Содержание

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. Постановка задачи

1.1 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 – 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.

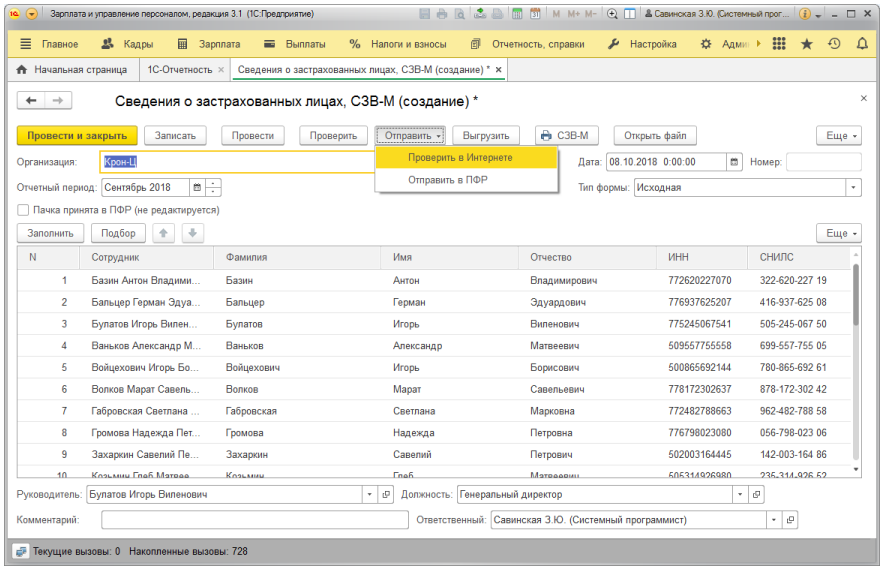


Рисунок 1.1 – интерфейс 1С:Зарплата и Кадры

Microsoft Dynamics 365 Human Resources – 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.

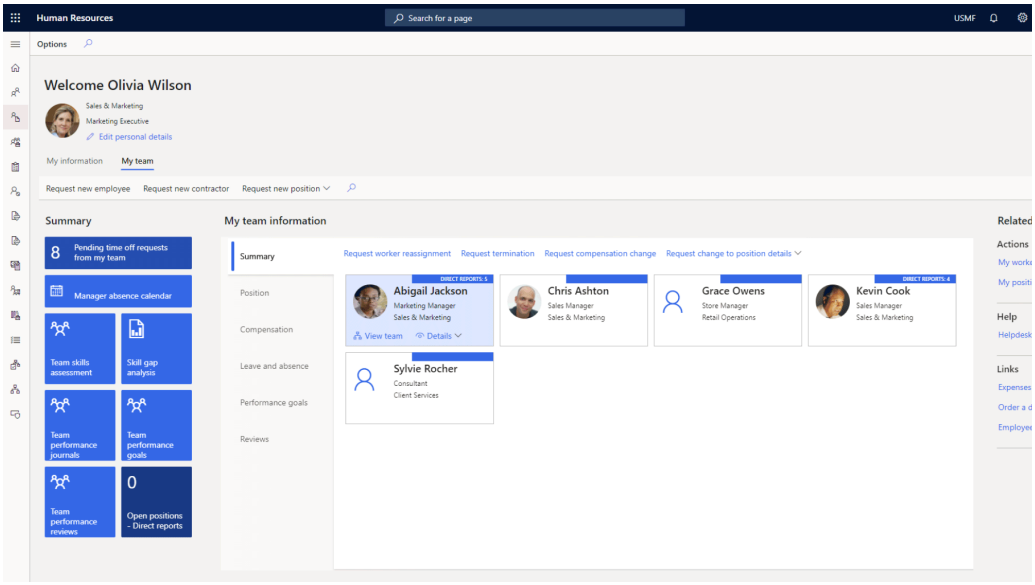


Рисунок 1.2 – интерфейс Microsoft Dynamics 365 Human Resources

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.

