МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ  
Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ   
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра информационных систем и технологий

Специальность 1-40 05 01 «Информационные системы и технологии»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА КУРСОВОГО ПРОЕКТА**

по дисциплине «Базы данных»

Тема: «Реализация базы данных для музыкальной площадки с использованием технологии применения мультимедийных типов данных»

**Исполнитель**

студент 2 курса 2 группы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.И. Клецкий

подпись, дата

**Руководитель**

ассистент \_\_\_\_\_\_\_\_\_\_\_\_ Н.И. Уласевич

(должность, уч. звание) (подпись, дата)

Допущен(а) к защите \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата, подпись

Содержание

[Введение 5](#_Toc166801716)

[1 Постановка задачи 6](#_Toc166801717)

[1.1 Описание функционала и объектов базы данных 6](#_Toc166801718)

[1.2 Описание используемых технологий 6](#_Toc166801719)

[2 Проектирование базы данных 8](#_Toc166801720)

[2.1 Описание структуры базы данных 8](#_Toc166801721)

[3 Разработка объектов базы данных 10](#_Toc166801722)

[3.1 Таблицы 10](#_Toc166801723)

[3.2 Процедуры 11](#_Toc166801724)

[3.3 Функции 13](#_Toc166801725)

[3.4 Тригеры 14](#_Toc166801726)

[3.5 Представления 14](#_Toc166801727)

[3.6 Роли 15](#_Toc166801728)

[3.7 Пользователи 16](#_Toc166801729)

[3.8 Последовательности 16](#_Toc166801730)

[4 Описание процедур импорта и экспорта 17](#_Toc166801731)

[4.1 Процедура импорта данных из JSON-файла 17](#_Toc166801732)

[4.2 Процедура экспорта данных в формате JSON 17](#_Toc166801733)

[5 Тестирование производительности 19](#_Toc166801734)

[6 Описание технологии и ее применения в базе данных 21](#_Toc166801735)

[7 Руководство пользователя 23](#_Toc166801736)

[Заключение 24](#_Toc166801737)

[Список используемых источников 25](#_Toc166801738)

[Приложение А Листинг создания таблиц 26](#_Toc166801739)

[Приложение Б Листинг создания процедур 28](#_Toc166801740)

[Приложение В Листинг создания функций 35](#_Toc166801741)

[Приложение Г Листинг создания триггеров 41](#_Toc166801742)

[Приложение Д Листинг процедуры для заполнения таблиц 43](#_Toc166801743)

# Введение

Современное хранение информации в основном осуществляется в цифровом формате, и для эффективного управления и хранения такой информации широко используются различные базы данных. В частности, базы данных успешно применяются в приложениях для хранения и управления данными.

В рамках данного учебно-проектного задания предлагается разработать базу данных для ресторана с использованием системы управления базами данных Oracle и технологии шифрования и маскирования данных. Для того чтобы база данных была эффективной и соответствовала целям проекта, ее данные должны отвечать следующим требованиям:

* Быть полезными, т.е. сокращать информационную энтропию системы;
* Обладать полнотой, достаточной для осуществления качественного управления;
* Обеспечивать точность, достоверность и непротиворечивость данных, т.е. в базе данных не должно быть заведомо ошибочных, противоречивых или устаревших данных;
* Быть актуальными, т.е. регулярно обновляться и поддерживаться в актуальном состоянии.
* Безопасность: база данных должна обладать высоким уровнем защиты от несанкционированного доступа и быть защищена от вирусов, злоумышленников и других угроз безопасности;
* Масштабируемость: база данных должна быть способна масштабироваться и расширяться в соответствии с ростом бизнеса и увеличением объема данных. Это позволит обеспечить стабильность и надежность работы базы данных в будущем.

В соответствии с заданием курсового проекта для проектирования базы данных ресторана используется система управления базами данных Oracle Database и технология шифрования и маскирования данных.

# Постановка задачи

* 1. **Описание функционала и объектов базы данных**

Целью данного курсового проекта является проектирование базы данных для блого-социальной сети с использованием технологии полнотекстового поиска. В качестве модели данных следует использовать реляционную модель. Проектирование необходимо произвести таким образом, чтобы конечные данные соответствовали общим требованиям к информации в базе данных

Функционал должен позволять:

* Регистрация нового пользователя;
* Добавление нового блога;
* Добавление нового комментария;
* Добавление нового поста;
* Изменение информации о пользователе;
* Изменение блогов;
* Изменения постов;
* Изменение комментариев;
* Удаление пользователей;
* Удаление блогов;
* Удаление постов;
* Удаление комментариев;
* Возможность ставить и убирать лайки;
* Возможность добавлять посты в избранное;
* Возможность убирать посты из избранного;
* Возможность подписываться на блог;
* Возможность отписаться от блога.

Для реализации функционала необходимо разработать ряд объектов базы данных. К таким объектам относятся:

* роли;
* пользователи;
* таблицы;
* представления;
* табличные пространства;
* индексы;
* хранимые процедуры;
* функции;
* триггеры.
  1. **Описание используемых технологий**

Существует огромное количество систем управления базами данных (СУБД), которые могут быть использованы для организации работы с данными в компьютерных системах. Некоторые из наиболее популярных СУБД включают в себя Oracle Database, MS SQL Server, NoSQL, MySQL, PostgreSQL и другие.

В данной работе было принято решение использовать Oracle Database в качестве СУБД, так как она предоставляет широкий спектр возможностей для оперирования объектами базы данных, включая поддержку стандарта SQL, хранение и управление табличными данными, поддержку транзакций и многое другое.

Одним из ключевых аспектов при работе с базами данных является безопасность. В процессе выбора технологии СУБД было уделено внимание технологиям шифрования данных. Шифрование может быть использовано для защиты данных от несанкционированного доступа путем кодирования данных с использованием алгоритмов шифрования.

# Проектирование базы данных

## Описание структуры базы данных

Для реализации поставленной задачи было создано табличное пространство SocNet. Диаграмма структуры полученной базы данных, разработанной в СУБД «Oracle database», представлена на рисунке 2.1. Для базы данных было разработано 8 таблиц, которые связаны друг с другом внешними ключами.

Таблица USER хранит данные о пользователях.

Таблица ROLE содержит информацию о ролях.

Таблица SUBSCRIOTION хранит информацию о подписках.

Таблица BLOG хранит информацию о блогах.

Таблица POST хранит информацию о постах.

Таблица COMMENT хранит информацию о комментариях.

Таблица LIKE содержит информацию о лайках.

Таблица FAVORITE хранит избранные посты.

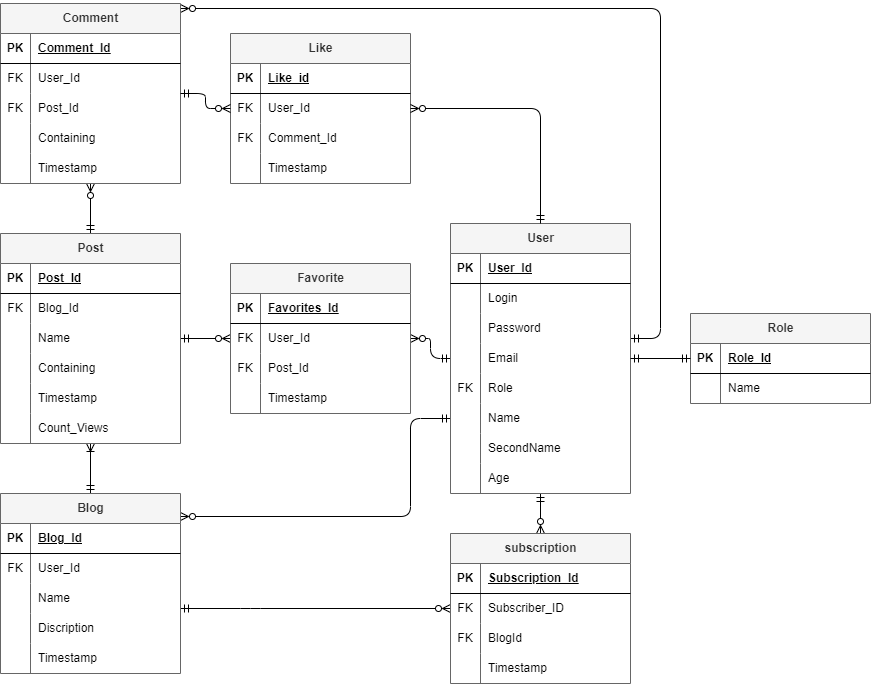


Рисунок 2.1 – Диаграмма структуры базы данных

Все таблицы связаны между собой с помощью внешних ключей, что позволяет эффективно и быстро обрабатывать данные и выполнять запросы к базе данных.

