Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут» імені Ігоря Сікорського
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

Лабораторна робота № 2

із дисципліни: «Технології розроблення програмного забезпечення»

на тему: «ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ. СЦЕНАРІЇ ВАРІАНТІВ ВИКОРИСТАННЯ. ДІАГРАМИ UML. ДІАГРАМИ КЛАСІВ. КОНЦЕПТУАЛЬНА МОДЕЛЬ СИСТЕМИ**»**

Виконав:
студент групи ІА-34
Чунаков А.Р.
Перевірив:
Мягкий Михайло Юрійович

Київ-2025

Мета: Розібрати тему лабораторної роботи, розробити потрібні діаграми для специфікації, візуалізації, проектування та документування програми. Навчитися правильно оформлювати діаграми, та реалізувати шаблон Ропезиторій у застосунку.

Короткі теоретичні відомості

Діаграма прецедентів (або діаграма варіантів використання) (англ. Use case diagram) — в UML, діаграма, на якій зображено відношення між акторами та прецедентами в системі.

Актор – є дійовою особою на діаграмі прецедентів (наприклад користувач чи адмін системи).

Прецеденти у свою чергу являють собою варіанти використання функціоналу системи.

Між акторами та прецедентами є **зв'язки**, які їх з'єднують. Є декілька видів звязків: асоціації, включення, розширення, узагальнення.

Уніфікована мова моделювання (UML) — це стандартна графічна мова опису, призначена для візуалізації, специфікації, проектування та документування компонентів програмних систем. Вона дозволяє представити архітектуру проекту за допомогою набору діаграм, які відображають як статичну структуру системи (класи, об'єкти, компоненти), так і її динамічну поведінку (процеси, взаємодію, стани), забезпечуючи тим самим єдине розуміння логіки та функціональних вимог між усіма учасниками розробки — від бізнес-аналітиків до програмістів.

Діаграма класів (Class Diagram) — це ключовий тип структурних діаграм UML, що моделює статичну архітектуру програмної системи, відображаючи її будову у вигляді набору класів, інтерфейсів та відношень між ними. Вона деталізує внутрішню структуру сутностей, включаючи їхні атрибути (властивості) та методи (операції), а також визначає характер взаємодії між об'єктами через механізми успадкування, асоціації, агрегації чи залежності. Ця діаграма є фундаментальною для об'єктно-орієнтованого проектування, оскільки слугує безпосередньою "кресленням" для написання програмного коду та формування структури бази даних.

Клас (на діаграмі класів UML) — це графічний елемент, представлений у вигляді прямокутника з трьома секціями, що слугує для візуалізації типу об'єктів системи. Перша секція містить унікальне ім'я класу, яке ідентифікує його в моделі. Друга секція перелічує атрибути (властивості) класу, визначаючи їхні

імена, типи даних та рівні доступу (видимість). Третя секція описує операції (методи), які клас може виконувати, вказуючи їх сигнатуру: вхідні параметри та тип даних, що повертається. Цей елемент є базовою одиницею інкапсуляції при проектуванні структури програмного забезпечення.

Патерн Репозиторій (Repository Pattern) — це архітектурний шаблон проектування, який створює рівень абстракції між бізнес-логікою програми та шаром доступу до даних (Data Access Layer). Його головна мета — ізолювати доменну логіку від деталей механізму зберігання (таких як SQL-запити, з'єднання з БД чи XML-файли), надаючи інтерфейс для роботи з даними так, ніби вони знаходяться в колекції в пам'яті (наприклад, методи Add, Remove, Get, Find). Використання репозиторію забезпечує слабку зв'язність компонентів (loose coupling), полегшує написання модульних тестів (Unit Tests) через можливість підміни реальної бази даних на імітацію (mock), та централізує логіку побудови запитів в одному місці.

