ОТЧЕТ  
ПО ЛАБОРАТОРНОЙ РАБОТЕ №1  
по дисциплине «Разработка приложений баз данных»  
«Использование технологии ODBC для создания приложений баз данных»  
**Вариант 15**

Факультет: АВТ Преподаватель: Менжулин С. А.

Группа: АВТ-943

Студент: Фокин Е.К.

Цель работы:

Получение практических навыков ODBC PostgreSQL.

Задание:

1. Разработать структуру БД в соответствии с вариантом задания. Задания составлены так, что после приведения к 3NF база данных должна содержать минимум 3 таблицы, отсутствуют ограничения на создание дополнительных таблиц или представлений.
2. При помощи средств администрирования PostgreSQL создать пользователя, задать ему пароль и создать принадлежащую ему базу данных.
3. Разработать программу на языке С++, использующую СУБД PostgreSQL для хранения данных, которая:
   1. При запуске программа подключается при помощи ODBC к базе данных, используя имя пользователя и пароль из п.2;
   2. Проверяет существование всех необходимых таблиц, в случае их отсутствия – создает;
   3. Для отображения разработанных классов использует один из архитектурных шаблонов: Data Mapper, Active Record, Table Data Gateway;
   4. Обслуживает пользовательский интерфейс (gui/cli, на выбор), содержащий набор операций в соответствии с семантикой варианта задания (предметной области), без привязки к реляционной модели. Запрещено использовать суррогатные первичные ключи в интерфейсе;

Задание варианта:

1. **БД «Гостиница»**

База данных должна содержать сведения о следующих объектах:

Распределение номеров с указанием общего количества мест в номере, количества свободных мест и проживающих.

Гости - фамилия, имя, отчество, пол, адрес, дата рождения, номер паспорта, дата выдачи, учреждение, выдавшее паспорт, номер комнаты, номер стоянки, регистрационный номер автомобиля, дата въезда, дата выезда, список оказанных услуг (наименование услуги, количество, цена).

Адресные данные горничных и расписание их дежурств.

Ход работы:

В ходе лабораторной работы будем использовать шаблон Data Mapper.

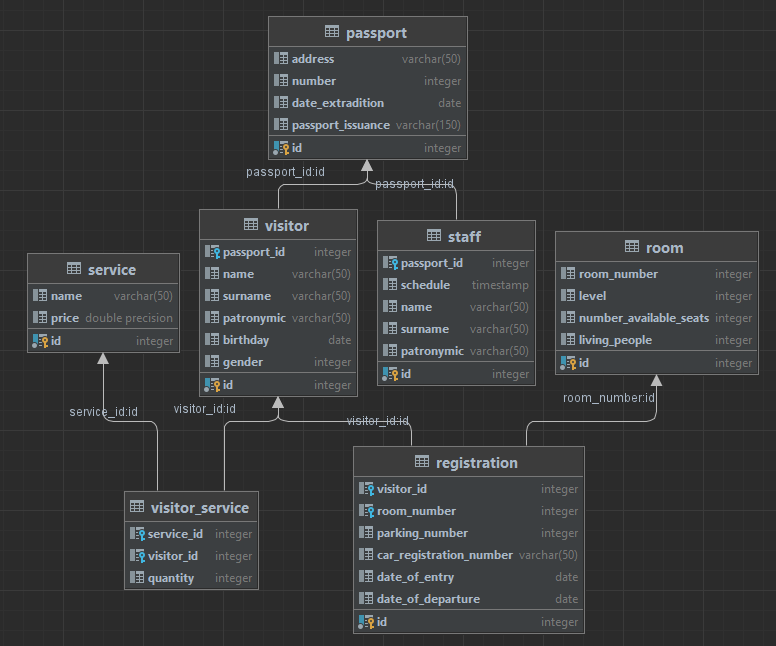


Рис. 1 Схема базы данных

Для начала, создадим пользователя, зададим ему пароль и добавим к нему базу данных

После установки специальных драйверов для ODBC, настроим подключение (Рис. 2):

