**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**КЫРГЫЗСКОЙ РЕСПУБЛИКИ**

**КЫРГЫЗСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. И.Раззакова**

**ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ**

Кафедра: **Программное обеспечение компьютерных систем**

**Курсовая работа**

по дисциплине **«Объектно-ориентированное программирование»**

по теме **«Система управления каталогом автомобильных запчастей»**

Выполнил: студент группы ПИ-3-21

Джанышбекова Акмарал

Проверил: Мусабаев Э. Б.

**Бишкек 2024**

Оглавление

[**Введение** 3](#_Toc167437108)

[**Цель** 3](#_Toc167437109)

[**1.** **Анализ и формирование требований** 3](#_Toc167437110)

[**1.1.** **Пользовательские требования** 3](#_Toc167437111)

[**1.2.** **Функциональные требования** 3](#_Toc167437112)

[**1.3.** **Нефункциональные требования** 3](#_Toc167437113)

[**2.** **Диаграмма USE CASE** 4](#_Toc167437114)

[**3.** **Диаграмма классов** 4](#_Toc167437115)

[**4.** **Описание классов и их назначения** 5](#_Toc167437116)

[**4.1.** **Абстрактный класс Users** 5](#_Toc167437117)

[**4.2. Класс ServiceWorker** 6](#_Toc167437118)

[**4.3. Класс PartStoreWorker** 8](#_Toc167437119)

[**4.4. Класс Country** 11](#_Toc167437120)

[**4.5. Класс CarBrand** 12](#_Toc167437121)

[**4.6. Класс Manufacturer** 13](#_Toc167437122)

[**4.7. Класс Part** 14](#_Toc167437123)

[**4.8. Класс Component** 15](#_Toc167437124)

[**4.9. Класс Assembly** 16](#_Toc167437125)

[**4.10. Класс MakeAssembly** 17](#_Toc167437126)

[**4.11. Класс History** 18](#_Toc167437127)

[**4.12. Класс Car** 20](#_Toc167437128)

[**4.13. Класс MakeCar** 23](#_Toc167437129)

[**5.** **Подключение базы данных к проекту** 25](#_Toc167437130)

# **Введение**

Современные автомобили состоят из множества деталей, и управление этими запчастями требует эффективной системы. Автосервисы и магазины запчастей нуждаются в надежном способе учета и замены агрегата в машине, чтобы обеспечивать качественное обслуживание и учитывать свои запасы. Для этого мы создаем систему управления каталогом запчастей автомобилей.

# **Цель**

Представить обзор и работу структуры классов автомобильного каталога, который содержит информацию о запчастях автомобиля, их производителях, странах производства, а также о пользователях системы, для обеспечения эффективного управления данными, учета взаимозаменяемости агрегатов, и предоставления возможности быстрого поиска и доступа к информации для автосервисов и магазинов запчастей.

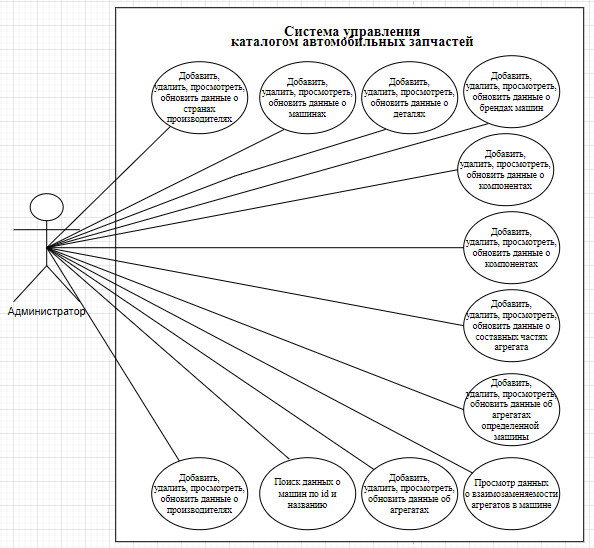
1. **Анализ и формирование требований**
   1. **Пользовательские требования**

Пользователи системы: администратор

Администратор должен иметь возможность выполнять следующие функции:

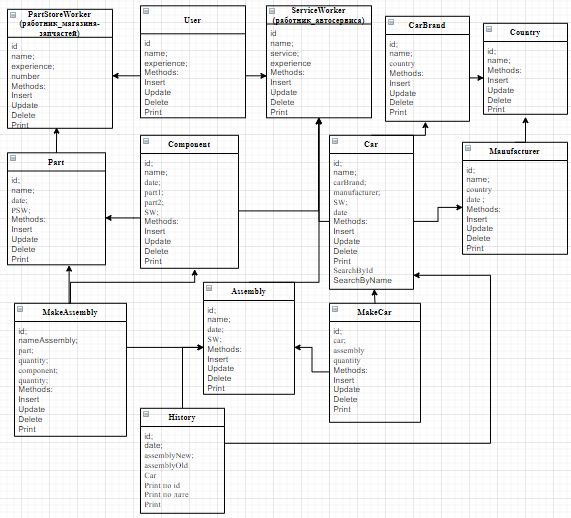
1. Добавление, удаление, просмотр, обновление данных о сотруднике Автосервиса и магазина Запчастей
2. Добавление, удаление, просмотр, обновление данных о странах производителях
3. Добавление, удаление, просмотр, обновление данных о брендах машин
4. Добавление, удаление, просмотр, обновление данных о производителях
5. Добавление, удаление, просмотр, обновление, поиск данных о машинах
6. Добавление, удаление, просмотр, обновление данных об агрегатах (сборка) определенной машины (то есть из каких частей состоит машина)
7. Добавление, удаление, просмотр, обновление данных об агрегатах (сборка)
8. Добавление, удаление, просмотр, обновление данных о деталях
9. Добавление, удаление, просмотр, обновление данных о компонентах
10. Добавление, удаление, просмотр, обновление данных о составе агрегата, то есть из каких деталей и компонентов состоит выбранных агрегат
11. Просмотр данных об учете взаимозаменяемости агрегатов в машине
    1. **Функциональные требования**
12. Обеспечить возможность полного доступа к информации о деталях, компонентов и его составляющих деталях и запчастей каждой машины.
13. Система автоматически введет учет о замене агрегата в машине и сохраняет эту информацию в базе данных.
14. Получение данных об учете взаимозаменяемости агрегатов в машине по указанной дате или id машины.
15. Поиск данных о машине по его названию и по id.
    1. **Нефункциональные требования**

Система реализована на языке С++ в виде консольного приложения.

1. **Диаграмма USE CASE**
2. **Диаграмма классов**

Система содержит 13 классов:

1. User
2. PartStoreWorker
3. ServiceWorker
4. Country
5. CarBrand
6. Manufacturer
7. Car
8. Component
9. Part
10. Assembly
11. MakeCar
12. MakeAssembly
13. History



1. **Описание классов и их назначения**

## **4.1.** **Абстрактный класс Users**

**Назначение класса**

Класс Users предназначен для управления данными сотрудников системы автомобильного каталога. Он содержит информацию о пользователях, таких как их идентификаторы, имена и опыт работы. Класс является абстрактным, предоставляя интерфейс для работы с пользователями, который должен быть реализован в производных классах.