Для более наглядного представления возможностей каждой роли в системе, необходимо предоставить диаграмму вариантов использования продукта, показа она на рисунке 2.2. На данной диаграмме должны быть отображены все возможности, которые доступны конкретному пользователю с учетом его роли, будь это обычный пользователь или администратор.

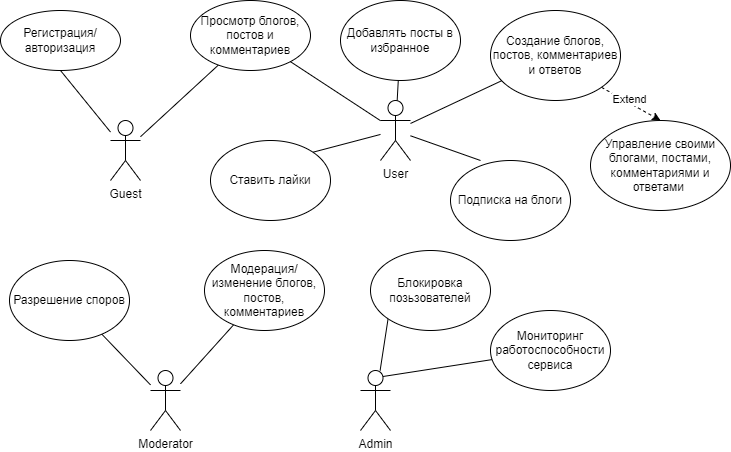


Рисунок 2.1 – Диаграмма вариантов использования

Такая диаграмма позволит легче воспринимать возможности каждой роли и понимать, какие возможности расширяются при переходе на более высокую роль в системе.

# Разработка объектов базы данных

## Таблицы

Таблицы являются неотъемлемой частью реляционной базы данных. Краткая характеристика каждой из таблиц была предоставлена в разделе 2, а код их создания можно увидеть в Приложении А. Ниже мы рассмотрим каждую таблицу подробнее.

Таблица USER состоит из восьми столбцов и представлена на таблице 3.1:

Таблица 3.1 «USER»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| USER\_ID | NUMBER | Хранит идентификатор пользователя |
| LOGIN | NVARCHAR2 | Хранит логин пользователя |
| PASSWORD | NVARCHAR2 | Хранит пароль пользователя |
| EMAIL | NVARCHAR2 | Хранит электронную почту пользователя |
| ROLE\_ID | NUMBER | Хранит роль пользователя, внешний ключ к таблице ROLE |
| NAME | NVARCHAR2 | Хранит имя пользователя |
| SECONDNAME | NVARCHAR2 | Хранит фамилию пользователя |
| AGE | NUMBER | Хранит возраст пользователя |

Таблица ROLE состоит из двух столбцов и показана на таблице 3.2:

Таблица 3.2 «ROLE»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| ROLE\_ID | NUMBER | Хранит идентификатор роли |
| NAME | VARCHAR2 | Хранит название роли |

Таблица BLOG состоит из четырех столбцов и представлена на таблице 3.3:

Таблица 3.3 «BLOG»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| BLOD\_ID | NUMBER | хранит идентификатор блога |
| USER\_ID | NUMBER | хранит идентификатор пользователя, внешний ключ к таблице USER |
| NAME | NVARCHAR2 | хранит название блога |
| DISCRIPTION | NVARCHAR2 | хранит описание блога |
| TIMRSTAMP | DATE | Хранит время создания блога |

Таблица POST состоит из шести столбцов и представлена на таблице 3.4:

Таблица 3.4 «POST»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| POST\_ID | NUMBER | хранит идентификатор поста |
| BLOG\_ID | NUMBER | хранит идентификатор блога, внешний ключ к таблице BLOG |
| NAME | NVARCHAR2 | Хранит название поста |
| CONTAINING | CLOB | Хранит содержание поста |
| TIMESTAMP | DATE | Хранит время создания |
| COUNTVIEWS | NUMBER | Хранит количество просмотров |

Таблица COMMENT состоит из пяти столбцов и представлена на таблице 3.5:

Таблица 3.5 «COMMENT»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| COMMENT\_ID | NUMBER | Хранит идентификатор комментария |
| USER\_ID | NUMBER | Хранит идентификатор пользователя, внешний ключ к таблице USER |
| POST\_ID | NUMBER | Хранит идентификатор поста, внешний ключ к таблице POST |
| CONTAINING | CLOB | Хранит содержание комментария |
| TIMESTAMP | DATE | Хранит время создания комментария |

Таблица LIKE состоит из четырех столбцов и представлена на таблице 3.6:

3.6 «LIKE»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| LIKE\_ID | NUMBER | Хранит идентификатор лайка |
| USER\_ID | NUMBER | Хранит идентификатор пользователя, внешний ключ к таблице USER |
| COMMENT\_ID | NUMBER | Хранит идентификатор комментария, внешний ключ к таблице COMMENT |
| TIMESTAMP | DATE | Хранит время создания |

Таблица FAVORITE состоит из четырех столбцов и представлена на таблице 3.7:

Таблица 3.7 «FAVORITE»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| FAVORITE\_ID | NUMBER | Хранит идентификатор избранного |
| USER\_ID | NUMBER | Хранит идентификатор пользователя, внешний ключ к таблице USER |
| POST\_ID | NUMBER | Хранит идентификатор поста, внешний ключ к таблице POST |
| TIMESTAMP | DATE | Хранит время создания |

Таблица SUBSCRIPTION состоит из четырёх столбцов и представлена на таблице 3.8:

Таблица 3.8 «SUBSCRIPTION»

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| SUBSCRIPTION\_ID | NUMBER | Хранит идентификатор подписки |
| SUBSCRIBER\_ID | NUMBER | Хранит идентификатор пользователя, внешний ключ к таблице USER |
| BLOG\_ID | NUMBER | Хранит идентификатор блога, внешний ключ к таблице BLOG |
| TIMESTAMP | DATE | Хранит время создания |

## Процедуры

Всего было разработано 17 процедуры для работы с данными. Краткое описание процедур описано далее.

Пример создания хранимой процедуры UPDUSERBUID, которая обновляет информацию о пользователе в листинге 3.1

|  |
| --- |
| CREATE OR REPLACE PROCEDURE UpdUserById (    p\_user\_id IN NUMBER,    p\_password IN VARCHAR2 DEFAULT NULL,    p\_email IN VARCHAR2 DEFAULT NULL,    p\_name IN VARCHAR2 DEFAULT NULL,    p\_secondname IN VARCHAR2 DEFAULT NULL,    p\_age IN NUMBER DEFAULT NULL,    p\_role\_id IN NUMBER DEFAULT NULL,    l\_updated\_count OUT NUMBER  ) AS    p\_hash\_password NVARCHAR2(255);  BEGIN    IF p\_password IS NOT NULL THEN       p\_hash\_password := hash\_password(p\_password);    ELSE p\_hash\_password := NULL;    END IF;    UPDATE User\_Table    SET password = NVL(p\_hash\_password, password),        email = NVL(p\_email, email),        name = NVL(p\_name, name),        secondname = NVL(p\_secondname, secondname),        age = NVL(p\_age, age),        role\_id = NVL(p\_role\_id, role\_id)    WHERE user\_id = p\_user\_id;      l\_updated\_count := SQL%ROWCOUNT;      IF l\_updated\_count > 0 THEN      DBMS\_OUTPUT.PUT\_LINE('Пользователь с ID: ' || p\_user\_id || ' обновлен.');    ELSE      DBMS\_OUTPUT.PUT\_LINE('Пользователь с ID: ' || p\_user\_id || ' не найден.');    END IF;  EXCEPTION      WHEN DUP\_VAL\_ON\_INDEX THEN        DBMS\_OUTPUT.PUT\_LINE('Ошибка: Пользователь с таким email уже существует.');        l\_updated\_count := -1;      WHEN OTHERS THEN      DBMS\_OUTPUT.PUT\_LINE('Ошибка: ' || SQLERRM);      l\_updated\_count := -1;  END UpdUserById; |

Листинг 3.1 – Процедура UPBUSERBYID

Данная процедура UPDUSERBYID предназначена для обновления информации о пользователе. Входными параметрами являются p\_user\_id – идентификатор пользователя, для которого необходимо обновить информацию, p\_password – новый пароль пользователя, p\_email – новую почту пользователя, p\_name – новое имя пользователя, p\_secondname – новая фамилия пользователя, p\_role\_id – новая роль пользователя, p\_age – новая дата рождения. Выходным параметром является p\_updated\_count – показывает успешность или неуспешность операции. Внутри процедуры осуществляется выборка из таблицы POST, где происходит поиск нужного поста и добавление к счетчику просмотров 1. Скрипт следующих процедур приложен в приложении Б.

