**Содержание**

[Введение 3](#_Toc152881671)

[1. Разбор и постановка задачи 4](#_Toc152881672)

[1.1 Аналитический обзор аналогов 4](#_Toc152881673)

[1.2 Описание функционала базы данных 6](#_Toc152881674)

[1.3 Разработка функциональных требований, определение вариантов использования 7](#_Toc152881675)

[2. Проектирование базы данных 8](#_Toc152881676)

[2.1 Схема базы данных 8](#_Toc152881677)

[2.2 Таблицы базы данных 9](#_Toc152881678)

[3. Разработка объектов базы данных 13](#_Toc152881679)

[3.1 Создание пользователей, ролей, и таблиц базы данных 13](#_Toc152881680)

[3.2 Пользовательские процедуры 14](#_Toc152881681)

[3.3 Индексы базы данных 15](#_Toc152881682)

[3.4 Последовательности 16](#_Toc152881683)

[3.5 Триггеры 16](#_Toc152881684)

[3.6 Директории 17](#_Toc152881685)

[4. Описание процедур импорта и экспорта 18](#_Toc152881686)

[4.1 Экспорт в JSON 18](#_Toc152881687)

[4.2 Импорт из JSON 19](#_Toc152881688)

[5. Тестирование производительности 21](#_Toc152881689)

[6. Описание технологии и ее применения в базе данных 23](#_Toc152881690)

[6.1 Шифрование данных в Oracle 23](#_Toc152881691)

[6.2 Функции шифрования и дешифрование данных 23](#_Toc152881692)

[6.3 Аудит SYS операций 24](#_Toc152881693)

[6.4 Включение аудита базы данных 24](#_Toc152881694)

[6.5 Защита словаря данных 24](#_Toc152881695)

[6.6 Защита слушателя 25](#_Toc152881696)

[7. Краткое описание приложения для демонстрации 26](#_Toc152881697)

[8. Руководство пользователя 27](#_Toc152881698)

[8.1 Пример работы приложения от лица пользователя 27](#_Toc152881699)

[8.2 Пример работы приложения от лица администратора 28](#_Toc152881700)

[Заключение 29](#_Toc152881701)

[Список используемых источников 30](#_Toc152881702)

[Приложение А 31](#_Toc152881703)

[Приложение Б 34](#_Toc152881704)

Введение

Базы данных широко используются в различных сферах деятельности, в том числе в управлении персоналом. Учет кадров – это процесс сбора, хранения, обработки и анализа информации о работниках организации, их трудовой деятельности и социально-экономическом положении.

Современные компании активно используют базы данных для эффективного управления и хранения информации о своем персонале. В рамках данного курсового проекта предлагается разработать базу данных для страховой компании с использованием системы управления базами данных Oracle 21c.

Целью данной работы являлась разработка реляционной базы данных для учета кадровых данных с применением настройки системы безопасности сервера СУБД, которая предназначена для хранения информации о кадрах, а также для обеспечения быстрого доступа к хранимой информации. Для достижения этой цели необходимо выполнить следующие задачи:

* Провести анализ предметной области и определить основные сущности и атрибуты базы данных;
* Разработать концептуальную модель базы данных в нотации ER-диаграмм;
* Преобразовать концептуальную модель в логическую модель в нотации реляционной алгебры;
* Реализовать логическую модель в физической модели с использованием oracle;
* Наполнить базу данных тестовыми данными;
* Разработать хранимые процедуры, функции и триггеры для реализации бизнес-логики базы данных;
* Разработать запросы для получения необходимой информации из базы данных.

1. Разбор и постановка задачи
   1. Аналитический обзор аналогов

В настоящее время существует множество программных продуктов, которые предназначены для автоматизации кадрового учета и управления персоналом. Они позволяют решать различные задачи, связанные с формированием штатного расписания, приемом и увольнением сотрудников, регистрацией личных данных, расчетом заработной платы, обучением и аттестацией, планированием мотивации и карьерного роста, а также формированием кадровой отчетности и аналитики.

[1С:Зарплата и Кадры](http://unitoria.ru/blog/kadry/top-5-programm-dlya-kadrovogo-ucheta-v-2021-godu) – это наиболее распространенный и известный продукт на российском рынке, который обеспечивает полный функционал кадрового учета и расчета заработной платы в соответствии с законодательством РФ. Программа позволяет вести учет сотрудников, их личных данных, кадровых документов, отпусков, больничных, командировок, квалификационных категорий, аттестаций, наград и дисциплинарных взысканий. Также программа формирует различные виды кадровой отчетности, в том числе для государственных органов и фондов. Программа имеет удобный и привычный интерфейс, а также возможность интеграции с другими продуктами 1С. Однако программа не содержит функционала для управления персоналом, такого как подбор, обучение, оценка, мотивация и развитие сотрудников. Кроме того, программа имеет устаревший дизайн и не поддерживает облачные технологии. Интерфейс программы представлен на рисунке 1.1.

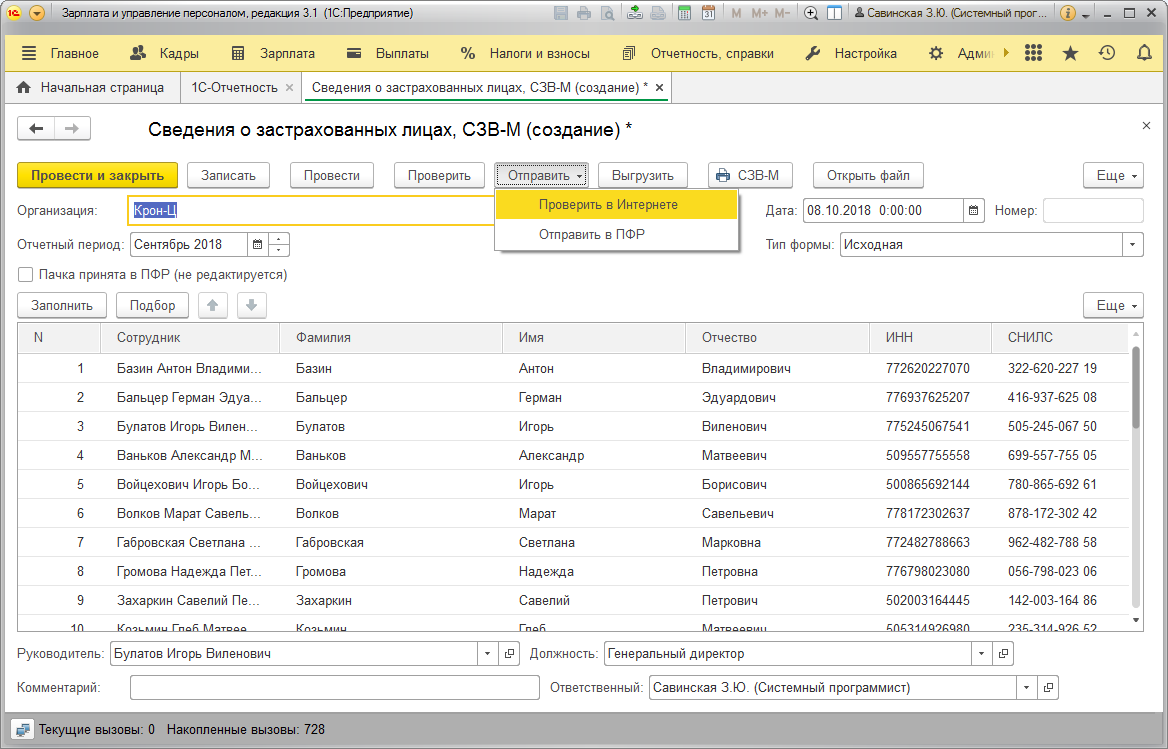


Рисунок 1.1 – интерфейс [1С:Зарплата и Кадры](http://unitoria.ru/blog/kadry/top-5-programm-dlya-kadrovogo-ucheta-v-2021-godu)

[Microsoft Dynamics 365 Human Resources](https://wiseadvice-it.ru/o-kompanii/blog/articles/programmy-dlya-vedeniya-kadrovogo-ucheta/) – это современный и мощный продукт от корпорации Microsoft, который предоставляет комплексный HRM-функционал для работы с бизнес-процессами управления персоналом и расчета заработной платы. Программа позволяет вести подбор персонала, обучение, оценку и аттестацию сотрудников, разработку мотивации и KPI, охрану труда, планирование, учет и аналитику персонала. Также программа имеет современный интерфейс и синхронизацию с сервисами Microsoft, такими как Outlook, Teams, Power BI и другие. Однако программа имеет ограниченный функционал расчета заработной платы, который не учитывает специфику белорусского законодательства и требует дополнительной настройки и интеграции. Кроме того, программа имеет высокие финансовые издержки внедрения и использования, а также небольшой объем внедрений на предприятиях и пользовательского опыта. Интерфейс программы представлен на рисунке 1.2.

