## *Лабораторна робота 2.* **Проєктування концептуальної моделі бази даних**

Мета:

Ознайомити студентів з процесом розробки концептуальної моделі бази даних для заданої предметної області, використовуючи нотацію UML (Unified Modeling Language).

Завдання:

1. Вивчення предметної області:

Переглянути виконання пункту 1 лабораторної роботи 1, тобто опис предметної області, на основі якої буде створюватися база даних.

2. Визначення основних сутностей:

Виявити ключові сутності предметної області (наприклад, студент, курс, викладач). Мінімум 5 основних сутностей, не враховуючи довідникові.

3. Визначення атрибутів сутностей:

Описати атрибути для кожної сутності (наприклад, ім'я, дата народження для студента).

4. Встановлення зв'язків між сутностями:

Описати зв'язки між сутностями (наприклад, студент записується на курс).

5. Побудова діаграми класів UML:

Використовуючи нотацію UML, створити діаграму класів, яка відображає сутності, їхні атрибути та зв'язки між ними.

Результат. Студенти повинні подати:

1. Опис основних сутностей предметної області
2. Опис атрибутів сутностей
3. Опис зв’язків між сутностями предметної області
4. Діаграму класів UML, що відображає концептуальну модель бази даних для заданої предметної області.

### *Теоретичні відомості*

Концептуальна модель (схема) – це представлення усього вмісту бази даних, яке описане за допомогою концептуальної мови визначення даних, плюс засоби безпеки і правила забезпечення цілісності. Саме концептуальної мови визначення даних, тому що її визначення має відноситися тільки до змісту інформації.

Концептуальна модель бази даних представляє високорівневий, абстрактний опис структури даних, що відображає основні сутності, атрибути та зв'язки між ними, без деталізації технічних аспектів, таких як типи даних або індекси. Вона є незалежною від конкретної СУБД або фізичного зберігання даних.

Основні компоненти концептуальної моделі:

– Сутності – абстракції реальних об'єктів або концепцій, які мають значення для бізнесу (наприклад, клієнт, продукт, замовлення). Тобто сутність – це множина об’єктів, що володіють однаковим набором властивостей, а формальний опис екземпляра сутності являє собою множину елементів даних, що відповідають конкретним значенням його властивостей (атрибутів).

– Атрибути – властивості сутностей, які містять конкретні характеристики об’єктів (наприклад, ім'я клієнта, ціна продукту).

– Зв'язки визначають, як сутності пов'язані між собою (наприклад, замовлення робиться клієнтом на продукт).

*Інструменти та методи створення*

Існує кілька методів створення концептуальної моделі, кожен із яких допомагає організувати дані та зв’язки між ними. Розглянемо основні методи створення концептуальної моделі бази даних:

1. ER-моделювання (Entity-Relationship Model, ERM)

Метод сутність-зв'язок є найпоширенішим підходом для розробки концептуальної моделі бази даних. Він полягає у графічному представленні сутностей (об’єктів), атрибутів (властивостей сутностей) та зв'язків між ними.

Основні поняття:

Сутність (Entity): Об’єкт, про який потрібно зберігати дані (наприклад, Клієнт, Товар).

Атрибут (Attribute): Характеристика сутності (наприклад, ім’я, ціна).

Зв’язок (Relationship): Описує, як сутності взаємодіють між собою (наприклад, клієнт купує товар).

Основні кроки ER-моделювання:

– Визначити ключові сутності.

– Визначити атрибути для кожної сутності.

– Визначити зв’язки між сутностями та їх кардинальність (один до одного, один до багатьох, багато до багатьох).

– Накреслити ER-діаграму, яка графічно відображає ці компоненти.

Приклад:

Розглянемо приклад концептуальної моделі для системи управління замовленнями:

– Сутності: Клієнт, Замовлення, Продукт.

– Атрибути: Клієнт (ID, Ім'я, Адреса), Замовлення (ID, Дата), Продукт (ID, Назва, Ціна).

– Зв'язки: Клієнт робить одне або кілька замовлень; кожне замовлення містить один або кілька продуктів.

2. Об’єктно-орієнтоване моделювання (Object-Oriented Model, OOM)

Цей метод використовує концепції об’єктно-орієнтованого програмування (ООП) для створення концептуальної моделі бази даних. Об’єктно-орієнтоване моделювання застосовується в тих випадках, коли база даних повинна підтримувати складні структури даних, які можуть бути представлені у вигляді об'єктів.

Основні поняття:

Об’єкт: Сутність, яка має атрибути і методи (поведінка).

Класи: Групування об'єктів, які мають спільні атрибути та методи.

Успадкування: Можливість одного класу наслідувати атрибути й методи іншого класу.

Основні кроки об’єктно-орієнтованого моделювання:

– Визначення об'єктів (схожих на сутності в ER-моделюванні).

– Опис атрибутів і методів для кожного об'єкта.

– Визначення зв’язків між об'єктами (асоціації, агрегації, наслідування).

– Створення UML-діаграми (Unified Modeling Language).

3. Моделювання на основі потоків даних (Data Flow Diagrams, DFD)

Метод потоків даних зосереджується на тому, як дані переміщуються через систему. Концептуальна модель бази даних у такому підході показує, які дані надходять у систему, як вони обробляються та зберігаються.

Основні поняття:

Процеси: Описують, як система обробляє дані.

Джерела/отримувачі даних: Вказують, звідки надходять дані та куди вони йдуть.

Потоки даних: Описують, як дані переміщуються між процесами та джерелами.

Основні кроки DFD:

– Визначення основних процесів у системі.

– Визначення джерел і отримувачів даних.