ADDBLOG - процедура для добавления нового блога.

ADDCOMMENT - процедура для добавления нового комментария.

ADDPOST - процедура для добавления нового поста.

ADDUSER - процедура для добавления нового пользователя.

DELBLOG - процедура для удаления блога.

DELCOMMENT - процедура для удаления комментария.

DELPOST - процедура для удаления поста.

DELUSER - процедура для удаления пользователя.

SEND\_MESSAGE - процедура для оповещения пользователей.

UBDBLOG - процедура для обновления блога.

UBDCOMMENT - процедура для обновления комментария.

UPDFAVORITE - процедура для добавления или удаления избранного.

UPDLIKE - процедура для добавления или удаления лайка.

UPDPOST - процедура для обновления поста.

UPDSUVSCRIPTION - процедура для добавления или удаления подписки.

UPDUSER - процедура для обновления пользователя.

## Функции

Всего было разработано 19 функций для работы с данными. Краткое описание функций описано далее.

Пример создания хранимой функции GETCOUNTLIKE, которая добавляет блюда представлена в листинге 3.2

|  |
| --- |
| CREATE OR REPLACE FUNCTION GETCOUNTLIKE(    p\_com\_id IN NUMBER  ) RETURN INTEGER AS l\_count\_like INTEGER;  BEGIN    SELECT COUNT(\*) INTO l\_count\_like FROM    LIKE\_TABLE l WHERE l.COMMENT\_ID = p\_com\_id;    return l\_count\_like;  END; |

Листинг 3.2 – Функция GETCOUNTLIKE

Данная процедура GETCOUNTLIKE предназначена для подсчета количества лайков для комментария. Входным параметром является p\_com\_id - идентификатор комментария, для которого необходимо посчитать количество лайков. Выходным параметром является p\_count\_like – количество лайков.

Внутри процедуры осуществляется выборка из таблицы LIKE, где происходит поиск лайков для нужного комментария и считается их количество. Скрипт следующих функций приложен в приложении В.

GETALLBLOGS – функция для получения всех блогов.

GETALLPOSTS – функция для получения всех постов

GETBLOGBUID – функция для получения блога по идентификатору.

GETBLOGFORUSER – функция для получения всех блогов пользователя.

GETCOMMENTBYID функция для получения комментария по идентификатору.

GETCOMMENTSFORPOST - функция для получения всех комментариев поста.

GETCOUNTFAVFORPOST - функция для получения количества лайков поста.

GETCOUNTFAVFORUSER - функция для получения всех избранных постав пользователя.

GETCOUNTLIKE - функция для получения количества лайков комментария.

GETCOUNTSUBFORBLOG - функция для получения количества подписок блога.

GETCOUNTSUBFORUSER - функция для получения количества подписок пользователя.

GETPOSTBYID - функция для получения поста по идентификатору.

GETPOSTFORBLOG - функция для получения постов блога.

GETUSERBEEMAIL - функция для получения пользователя по почте.

GETUSERBYID - функция для получения пользователя по идентификатору.

GETUSERBYLOGIN - функция для получения пользователя по логину.

HASH\_PASSWORD - функция для получения зашифрованного пароля.

## Тригеры

Всего была разработано 5 триггера для проверки на вставку пустых строк и для обновления времени при изменении комментария или поста. Реализация триггеров представлена в приложение Г.

## Представления

Было создано три представления, показанные в листинге 3.3. Каждое из них представляет собой поименованный запрос к базе данных, которые будут часто выполнятся. А именно подписки пользователя, список избранного и популярные посты.

|  |
| --- |
| CREATE VIEW user\_subscriptions AS  SELECT  u.USER\_ID, u.LOGIN, b.BLOG\_ID, b."NAME"  FROM  USER\_TABLE u  JOIN  SUBSCRIPTION s ON u.USER\_ID = s.SUBSCRIBER\_ID  JOIN  BLOG b ON s.BLOG\_ID = b.BLOG\_ID;  CREATE VIEW user\_favorite\_posts AS  SELECT u.USER\_ID, u.LOGIN, p.POST\_ID, p."NAME",  p.CONTAINING,  p.BLOG\_ID, p."TIMESTAMP"  FROM USER\_TABLE u  JOIN FAVORITE f ON u.USER\_ID = f.USER\_ID  JOIN POST p ON f.POST\_ID = p.POST\_ID;  CREATE VIEW popular\_posts AS  SELECT  p.POST\_ID, p.BLOG\_ID, p."NAME", p."TIMESTAMP", p.COUNTVIEWS,      COUNT(DISTINCT f.FAVORITE\_ID) AS like\_count,      COUNT(DISTINCT c.COMMENT\_ID) AS comment\_count  FROM  POST p  LEFT JOIN FAVORITE f ON p.POST\_ID = f.POST\_ID  LEFT JOIN COMMENT\_TABLE c ON p.POST\_ID = c.POST\_ID  GROUP BY p.POST\_ID, p.BLOG\_ID, p."NAME", p."TIMESTAMP", p.COUNTVIEWS  ORDER BY like\_count DESC, comment\_count DESC  FETCH FIRST 20 ROWS ONLY; |

Листинг 3.3 – Представления базы данных

Одним из преимуществ использования представлений – они позволяют создавать запросы с использованием более простых и понятных структур, скрывая сложные детали и объединения таблиц.

## Роли

Создание ролей — это важный аспект безопасности баз данных. Роли используются для управления доступом пользователей к объектам базы данных, таким как таблицы, представления, процедуры и т.д. В Oracle 19c для создания ролей используется команда CREATE ROLE.

Всего было реализовано ролей 4 ролей:

* ADMINR – для админов,
* MODEROTORR – для модераторов,
* USERR – для пользователей,
* GUESTR – для гостей,

Каждой роли были назначены определенные привилегии, которые позволяют пользователям выполнять определенные операции в базе данных. Например, роль GUESR имеет доступ только к операциям, связанным получением постов, блогов и другого для просмотра их содержания. Роль ADMINR имеет права на создание, удаление и обновления всех объектов базы данных.

Таким образом, создание ролей является важным шагом в обеспечении безопасности баз данных и помогает управлять доступом пользователей к объектам базы данных. Важно тщательно продумать и назначить правильные привилегии для каждой роли, чтобы предотвратить несанкционированный доступ к данным и защитить конфиденциальность данных.

## Пользователи

Создание пользователей является важным шагом в обеспечении безопасности и контроля доступа к базе данных. В данной базе данных созданы четыре пользователей, каждый из которых имеет свои уникальные права доступа.

Первый пользователь GUESTUSER создан для не авторизованных посетителей сервиса. Им будет предоставлен доступ к просмотру блогов, постов и комментариев, а также авторизацию.

Второй пользователь USERUSER создан для авторизованных пользователей сервиса. Он предоставляет полные возможности сервиса.

Третий пользователь MADERATORUSER создан для модераторов. Он имеет права на изменение блогов, постов и т.д.

Четвертый пользователь ADMINUSER создан для администратора базы данных. Он имеет полный доступ ко всей базе данных.

## Последовательности

Для автоматического инкрементирования числовых идентификаторов в таблицах можно использовать последовательности. Для этого мы создадим последовательности для каждой из наших таблиц.

# Описание процедур импорта и экспорта

При всей своей отлаженности и очевидности, классический способ хранения и представления объектов развитой структуры имеет и вполне определенные недостатки и может вызывать проблемы, с которыми сталкивался любой разработчик, пытавшийся реализовать таким способом достаточно сложную систему. В некоторых ситуациях, решить эти проблемы позволяет хранение объекта в виде JSON.

## Процедура импорта данных из JSON-файла

Для реализации импорта данных из json, была разработана функция, результатом которой стало возращение заполнение таблицы. Скрипт процедуры представлен в листинге 4.1.