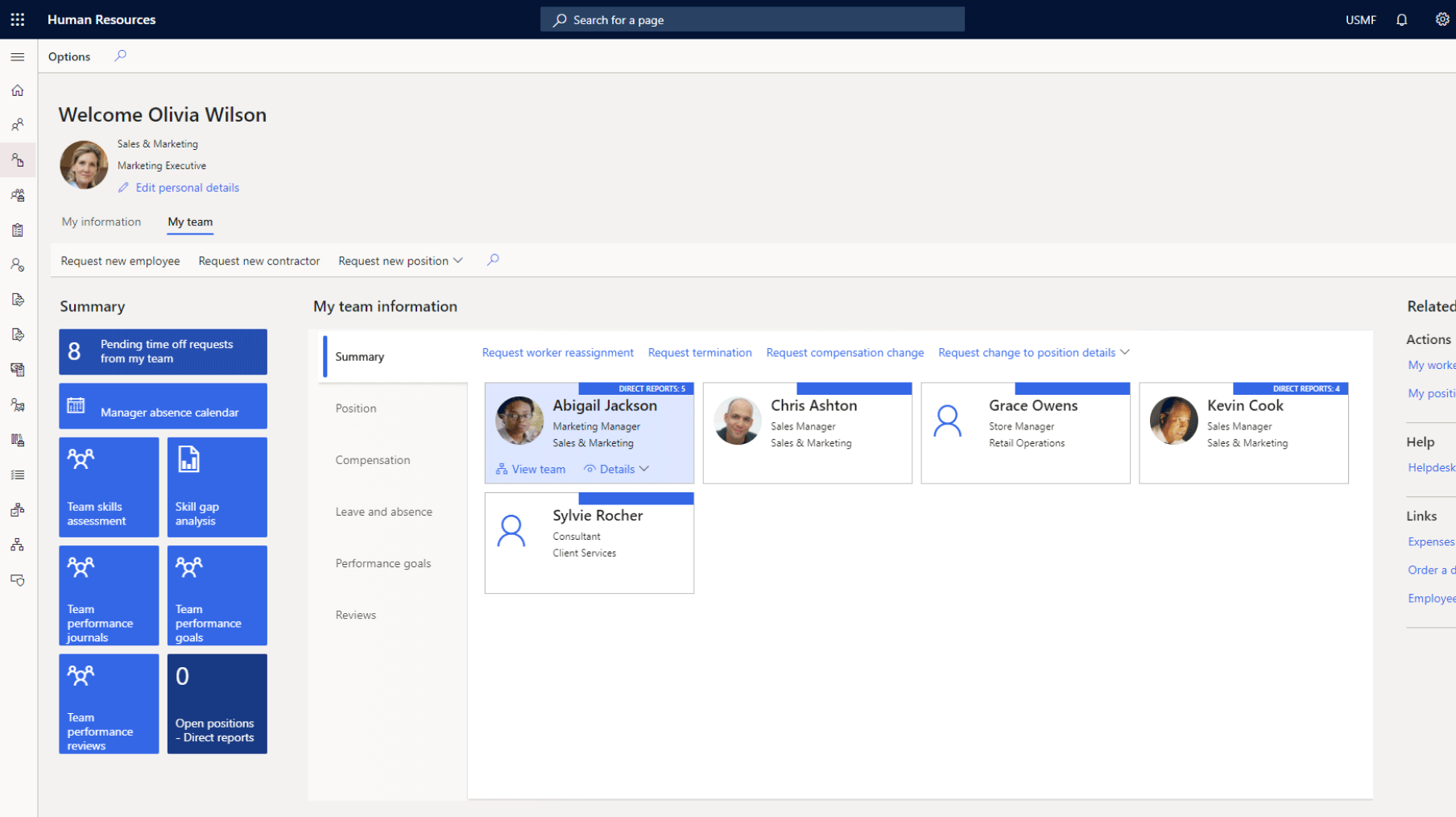


Рисунок 1.2 – интерфейс [Microsoft Dynamics 365 Human Resources](https://wiseadvice-it.ru/o-kompanii/blog/articles/programmy-dlya-vedeniya-kadrovogo-ucheta/)

SAP SuccessFactors – это продвинутый и инновационный продукт от компании SAP, который является лидером в области ERP-систем. Программа предоставляет полноценный HRM-функционал для работы с процессами управления персоналом и расчета заработной платы. Программа позволяет вести подбор персонала и развитие персонала, штатное расписание и структуру компании, кадровый учет с набором основных документов, планирование системы мотивации и KPI, кадровую аналитику и отчетность. Программа также имеет возможность расчета заработной платы, однако он не адаптирован к российскому законодательству и требует дополнительной настройки и интеграции. Программа имеет преимущество в том, что она создает единую систему автоматизации, если помимо этой системы на предприятии все автоматизировано на базе SAP. Однако программа также имеет высокие финансовые издержки внедрения и использования, а также смещает акцент с регламентированного кадрового учета на HR-процессы. Интерфейс программы представлен на рисунке 1.3.

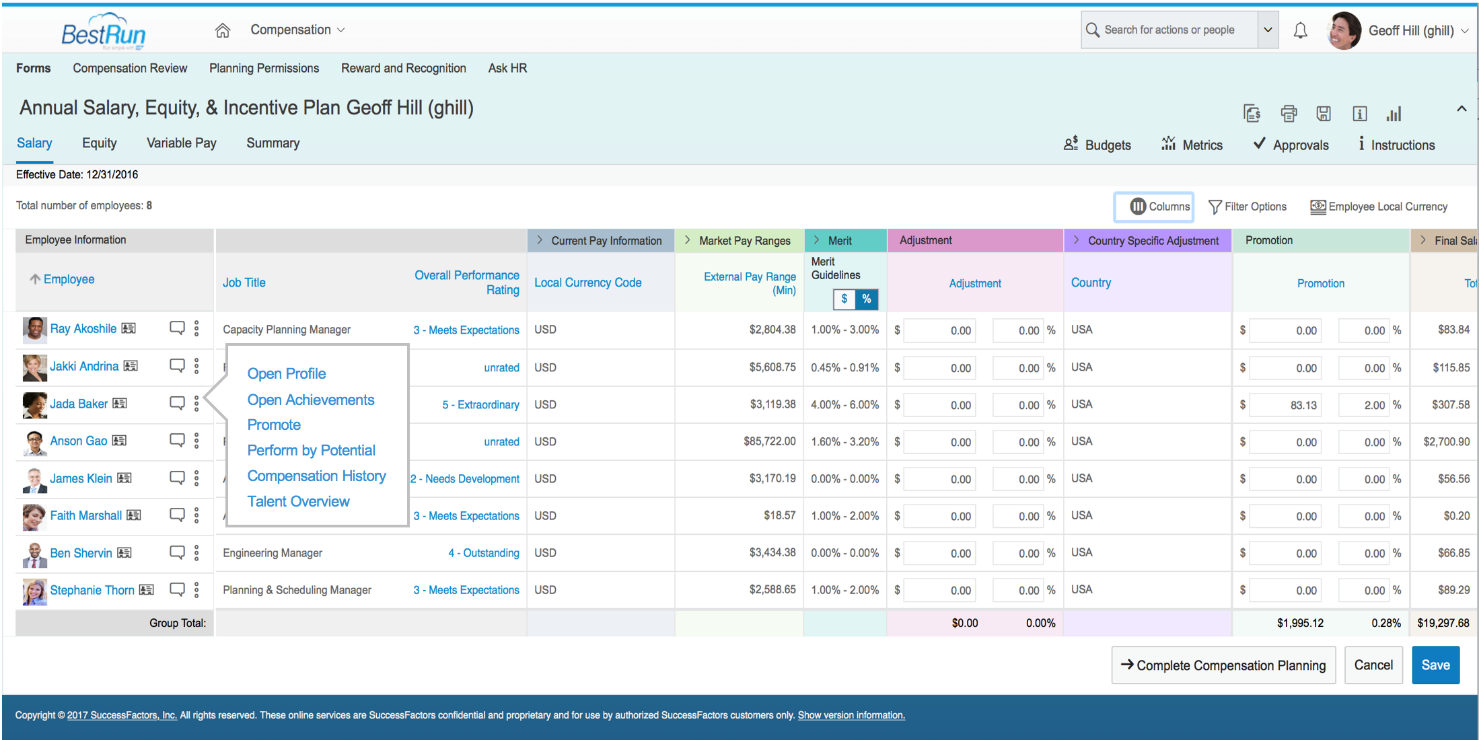


Рисунок 1.4 – интерфейс [SAP SuccessFactors](https://habr.com/ru/companies/otus/articles/659657/)

Таким образом, можно сделать вывод, что существующие аналоги базы данных для учета кадровых данных имеют свои преимущества и недостатки, и не полностью удовлетворяют потребностям российских предприятий. Поэтому целесообразно разработать собственную базу данных для учета кадровых данных с применением настройки системы безопасности сервера СУБД,

* 1. Описание функционала базы данных

Для выполнения данного курсового проекта были использованы следующие технологии:

Oracle 21c - реляционная система управления базами данных (СУБД), которая обеспечивает надежное и эффективное хранение, управление и защиту данных. Она выбрана для создания и управления базой данных, которая использовалась в данном проекте.

