000000009	0ae9589e-0003-4000-8000-000000000003	1
0ae9589e-0004-4000-8000-000000000004	0ae9589e-0003-4000-8000-000000000003	1
0ae9589e-0005-4000-8000-000000000005	0ae9589e-0003-4000-8000-000000000003	1
0ae9589e-0006-4000-8000-000000000006	0ae9589e-0005-4000-8000-000000000005	2

Полученные выходные данные:

0ae9589e-0003-4000-8000-000000000003	0ae9589e-0002-4000-8000-000000000002	0
0ae9589e-0007-4000-8000-000000000007	0ae9589e-0003-4000-8000-000000000003	1
0ae9589e-0008-4000-8000-000000000008	0ae9589e-0003-4000-8000-000000000003	1
0ae9589e-0009-4000-8000-000000000009	0ae9589e-0003-4000-8000-000000000003	1
0ae9589e-0004-4000-8000-000000000004	0ae9589e-0003-4000-8000-000000000003	1
0ae9589e-0005-4000-8000-000000000005	0ae9589e-0003-4000-8000-000000000003	1
0ae9589e-0006-4000-8000-000000000006	0ae9589e-0005-4000-8000-000000000005	2

3.8.9 Функция get_current_subordinates_id_by_employee_id

1) Сотрудник не имеет текущей позиции и подчиненных.

Тестовый сценарий:

```
select * from get_current_subordinates_id_by_employee_id
↪ ('21cc75fe-63ea-43c4-8232-c93ffccd30b0');
```

Ожидаемые выходные данные:

[P0001] ERROR: Employee with id 21cc75fe-63ea-43c4-8232-c93ffccd30b0 not found or has no current position

Полученные выходные данные:

[P0001] ERROR: Employee with id 21cc75fe-63ea-43c4-8232-c93ffccd30b0 not found or has no current position

2) Сотрудник является корнем текущей иерархии позиций.

Тестовый сценарий:

```
select employee_id, position_id, level from
↪ get_current_subordinates_id_by_employee_id
↪ ('d82bbccc-7456-445e-9834-eeffad9d5a0e');
```

Ожидаемые выходные данные:

0ae9589e-0003-4000-8000-000000000003	0ae9589e-0002-4000-8000-000000000002	0
0ae9589e-0007-4000-8000-000000000007	0ae9589e-0003-4000-8000-000000000003	1
0ae9589e-0008-4000-8000-000000000008	0ae9589e-0003-4000-8000-000000000003	1
0ae9589e-0009-4000-8000-000000000009	0ae9589e-0003-4000-8000-000000000003	1
0ae9589e-0004-4000-8000-000000000004	0ae9589e-0003-4000-8000-000000000003	1
0ae9589e-0005-4000-8000-000000000005	0ae9589e-0003-4000-8000-000000000003	1
0ae9589e-0006-4000-8000-000000000006	0ae9589e-0005-4000-8000-000000000005	2

Полученные выходные данные:

0ae9589e-0003-4000-8000-000000000003	0ae9589e-0002-4000-8000-000000000002	0
0ae9589e-0007-4000-8000-000000000007	0ae9589e-0003-4000-8000-000000000003	1
0ae9589e-0008-4000-8000-000000000008	0ae9589e-0003-4000-8000-000000000003	1
0ae9589e-0009-4000-8000-000000000009	0ae9589e-0003-4000-8000-000000000003	1
0ae9589e-0004-4000-8000-000000000004	0ae9589e-0003-4000-8000-000000000003	1
0ae9589e-0005-4000-8000-000000000005	0ae9589e-0003-4000-8000-000000000003	1
0ae9589e-0006-4000-8000-000000000006	0ae9589e-0005-4000-8000-000000000005	2

3) Сотрудник является листом текущей иерархии позиций.

