**Лабораторные работы по курсу**

**Базы данных**

**Лабораторная работа 8**

**«Разработка программы на языке С**

**для работы с базой данных»**

**Москва, 2024**

**Оглавление**

[1. Теоретическая часть 3](#_Toc156254148)

[1.1. Создание проекта 3](#_Toc156254149)

[1.1.1. Настройка среды 3](#_Toc156254150)

[1.1.2. Работа с библиотекой *libpq* 3](#_Toc156254151)

[1.1.3. Приложение по работе с базой данных 4](#_Toc156254152)

[1.2. SQL-инъекции 6](#_Toc156254153)

[1.3. Защита от SQL-инъекций 8](#_Toc156254154)

[2. Практическая часть 9](#_Toc156254155)

[2.1. Задание 1. 9](#_Toc156254156)

[2.2. Задание 2. 9](#_Toc156254157)

[2.3. Задание 3. 10](#_Toc156254158)

[Контрольные вопросы 10](#_Toc156254159)

[Список использованной литературы 10](#_Toc156254160)

1. Теоретическая часть

В современном мире базы данных редко встречаются отдельно от программного обеспечения, позволяющего с ними взаимодействовать. Подобные программы разрабатываются на разных языках программирования и служат для корректного взаимодействия между пользователем и непосредственно базой данных. В данной лабораторной работе будет рассмотрено создание программного интерфейса для работы с базой данных. Программа будет разработана на языке С.

* 1. Создание проекта
     1. Настройка среды

При разработке программного обеспечения требуется библиотека для взаимодействия с *postgresql*. В текущей лабораторной работе будет использоваться библиотека libpq. Проверить ее наличие в системе можно с помощью команды:

dpkg -s libpq-dev | grep Version

В случае отсутствия библиотеки, необходимо выполнить следующие команды:

sudo apt-get update

sudo apt-get -y install libpq-dev

Для работы с программным кодом удобно использовать программу Visual Studio Code.

Изображение выглядит как Графика, символ, Цвет электрик, Красочность

Автоматически созданное описание

Рисунок 1 Ярлык программы VS Code

Запустив Visual Studio Code, создайте новый файл (*Ctrl+N* или *File -> New File)*. Сохраните его под имением *main.c* в новую директорию, в которой будут располагаться все файлы проекта.

Запишите в него следующий код:

1. #include <stdio.h>
3. int main() {
4. printf("I love MPSU!**\n**");
5. return 0;
6. }

Для компиляции программы создайте файл сборки – *Makefile*. Для этого в директории проекта создайте файл *Makefile* изапишите в него следующий код:

1. outfile=main
2. all:
3. gcc -o $(outfile) main.c
4. clean:
5. rm -rf $(outfile)

Для запуска компиляции откройте командную строку, перейдите в директорию проекта и выполните команду *make*. Для удобства открыть её возможно с использованием программы *vscode*. Для этого в нижней панели перейдите во вкладку TERMINAL. Если панели нет, то вытяните её за верхний край.

После выполнения команды make в директории проекта должен появиться исполняемый файл *main*. Запустите его. В случае успеха, на экране должна появиться надпись “I love MPSU!”. Чтобы с помощью одной команды скомпилировать и запустить файл возможно выполнить следующую команду:

make && ./main

* + 1. Функции библиотеки *libpq*

Для взаимодействия программного обеспечения с базой данных используется библиотека *libpq*. Она содержит в себе набор функций, позволяющих подключаться к базе данных, выполнять SQL запросы и производить другие действия.

Ниже приведены основные функции

|  |  |
| --- | --- |
| Название функции | Описание действия |
| PGconn \*PQconnectdbParams(  const char \* const \*keywords,  const char \* const \*values,  int expand\_dbname); | Подключение к базе данных. |
| PGresult \*PQexec(PGconn \*conn,  const char \*command); | Передача команды серверу и ожидание результата. |
| PGresult \*PQexecParams(  PGconn \*conn,  const char \*command,  int nParams,  const Oid \*paramTypes,  const char \* const \*paramValues,  const int \*paramLengths,  const int \*paramFormats,  int resultFormat); | Передача команды серверу и ожидание результата. Имеет возможность передать параметры отдельно от текста SQL-команды. |
| void PQclear(PGresult \*res); | Освобождение области памяти, выделенной под PGresult |
| int PQntuples(const PGresult \*res); | Возвращает число строк (кортежей) в полученной выборке. |
| int PQnfields(const PGresult \*res); | Возвращает число столбцов (полей) в каждой строке полученной выборки. |
| ExecStatusType PQresultStatus(const PGresult \*res); | Возвращает статус выполнения команды. |
| char \*PQerrorMessage(const PGconn \*conn); | Возвращает сообщение об ошибке. |
| void PQfinish(PGconn \*conn); | Закрывает соединение с сервером. Освобождает память, используемую объектом PGconn. |

Более подробно об этих функциях можно прочитать в документации [1]

* + 1. Разработка программного обеспечения для работы с базой данных

Измените файл *main.c*, записав в него следующий программный код.

