Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Пєтухова Євгенія Олеговича*
(ПІБ)

академічної групи *122-18ск-1*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки*
(назва освітньої програми)

на тему: *Розробка інформаційної системи організації бронювань та замовлень номерів готелю*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Гуліна І.Г.*			
розділів:				
спеціальний	*доц. Гуліна І.Г.*			
економічний	*проф. Вагонова О.Г.*			
Рецензент	*доц. Шедловський І.А.*			
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2021

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

І.М. Удовик
(підпис) (прізвище, ініціали)

« » 2021 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студенту 122-18ск-1 *Пєтухову Євгенію Олеговичу*
(група) (прізвище та ініціали)

Тема кваліфікаційної роботи *Розробка інформаційної системи організації бронювань та замовлень номерів готелю*

затверджена наказом ректора НТУ «ДП» від *07.06.2021 р.* № *317-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів виробничої практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі.*** ***Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2021 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2021 р.***

Завдання видав _____ *доц. Гуліна І.Г.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Пєтухов Є.О.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2021 р.*

Термін подання дипломного проекту до ДЕК *16.06.2021 р.*

РЕФЕРАТ

Пояснювальна записка: 82 с., 26 рис., 6 табл., 3 дод., 26 джерел.

Об’єкт розробки: інформаційна система готелю.

Мета кваліфікаційної роботи: підвищення якості роботи адміністратора готелю за рахунок створення та використання відповідної інформаційної системи для надання можливості перегляду детальної інформації заявок, клієнтів та номерів.

У вступі розглянуто аналіз проблеми, новизну та актуальність даного програмного забезпечення, завдання, що необхідно виконати.

У першому розділі розглянуто призначення та область застосування програми, детальна постановка завдання, технічні та інструментальні засоби, технології та мови програмування.

У другому розділі виконано проектування та розробку системи, детальний опис її основних етапів, розробку алгоритму взаємодії програми з сервером та бази даних, описані класи, використані бібліотеки та показаний інтерфейс користувача програми.

В економічному розділі визначено трудомісткість розробленої інформаційної системи, проведений підрахунок вартості роботи по створенню програми та розраховано час на його створення.

Практичне значення полягає в автоматизації процесса роботи адміністратора готелю, заповнення форм заявок, зборі інформації у загальну базу даних та її систематизацію.

Перелік ключових слів: ІНФОРМАЦІЙНА СИСТЕМА ГОТЕЛЮ, JAVA, HIBERNATE, SWING, MYSQL.

ABSTRACT

Explanatory note: 82 p., 26 figs., 6 tabls., 3 apps., 26 sources.

Development object: hotel information system.

The purpose of the qualification work: to improve the quality of work of the hotel administrator by creating an appropriate information system to provide the ability to view detailed information of applications, customers and rooms.

The introduction discusses the analysis of the problem, the novelty and relevance of this software, the tasks to be performed.

The first section discusses the purpose and scope of the program, detailed task statement, hardware and tools, technologies and programming languages.

The second section performs the design and development of the system, a detailed description of its main stages, the development of the algorithm of interaction of the program with the server and database, describes the classes, used libraries and shows the user interface of the program.

The economic section determines the complexity of the developed information system, calculates the cost of work to create a program and calculates the time for its creation.

The practical significance lies in the automation of the process of work of the hotel administrator, filling in application forms, collecting information in a common database and its systematization.

Keywords: HOTEL INFORMATION SYSTEM, JAVA, HIBERNATE, SWING, MYSQL.

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ

ЕОМ - електронно-обчислювальна машина;

БД - база даних;

ІС - інформаційна система;

ПЗ - програмне забезпечення;

СУБД - система управління базами даних;

ІТ - інформаційні технології.

ВСТУП

Бази даних складають в даний час основу комп'ютерного забезпечення інформаційних процесів, що входять практично в усі сфери людської діяльності. Дійсно, процеси обробки інформації мають загальну природу і спираються на опис фрагментів реальності, виражене у вигляді сукупності взаємопов'язаних даних. Бази даних є ефективним засобом представлення структур даних і маніпулювання ними. Концепція баз даних припускає використання інтегрованих засобів зберігання інформації, що дозволяють забезпечити централізоване управління даними і обслуговування ними багатьох користувачів. При цьому БД повинна підтримуватися в комп'ютерному середовищі єдиним програмним забезпеченням, що називається системою управління базами даних. Одне з основних призначень СУБД - підтримка програмними засобами подання, відповідної інформації, що міститься в БД. При цьому, предметною областю називається фрагмент реальності, який описується чи моделюється за допомогою БД і її додатків. У предметної області виділяються інформаційні об'єкти - ідентифікуються об'єкти реального світу, процеси, системи, поняття і т.д., відомості про яких зберігаються в БД. Тому завдання створення інформаційних систем на основі баз даних нерозривно пов'язані зі спеціальністю 122 "Комп'ютерні науки».

Метою кваліфікаційної роботи є підвищення якості роботи адміністратора готелю за рахунок створення та використання відповідної інформаційної системи.

Велика завантаженість адміністраторів призводить до низької продуктивності праці і збільшення ймовірності допущення помилок в роботі, що в свою чергу викликано великими часовими витратами на обробку інформації, виконанням рутинної роботи і недостатньою комфортністю умов праці. Очевидно, що використання автоматизованої системи обліку набагато ефективніше.

Розмір готелю мало впливає на її вимоги до функціональних можливостей системи автоматизації. Більше значення тут відіграє рівень готелю і комплекс послуг, нею надаються. Так, для невеликого бутик-готелю безперечно важливими будуть можливості ведення програм частого гостя, організації он-лайн бронювання на власному інтернет-сайті, здійснення клубних програм, інтеграції з системами платного телебачення, телефонії, доступу в інтернет з номера і інші можливості, пропоновані повнофункціональними системами управління. З іншого боку, для невеликого готелю економічного класу буде цілком достатньо елементарних функцій управління бронюванням, поселенням і розрахунками.

Виконання цих заходів дозволить поліпшити репутацію, імідж готелю, що в свою чергу дозволить збільшити число клієнтів, зробити час проводиться, клієнтами в готелі більш комфортним і приємним, дасть можливість працювати більш ефективно, підвищувати свої доходи і рівень обслуговування гостей.

В кваліфікаційній роботі бакалавра вирішується завдання проектування бази даних (зміст БД), реалізації БД в середовищі СУБД MySQL і розробки програми на мові Java для роботи з базою даних.

Темою кваліфікаційної роботи є «Розробка інформаційної системи організації бронювань та замовлень номерів готелю».

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної галузі

Для того, щоб краще зрозуміти структуру і роботу інформаційних систем ресторанно-готельного комплексу необхідно чітко усвідомити роботу самого комплексу.

Готель є складовою частиною індустрії гостинності, яка в свою чергу відноситься до більш великої індустрії - туризму.

Дамо визначення терміну "готель". Готель може бути визначена як комерційне підприємство, основним завданням якого є надання туристам послуг з розміщення та харчування, а також надання в оренду конференц-залів. Класифікувати готелі досить складно через їх великої різноманітності. Але кілька основних класифікаційних ознак все ж існує.

Класифікація готелів:

- Розмір.

Зазвичай готелі групуються за розміром на чотири основні категорії:

• до 150 номерів;

• від 150 до 300 номерів;

• від 300 до 600 номерів;

• більше 600 номерів.

Така класифікація дозволяє готелям однакового розміру порівнювати свої операційні результати і статистичні дані.

- Цільовий ринок.

При створенні готелі одним з найбільш важливих питань є "хто буде проживати в ній? Для кого вона призначена? "Тобто необхідно визначити цільовий ринок. У число найпоширеніших готелів за цією ознакою звичайно включають: комерційні, готелі при аеропортах, готелі-люкс, готелі-курорти,

придорожні, готелі-казино, постоялі двори. Залежно від типу готелів формується і наступний ознака.

- Рівень сервісу.

Ще одним класифікаційним ознакою є рівень і кількість пропонованих послуг. Згідно з цим ознакою готелі бувають що пропонують:

• Вищий рівень послуг - готелі розраховані на прийом гостей, що представляють вищі політичні, управлінські кола, знаменитостей, добре забезпечених людей.

• Середній рівень послуг - такі готелі розраховані на найбільший сегмент подорожуючих. Найчастіше вони мають від 150 до 200 номерів.

• Обмежений рівень послуг - призначені для прийому осіб з невеликим достатком. Нерідко такі готелі можна зустріти в невеликих селищах, у великих автострад. Основне завдання і послуга - "надати дах над головою".

- Підпорядкованість.

Готелі можуть функціонувати як незалежні комерційні підприємства або входити в готельні ланцюги, що надають їм певні переваги.

Готельні ланцюги - це об'єднання готелів, що працюють за контрактом на управління. Такий контракт полягає між власником конкретного готелю і компанією, що надає послуги в галузі управління. Нерідко одна і та ж компанія здійснює керування великим числом готелів.

Інформаційні системи, що існують в готелі, об'єднують найрізноманітніші її служби. Таким чином, на початку повинна існувати деяка формальна структура або внутрішня організація, яка з одного боку буде координувати зусилля працівників для досягнення мети організації і з іншого боку дозволить використовувати досягнення науково-технічного прогресу для скорочення часу на виконання різного роду операцій. Перш ніж проектувати БД необхідно чітко знати підпорядкованість і зв'язку всередині підприємства. Організаційна схема це найбільш наочний спосіб представлення цієї інформації. Класик менеджменту Анрі Файоль сформулював наступні принципи створення хорошої організації.

• Єдність управління. Незалежно від структури організації, ступеня децентралізації делегування повноважень нести повну і абсолютну відповідальність за діяльність всього підприємства повинен одна людина.

• Скалярний метод передачі повноважень. Повна і абсолютна відповідальність означає право не тільки керувати, а й передавати, делегувати частину повноважень іншим особам по лінії керівництва.

• Єдність підпорядкування .У будь-якого службовця може і повинен бути тільки один керівник.

• Принцип відповідності. Делеговані повноваження повинні відповідати рівню відповідальності. Якщо на особу покладаються певні обов'язки, то його необхідно наділити і відповідними повноваженнями, необхідними для виконання цих обов'язків.

• Гнучкість. Структура організації повинна дозволяти вносити в неї корективи зв'язку зі зміною методів, завдань, цілей, масштабів комерційної діяльності, появою нових технологій і ресурсів.

• Доступність всіх рівнів організації. Будь-який співробітник організації повинен мати право і можливість подати скаргу, висловити зауваження або пред'явити рекламації відповідному керівнику.

У роботі розглядається інформаційна система для допомоги адміністратора готелю.

База даних - це сукупність відомостей про реальні об'єкти, процеси, події чи явища, що відносяться до певної теми або завдання, організована таким чином, щоб забезпечити зручне представлення цієї сукупності, як в цілому, так і будь-який її частини. Реляційна база даних являє собою безліч взаємопов'язаних таблиць, кожна з яких містить інформацію про об'єкти певного типу. Кожен рядок таблиці містить дані про один об'єкт (наприклад, клієнта, автомобілі, документі), а стовпці таблиці містять різні характеристики цих об'єктів - атрибути (наприклад, найменування та адреси клієнтів, марки і ціни автомобілів). Рядки таблиці називаються записами, всі записи мають однакову структуру - вони складаються з полів, в яких зберігаються атрибути

об'єкта. Кожне поле в записі містить одну характеристику об'єкта і має строго певний тип даних (наприклад, текстовий рядок, число, дата). Всі записи мають одні і ті ж поля, тільки в них містяться різні значення атрибутів.

Будь-яка СУБД дозволяє виконувати чотири найпростіші операції з даними:

- додати в таблицю одну або кілька записів;
- удалить з таблиці одну або кілька записів;
- обновіть значення деяких полів в одній або декількох записах;
- знайти одну або кілька записів, що задовольняють заданій умові.

Функціонування практично будь-якого сучасного підприємства немислимо без маніпуляції даними, пов'язаними з його виробничою діяльністю. Нерідко ефективність його діяльності та конкурентоспроможність на ринку товарів або послуг безпосередньо пов'язані з тим, чи актуальні ці дані і чи доступні вони звертаються до них користувачам (причому нерідко не тільки користувачам локальної мережі, але і відвідувачам корпоративного Web-сервера і співробітникам, які звертаються до них за допомогою мобільних пристроїв). З цією метою застосовуються різні архітектури фізичного зберігання даних, такі як Storage Area Network (SAN) або Network Attached storage (NAS), а також системи управління базами даних, призначені для логічної організації даних і здійснення доступу до них. Корпоративні дані більшості компаній зараз зберігаються в реляційних СУБД.

У найпростішому випадку інформаційна система, яка використовує СУБД, складається з двох основних компонентів: сервера баз даних, керуючого даними і виконує надходять від клієнтських додатків запити, і самих клієнтських додатків, що забезпечують інтерфейс користувача і посилають запити до сервера. Саме сервер баз даних може маніпулювати файлами, в яких зберігаються дані, виконувати запити користувачів, підтримувати кількість посилань цілісність даних, забезпечувати доступ до них, здійснювати резервне копіювання даних і протоколювати операції, пов'язані з їх зміною.

Функціональні можливості підсистеми дозволяють формувати готівкові та безготівкові рахунки за тарифами готелю за надані послуги, проводити реєстрацію, поселення, формувати замовлення на поселення (бронювання номерів), вести контроль наданих гостям послуг, вести облік телефонних переговорів, а також формувати і роздруковувати необхідні звіти по обліку гостей і оплати наданих послуг.

Завдання Адміністратора - введення (реєстрація) первинних документів з бронювання та поселення гостей. Далі система автоматично обробляє їх таким чином, як зазначено в її налаштування. Це в значній мірі полегшує роботу адміністратора з контролю над станом системи, а сам облік робиться схожим на ручну обробку документів.

1.2. Призначення розробки та галузь застосування

Створена інформаційна система призначена для підвищення якості та ефективності роботи адміністратора готелю та надання можливості перегляду детальної інформації замовлень, клієнтів та номерів.

В якості вхідної інформації для вирішення завдання використовується сукупність довідників по системі управління готелем, що описують функції діяльності адміністраторів і керівного персоналу готелю в наступних напрямках:

• підвищення продуктивності праці і зниження ймовірності помилок персоналу за рахунок надійного оперативного інформаційного забезпечення та автоматизації рутинних операцій на різних етапах роботи;

• оптимізація управління номерним фондом готелю.

1.3. Підстава для розробки

Відповідно до освітньої програми, згідно навчального плану та графіків навчального процесу, в кінці навчання студент виконує кваліфікаційну роботу

(проект). Тема роботи узгоджується з керівником проекту, випускаючою кафедрою, та затверджується наказом ректора.

Отже, підставами для розробки (виконання кваліфікаційної роботи) є:

- Освітня програма зпеціальності 122 «Комп’ютерні науки»;
- навчальний план та графік навчальногопроцесу;
- наказ ректора Національного технічного університету «Дніпровська політехніка» № 317-с від 07.06.2021 р;
- завдання на кваліфікаційну роботу на тему «Розробка інформаційної системи організації бронювань та замовлень номерів готелю».

1.4. Постановка завдання

Завданням кваліфікаційної роботи є створення інформаційної системи, а саме бази даних і графічного додатку для адміністратора готелю. Додаток має забезпечувати роботу з базою даних, пошук вільних і зайнятих номерів; пошук гостя по довільною ознакою; пошук дати заселення і виселення з конкретного номера.

До функцій, які повинні бути реалізовані, відносяться:

• оформлення заявки на розміщення клієнта;

• підтримка в актуальному стані нормативно-довідкової інформації (НДІ) системи (опис категорій і номерів, прейскурант цін на проживання);

• забезпечення зручного введення даних;

• перегляд вже введених даних;

• реалізація процесу пошуку необхідної інформації.

Номери в готелі можуть бути одномісними, двомісними, або двомісним люксом відповідно до цього і оплата за номери різна. В одному номері можуть проживати кілька гостей. Номер будь-якого типу оформляється на одну людину.Основними задачами при автоматизації робочого місця адміністратора готелю є:

• правильно організоване розміщення гостей в номерах;

• додавання і зберігання даних;

• пошук гостя по довільною ознакою;

• вибір гостей від'їжджаючих сьогодні, звільнення номера;

Систематизація даних дозволяє швидко провести пошук за характером параметрів, а так само в зручній формі зберігати і проводити аналіз інформації.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Для досягнення поставлених цілей програмне забезпечення, що розробляється, повинно підтримувати зручне та надійне внесення, редагування та зберігання інформації в системі.

Система, що розробляється в кваліфікаційній роботі пов инна забезпечувати виконання таких функції як:

• оформлення заявки на розміщення клієнта;

• підтримка в актуальному стані нормативно-довідкової інформації (НДІ) системи (опис категорій і номерів, прейскурант цін на проживання);

• забезпечення зручного введення даних;

• перегляд вже введених даних;

• реалізація процесу пошуку необхідної інформації.

1.5.2. Вимоги до інформаційної безпеки

Для уникнення некоректної роботи програми необхідно реалізувати:

- контроль вхідних даних;
- обробку виняткових ситуацій;
- виведення повідомлень про помилки;
- можливість повторного введення даних;
- можливість безперервної роботи протягом не менше 120 годин (5 діб);
- платформну незалежність;

– вірогідність виникнення не більше 2 логічних помилок на 1000 операторів за 1 рік експлуатації;

– забезпечення неушкодженого стану даних у випадку відмови.

1.5.3. Вимоги до складу та параметрів технічних засобів

Для нормального функціонування програми необхідно, щоб обчислювальна машина, на якій буде функціонувати шнформаційна система, відповідала наступним вимогам:

– процесор класу IntelXeon з тактовою частотою не менш 2.4 ГГц;

– не менше 2 GB оперативної пам'яті;

– монітор з діагоналлю не менше 17";

– 2 Гб вільного місця на жорсткому диску (та додатковий простір для зберігання БД);

– доступ до мережі Internet;

– клавіатура;

– маніпулятор «миша».