Тестовый сценарий:

```
select employee_id, position_id, level from
↪ get_current_subordinates_id_by_employee_id
↪ ('2c354bc1-5c3d-41b3-b406-4540b3fac852');
```

Ожидаемые выходные данные:

2c354bc1-5c3d-41b3-b406-4540b3fac852	4eae2daf-0007-4000-8000-000000000007	0

Полученные выходные данные:

2c354bc1-5c3d-41b3-b406-4540b3fac852	4eae2daf-0007-4000-8000-000000000007	0

4) Сотрудник не является ни листом, ни корнем текущей иерархии позиций.

Тестовый сценарий:

```
select employee_id, position_id, level from
↪ get_current_subordinates_id_by_employee_id
↪ ('2c354bc1-5c3d-41b3-b406-4540b3fac852');
```

Ожидаемые выходные данные:

ad5887f2-0bec-4c4a-b8e0-318fb303d99e	0ae9589e-0003-4000-8000-000000000003	0
446a4190-0a9a-49db-b205-2099138c15ef	0ae9589e-0007-4000-8000-000000000007	1
6ff7e9d8-3c9f-4d09-82ef-f4f963972f88	0ae9589e-0008-4000-8000-000000000008	1
4b5377fa-cb51-4954-8552-54acb626c108	0ae9589e-0004-4000-8000-000000000004	1
2f0083df-d72c-4c50-a9ea-72c6e742537d	0ae9589e-0006-4000-8000-000000000006	2

Полученные выходные данные:

ad5887f2-0bec-4c4a-b8e0-318fb303d99e	0ae9589e-0003-4000-8000-000000000003	0
446a4190-0a9a-49db-b205-2099138c15ef	0ae9589e-0007-4000-8000-000000000007	1
6ff7e9d8-3c9f-4d09-82ef-f4f963972f88	0ae9589e-0008-4000-8000-000000000008	1
4b5377fa-cb51-4954-8552-54acb626c108	0ae9589e-0004-4000-8000-000000000004	1
2f0083df-d72c-4c50-a9ea-72c6e742537d	0ae9589e-0006-4000-8000-000000000006	2

3.8.10 Функция change_parent_id_with_subordinates

1) Новый руководитель в иерархии позиций не существует.

Тестовый сценарий:

```
select change_parent_id_with_subordinates
    ('0ae9589e-0005-4000-8000-000000000005',
    '0ae9589e-0005-4000-8000-0000000000ab');
```

Ожидаемые выходные данные:

[P0001] ERROR: New parent id is not an employee

Полученные выходные данные:

[P0001] ERROR: New parent id is not an employee

2) Новый руководитель существует в иерархии позиций.

Тестовый сценарий:

```
select change_parent_id_with_subordinates
    ('0ae9589e-0005-4000-8000-000000000005',
    '0ae9589e-0004-4000-8000-000000000004');
select id, parent_id from position where id =
    '0ae9589e-0005-4000-8000-000000000005';
```

Ожидаемые выходные данные:

0ae9589e-0005-4000-8000-000000000005	0ae9589e-0004-4000-8000-000000000004

Полученные выходные данные:

0ae9589e-0005-4000-8000-000000000005	0ae9589e-0004-4000-8000-000000000004

3.8.11 Функция change_parent_id_without_subordinates

1) Новый руководитель не существует в иерархии позиций.

Тестовый сценарий:

```
select change_parent_id_without_subordinates
    ('0ae9589e-0005-4000-8000-000000000005',
    '0ae9589e-0004-4000-8000-0000000000a4');
```

Ожидаемые выходные данные:

[P0001] ERROR: New parent id is not an employee

Полученные выходные данные:

[P0001] ERROR: New parent id is not an employee

2) Новый руководитель существует в иерархии позиций.