**Описание полей**

|  |  |
| --- | --- |
| Наименование | Назначение |
| vector<int> id | Вектор, хранящий уникальные идентификаторы пользователей. |
| vector<string> name | Вектор, хранящий имена сотрудников. |
| vector<int> experience | Вектор, хранящий информацию о стаже работы пользователей |

**Описание методов**

|  |  |
| --- | --- |
| Наименование | Назначение |
| virtual void Delete(int id) = 0 | Чисто виртуальный метод, который должен удалять пользователя по его идентификатору. |
| virtual void Print() = 0 | Чисто виртуальный метод, который должен выводить информацию обо всех пользователях. |

Листинг класса

#pragma once

#include <iostream>

#include <iostream>

#include <vector>

using namespace std;

class Users

{

protected:

vector<int> id;

vector<string> name;

vector<int> experience;

public:

virtual void Delete(int id) = 0;

virtual void Print() = 0;

};

## **4.2. Класс ServiceWorker**

**Назначение класса**

Класс ServiceWorker предназначен для управления данными сотрудников автосервиса в системе автомобильного каталога. Этот класс наследуется от базового класса Users и расширяет его функциональность, добавляя специфические для сотрудников автосервиса поля и методы.

**Описание полей**

|  |  |
| --- | --- |
| Наименование | Назначение |
| vector<string> service; | Вектор, хранящий информацию об сервисах, которые могут предоставить сотрудники. |
| Поля из класса User |  |

**Описание методов**

|  |  |  |
| --- | --- | --- |
| Наименование | Код метода | Назначение |
| void Insert (string name, int experience, stringservice); | try {  sql::PreparedStatement\* pstmt;  sql::Connection\* con;  con = ConectDB();  pstmt = con->prepareStatement("INSERT INTO ServiceWorkers (name,experience,service) VALUES(?,?,?)");  pstmt->setString(1, name);  pstmt->setInt(2, experience);  pstmt->setString(3, service);  pstmt->execute();  cout << "Saved" << endl;  delete pstmt;  delete con;  }  catch (sql::SQLException e)  {  cout << "Cannot Insert. Error message: " << e.what() << endl;  system("pause");  } | Метод для добавления нового сотрудника автосервиса. Принимает имя, стаж работы и название автосервиса, в который будет добавлен сотрудник. |
| void Update(int id, string name, int experience, string service); | sql::PreparedStatement\* pstmt;  sql::Connection\* con;  try  {  con = ConectDB();  pstmt = con->prepareStatement("UPDATE ServiceWorkers SET name = ?, experience = ?, service = ? WHERE id = ?");  pstmt->setString(1, name);  pstmt->setInt(2, experience);  pstmt->setString(3, service);  pstmt->setInt(4, id);  pstmt->execute();  cout << "Updated" << endl;  delete pstmt;  delete con;  }  catch (sql::SQLException e)  {  cout << "Cannot Update. Error message: " << e.what() << endl;  system("pause");  } | Метод для обновления данных сотрудника автосервиса. Принимает идентификатор сотрудника, новое имя, стаж работы и название автосервиса для обновления соответствующих данных. |
| void Delete(int id); | sql::PreparedStatement\* pstmt;  sql::Connection\* con;  try  {  con = ConectDB();  pstmt = con->prepareStatement("DELETE FROM ServiceWorkers WHERE id = ?");  pstmt->setInt(1, id);  pstmt->execute();  cout << "Deleted" << endl;  delete pstmt;  delete con;  }  catch (sql::SQLException e)  {  cout << "Cannot Delete. Error message: " << e.what() << endl;  system("pause");  } | Метод для удаления сотрудника автосервиса по его идентификатору. Реализует чисто виртуальный метод из базового класса Users |
| void Print() | sql::Statement\* stmt;  sql::ResultSet\* res;  sql::Connection\* con;  try  {  con = ConectDB();  stmt = con->createStatement();  res = stmt->executeQuery("SELECT \* FROM ServiceWorkers");  cout << "ID\tName\tExperience\tService" << endl;  cout << "---------------------------------" << endl;  while (res->next())  {  cout << res->getInt("id") << "\t";  cout << res->getString("name") << "\t\t";  cout << res->getInt("experience") << "\t";  cout << res->getString("service") << endl;  }  delete res;  delete stmt;  delete con;  }  catch (sql::SQLException e)  {  cout << "Cannot retrieve data. Error message: " << e.what() << endl;  system("pause");  } | Метод для вывода информации обо всех сотрудниках автосервиса. Реализует чисто виртуальный метод из базового класса Users |

**Листинг класса**

#pragma once

#include "Connect.h"

#include "User.h"

#include <iostream>

#include <vector>

class ServiceWorker: public Users

{

private:

vector<string> service;

public:

void Insert(string name, int experience, string service);

void Update(int id, string name, int experience, string service);

void Delete(int id);

void Print();

};

## **4.3. Класс PartStoreWorker**

**Назначение класса**

Класс PartStoreWorker предназначен для управления данными работников магазинов запчастей в системе автомобильного каталога. Этот класс наследуется от абстрактного класса Users и реализует методы для добавления, обновления, удаления и вывода информации о работниках.

**Описание** **полей**

|  |  |
| --- | --- |
| Наименование | Назначение |
| vector<string> number; | Вектор, хранящий номера телефонов работников магазина запчастей. |
| Поля из класса User |  |