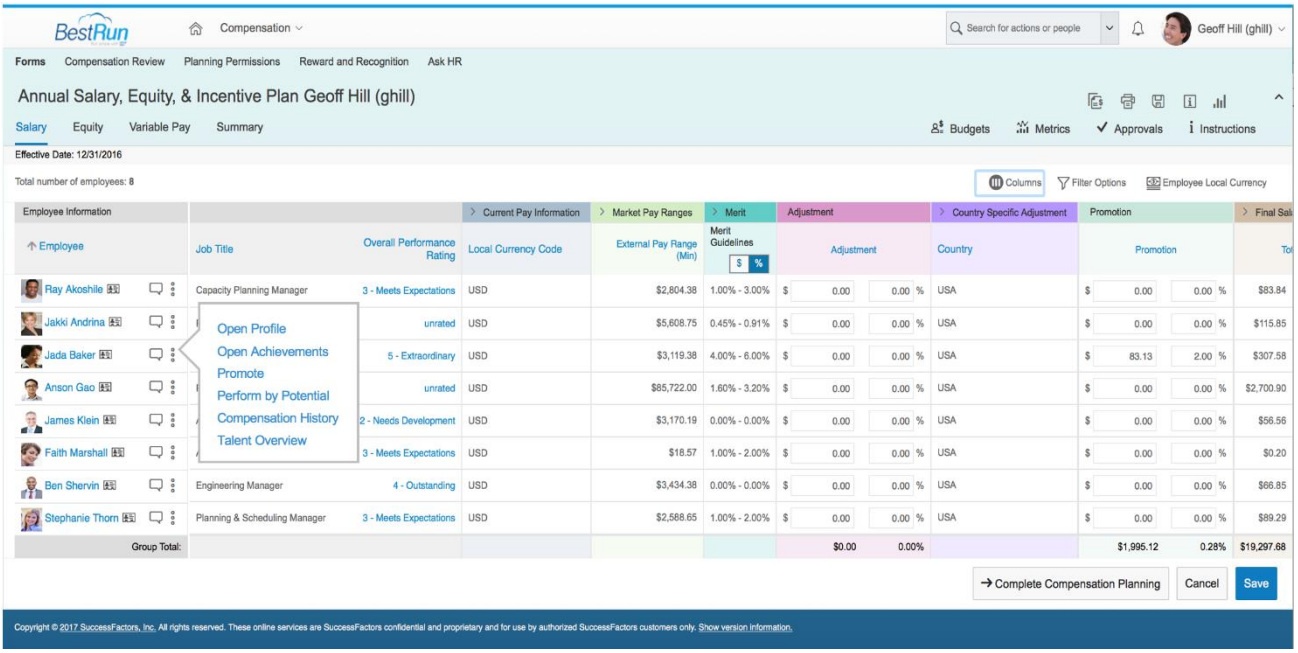


Рисунок 1.4 – интерфейс SAP SuccessFactors

Таким образом, можно сделать вывод, что существующие аналоги базы данных для учета кадровых данных имеют свои преимущества и недостатки, и не полностью удовлетворяют потребностям современных предприятий.

1.2 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 используется 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.

1.3 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.