Наведені вище технічні характеристики є рекомендованими, тобто при наявності технічних засобів не нижче зазначених, розроблений програмний виріб буде функціонувати відповідно до вимог щодо надійності, швидкості обробки даних і безпеки, висунутими замовником.Також слід зазначити, що розташування резервної копії БД може бути організовано за рахунок хмарних технологій.

1.5.4. Вимоги до інформаційної та програмної сумісності

Для нормального функціонування програми необхідно, щоб програмне забезпечення обчислювальної машини, на якій буде функціонувати розроблена інформаційна система мала ОС сімейства Windows.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення системи

Створена інформаційна система призначена для підвищення якості та ефективності роботи адміністратора готелю та надання можливості перегляду детальної інформації заявок, клієнтів та номерів.

До функцій, які повинні бути реалізовані, відносяться:

• оформлення заявки на розміщення клієнта;

• підтримка в актуальному стані нормативно-довідкової інформації (НДІ) системи (опис категорій і номерів, прейскурант цін на проживання);

• забезпечення зручного введення даних;

• перегляд вже введених даних;

• правильно організоване розміщення гостей в номерах;

• додавання і зберігання даних;

• пошук гостя по довільною ознакою;

• вибір гостей від'їжджаючих сьогодні, звільнення номера;

• реалізація процесу пошуку необхідної інформації.

Номери в готелі можуть бути одномісними, двомісними, або двомісним люксом відповідно до цього і оплата за номери різна. В одному номері можуть проживати кілька гостей. Номер будь-якого типу оформляється на одну людину.

2.2. Опис застосованих математичних методів

Оскільки особливості предметної галузі розв'язуваної задачі не передбачають застосування математичних методів, при розробці системи специфічні математичні методи не використовувалися.

2.3. Опис використаних технологій та мов програмування

Зберігання даних в інформаційних системах можна організувати наступним чином:

• набір файлів на диску;

• реляційна база даних;

• нереляційних база даних (ієрархічні СУБД і ін.).

Найбільш універсальним, масштабується і простим для підтримки є використання в системі, що розробляється реляційної СУБД. Порівняльні характеристики найбільш поширених на сьогоднішній день реляційних СУБД приведені в таблиці 2.1.

Проектована база даних передбачає роботу з порівняно невеликими обсягами даних (створення не більше декількох сотень записів на добу). При цьому важливим критерієм є масштабованість і можливо низька вартість рішення. Виходячи з цих критеріїв, згідно з таблицею 2.1, для вирішення поставленого завдання найбільш доцільним є використання СУБД MySQL.

Для створення бази даних була вибрана мова структурованих запитів SQL, оскільки має такі переваги: незалежність від конкретних СУБД; переносимість з однієї обчислювальної системи на іншу.

Таблиця 2.1.

Порівняльні характеристики реляційних СУБД

СУБД / Характеристики	**MS SQL Server**	**Oracle**	**MySQL**	**SQLite**
Об’єм данних	Великі	Великі	Середні	Невеликі
Швидкодія	Висока	Висока	Середня	Середня
Вимогливість до ресурсів	Середня	Висока	Низька	Низька
Масштабованість	Висока	Висока	Висока	Низька
Вартість	Середня	Висока	Безкоштовно	Безкоштовно

SQL використовується в СУБД, призначених для різних обчислювальних систем: від персональних комп'ютерів і робочих станцій до локальних мереж, міні-комп'ютерів і великих ЕОМ; однопользовательские додатки на основі SQL можуть бути перенесені в більш великі системи; можливість різного представлення даних.

За допомогою SQL творець бази може зробити так, що різні користувачі бази даних будуть бачити різні уявлення її структури і вмісту; повноцінність як мови, призначеного для роботи з базами даних.

Спочатку SQL був задуманий як мова інтерактивних запитів, але зараз він вийшов далеко за рамки читання даних; наявність стандартів (наявність стандартів і набору тестів для виявлення сумісності і відповідності конкретній реалізації SQL загальноприйнятому стандарту тільки сприяє "стабілізації" мови); декларативність (за допомогою SQL програміст описує тільки те, які дані потрібно витягнути або модифікувати);

можливість спільного використання даних декількома користувачами, що працюють паралельно; SQL є повноцінним і логічним мовою, призначеним для таких цілей:

- створення бази даних;
- управління захистом бази даних;
- зміна даних;
- читання даних;

MySQL є рішенням для малих і середніх додатків. Входить до складу серверів WAMP, LAMP і в портативні збірки серверів Денвер, XAMPP. Зазвичай MySQL використовується як сервер, до якого звертаються локальні або видалені клієнти, проте в дистрибутив входить бібліотека внутрішнього сервера, що дозволяє включати MySQL в автономні програми.

Гнучкість СУБД MySQL забезпечується підтримкою великої кількості типів таблиць: користувачі можуть вибрати як таблиці типу MyISAM, що підтримують повнотекстовий пошук, так і таблиці InnoDB, що підтримують

транзакції на рівні окремих записів. Завдяки відкритій архітектурі і GPL-ліцензуванню, в СУБД MySQL постійно з'являються нові типи таблиць.

MySQL має API для мов C, C++, Java, Лисп, Perl, PHP, Python, Ruby, Smalltalk і Tcl, бібліотеки для мов платформи.net, а також забезпечує підтримку для ODBC за допомогою ODBC-драйвера MyODBC.

XAMPP є повністю безкоштовним і простим в установці дистрибутивом Apache, MySQL, FileZilla, Mercury і Tomcat. Він створений з відкритим вихідним кодом, щоб бути неймовірно простим в установці і у використанні.

Мета XAMPP є створення простого в установці дистрибутива для розробників, щоб легко увійти в світ Apache. Щоб зробити його зручним для розробників, XAMPP налаштований з усіма включеними функціями. В даний час є дистрибутиви для Windows, Linux, і OS X.

Для установки XAMPP необхідно завантажити один файл формату zip, tar або exe. Компоненти програми не вимагають настройки. Програма регулярно оновлюється для включення до складу новітніх версій Apache / MySQL / PHP і Perl. Також в складі XAMPP присутні інші модулі, включаючи OpenSSL і phpMyAdmin.

Інтерфейс програми настільки простий, що її називають «складанням для ледачих» («lazy man's WAMP / LAMP installation»). Установка XAMPP займає менше часу, ніж установка кожного компонента окремо.

Даний web-сервер поширюється в повній, стандартної і зменшеною (відомої як XAMPP Lite) версіях. Всі додаткові модулі також доступні для скачування.

З додаткових можливостей можна відзначити, що сама компанія випускає пакети оновлень, що випускаються у вигляді zip, 7-zip, tar або exe, які дозволяють відновити всі компоненти з однієї версії збірки xampp на новішу.

Спочатку XAMPP створювався як інструмент для розробників, дозволяючи веб-дизайнерам і програмістам тестувати свою роботу, не використовуючи Інтернет. Для спрощення роботи деякі можливості і настройки безпеки відключені за замовчуванням, і в цілому XAMPP рекомендується до

використання тільки в дуже дружньому оточенні. Однак XAMPP іноді використовується і у всесвітній павутині. Також програма підтримує створення і управління базами даних MySQL і SQLite.

XAMPP, як і інші подібні пакети, можна використовувати для установки власної копії Вікіпедії на комп'ютер.

Установка сайту безпосередньо на XAMPP-сервер полягає в копіюванні файлів сайту в папку htdocs.

Apache HTTP - вільний веб-сервер. Apache є кросплатформним ПО, підтримує операційні системи Linux, BSD, Mac OS, Microsoft Windows, Novell NetWare, BeOS.

Основними достоїнствами Apache вважаються надійність і гнучкість конфігурації. Він дозволяє підключати зовнішні модулі для надання даних, використовувати СУБД для аутентифікації користувачів, модифікувати повідомлення про помилки і т. д. Підтримує IPv6.

Ядро Apache включає в себе основні функціональні можливості, такі як обробка конфігураційних файлів, протокол HTTP і система завантаження модулів. Ядро (на відміну від модулів) повністю розробляється Apache Software Foundation, без участі сторонніх програмістів.

Теоретично, ядро apache може функціонувати в чистому вигляді, без використання модулів. Однак, функціональність такого рішення вкрай обмежена.

Ядро Apache повністю написано на мові програмування C.

Система конфігурації Apache заснована на текстових конфігураційних файлах. Має три умовних рівня конфігурації:

• Конфігурація сервера (httpd.conf).

• Конфігурація віртуального хоста (httpd.conf с версії 2.2, extra / httpd-vhosts.conf).

• Конфігурація рівня директорії (.htaccess).

Має власну мову конфігураційних файлів, заснований на блоках директив. Практично всі параметри ядра можуть бути змінені через

конфігураційні файли, аж до управління MPM. Велика частина модулів має власні параметри.

Частина модулів використовує в своїй роботі конфігураційні файли операційної системи (наприклад / etc / passwd і / etc / hosts).

Крім цього, параметри можуть бути задані через ключі командного рядка.

Існує безліч модулів, що додають до Apache підтримку різних мов програмування і систем розробки.

До них відносяться: PHP (mod_php), Python (mod python, mod wsgi), Ruby (apache-ruby), Perl (mod perl), ASP (apache-asp) і Tcl (rivet).

Крім того, Apache підтримує механізми CGI і FastCGI, що дозволяє виконувати програми на практично всіх мовах програмування, в тому числі C, C ++, Lua, sh, Java.

Apache має різні механізми забезпечення безпеки і розмежування доступу до даних. Основними є:

• Обмеження доступу до певних директорій або файлів.

• Механізм авторизації користувачів для доступу до директорії на основі HTTP-аутентифікації (mod_auth_basic) і digest-аутентифікації (mod_auth_digest).

• Обмеження доступу до певних директорій або всьому сервера, засноване на IP-адреси користувачів.

• Заборона доступу до певних типів файлів для всіх або частини користувачів, наприклад заборона доступу до конфігураційним файлів і файлів баз даних.

Існують модулі, що реалізують авторизацію через СУБД або PAM.

У деяких MPM-модулях є можливість запуску кожного процесу Apache використовуючи різні uid і gid з відповідними цим користувачам і групам користувачів.

Також, існує механізм suexec, який використовується для запуску скриптів і CGI-додатків з правами і ідентифікаційними даними користувача.

Для реалізації шифрування даних, що передаються між клієнтом і сервером використовується механізм SSL, реалізований через бібліотеку

OpenSSL. Для підтвердження автентичності веб-сервера використовуються сертифікати X.509.

Існують зовнішні засоби забезпечення безпеки, наприклад mod_security.

За базу даних використовується MySQL з веб-інтерфейсом phpMyAdmin, потрапити в який можна включивши сервер Apache.

phpMyAdmin - веб-додаток з відкритим кодом, написаний на мові PHP і представляє собою веб-інтерфейс для адміністрування СУБД MySQL. PHPMyAdmin дозволяє через браузер і не тільки здійснювати адміністрування сервера MySQL, запускати команди SQL і переглядати вміст таблиць і баз даних. Додаток користується великою популярністю у веб-розробників, так як дозволяє управляти СУБД MySQL без безпосереднього введення SQL команд, надаючи дружній інтерфейс.

На сьогоднішній день PHPMyAdmin широко застосовується на практиці. Останнє пов'язано з тим, що розробники інтенсивно розвивають свій продукт, враховуючи всі нововведення СУБД MySQL. Переважна більшість російських провайдерів використовують цю програму в якості панелі управління для того, щоб надати своїм клієнтам можливість адміністрування виділених їм баз даних.

Додаток поширюється під ліцензією GNU General Public License і тому багато інших розробники інтегрують його в свої розробки, наприклад XAMPP, Denwer, AppServ, Open Server.

Проект на даний момент часу локалізований на більш ніж 62 мовах.

Java - сильно типізований об'єктно-орієнтована мова програмування, розроблений компанією Sun Microsystems (в подальшому придбаної компанією Oracle). Програми Java зазвичай транслюються в спеціальний байт-код, тому вони можуть працювати на будь-якої комп'ютерної архітектурі, за допомогою віртуальної Java-машини. Дата офіційного випуску - 23 травня 1995 року.

Програми на Java транслюються в байт-код Java, який виконується віртуальною машиною Java (JVM) - програмою, обробній байтовий код і передавальної інструкції обладнанню як інтерпретатор.

Перевагою подібного способу виконання програм є повна незалежність байт-коду від операційної системи і устаткування, що дозволяє виконувати Java-додатки на будь-якому пристрої, для якого існує відповідна віртуальна машина. Іншою важливою особливістю технології Java є гнучка система безпеки, в рамках якої виконання програми повністю контролюється віртуальною машиною. Будь-які операції, які перевищують встановлені повноваження програми (наприклад, спроба несанкціонованого доступу до даних або з'єднання з іншим комп'ютером), викликають негайне переривання.

Часто до недоліків концепції віртуальної машини відносять зниження продуктивності. Ряд удосконалень кілька збільшив швидкість виконання програм на Java:

• застосування технології трансляції байт-коду в машинний код безпосередньо під час роботи програми (JIT-технологія) з можливістю збереження версій класу в машинному коді,

• широке використання переносних орієнтованого коду (native-код) в стандартних бібліотеках,

• апаратні засоби, що забезпечують прискорену обробку байт-коду (наприклад, технологія Jazelle, підтримувана деякими процесорами фірми ARM).

За даними сайту shootout.alioth.debian.org, для семи різних завдань час виконання на Java становить в середньому в півтора-два рази більше, ніж для C/C++, в деяких випадках Java швидше, а в окремих випадках в 7 разів повільніше. З іншого боку, для більшості з них споживання пам'яті Java-машиною було в 10-30 разів більше, ніж програмою на C/C ++. Також примітно дослідження, проведене компанією Google, згідно з яким відзначається істотно нижча продуктивність і більше споживання пам'яті в тестових прикладах на Java в порівнянні з аналогічними програмами на C ++.

Ідеї, закладені в концепцію і різні реалізації середовища віртуальної машини Java, надихнули безліч ентузіастів на розширення переліку мов, які могли б бути використані для створення програм, що виконуються на

віртуальній машині. Ці ідеї знайшли також вираз в специфікації загальномовна інфраструктури CLI, закладеної в основу платформи .NET компанією Microsoft.

Основні можливості Java:

1) автоматичне керування пам'яттю;

2) розширені можливості обробки виняткових ситуацій;

3) багатий набір засобів фільтрації введення-виведення;

4) набір стандартних колекцій: масив, список, стек і т. П .;

5) наявність простих засобів створення мережевих додатків (у тому числі з використанням протоколу RMI);

6) наявність класів, що дозволяють виконувати HTTP-запити і обробляти відповіді;

7) вбудовані в мову засоби створення багатопоточних додатків, які потім були перенести на багато мов (наприклад, python);

8) уніфікований доступ до баз даних:

a) на рівні окремих SQL-запитів - на основі JDBC, SQLJ;

b) на рівні концепції об'єктів, що володіють здатністю до зберігання в базі даних - на основі Java Data Objects (англ.) і Java Persistence API;

9) підтримка узагальнень (починаючи з версії 1.5);

10) підтримка лямбда, замикань, вбудовані можливості функціонального програмування;

11) безліч варіантів реалізації багатопотокових програм.

2.4. Опис структури системи та алгоритмів її функціонування

2.4.1. Створення концептуальної моделі бази даних

Концептуальне проектування - побудова формалізованої моделі предметної області. Така модель будується з використанням стандартних мовних засобів, зазвичай графічних.

Розробляється база даних містить в собі дані про здаються номерах, вартості номерів, дані про клієнтів, а також дані про заявки.

База даних повинна виконувати такі основні завдання:

- зберігати відомості про всі номерах готелю;

- зберігати відомості про заброньовані номери клієнтів;

- забезпечувати пошук потрібного номера;

- забезпечувати оформлення номера на потрібного покупця.

База даних повинна бути налаштованою, тобто в ній має бути присутня можливість зміни, доповнення наступних параметрів:

- клієнта і (або) номера (при в'їзді або від'їзді клієнта);

Необхідно реалізувати базу пошуку за такими параметрами:

- зберігати відомості дати бронювання номера;

- зберігати відомості дати від'їзду покупця номера.

Для здійснення процесу оформлення номера необхідно ввести прізвище, ім'я, по батькові, номер телефону, і адреса проживання клієнта. Після покупки номера клієнтом необхідна збереження інформації про клієнта, але крім цього, про номер. Повинна бути можливість переглядати список зареєстрованих клієнтів, номерів готелю, всіх оформлених заявок (тільки для адміністрації).

Основними об'єктами (сутностями) в описі предметної області з точки зору бази даних є:

• готель;

• номер;

• інформацію про номери;

• замовлення;

• клієнт;

Атрибутами номера є:

• номер кімнати;

• тип номера;

Атрибутами опису номера є:

• тип номера;

• вартість номера;

Атрибутами замовлення є:

• номер замовлення;

• дата замовлення;

• дата прибуття;

• дата виїзду;

• ПІБ клієнта;

• номер кімнати;

• вартість проживання;

Атрибутами клієнтів є:

• номер клієнта;

• ПІБ;

• номер телефону;

• адреса;

На опис предметної області, а також описаних сутностей і їх атрибутів можна виділити наступні види зв'язків між сутностями бази даних, зображені на рис. 2.1.

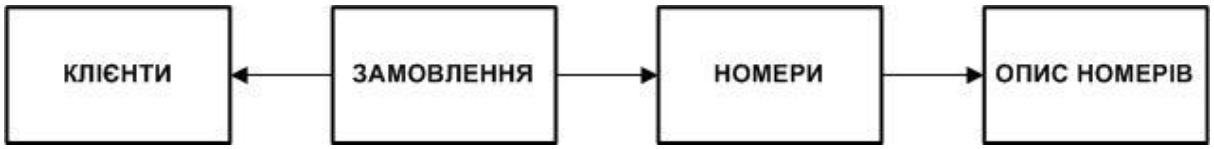


Рис 2.1. Інфологічна модель бази даних

2.4.2. Створення логічної моделі

Метою побудови логічної моделі є отримання графічного представлення логічної структури досліджуваної предметної області.

