ТЕХНІЧНИЙ КОЛЕДЖ ТЕРНОПІЛЬСЬКОГО НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ ІМЕНІ ІВАНА ПУЛЮЯ

Циклова комісія програмних систем і комплексів

**КУРСОВА РОБОТА**

з дисципліни:

**«Об’єктно-орієнтоване програмування»**

на тему: **«Програма автоматизації роботи готельного комплексу»**

Студента  3  курсу групи КН-321 спеціальності 122 «Комп’ютерні науки»

Ярмош І.О.

(прізвище та ініціали)

Керівник: викладач   Слободян  Р.О.         
Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_ Оцінка: ECTS \_\_\_\_\_\_

Члени комісії: \_\_\_\_\_\_\_\_\_\_\_ Р.О. Слободян

(підпис)

\_\_\_\_\_\_\_\_\_ Г.Я. Марціяш

(підпис)

м. Тернопіль – 2019

ЗМІСТ

[ВСТУП 5](#_Toc26434668)

[1 ТЕХНІЧНЕ ЗАВДАННЯ 7](#_Toc26434669)

[1.1 Найменування та область застосування 7](#_Toc26434670)

[1.2 Підстави для розробки 7](#_Toc26434671)

[1.3 Призначення розробки 7](#_Toc26434672)

[1.4 Вимоги до програми чи програмного виробу 8](#_Toc26434673)

[1.5 Вимоги до програмної документації 9](#_Toc26434674)

[1.6 Техніко-економічні показники 10](#_Toc26434675)

[1.7 Стадії та етапи розробки 10](#_Toc26434676)

[1.8 Порядок контролю та прийому 11](#_Toc26434677)

[2 РОЗРОБКА ТЕХНІЧНОГО ТА РОБОЧОГО ПРОЄКТУ 13](#_Toc26434678)

[2.1 Розробка загальної структури і варіантів використання програми 13](#_Toc26434679)

[2.2 Розробка системи класів 16](#_Toc26434680)

[2.3 Розробка методів 17](#_Toc26434681)

[2.4 Проєктування і опис інтерфейсу користувача 19](#_Toc26434682)

[2.5 Опис файлової структури програми 24](#_Toc26434683)

[2.6 Опис структури бази даних програми 24](#_Toc26434684)

[3 ТЕСТУВАННЯ ПРОГРАМИ І РЕЗУЛЬТАТИ ЇЇ ВИКОНАННЯ 26](#_Toc26434685)

[ВИСНОВКИ 30](#_Toc26434686)

[ПЕРЕЛІК ПОСИЛАНЬ 31](#_Toc26434687)

[Додаток А Лістинг файлу «main.cpp» 32](#_Toc26434688)

[Додаток Б Лістинг файлу «mainwindow.h» 33](#_Toc26434689)

[Додаток В Лістинг файлу «mainwindow.cpp» 34](#_Toc26434690)

[Додаток Г Лістинг файлу «addnewrow.h» 35](#_Toc26434691)

Додаток Д Лістинг файлу «addnewrow.cpp» 36

Додаток Е Лістинг файлу «deleterow.h» 38

Додаток Ж Лістинг файлу «deleterow.cpp» 39

Додаток И Лістинг файлу «secondwindow.h» 42

Додаток К Лістинг файлу «secondwindow. cpp» 43

Додаток Л Лістинг файлу «viewalldata.h» 47

Додаток М Лістинг файлу «viewalldata.cpp» 48

Додаток Н DVD-диск із програмним продуктом 50

ВСТУП

Протягом останніх декількох десятиліть комп'ютерні технології розвивались дуже швидко. Мови програмування також потерпіли значної еволюції. Поява більш потужних комп'ютерів дала життя більш об'ємним і складним програмам, які, водночас, висвітлювали нові проблеми в області керування програмами, а також їх подальшому супроводу.

Об'єктно-орієнтоване програмування сягає своїм корінням до створення мови програмування в 1960-тих роках, одночасно з посиленням дискусій про [кризу програмного забезпечення](https://uk.wikipedia.org/w/index.php?title=%D0%9A%D1%80%D0%B8%D0%B7%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F&action=edit&redlink=1). Разом із тим, як ускладнювалось апаратне та програмне забезпечення, було дуже важко зберегти якість програм. Об'єктно-орієнтоване програмування частково розв'язує цю проблему шляхом наголошення на модульності програми.

На відміну від традиційних поглядів, коли програму розглядали як набір [підпрограм](https://uk.wikipedia.org/wiki/%D0%9F%D1%96%D0%B4%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%B0), або як перелік [інструкцій](https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%86%D1%96%D1%8F) комп'ютеру, ООП програми можна вважати сукупністю об'єктів. Відповідно до парадигми об'єктно-орієнтованого програмування, кожен об'єкт здатний отримувати [повідомлення](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BC%D1%96%D0%BD_%D0%BF%D0%BE%D0%B2%D1%96%D0%B4%D0%BE%D0%BC%D0%BB%D0%B5%D0%BD%D0%BD%D1%8F%D0%BC%D0%B8), обробляти дані, та надсилати повідомлення іншим об'єктам. Кожен об'єкт — своєрідний незалежний автомат з окремим призначенням та відповідальністю.

Сучасна економіка привела до перегляду об’ємів витрат на комп’ютерну техніку та на спеціалізоване програмне забезпечення. Набагато краще і вигідніше стало розробляти прості програмні засоби, орієнтовані на розв’язання вузько спеціалізованих завдань. Тому, перед навчальними закладами постала мета підготувати спеціалістів, які спроможні створювати гнучкі програмні засоби, не витрачаючи на їх розробку та впровадження багато фінансів та часу.

Як приклад, в даній курсовій роботі представлено розробку спеціалізованого програмного рішення мовою С++ із використанням бази даних та принципів об’єктно-орієнтованого програмування. Ціль його розробки – автоматизація роботи готельного комплексу, що значно полегшує їхню роботу.

Об'єктно-орієнтоване програмування - це парадигма програмування, заснована на поданні програми у вигляді сукупності взаємодіючих об'єктів, кожен з яких є екземпляром певного класу, а класи є членами певної ієрархії наслідування. Спочатку розробляється клас, а на його основі при виконанні програми створюються конкретні об'єкти (екземпляри класів). На основі класів можна створювати нові, які розширюють базовий клас і таким чином створюється ієрархія класів. Це дозволяє швидко конструювати достатньо функціональний програмний засіб.

# ТЕХНІЧНЕ ЗАВДАННЯ

## Найменування та область застосування

Найменування програми – «Програма автоматизації роботи готельного комплексу».

Стисла назва – «HotelH».

Область застосування програми – обслуговування клієнтів за допомогою автоматизованої програми для роботи з інформацією у готелі .

## Підстави для розробки

Підставами для проведення розробки являється індивідуальне завдання на курсову роботу з дисципліни «Об’єктно-орієнтоване програмування».

Найменування теми курсової роботи – «Програма автоматизації роботи готельного комплексу».

Виконавець – студент групи КН-321 Ярмош Ігор Олегович.

## Призначення розробки

Підвищення ефективності і полегшення роботи працівників готельного комплексу.

Відповідно функціональне призначення – спрощення і автоматизація роботи готельного комплексую

**1.4Вимоги до програми чи програмного виробу**

### 1.4.1Вимоги до функціональних характеристик

Програма автоматизації роботи готельного комплексу повинна автоматизувати роботу готелю,покращити облік кімнат і клієнтів які знаходяться в ньому. Для цього потрібно авторизуватися адміністратору та ввести дані,які потрібні для ведення обліку.

### 1.4.2Вимоги до надійності

Надійне функціонування програми повинне забезпечуватись наявністю блоку перевірки правильності введених даних. Це збереже від помилок, що виникають при виконанні рутинної роботи користувачем.

Також необхідно забезпечити достатню стійкість програми до некваліфікованих дій користувача, що досягається супроводом будь-яких операцій в програмі чіткими і зрозумілими повідомленнями.

### 1.4.3 Умови до експлуатації

Розроблений програмний продукт не потребуватиме яких-небудь видів обслуговування, для його використання достатньо буде навиків користувача ПК.

### 1.4.4 Вимоги до інформаційної і програмної сумісності

В склад технічних засобів повинні входити: персональний комп’ютер із такими мінімальними характеристиками:

− Процесор Pentium III;

− ОЗУ: 128Mb, або вище;

− обсяг дискової пам’яті – 80 ГБ.

− монітор;

− клавіатура;

− мишка.

### 1.4.5 Вимоги до інформаційної і програмної сумісності

Вихідні коди програми повинні бути реалізовані мовою С++. В якості середовища розробки програми повинне використовуватись середовище Qt Creator 4.8.0.

Системні програмні засоби і утиліти, які буде використовувати програма, повинні забезпечуватись операційною системою сімейства Windows версії не нижче Windows 7, Linux або MacOS оскільки середовище Qt є кросс-платформне.