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <string.h>
4. #include <libpq-fe.h>
6. */\**
7. *Функция выхода из программы с сообщением об ошибке.*
8. *\*/*
9. void err\_exit(PGconn \*conn) {
10. PQfinish(conn); *//Закрытие соединения с сервером базы данных*
11. exit(1); *//Функция возвращает значение 1 -> программа завершилась аварийно*
12. }
14. */\**
15. *Функция печати результата запроса на экран*
16. *\*conn -> дескриптор подключения к серверу*
17. *query -> текст запроса*
18. *\*/*
19. void print\_query(PGconn \*conn, char\* query){
20. PGresult \*res = PQexec(conn, query); *//Передача запроса на сервер*
21. int rows = PQntuples(res); *//Получение числа строк в результате запроса*
22. int cols = PQnfields(res); *//Получение числа столбцов в результате запроса*

25. *// Проверка наличия возвращенных запросом строк.*
26. *// В случае их отсутствие - аварийное завершение программы*
27. if (PQresultStatus(res) != PGRES\_TUPLES\_OK) {
28. printf("No data retrieved**\n**");
29. PQclear(res);
30. err\_exit(conn);
31. }
33. *//Вывод результата запроса на экран*
34. for(int i=0; i<rows; i++) {
35. for(int j=0; j<cols; j++) {
36. printf("%s ", PQgetvalue(res, i, j));
37. }
38. printf("**\n**");
39. }
40. PQclear(res);
41. }
43. int main() {
44. *//Подключение к серверу базы данных*
45. PGconn \*conn = PQconnectdb("user=SAB password=123456 dbname=students");
47. *//Проверка статуса подключения. В случае ошибки - аварийное завершение программы*
48. if (PQstatus(conn) == CONNECTION\_BAD) {
49. fprintf(stderr, "Connection to database failed: %s**\n**",PQerrorMessage(conn));
50. err\_exit(conn);
51. }
52. print\_query(conn, "SELECT \* FROM student LIMIT 5;");
53. PQfinish(conn);
54. return 0;
55. }

Данный код осуществляет подключение к серверу БД students, выполнение требуемого запроса (выборки пяти записей из БД) и вывод результатов на экран пользователя.

Подключение к запущенному серверу PostgreSQL происходит к базе данных – students, пользователь – SAB, пароль для подключения – 123456. Данный пользователь был создан в одной из предыдущих лабораторных работ. Если этого не было сделано, то создайте его, чтобы было возможно подключиться к базе данных. Для этого выполните следующий запрос.

CREATE ROLE "SAB" WITH

LOGIN

SUPERUSER

CREATEDB

CONNECTION LIMIT -1

PASSWORD '123456';

Для компиляции программы измените *Makefile*:

1. libpath=-I/usr/include/postgresql
2. libs=-lpq
3. outfile=main
4. all:
5. gcc -o $(outfile) main.c $(libpath) $(libs)
6. clean:
7. rm -rf $(outfile)

В случае успешного подключения и выполнения запроса в консоль будет выведена информация о студентах из учебной базы данных.

* 1. SQL-инъекции

В процессе разработки приложений уделяется большое внимание вопросам безопасности. Проблема SQL-инъекций является одной из угроз сохранению и утечки данных. Рассмотрим эту проблему на нашем примере.

Добавим в программу функцию, позволяющую выводить на экран оценку студента, по его номеру студенческого билета и названию дисциплины.

