

Работа с PostgreSQL из C++

Владислав Хорев
Ведущий программист, Andersen



Проверка связи



Поставьте “+”, если меня видно и слышно



Если у вас нет звука:

- убедитесь, что на вашем устройстве и на колонках включен звук
- обновите страницу вебинара (или закройте страницу и заново присоединитесь к вебинару)
- откройте вебинар в другом браузере
- перезагрузите компьютер (ноутбук) и заново попытайтесь зайти

Владислав Хорев

О спикере:

- Ведущий программист в компании Andersen
- Работает в IT с 2011 года
- Опыт разработки на C++ более 11 лет



Вспоминаем прошное занятие

Вопрос: Что делает оператор GROUP BY?



Вспоминаем прошрое занятие

Вопрос: Что делает оператор GROUP BY?

Ответ: Оператор GROUP BY задает агрегацию по нужным столбцам.



Вспоминаем прошрое занятие

Вопрос: Что делает оператор JOIN?



Вспоминаем прошрое занятие

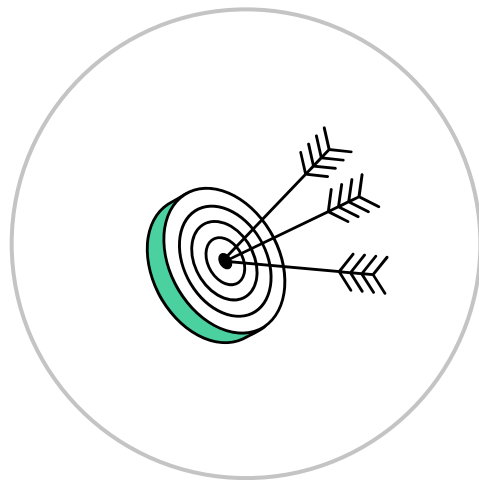
Вопрос: Что делает оператор JOIN?

Ответ: Оператор JOIN объединяет таблицы по определённомu столбцу или связке столбцов, как правило, по первичному ключу.



Цели занятия

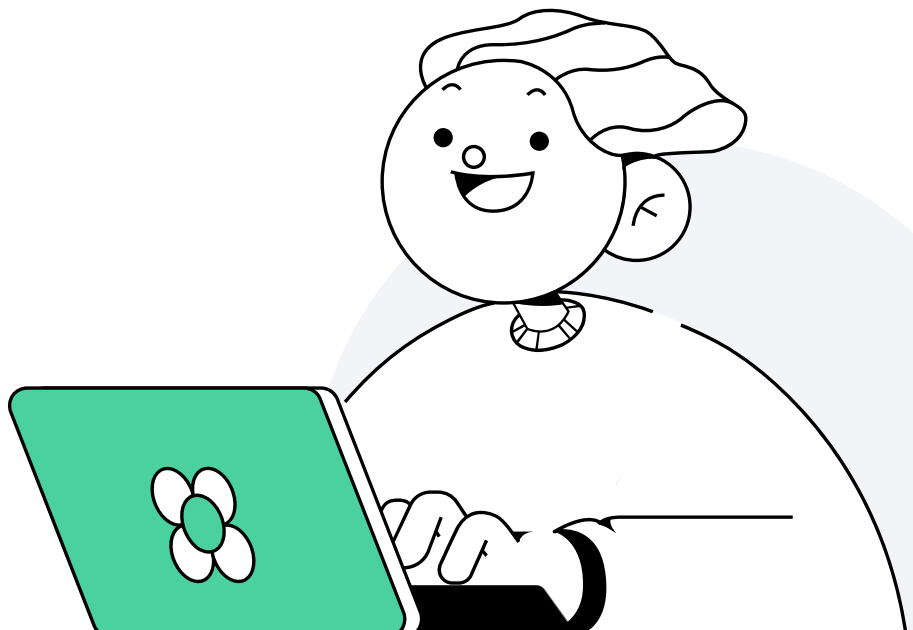
- Разберёмся как подключаться к PostgreSQL из C++
- Узнаем, как выполнять SQL-запросы из C++
- Выясним, как защитить себя от атаки SQL Injection



План занятия

- 1 Установка и подключение libpq++
- 2 Select-запросы
- 3 Insert- и Update-запросы
- 4 Защита от атаки SQL Injection
- 5 Домашнее задание

*Нажми на нужный раздел для перехода



Установка и подключение libpq++



1

PostgreSQL

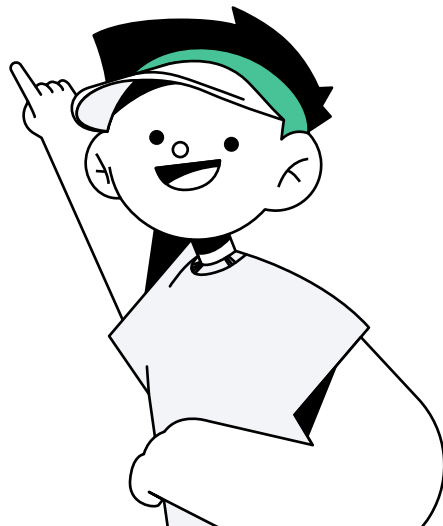
PostgreSQL имеет драйвера и интерфейсы для многих языков программирования, включая C++.