Обрана тема лабораторної роботи: **SQL IDE**

9. **Sql IDE** (singleton, command, observer, bridge, visitor, SOA)

Інтегроване середовище розробки скриптів SQL повинна вміти працювати з більшістю сучасних баз даних (Microsoft SQL server, Oracle, IBM DB2, MySQL, PostgreSQL), виділяти синтаксичні одиниці конструкцій, виконувати і показувати результат їх виконання, отримувати схеми даних з джерела (список таблиць, стовпців, ключів і т.д.), порівнювати схеми даних з різних джерел.

2. Аналіз теми: SQL IDE

Тема роботи — «SQL IDE», що є інтегрованим середовищем розробки, призначеним для роботи з різними системами управління базами даних. Цей проєкт є настільним додатком-клієнтом, який надає користувачу єдиний інтерфейс для підключення до різноманітних СУБД, таких як PostgreSQL, MySQL, Oracle та SQLite. Основний функціонал програми включає редактор SQL-запитів з підсвічуванням синтаксису, механізм виконання цих запитів з відображенням результатів у таблиці, а також інструменти для аналізу структури (схеми) підключеної бази даних.

Схема прецеденту:

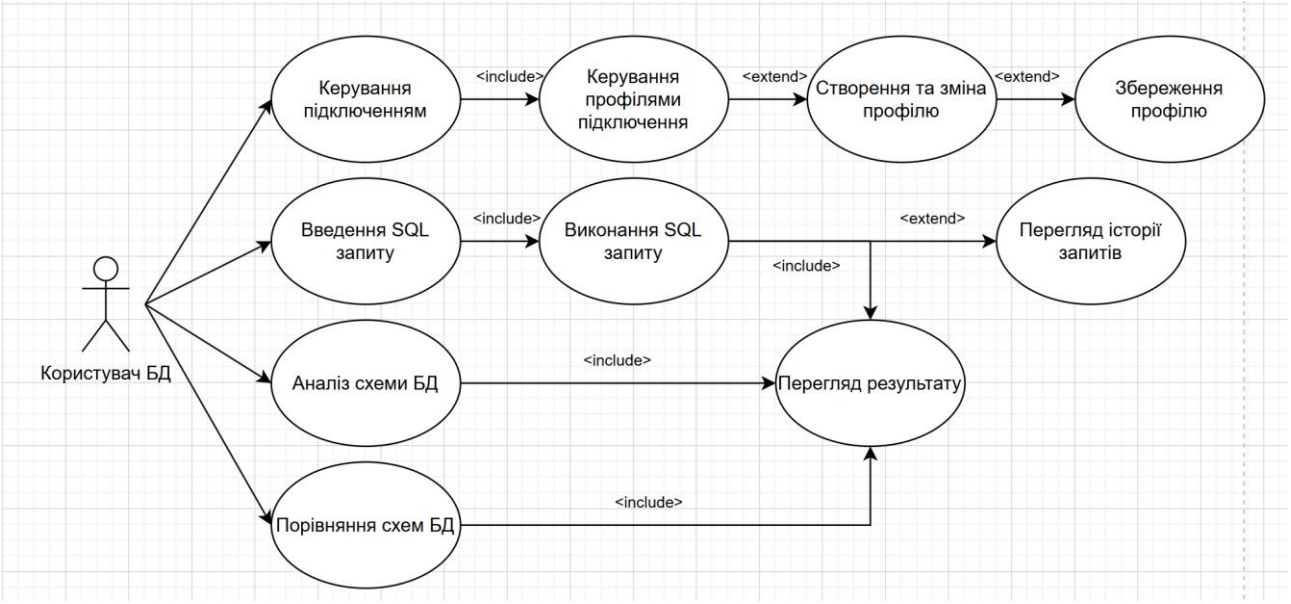

Рис1.1 – use-case діаграма функціональних вимог до системи

Пояснення до Діаграми Варіантів Використання (з розширеним функціоналом)

Ця діаграма варіантів використання (Use Case Diagram) ілюструє функціональні можливості SQL IDE. Вона відображає взаємодію основного актора з системою та залежності між різними варіантами використання.