Вимоги до захисту інформації і програми не пред’являються.

## 1.5 Вимоги до програмної документації

По закінченню розробки програмного забезпечення потрібно підготувати таку документацію:

- інструкція інсталяції програми;

- загальні відомості про можливості програми;

- інструкція з експлуатації.

## 1.6 Техніко-економічні показники

Розрахунок економічної ефективності і вартості розробки програмного продукту не проводиться.

Приблизне число використань розробленої програми в рік – 500-1500 раз

## 1.7 Стадії та етапи розробки

В загальному випадку, життєвий цикл визначається моделлю й описується у формі методології (методу). Модель або парадигма життєвого циклу визначає загальну організацію і, як правило, основні його фази та принципи переходу між ними. Методологія (метод) визначає комплекс робіт, їх детальний зміст і рольову відповідальність спеціалістів на всіх етапах вибраної моделі.

Життєвий цикл програмного забезпечення супроводжується розробленням, обігом та використанням програмної документації.

Програмна документація — сукупність документів, що містять відомості, необхідні для розробки, виготовлення, супроводу та експлуатації програм. Програмна документація є одним з видів технічної документації.

Комплекс державних стандартів, що встановлюють взаємопов'язані правила розробки, оформлення та обігу програм і програмної документації називається «Єдина система програмної документації» .

При розробці даного програмного забезпечення буде використовуватись спіральна модель життєвого циклу.

Розробка ітераціями відображає об'єктивно існуючий спіральний цикл створення системи. Неповне завершення робіт на кожному етапі дозволяє переходити на наступний етап, не чекаючи повного завершення роботи на поточному. При ітеративному способі розробки відсутню роботу можна буде виконати на наступній ітерації. Головне ж завдання - щонайшвидше показати користувачам системи працездатний продукт, тим самим активізуючи процес уточнення і доповнення вимог.

Виходячи з можливості внесення змін, як в процес, так і в проміжний продукт було створено спіральну модель ЖЦ .

Внесення змін орієнтоване на задоволення потреби користувачів одразу, як тільки буде встановлено, що створені артефакти або елементи документації не відповідають дійсному стану розробки.

Дана модель ЖЦ допускає аналіз продукту на витку розробки, його перевірку, оцінку правильності та прийняття рішення про перехід на наступний виток або повернення на попередній виток для доопрацювання на ньому проміжного продукту.

Відмінність цієї моделі від каскадної полягає в можливості багато разів повертатися до процесу формулювання вимог і до повторної розробки версії системи з будь-якого процесу моделі.

Для програмного продукту така модель не дуже підходить з декількох причин. По-перше, висловлення вимог замовником носить суб'єктивний характер, вимоги можуть багаторазово уточнюватися протягом розробки ПС і навіть після завершення та випробовування, і часом може з'ясуватися, що замовник «хотів зовсім інше». По-друге, змінюються обставини та умови використання системи, тому загальновизнаним законом програмної інженерії є закон еволюції, який сформулюємо так: кожна діюча ПС з часом потребує внесення змін або виводиться з експлуатації.

При необхідності внесення змін до системи на кожному витку з метою отримання нової версії системи обов'язково вносяться зміни в заздалегідь зафіксовані вимоги, після чого повертаються на попередній виток спіралі для продовження реалізації нової версії системи з урахуванням усіх змін.

## 1.8 Порядок контролю та прийому

Прийом розробленого програмного забезпечення повинен відбуватися на об’єкті Замовника в терміни, які зазначені в індивідуальному завданні.

Для прийому роботи Виконавець повинен представити:

− діючу програму, яка повністю відповідає даному технічному завданню;

− вихідний програмний код, записаний разом із програмою на оптичний носій інформації.

Прийом програмного забезпечення повинен відбуватися перед комісією з двох чоловік (один з яких – Замовник) у такій послідовності:

− доповідь Виконавця про виконану роботу;

− демонстрація Виконавцем роботи програми;

− контрольні випробовування роботи програми;

− відповіді на запитання і зауваження комісії.

# РОЗРОБКА ТЕХНІЧНОГО ТА РОБОЧОГО ПРОЄКТУ

## Розробка загальної структури і варіантів використання програми

Під час проходження етапу аналізу спочатку було визначено такі питання:

* Хто буде взаємодіяти з програмою?
* Хто вводитиме інформацію?
* Хто буде редагувати інформацію?
* Чи буде ще хто-небудь взаємодіяти з програмою?

В даному випадку з програмою HotelH має працювати тільки один користувач – адміністратор готельного комплексу. Таким чином, один і той самий користувач вводить інформацію, і переглядає її у різних видах.

Наступне, що потрібно було з'ясувати, це групу дій, які ініціюватиме діючий суб'єкт. Виходячи із технічного завдання, діючий суб'єкт – адміністратор буде виконувати такі дії:

– почати роботу з програмою;

* авторизуватися;

– додати запис ;

– видалити запис;

– переглянути базу даних;

– відкрити базу даних;

– пошук даних;

Таким чином, діаграму варіантів використання, яка отримується внаслідок наведеного переліку дій, можна подати так, як це показано на рисунку 2.1.



Рисунок 2.1 – Діаграма варіантів використання для програми HotelH.

**Запустити програму**. Ця дія, здавалося б, є дуже очевидною для того, щоб про неї зовсім не згадувати, але все ж таки... Коли запускається програма, на екран повинно виводитися меню, з якого користувач може вибрати потрібну дію. Це може називатися екраном інтерфейсу користувача.

**Авторизація.** На екрані має відобразитися форма авторизації в якій адміністратор повинен підтвердити свою особу,після чого зможе продовжити роботу з самою програмою.

**Додати запис.** На екрані має відобразитися форма,у якій програма добавляє запис в вибрану таблицю (форма заповнення залежить від вибраної таблиці) та дві кнопки :

* добавити;
* закрити вікно;

**Видалити запис.** На екрані має відобразитися форма,у якій програма видаляє запис з вибраної таблиці та поле у якому вказується номер запису за 2 кнопки :

* видалити;
* закрити;

**Переглянути базу даних.** На екрані має відобразитися форма,у якій програма виводить дані з вибраної таблиці та кнопка:

* закрити;

**Відкрити базу даних.** На екрані має відобразитися вікно програми Провідник у якій потрібно вибрати файл з розширенням “.db” для перегляду інформації з обраної бази даних,а саме з вибраних стовбців і таблиць.

**Пошук даних.** На екрані є форма,у якій ми можемо записувати які дані ми хочемо побачити,але пошук виконується з вибраної таблиці і її стовбця.

## Розробка системи класів

Наступний етап розробки програмного забезпечення – етап проектування. Даний етап поділяється на логічне та фізичне проектування. Логічне полягає у розробці структури класів, коли визначаються поля для збереження складових об'єктів, алгоритми методів, що описують поведінку об'єктів. Насамперед необхідно передбачити класи, з яких буде складатися програма. Для цього спочатку проаналізуємо перелік іменників з опису варіантів використання, потім внесемо деякі їх уточнення, визначимо їх атрибути, перейдемо від дієслів до повідомлень, і, на завершення, розробимо діаграми класів і послідовностей.

**Аналіз переліку іменників з опису варіантів використання**. Розглянемо перелік усіх іменників, які візьмемо з опису варіантів використання:

1. База даних.
2. Вікно авторизації
3. Головне вікно.
4. Вікно додати рядок.
5. Вікно видалити рядок.
6. Вікно переглянути інформацію .

**Уточнення переліку іменників.** З різних причин багато іменників не зможуть стати класами. Давайте проведемо відбір тільки тих іменників, які можуть претендувати на те, щоб бути класом.

Одним із таких претендентів може бути клас “База даних”, який буде містити підключення до сервера на якому буде зберігатись вся інформація в таблицях: таблиця користувачі, таблиця кімнати . Наступні 5 класів це класи вікон, які містять інформацію про усі функції вікон і їх можливості. Отже, складемо перелік класів, які було тільки, що уточнено:

1. База даних.
2. Вікно авторизації
3. Головне вікно.
4. Вікно додати рядок.
5. Вікно видалити рядок.
6. Вікно переглянути інформацію .

**Визначення** **атрибутів**. Багато іменників, яким відмовлено в реєстрації як кандидатами у класи, будуть потенційними кандидатами в атрибути (компонентні дані) класів.

**Перехід від дієслів до повідомлень.** Тепер подивимося, що нам дають варіанти використання для з'ясування того, якими повідомленнями обмінюватимуться класи. Оскільки повідомлення – це, по суті справи, є викликом методу в об'єкті, то визначення повідомлень зводиться до визначення методів класу, що приймає те або інше повідомлення. Як і у випадку з іменниками, далеко не кожне дієслово стає кандидатом у повідомлення.

## Розробка методів

Етап починається із створення основного інтерфейсу проекту, бібліотеки класів, описуються класи, розробляються методи, уточнюються вимоги до програми та вносяться певні зміни. Одержаний прототип продукту тестується й налагоджується. В програмі використовуються класи та методи Qt Framework[3], які суттєво полегшили роботу з організацією роботи усіх складових програми.