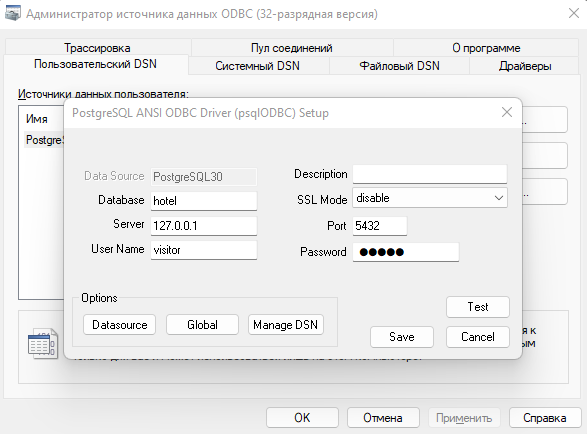


Рис. 2 Настройка подключения

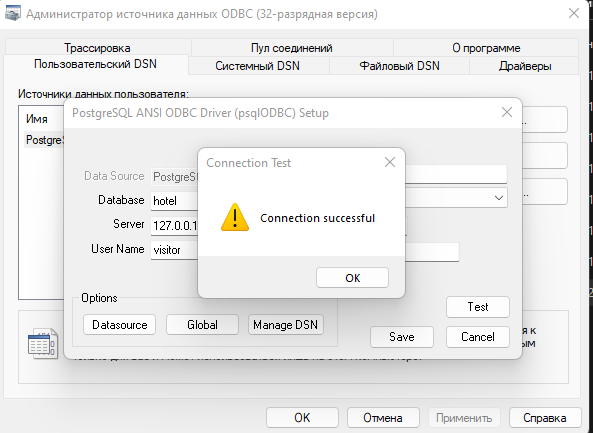


Рис. 3 Проверка подключения

В программе определим классы для каждой из сущностей

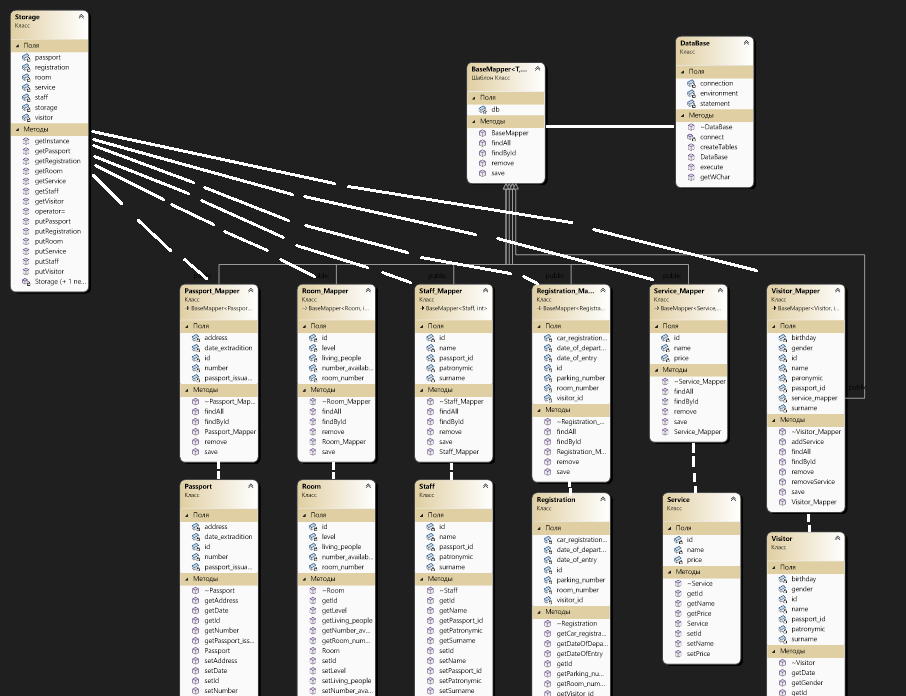


Рис. 4 Диаграмма классов

Для доступа к базе данных создадим класс DataBase, который имеет методы для подключения к базе и выполнения sql запросов – методы connect() и execute(), соответственно.

Также для каждого из классов сущностей создадим класс “mapper”, который будет отвечать за сохранение, получение, удаление сущностей из базы.

Представим полученные классы на диаграмме, чтобы увидеть их взаимодействие.