Тестовый сценарий:

```
create temp table children as select * from position where parent_id =
    '0ae9589e-0003-4000-8000-000000000003';
select change_parent_id_without_subordinates
    ('0ae9589e-0003-4000-8000-000000000003',
    '0ae9589e-0001-4000-8000-000000000001');
select id, parent_id from position where id =
    '0ae9589e-0003-4000-8000-000000000003' or id in (select id from
    children);
drop table children;
```

Ожидаемые выходные данные:

0ae9589e-0003-4000-8000-000000000003	0ae9589e-0001-4000-8000-000000000001
0ae9589e-0007-4000-8000-000000000007	0ae9589e-0002-4000-8000-000000000002
0ae9589e-0008-4000-8000-000000000008	0ae9589e-0002-4000-8000-000000000002
0ae9589e-0009-4000-8000-000000000009	0ae9589e-0002-4000-8000-000000000002
0ae9589e-0004-4000-8000-000000000004	0ae9589e-0002-4000-8000-000000000002
0ae9589e-0006-4000-8000-000000000006	0ae9589e-0002-4000-8000-000000000002

Полученные выходные данные:

0ae9589e-0003-4000-8000-000000000003	0ae9589e-0001-4000-8000-000000000001
0ae9589e-0007-4000-8000-000000000007	0ae9589e-0002-4000-8000-000000000002
0ae9589e-0008-4000-8000-000000000008	0ae9589e-0002-4000-8000-000000000002
0ae9589e-0009-4000-8000-000000000009	0ae9589e-0002-4000-8000-000000000002
0ae9589e-0004-4000-8000-000000000004	0ae9589e-0002-4000-8000-000000000002
0ae9589e-0006-4000-8000-000000000006	0ae9589e-0002-4000-8000-000000000002

3.9 Интерфейс доступа к базе данных

Приложение представляет API для взаимодействия с базой данных, реализованный с использованием библиотеки Swashbuckle [18]. Реализованные запросы представлены на рисунках 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9.

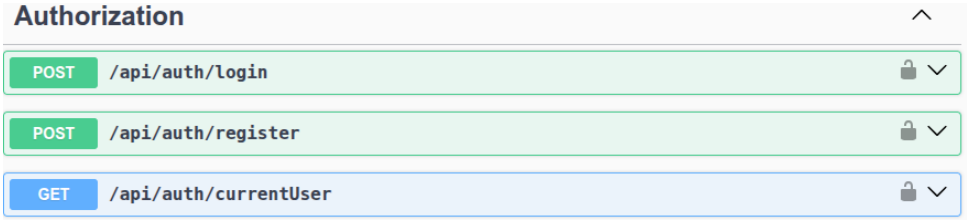

Рисунок 3.1 — Запросы для аутентификации, регистрации и получения данных о текущем пользователе.

Рисунок 3.2 — Запросы для работы с записями о компаниях.

Рисунок 3.3 — Запросы для работы с записями о полученных сотрудниками образованиями.

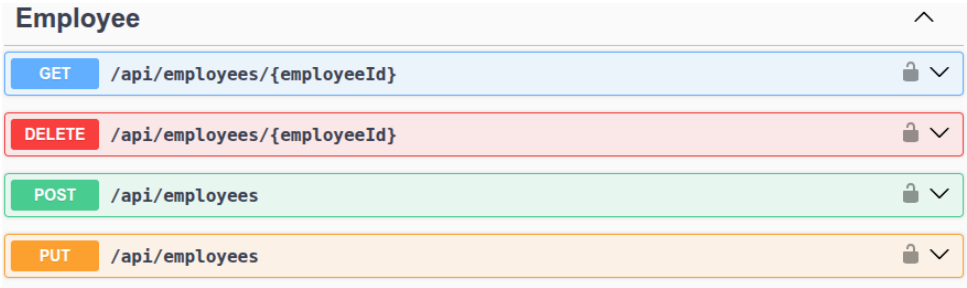

Рисунок 3.4 — Запросы для работы с записями о сотрудниках.

Рисунок 3.5 — Запросы для работы с записями о позициях.