**Описание** **методов**

|  |  |  |
| --- | --- | --- |
| Наименование | Код метода | Назначение |
| void Insert(string name, int experience, string number); | try {  sql::PreparedStatement\* pstmt;  sql::Connection\* con;  con = ConectDB();  pstmt = con->prepareStatement("INSERT INTO PartStoreWorkers (name, experience, number) VALUES(?, ?, ?)");  pstmt->setString(1, name);  pstmt->setInt(2, experience);  pstmt->setString(3, number);  pstmt->execute();  cout << "Saved" << endl;  delete pstmt;  delete con;  }  catch (sql::SQLException e)  {  cout << "Cannot Insert. Error message: " << e.what() << endl;  system("pause");  } | Метод для добавления нового работника магазина запчастей. Вставляет имя, опыт и номер телефона в базу данных. |
| void Update(int id, string name, int experience, string number); | sql::PreparedStatement\* pstmt;  sql::Connection\* con;  try  {  con = ConectDB();  pstmt = con->prepareStatement("UPDATE PartStoreWorkers SET name = ?, experience = ?, number = ? WHERE id = ?");  pstmt->setString(1, name);  pstmt->setInt(2, experience);  pstmt->setString(3, number);  pstmt->setInt(4, id);  pstmt->execute();  cout << "Updated" << endl;  delete pstmt;  delete con;  }  catch (sql::SQLException e)  {  cout << "Cannot Update. Error message: " << e.what() << endl;  system("pause");  } | Метод для обновления информации о существующем работнике магазина запчастей. Обновляет имя, опыт и номер телефона по идентификатору. |
| void Delete(int id); | sql::PreparedStatement\* pstmt;  sql::Connection\* con;  try  {  con = ConectDB();  pstmt = con->prepareStatement("DELETE FROM PartStoreWorkers WHERE id = ?");  pstmt->setInt(1, id);  pstmt->execute();  cout << "Deleted" << endl;  delete pstmt;  delete con;  }  catch (sql::SQLException e)  {  cout << "Cannot Delete. Error message: " << e.what() << endl;  system("pause");  } | Метод для удаления работника магазина запчастей по его идентификатору.  Реализует чисто виртуальный метод из базового класса Users |
| void Print(); | sql::Statement\* stmt;  sql::ResultSet\* res;  sql::Connection\* con;  try  {  con = ConectDB();  stmt = con->createStatement();  res = stmt->executeQuery("SELECT \* FROM PartStoreWorkers");  cout << "ID\tName\tExperience\tNumber" << endl;  cout << "---------------------------------" << endl;  while (res->next())  {  cout << res->getInt("id") << "\t";  cout << res->getString("name") << "\t\t";  cout << res->getInt("experience") << "\t";  cout << res->getString("number") << endl;  }  delete res;  delete stmt;  delete con;  }  catch (sql::SQLException e)  {  cout << "Cannot retrieve data. Error message: " << e.what() << endl;  system("pause");  } | Метод для вывода информации обо всех работниках магазина запчастей. Реализует чисто виртуальный метод из базового класса Users |

**Листинг** **класса**

#pragma once

#include "Connect.h"

#include "User.h"

#include <iostream>

#include <vector>

class PartStoreWorker : public Users

{

private:

vector<string> number;

public:

void Insert(string name, int experience, string number);

void Update(int id, string name, int experience, string number);

void Delete(int id);

void Print();

};

Примечание: в последующих описании методов классов «Код методов» в таблицах будет опускаться, если там будут методы для создания, обновления, удаления и просмотра записей.

## **4.4. Класс Country**

**Назначение** **класса**

Класс Country предназначен для управления информацией о странах-производителях, связанных с автомобильным каталогом.

**Описание** **полей**

|  |  |
| --- | --- |
| Наименование | Назначение |
| vector<string> name; | Вектор, хранящий названия стран. |

**Описание** **методов**

|  |  |
| --- | --- |
| Наименование | Назначение |
| void Insert(string name); | Метод для добавления новой страны в каталог. Принимает название страны в качестве аргумента и добавляет его в вектор name. |
| void Update(int id, string name); | Метод для обновления названия страны. |
| void Delete(int id); | Метод для удаления страны из каталога. |
| void Print(); | Метод для вывода всех стран из каталога. |

**Листинг** **класса**

#pragma once

#include "Connect.h"

#include <iostream>

#include <vector>

#include <string>

using namespace std;

class Country

{

private:

vector<string> name;

public:

void Insert(string name);

void Update(int id, string name);

void Delete(int id);

void Print();

};

## **4.5. Класс CarBrand**

**Назначение** **класса**

Класс CarBrand предназначен для управления данными о марках автомобилей в системе автомобильного каталога. Он предоставляет методы для добавления, обновления, удаления и вывода информации о марках автомобилей и связанных с ними странах-производителях.

**Описание** **полей**

|  |  |
| --- | --- |
| Наименование | Назначение |
| vector<string> names; | Вектор, хранящий названия марок автомобилей. |
| vector<int> id\_country; | Вектор, хранящий идентификаторы стран, в которых производятся марки автомобилей. |

**Описание** **методов**

|  |  |
| --- | --- |
| Наименование | Назначение |
| void Insert(string name, int id\_country); | Метод для добавления новой марки автомобиля в базу данных. Проверяет существование указанной страны перед добавлением. |
| void Update(int id, string name, int id\_country); | Метод для обновления информации о марке автомобиля по её идентификатору. Проверяет существование указанной страны перед обновлением. |
| void Delete(int id); | Метод для удаления марки автомобиля по её идентификатору из базы данных. |
| void Print(); | Метод для вывода всех марок автомобилей из базы данных с указанием их идентификаторов, названий и стран-производителей. |

**Листинг** **класса**

#pragma once

#include "Connect.h"

#include <iostream>

#include <vector>

#include <string>

using namespace std;

class CarBrand

{

private:

vector<string> names;

vector<int> id\_country;

public:

void Insert(string name, int id\_country);

void Update(int id, string name, int id\_country);

void Delete(int id);

void Print();

};

## **4.6. Класс Manufacturer**

**Назначение** **класса**

Класс Manufacturer предназначен для управления данными о производителях автомобилей в системе автомобильного каталога.

**Описание** **полей**

|  |  |
| --- | --- |
| Наименование | Назначение |
| vector<string> names; | Вектор, хранящий названия производителей. |
| vector<int> id\_country; | Вектор, хранящий идентификаторы стран, в которых находятся производители. |
| vector<string> date; | Вектор, хранящий даты регистрации производителей. |

**Описание** **методов**

|  |  |
| --- | --- |
| Наименование | Назначение |
| void Insert(string name, int id\_country, string date); | Метод для добавления нового производителя в базу данных. Проверяет существование указанной страны перед добавлением. |
| void Update(int id, string name, int id\_country, string date); | Метод для обновления информации о производителе по его идентификатору. Проверяет существование указанной страны перед обновлением. |
| void Delete(int id); | Метод для удаления производителя по его идентификатору из базы данных. |
| void Print(); | Метод для вывода всех производителей из базы данных с указанием их идентификаторов, названий, стран-производителей и дат регистрации. |

**Листинг** **класса**

#pragma once

#include "Connect.h"

#include <iostream>

#include <vector>

#include <string>

using namespace std;

class Manufacturer

{

private:

vector<string> names;

vector<int> id\_country;

vector<string> date;

public:

void Insert(string name, int id\_country, string date);

void Update(int id, string name, int id\_country, string date);

void Delete(int id);

void Print();

};

## **4.7. Класс Part**

**Назначение класса**

Класс Part предназначен для управления данными о запчастях в системе автомобильного каталога. Он предоставляет методы для добавления, обновления, удаления и вывода информации о запчастях, включая их названия, идентификаторы связанных работников склада запчастей и даты.