Вставлять весь проект в приложение будет нецелесообразным, из-за большого количества кода и одинаковых повторяющихся методов, поэтому было принято решение вставить только основные «куски» кода

Вывод:

В ходе лабораторной работы была создана база данных, а также таблицы для бизнес-сущностей и определены связи между ними. Также было создано приложение на языке C++, использующее ODBC для подключения к базе, позволяющее просматривать, создавать, удалять и искать сущности по заданным параметрам.

Приложение:

Класс DataBase для подключения к БД:

DataBase.h

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#ifndef DataBase\_HEADER

#define DataBases\_HEADER

#include <iostream>

#include "windows.h"

#include <sql.h>

#include <sqlext.h>

#include <string>

using namespace std;

class DataBase {

private:

SQLHENV environment;

SQLHDBC connection;

SQLHSTMT statement;

private:

void connect();

public:

DataBase();

~DataBase();

const SQLHSTMT& execute(const char\* sql);

const wchar\_t\* getWChar(const char\* sql);

void createTables();

};

#endif

#endif

DataBase.cpp

#include "DataBase.h"

void DataBase::connect()

{

SQLWCHAR name[] = L"PostgreSQL30";

SQLWCHAR user\_name[] = L"visitor";

SQLWCHAR pass[] = L"12345";

if (SQLAllocHandle(SQL\_HANDLE\_ENV, SQL\_NULL\_HANDLE, &environment) == SQL\_ERROR)

{

fwprintf(stderr, L"Unable to allocate an environment handle\n");

exit(-1);

}

if (SQLSetEnvAttr(environment, SQL\_ATTR\_ODBC\_VERSION, (SQLPOINTER)SQL\_OV\_ODBC3, 0) == SQL\_ERROR) {

fwprintf(stderr, L"Unable to set an environment variable\n");

exit(-1);

}

if (SQLAllocHandle(SQL\_HANDLE\_DBC, environment, &connection) == SQL\_ERROR) {

fwprintf(stderr, L"Unable to allocate an connection handle\n");

exit(-1);

}

if (SQLConnect(connection, name, SQL\_NTS, user\_name, SQL\_NTS, pass, SQL\_NTS) == SQL\_ERROR) {

fwprintf(stderr, L"Unable to connect\n");

exit(-1);

}

if (SQLAllocHandle(SQL\_HANDLE\_STMT, connection, &statement) == SQL\_ERROR) {

fwprintf(stderr, L"Unable to allocate stmt handle\n");

exit(-1);

}

}

DataBase::DataBase() {

connect();

createTables();

}

DataBase::~DataBase()

{

if (this->statement != SQL\_NULL\_HSTMT)

SQLFreeHandle(SQL\_HANDLE\_STMT, this->statement);

if (this->connection != SQL\_NULL\_HDBC) {

SQLDisconnect(this->connection);

SQLFreeHandle(SQL\_HANDLE\_DBC, this->connection);

}

if (this->environment != SQL\_NULL\_HENV)

SQLFreeHandle(SQL\_HANDLE\_ENV, this->environment);

}

const wchar\_t\* DataBase::getWChar(const char\* c)

{

const size\_t cSize = strlen(c) + 1;

wchar\_t\* wc = new wchar\_t[cSize];

mbstowcs(wc, c, cSize);

return wc;

}

void DataBase::createTables()

{

SQLRETURN ret;

ret = SQLExecDirect(statement,

(SQLWCHAR\*)getWChar("CREATE TABLE IF NOT EXISTS room(\

id serial primary key,\

room\_number integer,\

level integer,\

number\_available\_seats integer,\

living\_people integer\

)"), SQL\_NTS);

ret = SQLExecDirect(statement,

(SQLWCHAR\*)getWChar("CREATE TABLE IF NOT EXISTS passport(\

id serial primary key,\

address varchar(50),\

number integer,\

date\_extradition date,\

passport\_issuance varchar(50)\

)"), SQL\_NTS);

ret = SQLExecDirect(statement,

(SQLWCHAR\*)getWChar("CREATE TABLE if not exists visitor(\

id serial primary key,\

passport\_id int references passport(id),\

name varchar(50),\

surname varchar(50),\

patronymic varchar(50),\

gender int,\

birthday date"), SQL\_NTS);