– Визначення потоків даних між джерелами, процесами та базою даних.

4. Моделювання на основі семантичних об’єктів (Semantic Object Model, SOM)

Цей підхід акцентує увагу на значенні (семантиці) об'єктів і відносин між ними. Він фокусується на реальних об'єктах і процесах, що відбуваються в предметній області, і дозволяє визначити важливі семантичні аспекти системи.

Основні кроки семантичного моделювання:

– Визначити об’єкти, які мають ключове значення в предметній області.

– Визначити взаємозв’язки між цими об'єктами.

– Описати всі значущі атрибути та властивості.

5. Інформаційно-логічне моделювання (Information Engineering, IE)

Цей підхід використовує систематичний процес моделювання даних, починаючи з високорівневого подання і поступово деталізуючи його. Інформаційно-логічне моделювання включає три основні етапи:

– Концептуальна модель: Визначення сутностей та їхніх зв'язків.

– Логічна модель: Відображення концептуальної моделі на модель бази даних із зазначенням атрибутів, первинних та зовнішніх ключів.

– Фізична модель: Перетворення логічної моделі на фізичну, з урахуванням вибраної СУБД та її особливостей.

Підсумок:

– ER-моделювання є найпопулярнішим підходом для створення концептуальних моделей баз даних.

– Об'єктно-орієнтоване моделювання підходить для проєктів з використанням ООП.

– Моделювання потоків даних використовується для аналізу руху даних у системі.

– Семантичне моделювання фокусується на значеннях об'єктів і їхніх відносинах.

– Інформаційно-логічне моделювання поєднує кілька рівнів деталізації для побудови бази даних.

Вибір методу залежить від складності предметної області, вимог до системи та специфіки проєкту.

Метод проєктування концептуальної моделі бази даних *з використанням* уніфікованої мови моделювання (Unified Modeling Language, UML) *діаграми класів* дозволяє представити структуру бази даних у вигляді об’єктно-орієнтованого підходу.

UML є графічною мовою для візуалізації, специфікування, конструювання та документування систем, в яких велика роль належить програмному забезпеченню. Діаграма класів UML допомагає візуалізувати основні сутності (класи), їх атрибути, методи та взаємозв'язки, що дозволяє краще зрозуміти архітектуру майбутньої системи.

*Кроки проєктування концептуальної моделі бази даних за допомогою UML діаграми класів:*

1. Аналіз предметної області

Перше, що потрібно зробити, це провести аналіз предметної області. На цьому етапі визначається, які об’єкти (сутності) існують у системі, які дані необхідно зберігати,

і як ці об'єкти взаємодіють між собою. Наприклад, якщо це система для управління бібліотекою, можна визначити такі об'єкти, як "Книга", "Автор", "Читач".

2. Визначення класів

Кожен об’єкт у предметній області перетворюється в клас на UML діаграмі. Клас представляє сутність, яка має певні атрибути і поведінку. Класам присвоюються назви, які відповідають сутностям предметної області.

Наприклад, для бібліотеки це можуть бути класи: "Книга", "Автор", "Читач".

3. Визначення атрибутів

Після визначення класів потрібно додати до них атрибути. Атрибути представляють характеристики класу. Кожен атрибут має певний тип (наприклад, "int" для чисел, "String" для тексту).

– Для класу "Книга" атрибути можуть бути: Назва, ISBN, Рік видання.

– Для класу "Автор": Ім'я, Прізвище.

4. Визначення асоціацій (зв’язків) між класами

У UML діаграмі класів зв’язки між класами відображаються за допомогою асоціацій. Вони показують, як сутності взаємодіють одна з одною. Наприклад, в бібліотеці клас "Автор" буде мати зв'язок з класом "Книга", оскільки автор може написати багато книг, а книга може мати кількох авторів.

Основні типи зв’язків:

– Асоціація (Association) – найзагальніший зв’язок, який вказує на існування взаємодії між класами.

– Агрегація (Aggregation) – частковий зв’язок між класами, коли один клас є частиною іншого (наприклад, "Читач" орендує "Книгу").

– Композиція (Composition) – сильніший зв’язок, коли один клас не може існувати без іншого (наприклад, "Глава книги" не може існувати без "Книги").

– Успадкування (Inheritance) – один клас може успадковувати атрибути і поведінку іншого класу (наприклад, "Співробітник" може бути підкласом "Людина").

5. Визначення кардинальності

Кардинальність вказує на кількість екземплярів сутностей, які можуть бути пов’язані між собою. Це важливий аспект моделювання баз даних, який визначає, чи можуть сутності мати відношення "один до одного", "один до багатьох" або "багато до багатьох".

Наприклад, зв’язок між класом "Автор" і "Книга" може мати кардинальність багато до багатьох (один автор може написати кілька книг, одна книга може бути написана колективом авторів).

6. Визначення методів (функцій)

Класи також можуть мати методи, які відображають їхню поведінку. Хоча в концептуальній моделі бази даних зазвичай більше уваги приділяється атрибутам і зв’язкам, у UML діаграмі класів можна додати методи, які вказують на операції, що можуть виконуватись з даними.

Наприклад, для класу "Читач" може бути метод "Орендувати книгу()".

7. Побудова UML діаграми класів

На основі виконаного аналізу створюється діаграма класів у UML, яка відображає всі класи, їх атрибути, зв’язки та кардинальність. Це допомагає візуалізувати структуру бази даних до її фізичної реалізації.

8. Перевірка та уточнення

На цьому етапі важливо перевірити модель, щоб переконатися, що всі сутності, атрибути і зв’язки правильно відображають предметну область. Якщо знайдено неточності або додаткові вимоги, вносяться корективи в модель.