CREATE OR REPLACE FUNCTION import\_json\_roles (

  p\_json\_data CLOB

) RETURN VARCHAR2

IS

  l\_message VARCHAR2(4000);

BEGIN

  FOR rec IN (

    SELECT \*

    FROM JSON\_TABLE(

      p\_json\_data,

      '$[\*]'

      COLUMNS (

        name NVARCHAR2(50) PATH '$.Name'

      )

    )

  ) LOOP

    INSERT INTO Role\_Table (Name) VALUES (rec.name);

  END LOOP;

  COMMIT;

  l\_message := 'Данные JSON успешно импортированы.';

  DBMS\_OUTPUT.PUT\_LINE(l\_message);

  RETURN l\_message;

EXCEPTION

  WHEN OTHERS THEN

    l\_message := 'Ошибка при импорте JSON: ' || SQLERRM;

    DBMS\_OUTPUT.PUT\_LINE(l\_message);

    RETURN l\_message;

END;

Листинг 4.1 – функция import\_json\_roles

## Процедура экспорта данных в формате JSON

Для реализации экспорта данных в json, была разработана функция, результатом которой стало возращение json строки, с записанными строками из определенной таблицы в json формат. Скрипт процедуры представлен в листинге 4.2.

|  |
| --- |
| CREATE OR REPLACE FUNCTION export\_json\_roles RETURN CLOB  IS    l\_json\_data CLOB;  BEGIN    SELECT JSON\_ARRAYAGG(      JSON\_OBJECT(        'roleId' VALUE role\_id,        'name'   VALUE Name      )    )    INTO l\_json\_data    FROM Role\_Table;    DBMS\_OUTPUT.PUT\_LINE('Данные экспортированы в JSON.');    RETURN l\_json\_data;  EXCEPTION    WHEN OTHERS THEN      DBMS\_OUTPUT.PUT\_LINE('Ошибка при экспорте JSON: ' || SQLERRM);      RETURN NULL;  END; |

Листинг 4.2 – функция export\_json\_roles

Таким образом, пользователи базы данных могут легко экспортировать и импортировать данные в форматах JSON, что делает управление базой данных более удобным и эффективным.

# Тестирование производительности

Для тестирования производительности базы данных, нам нужно заполнить таблицы достаточным количеством записей. Для данного случая мы заполнили таблицу Comment на 100.000 строк, скрипт заполнения таблиц приложен в приложение Д.

Далее, для улучшения производительности запросов, мы создали несколько индексов. Для таблицы POST мы создали индекс на столбец BLOG\_ID. Для таблицы Comment мы создали индекс на столбе POST\_ID. Пример для создания индексов показан в листинге 5.1.

Теперь давайте рассмотрим производительность базы данных в контексте запросов на выборку данных.

|  |
| --- |
| CREATE INDEX idx\_comment\_postid ON Comment\_Table(Post\_Id); |

Листинг 5.1 – Индекс для таблицы

При выполнении запроса, показанного далее в листинге 5.1 на выборку данных из нескольких таблиц без использования индексов, время ожидания может значительно увеличиваться из-за необходимости выполнения большого количества операций сканирования таблиц. Так, при выполнении запроса на выборку данных из таблицы Comment без использования индексов, время выполнения, представленное на рисунке 5.1, составило около 2415 миллисекунд.



Рисунок 5.1 – Оценка запроса без использования индексов

Создание индексов на столбцах POST\_ID позволило сократить время выполнения запроса на выборку данных из таблицы Comment до 1499 миллисекунд, как видно на рисунке 5.2.



Рисунок 5.2 – Оценка запроса с использованием индексов

Помимо них реализованна поэтапная загрузка в функциях, которая еще больше снижает затрачивамое время, как видно на рисунке 5.3. Листинг функции представлен на листинге 5.2.



Рисунок 5.3 – Поэтапная загрузка данных

CREATE OR REPLACE FUNCTION GetCommentsForPost(

    p\_post\_id IN NUMBER,

    p\_start\_row IN NUMBER,

    p\_count\_row IN NUMBER

) RETURN SYS\_REFCURSOR AS

    l\_cursor SYS\_REFCURSOR;

BEGIN

    OPEN l\_cursor FOR

        SELECT \* FROM (

            SELECT

                com.\*,

                GETCOUNTLIKE(com.COMMENT\_ID) AS like\_count,

                ROW\_NUMBER() OVER (ORDER BY GETCOUNTLIKE(com.COMMENT\_ID) DESC) as rn

            FROM COMMENT\_TABLE com

            WHERE com.POST\_ID = p\_post\_id

        )

        WHERE rn BETWEEN p\_start\_row AND (p\_start\_row + p\_count\_row);

    return l\_cursor;

END;

Листинг 5.2 – функция с поэтапной загрузкой

Таким образом, использование индексов является важным приемом оптимизации производительности базы данных и может сократить время выполнения запросов на выборку данных из таблиц.

# Описание технологии и ее применения в базе данных

В эпоху информационной перегрузки эффективный поиск информации становится критически важным. Традиционные реляционные базы данных, основанные на точном соответствии, не всегда справляются с задачей поиска по текстовым данным. Oracle Text – это мощный компонент СУБД Oracle, предоставляющий функционал полнотекстового поиска, позволяющий находить информацию на основе смысла, а не только точного совпадения слов.

Oracle Text предоставляет широкий спектр возможностей для реализации полнотекстового поиска: индексирование текстовых данных, лингвистический анализ, поддержка различных языков, ранжирование результатов и многое другое.

Oracle Text состоит из следующих основных компонентов:

* Лексический анализатор: разбивает текст на слова с учетом морфологии.
* Список слов: содержит информацию о словах, их синонимах и других лингвистических характеристиках.
* Индекс: специализированный индекс, хранящий информацию о словах и их позициях в документах.

Создание полнотекстового индекса показан в листинге 6.1.

begin

    ctx\_ddl.create\_preference('my\_wordlist', 'BASIC\_WORDLIST');

    ctx\_ddl.create\_preference('my\_lexer', 'AUTO\_LEXER');

    ctx\_ddl.set\_attribute('my\_lexer', 'INDEX\_STEMS','YES');

end;

create index post\_text\_index ON Post(Containing) indextype is ctxsys.context parameters ('LEXER my\_lexer WORDLIST my\_wordlist');

Листинг 6.1 – Создание полнотекстового индекса

Для реализации поиска, была разработана функция, результатом которой стало возращение курсора с найденными строками, отсортированные по релевантности. Скрипт функции представлен в листинге 6.2.

CREATE OR REPLACE FUNCTION SEARCH(p\_text IN NVARCHAR2) RETURN SYS\_REFCURSOR AS

  l\_cursor SYS\_REFCURSOR;

BEGIN

  OPEN l\_cursor FOR

    SELECT p.\*,

           SCORE(1) AS score\_containing

    FROM Post p

    WHERE CONTAINS(p.CONTAINING,'$' || p\_text, 1) > 0

    ORDER BY score\_containing DESC;

  RETURN l\_cursor;

EXCEPTION

  WHEN OTHERS THEN

    DBMS\_OUTPUT.PUT\_LINE('Ошибка: ' || SQLERRM);

    RETURN NULL;

END;

Листинг 6.2 – функция поиска

Таким образом, применение полнотекстового поиска на основе Oracle Text значительно расширяет функциональные возможности баз данных, улучшает доступность и качество информации для конечных пользователей.

# Руководство пользователя

Это руководство пользователя предназначено для того, чтобы помочь вам эффективно использовать базу данных Oracle, разработанную в рамках данной курсовой работы. В нем подробно описаны шаги по установке, настройке и эксплуатации системы, а также приведены основные команды и примеры для выполнения основных операций. Для подключения к базе данных могут быть использованы пользователи, описанные ниже.

При входе на сервис, мы будем иметь статус гостя. Из функционала ему доступно авторизация или регистрация для получения статуса пользования, а также просмотр блогов, постов и комментариев.

После регистрации и последующей авторизации, пользователь уже имеет одноименный статус – Пользователь. У него открываются новые возможности, он может создавать блоги, посты, комментарии. Изменять созданные им объекты и их удалять. Так же появляется возможности добавлять посты в список избранного, подписываться на блоги и ставить лайки комментариям.

Если вход на площадку был произведен с аккаунта модератора, то мы сможем просматривать и редактировать все блоги, посты и комментарии пользователей, однако сами создать что-то мы не можем.

Вход с учетной записью администратора дает нам полное управление базой данных аукционной площадки.

Для управления информацией в базе данных предоставлен набор функций и процедур, описанных в главе 3.

# Заключение

В проекте была успешно реализована база данных блого-социальной сети, которая включает управление информацией о сотрудниках, меню, заказах, столиках и анализ выполненных заказов. Были применены разные возможности базы данных от простых операций вставки и удаления до написания сложных триггеров и процедур. Важным аспектом проекта было написание полнотекстового поиска данных. Была построена иерархия процедур доступа к данным, которая обеспечивает различный уровень доступа для разных пользователей, таких как администраторы, модераторы, пользователи и гости. Это позволяет каждому пользователю получить доступ только к тем данным, которые ему необходимы. Было проведено тестирование производительности базы данных на большом объеме данных, что позволило оптимизировать ее структуру и проанализировать планы запросов к таблицам. Применение технологий базы данных в данной работе позволило значительно упростить и ускорить процессы управления данными сервиса.