2. Проектирование базы данных

2.1 Схема базы данных

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

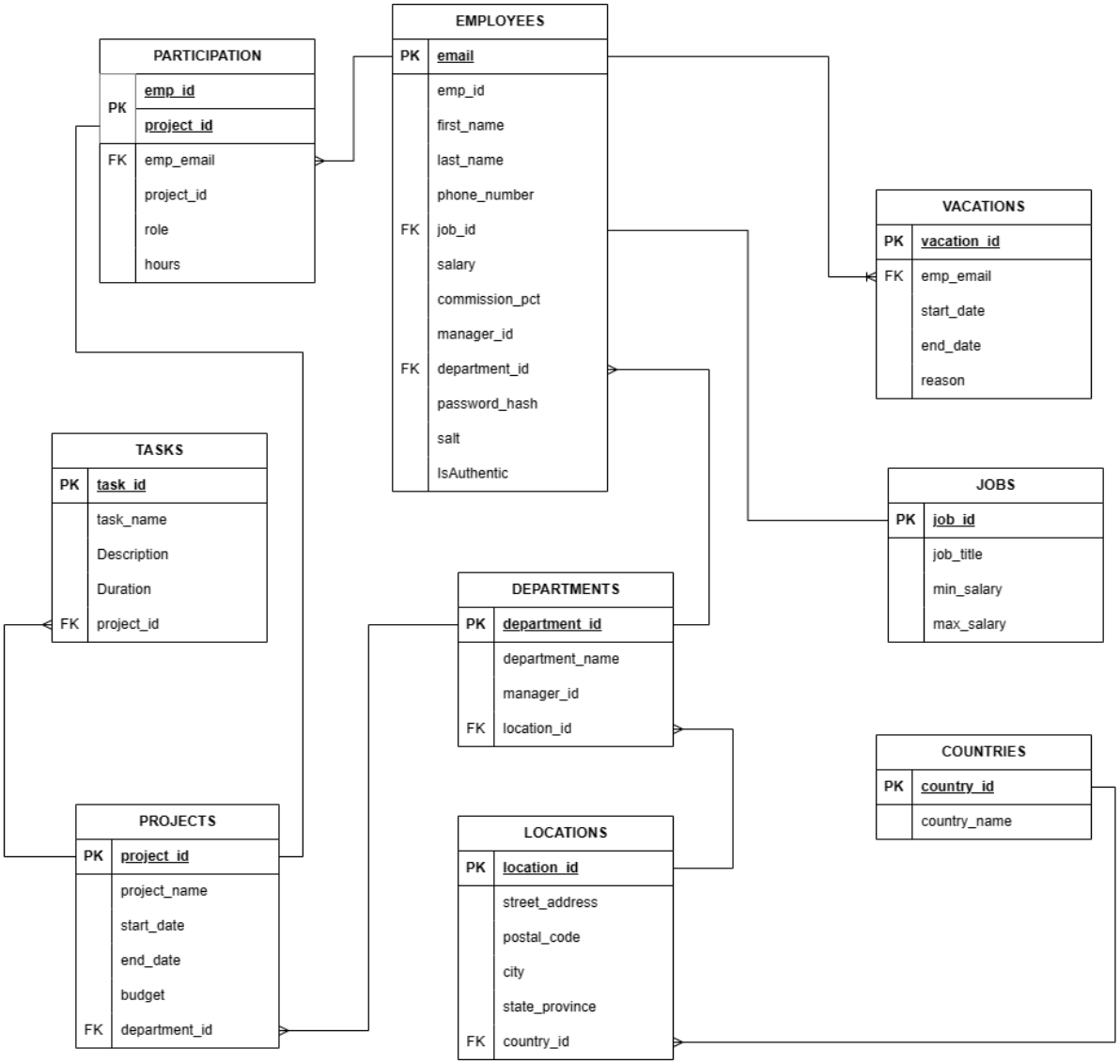


Рисунок 2.1 – UML диаграмма базы данных

Для наглядного представления работы пользователей с такой схемой базы данных в приложении А представлена диаграмма вариантов использования продукта.

2.2 Таблицы базы данных

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

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
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
salt	VARCHAR2(128)	Соль для пароля
IsAuthentic	NUMBER(1) NOT NULL	Активирован ли аккаунт

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
max_salary	NUMBER(6)	Максимальная зарплата

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
City	VARCHAR2(30) NOT NULL	Город местоположения
state_province	VARCHAR2(25)	Область местоположения
country_id	CHAR(2) NOT NULL	Идентификатор страны

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
end_date	DATE	Дата окончания проекта (может быть null)
start_date	DATE NOT NULL	Дата начала проекта

Продолжение таблицы 2.6

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_email	VARCHAR2(25)	Электронная почта сотрудника
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
Reason	VARCHAR2(500)	Причина отпуска
emp_email	VARCHAR2(25)	Электронная почта сотрудника
start_date	DATE NOT NULL	Дата начала отпуска
end_date	DATE NOT NULL	Дата окончания отпуска

Эта таблица хранит информацию о отпусках сотрудников.