DataGrip - мощный инструмент для работы с базами данных, который предоставляет широкие возможности для разработки, администрирования и анализа данных. Он используется для создания схемы базы данных, написания запросов, выполнения тестирования и оптимизации запросов.

Исходя из описанных выше технологий, можно сделать вывод о том, что для реализации данного проекта были использованы современные инструменты и технологии, что позволило повысить эффективность работы. Oracle 21c предоставляет высокую производительность и надежность в работе с базами данных, а DataGrip обеспечивает удобное и интуитивно понятное управление базами данных. В целом, использование таких технологий позволило реализовать проект на высоком уровне и обеспечить его успешное выполнение.

* 1. Разработка функциональных требований, определение вариантов использования

К основным функциональным требованиям относятся:

* База данных должна быть реализована в СУБД Oracle.
* Доступ к данным должен осуществляться только через соответствующие процедуры.
* Количество объектов БД (таблиц, представлений, индексов, пользователей и пр.) регламентируется задачей.
* Должен быть проведен импорт данных из JSON файлов, экспорт данных в формат JSON.
* Необходимо протестировать производительность базы данных на таблице, содержащей не менее 100 000 строк, и внести изменения в структуру в случае необходимости. Необходимо проанализировать планы запросов к таблице.
* Применить технологию базы данных согласно выбранной теме: подробно описать применяемые системные пакеты, утилиты или технологии; показать применение указанной технологии в базе данных.
* Листинги проекта должны содержать комментарии.

1. Проектирование базы данных
   1. Схема базы данных

Для реализации базы данных учета кадров необходимо создать следующие таблицы: сотрудники, должности, отделы, местоположение, страны, регионы, проекты, задачи, таблица участия сотрудников в проектах и таблица оценок. На рисунке 2.1 представлена UML-диаграмма, соответствующая данной схеме базы данных:

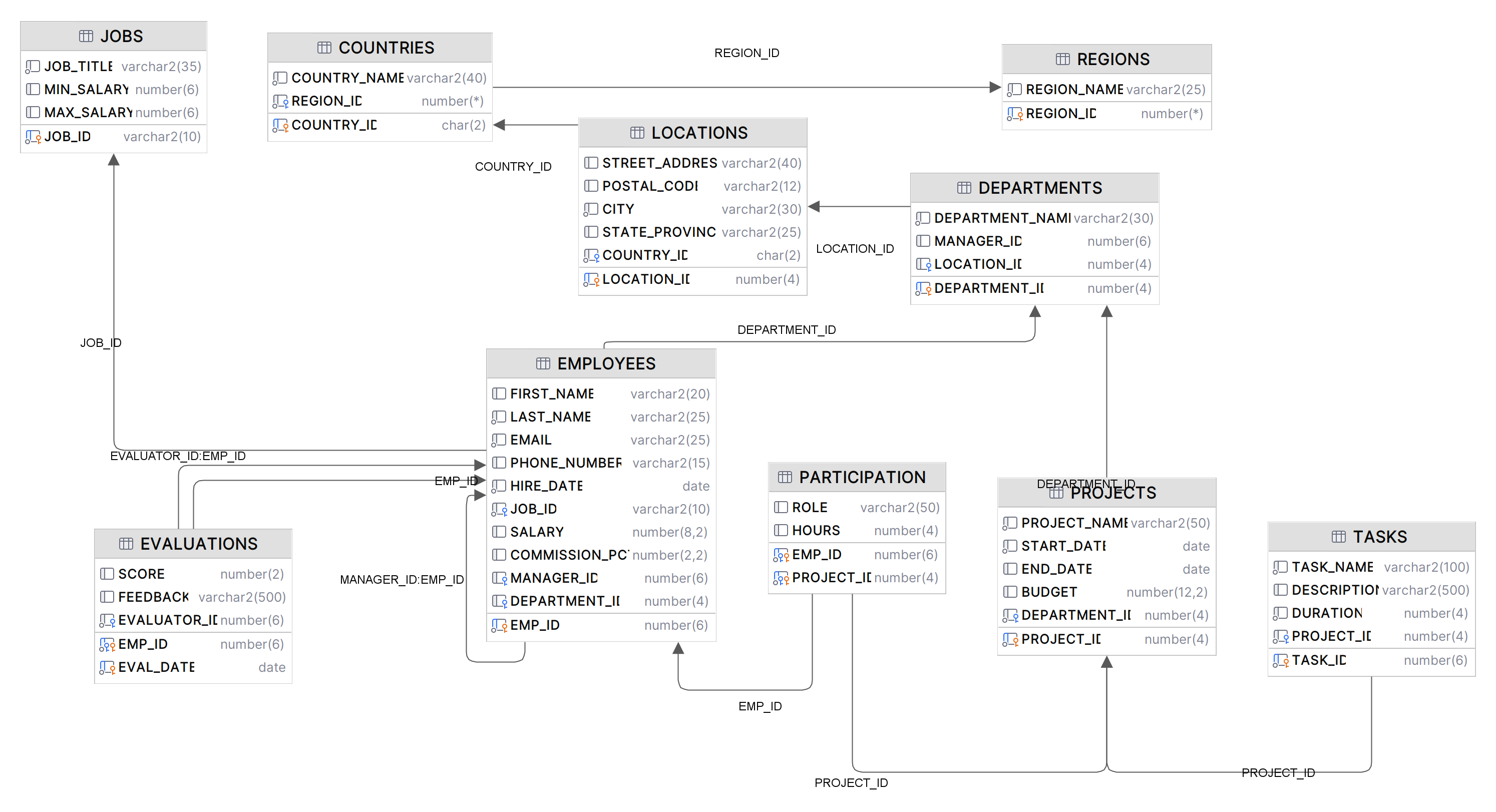


Рисунок 2.1 – UML-диаграмма базы данных страховой компании

Для наглядного представления работы пользователей с такой схемой базы данных представлена диаграмма вариантов использования продукта на рисунке 2.2.

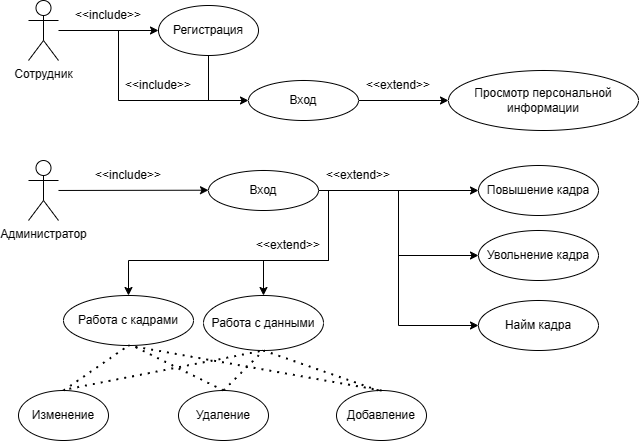


Рисунок 2.2 – Диаграмма вариантов использования

Таким образом будут реализованы базовые функции реализации работы базы данных. Клиент будет взаимодействовать с страховым агентом, совместно оформлять страховые полисы (выполнение идёт со стороны страхового агента). После клиент сможет рассматривать свои оформленные полисы, оставлять заяви на оценку возможного нового страхования и отслеживать процесс рассмотрения его заявлений на выплату по страховым полисам.

* 1. Таблицы базы данных

Каждая таблица базы данных будет описана в далее в таблицах. Скрипт создания таблиц представлен в приложении А.

Таблица 2.1 – EMPLOYEES

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| emp\_id | NUMBER(6) | Идентификатор сотрудника |
| first\_name | VARCHAR2(20) | Имя сотрудника |
| last\_name | VARCHAR2(25) NOT NULL | Фамилия сотрудника |
| email | VARCHAR2(25) NOT NULL | Электронная почта сотрудника |
| phone\_number | VARCHAR2(25) NOT NULL | Номер телефона сотрудника |
| hire\_date | VARCHAR2(15) | Дата найма сотрудника |
| job\_id | DATE NOT NULL | Идентификатор должности сотрудника |
| salary | VARCHAR2(10) NOT NULL | Зарплата сотрудника |
| commission\_pct | NUMBER(8,2) | Процент комиссии сотрудника |
| manager\_id | NUMBER(6) | Идентификатор менеджера сотрудника |
| department\_id | NUMBER(4) | Идентификатор отдела сотрудника |
| password\_hash | VARCHAR2(128) | Пароль |

Эта таблица хранит информацию о сотрудниках.

Таблица 2.2 – JOBS

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| job\_id | VARCHAR2(10)PRIMARY KEY | Идентификатор должности |
| job\_title | VARCHAR2(35) NOT NULL | Название должности |
| min\_salary | NUMBER(6) | Минимальная зарплата для должности |
| max\_salary | NUMBER(6) | Максимальная зарплата для должности |

Эта таблица хранит информацию о должностях.

Таблица 2.3 – DEPARTMENTS

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| department\_id | NUMBER(4)PRIMARY KEY | Идентификатор отдела |
| department\_name | VARCHAR2(30) NOT NULL | Название отдела |
| manager\_id | NUMBER(6) | Идентификатор менеджера отдела |
| location\_id | NUMBER(4) | Идентификатор местоположения отдела |

Эта таблица хранит информацию об отделах.