Як уже було зазначено вище, багато атрибутів для кожного з класів є похідними з тих іменників, які самі не стали класами. Інші атрибути можуть бути виведені з асоціацій в діаграмі класів.

**Основні методи.** Такими методами є методи роботи з базою даних, інформацією яку хоче отримати користувач. Наприклад метод connect створює з’єднання з базою даних через яке інший метод такий як get може працювати з базою даних беручи з відти інформацію, яку потрібно побачити та додати іншу інформацію.

Визначивши усі необхідні атрибути і методи класів, а також допоміжні класи, можна з допомогою програми StarUML побудувати повну діаграму класів. Розроблена діаграма класів зображена на рисунку 2.2.

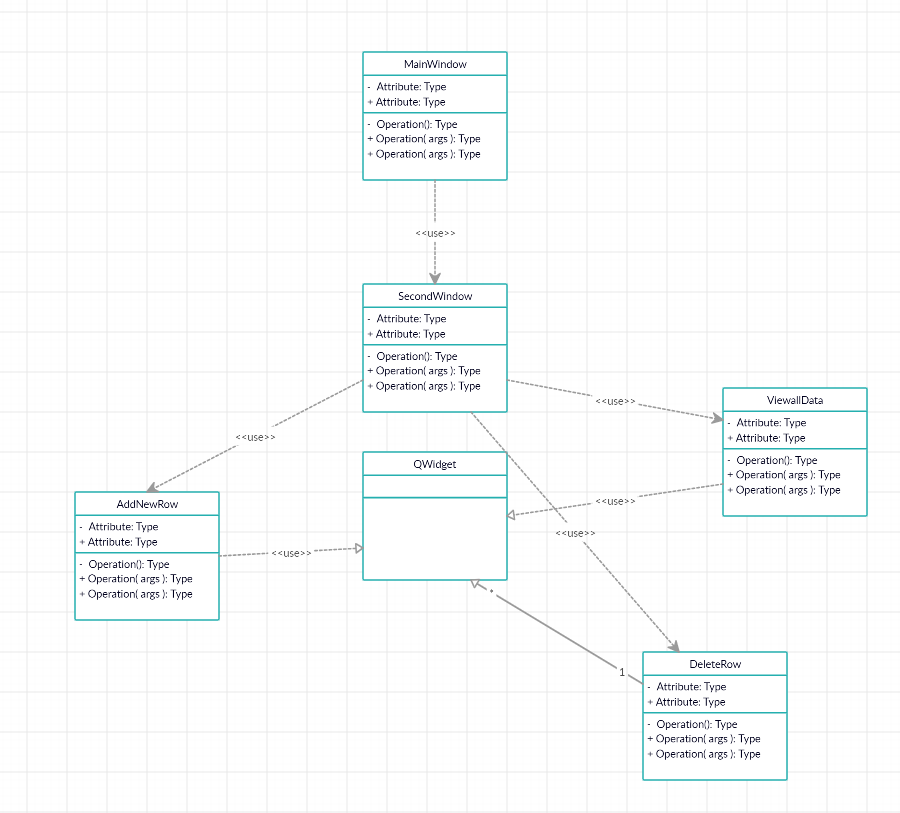


Рисунок 2.2 – UML-діаграма класів програми HotelH

Маючи діаграми варіантів використання, детальні їх описи, діаграми класів, а також попередні плани щодо створення коду програми, можна приступити до написання коду програми.

Написання коду програми починається із написання заголовного (\*.h) файлу у якому необхідно визначити тільки інтерфейсні частини класів, але не подробиці їх реалізації. Оголошення в заголовному файлі – це загальнодоступна частина класів. Тіла функцій, які є розташовані в \*.срр-файлах, називаються реалізацією і користувачам (стороннім програмістам) недоступні.

Написання заголовного файлу – це проміжний крок між етапами плануванням і звичайним кодуванням методів.

Більшість оголошень появляються безпосередньо з класів, створених за допомогою взятих з описів варіантів використання іменників, і відображаються на діаграмі класів.

Написання коду програми розглянутих методів має починатися тільки на даному етапі розроблення ПЗ і ні кроком раніше, тому що тільки зараз відомі ім'я кожного класу чи методу, їх призначення і навіть, можливо, передбачити аргументи, що їм передаються.

## Проєктування і опис інтерфейсу користувача

Інтерфейс користувача (ІК) - це сукупність засобів, за допомогою яких користувач взаємодіє з різними пристроями (з комп’ютером або побутовою технікою) або іншим складним інструментарієм (системою). Інтерфейс користувача - це такий різновид інтерфейсів, в якому з одного боку - людина, з іншого - машина (пристрій, програмне забезпечення). За визначенням Національного банку стандартизованих науково-технічних термінів, інтерфейс користувача - це комплекс апаратних і програмних засобів, що забезпечує взаємодію користувача з комп'ютером .

ІК часто розуміють лише як зовнішній вигляд програмного забезпечення (ПЗ), але таке розуміння є надто вузьким, оскільки саме за допомогою інтерфейсу користувач сприймає програму в цілому та використовує її функціональність. ІК забезпечує підтримку прийняття рішень у визначеній предметній галузі та визначає порядок використання ПЗ і документації до нього. В дійсності, ІК об'єднує усі елементи і компоненти ПЗ, які здатні впливати на взаємодію користувача з програмним забезпеченням . До таких елементів належать: набір задач, які користувач розв’язує за допомогою ПЗ; використовувана програмним забезпеченням метафора (наприклад, "робочий стіл" у операційній системі Windows); елементи управління ПЗ; навігація між блоками ПЗ; візуальний (і не тільки) дизайн вікон та екранних форм програми та інші складові .

При створенні ПЗ використовуватиметься середовище Qt Designer[2]. Основні класи, які використовуватимуться для роботи з вікнами: QMainWindow, QWidget, QMessagebox,QSqlTableModel.

Також при оформлені та створенні вікна будуть використовуватись такі елементи Qt Designer:

* LineEdit – поле для вводу;
* Label – поле для показу тексту;
* TableView – інструмент для виводу таблиці на екран;
* PushButton – кнопки;
* DateEdit – для спрощення вводу дати;
* DateTimeEdit – для спрощення вводу дати і часу;
* Layot – інтрумент для створення шарів на формі (для полегшення роботи з елементами форми);
* GroupBox – впорядковує елементи по групам.

Переглянути як ці елементи працюють можна в 3 розділі даної записки.

При оформлені та створенні вікна “Додати запис” використовуватись такі елементи Qt Designer: Label, PushButton, LineEdit, які зображені на рисунку 2.3.

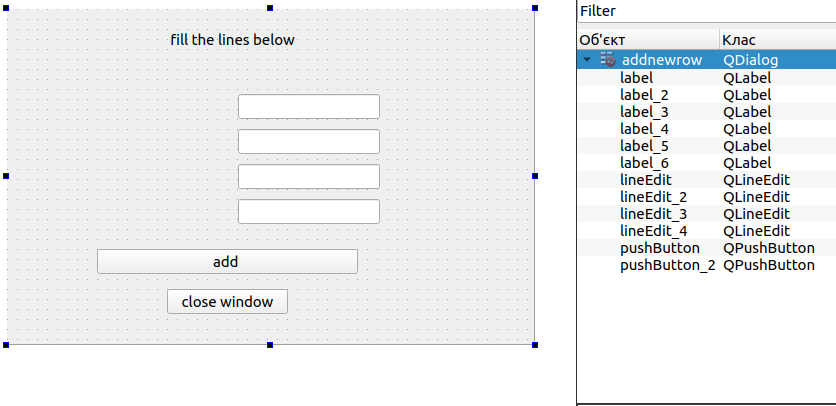


Рисунок 2.3 – Вікно “Додати запис”

При оформлені та створенні вікна “Видалити запис” використовуватись такі елементи Qt Designer: TableView, Label, PushButton, LineEdit, які зображені на рисунку 2.4.

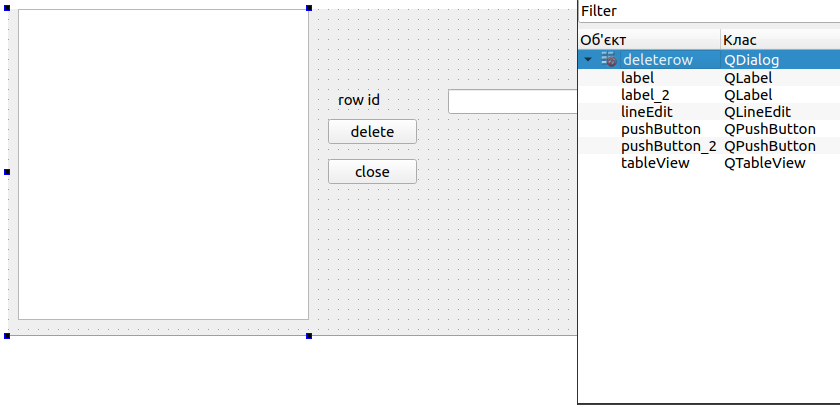


Рисунок 2.4 – Вікно “Видалити запис ”

При оформлені та створенні вікна “Авторизація ” використовуватись такі елементи Qt Designer: LineEdit, GroupBox, Label, PushButton, LineEdit,MenuBar,

StatusBar,які зображені на рисунку 2.5.

