МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний економічний університет
Факультет комп’ютерних інформаційних технологій
Кафедра комп'ютерної інженерії

Бубенко Володимир Тарасович

Програмний модуль редагування структурованих даних заданих у вигляді графа / The software module for structured data editing in the graph form

напрям підготовки: 6.050102 - Комп’ютерна інженерія
фахове спрямування - Комп’ютерні системи та мережі
Бакалаврська робота

Виконав студент групи КСМз-41/2
В.Т. Бубенко

Науковий керівник:
Якименко І.З.

Тернопіль - 2018

РЕЗЮМЕ

Дипломний проект містить 101 сторінки пояснюючої записки, __ рисунків, ___ таблиць, 2 додатків та 4 графічні схеми формату А3.

Метою дипломної роботи є дослідження та розробка модуля редагування структурованих даних заданих у вигляді графа.

Методи досліджень базуються на теорії алгоритмів (для аналізу розроблених методів та алгоритмів), теорії графів, технологій структурного та об’єктно-орієнтованого програмування.

Зі збільшенням обсягу даних, які зберігаються локально, так і переданих по мережі, виникає проблема нестачі потужності обчислювальних ресурсів. Обчислювальні машини виявляються нездатні обробити необхідні дані в встановлені або допустимі часові інтервали. У випадках, коли тимчасові рамки відсутні, може виникнути проблема нестачі пам'яті, коли необхідна для поточних обчислень інформація просто нездатна вміститися в наданому обсязі пам'яті. Тому виникає потреба в розробці модуля редагування структурованих даних заданих у вигляді графа.

Створено графічний інтерфейс програмного забезпечення і спроектована схема бази даних з всіма необхідними інструментами і технологіями.

Ключові слова: СТРУКТУРОВАНІ ДАНІ, СЕРЕДОВИЩЕ ПРОЕКТУВАННЯ, БАЗА ДАНИХ, АЛГОРИТМ, АТРИБУТИ КЛАСУ.

RESUME

Diploma project contains 76 pages explanatory notes, 9 figures, 9 tables, 2 apps and 2 graphic scheme A3.

The purpose of the thesis is to research and develop a module for editing structured data given in the form of a graph.

Research methods are based on the theory of algorithms (for the analysis of developed methods and algorithms), the theory of graphs, technologies of structural and object-oriented programming.

With the increase in the amount of data stored locally and transmitted over the network, there is a problem of lack of power computing resources. Computing machines are unable to process the necessary data in established or allowable time intervals. In cases where there is no timeframe, there may be a problem with the lack of memory when the information needed for current computing is simply unable to fit in the amount of memory provided. Therefore, there is a need to develop a module for editing structured data given in the form of a graph.

A graphical software interface is created and a database schema with all the necessary tools and technologies is designed.

Keywords: STRUCTURED DATA, DESIGN ENVIRONMENT, DATA BASE, ALGORITHM, CLASS ATTRIBUTE.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ІС – Інформаційна система

СУБД - Система управління базами даних

ACID - Atomicity, Consistency, Isolation, Durability (Атомарність, Узгодженість, Ізольованість, Довговічність)

API - Application Programming Interface (Програмний інтерфейс програми)

AWT - Abstract Window Toolkit (Абстрактний віконний інструмент)

DAO - Data Access Object (Об'єкт доступу до даних)

DOM - Document Object Model (Об'єктна модель документа)

DTD - Document Type Definition (Визначення типу документа)

FK - Foreign Key (Зовнішній ключ)

GC - Garbage Collection (Прибирання сміття)

GUI - Graphical user interface (Графічний інтерфейс користувача)

IDE - Integrated Development Environment (Інтегроване середовище розробки)

JAR - Java ARchive (Java-архів)

JDBC - Java DataBase Connectivity (З'єднання з базами даних на Java)

JDK - Java Development Kit (Набір розробника Java)

JRE - Java Runtime Environment (Середовище виконання для Java)

JVM - Java Virtual Machine (Віртуальна машина Java)

PK - Primary Key (Первинний ключ)

PL/SQL - Procedural Language / Structured Query Language (Мова програмування, процедурне розширення мови SQL)

SAX - Simple API for XML (Просте API для XML)

SQL - Structured Query Language (Мова структурованих запитів)

SWT - Standard Window Toolkit (Стандартний віконний інструмент)

XML - eXtensible Markup Language (Розширювана мова розмітки)

ВСТУП

Зі збільшенням обсягу даних, які зберігаються локально, так і переданих по мережі, виникає проблема нестачі потужності обчислювальних ресурсів.

Обчислювальні машини виявляються нездатні обробити необхідні дані в встановлені або допустимі часові інтервали. У випадках, коли тимчасові рамки відсутні, може виникнути проблема нестачі пам'яті, коли необхідна для поточних обчислень інформація просто нездатна вміститися в наданому обсязі пам'яті. У зв'язку з цим виникає таке поняття, як великі дані.

Великі дані (англ. Big data) - сукупність методів, підходів і засобів обробки структурованих і неструктурованих даних великих обсягів і істотного різноманіття, і подальшого їх інтерпретації з отриманням ефективних результатів. Під «великими обсягами» розуміються такі обсяги даних, обробка яких стандартними і апаратними засобами представляється вкрай складною [2].

Це ті обсяги, які не можуть бути розміщені і оброблені на окремій обчислювальній машині - терабайти або петабайти даних.

Виникає питання обробки даних таких обсягів. Способів вирішення такого питання кілька. В першу чергу, можна нарощувати обчислювальні потужності окремих машин. Але таке зростання має межу, притому цілком певний - кожна віртуальна машина, в залежності від своєї архітектури, має ті чи інші обмеження на встановлюваний об'єм пам'яті, число процесорів і т.д., отже, такий підхід дозволить вирішити лише приватні проблеми, що виникають в конкретних завданнях. Інший підхід - оптимізація алгоритмів і програмних продуктів. І якщо попередній був орієнтований на «грубу силу», то цей спосіб має на увазі швидше відточування існуючих інструментів і методів, упор на якість, а не кількість. Недолік такого підходу - його швидкість.

Оптимізація, віднімаючи чимало часу, може дати результат лише в соті частки відсотка або взагалі не мати практичного сенсу.

Найбільш правильним рішенням проблеми великих даних є розпаралелювання обчислень. В цьому випадку формуються обчислювальні мережі, що складаються з обчислювальних машин, що працюють одночасно один з одним і виконують операції паралельно. Даний підхід не обмежує обсяг пам'яті і число процесорів, тим самим вирішуючи проблему нестачі ресурсів. Однак, вимагає розробки відповідного програмного забезпечення, орієнтованого на паралельні обчислення, а також адаптації існуючих алгоритмів.

У багатьох випадках дані зручно представляти у вигляді графів, де вершинами будуть якісь сутності предметної області, а ребрами - зв'язки між цими сутностями. Найбільш яскравий приклад - соціальні мережі. І тривіальний граф - з вершинами, що представляють собою облікові записи певних людей. Ребра в цьому випадку будуть означати, скажімо, знайомство одну людину з іншим. Зручність подання даних в такому вигляді пояснюється тим, що графи досить добре вивчені, і існує безліч алгоритмів їх обробки - обходу, пошуку шляхів, поділу, кластеризації. Крім того, частина цих алгоритмів добре піддається розпаралеленню для виконання в розподіленому середовищі. Це дає можливості для обробки великих даних, представлених у вигляді графів, у реальному часі.

1 АНАЛІЗ ПІДХОДІВ РЕДАГУВАННЯ СТРУКТУРОВАНИХ ДАНИХ ЗАДАНИХ У ВИГЛЯДІ ГРАФА

1.1 Подання структурованих даних в реляційних базах даних

Існує ряд завдань, що вимагають подання структурованих даних в реляційних базах даних (РБД). Перш за все це пов'язано з інформацією, представленою у вигляді графових або мережевих структурах. В даний час відсутні рішення за оптимальними схемами зберігання настільки складних структурованих даних. Існують два основні методи представлення графів (в пам'яті) ЕОМ: матричний, (тобто представлення масивами) і зв'язковими нелінійними списками. Вибирається метод представлення в залежності від природи даних і операцій, які виконуються над ними. Якщо завдання вимагає великого числа включень і виключень вузлів, то використовується представлення графа зв'язковими списками; в іншому випадку - матричне подання. Зберігання графів в РБД ускладнюється великою кількістю посилань на елементи графа, невідомим заздалегідь. Основна складність полягає в ефективному зберіганні і підтримці цілісності цих посилань. На сьогоднішній день виявлено два способи зберігання деревовидних структур:

1) у вигляді матриці суміжності з використанням допоміжної таблиці;

2) за методом вкладених множин.

Як виявилося, дані схеми не застосовні прямо до довільного графу, так як в них використовуються обмеження ієрархічної структури, зокрема не передбачається і не враховується можливість ненапрямлених зв'язків, петель. Розглянемо можливі способи зберігання графів в РБД. Використовуючи матрицю суміжності для зберігання графів в РСУБД, запишемо її не в класичному вигляді, а представимо зв'язковим списком вершин. Ключем таких реляційних відносини є поєднання ідентифікаторів вихідної вершини (одного з батьків) і входить (нащадка). Для ненаправленного графа, який передбачає можливе повторення цих сполучень, введемо окреме ідентифікаційне поле (рисунок 1.1). В цьому випадку

ID_Out і ID_In, що представляють позначення вхідних та вихідних вершин по матриці, відповідають ID_el вузлів графа в основній таблиці з даними. Подання графа списками ребер і вершин є одним з найбільш поширених при аналітичній роботі з графами і широко використовується в програмуванні, проте у випадку з РБД фактично відповідає розглянутому. Випадок зберігання графа з використанням зв'язкового (облікового) представлення зводиться до зберігання структури і нагадує нелінійний розгалужений список. Отримана структура має два типи елементів: (елемент графа-посилання) і (посилання-посилання). Проте, отримана структура не є нелінійним розгалуженим списком, так як в ній можуть існувати петлі і назад спрямовані зв'язки (рисунок 1.2). Зв'язки B → D і D → C не відповідають поданням у вигляді дерева і не відповідають теорії нелінійних розгалужених списків. Реляційна схема зберігання наведена на рисунку 1.3. Основні дані (елементи списку) зберігаються в таблиці «Граф». Пов'язана таблиця «Граф: елементи графа» використовується спільно з основною для окремого зберігання елементів, відсутніх в разі поєднань посилання-посилання. Поле «Type» має бінарне значення: 0 - посилання-елемент, 1 - посилання-посилання. Нарешті, ще одним варіантом зберігання є представлення графа у вигляді лісу зв'язних дерев (рисунок 1.4). У разі такого представлення раціонально використовувати схему зберігання дерев, удосконаливши її для подання кількох пов'язаних ієрархій. Граф: матриця суміжності PK ID_link ID_Out ID_In Граф: елементи графа PK ID_el Data (рисунок 1.1). Подання графа з використанням матриці суміжності A D C

