МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ

Кафедра програмної інженерії

# КУРСОВА РОБОТА

|  |  |  |
| --- | --- | --- |
| з | Бази даних | |
|  | (назва дисципліни) | |
| на тему: | | «Розробка бази даних інформаційної системи «Агентство нерухомості» з використанням СУБД Apache Ignite» |
|  | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студента | | 2 | | курсу, групи | 6.1219-2 |
| спеціальності | | | 121 інженерія програмного забезпечення | | |
|  | | | (шифр і назва спеціальності) | | |
| Наумова Дмитра Павловича | | | | | |
| (ініціали та прізвище) | | | | | |
| Керівник | доцент кафедри програмної інженерії, кандидат фізико-математичних наук Кудін О. В. | | | | |
|  | (посада, вчене звання, науковий ступінь, прізвище та ініціали) | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Національна шкала: | |  | | |
| Кількість балів: |  | | Оцінка ECTS: |  |
|  |  | |  |  |
| Члени комісії: |  |  | |  | |
|  | (підпис) |  | | (ініціали та прізвище) | |
|  |  |  | |  | |
|  |  |  | |  | |
|  | (підпис) |  | | (ініціали та прізвище) | |
|  |  |  | |  | |
|  |  |  | |  | |
|  | (підпис) |  | | (ініціали та прізвище) | |

Запоріжжя – 2021

|  |  |  |
| --- | --- | --- |
| **МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**  **ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ** | | |
| Факультет | математичний | |
| Кафедра | програмної інженерії | |
| Дисципліна | | бази даних |
| Спеціальність | | 121 інженерія програмного забезпечення |
| Освітня програма | | (шифр і назва)  програмна інженерія |

**ЗАВДАННЯ**

**на курсову роботу студентові (студентці)**

|  |
| --- |
| Наумов Дмитро Павлович |

(прізвище, ім’я та по-батькові)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1. Тема роботи | Розробка бази даних інформаційної системи «Агентство | | | | | |
|  | нерухомості» з використанням СУБД Apache Ignite | | | | | |
| 2. Строк здачі студентом закінченої роботи | | | |  | | |
| 3. Вихідні дані до роботи | | | 1. Постановка задачі. | | | |
| 2. Перелік літератури. | | | | | | |
| 3. Практична реалізація коду | | | | | | |
| 4. Зміст роботи | | | | |  | |
| 1. Постановка задачі. | | | | | | |
| 2. Основні теоретичні відомості. | | | | | | |
| 3. Система управління базою даних Apache Ignite | | | | | | |
| 5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень) | | | | | |  |
| 6. Дата видачі завдання | | |  | | | | | |

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Назва етапів курсової роботи** | **Термін виконання етапів роботи** | **Примітка** |
| 1. | Розробка плану роботи |  |  |
|  |  |  |  |
| 2. | Збір вихідних даних |  |  |
|  |  |  |  |
| 3. | Обробка методичних джерел |  |  |
|  |  |  |  |
| 4. | Розробка першого розділу |  |  |
|  |  |  |  |
| 5. | Розробка другого розділу |  |  |
|  |  |  |  |
| 6. | Розробка третього розділу |  |  |
|  |  |  |  |
| 7. | Оформлення курсової роботи |  |  |
|  |  |  |  |
| 8. | Захист курсової роботи |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | |  |  | Д. П. Наумов |
|  |  | (підпис) |  | (ініціали та прізвище) |
|  |  | | | |
| Керівник роботи | |  |  | О. В. Кудін |
|  |  | (підпис) |  | (ініціали та прізвище) |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

**РЕФЕРАТ**

Курсова робота «розробкa бази даних інформаційної системи «Агентство нерухомості»»: \_\_с., \_\_ рис., \_\_ табл., \_\_ джерел, \_\_ додаток.

БАЗА ДАНИХ, NOSQL, APACHE IGNITE.

Об’єкт дослідження – процес розробки бази даних інформаційної системи «Агентство нерухомості».

Предмет дослідження – СУБД Apache Ignite.

Мета роботи – розробити базу даних та клієнтський додаток інформаційної системи «Агентство нерухомості».

Методи дослідження – методи програмної інженерії, системний аналіз.

У роботі розроблено базу даних інформаційної системи «Агентство нерухомості» з використанням нереляційної СУБД Apache Ignite та клієнтський web-додаток з використанням ASP.NET.

Результати роботи можуть бути використані для подальшого застосування у сфері проектування та розробки баз даних.

**ЗМІСТ**

Завдання на курсову роботу 2

Реферат 4

Вступ 7

[1 ПРОЕКТУВАННЯ БАЗИ ДАНИХ 7](#_Toc68266503)

[1.1 Аналіз існуючих СУБД 7](#_Toc68266504)

[1.2 Опис предметної області 7](#_Toc68266505)

[1.3 Побудова ER-діаграм 7](#_Toc68266506)

[Висновки до розділу 1 7](#_Toc68266507)

[2 РЕАЛІЗАЦІЯ БАЗИ ДАНИХ ІНФОРМАЦІЙНОЇ СИСТЕМИ «АГЕНТСТВО НЕРУХОМОСТІ» З ВИКОРИСТАННЯМ СУБД APACHE IGNITE 8](#_Toc68266508)

[2.1 Назва підрозділу 8](#_Toc68266509)

[Висновки до розділу 2 8](#_Toc68266510)

[Висновки 9](#_Toc68266511)

[Перелік посилань 10](#_Toc68266512)

[Додаток А 11](#_Toc68266513)

[Програмна реалізація бази даних 11](#_Toc68266514)

# ВСТУП

Сучасний розвиток інформаційних систем призводить до того, що останнім необхідні засоби для ефективної організації та збереження великих обсягів даних.

