

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

**Лабораторна робота №2**

з дисципліни

**«Бази даних і засоби управління»**

Тема: «**Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL**»

Виконав: студент 3 курсу

Група: КВ-03

Дунець А.М

Перевірив: Петрашенко А.В.

Київ – 2023

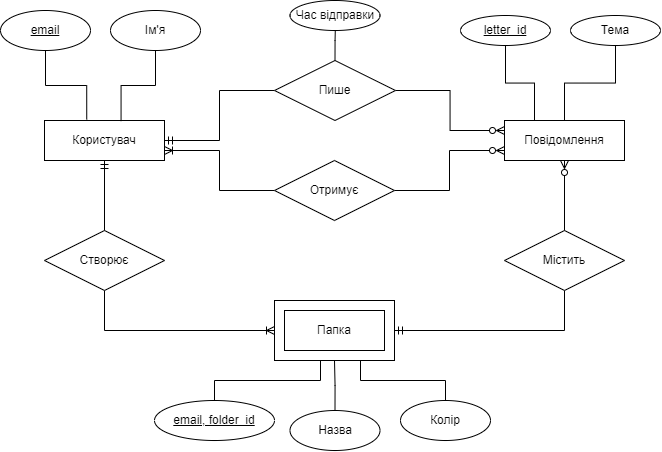
*Метою роботи* є здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

*Загальне завдання* роботи полягає у наступному:

1. Реалізувати функції перегляду, внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу.
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.
3. Забезпечити реалізацію пошуку за декількома атрибутами з двох та більше сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових – як шаблон функції LIKE оператора SELECT SQL, для логічного типу – значення True/False, для дат – у рамках діапазону дат.
4. Програмний код виконати згідно шаблону MVC (модель-подання-контролер).

Репозиторій github: <https://github.com/dynetskov/dataBases.git>

**Діаграма “сутність-звʼязок” для предметної галузі “Електронна пошта”(нотація Crow’s foot)**



*Опис моделі:*

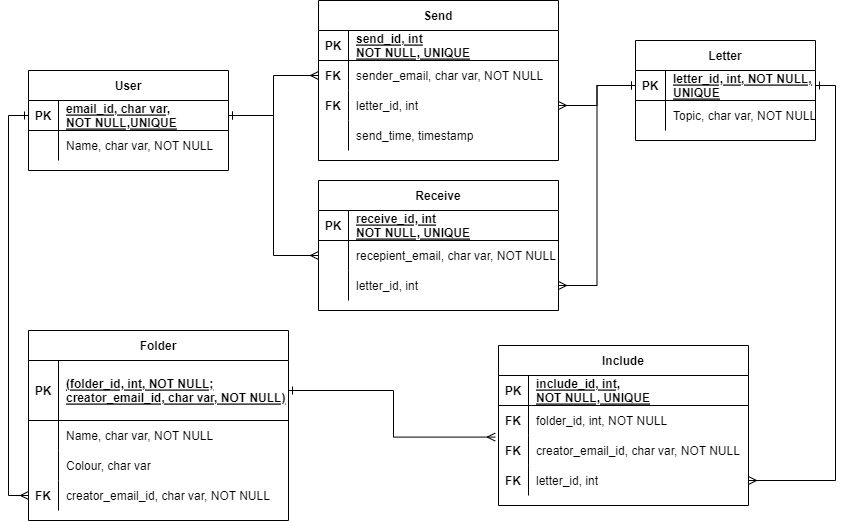
Електронна пошта має три сутності:Користувач, Повідомлення, Папка. Сутність Папка не може існувати без сутності користувач, тому вона слабка.

Сутність **Користувач** містить такі атрибути: email, Ім’я. Вона має з сутністю **Повідомлення** два зв’язки. Перший з них це зв’язок **Пише**, тип якого це 1:N(N >= 0), цей зв'язок має свій власний атрибут: час надсилання.