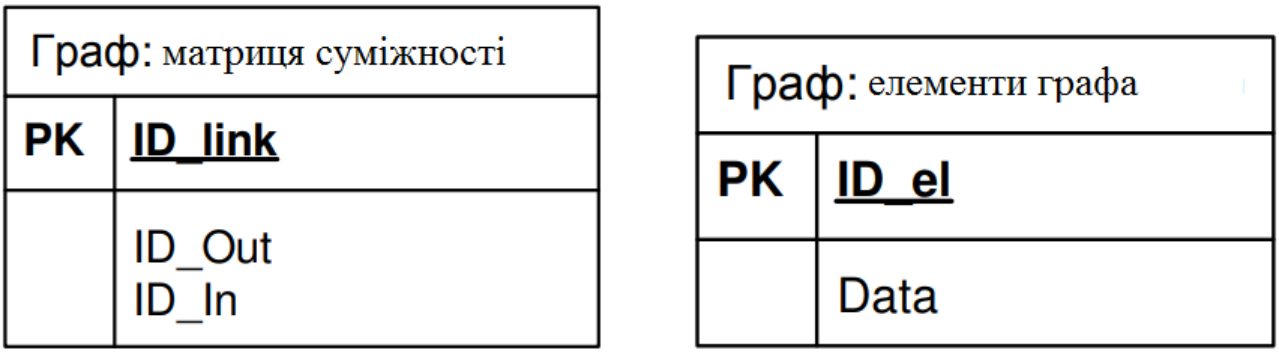

Рисунок 1.1 – Представлення графа з використанням матриці суміжності

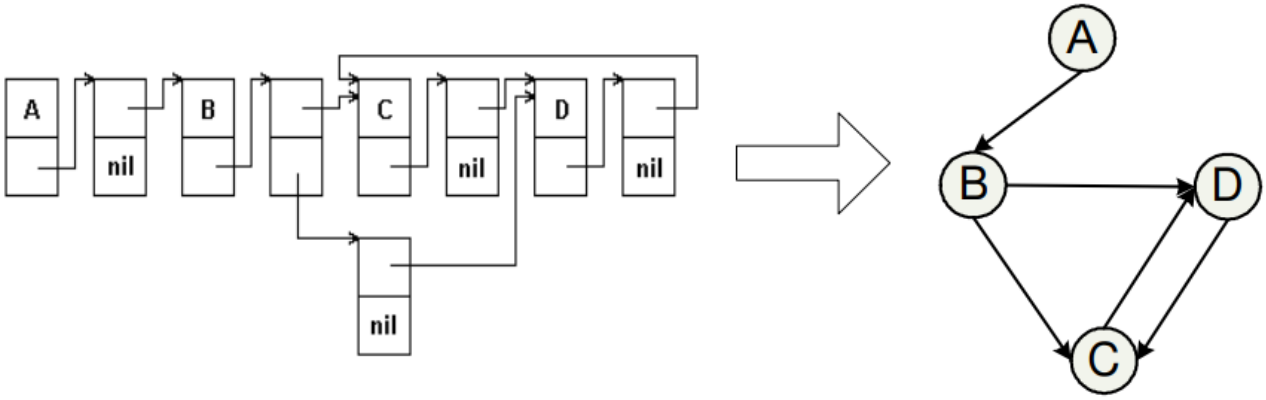

Рисунок 1.2 – Представлення графа списком

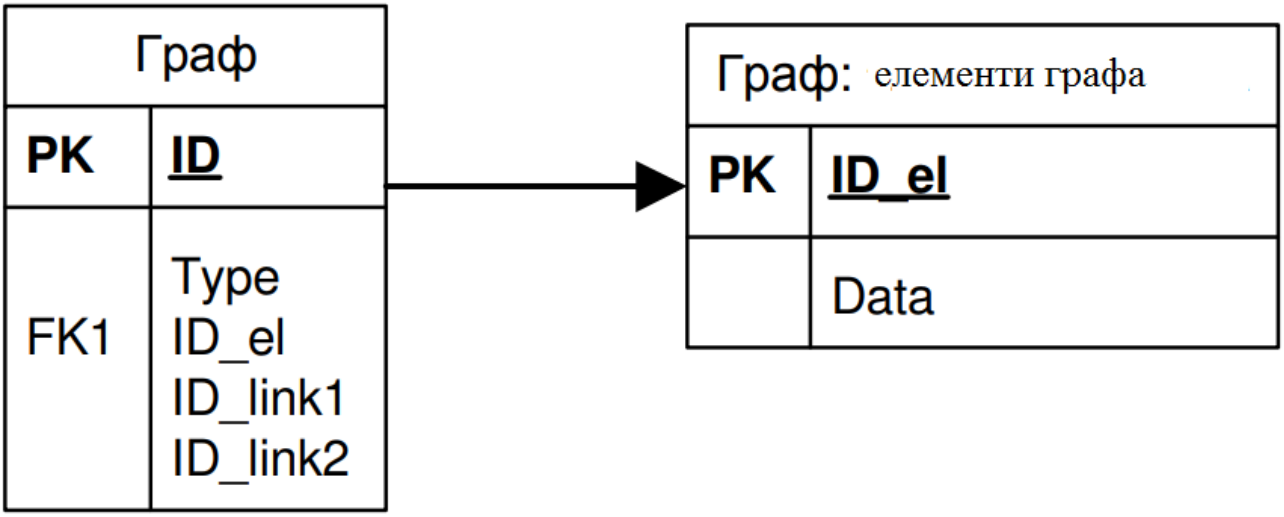

Рисунок 1.3 – Представлення графа списком в РСУБД

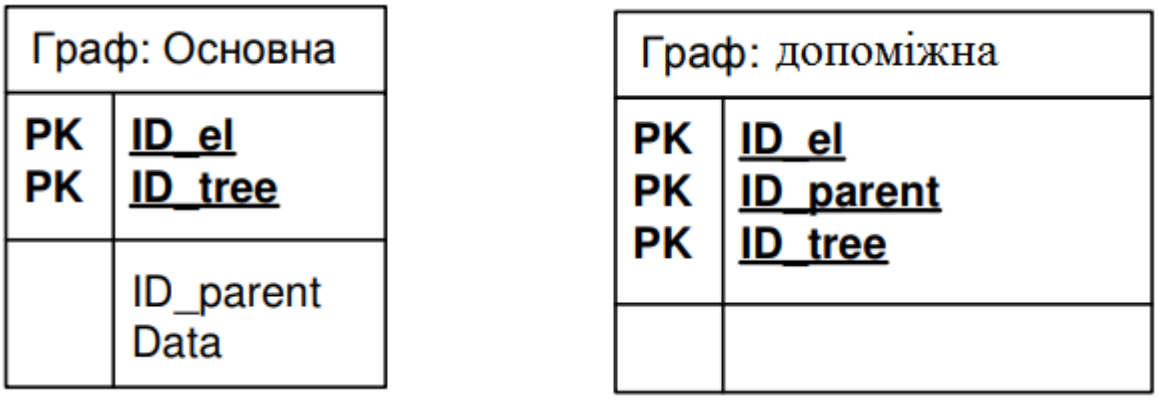

Рисунок 1.4 – Представлення по методу зв’язних ієрархій: з допомогою матриці суміжності

Автором виявлені дві найбільш раціональні схеми зберігання ієрархічних структур, що мають різну ефективність, в залежності від типу розв'язуваної задачі. В одному випадку для зберігання дерева використовуються дві таблиці: основна і допоміжна. В основній таблиці зберігається тільки безпосередній предок вузла, а в допоміжній кожному вузлу співвіднесені всі його предки по напрямку до вершини. Для зберігання графа розширимо цю схему на ідентифікатор дерева, що використовується спільно з номером вузла, до якого він належить. Таким чином, застосовувана для зберігання графів схема буде мати вигляд, представлений на рисунку 1.4. В іншому випадку, згідно з методом вкладених множин (МВМ), кожному елементу ієрархії в залежності від його місця присвоюються два числових значення: ліве і праве. Кожне дерево, що входить в граф, отримує ці значення незалежно від структури самого графа (рисунок 1.5).

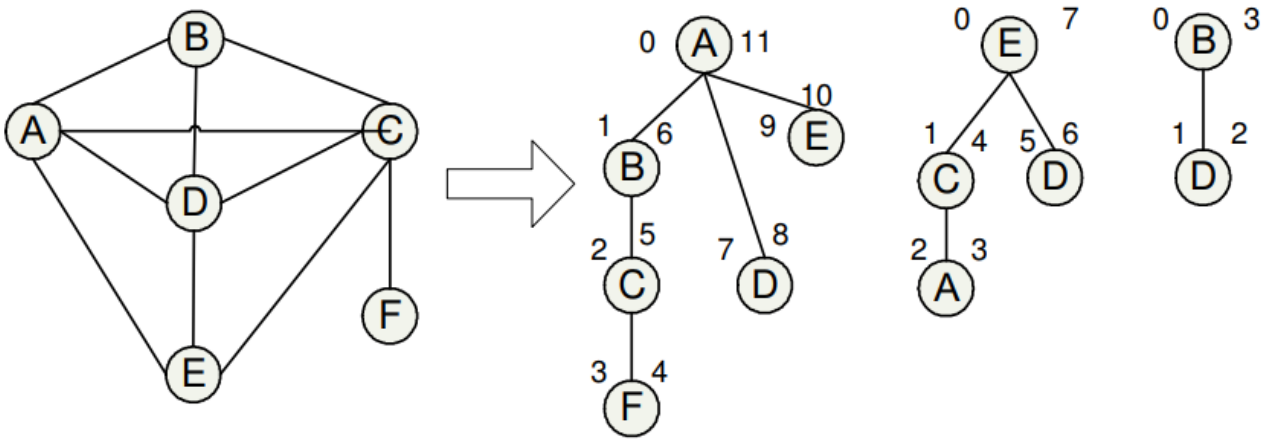

Рисунок 1.5 - Представлення в вигляді зв’язних дерев МВМ з окремою нумерацією

В цьому випадку модернізація схеми зберігання ієрархій зводиться до додавання нового поля - ідентифікатора дерева (рисунок 1.6). Логічно використовувати для всіх дерев графа єдину, наскрізну нумерацію, яка враховує всі дерева та його складові.

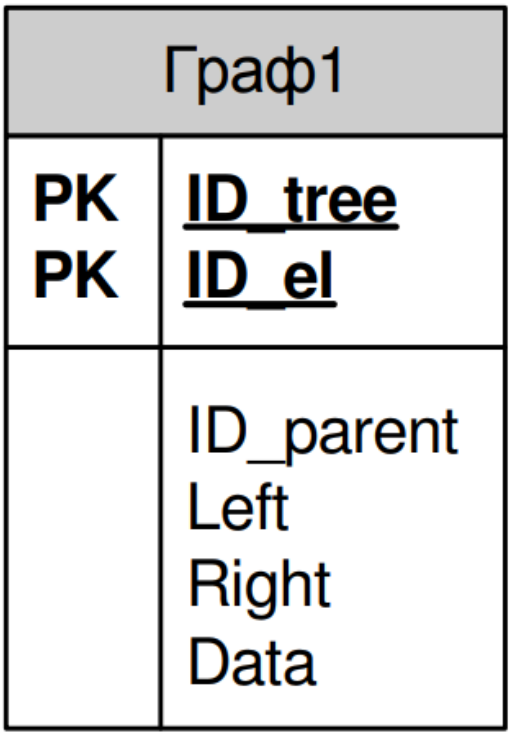

Рисунок 1.6 - Представлення у вигляді зв’язних дерев для РБД

Представлення матрицею суміжності і списками ребер і вершин раціонально об'єднати воєдино, оскільки для цих представлень використовується одна і та ж схема зберігання. Назвемо цей об'єднаний спосіб «зберіганням за допомогою списків ребер і вершин», що найбільш точно відповідає суті схеми зберігання. Виключимо з остаточного розгляду чітке (списочне) представлення графа через виявлену особливість складності роботи з ним.

1.2. Порівняння методів представлення графів

Таким чином, остаточний список представлень графа виглядає наступним чином:

1) подання графа списками ребер і вершин;

2) представлення графів у вигляді лісу пов'язаних дерев на підставі методу матриці суміжності з допоміжною таблицею;

3) представлення графів у вигляді лісу пов'язаних дерев на підставі методу вкладених множин з незалежної нумерацією;

4) подання графів у вигляді лісу зв'язних дерев на підставі методу вкладених множин з наскрізною нумерацією.

Для вирішення завдань, пов'язаних з графами, потрібно зробити над ними певні дії. Автором проаналізовані дані схеми зберігання щодо найбільш поширених задач над графами, визначені рішення кожної з них для кожної схеми зберігання. Критерієм порівняння вибрано число звернень до диску (дискових операцій), що визначає, відповідно, і швидкість роботи програми. Окремо в кожному випадку вказується, чи проводиться ця операція з використанням аналітичних методів і/або в рамках циклічного запиту. Вихідні дані: E1 - число елементів в графі; E2 - число дуг в графі; M1 - число рядків таблиці, що зберігає елементи графа (для всіх представлень M1 = E1); M2 - число рядків таблиці (якщо вони є), що зберігає дуги графа (для подання у вигляді списку ребер і списку вершин M2 = E2; для подання у вигляді лісу дерев по методу матриці суміжності M2> E2> E1; .в представленнях по методу вкладених множин допоміжна таблиця відсутня); N1 - число стовпців таблиці, яка містить елементи графа; N2 - число стовпців таблиці, що зберігає дуги графа за методом списку вершин і ребер; N2 +1 - число стовпців таблиці, що зберігає дуги графа в поданні у вигляді лісу дерев з використанням методу матриці суміжності; N1 + 1 - число стовпців в таблиці в поданні у вигляді лісу дерев з використанням методу матриці суміжності; N1 + 3 - число стовпців в таблиці в поданні у вигляді лісу дерев за методом вкладених множин.

Операції пошуку шляху, знаходження циклів і виділення підграфа не розглядалися в зв'язку з неможливістю визначити для них точну кількість звернень до диску, але розглядалися що входять до них операції виділення вершин з тільки вхідними та вихідними дугами. Результати проведеного порівняння представлені у зведеній таблиці 1.1, де жирним курсивним шрифтом виділені найкращі варіанти зберігання на кожному етапі.

Таблиця 1.1 – Порівняння методів представлення графа

Метод	Списки ребер і вершин	Ліс дерев і метод матриці суміжності	Ліс дерев і	Ліс дерев і метод вкладених множин (сквозна нумерація)
Операція додавання вершини	$\boldsymbol{U_1=1}$	$\boldsymbol{U_2=1}$	$U_3= M_3$	$U_3= M_1$
Операція видалення вершини	$\boldsymbol{U_1=1}$	$\boldsymbol{U_2=1}$	$U_3= M_3$	$U_3= M_1$
Операція додавання дуги	$\boldsymbol{U_1=1}$	$U_2= M_1+ 1$ або $U_2= M_1+2$	$U_3= M_1+ M_3$	$U_4= 2M_1$
Операція видалення вершини	$\boldsymbol{U_1=1}$	$U_2= M_1+2$	$U_3= M_1+ M_3+2$	$U_4= 2M_1+2$
Визначення суміжності вершин	$U_1=M_2$	$\boldsymbol{U_2= M_1}$	$\boldsymbol{U_3= M_1}$	$\boldsymbol{U_4= M_1}$
Визначення інцедентності вузла до ребра	$U_1= M_2$	**0**	**0**	**0**
Виділення вершин тільки з вхідними дугами	$U_1= M_1+ M_2$	$\boldsymbol{U_2= M_1}$	$\boldsymbol{U_3= M_1}$	$\boldsymbol{U_4= M_1}$
Виділення вершин тільки з вихідними дугами	$U_1= M_1+ M_2$	$\boldsymbol{U_2= M_1}$	$\boldsymbol{U_3= M_1}$	$\boldsymbol{U_4= M_1}$

Варто зауважити, що представлення у вигляді списку ребер і вершин дозволяє найбільш зручним чином зберігати інформацію по навантажених графах, ставлячи її в пряму відповідність дуг. Проведений аналіз виявив корисність з точки зору складання запитів методу подання до вигляді лісу дерев за методикою матриці суміжності і найкращу поведінку в задачі пошуку шляху в графі методу подання до вигляді лісу дерев з використанням методу вкладених множин та

наскрізної нумерації. Результати показують необхідність розвитку методів представлення графів і оптимізації завдань по роботі з ними в РСУБД, так як жодна з розглянутих схем зберігання не може бути визнана оптимальною.

1.3 Аналіз аналогів для редагування структурованих даних

Всі організації з року в рік генерують все більшу кількість даних. Ці дані стають більш складними, з'являються різні складні відносини між ними. Такі структури даних добре представляються у вигляді графа, так як вершини графа це самі дані, а дуги - це відносини між ними.

Організації у яких з'являється гостра необхідність в зборі та редагуванні структурованих даних часто створюють своє власне програмне забезпечення (ПЗ). Але далеко не всі мають можливість створити власне ПЗ для редагування даних з їх власною структурою. Таким чином доводиться редагувати свої структуровані дані вручну використовуючи звичайні текстові редактори такі як notepad ++ або Geany. Такий підхід багато в чому не зручний так як на редактора цих даних покладається відповідальність за цілісність і правильність даних з якими він працює. Людині зазвичай складно орієнтуватися у великих кількостях даних зі складною структурою. Це веде до неминучих помилок, які можуть трапиться тому, що людині просто складно стежити за усією структурою в цілому. Розширювана мова розмітки XML добре підходить для вирішення проблем, пов'язаних зі зберіганням даних зі складною структурою.

Щоб вирішити проблему з якою стикаються безліч організацій, але у яких немає можливості створити власне програмне забезпечення, було вирішено створити програмне забезпечення, що дозволяє редагувати структуровані дані. Для зручності користувачів у програмі буде реалізований зручний і зрозумілий графічний інтерфейс.