Прикладом інформаційних систем може бути агентство нерухомості.

Агентство займається оформленням контрактів з нерухомості. Ведеться облік продавців, агентів та клієнтів. До кожного об’єкту нерухомості наводиться його розгорнутий опис. За згодою покупця і продавця агентство укладає угоду. Заробітна плата агентів обчислюється залежно від кількості укладених угод.

Отже, актуальною задачею є розробка баз даних для автоматизації обліку в інформаційних системах.

Метою курсової роботи є розробка бази даних та клієнтського додатку інформаційної системи «Агентство нерухомості».

Задачі, які необхідно розв’язати для досягнення поставленої мети:

* проаналізувати існуючі системи керування базами даних;
* проаналізувати предметну область;
* сформулювати вимоги до інформаційної системи;
* спроектувати схему даних;
* спроектувати макет клієнтського програмного застосунку;
* реалізувати базу даних засобами обраної СУБД;
* реалізувати клієнтський програмний застосунок;
* протестувати інформаційну систему.

Об’єкт дослідження – процес розробки інформаційної системи «Агентство нерухомості».

Предмет дослідження – СУБД Apache Ignite.

Методи дослідження – методи програмної інженерії, системний аналіз.

Структурно робота складається з трьох розділів. У першому розділі наведено аналіз існуючих СУБД, спроектовано базу даних, виходячи з опису предметної області та поставлених вимог до інформаційної системи. У другому розділі базу даних реалізовано з використанням СУБД Apache Ignite. У третьому розділі розроблено клієнтський web-додаток.

# 1 ПРОЕКТУВАННЯ БАЗИ ДАНИХ

**1.1 Аналіз існуючих СУБД**

За використовуваною моделлю даних СУБД поділяються на такі типи:

– Реляційні

– Документ-орієнтовані

– Ієрархічні

– Мережеві

– «Ключ-значення»

– Комбіновані.

(Далі – спосіб зберігання даних, переваги і недоліки кожної моделі)

Реляційна модель даних є найбільш поширеною; відомими СУБД з цією моделлю є MySQL, SQLite, PostgeSQL, Microsoft SQL Server. Модель була запропонована в 1970р. Едгаром Коддом. Дані зберігаються у вигляді *n*-них відношень, а окремі записи є кортежами довжини *n*, де *n –* кількість атрибутів сутності. Елементи кортежу можуть набувати значень із домену атрибута – множини його допустимих значень.

Для візуального сприйняття використовуються терміни «таблиця», «рядок», «стовпець». Таблиця є множиною рядків, тому повторення останніх не допускається. Усі дані одного стовпця мають бути однакового типу та належати до множини допустимих значень.

Реляційні СУБД дозволяють створювати таблиці, використовуючи попередньо визначену множину типів даних. Крім того, реляційна модель надає можливість використання первинних ключів, які забезпечують унікальність атрибута і рядка в цілому, та зовнішніх ключів, які дозволяють встановлювати логічні зв’язки між сутностями та обмежити домен атрибута.

Для пошуку даних використовуються SQL-запити, до основного функціоналу яких належить вибір даних за певними критеріями, поєднання даних з декількох таблиць, упорядкування та групування рядків, обчислення підсумкових функцій (максимальне, мінімальне, середнє значення, сума тощо). Операції додавання, зміни та видалення даних також реалізуються у вигляді SQL-запитів.

Сучасні версії реляційних СУБД мають широкий набір функцій, тому майже будь-яку операцію можна виконати на боці сервера бази даних, що забезпечує мінімальне мережеве навантаження. Це є критично важливим, якщо клієнтське ПЗ і сервер бази даних знаходяться на різних вузлах мережі.

З урахуванням особливостей зберігання даних в реляційних СУБД, необхідно зазначити, що

Отже, вище було наведено класифікацію систем управління базами даних, розглянуто особливості, переваги та недоліки кожного типу. ..

**1.2 Опис предметної області**

В даній роботі розглядається предметна область «Агентство нерухомості». Агентство є посередником купівлі-продажу об’єктів нерухомості. Ведеться облік нерухомості різних типів, як-от будинки, квартири, земельні ділянки. Клієнти можуть переглядати колекцію об’єктів нерухомості, публікувати свої об’єкти або замовлення з певним переліком вимог. Працівники агентства за згодою продавця і покупця укладають договори купівлі-продажу. Заробітна плата працівників залежить від вартості укладених угод.

Для підвищення рівня доступності та конкурентоспроможності, а також для автоматизації обліку на підприємстві, необхідно розробити автоматизовану інформаційну систему. Система має забезпечувати наступні функції:

– Перегляд наявних об’єктів нерухомості;

– Пошук об’єктів за певними критеріями;

– Публікація, редагування і видалення об’єктів нерухомості;

– Додавання об’єкта нерухомості в закладки;

– Публікація замовлень з описом вимог до об’єкта;

– Перегляд, редагування та скасування замовлень;

– Запит на відвідування об’єкта для детального ознайомлення;

– Укладання договору купівлі-продажу;

– Модерація оголошень та облікових записів користувачів;

– Створення, редагування та видалення облікових записів;

– Підбір об’єктів нерухомості на основі вподобань користувачів.

Для забезпечення наведеного вище функціоналу необхідно реалізувати наступний набір сутностей (табл. 1.1).