Другий зв’язок **Отримує** має тип M:N, де M > 0, N >= 0. Також **Користувач** має зв’язок **Створює** типу 1:N(N > 0) з сутністю **Папка**

Сутність **Папка** має такі 3 атрибути: складений ключовий атрибут (email, folder\_id), Назва, Колір. **Папка** має зв’язок **Містить** типу 1:N(N >= 0) з сутністю **Повідомлення**.

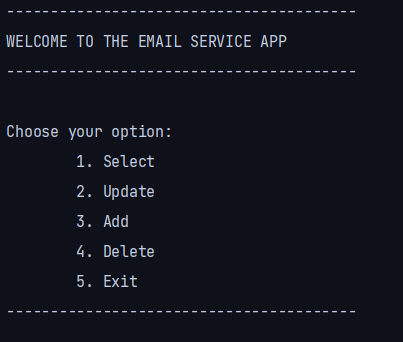
Сутність **Повідомлення** має 3 зв’язки, які вже були перераховані та містить такі атрибути: letter\_id, Тема.



*Схема бази даних(таблиці) PostgreSQL*

Використано мову програмування **C++** та офіційне API PostgreSQL– бібліотека **libpqxx**. Середовище розробки програмного забезпечення – JetBrains Clion.

**Вигляд основного меню консольної програми**

****

**Пункт 1:**

Відповідає за вивід бажаних даних із таблиць з можливим накладанням вимог до пошуку(WHERE …).

**Пункт 2:**

Відповідає за оновлення бажаних даних в таблиці.

**Пункт 3:**

Відповідає за додавання нового рядку в таблицю.

**Пункт 4:**

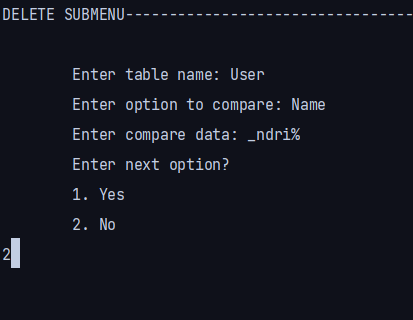
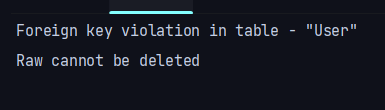
Відповідає за видалення даних із таблиці.

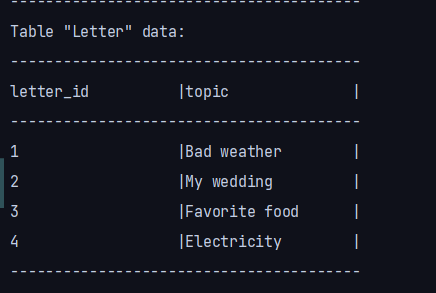
**Пункт 5:**

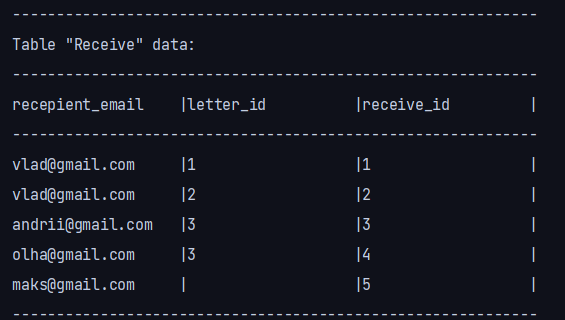
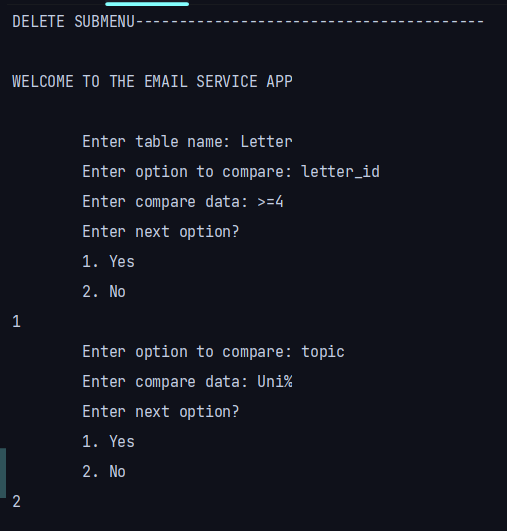
Завершення роботи програми