Логічна модель предметної області ілюструє сутності, а також їх взаємовідносини між собою.

Сутності описують об'єкти, які є предметом діяльності предметної галузі, і суб'єкти, що здійснюють діяльність в рамках предметної галузі. Властивості об'єктів і суб'єктів реального світу описуються за допомогою атрибутів.

Взаємини між сутностями ілюструються за допомогою зв'язків. Правила та обмеження взаємин описуються за допомогою властивостей зв'язків. Зазвичай зв'язку визначають якої залежності між сутностями, який вплив однієї сутності на іншу.

Логічні взаємозв'язки являють собою зв'язки між сутностями. Вони визначаються дієсловами, які показують, як одна сутність відноситься до іншої.

У всіх цих випадках взаємозв'язку відображають взаємодію між двома сутностями, зване «один-до-багатьох». Це означає, що один екземпляр першої суті взаємодіє з декількома екземплярами іншої сутності. Взаємозв'язку відображаються лініями, що з'єднують дві сутності з точкою на одному кінці і дієсловом, що розташовуються над лінією.

Крім взаємозв'язку «один-ко-многим» існує ще один тип - це «багато-до-багатьох». Цей тип зв'язку описує ситуацію, при якій екземпляри сутностей можуть взаємодіяти з декількома екземплярами інших сутностей. Зв'язок «багато-до-багатьох» використовують на початкових стадіях проектування. Цей тип взаємозв'язку відображається суцільною лінією з точками на обох кінцях.

Зв'язок «багато-до-багатьох» може не враховувати певні обмеження системи, тому може бути замінена на «один-ко-многим» при подальшому перегляді проекту.

Логічна модель бази даних інформаційної системи що розробляється наведено на рис. 2.2.

Рис. 2.2. Логічна модель бази даних інформаційної системи

Таблиця - це деяка регулярна структура, що складається з кінцевого набору полів по вертикалі і однотипних записів по горизонталі. У реляційної моделі даних таблиця називається відношенням.

Атрибут - це інформаційне відображення властивостей об'єкта. Кожен об'єкт характеризується рядом основних атрибутів.

Первинний ключ (Primary Key) - це атрибут (або група атрибутів), які єдиним чином ідентифікують кожен рядок в таблиці.

Альтернативний (Foreign Key) ключ - це атрибут (або група атрибутів), неспівпадаючий з первинним ключем і унікально ідентифікує екземпляр об'єкта.

Мобільний зв'язок - це функціональна залежність між сутностями. Підтримка несуперечності функціональних залежностей між сутностями називається посилальної цілісністю.

Таблиця 2.2.

Атрибути і первинні ключі сутностей логічної моделі

Сутність	Первиний ключ	Атрибути
Клієнт	Номер клієнта	Номер клієнта ПІБ Номер телефону Адреса
Замовлення	Номер замовлення	Номер замовлення Дата замовлення Дата прибуття Дата виїзду ПІБ клієнта Номер кімнати Вартість проживання
Номери	Номер кімнати	Номер кімнати Тип номеру
Опис номеру	Ідентифікатор	Ідентифікатор Тип номеру Вартість номеру

2.4.3. Створення фізичної моделі

Фізична модель визначає спосіб розміщення даних в середовищі зберігання і способи доступу до цих даних, які підтримуються на фізичному рівні.

Атрибут - це інформаційне відображення властивостей об'єкта. Кожен об'єкт характеризується рядом основних атрибутів і в фізичної моделі кожному атрибуту відповідає поле записи.

Таблиця 2.3.

Таблиця «Номери»

№ п/п	Найменування поля	Примітка	Тип
1.	Id	Номер кімнати	int(11)
2.	Type	Тип кімнати	varchar(45)

Таблиця 2.4.

Таблиця «Опис номера»

№ п/п	Найменування поля	Примітка	Тип
1.	Id	Ідентифікатор	int(11)
2.	Type	Тип кімнати	varchar(45)
3.	Cost	Вартість кімнати	int(11)

Таблиця 2.5.

Таблиця «Замовлення»

№ п/п	Найменування поля	Примітка	Тип
1.	Id	Номер заказа	int(11)
2.	Date_arrive	Дата прибуттяя	date
3.	Date_leave	Дата виїзду	date
4.	Date_order	Дата замовлення	date
5.	Client_FIO	ПІБ клієнта	varchar(45)
6.	Room_number	Номер кімнати	int(11)
7.	Order_cost	Вартість проживання	int(11)

Таблиця 2.6.

Таблиця «Клієнти»

№ п/п	Найменування поля	Примітка	Тип
1.	Id	Номер клієнта	int(11)
2.	FIO	ПІБ	varchar(45)
3.	Phone	Номер телефону	int(11)
4.	Adress	Адреса	varchar(300)

Фізична модель бази даних наведена на рис. 2.3.

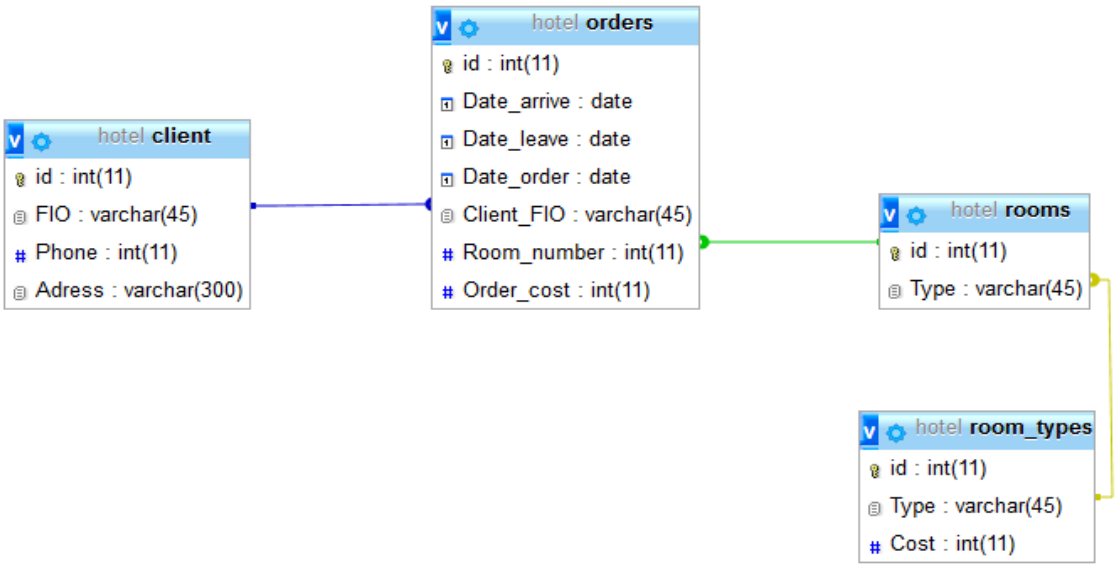


Рис 2.3. Фізична модель бази даних

Для створення фрагмента автоматизованої інформаційної системи робочого місця адміністратора готелю, необхідно створити базу даних з наступними таблицями: КЛІЄНТИ, ЗАМОВЛЕННЯ, НОМЕРИ, ОПИС НОМЕРА.

Першою створюємо таблицю НОМЕРА. У цій таблиці міститься інформація про номери готелі. В поле Id вказано номер кімнати, а в поле тип вказується вид кімнати (одномісна, двомісна, люкс).

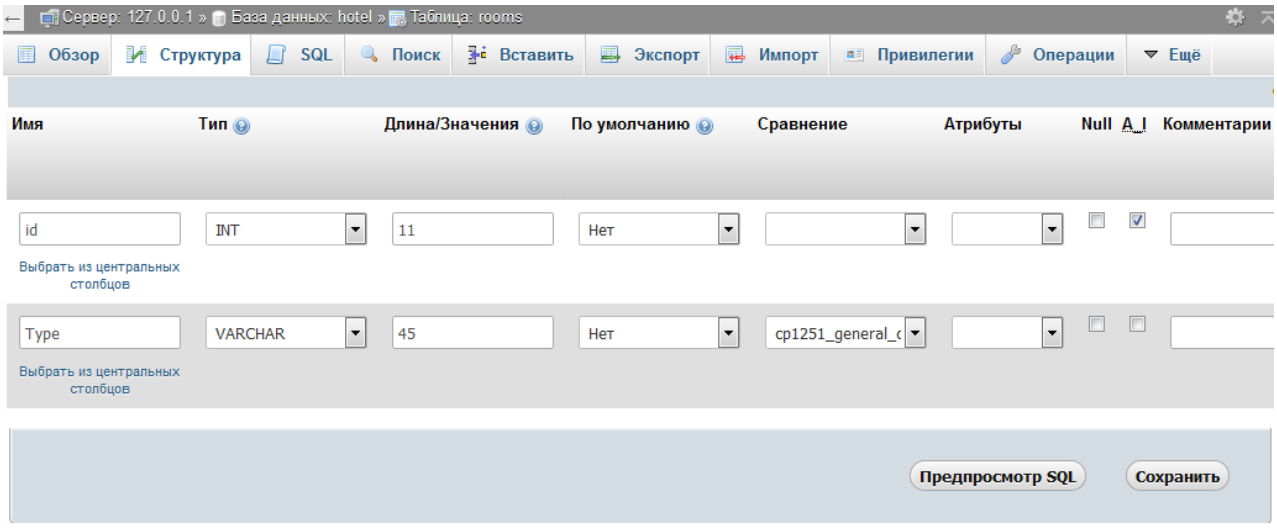


Рис. 2.4. Створення таблиці «НОМЕРИ»

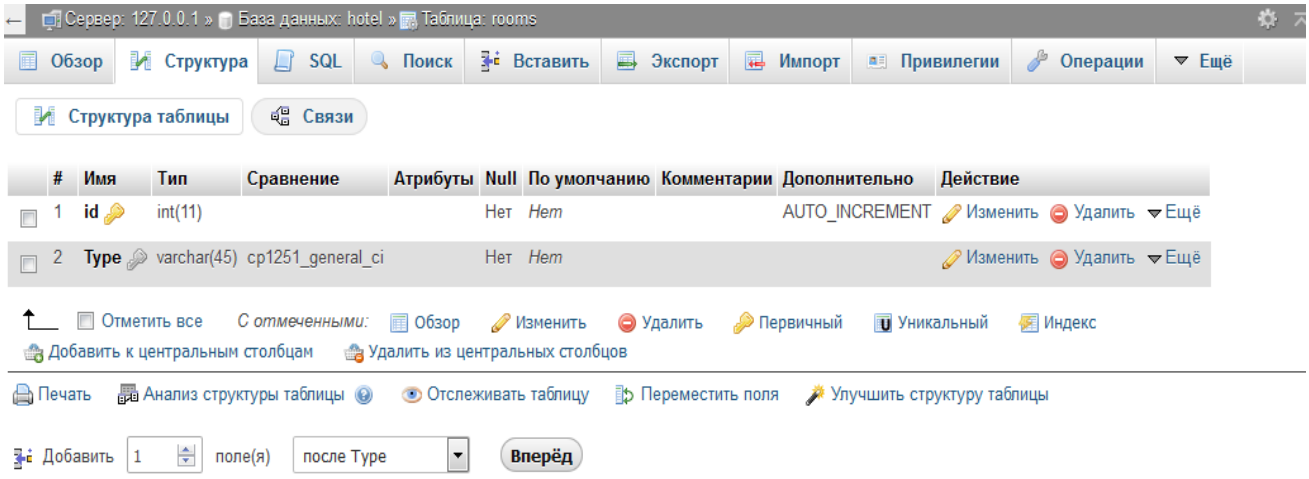


Рис. 2.5. Готова таблиця «НОМЕРИ»

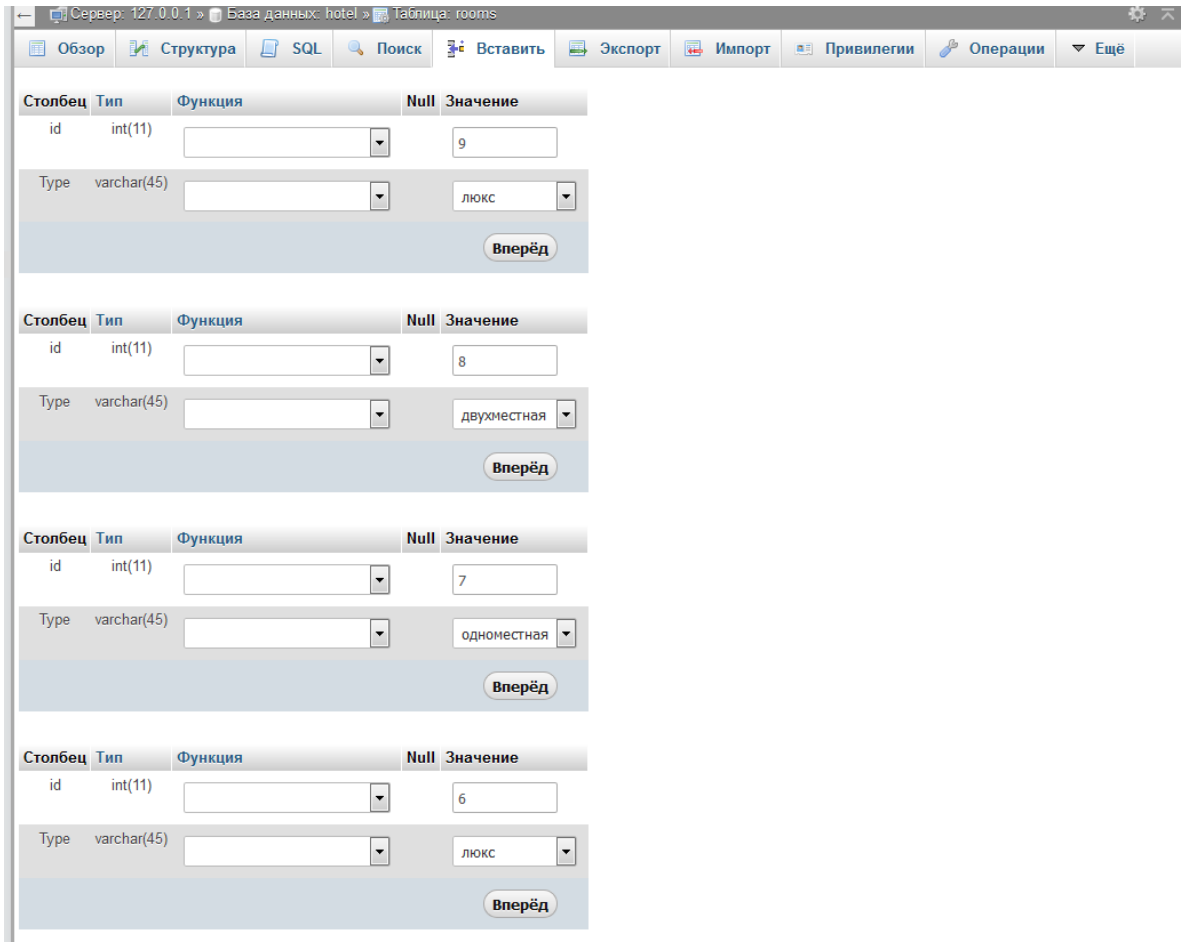


Рис. 2.6. Заповнення таблиці «НОМЕРИ»

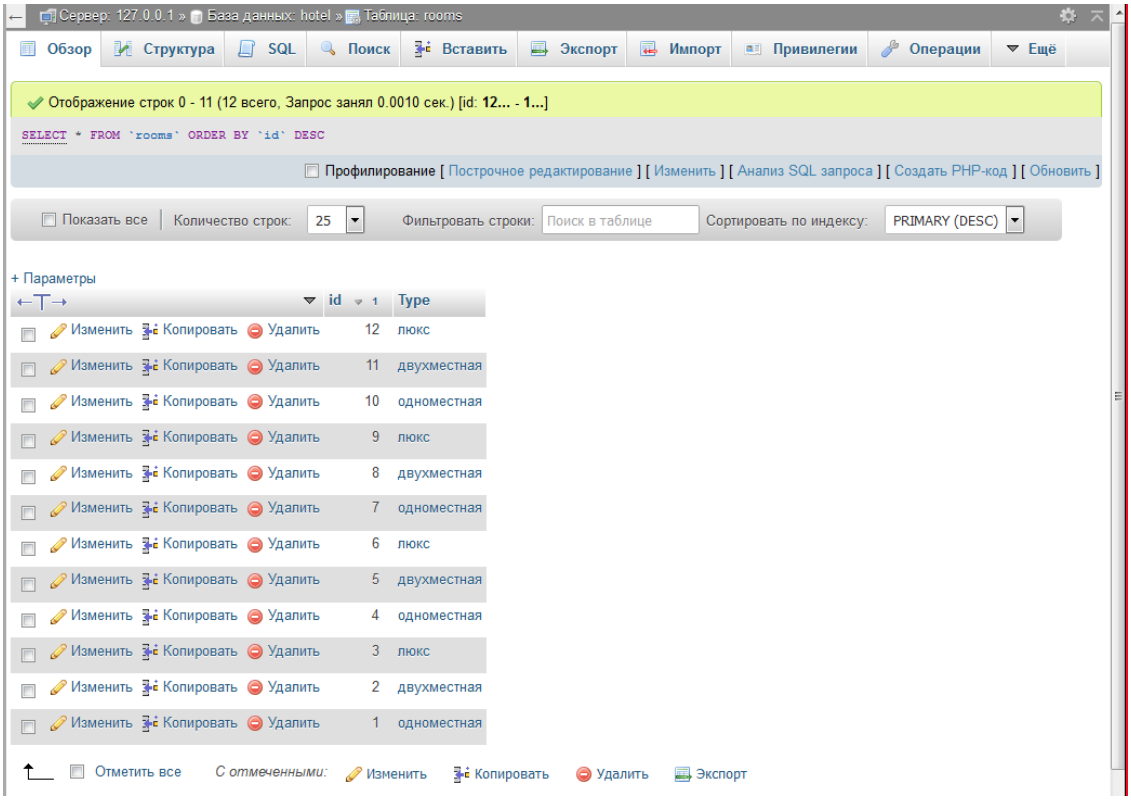


Рис. 2.7. Заповнена таблиця «НОМЕРИ»

Таблиця «ОПИС НОМЕРА», в котрій міститься інформація о типах кімнат та вартість проживання за добу.