Таблица 2.4 – LOCATIONS

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| location\_id | NUMBER(4) PRIMARY KEY | Идентификатор местоположения |
| street\_address | VARCHAR2(40) | Улица местоположения |
| postal\_code | VARCHAR2(12) | Почтовый индекс местоположения |
| City | VARCHAR2(30) NOT NULL | Город местоположения |
| state\_province | VARCHAR2(25) | Область местоположения |
| country\_id | CHAR(2) NOT NULL | Идентификатор страны местоположения |

Эта таблица хранит информацию местоположений отдел.

Таблица 2.5 – COUNTRIES

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| country\_id | CHAR(2) PRIMARY KEY | Идентификатор страны |
| country\_name | VARCHAR2(40) NOT NULL | Название страны |

Эта таблица хранит информацию о странах.

Таблица 2.6 – PROJECTS

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| project\_id | NUMBER(4) PRIMARY KEY | Идентификатор проекта |
| project\_name | VARCHAR2(50) NOT NULL | Название проекта |
| start\_date | DATE NOT NULL | Дата начала проекта |
| end\_date | DATE | Дата окончания проекта (может быть null) |
| budget | NUMBER(12,2) | Бюджет проекта |
| department\_id | NUMBER(4) NOT NULL | Идентификатор отдела, ответственного за проект |

Эта таблица хранит информацию о проектах.

Таблица 2.7 – PARTICIPATION

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| emp\_id | NUMBER(6) NOT NULL | Идентификатор сотрудника, участвующего в проекте |
| project\_id | NUMBER(4) NOT NULL | Идентификатор проекта, в котором участвует сотрудник |
| role | VARCHAR2(50) | Роль сотрудника в проекте |
| Hours | NUMBER(4) | Количество часов, затраченных сотрудником на проект |
| PRIMARY KEY (emp\_id, project\_id) |  | Составной первичный ключ из emp\_id и project\_id |

Эта таблица хранит информацию об участии сотрудников в проектах.

Таблица 2.8 – TASKS

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| task\_id | NUMBER(6) PRIMARY KEY | Идентификатор задачи |
| task\_name | VARCHAR2(100) NOT NULL | Название задачи |
| Description | VARCHAR2(500) | Описание задачи |
| Duration | NUMBER(4) NOT NULL | Продолжительность задачи в часах |
| project\_id | NUMBER(4) NOT NULL | Идентификатор проекта, к которому относится задача |

Эта таблица хранит информацию о задачах сотрудников.

Таблица 2.9 – VACATIONS

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| vacation\_id | NUMBER(6)PRIMARY KEY | Идентификатор отпуска |
| emp\_id | NUMBER(6)NOT NULL | Идентификатор сотрудника |
| start\_date | DATE NOT NULL | Дата начала отпуска |
| end\_date | DATE NOT NULL | Дата окончания отпуска |
| Reason | VARCHAR2(500) | Причина отпуска |

Эта таблица хранит информацию о отпусках сотрудников.

1. Разработка объектов базы данных
   1. Создание пользователей, ролей, и таблиц базы данных

Для работы с базой данных необходимо выделить несколько пользователей с определёнными привилегиями. Начало разработки базы данных начинается с созданий её объектов. Для этого создадим пользователя и администратора. Администратору будут выданы привилегии создания сессии, создание и изменение основных объектов базы данных: таблицы, представления, курсоры, последовательностей и т.д. Пользователю будут выданы права на вызов определенных процедур. С выданными права пользователь сможет использовать основную функциональность базы данных. Скрипт создания представлен на листинге 3.1.

|  |
| --- |
| *-- Создание профиля для пользователя* CREATE PROFILE user\_profile LIMIT  FAILED\_LOGIN\_ATTEMPTS 5  SESSIONS\_PER\_USER 10  CPU\_PER\_SESSION UNLIMITED  CPU\_PER\_CALL 3000  CONNECT\_TIME 45  IDLE\_TIME 15  LOGICAL\_READS\_PER\_SESSION DEFAULT  LOGICAL\_READS\_PER\_CALL DEFAULT  PRIVATE\_SGA 15K  COMPOSITE\_LIMIT 5000000;  *-- Создание профиля для администратора* CREATE PROFILE admin\_profile LIMIT  SESSIONS\_PER\_USER UNLIMITED  CPU\_PER\_SESSION UNLIMITED  CPU\_PER\_CALL UNLIMITED  CONNECT\_TIME UNLIMITED  IDLE\_TIME UNLIMITED  LOGICAL\_READS\_PER\_SESSION DEFAULT  LOGICAL\_READS\_PER\_CALL DEFAULT  PRIVATE\_SGA UNLIMITED  COMPOSITE\_LIMIT UNLIMITED;  *-- Создание пользователя с ролью администратора и профилем администратора* CREATE USER admin\_user IDENTIFIED BY admin\_password  DEFAULT TABLESPACE USERS\_TBS  TEMPORARY TABLESPACE TEMP\_TBS  PROFILE admin\_profile;  CREATE USER user\_user IDENTIFIED BY user\_password  DEFAULT TABLESPACE USERS\_TBS  TEMPORARY TABLESPACE TEMP\_TBS  PROFILE user\_profile; |

Листинг 3.1 – Создание пользователя для менеджера

* 1. Пользовательские процедуры

Следующим этапом будет разработка основных процедур, для осуществления работы с базой данных. После этого станет ясно какие дополнительные объекты базы данных будет необходимо создать для оптимизации выполнения процедур. Для наглядности работы системы определим ожидаемый действия всех пользователей.

На листинге 3.2 представлена процедура для повышения сотрудника в должности. Процедура promote\_employee обновляет должность сотрудника в таблице employees. Затем процедура выполняет операцию UPDATE, чтобы изменить должность сотрудника на новую, и, если обновление проходит успешно, происходит COMMIT, который сохраняет изменения в базе данных.

|  |
| --- |
| create PROCEDURE *promote\_employee* (  p\_emp\_id IN employees.emp\_id%TYPE,  p\_new\_job\_id IN employees.job\_id%TYPE ) IS BEGIN  UPDATE employees SET job\_id = p\_new\_job\_id WHERE emp\_id = p\_emp\_id;  COMMIT; EXCEPTION  WHEN OTHERS THEN  ROLLBACK; RAISE; END *promote\_employee*; / |

Листинг 3.2 – Процедура для повышения сотрудника в должности

На листинге 3.3 представлена процедура для найма сотрудника. Процедура insert\_into\_employees вставляет новую запись в таблицу employees. Значение password\_hash шифруется функцией ENCRYPT перед вставкой. Если вставка проходит успешно, происходит COMMIT, который сохраняет изменения в базе данных.

|  |
| --- |
| create PROCEDURE *insert\_into\_employees* (p\_first\_name VARCHAR2,p\_last\_name VARCHAR2,p\_email VARCHAR2,p\_phone\_number VARCHAR2,p\_hire\_date DATE,p\_job\_id VARCHAR2,p\_salary NUMBER,p\_commission\_pct NUMBER,p\_manager\_id NUMBER,p\_department\_id NUMBER,p\_password\_hash VARCHAR2 ) IS v\_emp\_id employees.emp\_id%TYPE; BEGIN  INSERT INTO employees (  emp\_id, first\_name, last\_name, email, phone\_number, hire\_date, job\_id, salary, commission\_pct, manager\_id, department\_id, password\_hash  ) VALUES (  v\_emp\_id, p\_first\_name, p\_last\_name, p\_email, p\_phone\_number, p\_hire\_date, p\_job\_id, p\_salary, p\_commission\_pct, p\_manager\_id, p\_department\_id, *ENCRYPT*(p\_password\_hash)  );  COMMIT; EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE; END *insert\_into\_employees*; / |

Листинг 3.3 – Процедура для найма сотрудника

В листинге 3.4 представлена процедура для увольнения сотрудника. Процедура fire\_employee удаляет сотрудника из таблицы employees.

|  |
| --- |
| create PROCEDURE *fire\_employee* (p\_emp\_id NUMBER) IS BEGIN  *-- Удалить данные о сотруднике из таблицы employees* DELETE FROM employees WHERE emp\_id = p\_emp\_id;  *-- Вывести сообщение об успешном увольнении сотрудника* DBMS\_OUTPUT.*PUT\_LINE*('Employee ' || p\_emp\_id || ' has been fired.');  COMMIT; EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE; END *fire\_employee*; / |

Листинг 3.4 – Процедура для увольнения сотрудника

Так же, были реализованы процедуры для изменения, удаления и вставки данных во все таблицы. Некоторые вспомогательные процедуру представлены ниже в таблице 3.1.

Таблица 3.1 – Вспомогательные процедуры

|  |  |
| --- | --- |
| Название процедуры | Описание процедуры |
| encrypt(p\_plain\_text VARCHAR2) | шифрует обычный текст. |
| decrypt(p\_encrypted\_text RAW) | расшифровывает зашифрованный текст |
| export\_json() | экспортирует данные из таблицы employees в файл JSON |
| import\_json() | импортирует данные из файла JSON в таблицу employees |

* 1. Индексы базы данных

Индексы — это структуры данных, построенные на основе одного или нескольких столбцов таблицы. Индекс предоставляет быстрый доступ к данным таблицы и позволяет ускорить выполнение операций SELECT, UPDATE и DELETE, так как он содержит отсортированные значения из одного или нескольких столбцов и указатели на строки таблицы.