При побудові UML діаграми класів важливо дотримуватися певних вимог та правил для коректного відображення структури системи, зв'язків між класами та їхньої взаємодії. Як мова графічного візуального моделювання UML має свою нотацію – прийняті позначення. Дотримання нотації допомагає забезпечити зрозумілу та точну інтерпретацію моделі, а також сприяють її подальшій реалізації в програмному коді або базі даних.

*Основні вимоги до побудови UML діаграми класів.*

1. Чітке визначення класів:

– Кожен клас представляє конкретну сутність або об'єкт системи.

– Кожен клас зображується у вигляді прямокутника, який ділиться на три горизонтальні секції.

– Ім'я класу записується вгорі прямокутника (перша секція). Кожне слово в назві класу прийнято писати з великої літери без пробілів. Це має бути іменник або словосполучення в однині, який найбільш точно характеризує предмет.

– Під ім'ям класу (друга секція) розміщуються атрибути (характеристики об'єкта).

– У нижній частині прямокутника (третя секція) можуть бути вказані методи (функції), які описують поведінку класу.

Атрибути та методи можуть бути відсутніми, тобто секції 2 та 3 можуть бути порожніми.

2. Правильне позначення атрибутів і методів:

– Атрибути повинні бути чітко визначені. Позначаються довільним текстовим рядком. Ім'я повинно починатися з маленької літери, якщо воно містить кілька слів, то інші слова, крім першого, пишуться з великої літери.

– Методи повинні мати параметри (якщо вони потрібні) і результат виконання (тип, що повертається).

3. Правильне визначення зв'язків:

– Взаємодії між класами (зв'язки) повинні бути логічно визначені на основі предметної області.

– Кардинальність або кратність (один до одного, один до багатьох і т. д.) має бути чітко вказана для кожного зв'язку.

4. Позначення видимості атрибутів і методів:

– "+" (плюс) перед атрибутом або методом позначає публічну видимість (доступний всім).

– "-" (мінус) означає приватну видимість (доступний тільки всередині класу).

– "#" (геш) означає захищену видимість (доступний всередині класу та його нащадків).

– "~" означає пакетний атрибут, він є відкритим, але тільки в межах свого пакета.

Квантор видимості може бути опущений.

5. Забезпечення логічної організації діаграми:

– Класи повинні бути розташовані так, щоб діаграма була легко читабельною.

– Стрілки і зв'язки не повинні перетинатися або перекривати один одного без потреби.

*Стрілки та зв'язки у UML діаграмі класів (відношення між класами)*

Зв'язки між класами відображають їх взаємодію, і для цього використовуються різні типи стрілок, кожна з яких має своє значення.

1. Асоціація (Association):

Асоціація відображає загальний зв'язок між двома класами, що показує, як одна сутність взаємодіє з іншою.

Вид стрілки: Проста лінія.

Кардинальність: На кожному кінці зв’язку вказуються числа, що показують кількість об’єктів, які можуть брати участь у зв’язку (1:1, 1:N, M:N).

Приклад (рис. 2.1): Клас `Пасажир` асоційований із класом `Квиток` (1 пасажир може купити кілька квитків, а один квиток може бути придбаний на одного пасажира).


Рисунок 2.1 – Приклад асоціації між класами

Асоціація може бути рефлексивною, це означає, що один екземпляр класу звертається до іншого екземпляру цього класу. Наприклад, менеджер підпорядкований топ-менеджеру (рис. 2.2).


Рисунок 2.2 – Приклад рефлексивної асоціації

До Асоціації застосовні чотири базових доповнення: ім'я, роль, кратність і агрегування.

Асоціації можна присвоїти ім'я. Зазвичай в якості імені використовується дієслово або пропозиція, що пояснює причину виникнення зв'язку. Закінчення лінії асоціації в місці, де вона з'єднується з класом, називається роллю асоціації.

Назва ролі асоціації – це зазвичай іменник, яке уточнює, описує роль, в якій один клас бере участь у зв'язку з іншим класом. Роль може бути вказана для обох класів, що беруть участь у зв'язку, або тільки для одного (рис. 2.3).

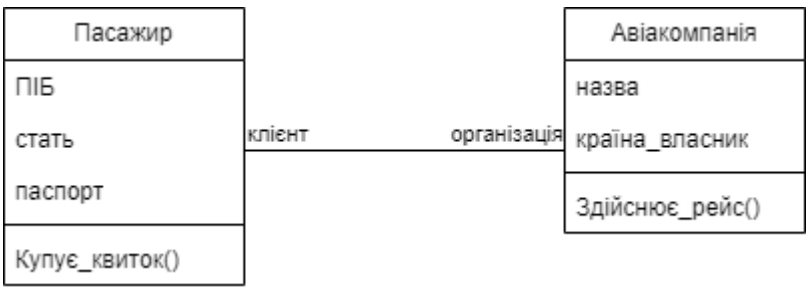


Рисунок 2.3 – Приклад асоціації між класами із вказаними ролями

Кратність (кардинальність, потужність) визначається для класів і вказує допустиму кількість об'єктів (екземплярів класу), що беруть участь у відношенні (рис. 2.4).

Приклади індикаторів потужності представлено в таблиці 2.1.