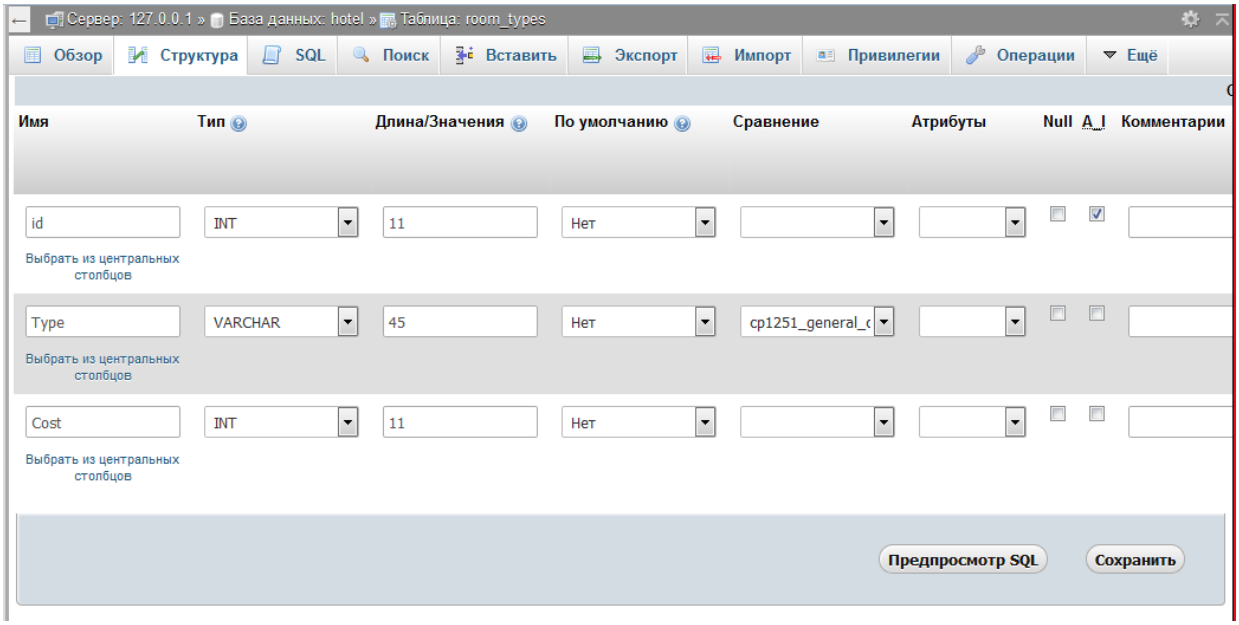


Рис. 2.8. Створення таблиці «ОПИС НОМЕРА»

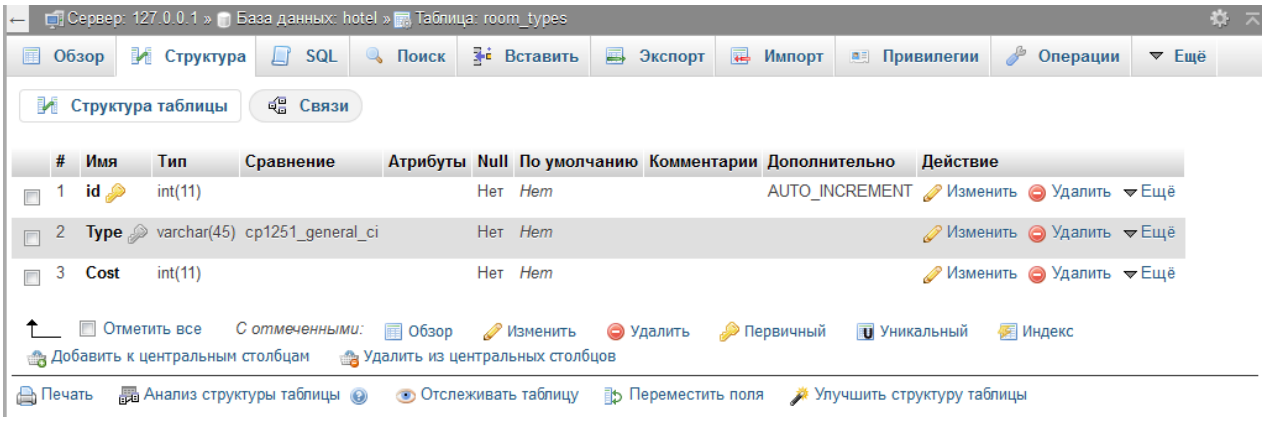


Рис. 2.9. Готова таблиця «ОПИС НОМЕРА»

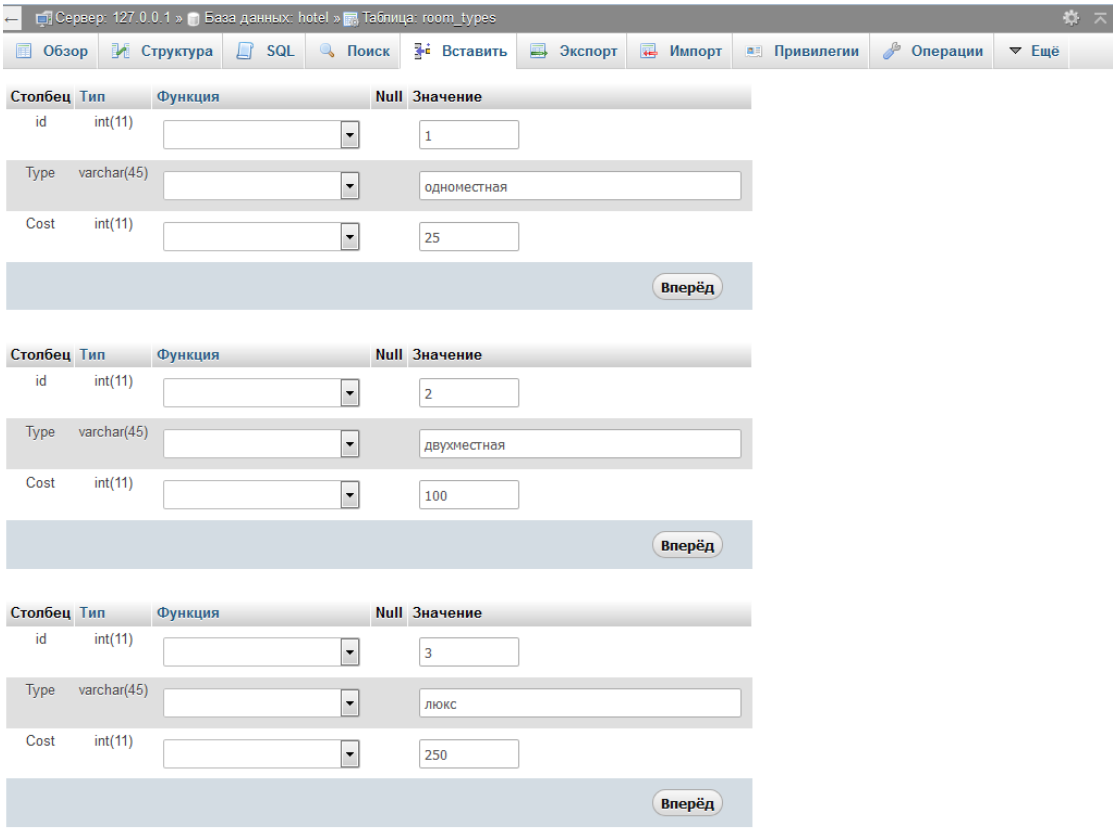


Рис. 2.10. Заповнення таблиці «ОПИС НОМЕРА»

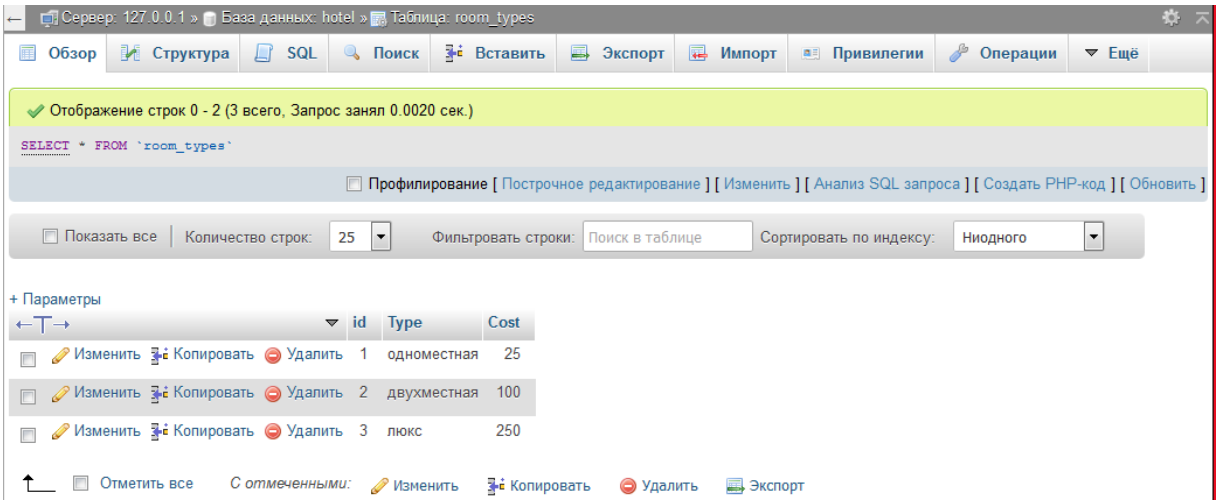


Рис. 2.11. Заповнена таблиця «ОПИС НОМЕРА»

Таблиця «КЛІЄНТИ» містить інформацію про клієнтів: ПІБ, номер телефону та адресу проживання.

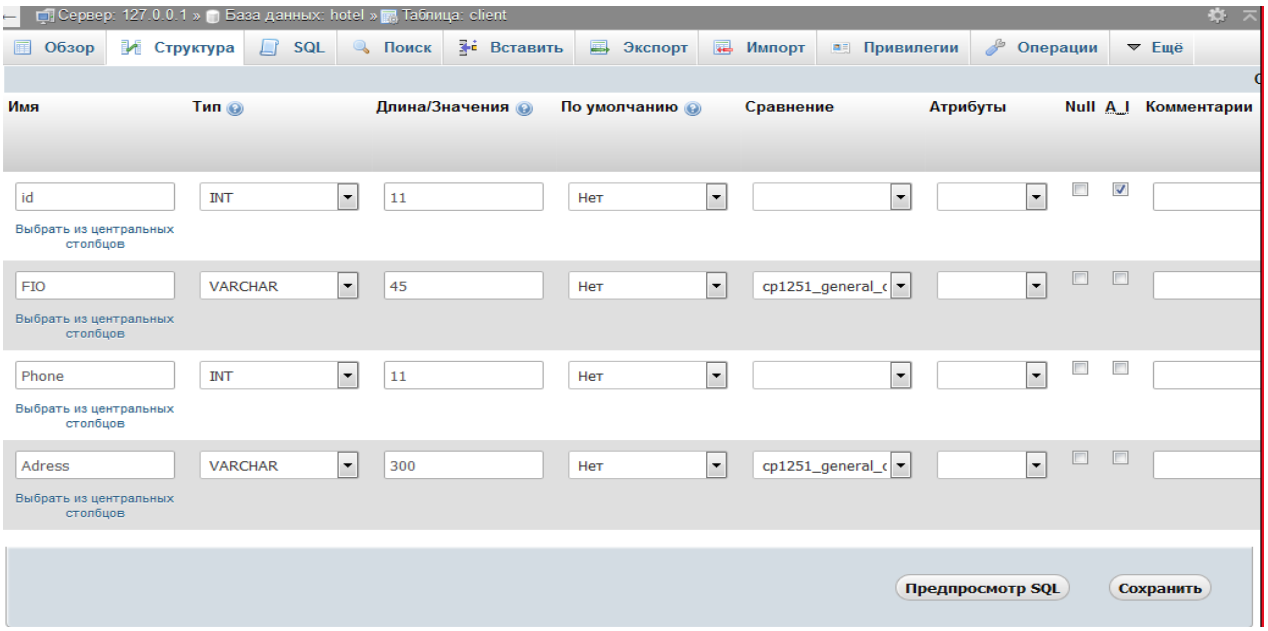


Рис. 2.12. Створення таблиці «КЛІЄНТИ»

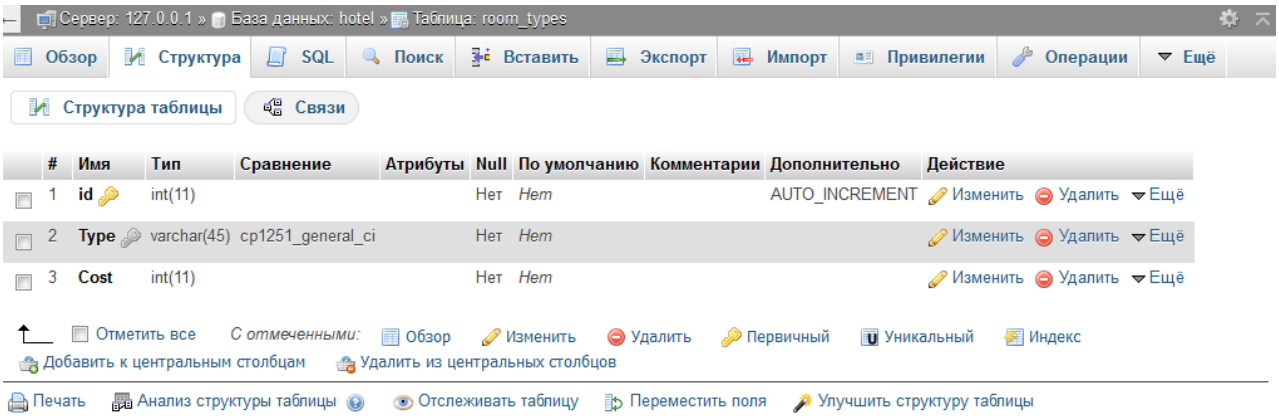


Рис. 2.13. Готова таблиця «КЛІЄНТИ»

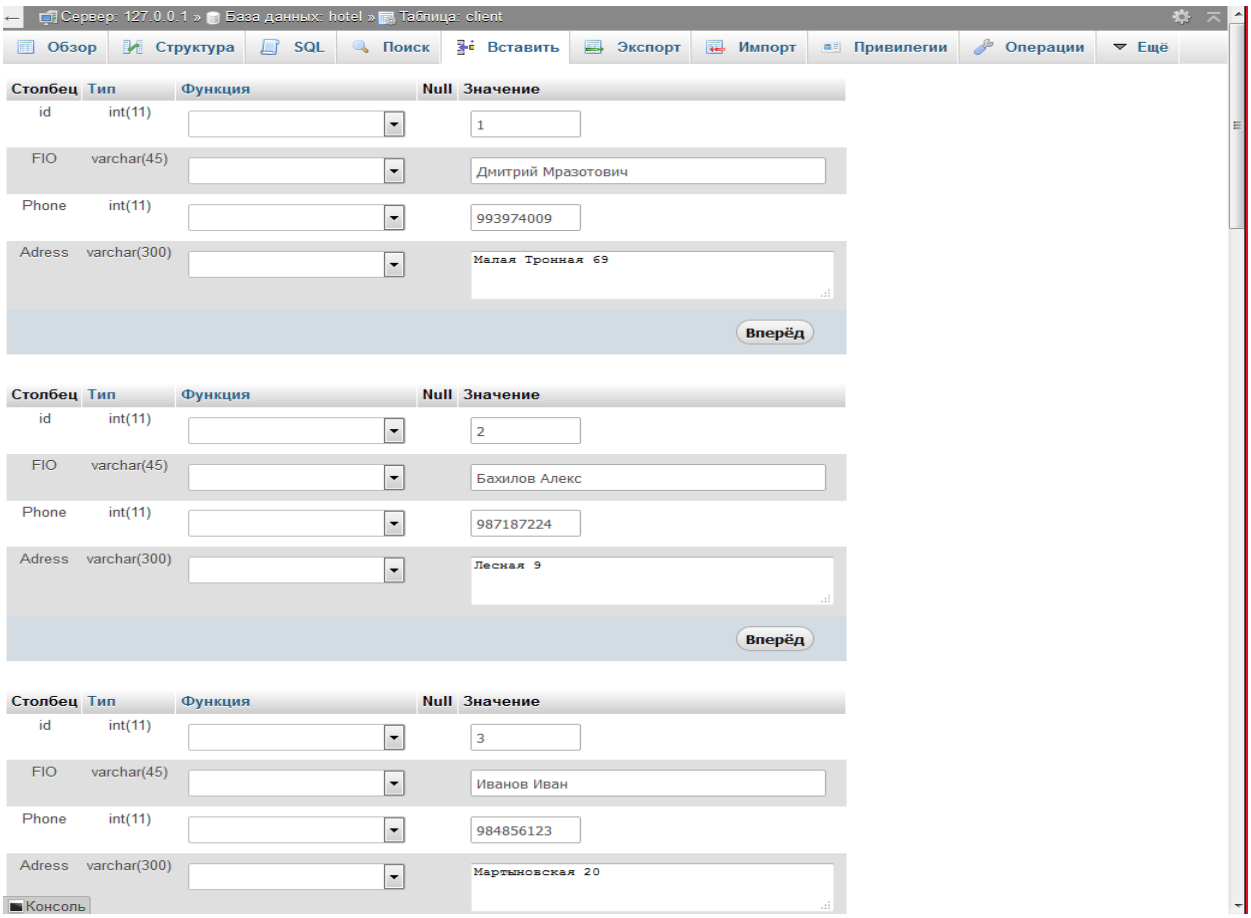


Рис. 2.14. Заповнення таблиці «КЛІЄНТИ»