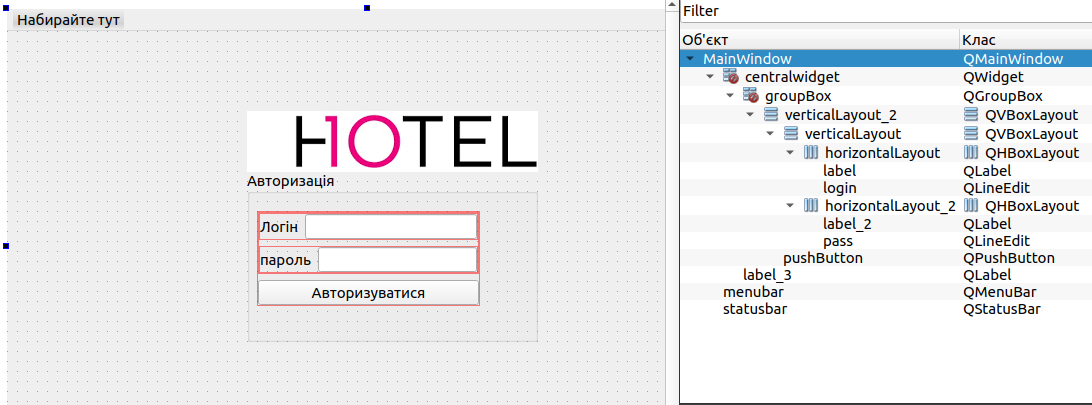


Рисунок 2.5 – Вікно “Авторизація”

При оформлені та створенні вікна “Головне вікно” використовуватись такі елементи Qt Designer: LineEdit, Label, PushButton, LineEdit, ListWidget,ComboBox, які зображені на рисунку 2.6.

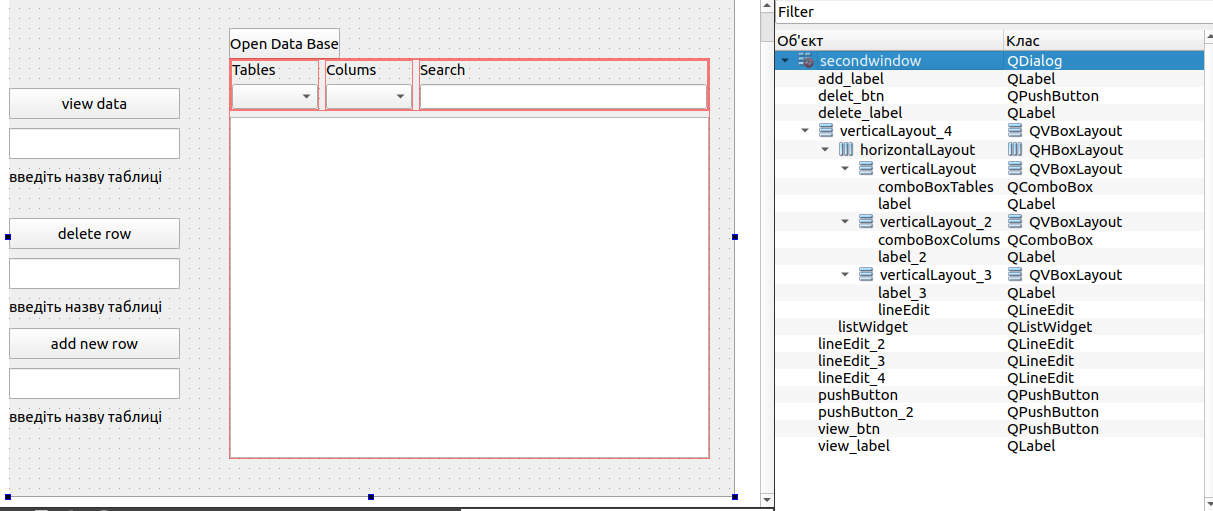


Рисунок 2.6 – Вікно “Головне вікно”

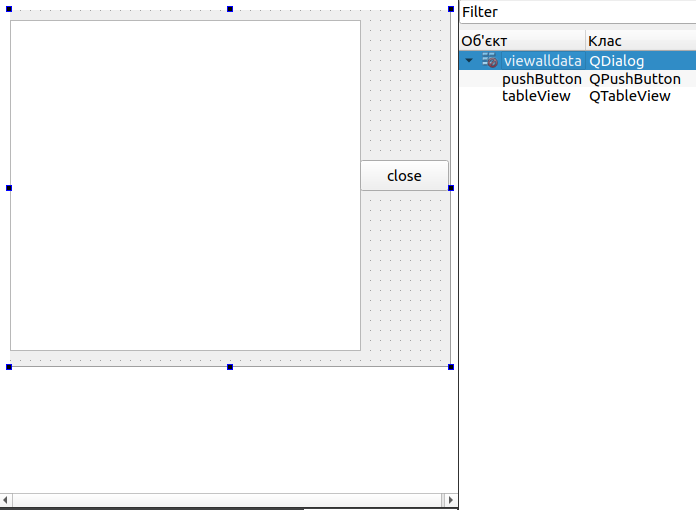
При оформлені та створенні вікна “Переглянути інформацію” використовувались такі елементи Qt Designer:PushButton, TableWiew, які зображені на рисунку 2.7.

Рисунок 2.7 – Вікно “Переглянути інформацію”

## Опис файлової структури програми

Програма складається із таких файлів:

* «main.cpp» -лістинг даного файла знаходиться в додатку А. В цьому файлі міститься головна функція програми ;
* «mainwindow.h» - лістинг даного файла знаходиться в додатку Б;
* «mainwindow.cpp» - лістинг даного файла знаходиться в додатку В;
* «addnewrow.h» - лістинг даного файла знаходиться в додатку Г;
* «addnewrow.cpp» - лістинг даного файла знаходиться в додатку Д;
* «deleterow.h» - лістинг даного файла знаходиться в додатку Е;
* «deleterow.cpp» - лістинг даного файла знаходиться в додатку Ж;
* «secondwindow.h» - лістинг даного файла знаходиться в додатку И;
* «secondwindow.cpp» - лістинг даного файла знаходиться в додатку К;
* «viewalldata.h» - лістинг даного файла знаходиться в додатку Л;
* «viewalldata.cpp» - лістинг даного файла знаходиться в додатку М:
* DVD-диск із програмним продуктом знаходиться в додатку H.

## Опис структури бази даних програми

База даних складається із шести таблиць, які зображено на ER-діаграмі рисунку 2.8. В якості системи управління бази даних було використано SQLite — полегшена [реляційна система керування базами даних](https://uk.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D1%96%D0%B9%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D0%BA%D0%B5%D1%80%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B1%D0%B0%D0%B7%D0%B0%D0%BC%D0%B8_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85).

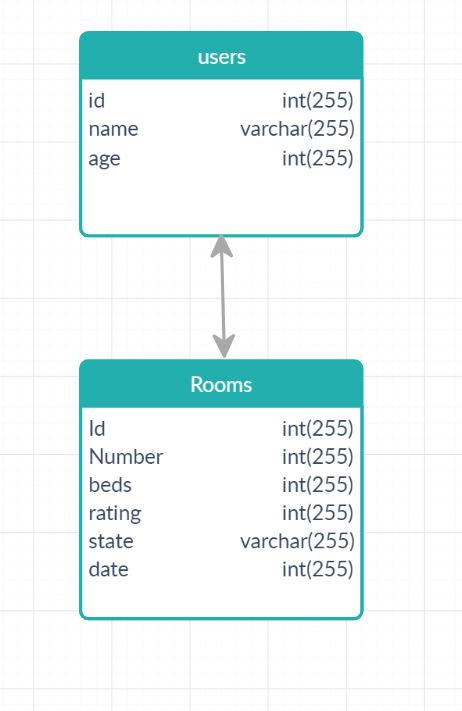


Рисунок 2.8 – ER-діаграма бази даних

# ТЕСТУВАННЯ ПРОГРАМИ І РЕЗУЛЬТАТИ ЇЇ ВИКОНАННЯ

В ході тестування програми було виявлено і виправлено незначні помилки. Запустивши програму на екрані монітора відображається вікно авторизації представлене на рисунку 3.1.