PostgreSQL

PostgreSQL имеет драйвера и интерфейсы для многих языков программирования, включая C++.

Существует несколько реализаций API для C++.

Одна из самых популярных - это **библиотека libpq++**, также известная как libpqxx.





libpq++ - это официальный API для C++ клиента PostgreSQL



libpq++ - это официальный API для C++ клиента PostgreSQL

Эта библиотека - с открытым исходным кодом под лицензией BSD.
Исходный код libpq++ доступен на Github.



libpq++ - это официальный API для C++ клиента PostgreSQL

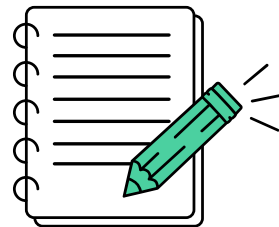
Эта библиотека - с открытым исходным кодом под лицензией BSD.
Исходный код libpq++ доступен на Github.

libpq++ собирается с помощью **CMake**, следовательно, его можно легко добавить в свой проект.

Подключение librq++ к вашему CMake-проекту

1

Скачайте свежий релиз librq++ с Github.com и распакуйте его в отдельный каталог



Подключение librq++ к вашему CMake-проекту

- 1 Скачайте свежий релиз librq++ с Github.com и распакуйте его в отдельный каталог
- 2 Добавьте путь к этому каталогу в ваш файл CMakeLists.txt



Подключение librq++ к вашему CMake-проекту

- 1 Скачайте свежий релиз librq++ с Github.com и распакуйте его в отдельный каталог
- 2 Добавьте путь к этому каталогу в ваш файл CMakeLists.txt
- 3 Включите поддержку C++17 в своем проекте