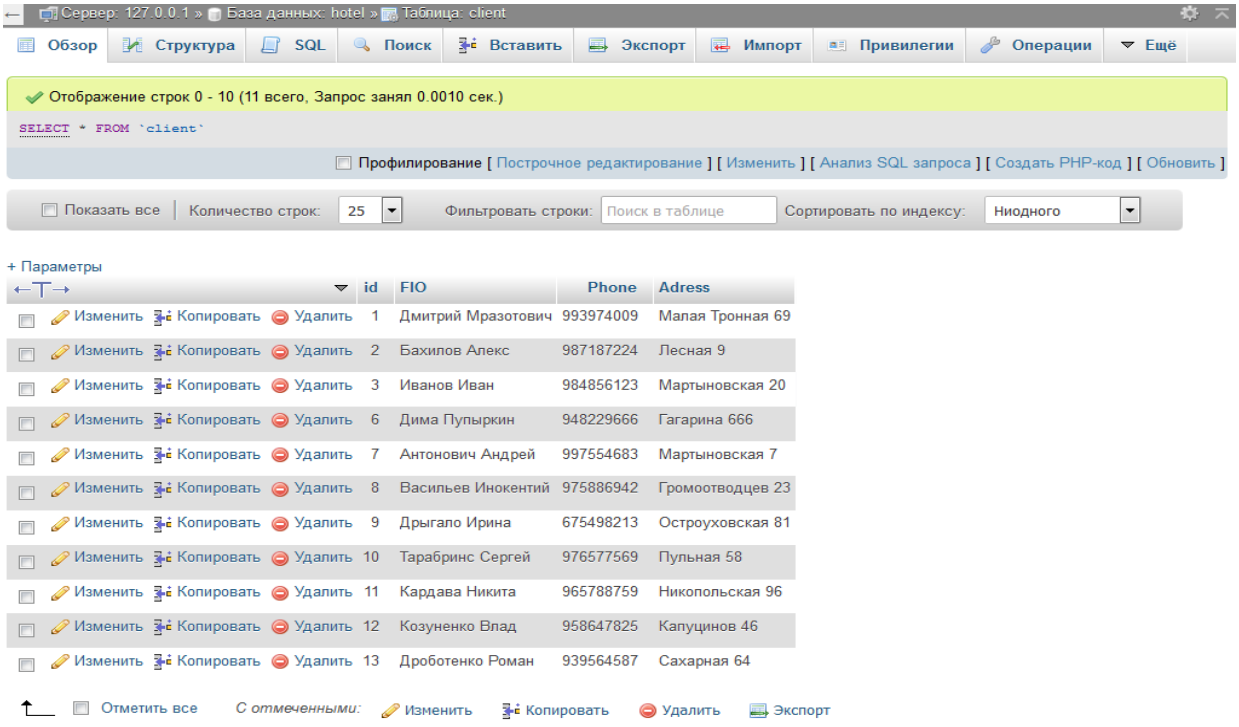


Рис. 2.15. Заповнена таблиця «КЛІЄНТИ»

Таблица «ЗАМОВЛЕННЯ» містить інформацію про дату прибуття, виїзду, дату оформлення заказу, ПІБ клієнта, номер кімнати та загальна вартість проживання.

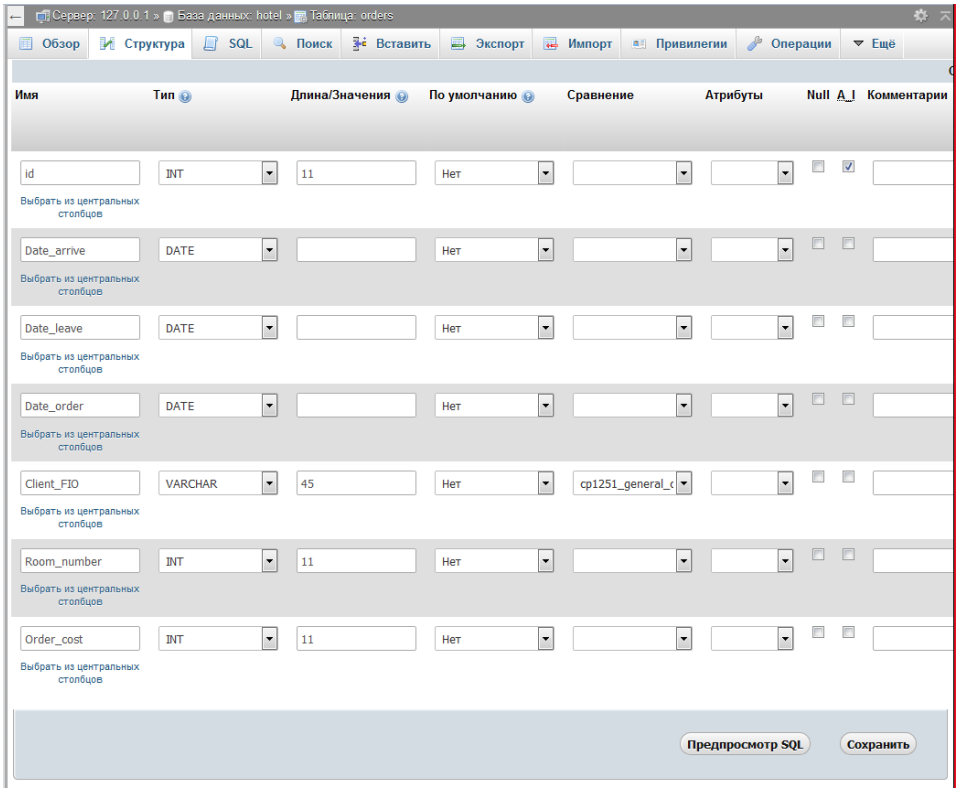


Рис. 2.16. Створення таблиці «ЗАМОВЛЕННЯ»

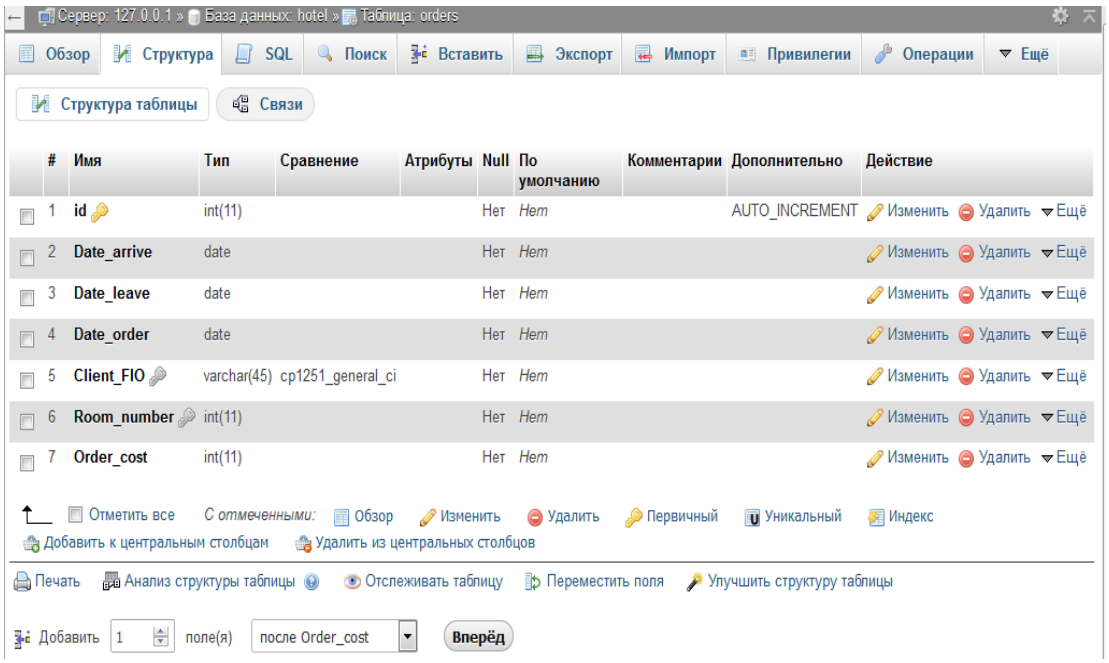


Рис. 2.17. Готова таблиця «ЗАМОВЛЕННЯ»

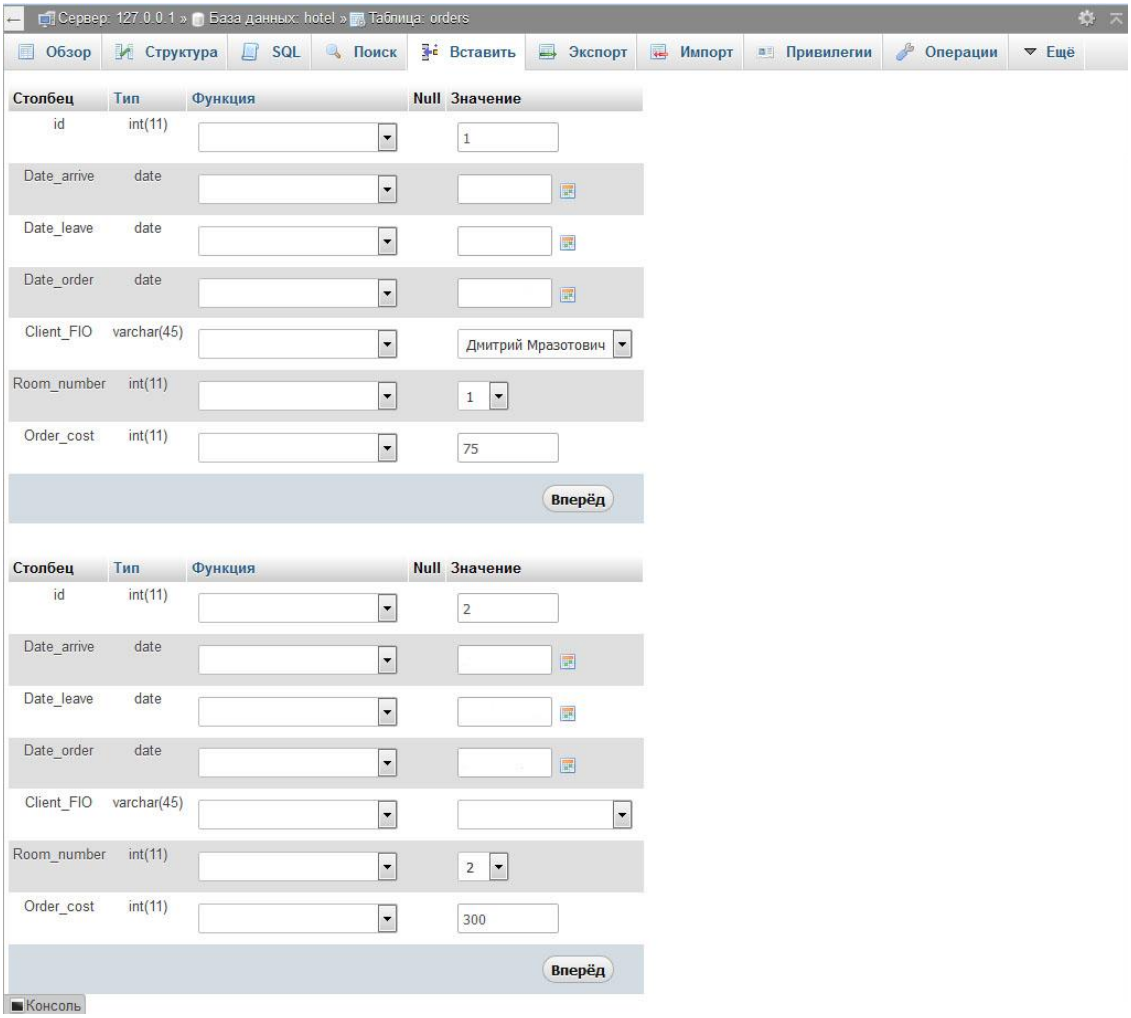


Рис. 2.18. Заповнення таблиці «ЗАМОВЛЕННЯ»

2.5. Обгрунтування та організація вхідних та вихідних даних програми

В якості вхідних даних в програмі використовується інформація про клієнтів, дати замовлення та інша інформація, що заноситься до БД (див. п.2.4.).

В якості вихідних даних в програмі використовуються файли звіту роботи програми.

2.6. Опис роботи розробленої системи

2.6.1. Використані технічні засоби

При розробці інформаційної системи допомоги адміністратора готелю було використано робочу станцію на базі ноутбука Dell Inspiron 3584 з такими

характеристиками: Екран 15.6" TN (1920x1080) Full HD, / Intel Core i3-7020U (2.3 ГГц) / RAM 4 ГБ / HDD 1 ТБ / Intel UHD Graphics 620 / LAN / Wi-Fi / Bluetooth / вебкамера.

2.6.2. Використані програмні засоби

2.6.2.1. Фреймворк Hibernate

Hibernate - бібліотека для мови програмування Java, призначена для вирішення завдань об'єктно-реляційного відображення (ORM), поширюється вільно на умовах GNU Lesser General Public License.

Метою Hibernate є звільнення розробника від значного обсягу порівняно низкоуровневого програмування при роботі в об'єктно-орієнтованих засобах в реляційної базі даних. Розробник може використовувати Hibernate як в процесі проектування системи класів і таблиць «з нуля», так і для роботи з уже існуючою базою даних.

Бібліотека не тільки вирішує завдання зв'язку класів Java з таблицями бази даних (і типів даних Java з типами даних SQL), але і також надає кошти для автоматичної генерації і оновлення набору таблиць, побудови запитів і обробки отриманих даних і може значно зменшити час розробки, яке зазвичай витрачається на ручне написання SQL- і JDBC-коду. Hibernate автоматизує генерацію SQL-запитів і звільняє розробника від ручної обробки результуючого набору даних і перетворення об'єктів, максимально полегшуючи перенесення (портирование) додатки на будь-які бази даних SQL.

Hibernate забезпечує прозору підтримку збереження даних (persistence) для «POJO» (тобто для стандартних Java-об'єктів); єдине сувора вимога для зберігається класу - наявність конструктора за замовчуванням (без параметрів). Для коректної поведінки в деяких додатках потрібно також приділити увагу методам equals () і hashCode ().

Mapping (зіставлення, проектування) Java-класів з таблицями бази даних здійснюється за допомогою конфігураційних XML-файлів або Java-анотацій. При використанні файлу XML Hibernate може генерувати скелет вихідного коду для класів тривалого зберігання. У цьому немає необхідності, якщо використовується анотація. Hibernate може використовувати файл XML або анотації для підтримки схеми бази даних.

Забезпечуються можливості по організації відносини між класами «один-ко-многим» і «багато-до-багатьох». На додаток до управління зв'язками між об'єктами Hibernate також може управляти рефлексивними відносинами, де об'єкт має зв'язок «один-ко-многим» з іншими екземплярами свого власного типу даних.

Hibernate підтримує відображення для користувача типів значень. Це робить можливими такі сценарії:

• Перевизначення типу за замовчуванням SQL, Hibernate вибирає при відображенні стовпчика властивості.

• Проекція перераховується типу Java на поле БД, ніби вони є звичайними властивостями.

• Проекція одного властивості в кілька колонок.

Колекції об'єктів даних, як правило, зберігаються у вигляді колекцій Java-об'єктів, таких, як набір (Set) і список (List). Підтримуються узагальнені класи (Generics), введені в Java 5. Hibernate може бути налаштований на «ліниві» (відкладені) завантаження колекцій. Відкладені завантаження є варіантом за умовчанням, починаючи з Hibernate 3.

Пов'язані об'єкти можуть бути налаштовані на каскадні операції. Наприклад, батьківський клас Album (музичний альбом) може бути налаштований на каскадне збереження і / або видалення свого нащадка Track. Це може скоротити час розробки і забезпечити цілісність. Функція перевірки зміни даних (dirty checking) дозволяє уникнути непотрібного запису дій в базу даних, виконуючи SQL-оновлення тільки при зміні полів персистентних об'єктів.

Успіх бібліотеки Hibernate підштовхнув JCP до розробки специфікації JDO, що стала однією з стандартних технологій ORM на платформі JavaEE. Також Hibernate сумісна з JSR-220/317 і надає стандартні засоби JPA.

Hibernate забезпечує використання SQL-подібного мови Hibernate Query Language (HQL), який дозволяє виконувати SQL-подібні запити, записані поруч з об'єктами даних Hibernate. Запити критеріїв надаються як Об'єктно-орієнтована альтернатива до HQL.

2.6.2.2. Фреймворк Swing

Swing - бібліотека для створення графічного інтерфейсу для програм на мові Java. Swing був розроблений компанією Sun Microsystems. Він містить ряд графічних компонентів (англ. Swing widgets), таких як кнопки, поля введення, таблиці і т. Д.

Swing відноситься до бібліотеки класів JFC, яка представляє собою набір бібліотек для розробки графічних оболонок. До цих бібліотек відносяться Java 2D, Accessibility-API, Drag & Drop-API і AWT.

Swing надає більш гнучкі інтерфейсні компоненти, ніж більш рання бібліотека AWT. На відміну від AWT, компоненти Swing розроблені для однаковою крос-платформної роботи, в той час як компоненти AWT повторюють інтерфейс виконуваної платформи без змін. AWT ж використовує тільки стандартні елементи ОС для відображення, тобто для кожного елементу створюється окремий об'єкт ОС (вікно), в зв'язку з чим, AWT не дозволяє створювати елементи довільної форми (можливо використовувати тільки прямокутні компоненти), елементи керування на основі AWT завжди відображаються поверх Swing-елементів (так як все Swing компоненти відображаються на поверхні контейнера).