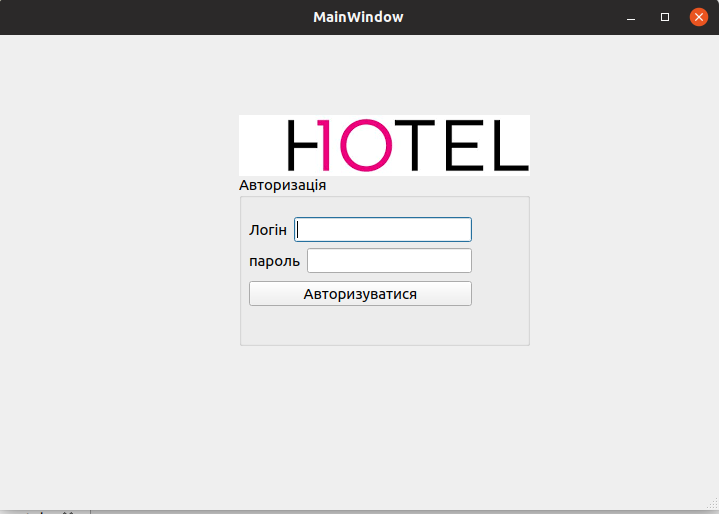


Рисунок 3.1 – Вікно авторизації програми “HotelH”

У цьому вікні бачимо 2 поля :

− Логін;

− пароль;

Та одно кнопку:

− Авторизуватися;

Ввівши правильний логін і пароль зможемо перейти в наступне вікно зображене на рисунку 3.2,яке є головним вікном програми.

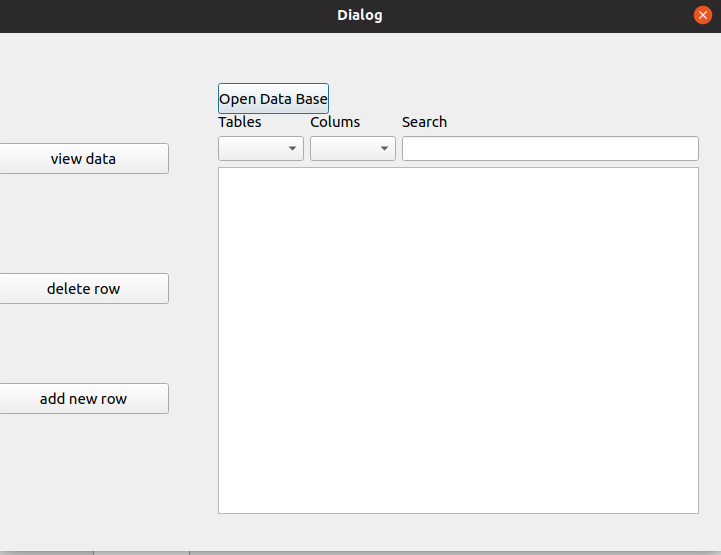


Рисунок 3.2 – Головне вікно програми

Нажавши на кнопку «add new row(Додати новий запис)» і вибравши таблицю з якою ми хочемо працювати ми зможемо перейти в вікно, яке зображене на рисунку 3.3.

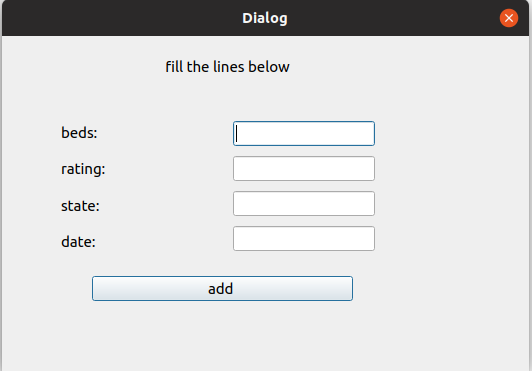


Рисунок 3.3 – Вікно добавлення записів в вибрану таблицю

Нажавши на кнопку «delete row(видалити рядок)» ми зможемо перейти в вікно, яке зображене на рисунку 3.4.

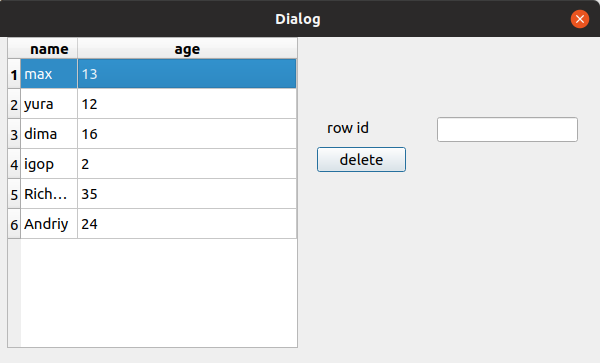


Рисунок 3.4 – Видалення запису з вибраної таблиці

Нажавши на кнопку «view data(перегляд даних)» ми зможемо перейти в вікно, яке зображене на рисунку 3.5.

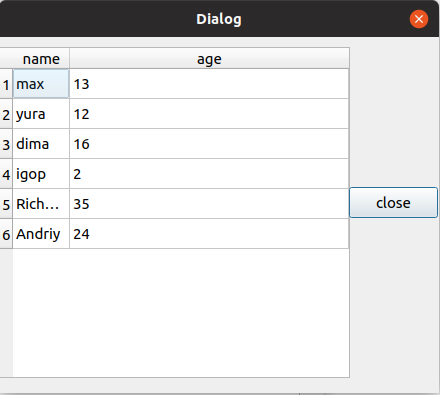


Рисунок 3.5 – Перегляд даних з вибраної таблиці

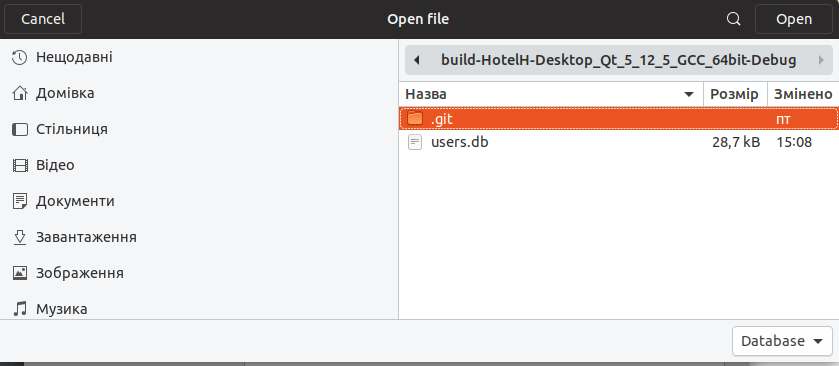
Нажавши на кнопку «Open data base(відкрити базу даних)» ми зможемо перейти в вікно провідника, яке зображене на рисунку 3.6.

Рисунок 3.6 – Вибір бази даних для перегляду інформації

ВИСНОВКИ

В даній курсовій роботі було описано етапи розробки програми HotelH, яка призначена для автоматизації роботи готельного комплексу. Розроблена програма відповідає усім поставленим вимогам технічного завдання. Під час виконання курсової роботи було закріплено і поглиблено знання, одержанні за час вивчення предмету «Об’єктно-орієнтоване програмування», більш детально вивчено мову програмування С++. Для розробки даного програмного забезпечення використовувались принципи і концепції об’єктно-орієнтованого програмування. Використане середовище програмування Qt , його особливі можливості і переваги, що дозволило швидко конструювати програмний засіб. Також під час розробки використано стандартні бібліотеки Qt, що дозволило в значній мірі пришвидшити розробку даної програми.

ПЕРЕЛІК ПОСИЛАНЬ

1. Грицюк Ю.І. Рак Т.Є. Об’єктно-орієнтоване програмування мовою С++ : навч. посіб. Львів : Вид-во Львівського ДУ БЖД, 2011. 404 с.
2. Вайсфельд М. Объектно-ориентированное мышление. СПб. : Питер, 2014. 304 с.
3. Build with Qt. URL: https://www1.qt.io/built-with-qt (дата звернення: 15.12.2019).
4. EVILEG : вебсайт. URL: https://evileg.com/uk/knowledge/qt (дата звернення: 15.12.2019).

