

**Федеральное государственное автономное образовательное
учреждение высшего образования «Санкт-Петербургский государственный
электротехнический университет «ЛЭТИ»
им. В.И. Ульянова (Ленина)»**

кафедра информационных систем

КУРСОВАЯ РАБОТА
по дисциплине “Управление данными”
Тема: “Проектирование базы данных”

Выполнил: Рудаков Артём Алексеевич

Группа: №2373

Вариант: №19

Санкт-Петербург, 2024

Задание на курсовую работу

Спроектировать базу данных, построить программу, обеспечивающую взаимодействия с ней в режиме диалога, для работников технического архива предприятия.

Технический архив содержит стеллажи, полки и ячейки, в которых хранится документация.

Ячейка архива может быть пустой или хранить все экземпляры одного документа.

Каждый экземпляр документации имеет инвентарный номер и название.

В БД должна содержаться следующая информация:

- номер стеллажа;
- номер полки;
- номер ячейки;
- название документа и темы, к которой он относится;
- инвентарный номер;
- количество экземпляров документа, содержащихся в ячейке;
- даты поступления документов в архив и запросов к ним.

За документом могут обратиться абоненты архива, характеризующиеся ФИО, номером и телефоном отдела, где они работают.

При работе с БД могут потребоваться следующие сведения:

1. определить название наиболее часто требуемого документа;
2. определить общее количество документов на заданную тему;
3. определить тему по названию документа;
4. определять название документа, который имеется в максимальном количестве экземпляров;

5. определять отдел, работника которого наиболее часто обращаются к архиву;
6. установить ФИО абонента, обращавшегося последним к указанному документу.

Администратор БД может вносить следующие изменения: – добавление нового документа; – изменение номера телефона указанного отдела; – удаление экземпляра некоторого документа.

Необходимо предусмотреть возможность выдачи справки об абонентах отдела, пользующихся архивом, и отчета о работе архива (число единиц хранения, названия документов, поступивших в архив за последний месяц, количество экземпляров каждого документа, место его хранения).

Введение

Целью курсовой работы является разработка базы данных. Была выбрана СУБД PostgreSQL. Для доступа к базе данных, внесения изменений в неё со стороны администратора и получения информации со стороны абонента был разработан клиент на языке C++. Все манипуляции с данными в процессе выполнения курсовой работы были выполнены в программе на C++ с помощью библиотеки, дающей доступ к работе с базами данных PostgreSQL. Клиент будет запрашивать данные для входа в личный кабинет абонента технического архива, только после чего будет возможна работа с базой данных, что предусматривает защиту базы данных от внешних вмешательств. Также приложение поддерживает проверку допустимых в поле ввода значений.

Анализ предметной области

Предметная область проекта – технический архив, содержащий стеллажи, полки, ячейки, документы, экземпляры документов. В архиве работают сотрудники из разных отделов. Сотрудники могут запрашивать документы из архива.