Компоненти Swing підтримують специфічні динамічно підключаються види і поведінки (с англ. Plugable look-and-feel), завдяки якому можлива адаптація до графічного інтерфейсу платформи (тобто до компоненту можна

динамічно скористатись іншим, специфічний для операційної системи, в тому числі і створений програмістом вигляд і поведінку). Таким чином, функції, які залежать Swing, можуть виглядати як рідні додатки для даної операційної системи. Основним мінусом таких «легких» (англ. Lightweight) компонентів є щодо повільна робота. Позитивна сторона - універсальність інтерфейсу створених додатків на всіх платформах.

«Lightweight» означає, що компоненти Swing отрісовиваємих самими компонентами на поверхні батьківського вікна, без використання компонентів операційної системи. На відміну від «Важких» компонентів AWT, в додатку Swing може бути тільки одне вікно, і всі інші компоненти отрісовиваємих на найближчому батьку, що має власне вікно (наприклад, на JFrame). У додатку можуть поєднуватися Swing і AWT елементи, хоча це може породжувати деякі проблеми - зокрема, компоненти AWT завжди перекривають Swing елементи, а також закривають собою спливаючі меню JPopupMenu і JComboBox. Для запобігання цьому, у цих компонентів є методи setLightWeightPopupEnabled (boolean), що дозволяють заборонити використання «легковагих» спливаючих елементів. При установці властивості в true (setLightWeightPopupEnabled (true)), AWT елементи не будуть перекривати меню.

2.6.2.3. Інтегроване середовище розробки Intellij IDEA

IntelliJ IDEA - інтегроване середовище розробки програмного забезпечення на багатьох мовах програмування, зокрема Java, JavaScript, Python, розроблена компанією JetBrains.

Мови: Java, JavaScript, CoffeeScript, HTML / XHTML / HAML, CSS / SASS / LESS, XML / XSL / XPath, YAML, ActionScript / MXML, Python, Ruby, Haxe, Groovy, Scala, SQL, PHP, Kotlin, Clojure , Ci, C ++.

Інтерфейс середовища розробки Intellij IDEA зображений на рис. 2.1.

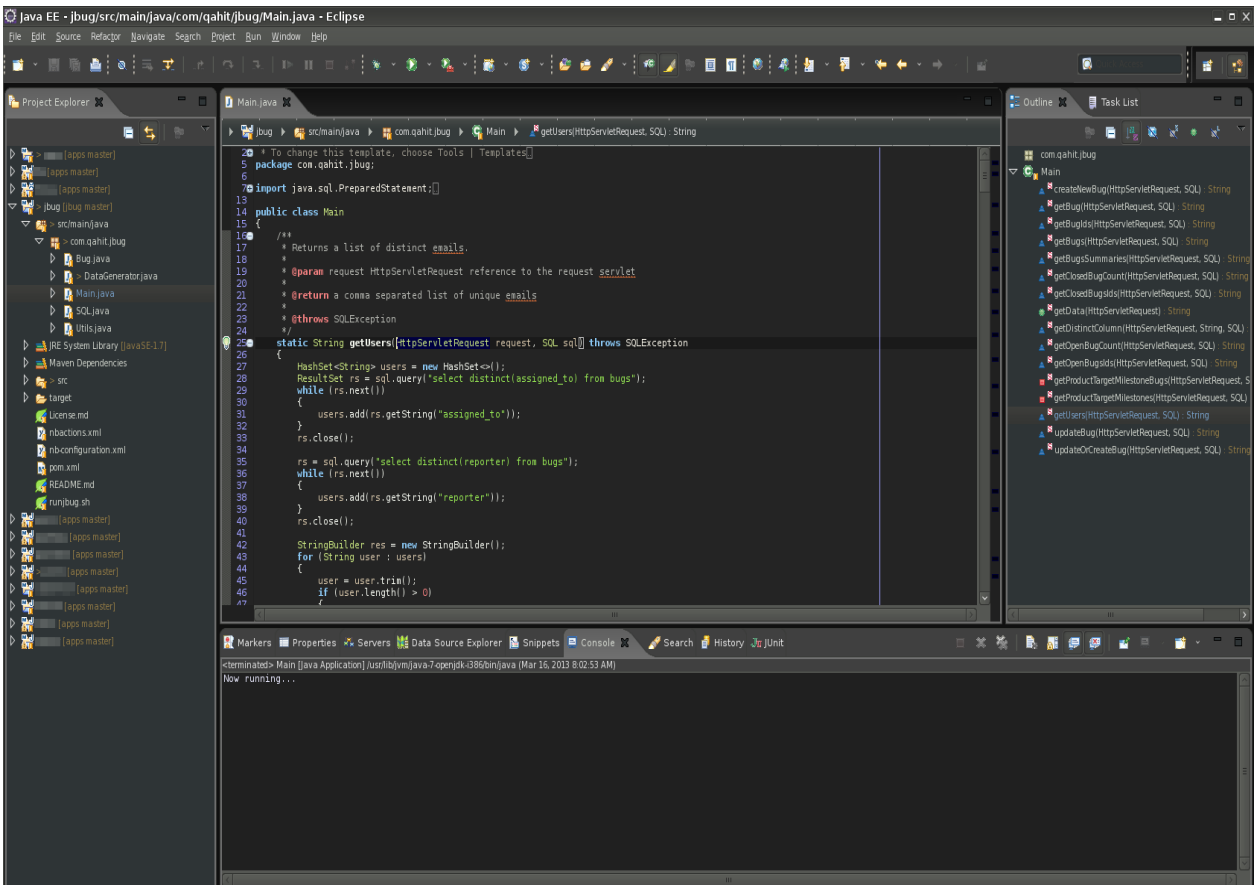


Рис. 2.4. Інтерфейс Intellij IDEA

Перша версія з'явилася в січні 2001 року і швидко стала популярною, як перша середа для Java з широким набором інтегрованих інструментів для рефакторинга, які дозволяли програмістам швидко реорганізовувати вихідні тексти програм. Дизайн середовища орієнтований на продуктивність роботи програмістів, дозволяючи сконцентруватися на функціональних завданнях, в той час як IntelliJ IDEA бере на себе виконання рутинних операцій.

Починаючи з шостої версії продукту IntelliJ IDEA надає інтегрований інструментарій для розробки графічного інтерфейсу користувача. Серед інших можливостей, середа добре сумісна з багатьма популярними вільними інструментами розробників, такими як CVS, Subversion, Apache Ant, Maven і JUnit. У лютому 2007 року розробники IntelliJ анонсували ранню версію плагіна для підтримки програмування на мові Ruby.

Починаючи з версії 9.0, середа доступна в двох редакціях: Community Edition і Ultimate Edition. Community Edition є повністю вільною версією,

доступною під ліцензією Apache 2.0, в ній реалізована повна підтримка Java SE, Groovy, Scala, а також інтеграція з найбільш популярними системами управління версіями. В редакції Ultimate Edition реалізована підтримка Java EE, UML-діаграм, підрахунок покриття коду, а також підтримка інших систем управління версіями, мов та фреймворків.

Також, що не менш важливо, має ряд широко поширених фреймворків (наприклад, Spring), які легко настроюються і можуть бути підключені до проекту в пару клацань.

2.6.3. Виклик та завантаження програми

Спосіб виклику програми з відповідного носія даних та умови його завантаження є стандартними для запуску виконуючих файлів при роботі в ОС Windows. Додаткових чи специфічних вимог щодо запуску програми не встановлено, програма не потребує спеціального завантаження та налаштування. Для роботи потрібен ПК чи ноутбук з ОС Windows.

2.6.4. Опис інтерфейсу користувача

При запуску програми з'являється головне вікно. Інтерфейс побудований таким чином, що будь-який користувач, навіть не працював до цього з комп'ютером, може здогадатися про призначення кожного компонента. Інтерфейс має дуже зручний вид та інтуітивно зрозумілий інтерфейс. Вікно головної форми має меню у відповідність з рис. 2.20, що дозволяє здійснювати навігацію по програмі

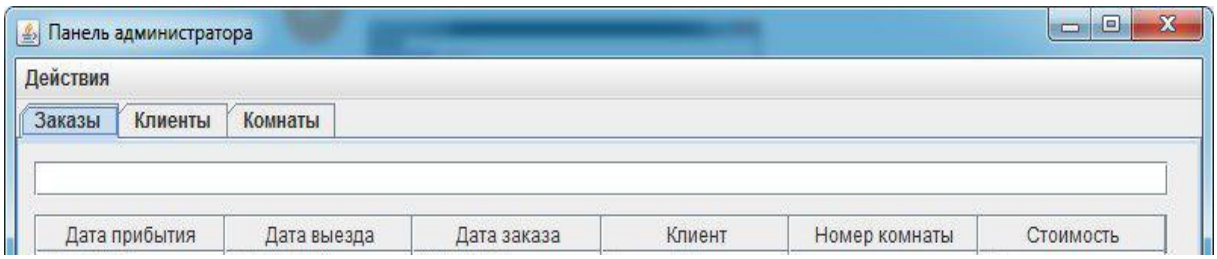
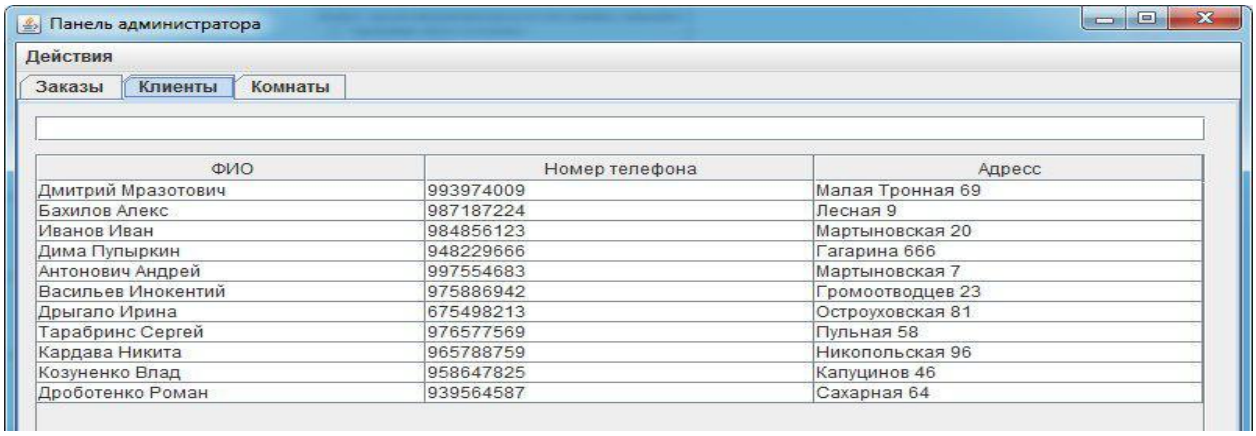


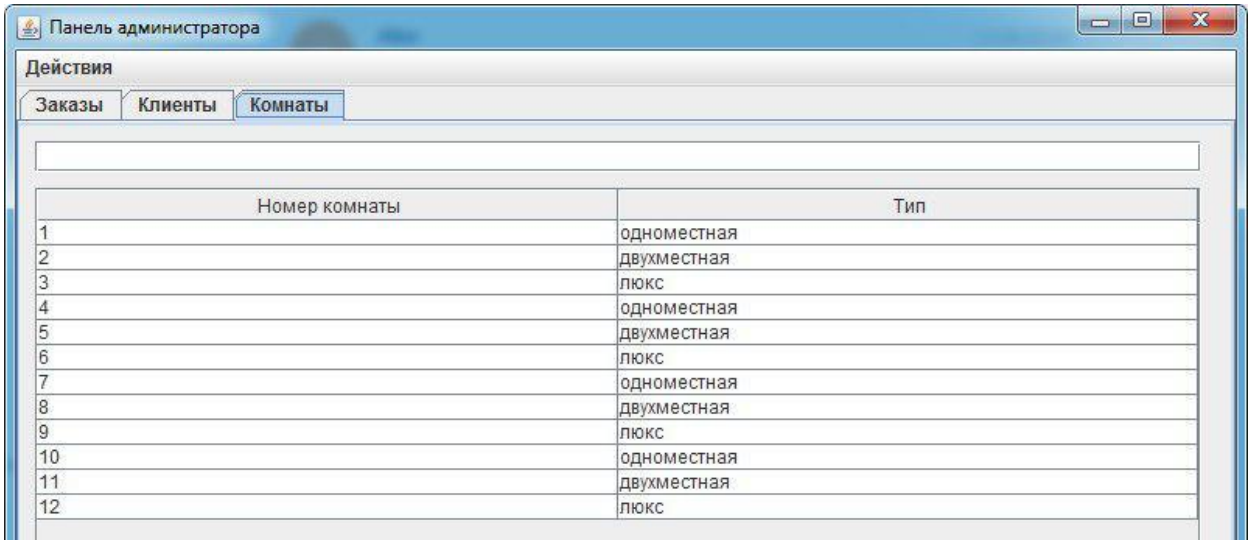
Рис.2.20. Главное окно

Після відкриття головної форми у нас з'являється можливість почати роботу в програмі. На головній формі розташовано три вкладки на відображення інформації про замовлення, клієнтів і номерах. На рис. 2.21. і рис. 2.22. зображені вкладки Клієнти і Кімнати.



ФИО	Номер телефона	Адресс
Дмитрий Мразотович	993974009	Малая Тронная 69
Бахилов Алекс	987187224	Лесная 9
Иванов Иван	984856123	Мартыновская 20
Дима Пупыркин	948229666	Гагарина 666
Антонович Андрей	997554683	Мартыновская 7
Васильев Инокентий	975886942	Громоотводцев 23
Дрыгало Ирина	675498213	Остроуховская 81
Тарабринс Сергей	976577569	Пульная 58
Кардава Никита	965788759	Никопольская 96
Козуненко Влад	958647825	Капуцинов 46
Дроботенко Роман	939564587	Сахарная 64

Рис. 2.21. Вкладка Клієнти



Номер комнаты	Тип
1	одноместная
2	двухместная
3	люкс
4	одноместная
5	двухместная
6	люкс
7	одноместная
8	двухместная
9	люкс
10	одноместная
11	двухместная
12	люкс

Рис. 2.22. Вкладка Комнаты

На кожній з вкладок може здійснюватися пошук по довільному параметру. На рис. 2.23. зображений пошук за таблицею «Кімнати».

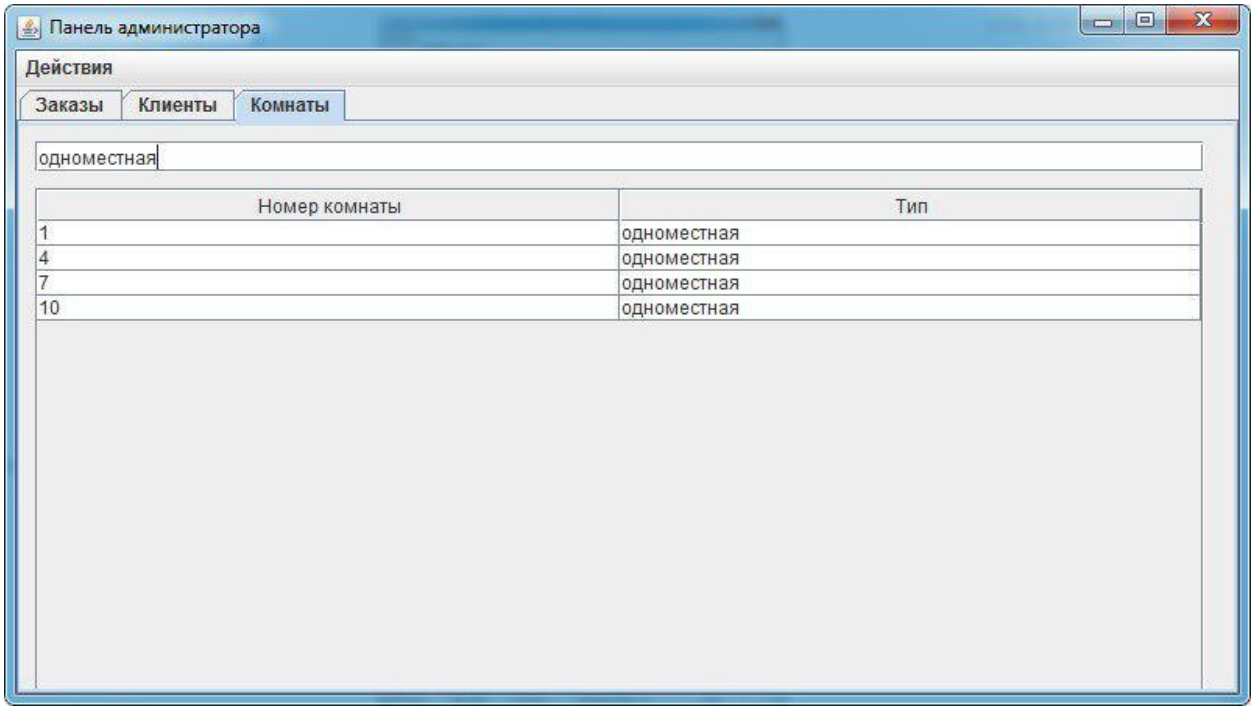


Рис. 2.23. Пошук по таблиці Кімнати

На головній формі присутній меню, що дозволяє здійснювати навігацію по програмі. Меню зображено на рис. 2.24.

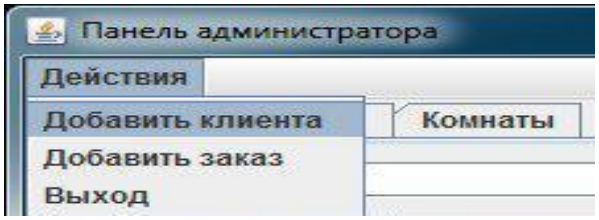


Рис. 2.24. Меню

Для того, щоб відкрити форму з занесенням інформації про нового клієнта, необхідно на головній формі натиснути пункт «Додати клієнта» в меню «Дії», і відкриється форма відповідно до рис. 2.25.