Таблиця 2.1 – Приклади індикаторів кардинальності

| Індикатор кардинальності | Опис |
|---|---|
| 0..1 | нуль або один |
| 1 | рівно один |
| 1..* | один або багато |
| * | багато |

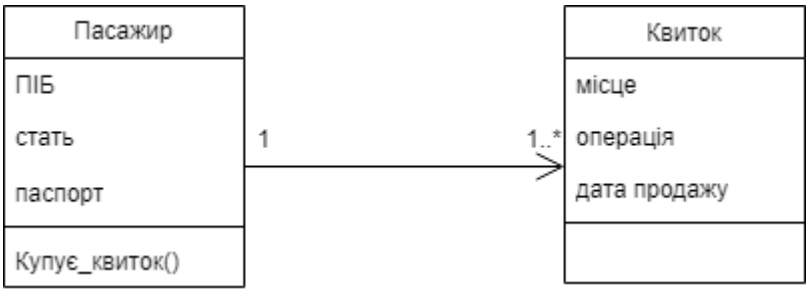


Рисунок 2.4 – Асоціація між класами із вказаною кардинальністю

2. Агрегація (Aggregation):

Агрегація показує відношення "частина-ціле", де один клас є частиною іншого. Однак частини можуть існувати незалежно від цілого.

Вид стрілки: Суцільна лінія з порожнім ромбом на кінці, що вказує на ціле.

Приклади: Клас «Машина» може мати кілька об’єктів «Колесо», але колеса можуть існувати незалежно від машини; клас «Авіакомпанія» може мати кілька об'єктів «Літак», але літаки можуть існувати незалежно від авіакомпанії, наприклад, літаки можна брати у лізинг (рис. 2.5).

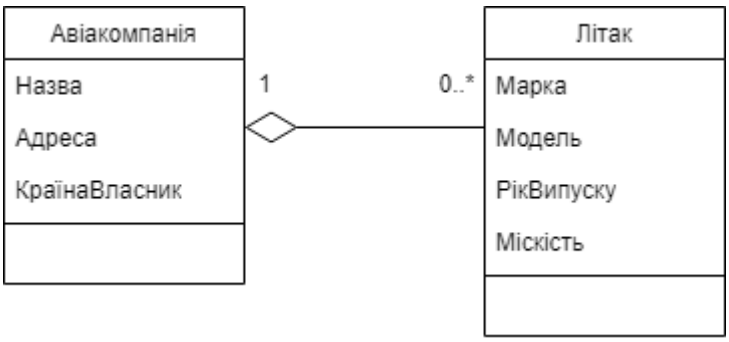


Рисунок 2.5 – Приклад зв’язку агрегації між класами

3. Композиція (Composition):

Композиція також показує відношення "частина-ціле", але частина не може існувати без цілого. Якщо ціле знищується, частини також зникають.

Вид стрілки: Суцільна лінія з заповненим ромбом на кінці, що вказує на ціле.

Приклади: Класи «Книга» і «Глава». Глава не може існувати без книги; класи «Рейс» і «Клас». Класи не можуть існувати без рейсу авіакомпанії (рис. 2.6).

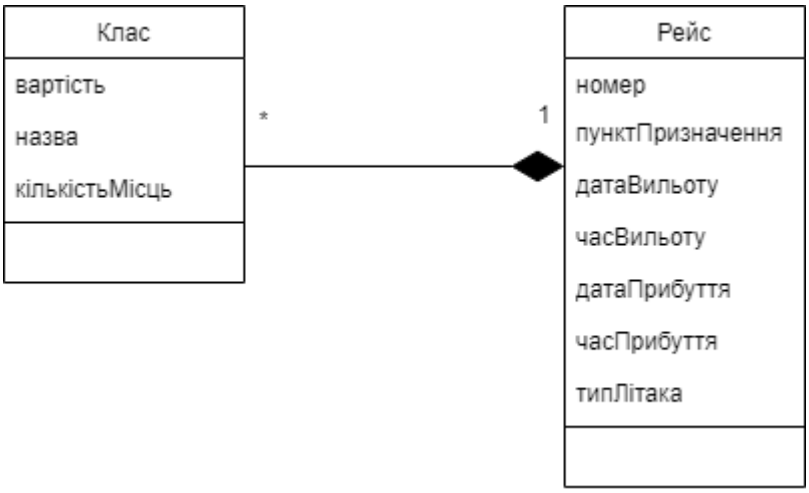


Рисунок 2.6 – Приклад зв’язку композиції між класами

4. Успадкування (Generalization):

Це зв'язок, який вказує на відносини батьківського і дочірнього класу. Дочірній клас наслідує атрибути і методи батьківського класу.

Вид стрілки: Суцільна лінія зі стрілкою з відкритим трикутником на кінці, яка вказує на батьківський клас.

Приклад: класи «Касир», «Льотчик» і «Працівники адміністрації» успадковуються від класу «Співробітник» (рис. 2.7).