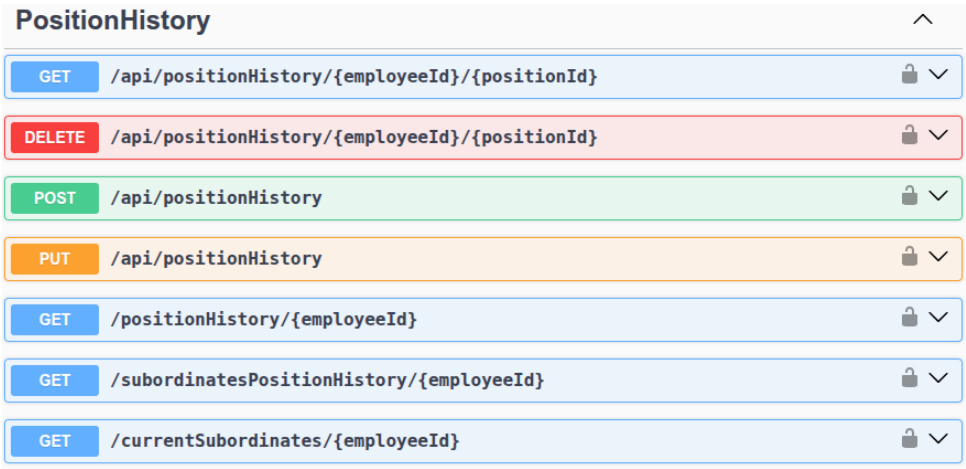

Рисунок 3.6 — Запросы для работы с записями об истории изменения позиций сотрудников.

Рисунок 3.7 — Запросы для работы с записями о должностях.

Рисунок 3.8 — Запросы для работы с записями об истории изменения должностей сотрудников.

Рисунок 3.9 — Запросы для работы с оценками сотрудников.

Вывод

В данном разделе было проведено сравнение объектно-реляционных СУБД и выбрана СУБД для работы с базой данных. Были определены средства реализации, а также созданы

таблицы, триггеры, хранимые процедуры и функции, ролевая модель. Было проведено тестирование триггеров и функций, все тесты были успешно пройдены. Была описана реализация доступа к данным в приложении, а также предоставляемый приложением интерфейс доступа к базе данных.

4 Исследовательская часть

В данном разделе поставлена задача исследования зависимости времени выполнения запроса от количества записей в таблице и количества одновременно выполняемых запросов, а также при наличии кэширования и без него.

В обоих случаях измерялось время выполнения запроса получения информации об образованиях сотрудника. Для реализации кэширования использовалась резидентная СУБД Redis [14].

4.1 Характеристики устройства

Исследования проводились на машине со следующими характеристиками:

— процессор Intel(R) Core(TM) i5-10210U, тактовая частота 1.60 ГГц;

— оперативная память: 16 ГБ;

— операционная система: Ubuntu 22.04.4 LTS.

4.2 Зависимость времени выполнения от количества записей

В таблице 4.1 представлены результаты измерения зависимости времени выполнения запросов от количества записей в таблице education. Представленые данные получены путем измерения времени выполнения 100 запросов 100 раз и последующего усреднения.

Таблица 4.1 — Время выполнения запросов в зависимости от количества записей в таблице

Количество записей в таблице education	Среднее время выполнения 100 запросов, мс	
	с кэшированием	без кэширования
500	1246.17	2440.40
1000	1109.59	2847.10
1500	1168.45	3272.73
2000	1237.17	3514.95
2500	1270.37	3879.98
3000	1296.57	4562.16
3500	1468.89	4966.40
4000	1413.22	5295.74
4500	1426.80	5544.67
5000	1455.55	5711.20
5500	1463.08	6020.39
6000	1583.57	6259.69
6500	1542.33	6636.07
7000	1587.09	7038.91
7500	1680.85	7204.35
8000	1690.19	7437.81
8500	1707.12	8070.07
9000	1728.39	8259.58
9500	1874.58	8650.11
10000	1948.10	9039.44

На рисунке 4.1 представлены графики зависимости времени выполнения запросов от числа записей в таблице, при наличии кэширования и без него.