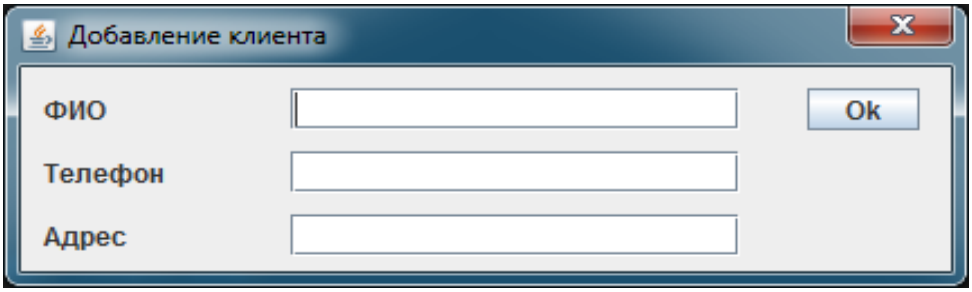


Рис. 2.25. Додавання клієнта

Для того, щоб відкрити форму з занесенням інформації про нове замовлення, необхідно на головній формі натиснути пункт «Додати замовлення» в меню «Дії», і відкриється форма відповідно до рис. 2.26.

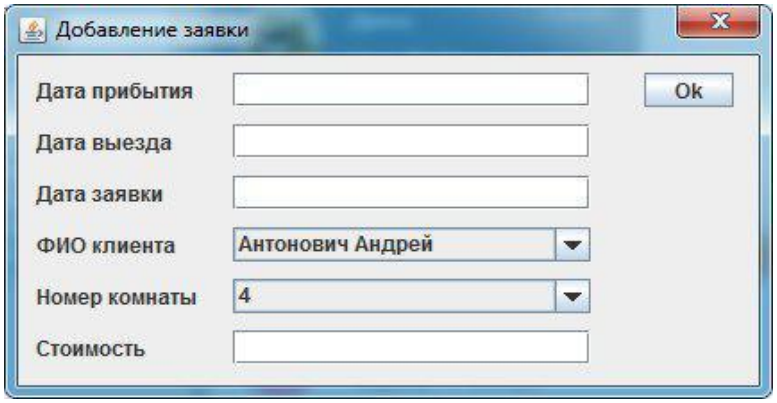


Рис. 2.26. Додавання замовлення

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Початкові дані:

1. передбачуване число операторів програми – 1300;
2. коефіцієнт складності програми – 1,6;
3. коефіцієнт корекції програми в ході її розробки – 0,05;
4. годинна заробітна плата програміста– 60 грн/год;
5. коефіцієнтзбільшеннявитратпрацівнаслідокнедостатньогоопи сузадачі – 1,2;
6. коефіцієнткваліфікаціїіпрограміста, обумовленийвід стажу роботи з даноїспеціальності – 1,2;
7. вартість машино-години ЕОМ –13 грн/год.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д, \text{ людино-годин,} \quad (3.1)$$

де t_o- витрати праці на підготовку й опис поставленої задачі (приймається 50 людино-годин);

t_u - витрати праці на дослідження алгоритму рішення задачі;

t_a- витрати праці на розробку блок-схеми алгоритму;

t_n-витрати праці на програмування по готовій блок-схемі;

$t_{отл}$-витрати праці на налагодження програми на ЕОМ;

$t_д$ - витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у програмногму забезпеченні, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p),$$

де q - передбачуване число операторів (1300);

C - коефіцієнт складності програми (1,6);

p - коефіцієнт корекції програми в ході її розробки (0,05).

Звідси умовне число операторів в програмі:

$$Q = 1{,}6 \cdot 1300 \cdot (1 + 0{,}05) = 2184$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75..85) \cdot k}\ \text{, людино-годин,}$$

де B - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

k - коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності.При стажі роботи від 3 до 5 років він складає 1,2.

Приймемо збільшення витрат праці внаслідок недостатнього опису завдання не більше 50% (B = 1,2). З урахуванням коефіцієнта кваліфікації k = 1,2, отримуємо витрати праці на вивчення опису завдання:

$$t_u = (2184 \cdot 1{,}2) / (75 \cdot 1{,}2) = 29{,}12 \text{ людино-годин}$$

Витрати праці на розробку алгоритму рішення задачі визначаються за формулою:

$$t_a = \frac{Q}{(20\ldots25)\cdot k}\ \text{, людино-годин,}\quad (3.2)$$

де Q – умовне число операторів програми;

k – коефіцієнт кваліфікації програміста.

Підставивши відповідні значення в формулу (3.2), отримаємо:

$$t_a = 2184 / (20 \cdot 1{,}2) = 91 \text{людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20\ldots25)\cdot k}\ \text{, людино-годин.}$$

$$t_n = 2184 / (20 \cdot 1{,}2) = 91 \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

- за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4..5)\cdot k}\ \text{, людино-годин.}$$

$$t_{отл} = 2184 / (5 \cdot 1{,}2) = 364 \text{ чел.-ч.}$$

- за умови комплексного налагодження завдання:

$$t^k_{отл} = 1{,}5 \cdot t_{отл}\ \text{, людино-годин.}$$

$$t^{k}_{отл} = 1,5 \cdot 364 = 546 \text{ людино-годин.}$$

Витрати праці на підготовку документації визначаються за формулою:

$$t_{д} = t_{др} + t_{до} \text{ , людино-годин,}$$

де $t_{др}$-трудомісткість підготовки матеріалів і рукопису:

$$t_{др} = \frac{Q}{(15..20) \cdot k} \text{ , людино-годин,}$$

$t_{до}$ - трудомісткість редагування, печатки й оформлення документації:

$$t_{до} = 0,75 \cdot t_{др} \text{ , людино-годин.}$$

Підставляючи відповідні значення, отримаємо:

$$t_{др} = 2184 / (18 \cdot 1,2) = 101,11 \text{ людино-годин.}$$

$$t_{до} = 0,75 \cdot 101,11 = 75,83 \text{ людино-годин.}$$

$$t_{д} = 101,11 + 75,83 = 176,94 \text{ людино-годин.}$$

Повертаючись до формули (3.1), отримаємо повну оцінку трудомісткості розробки програмного забезпечення:

$$t = 50 + 29,12 + 91 + 91 + 364 + 176,94 = 802,06 \text{ людино-годин.}$$

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ $K_{ПО}$ включають витрати на заробітну плату виконавця програми $З_{ЗП}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ:

$$K_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн.}$$

Заробітна плата виконавців визначається за формулою:

$$З_{ЗП} = t \cdot C_{ПР}, \text{ грн,}$$

де: t - загальна трудомісткість, людино-годин;

$C_{ПР}$ - середня годинна заробітна плата програміста, грн/година

З урахуванням того, що середня годинна зарплата програміста становить 60 грн / год, отримуємо:

$$З_{ЗП} = 802{,}06 \cdot 60 = 48123{,}87 \text{ грн.}$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$З_{мв} = t_{отл} \cdot C_{мч}, \text{ грн,} \quad (3.3)$$

де $t_{отл}$ - трудомісткість налагодження програми на ЕОМ, год;

$C_{мч}$ - вартість машино-години ЕОМ, грн/год (13 грн/год).

Підставивши в формулу (3.3) відповідні значення, визначимо вартість необхідного для налагодження машинного часу:

$$З_{мв} = 364 \cdot 13 = 4732 \text{ грн.}$$

Звідси витрати на створення програмного продукту:

$$К_{ПО} = 48123{,}87 + 4732 = 52855{,}87 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p}, \text{ міс.}$$

де B_k- число виконавців (дорівнює 1);

F_p - місячний фонд робочого часу (при 40 годинному робочому тижні F_p=176 годин).

Звідси витрати на створення програмного продукту:

$$T = 802{,}06 / 1 \cdot 176 \approx 4{,}5 \text{ міс.}$$

Висновок: програмне забезпечення розроблено для підвищення якості роботи адміністратора готелю. Вартість даного програмного забезпечення становить 52 тис. грн. і не вимагає додаткових витрат як при розробці програми. Очікуваний час розробки становить 4,5 місяці. Цей термін пов'язаний зі значним числом операторів, і включає час на дослідження і розробку алгоритму вирішення поставленого завдання, програмування по готовому алгоритму, налагодження програми і підготовку документації.

ВИСНОВКИ

В кваліфікаційній роботі було розроблено інформаційну систему для роботи адміністратора готелю. Розроблена система реалізована з використанням СУБД MySQL, мови програмування Java.

У роботі була розроблена структура таблиць бази даних, яка надалі була заповнена довільними даними про клієнтів, кімнатах і замовленнях.

Для наочного уявлення отриманих даних аналізу було розроблено додаток для відображення і заповнення бази даних, також реалізовані Hibernate-запити, використання яких дозволяє отримати дані з БД у вигляді об'єктів, описаних в mapping-класах.

При впровадженні даної системи в роботу адміністратора готелю буде підвищено ефективність ведення, зберігання і обробки даних що надходять. Також у кваліфікаційній роботі було визначено трудомісткість розробленого програмного продукту 615 люд-год, проведений підрахунок вартості роботи по створенню програми 43 863 грн та розраховано час на його створення 3 місяці.

Впровадження даного програмного продукту є економічно вигідним, оскільки його повна вартість є помірною за рахунок використання некомерційних інструментів для розробки, відсутності необхідності конфігурації і обслуговування робочих місць користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Джошуа Блох. Java. Эффективное программирование; — М.: «Вильямс», 2014. – 440 с.
2. Роберт Седжвик, Кевин Уэйн. Алгоритмы на Java; — М.: «Вильямс», 2013. – 843 с.
3. Герберт Шилдт. Java 8. Полное руководство, 9-е издание = Java 8. The Complete Reference, 9th Edition. — М.: «Вильямс», 2015. — 1376 с.
4. Брюс Эккель. Философия Java; — М.: «Вильямс», 2015. - 1168 с.
5. Барри Берд. Java 8 для чайников = Java For Dummies, 6th edition. - М .: «Диалектика», 2015. – 400 с.
6. Джеймс Гослинг, Билл Джой, Гай Стил, Гилад Брача, Алекс Бакли. Язык программирования Java SE 8. Подробное описание, 5-е издание = The Java Language Specification, Java SE 8 Edition (5th Edition) (Java Series). — М.: «Вильямс», 2015. — 672 с.
7. Патрик Нимейер, Дэниэл Леук. Программирование на Java. Исчерпывающее руководство для профессионалов; — М.: «Вильямс», 2014. – 1216 с.
8. Mark Matthews, Jim Cole, Joseph D. Gradecki. MySQL and Java Developer's Guide; — М.: «Вильямс», 2003. – 432 с.
9. Боуман, С.Эмерсон, М.Дарновски. Практическое руководство SQL; — М.: «Вильямс», 2000. – 321 с.
10. Грофф Дж. Р., Вайнберг П.Н., Оппелъ Э. Дж. SQL. Полное руководство; — М.: «Вильямс», 2015. – 959 с.
11. Ларри Ульман. MySQL. Руководство по изучению языка; — М.: «Вильямс», 2004. – 352 с.
12. Д. Кренке. Теория и практика построения баз данных; — М.: «Вильямс», 2005. – 864 с.
13. Леон Аткинсон. MySQL. Библиотека профессионала; — М.: «Вильямс», 2002. – 624 с.

14. Marc Delisle. Mastering phpMyAdmin 3.1 for Effective MySQL Management; — М.: «Вильямс», 2009. – 352 с.

15. Andy Oppel. Databases A Beginner's Guide; — М.: «Вильямс», 2009. – 496 с.

16. Советов Б.Я., Цехановский В.В., Чертовской В.Д. Базы данных. Теория и практика; — М.: «Вильямс», 2014. – 464с.

17. Базы данных: основные понятия [Электрон. ресурс]. – Способ доступа: URL: http://www.webmasterwiki.ru/MySQL

18. Документация по MySQL [Электрон. ресурс]. – Способ доступа: URL: http://www.sql.ru/docs/mysql/rus_ref/

19. Apache: установка и настройка веб-сервера [Электрон. ресурс]. – Способ доступа: URL: http://www.internet-technologies.ru/articles/article_1747.htmlp/

20. Установка и настройка сервера XAMPP на Windows [Электрон. ресурс]. – Способ доступа: URL: http://makegood.ru/tools/xampp/

21. XAMPP [Электрон. ресурс]. – Способ доступа: URL: http://htmlbook.ru/webserver/xampp

22. Java Hibernate [Электрон. ресурс]. – Способ доступа: URL: http://alfalavista.ru/idxfldr/2013-06-18-22-25-47/302-java-hibernate.html

23. Г. Шилдт. Swing руководство для начинающих; — М.: «Вильямс», 2007. – 705 с.

24. Библиотека Swing [Электрон. ресурс]. – Способ доступа: URL: http://java-online.ru/libs-swing.xhtml

25. Java Swing Tutorial [Электрон. ресурс]. – Способ доступа: URL: https://www.javatpoint.com/java-swing

26. Справочник по Java Swing GUI [Электрон. ресурс]. – Способ доступа: URL: http://www.darkraha.com/rus/java/swing/swing01.php

ДОДАТОК А

КОД ПРОГРАМИ

Main.java

```
import UI.MainFrame;

/**
/
public class Main {
    public static void main(String[] args) {
        new MainFrame();
    }
}
```

MainFrame.java

```
package UI;

/**
/
import Work.MySessionFactory;
import org.hibernate.SessionFactory;

import javax.swing.*;

public class MainFrame extends JFrame {

    public MainFrame() {
        setTitle("Панель администратора");
        setBounds(200, 300, 800, 450);
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        SessionFactory sf = MySessionFactory.getSessionFactory();

        JPanel panelF = new JPanel();
        JPanel panelS = new JPanel();
        JPanel panelT = new JPanel();

        JTabbedPane jtp = new JTabbedPane();

        Panel1 pan1 = new Panel1(sf);
        Panel2 pan2 = new Panel2(sf);
        Panel3 pan3 = new Panel3(sf);

        jtp.add("Заказы", panelF.add(pan1));
        jtp.add("Клиенты", panelS.add(pan2));
```

```
        jtp.add("Комнаты", panelT.add(pan3));

        getContentPane().add(jtp);
        setJMenuBar(new Menu(sf, pan2.tm, pan1.tm));

        setVisible(true);
    }
}
```

Menu.java

```
package UI;

import Work.ClientTable;
import Work.OrderTable;
import org.hibernate.SessionFactory;
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class Menu extends JMenuBar {
    public Menu(SessionFactory sf, ClientTable clientModel, OrderTable orderModel)
{
        JMenu actions = new JMenu("Действия");

        JMenuItem itm1 = new JMenuItem("Добавить клиента");
        actions.add(itm1);
        JMenuItem itm2 = new JMenuItem("Добавить заказ");
        actions.add(itm2);

        add(actions);

        itm1.addActionListener(e -> {
            AddClientFrame dialog = new AddClientFrame(sf.openSession(),
clientModel);
            dialog.setVisible(true);
        });

        itm2.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                AddOrderFrame dialog1 = new AddOrderFrame(sf.openSession(),
orderModel);
                dialog1.setVisible(true);
            }
        });
```

```
    }
}
```

Panel1.java

```
package UI;

import Work.OrderTable;
import org.hibernate.SessionFactory;

import javax.swing.*;
import javax.swing.table.TableRowSorter;

/**
/
public class Panel1 extends JPanel {
    OrderTable tm = null;

    public Panel1(SessionFactory sf) {
        setLayout(null);

        tm = new OrderTable(sf);
        JTable tbl = new JTable(tm);
        JScrollPane scr = new JScrollPane(tbl);
        scr.setBounds(10, 40, 750, 400);
        add(scr);

        TableRowSorter<OrderTable> sorter = new TableRowSorter<>(tm);
        tbl.setRowSorter(sorter);

        JTextField filter = new JTextField();
        filter.setBounds(10, 10, 750, 20);
        add(filter);
        filter.addCaretListener(e -> {
            try {
                sorter.setRowFilter(RowFilter.regexFilter("(?i)" + filter.getText()));
                sorter.setSortKeys(null);
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        });
    }


}
```

Panel2.java

```
package UI;
```

```
import Work.ClientTable;
import org.hibernate.SessionFactory;

import javax.swing.*;
import javax.swing.table.TableRowSorter;

/**
/
public class Panel2 extends JPanel {
    ClientTable tm = null;
    JTable tbl = null;

    public Panel2(SessionFactory sf) {
        setLayout(null);

        tm = new ClientTable(sf);
        tbl = new JTable(tm);
        JScrollPane scr = new JScrollPane(tbl);
        scr.setBounds(10, 40, 750, 400);
        add(scr);

        TableRowSorter<ClientTable> sorter = new TableRowSorter<>(tm);
        tbl.setRowSorter(sorter);

        JTextField filter = new JTextField();
        filter.setBounds(10, 10, 750, 20);
        add(filter);
        filter.addCaretListener(e -> {
            try {
                sorter.setRowFilter(RowFilter.regexFilter("(?i)" + filter.getText()));
                sorter.setSortKeys(null);
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        });
    }
}
```

Panel3.java

```
package UI;

import Work.RoomsTable;
import org.hibernate.SessionFactory;

import javax.swing.*;
```

```
import javax.swing.table.TableRowSorter;

/**
 * Created by Alex on 17.06.2017.
 */
public class Panel3 extends JPanel {

    public Panel3(SessionFactory sf) {
        setLayout(null);

        RoomsTable tm = new RoomsTable(sf);
        JTable tbl = new JTable(tm);
        JScrollPane scr = new JScrollPane(tbl);
        scr.setBounds(10, 40, 750, 400);
        add(scr);
        TableRowSorter<RoomsTable> sorter = new TableRowSorter<>(tm);
        tbl.setRowSorter(sorter);
        JTextField filter = new JTextField();
        filter.setBounds(10, 10, 750, 20);
        add(filter);
        filter.addCaretListener(e -> {
            try {
                sorter.setRowFilter(RowFilter.regexFilter("(?i)" + filter.getText()));
                sorter.setSortKeys(null);
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        });
    }
}
```