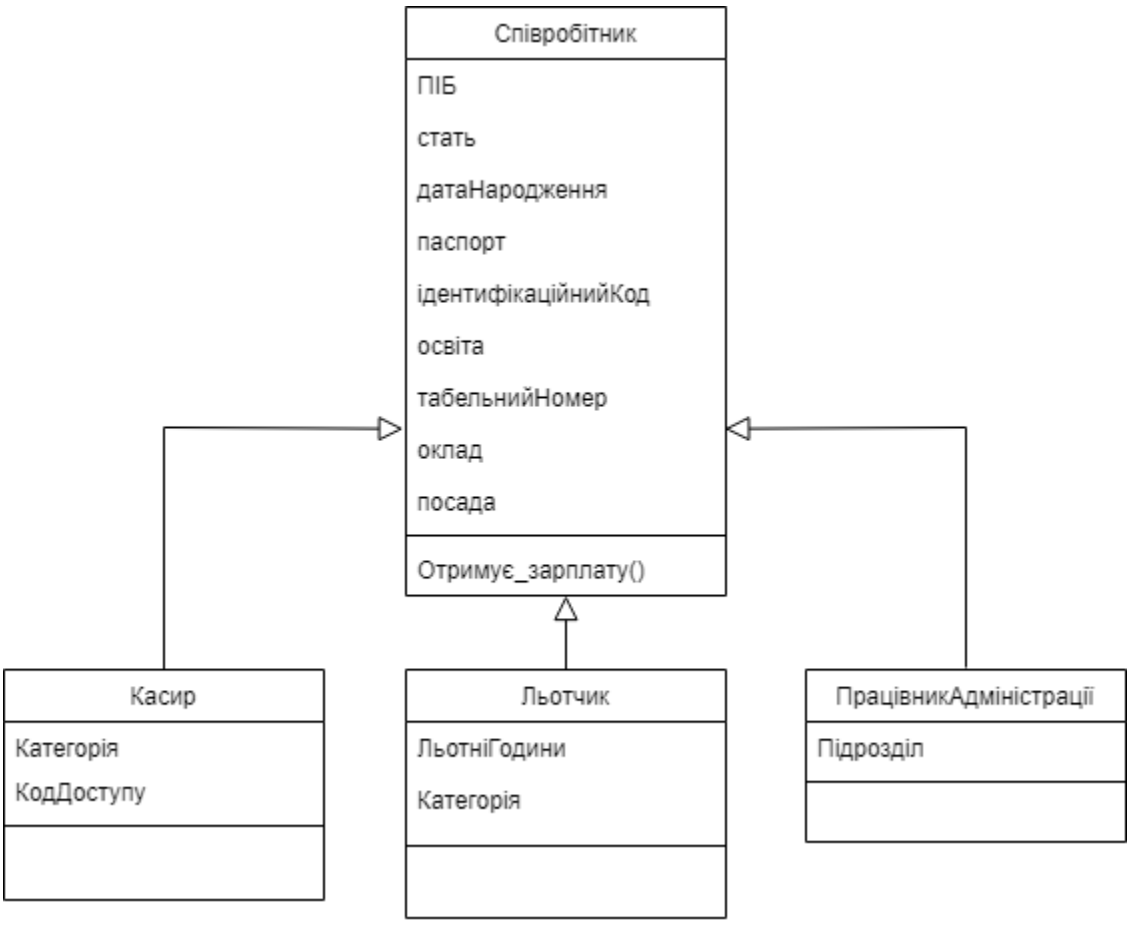


Рисунок 2.7 – Приклад зв’язку успадкування між класами

5. Залежність (Dependency):

Залежність показує, що один клас використовує інший для виконання певної функції. Це слабший тип зв'язку, який зазвичай показує тимчасову взаємодію.

Вид стрілки: Пунктирна лінія зі стрілкою, що вказує на клас, від якого є залежність.

Приклад: Клас «Пасажир» може тимчасово використовувати клас «БронюванняКвитка» для отримання інформації (рис. 2.8).

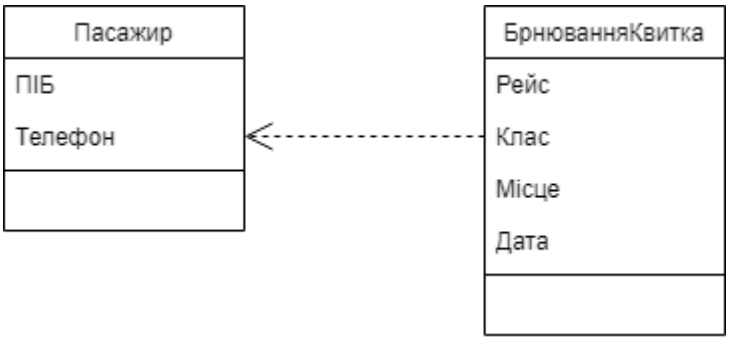


Рисунок 2.8 – Приклад залежності між класами

Підсумок:

– Асоціація: відображає загальний зв'язок між класами.

– Успадкування: показує відношення батьківського і дочірнього класу.

– Агрегація: відношення "частина-ціле", де частина може існувати окремо.

– Композиція: сильніший зв'язок "частина-ціле", де частина не може існувати без цілого.

– Залежність: слабший зв'язок, що показує використання одного класу іншим.

Ці стрілки і зв'язки допомагають чітко візуалізувати відносини між класами в UML діаграмі, що є критично важливим під час проєктування систем або баз даних.

*Приклад UML діаграми класів*

Наведемо приклад *UML* діаграми класів для предметної області «Авіакомпанія» (рис. 2.9).

При аналізі предметної області виділено наступні сутності (класи): «Авіакомпанія», «Рейс», «Квиток», «Клас», «Пасажир» та «Співробітник».

Для кожної сутності виділено відповідні атрибути (властивості). Наприклад для сутності «Авіакомпанія»: «назва», «країна власник». Для решти сутностей представлено на рисунку.

Наступним кроком було визначено зв’язки між сутностями та їх типи. Наприклад, клас «Пасажир» асоційований із класом «Квиток»: пасажир купує квиток; клас «Співробітник» асоційований із класом «Квиток»: співробітник продає квиток; клас «Рейс» пов’язаний із класом «Клас» відношенням композиції: класи є частиною рейсу та не можуть існувати без рейсу; аналогічно клас «Авіакомпанія» пов’язаний із класом «Рейс» зв’язком композиції: рейси є частиною авіакомпанії, оскільки рейси здійснюються певною авіакомпанією, вони не можуть існувати самі по собі; класи «Авіакомпанія» та «Літак» пов’язані зв’язком агрегації: авіакомпанія має літаки, але літаки можуть існувати без авіакомпанії, тобто взяті у лізинг, оренду або найняті на певний чартерний рейс.