Таблиця 1.1 – Набір сутностей предметної області.

|  |  |
| --- | --- |
| Назва сутності | Опис |
| Person | Особа – містить інформацію про ім’я, прізвище, номер телефону, адресу електронної пошти, адресу проживання, дату реєстрації. |
| Credential | Облікові дані – до них належить пароль, стан облікового запису (нормальний, деактивований або заблокований) та рівень доступу до системи (звичайний користувач, агент, модератор, адміністратор). |
| Agent | Агент – містить інформацію про загальну кількість угод, кількість угод за останній місяць, заробітну плату. |
| EstateObject | Об’єкт нерухомості – містить загальний опис, ціну, адресу, дату публікації, ідентифікатор продавця, актуальність, тип об’єкта, стан, теги, посилання на фотографії. |

Продовження таблиці 1.1

|  |  |
| --- | --- |
| Назва сутності | Опис |
| House | Будинок – описова сутність, містить характерні для будинку атрибути: площа земельної ділянки та житлова площа (одиниця виміру – квадратний метр), кількість поверхів, кількість кімнат. |
| Flat | Квартира – описова сутність, містить атрибути, характерні для квартири: житлова площа (одиниця виміру – квадратний метр), кількість кімнат, номер поверху. |
| Landplot | Земельна ділянка – описова сутність, містить інформацію про площу ділянки (одиниця виміру – ар). |
| Location | Локація, або район місцезнаходження – містить інформацію про область, населений пункт та район (якщо є поділ на райони). Вважається, що агентство працює лише на території України, тому назва країни не вказується. |
| ClientWish | Побажання клієнта – містить ідентифікатор клієнта, актуальність, дату публікації, інформацію про тип об’єкта, бажане місцезнаходження, максимальну ціну, необхідний стан об’єкта, теги. |
| Match | Співпадіння – сутність для опису відношення між побажаннями клієнта та об’єктами нерухомості, що полягає у відповідності характеристик об’єкта вимогам, наведеним у побажанні клієнта. Містить ідентифікатор об’єкта та ідентифікатор клієнтського побажання. Передбачається, що користувач зможе переглядати загальний перелік співпадінь на головній сторінці або окремий перелік для кожного опублікованого побажання. |

Кінець таблиці 1.1

|  |  |
| --- | --- |
| Назва сутності | Опис |
| Order | Заявка на огляд об’єкта – містить дату подання, ідентифікатор клієнта, ідентифікатор об’єкта та ідентифікатор агента, який погодиться супроводжувати об’єкт нерухомості. |
| Bookmark | Помітка, або закладка – містить ідентифікатор клієнта та ідентифікатор об’єкта. |
| Deal | Договір – містить дату укладання, ідентифікатори об’єкта нерухомості, продавця, покупця, агента, фінальну ціну об’єкта. |

А

Отже, вище було розглянуто інформаційну систему «Агентство нерухомості», сформовано перелік вимог до програмного забезпечення даної системи.

**1.3 Побудова ER-діаграм**

ER-діаграма є графічним представленням сутностей і відношень предметної області. Діаграма дозволяє відобразити сутності, їх атрибути, відношення та характеристики відношень (тип відношення, кратність). Сутності зображуються у вигляді прямокутників, а відношення – у вигляді ліній з назвою відношення посередині лінії та зазначенням кратності відношення з обох боків від назви. Якщо до діаграми включаються переліки атрибутів сутностей, то лініями відношень з’єднуються атрибути, які є ключовими (як правило, зовнішній ключ однієї сутності-учасника відношення поєднується з первинним ключем іншої сутності).

Можливими варіантами кратності відношень є:

– 1 : 1 (один до одного);

– 1 : N (один до багатьох);

– M : N (багато до багатьох).

Прикладом відношення 1 : 1 може бути відношення «обліковий запис має ідентифікаційний номер телефону». За одним обліковим записом закріплено один номер, і один номер може використовуватися лише в одному обліковому записі.

Прикладом відношення 1 : N може бути відношення «публікація в соціальній мережі має декілька коментарів». Як правило, одна публікація може мати декілька коментарів, або не мати жодного, але коментар належить лише одній публікації.

Відношення «користувач коментує публікацію в соціальній мережі» є типовим відношенням кратності M : N. Як правило, один користувач може написати багато коментарів до різних публікацій, і одна публікація може мати декілька коментарів. Такі відношення зазвичай реалізуються з використанням додаткових сутностей. У наведеному вище прикладі відношення M : N буде поділено на 2 відношення: «Користувач пише коментарі» та «Публікація має коментарі» – обидва з кратністю 1 : N.

Згідно з переліком сутностей (див. табл. 1.1) визначимо відношення предметної області (табл. 1.2).