Применение индексов представляет собой компромисс между ускорением получения результатов запросов и замедлением обновлений и вставок данных. Первая часть этого компромисса – ускорение запросов – довольно очевидна: если поиск выполняется по отсортированному индексу вместо полного сканирования всей таблиц, то запрос проходит намного быстрее. Но всякий раз, когда вы обновляете, вставляете или удаляете строку таблицы с индексами, индексы также должны быть обновлены соответствующим образом. То есть такие операции на таблицах с индексами обходятся дороже.

* 1. Последовательности

Последовательности в SQL используются для генерации уникальных числовых идентификаторов. Они часто используются для автоматического создания первичных ключей. Создание последовательностей представлено в листинге 3.5.

Каждая последовательность начинается с 1 и увеличивается на 1 при каждом вызове. Это обеспечивает уникальность каждого идентификатора и предотвращает возможные конфликты при вставке новых записей в таблицы.

|  |
| --- |
| *-- Создание последовательности для поля ID в* CREATE SEQUENCE job\_id\_seq START WITH 1 INCREMENT BY 1; *-- таблице jobs* CREATE SEQUENCE department\_id\_seq START WITH 1 INCREMENT BY 1; *-- в таблице departments* CREATE SEQUENCE location\_id\_seq START WITH 1 INCREMENT BY 1; *-- в таблице locations* CREATE SEQUENCE project\_id\_seq START WITH 1 INCREMENT BY 1; *-- в таблице projects* CREATE SEQUENCE task\_id\_seq START WITH 1 INCREMENT BY 1; *-- в таблице tasks* CREATE SEQUENCE emp\_id\_seq START WITH 1 INCREMENT BY 1; *-- в таблице employees* CREATE SEQUENCE vacation\_id\_seq START WITH 1 INCREMENT BY 1; *-- в таблице vacations* CREATE SEQUENCE countries\_id\_seq START WITH 1 INCREMENT BY 1; *-- в таблице countries* CREATE SEQUENCE participation\_id\_seq START WITH 1 INCREMENT BY 1; *-- в таблице participation* |

Листинг 3.5 – Создание последовательностей

Создание этих последовательностей позволяет нам автоматизировать процесс присвоения идентификаторов, что упрощает вставку новых данных и уменьшает вероятность ошибок. Это также улучшает эффективность работы с базой данных, поскольку нам не нужно вручную контр

* 1. Триггеры

Триггеры в SQL - это специальные процедуры, которые автоматически запускаются при определенных событиях в базе данных, таких как вставка, обновление или удаление записей. Они играют важную роль в поддержании целостности данных и могут быть использованы для автоматизации определенных операций.

В данной работе триггеры обеспечивают автоматическое присвоение уникальных идентификаторов при вставке новых записей в таблицы. Это упрощает процесс вставки данных и уменьшает вероятность ошибок, связанных с дублированием идентификаторов. Таким образом, триггеры помогают обеспечить целостность данных и улучшить эффективность работы с базой данных. Листинг создания триггеров представлен в приложении Б.

* 1. Директории

Для реализации экспорта и импорта в JSON создадим необходимую директорию. Реализация представлена дальше в листинге 3.6

|  |
| --- |
| CREATE DIRECTORY MY\_DIRECTORY AS 'C:\Humanix\Files\JSON'; |

Листинг 3.6 – Создание директории для хранения результатов импорта

В ней будут создаваться файлы JSON экспорта и импорта. Объект Directory является логической ссылкой в базе данных на каталог файловой системы сервера, где установлена БД Oracle. Владельцем всех объектов Directory в базе данных является пользователь SYS, даже если объект Directory создан другим пользователем. Имена объектов Directory уникальны внутри всей БД. Все объекты Directory хранятся в табличном пространстве SYS.

1. Описание процедур импорта и экспорта
   1. Экспорт в JSON

Процедура export\_json предназначена для экспорта данных из таблицы employees в формате JSON. Процедура открывает файл employees\_export.json для записи и затем проходит по всем записям в таблице employees. Каждая запись преобразуется в JSON-объект с помощью функции JSON\_OBJECT, где каждому полю записи соответствует ключ JSON-объекта. Полученный JSON-объект записывается в файл. После обработки всех записей файл закрывается. Если в процессе работы процедуры происходит ошибка, файл также закрывается, и ошибка передается вызывающей стороне. Это обеспечивает корректное завершение работы с файлом даже в случае возникновения ошибок. Скрипт создания процедуры export\_json представлен на листинге 4.1.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE *export\_json* AS  v\_file UTL\_FILE.FILE\_TYPE;  v\_data employees%ROWTYPE;  v\_json VARCHAR2(32767); BEGIN  v\_file := UTL\_FILE.*FOPEN*('MY\_DIRECTORY', 'employees\_export.json', 'w', 32767); FOR v\_data IN (SELECT *\** FROM employees) LOOP  *-- Преобразуем каждую запись в JSON* v\_json := *JSON\_OBJECT*(  'emp\_id' VALUE v\_data.emp\_id,  'first\_name' VALUE v\_data.first\_name,  'last\_name' VALUE v\_data.last\_name,  'email' VALUE v\_data.email,  'phone\_number' VALUE v\_data.phone\_number,  'hire\_date' VALUE *TO\_CHAR*(v\_data.hire\_date, 'YYYY-MM-DD'),  'job\_id' VALUE v\_data.job\_id,  'salary' VALUE v\_data.salary,  'commission\_pct' VALUE v\_data.commission\_pct,  'manager\_id' VALUE v\_data.manager\_id,  'department\_id' VALUE v\_data.department\_id  ); UTL\_FILE.*PUT\_LINE*(v\_file, v\_json);*-- Записываем JSON в файл*  END LOOP; UTL\_FILE.*FCLOSE*(v\_file); *-- Закрываем файл* EXCEPTION  WHEN OTHERS THEN  *-- Если произошла ошибка, закрываем файл* IF UTL\_FILE.*IS\_OPEN*(v\_file) THEN  UTL\_FILE.*FCLOSE*(v\_file);  END IF;  RAISE; END *export\_json*; / |

Листинг 4.1 – Процедура экспорта

Этот файл можно использовать для обмена данными между различными системами или для резервного копирования данных.

* 1. Импорт из JSON

Процедура import\_json предназначена для импорта данных в формате JSON в таблицу employees. Процедура открывает файл employees\_import.json для чтения и считывает все строки в переменную v\_data. Затем эти данные преобразуются в массив JSON-объектов. Процедура проходит по каждому JSON-объекту в массиве, извлекает данные и вставляет их в таблицу employees. Если в процессе работы процедуры происходит ошибка, все изменения откатываются, и ошибка передается вызывающей стороне. Это обеспечивает корректное завершение работы с файлом и таблицей даже в случае возникновения ошибок. Скрипт создания процедуры import\_json представлен на листинге 4.2.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE *import\_json* IS  v\_file UTL\_FILE.FILE\_TYPE;  v\_data CLOB;  v\_line VARCHAR2(32767);  v\_json JSON\_ARRAY\_T;  v\_json\_obj JSON\_OBJECT\_T;  v\_emp\_rec employees%ROWTYPE; BEGIN v\_file := UTL\_FILE.*FOPEN*('MY\_DIRECTORY', 'employees\_import.json', 'r'); BEGIN  LOOP  UTL\_FILE.*GET\_LINE*(v\_file, v\_line);  v\_data := v\_data || v\_line;  END LOOP;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  NULL; *-- Игнорируем исключение, так как это означает, что мы достигли конца файла* END;UTL\_FILE.*FCLOSE*(v\_file); v\_json := JSON\_ARRAY\_T.*parse*(v\_data); FOR i IN 0 .. v\_json.*get\_size*() - 1 LOOP  v\_json\_obj := *JSON\_OBJECT\_T*(v\_json.*get*(i));  v\_emp\_rec.emp\_id := v\_json\_obj.*get\_number*('emp\_id');  v\_emp\_rec.first\_name := v\_json\_obj.*get\_string*('first\_name');  v\_emp\_rec.last\_name := v\_json\_obj.*get\_string*('last\_name');  v\_emp\_rec.email := v\_json\_obj.*get\_string*('email');  v\_emp\_rec.phone\_number := v\_json\_obj.*get\_string*('phone\_number');  v\_emp\_rec.hire\_date := *TO\_DATE*(v\_json\_obj.*get\_string*('hire\_date'), 'YYYY-MM-DD');  v\_emp\_rec.job\_id := v\_json\_obj.*get\_string*('job\_id');  v\_emp\_rec.salary := v\_json\_obj.*get\_number*('salary');  v\_emp\_rec.commission\_pct := v\_json\_obj.*get\_number*('commission\_pct');  v\_emp\_rec.manager\_id := v\_json\_obj.*get\_number*('manager\_id');  v\_emp\_rec.department\_id := v\_json\_obj.*get\_number*('department\_id');  v\_emp\_rec.password\_hash := v\_json\_obj.*get\_string*('password\_hash');   INSERT INTO employees VALUES v\_emp\_rec;  END LOOP;  COMMIT; EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE; END *import\_json*; / |