ret = SQLExecDirect(statement,

(SQLWCHAR\*)getWChar("CREATE TABLE IF NOT EXISTS registration(\

id serial primary key,\

visitor\_id integer references visitor(id),\

room\_number integer references room(id),\

parking\_number integer,\

car\_registration\_number varchar(50),\

date\_of\_entry date,\

date\_of\_departure date"), SQL\_NTS);

ret = SQLExecDirect(statement,

(SQLWCHAR\*)getWChar("CREATE TABLE IF NOT EXISTS service(\

id serial primary key,\

name varchar(50),\

price float"), SQL\_NTS);

ret = SQLExecDirect(statement,

(SQLWCHAR\*)getWChar("CREATE TABLE IF NOT EXISTS visitor\_service(\

service\_id integer references service(id),\

visitor\_id integer references visitor(id),\

quantity integer"), SQL\_NTS);

ret = SQLExecDirect(statement,

(SQLWCHAR\*)getWChar("CREATE TABLE IF NOT EXISTS staff(\

id serial primary key,\

passport\_id integer references passport(id),\

schedule timestamp,\

name varchar(50),\

surname varchar(50),\

patronymic varchar(50)"), SQL\_NTS);

}

const SQLHSTMT& DataBase::execute(const char\* sql)

{

SQLRETURN ret = SQLExecDirect(statement,

(SQLWCHAR\*)getWChar(sql), SQL\_NTS);

if (ret == SQL\_SUCCESS) {

return this->statement;

}

return nullptr;

}

Класс BaseMapper для взаимодействия с БД:

BaseMapper:

#pragma once

#ifndef BaseMapper\_HEADER

#define BaseMapper\_HEADER

#include "DataBase.h"

#include <vector>

template<class T, class TypeID>

class BaseMapper {

protected:

DataBase& db;

public:

BaseMapper(DataBase& db);

virtual void save(const T& date) = 0;

virtual void remove(TypeID typeID) = 0;

virtual T\* findById(TypeID typeID) = 0;

virtual std::vector<T\*> findAll() = 0;

};

template<class T, class TypeID>

BaseMapper<T, TypeID>::BaseMapper(DataBase& db) : db(db) {}

#endif

Класс Storage для хранения данных в оперативной памяти:

Storage.h

#pragma once

#ifndef Storage\_HEADER

#define Storage\_HEADER

#include <map>

#include "Room.h"

#include "Registration.h"

#include "Visitor.h"

#include "Passport.h"

#include "Staff.h"

#include "Service.h"

using namespace std;

class Storage

{

private:

map<int, Room > room;

map<int, Registration > registration;

map<int, Visitor > visitor;

map<int, Passport > passport;

map<int, Staff > staff;

map<int, Service > service;

Storage();

static Storage\* storage;

public:

Storage(Storage& other) = delete;

void operator = (Storage&) = delete;

static Storage\* getInstance();

Room\* getRoom(int);

Registration\* getRegistration(int);

Visitor\* getVisitor(int);

Passport\* getPassport(int);

Staff\* getStaff(int);

Service\* getService(int);

void putRoom(Room& room);

void putRegistration(Registration& registration);

void putVisitor(Visitor& visitor);

void putPassport(Passport& passport);

void putStaff(Staff& staff);

void putService(Service& service);

};

#endif

Storage.cpp

#include "Storage.h"

Storage\* Storage::storage;

Storage::Storage()

{

}

Storage\* Storage::getInstance()

{

if (storage == nullptr)

{

storage = new Storage();

}

return storage;

}

Room\* Storage::getRoom(int id)

{

return &(this->room.find(id)->second);

}

Registration\* Storage::getRegistration(int id)

{

return &(this->registration.find(id)->second);

}

Visitor\* Storage::getVisitor(int id)

{

return &(this->visitor.find(id)->second);

}

Passport\* Storage::getPassport(int id)

{

return &(this->passport.find(id)->second);

}

Staff\* Storage::getStaff(int id)

{

return &(this->staff.find(id)->second);

}

Service\* Storage::getService(int id)

{

return &(this->service.find(id)->second);

}

void Storage::putRoom(Room& room)

{

auto it = this->room.find(room.getId());

if (it == this->room.end())