**Описание** **полей**

|  |  |
| --- | --- |
| Наименование | Назначение |
| vector<string> names; | Вектор, хранящий названия деталей. |
| vector<int> id\_PSW; | Вектор, хранящий идентификаторы работников склада запчастей, связанных с деталями. |
| vector<string> date; | Вектор, хранящий дату производства, связанные с деталями. |

**Описание** **методов**

|  |  |
| --- | --- |
| Наименование | Назначение |
| void Insert(string name, int id\_PSW, string date); | Метод для добавления новой детали в базу данных. |
| void Update(int id, string name, int id\_PSW, string date); | Метод для обновления информации о деталях по её идентификатору. |
| void Delete(int id); | Метод для удаления детали. |
| void Print(); | Метод для вывода всех запчастей из базы данных. |

**Листинг** **класса**

#pragma once

#include "Connect.h"

#include <iostream>

#include <vector>

#include <string>

using namespace std;

class Part

{

private:

vector<string> names;

vector<int> id\_PSW;

vector<string> date;

public:

void Insert(string name, int id\_PSW, string date);

void Update(int id, string name, int id\_PSW, string date);

void Delete(int id);

void Print();

};

## **4.8. Класс Component**

**Назначение** **класса**

Класс Component предназначен для управления данными о компонентах автомобилей в системе автомобильного каталога. Он предоставляет методы для добавления, обновления, удаления и вывода информации о компонентах, включая их названия, даты, связанные части и ответственных сотрудников.

**Описание** **полей**

|  |  |
| --- | --- |
| Наименование | Назначение |
| vector<string> names; | Вектор, хранящий названия компонентов. |
| vector<string> date; | Вектор, хранящий даты создания, связанные с компонентами. |
| vector<int> id\_part1; | Вектор, хранящий идентификаторы первой части, связанной с компонентом детали. |
| vector<int> id\_part2; | Вектор, хранящий идентификаторы второй части, связанной с компонентом детали. |
| vector<int> id\_SW; | Вектор, хранящий идентификаторы сотрудников, связанных с компонентом. |

**Описание** **методов**

|  |  |
| --- | --- |
| Наименование | Назначение |
| void Insert(string name, string date, int id\_part1, int id\_part2, int id\_SW); | Метод для добавления нового компонента в базу данных. Принимает название компонента, дату, идентификаторы двух связанных частей и идентификатор сотрудника. |
| void Update(int id, string name, string date, int id\_part1, int id\_part2, int id\_SW); | Метод для обновления информации о компоненте по его идентификатору. |
| void Delete(int id); | Метод для удаления компонента по его идентификатору из базы данных. |
| void Print(); | Метод для вывода всех компонентов из базы данных с указанием их идентификаторов, названий, дат, связанных частей и ответственных сотрудников. |

**Листинг** **класса**

#pragma once

#include "Connect.h"

#include <iostream>

#include <vector>

#include <string>

using namespace std;

class Component

{

private:

vector<string> names;

vector<string> date;

vector<int> id\_part1;

vector<int> id\_part2;

vector<int> id\_SW;

public:

void Insert(string name, string date, int id\_part1, int id\_part2, int id\_SW);

void Update(int id, string name, string date, int id\_part1, int id\_part2, int id\_SW);

void Delete(int id);

void Print();

};

## **4.9. Класс Assembly**

**Назначение** **класса**

Класс Assembly предназначен для управления данными о сборках (агрегатов) автомобилей в системе автомобильного каталога.

**Описание** **полей**

|  |  |
| --- | --- |
| Наименование | Назначение |
| vector<string> names; | Вектор, хранящий названия сборок автомобилей. |
| vector<string> date; | Вектор, хранящий даты сборок автомобилей. |
| vector<int> id\_SW; | Вектор, хранящий идентификаторы служащих, ответственных за сборку. |

**Описание методов**

|  |  |
| --- | --- |
| Наименование | Назначение |
| void Insert(string name, string date, int id\_SW); | Метод для добавления новой записи о сборке автомобиля в базу данных. |
| void Update(int id, string name, string date, int id\_SW); | Метод для обновления информации о сборке автомобиля по её идентификатору. |
| void Delete(int id); | Метод для удаления записи о сборке автомобиля по её идентификатору из базы данных. |
| void Print(); | Метод для вывода всех записей о сборках автомобилей из базы данных. |

**Листинг** **класса**

#pragma once

#include "Connect.h"

#include <iostream>

#include <vector>

#include <string>

using namespace std;

class Assembly

{

private:

vector<string> names;

vector<string> date;

vector<int> id\_SW;

public:

void Insert(string name, string date, int id\_SW);

void Update(int id, string name, string date, int id\_SW);

void Delete(int id);

void Print();

};

## **4.10. Класс MakeAssembly**

**Назначение** **класса**

Класс MakeAssembly предназначен для управления данными о сборках автомобилей в системе автомобильного каталога. Он предоставляет методы для добавления, обновления, удаления и вывода информации о сборках, включая их компоненты и количество.

**Описание** **полей**

|  |  |
| --- | --- |
| Наименование | Назначение |
| vector<int> id\_nameAssembly; | Вектор, хранящий идентификаторы сборок автомобилей. |
| vector<int> id\_part; | Вектор, хранящий идентификаторы деталей, входящих в сборку. |
| vector<int> quantity\_part; | Вектор, хранящий количество каждой детали в сборке. |
| vector<int> id\_component; | Вектор, хранящий идентификаторы компонентов, входящих в сборку. |
| vector<int> quantity\_component; | Вектор, хранящий количество каждого компонента в сборке. |

**Описание** **методов**

|  |  |
| --- | --- |
| Наименование | Назначение |
| void Insert(int id\_nameAssembly, int id\_part, int quantity\_part, int id\_component, int quantity\_component); | Метод для добавления новой записи о сборке автомобиля в базу данных. |
| void Update(int id, int id\_nameAssembly, int id\_part, int quantity\_part, int id\_component, int quantity\_component); | Метод для обновления информации о существующей записи о сборке автомобиля в базе данных. |
| void Delete(int id); | Метод для удаления записи о сборке автомобиля из базы данных по её идентификатору. |
| void Print(); | Метод для вывода информации о всех сборках автомобилей, включая их запчасти и компоненты, из базы данных. |

**Листинг** **класса**

#pragma once

#include "Connect.h"

#include <iostream>

#include <vector>

#include <string>

using namespace std;

class MakeAssembly

{

private:

vector<int> id\_nameAssembly;

vector<int> id\_part;

vector<int> quantity\_part;

vector<int> id\_component;

vector<int> quantity\_component;

public:

void Insert(int id\_nameAssembly, int id\_part, int quantity\_part, int id\_component, int quantity\_component);

void Update(int id, int id\_nameAssembly, int id\_part, int quantity\_part, int id\_component, int quantity\_component);

void Delete(int id);

void Print();

};

## **4.11. Класс History**

**Назначение** **класса**

Класс History предназначен для работы с историей замен агрегатов в автомобилях в системе автомобильного каталога, то есть идет учет взаимозаменяемости. Он предоставляет методы для вывода информации обо всех заменах агрегатов, а также о заменах, совершенных для конкретного автомобиля или в определенную дату.