# Список используемых источников

1. Полнотекстовый поиск при помощи Oracle Text [Электронный ресурс]. – Режим доступа: oraclestart.blogspot.com/2013/08/oracle-text.html?m=1 – Дата доступа: 30.04.2024.
2. Text Reference [Электронный ресурс]. – Режим доступа: https://docs.oracle.com/cd/B28359\_01/text.111/b28304/toc.htm – Дата доступа: 26.04.2024.
3. Работа с файлами в Oracle [Электронный ресурс]. – Режим доступа: https://docs.oracle.com/cd/F49540\_01/DOC/server.815/a68001/utl\_file.htm. – Дата доступа: 1.05.2024.
4. Работа с данными формата JSON в Oracle [Электронный ресурс]. – Режим доступа: https://docs.oracle.com/en/database/oracle/oracle-database/19/arpls/DBMS\_JSONGEN.html. – Дата доступа: 2.05.2024.
5. Oracle Developer Tools for VS Code [Электронный ресурс]. – Режим доступа: Shttps://medium.com/oracledevs/oracle-developer-tools-for-vs-code-f26d31535925hay – Дата доступа: 10.02.2024.

# Приложение А Листинг создания таблиц

|  |
| --- |
| CREATE TABLE Role\_Table (    role\_id NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY PRIMARY KEY,    Name NVARCHAR2(50) UNIQUE NOT NULL  );  INSERT INTO Role\_table (Name) VALUES ('Администратор');  INSERT INTO Role\_table (Name) VALUES ('Модератор');  INSERT INTO Role\_table (Name) VALUES ('Пользователь');  INSERT INTO Role\_table (Name) VALUES ('Гость');  CREATE TABLE User\_table (    User\_id NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY PRIMARY KEY,    Login NVARCHAR2(50) UNIQUE NOT NULL,    Password NVARCHAR2(255) NOT NULL,    Email NVARCHAR2(100) UNIQUE,    Name NVARCHAR2(50) NOT NULL,    SecondName NVARCHAR2(50),    Age NUMBER(3),    Role\_Id NUMBER REFERENCES Role\_Table(Role\_Id)  );  CREATE TABLE Blog (    Blog\_Id NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY PRIMARY KEY,    User\_Id NUMBER REFERENCES User\_Table(User\_Id) ON DELETE CASCADE,    Name NVARCHAR2(100) NOT NULL,    Discription NVARCHAR2(255),    Timestamp DATE DEFAULT SYSDATE  );  CREATE TABLE Post (    Post\_Id NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY PRIMARY KEY,    Blog\_Id NUMBER REFERENCES Blog(Blog\_Id) ON DELETE CASCADE,    Name NVARCHAR2(100) NOT NULL,    Containing CLOB,    Timestamp DATE DEFAULT SYSDATE,    CountViews NUMBER DEFAULT 0  );  CREATE TABLE Comment\_Table (    Comment\_Id NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY PRIMARY KEY,    User\_Id NUMBER REFERENCES User\_Table(User\_Id) ON DELETE CASCADE,    Post\_Id NUMBER REFERENCES Post(Post\_Id) ON DELETE CASCADE,    Containing CLOB,    Timestamp DATE DEFAULT SYSDATE  );  CREATE TABLE LIKE\_TABLE (    LIKE\_ID NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY PRIMARY KEY,    User\_Id NUMBER REFERENCES User\_Table(User\_Id) ON DELETE CASCADE,    Comment\_Id NUMBER REFERENCES Comment\_Table(Comment\_Id) ON DELETE CASCADE,    Timestamp DATE DEFAULT SYSDATE  )  CREATE TABLE Favorite (    Favorite\_Id NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY PRIMARY KEY,    User\_Id NUMBER REFERENCES User\_Table(User\_Id),    Post\_Id NUMBER REFERENCES Post(Post\_Id),    Timestamp DATE DEFAULT SYSDATE  );  CREATE TABLE Subscription (    Subscription\_Id NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY PRIMARY KEY,    Subscriber\_Id NUMBER REFERENCES User\_Table(User\_Id) ON DELETE CASCADE,    Blog\_Id NUMBER REFERENCES Blog(Blog\_Id) ON DELETE CASCADE,    Timestamp Date DEFAULT SYSDATE  ); |