{

this->room[room.getId()] = room;

return;

}

auto& r = it->second;

r.setLevel(room.getLevel());

r.setLiving\_people(room.getLiving\_people());

r.setNumber\_available\_seats(room.getNumber\_available\_seats());

r.setRoom\_number(room.getRoom\_number());

}

void Storage::putPassport(Passport& passport\_info)

{

auto it = this->passport.find(passport\_info.getId());

if (it == this->passport.end()) {

this->passport[passport\_info.getId()] = passport\_info;

return;

}

auto& pa = it->second;

pa.setAddress(passport\_info.getAddress());

pa.setNumber(passport\_info.getNumber());

pa.setDate(passport\_info.getDate());

pa.setPassport\_issuance(passport\_info.getPassport\_issuance());

}

void Storage::putStaff(Staff& staff)

{

auto it = this->staff.find(staff.getId());

if (it == this->staff.end()) {

this->staff[staff.getId()] = staff;

return;

}

auto& st = it->second;

st.setPassport\_id(staff.getPassport\_id());

st.setName(staff.getName());

st.setSurname(staff.getSurname());

st.setPatronymic(staff.getPatronymic());

}

void Storage::putRegistration(Registration& registration)

{

auto it = this->registration.find(registration.getId());

if (it == this->registration.end()) {

this->registration[registration.getId()] = registration;

return;

}

auto& re = it->second;

re.setVisitor\_id(registration.getVisitor\_id());

re.setRoom\_number(registration.getRoom\_number());

re.setParking\_number(registration.getParking\_number());

re.setCar\_registration\_number(registration.getCar\_registration\_number());

re.setDateOfEntry(registration.getDateOfEntry());

re.setDateOfDeparture(registration.getDateOfDeparture());

}

void Storage::putVisitor(Visitor& visitor)

{

auto it = this->visitor.find(visitor.getId());

if (it == this->visitor.end()) {

this->visitor[visitor.getId()] = visitor;

return;

}

auto& vi = it->second;

vi.setPassport\_id(visitor.getPassport\_id());

vi.setName(visitor.getName());

vi.setSurname(visitor.getSurname());

vi.setPatronymic(visitor.getPatronymic());

vi.setDate(visitor.getDate());

}

void Storage::putService(Service& service)

{

auto it = this->service.find(service.getId());

if (it == this->service.end()) {

this->service[service.getId()] = service;

return;

}

auto& se = it->second;

se.setName(service.getName());

se.setPrice(service.getPrice());

}

Пример класса, определяющего таблицу в БД:

Visitor.h

#pragma once

#ifndef Visitor\_HEADER

#define Visitor\_HEADER

#include <iostream>

#include <string>

#include <sstream>

#include "Date.h"

using namespace std;

class Visitor

{

private:

int id;

int passport\_id;

bool gender;

string name;

string surname;

string patronymic;//Вроде отчество

Date birthday;

public:

Visitor();

~Visitor();

int getId() const;

void setId(int);

int getPassport\_id() const;

void setPassport\_id(int);

string getName() const;

void setName(const string& name);

string getSurname() const;

void setSurname(const string& surname);

string getPatronymic() const;

void setPatronymic(const string& patronymic);

bool getGender() const;

void setGender(bool);

Date getDate() const;

void setDate(Date);

std::string toStringDate() const;

Date setStringDate(std::string& date);

};

#endif

Visitor.cpp

#include "Visitor.h"

Visitor::Visitor()

{

}

Visitor::~Visitor()

{

}

int Visitor::getId () const

{

return this->id;

}

void Visitor::setId(int id)

{

this->id = id;

}

int Visitor::getPassport\_id() const

{

return this->passport\_id;

}

void Visitor::setPassport\_id(int passport\_id)

{

this->passport\_id = passport\_id;

}

string Visitor::getName() const

{

return this->name;

}

void Visitor::setName(const string& name)

{

this->name = name;

}

string Visitor::getSurname() const

{

return this->surname;

}

void Visitor::setSurname(const string& surname)

{

this->surname = surname;

}

string Visitor::getPatronymic() const

{

return this->patronymic;

}

void Visitor::setPatronymic(const string& patronymic)

{

this->patronymic = patronymic;

