

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

**Лабораторна робота №** 3

з дисципліни “Бази даних”

# тема “Засоби оптимізації роботи СУБД PostgreSQL”

|  |  |
| --- | --- |
| Виконав  Студент II курсу  групи КП-03  Довженко Роман Олександрович  (*прізвище, ім’я, по батькові*)  варіант №2 |  |

Київ 2021

**Мета роботи**

Метою роботи є здобуття практичних навичок використання засобів

оптимізації СУБД PostgreSQL.

**Постановка завдання**

**Загальне завдання роботи полягає у наступному:**

1. Перетворити модуль “Модель” з шаблону MVC лабораторної роботи No2

у вигляд об’єктно-реляційної проекції (ORM).

2. Створити та проаналізувати різні типи індексів у PostgreSQL.

3. Розробити тригер бази даних PostgreSQL.

**Деталізоване завдання:**

Вимоги до пункту завдання No1

Для перетворення функцій, що реалізують запити до об’єктної бази даних,

необхідно встановити бібліотеку sqlAlchemy, налаштувати програму на роботу

з ORM, розробити класи-сутності для об’єктів-сутностей, представлених

відповідними таблицями БД та пов’язаних зв’язками 1:М, М:М та 1:1 виконати

опис схеми бази даних. Особливу увагу приділити контролю зовнішніх зв’язків

між таблицями засобами ORM.

Замінити виклики запитів мовою SQL на відповідні запити засобами

SQLAlchemy по роботі з об’єктами. Обов’язковим є реалізація вставки,

вилучення та редагування екземплярів класів-сутностей. Розробка запитів на

генерацію даних та пошук екземплярів класів-сутностей вітається, але не є

обов’язковою.

Інтерфейси функцій (вхідні та вихідні аргументи функцій модуля “Модель”)

мають залишитись без змін.

Вимоги до пункту завдання No2

Відповідно до варіанту індексування продемонструвати на прикладах запитів

SQL SELECT підвищення швидкодії їх виконання з використанням індексів, а

також пояснити чому для деяких випадків індексування використовувати

недоцільно. При цьому для наочного представлення слід використати функцію

генерування рандомізованих даних з лабораторної роботи No2, створивши

необхідну кількість тестових даних. Навести 4-5 прикладів запитів SELECT (із виведенням результуючих даних), що містять фільтрацію, агрегатні функції,

групування та сортування (у необхідних комбінаціях).

Вимоги до пункту завдання No3

Створити тригер бази даних PostgreSQL відповідно до варіанта. Тригерна

функція має включати обробку запису, що модифікується (вставляється або

вилучається), умовні оператори, курсорні цикли та обробку виключних

ситуацій. Виконати відлагодження тригера при різних вхідних даних, навівши

2-3 приклади його використання.

**Результат роботи**

**Структура БД**

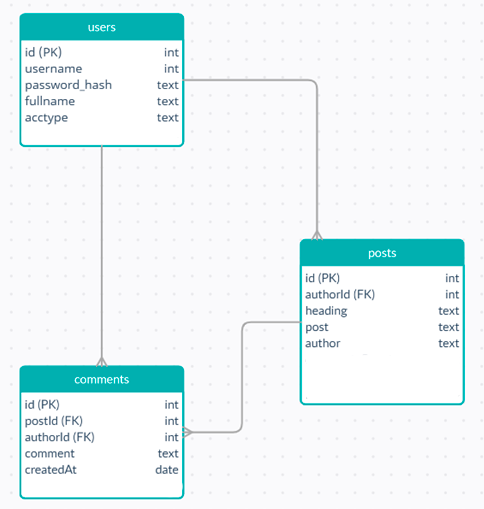
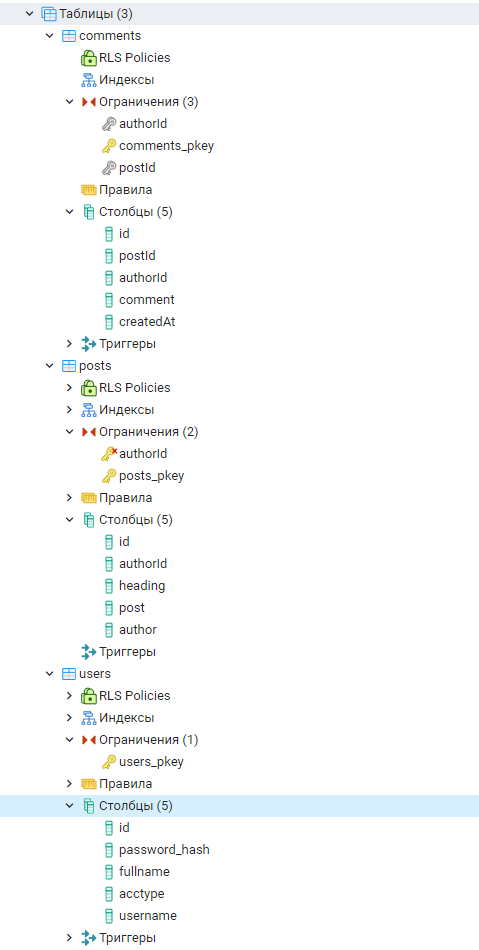
****

Рис. Структура бази даних

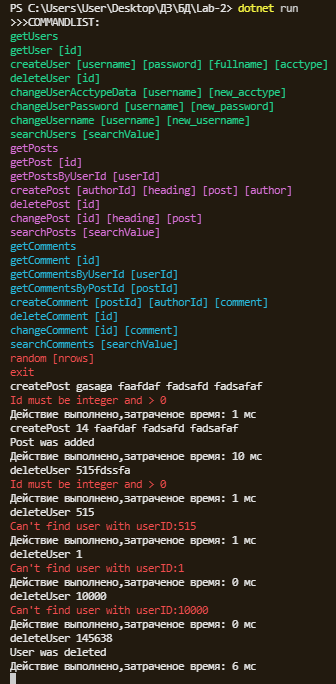
1. Таблиця users
   1. Primary Key - id - integer field
   2. username - text field
   3. password\_hash - text field
   4. fullname - text field
   5. acctype - text field
2. Таблиця users
   1. Primary Key - id - integer field
   2. Foreign Key - authorId - integer field
   3. heading - text field
   4. post - text field
   5. author - text field
3. Таблиця comments
   1. Primary Key - id - integer field
   2. Foreign Key - postId - integer field
   3. Foreign Key - authorId - integer field
   4. comment - text field
   5. createdAt - text(datetime ISO 8601) field