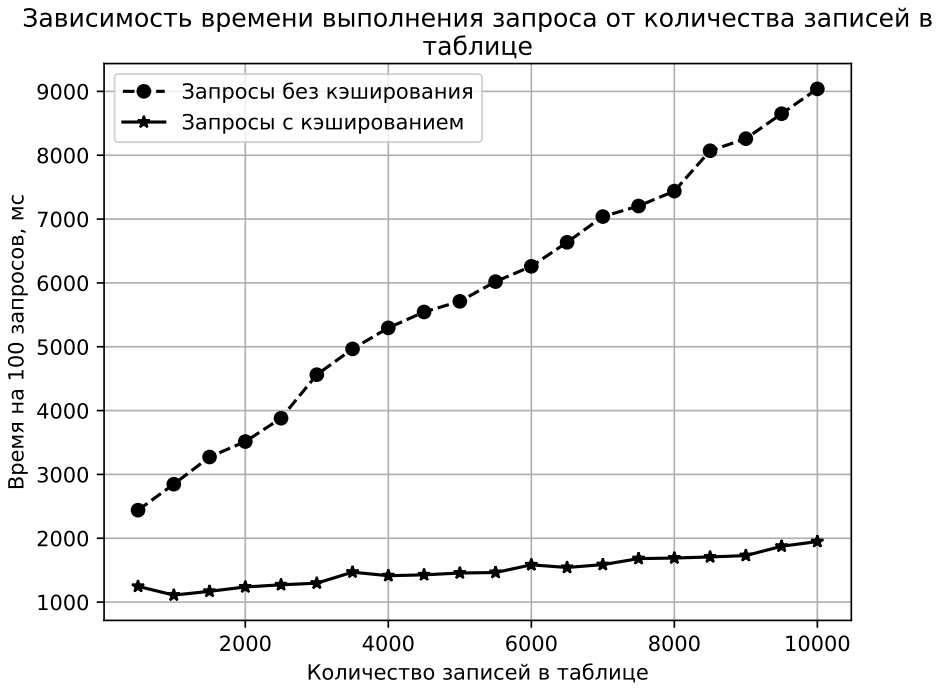

Рисунок 4.1 — Зависимость времени выполнения запросов от количества записей в таблице.

Как видно из полученных данных, время выполнения запросов растет с количеством данных в таблице. При этом использование кэширования сокращает время выполнения запроса в среднем в 4 раза при 5000 записей в таблице.

4.3 Зависимость времени выполнения от количества одновременно выполняемых запросов

В таблице 4.2 представлены результаты измерения зависимости времени выполнения запросов от количества одновременно выполняемых запросов к таблице education. Представленые данные получены в результате измерения времени непрерывного выполнения запросов указанным числом клиентов и числа выполненных запросов за определенный временной интервал. После этого получалось усредненное время выполнения 10 запросов клиентом. Количество записей в таблице — 5000.

Таблица 4.2 — Время выполнения запросов в зависимости от количества одноврменно выполняющих запросы клиентов

Количество клиентов	Среднее время выполнения 10 запросов, мс	
	с кэшированием	без кэширования
5	280.11	495.75
10	426.28	625.69
15	638.83	843.32
20	827.79	1062.09
25	1221.80	1365.25
30	1323.70	1540.77
35	1357.12	1675.56
40	1534.46	1874.32
45	1730.39	2107.04
50	1921.09	2304.87

На рисунке 4.2 представлены графики зависимости времени выполнения запросов от числа одновременно выполняемых запросов, при наличии кэширования и без него.