Таблиця 1.2 – відношення предметної області «Агентство нерухомості»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Учасник відношення 1 | Учасник відношення 2 | Назва | Кратність | Пояснення |
| 1 | 2 | 3 | 4 | 5 |
| Person | Agent | is a | 1:1 | Особа може бути агентом, одному обл. запису відповідає не більше ніж один обліковий запис агента. |
| Person | Credential | has | 1:1 | Особа має облікові дані, одній особі відповідає один набір облікових даних. |

Продовження таблиці 1.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| Person | Location | lives in | N:1 | Декілька осіб можуть проживати в одній локації. |
| Person | ClientWish | has | 1:N | Особа може опублікувати декілька побажань. |
| ClientWish | Location | attached to | N:1 | Декілька побажань клієнта можуть бути закріпленими за однією локацією. |
| Person | EstateObject | sells | 1:N | Одна особа може продавати декілька об’єктів нерухомості, у кожного опублікованого об’єкта нерухомості може бути лише 1 продавець. |
| EstateObject | House | is a | 1:1 | Об’єкт нерухомості може бути будинком. Одному об’єкту нерухомості може відповідати не більше ніж один опис будинку. |
| EstateObject | Flat | is a | 1:1 | Об’єкт нерухомості може бути квартирою. Одному об’єкту нерухомості може відповідати не більше ніж один опис квартири. |
| EstateObject | Landplot | is a | 1:1 | Об’єкт нерухомості може бути земельною ділянкою. Одному об’єкту нерухомості може відповідати не більше ніж один опис земельної ділянки. |
| EstateObject | Location | placed in | N:1 | Декілька об’єктів нерухомості можуть знаходитися в одній локації. |
| EstateObject | Deal | sold by | 1:1 | Об’єкт може бути проданий за договором. Одному об’єкту відповідає не більше ніж один договір купівлі-продажу. |
| Person | Deal | buys | 1:N | Особа може купити декілька об’єктів нерухомості. Одному договору відповідає один покупець. |
| Person | Deal | sells | 1:N | Особа може продати декілька об’єктів нерухомості. Одному договору відповідає один продавець. |
| Agent | Deal | serves | 1:N | Агент може обслуговувати декілька договорів. Одному договору відповідає один агент. |

Кінець таблиці 1.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| Person | Order | has | 1:N | Одна особа може зробити декілька запитів на огляд різних об’єктів нерухомості. Одному запиту відповідає одна особа, що подала запит. |
| Agent | Order | serves | 1:N | Один агент може супроводжувати декілька замовлень. Одному замовленню відповідає один агент. |
| EstateObject | Order | has | 1:N | Один об’єкт нерухомості може бути замовлений для огляду декілька разів. Одному замовленню відповідає один об’єкт нерухомості. |
| Person | Bookmark | has | 1:N | Один користувач може зробити деілька закладок. Одній закладці відповідає один користувач. |
| EstateObject | Bookmark | has | 1:N | Один об’єкт нерухомості може бути доданий у закладки декілька разів. Одна закладка може посилатися на один об’єкт. |
| ClientWish | Match | has | 1:N | Одне побажання клієнта може мати декілька співпадінь з опублікованими об’єктами нерухомості. Одне співпадіння відповідає одному побажанню клієнта. |
| EstateObject | Match | has | 1:N | Один об’єкт нерухомості може відповідати різним побажанням клієнтів. Одне співпадіння відповідає одному об’єкту нерухомості. |

Далі, з урахуванням опису сутностей (див. табл. 1.1) визначимо їх атрибути та побудуємо ER-діаграму (рис. 1.1). Необхідно враховувати, що далі буде використовуватися модель даних key-value, тому кожна сутність повинна мати один ключовий артибут.