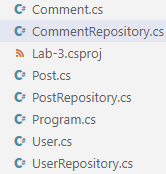


**Код програми**

|  |
| --- |
| Program.cs |
| using System;  using Npgsql;  using System.Collections.Generic;  using System.Diagnostics;  namespace Lab\_3  {  class Program  {  static void PrintInfo()  {  Console.WriteLine(">>>COMMANDLIST:");  Console.ForegroundColor = ConsoleColor.Green;  Console.WriteLine("getUsers");  Console.WriteLine("getUser [id]");  Console.WriteLine("createUser [username] [password] [fullname] [acctype]");  Console.WriteLine("deleteUser [id]");  Console.WriteLine("changeUserAcctypeData [username] [new\_acctype]");  Console.WriteLine("changeUserPassword [username] [new\_password]");  Console.WriteLine("changeUsername [username] [new\_username]");  Console.WriteLine("searchUsers [searchValue]");  Console.ForegroundColor = ConsoleColor.Magenta;  Console.WriteLine("getPosts");  Console.WriteLine("getPost [id]");  Console.WriteLine("getPostsByUserId [userId]");  Console.WriteLine("createPost [authorId] [heading] [post] [author]");  Console.WriteLine("deletePost [id]");  Console.WriteLine("changePost [id] [heading] [post]");  Console.WriteLine("searchPosts [searchValue]");  Console.ForegroundColor = ConsoleColor.Cyan;  Console.WriteLine("getComments");  Console.WriteLine("getComment [id]");  Console.WriteLine("getCommentsByUserId [userId]");  Console.WriteLine("getCommentsByPostId [postId]");  Console.WriteLine("createComment [postId] [authorId] [comment]");  Console.WriteLine("deleteComment [id]");  Console.WriteLine("changeComment [id] [comment]");  Console.WriteLine("searchComments [searchValue]");  Console.ForegroundColor = ConsoleColor.Red;  Console.WriteLine("random [nrows]");  Console.WriteLine("exit");  Console.ForegroundColor = ConsoleColor.White;  }  static void UserEditProcess(string[] parts, UserRepository ur, PostRepository pr, CommentRepository cr)  {  if(parts[0] == "getUsers" && parts.Length == 1)  {  Console.ForegroundColor = ConsoleColor.Yellow;  Console.WriteLine("Users list:");  foreach(User u in ur.GetAll())  {  Console.WriteLine(u);  }  Console.ForegroundColor = ConsoleColor.White;  }  else if(parts[0] == "getUser" && parts.Length == 2)  {  int id;  bool success = int.TryParse(parts[1], out id);  if(success && id > 0)  {  Console.ForegroundColor = ConsoleColor.Yellow;  User u = ur.GetById(id);  if(u == default)  {  Console.WriteLine("Can't find user");  }  else  {  Console.WriteLine(ur.GetById(id));  }  Console.ForegroundColor = ConsoleColor.White;  }  else  {  Console.ForegroundColor = ConsoleColor.Red;  Console.WriteLine("Id must be integer and > 0");  Console.ForegroundColor = ConsoleColor.White;  }  }  else if(parts[0] == "createUser" && parts.Length == 5)  {  if(ur.UserExists(parts[1]))  {  Console.ForegroundColor = ConsoleColor.Red;  Console.WriteLine($"User with username:{parts[1]} already exist");  Console.ForegroundColor = ConsoleColor.White;  }  else  {  User user = new User()  {  username = parts[1],  password\_hash = parts[2],  fullname = parts[3],  acctype = parts[4]  };  Console.WriteLine($"User was added");  ur.Insert(user);  }  }  else if(parts[0] == "deleteUser" && parts.Length == 2)  {  int id;  bool success = int.TryParse(parts[1], out id);  if(success && id > 0)  {  if(ur.DeleteById(id) == 1)  {  List<Post> userpostlist = pr.GetByUserId(id);  List<Comment> usercommentlist = cr.GetByAuthorId(id);  foreach(Post p in userpostlist)  {  cr.DeleteByPostId(p.id);  pr.DeleteById(p.id);  }  foreach(Comment c in usercommentlist)  {  cr.DeleteById(c.id);  }  Console.WriteLine($"User was deleted");  }  else  {  Console.ForegroundColor = ConsoleColor.Red;  Console.WriteLine($"Can't find user with userID:{id}");  Console.ForegroundColor = ConsoleColor.White;  }  }  else  {  Console.ForegroundColor = ConsoleColor.Red;  Console.WriteLine("Id must be integer and > 0");  Console.ForegroundColor = ConsoleColor.White;  }  }  else if(parts[0] == "changeUserAcctypeData" && parts.Length == 3)  {  if(ur.ChangeAcctypeData(parts[1], parts[2]) == 1)  {  Console.WriteLine("Acctype data was changed");  }  else  {  Console.ForegroundColor = ConsoleColor.Red;  Console.WriteLine($"Can't find user with username:{parts[1]}");  Console.ForegroundColor = ConsoleColor.White;  }  }  else if(parts[0] == "changeUserPassword" && parts.Length == 3)  {  if(ur.ChangePassword(parts[1], parts[2]) == 1)  {  Console.WriteLine("Password was changed");  }  else  {  Console.ForegroundColor = ConsoleColor.Red;  Console.WriteLine($"Can't find user with username:{parts[1]}");  Console.ForegroundColor = ConsoleColor.White;  }  }  else if(parts[0] == "changeUsername" && parts.Length == 3)  {  if(ur.ChangeUsername(parts[1], parts[2]) == 1)  {  Console.WriteLine("Username was changed");  }  else  {  Console.ForegroundColor = ConsoleColor.Red;  Console.WriteLine($"Can't find user with username:{parts[1]}");  Console.ForegroundColor = ConsoleColor.White;  }  }  else if(parts[0] == "searchUsers" && parts.Length == 2)  {  Console.WriteLine("Founded results:");  foreach(User u in ur.SearchUsers(parts[1]))  {  Console.WriteLine(u);  }  }  else  {  Console.ForegroundColor = ConsoleColor.Red;  Console.WriteLine($"Invalid command");  Console.ForegroundColor = ConsoleColor.White;  }  }  static void PostEditProcess(string[] parts, PostRepository pr, CommentRepository cr)  {  if(parts[0] == "getPosts" && parts.Length == 1)  {  Console.ForegroundColor = ConsoleColor.Yellow;  Console.WriteLine("Posts list:");  foreach(Post p in pr.GetAll())  {  Console.WriteLine(p);  }  Console.ForegroundColor = ConsoleColor.White;  }  else if(parts[0] == "getPost" && parts.Length == 2)  {  int id;  bool success = int.TryParse(parts[1], out id);  if(success && id > 0)  {  Console.ForegroundColor = ConsoleColor.Yellow;  Post p = pr.GetById(id);  if(p == default)  {  Console.WriteLine("Can't find post");  }  else  {  Console.WriteLine(pr.GetById(id));  }  Console.ForegroundColor = ConsoleColor.White;  }  else  {  Console.ForegroundColor = ConsoleColor.Red;  Console.WriteLine("Id must be integer and > 0");  Console.ForegroundColor = ConsoleColor.White;  }  }  else if(parts[0] == "getPostByUserId" && parts.Length == 2)  {  int id;  bool success = int.TryParse(parts[1], out id);  if(success && id > 0)  {  Console.ForegroundColor = ConsoleColor.Yellow;  List<Post> posts = pr.GetByUserId(id);  Console.WriteLine($"Posts by author with id:{id}");  foreach(Post p in posts)  {  Console.WriteLine(p);  }  Console.ForegroundColor = ConsoleColor.White;  }  else  {  Console.ForegroundColor = ConsoleColor.Red;  Console.WriteLine("Id must be integer and > 0");  Console.ForegroundColor = ConsoleColor.White;  }  }  else if(parts[0] == "createPost" && parts.Length == 5)  {  int id;  bool success = int.TryParse(parts[1], out id);  if(success && id > 0)  {  Post post = new Post()  {  authorId = Convert.ToInt32(parts[1]),  heading = parts[2],  post = parts[3],  author = parts[4]  };  Console.WriteLine($"Post was added");  pr.Insert(post);  }  else  {  Console.ForegroundColor = ConsoleColor.Red;  Console.WriteLine("Id must be integer and > 0");  Console.ForegroundColor = ConsoleColor.White;  }  }  else if(parts[0] == "deletePost" && parts.Length == 2)  {  int id;  bool success = int.TryParse(parts[1], out id);  if(success && id > 0)  {  if(pr.DeleteById(id) == 1)  {  cr.DeleteByPostId(id);  Console.WriteLine($"Post was deleted");  }  else  {  Console.ForegroundColor = ConsoleColor.Red;  Console.WriteLine($"Can't find post with ID:{id}");  Console.ForegroundColor = ConsoleColor.White;  }  }  else  {  Console.ForegroundColor = ConsoleColor.Red;  Console.WriteLine("Id must be integer and > 0");  Console.ForegroundColor = ConsoleColor.White;  }  }  else if(parts[0] == "changePost" && parts.Length == 4)  {  int id;  bool success = int.TryParse(parts[1], out id);  if(success && id > 0)  {  if(pr.UpdateData(id, parts[2], parts[3]) == 1)  {  Console.WriteLine("Post data was changed");  }  else  {  Console.ForegroundColor = ConsoleColor.Red;  Console.WriteLine($"Can't find post with id:{parts[1]}");  Console.ForegroundColor = ConsoleColor.White;  }  }  else  {  Console.ForegroundColor = ConsoleColor.Red;  Console.WriteLine("Id must be integer and > 0");  Console.ForegroundColor = ConsoleColor.White;  }  }  else if(parts[0] == "searchPosts" && parts.Length == 2)  {  Console.WriteLine("Founded results:");  foreach(Post p in pr.SearchPosts(parts[1]))  {  Console.WriteLine(p);  }  }  else  {  Console.ForegroundColor = ConsoleColor.Red;  Console.WriteLine($"Invalid command");  Console.ForegroundColor = ConsoleColor.White;  }  }  static void CommentEditProcess(string[] parts, CommentRepository cr)  {  if(parts[0] == "getComments" && parts.Length == 1)  {  Console.ForegroundColor = ConsoleColor.Yellow;  Console.WriteLine("Comments list:");  foreach(Comment c in cr.GetAll())  {  Console.WriteLine(c);  }  Console.ForegroundColor = ConsoleColor.White;  }  else if(parts[0] == "getComments" && parts.Length == 2)  {  int id;  bool success = int.TryParse(parts[1], out id);  if(success && id > 0)  {  Console.ForegroundColor = ConsoleColor.Yellow;  Comment c = cr.GetById(id);  if(c == default)  {  Console.WriteLine("Can't find comment");  }  else  {  Console.WriteLine(cr.GetById(id));  }  Console.ForegroundColor = ConsoleColor.White;  }  else  {  Console.ForegroundColor = ConsoleColor.Red;  Console.WriteLine("Id must be integer and > 0");  Console.ForegroundColor = ConsoleColor.White;  }  }  else if(parts[0] == "getCommentsByUserId" && parts.Length == 2)  {  int id;  bool success = int.TryParse(parts[1], out id);  if(success && id > 0)  {  Console.ForegroundColor = ConsoleColor.Yellow;  List<Comment> comments = cr.GetByAuthorId(id);  Console.WriteLine($"Comments by author with id:{id}");  foreach(Comment c in comments)  {  Console.WriteLine(c);  }  Console.ForegroundColor = ConsoleColor.White;  }  else  {  Console.ForegroundColor = ConsoleColor.Red;  Console.WriteLine("Id must be integer and > 0");  Console.ForegroundColor = ConsoleColor.White;  }  }  else if(parts[0] == "getCommentsByPostId" && parts.Length == 2)  {  int id;  bool success = int.TryParse(parts[1], out id);  if(success && id > 0)  {  Console.ForegroundColor = ConsoleColor.Yellow;  List<Comment> comments = cr.GetByPostId(id);  Console.WriteLine($"Comments by post with id:{id}");  foreach(Comment c in comments)  {  Console.WriteLine(c);  }  Console.ForegroundColor = ConsoleColor.White;  }  else  {  Console.ForegroundColor = ConsoleColor.Red;  Console.WriteLine("Id must be integer and > 0");  Console.ForegroundColor = ConsoleColor.White;  }  }  else if(parts[0] == "createComment" && parts.Length == 4)  {  int postid;  bool success1 = int.TryParse(parts[1], out postid);  int authorid;  bool success2 = int.TryParse(parts[1], out authorid);  if(success1 && postid > 0 && success2 && authorid > 0)  {  Comment comment = new Comment()  {  authorId = Convert.ToInt32(parts[2]),  postId = Convert.ToInt32(parts[1]),  comment = parts[3]  };  Console.WriteLine($"Comment was added");  cr.Insert(comment);  }  else  {  Console.ForegroundColor = ConsoleColor.Red;  Console.WriteLine("Id must be integer and > 0");  Console.ForegroundColor = ConsoleColor.White;  }  }  else if(parts[0] == "deleteComment" && parts.Length == 2)  {  int id;  bool success = int.TryParse(parts[1], out id);  if(success && id > 0)  {  if(cr.DeleteById(id) == 1)  {  Console.WriteLine($"Comment was deleted");  }  else  {  Console.ForegroundColor = ConsoleColor.Red;  Console.WriteLine($"Can't find comment with ID:{id}");  Console.ForegroundColor = ConsoleColor.White;  }  }  else  {  Console.ForegroundColor = ConsoleColor.Red;  Console.WriteLine("Id must be integer and > 0");  Console.ForegroundColor = ConsoleColor.White;  }  }  else if(parts[0] == "changeComment" && parts.Length == 3)  {  int id;  bool success = int.TryParse(parts[1], out id);  if(success && id > 0)  {  if(cr.UpdateData(id, parts[2]) == 1)  {  Console.WriteLine("Comment data was changed");  }  else  {  Console.ForegroundColor = ConsoleColor.Red;  Console.WriteLine($"Can't find comment with id:{parts[1]}");  Console.ForegroundColor = ConsoleColor.White;  }  }  else  {  Console.ForegroundColor = ConsoleColor.Red;  Console.WriteLine("Id must be integer and > 0");  Console.ForegroundColor = ConsoleColor.White;  }  }  else if(parts[0] == "searchComments" && parts.Length == 2)  {  