Обоснование модели данных

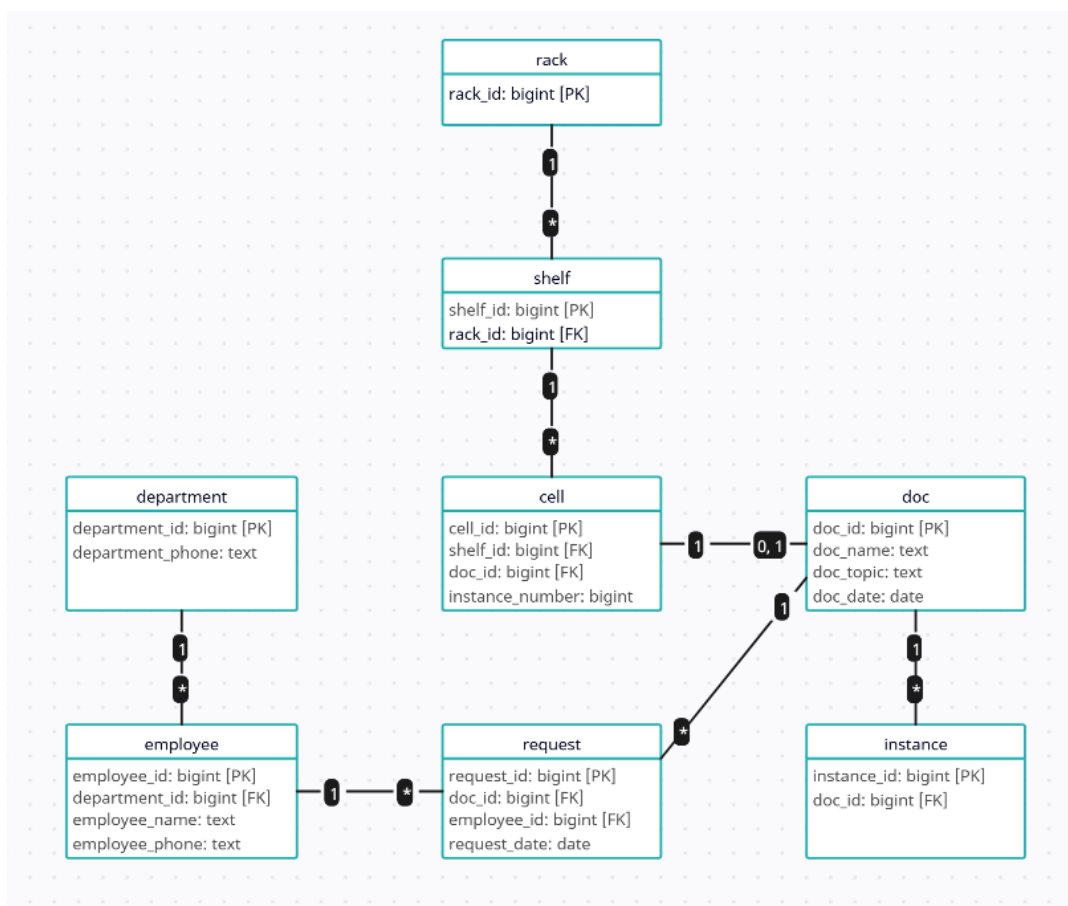


Рис. 1. ER модель

Стеллаж (rack) содержит много полок (shelf) – один ко многим; полка (shelf) содержит много ячеек (cell) – один ко многим; ячейка (cell) может либо хранить документ (doc), либо нет – отношение один к одному или нулю; документ (doc) может иметь много экземпляров (instance) – один ко многим; к документу (doc) может производиться много запросов (request) – один ко многим; отдел архива (department) может иметь много сотрудников (employee) – один ко многим; сотрудник (employee) может создавать много запросов (request) – один ко многим.

Обоснование выбора СУБД

Была выбрана PostgreSQL, т.к. данная СУБД является реляционной, что позволяет связывать объекты базы данных. Также PostgreSQL является серверной СУБД, что позволит хранить базу данных на сервере, к которому

смогут обращаться несколько работников архива, а не только тот, кто хранит БД на своём компьютере.

Процесс создания базы данных

Созданы таблицы (Tables):

- 1) Таблица **rack** – стеллаж, содержит **rack_id [PK]**.
- 2) Таблица **shelf** – полка, содержит **shelf_id [PK]**, **rack_id**.
- 3) Таблица **cell** – ячейка, содержит **cell_id [PK]**, **shelf_id**, **doc_id**, **instance_number**.
- 4) Таблица **doc** – документ, содержит **doc_id [PK]**, **doc_name**, **doc_topic**, **doc_date**.
- 5) Таблица **instance** – экземпляр документа, содержит **instance_id [PK]**, **doc_id**.
- 6) Таблица **request** – запрос к документу, содержит **request_id [PK]**, **doc_id**, **employee_id**, **request_date**.
- 7) Таблица **department** – отдел архива, содержит **department_id [PK]**, **department_phone**.
- 8) Таблица **employee** – абонент архива, содержит **employee_id [PK]**, **department_id**, **employee_name**, **employee_phone**.

Добавлены ограничения (Constraints):

- 1) Уникальные ключи **_id** (Primary Key) для каждой таблицы.
- 2) Уникальное значение **doc_id** (UNIQUE) для каждой ячейки (в таблице **cell**).
- 3) Уникальные ключи **_id** (Primary Key) для каждой таблицы.
- 4) Внешние ключи (Foreign Key) для следующих отношений:
 - **shelf.rack_id** → **rack.rack_id**;
 - **cell.shelf_id** → **shelf.shelf_id**;
 - **cell.doc_id** → **doc.doc_id**;
 - **instance.doc_id** → **doc.doc_id**;

- **employee.department_id → department.department_id;**
- **request.doc_id → doc.doc_id;**
- **request.employee_id → employee.employee_id.**

Табл. 1. Таблицы

№	Название	Столбцы
1	rack	rack_id [PK]
2	shelf	shelf_id [PK], rack_id
3	cell	cell_id [PK], shelf_id, doc_id, instance_number
4	doc	doc_id [PK], doc_name, doc_topic, doc_date
5	instance	instance_id [PK], doc_id
6	request	request_id [PK], doc_id, employee_id, request_date
7	employee	employee_id [PK], department_id, employee_name, employee_phone
8	department	department_id [PK], department_phone

Описание функций групп пользователей

Табл. 2. Назначение прав доступа

Объект	Абоненты	Администратор
Таблица rack	S	SUID
Таблица shelf	S	SUID
Таблица cell	S	SUID
Таблица doc	S	SUID
Таблица instance	S	SUID
Таблица request	S	SUID
Таблица employee	S	SUID
Таблица department	S	SUID

Описание функция управления данными

Функции абонента архива:

- определить название наиболее часто требуемого документа;
- определить общее количество документов на заданную тему;

- определить тему по названию документа;
- определять название документа, который имеется в максимальном количестве экземпляров;
- определять отдел, работника которого наиболее часто обращаются к архиву;
- установить ФИО абонента, обращавшегося последним к указанному документу.

Администратор БД может вносить следующие изменения:

- добавление нового документа;
- изменение номера телефона указанного отдела;
- удаление экземпляра некоторого документа.

Организация защиты БД

Защиту базы данных от внешних вмешательств предусматривает система входа в личный кабинет работника архива. Некорректные данные проверяются компилятором C++ и не посылаются в БД в случае, если они не прошли проверку.

Заключение

Были разработаны БД на основе СУБД PostgreSQL и клиентское приложение на C++ для взаимодействия с БД.

Список используемых источников

- 1) Ульман Д., Уидом Д. Системы баз данных. Полный курс С. М.: Вильямс, 2017. – 1088 с.
- 2) Документация PostgreSQL <<https://postgrespro.ru/docs/postgresql>>
- 3) Взаимодействие C++ с PostgreSQL
<<https://www.postgresql.org/docs/7.2/libpqplusplus.html>>

Приложение А. Руководство пользователя БД

Введение

Область применения – сотрудники технического архива. Сотрудник, имеющий доступ к БД должен знать перечень применяемых функций по отношению к БД. Сотрудник должен быть в базе данных с указанным ФИО и номером телефона для получения доступа.

Назначение и условия применения

Функции, для автоматизации которых предназначено данное программное обеспечение:

- Посчитать число документов по названию темы;
- Найти тему по названию документа;
- Найти название документа с наибольшим количеством экземпляров;
- Найти наиболее требуемый документ;
- Найти отдел, сотрудник которого, наиболее часто обращается к архиву;
- Найти ФИО сотрудника, обращавшегося последним к определённому документу;
- Выполнить запрос к документу;
- Обновить номер телефона отдела;
- Удалить экземпляр документа;
- Добавить новый документ;
- Получить справку об абонентах архива;
- Получить справку о работе архива.