**Описание** **методов**

|  |  |  |
| --- | --- | --- |
| Наименование | Код методов | Назначение |
| void Print(int id\_car); | sql::PreparedStatement\* pstmt;  sql::ResultSet\* result;  sql::Connection\* con;  try {  con = ConectDB();  pstmt = con->prepareStatement("SELECT Histories.id, Histories.date, NewAssemblies.name AS New,OldAssemblies.name AS Old \  FROM Histories \  INNER JOIN Assemblies AS NewAssemblies ON Histories.id\_assembly\_new = NewAssemblies.id \  INNER JOIN Assemblies AS OldAssemblies ON Histories.id\_assembly\_old = OldAssemblies.id \  WHERE Histories.id\_car = ?;");  pstmt->setInt(1, id\_car);  result = pstmt->executeQuery();  cout << "ID\t Date \t New \t Old" << endl;  cout << "-----------------" << endl;  while (result->next())  { cout << result->getInt("id") << "\t" << result->getString("date") << "\t" << result->getString("New")<< "\t" << result->getString("Old") << endl; }  delete result; delete pstmt; delete con;}  catch (sql::SQLException e)  {  cout << "Cannot retrieve data. Error message: " << e.what() << endl;  system("pause");  } | Метод для вывода информации о заменах агрегатов для конкретного автомобиля по его идентификатору. Выводит идентификаторы замен, даты, а также названия новых и старых агрегатов. |
| void Print(string date); | sql::PreparedStatement\* pstmt;  sql::ResultSet\* result;  sql::Connection\* con;  try  { con = ConectDB();  pstmt = con->prepareStatement("SELECT Histories.id, Cars.name, NewAssemblies.name AS New, OldAssemblies.name AS Old \  FROM Histories\  INNER JOIN Assemblies AS NewAssemblies ON Histories.id\_assembly\_new = NewAssemblies.id\  INNER JOIN Assemblies AS OldAssemblies ON Histories.id\_assembly\_old = OldAssemblies.id\  INNER JOIN Cars ON Histories.id\_assembly\_old = Cars.id\  WHERE Histories.date = ?;");  pstmt->setString(1, date);  result = pstmt->executeQuery();  cout << "ID\t Car \t New \t Old" << endl;  cout << "-----------------" << endl;  while (result->next())  { cout << result->getInt("id") << "\t" << result->getString("name") << "\t" << result->getString("New")<< "\t" << result->getString("Old") << endl; }  delete result; delete pstmt; delete con;}  catch (sql::SQLException e)  { cout << "Cannot retrieve data. Error message: "<< e.what() << endl; system("pause");} | Метод для вывода информации о заменах агрегатов, совершенных в определенную дату. |
| void Print(); | sql::PreparedStatement\* pstmt;  sql::ResultSet\* result; sql::Connection\* con;  try { con = ConectDB();  pstmt = con->prepareStatement("SELECT Histories.id, Histories.date, Cars.name as Cars , NewAssemblies.name AS New,OldAssemblies.name AS Old \  FROM Histories \  INNER JOIN Assemblies AS NewAssemblies ON Histories.id\_assembly\_new = NewAssemblies.id \  INNER JOIN Assemblies AS OldAssemblies ON Histories.id\_assembly\_old = OldAssemblies.id\  INNER JOIN Cars ON Histories.id\_car = Cars.id ;");  result = pstmt->executeQuery();  cout << "ID\t Date \t Cars\t New \t Old" << endl; cout << "-----------------" << endl;  while (result->next())  { cout << result->getInt("id") << "\t" << result->getString("date") << "\t" << result->getString("Cars")<< "\t" << result->getString("New") << "\t" << result->getString("Old") << endl; }  delete result; delete pstmt; delete con;}  catch (sql::SQLException e) {  cout << "Cannot retrieve data. Error message: " << e.what() << endl; system("pause"); } | Метод для вывода информации обо всех заменах агрегатов, включая идентификаторы, даты, названия автомобилей и заменяемых агрегатов. |

**Листинг** **класса**

#pragma once

#include<iostream>

#include<vector>

#include"Connect.h"

class History

{

public:

void Print(int id\_car);

void Print(string date);

void Print();

};

## **4.12. Класс Car**

**Назначение** **класса**

Класс Car предназначен для управления данными о автомобилях в системе автомобильного каталога. Он предоставляет методы для добавления, обновления, удаления и вывода информации о автомобилях, а также для поиска автомобилей по их идентификатору или названию.

**Описание** **полей**

|  |  |
| --- | --- |
| Наименование | Назначение |
| vector<string> names; | Вектор, хранящий названия автомобилей. |
| vector<int> id\_carBrand; | Вектор, хранящий идентификаторы марок автомобилей. |
| vector<int> id\_manufacturer; | Вектор, хранящий идентификаторы производителей автомобилей. |
| vector<int> id\_SW; | Вектор, хранящий идентификаторы служащих, ответственных за обслуживание автомобилей. |
| vector<string> date; | Вектор, хранящий даты добавления информации об автомобилях. |