3. Разработка объектов базы данных

3.1 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, последовательностей и т.д. Пользователю будут выданы права на вызов получение данных из определённых таблиц. С выданными правами 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;

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

3.2 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.

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 hire_employee 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 hire_employee (p_first_name VARCHAR2,p_last_name
VARCHAR2,p_email VARCHAR2,p_phone_number VARCHAR2,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;
     v_salt employees.salt%TYPE;
BEGIN
    INSERT INTO employees (
        emp_id, first_name, last_name, email, phone_number,
job_id, salary, commission_pct, manager_id, department_id,
password_hash, salt, ISAUTHENTIC
    ) VALUES (
        v_emp_id, p_first_name, p_last_name, p_email,
p_phone_number, p_job_id, p_salary, p_commission_pct,
```

```
p_manager_id, p_department_id, ENCRYPT(p_password_hash, v_salt),
v_salt, 0
    );
    COMMIT;
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK; RAISE;
END hire_employee;
/
```

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, для всех таблиц были реализованы процедуры, которые обеспечивают базовые операции управления данными в базе данных, такие как вставка, обновление и удаление данных. Некоторые вспомогательные процедуры 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
generate_salt(p_email VARCHAR2)	генерирует соль на основе электронной почты сотрудника
export_json()	экспортирует данные из таблицы employees в файл JSON
import_json()	импортирует данные из файла JSON в таблицу employees

3.3 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.

3.4 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 последовательностей
CREATE SEQUENCE department_id_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE location_id_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE project_id_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE task_id_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE emp_id_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE vacation_id_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE participation_id_seq START WITH 1 INCREMENT BY 1;
```

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

3.5 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.

3.6 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.

4. Описание процедур импорта и экспорта

4.1 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);
    v_count NUMBER := 0;
    v_total NUMBER := 0;
BEGIN
    -- Получаем общее количество записей
    SELECT COUNT(*) INTO v_total FROM employees;

    -- Открываем файл для записи
    v_file := UTL_FILE.FOPEN('MY_DIRECTORY', 'employees.json',
'w', 32767);

    -- Записываем открывающую квадратную скобку
    UTL_FILE.PUT_LINE(v_file, '[');

    -- Начинаем цикл по всем записям в таблице employees
    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,
            '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,
            'password_hash' VALUE v_data.PASSWORD_HASH,
            'isauthentic' value v_data.ISAUTHENTIC
```

```
        );

     -- Увеличиваем счетчик
        v_count := v_count + 1;

        -- Записываем JSON в файл
        IF v_count < v_total THEN
            UTL_FILE.PUT_LINE(v_file, v_json || ',');
        ELSE
            UTL_FILE.PUT_LINE(v_file, v_json);
        END IF;

        -- Записываем JSON в файл
--         UTL_FILE.PUT_LINE(v_file, v_json);
    END LOOP;


    -- Записываем закрывающую квадратную скобку
    UTL_FILE.PUT_LINE(v_file, ']');
    -- Закрываем файл
    UTL_FILE.FCLOSE(v_file);
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.

4.2 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
    -- Открываем файл для чтения
    v_file := UTL_FILE.FOPEN('MY_DIRECTORY', 'employees.json',
'r');

    -- Читаем данные из файла
    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;

    -- Закрываем файл
    UTL_FILE.FCLOSE(v_file);

    -- Парсим JSON
    v_json := JSON_ARRAY_T.parse(v_data);

    -- Извлекаем данные из JSON и вставляем их в таблицу
    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.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');
        v_emp_rec.ISAUTHENTIC :=
v_json_obj.get_string('isauthentic');

        INSERT INTO employees VALUES v_emp_rec;
    END LOOP;