Подготовка к работе

Чтобы работать с БД, нам нужны данные.

- Добавим документы в doc:

	doc_id [PK] bigint	doc_name text	doc_topic text	doc_date date
1	1	Technical manual for the operation of equipment X	Equipment operation	2024-11-07
2	2	Maintenance instructions for equipment Y	Technical maintenance	2024-11-07
3	3	Regulations for the preventive maintenance of equipment...	Technical maintenance	2024-11-07
4	4	Production safety manual	Safety	2024-11-07
5	5	Calibration methodology for measuring instruments	Calibration	2024-11-07
6	6	Report on the testing of equipment A	Testing	2024-11-07
7	7	Technical requirements for system B	Technical requirements	2024-11-07
8	8	Modernization plan for equipment C	Modernization	2024-11-07
9	9	User guide for software D	Software	2024-11-07
10	10	Safety recommendations for equipment Y	Safety	2024-11-07

Рис. 2. Таблица doc (документы)

Документы под номерами 2 и 3 имеют одинаковую тему. 4 и 10 – тоже.

- Добавим **стеллажи** в **rack**:

	rack_id [PK] bigint
1	1
2	2

Рис. 3. Таблица rack (стеллажи)

- Добавим **полки** в **shelf**:

	shelf_id [PK] bigint	rack_id bigint
1	1	1
2	2	1
3	3	2

Рис. 4. Таблица shelf (полки)

Полка 1 находится в стеллаже 1.

Полка 2 находится в стеллаже 1.

Полка 3 находится в стеллаже 2.

- Добавим **экземпляры** в **instance**:

	instance_id [PK] bigint	doc_id bigint
1	1	3
2	2	7
3	3	1
4	4	2
5	5	1
6	6	5
7	7	9
8	8	3
9	9	6
10	10	7
11	11	1
12	12	3
13	13	5
14	14	2
15	15	1

Рис. 5. Таблица instance (экземпляры)

- Добавим **ячейки** в **cell**:

	cell_id [PK] bigint	shelf_id bigint	doc_id bigint	instance_number bigint
1	1	1	3	3
2	2	1	1	4
3	3	1	9	1
4	4	1	10	0
5	5	2	2	2
6	6	2	4	0
7	7	2	6	1
8	8	3	[null]	0
9	9	3	8	0
10	10	3	7	2

Рис. 6. Таблица cell (ячейки)

Всего 10 ячеек: четыре на 1 полке, три на 2 полке и три на 3 полке.

Ячейка 8 пуста, число экземпляров документа, следовательно, равно 0.

Максимальное количество экземпляров у документа 1 в ячейке 2.

- Добавим **отделы** в **department**:

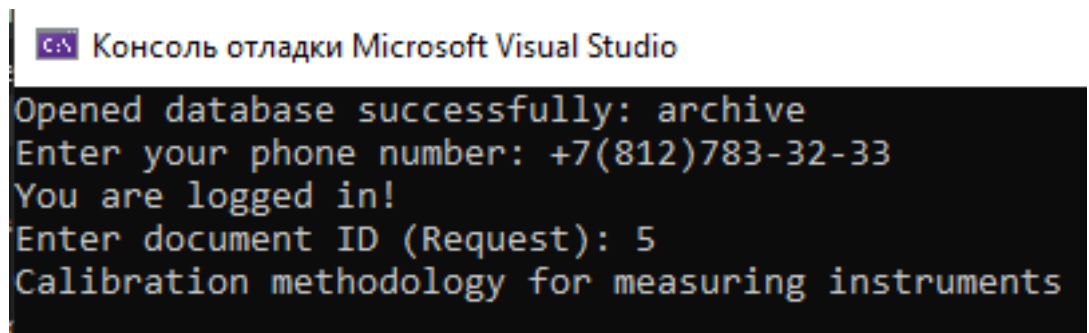
	department_id [PK] bigint	department_phone text
1	1	+7(812)923-97-24
2	2	+7(812)807-65-41

Рис. 7. Таблица department (отделы)

- Добавим запросы в request:

	request_id [PK] bigint	doc_id bigint	employee_id bigint	request_date date
1	1	1	3	2024-11-07
2	2	3	2	2024-11-07
3	3	9	1	2024-11-07
4	4	5	1	2024-11-07
5	5	10	3	2024-11-07
6	6	1	2	2024-11-07
7	7	5	2	2024-11-07
8	8	6	2	2024-11-07

Рис. 8. Таблица request (запросы)



```

C:\> Консоль отладки Microsoft Visual Studio
Opened database successfully: archive
Enter your phone number: +7(812)783-32-33
You are logged in!
Enter document ID (Request): 5
Calibration methodology for measuring instruments

```

Рис. 9. Процесс выполнения запроса с выводом названия документа

Описание операций

- Определить название наиболее часто требуемого документа:

QueryQuery History

1

SELECT doc_name FROM doc

2

WHERE doc_id =

3

(SELECT doc_id FROM

4

(SELECT doc_id, COUNT(doc_id)

5

FROM request

6

GROUP BY doc_id

7

ORDER BY doc_id)

8

WHERE count =

9

(SELECT MAX(count) FROM

10

(SELECT doc_id, COUNT(doc_id)

11

FROM request

12

GROUP BY doc_id

13

ORDER BY doc_id)

14

)

15

LIMIT 1);

Data OutputMessagesNotifications

≡+

📄

▼

📋

▼

🗑

🗄

⬇

📈

SQL

	doc_name	
	text	🔒
1	Technical manual for the operation of equipment X	

Рис. 10. SQL запрос и его результат

Наиболее часто запрашиваемым документом оказался документ под номером 1 - "Technical manual for the operation of equipment X". Он встречается в таблице запросов 2 раза. В этом можно убедиться, взглянув на Рис. 8.