Як було відзначено в попередньому пункті багато організацій, які мають

необхідність редагувати структурованих дані, часто вирішують цю проблему за допомогою свого власного програмного забезпечення. Але для інших організацій ця проблема є досить вагомою, так як на ринку немає універсальних програмних продуктів для її вирішення. Для редагування структурованих даних які зберігаються в форматі XML, можна використовувати будь-які текстові редактори, наприклад, такі як notepad ++ або Geany. У такого підходу є деякі мінуси:

– відповідальність за дотримання типів даних (рядок, число, логічне значення) у кожного об'єкта лягає на редактора даних;

– відповідальність за дотримання цілісності структури даних знову ж лягає на редактора даних;

– користувач повинен розуміти принцип побудови xml документа.

У переваги даного підходу редагування даних можна віднести те, що не потрібно розробляти власне програмне забезпечення.

1.3.1 Переваги роботи при використанні спеціалізованого програмного забезпечення.

Щоб краще показати переваги роботи в спеціальному програмному забезпеченні над роботою в звичайних текстових редакторах, нижче приведена порівняльна таблиця 1.2 двох цих підходів.

Таблиця 1.2 – Порівняльна таблиця підходів редагування даних

	Звичайні текстові редактори	Спеціальне програмне забезпечення
1	2	3
Перевірка на помилки в xml розмітці	-	+

Продовження таблиці 1.2

1	2	3
Перевірка на помилки в типах даних	-	+
Перевірка на помилки в цілісності структури даних	-	+
Необхідність розуміння XML розмітки	+	-

З таблиці 1.2 видно, що звичайні текстові редактори мають безліч недоліків, тому доцільно розробити власне програмне забезпечення, яке не матиме цих недоліків.

1.4 Постановка задач проектування

Розроблюване програмне забезпечення повинно дозволити кінцевому користувачеві редагувати (додавати змінні, видаляти компоненти) структуровані дані в найбільш зручному вигляді і не турбуватися про випадкові помилки. Програмне забезпечення повинно мати графічний інтерфейс, який дасть можливість навіть недосвідченому користувачу працювати в програмі. Структура даних представлена у вигляді графа вершинами якого є компоненти інформаційної системи, а дугами - відносини між ними.

Програмний інтерфейс повинен бути досить простим і інтуїтивно зрозумілим. Для цього необхідно розв’язати низку взаємнопов’язаних задач, серед яких виділяються:

—проаналізувати існуючі аналоги на ринку;

—провести проектування схем реляційної бази даних яка дозволяє зберігати дані у вигляді графа;

—провести проектування структури графічного інтерфейсу, що дозволяє редагувати структуровані дані;

—здійснити вибір програмних засобів розробки і мови програмування для проектування, а саме:

1) вибір мови програмування;
2) вибір середовища розробки;
3) вибір системи управління базами даних;

—провести реалізацію програмного забезпечення для редагування структурованих даних за допомогою обраних засобів і технологій, а саме:

1) добавлення компонентів;
2) редагування компонентів;
3) видалення компонентів;

—провести тестування розробленого програмного забеспечення.

Завдання обробки інформації, структура якої визначена в повному обсязі, виникають сьогодні в самих різних областях. Прикладом можуть служити корпоративні каталоги даних, які складають основу більшості сучасних інформаційних систем, інтегруючи різні бізнес-додатки в єдиний інформаційний простір і забезпечуючи збереження і доступність інформації для різних груп користувачів.

Одним прикладом можуть служити електронні бібліотеки та електронні колекції. Призначення систем такого роду полягає в тому, щоб зберігати різнорідні інформаційні ресурси. При цьому завдання опису структури даних може вирішуватися як на системному рівні, так і на рівні кінцевого користувача. Гнучкість опису різнорідних даних визначає ефективність зберігання та

вилучення інформації в електронній бібліотеці. Загальним для всіх таких систем є наявність каталогу даних або каталогу об'єктів. Зазвичай основними частинами каталогу є система підтримки метаданих та система зберігання та вилучення даних.

Системи підтримки метаданих забезпечують адекватне моделювання і опис інформації як на рівні опису окремих сутностей (об'єктів), так і на рівні опису структури всього каталогу. Метаопису задають специфіку занесення, пошуку та вилучення даних. Відомо безліч різноманітних підходів до побудови систем управління даними з різним ступенем структурованості [3, 6, 8, 11, 18–22]. Подібні системи реалізуються на основі різних технологій. Найпопулярнішою платформою для обробки даних сьогодні залишаються реляційні СУБД [1, 2].

Що ми розуміємо під квазіструктуровані (напівструктуровані) дані? Зазвичай мова йде про інформацію, в якій можна виділити ту чи іншу структуру, однак структура ця заздалегідь цілком або частково невідома, або може змінюватися з плином часу. Можливо кілька варіантів.

Дані, структура яких навіть апріорно невідома. У цьому випадку формування структури каталогу даних і його заповнення відбувається одночасно в міру надходження інформації. Прикладом може служити набір, довільним чином пов'язаних документів в різних форматах з довільними описами кожного документа.

Структура даних відома частково. Відомі основні описи базових сутностей або класів системи, разом з тим є необхідність зберігання та обробки заздалегідь невизначеної додаткової інформації. Так, інформація про покупців або продавців, може супроводжуватися набором приміток про кожну операцію, контактними телефонами, додатковим умовами і т.д.

Структура даних відома і чітко визначена, але може змінюватися з плином часу. Найпростішим прикладом може служити набір реквізитів рахунку-фактури або реквізитів організації, які змінюються зі зміною законодавства.

Подібна досить умовна класифікація побудована за принципом «посилення» чіткості структури інформації, разом з тим, вона відображає характерні завдання обробки напівструктурованих даних.

Дослідження систем обробки слабоструктурованих даних і мов запитів до них ведуться досить давно. Як приклад можна назвати систему Lore з цих слів Lorel [6, 7] або систему інтеграції даних з гетерогенних джерел YAT [9, 10, 11]. Закладені в Lore ідеї знайшли продовження в багатьох системах обробки різнорідних даних. Розроблено ряд мов запитів до слабкоструктурованих даних: XML-QL [12, 14], XQL [15], Quilt [16, 17], X-Query [13]. Докладні огляди цих технологій можна знайти в [4, 5].

Реляційні системи ефективно обробляють дані, описані в термінах відносин і їх зв'язків. При цьому метадані у вигляді описів відносин, атрибутів відносин, первинних і вторинних ключів, підтримуються на системному рівні СУБД. У «класичному» варіанті реляційної СУБД підтримка слабоструктурованих даних не передбачена: в рамках реляційної алгебри описати їх неможливо. Для їх моделювання необхідний прикладний рівень - надбудови над СУБД, які дозволяли б вирішити завдання моделювання і обробки різнорідної інформації; в свою чергу ці надбудови повинні використовувати тільки стандартні засоби реляційних систем.

2 МОДЕЛЮВАННЯ СИСТЕМИ РЕДАГУВАННЯ СТРУКТУРОВАНИХ ДАНИХ ЗАДАНИХ У ВИГЛЯДІ ГРАФА

2.1 Система моделювання Object Exchange Model

2.1.1 Модель Object Exchange Model

В рамках проекту Lore використана модель представлення даних OEM (Object Exchange Model), розроблена в ході досліджень за проектом Tsimmis [23, 24]. Закладені в моделі OEM ідеї з тими чи іншими модифікаціями використовуються практично у всіх системах управління слабоструктурованими даними. Інформація в OEM-структурі представляється у вигляді графа з іменованими ребрами, а вузли відповідають об'єктам. Для визначеності будемо називати їх node-об'єктами або *n*-об'єктами. У найпростішому варіанті вони можуть бути атомарними або контейнерами. Атомарні *n*-об'єкти мають лише вхідні зв'язку і значення певного типу, але не мають залежних об'єктів. У контейнерів немає значень, але є залежні об'єкти (вихідні зв'язки). Кожен *n*-об'єкт має унікальний ідентифікатор. На зв'язку між ними, в загальному випадку, не накладається ніяких обмежень. На рисунку 2.1 наведено фрагмент OEM-бази даних.

Метадані в OEM-структурі визначаються даними і формуються в міру виникнення самих даних. Семантична інформація зберігається в самій структурі бази даних. Універсальність підходу дозволяє легко моделювати інформацію довільної структури. Візьмемо цю модель в якості відправної точки і побудуємо на її основі систему моделювання та зберігання напівструктурованих даних.

Найпростішу OEM-структуру реалізувати у вигляді двох основних таблиць - таблиці *n*-об'єктів і таблиці зв'язків і набору допоміжних таблиць значень (рисунок 2.1а).

Таблиця об'єктів tbl_object містить список *n*-об'єктів - вершин графів OEM-структури. У найпростішому варіанті кожен з них має унікальний ідентифікатор і тип, який може приймати два значення: контейнер і атомарний. Таблиця зв'язків

tbl_links описує зв'язки між *n*-об'єктами - ребра графів. Ім'я зв'язку і набір ідентифікаторів батьківського і залежного *n*-об'єктів утворюють первинний ключ таблиці зв'язків. Значення *n*-об'єктів зберігаються в таблицях значень, в кожній таблиці - значення свого типу. Вибір таблиці, в якій зберігається значення, визначається атрибутом obj_val_type таблиці об'єктів. Часто значення всіх n-об'єктів призводять до строковому типу. Це дозволяє зберігати їх безпосередньо в таблиці tbl_object, для чого в неї необхідно додати атрибут obj_val, зазвичай має тип varchar. Приведення типів в цьому випадку забезпечується прикладними завданнями і грунтується на атрибуті tbl_object.obj_val_type (рисунок 2.1 б).

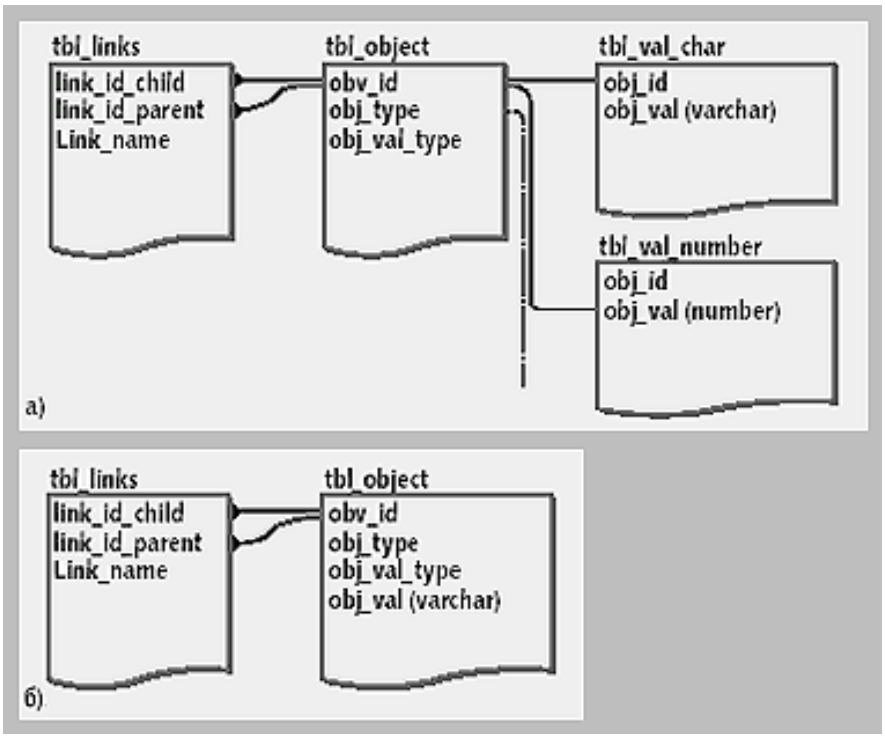

Рисунок 2.1 – Схема даних для OEM-структури: а) значення об'єктів кожного типу в окремих таблицях; б) всі значення об'єктів наведені до символьного типу

За допомогою такого підходу можна моделювати дані будь-якої структури і складності. Разом з тим, відсутність явної метаінформації ускладнює введення, вилучення та обробку даних, що мають подібний семантичний опис. Те ж саме можна сказати і про дані з «розмитою» структурою, коли опис однієї частини інформації суворо детерміновані, а інші - заздалегідь не визначені. Додаткова

метаінформація необхідна користувачам і додаткам для уявлення про структурах даних, які можуть зберігатися в базі даних. Вона необхідна для визначення набору можливих / обов'язкових n-об'єктів, їх значень, зв'язків між ними.

В якості найпростішого прикладу може служити інформація про клієнтів, де завжди можна виділити загальні атрибути, частина з яких повинна бути обов'язково оброблена. Покупці, будучи в свою чергу клієнтами, також мають загальні (тільки для покупців) атрибути. Для коректного введення даних про клієнтів, покупців, продавців необхідно знати набір атрибутів, які необхідно або бажано заповнити. При отриманні даних також необхідна інформація про набір атрибутів. Разом з тим, ці правила не повинні накладати обмеження на введення довільних атрибутів з довільними значеннями, оскільки кожен клієнт, в тому числі і покупець, може мати індивідуальну інформацію.

Тому запропоновану схему необхідно модифікувати з метою введення додаткового метаописання даних, не втрачаючи при цьому гнучкості моделювання неструктурованою інформації.

Стосовно до OEM-структури додаткова метаінформація повинна описувати зумовлені семантичні зв'язки між *n*-об'єктами. Варіантів організації таких зв'язків досить багато. Розглянемо один з них, пов'язаний з перетворенням OEM-бази даних в своєрідний каталог слабоструктурованих об'єктів. Нескладно бачити, що в термінах «об'єктного» моделювання *n*-об'єкти - контейнери відповідають екземплярам будь-якого класу, а атомарні *n*-об'єкти відповідають атрибутам примірників класу. Додамо в схему даних можливості формального визначення класів; фактично йдеться про створення каталогу класів.

Створимо в схемі даних дві таблиці описів класів і їх атрибутів відповідно (рисунок 2.2). Таблиця описів класів tbl_classdef складається всього з трьох атрибутів і визначає ієрархію класів, необхідну для моделювання даних. Атрибут tbl_classdef.cdef_id є ідентифікатором класу. Tbl_classdef.cdef_id_parent визначає відносини спадкування в ієрархії класів. Таблиця tbl_attdef визначає атрибути класів.