Console.WriteLine("Founded results:");  foreach(Comment c in cr.SearchComments(parts[1]))  {  Console.WriteLine(c);  }  }  else  {  Console.ForegroundColor = ConsoleColor.Red;  Console.WriteLine($"Invalid command");  Console.ForegroundColor = ConsoleColor.White;  }  }  static void Main(string[] args)  {  Stopwatch sWatch = new Stopwatch();  string connect\_parameters = "Server=localhost;Port=5432;User Id=postgres;Password=admin;Database=facebook;";  NpgsqlConnection connection = new NpgsqlConnection(connect\_parameters);  connection.Open();  UserRepository ur = new UserRepository(connection);  PostRepository pr = new PostRepository(connection);  CommentRepository cr = new CommentRepository(connection);  PrintInfo();  string command = "";  while(command != "exit")  {  command = Console.ReadLine();  string[] parts = command.Split(' ');  if(parts[0] == "random" && parts.Length == 2)  {  int nrows;  bool success = int.TryParse(parts[1], out nrows);  if(success && nrows > 0)  {  sWatch.Reset();  sWatch.Start();  //Console.WriteLine(nrows);  NpgsqlCommand comm = connection.CreateCommand();  comm.CommandText =  @"  SELECT fillrandomvalues(@nrows)  ";  comm.Parameters.AddWithValue("@nrows", nrows);  comm.ExecuteNonQuery();  sWatch.Stop();  Console.WriteLine ($"Действие выполнено,затраченое время: {sWatch.ElapsedMilliseconds.ToString()} мс");  continue;  }  else  {  Console.ForegroundColor = ConsoleColor.Red;  Console.WriteLine("Nrows must be integer and > 0");  Console.ForegroundColor = ConsoleColor.White;  }  }  string[] usersCommand = new string[]{"getUsers", "getUser", "createUser", "deleteUser", "changeUserAcctypeData", "changeUserPassword", "changeUsername", "searchUsers"};  string[] postsCommand = new string[]{"getPosts", "getPost", "getPostsByUserId", "createPost", "deletePost", "changePost", "searchPosts"};  string[] commentsCommand = new string[]{"getComments", "getComment", "getCommentsByUserId", "getCommentsByPostId", "createComment", "deleteComment", "changeComment", "searchComments"};  bool marker = false;  foreach(string i in usersCommand)  {  if(parts[0] == i)  {  marker = true;  sWatch.Reset();  sWatch.Start();  UserEditProcess(parts, ur, pr, cr);  sWatch.Stop();  Console.WriteLine ($"Действие выполнено,затраченое время: {sWatch.ElapsedMilliseconds.ToString()} мс");  }  }  foreach(string i in postsCommand)  {  if(parts[0] == i)  {  marker = true;  sWatch.Start();  PostEditProcess(parts, pr, cr);  sWatch.Stop();  Console.WriteLine ($"Действие выполнено,затраченое время: {sWatch.ElapsedMilliseconds.ToString()} мс");  }  }  foreach(string i in commentsCommand)  {  if(parts[0] == i)  {  marker = true;  sWatch.Start();  CommentEditProcess(parts, cr);  sWatch.Stop();  Console.WriteLine ($"Действие выполнено,затраченое время: {sWatch.ElapsedMilliseconds.ToString()} мс");  }  }  if(command != "exit" && marker == false)  {  Console.ForegroundColor = ConsoleColor.Red;  Console.WriteLine("Invalid Command");  Console.ForegroundColor = ConsoleColor.White;  }  }  connection.Close();  }  }  } |
| User.cs |
| using System;  namespace Lab\_3  {  public class User  {  public int id;  public string username;  public string password\_hash;  public string fullname;  public string acctype;  public string createdAt;  // public User(string username, string hash, string fullname, string acctype, string createdAt)  // {  // this.username = username;  // this.password\_hash = hash;  // this.fullname = fullname;  // this.acctype = acctype;  // this.createdAt = createdAt;  // }  public override string ToString()  {  return $"[{id}] [{username}] [{fullname}] [{acctype}]";  }  }  }  //add post Author Heading Post  //remove post id  //add comment Author postId comment  //remove comment id |
| Post.cs |
| using System.Collections.Generic;  namespace Lab\_3  {  public class Post  {  public int id;  public string author;  public int commentsCount = 0;  public string createdAt;  public string heading;  public string post;  public int authorId;  public int pinned\_commentId = -1;  public List<Comment> commentList = null;  // public Post(string heading, string post, string author, int commentsCount, string createdAt, int authorId)  // {  // this.author = author;  // this.commentsCount = commentsCount;  // this.createdAt = createdAt;  // this.heading = heading;  // this.post = post;  // this.authorId = authorId;  // }  public override string ToString()  {  return $"[{id}] [{authorId}] [{heading}] [{post}] [{author}]";  }  }  } |
| Comment.cs |
| namespace Lab\_3  {  public class Comment  {  public int id;  public string author;  public string createdAt;  public string comment;  public int postId;  public int authorId;  // public Comment(string author, string comment, string createdAt, int postId, int authorId)  // {  // this.author = author;  // this.postId = postId;  // this.createdAt = createdAt;  // this.comment = comment;  // this.authorId = authorId;  // }  public override string ToString()  {  return $"[{id}] [{postId}] [{authorId}] [{comment}]";  }  }  } |
| CommentRepository.cs |
| using Npgsql;  using System;  using static System.Math;  using System.Collections.Generic;  namespace Lab\_3  {  public class CommentRepository  {  private NpgsqlConnection connection;  public CommentRepository(NpgsqlConnection connection)  {  this.connection = connection;  }  public List<Comment> GetAll()  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"SELECT \* FROM comments";  List<Comment> list = ReadCommentsFromCommand(command);  return list;  }  public List<Comment> GetPage(int pageNumber, int pageSize)  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"SELECT \* FROM comments LIMIT @pageSize OFFSET @pageSize \* (@pageNumber - 1)";  command.Parameters.AddWithValue(@"pageSize", pageSize);  command.Parameters.AddWithValue(@"pageNumber", pageNumber);  List<Comment> list = ReadCommentsFromCommand(command);  return list;  }  public List<Comment> GetPostPage(int pageNumber, int pageSize, Post post)  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"SELECT \* FROM comments WHERE postId = @postId LIMIT @pageSize OFFSET @pageSize \* (@pageNumber - 1)";  command.Parameters.AddWithValue(@"pageSize", pageSize);  