- Определить общее количество документов на заданную тему:

```

239
240  int count_docs_by_topic(std::string topic, pqxx::connection& C) {
241      int retval = 0;
242      try {
243          pqxx::work W(C);
244          std::string query = "SELECT COUNT(*) FROM doc WHERE doc_topic = '" + topic + "';";
245          pqxx::result R = W.exec(query);
246          if (!R.empty()) {
247              return R[0][0].as<int>();
248          }
249      }
250      catch (std::exception& e) {
251          std::cerr << e.what() << std::endl;
252          return 0;
253      }
254      return retval;
255  }
256

```

Рис. 11. Функция, посылающая SQL запрос

Для определения этих сведений написана функция, посылающая SQL запрос (обведён красным) на сервер. Запрос содержит пользовательскую переменную – название темы – которую пользователь вводит с клавиатуры.

```

C# Консоль отладки Microsoft Visual Studio
Opened database successfully: archive
Enter your phone number: +7(812)783-32-33
You are logged in!
Количество документов на тему Safety - 2

```

Рис. 12. Результат выполнения SQL запроса

Выведено сообщение для темы Safety.

- Определить тему по названию документа:

```

256
257  std::string find_topic_by_doc_name(std::string name, pqxx::connection& C) {
258      std::string retval = "";
259      try {
260          pqxx::work W(C);
261          std::string query = "SELECT doc_topic FROM doc WHERE doc_name = '" + name + "';";
262          pqxx::result R = W.exec(query);
263          if (!R.empty()) {
264              return R[0][0].as<std::string>();
265          }
266      }
267      catch (std::exception& e) {
268          std::cerr << e.what() << std::endl;
269          return "";
270      }
271      return retval;
272  }

```

Рис. 13. Функция, посылающая SQL запрос

Запрос содержит пользовательскую переменную – название документа – которую пользователь вводит с клавиатуры.

```

C# Консоль отладки Microsoft Visual Studio
Opened database successfully: archive
Enter your phone number: +7(812)645-91-72
You are logged in!
Тема документа 'Maintenance instructions for equipment Y' - Technical maintenance

```

Рис. 14. Результат выполнения SQL запроса

- Определять название документа, который имеется в максимальном количестве экземпляров

Рис. 15. Функция, посылающая SQL запрос

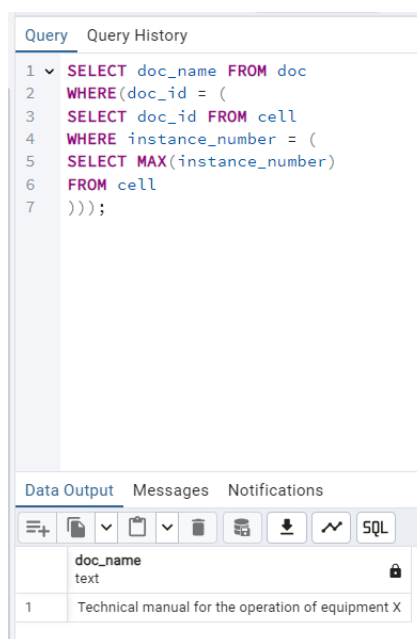


Рис. 16. SQL запрос и его результат

- Определять отдел, работника которого наиболее часто обращаются к архиву:



Рис. 17. SQL запрос и его результат

Данный SQL запрос ищет номер отдела (номер 1), где работает абонент архива (номер 2), который наиболее часто обращался к архиву (Рис. 8).

- Установить ФИО абонента, обращавшегося последним к указанному документу:

```

354
355 std::string find_empl_name_by_last_request(std::string doc_name, pqxx::connection& c) {
356     try {
357         pqxx::work W(c);
358         std::string query = "SELECT employee_name FROM employee\
359 WHERE employee_id = (\
360 SELECT employee_id FROM request\
361 WHERE doc_id = (\
362 SELECT doc_id FROM doc\
363 WHERE doc_name = '" + doc_name + "'\
364 )\
365 ORDER BY request_id DESC\
366 LIMIT 1);";
367         pqxx::result R = W.exec(query);
368         if (!R.empty()) {
369             return R[0][0].as<std::string>();
370         }
371     } catch (std::exception& e) {
372         std::cerr << e.what() << std::endl;
373         return "";
374     }
375     return "";
376 }
377
378
  
```

Рис. 18. Функция, посылающая SQL запрос

Поскольку таблица с запросами является последовательной, можно определить имя последнего обращавшегося к документу абонента через последнюю запись в таблице запросов.

Консоль отладки Microsoft Visual Studio

```

Opened database successfully: archive
Enter your phone number: +7(812)645-91-72
You are logged in!
Последний абонент, обращавшийся к документу 'Report on the testing of equipment A' - Petrov P.P.