    COMMIT;
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        RAISE;
END import_json;
/
```

Листинг 4.2 – Процедура импорта

В случае возникновения ошибки процедура откатывает изменения.

5. Тестирование производительности

Для проверки производительности базы данных необходимо заполнить ее большим количеством различных данных и узнать время выполнения одного запроса. 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 (first_name, last_name, email,
phone_number, job_id, salary, commission_pct, manager_id,
department_id, ISAUTHENTIC) VALUES (
      DBMS_RANDOM.STRING('A', 10),
      DBMS_RANDOM.STRING('A', 10),
      'email' || TO_CHAR(v_counter) || '@example.com',
      '1234567890',
      'AD_PRES',
      DBMS_RANDOM.VALUE(2000, 8000),
      DBMS_RANDOM.VALUE(0, 0.2),
      null,
      10,
      0
    );
    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. Результат представлен на рисунке 5.1.

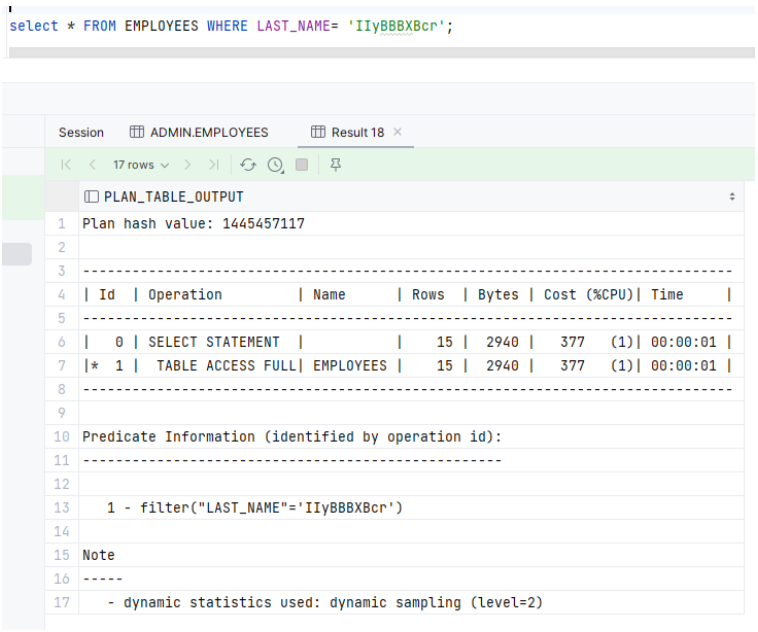


Рисунок 5.1 – Получения плана выполнения запроса

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).

Проведенный анализ производительности базы данных с таблицей SERVICE_TYPES, содержащей большое количество строк, позволяет сделать следующие выводы. Первоначально, до создания индексов, запрос к таблице осуществлялся полным сканированием, что может привести к замедлению при работе с большим объемом данных.

Создание индексов на соответствующих столбцах таблицы существенно улучшило время выполнения запроса. Индексы позволяют эффективнее организовывать доступ к данным, снижая стоимость выполнения операций. Таким образом, база данных оказывается более подготовленной к обработке больших объемов данных, что является важным аспектом в условиях активной работы системы с множеством запросов и операций.

6. Описание технологии и ее применения в базе данных

6.1 Шифрование данных в Oracle

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

```
CREATE OR REPLACE FUNCTION encrypt(p_plain_text VARCHAR2, p_salt
VARCHAR2) RETURN RAW IS
    encryption_key RAW(256) :=
HEXTORAW('0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456
789ABCDEF');
    encrypted_raw RAW(2048);
BEGIN
    encrypted_raw := DBMS_CRYPTO.ENCRYPT(
        src => UTL_I18N.STRING_TO_RAW(p_plain_text || p_salt,
'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, p_salt
VARCHAR2) RETURN VARCHAR2 IS
```