**Описание** **методов**

|  |  |  |
| --- | --- | --- |
| Наименование | Код методов | Назначение |
| template<typename T>  void Search(T value, const string& columnName); | try {  sql::Connection\* con = ConectDB();  sql::PreparedStatement\* pstmt = con->prepareStatement(  "SELECT Cars.id, Cars.name, CarBrands.name as carBrand\_name, Manufacturers.name as manufacturer\_name, ServiceWorkers.name as serviceWorker\_name, Cars.date "  "FROM Cars "  "INNER JOIN CarBrands ON Cars.id\_carBrand = CarBrands.id "  "INNER JOIN Manufacturers ON Cars.id\_manufacturer = Manufacturers.id "  "INNER JOIN ServiceWorkers ON Cars.id\_SW = ServiceWorkers.id "  "WHERE Cars." + columnName + "=?");  if constexpr (is\_same<T, int>::value) {  pstmt->setInt(1, value); }  else if constexpr (is\_same<T, string>::value) {  pstmt->setString(1, value); }  sql::ResultSet\* res = pstmt->executeQuery(); bool found = false;  while (res->next()) {  found = true;  cout << "ID: " << res->getInt("id") << endl;  cout <<"Name:"<<res->getString("name") << endl;  cout<<"CarBrand:"<<res>getString("carBrand\_name")<<end;  cout<<"Manufacturer:"<<res>getString("manufacturer\_name") << endl;cout <<"ServiceWorker:"<< res->getString("serviceWorker\_name")<< endl;  cout << "Date: " << res->getString("date") << endl;  cout << "---------------------------------" << endl; }  if (!found) {  cout << "No cars found with the " << columnName << " " << value << "." << endl;  }  delete res; delete pstmt; delete con; }  catch (sql::SQLException e) {  cout << "Cannot retrieve data. Error message: " << e.what() << endl;  system("pause"); } | Шаблонный метод для поиска автомобилей по определенному значению и имени столбца в базе данных. Он принимает на вход значение и тип данных, а также имя столбца, по которому производится поиск. Это обеспечивает гибкость в поиске автомобилей по различным критериям без необходимости создания отдельных методов для каждого типа данных. |
| void Insert(string name, int id\_carBrand, int id\_manufacturer, int id\_SW, string date); | try {  sql::Connection\* con = ConectDB();  sql::PreparedStatement\* pstmt = con->prepareStatement("INSERT INTO Cars (name, id\_carBrand, id\_manufacturer, id\_SW, date) VALUES(?, ?, ?, ?, ?)");  pstmt->setString(1, name);  pstmt->setInt(2, id\_carBrand);  pstmt->setInt(3, id\_manufacturer);  pstmt->setInt(4, id\_SW);  pstmt->setString(5, date);  pstmt->execute();  cout << "Saved" << endl;  delete pstmt; delete con; }  catch (sql::SQLException e) { cout << "Cannot Insert. Error message: " << e.what() << endl;  system("pause"); } | Метод для добавления нового автомобиля в базу данных. |
| void Update(int id, string name, int id\_carBrand, int id\_manufacturer, int id\_SW, string date); | try {  sql::Connection\* con = ConectDB();  sql::PreparedStatement\* pstmt = con->prepareStatement("UPDATE Cars SET name = ?, id\_carBrand = ?, id\_manufacturer = ?, id\_SW = ?, date = ? WHERE id = ?");  pstmt->setString(1, name);  pstmt->setInt(2, id\_carBrand);  pstmt->setInt(3, id\_manufacturer);  pstmt->setInt(4, id\_SW);  pstmt->setString(5, date);  pstmt->setInt(6, id);  pstmt->execute();  cout << "Updated" << endl;  delete pstmt;  delete con;} catch (sql::SQLException e) {  cout << "Cannot Update. Error message: " << e.what() << endl;  system("pause");  } | Метод для обновления информации о существующем автомобиле по его идентификатору. |
| void Delete(int id); | try {  sql::Connection\* con = ConectDB();  sql::PreparedStatement\* pstmt = con->prepareStatement("DELETE FROM Cars WHERE id = ?");  pstmt->setInt(1, id);  pstmt->execute();  cout << "Deleted" << endl;  delete pstmt;  delete con;  }  catch (sql::SQLException e) {  cout << "Cannot Delete. Error message: " << e.what() << endl;  system("pause");  } | Метод для удаления информации о автомобиле по его идентификатору. |
| void Print(); | try {  sql::Connection\* con = ConectDB();  sql::Statement\* stmt = con->createStatement();  sql::ResultSet\* res = stmt->executeQuery(  "SELECT Cars.id, Cars.name, CarBrands.name as carBrand\_name, Manufacturers.name as manufacturer\_name, ServiceWorkers.name as serviceWorker\_name, Cars.date "  "FROM Cars "  "INNER JOIN CarBrands ON Cars.id\_carBrand = CarBrands.id "  "INNER JOIN Manufacturers ON Cars.id\_manufacturer = Manufacturers.id "  "INNER JOIN ServiceWorkers ON Cars.id\_SW = ServiceWorkers.id");  cout << "ID\tName\tCarBrand\tManufacturer\tServiceWorker\tDate" << endl; cout << "---------------------------------------------------------------------" << endl;  while (res->next())  { cout << res->getInt("id") << "\t";  cout << res->getString("name") << "\t";  cout << res->getString("carBrand\_name") << "\t\t";  cout << res->getString("manufacturer\_name") << "\t\t";  cout << res->getString("serviceWorker\_name") << "\t\t";  cout << res->getString("date") << endl; }  delete res; delete stmt; delete con; }  catch (sql::SQLException e) {  cout << "Cannot retrieve data. Error message: " << e.what() << endl;  system("pause"); } | Метод для вывода информации о всех автомобилях в системе, включая название, марку, производителя, сервисного работника и дату. |
| void SearchById(int id); | Search(id, "id"); | Метод для поиска автомобиля по его идентификатору и вывода информации о нем. |
| void SearchByName(string name); | Search(name, "name"); | Метод для поиска автомобиля по его названию и вывода информации о нем. |

**Листинг** **класса**

#pragma once

#include "Connect.h"

#include <iostream>

#include <vector>

#include <string>

using namespace std;

class Car

{

private:

vector<string> names;

vector<int> id\_carBrand;

vector<int> id\_manufacturer;

vector<int> id\_SW;

vector<string> date;

template<typename T>

void Search(T value, const string& columnName);

public:

void Insert(string name, int id\_carBrand, int id\_manufacturer, int id\_SW, string date);

void Update(int id, string name, int id\_carBrand, int id\_manufacturer, int id\_SW, string date);

void Delete(int id);

void Print();

void SearchById(int id);

void SearchByName(string name);

};

## **4.13. Класс MakeCar**

**Назначение** **класса**

Класс MakeCar предназначен для управления данными о производстве автомобилей в системе автомобильного каталога. Он обеспечивает возможность добавления, обновления, удаления и вывода информации о производстве конкретных автомобилей и их комплектующих.

**Описание** **методов**

|  |  |
| --- | --- |
| Наименование | Назначение |
| void Print(int id\_car); | Метод для вывода информации об агрегатах, используемых для производства конкретного автомобиля по его идентификатору. |
| void Print(float id\_assem); | Метод для вывода информации о всех автомобилях, производство которых использует определенный агрегат по её идентификатору. |
| void Insert(int id\_car, int id\_assem,int quantity); | Метод для добавления новой записи о производстве определенного автомобиля с указанным количеством агрегатов. |
| void Update(int id, int id\_assem,int quantity); | Метод для обновления информации о производстве автомобиля по его идентификатору, включая изменение используемых агрегаторов и их количество. Также выполняет запись в историю, если агрегат был изменен.  sql::PreparedStatement\* pstmt;  sql::Connection\* con;  sql::ResultSet\* result;  con = ConectDB();  int old\_id\_assem;  int id\_car;  pstmt = con->prepareStatement("Select id\_assembly ,id\_car from MakeCars where id=?");  pstmt->setInt(1, id);  result = pstmt->executeQuery();  if (result->next())  { old\_id\_assem = result->getInt("id\_assembly");  id\_car = result->getInt("id\_car");  cout << old\_id\_assem << " " << id\_assem; }  pstmt = con->prepareStatement("UPDATE MakeCars SET id\_assembly=?,quantity=? WHERE id = ?");  pstmt->setInt(1, id\_assem);  pstmt->setInt(2, quantity);  pstmt->setInt(3, id);  pstmt->execute();  cout << "Updated" << endl;  if (old\_id\_assem != id\_assem) {  pstmt = con->prepareStatement("INSERT INTO Histories(date, id\_assembly\_new, id\_assembly\_old, id\_car) VALUES (NOW(), ?, ?, ?)");  pstmt->setInt(1, id\_assem);  pstmt->setInt(2, old\_id\_assem);  pstmt->setInt(3, id\_car);  pstmt->execute(); }  delete pstmt; delete con; delete result; |
| void Delete(int id); | Метод для удаления информации о производстве автомобиля по его идентификатору. |

1. **Подключение базы данных к проекту**

В проекте используется реляционная база данных MySQL. Выбор обусловлен тем, что MySQL является одной из самых производительных и масштабируемых серверных СУБД. Она поддерживает широкий набор функциональных возможностей, соответствующих всем основным концепциям реляционных баз данных, и предоставляет надежное и устойчивое решение для управления данными. MySQL легко интегрируется с различными языками программирования, включая C++, что делает её идеальным выбором для этого проекта.