```

Рис. 19. Результат выполнения SQL запроса

- Добавление нового документа:

Query Query History

```

1  INSERT INTO doc (doc_name, doc_topic, doc_date)
2  VALUES ('New Document', 'Topic', CURRENT_DATE);

```

Рис. 20. SQL запрос

```

377 }
378 void fill_doc(std::string doc_name, std::string doc_topic, pqxx::connection& C) {
379     try {
380         pqxx::work W(C);
381         std::string query = "INSERT INTO doc (doc_name, doc_topic, doc_date) VALUES ('" + doc_name + "', '" + doc_topic + "', CURRENT_DATE);";
382         W.exec(query);
383         W.commit();
384     }
385     catch (std::exception& e) {
386         std::cerr << e.what() << std::endl;
387     }
388 }
389

```

Рис. 21. Функция, посылающая SQL запрос

- Изменение номера телефона указанного отдела:

Query Query History

```

1  UPDATE department
2  SET department_phone = 'new_phone_number'
3  WHERE department_id = '2';

```

Рис. 22. SQL запрос

```

389 void update_department_phone(int department_id, std::string new_phone, pqxx::connection& C) {
390     try {
391         pqxx::work W(C);
392         std::string query = "UPDATE department SET department_phone = '" + new_phone + "' WHERE department_id = '" + std::to_string(department_id) + "'";
393         W.exec(query);
394         W.commit();
395     }
396     catch (std::exception& e) {
397         std::cerr << e.what() << std::endl;
398     }
399 }
400
401

```

Рис. 23. Функция, посылающая SQL запрос

- Удаление экземпляра некоторого документа:

Query	Query History
1	UPDATE cell SET instance_number = instance_number - 1
2	WHERE doc_id = 7 AND instance_number > 0;
3	

Рис. 24. SQL запрос для doc_id = 7

```

481 void delete_doc_inst(int doc_id, pqxx::connection& C) {
482     try {
483         pqxx::work W(C);
484         std::string query = "UPDATE cell SET instance_number = instance_number - 1 WHERE doc_id = '" + std::to_string(doc_id) + "' AND instance_number > 0;";
485         W.exec(query);
486         W.commit();
487     }
488     catch (std::exception& e) {
489         std::cerr << e.what() << std::endl;
490     }
491 }
492
493
494

```

Рис. 25. Функция, посылающая SQL запрос

- Выдать справку об абонентах архива:

```

438
439 void get_report_1(pqxx::connection& C) {
440     try {
441         pqxx::work W(C);
442         std::string query = "SELECT * FROM employee;";
443         pqxx::result R = W.exec(query);
444         for (auto row : R) {
445             for (auto field : row) {
446                 std::cout << field.c_str() << '\t';
447             }
448             std::cout << '\n';
449         }
450         W.commit();
451     }
452     catch (std::exception& e) {
453         std::cerr << e.what() << std::endl;
454     }
455 }

```

Рис. 26. Функция, посылающая SQL запрос

```

C:\> Консоль отладки Microsoft Visual Studio

Opened database successfully: archive
Enter your phone number: +7(812)783-83-16
You are logged in!
1      1      Ivanov I.I.      +7(812)783-32-33
2      1      Petrov P.P.      +7(812)645-91-72
3      2      Sidorov S.S.      +7(812)783-83-16

```

Рис. 27. Справка об абонентах архива

- Выдать справку о работе архива (число единиц хранения, названия документов, поступивших в архив за последний месяц, количество экземпляров каждого документа, место его хранения):

```

457 void get_report_2(pqxx::connection& C) {
458     try {
459         pqxx::work W(C);
460         std::string query = "SELECT COUNT(*) cell_id FROM cell;";
461         pqxx::result R = W.exec(query);
462         if (!R.empty()) {
463             std::cout << "Число единиц хранения - " << R[0][0].as<int>() << std::endl;
464         }
465         query = "\
466 SELECT cell_id, instance_number, doc_name FROM\
467 (\
468 SELECT cell_id, cell.doc_id, instance_number, doc_name\
469 FROM cell JOIN doc ON cell.doc_id = doc.doc_id\
470 )\
471 WHERE doc_id IN\
472 (\
473 SELECT doc_id FROM doc WHERE doc_date >= doc_date - 31\
474 )\
475 ORDER BY cell_id;";
476 R = W.exec(query);
477 std::cout << "Ячейка\tКол-во\tНазвание документа\n";
478 for (auto row : R) {
479     for (auto field : row) {
480         std::cout << field.c_str() << '\t';
481     }
482     std::cout << '\n';
483 }
484 W.commit();
485 }
486 catch (std::exception& e) {
487     std::cerr << e.what() << std::endl;
488 }
489 }

```

Рис. 28. Функция, посылающая SQL запрос

Консоль отладки Microsoft Visual Studio

```

Opened database successfully: archive
Enter your phone number: +7(812)783-83-16
You are logged in!
Число единиц хранения - 10
Ячейка  Кол-во  Название документа
1        3      Regulations for the preventive maintenance of equipment Z
2        4      Technical manual for the operation of equipment X
3        1      User guide for software D
4        0      Safety recommendations for equipment Y
5        2      Maintenance instructions for equipment Y
6        0      Production safety manual
7        1      Report on the testing of equipment A
9        0      Modernization plan for equipment C
10       2      Technical requirements for system B

```

Рис. 29. Справка о работе архива

Аварийные ситуации

В случае несоблюдения условий выполнения запроса на сервер, программа пропускает запрос, не подвергая базу данных ненужным изменениям. Пользователь должен повторить попытку с корректными значениями.