1. /\*
2. Функция поиска и вывода на экран информации о студенте из базы
3. по номеру его студенческого билета и названию дисциплины
4. \*conn -> дескриптор подключения к серверу
5. st\_id -> номер студенческого билета
6. f\_name -> название дисиплины
7. \*/
8. void find\_students\_by\_ID(PGconn \*conn, int st\_id, char\* f\_name)
9. {
10. // создание динамического массива, для помещения в него запроса
11. char\* query = (char\*)malloc(512\*sizeof(char));
12. // Удаление символа окончания строки из введенной фамилии студента
13. int pos = strlen(f\_name) - 1;
14. if (f\_name[pos] == '**\n**')
15. f\_name[pos]='**\0**';
17. // Запрос, возвращающий ФИ, название дисциплины и оценку студента
18. // значения st\_id и f\_name подставляются в запрос
19. // на место %d и %s соответственно
20. sprintf(query, " **\**
21. SELECT SURNAME, **\**
22. NAME, **\**
23. FIELD\_NAME, **\**
24. MARK **\**
25. FROM STUDENT S **\**
26. JOIN PUBLIC.FIELD\_COMPREHENSION F\_C ON S.STUDENT\_ID = F\_C.STUDENT\_ID **\**
27. JOIN PUBLIC.FIELD F ON F.FIELD\_ID = F\_C.FIELD **\**
28. WHERE S.STUDENT\_ID = %d AND F.FIELD\_NAME = **\'**%s**\'**" , st\_id, f\_name);
30. PGresult \*res = PQexec(conn, query);
32. if (PQresultStatus(res) != PGRES\_TUPLES\_OK) {
33. printf("No data retrieved**\n**");
34. PQclear(res);
35. err\_exit(conn);
36. }
37. // Вывод на экран найенной информации
38. int rows = PQntuples(res);
39. int cols = PQnfields(res);
41. for(int i=0; i<rows; i++) {
42. for(int j=0; j<cols; j++) {
43. printf("%s ", PQgetvalue(res, i, j));
44. }
45. printf("**\n**");
46. }
47. PQclear(res);
48. free(query);
49. }

Для обращения к ней заменим в основной программе main() строку с вызовом функции *print\_query(…)* на последовательность команд:

1. int st\_id;
2. printf("Enter student ID number: **\n**");
3. scanf("%d", &st\_id);
4. getchar();
5. printf("Enter field name: **\n**");
6. fgets(query, 256, stdin);
8. find\_students\_by\_ID(conn, st\_id, query);

В этой части кода происходит ввод номера студенческого билета и строки, содержащей название дисциплины.[[1]](#footnote-1) Далее следует вызов функции *find\_students\_by\_ID().* На вход функция принимает значение дескриптора соединения, номер студенческого билета и название дисциплины.

При корректном вызове функции программа выберет из базы данных студента и выведет его оценку по дисциплине на экран.

Enter student ID number:

856271

Enter field name:

Базы данных

Михеев Александр Базы данных 2

Однако, злоумышленники при вводе требуемой информации (в нашем случае при вводе названия дисциплины) могут задать определенное значение, которое позволит им получить доступ к закрытым данным или внести изменения в БД (в нашем случае изменить оценку).

Попробуем ввести в программу вместо названия дисциплины следующее значение:

1' OR 1=1; UPDATE FIELD\_COMPREHENSION F\_C SET MARK = 5 FROM FIELD F WHERE F.FIELD\_ID = F\_C.FIELD AND STUDENT\_ID = 856271 AND FIELD\_NAME = 'Базы данных' ;--

Строка вывода программы будет выглядеть следующим образом:

Enter student ID number:

856271

Enter field name:

1' OR 1=1; UPDATE FIELD\_COMPREHENSION F\_C SET MARK = 5 FROM FIELD F WHERE F.FIELD\_ID = F\_C.FIELD AND STUDENT\_ID = 856271 AND FIELD\_NAME = 'Базы данных' ;--

No data retrieved

Если запустить программу еще раз, введя данные студента 856271 и дисциплину «Базы данных», то оценка будет уже другая.

Михеев Александр Базы данных 5

Давайте разберемся как это произошло. В функцию *find\_students\_by\_ID()* в качестве входных аргументов передаются введённые пользователем student\_ID и field\_name и их значения подставляются в формируемый внутри функции запрос.

Рассмотрим получившийся в результате слияния строк SQL запрос.