# Приложение Б Листинг создания процедур

|  |
| --- |
| CREATE OR REPLACE PROCEDURE AddBlog (    p\_user\_id IN NUMBER,    p\_name IN VARCHAR2,    p\_description IN VARCHAR2,    p\_blog\_id OUT NUMBER  ) AS  BEGIN    INSERT INTO Blog (blog\_id, user\_id, name, Discription)    VALUES (blog\_seq.NEXTVAL, p\_user\_id, p\_name, p\_description)    RETURNING blog\_id INTO p\_blog\_id;    DBMS\_OUTPUT.PUT\_LINE('Блог добавлен. ID блога: ' || p\_blog\_id);  EXCEPTION    WHEN OTHERS THEN      DBMS\_OUTPUT.PUT\_LINE('Ошибка: ' || SQLERRM);      p\_blog\_id := -1;  END;  CREATE OR REPLACE PROCEDURE AddComment (    p\_post\_id IN NUMBER,    p\_user\_id IN NUMBER,    p\_containing IN VARCHAR2,    p\_comment\_id OUT NUMBER  ) AS  BEGIN    INSERT INTO Comment\_Table (comment\_id, post\_id, user\_id, containing)    VALUES (comment\_seq.NEXTVAL, p\_post\_id, p\_user\_id, p\_containing)    RETURNING comment\_id INTO p\_comment\_id;    DBMS\_OUTPUT.PUT\_LINE('Коментарий добавлен');    EXCEPTION      WHEN OTHERS THEN      DBMS\_OUTPUT.PUT\_LINE('Ошибка: ' || SQLERRM);      p\_comment\_id:=-1;  END;  CREATE OR REPLACE PROCEDURE AddPost (    p\_blog\_id IN NUMBER,    p\_name IN VARCHAR2,    p\_containing IN CLOB,    p\_post\_id OUT NUMBER  ) AS  BEGIN    INSERT INTO Post (post\_id, blog\_id, name, containing)    VALUES (post\_seq.NEXTVAL, p\_blog\_id, p\_name, p\_containing)    RETURNING post\_id INTO p\_post\_id;    DBMS\_OUTPUT.PUT\_LINE('Пост добавлен');      EXCEPTION      WHEN OTHERS THEN      DBMS\_OUTPUT.PUT\_LINE('Ошибка: ' || SQLERRM);      p\_post\_id:=-1;  END;  CREATE OR REPLACE PROCEDURE AddUser (    login IN VARCHAR2,    password IN VARCHAR2,    email IN VARCHAR2,    name IN VARCHAR2,    secondname IN VARCHAR2,    age IN NUMBER,    RoleId IN NUMBER,    UserId OUT number  )  AS  BEGIN    INSERT INTO User\_Table(login, password, email, name, secondname, age, role\_id)     VALUES(AddUser.login,  hash\_password(AddUser.password), AddUser.email, AddUser.name, AddUser.secondname, AddUser.age, AddUser.RoleId);    DBMS\_OUTPUT.PUT\_LINE('Пользователь добавлен');    COMMIT;    select user\_id into UserId from user\_table where login = AddUser.login;  EXCEPTION      WHEN DUP\_VAL\_ON\_INDEX THEN BEGIN        IF SQLERRM LIKE '%unique\_login%' THEN          DBMS\_OUTPUT.PUT\_LINE('Ошибка: Пользователь с таким логином уже существует.');          UserId:=-1;        ELSIF SQLERRM LIKE '%unique\_email%' THEN          DBMS\_OUTPUT.PUT\_LINE('Ошибка: Пользователь с таким email уже существует.');          UserId:= -2;        ELSE          RAISE;        END IF;      END;    WHEN OTHERS THEN BEGIN      DBMS\_OUTPUT.PUT\_LINE('Ошибка: ' || SQLERRM);      UserId:= -3;      END;  END AddUser;  CREATE OR REPLACE PROCEDURE ADDVIEWPOST(    p\_post\_id IN NUMBER,    l\_updated\_count OUT NUMBER  )   AS   BEGIN    UPDATE POST    SET CountViews = CountViews + 1 WHERE post\_id = p\_post\_id;      l\_updated\_count := SQL%ROWCOUNT;    END ADDVIEWPOST;  CREATE OR REPLACE PROCEDURE DelBlog (    p\_blog\_id IN NUMBER,  l\_deleted\_count OUT NUMBER  ) AS  BEGIN    DELETE FROM Blog    WHERE blog\_id = p\_blog\_id;      l\_deleted\_count := SQL%ROWCOUNT;        DBMS\_OUTPUT.PUT\_LINE('Блог удален');    EXCEPTION      WHEN NO\_DATA\_FOUND THEN      DBMS\_OUTPUT.PUT\_LINE('Блог не найден');      l\_deleted\_count:=-1;      WHEN OTHERS THEN      DBMS\_OUTPUT.PUT\_LINE('Ошибка: ' || SQLERRM);      l\_deleted\_count:=-1;  END;  CREATE OR REPLACE PROCEDURE DelComment (    p\_comment\_id IN NUMBER,    l\_deleted\_count OUT NUMBER  ) AS  BEGIN    DELETE FROM Comment\_Table    WHERE comment\_id = p\_comment\_id;    l\_deleted\_count := SQL%ROWCOUNT;    DBMS\_OUTPUT.PUT\_LINE('Коментарий удален');    EXCEPTION      WHEN NO\_DATA\_FOUND THEN      DBMS\_OUTPUT.PUT\_LINE('Коментарий не найден');      l\_deleted\_count:=-1;      WHEN OTHERS THEN      DBMS\_OUTPUT.PUT\_LINE('Ошибка: ' || SQLERRM);      l\_deleted\_count:=-1;  END;  CREATE OR REPLACE PROCEDURE DelPost (    p\_post\_id IN NUMBER,    l\_deleted\_count OUT NUMBER  ) AS  BEGIN    DELETE FROM Post    WHERE post\_id = p\_post\_id;    l\_deleted\_count := SQL%ROWCOUNT;      DBMS\_OUTPUT.PUT\_LINE('Пост удален');      EXCEPTION      WHEN NO\_DATA\_FOUND THEN      DBMS\_OUTPUT.PUT\_LINE('пост не найден');      l\_deleted\_count:=-1;      WHEN OTHERS THEN      DBMS\_OUTPUT.PUT\_LINE('Ошибка: ' || SQLERRM);      l\_deleted\_count:=-1;  END;  CREATE OR REPLACE PROCEDURE DelUserById (    p\_user\_id IN NUMBER,    l\_deleted\_count OUT NUMBER  ) AS  BEGIN    DELETE FROM User\_Table    WHERE user\_id = p\_user\_id;      l\_deleted\_count := SQL%ROWCOUNT;      IF l\_deleted\_count > 0 THEN      DBMS\_OUTPUT.PUT\_LINE('Пользователь с ID: ' || p\_user\_id || ' удален.');    ELSE      DBMS\_OUTPUT.PUT\_LINE('Пользователь с ID: ' || p\_user\_id || ' не найден.');    END IF;    EXCEPTION    WHEN OTHERS THEN      DBMS\_OUTPUT.PUT\_LINE('Ошибка: ' || SQLERRM);      l\_deleted\_count:= -1;  END DelUserById;  CREATE OR REPLACE PROCEDURE send\_message(    user\_id IN NUMBER,    v\_message IN NVARCHAR2  ) AS  BEGIN    DBMS\_OUTPUT.PUT\_LINE(v\_message);  END;  CREATE OR REPLACE PROCEDURE UpdBlog (    p\_blog\_id IN NUMBER,    p\_name IN VARCHAR2 DEFAULT NULL,    p\_description IN VARCHAR2 DEFAULT NULL,    p\_updated\_count OUT NUMBER  ) AS  BEGIN    UPDATE Blog    SET name = NVL(p\_name, name),        discription = NVL(p\_description, discription)    WHERE blog\_id = p\_blog\_id;    p\_updated\_count := SQL%ROWCOUNT;    DBMS\_OUTPUT.PUT\_LINE('Блог обновлен');  EXCEPTION    WHEN OTHERS THEN      DBMS\_OUTPUT.PUT\_LINE('Ошибка: ' || SQLERRM);      p\_updated\_count := -1;  END;  CREATE OR REPLACE PROCEDURE UpdComment (    p\_comment\_id IN NUMBER,    p\_containing IN VARCHAR2 DEFAULT NULL,    l\_updated\_count OUT NUMBER  ) AS  BEGIN    UPDATE Comment\_Table    SET containing = NVL(p\_containing, containing)    WHERE comment\_id = p\_comment\_id;      l\_updated\_count := SQL%ROWCOUNT;    DBMS\_OUTPUT.PUT\_LINE('Коментарий обновлен');    EXCEPTION      WHEN OTHERS THEN      DBMS\_OUTPUT.PUT\_LINE('Ошибка: ' || SQLERRM);      l\_updated\_count:=-1;  END;  CREATE OR REPLACE PROCEDURE UpdFavorite (    UserId IN NUMBER,    PostId IN NUMBER,    res OUT NUMBER  )  AS  BEGIN    INSERT INTO Favorite(user\_id, post\_id) VALUES(UserId, PostId);    DBMS\_OUTPUT.PUT\_LINE('Пост добавлен в избранное.');    COMMIT;    res:=1;  EXCEPTION    WHEN DUP\_VAL\_ON\_INDEX THEN      DELETE FROM Favorite      WHERE user\_id = UserId        AND post\_id = PostId;      COMMIT;      DBMS\_OUTPUT.PUT\_LINE('Пост удален из избранного.');      res:=0;    WHEN OTHERS THEN      DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка' || SQLERRM);      res:=-1;  END UpdFavorite;  CREATE OR REPLACE PROCEDURE UpdLike (    p\_user\_id IN NUMBER,    p\_comment\_id IN NUMBER,    res OUT NUMBER  )  AS  BEGIN    INSERT INTO Like\_Table(user\_id, comment\_id) VALUES(p\_user\_id, p\_comment\_id);    DBMS\_OUTPUT.PUT\_LINE('Лайк добавлен');    COMMIT;    res:=1;  EXCEPTION    WHEN DUP\_VAL\_ON\_INDEX THEN      DELETE FROM Like\_Table      WHERE user\_id = p\_user\_id        AND comment\_id = p\_comment\_id;      COMMIT;      DBMS\_OUTPUT.PUT\_LINE('Лайк удален');      res:=0;    WHEN OTHERS THEN      DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка' || SQLERRM);      res:=-1;  END UpdLike;  CREATE OR REPLACE PROCEDURE UpdPost (    p\_post\_id IN NUMBER,    p\_name IN VARCHAR2 DEFAULT NULL,    p\_containing IN CLOB DEFAULT NULL,    l\_updated\_count OUT NUMBER  ) AS  BEGIN    UPDATE Post    SET name = NVL(p\_name, name),        containing = NVL(p\_containing, containing)    WHERE post\_id = p\_post\_id;    l\_updated\_count := SQL%ROWCOUNT;      DBMS\_OUTPUT.PUT\_LINE('Пост обновлен');    EXCEPTION      WHEN OTHERS THEN      DBMS\_OUTPUT.PUT\_LINE('Ошибка: ' || SQLERRM);      l\_updated\_count:=-1;  END;  CREATE OR REPLACE PROCEDURE UpdSubscription (    SubscriberId IN NUMBER,    BlogId IN NUMBER,    res OUT NUMBER  )  AS  BEGIN    INSERT INTO SUBSCRIPTION(subscriber\_id, blog\_id) VALUES(SubscriberId, BlogId);    DBMS\_OUTPUT.PUT\_LINE('Пользователь подписался.');    COMMIT;    res:= 1;  EXCEPTION    WHEN DUP\_VAL\_ON\_INDEX THEN BEGIN      DELETE FROM SUBSCRIPTION      WHERE subscriber\_id = SubscriberId        AND blog\_id = BlogId;      COMMIT;    DBMS\_OUTPUT.PUT\_LINE('Пользователь отписался.');    res:= 0;    END;    When OTHERS Then BEGIN    DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка: ' || SQLERRM);    res:=-1;    END;  END UpdSubscription;  CREATE OR REPLACE PROCEDURE UpdUserById (    p\_user\_id IN NUMBER,    p\_password IN VARCHAR2 DEFAULT NULL,    p\_email IN VARCHAR2 DEFAULT NULL,    p\_name IN VARCHAR2 DEFAULT NULL,    p\_secondname IN VARCHAR2 DEFAULT NULL,    p\_age IN NUMBER DEFAULT NULL,    p\_role\_id IN NUMBER DEFAULT NULL,    l\_updated\_count OUT NUMBER  ) AS    p\_hash\_password NVARCHAR2(255);  BEGIN    IF p\_password IS NOT NULL THEN       p\_hash\_password := hash\_password(p\_password);    ELSE p\_hash\_password := NULL;    END IF;    UPDATE User\_Table    SET password = NVL(p\_hash\_password, password),        email = NVL(p\_email, email),        name = NVL(p\_name, name),        secondname = NVL(p\_secondname, secondname),        age = NVL(p\_age, age),        role\_id = NVL(p\_role\_id, role\_id)    WHERE user\_id = p\_user\_id;      l\_updated\_count := SQL%ROWCOUNT;      IF l\_updated\_count > 0 THEN      DBMS\_OUTPUT.PUT\_LINE('Пользователь с ID: ' || p\_user\_id || ' обновлен.');    ELSE      DBMS\_OUTPUT.PUT\_LINE('Пользователь с ID: ' || p\_user\_id || ' не найден.');    END IF;  EXCEPTION      WHEN DUP\_VAL\_ON\_INDEX THEN        DBMS\_OUTPUT.PUT\_LINE('Ошибка: Пользователь с таким email уже существует.');        l\_updated\_count := -1;      WHEN OTHERS THEN      DBMS\_OUTPUT.PUT\_LINE('Ошибка: ' || SQLERRM);      l\_updated\_count := -1;  END UpdUserById; |