Приложение Б (Программный код на C++)

```
bool login(pqxx::connection& C, int& id) {
    std::string phone_number = "";
    std::cout << "Enter your phone number: ";
    std::cin >> phone_number;

    try {

        pqxx::work W(C);
        std::string query = "SELECT COUNT(*) FROM employee WHERE employee_phone = " + phone_number + " ";
        pqxx::result R = W.exec(query);
        if (!R.empty()) {
            if (R[0][0].as<int>() != 0) {
                query = "SELECT employee_id FROM employee WHERE employee_phone = " + phone_number + " ";
                pqxx::result R = W.exec(query);
                id = R[0][0].as<int>();
                std::cout << "You are logged in!\n";
                return true;
            }
        }
        std::cout << "Wrong phone number!\n";
    }
    catch (std::exception& e) {
        std::cerr << e.what() << std::endl;
        return 0;
    }
    return false;
}

void fill_doc(pqxx::connection& C) {
    try {
        pqxx::work W(C);
        std::string query = "INSERT INTO doc (doc_name, doc_topic, doc_date) VALUES ('Technical manual for the operation of equipment X',
'Equipment operation', CURRENT_DATE);";
        W.exec(query);
        query = "INSERT INTO doc (doc_name, doc_topic, doc_date) VALUES ('Maintenance instructions for equipment Y', 'Technical
maintenance', CURRENT_DATE);";
        W.exec(query);
        query = "INSERT INTO doc (doc_name, doc_topic, doc_date) VALUES ('Regulations for the preventive maintenance of equipment Z',
'Technical maintenance', CURRENT_DATE);";
        W.exec(query);
        query = "INSERT INTO doc (doc_name, doc_topic, doc_date) VALUES ('Production safety manual', 'Safety', CURRENT_DATE);";
        W.exec(query);
        query = "INSERT INTO doc (doc_name, doc_topic, doc_date) VALUES ('Calibration methodology for measuring instruments', 'Calibration',
CURRENT_DATE);";
        W.exec(query);
        query = "INSERT INTO doc (doc_name, doc_topic, doc_date) VALUES ('Report on the testing of equipment A', 'Testing',
CURRENT_DATE);";
        W.exec(query);
        query = "INSERT INTO doc (doc_name, doc_topic, doc_date) VALUES ('Technical requirements for system B', 'Technical requirements',
CURRENT_DATE);";
        W.exec(query);
        query = "INSERT INTO doc (doc_name, doc_topic, doc_date) VALUES ('Modernization plan for equipment C', 'Modernization',
CURRENT_DATE);";
        W.exec(query);
        query = "INSERT INTO doc (doc_name, doc_topic, doc_date) VALUES ('User guide for software D', 'Software', CURRENT_DATE);";
        W.exec(query);
        query = "INSERT INTO doc (doc_name, doc_topic, doc_date) VALUES ('Safety recommendations for equipment Y', 'Safety',
CURRENT_DATE);";
        W.exec(query);
        W.commit();
    }
    catch (std::exception& e) {
        std::cerr << e.what() << std::endl;
    }
}

void fill Rack(pqxx::connection& C) {
```

```

        pqxx::work W(C);
        std::string query = "INSERT INTO rack DEFAULT VALUES;";
        W.exec(query);
        W.exec(query);
        W.commit();
    }
    catch (std::exception& e) {
        std::cerr << e.what() << std::endl;
    }
}

void fill_shelf(pqxx::connection& C) {
    try {
        pqxx::work W(C);
        std::string query = "INSERT INTO shelf (rack_id) VALUES (1);";
        W.exec(query);
        query = "INSERT INTO shelf (rack_id) VALUES (1);";
        W.exec(query);
        query = "INSERT INTO shelf (rack_id) VALUES (2);";
        W.exec(query);
        W.commit();
    }
    catch (std::exception& e) {
        std::cerr << e.what() << std::endl;
    }
}

void fill_instance(pqxx::connection& C) {
    try {
        pqxx::work W(C);
        std::string query = "INSERT INTO instance (doc_id) VALUES (3);";
        W.exec(query);
        query = "INSERT INTO instance (doc_id) VALUES (7);";
        W.exec(query);
        query = "INSERT INTO instance (doc_id) VALUES (1);";
        W.exec(query);
        query = "INSERT INTO instance (doc_id) VALUES (2);";
        W.exec(query);
        query = "INSERT INTO instance (doc_id) VALUES (1);";
        W.exec(query);
        query = "INSERT INTO instance (doc_id) VALUES (5);";
        W.exec(query);
        query = "INSERT INTO instance (doc_id) VALUES (9);";
        W.exec(query);
        query = "INSERT INTO instance (doc_id) VALUES (3);";
        W.exec(query);
        query = "INSERT INTO instance (doc_id) VALUES (6);";
        W.exec(query);
        query = "INSERT INTO instance (doc_id) VALUES (7);";
        W.exec(query);
        query = "INSERT INTO instance (doc_id) VALUES (1);";
        W.exec(query);
        query = "INSERT INTO instance (doc_id) VALUES (3);";
        W.exec(query);
        query = "INSERT INTO instance (doc_id) VALUES (5);";
        W.exec(query);
        query = "INSERT INTO instance (doc_id) VALUES (2);";
        W.exec(query);
        query = "INSERT INTO instance (doc_id) VALUES (1);";
        W.exec(query);
        W.commit();
    }
    catch (std::exception& e) {
        std::cerr << e.what() << std::endl;
    }
}

void fill_cell(pqxx::connection& C) {
    try {
        pqxx::work W(C);