command.Parameters.AddWithValue(@"pageNumber", pageNumber);  command.Parameters.AddWithValue(@"postId", post.id);  List<Comment> list = ReadCommentsFromCommand(command);  return list;  }  public List<Comment> SearchComments(string searchValue)  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"SELECT \* FROM comments WHERE comment LIKE @filter";  command.Parameters.AddWithValue("@filter", '%' + searchValue + '%');    List<Comment> list = ReadCommentsFromCommand(command);  return list;  }  static List<Comment> ReadCommentsFromCommand(NpgsqlCommand command)  {  NpgsqlDataReader reader = command.ExecuteReader();  List<Comment> list = new List<Comment>();  while (reader.Read())  {  Comment comment = ReadComments(reader);  list.Add(comment);  }  reader.Close();  return list;  }  public int GetTotalPages(int size)  {  long nOperators = CountComments();  double pageSize = size;  return (int)Ceiling(nOperators / pageSize);  }  public int GetTotalPostPages(int size, Post post)  {  long nOperators = CountPostComments(post.id);  double pageSize = size;  return (int)Ceiling(nOperators / pageSize);  }  public long CountComments()  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"SELECT COUNT(\*) FROM comments";  return (long)command.ExecuteScalar();  }  public long CountPostComments(int postId)  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"SELECT COUNT(\*) FROM comments WHERE postId = @postId";  command.Parameters.AddWithValue(@"postId", postId);  return (long)command.ExecuteScalar();  }  public bool CommentExists(int id)  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"SELECT \* FROM comments WHERE id = @id";  command.Parameters.AddWithValue("@id", id);  NpgsqlDataReader reader = command.ExecuteReader();  bool result = reader.Read();  return result;  }  public void Insert(Comment comment)  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText =  @"  INSERT INTO comments (postid, authorid, comment)  VALUES (@postId, @authorId, @comment);  ";  command.Parameters.AddWithValue("@comment", comment.comment);  command.Parameters.AddWithValue("@postId", comment.postId);  command.Parameters.AddWithValue("@authorId", comment.authorId);  command.ExecuteNonQuery();  }  public Comment GetById(int id)  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"SELECT \* FROM comments WHERE id = @id";  command.Parameters.AddWithValue("@id", id);    NpgsqlDataReader reader = command.ExecuteReader();    if (reader.Read())  {  Comment co = ReadComments(reader);  reader.Close();  return co;  }  else  {  reader.Close();  return null;  }  }  public int DeleteById(int id)  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"DELETE FROM comments WHERE id = @id";  command.Parameters.AddWithValue("@id", id);    int nChanged = command.ExecuteNonQuery();    if (nChanged == 0)  {  return 0;  }  else  {  return 1;  }  }  public void DeleteByPostId(int id)  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"DELETE FROM comments WHERE postId = @id";  command.Parameters.AddWithValue("@id", id);    int nChanged = command.ExecuteNonQuery();    if (nChanged == 0)  {  return;  }  else  {  DeleteByPostId(id);  }  }  public List<Comment> GetByPostId(int id)  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"SELECT \* FROM comments WHERE postId = @id";  command.Parameters.AddWithValue("@id", id);    NpgsqlDataReader reader = command.ExecuteReader();  List<Comment> list = new List<Comment>();  while(reader.Read())  {  Comment co = ReadComments(reader);  list.Add(co);  }  reader.Close();  return list;  }  public List<Comment> GetByAuthorId(int id)  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"SELECT \* FROM comments WHERE authorId = @id";  command.Parameters.AddWithValue("@id", id);    NpgsqlDataReader reader = command.ExecuteReader();  List<Comment> list = new List<Comment>();  while(reader.Read())  {  Comment co = ReadComments(reader);  list.Add(co);  }  reader.Close();  return list;  }  public int UpdateData(int id, string comment)  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"UPDATE comments SET comment = @comment WHERE id = @id";    command.Parameters.AddWithValue("@id", id);  command.Parameters.AddWithValue("@comment", comment);  int nChanged = command.ExecuteNonQuery();  if (nChanged == 0)  {  return 0;  }  else  {  return 1;  }  }  static Comment ReadComments(NpgsqlDataReader reader)  {  // public Comment(string author, string comment, string createdAt, int postId, int authorId)  // Console.WriteLine(@"{reader.GetString(0)}, {reader.GetString(2)}, {reader.GetString(3)}, {reader.GetString(4)}, {reader.GetString(5)}");  Comment Comment = new Comment(){  postId = Convert.ToInt32(reader.GetInt32(1)),  comment = reader.GetString(3),  authorId = Convert.ToInt32(reader.GetInt32(2))  };  Comment.id = Convert.ToInt32(reader.GetInt32(0));    return Comment;  }  }  } |
| PostRepository.cs |
| using Npgsql;  using System;  using static System.Math;  using System.Collections.Generic;  namespace Lab\_3  {  public class PostRepository  {  private NpgsqlConnection connection;  public PostRepository(NpgsqlConnection connection)  {  this.connection = connection;  }  public List<Post> GetAll()  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"SELECT \* FROM posts";  List<Post> list = ReadPostsFromCommand(command);  return list;  }  public int GetTotalPages(int size)  {  long nOperators = CountPosts();  double pageSize = size;  return (int)Ceiling(nOperators / pageSize);  }  public int GetSearchPagesCount(int size, string searchValue)  {  if(string.IsNullOrEmpty(searchValue))  {  return this.GetTotalPages(size);  }  return (int)Ceiling(this.SearchPosts(searchValue).Count / (double)size);  }  public List<Post> GetSearchPage(int pageNumber, int pageSize, string searchValue)  {  if(string.IsNullOrEmpty(searchValue))  {  return this.GetPage(pageNumber, pageSize);  }  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"SELECT \* FROM posts WHERE post LIKE @filter OR heading LIKE @filter LIMIT @pageSize OFFSET @pageSize \* (@pageNumber - 1)";  command.Parameters.AddWithValue(@"pageSize", pageSize);  command.Parameters.AddWithValue(@"pageNumber", pageNumber);  