# Приложение В Листинг создания функций

|  |
| --- |
| CREATE OR REPLACE FUNCTION GetAllBlogs RETURN SYS\_REFCURSOR AS    l\_cursor SYS\_REFCURSOR;  BEGIN    OPEN l\_cursor FOR      SELECT \*      FROM Blog;      RETURN l\_cursor;  END;  CREATE OR REPLACE FUNCTION SEARCH(p\_text IN NVARCHAR2) RETURN SYS\_REFCURSOR AS    l\_cursor SYS\_REFCURSOR;  BEGIN    OPEN l\_cursor FOR      SELECT p.\*,             SCORE(1) AS score\_containing      FROM Post p      WHERE CONTAINS(p.CONTAINING,'$' || p\_text, 1) > 0      ORDER BY score\_containing DESC;    RETURN l\_cursor;  EXCEPTION    WHEN OTHERS THEN      DBMS\_OUTPUT.PUT\_LINE('Ошибка: ' || SQLERRM);      RETURN NULL;  END;  CREATE OR REPLACE FUNCTION hash\_password (      p\_password IN NVARCHAR2  ) RETURN NVARCHAR2  AS      p\_hash\_raw RAW(32);      p\_hash\_hex NVARCHAR2(255);  BEGIN      p\_hash\_raw := DBMS\_CRYPTO.HASH(UTL\_I18N.STRING\_TO\_RAW(p\_password, 'AL32UTF8'), DBMS\_CRYPTO.HASH\_SH256);      p\_hash\_hex := UTL\_ENCODE.base64\_encode(p\_hash\_raw);      RETURN p\_hash\_hex;  END;  CREATE OR REPLACE FUNCTION GetUserByLogin (    p\_login IN VARCHAR2  ) RETURN User\_Table%ROWTYPE AS    l\_user User\_Table%ROWTYPE;  BEGIN    SELECT \* INTO l\_user    FROM User\_Table    WHERE login = p\_login;      RETURN l\_user;  EXCEPTION    WHEN NO\_DATA\_FOUND THEN      DBMS\_OUTPUT.PUT\_LINE('Пользователь с логином: ' || p\_login || ' не найден.');      RETURN NULL;    WHEN OTHERS THEN      DBMS\_OUTPUT.PUT\_LINE('Ошибка: ' || SQLERRM);      RETURN NULL;  END GetUserByLogin;  CREATE OR REPLACE FUNCTION GetUserById (    p\_user\_id IN NUMBER  ) RETURN User\_Table%ROWTYPE AS    l\_user User\_Table%ROWTYPE;  BEGIN    SELECT \* INTO l\_user    FROM User\_Table    WHERE user\_id = p\_user\_id;      RETURN l\_user;  EXCEPTION    WHEN NO\_DATA\_FOUND THEN      DBMS\_OUTPUT.PUT\_LINE('Пользователь с ID: ' || p\_user\_id || ' не найден.');      RETURN NULL;    WHEN OTHERS THEN      DBMS\_OUTPUT.PUT\_LINE('Ошибка: ' || SQLERRM);      RETURN NULL;  END GetUserById;  CREATE OR REPLACE FUNCTION GetUserByEmail (    p\_email IN VARCHAR2  ) RETURN User\_Table%ROWTYPE AS    l\_user User\_Table%ROWTYPE;  BEGIN    SELECT \* INTO l\_user    FROM User\_Table    WHERE email = p\_email;      RETURN l\_user;  EXCEPTION    WHEN NO\_DATA\_FOUND THEN      DBMS\_OUTPUT.PUT\_LINE('Пользователь с email: ' || p\_email || ' не найден.');      RETURN NULL;    WHEN OTHERS THEN      DBMS\_OUTPUT.PUT\_LINE('Ошибка: ' || SQLERRM);      RETURN NULL;  END GetUserByEmail;  CREATE OR REPLACE FUNCTION GETPOSTFORBLOG(      p\_blog\_id IN NUMBER,      p\_start\_row IN NUMBER,      p\_count\_row IN NUMBER,      p\_sort\_column VARCHAR2 DEFAULT 'COUNTVIEWS',      p\_sort\_direction VARCHAR2 DEFAULT 'DESC'  ) RETURN SYS\_REFCURSOR AS      l\_cursor SYS\_REFCURSOR;  BEGIN      OPEN l\_cursor FOR          SELECT \* FROM (              SELECT                  p.\*,                  ROW\_NUMBER() OVER (ORDER BY                      CASE when p\_sort\_direction = 'ASC' then                         case WHEN p\_sort\_column = 'COUNTVIEWS' THEN p.COUNTVIEWS                              WHEN p\_sort\_column = 'FAVORITE\_COUNT' then favorite\_count                              when p\_sort\_column = 'CREATION\_DATE'THEN (p."TIMESTAMP" - TO\_DATE('1970-01-01', 'YYYY-MM-DD')) \* 86400                          end                        end,                      CASE when p\_sort\_direction = 'DESC' then                         case WHEN p\_sort\_column = 'COUNTVIEWS' THEN p.COUNTVIEWS                              WHEN p\_sort\_column = 'FAVORITE\_COUNT' then favorite\_count                              when p\_sort\_column = 'CREATION\_DATE'THEN (p."TIMESTAMP" - TO\_DATE('1970-01-01', 'YYYY-MM-DD')) \* 86400                          end                        end DESC) as rn              FROM (                  SELECT ps.\*, GETCOUNTFAVFORPOST(ps.POST\_ID) AS favorite\_count                  FROM POST ps                  WHERE BLOG\_ID = p\_blog\_id              ) p          )      WHERE rn BETWEEN p\_start\_row AND (p\_start\_row + p\_count\_row);      return l\_cursor;  END;  CREATE OR REPLACE FUNCTION GetPostById (    p\_post\_id IN NUMBER  ) RETURN Post%ROWTYPE AS    l\_post Post%ROWTYPE;  BEGIN    SELECT \* INTO l\_post    FROM Post    WHERE post\_id = p\_post\_id;      RETURN l\_post;    EXCEPTION    WHEN NO\_DATA\_FOUND THEN    DBMS\_OUTPUT.PUT\_LINE('пост не найден');    RETURN NULL;    WHEN OTHERS THEN    DBMS\_OUTPUT.PUT\_LINE('Ошибка: ' || SQLERRM);    RETURN NULL;  END;  CREATE OR REPLACE FUNCTION GETCOUNTSUBFORUSER(    p\_user\_id IN NUMBER  ) RETURN INTEGER AS l\_count\_sub INTEGER;  BEGIN    SELECT COUNT(\*) INTO l\_count\_sub FROM    SUBSCRIPTION s WHERE s.SUBSCRIBER\_ID = p\_user\_id;    return l\_count\_sub;  END;  CREATE OR REPLACE FUNCTION GETCOUNTSUBFORBLOG(    p\_blog\_id IN NUMBER  ) RETURN INTEGER AS l\_count\_sub INTEGER;  BEGIN    SELECT COUNT(\*) INTO l\_count\_sub FROM    SUBSCRIPTION s WHERE s.BLOG\_ID = p\_blog\_id;    return l\_count\_sub;  END;  CREATE OR REPLACE FUNCTION GETCOUNTLIKE(    p\_com\_id IN NUMBER  ) RETURN INTEGER AS l\_count\_like INTEGER;  BEGIN    SELECT COUNT(\*) INTO l\_count\_like FROM    LIKE\_TABLE l WHERE l.COMMENT\_ID = p\_com\_id;    return l\_count\_like;  END;  CREATE OR REPLACE FUNCTION GETCOUNTFAVFORUSER(    p\_user\_id IN NUMBER  ) RETURN INTEGER AS l\_count\_fav INTEGER;  BEGIN    SELECT COUNT(\*) INTO l\_count\_fav FROM    FAVORITE f WHERE f.USER\_ID = p\_user\_id;    return l\_count\_fav;  END;  CREATE OR REPLACE FUNCTION GETCOUNTFAVFORPOST(    p\_post\_id IN NUMBER  ) RETURN INTEGER AS l\_count\_fav INTEGER;  BEGIN    SELECT COUNT(\*) INTO l\_count\_fav FROM    FAVORITE f WHERE f.POST\_ID = p\_post\_id;    return l\_count\_fav;  END;  CREATE OR REPLACE FUNCTION GetCommentsForPost(      p\_post\_id IN NUMBER,      p\_start\_row IN NUMBER,      p\_count\_row IN NUMBER  ) RETURN SYS\_REFCURSOR AS      l\_cursor SYS\_REFCURSOR;  BEGIN      OPEN l\_cursor FOR          SELECT \* FROM (              SELECT                  com.\*,                  GETCOUNTLIKE(com.COMMENT\_ID) AS like\_count,                  ROW\_NUMBER() OVER (ORDER BY GETCOUNTLIKE(com.COMMENT\_ID) DESC) as rn              FROM COMMENT\_TABLE com              WHERE com.POST\_ID = p\_post\_id          )          WHERE rn BETWEEN p\_start\_row AND (p\_start\_row + p\_count\_row);      return l\_cursor;  END;  CREATE OR REPLACE FUNCTION GetCommentById (    p\_comment\_id IN NUMBER  ) RETURN Comment\_Table%ROWTYPE AS    l\_comment Comment\_Table%ROWTYPE;  BEGIN    SELECT \* INTO l\_comment    FROM Comment\_Table    WHERE comment\_id = p\_comment\_id;      RETURN l\_comment;      EXCEPTION    WHEN NO\_DATA\_FOUND THEN    DBMS\_OUTPUT.PUT\_LINE('Коментарий не найден');    RETURN NULL;    WHEN OTHERS THEN    DBMS\_OUTPUT.PUT\_LINE('Ошибка: ' || SQLERRM);    RETURN NULL;  END;  CREATE OR REPLACE FUNCTION GETBLOGFORUSER(    p\_user\_id IN NUMBER,    p\_start\_row IN NUMBER,    p\_count\_row IN NUMBER) RETURN SYS\_REFCURSOR AS   l\_cursor SYS\_REFCURSOR;   BEGIN    OPEN l\_cursor FOR      SELECT \* FROM (        SELECT b.\*, ROWNUM as rn FROM Blog b        WHERE USER\_ID = p\_user\_id ORDER BY b."TIMESTAMP" ASC      )      WHERE rn BETWEEN p\_start\_row AND (p\_start\_row + p\_count\_row);    return l\_cursor;  end;  CREATE OR REPLACE FUNCTION GetBlogById (    p\_blog\_id IN NUMBER  ) RETURN Blog%ROWTYPE AS    l\_blog Blog%ROWTYPE;  BEGIN    SELECT \* INTO l\_blog    FROM Blog    WHERE blog\_id = p\_blog\_id;      RETURN l\_blog;      EXCEPTION    WHEN NO\_DATA\_FOUND THEN    DBMS\_OUTPUT.PUT\_LINE('Блог не найден');    RETURN NULL;    WHEN OTHERS THEN    DBMS\_OUTPUT.PUT\_LINE('Ошибка: ' || SQLERRM);    RETURN NULL;  END;  CREATE OR REPLACE FUNCTION GetAllPosts RETURN SYS\_REFCURSOR AS    l\_cursor SYS\_REFCURSOR;  BEGIN    OPEN l\_cursor FOR      SELECT \*      FROM Post;      RETURN l\_cursor;  END; |