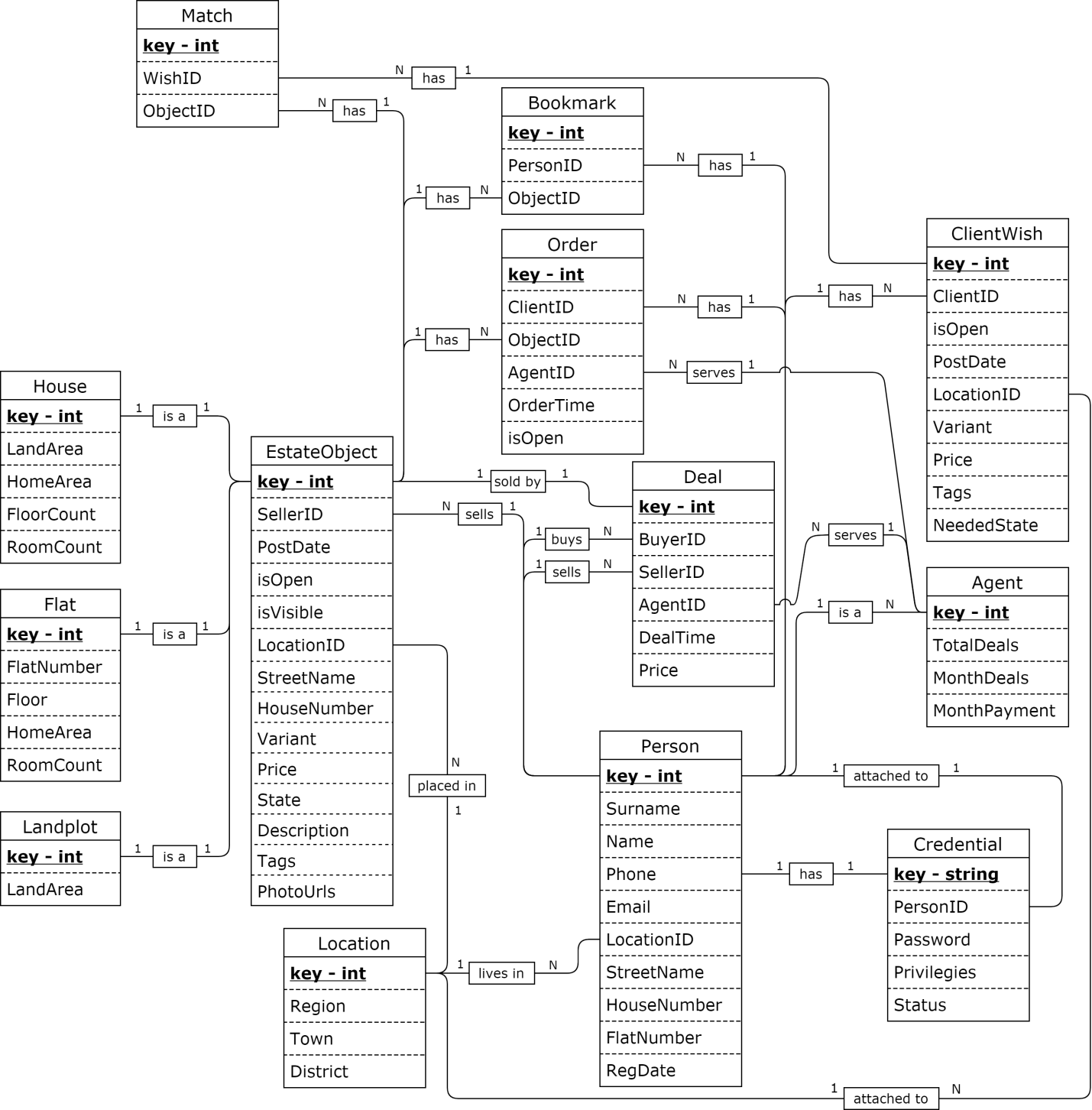


Рисунок 1.1 – ER-діаграма «Агентство нерухомості»

**Висновки до розділу 1**

В даному розділі було розглянуто класифікацію систем управління базами даних, проведено аналіз предметної області, складено перелік вимог до інформаційної системи. Побудовано ER-діаграму для відображення сутностей і відношень предметної області. ER-діаграма буде використовуватися при подальшій розробці інформаційної системи.

# 2 РЕАЛІЗАЦІЯ БАЗИ ДАНИХ ІНФОРМАЦІЙНОЇ СИСТЕМИ «АГЕНТСТВО НЕРУХОМОСТІ» З ВИКОРИСТАННЯМ СУБД APACHE IGNITE

**2.1 Теоретичні відомості**

Apache Ignite (далі – Ignite) є розподіленою СУБД, що використовує модель даних key-value. Написана з використанням мови програмування Java. СУБД допускає як одиничну, так і багатовузлову конфігурацію мережі, а дані можуть зберігатися розподілено або в режимі реплікації. При розподіленому режимі кожен вузол мережі зберігає окрему порцію даних; в режимі реплікації кожен вузол містить копію всіх наявних даних. Багатовузлова конфігурація використовується при великих навантаженнях на інформаційну систему, тоді як на початкових етапах розробки і тестування доцільно використовувати одиничну конфігурацію.

За замовченням Ignite зберігає дані виключно в оперативній пам’яті. Це означає, що інформація буде втрачена при збої процесу або вимкненні комп’ютера. Для збереження даних на постійному носії необхідно застосувати файл конфігурації з відповідними параметрами (див. розділ 2.2) при запуску вузла.

Дана СУБД використовує модель key-value, але для маніпуляцій з даними дозволяється використання мови SQL. Оскільки мова SQL є досить широко використовуваною, перехід від іншої бази даних до Ignite значно полегшується.

Ignite надає можливість застосування транзакцій для забезпечення стійкості бази даних. Транзакції є доцільними при виконанні декількох послідовних логічно пов’язаних запитів до бази даних (наприклад, перевірити наявність запису з ключем N, якщо немає, то додати новий запис).

Ignite надає програмні інтерфейси для різних середовищ і мов програмування, наприклад, Java, C#/.NET, C++, Python, Node.js. Крім того, існує можливість вбудованого використання Ignite у програмі, але такий підхід уповільнює запуск програми, оскільки при кожному її виконанні запускається вузол Ignite.

Програмні інтерфейси Ignite є зручними у використанні тому, що застосовуються технології відповідної мови програмування і допускається застосування користувацьких типів даних. Тобто СУБД самостійно вирішує задачу розпізнавання і приведення типів. Щодо технологій, якщо взяти до уваги платформу .NET, Ignite дозволяє використовувати LINQ – типовий для платформи інтерфейс взаємодії з даними.

**2.2 Налаштування серверу БД**

В даній роботі буде використовуватися клієнт-серверний підхід до розгортання СУБД. Передбачається виконання роботи на ОС Windows. Для початку, встановимо необхідне ПЗ:

– Java runtime environment 8;

– Apache Ignite.

Оскільки за замовченням Ignite не зберігає дані на диску, створимо файл конфігурації (config.xml), в якому зазначимо необхідні опції (див. додаток А). Файл бажано помістити в директорію з виконавчими скриптами Ignite (bin). Для запуску серверного вузла застосовується команда:

ignite.bat config.xml

За замовченням вузол запускається в незахищеному режимі – при підключенні ім’я користувача і пароль не потрібні. Крім того, мережа вузлів при першому запуску знаходиться в неактивному стані. Для активації необхідно застосувати утиліту control.bat або відповідний метод в коді програми.

Після запуску вузла СУБД є повністю готовою до роботи. Для взаємодії з СУБД можна використовувати самостійні утиліти (наприклад, sqlline) або програмні інтерфейси.

Отже, вище було розглянуто процес налаштування та запуску серверного вузла СУБД Apache Ignite. Дана СУБД вимагає незначної кількості дій, що значно полегшує її використання.