```

```

std::string query = "INSERT INTO cell (shelf_id, doc_id) VALUES (1, 3);";
W.exec(query);
query = "UPDATE cell SET instance_number = (SELECT COUNT(*) FROM instance WHERE instance.doc_id = cell.doc_id);";
W.exec(query);
query = "INSERT INTO cell (shelf_id, doc_id) VALUES (1, 1);";
W.exec(query);
query = "UPDATE cell SET instance_number = (SELECT COUNT(*) FROM instance WHERE instance.doc_id = cell.doc_id);";
W.exec(query);
query = "INSERT INTO cell (shelf_id, doc_id) VALUES (1, 9);";
W.exec(query);
query = "UPDATE cell SET instance_number = (SELECT COUNT(*) FROM instance WHERE instance.doc_id = cell.doc_id);";
W.exec(query);
query = "INSERT INTO cell (shelf_id, doc_id) VALUES (1, 10);";
W.exec(query);
query = "UPDATE cell SET instance_number = (SELECT COUNT(*) FROM instance WHERE instance.doc_id = cell.doc_id);";
W.exec(query);
query = "INSERT INTO cell (shelf_id, doc_id) VALUES (2, 2);";
W.exec(query);
query = "UPDATE cell SET instance_number = (SELECT COUNT(*) FROM instance WHERE instance.doc_id = cell.doc_id);";
W.exec(query);
query = "INSERT INTO cell (shelf_id, doc_id) VALUES (2, 4);";
W.exec(query);
query = "UPDATE cell SET instance_number = (SELECT COUNT(*) FROM instance WHERE instance.doc_id = cell.doc_id);";
W.exec(query);
query = "INSERT INTO cell (shelf_id, doc_id) VALUES (2, 6);";
W.exec(query);
query = "UPDATE cell SET instance_number = (SELECT COUNT(*) FROM instance WHERE instance.doc_id = cell.doc_id);";
W.exec(query);
query = "INSERT INTO cell (shelf_id, doc_id) VALUES (3, NULL);";
W.exec(query);
query = "UPDATE cell SET instance_number = (SELECT COUNT(*) FROM instance WHERE instance.doc_id = cell.doc_id);";
W.exec(query);
query = "INSERT INTO cell (shelf_id, doc_id) VALUES (3, 8);";
W.exec(query);
query = "UPDATE cell SET instance_number = (SELECT COUNT(*) FROM instance WHERE instance.doc_id = cell.doc_id);";
W.exec(query);
query = "INSERT INTO cell (shelf_id, doc_id) VALUES (3, 7);";
W.exec(query);
query = "UPDATE cell SET instance_number = (SELECT COUNT(*) FROM instance WHERE instance.doc_id = cell.doc_id);";
W.exec(query);
W.commit();
}
catch (std::exception& e) {
    std::cerr << e.what() << std::endl;
}
}

void fill_department(pqxx::connection& C) {
    try {
        pqxx::work W(C);
        std::string query = "INSERT INTO department (department_phone) VALUES ('+7(812)923-97-24');";
        W.exec(query);
        query = "INSERT INTO department (department_phone) VALUES ('+7(812)807-65-41');";
        W.exec(query);
        W.commit();
    }
    catch (std::exception& e) {
        std::cerr << e.what() << std::endl;
    }
}

void fill_employee(pqxx::connection& C) {
    try {
        pqxx::work W(C);
        std::string query = "INSERT INTO employee (department_id, employee_name, employee_phone) VALUES ('1', 'Ivanov I.I.', '+7(812)783-32-33');";
        W.exec(query);
        query = "INSERT INTO employee (department_id, employee_name, employee_phone) VALUES ('1', 'Petrov P.P.', '+7(812)645-91-72');";
        W.exec(query);
        query = "INSERT INTO employee (department_id, employee_name, employee_phone) VALUES ('2', 'Sidorov S.S.', '+7(812)783-83-16');";

```

```

        W.exec(query);
        W.commit();
    }
    catch (std::exception& e) {
        std::cerr << e.what() << std::endl;
    }
}

void select_all(std::string tb_name, pqxx::connection& C) {
    try {
        pqxx::work W(C);
        std::string query = "SELECT * FROM " + tb_name + ";";
        pqxx::result R = W.exec(query);
        for (auto row : R) {
            for (auto field : row) {
                std::cout << field.c_str() << '\t';
            }
            std::cout << '\n';
        }
        W.commit();
    }
    catch (std::exception& e) {
        std::cerr << e.what() << std::endl;
    }
}

void truncate_cascade(std::string tb_name, pqxx::connection& C) {
    try {
        pqxx::work W(C);
        std::string query = "TRUNCATE " + tb_name + " CASCADE;";
        pqxx::result R = W.exec(query);
        W.commit();
    }
    catch (std::exception& e) {
        std::cerr << e.what() << std::endl;
    }
}

int count_does_by_topic(std::string topic, pqxx::connection& C) {
    int retval = 0;
    try {
        pqxx::work W(C);
        std::string query = "SELECT COUNT(*) FROM doc WHERE doc_topic = '" + topic + "';";
        pqxx::result R = W.exec(query);
        if (!R.empty()) {
            return R[0][0].as<int>();
        }
    }
    catch (std::exception& e) {
        std::cerr << e.what() << std::endl;
        return 0;
    }
    return retval;
}

std::string find_topic_by_doc_name(std::string name, pqxx::connection& C) {
    std::string retval = "";
    try {
        pqxx::work W(C);
        std::string query = "SELECT doc_topic FROM doc WHERE doc_name = '" + name + "';";
        pqxx::result R = W.exec(query);
        if (!R.empty()) {
            return R[0][0].as<std::string>();
        }
    }
    catch (std::exception& e) {
        std::cerr << e.what() << std::endl;
        return "";
    }
}