command.Parameters.AddWithValue("@filter", '%' + searchValue + '%');  List<Post> list = ReadPostsFromCommand(command);  return list;  }  public List<Post> GetSearchMyPage(int pageNumber, int pageSize, int authorId, string searchValue)  {  if(string.IsNullOrEmpty(searchValue))  {  return this.GetMyPage(pageNumber, pageSize, authorId);  }  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"SELECT \* FROM posts WHERE authorId = @authorId AND (heading LIKE @filter OR post LIKE @filter) LIMIT @pageSize OFFSET @pageSize \* (@pageNumber - 1)";  command.Parameters.AddWithValue("@authorId", authorId);  command.Parameters.AddWithValue(@"pageSize", pageSize);  command.Parameters.AddWithValue(@"pageNumber", pageNumber);  command.Parameters.AddWithValue("@filter", '%' + searchValue + '%');  List<Post> list = ReadPostsFromCommand(command);  return list;  }  public List<Post> SearchPosts(string searchValue)  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"SELECT \* FROM posts WHERE post LIKE @filter OR heading LIKE @filter";  command.Parameters.AddWithValue("@filter", '%' + searchValue + '%');    List<Post> list = ReadPostsFromCommand(command);  return list;  }  public List<Post> SearchMyPosts(string searchValue, int authorId)  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"SELECT \* FROM posts WHERE authorId = @authorId AND (heading LIKE @filter OR post LIKE @filter)";  command.Parameters.AddWithValue("@authorId", authorId);  command.Parameters.AddWithValue("@filter", '%' + searchValue + '%');    List<Post> list = ReadPostsFromCommand(command);  return list;  }  public int GetSearchMyTotalPages(int size, int authorId, string searchValue)  {  long nOperators = CountMySearchPosts(authorId, searchValue);  double pageSize = size;  return (int)Ceiling(nOperators / pageSize);  }  public int GetMyTotalPages(int size, int authorId)  {  long nOperators = CountMyPosts(authorId);  double pageSize = size;  return (int)Ceiling(nOperators / pageSize);  }  public bool PostExists(int id)  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"SELECT \* FROM posts WHERE id = @id";  command.Parameters.AddWithValue("@id", id);  NpgsqlDataReader reader = command.ExecuteReader();  bool result = reader.Read();  return result;  }  public void Insert(Post post)  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText =  @"  INSERT INTO public.posts(authorid, heading, post, author)  VALUES (@authorId, @heading, @post, @author);  ";  command.Parameters.AddWithValue("@heading", post.heading);  command.Parameters.AddWithValue("@post", post.post);  command.Parameters.AddWithValue("@author", post.author);  command.Parameters.AddWithValue("@authorId", post.authorId);  command.ExecuteNonQuery();  }  public long CountPosts()  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"SELECT COUNT(\*) FROM posts";  return (long)command.ExecuteScalar();  }  public long CountMySearchPosts(int authorId, string searchValue)  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"SELECT COUNT(\*) FROM posts WHERE authorId = @authorId AND (heading LIKE @filter OR post LIKE @filter)";  command.Parameters.AddWithValue("@authorId", authorId);  command.Parameters.AddWithValue("@filter", '%' + searchValue + '%');  return (long)command.ExecuteScalar();  }  public long CountMyPosts(int authorId)  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"SELECT COUNT(\*) FROM posts WHERE authorId = @authorId";  command.Parameters.AddWithValue(@"authorId", authorId);  return (long)command.ExecuteScalar();  }  public Post GetById(int id)  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"SELECT \* FROM posts WHERE id = @id";  command.Parameters.AddWithValue("@id", id);    NpgsqlDataReader reader = command.ExecuteReader();    if (reader.Read())  {  Post po = ReadPost(reader);  reader.Close();  return po;  }  else  {  reader.Close();  return null;  }  }  public int DeleteById(int id)  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"DELETE FROM posts WHERE id = @id";  command.Parameters.AddWithValue("@id", id);    int nChanged = command.ExecuteNonQuery();    if (nChanged == 0)  {  return 0;  }  else  {  return 1;  }  }  public List<Post> GetByUserId(int id)  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"SELECT \* FROM posts WHERE authorId = @id";  command.Parameters.AddWithValue("@id", id);    NpgsqlDataReader reader = command.ExecuteReader();  List<Post> list = new List<Post>();  while(reader.Read())  {  Post po = ReadPost(reader);  list.Add(po);  };  reader.Close();  return list;  }  public int UpdateData(int id, string heading, string post)  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"UPDATE posts SET heading = @heading, post = @post WHERE id = @id";    command.Parameters.AddWithValue("@id", id);  command.Parameters.AddWithValue("@heading", heading);  command.Parameters.AddWithValue("@post", post);  int nChanged = command.ExecuteNonQuery();  if (nChanged == 0)  {  return 0;  }  else  {  return 1;  }  }  static Post ReadPost(NpgsqlDataReader reader)  {  Post Post = new Post(){  author = reader.GetString(4),  heading = reader.GetString(2),  post = reader.GetString(3),  authorId = Convert.ToInt32(reader.GetInt32(1)),  };  Post.id = Convert.ToInt32(reader.GetInt32(0));    return Post;  }  public List<Post> GetPage(int pageNumber, int pageSize)  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"SELECT \* FROM posts LIMIT @pageSize OFFSET @pageSize \* (@pageNumber - 1)";  command.Parameters.AddWithValue(@"pageSize", pageSize);  command.Parameters.AddWithValue(@"pageNumber", pageNumber);  List<Post> list = ReadPostsFromCommand(command);  return list;  }  public List<Post> GetMyPage(int pageNumber, int pageSize, int authorId)  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"SELECT \* FROM posts WHERE authorId = @authorId LIMIT @pageSize OFFSET @pageSize \* (@pageNumber - 1)";  command.Parameters.AddWithValue(@"pageSize", pageSize);  command.Parameters.AddWithValue(@"pageNumber", pageNumber);  command.Parameters.AddWithValue(@"authorId", authorId);  List<Post> list = ReadPostsFromCommand(command);  return list;  }  static List<Post> ReadPostsFromCommand(NpgsqlCommand command)  {  NpgsqlDataReader reader;  reader = command.ExecuteReader();  List<Post> list = new List<Post>();  while (reader.Read())  {  Post post = ReadPost(reader);  list.Add(post);  }  reader.Close();  return list;  }  }  } |