}

bool Visitor::getGender() const

{

return this->gender;

}

void Visitor::setGender(bool gender)

{

this->gender = gender;

}

Date Visitor::getDate() const

{

return this->birthday;

}

void Visitor::setDate(Date date)

{

this->birthday.day = date.day;

this->birthday.month = date.month;

this->birthday.year = date.year;

}

std::string Visitor::toStringDate() const

{

return std::to\_string(this->birthday.year) +

"-" + std::to\_string(this->birthday.month) +

"-" + std::to\_string(this->birthday.day);

}

Date Visitor::setStringDate(std::string& date)

{

Date strDate;

std::stringstream in(date);

std::getline(in, date, '-');

strDate.year = stoi(date);

std::getline(in, date, '-');

strDate.month = stoi(date);

std::getline(in, date, '-');

strDate.day = stoi(date);

return strDate;

}

Пример класса-маппера таблицу в БД:

Visitor\_Mapper.h

#pragma once

#ifndef Visitor\_Mapper\_HEADER

#define Visitor\_Mapper\_HEADER

#include "BaseMapper.h"

#include "Visitor.h"

#include "Service\_Mapper.h"

#define ID\_LEN 8

#define NAME\_LEN 1024

class Visitor\_Mapper : public BaseMapper<Visitor, int>

{

private:

SQLINTEGER id;

SQLINTEGER passport\_id;

SQLINTEGER gender;

SQLCHAR name[64];

SQLCHAR surname[64];

SQLCHAR paronymic[64];

SQLCHAR birthday[64];

Service\_Mapper& service\_mapper;

public:

Visitor\_Mapper(DataBase& db, Service\_Mapper& service\_mapper);

~Visitor\_Mapper();

void save(const Visitor& visitor) override;

void remove(int id) override;

Visitor\* findById(int id) override;

std::vector<Visitor\*> findAll() override;

void addService(int service\_id, int visitor\_id, int quantity);

void removeService(int service\_id, int visitor\_id, int quantity);

};

#endif

Visitor\_Mapper.cpp

#include "Visitor\_Mapper.h"

#include "Storage.h"

Visitor\_Mapper::Visitor\_Mapper(DataBase& db, Service\_Mapper& service\_mapper) : BaseMapper(db), service\_mapper(service\_mapper)

{

}

Visitor\_Mapper::~Visitor\_Mapper()

{

}

void Visitor\_Mapper::save(const Visitor& visitor)

{

char buf[1024];

if (visitor.getId() != -1) {

sprintf(buf,

"UPDATE visitor SET passport\_id = '%d', \"name\" = '%s', surname = '%s', patronymic = '%s', gender = '%d', birthday = '%s' WHERE id = '%d'",

visitor.getPassport\_id(),

visitor.getName().c\_str(),

visitor.getSurname().c\_str(),

visitor.getPatronymic().c\_str(),

visitor.getGender(),

visitor.toStringDate().c\_str(),

visitor.getId());

}

else {

sprintf(buf, "INSERT INTO visitor(passport\_id, name, surname, patronymic, gender, birthday) VALUES ('%d', '%s', '%s', '%s', '%d', '%s')",

visitor.getPassport\_id(),

visitor.getName().c\_str(),

visitor.getSurname().c\_str(),

visitor.getPatronymic().c\_str(),

visitor.getGender(),

visitor.toStringDate().c\_str());

}

db.execute(buf);

}

void Visitor\_Mapper::remove(int id)

{

char buf[64];

sprintf(buf, "DELETE FROM visitor WHERE id = %d", id);

db.execute(buf);

}

Visitor\* Visitor\_Mapper::findById(int id)

{

char buf[64];

sprintf(buf, "SELECT vi.id, vi.passport\_id, vi.name, vi.surname, vi.patronymic, vi.birthday FROM visitor AS vi WHERE vi.id = %d", id);

SQLHSTMT statement = db.execute(buf);