**Завдання 1**

Приклад спроби вилучення інформації із батьківської таблиці із виведенням помилки:



Приклад спроби вилучення інформації із батьківської таблиці із успішним вилученням:

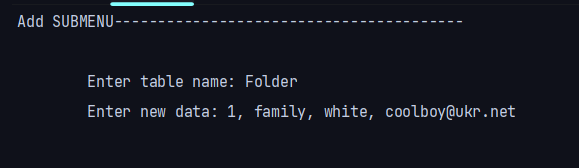


Приклад невдалого вставлення даних у дочірню таблицю:

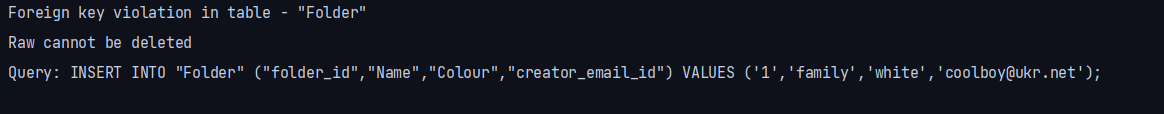
Батьківська таблиця:



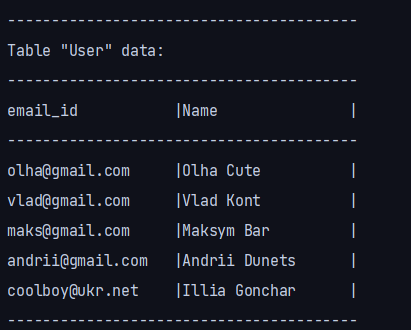
Ввід даних:



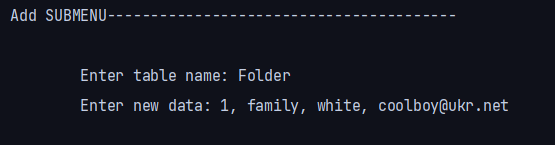
Помилка:



Тепер додамо цього користувача в батьківську таблицю:



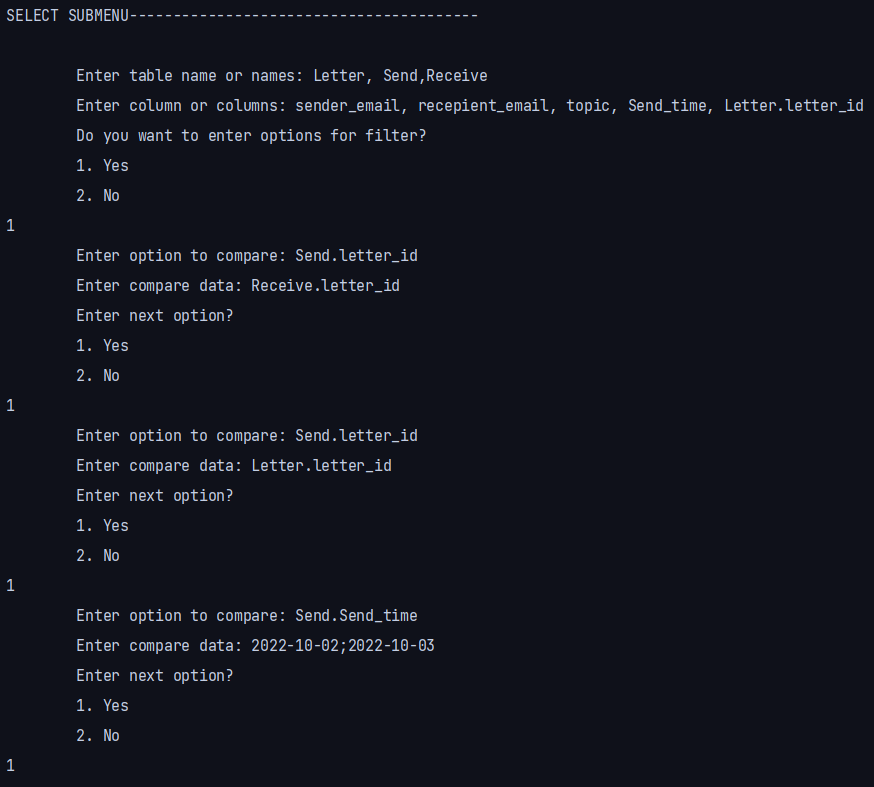
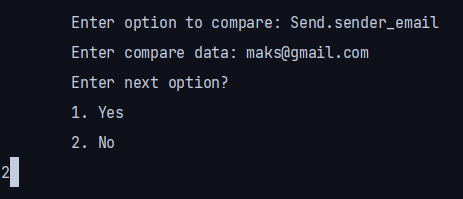
Спробуємо додати нові дані в дочірню таблицю ще раз:



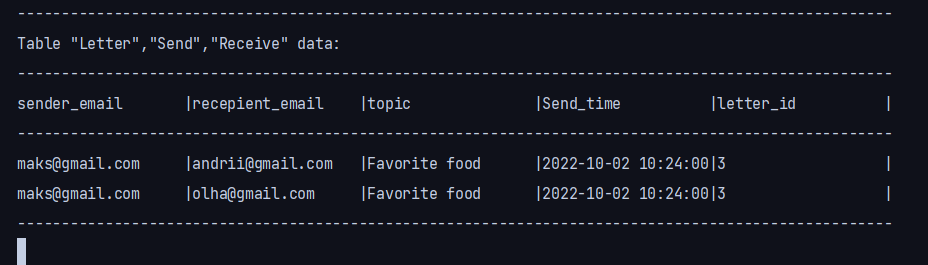
Результат успішний

**Завдання 3**

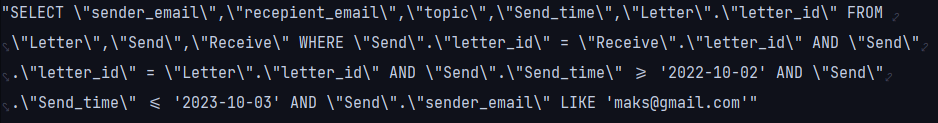
Приклад пошуково запиту 1:

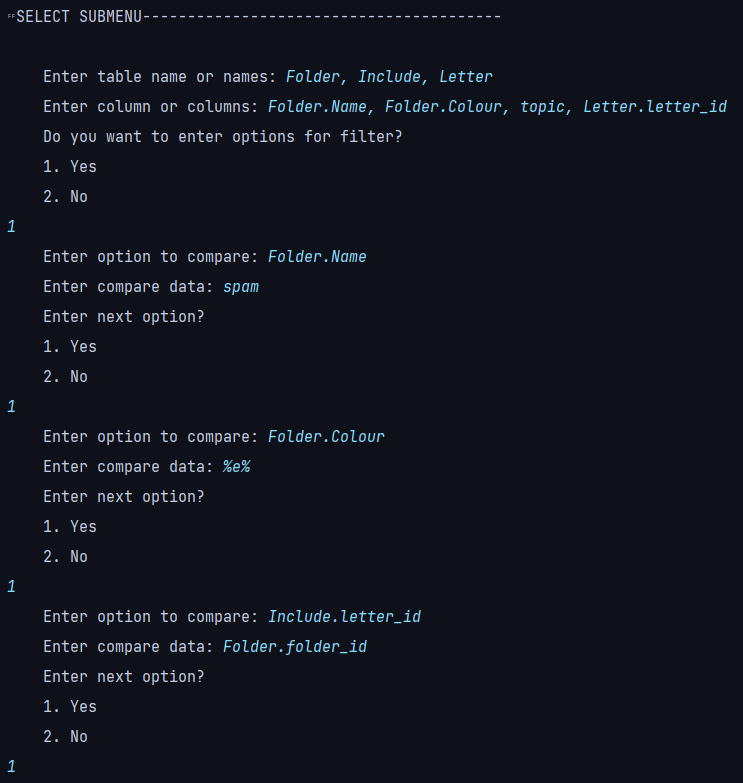
 

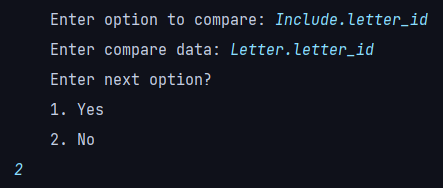
Результат:



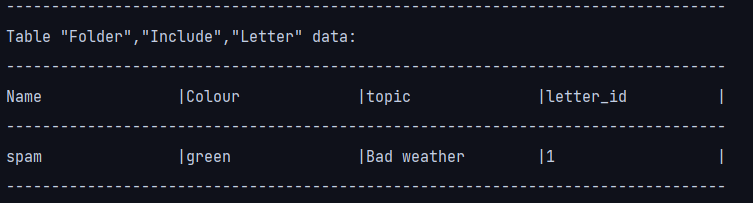
SQL запит, сформований програмою:



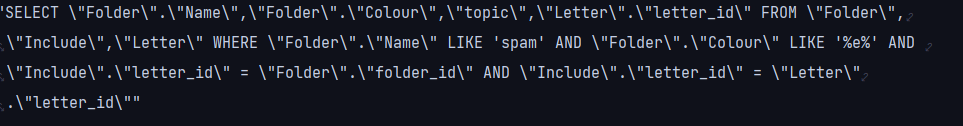
Приклад 2:



Результат:



SQL запит, сформований програмою:



**Завдання 4**

Програмник код модуля «Model» а також додатковий «Persistence layer» PostgreSQL для цього модуля:

**model.h:**

/\*\*

 \* @file model.h

 \* @author Dunets Andrii (andrijdunec@gmail.com)

 \* @brief Model part for MVC pattern

 \* @date 2023-01-08

 \*/

#ifndef EMAIL\_SERVICE\_APP\_MODEL\_H

#define EMAIL\_SERVICE\_APP\_MODEL\_H

#include "db\_backend.h"

**class** model

{

**private**:

db\_backend db;

std::string exception;

**public**:

model() = **delete**;

model(**const** std::string &dbName, **const** std::string &userName, **const** std::string &userPass)

: db(dbName, userName, userPass)

{

}

~model()

{

db.db\_disconnect();

}

std::vector<std::string> model\_getTableNames()

{

**return** db.db\_selectTablesName();

}

std::vector<std::vector<std::string>>

model\_selectTable(**const** std::string &tableName, std::string **const** &columns, std::string **const** &where = "")

{

**return** db.db\_selectTable(tableName, columns, where);

}

**void** model\_updateTable(**const** std::string &tableName, **const** std::string &columns, **const** std::string &new\_,

**const** std::string &where)

{

**try**

{

db.db\_updateTable(tableName, columns, new\_, where);

}

**catch** (std::exception **const** &e)

{

**throw** exception;

}

}

**void** model\_delete(**const** std::string &tableName, **const** std::string &where)

{

db.db\_delete(tableName, where);

}

std::string formWhere(**const** std::string &option, **const** std::string &toCmp, **const** **int** &colType)

{

**if** (colType != INT\_ID && colType != TEXT\_ID && colType != DATE\_ID)

{

**throw** std::string{"Invalid type of statement - "+option};

}

**if** (colType == TEXT\_ID)

{

exception = "Invalid statement input for VARCHAR type";

**return** option+" LIKE "+toCmp;

}

std::string delim;

**if** (colType == DATE\_ID)

{

exception = "Invalid statement input for TIMESTAMP type";

delim = ";'";

} **else**

{

exception = "Invalid statement input for INT type";

delim = "-'";

}

**char** text[toCmp.size()+1];

std::string delimEnd = "'";