1. Актор (Actor)

- **Користувач БД**: Єдиний актор системи, який представляє будь-яку особу (розробника, адміністратора), що використовує додаток для роботи з базами даних.

2. Основні Варіанти Використання (Use Cases)

Діаграма деталізує чотири основні групи сценаріїв:

1) "Керування підключенням"

- Керування підключенням: Базовий сценарій, з якого починається робота. Користувач ініціює процес з'єднання з базою даних.
- Керування профілями підключення: Цей варіант використання пов'язаний з базовим відношенням <<include>>. Це означає, що процес підключення нерозривно пов'язаний з вибором або налаштуванням параметрів (профілю).
- Створення та зміна профілю: Це додаткова функціональність (<<extend>>), яка дозволяє користувачу не просто вибрати існуючий профіль, а й створити новий або відредагувати поточний.
- Збереження профілю: Логічне завершення процесу створення/зміни (<<extend>>), що відповідає за запис налаштувань у внутрішню базу даних програми.

2) Група "Робота з SQL запитами"

- Введення SQL запиту: Дія користувача з написання коду в редакторі.
- Виконання SQL запиту: Цей сценарій обов'язково включає (<<include>>) попереднє введення тексту запиту.
- Перегляд історії запитів: Додаткова функція (<<extend>>), доступна користувачу. Вона дозволяє переглянути список раніше виконаних команд, що спрощує повторне використання коду.
- Перегляд результату: Завершальний етап виконання запиту (<<include>>), де система відображає дані або статус операції.

3) Група "Аналіз структури"

- Аналіз схеми БД: Функція для отримання структури поточної бази даних (таблиць, стовпців). Вона також завершується обов'язковим переглядом результату (<<include>>).

4) Група "Порівняння"

- Порівняння схем БД: Сценарій для аналізу відмінностей між двома базами даних, який також включає відображення звіту про порівняння (<<include>>).

3. Діаграма класів:

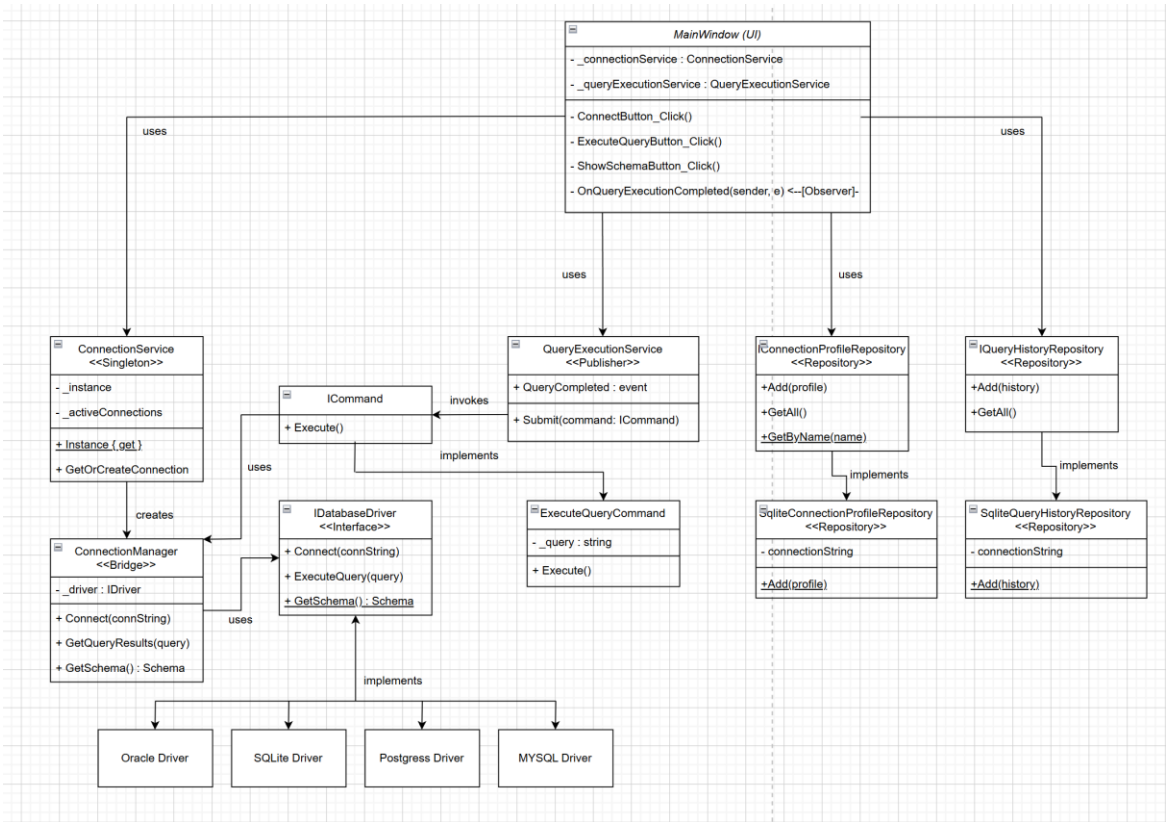

Рис. 1.2 – Діаграма класів системи

Ця діаграма класів ілюструє ключові компоненти, що складають SQL IDE, та зв'язки між ними. Архітектура побудована навколо набору класичних патернів проєктування, щоб забезпечити гнучкість, розширюваність та низьку зв'язність компонентів.

1) Рівень UI (MainWindow)

MainWindow (UI): Це головний клас, що відповідає за інтерфейс користувача. Він виступає в ролі "контролера" або "презентера".

- Він має посилання (uses) на два основні сервіси: ConnectionService та QueryExecutionService.
- Він містить обробники подій, які ініціюють логіку у відповідь на дії користувача.
- Він реалізує патерн Observer через метод OnQueryExecutionCompleted. Цей метод підписаний на подію QueryCompleted від QueryExecutionService і оновлює UI, коли отримує сповіщення.

2) Сервісний Рівень (Singleton, Observer, Command)

- ConnectionService (<<Singleton>>):
  - Цей клас реалізує патерн Singleton. Він гарантує, що у всього додатка є лише один екземпляр цього сервісу, доступний через статичну властивість Instance.
  - Його головна відповідальність — керувати активними підключеннями. Метод GetOrCreateConnection створює та повертає екземпляр ConnectionManager.
- QueryExecutionService (<<Publisher>>):
  - Цей клас виступає у двох ролях: Invoker у патерні Command та Publisher у патерні Observer.
  - Як Invoker, він приймає об'єкти, що реалізують ICommand (через метод Submit), і викликає їх виконання (invokes).
  - Як Publisher, він містить подію QueryCompleted, на яку можуть підписатися інші класи (наприклад, MainWindow), щоб отримати сповіщення про завершення запиту.

- ICommand (Інтерфейс) та ExecuteQueryCommand:
  - Ці класи реалізують патерн Command.
  - ICommand — це інтерфейс, що інкапсулює дію в один метод Execute().
  - ExecuteQueryCommand — це конкретна команда, що реалізує цей інтерфейс. Вона містить усю логіку, необхідну для виконання SQL-запиту. Для цього вона використовує ConnectionManager, щоб отримати доступ до активного з'єднання з БД (хоча на схемі зв'язок "uses" помилково веде від ICommand, логічно він належить ExecuteQueryCommand).

3) Рівень Доступу до Даних (Bridge)

Це ядро всієї архітектури, що забезпечує підтримку різних СУБД.