SQLRETURN retcode;

retcode = SQLBindCol(statement, 1, SQL\_C\_LONG, &this->id, ID\_LEN, nullptr);

retcode = SQLBindCol(statement, 2, SQL\_C\_LONG, &this->passport\_id, ID\_LEN, nullptr);

retcode = SQLBindCol(statement, 3, SQL\_C\_CHAR, &this->name, NAME\_LEN, nullptr);

retcode = SQLBindCol(statement, 4, SQL\_C\_CHAR, &this->surname, NAME\_LEN, nullptr);

retcode = SQLBindCol(statement, 5, SQL\_C\_CHAR, &this->paronymic, NAME\_LEN, nullptr);

retcode = SQLBindCol(statement, 6, SQL\_C\_CHAR, &this->birthday, NAME\_LEN, nullptr);

retcode = SQLFetch(statement);

if (retcode == SQL\_SUCCESS || retcode == SQL\_SUCCESS\_WITH\_INFO) {

Visitor visitor;

visitor.setId(id);

visitor.setPassport\_id(passport\_id);

visitor.setName(reinterpret\_cast<char\*>(name));

visitor.setSurname(reinterpret\_cast<char\*>(surname));

visitor.setPatronymic(reinterpret\_cast<char\*>(paronymic));

string str = reinterpret\_cast<char\*>(birthday);

SQLFreeStmt(statement, SQL\_CLOSE);

return Storage::getInstance()->getVisitor(visitor.getId());

}

else if (retcode != SQL\_NO\_DATA) {

std::cout << "SearchById error in order\_mapper\n";

}

return nullptr;

}

vector<Visitor\*> Visitor\_Mapper::findAll()

{

std::vector<Visitor\*> Visitor\_vector;

SQLHSTMT statement = db.execute("SELECT vi.id, vi.passport\_id, vi.name, vi.surname, vi.patronymic, vi.gender, vi.birthday FROM visitor AS vi ORDER BY vi.id ASC");

SQLRETURN retcode;

retcode = SQLBindCol(statement, 1, SQL\_C\_LONG, &id, ID\_LEN, nullptr);

retcode = SQLBindCol(statement, 2, SQL\_C\_LONG, &passport\_id, ID\_LEN, nullptr);

retcode = SQLBindCol(statement, 3, SQL\_C\_CHAR, &name, NAME\_LEN, nullptr);

retcode = SQLBindCol(statement, 4, SQL\_C\_CHAR, &surname, NAME\_LEN, nullptr);

retcode = SQLBindCol(statement, 5, SQL\_C\_CHAR, &paronymic, NAME\_LEN, nullptr);

retcode = SQLBindCol(statement, 6, SQL\_C\_LONG, &gender, ID\_LEN, nullptr);

retcode = SQLBindCol(statement, 7, SQL\_C\_CHAR, &birthday, NAME\_LEN, nullptr);

while (true)

{

retcode = SQLFetch(statement);

if (retcode == SQL\_SUCCESS || retcode == SQL\_SUCCESS\_WITH\_INFO) {

Visitor visitor;

visitor.setId(id);

visitor.setPassport\_id(passport\_id);

visitor.setName(reinterpret\_cast<char\*>(name));

visitor.setSurname(reinterpret\_cast<char\*>(surname));

visitor.setPatronymic(reinterpret\_cast<char\*>(paronymic));

visitor.setGender(gender);

string str = reinterpret\_cast<char\*>(birthday);

visitor.setDate(visitor.setStringDate(str));

Storage::getInstance()->putVisitor(visitor);

Visitor\_vector.push\_back(Storage::getInstance()->getVisitor(visitor.getId()));

}

else {

if (retcode != SQL\_NO\_DATA) {

std::cout << "Error fetching genres\n";

}

break;

}

}

SQLFreeStmt(statement, SQL\_CLOSE);

return Visitor\_vector;

}

void Visitor\_Mapper::addService(int service\_id, int visitor\_id, int quantity)

{

char buf[128];

sprintf(buf, "INSERT INTO visitor\_service(service\_id, visitor\_id, quantity) values (%d, %d, %d)", service\_id, visitor\_id, quantity);

db.execute(buf);

}

void Visitor\_Mapper::removeService(int service\_id, int visitor\_id, int quantity)

{

char buf[128];

sprintf(buf, "DELETE FROM visitor\_service WHERE (service\_id = %d AND visitor\_id = %d AND quantity = %d)", service\_id, visitor\_id, quantity);

db.execute(buf);

}