Далі визначимо кардинальність зв’язків: один співробітник продає багато квитків, а один і той же квиток може бути проданий тільки одним співробітником – зв’язок один до багатьох; один рейс може мати багато класів, але один і той же клас

може бути тільки на одному рейсі, оскільки містить інформацію про вартість і кількість місць – зв'язок один до багатьох; один пасажир може купити кілька квитків, але один квиток продається на одного пасажира – зв'язок один до багатьох; один клас відноситься до одного рейсу, в той час як один рейс може мати кілька класів (економ, бізнес та інші) – зв'язок багато до одного; авіакомпанія може здійснювати багато рейсів, один вид рейсу здійснюється однією авіакомпанією – зв'язок один до багатьох.

Методи для проєктування бази даних не визначаються.

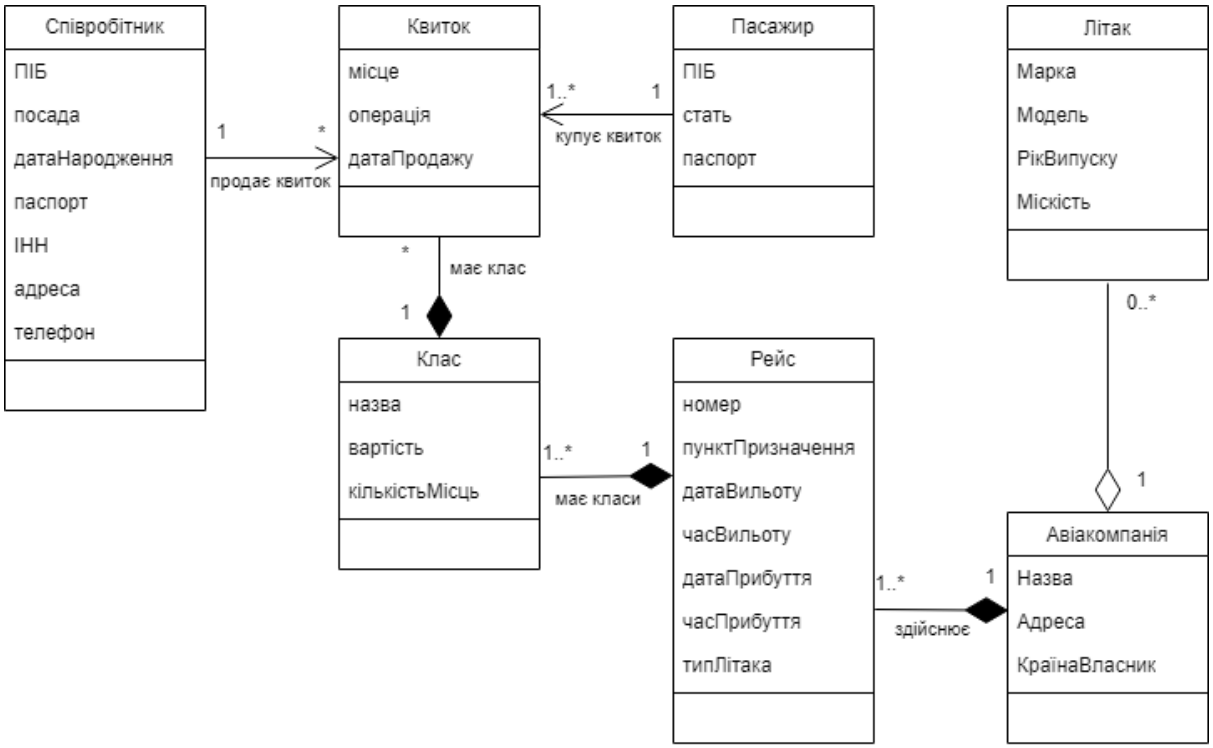


Рисунок 2.9 – Приклад UML діаграми класів діяльності авіакомпанії

*Інструменти для створення UML діаграм:*

– StarUML

StarUML – це популярний інструмент для моделювання систем за допомогою діаграм UML. Його основна мета – допомогти розробникам створювати та документувати програмні системи, використовуючи різні типи діаграм для візуалізації архітектури програмного забезпечення та бізнес-процесів.

Основні характеристики StarUML: підтримка UML, моделювання баз даних, плагіни та розширюваність, генерація коду, зворотне проєктування, підтримка стандартів, кросплатформеність.

Переваги: легкість у використанні, широка підтримка UML-діаграм, генерація коду та зворотне проєктування, розширюваність через плагіни.

Недоліки: платна ліцензія (існує пробна версія), обмежений функціонал порівняно з деякими іншими потужними CASE-інструментами (наприклад, Rational Rose або Enterprise Architect).

StarUML – це зручний інструмент для моделювання та проєктування, що підходить для розробників програмного забезпечення, архітекторів систем та аналітиків.

– Lucidchart

Lucidchart – це хмарний інструмент для створення діаграм, що дозволяє користувачам легко візуалізувати інформацію, моделювати процеси та працювати над

проєктами спільно. Він широко використовується в різних сферах, зокрема у програмному забезпеченні, бізнес-аналітиці та освіті для створення схем, блок-схем, діаграм, моделей баз даних та багатьох інших візуальних інструментів.

Основні характеристики Lucidchart: хмарна платформа, спільна робота в режимі реального часу, шаблони та готові елементи, підтримка різних типів діаграм, інтеграції з багатьма популярними інструментами для бізнесу та розробки, автоматичне створення діаграм, імпорт та експорт, безпека.