| UserRepository.cs |
| using Npgsql;  using System;  using static System.Math;  using System.Collections.Generic;  namespace Lab\_3  {  public class UserRepository  {  private NpgsqlConnection connection;  public UserRepository(NpgsqlConnection connection)  {  this.connection = connection;  }  public List<User> GetAll()  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"SELECT \* FROM users";  List<User> list = ReadUsersFromCommand(command);  return list;  }  public void Insert(User user)  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText =  @"  INSERT INTO users (password\_hash, fullname, acctype, username)  VALUES (@password\_hash, @fullname, @acctype, @username);  ";  command.Parameters.AddWithValue("@username", user.username);  command.Parameters.AddWithValue("@password\_hash", user.password\_hash);  command.Parameters.AddWithValue("@fullname", user.fullname);  command.Parameters.AddWithValue("@acctype", user.acctype);  command.ExecuteNonQuery();  }  public int GetTotalPages(int size)  {  long nOperators = CountUsers();  double pageSize = size;  return (int)Ceiling(nOperators / pageSize);  }  public List<User> GetPage(int pageNumber, int pageSize)  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"SELECT \* FROM users LIMIT @pageSize OFFSET @pageSize \* (@pageNumber - 1)";  command.Parameters.AddWithValue("@pageSize", pageSize);  command.Parameters.AddWithValue("@pageNumber", pageNumber);  List<User> list = ReadUsersFromCommand(command);  return list;  }  public int GetSearchTotalPages(int size, string searchValue)  {  if(string.IsNullOrEmpty(searchValue))  {  return this.GetTotalPages(size);  }  return (int)Ceiling(this.SearchUsers(searchValue).Count / (double)size);  }  public List<User> GetSearchPage(int pageNumber, int pageSize, string searchValue)  {  if(string.IsNullOrEmpty(searchValue))  {  return this.GetPage(pageNumber, pageSize);  }  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"SELECT \* FROM users WHERE username LIKE @filter OR fullname LIKE @filter LIMIT @pageSize OFFSET @pageSize \* (@pageNumber - 1)";  command.Parameters.AddWithValue("@pageSize", pageSize);  command.Parameters.AddWithValue("@pageNumber", pageNumber);  command.Parameters.AddWithValue("@filter", '%' + searchValue + '%');  List<User> list = ReadUsersFromCommand(command);  return list;  }  public bool UserExists(string username)  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"SELECT \* FROM users WHERE username = @username";  command.Parameters.AddWithValue("@username", username);  NpgsqlDataReader reader = command.ExecuteReader();  bool result = reader.Read();  reader.Close();  return result;  }  public long CountUsers()  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"SELECT COUNT(\*) FROM users";  return (long)command.ExecuteScalar();  }  public User GetByUsername(string username)  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"SELECT \* FROM users WHERE username = @username";  command.Parameters.AddWithValue("@username", username);    NpgsqlDataReader reader = command.ExecuteReader();    if (reader.Read())  {  User user = ReadUser(reader);  reader.Close();  return user;  }  else  {  reader.Close();  return null;  }  }  public User GetById(int id)  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"SELECT \* FROM users WHERE id = @id";  command.Parameters.AddWithValue("@id", id);    NpgsqlDataReader reader = command.ExecuteReader();    if (reader.Read())  {  User user = ReadUser(reader);  reader.Close();  return user;  }  else  {  reader.Close();  return null;  }  }  public int DeleteById(int id)  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"DELETE FROM users WHERE id = @id";  command.Parameters.AddWithValue("@id", id);    int nChanged = command.ExecuteNonQuery();    if (nChanged == 0)  {  return 0;  }  else  {  return 1;  }  }  public int ChangeAcctypeData(string username, string acctype)  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"UPDATE users SET acctype = @acctype WHERE username = @username";    command.Parameters.AddWithValue("@username", username);  command.Parameters.AddWithValue("@acctype", acctype);  int nChanged = command.ExecuteNonQuery();  if (nChanged == 0)  {  return 0;  }  else  {  return 1;  }  }  public int ChangePassword(string username, string password\_hash)  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"UPDATE users SET password\_hash = @password\_hash WHERE username = @username";    command.Parameters.AddWithValue("@username", username);  command.Parameters.AddWithValue("@password\_hash", password\_hash);  int nChanged = command.ExecuteNonQuery();  if (nChanged == 0)  {  return 0;  }  else  {  return 1;  }  }  public int ChangeUsername(string username, string newUsername)  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"UPDATE users SET username = @newUsername WHERE username = @username";    command.Parameters.AddWithValue("@username", username);  command.Parameters.AddWithValue("@newUsername", newUsername);  int nChanged = command.ExecuteNonQuery();  if (nChanged == 0)  {  return 0;  }  else  {  return 1;  }  }  public List<User> SearchUsers(string searchValue)  {  NpgsqlCommand command = this.connection.CreateCommand();  command.CommandText = @"SELECT \* FROM users WHERE username LIKE @filter OR fullname LIKE @filter";  command.Parameters.AddWithValue("@filter", '%' + searchValue + '%');    List<User> list = ReadUsersFromCommand(command);  return list;  }  static List<User> ReadUsersFromCommand(NpgsqlCommand command)  {  NpgsqlDataReader reader;  reader = command.ExecuteReader();  List<User> list = new List<User>();  while (reader.Read())  {  User user = ReadUser(reader);  list.Add(user);  }  reader.Close();  return list;  }  static User ReadUser(NpgsqlDataReader reader)  {  User User = new User(){  username = reader.GetString(4),  password\_hash = reader.GetString(1),  fullname = reader.GetString(2),  acctype = reader.GetString(3)  };  User.id = Convert.ToInt32(reader.GetInt32(0));    return User;  }  }  } |