```
    encryption_key RAW(256) :=
HEXTORAW('0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456
789ABCDEF');
    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 separate_string(UTL_I18N.RAW_TO_CHAR(decrypted_raw,
'AL32UTF8'), p_salt);
END decrypt;
```

Листинг 6.2 – Функция для дешифрования данных

6.2 Аутентификация

Аутентификация пользователей является важной частью любой системы управления базами данных. Это процесс проверки идентификации пользователя, который пытается получить доступ к системе. Один из распространенных методов аутентификации - это аутентификация посредством электронной почты.

Аутентификация посредством электронной почты - это процесс, при котором пользователь вводит свой адрес электронной почты и пароль для входа в систему. Этот метод обеспечивает уровень безопасности, поскольку только пользователь, знающий правильный адрес электронной почты и пароль, сможет получить доступ.

После найма сотрудника на работу и создания его аккаунта администратором, пользователь обязан активироваться в системе посредством ввода аутентификационного кода, полученного по электронной почте.

6.3 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 база данных будет взломана, эти учетные записи будут использованы хакерами в первую очередь. Дабы этого избежать необходимо включить аудит SQL-команд из этих привилегированных учетных записей следующей командой, представленной на листинге 6.3.

```
alter system set audit_sys_operations=true scope=spfile;
```

Листинг 6.3 – Включение аудита команд

6.4 Включение аудита базы данных

Опять же, 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

6.5 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

6.6 Защита слушателя

Для слушателя всегда следует применять пароль, чтобы воспрепятствовать подключению неправомочных пользователей к базе данных Oracle. 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.