- ConnectionManager (<<Bridge>>):
  - Це основний клас патерну Bridge. Він діє як абстракція, що відокремлює логіку *що* ми хочемо зробити (наприклад, GetSchema) від того, *як* це зробити.
  - Він має посилання на інтерфейс IDatabaseDriver і делегує йому всю "важку" роботу.
- IDatabaseDriver (<<Interface>>):
  - Це інтерфейс реалізації патерну Bridge. Він визначає контракт (набір методів Connect, ExecuteQuery, GetSchema), який повинен реалізувати кожен конкретний драйвер бази даних.
- Oracle Driver, SQLite Driver, Postgress Driver, MYSQL Driver:
  - Це конкретні реалізації патерну Bridge. Кожен з цих класів реалізує інтерфейс IDatabaseDriver, містить специфічний для своєї СУБД код.

4) Рівень Внутрішніх Даних (Repository) Цей рівень відповідає за збереження налаштувань програми, профілів підключення та історії запитів, ізолюючи логіку програми від способу зберігання даних.

IConnectionProfileRepository та IQueryHistoryRepository (<<Interface>>):

- Цей шар визначає контракти (інтерфейси) для доступу до внутрішніх даних.

- Вони декларують методи (наприклад, Add, GetAll, GetByName), які необхідні додатку для роботи з даними, але приховують деталі реалізації (наприклад, чи це файл, чи база даних).
- MainWindow залежить саме від цих інтерфейсів, що забезпечує слабку зв'язність (loose coupling).

SqliteConnectionProfileRepository та SqliteQueryHistoryRepository:

- Це конкретні реалізації патерну Repository.
- Вони містять безпосередній код взаємодії з локальною базою даних SQLite (ide_settings.db).
- Ці класи реалізують методи інтерфейсів, виконуючи INSERT та SELECT запити для збереження та відновлення стану програми (профілів та історії) між сесіями.

4. Use Cases

Сценарій 1: Керування підключенням

Назва: Керувати підключенням

Актор: Користувач БД

Мета: Встановити активне з'єднання з цільовою базою даних.

Тригер: Користувач запускає програму та натискає кнопку "Підключитись".

Основний потік:

1. Користувач обирає тип цільової СУБД з випадаючого списку в інтерфейсі.
2. Користувач вводить коректний рядок підключення у відповідне текстове поле.
3. Користувач натискає кнопку для підтвердження підключення.
4. Система валідує введені дані.
5. Система, використовуючи відповідний для обраного типу СУБД драйвер, намагається встановити з'єднання з фізичною базою даних.
6. Успішний результат: Система активує основний робочий простір (редактор запитів, кнопки) та виводить у рядку стану повідомлення про успішне підключення.

Альтернативний потік (Помилка):

7. На кроці 5 система не може встановити з'єднання (наприклад, невірний пароль або сервер недоступний).
8. Система перехоплює помилку.
9. Система виводить діалогове вікно з текстом помилки. Робочий простір залишається неактивним.

Сценарій 2 (простий): Аналіз схеми БД

Назва: Аналізувати схему БД

Актор: Користувач БД

Мета: Отримати та переглянути структуру (таблиці та стовпці) поточної бази даних.

Передумова: Сценарій UC-1 успішно завершено, з'єднання встановлено.

Основний потік:

1. Користувач натискає кнопку "Показати схему".
2. Система звертається до активного підключення і запитує метадані (схему).
3. Спеціалізований драйвер для поточної СУБД виконує необхідні системні запити для отримання списку таблиць та їх стовпців.
4. Система отримує ці дані та будує внутрішню об'єктну модель схеми.
5. Спеціальний компонент обходить цю модель та перетворює її у відформатоване текстове представлення.
6. Успішний результат: Система відображає отриманий текст зі структурою схеми у діалоговому вікні.

Альтернативний потік: Втрата з'єднання з сервером БД

- Умова: Потік починається на кроці 3 основного потоку, якщо під час виконання системного запиту сервер стає недоступним (наприклад, зникло живлення сервера, обірвався мережевий кабель або впав VPN).
- Кроки:
    1. Драйвер СУБД намагається виконати запит до системних таблиць, але не отримує відповіді протягом встановленого часу (Timeout) або отримує помилку мережі.
    2. Система (рівень ConnectionManager) перехоплює виняткову ситуацію (Exception).