В данном проекте, где используется язык C++, существуют удобные библиотеки и драйверы для подключения к MySQL, такие как MySQL Connector/C++.

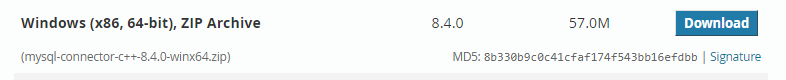
Пошаговый процесс подключения базы данных MySQL к проекту:

Шаг. 1. Скачивание драйвера MySQL Connector/C++.

Для начало переходим по ссылке для скачивания с официального сайта:

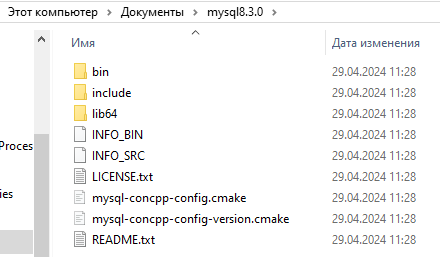
<https://dev.mysql.com/downloads/connector/cpp/>

далее скачиваем ZIP файл, который показан на рисунке



Шаг. 2. Распаковка архива

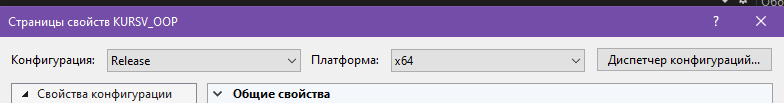
После успешного скачивания необходимо разархивировать папку, после этого в папке появятся файлы, которые показаны на рисунке



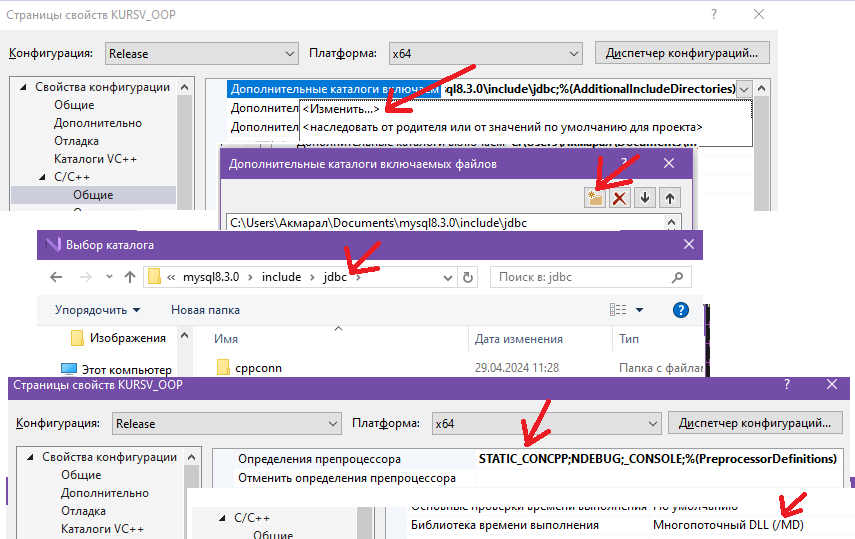
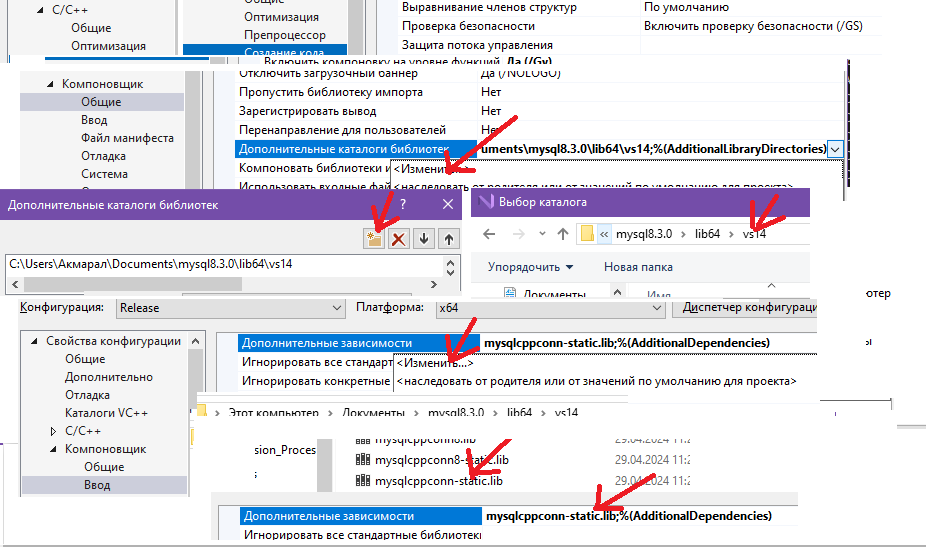
Далее заходим в проект, открываем окно свойств проекта

Шаг. 3. Подключение файлов к проекту

Конфигурация должна быть Release, а платформа x64



Потом подключаем дополнительные каталоги как показано на рисунке, следуем всем красным стрелкам, которые показаны на рисунке



После этих действий необходимо пересобрать проект и создать класс Connect и написать код, который показан ниже:

#pragma once

#include <stdlib.h>

#include <iostream>

#include "mysql\_connection.h"

#include <cppconn/driver.h>

#include <cppconn/exception.h>

#include <cppconn/prepared\_statement.h>

using namespace std;

const string server = "tcp://127.0.0.1";

const string username = "root";

const string password = "";

const string DB = "car\_business";

sql::Connection\* ConectDB();

#include "Connect.h"

sql::Driver\* driver;

sql::Connection\* con;

sql::ConnectOptionsMap connection\_properties;

sql::Connection\* ConectDB()

{ connection\_properties["hostName"] = server;

connection\_properties["userName"] = username;

connection\_properties["password"] = password;

connection\_properties["schema"] = DB;

connection\_properties["port"] = 3307;

try {

driver = get\_driver\_instance();

con = driver->connect(connection\_properties);

return con; }

catch (sql::SQLException e)

{ cout << "Could not connect to server. Error message: " << e.what() << endl; system("pause");

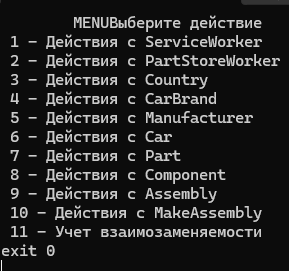
exit(1); }}

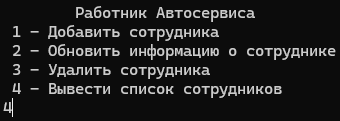
После этого в MySQL создадим базу данных car\_business и его таблицы:

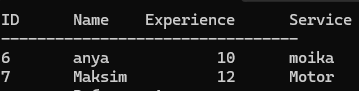


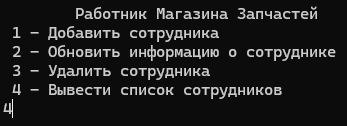
1. **Скриншоты работы программы**

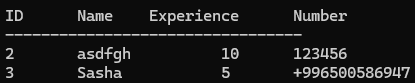
Главное окно программы со списком всех доступных функций

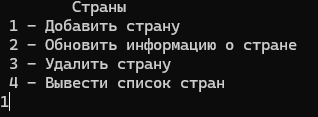
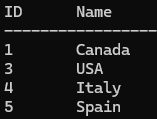


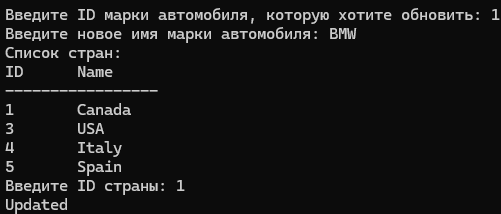
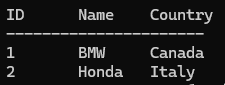
1 – действие

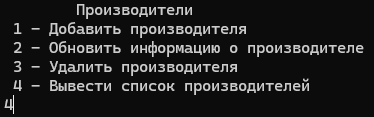


2 – действие

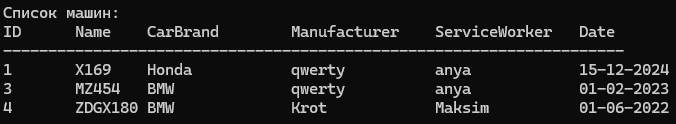
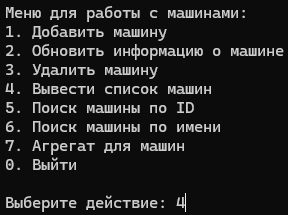


3 – действие

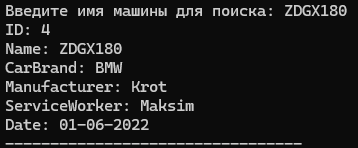
4 – действие

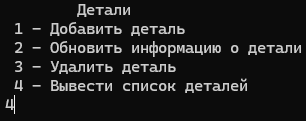
5 – действие

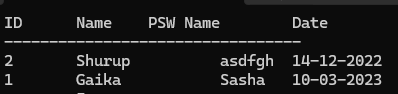
6 – действие

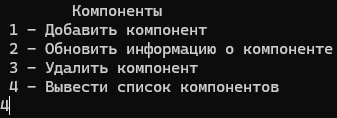
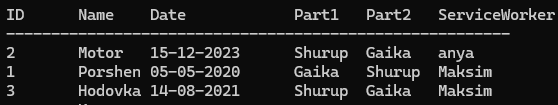


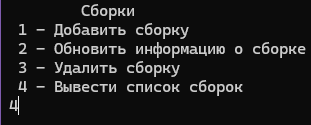
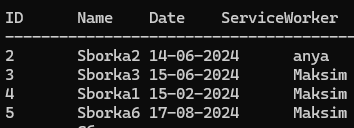
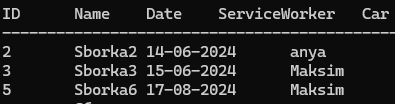


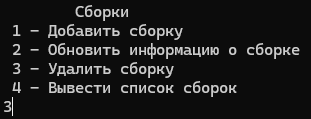


7 – действие

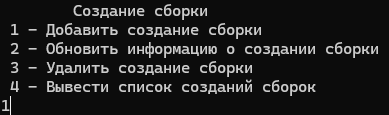


8 – действие

9 – действие

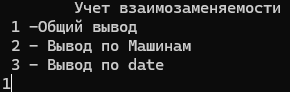


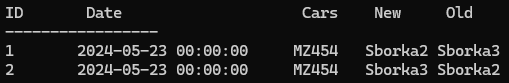
10 – действие





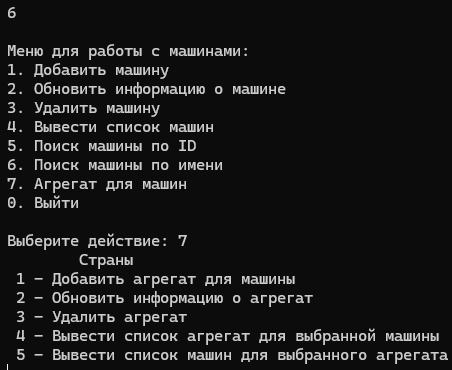
11 – действие

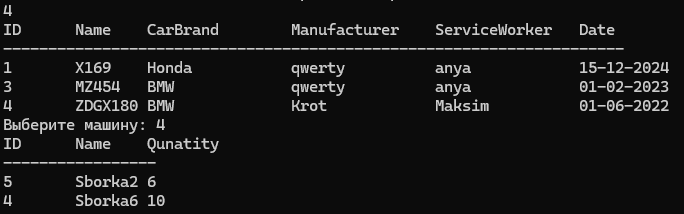




В эту таблицу данные будут добавляться после изменения Агрегата(сборки) определенной машины на другой Агрегат(сборку), продемонстрируем это:

Выбираем действие 6

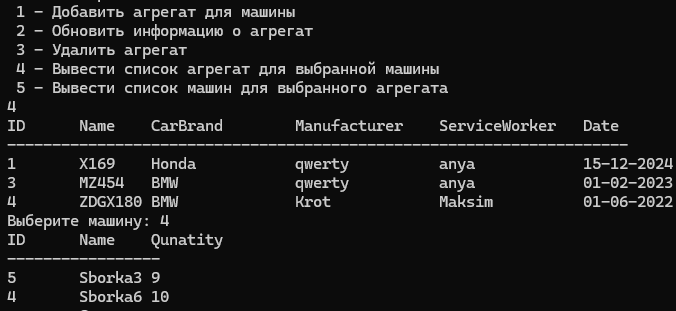
Выбираем действие 4, для того чтобы узнать в выбранном машине какие сборки(агрегаты) есть



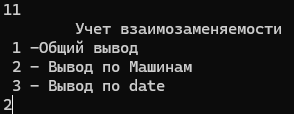
Теперь изменим один из агрегатов на другую, для этого нажимаем на действие 2,

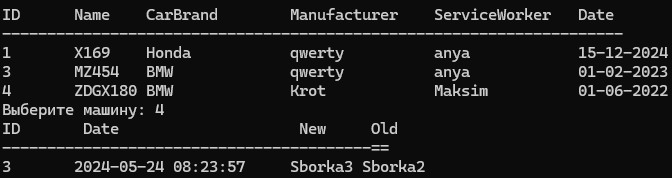


Здесь мы меняем Sborka2 на Sborka3



Теперь проверим сохранились ли эти изменения в таблице





Как видно из рисунка учет о взаимозаменяемости агрегатов совершилось.