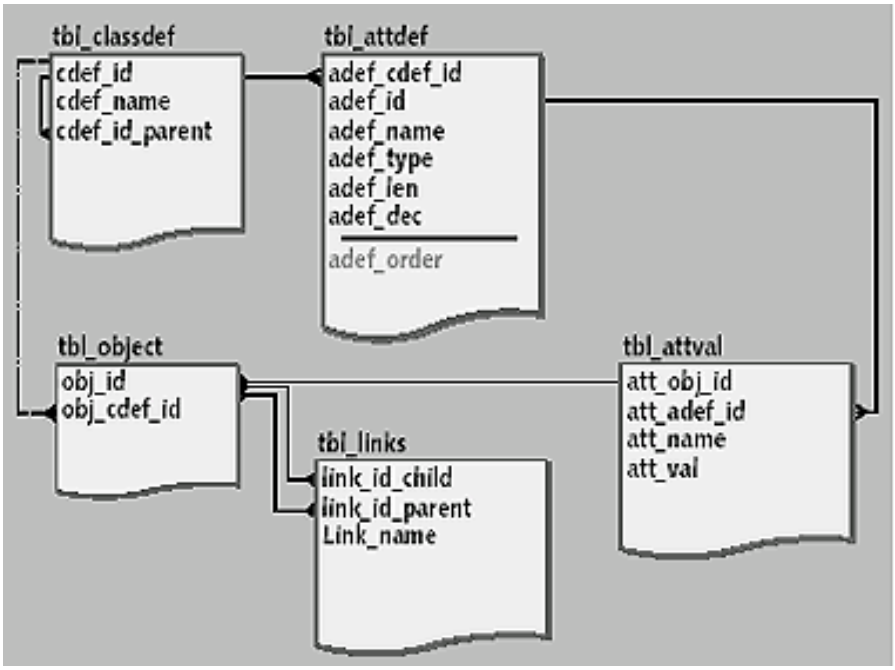

Рисунок 2.2 – Розширена реляційна схема для обробки квазіструктурірованих даних

Можливо два варіанти заповнення таблиці tbl_attdef. Припустимо, що клас C_n є батьківським для класу C_{n1}. У першому випадку для визначення класу C_{n1} таблиця tbl_attdef містить тільки визначення атрибутів, що відрізняють його від визначення батьківського класу C_n. Щоб отримати повний список атрибутів класу C_{n1} необхідно отримати список всіх атрибутів для всіх класів вгору по ієрархії, починаючи з визначення C_{n1}. Таким чином, всі зміни в термінах батьківських класів автоматично стають доступні для класів спадкоємців. Ідентифікатор атрибута adef_id буде однаковим для всіх класів в ієрархії. Кожному атрибуту відповідає одна запис в таблиці tbl_attdef (рисунок 2.3).

tbl_classdef

CDEF_ID	CDEF_ID_PARENT
A	
B	
C	A
D	C

tbl_attdef

ADEF_ID	ADEF_CDEF_ID	ADEF_NAME
01	A	Att_A1
02	A	Att_A2
03	B	Att_B1
04	B	Att_B2
05	C	Att_C1
06	D	Att_D1
07	D	Att_D2

Рисунок 2.3 – Перший варіант опису атрибутів класу

Клас A є батьківським для класу C, від якого в свою чергу проведений клас D (рисунок 4). Відповідно класу A належать атрибути з ідентифікаторами 01, 02; класу C - 01, 02, 05; класу D - 01, 02, 05, 06, 07.

Другий варіант передбачає, що таблиця tbl_attdef містить повний опис атрибутів для кожного класу. Необхідність такого підходу може виникнути при наявності у атрибутів класів індивідуальних ознак, які можуть відрізнятися у батьківських і породжених класів. Як приклад можна привести порядок проходження атрибутів у визначенні класу, що важливо, наприклад, при описі XML-документів. Така необхідність може також виникнути при перевизначенні атрибутів, збільшенні довжини строкових атрибутів для породжених класів і т.д. Цілісність ієрархії класів в цьому випадку необхідно підтримувати штучно, за допомогою написання відповідних тригерів для таблиці tbl_attdef. Опис атрибутів, визначених для C_n, повинно повторюватися для C_{n1}. Разом з тим, є первинним ключем ідентифікатор adef_id повинен змінюватися для одного і того ж атрибута при переміщенні по ієрархії класів (для C_n і C_{n1} він повинен бути різним). Таким чином, необхідно також забезпечити ідентифікацію для успадкованих атрибутів уздовж ієрархії класів. Це можна зробити, додавши в таблицю tbl_attdef додатковий атрибут adef_id_parent, значення якого заповнюється при першому визначенні атрибута (в цьому випадку adef_id_parent = adef_id) і залишається незмінним для відповідних атрибутів класів спадкоємців (рисунок 5). Таблиця tbl_attdef містить опис всіх атрибутів для кожного класу.

Далі розглянемо схему зберігання даних (рисунок 3). Винесемо значення атрибутів в окрему таблицю, розвантаживши тим самим таблицю об'єктів. Кожен запис tbl_object тепер відповідає об'єкту каталогу і контейнеру в первісної OEM-схемі. Далі для визначеності об'єкти каталогу будемо назвати k-об'єктами. У таблиці об'єктів міститься посилання на визначення класу, екземпляр якого описує запис (атрибут obj_cdef_id). Таблиця зв'язків описує тільки відносини між *k*-об'єктами, оскільки зв'язку між *k*-об'єктами і атрибутами описані тепер в таблицях визначення класів. Значення атрибутів містяться в таблиці tbl_attval,

причому її записи відповідають тільки тим атрибутам *k*-об'єктів, для яких визначені значення.

Яким же чином досягається гнучкість опису різнорідних даних в, здавалося б, жорстко певній структурі, з явними описами визначень класів? Для цього досить дозволити визначати k-об'єкти без вказівки класу, тобто не визначаючи що значення tbl_obect.obj_cdef_id. Аналогічно можна вчинити і з атрибутами, дозволивши додавати довільні записи в таблицю tbl_attval з невизначеним значенням атрибута tbl_attval.att_adef_Id, що вказує на відповідний запис в таблиці атрибутів класів. На рисунку 2.2 ці зв'язки показані пунктиром.

Таким чином, можна створювати *k*-об'єкти невизначеною структури з довільним набором атрибутів або додавати довільні атрибути до *k*-об'єктів заздалегідь визначених класів. Розумним також представляється привести значення всіх атрибутів *k*-об'єктів до строковому типу. З одного боку це не забороняє робити операцію приведення типу згідно з визначеннями в tbl_attdef, з іншого - надає універсальність і додаткові можливості при описі нестандартних даних. Запропоновану схему можна ще більше розширити, додавши додаткову типізацію зв'язків об'єктів або встановивши додаткові правила на порядок проходження значень атрибутів в *k*-об'єкті, в залежності від конкретних вимог до гнучкості опису даних.

2.2 Каталог об'єктів інтегрований з реляційної схеми

Реляційні системи ефективно обробляють відносини з встановленими зв'язками. В рамках реляційної моделі можна описати інформацію будь-який заздалегідь відомої структури. Логічно було б використовувати в каталозі об'єктів всі переваги реляційної моделі для структурованої частини інформації, не втрачаючи можливостей системи по обробці неструктурованих даних. Це дозволило б інтегрувати і повторно використовувати вже наявні в реляційної

середовищі дані різних інформаційних систем і при цьому оптимізувати доступ до інформації.

Фактично мова йде про те, щоб описати в об'єктному вигляді реляционную схему. Для вирішення цього завдання необхідно встановити семантичні зв'язки між атрибутами реляційних таблиць і атрибутами класів, описаних в каталозі об'єктів. Це можна зробити, представивши набір атрибутів класу у вигляді ідентифікованого набору кортежів з різних відносин [8]. Нескладно бачити, що при певній і заздалегідь відомої структурі даних, в такому вигляді можна представити визначення будь-якого класу. Так, *k*-об'єкт «замовлення» можна уявити у вигляді набору кортежів з відносин «накладні», «список товарів», «клієнти», «деталі доставки», «менеджери» і т.д. В кожному відношенні кожного *k*-об'єкту будуть відповідати один або кілька кортежів, причому одне відношення може містити кортежі, що відносяться до різних *k*-об'єктів різних класів; *k*-об'єкти «покупець» і «продавець» можуть посилатися на кортежі одного і того ж відносини «клієнти».

Відомі труднощі, які виникають при установці посилань атрибутів *k*-об'єктів на атрибути таблиць. Припустимо, що кожен *k*-об'єкт типу «замовлення» має атрибут «покупець», який зберігається в реляційної таблиці tbl_cust. Логічно припустити, що кожного запису в tbl_cust буде відповідати тільки один екземпляр класу «замовлення», так як в цьому випадку дані примірників класу будуть незалежні один від одного. При необхідності послатися на одного клієнта з різних об'єктів можна створити в каталозі об'єктів клас «клієнт» і посилатися на один і той же екземпляр цього класу з різних *k*-об'єктів типу «замовлення». При цьому як і раніше один запис tbl_cust відповідає одному об'єкту класу «клієнт». Такий підхід є «правильним», але часто непридатним на практиці. У реляційних таблицях він призводить до дублювання записів, що не застосовується при використанні даних з реальних баз даних, на основі яких вже функціонують додатки.

Ці труднощі можна усунути, дозволивши посилання різних *k*-об'єктів на один і той же кортеж таблиці. Мінусом такого рішення є те, що значення

атрибутів різних об'єктів стають залежними один від одного. Так, при зміні даних з'являється можливість інтеграції будь-яких реляційних таблиць в каталог об'єктів. Таким чином, необхідно описати реляційну схему в каталозі об'єктів і встановити реляційні зв'язку типу «багато до багатьох» між списком об'єктів tbl_object і таблицями даних. Зауважимо, що посилання «один запис - один об'єкт» є окремим випадком цього відносини.

Для встановлення зв'язків необхідна наявність первинних ключів у всіх таблицях даних і формування додаткових таблиць посилань об'єктів на кортежі таблиць даних. Як приклад на рисунку 2.3 приведена частина реляційної схеми, яка описує замовлення і частина таблиць каталогу об'єктів. Таблиці зв'язку score_link, pay_link, cust_link формують взаємні посилання *k*-об'єктів і відповідних записів таблиць даних.

Для опису в каталозі об'єктів посилань на таблиці даних необхідно модифікувати структуру таблиці tbl_attdef. Додамо в неї атрибут adef_tblname. Запис в tbl_attdef тепер може описувати і атрибут деякого відносини, ім'я якого і визначається adef_tblname. Якщо adef_tblname не визначене, отже, значення атрибута зберігається в tbl_attval. Таким чином, об'єднання кортежів різних відносин в додаткову логічну структуру об'єкта вимагає існування в кожній таблиці даних первинного ключа, і, отже, не порушувати вихідну реляційну схему. Для вирішення задач даного класу необхідно:

- отримати список об'єктів заданого класу;
- отримати набір пар «атрибут-значення» для деякого об'єкта;
- отримати таблицю значень атрибутів для об'єктів заданого класу.

Всі інші завдання зазвичай є комбінацією перерахованих. Що стосується завдань 2 і 3, то вони вирішуються стандартними засобами будь-якої реляційної бази даних за допомогою нескладних SQL-запитів. наприклад:

```
select * from tbl_object where
 obj_cdef_id = <ід. класу>
select att_name, att_val from tbl_attval
```

where obj_id = <ід. об'єкта>

При цьому доступний весь спектр можливостей оптимізації таких запитів в СУБД.

У задачі 3 мова йде про формування результуючих відносини, атрибути яких відповідають атрибутам класу, описаних в таблиці tbl_classdef каталогу об'єктів. Відповідно кожному *k*-об'єкту з каталогу відповідає кортеж цього відношення, значення атрибутів повинні бути значення з таблиці tbl_attval. На практиці таке подання даних часто дуже наочно і зручно для обробки. Для реляційних СУБД формування результатів запиту у вигляді деякого уявлення таблиць - елементарна операція. На жаль, рішення цього завдання вимагає значних зусиль при формуванні результуючої таблиці із запропонованих схем даних. Так, для каталогу об'єктів спочатку потрібно отримати необхідний набір об'єктів. Припустимо, він існує у вигляді тимчасової таблиці tmp_obj_list. Потім необхідно сформувати і виконати запит такого типу:

```
SELECT b.att_val as acode1, c.att_val as acode 2,
 d.att_val as acode3 ...
FROM tmp_obj_list a LEFT OUTER JOIN tbl_attval b
 ON a.obj_id = b.obj_id
tmp_obj_list a LEFT OUTER JOIN tbl_attval c ON
 a.obj_id = c.obj_id
tmp_obj_list a LEFT OUTER JOIN tbl_attval d ON
 a.obj_id = d.obj_id
...
```

При досить великій кількості об'єктів в результуючій таблиці tmp_obj_list, або великій кількості атрибутів у вихідному наборі такий запит, безсумнівно, буде трудомісткою операцією для сервера баз даних. Підвищити продуктивність виконання запитів можна, застосувавши, коли це допустимо, комбіновану схему

каталогу об'єктів. У цьому випадку запит для прикладу на рисунку 6 можна модифікувати:

```
SELECT b. *, C.att_val as acode 2, d.att_val
 as acode3 ...
FROM tmp_obj_list a LEFT OUTER JOIN cust_link b
INNER JOIN tbl_cust e
ON e.cust_id = b.cust_id
ON a.obj_id = b.obj_id
tmp_obj_list a LEFT OUTER JOIN tbl_attval c ON
 a.obj_id = c.obj_id
tmp_obj_list a LEFT OUTER JOIN tbl_attval d a.obj_id = d.obj_id
...
```

Тут разом з атрибутами об'єктів витягується вся інформація про клієнтів.

2.3 Витяг списку об'єктів по шляху доступу для моделі Object Exchange Model

Отримання списку об'єктів по шляху доступу, взагалі кажучи, є нетривіальним. Шлях, як правило, представляє послідовний набір зв'язків між *n*-об'єктами (ребер графа OEM-моделі), який починається з деякого *n*-об'єкта, оголошеного точкою доступу (... .....). Стандартним також є вимога обробки шляхів доступу з шаблонами, як це зроблено в більшості мов запиту до слабкоструктурованих даними [5, 7, 14, 15]. На жаль, стандартними засобами реляційної моделі це завдання не вирішується. Алгоритм добування даних необхідно реалізовувати або на рівні додатку, або на сервері баз даних за

допомогою збережених процедур. Разом з тим, хотілося б максимально використовувати переваги реляційної технології для її вирішення.

Одним з основних методів оптимізації запитів в системах обробки квазіструктурірованних даних є побудова путівників по даним, таких як DataGuides [23] в Lore. Путівник по даним являє собою граф, побудований на основі OEM-схеми, узагальнюючий вихідну базу даних. У DataGuides підсумовані інформація про зв'язки між *n*-об'єктами. Кожен можливий шлях у вихідній базі даних існує в DataGuides, і, навпаки, будь-якого шляху в DataGuides відповідає хоча б один шлях у вихідній базі. Тому цей путівник може розглядатися як індекс по всіх можливих маршрутах в OEM-схемі (path index). На рисунку 2.4 наведено фрагмент DataGuides. Розміри путівника по даним зазвичай значно менше розмірів основної бази даних. Путівник сам є OEM-структурою; це дозволяє Lore зберігати його як OEM-базу даних і обробляти допомогою звичайних методів тестування.