3. Процес побудови моделі схеми припиняється (об'єкт SchemaRoot не створюється).
4. Система виводить спливаюче вікно з повідомленням про помилку.
5. Сценарій завершується, система повертається до стану очікування дій користувача (користувач може спробувати підключитися знову).

Сценарій 3 (складний): Виконання SQL-запиту

Назва: UC-3: Виконання запиту

Актор: Користувач БД

Мета: Виконати SQL-запит до підключеної БД та отримати результат.

Опис: Це найскладніший сценарій, оскільки він задіює всю основну архітектуру системи та декілька архітектурних патернів (Команда, Спостерігач, Міст).
Через свою складність він представлений у вигляді діаграми послідовності.

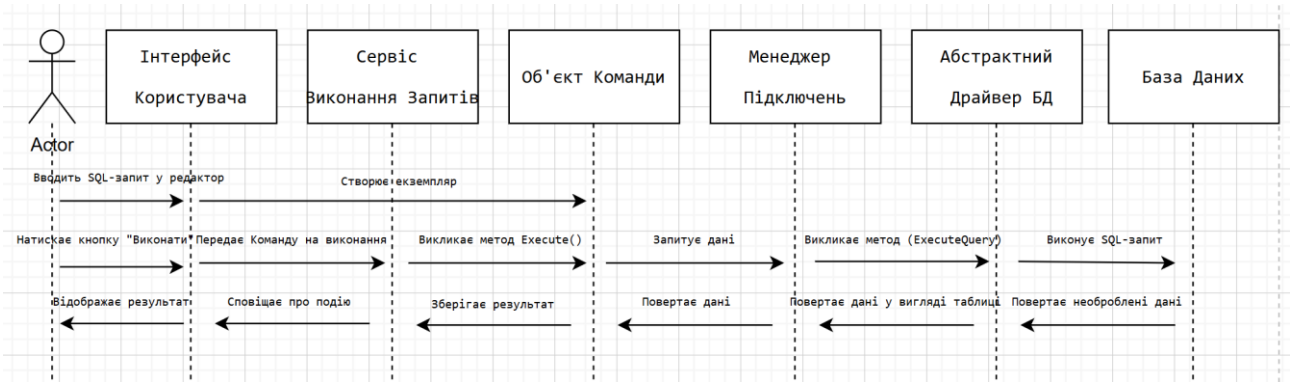

Рис. 1.3 –Діаграма послідовностей для сценарію Виконання SQL-запиту

Детальний опис кроків:

1. Користувач вводить текст SQL-запиту у текстовий редактор в інтерфейсі.
2. Користувач натискає кнопку "Виконати запит".
3. Інтерфейс Користувача (UI) створює спеціальний "Об'єкт Команди" (патерн Command), інкапсулюючи в ньому текст запиту.
4. UI передає цей об'єкт централізованому "Сервісу Виконання Запитів".
5. Сервіс викликає метод виконання у переданої команди.
6. Команда, у свою чергу, звертається до активного "Менеджера Підключень" і просить його виконати запит.

7. Менеджер Підключень не знає про специфіку бази даних; він делегує виконання "Абстрактному Драйверу БД" (патерн Bridge).
8. Конкретний драйвер (наприклад, драйвер для PostgreSQL або SQLite) формує і відправляє запит до фізичної Бази Даних.
9. База Даних повертає результат.
10. Драйвер форматує результат у стандартизований вигляд (наприклад, таблицю даних) і повертає його Менеджеру Підключень, а той – Об'єкту Команди.
11. "Сервіс Виконання Запитів" отримує контроль назад і генерує подію (патерн Observer), сповіщаючи всіх підписаних слухачів про те, що запит завершено (успішно або з помилкою), і додаючи до події отримані дані.
12. Інтерфейс Користувача, будучи підписаним на цю подію, отримує сповіщення.
13. UI оновлює відповідні елементи – заповнює таблицю результатів та оновлює текст у рядку стану, показуючи результат користувачу.