Додаток А  
Лістинг файлу «main.cpp»

|  |
| --- |
| #include "mainwindow.h" |
|  |  |
|  | #include <QApplication> |
|  |  |
|  | int main(int argc, char \*argv[]) |
|  | { |
|  | QApplication a(argc, argv); |
|  | MainWindow w; |
|  | w.show(); |
|  | return a.exec(); |
|  | } |

Додаток Б  
Лістинг файлу «mainwindow.h»

|  |
| --- |
| #ifndef MAINWINDOW\_H |
|  | #define MAINWINDOW\_H |
|  |  |
|  | #include <QMainWindow> |
|  | #include <secondwindow.h> |
|  | #include<QtSql> |
|  | #include<QSqlQuery> |
|  |  |
|  |  |
|  |  |
|  | QT\_BEGIN\_NAMESPACE |
|  | namespace Ui { class MainWindow; } |
|  | QT\_END\_NAMESPACE |
|  |  |
|  | class MainWindow : public QMainWindow |
|  | { |
|  | Q\_OBJECT |
|  |  |
|  | public: |
|  | MainWindow(QWidget \*parent = nullptr); |
|  | ~MainWindow(); |
|  |  |
|  | private slots: |
|  | void on\_pushButton\_clicked(); |
|  |  |
|  | private: |
|  | Ui::MainWindow \*ui; |
|  | secondwindow \* window; |
|  | }; |
|  | #endif // MAINWINDOW\_H |

Додаток В  
Лістинг файлу «mainwindow.cpp»

|  |
| --- |
| #include "mainwindow.h" |
|  | #include "ui\_mainwindow.h" |
|  | #include <QMessageBox> |
|  | #include <secondwindow.h> |
|  |  |
|  | MainWindow::MainWindow(QWidget \*parent) |
|  | : QMainWindow(parent) |
|  | , ui(new Ui::MainWindow) |
|  | { |
|  | ui->setupUi(this); |
|  | } |
|  |  |
|  | MainWindow::~MainWindow() |
|  | { |
|  | delete ui; |
|  | } |
|  |  |
|  |  |
|  | void MainWindow::on\_pushButton\_clicked() |
|  | { |
|  | QString login = ui->login->text(); |
|  | QString password = ui->pass->text(); |
|  |  |
|  | if(login == "admin" && password == "test"){ |
|  | //QMessageBox::information(this,"message","YcTTiX"); Вивід повідомлення в окремому вікні |
|  | ui->statusbar->showMessage("ВИ успішно авторизувалися"); //Вивід повідомлення внизу вікна |
|  |  |
|  | hide(); |
|  | window = new secondwindow(this); |
|  | window->show(); |
|  |  |
|  | }else{ |
|  | //QMessageBox::warning(this,"oh","gg"); |
|  | ui->statusbar->showMessage("пароль або логін невірний!"); |
|  |  |
|  |  |
|  | } |
|  |  |
|  | } |

Додаток Г  
Лістинг файлу «addnewrow.h»

|  |
| --- |
| #ifndef ADDNEWROW\_H |
|  | #define ADDNEWROW\_H |
|  |  |
|  | #include <QDialog> |
|  | #include<QtSql> |
|  | #include<QSqlQuery> |
|  | #include<QFileDialog> |
|  | #include<QRegExp> |
|  | #include <QDebug> |
|  | namespace Ui { |
|  | class addnewrow; |
|  | } |
|  |  |
|  | class addnewrow : public QDialog |
|  | { |
|  | Q\_OBJECT |
|  |  |
|  | public: |
|  | explicit addnewrow(QWidget \*parent = nullptr); |
|  | void getTableName(const QString& tableName); |
|  | ~addnewrow(); |
|  |  |
|  | private slots: |
|  | void on\_pushButton\_clicked(); |
|  |  |
|  |  |
|  |  |
|  | void on\_pushButton\_2\_clicked(); |
|  |  |
|  | private: |
|  | Ui::addnewrow \*ui; |
|  | QString table; |
|  | QSqlDatabase dataBase; |
|  | }; |
|  |  |
|  | #endif // ADDNEWROW\_H |

Додаток Д  
Лістинг файлу «addnewrow.cpp»

|  |
| --- |
| #include "addnewrow.h" |
|  | #include "ui\_addnewrow.h" |
|  |  |
|  | addnewrow::addnewrow(QWidget \*parent) : |
|  | QDialog(parent), |
|  | ui(new Ui::addnewrow) |
|  | { |
|  |  |
|  | ui->setupUi(this); |
|  | ui->lineEdit->hide(); |
|  | ui->lineEdit\_2->hide(); |
|  | ui->lineEdit\_3->hide(); |
|  | ui->lineEdit\_4->hide(); |
|  | ui->pushButton\_2->hide(); |
|  | } |
|  |  |
|  | void addnewrow::getTableName(const QString &tableName) { |
|  | table = tableName; |
|  | if (tableName == "Rooms") { |
|  | ui->label->setText("beds: "); |
|  | ui->label\_2->setText("rating: "); |
|  | ui->label\_3->setText("state: "); |
|  | ui->label\_4->setText("date: "); |
|  |  |
|  | ui->lineEdit->show(); |
|  | ui->lineEdit\_2->show(); |
|  | ui->lineEdit\_3->show(); |
|  | ui->lineEdit\_4->show(); |
|  | } else if (tableName == "users") { |
|  |  |
|  | ui->label->setText("name: "); |
|  | ui->label\_2->setText("age: "); |
|  |  |
|  | ui->lineEdit->show(); |
|  | ui->lineEdit\_2->show(); |
|  | } |
|  | } |
|  |  |
|  | addnewrow::~addnewrow() |
|  | { |
|  | delete ui; |
|  | } |
|  |  |
|  |  |
|  | void addnewrow::on\_pushButton\_clicked() |
|  | { |
|  | ui->label\_6->clear(); |
|  | QString path = "/home/igop/Navchannya/Kursova/build-HotelH-Desktop\_Qt\_5\_12\_5\_GCC\_64bit-Debug/users.db"; |
|  | dataBase=QSqlDatabase::addDatabase("QSQLITE"); |
|  | dataBase.setDatabaseName(path); |
|  |  |
|  | if(!dataBase.open()){ |
|  | qDebug () << "can`t open db"; |
|  | return; |
|  | } |
|  | QSqlQuery query; |
|  |  |
|  | if (table == "users") { |
|  | QString age = ui->lineEdit\_2->text(); |
|  | QString name = ui->lineEdit->text(); |
|  |  |
|  | query.prepare("INSERT INTO users (name, age)" |
|  | "VALUES (:name, :age)"); |
|  | query.bindValue(":name", name); |
|  | query.bindValue(":age", age); |
|  | if(!query.exec()){ |
|  | qDebug() << "error insert into "; |
|  | qDebug() << query.lastError().text(); |
|  | } |
|  |  |
|  | } else if (table == "Rooms") { |
|  | QString beds = ui->lineEdit->text(); |
|  | QString rating = ui->lineEdit\_2->text(); |
|  | QString state = ui->lineEdit\_3->text(); |
|  | QString date = ui->lineEdit\_4->text(); |
|  | query.prepare("INSERT INTO Rooms (beds, rating, state, date)" |
|  | "VALUES (:beds, :rating, :state, :date)"); |
|  | query.bindValue(":beds", beds.toInt()); |
|  | query.bindValue(":rating", rating.toInt()); |
|  | query.bindValue(":state", state); |
|  | query.bindValue(":date", date); |
|  |  |
|  | if(!query.exec()){ |
|  | qDebug() << "error insert into "; |
|  | qDebug() << query.lastError().text(); |
|  | } |
|  |  |
|  | } |
|  | dataBase.close(); |
|  | ui->label\_6->setText("запис добавлено"); |
|  |  |
|  | ui->pushButton\_2->show(); |
|  | } |
|  |  |
|  |  |
|  |  |
|  | void addnewrow::on\_pushButton\_2\_clicked() |
|  | { |
|  | this->hide(); |
|  | } |

Додаток Е  
Лістинг файлу «deleterow.h»

|  |
| --- |
| #ifndef DELETEROW\_H |
|  | #define DELETEROW\_H |
|  |  |
|  | #include <QDialog> |
|  | #include<QtSql> |
|  | #include<QSqlQuery> |
|  | #include<QFileDialog> |
|  | #include<QRegExp> |
|  | #include <QDebug> |
|  | namespace Ui { |
|  | class deleterow; |
|  | } |
|  |  |
|  | class deleterow : public QDialog |
|  | { |
|  | Q\_OBJECT |
|  |  |
|  | public: |
|  | explicit deleterow(QWidget \*parent = nullptr); |
|  | void getTableName(const QString& tableName); |
|  | ~deleterow(); |
|  |  |
|  | private slots: |
|  | void on\_pushButton\_clicked(); |
|  |  |
|  | void on\_pushButton\_2\_clicked(); |
|  |  |
|  | private: |
|  | Ui::deleterow \*ui; |
|  | QString table; |
|  | QSqlDatabase dataBase; |
|  | QSqlTableModel \*model; |
|  |  |
|  | void setupModel(const QString &tableName, const QStringList &headers); |
|  | void createUI(); |
|  | }; |
|  |  |
|  | #endif // DELETEROW\_H |