7. 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 C-подобным синтаксисом — C#, 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.

8. Руководство пользователя

8.1 Пример работы приложения от лица пользователя

После открытия приложения пользователя встречает форма для входа, которая представлена на рисунке 8.1. 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 зависимости от роли, которую он имеет.

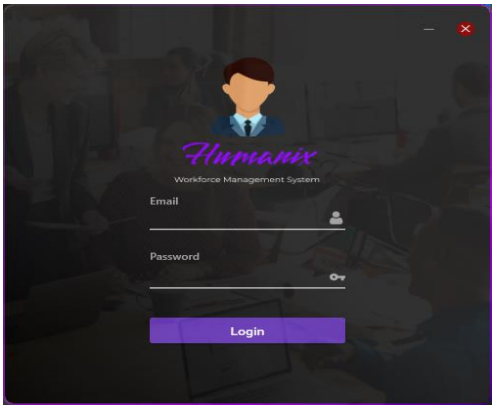


Рисунок 8.1 – Авторизация в приложении

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 8.2.

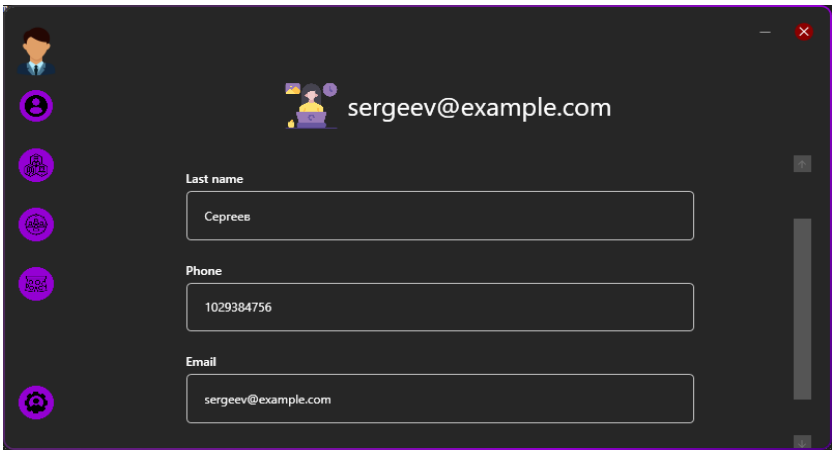


Рисунок 8.2 – Страница профиля

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.

8.2 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 8.3.

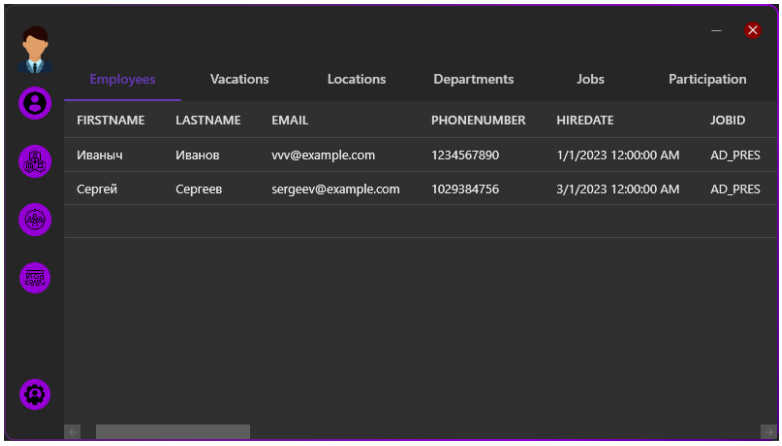


Рисунок 8.3 – Панель администратора

В случае авторизации без прав администратора, пользователь имеет доступ только к общим данным, таким как список сотрудников в его отделе отпуска и т.д. Пользователь не может влиять на состав и статус сотрудников, а также на другие аспекты работы системы.

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.

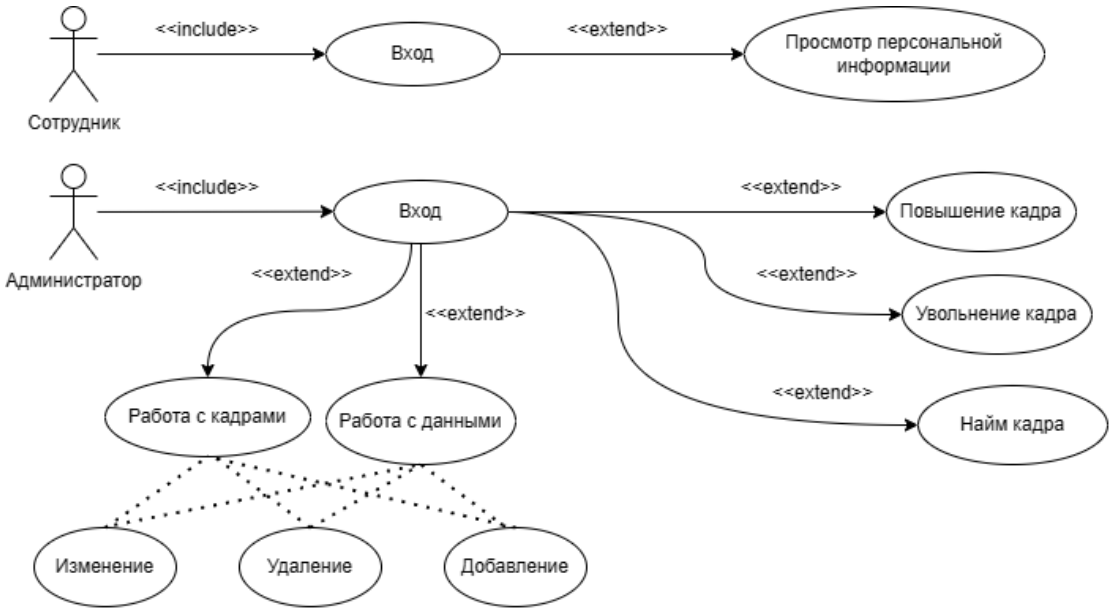
2. Нанда А., Фейерштейн С. Oracle PL/SQL для администраторов баз данных. СПб: Символ-Плюс, 2008. Стр. 114 – 148. – Дата доступа: 27.09.2023.

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

Диаграмма вариантов использования



Приложение Б

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)
    salt VARCHAR2(128), -- Соль для пароля сотрудника
    IsAuthentic NUMBER(1) NOT NULL
) 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 В

Листинг создания триггеров

```
CREATE OR REPLACE TRIGGER emp_id_trg
    BEFORE INSERT ON employees FOR EACH ROW
    BEGIN
        :NEW.salt := generate_salt(:NEW.email);
        SELECT emp_id_seq.NEXTVAL INTO :new.emp_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 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;/
```