Переваги: простий у використанні інтерфейс, доступний з будь-якого пристрою через хмарну платформу, широкий вибір інтеграцій з іншими інструментами, можливості для командної роботи в реальному часі, гнучкість у створенні діаграм різних типів.

Недоліки: обмеженість безкоштовної версії (деякі функції доступні лише в платних підписках), не завжди підходить для дуже складних діаграм великих масштабів.

Lucidchart – це потужний інструмент для візуалізації даних і процесів, який ідеально підходить для тих, хто шукає просте, але гнучке рішення для створення діаграм і схем у хмарному середовищі.

– Visual Paradigm

Visual Paradigm – це комплексне програмне забезпечення для візуального моделювання та проєктування, яке використовується для аналізу, проєктування систем, процесів і баз даних. Він підтримує широкий спектр моделей і діаграм, що дозволяє використовувати його у різних сферах, від розробки програмного забезпечення до управління бізнес-процесами.

Основні особливості Visual Paradigm: підтримка UML і BPMN(Business Process Model and Notation), інструменти для управління вимогами, підтримка ERD (Entity-Relationship Diagram), автоматична генерація коду та зворотне проєктування, інструменти для розробки баз даних, Agile і Scrum інструменти, співпраця в команді, інтеграція з іншими інструментами популярними інструментами, інструменти для керування архітектурою підприємства (Enterprise Architecture), система версій.

Visual Paradigm підтримує контроль версій для проєктів, що дозволяє відслідковувати зміни і відновлювати попередні версії моделей, що зручно для команд, які працюють над проєктами в декілька етапів.

Переваги: широкий функціонал для різних типів моделювання (UML, BPMN, ERD, ArchiMate тощо), можливість інтеграції з іншими розробницькими інструментами, підтримка командної роботи та керування версіями проєктів, автоматичне генерування коду і SQL-скриптів, простий інтерфейс для створення діаграм і моделей.

Недоліки: складність, вартість (хоча є безкоштовна версія, більшість просунутих функцій доступні тільки в платних планах).

Visual Paradigm – це потужний і універсальний інструмент для моделювання, який підходить як для розробників програмного забезпечення, так і для бізнес-аналітиків. Завдяки широкій підтримці різних типів діаграм, він дозволяє моделювати процеси, системи та бази даних на будь-якому етапі розробки або бізнес-аналізу.

– Draw.io

Draw.io (нині відомий як diagrams.net) – це безкоштовний онлайн-інструмент для створення діаграм і схем, який широко використовується для візуалізації різних типів моделей і процесів. Він є однією з найпопулярніших платформ для створення діаграм через свою простоту, гнучкість і доступність.

Основні особливості: простота використання, інтегрується з різними хмарними платформами, такими як Google Drive, OneDrive, Dropbox, GitHub і інші; офлайн-режим,

співпраця в реальному часі, експорт та імпорт діаграм, шаблони та бібліотеки, інтеграція з Jira та Confluence, безпека та конфіденційність, відкритий вихідний код.

Переваги: доступність, широкий набір функцій, гнучкість, простий інтерфейс, безпека.

Недоліки: відсутність деяких просунутих функцій, залежність від Інтернету в онлайн-версії.

Draw.io (diagrams.net) – це один із найкращих безкоштовних інструментів для створення діаграм і схем. Завдяки простоті, широкій підтримці форматів і можливості інтеграції з хмарними сервісами, він є чудовим вибором для індивідуальних користувачів та команд. Хоча він може не мати таких розширених можливостей, як деякі платні інструменти, він забезпечує всі необхідні функції для створення високоякісних діаграм.

Приклад роботи з онлайн версією diagrams.net

Після переходу на сайт програмного забезпечення, обираємо потрібну дію: створити нову діаграму чи відкрити існуючу.

У вікні створення нової діаграми можна обрати шаблон нової діаграми або обрати за замовчуванням «Порожня діаграма», ввести її назву та натиснути кнопку «Створити», обравши відповідний шлях.

Зліва у вертикальній частині знайдете перелік фігур, які можна перетягувати на діаграму. Для створення UML діаграми класів необхідно зліва прокрутити до пункту UML та розгорнути його (рис. 2.10).