**2.3 Програмна реалізація сутностей**

Як було зазначено в розділі 2.1, Ignite надає можливість зберігання користувацьких типів даних. Це спрощує розробку програмного забезпечення тому, що код можна використовувати повторно.

Для кращого розуміння процесу реалізації сутностей необхідно розглянути практичні приклади. Тут і далі будемо використовувати мову програмування C#.

По-перше, створимо новий проект з використанням мови C#, платформи .NET core 3.1. Підключимо бібліотеку Apache.Ignite.Core.dll – файл бібліотеки повинен знаходитися в директорії IGNITE\_HOME/platforms/dotnet/bin (тут і далі IGNITE\_HOME – коренева директорія Ignite).

Для початку взаємодії з СУБД необхідно налаштувати клієнтський вузол.

IIgniteClient client;

client = Ignition.StartClient (new IgniteClientConfiguration

{Endpoints = new[] {"127.0.0.1:10800"}}

);

IIgniteClient є клієнтським інтерфейсом Ignite, який є посередником у взаємодії програми користувача з сервером СУБД.

Метод Ignition.StartClient() створює новий екземпляр клієнта та здійснює підключення до серверного вузла.

До методу Ignition.StartClient() в якості параметра передається конфігурація клієнтського вузла. В даному випадку зазначається лише IP-адреса серверного вузла та номер використовуваного порту.

Після виконання вказаних вище операцій, якщо підключення вдалося, то до змінної client буде записано екземпляр клієнтського інтерфейсу. В іншому випадку отримаємо System.AggregateException з детальним описом помилки.

Як було зазначено в розділі 2.2, при першому запуску мережа вузлів Ignite знаходиться в неактивному стані. Для активації застосовується наступний метод:

client.GetCluster().SetActive(true);

Щоб закрити створене підключення до серверу СУБД, необхідно викликати метод Dispose() клієнтського інтерфейсу.

client.Dispose();

Тепер, отримавши доступ до СУБД, можна почати реалізацію сутностей. Користуючись ER-діаграмою (рис. 1.1), для прикладу створимо клас сутності Location:

public class Location {

public string Region;

public string Town;

public string District;

}

Ignite зберігає дані у вигляді структури даних «кеш» (ICache, ICacheClient), яка є множиною пар ключ-значення. В мові C# існує схожа структура даних під назвою Dictionary.

ICache та ICacheClient є аналогами таблиць, використовуваних в реляційній моделі даних. Різниця між ICache та ICacheClient полягає в тому, що інтерфейс ICache застосовується в серверних вузлах, запущених у користувацькій програмі, а ICacheClient – у вузлах-клієнтах.

Щоб створити кеш без SQL-надбудови, застосовується метод клієнтського інтерфейсу CreateCache() з параметром типу string:

ICacheClient LocationCache = client.CreateCache<int, Location>("location");

В даному випадку створюється кеш з ключами типу int та значеннями типу Location. В методі CreateCache() зазначаються типи ключів та значень, а в якості параметра передається назва кешу.

Наведений вище підхід є найпростішим варіантом створення кешу. Допускається при реалізації сутностей, які не мають відношень з іншими сутностями, і якщо не передбачається застосування SQL-запитів.

Щоб мати змогу виконувати SQL-запити, необхідно зазначити атрибут QuerySqlField у полях класу сутності або задати конфігурацію кешу, в якій зазначені поля SQL.

Розглянемо перший варіант:

public class Location {

[QuerySqlField] public string Region;

[QuerySqlField] public string Town;

[QuerySqlField] public string District;

}

CacheClientConfiguration locationCfg = new CacheClientConfiguration {

GroupName = "estateagency",

Name = "location",

AtomicityMode = CacheAtomicityMode.Transactional,

QueryEntities = new[] {

new QueryEntity {

TableName = "Locations",

KeyType = typeof(int),

ValueType = typeof(Location)

}

}

};

ICacheClient LocationCache = client.CreateCache<int, Location>(locationCfg);

В даному випадку в кожному полі класу було зазначено атрибут QuerySqlField, що робить поле доступним для SQL-запитів. Далі було створено конфігурацію кешу, де зазначено назву групи, власну назву кешу, транзакційний режим роботи, набір сутностей SQL. В даному випадку кеш містить лише одну сутність SQL з назвою таблиці «Locations», ключем типу int та значенням типу Location. Поля SQL автоматично отримуються з класу Location.

Робити всі поля класу видимими для SQL необов’язково. Достатньо зазначити атрибут QuerySqlField в одному полі класу. Якщо таких полів не буде і в конфігурації кешу буде зазначено поле QueryEntities, то при створенні кешу станеться помилка.

Другий варіант передбачає визначення полів SQL безпосередньо в конфігурації кешу. Може бути корисним, якщо з певних причин клас сутності неможливо змінити.

CacheClientConfiguration locationCfg = new CacheClientConfiguration {

GroupName = "estateagency",

Name = "location",

AtomicityMode = CacheAtomicityMode.Transactional,

QueryEntities = new[] {

new QueryEntity {

TableName = "Locations",

KeyType = typeof(int),

ValueType = typeof(Location),

Fields = new[] {

new QueryField ("Region", typeof(string)),

new QueryField ("Town", typeof(string)),

new QueryField ("District", typeof(string))

}

}

}

};

ICacheClient LocationCache = client.CreateCache<int, Location>(locationCfg);

В даному варіанті в об’єкті QueryEntity, що відповідає сутності Location, зазначається поле Fields, що містить поля, які мають бути доступними для SQL. При конфігурації кешу автоматично встановлюється відповідність між SQL-полями та полями класу з однаковими назвами.