Додаток Ж  
Лістинг файлу «deleterow.cpp»

|  |
| --- |
| #include "deleterow.h" |
|  | #include "ui\_deleterow.h" |
|  |  |
|  | deleterow::deleterow(QWidget \*parent) : |
|  | QDialog(parent), |
|  | ui(new Ui::deleterow) |
|  | { |
|  | ui->setupUi(this); |
|  | ui->label\_2->hide(); |
|  | ui->pushButton\_2->hide(); |
|  |  |
|  | } |
|  | void deleterow::setupModel(const QString &tableName, const QStringList &headers) { |
|  | model = new QSqlTableModel(this); |
|  | model->setTable(tableName); |
|  |  |
|  | for (int i = 0, j =0; i < model->columnCount(); i++, j++) { |
|  | model->setHeaderData(i, Qt::Horizontal, headers[j]); |
|  | } |
|  | model->setSort(0, Qt::AscendingOrder); |
|  | } |
|  |  |
|  | void deleterow::createUI() |
|  | { |
|  | ui->tableView->setModel(model); |
|  | ui->tableView->setColumnHidden(0, true); |
|  | ui->tableView->setSelectionBehavior(QAbstractItemView::SelectRows); |
|  | ui->tableView->setSelectionMode(QAbstractItemView::SingleSelection); |
|  | ui->tableView->resizeColumnsToContents(); |
|  | ui->tableView->setEditTriggers(QAbstractItemView::NoEditTriggers); |
|  | ui->tableView->horizontalHeader()->setStretchLastSection(true); |
|  |  |
|  | model->select(); |
|  | } |
|  | void deleterow::getTableName(const QString &tableName) { |
|  | table = tableName; |
|  |  |
|  | if (tableName == "users") { |
|  | dataBase = QSqlDatabase::addDatabase("QSQLITE"); |
|  | dataBase.setDatabaseName("/home/igop/Navchannya/Kursova/build-HotelH-Desktop\_Qt\_5\_12\_5\_GCC\_64bit-Debug/users.db"); |
|  | if (!dataBase.open()) { |
|  | qDebug()<<dataBase.lastError().text(); |
|  | } |
|  |  |
|  | this->setupModel("users", |
|  | QStringList() << trUtf8("id") |
|  | << trUtf8("name") |
|  | << trUtf8("age") |
|  | ); |
|  |  |
|  | this->createUI(); |
|  | dataBase.close(); |
|  | } else if (tableName == "Rooms") { |
|  | dataBase = QSqlDatabase::addDatabase("QSQLITE"); |
|  | dataBase.setDatabaseName("/home/igop/Navchannya/Kursova/build-HotelH-Desktop\_Qt\_5\_12\_5\_GCC\_64bit-Debug/users.db"); |
|  | if (!dataBase.open()) { |
|  | qDebug()<<dataBase.lastError().text(); |
|  | } |
|  |  |
|  | this->setupModel("Rooms", |
|  | QStringList() << trUtf8("id") |
|  | << trUtf8("beds") |
|  | << trUtf8("rating") |
|  | << trUtf8("state") |
|  | << trUtf8("date") |
|  | ); |
|  |  |
|  | this->createUI(); |
|  | dataBase.close(); |
|  | } |
|  |  |
|  | } |
|  |  |
|  | deleterow::~deleterow() |
|  | { |
|  | delete ui; |
|  | } |
|  |  |
|  | void deleterow::on\_pushButton\_clicked() |
|  | { |
|  | QString strId = ui->lineEdit->text(); |
|  | int id = strId.toInt(); |
|  | dataBase = QSqlDatabase::addDatabase("QSQLITE"); |
|  | dataBase.setDatabaseName("/home/igop/Navchannya/Kursova/build-HotelH-Desktop\_Qt\_5\_12\_5\_GCC\_64bit-Debug/users.db"); |
|  | if (!dataBase.open()) { |
|  | qDebug()<<dataBase.lastError().text(); |
|  | } |
|  | QSqlQuery query; |
|  | if (table == "users") { |
|  | if (!query.exec("DELETE FROM users WHERE id = '"+strId+"' ;")) { |
|  | qDebug() <<"error deleting"; |
|  | } |
|  | ui->label\_2->setText("deleting status(200)"); |
|  |  |
|  | } else if (table == "Rooms") { |
|  | if (!query.exec("DELETE FROM Rooms WHERE Number = '"+strId+"' ;")) { |
|  | qDebug() <<"error deleting"; |
|  | } |
|  | ui->label\_2->setText("deleting status(200)"); |
|  | } |
|  | dataBase.close(); |
|  | ui->pushButton\_2->show(); |
|  |  |
|  | } |
|  |  |
|  | void deleterow::on\_pushButton\_2\_clicked() |
|  | { |
|  | this->hide(); |
|  | } |

Додаток И  
Лістинг файлу «secondwindow.h»

|  |
| --- |
| #ifndef SECONDWINDOW\_H |
|  | #define SECONDWINDOW\_H |
|  |  |
|  | #include <QDialog> |
|  | #include<QtSql> |
|  | #include<QSqlQuery> |
|  | #include<QFileDialog> |
|  | #include<QRegExp> |
|  | #include <QDebug> |
|  |  |
|  | namespace Ui { |
|  | class secondwindow; |
|  | } |
|  |  |
|  | class secondwindow : public QDialog |
|  | { |
|  | Q\_OBJECT |
|  |  |
|  | QSqlDatabase dataBase; |
|  | QSqlQuery sqlQuery; |
|  | QString pathDB; |
|  | QStringList \* list; |
|  | public: |
|  | explicit secondwindow(QWidget \*parent = nullptr); |
|  | ~secondwindow(); |
|  |  |
|  | private slots: |
|  | void on\_pushButton\_clicked(); |
|  |  |
|  | void on\_comboBoxTables\_activated(int index); |
|  |  |
|  | void on\_comboBoxColums\_activated(int index); |
|  |  |
|  | void on\_lineEdit\_textChanged(const QString &arg1); |
|  |  |
|  |  |
|  | void on\_pushButton\_2\_clicked(); |
|  |  |
|  | void on\_delet\_btn\_clicked(); |
|  |  |
|  | void on\_view\_btn\_clicked(); |
|  |  |
|  | private: |
|  | Ui::secondwindow \*ui; |
|  |  |
|  | void fillingData(); |
|  | void addTables(); |
|  | void makeRequest(); |
|  | void addColumns(); |
|  | void addValues(int index); |
|  |  |
|  | void refleshLits(); |
|  |  |
|  | }; |
|  |  |
|  | #endif // SECONDWINDOW\_H |

Додаток К  
Лістинг файлу «secondwindow. cpp»