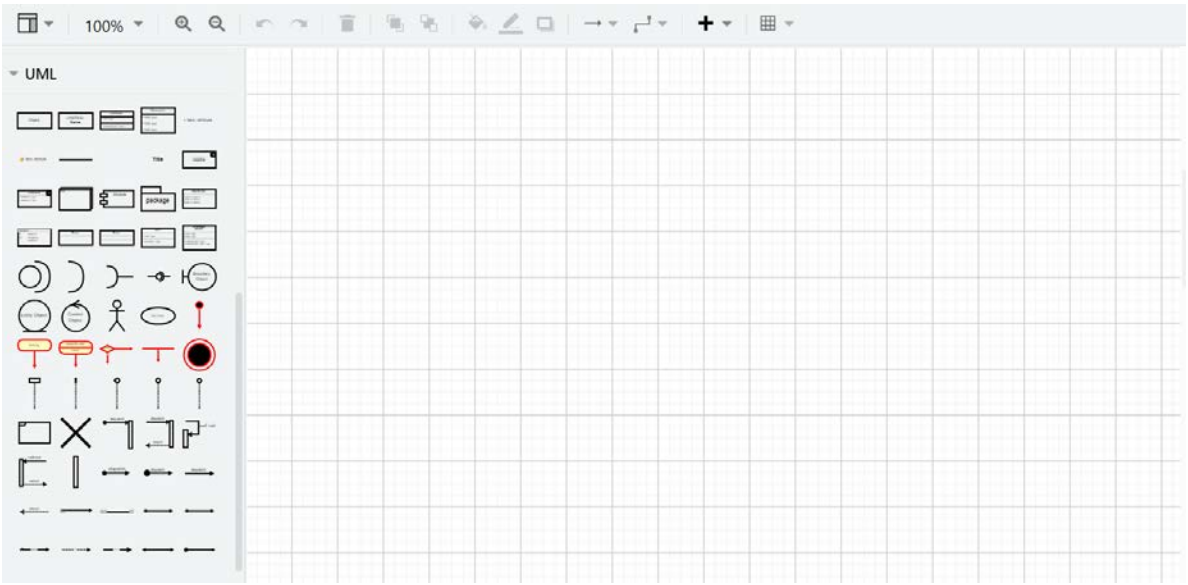


Рисунок 2.10 – Створення нової UML діаграми класів

Підводячи курсор до відповідного інструменту, підсвічується його назва та інструмент показується у збільшеному вигляді (рис. 2.11). Натискаємо на потрібний інструмент та перетягуємо не відпускаючи на поле діаграми.

При натисканні на елемент діаграми правою кнопкою «миші», з’являється контексте меню з налаштуваннями. Таким чином можна змінити напрям стрілки на протилежний. З правої сторони робочого вікна знаходяться налаштування стилю, тексту тощо.

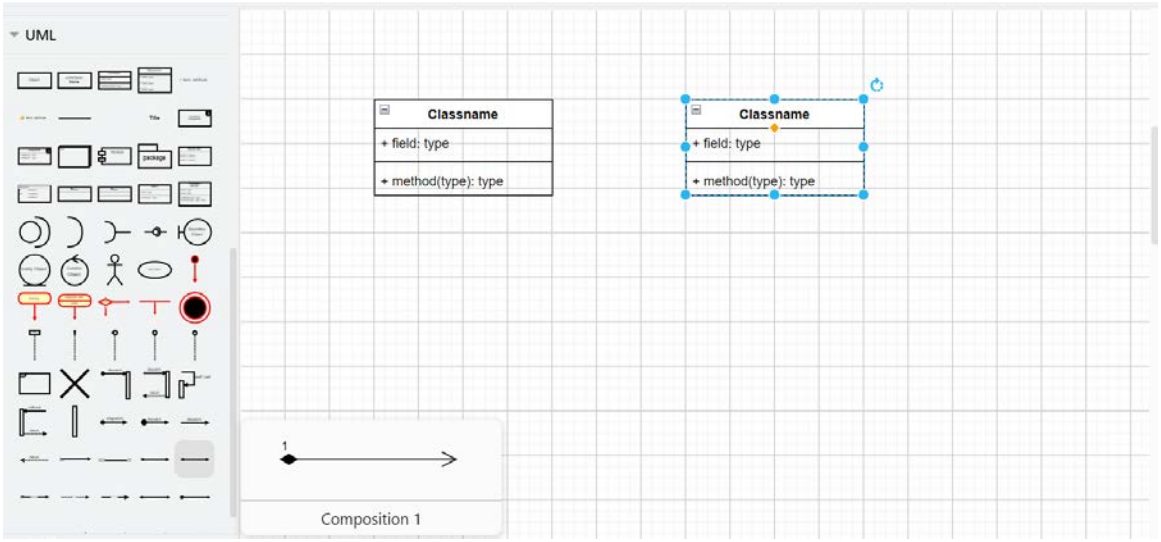

Рисунок 2.11 – Приклад вибору інструмента в робочому вікні

Висновки. Метод створення концептуальної моделі бази даних через UML діаграму класів є потужним способом візуалізації структури системи та її взаємозв'язків на ранньому етапі проєктування. Він дозволяє побудувати абстрактну модель, яка згодом легко перетворюється в логічну і фізичну модель бази даних.

*Структура звіту до лабораторної роботи*

1. Завдання (опис предметної області)

2. Виділені суттєві сутності, що характеризують предметну область, та їх характеристика. Наприклад в таблиці, як представлено в табл. 2.2.

Таблиця 2.2 – Сутності, що характеризують предметну область

| Сутність | Характеристика |
|---|---|
| Авіакомпанія | Містить інформацію про авіакомпанію, що здійснює рейси |
| Рейс | Містить інформацію про рейси, які обслуговує дана компанія |
| Квиток | Містить інформацію про квитки на певний клас рейсу авіакомпанії |
| … | … |

3. Опис атрибутів сутностей у вигляді таблиці 2.3. *Мінімум 5 атрибутів для кожної із основних сутностей.*

Таблиця 2.3 – Властивості сутностей, що характеризують предметну область

| Сутність | Атрибут |
|---|---|
| Рейс | номер |
| | пункт призначення |
| | дата вильоту |
| | час вильоту |
| | … |
| Пасажир | ПІБ |
| | стать |
| | паспорт |
| | … |
| … | … |

4. Опис зв’язків між сутностями предметної області. Наприклад, пасажир купує квиток, співробітник продає квиток, авіакомпанія здійснює рейси. Результат подати у вигляді таблиці 2.4.

Таблиця 2.4 – Опис зв’язків між сутностями

| Сутність 1 | Назва зв’язку | Сутність 2 |
|---|---|---|
| Авіакомпанія | Здійснює | Рейс |
| Пасажир | Купує квиток | Квиток |
| Співробітник | Продає квиток | Квиток |
| … | … | … |

5. Розроблена діаграма класів UML, що відображає концептуальну модель бази даних для заданої предметної області. Приклад представлено на рисунку 2.9.

Звіт до лабораторної роботи 2 можна здати онлайн на сайті ДО edu.op.edu.ua до початку вашого заняття.