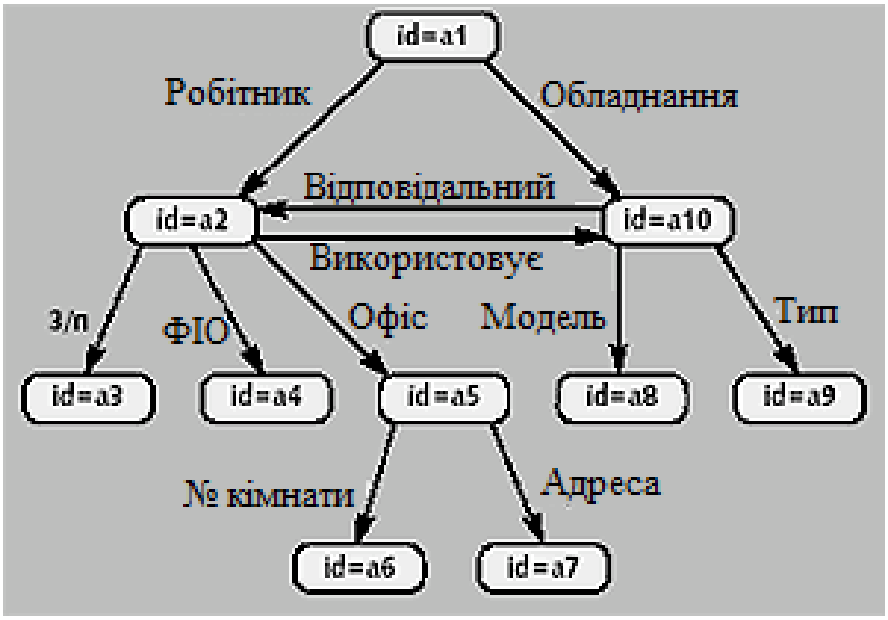

Рисунок 2.4 – DataGuides для OEM-бази з прикладу на рисунку 2.1

Після формування DataGuides для кожного його об'єкта створюють набори ідентифікаторів *n*-об'єктів у вихідній базі даних (target set), доступні за відповідним шляху в путівнику. Так для об'єкта «a10» див рисунок 2.4 набір

ідентифікаторів буде дорівнює «09, 10», для об'єкта «a4» - «03, 06, 07» і т.д. Далі формують дві таблиці посилань: targetHash (таблиця сформованих наборів ідентифікаторів); objectHash (таблиця посилань об'єктів вихідної бази даних на об'єкти DataGuides, в наборах ідентифікаторів яких він міститься).

Алгоритм виконання запиту з використанням DataGuides виглядає так.

1. Визначити наявність відповідного шляху в DataGuides.
2. Якщо шлях знайдений, знайти відповідне посилання в таблиці targetHash.
3. Отримати набір ідентифікаторів *n*-об'єктів.
4. Отримати набір значень *n*-об'єктів з набору ідентифікаторів, якщо це необхідно.

На практиці важливий окремий випадок OEM-схеми, в якій всі *n*-об'єкти об'єднані в ієрархічну структуру: кожен *n*-об'єкт має тільки один вхідний зв'язок від *n*-об'єкта більш високого рівня і скільки завгодно вихідних до *n*-об'єктів нижчого рівня. В цьому випадку можна об'єднати DataGuides і targetHash в одну таблицю, так як однієї записи в targetHash відповідає тільки один шлях в DataGuides. Результуюча таблиця (tbl_path) містить всі можливі шляхи від *n*-об'єкта - точки доступу до всіх об'єктів DataGuides і відповідні їм набори ідентифікаторів.

З використанням tbl_path вилучення даних відбувається в два етапи. Спочатку на основі запиту з таблиці tbl_path формується список n-об'єктів. При цьому можливе використання шаблонів. Якщо на результуючий список необхідно накласти фільтр або потрібно витягти додатково все або деякі атрибути об'єктів, то це відбувається за списком *n*-об'єктів, отриманого на першому етапі. До недоліків цього підходу слід віднести труднощі використання tbl_path при великих рівнях вкладеності *n*-об'єктів; це призводить до збільшення довжини шляхів і, безсумнівно, позначається на продуктивності. Перевагою ж є простота пошуку та ефективність вилучення даних за довільним шляхом.

З метою оптимізації продуктивності можна штучно виділити в вихідній OEM-структурі певну ієрархію об'єктів, створивши для неї tbl_path. Можна також створити кілька таблиць шляхів, кожну для зв'язків між *n*-об'єктами тільки

певного типу (типів). Це збільшує накладні витрати на підтримку декількох таблиць, але може дати значний виграш при пошуку.

Для визначення наявності відповідного шляху в DataGuides необхідно:

- якщо шлях знайдений, знайти відповідне посилання в таблиці targetHash;
- отримати набір ідентифікаторів *n*-об'єктів;
- отримати набір значень *n*-об'єктів з набору ідентифікаторів, якщо це необхідно.

Витяг списку об'єктів по шляху доступу для каталогу об'єктів.

Сказане стосується і для каталогу об'єктів (див. рисунок 2.2); слід враховувати тільки, що в якості вузлів графа виступають об'єкти каталогу (*k*-об'єкти). Разом з тим наявність додаткової метаінформації дозволяє в цьому випадку застосувати додаткові способи побудови DataGuides. Розглянемо один з можливих варіантів. Уявімо об'єкт каталогу у вигляді частини OEM-схеми. *k*-об'єкт - екземпляр класу «персона» в OEM-схемі представлений у вигляді графа з ребром «персона» і двох *n*-об'єктів (на рисунку вони позначені як (а) і (б)). *n*-об'єкти з'єднані єдиним зв'язком, що відповідає класу вихідного *к*-об'єкта - «персона». Вхідні зв'язки *n*-об'єкта (а) відповідають входам зв'язків вихідного *к*-об'єкта. Ідентифікатор *n*-об'єкта (а) не визначений. Ідентифікатор *n*-об'єкта (б) дорівнює ідентифікатору вихідного *k*-об'єкта. З *n*-об'єкта (б) виходять зв'язки вихідного *k*-об'єкта, а також зв'язки з іменами атрибутів вихідного *k*-об'єкта до атомарних об'єктів (їх ідентифікатори також не визначені), які мають значення атрибутів вихідного *k*-об'єкта «персона». Нескладно бачити, що таке еквівалентне уявлення дозволяє врахувати клас вихідного *k*-об'єкта безпосередньо в OEM-структурі. Він описується зв'язком між *n*-об'єктами (а) і (б).

Побудуємо на основі цього еквівалентного OEM-уявлення об'єктів каталогу DataGuides. Поставимо у відповідність кожному k-об'єкту граф, що складається з:

- *n*-об'єкта (а) з невизначеним ідентифікатором, що входять зв'язками вихідного *k*-об'єкта і одним вихідним зв'язком;
- *n*-об'єкта (б) з одним вхідним зв'язком, з вихідними зв'язками вихідного *k*-об'єкта і ідентифікатором, рівним ідентифікатору вихідного *k*-об'єкта;

- зв'язку між *n*-об'єктами (а) і (б), що відповідає класу *k*-об'єкта.

Так як ідентифікатор *n*-об'єкта (а) не визначений, що не будемо враховувати його при побудові таблиць targetHash, objectHash. Це накладає обмеження на шляху, які можна використовувати для отримання списку *k*-об'єктів. Кожен шлях має закінчуватися ребром, що відповідає класу *k*-об'єкта і всі пари <вхідний зв'язок>. <Тип об'єкта> («робітник персона» для рисунку 8), зазначені в дорозі, повинні бути повними. Таким чином, DataGuides формується з урахуванням метаінформації каталогу об'єктів.

На практиці часто всі об'єкти каталогу мають імена, що полегшує для користувачів ідентифікацію об'єктів і навігацію по каталогу. В цьому випадку можна побудувати DataGuides, для якого зв'язок між об'єктами (а) і (б) приймає значення імені *k*-об'єкта.

Важливим є випадок, в якому каталог об'єктів являє собою ієрархічну структуру з іменованими об'єктами і неіменованого зв'язками. Для цього варіанту каталогу об'єктів DataGuides містить тільки імена k-об'єктів, що значно спрощує його структуру. Також для ієрархічних каталогів об'єктів можлива побудова індексу шляхів (таблиці tbl_path), як було описано для OEM – моделі.

3 РЕАЛІЗАЦІЯ АЛГОРИТМУ ОПРАЦЮВАННЯ СТРУКТУРОВАНИХ ДАНИХ ЗАДАНИХ У ВИГЛЯДІ ГРАФА

3.1 Вибір мови програмування

Розробка програмного забезпечення здійснювалася за допомогою мови програмування Java. Java є строго типізований об'єктно-орієнтованою мовою програмування. Програми на Java компілюються за допомогою Javac в спеціальний байт-код, який виконується віртуальною машиною Java (JVM), таким чином мову Java це одночасно і компільована і інтерпретована мова програмування [1]. Перевагою особливості мови є повна незалежність байт-коду від операційної системи і устаткування, що дає можливість виконувати програми на мові Java на будь-якому пристрої, для якого існує віртуальна машина. Також важливою особливістю мови Java це гнучка система безпеки, в рамках якої віртуальна машина повністю контролює виконання програми. Будь-які операції, які перевищують встановлені повноваження програми (такі як, спроба несанкціонованого доступу до даних), викликають негайне переривання.

Мова програмування Java є одною з найбільш популярних мов для створення прикладних програм.Існує три версії Java платформи:

– Java Standard Edition (Java SE), це видання є основним в Java, воно містить (JRE), API, компілятори; воно добре може підійти для створення призначеного для користувача програмного забезпечення, в першу чергу - для настільних систем;

– Java Enterprise Edition (Java EE), являє собою стек специфікацій, а також документації, яка необхідна для створення програмного забезпечення рівня підприємства;

– Java Micro Edition (Java ME), призначена для використання в пристроях, в яких обмежена по обчислювальній потужності, наприклад: в мобільних телефонах, КПК, вбудованих системах (в даний час використовується досить рідко).

Важливою перевагою мови Java над мовою C ++. Це автоматичне очищення пам'яті від невикористовуваних об'єктів. Ця технологія називається Garbage Collection (Прибирання сміття) і призначена для того щоб уникнути витоків пам'яті в програмі.

3.1.1 Вибір графічної бібліотеки

У мові програмування Java є три бібліотеки візуальних компонентів для створення графічного інтерфейсу користувача. Найперша з них це AWT. При її проектуванні був допущено безліч недоліків, і тим самим з нею досить складно працювати. Бібліотека Swing розроблена на базі AWT і замінює більшість її компонентів своїми, спроектованими більш ретельно і зручно. Третьою бібліотекою для створення графічного інтерфейсу є SWT.

3.1.2 Бібліотека AWT

AWT була найпершою бібліотекою для створення графічного інтерфейсу на Java. Ця бібліотека реалізує деякий програмний прошарок, яка дозволяє викликати з програм, написаних на Java методи для роботи з інтерфейсом написаних на мові C. Методи цієї бібліотеки створюють і далі використовують графічні компоненти операційної системи, в якій вони запускаються [2]. У такому підході є як позитивні моменти, так і негативні. Приклад позитивного моменту в тому, що програма, яка написана на мові Java буде дуже схоже на всі інші програми в операційній системі. А негативний момент в тому, що не існує ніяких гарантій що існуючі відмінності в різних компонентах і шрифт не будуть псувати зовнішній вигляд програми при запуску її на іншій платформі.

Переваги:

- вбудована в стандартний JDK;
- достатньо висока швидкість роботи;
- графічні компоненти такі ж, як і стандартні в операційній системі.

Недоліки:

– при використанні будь-яких нативних графічних компонентів накладається обмеження на використання їх властивостей. Деякі компоненти бібліотеки можуть не працювати на деяких платформах;

– деякі властивості, наприклад, спливаючі підказки і іконки, в бібліотеці AWT повністю відсутні;

– мало стандартних графічних компонентів в бібліотеці AWT, розробнику необхідно реалізовувати свої власні графічні елементи;

– графічний інтерфейс виглядає по-різному на різних платформах (можуть бути проблеми з відображення на різних платформах).

3.1.3 Бібліотека Swing

Бібліотека графічних компонентів Swing була створена з метою виправити всі недоліки попередньої бібліотеки AWT. Компоненти Swing повністю написані на мові Java а не на C як у попередній бібліотеці. Для відтворення компонентів використовується 2D, що дає деякі переваг. Бібліотека Swing має набагато більший набір стандартних графічних компонентів у порівнянні з бібліотекою AWT як за різноманітністю, так і по функціональності [3]. Використовуючи бібліотеку Swing стало набагато легше створювати нові компоненти, для цього потрібно просто успадковуватися від вже існуючих класів і змінити їх у міру необхідності.

Переваги:

– вбудована в стандартний JDK;

– велику кількість книжок і статей по бібліотеці Swing;

– багато середовищ розробки мають вбудовані редактори форми;

– існує багато розширень на базі Swing такі як SwingX;

– підтримка різних стилів.

Недоліки:

– вікно з великою кількістю компонентів може почати пригальмовувати.

3.1.4 Бібліотека SWT

Бібліотека SWT була розроблена в компанії IBM, так як бібліотека Swing занадто повільної для них, також вони цю бібліотеку просували разом з їх середовищем розробки Eclipse. SWT, на зразок з AWT, використовує компоненти операційної системи, в якій вона запускається, але для кожної платформи були розроблені свої власні інтерфейси взаємодії [4]. Так що для кожної нової системи доведеться надати ще одну окрему JAR-бібліотеку з відповідною для SWT версією. Це дозволяє краще використовувати існуючі функції графічних компонентів.

Переваги:

– використовується компоненти операційної системи;

– є редактор форм в популярному середовищі розробки Eclipse;

– висока швидкість роботи;

– обширна документація по використанню бібліотеки;

– можливе використання компонентів з бібліотек AWT і Swing.

Недоліки:

– для кожної операційної системи необхідно надавати окрему бібліотеку;

– необхідно бути вкрай обережним при використанні ресурсів і вчасно їх звільняти;

– досить складна архітектура, відносно великий поріг входження.

Після аналізу головних бібліотек для створення графічного інтерфейсу, була обрана бібліотека Swing, так як вона є досить потужною і добре документованою.