SELECT SURNAME,

NAME,

FIELD\_NAME,

MARK

FROM STUDENT S

JOIN PUBLIC.FIELD\_COMPREHENSION F\_C ON S.STUDENT\_ID = F\_C.STUDENT\_ID

JOIN PUBLIC.FIELD F ON F.FIELD\_ID = F\_C.FIELD

WHERE S.STUDENT\_ID = 856271 AND F.FIELD\_NAME = '

*1' OR 1=1;*

*UPDATE FIELD\_COMPREHENSION F\_C SET MARK = 5*

*FROM FIELD F*

*WHERE F.FIELD\_ID = F\_C.FIELD AND STUDENT\_ID = 856271 AND FIELD\_NAME = 'Базы данных' ;*

*--* '

Вставленная строка (выделена курсивом) будет расположена между кавычками. Выражение F.FIELD\_NAME = '*1' OR 1=1* всегда истинно, поэтому для студента с заданным student\_ID будет выполнена команда, идущая после точки с запятой в этой строке. Произойдет изменение оценки данного студента на 5 по дисциплине «Базы данных». В самом конце с помощью символа «--» будет закомментирован апостроф, завершающий строку. Таким образом с помощью внесенного SQL кода (*SQL инъекции*) возможно произвести изменение в базе данных.

* 1. Защита от SQL-инъекций

Самым простым способом защиты от SQL инъекций является проверка введенного значения на корректность. Например, если вводится число, то необходимо убедиться, что введенное значение именно число. Также существует возможность ограничить набор допустимых символов. При разработке графического интерфейса можно минимизировать ввод данных в поля, заменив его на выбор из вариантов.

Чаще всего в качестве защиты от SQL инъекций выбирают один из следующих способов:

1. Изменение кавычек на апострофы.

Если для пользователя не так важен вводимый символ, то возможно изменить символ **'** на **`**. Таким образом вредоносный код будет принят как одна длинная строка и добавлен в базу данных.

Приведем функцию, изменяющую символ апострофа.

1. /\*
2. Функция изменения символа апострофа
3. \*query -> текст запроса
4. \*/
5. void correct\_apostrof(char\* query){
6. for(int i = 0; i < strlen(query); ++i)
7. if(query[i] == '**\'**')
8. query[i] = '`';
9. }
10. Экранирование кавычек

Если для пользователя важен именно символ кавычки, то его для ввода в базу данных необходимо экранировать. Напомним, что кавычки в PostgreSQL экранируются повторением данного символа. Например, если мы хотим внести в таблицу строку имя John O'Brien, то запрос на добавление будет выглядеть следующим образом:

INSERT INTO ApostropheTest(Name, Surname)

VALUES ('John', 'O''Brien')



Приведем возможный вариант программной реализации функции.

1. /\*
2. Функция экранирования кавычек
3. \*query -> текст запроса
4. \*query\_new -> текст запроса после изменения
5. \*/
6. void add\_escape\_quotes(char\* query, char\* query\_new){
7. int j = 0;
8. for(int i = 0; i < strlen(query); ++i){
9. if(query[i] == '**\'**'){
10. query\_new[j] = '**\'**';
11. query\_new[j+1] = '**\'**';
12. j++;
13. }
14. else
15. query\_new[j] = query[i];
16. j++;
17. }
18. }
19. Использование параметризированных запросов.

С помощью функции *PQexecParams* возможно отправить на сервер подготовленный запрос. Функция очень похожа на PQexec, но предлагает дополнительную функциональность: значения параметров могут быть указаны отдельно от самой строки-команды. Это позволяет избежать использования кавычек и экранирующих символов, что является трудоёмким методом, часто приводящим к ошибкам. *PQexecParams* позволяет включать не более одной SQL-команды в строку запроса. (В ней могут содержаться точки с запятой, однако может присутствовать не более одной непустой команды.) Таким образом, выполнение атак с использованием SQL-инъекций становится гораздо более трудоемкой задачей.

Ниже приведен программный код реализации функции поиска оценок студента, но с использованием параметризированных запросов. Значения передаются в функцию с помощью параметров $1 и $2.