**if** (strstr(toCmp.c\_str(), ".") != **nullptr**)

{

delimEnd = "";

sprintf(text, "%s", toCmp.c\_str());

} **else**

{

sprintf(text, "%s", toCmp.c\_str()+1);

}

**char** \*firstNum = strtok(text, delim.c\_str());

**char** \*secondNum = **nullptr**;

**if** (firstNum != **nullptr**)

{

**if** (firstNum[0] == '<' || firstNum[0] == '>')

{

**if** (firstNum[1] == '=')

{

**return** option+" "+firstNum[0]+"= "+delimEnd+(firstNum+2)+delimEnd;

}

**return** option+" "+firstNum[0]+" "+delimEnd+(firstNum+1)+delimEnd;

}

secondNum = strtok(**nullptr**, "'");

**if** (secondNum != **nullptr**)

{

**return** option+" >= "+delimEnd+firstNum+delimEnd+" AND "+option+" <= "+delimEnd+secondNum+delimEnd;

}

**return** option+" = "+delimEnd+firstNum+delimEnd;

}

**throw** exception;

}

**void** model\_add(**const** std::string &tableName, **const** std::string &data)

{

std::vector<std::string> vectorNames = model\_getColumnNames(tableName, "\*");

std::string stringNames;

**for** (**auto** &col: vectorNames)

{

stringNames += "\""+col+"\",";

}

stringNames.pop\_back();

db.db\_add(tableName, stringNames, data);

}

**unsigned** **long** model\_columnsNumber(**const** std::string &tableName)

{

**return** model\_getColumnNames(tableName, "\*").size();

}

**int** model\_getColumnType(**const** std::string &tableName, **const** std::string &column)

{

**char** text[column.size()+1];

sprintf(text, "%s", column.c\_str());

**if** (strstr(column.c\_str(), ".") == **nullptr**)

{

**return** db.db\_getColumnType(tableName, column);

}

**char** \*word = strtok(text, ".");

**return** db.db\_getColumnType(word, strtok(**nullptr**, "\0"));

}

std::vector<std::string> model\_getColumnNames(**const** std::string &tableName, **const** std::string &columns)

{

**return** db.db\_getColumnNames(tableName, columns);

}

};

#endif //EMAIL\_SERVICE\_APP\_MODEL\_H

**db\_backend.h**

/\*\*

 \* @file db\_backend.h

 \* @author Dunets Andrii (andrijdunec@gmail.com)

 \* @brief Persistence PostgreSQL layer for MVC pattern

 \* @date 2023-01-08

 \*/

#ifndef EMAIL\_SERVICE\_APP\_DB\_BACKEND\_H

#define EMAIL\_SERVICE\_APP\_DB\_BACKEND\_H

#include "pqxx/pqxx"

#include "pqxx/except"

#define INT\_ID (23)

#define TEXT\_ID (1043)

#define DATE\_ID (1114)

**using** db\_result = pqxx::result;

**class** db\_backend

{

**private**:

pqxx::connection con;

**public**:

db\_backend() = **delete**;

db\_backend(**const** std::string &dbName, **const** std::string &userName, **const** std::string &userPass) **try**

: con(std::string***{***

"dbname = "+dbName+" user = "+userName+" password = "+userPass+" hostaddr = 127.0.0.1 port = 5432"***}***)

{

std::cout << "Opened database successfully: " << con.dbname() << std::endl;

}

**catch** (std::exception **const** &e)

{

**throw** std::string{"Can't open database "+dbName};

}

**void** db\_disconnect()

{

con.disconnect();

}

std::vector<std::string> db\_selectTablesName()

{

std::vector<std::string> names;

pqxx::work query(con);

db\_result res = query.exec("SELECT tablename FROM pg\_tables\n"

"WHERE schemaname = 'public';");

names.reserve(res.size());

**for** (**auto** row: res)

{

names.emplace\_back(row.front().c\_str());

}

**return** names;

}

std::vector<std::vector<std::string>> db\_selectTable(**const** std::string &tableName,