# Приложение Г Листинг создания триггеров

|  |
| --- |
| CREATE OR REPLACE TRIGGER notify\_subscribers  AFTER INSERT ON post  FOR EACH ROW  DECLARE    v\_blog\_id NUMBER;    v\_message NVARCHAR2(200);  BEGIN    v\_blog\_id := :NEW.blog\_id;    v\_message := 'Новый пост опубликован в блоге, на который вы подписаны!';    FOR sub IN (      SELECT SUBSCRIBER\_ID      FROM subscription      WHERE blog\_id = v\_blog\_id    ) LOOP      send\_message(sub.subscriber\_Id, v\_message);    END LOOP;  END;  CREATE OR REPLACE PROCEDURE send\_message(    user\_id IN NUMBER,    v\_message IN NVARCHAR2  ) AS  BEGIN    DBMS\_OUTPUT.PUT\_LINE(v\_message);  END;  CREATE OR REPLACE TRIGGER notify\_empty\_comment  BEFORE INSERT ON COMMENT\_TABLE  FOR EACH ROW  DECLARE      trimmed\_clob CLOB;  BEGIN      trimmed\_clob := TRIM(:new.Containing);      IF trimmed\_clob IS NULL OR trimmed\_clob = '' THEN          RAISE\_APPLICATION\_ERROR(-20001, 'Поле не может быть пустым.');      END IF;  END;  CREATE OR REPLACE TRIGGER notify\_empty\_post  BEFORE INSERT ON POST  FOR EACH ROW  DECLARE      trimmed\_clob CLOB;  BEGIN      trimmed\_clob := TRIM(:new.Containing);      IF trimmed\_clob IS NULL OR trimmed\_clob = '' THEN          RAISE\_APPLICATION\_ERROR(-20001, 'Поле не может быть пустым.');      END IF;  END;  CREATE OR REPLACE TRIGGER notify\_update\_time\_post  BEFORE UPDATE ON POST  FOR EACH ROW  BEGIN      :new.Timestamp := SYSTIMESTAMP;  END;  CREATE OR REPLACE TRIGGER notify\_update\_time\_comment  BEFORE UPDATE ON COMMENT\_TABLE  FOR EACH ROW  BEGIN      :new.Timestamp := SYSTIMESTAMP;  END; |

# Приложение Д Листинг процедуры для заполнения таблиц

|  |
| --- |
| INSERT INTO USER\_TABLE(login,PASSWORD,EMAIL,name,SECONDNAME,AGE,ROLE\_ID) VALUES(      'Пользователь','12345qwerty','adaw@gmail.com','Marat','petrov',19,3  );  INSERT INTO BLOG(USER\_ID,NAME,DISCRIPTION) VALUES(1,'BLOG1','blog1blog1');  INSERT INTO POST(BLOG\_ID,NAME,CONTAINING) VALUES(1,'post1','Сегодняшний день начался с яркого солнца и прохладного ветра.');  INSERT INTO POST(BLOG\_ID,NAME,CONTAINING) VALUES(1,'post1','В парке зацвели первые весенние цветы, наполнившие воздух ароматом.');  INSERT INTO POST(BLOG\_ID,NAME,CONTAINING) VALUES(1,'post1','Вечером планирую прогуляться по набережной и насладиться закатом.');  INSERT INTO POST(BLOG\_ID,NAME,CONTAINING) VALUES(1,'post1','На кухне раскатывается аромат свежей выпечки, готовится ужин для семьи.');  INSERT INTO POST(BLOG\_ID,NAME,CONTAINING) VALUES(1,'post1','После дождя в саду открылись новые почки на розовых кустах.');  INSERT INTO POST(BLOG\_ID,NAME,CONTAINING) VALUES(1,'post1','The sun is shining brightly, casting long shadows on the ground.');  INSERT INTO POST(BLOG\_ID,NAME,CONTAINING) VALUES(1,'post1','Spring has arrived, and flowers are blooming in every corner of the garden.');  INSERT INTO POST(BLOG\_ID,NAME,CONTAINING) VALUES(1,'post1','Tonight, I`m planning to have dinner with friends at a cozy restaurant downtown.');  INSERT INTO POST(BLOG\_ID,NAME,CONTAINING) VALUES(1,'post1','The sound of laughter fills the air as children play in the park.');  INSERT INTO POST(BLOG\_ID,NAME,CONTAINING) VALUES(1,'post1','I`m looking forward to a relaxing weekend, spent reading books and enjoying nature.');  DECLARE    v\_containing CLOB := 'Пример комментария.';  BEGIN  FOR i in 1..10 LOOP    FOR j IN 1..10000 LOOP      INSERT INTO Comment\_Table (User\_Id, Post\_Id, Containing)      VALUES (1, i, v\_containing || j);    END LOOP;  END LOOP;    COMMIT;    DBMS\_OUTPUT.PUT\_LINE('100000 комментариев добавлено.');  END; |