Окрім визначення полів, досить часто доводиться визначати індекси для пришвидшення доступу до даних. Так само як при визначенні SQL-полів, індекси можна визначити двома способами: в класі сутності або в конфігурації кешу.

При першому способі необхідно зазначити атрибут QuerySqlField з опцією IsIndexed на потрібному полі класу. Розглянемо приклад сутності Location:

public class Location {

**[QuerySqlField (IsIndexed = true)]** public string Region;

[QuerySqlField] public string Town;

[QuerySqlField] public string District;

}

При другому способі, в об’єкті QueryEntity, що відповідає сутності Location, необхідно визначити колекцію Indexes. Колекція містить індекси з зазначенням використовуваних полів класу.

Indexes = new QueryIndex[]

{

new QueryIndex("Region")

}

Після створення кешу, його можна отримати:

а) за допомогою методу IIgniteClient.GetCache(string).

ICacheClient LocationCache = client.GetCache<int, Location>("location");

б) за допомогою більш універсальних методів:

– IIgniteClient.GetOrCreateCache(CacheClientConfiguration);

– IIgniteClient.GetOrCreateCache(string).

ICacheClient LocationCache = client.GetOrCreateCache<int, Location>("location");

Для видалення існуючого кешу застосовується метод IIgniteClient.DestroyCache(string).

client.DestroyCache ("location");

Для отримання переліку назв існуючих кешів застосовується метод IIgniteClient.GetCacheNames().

ICollection<string> names = client.GetCacheNames();

Особливість Apache Ignite полягає в тому, що СУБД може зберігати різні типи даних в одному кеші. Допускається поєднання будь-яких користувацьких та системних типів даних, але найбільш доцільним є зберігання типів даних, що пов’язані наслідуванням. Якщо сутності використовують ключ одного типу, то даний ключ є спільним для всіх сутностей. Мається на увазі, що при існуванні в одному кеші сутностей типу А і типу В, що мають ключ типу *a*, то значення *x* ключа може зустрітися лише один раз. Якщо існує запис типу A з ключем *x*,то не може існувати запис типу В з таким же значенням ключа.

Розглянемо приклад сутностей EstateObject, House з предметної області «Агентство нерухомості» (див. табл. 1.1, рис. 1.1). Сутність House походить від сутності EstateObject. Спочатку визначимо класи сутностей:

public class EstateObject {

[QuerySqlField] public int SellerID;

[QuerySqlField] public DateTime PostDate;

[QuerySqlField] public bool isOpen;

[QuerySqlField] public bool isVisible;

[QuerySqlField] public int LocationID;

[QuerySqlField] public string StreetName;

[QuerySqlField] public string HouseNumber;

[QuerySqlField] public byte Variant;

[QuerySqlField] public int Price;

[QuerySqlField] public byte State;

[QuerySqlField] public ICollection<string> Tags;

[QuerySqlField] public ICollection<string> PhotoUrls;

}

public class House: EstateObject {

[QuerySqlField] public float LandArea;

[QuerySqlField] public float HomeArea;

[QuerySqlField] public short FloorCount;

[QuerySqlField] public short RoomCount;

}

Далі необхідно задати конфігурацію кешу та створити його.

CacheClientConfiguration objectCfg = new CacheClientConfiguration {

GroupName = "estateagency",

Name = "estateobject",

AtomicityMode = CacheAtomicityMode.Transactional,

QueryEntities = new[] {

new QueryEntity {

TableName = "EstateObjects",

KeyType = typeof(int),

ValueType = typeof(EstateObject)

},

new QueryEntity {

TableName = "Houses",

KeyType = typeof(int),

ValueType = typeof(House)

}

}

};

ICacheClient ObjectCache = client.CreateCache<int, EstateObject>(objectCfg);

ICacheClient HouseCache = client.GetCache<int, House>("estateobject");

Відповідно до наведених вище прикладів, необхідно реалізувати інші сутності предметної області. Програмний код наведено в додатку Б.

Отже, вище було розглянуто підключення до серверу БД та програмну реалізацію сутностей в моделі даних key-value із SQL-надбудовою. Реалізація сутностей включає в себе такі етапи:

– Визначення класу сутності;

– Конфігурація кешу;

– Створення кешу.

**2.4 Забезпечення цілісності відношень**

На відміну від реляційної моделі даних, модель key-value не забезпечує цілісність відношень. Важливість цієї функції полягає в захисті від появи записів, які не мають змісту при відсутності запису, на який встановлено посилання.

Наприклад, сутність об’єкта нерухомості має зберігати інформацію про продавця. Замість того, щоб зберігати всю інформацію, об’єкт нерухомості містить лише ID продавця – посилання на запис особи з відповідним ключем. Якщо такий запис не існує, то інформація про об’єкт нерухомості є неповною і, відповідно, некоректною, бо не містить інформації про особу-продавця.

Перевірка цілісності відношень має здійснюватися при додаванні, редагуванні та видаленні записів. В реляційній моделі даних перевірка здійснюється автоматично на рівні серверу БД, якщо в таблицях визначені зовнішні ключі. В СУБД моделі key-value, в тому числі в Apache Ignite, немає функціональності зовнішніх ключів, тому перевірку відношень необхідно виконувати на рівні клієнтської програми.