3.2 Схема структури даних

Кожен об'єкт має ідентифікатор (id), який присвоюється за наступним правилом: спочатку цифри ідентифікатора предка, а потім цифра самого об'єкта.

Кожне відношення має fromid (ідентифікатор предка) і toid (ідентифікатор нащадка).

На рисунку 3.1 представлена схема структури графа з усіма типами об'єктів (вершин) і відносин (дугами).

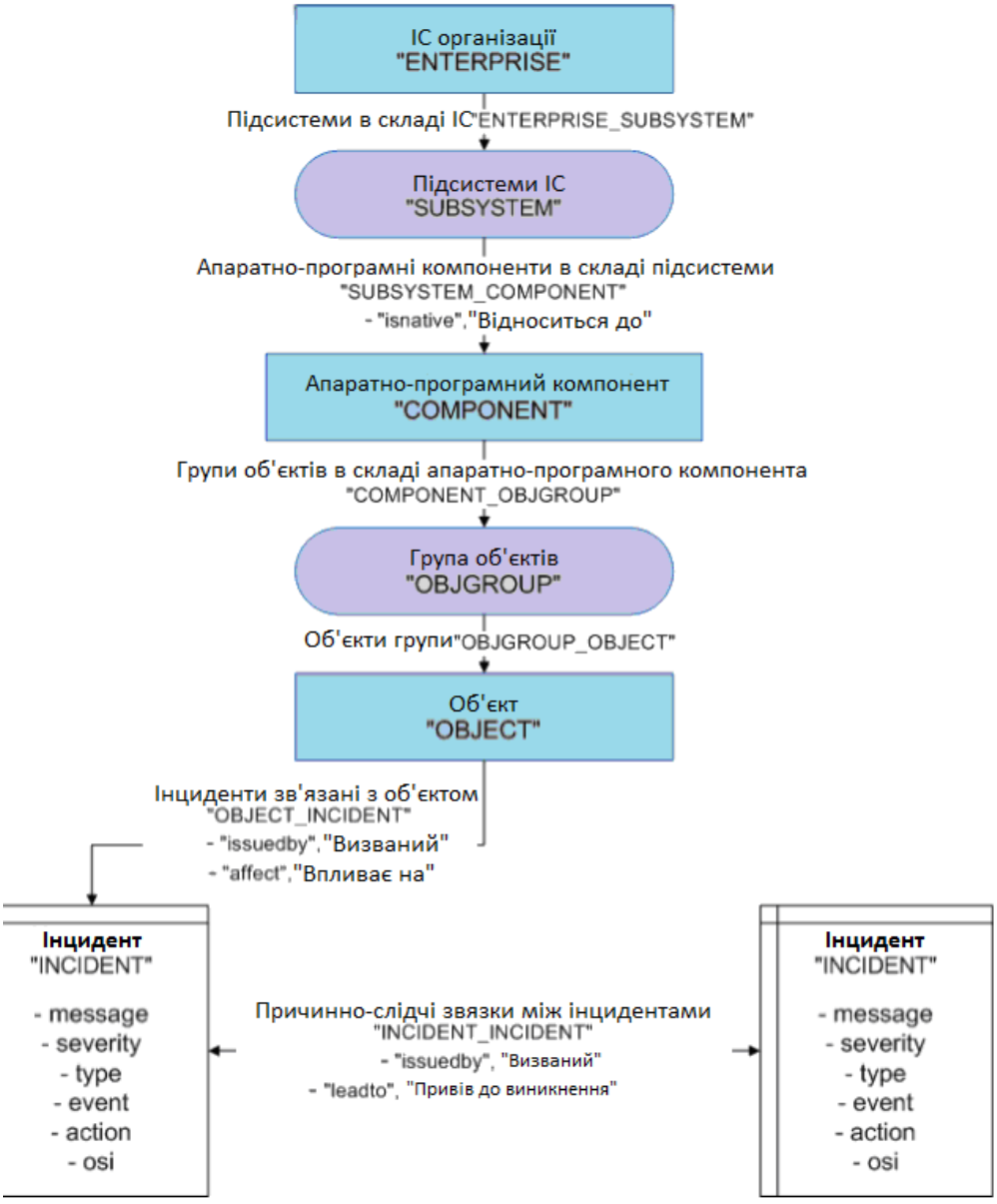

Рисунок 3.1 – Схема графа

3.2.1 Вибір аналізатора XML документів

XML (Extensible Markup Language) - це мова розмітки документів, який дозволяє структурувати дані з різною структурою, використовуючи для цього довільний набір інструкцій. XML був розроблений як мова з досить простим для розуміння формальним синтаксисом, який зручний для створення, а також обробки документів програмами і в той же час зручний для читання і редагування документів людиною [5].

Будь- який XML-аналізатор, будучи, транслятором даної мови розмітки і може бути розбитий на кілька різних модулів, які відповідають за синтаксичний, лексичний, а також семантичний аналіз всього вмісту документа. Очевидно, що, якщо було необхідно програмісту кожен раз писати всі ці блоки самому, переваги XML розмітки як в такому відпала б, так як головна його перевага, як це вже згадувалося, є стандартний підхід добування інформації з документа. Якщо XML-документ синтаксично правильно складений, то він може бути розібраний будь-яким універсальним аналізатором XML розмітки, і XML оброблювачу необхідно тільки використовувати отримані на виході дані, які пройшли синтаксичний аналіз, а далі інтерпретувати вміст XML документа, відповідно з його DTD структурою даних.

В даний час багато основних універсальних XML-аналізаторів є подієво-орієнтованими API SAX - Simple API for XML. Аналізатор виявляє деяку синтаксичну конструкцію даного XML-документа (символ, елемент, шаблон), фіксує початок і кінець оголошень елементів, після чого переглядає DTD-правила або знаходить помилки [6]. Інформація про збережену структуру документа передається йому в якості параметрів функції. Обробник події - це такий собі клас в програмі, який виконує дії, які необхідні для обробки XML документа і здійснює таким чином розбір вмісту. Після завершення даної функції управління знову передається XML-аналізатору і процес розбору документа триває.

При розробленому програмному забезпеченні використовується DOM аналізатор, так як будь-яка дія, яку вчиняє користувач в програмі тягне за собою безліч операцій над документом таких як пошук потрібного компонента пошук

всіх зв'язків з цим компонентом і додавання, зміна або видалення самого компонента. DOM аналізатор завантажує весь документ в пам'ять і будує дерево компонентів, в якому робота буде здійснюватися набагато швидше ніж проходячи весь файл кожного разу [7].

Так як робота здійснюється з одним і тим же файлом його можна завантажити один раз при завантаженні програми і зберегти посилання на Java об'єкт. Для цього можна використовувати патерн проектування одиночка (Singleton), який дає можливість контролювати, що об'єкт буде завантажений, і він буде єдиним об'єктом цього класу.

Структура паттерна Singleton заснована на використанні однієї копії об'єкта у всій програмі, доступ до якої можна отримати з будь-якої точки в програмі [8].

3.3 Тестування програмного забезпечення

Найпопулярніший вид тестування програмного забезпечення - це модульне тестування. Це тестування проводиться для окремих модулів [9].

Для модульного тестування програмного забезпечення використовувалася бібліотека JUnit 4.

Так як програма працює з XML файлом необхідно було додати спеціальний тестовий XML файл з такою ж структурою, але з явно відомими даними в ньому. Нижче представлений скріншот успішного проходження тестування.

На рисунку 3.2 видно, що тестування пройшло повністю успішно. Тести покрили всі дії (додавання, зміни, видалення, отримання всіх об'єктів, отримання об'єктів, пов'язаних з іншим об'єктом) над усіма видами об'єктів (Enterprise, Subsystem, Component, ObjGroup, Object, Incident).

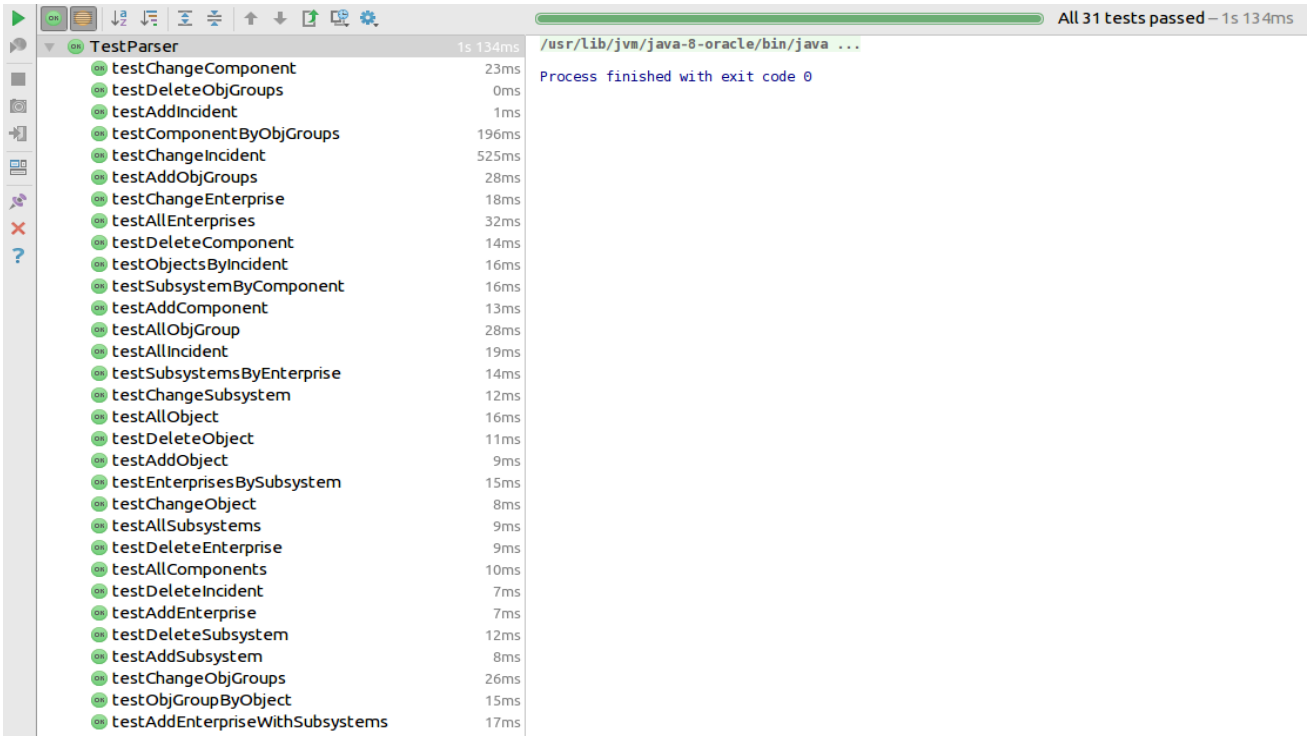

Рисунок 3.2 – Результати тестів

3.3.1 Проектування схеми бази даних

У розробленому програмному забезпеченні також використовується база даних. Бази даних має 9 таблиць, які наведені нижче. PK - primary key (первинний ключ) і FK - foreign key (зовнішній ключ). В таблицях 3.1 -3.4 приведені дані взаємодії структурованих даних заданих у вигляді графа.

Таблиця 3.1–Таблиця Enterprise

Назва	Тип
id (PK)	Numeric
enterprisename	Varchar
comment	Varchar

Таблиця 3.2 –Таблиця Subsystem

Назва	Тип
id (PK)	Numeric
subname	Varchar
comment	Varchar

Таблиця 3.3 – Таблиця Component

Назва	Тип
id (PK)	Numeric
compname	Varchar
comment	Varchar
vendor	Varchar

Таблиця 3.4 – Таблиця ObjGroup

Назва	Тип
id (PK)	Numeric
objgrpname	Varchar
belongsto	Varchar
id_component (FK)	Numeric

В таблиці 3.5 представлені назви та типи об’єктів, які формують базу даних.

Таблиця 3.5 – Таблиця Object

Назва	Тип
id (PK)	Numeric
objname	Varchar
belongsto	Varchar
id_objgroup (FK)	Numeric

В таблиці 3.6 представлені дані про назву та тип бази даних Incident.

Таблиця 3.6 – Таблиця Incident

Назва	Тип
id (PK)	Numeric
code	Varchar
message	Varchar
severity	Varchar
type	Varchar
event	Varchar
action	Varchar
osi	Numeric
comment	Varchar

В таблиці 3.7 представлені назви і типи Enterprise_Subsystem, які взаємодіють між собою.

Таблиця 3.7 – Таблиця Enterprise_Subsystem

Назва	Тип
id_enterprice (FK)	Numeric
id_subsystem (FK)	Numeric

В таблиці 3.8 вказані дані про Subsystem_Component, які є важливими при формувані бази даних.

Таблиця 3.8 – Таблиця Subsystem_Component

Назва	Тип
id_subsystem (FK)	Numeric
id_component (FK)	Numeric
isnative	Bool

Аналогічним чином формуються дані в таблиці 3.9, які вказують зв'язок з назвою та типом.

Таблиця 3.9 – Таблиця Object_Incident

Назва	Тип
id_obj (FK)	Numeric
id_incident (FK)	Numeric
issuedby	Bool
affect	Bool
comment	Varchar

На рисунку 3.2 показана підсумкова схема бази даних.