|  |
| --- |
| #include "secondwindow.h" |
|  | #include "ui\_secondwindow.h" |
|  | #include "addnewrow.h" |
|  | #include "deleterow.h" |
|  | #include "viewalldata.h" |
|  |  |
|  | secondwindow::secondwindow(QWidget \*parent) : |
|  | QDialog(parent), |
|  | ui(new Ui::secondwindow) |
|  | { |
|  | ui->setupUi(this); |
|  |  |
|  | list=new QStringList; |
|  |  |
|  | ui->lineEdit\_2->hide(); |
|  | ui->lineEdit\_3->hide(); |
|  | ui->lineEdit\_4->hide(); |
|  | ui->add\_label->hide(); |
|  | ui->delete\_label->hide(); |
|  | ui->view\_label->hide(); |
|  |  |
|  | /\* |
|  | dataBase = QSqlDatabase::addDatabase("QSQLITE"); |
|  | dataBase.setDatabaseName("users.db"); |
|  |  |
|  | if (dataBase.open()){ |
|  |  |
|  | sqlQuery=QSqlQuery(dataBase); |
|  | sqlQuery.exec("SELECT \*FROM users"); |
|  |  |
|  | //for (const QString &tableName:dataBase.tables()) |
|  | // qDebug() << tableName; |
|  |  |
|  | /\*int count = sqlQuery.record().count(); |
|  |  |
|  | for (int i =0;i<count;i++) |
|  | qDebug()<<sqlQuery.record().field(i).name();\*/ |
|  |  |
|  | //while (sqlQuery.next()) |
|  | // qDebug() << sqlQuery.value(1).toString();}\*/ |
|  |  |
|  | } |
|  |  |
|  | secondwindow::~secondwindow() |
|  | { |
|  | delete ui; |
|  | } |
|  |  |
|  | void secondwindow::on\_pushButton\_clicked() |
|  | { |
|  | pathDB=QFileDialog::getOpenFileName(this,"Open file","","Database (\*db)"); |
|  | dataBase=QSqlDatabase::addDatabase("QSQLITE"); |
|  | dataBase.setDatabaseName(pathDB); |
|  |  |
|  | if(dataBase.open()){ |
|  |  |
|  | sqlQuery=QSqlQuery(dataBase); |
|  | fillingData(); |
|  | refleshLits(); |
|  | } |
|  | } |
|  |  |
|  | void secondwindow::fillingData() |
|  | { |
|  | addTables(); |
|  | makeRequest(); |
|  | addColumns(); |
|  | addValues(0); |
|  | } |
|  |  |
|  | void secondwindow::addTables() |
|  | { |
|  | ui->comboBoxTables->clear(); |
|  |  |
|  | for (const QString &tableName:dataBase.tables()) |
|  | ui->comboBoxTables->addItem(tableName); |
|  |  |
|  | } |
|  |  |
|  | void secondwindow::makeRequest() |
|  | { |
|  |  |
|  | QString tableName=ui->comboBoxTables->currentText(); |
|  |  |
|  | if (!sqlQuery.exec("SELECT \* FROM '" + tableName + "';")) { |
|  | qDebug() << sqlQuery.lastError()<<" error selecting"; |
|  | } |
|  | } |
|  |  |
|  | void secondwindow::addColumns(){ |
|  |  |
|  | ui->comboBoxColums->clear(); |
|  | int columnCount = sqlQuery.record().count(); |
|  | for (int i=0;i<columnCount;i++) |
|  | ui->comboBoxColums->addItem(sqlQuery.record().field(i).name()); |
|  | } |
|  | void secondwindow::addValues(int index){ |
|  | ui->listWidget->clear(); |
|  |  |
|  | while(sqlQuery.next()) |
|  | ui->listWidget->addItem(sqlQuery.value(index).toString()); |
|  | } |
|  |  |
|  | void secondwindow::refleshLits(){ |
|  | list->clear(); |
|  | for (int i=0;i<ui->listWidget->count();i++) |
|  | list->append(ui->listWidget->item(i)->text()); |
|  |  |
|  |  |
|  | } |
|  | void secondwindow::on\_comboBoxTables\_activated(int index) |
|  | { |
|  | makeRequest(); |
|  | addColumns(); |
|  | addValues(0); |
|  |  |
|  | refleshLits(); |
|  | } |
|  |  |
|  | void secondwindow::on\_comboBoxColums\_activated(int index) |
|  | { |
|  | makeRequest(); |
|  | addColumns(); |
|  | addValues(index); |
|  |  |
|  | refleshLits(); |
|  | } |
|  |  |
|  | void secondwindow::on\_lineEdit\_textChanged(const QString &arg1) |
|  | { |
|  | QRegExp regExp(arg1,Qt::CaseInsensitive,QRegExp::Wildcard); |
|  | ui->listWidget->clear(); |
|  | ui->listWidget->addItems(list->filter(regExp)); |
|  | } |
|  |  |
|  |  |
|  |  |
|  | void secondwindow::on\_pushButton\_2\_clicked() |
|  | { |
|  | ui->view\_label->show(); |
|  | ui->lineEdit\_3->show(); |
|  |  |
|  | QString tableName = ui->lineEdit\_3->text(); |
|  | if (tableName != "") { |
|  | qDebug() << tableName; |
|  |  |
|  | addnewrow add; |
|  | add.setModal(true); |
|  | add.getTableName(tableName); |
|  | add.exec(); |
|  | ui->view\_label->hide(); |
|  | ui->lineEdit\_3->hide(); |
|  |  |
|  | } else if (tableName != "users" || tableName != "Rooms" || tableName != "" ) { |
|  | ui->view\_label->setText("такої таблиці не існує"); |
|  | } |
|  |  |
|  |  |
|  | } |
|  |  |
|  | void secondwindow::on\_delet\_btn\_clicked() |
|  | { |
|  | ui->delete\_label->show(); |
|  | ui->lineEdit\_2->show(); |
|  |  |
|  | QString tableName = ui->lineEdit\_2->text(); |
|  | if (tableName != "") { |
|  | qDebug() << tableName; |
|  |  |
|  | deleterow add; |
|  | add.setModal(true); |
|  | add.getTableName(tableName); |
|  | add.exec(); |
|  | ui->delete\_label->hide(); |
|  | ui->lineEdit\_2->hide(); |
|  | } else if (tableName != "users" || tableName != "Rooms" || tableName != "" ) { |
|  | ui->delete\_label->setText("такої таблиці не існує"); |
|  | } |
|  |  |
|  | } |
|  |  |
|  | void secondwindow::on\_view\_btn\_clicked() |
|  | { |
|  | ui->add\_label->show(); |
|  | ui->lineEdit\_4->show(); |
|  |  |
|  | QString tableName = ui->lineEdit\_4->text(); |
|  | if (tableName != "") { |
|  | qDebug() << tableName; |
|  |  |
|  | viewalldata add; |
|  | add.setModal(true); |
|  | add.getTableName(tableName); |
|  | add.exec(); |
|  | ui->add\_label->hide(); |
|  | ui->lineEdit\_4->hide(); |
|  | } else if (tableName != "users" || tableName != "Rooms" || tableName != "" ) { |
|  | ui->delete\_label->setText("такої таблиці не існує"); |
|  | } |
|  |  |
|  | } |

Додаток Л  
Лістинг файлу «viewalldata.h»

|  |
| --- |
| #ifndef VIEWALLDATA\_H |
|  | #define VIEWALLDATA\_H |
|  |  |
|  | #include <QDialog> |
|  | #include<QtSql> |
|  | #include<QSqlQuery> |
|  | #include<QFileDialog> |
|  | #include<QRegExp> |
|  | #include <QDebug> |
|  | namespace Ui { |
|  | class viewalldata; |
|  | } |
|  |  |
|  | class viewalldata : public QDialog |
|  | { |
|  | Q\_OBJECT |
|  |  |
|  | public: |
|  | explicit viewalldata(QWidget \*parent = nullptr); |
|  | void getTableName(const QString& tableName); |
|  | ~viewalldata(); |
|  |  |
|  | private slots: |
|  | void on\_pushButton\_clicked(); |
|  |  |
|  | private: |
|  | Ui::viewalldata \*ui; |
|  | QString table; |
|  | QSqlDatabase dataBase; |
|  | QSqlTableModel \*model; |
|  | void setupModel(const QString &tableName, const QStringList &headers); |
|  | void createUI(); |
|  | }; |
|  |  |
|  | #endif // VIEWALLDATA\_H |

Додаток М  
Лістинг файлу «viewalldata.cpp»

|  |
| --- |
| #include "viewalldata.h" |
|  | #include "ui\_viewalldata.h" |
|  |  |
|  | viewalldata::viewalldata(QWidget \*parent) : |
|  | QDialog(parent), |
|  | ui(new Ui::viewalldata) |
|  | { |
|  | ui->setupUi(this); |
|  | } |
|  |  |
|  | viewalldata::~viewalldata() |
|  | { |
|  | delete ui; |
|  | } |
|  | void viewalldata::setupModel(const QString &tableName, const QStringList &headers) { |
|  | model = new QSqlTableModel(this); |
|  | model->setTable(tableName); |
|  |  |
|  | for (int i = 0, j =0; i < model->columnCount(); i++, j++) { |
|  | model->setHeaderData(i, Qt::Horizontal, headers[j]); |
|  | } |
|  | model->setSort(0, Qt::AscendingOrder); |
|  | } |
|  |  |
|  | void viewalldata::createUI() |
|  | { |
|  | ui->tableView->setModel(model); |
|  | ui->tableView->setColumnHidden(0, true); |
|  | ui->tableView->setSelectionBehavior(QAbstractItemView::SelectRows); |
|  | ui->tableView->setSelectionMode(QAbstractItemView::SingleSelection); |
|  | ui->tableView->resizeColumnsToContents(); |
|  | ui->tableView->setEditTriggers(QAbstractItemView::NoEditTriggers); |
|  | ui->tableView->horizontalHeader()->setStretchLastSection(true); |
|  |  |
|  | model->select(); |
|  | } |
|  | void viewalldata::getTableName(const QString &tableName) { |
|  | table = tableName; |
|  |  |
|  | if (tableName == "users") { |
|  | dataBase = QSqlDatabase::addDatabase("QSQLITE"); |
|  | dataBase.setDatabaseName("/home/igop/Navchannya/Kursova/build-HotelH-Desktop\_Qt\_5\_12\_5\_GCC\_64bit-Debug/users.db"); |
|  | if (!dataBase.open()) { |
|  | qDebug()<<dataBase.lastError().text(); |
|  | } |
|  |  |
|  | this->setupModel("users", |
|  | QStringList() << trUtf8("id") |
|  | << trUtf8("name") |
|  | << trUtf8("age") |
|  | ); |
|  |  |
|  | this->createUI(); |
|  | dataBase.close(); |
|  | } else if (tableName == "Rooms") { |
|  | dataBase = QSqlDatabase::addDatabase("QSQLITE"); |
|  | dataBase.setDatabaseName("/home/igop/Navchannya/Kursova/build-HotelH-Desktop\_Qt\_5\_12\_5\_GCC\_64bit-Debug/users.db"); |
|  | if (!dataBase.open()) { |
|  | qDebug()<<dataBase.lastError().text(); |
|  | } |
|  |  |
|  | this->setupModel("Rooms", |
|  | QStringList() << trUtf8("id") |
|  | << trUtf8("beds") |
|  | << trUtf8("rating") |
|  | << trUtf8("state") |
|  | << trUtf8("date") |
|  | ); |
|  |  |
|  | this->createUI(); |
|  | dataBase.close(); |
|  | } |
|  |  |
|  | } |
|  | void viewalldata::on\_pushButton\_clicked() |
|  | { |
|  | this->hide(); |
|  | } |

Додаток Н  
DVD- диск із програмним продуктом