5. Розробити основні класи і структуру системи баз даних.

Клас	Призначення	Основні поля
ConnectionProfile	Описує профіль підключення, який зберіг користувач.	`Id`, `ProfileName`, `DatabaseType`, `ConnectionString`, `LastUsed`
QueryHistory	Описує один виконаний SQL-запит.	`Id`, `QueryText`, `ExecutionTime`, `WasSuccessful`, `ConnectionProfileId`
UserSetting	Описує налаштування інтерфейсу у форматі "Ключ-Значення"	`Id`, `SettingKey`, `SettingValue`

6. Класи даних повинні реалізувати шаблон Репозиторію для взаємодії з базою даних.

Реалізований шаблон Репозиторій знаходиться за посиланням нижче:

https://github.com/1chuny/SQL-IDE/tree/main/DataAccess/Repositories

Висновок: У ході виконання лабораторної роботи було успішно проаналізовано предметну область та спроєктовано архітектуру SQL IDE, де розробка комплекту UML-діаграм стала вирішальним етапом для переходу від абстрактних вимог до програмної реалізації. Створені діаграми варіантів використання, класів, послідовності та розгортання дозволили візуалізувати логіку роботи системи та чітко визначити зв'язки між компонентами, що стало фундаментом для коректного впровадження складних патернів проєктування, зокрема шаблону Репозиторій для роботи з внутрішніми даними. Такий підхід до специфікації та документування не лише структурував процес розробки, але й забезпечив створення гнучкої та масштабованої архітектури, мінімізуючи ризики виникнення логічних помилок при написанні коду завдяки детальному попередньому плануванню.

Контрольні питання:

1. Що таке UML?

UML (Unified Modeling Language), або Уніфікована мова моделювання, — це стандартизована, графічна нотація, розроблена для візуалізації, специфікації, конструювання та документування артефактів програмних систем. UML використовується для моделювання об'єктно-орієнтованих систем і надає багатий набір діаграм для опису системи з різних точок зору, від функціональних вимог користувача до архітектури та деталей реалізації.

2. Що таке діаграма класів UML?

Діаграма класів UML — це статична структурна діаграма, яка є основною будівельною одиницею в об'єктно-орієнтованому моделюванні, оскільки вона показує класи системи, їхні внутрішні структури (атрибути та операції) та зв'язки (асоціації, спадкування, агрегації, композиції) між ними, представляючи архітектуру системи на рівні, який безпосередньо корелює з кодом об'єктно-орієнтованих мов програмування.

3. Які діаграми UML називають канонічними?

Термін "канонічні діаграми" не є офіційним у специфікації UML, проте в практиці розробки програмного забезпечення під ним зазвичай мають на увазі основний, найбільш важливий набір діаграм, необхідних для всебічного опису системи, до якого традиційно відносять: діаграму класів (для структури), діаграму варіантів використання (для вимог), діаграму послідовності (для часової взаємодії) та діаграму станів (для поведінки об'єкта).

4. Що таке діаграма варіантів використання?

Діаграма варіантів використання (Use Case Diagram) — це поведінкова діаграма UML, яка моделює функціональні вимоги системи, графічно зображуючи її межі, акторів (зовнішніх користувачів або системи) та варіанти використання (функції або послуги, які система надає), а також відношення $<<include>>$ та $<<extend>>$ між самими варіантами використання, що допомагає зрозуміти, що система повинна робити, з точки зору взаємодії з користувачем.

5. Що таке варіант використання?

Варіант використання (Use Case) — це опис послідовності дій, які виконує система, щоб надати акторові (людині або іншій системі) певний значущий, вимірний результат (цінність). Він слугує для фіксації функціональності системи, описуючи, як зовнішній об'єкт взаємодіє з системою для досягнення конкретної мети, і є ключовим елементом визначення вимог.

6. Які відношення можуть бути відображені на діаграмі використання?