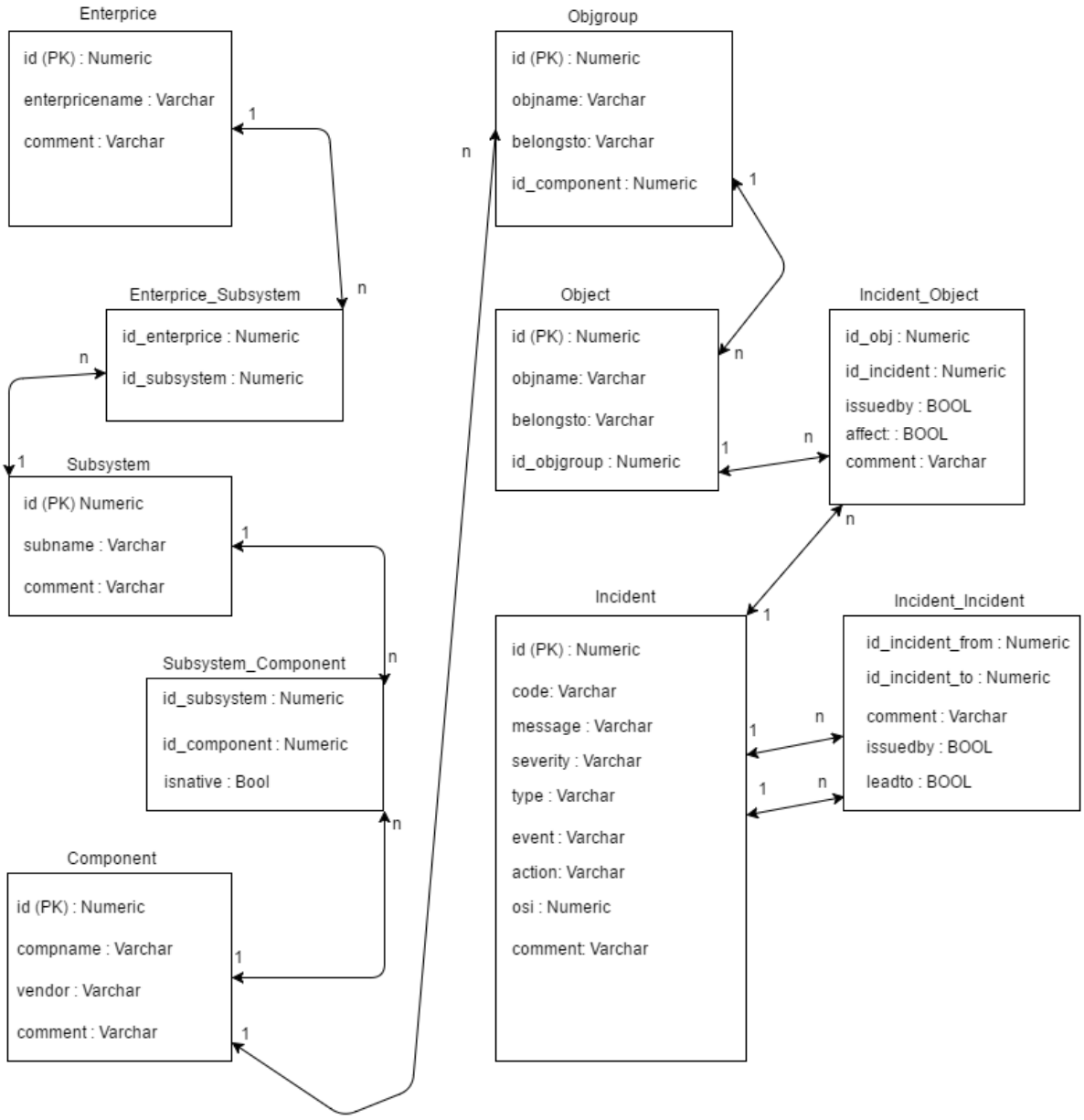

Рисунок 3.2 – Схема база даних

Одна з найпопулярніших моделей даних це реляційна модель. У такій моделі існує певний зв'язок між сутностями таблиць. Для зберігання і роботи з даними, така СУБД повинна мати певну структуру (таблиці). У таблицях кожен стовпець може містити дані різного типу.

Ця СУБД SQLite досить просто вбудовується в додатки, так як вона в основному базується на файлах, і надає досить широкий набір інструментів для

роботи з нею, особливо в порівнянні з мережевими СУБД. При роботі з цією системою звернення відбуваються до файлів безпосередньо (ті файли в яких безпосередньо зберігаються дані) [10]. Саме тому SQLite дуже швидка і потужна СУБД, яка має дуже широке поширення.

Переваги SQLite:

- вся файлова структура бази даних складається з одного файлу;
- дана СУБД повністю підтримує структуровану мову запитів SQL;
- зручна для розробки і тестування додатків.

Недоліки SQLite:

- відсутність системи користувачів. Ця особливість не настільки критична так як це рідко буває необхідно базам даних які вбудовуються в додатки;
- відсутність можливості для збільшення продуктивності, що також не настільки критично для бази даних яка призначена для вбудовування в додаток.

3.3.2 Вибір середовища розробки

- NetBeans це досить потужне середовище розробки з відкритим вихідним кодом. Дане середовище розробки орієнтоване на настільні додатки;
- IntelliJ IDEA розроблена компанією JetBrains. Вона поширюється відразу в двох версіях, одна з них повністю безкоштовна, а за другу необхідно платити щороку;
- Eclipse одне з найпопулярніших середовищ для розробки на Java. Дане середовище розробки орієнтоване на веб і мобільні додатки;
- JDeveloper це ще один продукт від компанії Oracle з великою кількістю переваг, серед яких підтримка хмарного сервісу Oracle і системи контролю версій. Це досить універсальний продукт, в якому можна працювати з такими технологіями: PL / SQL оброблювачем запитів, редакторами HTML, CSS, JavaScript.

Для розробки програмного забезпечення була обрано середовище розробки IntelliJ IDEA.

3.4 Структура програми

На рисунку 3.3 показана структура програми з наступними пакетами:

- Package gui - містить класи, що відповідають за відлагодження графічного інтерфейсу;
- Package gui.components - містить класи спадкоємці від стандартних класів в бібліотеці Swing;
- Package dao - містить класи, які необхідні для роботи з даними в середині програми;
- Package parsers - містить класи, які відповідають за обробку XML документа.

У пакеті dao знаходяться класи які зберігають інформацію про об'єкти і зв'язки між ними. Ці класи дають можливість зручно взаємодіяти з даними.

У пакеті gui знаходяться всі класи, що відповідають за графічний інтерфейс програми. Класи, назва яких закінчується на GUI відповідають за відображення вікон. Всього 8 вікон у програмі 6 з яких відповідають за редагування об'єктів (Enterprise, Subsystem, Component, ObjGroup, Object, Incident), також є головне вікно, в якому можна вибрати тип об'єкта для подальшого редагування. І останнє вікно - це вікно налаштувань. У всіх класах по редагуванню об'єктів є підкласи, які відповідають за асинхронне редагування об'єктів (додавання, зміна, видалення). Ці операції необхідно проводити в окремому потоці, так як вони можуть зайняти досить багато часу і тим самим блокувати графічний інтерфейс.

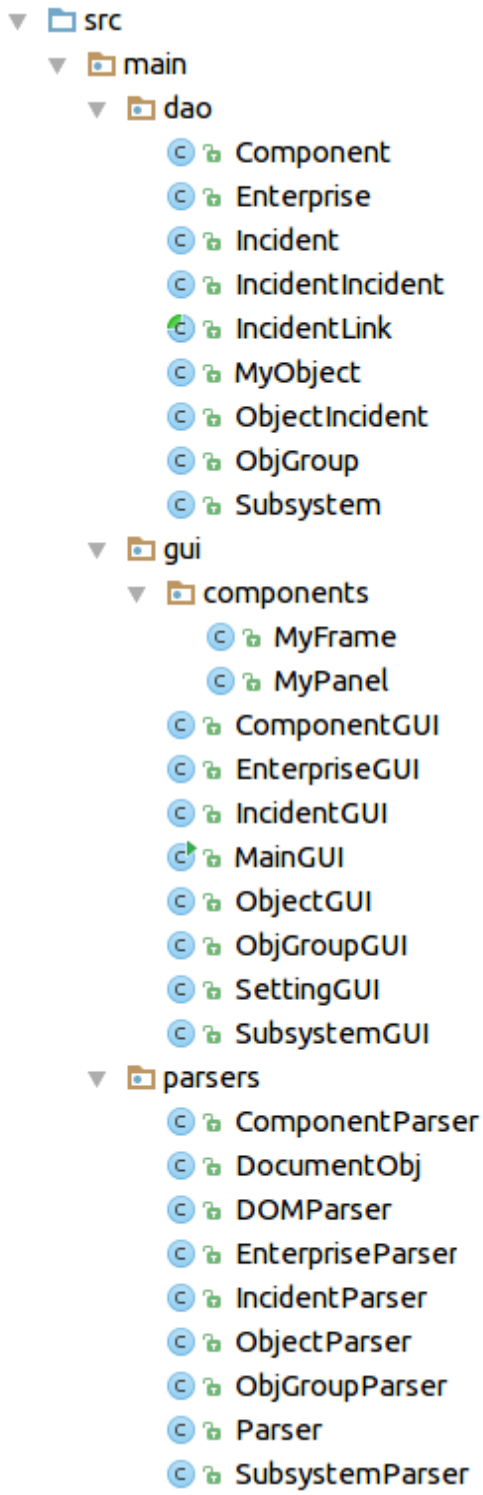

Рисунок 3.3 – Структура програми

У пакеті gui.components знаходяться класи, які успадковують стандартні класи в бібліотеці Swing. Це необхідно для створення власних елементів графічного інтерфейсу.

У пакеті parsers знаходяться класи, які реалізують основну логіку програми. Вони відповідають за редагування структурованих даних і підтримки їх

цілісності. Клас DocumentObj реалізує патерн проектування Singleton. Це дає можливість завантажити дані з XML файлу раз і працювати з ними з будь-якої точки програми. У цьому пакеті також присутні класи для роботи з даними об'єктів кожного типу (Enterprise, Subsystem, Component, ObjGroup, Object, Incident). Кожен з них вміє отримувати об'єкти свого типу, змінювати їх, додавати нові і видаляти непотрібні. Отримання об'єктів може бути декількох типів:

- отримання всіх об'єктів даного типу;
- отримання об'єкта по id;
- отримання всіх об'єктів, які пов'язані з заданим батьківським об'єктом.

При додаванні нових об'єктів спочатку перевіряється правильність введених даних, а потім генерується спеціальний ідентифікатор id за таким правилом: перші числа - це ідентифікатор предка, до якого справа додається число, яке дорівнює кількості вже пов'язаних з предком об'єктів. Також для додавання об'єкта необхідно вказати батька об'єкта (Для Incident і Component необхідно також вказати параметри відносини з предком).

При зміні об'єктів спочатку знаходиться потрібний об'єкт за ідентифікатором, потім змінюються параметри цього об'єкта. Якщо змінились якісь стосунки з об'єктом, то старий зв'язок видаляється і створюється новий зв'язок.

На рисунку 3.4 представлений пакет з класами тестів в них реалізовані тести, пов'язані з кожним типом об'єктів.

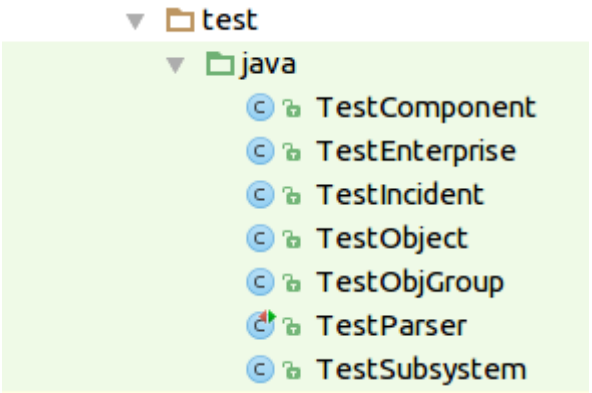

Рисунок 3.4– Структура тестів

Отриманий результат - це розроблене програмне забезпечення, яке дозволяє користувачеві редагувати дані представлені в вигляді графа.

Модель даних по якій розроблялася програма представляється у вигляді графа, вершини якого є об'єкти, а дуги - це відношення між цими об'єктами. Дана модель складається з шести типів об'єктів (вершин):

- організація (Enterprise),
- підсистеми ИС (Subsystem),
- апаратно-програмні компоненти підсистем (Component),
- групи об'єктів в складі компонентів (ObjGroup),
- атомарні об’єкти (Object),
- інциденти (Incident).

Також в моделі присутні шість типів відносин (дуг) між об'єктами:

- ІС – підсистема,
- підсистема – компонент,
- компонент – група об’єктів,
- група об’єктів – атомарний об’єкт,
- атомарний об’єкт – інцидент,
- інцидент – інцидент.

3.4.1 Представлення даних у XML розмітці

Структура даних для зберігання організацій:

```
<node id="1" type="ENTERPRISE">
<property name="enterprisename" value="Apple"/>
<property name="comment" value=""/>
</node>
```

Структура даних для зберігання підсистем ІС:

<node type="SUBSYSTEM" id="11">
<property name="subname" value=" Серверне обладнання " />
<property name="comment" value=" Конфігурація і управління серверним апаратним забезпеченням " />
</node>

Структура даних для зберігання апаратно-програмних компонентів підсистем:

<node type="COMPONENT" id="191">
<property name="compname" value="Lotus Domino 8.5.2" />
<property name="vendor" value="IBM" />
<property name="comment" value=" Серверне програмне забезпечення " />
</node>

Структура даних для збереження груп об'єктів:

<node type="OBJGROUP" id="1413">
<property name="objgname" value="Жесткий диск" />
<property name="belongsto" value=" " />
</node>

Структура даних для збереження обєктів:

<node type="OBJECT" id="13221">
<property name="objname" value="RJ11" />
<property name="belongsto" value=" " />
</node>

Структура даних для збереження інцидентів:

<node type="INCIDENT" id="191013">

<property name="code" value="INC_0026" />

<property name="message" value="No route found to domain from server" />

<property name="severity" value="Висока" />

<property name="type" value="Сервер Lotus Domino" />

<property name="event" value=" Не вдалося знайти маршрут для передачі поштового повідомлення в зазначений домен по протоколу NRPC " />

<property name="action" value=" Перевірте настройки документів Server і Domain; переконайтеся в наявності правильно сконфигурированного документа типу Connection " />

<property name="osi" value="7" />

<property name="comment" value=" Не вдалося знайти маршрут до вказаного поштового домену " />

</node>

3.4.2 Представлення даних відносин між об'єктами

Структура даних для зберігання відносин ІС - підсистема:

<edge fromid="2" toid="21" type="ENTERPRISE_SUBSYSTEM"/>

Структура даних для зберігання відносин підсистема - компонент:

<edge type="SUBSYSTEM_COMPONENT" fromid="11" toid="111">

<property name="isnative" value="true"/>

</edge>

Структура даних для зберігання відносин компонент - група об'єктів:

<edge fromid="214" toid="2141" type="COMPONENT_OBJGROUP"/>

Структура даних для зберігання відносин група об'єктів - атомарний об'єкт:

<edge fromid="2141" toid="21411" type="OBJGROUP_OBJECT"/>

Структура даних для зберігання відносин атомарний об'єкт - інцидент:

<edge fromid="21411" toid="214111" type="OBJECT_INCIDENT">
<property name="issuedby" value="true"/>
<property name="affect" value="false"/>
<property name="comment" value="Вызван"/>
</edge>

Структура даних для зберігання відносин інцидент - інцидент:

<edge fromid="214111" toid="214112" type="INCIDENT_INCIDENT">
<property name="issuedby" value="false"/>
<property name="leadto" value="true"/>
<property name="comment" value="Визвав"/>
</edge>

3.4.3 Графічний інтерфейс програми

Для редагування різних типів об'єктів (Enterprise, Subsystem, Component, ObjGroup, Object, Incident) створені спеціальні вікна, в яких можна редагувати вибраний об'єкт, разом з відносинами з цим об'єктом або додати новий об'єкт. На рисунку 3.5 показаний скріншот графічного інтерфейсу, який відповідає за редагування Enterprise.