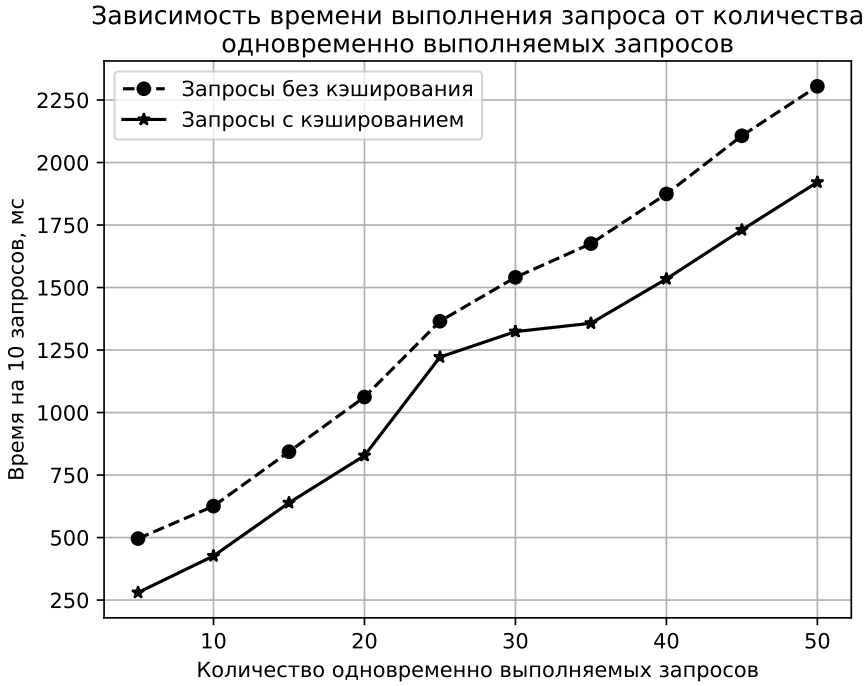

Рисунок 4.2 — Зависимость времени выполнения запросов от количества одновременно выполняемых запросов.

Из полученных данных видно, что время выполнения запросов растет с числом пользователей, при этом при использовании кэширования время выполнения в среднем в 1.3 раза меньше.

4.4 Вывод

По результатам замеров времени выполнения было выявлено, что время выполнения запроса растет как с ростом числа записей в таблице, так и с ростом числа одновременно выполняемых запросов. При этом использование кэширования позволяет снизить время выполнения запросов в обоих случаях.

ЗАКЛЮЧЕНИЕ

Поставленная цель была достигнута: разработана база данных для хранения и обработки оценок трудовой эффективности сотрудников группы компаний.

В ходе выполнения работы были выполнены следующие задачи:

— проведен анализ предметной области, выделины основные сущности и связи между ними, сформулированы требования и ограничения к разрабатываемой базе данных;

— спроектированы сущности базы данных и определины ограничения целостности данных, спроектирована ролевую модель на уровне базы данных;

— выбраны средства реализации базы данных и приложения;

— реализованы сущности базы данных и ограничения целостности. Описаны методы тестирования и разработаны тестовые случаи для проверки корректности работы функционала;