```

```

        return retval;
    }

std::string find_doc_name_with_largest_inst_num(pqxx::connection& C) {
    std::string retval = "";
    try {
        pqxx::work W(C);
        std::string query = "SELECT doc_name FROM doc\
                               WHERE(doc_id = (\
                                   SELECT doc_id FROM cell\
                                   WHERE instance_number = (\
                                       SELECT MAX(instance_number)\
                                       FROM cell\
                                   ))";

        pqxx::result R = W.exec(query);
        if (!R.empty()) {
            return R[0][0].as<std::string>();
        }
    }
    catch (std::exception& e) {
        std::cerr << e.what() << std::endl;
        return "";
    }
    return retval;
}

std::string find_doc_name_most_requested(pqxx::connection& C) {
    try {
        pqxx::work W(C);
        std::string query = "SELECT doc_name FROM doc\
                               WHERE doc_id =\
                               (SELECT doc_id FROM\
                               (SELECT doc_id, COUNT(doc_id)\
                               FROM request\
                               GROUP BY doc_id\
                               ORDER BY doc_id)\
                               WHERE count =\
                               (SELECT MAX(count) FROM\
                               (SELECT doc_id, COUNT(doc_id)\
                               FROM request\
                               GROUP BY doc_id\
                               ORDER BY doc_id)\
                               )\
                               LIMIT 1);";

        pqxx::result R = W.exec(query);
        if (!R.empty()) {
            return R[0][0].as<std::string>();
        }
    }
    catch (std::exception& e) {
        std::cerr << e.what() << std::endl;
        return "";
    }
    return "";
}

std::string find_department_most_requested(pqxx::connection& C) {
    try {
        pqxx::work W(C);
        std::string query = "SELECT department_id FROM employee\
                               WHERE employee_id = (\
                                   SELECT employee_id FROM(\
                                       SELECT employee_id, COUNT(employee_id)\
                                       FROM request\
                                       GROUP BY employee_id\
                                       ORDER BY employee_id)\
                                   WHERE count = (\
                                       SELECT MAX(count) FROM(\
                                           SELECT employee_id, COUNT(employee_id)\
                                           FROM request\
                                       )\
                                   )";
    }

```

```

        GROUP BY employee_id\
        ORDER BY employee_id));";
pqxx::result R = W.exec(query);
if (!R.empty()) {
    return R[0][0].as<std::string>();
}
}
catch (std::exception& e) {
    std::cerr << e.what() << std::endl;
    return "";
}
return "";
}

std::string find_empl_name_by_last_request(std::string doc_name, pqxx::connection& C) {
    try {
        pqxx::work W(C);
        std::string query = "SELECT employee_name FROM employee\
        WHERE employee_id = (\
        SELECT employee_id FROM request\
        WHERE doc_id = (\
        SELECT doc_id FROM doc\
        WHERE doc_name = " + doc_name + "\
        )\
        ORDER BY request_id DESC\
        LIMIT 1);";
        pqxx::result R = W.exec(query);
        if (!R.empty()) {
            return R[0][0].as<std::string>();
        }
    }
    catch (std::exception& e) {
        std::cerr << e.what() << std::endl;
        return "";
    }
    return "";
}

void fill_doc(std::string doc_name, std::string doc_topic, pqxx::connection& C) {
    try {
        pqxx::work W(C);
        std::string query = "INSERT INTO doc (doc_name, doc_topic, doc_date) VALUES (" + doc_name + ", " + doc_topic + ", " +
CURRENT_DATE);";
        W.exec(query);
        W.commit();
    }
    catch (std::exception& e) {
        std::cerr << e.what() << std::endl;
    }
}

void update_department_phone(int department_id, std::string new_phone, pqxx::connection& C) {
    try {
        pqxx::work W(C);
        std::string query = "UPDATE department SET department_phone = " + new_phone + " WHERE department_id = " +
std::to_string(department_id) + ";";
        W.exec(query);
        W.commit();
    }
    catch (std::exception& e) {
        std::cerr << e.what() << std::endl;
    }
}

void delete_doc_inst(int doc_id, pqxx::connection& C) {
    try {
        pqxx::work W(C);
        std::string query = "UPDATE cell SET instance_number = instance_number - 1 WHERE doc_id = " + std::to_string(doc_id) + " AND
instance_number > 0;";
        W.exec(query);
        W.commit();
    }
}

```



```

    }
    catch (std::exception& e) {
        std::cerr << e.what() << std::endl;
    }
}

std::string doRequest(pqxx::connection& C, int& empl_id) {
    std::string retval = "";
    int doc_id = 0;
    std::cout << "Enter document ID (Request): ";
    std::cin >> doc_id;
    try {
        pqxx::work W(C);
        std::string query = "SELECT doc_name FROM doc WHERE doc_id = " + std::to_string(doc_id) + ",";
        pqxx::result R = W.exec(query);
        if (!R.empty()) {
            retval = R[0][0].as<std::string>();
            query = "INSERT INTO request (doc_id, employee_id, request_date) VALUES (" + std::to_string(doc_id) + ", " +
std::to_string(empl_id) + ", CURRENT_DATE);";
            W.exec(query);
            W.commit();
            return retval;
        }
    }
    catch (std::exception& e) {
        std::cerr << e.what() << std::endl;
        return "";
    }
    return retval;
}

void get_report_1(pqxx::connection& C) {
    try {
        pqxx::work W(C);
        std::string query = "SELECT * FROM employee;";
        pqxx::result R = W.exec(query);
        for (auto row : R) {
            for (auto field : row) {
                std::cout << field.c_str() << "t";
            }
            std::cout << "n";
        }
        W.commit();
    }
    catch (std::exception& e) {
        std::cerr << e.what() << std::endl;
    }
}

void get_report_2(pqxx::connection& C) {
    try {
        pqxx::work W(C);
        std::string query = "SELECT COUNT(*) cell_id FROM cell;";
        pqxx::result R = W.exec(query);
        if (!R.empty()) {
            std::cout << "Число единиц хранения - " << R[0][0].as<int>() << std::endl;
        }
        query = "\
SELECT cell_id, instance_number, doc_name FROM\
(\
SELECT cell_id, cell.doc_id, instance_number, doc_name\
FROM cell JOIN doc ON cell.doc_id = doc.doc_id\
)\
WHERE doc_id IN\
(\
SELECT doc_id FROM doc WHERE doc_date >= doc_date - 31\
)\
ORDER BY cell_id;";
        R = W.exec(query);
    }
}

```

```

std::cout << "Ячейка\tКол-во\tНазвание документа\n";
for (auto row : R) {
    for (auto field : row) {
        std::cout << field.c_str() << "\t";
    }
    std::cout << "\n";
}
W.commit();
}
catch (std::exception& e) {
    std::cerr << e.what() << std::endl;
}
}

```