Розглянемо приклад сутності EstateObject з предметної області (див. табл. 1.1, табл. 1.2, рис. 1.1). Дана сутність містить посилання на запис особи-продавця (SellerID) та локації (LocationID). При операції додавання необхідно перевірити наявність записів в кешах Person і Location. З урахуванням того, що база даних може працювати в багатокористувацькому режимі, застосовуються транзакції. Будемо вважати, що існує кеш LocationCache з записами типу Location, PersonCache з записами типу Person, ObjectCache з записами типу EstateObject.

public static bool PutObject (int key, EstateObject obj){

using (var tx = client.GetTransactions().TxStart()){

if (!PersonCache.ContainsKey(obj.SellerID)){

tx.Commit();

return false;

}

if (!LocationCache.ContainsKey(obj.LocationID)){

tx.Commit();

return false;

}

ObjectCache.Put(key, obj);

return true;

}

}

Наведений вище метод передбачає прямий запис об’єкта нерухомості за ключем key, незалежно від наявності заданого ключа. Щоб уникнути перезапису вже існуючих об’єктів, можна додати перевірку наявності ключа або створити кеш, у якому зберігається лічильник, і автоматично визначати ключі для нових записів.

При зміні даних запису EstateObject (ключ не змінюється), необхідно перевірити наявність тих же записів, що при операції додавання. Зміна ключа може відбутися, але при реальній експлуатації інформаційної системи така необхідність виникає дуже рідко. Для сутності EstateObject це не потрібно взагалі, але, наприклад, ключ сутності Credential може змінюватися. Тоді буде необхідно змінити поле Phone сутності Person, яке посилається на ключ сутності Credential.

При видаленні запису EstateObject порушується зміст тих сутностей, які містять поля з посиланням на ключ даного запису:

– Deal (ключ);

– Order (поле ObjectID);

– Bookmark (поле ObjectID);

– Match (поле ObjectID).

При видаленні запису можливі такі рішення:

– Видалити записи з посиланням;

– Замінити значення посилань на NULL;

– Заборонити видалення;

– Нічого не робити.

В даному випадку Deal є важливим записом, тому видалення об’єкта нерухомості забороняється, якщо існує Deal з посиланням на даний об’єкт. В іншому випадку видалення можна дозволити, при цьому будуть видалятися записи Order, Bookmark, Match, що містили посилання на вилучений об’єкт нерухомості.

Приклад методу для видалення запису EstateObject:

public static bool DeleteObject (int key){

using (var tx = client.GetTransactions().TxStart()){

if (DealCache.ContainsKey(key)){

tx.Commit();

return false;

}

OrderCache.Query (new SqlFieldsQuery

($"delete from Orders where ObjectID={key};"));

BookmarkCache.Query (new SqlFieldsQuery

($"delete from Bookmarks where ObjectID={key};"));

MatchCache.Query (new SqlFieldsQuery

($"delete from Matches where ObjectID={key};"));

tx.Commit();

return true;

}

}

В даному випадку застосовується як функціональна взаємодія з інтерфейсом кешу, так і SQL-запити. SQL дозволяє перекласти навантаження на серверний вузол БД, зменшити кількість мережевих взаємодій і пришвидшити виконання операції. Однак, таке забезпечення цілісності відношень у будь-якому випадку буде працювати повільніше, ніж в реляційній моделі даних з підтримкою зовнішніх ключів. Цілісність відношень не є абсолютно необхідною, але варто її забезпечити для уникнення беззмістовних записів у базі даних. Можна досягти цілісності відношень іншими способами – наприклад, розробити клієнтську програму з обмеженою функціональністю та можливістю вибирати значення полів даних з підібраних заздалегідь.

Отже, вище було розглянуті типові рішення для забезпечення цілісності відношень в моделі даних key-value. Цілісність відношень в реляційній моделі забезпечується на рівні СУБД, тоді як в моделі key-value за це відповідає клієнтське ПЗ. Перевірку цілісності доцільно вбудовувати в методи для додавання, зміни та видалення записів.

**2.5 Розробка запитів до бази даних**

**Висновки до розділу 2**

Отже, текст висновків до другого розділу.

**Висновки**

Отже, далі наведено загальний висновок.

**Перелік посилань**

1. Горнаков С. Г., Осваиваем популярные системы управления сайтом (CMS). Москва : Пресс, 2009. 336 с.
2. Коноплянко З. Д. Інформатика та комп’ютерна техніка. Українсько-Англійсько-Російський термінологічний тлумачний словник. URL : [https://www.studmed.ru/view/konoplyanko-zd-nformatika-ta-kompyuterna-tehnka-ukrayinsko-anglysko-rosyskiy-termnologchniy-tlumachiny-slovnik\_25eba0 b4674.html](https://www.studmed.ru/view/konoplyanko-zd-nformatika-ta-kompyuterna-tehnka-ukrayinsko-anglysko-rosyskiy-termnologchniy-tlumachiny-slovnik_25eba0%20b4674.html) (дата звернення : 19.04.2019)
3. Створення Web-сторінок. Інструментарій для створення Web-сторінок*.* URL : [http://gymlit.in.ua/re\_%D0%A1%D1%82%D0%B2%D0%BE% D1%80%D0%B5%D0%BD%D0%BD%D1%8F\_Web-%D1%81%D1%82%D0% BE%D1%80% D1%96% D0%BD%D0 %BE%D0%BA](http://gymlit.in.ua/re_Ство%25%20D1%80ення_Web-ст%D0%25%20BEр%25%20D1%96%25%20D0%BD%D0%20%BEк) (дата звернення 16.03.2019)

# Додаток А

# Програмна реалізація бази даних