**const** std::string &columns, **const** std::string &where = "")

{

pqxx::work query(con);

**try**

{

std::string text{"SELECT "+columns+" FROM "+tableName+" "+where};

db\_result res = query.exec(text);

std::vector<std::vector<std::string>> data;

data.resize(res.size());

**for** (**auto** i = 0; i < data.size(); ++i)

{

data[i].resize(res[i].size());

**for** (**auto** j = 0; j < data[i].size(); ++j)

{

data[i][j] = res[i][j].c\_str();

}

}

**return** data;

} **catch** (pqxx::undefined\_table **const** &e)

{

**throw** std::string{"Invalid table names - "+tableName};

} **catch** (pqxx::undefined\_column **const** &e)

{

**throw** std::string{"Invalid columns - "+columns+" for tables - "+tableName};

}

}

**int** db\_getColumnType(**const** std::string &tableName, **const** std::string &column)

{

pqxx::work query(con);

**try**

{

**return** query.exec("SELECT "+column+" FROM "+tableName).column\_type(column);

} **catch** (pqxx::undefined\_table **const** &e)

{

**throw** std::string{"Invalid table name - "+tableName};

} **catch** (pqxx::undefined\_column **const** &e)

{

**throw** std::string{"Invalid column - "+column+" for table - "+tableName};

}

}

**void** db\_updateTable(**const** std::string &tableName, **const** std::string &columns, **const** std::string &new\_,

**const** std::string &where)

{

pqxx::work query(con);

**try**

{

query.exec("UPDATE "+tableName+" SET "+columns+" = "+new\_+" "+where);

query.commit();

} **catch** (pqxx::restrict\_violation **const** &e)

{

**throw** std::string{"Restrict violation for column - "+columns+" in table - "+tableName};

} **catch** (pqxx::unique\_violation **const** &e)

{

**throw** std::string{

"Unique violation for column - "+columns+" in table - "+tableName+"\nValue is not unique: "+new\_};

} **catch** (pqxx::foreign\_key\_violation **const** &e)

{

**throw** std::string{

"Foreign key violation for column - "+columns+" in table - "+tableName+"\nValue already exist: "+

new\_};

}

}

**void** db\_delete(**const** std::string &tableName, **const** std::string &where)

{

pqxx::work query(con);

**try**

{

query.exec("DELETE FROM "+tableName+" "+where+";");

query.commit();

} **catch** (pqxx::foreign\_key\_violation **const** &e)

{

**throw** std::string{"Foreign key violation in table - "+tableName+"\nRaw cannot be deleted"};

}

}

**void** db\_add(**const** std::string &tableName, **const** std::string &columns, **const** std::string &data)

{

pqxx::work query(con);

**try**

{

query.exec("INSERT INTO "+tableName+" ("+columns+") VALUES ("+data+");");

query.commit();

} **catch** (pqxx::foreign\_key\_violation **const** &e)

{

**throw** std::string{"Foreign key violation in table - "+tableName+"\nRaw cannot be added\nQuery: "+e.query()};

} **catch** (pqxx::sql\_error **const** &e)

{

**throw** std::string{"Invalid amount or data of parameters\nQuery: "+e.query()};

}

}

std::vector<std::string> db\_getColumnNames(**const** std::string &tableName, **const** std::string &columns)

{

**try**

{

pqxx::work query(con);

db\_result res = query.exec("SELECT "+columns+" FROM "+tableName);

std::vector<std::string> result;

result.reserve(res.columns());

**for** (**auto** i = 0; i < result.capacity(); ++i)

{

result.emplace\_back(res.column\_name(i));

}

**return** result;

} **catch** (pqxx::undefined\_table **const** &e)

{

**throw** std::string{"Invalid table name - "+tableName};

} **catch** (pqxx::undefined\_column **const** &e)

{

**throw** std::string{"Invalid columns - "+columns+" for table - "+tableName};

}

}

};

#endif //EMAIL\_SERVICE\_APP\_DB\_BACKEND\_H