— проведено исследование производительности базы данных при увеличении объема данных и количестве одновременно выполняемых запросов, а также с использованием кеширования и без него.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ASP.NET [Электронный ресурс]. Режим доступа: https://dotnet.microsoft.com/ru-ru/apps/aspnet. Дата обращения: 07.05.25.
2. C# [Электронный ресурс]. Режим доступа: https://learn.microsoft.com/ru-ru/dotnet/csharp/. Дата обращения: 07.05.25.
3. Client Connection Defaults [Электронный ресурс]. Режим доступа: https://www.postgresql.org/docs/current/runtime-config-client.html. Дата обращения: 01.05.25.
4. CREATE FUNCTION [Электронный ресурс]. Режим доступа: https://www.postgresql.org/docs/current/sql-createfunction.html. Дата обращения: 25.04.25.
5. Entity Framework Core [Электронный ресурс]. Режим доступа: https://learn.microsoft.com/ru-ru/ef/core/. Дата обращения: 07.05.25.
6. Happy job [Электронный ресурс]. Режим доступа: https://happy-job.ru/. Дата обращения: 16.03.25.
7. IBM DB2 [Электронный ресурс]. Режим доступа: https://www.ibm.com/db2?ysclid=mbkpdlgyt3113285185. Дата обращения: 05.05.25.
8. iSpring [Электронный ресурс]. Режим доступа: https://www.ispring.ru/. Дата обращения: 16.03.25.
9. Jinn [Электронный ресурс]. Режим доступа: https://jinn.ru/. Дата обращения: 16.03.25.
10. .NET [Электронный ресурс]. Режим доступа: https://dotnet.microsoft.com/ru-ru/. Дата обращения: 07.05.25.
11. Oracle Database [Электронный ресурс]. Режим доступа: https://www.oracle.com/database/. Дата обращения: 05.05.25.
12. PostgreSQL [Электронный ресурс]. Режим доступа: https://www.postgresql.org/. Дата обращения: 05.05.25.
13. Proaction [Электронный ресурс]. Режим доступа: https://proaction.pro/. Дата обращения: 16.03.25.
14. Redis [Электронный ресурс]. Режим доступа: https://redis.io/. Дата обращения: 15.05.25.

15. Redmine [Электронный ресурс]. Режим доступа: https://redmine.org/. Дата обращения: 16.03.25.

16. Row Security Policies [Электронный ресурс]. Режим доступа: https://www.postgresql.org/docs/current/ddl-rowsecurity.html. Дата обращения: 19.04.25.

17. Startexam [Электронный ресурс]. Режим доступа: https://www.ispring.ru/. Дата обращения: 16.03.25.

18. Swashbuckle [Электронный ресурс]. Режим доступа: https://github.com/domaindrivendev/Swashbuckle.AspNetCore#readme. Дата обращения: 08.05.25.

19. ToolKeeper [Электронный ресурс]. Режим доступа: https://toolkeeper.io/. Дата обращения: 16.03.25.

20. Weeek [Электронный ресурс]. Режим доступа: https://weeek.net/. Дата обращения: 01.04.25.

21. Системы менеджмента качества. Основные положения и словарь. ГОСТ Р ISO 9000:2008, 2009.

22. Ильченко С. В. Борщева А. В. Методы оценки эффективности трудовой деятельности персонала. *Вестник экспериментального образования*, 3(16), 2018.

23. Вишнякова М. В. *Мифы и правда о KPI*. ЛЕТОПИСЬ, Москва, 2017.

24. Семенов О. Ю. Воробович Н. П. Программные методы и средства планирования и управления проектами. *Вестник КрасГАУ*, 10, 2009.

25. Кузнецов С. Д. Объектно-реляционные базы данных: прошедший этап или недооцененные возможности? *Труды ИСП РАН*, (2), 2008.

26. Дейт К. Дж. *Введение в системы баз данных*. «Вильямс», Москва, 8 edition, 2005.

27. Митрофанова А. Е. Обоснование подходов к оценке результативности труда персонала организации. *Вестник ГУУ*, 12, 2016.

28. Комаров В. И. *Путеводитель по базам данных*. ДМК-Пресс, Москва, 2024.

29. Хеллевиг Й. *Вовлеченность персонала в России. Предварительная версия. Как построить корпоративную культуру, основанную на вовлеченности персонала, клиентоориентированности и инновациях*. Russia Advisory Group Oy, Хельсинки, 2012.

30. Османова З. О. Методы оценки компетенций персонала предприятия. *Научный вестник: финансы, банки, инвестиции*, 3(36), 2016.

31. Ехлаков Ю. П. *Управление программными проектами: учебник*. Изд-во Томск. гос. ун-та систем управления и радиоэлектроники, Томск, 2015.

32. Гананчян А. В. Уткин Г. С., Жильцов С. А. Постреляционная система управления базой данных на основе словарной технологии. *Космическая техника и технологии*, (39), 2022.

Приложение А

Презентация к курсовой работе на 13 слайдах.