Листинг 4.2 – Процедура импорта

1. Тестирование производительности

Для тестирования производительности мы добавляем в таблицу Employees 100000 строк. Скрипт вставки представлен на листинге 5.1.

|  |
| --- |
| DECLARE v\_counter NUMBER := 0; BEGIN  WHILE v\_counter <= 100000 LOOP  INSERT INTO employees (emp\_id, first\_name, last\_name, email, phone\_number, hire\_date, job\_id, salary, commission\_pct, manager\_id, department\_id) VALUES (  v\_counter,  DBMS\_RANDOM.*STRING*('A', 10),  DBMS\_RANDOM.*STRING*('A', 10),  'email' || *TO\_CHAR*(v\_counter) || '@example.com',  '1234567890',  *SYSDATE*,  'AD\_PRES',  DBMS\_RANDOM.*VALUE*(2000, 8000),  DBMS\_RANDOM.*VALUE*(0, 0.2),  null,  10  );  v\_counter := v\_counter + 1;  END LOOP; END;/ |

Листинг 5.1 – Вставка 100000 строк в таблицу

Для получения плана выполнения SQL-запроса в Oracle можно воспользоваться инструментом EXPLAIN PLAN используя функцию “Explain Plan”. [Результат будет отображаться в специальной вкладке “Plan” в формате, смешивающем древовидную структуру и таблицу.](https://www.jetbrains.com/help/datagrip/query-execution-plan.html) Результат представлен на рисунке 5.1.

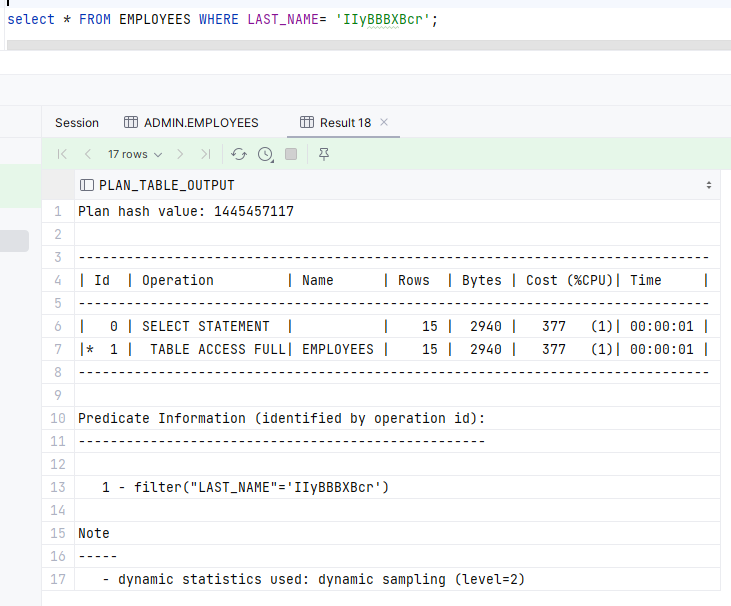


Рисунок 5.1 – Получения плана выполнения запроса

Каждая строка плана выполнения представляет операцию, которую база данных выполнит, а также ее стоимость, учитывая различные факторы, такие как использование индексов, объем данных и другие. Каждая операция оценивается по ожидаемому количеству возвращаемых строк. В данном случае план выполнения состоит из двух строк:

SELECT STATEMENT: это верхний уровень плана выполнения, который описывает операцию SELECT в целом.

TABLE ACCESS FULL: это операция, которая читает все строки таблицы ORDERS (без использования индексов), и возвращает их в качестве результата запроса. Стоимость этого плана выполнения равна 377 (без указания процента использования ресурсов ЦПУ) и означает, что база данных ожидает, что выполнение этого запроса займет примерно 377 единицы измерения работы (например, время CPU или количество операций ввода-вывода).

1. Описание технологии и ее применения в базе данных
   1. Шифрование данных в Oracle

Для шифрования данных в Oracle есть пакет DBMS\_CRYPTO позволяющий использовать алгоритмы симметричного и асимметричного шифрования, хэширования и цифровой подписи. DBMS\_CRYPTO поддерживает алгоритмы шифрования, такие как AES (Advanced Encryption Standard), DES (Data Encryption Standard), 3DES (Triple Data Encryption Standard), RSA (Rivest-Shamir-Adleman) и другие. В данном курсовом проекте мы будем использовать алгоритм AES (Advanced Encryption Standard) для шифрования и дешифрования данных в базе данных Oracle. 256 AES является одним из наиболее распространенных и надежных алгоритмов симметричного шифрования, который широко применяется в различных сферах, включая информационную безопасность, защиту данных и конфиденциальность. AES основан на блочном шифровании, где данные разбиваются на фиксированные блоки и каждый блок шифруется независимо. Алгоритм AES поддерживает различные длины ключей, включая 128, 192 и 256 бит, что позволяет выбирать уровень безопасности и производительности в соответствии с требованиями проекта.

* 1. Функции шифрования и дешифрование данных

Функция шифрования данных, используется, если в таблице нужно зашифровать конфиденциальные данные. Функция представлена в листинге 6.1

|  |
| --- |
| CREATE OR REPLACE FUNCTION *encrypt*(p\_plain\_text VARCHAR2) RETURN RAW IS  encryption\_key RAW(256) := *HEXTORAW*('0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF');  encrypted\_raw RAW(2048); BEGIN  encrypted\_raw := DBMS\_CRYPTO.*ENCRYPT*(  src => UTL\_I18N.*STRING\_TO\_RAW*(p\_plain\_text, 'AL32UTF8'),  typ => DBMS\_CRYPTO.ENCRYPT\_AES256 + DBMS\_CRYPTO.CHAIN\_CBC + DBMS\_CRYPTO.PAD\_PKCS5,  key => encryption\_key  );  RETURN encrypted\_raw; END *encrypt*; / |

Листинг 6.1 – Функция для шифрования данных

Реализация функции дешифрования представлены в листинге 6.2.

|  |
| --- |
| CREATE OR REPLACE FUNCTION *decrypt*(p\_encrypted\_text RAW) RETURN VARCHAR2 IS  encryption\_key RAW(256) := *HEXTORAW*('0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF');  decrypted\_raw RAW(2048); BEGIN  decrypted\_raw := DBMS\_CRYPTO.*DECRYPT*(  src => p\_encrypted\_text,  typ => DBMS\_CRYPTO.ENCRYPT\_AES256 + DBMS\_CRYPTO.CHAIN\_CBC + DBMS\_CRYPTO.PAD\_PKCS5,  key => encryption\_key  );  RETURN UTL\_I18N.*RAW\_TO\_CHAR*(decrypted\_raw, 'AL32UTF8'); END *decrypt*; / |

Листинг 6.2 – Функция для дешифрования данных

* 1. Аудит SYS операций

По умолчанию в базе данных Oracle не активирован аудит исполняемых команд SQL, выполняемый в привилегированном режиме от учетной записи SYS, а также при подключении пользователей с привилегиями (учетными записями) SYSDBA или SYSOPER. Так если ваша база данных будет взломана, эти учетные записи будут использованы хакерами в первую очередь. Но, к счастью аудит SQL-команд из этих привилегированных учетных записей очень просто включить следующей командой, представленной на листинге 6.3.

|  |
| --- |
| alter system set audit\_sys\_operations=true scope=spfile; |

Листинг 6.3 – Включение аудита команд

* 1. Включение аудита базы данных

Опять же, по умолчанию в Oracle аудит команд SQL не включен по умолчанию. По рекомендациям разработка аудит должен быть включен для всех команд SQL. Аудит базы данных включается с применением параметра audit\_trail. Команда представлена на листинге 6.4.

|  |
| --- |
| alter system set audit\_trail=DB,EXTENDED scope=spfile; |

Листинг 6.4 – Применение параметра audit\_trail

* 1. Защита словаря данных

Пользователи, которым предоставлены системные полномочия ANY, могут удалять таблицы словаря данных. Чтобы защитить словарь данных, конфигурационный параметр 07\_DICTIONARY\_ACCESSIBILITY в файле параметров необходимо установить в FALSE. Это ограничит выдачу полномочий ANY только тем пользователям, которые регистрируются с полномочиями SYSDBA. Подключение к базе данных с полномочиями SYSDBA. Изменение параметра представлено на листинге 6.5.

|  |
| --- |
| ALTER SYSTEM SET "\_07\_DICTIONARY\_ACCESSIBILITY" = FALSE SCOPE=SPFILE; *-- Перезапуск базы данных для применения изменений* SHUTDOWN IMMEDIATE; STARTUP; |

Листинг 6.5 – Изменение параметра 07\_DICTIONARY\_ACCESSIBILITY

* 1. Защита слушателя

Как было показано в статье блогов, для слушателя всегда следует применять пароль, чтобы воспрепятствовать подключению неправомочных пользователей к базе данных Oracle. Помните, что организовать атаку на незащищенную службу TNS Listener довольно просто. Как только пароль для слушателя установлен, привилегированные действия, такие как остановка или запуск слушателя, не смогут выполняться без ввода соответствующего пароля. Можно также запретить пользователю применять команду SET для вмешательства в функции слушателя. Для этого потребуется добавить строку, представленную на листинге 6.6, в файл конфигурации listener.ora.

|  |
| --- |
| ADMIN\_RESTRICTIONS=ON |

Листинг 6.6 – Параметр ADMIN\_RESTRICTIONS

По умолчанию этот параметр установлен в значение false. Следует также избегать удаленного управления службой слушателя, поскольку ее пароль не шифруется при передаче по сети. Пароль слушателя хранится в файле listener.ora, поэтому нужно защитить этот файл.