На діаграмі варіантів використання відображаються такі основні відношення: Асоціація (зв'язок між актором і варіантом використання), Узагальнення/Спадкування (для акторів або для варіантів використання), Включення ($<<include>>$) — коли один варіант використання обов'язково викликає інший (спільна функціональність), і Розширення ($<<extend>>$) — коли один варіант використання необов'язково додає функціональність до іншого за певних умов.

7. Що таке сценарій?

Сценарій (Scenario) — це конкретний примірник (екземпляр) варіанта використання, який являє собою одну чітку, послідовну стежку виконання (послідовність кроків або подій), що виконується системою та актором за певних умов для досягнення кінцевої мети варіанта використання. Сценарії деталізують поведінку, яку описує варіант використання, наприклад, "успішний сценарій" або "сценарій помилки".

8. Що таке діаграма класів?

Див. відповідь на питання 2. Діаграма класів — це статична структурна діаграма UML, яка зображує класи системи, їхні поля (атрибути), методи (операції) та всі типи зв'язків між ними (асоціація, спадкування, агрегація, композиція), слугуючи

ключовим кресленням для архітектури системи, яке визначає її словник та основні будівельні блоки.

9. Які зв’язки між класами ви знаєте?

Основними зв'язками між класами в UML є: Асоціація (загальний зв'язок), Узагальнення/Спадкування (відношення "є-різновидом"), Реалізація (між класом та інтерфейсом), Залежність (найслабший зв'язок, де зміна в одному класі може вплинути на інший), а також спеціальні типи асоціації — Агрегація (слабке володіння "частина-ціле") та Композиція (сильне володіння "частина-ціле").

10. Чим відрізняється композиція від агрегації?

Композиція та Агрегація обидві описують відношення "частина-ціле" (*has-a*), але відрізняються силою володіння: Композиція представляє сильне володіння, де частина не може існувати без цілого (об'єкт-ціле керує життєвим циклом частин, і вони знищуються разом із ним), тоді як Агрегація представляє слабке володіння, де частина може існувати незалежно від цілого і може бути спільно використовувана іншими об'єктами.

11. Чим відрізняється зв'язки типу агрегації від зв'язків композиції на діаграмах класів?

На діаграмах класів ці зв'язки розрізняються за типом ромба, розташованого на стороні "цілого" (контейнера): Агрегація позначається порожнім (незаповненим) ромбом, що символізує слабкий зв'язок, де об'єкти-частини не залежать від життєвого циклу об'єкта-цілого; тоді як Композиція позначається заповненим (чорним) ромбом, що символізує сильне володіння та залежність життєвого циклу.

12. Що являють собою нормальні форми баз даних?

Нормальні форми (НФ) — це набір правил і критеріїв, розроблених у теорії реляційних баз даних для оптимізації структури таблиць шляхом мінімізації надмірності даних (дублювання) та усунення аномалій (проблем при вставці, оновленні та видаленні даних), забезпечуючи, таким чином, логічну цілісність та ефективність зберігання даних.

13. Що таке фізична модель бази даних? Логічна?

Логічна модель бази даних — це абстрактне, незалежне від СУБД представлення структури даних, яке зосереджується на бізнес-вимогах і тому, що система повинна зберігати (сутності, атрибути, зв'язки), тоді як Фізична модель — це конкретне представлення структури, яке включає деталі реалізації у вибраній СУБД, такі як точні типи даних, індекси, обмеження та фізичні характеристики зберігання.

14. Який взаємозв'язок між таблицями БД та програмними класами?

Взаємозв'язок між таблицями БД та програмними класами базується на об'єктно-реляційній відповідності (ORM): як правило, Клас у програмі відповідає Таблиці в БД, Атрибут класу відповідає Стовпчику таблиці, а Об'єкт класу — Рядку (запису) таблиці. Відношення між класами (наприклад, асоціації) відображаються у БД за допомогою Зовнішніх ключів, що дозволяє програмному забезпеченню взаємодіяти з даними об'єктно-орієнтованим способом.