Завдання #1







Завдання #2

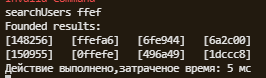
Мною були використані такі команди створення індексів:

CREATE INDEX n\_post

ON posts (id, authorId, heading, post, author)

CREATE UNIQUE INDEX n\_user

ON users (id, password\_hash, fullname, acctype, username)

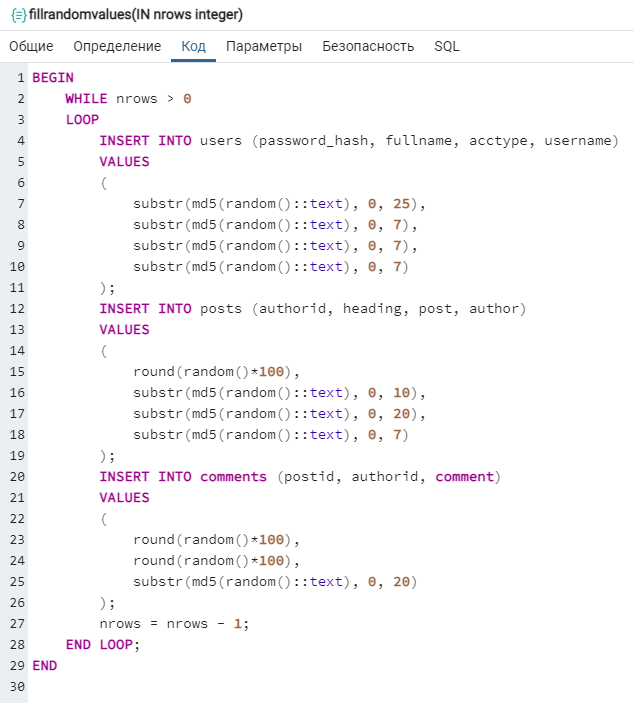


Найпростіший спосіб вирішення завдання пошуку записів у базі даних, які відповідають певному критерію— повний перебір. Але зі зростанням кількості записів продуктивність такого підходу помітно падатиме. Для підвищення продуктивності пошуку створюються допоміжні структури індекси. Використовуючи індекси, можна істотно підняти швидкість пошуку, тому що дані в індексі зберігаються у формі, що дозволяє нам у процесі пошуку не розглядати області, які не можуть містити шукані елементи.

Важливо, що використання індексів як скорочує час пошуку у абсолютному вираженні, а й зменшує алгоритмічну складність процесу пошуку. Це означає, що час, необхідний пошук за допомогою індексів, при зростанні обсягу бази даних зростатиме істотно повільніше, ніж із використанні повного перебору.

Але, у своїй, найчастіше значно уповільнюють операції вставки, оновлення, видалення, т.я. потрібно оновлювати сам індекс.

Завдання #3

**Висновок**

Здобув вміння проектувати бази даних та отримав практичні

навички створення реляційних баз даних за допомогою PostgreSQL.

Розробив модель «сутність-зв’язок» предметної галузі, обраної самостійно, відповідно до пункту «Вимоги до ER-моделі». Перетворив розроблену модель у схему бази даних (таблиці)

PostgreSQL. Виконав нормалізацію схеми бази даних до третьої нормальної форми. Ознайомився із інструментарієм PostgreSQL та pgAdmin 4.

Навчився створювати запити, валідацію даних уведених користувачем, обробку виняткових ситуацій. Навчився створювати функції та тригери на мові SQL за допомогою PostgreSQL та pgAdmin 4.

Здобув практичні навички використання засобів оптимізації СУБД PostgreSQL.