1. Краткое описание приложения для демонстрации

В качестве интерфейса прикладного программирования был выбран обширный API-интерфейс — Windows Presentation Foundation (WPF), позволяющий создавать красивые и производительные приложения с разнообразными элементами управления и реагировать на различные действия пользователя.

Для работы с WPF использовался объектно-ориентированный язык программирования с С-подобным синтаксисом — С#, разработанный для создания приложений на платформе Microsoft .NET Framework.

Для удовлетворения проектируемой системы различным атрибутам качества применяются различные архитектурные шаблоны (паттерны). В разрабатываемом приложении используется архитектурный шаблон Model-View-ViewModel (MVVM).

Шаблон MVVM имеет три основных слоя: модель, которая представляет бизнес-логику приложения, представление пользовательского интерфейса, и представление-модель, в котором содержится вся логика построения графического интерфейса и ссылка на модель, поэтому он выступает в качестве модели для представления.

1. Руководство пользователя
   1. Пример работы приложения от лица пользователя

После открытия приложения пользователя встречает форма для входа, которая представлена на рисунке 7.1. Здесь пользователь может ввести свой логин и пароль для авторизации в системе. После успешной авторизации система автоматически определяет роль пользователя и предоставляет ему соответствующие возможности в за0432исимости от роли, которую он имеет.

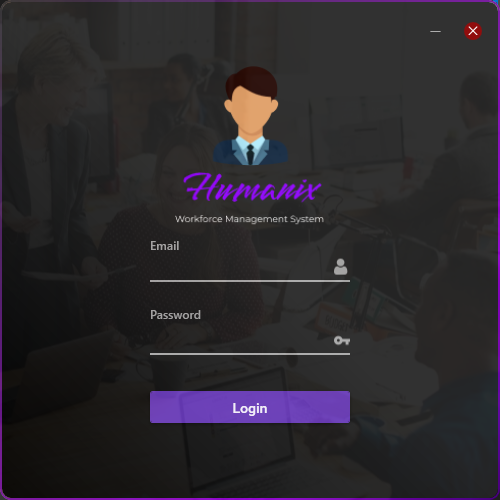


Рисунок 7.1 – Авторизация в приложении

После успешной авторизации в приложении открывается окно с профилем сотрудника. В данном окне пользователь имеет возможность отредактировать некоторую персональную информацию. Страница профиля представлена на рисунке 7.2.

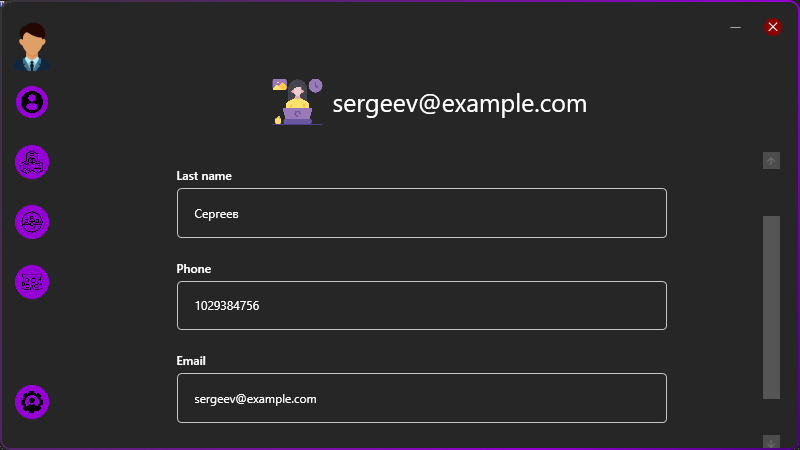


Рисунок 7.2 – Страница профиля

При помощи панели, представленной в левой части окна, пользователь имеет возможность открыть страницы со списком сотрудников в своем отделе, списком проектов, в которых он принимает участие, и списком отпусков.

* 1. Пример работы приложения от лица администратора

Если пользователь авторизован с правами администратора, то в панели, представленной в левой части окна, появляется дополнительный пункт меню, при нажатии на который открывается панель администратора. В ней, пользователь с права администратора имеет возможность просматривать содержимое всех таблиц, представленных в базе данных, а также, в случае таблицы сотрудников, увольнять, нанимать и повышать в должности. Панель администратора представлена на рисунке 7.3.

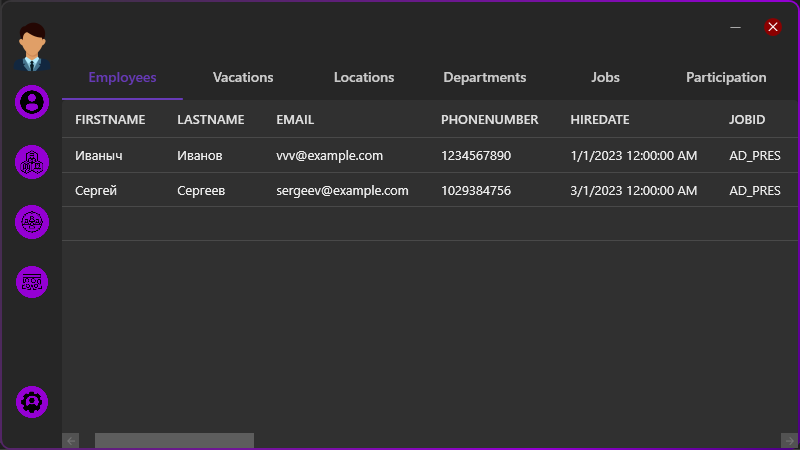


Рисунок 7.3 – Панель администратора

Заключение

В ходе выполнения курсового проекта была успешно разработана реляционная база данных для учета кадровых данных страховой компании с использованием системы управления базами данных Oracle 21c.

Был проведен анализ предметной области, в результате которого были определены основные сущности и атрибуты базы данных. На основе этого анализа была разработана концептуальная модель базы данных в нотации ER-диаграмм, которая затем была преобразована в логическую модель в нотации реляционной алгебры.

Эта логическая модель была реализована в физической модели с использованием Oracle. База данных была наполнена тестовыми данными для проверки ее работоспособности и эффективности.

Для реализации бизнес-логики базы данных были разработаны хранимые процедуры, функции и триггеры. Эти триггеры обеспечивают автоматическое присвоение уникальных идентификаторов при вставке новых записей в таблицы, что упрощает процесс вставки данных и уменьшает вероятность ошибок, связанных с дублированием идентификаторов.

Были разработаны запросы для получения необходимой информации из базы данных, что позволяет быстро и эффективно получать данные для анализа и принятия решений.

Таким образом, цель данной работы была успешно достигнута. Разработанная база данных обеспечивает эффективное управление и хранение информации о персонале страховой компании, обеспечивает быстрый доступ к хранимой информации и упрощает процесс учета кадровых данных.

Список используемых источников

1. Фейерштуйн С., Прибыл Б. Ф36 Oracle PL/SQL. Для профессионалов. 6-е издание — СПб.: Питер, 2015. — 1024 с.: ил. — (Серия «Бестселлеры O’Reilly»). Стр. 486 – 752 . – Дата доступа: 15.09.2023.
2. Нанда А., Фейерштейн С. Oracle PL/SQL для администраторов баз данных. СПб: Символ-Плюс, 2008. Стр. 114 – 148. – Дата доступа: 27.09.2023.
3. Работа с данными формата Xml в Oracle [Электронный ресурс]. – Режим доступа: https://docs.oracle.com/en/database/oracle/oracle-database/19/arpls/DBMS\_XMLGEN.html. – Дата доступа: 28.09.2023.
4. Oracle Advanced Analytics: анализ «больших данных» [Электронный ресурс]. – Режим доступа: https://bytemag.ru/oracle-advanced-analytics-analiiz-bolshiih-dannie-8198/. – Дата доступа: 30.09.2023.