AddClientFrame.java

```
package UI;

import Entities.ClientEntity;
import Work.ClientTable;
import org.hibernate.Session;
import javax.swing.*;

public class AddClientFrame extends JDialog
{
    JTextField FioField = null;
    JTextField PhoneField = null;
    JTextField AdressField = null;

    public AddClientFrame(Session session, ClientTable model)
```

```
{
    setTitle("Добавление клиента");
    setLayout(null);
    setModal(true);
    setBounds(500, 500, 430, 135);

    JLabel lbl1 = new JLabel("ФИО");
    lbl1.setBounds(10, 10, 100, 20);
    add(lbl1);

    JLabel lbl2 = new JLabel("Телефон");
    lbl2.setBounds(10, 40, 100, 20);
    add(lbl2);

    JLabel lbl3 = new JLabel("Адресс");
    lbl3.setBounds(10, 70, 100, 20);
    add(lbl3);

    FioField = new JTextField();
    FioField.setBounds(120, 10, 200, 20);
    add(FioField);

    PhoneField = new JTextField();
    PhoneField.setBounds(120, 40, 200, 20);
    add(PhoneField);

    AdressField = new JTextField();
    AdressField.setBounds(120, 70, 200, 20);
    add(AdressField);

    JButton btnOk = new JButton("Ok");
    btnOk.setBounds(350, 10, 50, 20);
    add(btnOk);
    btnOk.addActionListener(e -> {
        ClientEntity client = null;
        try {
            client = new ClientEntity();
            client.setFio(FioField.getText());
            client.setPhone(Integer.parseInt(PhoneField.getText()));
            client.setAdress(AdressField.getText());
            session.beginTransaction();
            session.save(client);
            session.getTransaction().commit();
            session.close();
```

```
            model.read();
            model.fireTableDataChanged();

            this.dispose();
          } catch (Exception e1) {
            JOptionPane.showMessageDialog(
            null, e1.getMessage(), "Ошибка при вставке",
JOptionPane.OK_OPTION);
          }
       });
   }
}
```

AddOrderFrame.java

```
package UI;

import Entities.ClientEntity;
import Entities.OrdersEntity;
import Entities.RoomsEntity;
import Work.OrderTable;
import org.hibernate.Session;

import javax.swing.*;
import java.sql.Date;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;

public class AddOrderFrame extends JDialog
{
   JTextField DateArr = null;
   JTextField DateLeav = null;
   JTextField DateOrd = null;
   JTextField OrdCost = null;

   public AddOrderFrame(Session session, OrderTable model)
   {
      ArrayList<ClientEntity> clients = new ArrayList<>(
            session.createSQLQuery("SELECT * FROM client ORDER BY
FIO").addEntity(ClientEntity.class).list());
      ArrayList<RoomsEntity> rooms = new ArrayList<>(
            session.createSQLQuery("SELECT * FROM rooms ORDER BY
id").addEntity(RoomsEntity.class).list());

      setTitle("Добавление заявки");
      setLayout(null);
```

```
        setModal(true);
        setBounds(500, 500, 430, 230);

        JLabel lbl1 = new JLabel("Дата прибытия");
        lbl1.setBounds(10, 10, 100, 20);
        add(lbl1);

        JLabel lbl2 = new JLabel("Дата выезда");
        lbl2.setBounds(10, 40, 100, 20);
        add(lbl2);

        JLabel lbl3 = new JLabel("Дата заявки");
        lbl3.setBounds(10, 70, 100, 20);
        add(lbl3);

        JLabel lbl4 = new JLabel("ФИО клиента");
        lbl4.setBounds(10, 100, 100, 20);
        add(lbl4);

        JLabel lbl5 = new JLabel("Номер комнаты");
        lbl5.setBounds(10, 130, 100, 20);
        add(lbl5);

        JLabel lbl6 = new JLabel("Стоимость");
        lbl6.setBounds(10, 160, 100, 20);
        add(lbl6);

        DateArr = new JTextField( new
Date(Calendar.getInstance().getTime().getTime()).toString());
        DateArr.setBounds(120, 10, 200, 20);
        add(DateArr);

        DateLeav = new JTextField( new
Date(Calendar.getInstance().getTime().getTime()).toString());
        DateLeav.setBounds(120, 40, 200, 20);
        add(DateLeav);

        DateOrd = new JTextField( new
Date(Calendar.getInstance().getTime().getTime()).toString());
        DateOrd.setBounds(120, 70, 200, 20);
        add(DateOrd);

        JComboBox fioBox = new JComboBox();
        fioBox.setBounds(120, 100, 200, 20);
        for(ClientEntity client : clients) {
```

```
            fioBox.addItem(client.getFio());
        }
        add(fioBox);

        JComboBox RoomNum = new JComboBox();
        RoomNum.setBounds(120, 130, 200, 20);
        for(RoomsEntity room : rooms) {
            RoomNum.addItem(Integer.toString(room.getId()));
        }
        add(RoomNum);


        OrdCost = new JTextField();
        OrdCost.setBounds(120, 160, 200, 20);
        add(OrdCost);

        JButton btnOk = new JButton("Ok");
        btnOk.setBounds(350, 10, 50, 20);
        add(btnOk);
        btnOk.addActionListener(e -> {
            OrdersEntity ordrs = null;

            try {
                SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd");
                java.util.Date utilDateArr = format.parse(DateArr.getText());
                java.sql.Date sqlDateArr = new java.sql.Date(utilDateArr.getTime());

                java.util.Date utilDateLeav = format.parse(DateLeav.getText());
                java.sql.Date sqlDateLeav = new java.sql.Date(utilDateLeav.getTime());

                java.util.Date utilDateOrder = format.parse(DateOrd.getText());
                java.sql.Date sqlDateOrder = new java.sql.Date(utilDateOrder.getTime());

                ordrs = new OrdersEntity();
                ordrs.setDateArrive(sqlDateArr);
                ordrs.setDateLeave(sqlDateLeav);
                ordrs.setDateOrder(sqlDateOrder);
                ordrs.setClientFio(fioBox.getSelectedItem().toString());

ordrs.setRoomNumber(Integer.parseInt(RoomNum.getSelectedItem().toString()));
                ordrs.setOrderCost(Integer.parseInt(OrdCost.getText()));

            session.beginTransaction();
            session.save(ordrs);
            session.getTransaction().commit();
```

```
            session.close();

            model.read();
            model.fireTableDataChanged();

            this.dispose();
        } catch (Exception e1) {
            JOptionPane.showMessageDialog(
                    null, e1.getMessage(), "Ошибка при вставке",
JOptionPane.OK_OPTION);
        }
        });
    }
}
```

ClientEntity.java

```
package Entities;

import javax.persistence.*;

/**
/
@Entity
@Table(name = "client", schema = "hotel", catalog = "")
public class ClientEntity {
    private int id;
    private String fio;
    private int phone;
    private String adress;

    @Id
    @Column(name = "id")
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    @Basic
    @Column(name = "FIO")
    public String getFio() {
        return fio;
    }
```

```
public void setFio(String fio) {
   this.fio = fio;
}

@Basic
@Column(name = "Phone")
public int getPhone() {
   return phone;
}

public void setPhone(int phone) {
   this.phone = phone;
}

@Basic
@Column(name = "Adress")
public String getAdress() {
   return adress;
}

public void setAdress(String adress) {
   this.adress = adress;
}

@Override
public boolean equals(Object o) {
   if (this == o) return true;
   if (o == null || getClass() != o.getClass()) return false;

   ClientEntity that = (ClientEntity) o;

   if (id != that.id) return false;
   if (phone != that.phone) return false;
   if (fio != null ? !fio.equals(that.fio) : that.fio != null) return false;
   if (adress != null ? !adress.equals(that.adress) : that.adress != null) return false;

   return true;
}

@Override
public int hashCode() {
   int result = id;
   result = 31 * result + (fio != null ? fio.hashCode() : 0);
   result = 31 * result + phone;
   result = 31 * result + (adress != null ? adress.hashCode() : 0);
```

```
        return result;
    }
}
```

OrdersEntity.java

```
package Entities;

import javax.persistence.*;
import java.sql.Date;

/**
/
@Entity
@Table(name = "orders", schema = "hotel", catalog = "")
public class OrdersEntity {
    private int id;
    private Date dateArrive;
    private Date dateLeave;
    private Date dateOrder;
    private String clientFio;
    private int roomNumber;
    private int orderCost;

    @Id
    @Column(name = "id")
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    @Basic
    @Column(name = "Date_arrive")
    public Date getDateArrive() {
        return dateArrive;
    }

    public void setDateArrive(Date dateArrive) {
        this.dateArrive = dateArrive;
    }

    @Basic
    @Column(name = "Date_leave")
```

```
public Date getDateLeave() {
   return dateLeave;
}

public void setDateLeave(Date dateLeave) {
   this.dateLeave = dateLeave;
}

@Basic
@Column(name = "Date_order")
public Date getDateOrder() {
   return dateOrder;
}

public void setDateOrder(Date dateOrder) {
   this.dateOrder = dateOrder;
}

@Basic
@Column(name = "Client_FIO")
public String getClientFio() {
   return clientFio;
}

public void setClientFio(String clientFio) {
   this.clientFio = clientFio;
}

@Basic
@Column(name = "Room_number")
public int getRoomNumber() {
   return roomNumber;
}
public void setRoomNumber(int roomNumber) {
   this.roomNumber = roomNumber;
}
@Basic
@Column(name = "Order_cost")
public int getOrderCost() {
   return orderCost;
}
public void setOrderCost(int orderCost) {
   this.orderCost = orderCost;
}
```

```
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        OrdersEntity that = (OrdersEntity) o;

        if (id != that.id) return false;
        if (roomNumber != that.roomNumber) return false;
        if (orderCost != that.orderCost) return false;
        if (dateArrive != null ? !dateArrive.equals(that.dateArrive) : that.dateArrive !=
null) return false;
        if (dateLeave != null ? !dateLeave.equals(that.dateLeave) : that.dateLeave !=
null) return false;
        if (dateOrder != null ? !dateOrder.equals(that.dateOrder) : that.dateOrder !=
null) return false;
        if (clientFio != null ? !clientFio.equals(that.clientFio) : that.clientFio != null)
return false;

        return true;
    }
    @Override
    public int hashCode() {
        int result = id;
        result = 31 * result + (dateArrive != null ? dateArrive.hashCode() : 0);
        result = 31 * result + (dateLeave != null ? dateLeave.hashCode() : 0);
        result = 31 * result + (dateOrder != null ? dateOrder.hashCode() : 0);
        result = 31 * result + (clientFio != null ? clientFio.hashCode() : 0);
        result = 31 * result + roomNumber;
        result = 31 * result + orderCost;
        return result;
    }
}
```

RoomsEntity.java

```
package Entities;

import javax.persistence.*;

/**
/
@Entity
@Table(name = "rooms", schema = "hotel", catalog = "")
public class RoomsEntity {
    private int id;
    private String type;
```

```
    @Id
    @Column(name = "id")
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    @Basic
    @Column(name = "Type")
    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        RoomsEntity that = (RoomsEntity) o;

        if (id != that.id) return false;
        if (type != null ? !type.equals(that.type) : that.type != null) return false;

        return true;
    }

    @Override
    public int hashCode() {
        int result = id;
        result = 31 * result + (type != null ? type.hashCode() : 0);
        return result;
    }
}
```

RoomTypesEntity.java

```
package Entities;

import javax.persistence.*;

/**
/
@Entity
@Table(name = "room_types", schema = "hotel", catalog = "")
public class RoomTypesEntity {
    private int id;
    private String type;
    private int cost;
    @Id
    @Column(name = "id")
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }

    @Basic
    @Column(name = "Type")
    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
    }

    @Basic
    @Column(name = "Cost")
    public int getCost() {
        return cost;
    }
    public void setCost(int cost) {
        this.cost = cost;
    }
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
```

```
        RoomTypesEntity that = (RoomTypesEntity) o;

        if (id != that.id) return false;
        if (cost != that.cost) return false;
        if (type != null ? !type.equals(that.type) : that.type != null) return false;
        return true;
    }
    @Override
    public int hashCode() {
        int result = id;
        result = 31 * result + (type != null ? type.hashCode() : 0);
        result = 31 * result + cost;
        return result;
    }
}
```

ClientTable.java

```
package Work;

import Entities.ClientEntity;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;

import javax.swing.table.AbstractTableModel;
import java.util.ArrayList;

/**
/
public class ClientTable extends AbstractTableModel {
    Session session;
    ArrayList<ClientEntity> client;

    public ClientTable(SessionFactory sf) {
        this.session = sf.openSession();
        Transaction tr = session.beginTransaction();
        try {
            read();
        } catch (HibernateException e) {
            if (tr != null) tr.rollback();
            e.printStackTrace();
        }
    }

    @Override
```

```
    public int getRowCount() {
        return client.size();
    }

    @Override
    public String getColumnName(int index)
    {
        String[] names = {"ФИО", "Номер телефона", "Адресс"};
        return names[index];
    }

    @Override
    public int getColumnCount() {
        return 3;
    }

    @Override
    public Object getValueAt(int rowIndex, int columnIndex) {
        ClientEntity ord = client.get(rowIndex);
        switch (columnIndex) {
            case 0: return ord.getFio();
            case 1: return ord.getPhone();
            case 2: return ord.getAdress();
            default: return null;
        }
    }
    public void read() {
        client = (ArrayList<ClientEntity>) session.createSQLQuery("SELECT * FROM
Client").addEntity(ClientEntity.class).list();
        fireTableDataChanged();
    }
}
```

MySessionFactory.java

```
package Work;

import org.hibernate.HibernateException;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;
import org.hibernate.metadata.ClassMetadata;
```

```
import java.util.Map;

/**
/
public class MySessionFactory {
   private static SessionFactory sessionFactory = buildSessionFactory();

   protected static SessionFactory buildSessionFactory() {
      // A SessionFactory is set up once for an application!
      final StandardServiceRegistry registry = new StandardServiceRegistryBuilder()
            .configure() // configures settings from hibernate.cfg.xml
            .build();
      try {
         sessionFactory = new MetadataSources( registry
).buildMetadata().buildSessionFactory();
      }
      catch (Exception e) {
         // The registry would be destroyed by the SessionFactory, but we had trouble
building the SessionFactory
         // so destroy it manually.
         StandardServiceRegistryBuilder.destroy( registry );

         throw new ExceptionInInitializerError("Initial SessionFactory failed" + e);
      }
      return sessionFactory;
   }


   public static SessionFactory getSessionFactory() {
      return sessionFactory;
   }

   public static void shutdown() {
      // Close caches and connection pools
      getSessionFactory().close();
   }
}
```

OrderTable.java

```
package Work;

import Entities.OrdersEntity;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
```

```
import javax.swing.table.AbstractTableModel;
import java.util.*;

/**
/
public class OrderTable extends AbstractTableModel {
    Session session;
    ArrayList<OrdersEntity> ordrs;

    public OrderTable(SessionFactory sf) {
        this.session = sf.openSession();
        Transaction tr = session.beginTransaction();
        try {
            read();
        } catch (HibernateException e) {
            if (tr != null) tr.rollback();
            e.printStackTrace();
        }finally {
            session.close();
        }
    }

    @Override
    public int getRowCount() {
        return ordrs.size();
    }

    @Override
    public String getColumnName(int index)
    {
        String[] names = {"Дата прибытия", "Дата выезда", "Дата заказа", "Клиент",
"Номер комнаты", "Стоимость"};
        return names[index];
    }

    @Override
    public int getColumnCount() {
        return 6;
    }

    @Override
    public Object getValueAt(int rowIndex, int columnIndex) {
        OrdersEntity ord = ordrs.get(rowIndex);
        switch (columnIndex) {
```

```
            case 0: return ord.getDateArrive();
            case 1: return ord.getDateLeave();
            case 2: return ord.getDateOrder();
            case 3: return ord.getClientFio();
            case 4: return ord.getRoomNumber();
            case 5: return ord.getOrderCost();
            default: return null;
        }
    }

    public void read() {
        ordrs = (ArrayList<OrdersEntity>) session.createSQLQuery("SELECT * FROM
Orders").addEntity(OrdersEntity.class).list();
    }
}
```

RoomsTable.java

```
package Work;

import Entities.RoomsEntity;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;

import javax.swing.table.AbstractTableModel;
import java.util.ArrayList;

/**
/
public class RoomsTable extends AbstractTableModel {
    Session session;
    ArrayList<RoomsEntity> rooms;

    public RoomsTable(SessionFactory sf) {
        this.session = sf.openSession();
        Transaction tr = session.beginTransaction();
        try {
            rooms = (ArrayList<RoomsEntity>) session.createSQLQuery("SELECT *
FROM rooms  ORDER BY `rooms`.`id`").addEntity(RoomsEntity.class).list();
        } catch (HibernateException e) {
            if (tr != null) tr.rollback();
            e.printStackTrace();
```

```
        }finally {
            session.close();
        }
    }

    @Override
    public int getRowCount() {
        return rooms.size();
    }

    @Override
    public String getColumnName(int index)
    {
        String[] names = {"Номер комнаты", "Тип"};
        return names[index];
    }

    @Override
    public int getColumnCount() {
        return 2;
    }

    @Override
    public Object getValueAt(int rowIndex, int columnIndex) {
        RoomsEntity ord = rooms.get(rowIndex);
        switch (columnIndex) {
            case 0: return ord.getId();
            case 1: return ord.getType();
            default: return null;
        }
    }
}
```

ДОДАТОК Б

ВІДГУК
керівника економічного розділу
на кваліфікаційну роботу бакалавра
на тему:
«Розробка інформаційної системи організації бронювань та замовлень номерів готелю»
студента групи 122-18ск-1 Пєтухова Євгенія Олеговича

Керівник економічного розділу
Зав. каф. ПЕП та ПУ, д.е.н. **О. Г. Вагонова**

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	**Опис**
Пояснювальні документи	
Диплом Пєтухов.doc	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Диплом Пєтухов.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF.
Програма	
diplom.zip	Архів. Містить коди програми і откомпільовану програму.
Презентація	
Презентація Пєтухов.ppt	Презентація кваліфікаційної роботи.