1. /\*
2. Функция поиска информации о студенте из базы
3. по его номеру студенческого билета и названию дисциплины
4. с использованием параметрического запроса
5. \*conn -> дескриптор подключения к серверу
6. st\_id -> номер студенческого билета
7. f\_name -> название дисиплины
8. \*/
9. void find\_students\_by\_ID\_Params(PGconn \*conn, int st\_id, char\* f\_name)
10. {
11. //Создание массива для задания параметров запроса
12. const char \*paramValues[2];
13. paramValues[0] = (char\*) &st\_id;
14. paramValues[1] = f\_name;
16. char \*query = "SELECT SURNAME, **\**
17. NAME, **\**
18. FIELD\_NAME, **\**
19. MARK **\**
20. FROM STUDENT S **\**
21. JOIN PUBLIC.FIELD\_COMPREHENSION F\_C ON S.STUDENT\_ID = F\_C.STUDENT\_ID **\**
22. JOIN PUBLIC.FIELD F ON F.FIELD\_ID = F\_C.FIELD **\**
23. WHERE S.STUDENT\_ID=$1 AND F.FIELD\_NAME=**\'**$2**\'**";
25. //Вызов параметрического запроса
26. PGresult \*res = PQexecParams(conn, query, 2, NULL, paramValues, NULL, NULL, 0);
28. if (PQresultStatus(res) != PGRES\_TUPLES\_OK) {
29. printf("No data retrieved**\n**");
30. PQclear(res);
31. err\_exit(conn);
32. }
34. int rows = PQntuples(res);
35. int cols = PQnfields(res);
36. for(int i=0; i<rows; i++) {
37. for(int j=0; j<cols; j++) {
38. printf("%s ", PQgetvalue(res, i, j));
39. }
40. printf("**\n**");
41. }
42. PQclear(res);
43. }
44. Практическая часть
    1. Задание 1.
       1. *Защита от SQL инъекций*

Используя функции для защиты от SQL инъекций, повторите пример из теоретической части и убедитесь в том, что она работает.

* 1. Задание 2.
     1. *Реализация функции для взаимодействия с базой данных*

Реализуйте любой из SQL запросов, который вы выполняли в одной из предыдущих работ в виде функции на языке С, взаимодействующей с базой данных. Предусмотрите защиту от SQL инъекции любым из приведенных выше способов.

* + 1. *Предупреждение об инъекции*

Добавьте функцию, срабатывающую в случае попытки внедрения SQL инъекции и выводящее значение, что текущему пользователю вынесено предупреждение.

* + 1. *Использование представлений*

Замените вызов SQL запроса из программного кода на вызов представления. Объясните преимущества и недостатки такого подхода.

* 1. Задание 3.
     1. *Доработка программного обеспечения – добавление аутентификации*

Создайте две роли – «Ваши инициалы junior» и «Ваши инициалы admin». Junior обладает только правом чтения из базы данных, admin – полные права. Добавьте в программу поле ввода логина/пароля для подключения к базе данных от имени различных пользователей. Типы пользователей – администратор, пользователь.

* + 1. *Доработка программного обеспечения – добавление CRUD операций*

От имени администратора сделайте возможность добавлять, изменять и удалять значения оценок у любого студента.

От имени пользователя сделайте возможность выполнять запрос на просмотр всех его оценок и поиск конкретной строки по введенным значениям фамилии, имени и номера группы. Используя разработанную функцию, попробуйте выполнить SQL инъекцию на изменение значение оценки от имени данного пользователя.

Контрольные вопросы

1. Как возможно установить соединение с базой данных на языке С?
2. Что такое SQL инъекция?
3. Какие методы защиты от SQL инъекций вы знаете?
4. Для чего предназначены функции *PQclear* и *PQfinish*
5. В чем преимущества использования параметризованных запросов?

Список использованной литературы

|  |  |
| --- | --- |
| [1] | «Документация libpq,» [В Интернете]. Available: https://postgrespro.ru/docs/postgresql/9.6/libpq. |
| [2] | «Исходный код СУБД postgres,» [В Интернете]. Available: https://github.com/postgres/postgres. [Дата обращения: 30 01 2023]. |
| [3] | Документация к PostgreSQL 15.1, 2022. |
| [4] | Е. Рогов, PostgreSQL изнутри, 1-е ред., Москва: ДМК Пресс, 2023, p. 662 . |
| [5] | Б. А. Новиков, Е. А. Горшкова и Н. Г. Графеева, Основы технологии баз данных, 2-е ред., Москва: ДМК пресс, 2020, p. 582. |
| [6] | Е. П. Моргунов, PostgreSQL. Основы языка SQL, 1-е ред., Санкт-Петербург: БХВ-Петербург, 2018, p. 336. |

1. Единственным «подводным камнем» является наличие функции *getchar()* между командами ввода. Проблема заключается в том, что функция *scanf* считывает все символы до символа окончания строки, остающемся в буфере ввода. И при вызове *fgets* данный символ попадает в строку *query* и функция завершает свою работу, т.к. считывание происходит до символа окончания строки **включая** его. Поэтому, с помощью функции getchar символ окончания строки считывается и не попадает в *fgets.* [↑](#footnote-ref-1)