Приложение А

Листинг создания таблиц

|  |
| --- |
| *-- Создаем таблицу сотрудников с первичным ключом emp\_id* CREATE TABLE employees (  emp\_id NUMBER(6) PRIMARY KEY , *-- Идентификатор сотрудника* first\_name VARCHAR2(20), *-- Имя сотрудника* last\_name VARCHAR2(25) NOT NULL, *-- Фамилия сотрудника* email VARCHAR2(25) , *-- Электронная почта сотрудника* phone\_number VARCHAR2(15), *-- Номер телефона сотрудника* hire\_date DATE NOT NULL, *-- Дата найма сотрудника* job\_id VARCHAR2(10) NOT NULL, *-- Идентификатор должности сотрудника* salary NUMBER(8,2), *-- Зарплата сотрудника* commission\_pct NUMBER(2,2), *-- Процент комиссии сотрудника* manager\_id NUMBER(6), *-- Идентификатор менеджера сотрудника* department\_id NUMBER(4), *-- Идентификатор отдела сотрудника* password\_hash VARCHAR2(128)  *-- CONSTRAINT employees\_pk PRIMARY KEY (emp\_id, email)* ) TABLESPACE USERS\_TBS;  *-- Создаем таблицу должностей с первичным ключом job\_id* CREATE TABLE jobs (  job\_id VARCHAR2(10) PRIMARY KEY, *-- Идентификатор должности* job\_title VARCHAR2(35) NOT NULL, *-- Название должности* min\_salary NUMBER(6), *-- Минимальная зарплата для должности* max\_salary NUMBER(6) *-- Максимальная зарплата для должности* ) TABLESPACE USERS\_TBS;  *-- Создаем таблицу отделов с первичным ключом department\_id* CREATE TABLE departments (  department\_id NUMBER(4) PRIMARY KEY, *-- Идентификатор отдела* department\_name VARCHAR2(30) NOT NULL, *-- Название отдела* manager\_id NUMBER(6), *-- Идентификатор менеджера отдела* location\_id NUMBER(4) *-- Идентификатор местоположения отдела* ) TABLESPACE USERS\_TBS;  *-- Создаем таблицу местоположений с первичным ключом location\_id* CREATE TABLE locations (  location\_id NUMBER(4) PRIMARY KEY, *-- Идентификатор местоположения* street\_address VARCHAR2(40), *-- Улица местоположения* postal\_code VARCHAR2(12), *-- Почтовый индекс местоположения* city VARCHAR2(30) NOT NULL, *-- Город местоположения* state\_province VARCHAR2(25), *-- Область местоположения* country\_id CHAR(2) NOT NULL *-- Идентификатор страны местоположения* ) TABLESPACE USERS\_TBS;  *-- Создаем таблицу стран с первичным ключом country\_id* CREATE TABLE countries (  country\_id CHAR(2) PRIMARY KEY, *-- Идентификатор страны* country\_name VARCHAR2(40) NOT NULL *-- Название страны* ) TABLESPACE USERS\_TBS;  *-- Создаем таблицу проектов с первичным ключом project\_id* CREATE TABLE projects (  project\_id NUMBER(4) PRIMARY KEY, *-- Идентификатор проекта* project\_name VARCHAR2(50) NOT NULL, *-- Название проекта* start\_date DATE NOT NULL, *-- Дата начала проекта* end\_date DATE, *-- Дата окончания проекта (может быть null)* budget NUMBER(12,2), *-- Бюджет проекта* department\_id NUMBER(4) NOT NULL *-- Идентификатор отдела, ответственного за проект* ) TABLESPACE USERS\_TBS;  *-- Создаем таблицу задач с первичным ключом task\_id* CREATE TABLE tasks (  task\_id NUMBER(6) PRIMARY KEY, *-- Идентификатор задачи* task\_name VARCHAR2(100) NOT NULL, *-- Название задачи* description VARCHAR2(500), *-- Описание задачи* duration NUMBER(4) NOT NULL, *-- Продолжительность задачи в часах* project\_id NUMBER(4) NOT NULL *-- Идентификатор проекта, к которому относится задача* ) TABLESPACE USERS\_TBS;  *-- Создаем таблицу участия сотрудников в проектах с составным первичным ключом из emp\_id и project\_id* CREATE TABLE participation (  emp\_id NUMBER(6) NOT NULL, *-- Идентификатор сотрудника, участвующего в проекте* project\_id NUMBER(4) NOT NULL, *-- Идентификатор проекта, в котором участвует сотрудник* role VARCHAR2(50), *-- Роль сотрудника в проекте* hours NUMBER(4), *-- Количество часов, затраченных сотрудником на проект* PRIMARY KEY (emp\_id, project\_id) *-- Составной первичный ключ из emp\_id и project\_id* ) TABLESPACE USERS\_TBS;   CREATE TABLE vacations (  vacation\_id NUMBER(6) PRIMARY KEY, *-- Идентификатор отпуска* emp\_id NUMBER(6) NOT NULL, *-- Идентификатор сотрудника* start\_date DATE NOT NULL, *-- Дата начала отпуска* end\_date DATE NOT NULL, *-- Дата окончания отпуска* reason VARCHAR2(500) *-- Причина отпуска* ) TABLESPACE USERS\_TBS; *-- Создаем внешние ключи для связи между таблицами  -- Связываем таблицу employees с таблицей jobs по атрибуту job\_id* ALTER TABLE employees ADD CONSTRAINT fk\_employees\_jobs FOREIGN KEY (job\_id) REFERENCES jobs (job\_id);  *-- Связываем таблицу employees с самой собой по атрибуту manager\_id* ALTER TABLE employees ADD CONSTRAINT fk\_employees\_managers FOREIGN KEY (manager\_id) REFERENCES employees (emp\_id);  *-- Связываем таблицу employees с таблицей departments по атрибуту department\_id* ALTER TABLE employees ADD CONSTRAINT fk\_employees\_departments FOREIGN KEY (department\_id) REFERENCES departments (department\_id);  *-- Связываем таблицу departments с таблицей locations по атрибуту location\_id* ALTER TABLE departments ADD CONSTRAINT fk\_departments\_locations FOREIGN KEY (location\_id) REFERENCES locations (location\_id);  *-- Связываем таблицу locations с таблицей countries по атрибуту country\_id* ALTER TABLE locations ADD CONSTRAINT fk\_locations\_countries FOREIGN KEY (country\_id) REFERENCES countries (country\_id);  *-- Связываем таблицу projects с таблицей departments по атрибуту department\_id* ALTER TABLE projects ADD CONSTRAINT fk\_projects\_departments FOREIGN KEY (department\_id) REFERENCES departments (department\_id);  *-- Связываем таблицу tasks с таблицей projects по атрибуту project\_id* ALTER TABLE tasks ADD CONSTRAINT fk\_tasks\_projects FOREIGN KEY (project\_id) REFERENCES projects (project\_id);  *-- Связываем таблицу participation с таблицей employees по атрибуту emp\_id* ALTER TABLE participation ADD CONSTRAINT fk\_participation\_employees FOREIGN KEY (emp\_id) REFERENCES employees (emp\_id);  *-- Связываем таблицу participation с таблицей projects по атрибуту project\_id* ALTER TABLE participation ADD CONSTRAINT fk\_participation\_projects FOREIGN KEY (project\_id) REFERENCES projects (project\_id);  *-- Связываем таблицу vacations с таблицей employees по атрибуту emp\_id* ALTER TABLE vacations ADD CONSTRAINT fk\_vacations\_emp\_id FOREIGN KEY (emp\_id) REFERENCES employees(emp\_id); |

Приложение Б

Листинг создания триггеров

|  |
| --- |
| *-- Создание триггеров* CREATE OR REPLACE TRIGGER emp\_id\_trg  BEFORE INSERT ON employees FOR EACH ROW  BEGIN  SELECT emp\_id\_seq.NEXTVAL INTO :new.**emp\_id** FROM dual;  END; /  CREATE OR REPLACE TRIGGER job\_id\_trg  BEFORE INSERT ON jobs FOR EACH ROW  BEGIN  SELECT job\_id\_seq.NEXTVAL INTO :new.**job\_id** FROM dual;  END; /  CREATE OR REPLACE TRIGGER department\_id\_trg  BEFORE INSERT ON departments FOR EACH ROW  BEGIN  SELECT department\_id\_seq.NEXTVAL INTO :new.**department\_id** FROM dual;  END; / CREATE OR REPLACE TRIGGER location\_id\_trg  BEFORE INSERT ON locations FOR EACH ROW  BEGIN  SELECT location\_id\_seq.NEXTVAL INTO :new.**location\_id** FROM dual;  END; / CREATE OR REPLACE TRIGGER country\_id\_trg  BEFORE INSERT ON countries FOR EACH ROW  BEGIN  SELECT country\_id\_trg.NEXTVAL INTO :new.**country\_id** FROM dual;  END; / CREATE OR REPLACE TRIGGER project\_id\_trg  BEFORE INSERT ON projects FOR EACH ROW  BEGIN  SELECT project\_id\_seq.NEXTVAL INTO :new.**project\_id** FROM dual;  END; / CREATE OR REPLACE TRIGGER task\_id\_trg  BEFORE INSERT ON tasks FOR EACH ROW  BEGIN  SELECT task\_id\_seq.NEXTVAL INTO :new.**task\_id** FROM dual;  END; /  CREATE OR REPLACE TRIGGER vacation\_id\_trg  BEFORE INSERT ON vacations FOR EACH ROW  BEGIN  SELECT vacation\_id\_seq.NEXTVAL INTO :new.**vacation\_id** FROM dual;  END; / |