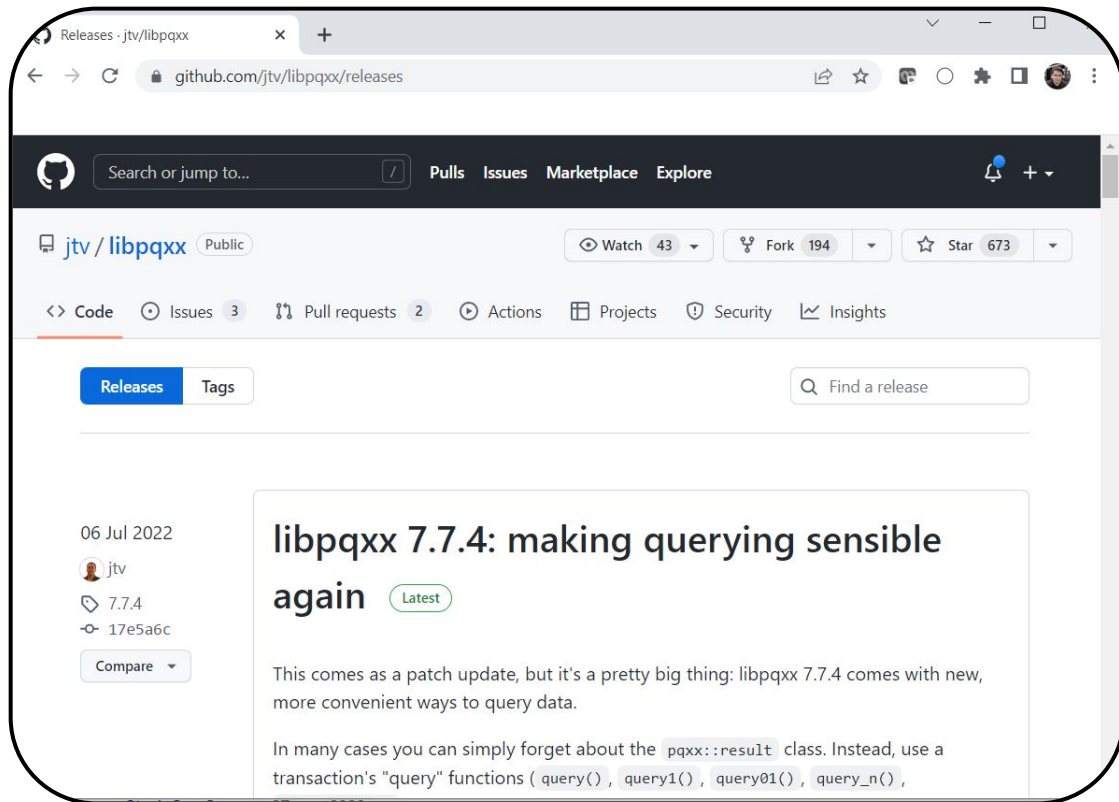


Подключение librq++ к вашему CMake-проекту

- 1 Скачайте свежий релиз librq++ с Github.com и распакуйте его в отдельный каталог
- 2 Добавьте путь к этому каталогу в ваш файл CMakeLists.txt
- 3 Включите поддержку C++17 в своем проекте
- 4 Слинкуйте ваш проект с librq++



Рекомендуется скачать самый последний релиз



Подключение libpq++

Ваш файл CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.20)
project(MyProject1)
add_executable(MyProject1 main.cpp)

add_subdirectory(C:/path/to/libpqxx libpqxx-build)      # Укажите путь к libpq++

target_compile_features(MyProject1 PRIVATE cxx_std_17)  # Включите C++17

target_link_libraries(MyProject1 pqxx)                  # Слинкуйте libpq++ к проекту
```

Подключение libpq++

Для работы с libpq++ подключите заголовочные файлы:

```
#include <pqxx/pqxx>

int main()
{
    return 0;
}
```

Подключение libpq++

Создайте объект соединения `pqxx::connection` и укажите данные для подключения к базе данных PostgreSQL:

```
#include <pqxx/pqxx>

int main()
{
    pqxx::connection c(
        "host=localhost "
        "port=5432 "
        "dbname=my_database "
        "user=my_database_user "
        "password=my_password_123");

    return 0;
}
```

Подключение libpq++

Совет: для упрощения обработки ошибок, оборачивайте вызовы к libpq++ в блок try, а в блоке catch ловите исключение pqxx::sql_error. Пример:

```
try
{
    pqxx::connection c(
        "host=localhost "
        "port=5432 "
        "dbname=my_database "
        "user=my_database_user "
        "password=my_password_123");

    // ...

} catch (pqxx::sql_error e)
{
    std::cout << e.what() << std::endl;
}
```


Select-запросы



2

Select-запросы

Предположим, мы имеем в базе данных таблицу с книгами **book** следующего формата:

id	title	author
1	Война и мир	Лев Толстой
2	Герой нашего времени	Михаил Лермонтов
3	Мастер и Маргарита	Михаил Булгаков

Select-запросы

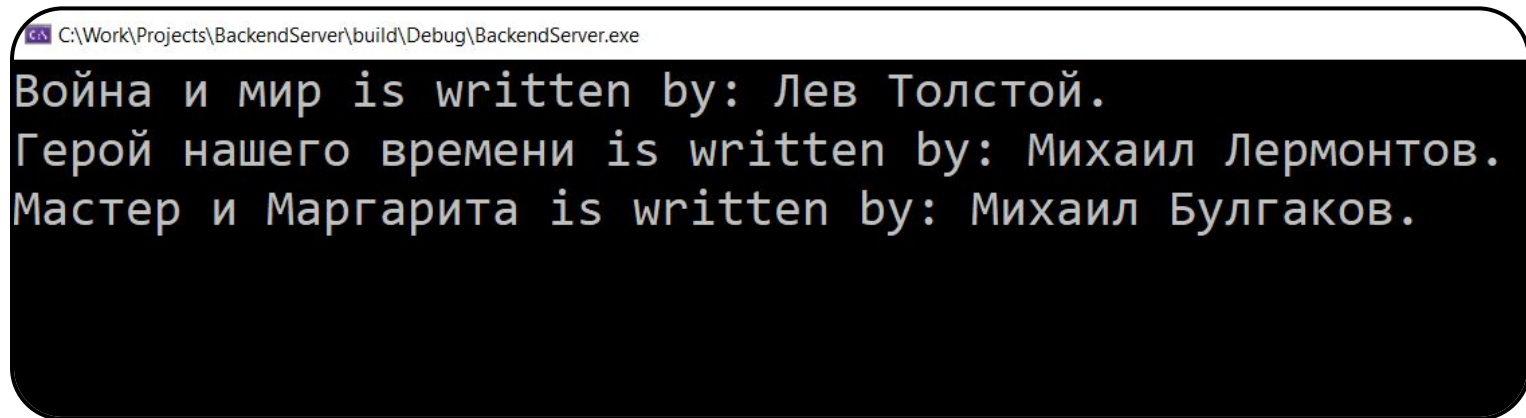
Для того, чтобы получить данные из этой таблицы, создайте объект **work** и выполните над ним запрос **query()**:

```
pqxx::work tx{ c };

for (auto [title, author] : tx.query<std::string, std::string>(
    "SELECT title, author FROM book"))
{
    std::cout << title << " is written by: " << author << ".\n";
}
```

Select-запросы

Вывод на экран:

A screenshot of a Windows command prompt window. The title bar at the top shows the file path "C:\Work\Projects\BackendServer\build\Debug\BackendServer.exe". The command prompt area has a black background with white text. It displays three lines of output, each on a new line: "Война и мир is written by: Лев Толстой.", "Герой нашего времени is written by: Михаил Лермонтов.", and "Мастер и Маргарита is written by: Михаил Булгаков.".

```
C:\Work\Projects\BackendServer\build\Debug\BackendServer.exe
Война и мир is written by: Лев Толстой.
Герой нашего времени is written by: Михаил Лермонтов.
Мастер и Маргарита is written by: Михаил Булгаков.
```

Select-запросы

Вы можете делать SELECT-запросы любой сложности:

```
pqxx::work tx{ c };  
  
for (auto [title, author] : tx.query<std::string, std::string>(  
    "SELECT id, author, title FROM book ORDER BY title LIMIT 2"))  
{  
    std::cout << title << " is written by: " << author << ".\n";  
}
```

Select-запросы

Если вам требуется только одна запись, вы можете использовать метод **query_value()**:

```
pqxx::work tx{ c };

std::string author = tx.query_value<std::string>("SELECT author FROM book WHERE id = 3");

std::cout << "The author you wanted is: " << author << ".\n";
```

Insert- и Update-запросы



3

Insert- и Update-запросы

Для того, чтобы добавить запись в таблицу или обновить имеющиеся записи, необходимо:

- 1 Создать объект **work**.
- 2 Вызвать метод **exec()**, указав в качестве аргумента запрос INSERT или UPDATE.
- 3 Завершить транзакцию, вызвав метод **commit()**.

Insert- и Update-запросы

Пример выполнения INSERT-запроса:

```
pqxx::work tx{ c };  
  
tx.exec("INSERT INTO book(title, author) "  
        "VALUES('Братья Карамазовы', 'Федор Достоевский')");  
  
tx.commit();
```

Insert- и Update-запросы

Пример выполнения UPDATE-запроса:

```
pqxx::work tx{ c };  
  
tx.exec("UPDATE book SET author='Фёдор' where author='Федор');  
  
tx.commit();
```

Защита от атаки SQL Injection



4

Что такое SQL Injection

Предположим вы захотите разрешить пользователю добавить свою книгу на сайт. Вы можете попросить пользователя ввести имя книги с клавиатуры, например так:

```
std::string newTitle, newAuthor;
std::cout << "Введите название книги и автора: "<< std::endl;
std::cin >> newTitle >> newAuthor;

pqxx::work tx{ c };

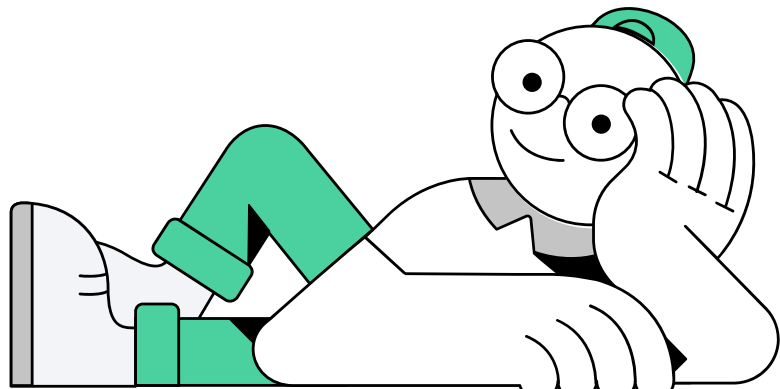
tx.exec("INSERT INTO book(title, author) "
        "VALUES('" + newTitle + "', '" + newAuthor + "')");

tx.commit();
```

Что такое SQL Injection

Если пользователь просто введет имя автора книги - то все отлично.

Но что если пользователь захочет выйти за пределы кавычек и попытается изменить сам SQL-запрос?



Что такое SQL Injection

В SQL символ комментирования - два дефиса "--".

Предположим пользователь введет в поле author следующую строку:

```
pwned'); drop database my_database; --
```

Что такое SQL Injection

В SQL символ комментирования - два дефиса “--”.

Предположим пользователь введет в поле author следующую строку:

```
pwned'); drop database my_database; --
```

Тогда итоговый запрос превратится в два запроса:

```
INSERT INTO book(title, author) VALUES('title', 'pwned'); drop database my_database; --')
```

Что такое SQL Injection

В SQL символ комментирования - два дефиса "--".

Предположим пользователь введет в поле author следующую строку:

```
pwned'); drop database my_database; --
```

Тогда итоговый запрос превратится в два запроса:

```
INSERT INTO book(title, author) VALUES('title', 'pwned'); drop database my_database; --')
```

- 1 Первый запрос добавит строку в вашу таблицу.

Что такое SQL Injection

В SQL символ комментирования - два дефиса “--”.

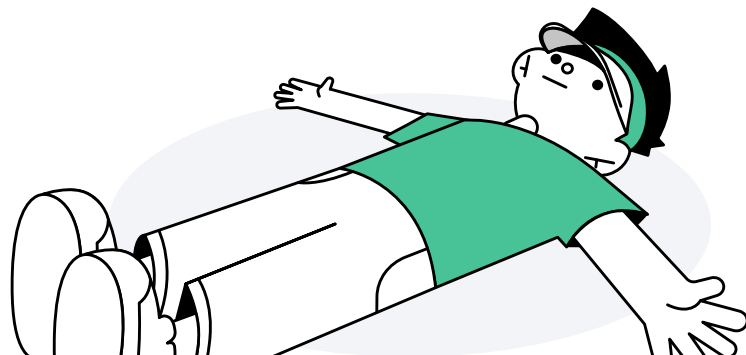
Предположим пользователь введет в поле author следующую строку:

```
pwned'); drop database my_database; --
```

Тогда итоговый запрос превратится в два запроса:

```
INSERT INTO book(title, author) VALUES('title', 'pwned'); drop database my_database; --')
```

- 1 Первый запрос добавит строку в вашу таблицу.
- 2 Второй запрос удалит вашу базу данных.



Два способа избежать SQL Injection

- 1 Экранирование данных
- 2 Prepared statement



Экранирование данных — замена в тексте управляющих символов на соответствующие текстовые подстановки.

Для этого libpq++ предоставляет специальный метод: **esc()**

Экранирование данных. Пример

```
std::string newTitle, newAuthor;
std::cout << "Введите название книги и автора: "<< std::endl;
std::cin >> newTitle >> newAuthor;

pqxx::work tx{ c };

tx.exec("INSERT INTO book(title, author) "
        "VALUES('" + tx.esc(newTitle) + "', '" + tx.esc(newAuthor) + "')");

tx.commit();
```



Другой способ - это подготовить шаблон запроса SQL, чтобы потом просто подставить в него нужные данные.

Такие шаблоны называются — Prepared statement

Prepared statement

Для того, чтобы подготовить шаблон, необходимо вызвать метод `prepare()` у объекта `connection`.

Это нужно сделать 1 раз при подключении к базе данных.

В дальнейшем вы можете вызывать prepared statement по имени сколько угодно раз, подставляя необходимые параметры.

Prepared statement

Пример создания prepared statement:

```
pqxx::connection c(  
    "host=localhost "  
    "port=5432 "  
    "dbname=my_database "  
    "user=my_database_user "  
    "password=my_password_123");  
  
c.prepare("insert_book", "INSERT INTO book(title, author) VALUES($1, $2)");
```

Prepared statement

Пример вызова prepared statement:

```
std::string newTitle, newAuthor;  
std::cout << "Введите название книги и автора: "<< std::endl;  
std::cin >> newTitle >> newAuthor;  
  
pqxx::work tx{ c };  
  
tx.exec_prepared("insert_book", newTitle, newAuthor);  
  
tx.commit();
```


Итоги



Итоги занятия

Сегодня мы

- 1 Разобрались, как подключаться к PostgreSQL из C++
- 2 Узнали, как выполнять SQL-запросы из C++
- 3 Выяснили, как защитить себя от атаки SQL Injection



Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).

- 1 Вопросы по домашней работе задавайте в чате группы
- 2 Задачи можно сдавать по частям
- 3 Зачёт по домашней работе ставят после того, как приняты все задачи



Дополнительные материалы

- [Документация по libpq++](#)



Задавайте вопросы и пишите отзыв о лекции

Владислав Хорев
Ведущий программист, Andersen