Щоб додати новий Enterprise потрібно натиснути кнопку "Додати новий" і заповнити всі поля. Після чого необхідно натиснути на кнопку додати. Щоб до Enterprise додати відносини Enterprise-Subsystem потрібно вибрати Subsystem

справа в списку і натиснути на "Додати до Enterprise", після натискання кнопки в списку Enterprise-Subsystem з'явиться нове ставлення, яке стосується форми введення.

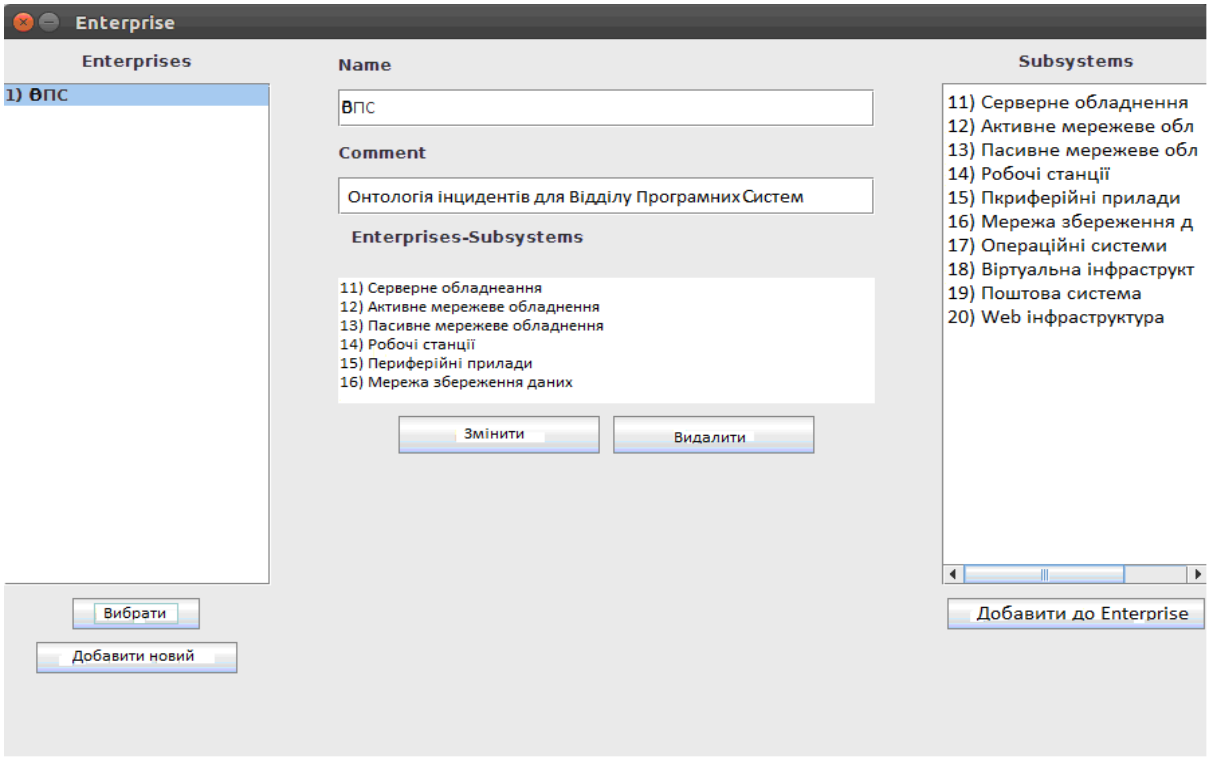

Рисунок 3.5– Вікно Enterprises

Якщо ви хочете відмінити відношення Enterprise-Subsystem потрібно натиснути на нього і з'явиться кнопка над списком "Прибрати Enterprise-Subsystem", яку потрібно натиснути, щоб прибрати це відношення зі списку.

Щоб змінити вже існуючий Enterprise потрібно знайти цей Enterprise в списку ліворуч (список Enterprise) натиснути на знайдений Enterprise і натиснути на кнопку "Вибрати". Після натискання на кнопку "Вибрати" всі параметри даного Enterprise з'являться у відповідних полях, а відносини Enterprise-Subsystem в списку. Після зміни параметрів Enterprise і його зв'язків потрібно натиснути на кнопку "Змінити".

Щоб видалити вже існуючий Enterprise потрібно знайти цей Enterprise в списку ліворуч (список Enterprises) натиснути на знайдений Enterprise і натиснути на кнопку "Вибрати". Після натискання на кнопку "Вибрати" потрібно натиснути

на кнопку "Видалити", після чого з'явиться вікно підтвердження видалення запису де потрібно натиснути на кнопку "Yes".

На рисунку 3.6 показаний скріншот графічного інтерфейсу, який відповідає за редагування Subsystem.

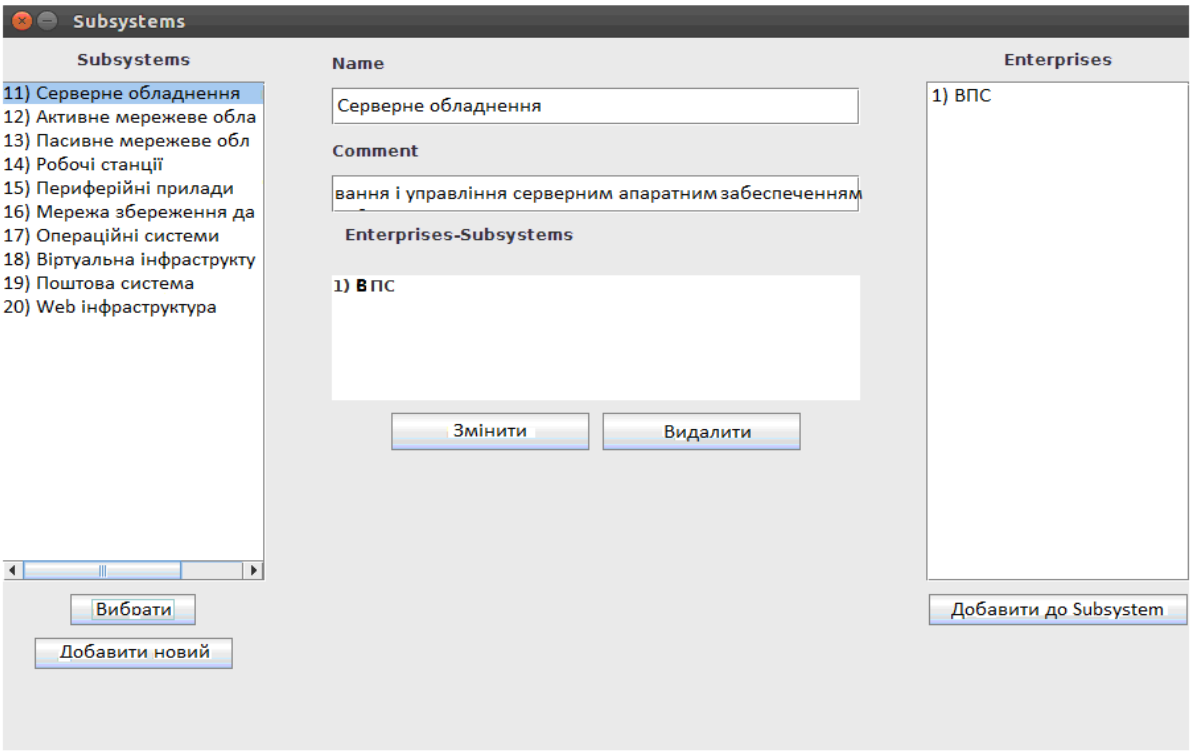

Рисунок 3.6 – Вікно Subsystems

Щоб додати новий Sybsustem потрібно натиснути кнопку "Додати новий" і заповнити всі поля. Після чого необхідно натиснути на кнопку додати. Щоб до Subsystem додати відносини Enterprise-Subsystem потрібно вибрати Enterprise справа в списку і натиснути на "Додати до Sybsustem", після натискання кнопки в списку Enterprise-Subsystems з'явиться нове ставлення, яке стосується форми введення.

Якщо ви хочете відмінити відношення Enterprise-Subsystem потрібно натиснути на нього і з'явиться кнопка над списком "Прибрати Enterprise-Subsystem", яку потрібно натиснути, щоб прибрати це відношення зі списку.

Щоб змінити вже існуючий Sybsustem потрібно знайти цей Sybsustem в списку ліворуч (список Sybsustems) натиснути на знайдений Sybsustem і

натиснути на кнопку "Вибрати". Після натискання на кнопку "Вибрати" всі параметри даного Sybsustem з'являться у відповідних полях, а відносини Enterprise-Subsystem в списку. Після зміни параметрів Sybsustem і його зв'язків потрібно натиснути на кнопку "Змінити".

Щоб видалити вже існуючий Sybsustem потрібно знайти цей Sybsustem в списку ліворуч (список Sybsustems) натиснути на знайдений Sybsustem і натиснути на кнопку "Вибрати". Після натискання на кнопку "Вибрати" потрібно натиснути на кнопку "Видалити", після чого з'явиться вікно підтвердження видалення запису де потрібно натиснути на кнопку "Yes". На рисунку 3.7 показаний скріншот графічного інтерфейсу редагування Components.

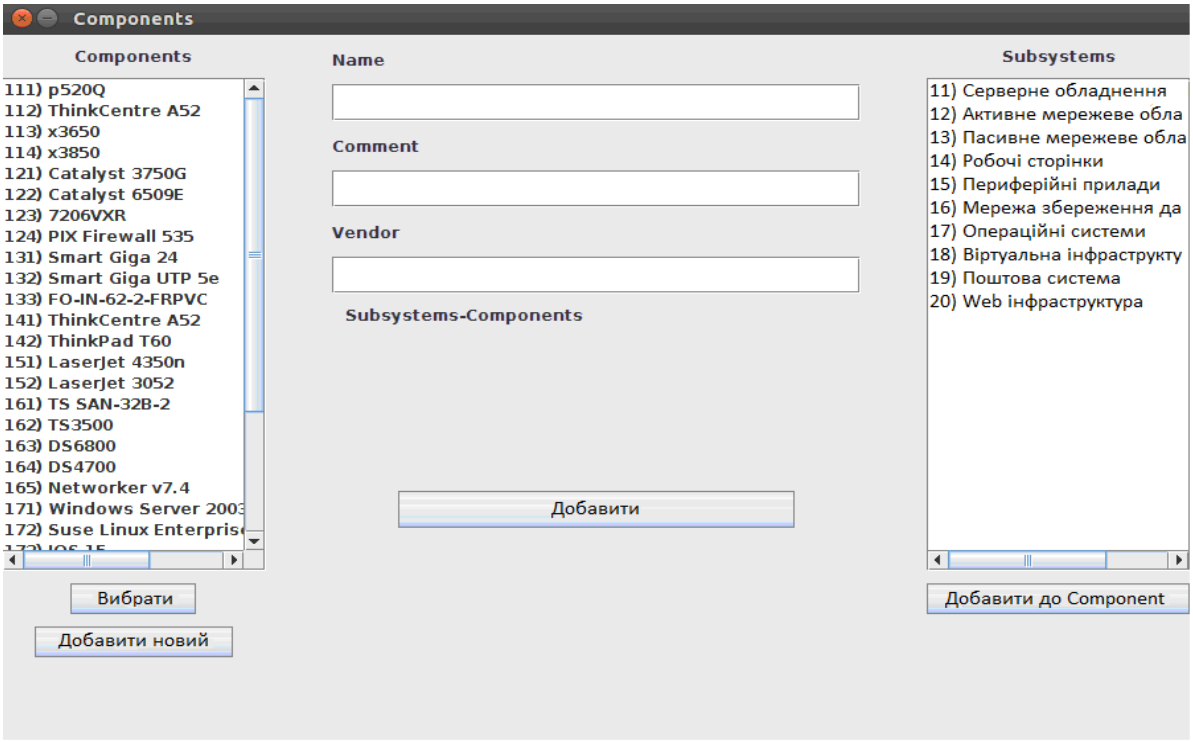

Рисунок 3.7– Вікно Components

Щоб додати новий Component потрібно натиснути кнопку "Додати новий" і заповнити всі поля. Після чого необхідно натиснути на кнопку додати. Щоб до Component додати відносини Subsystem-Component потрібно вибрати Subsystem справа в списку і натиснути на "Додати до Component", після натискання кнопки з'явиться вікно для вибору параметра відносини (isnative). На рисунку 3.8 нижче

представлено діалогове вікно, в яке можна вибрати для параметра isnative два значення true і false.

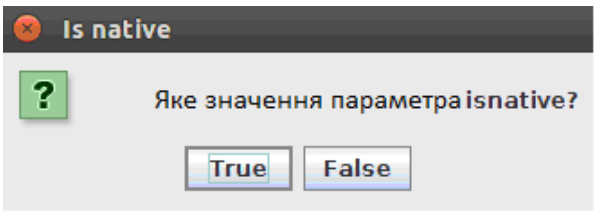

Рисунок 3.8 – Вікно Is native

Після вибору значення параметра діалоговому вікні потрібно натиснути на кнопку "ОК" і в списку Subsystems-Components, яке стосується форми введення, з'явиться цей показник.

Якщо ви хочете відмінити відношення Subsystem-Component потрібно натиснути на нього і з'явиться кнопка над списком "Прибрати Subsystem-Component", яку потрібно натиснути, щоб прибрати це відношення зі списку.

Щоб змінити вже існуючий Component потрібно знайти цей Component в списку ліворуч (список Components) натиснути на знайдений Component і натиснути на кнопку "Вибрати". Після натискання на кнопку "Вибрати" всі параметри даного Component з'являться у відповідних полях, а відносини Subsystem-Component в списку. Після зміни параметрів Component і його зв'язків потрібно натиснути на кнопку "Змінити".

Щоб видалити вже існуючий Component потрібно знайти цей Component в списку ліворуч (список Component) натиснути на знайдений Component і натиснути на кнопку "Вибрати". Після натискання на кнопку "Вибрати" потрібно натиснути на кнопку "Видалити", після чого з'явиться вікно підтвердження видалення запису де потрібно натиснути на кнопку "Yes".

На рисунку 3.9 показаний скріншот графічного інтерфейсу, який відповідає за редагування ObjectGroup

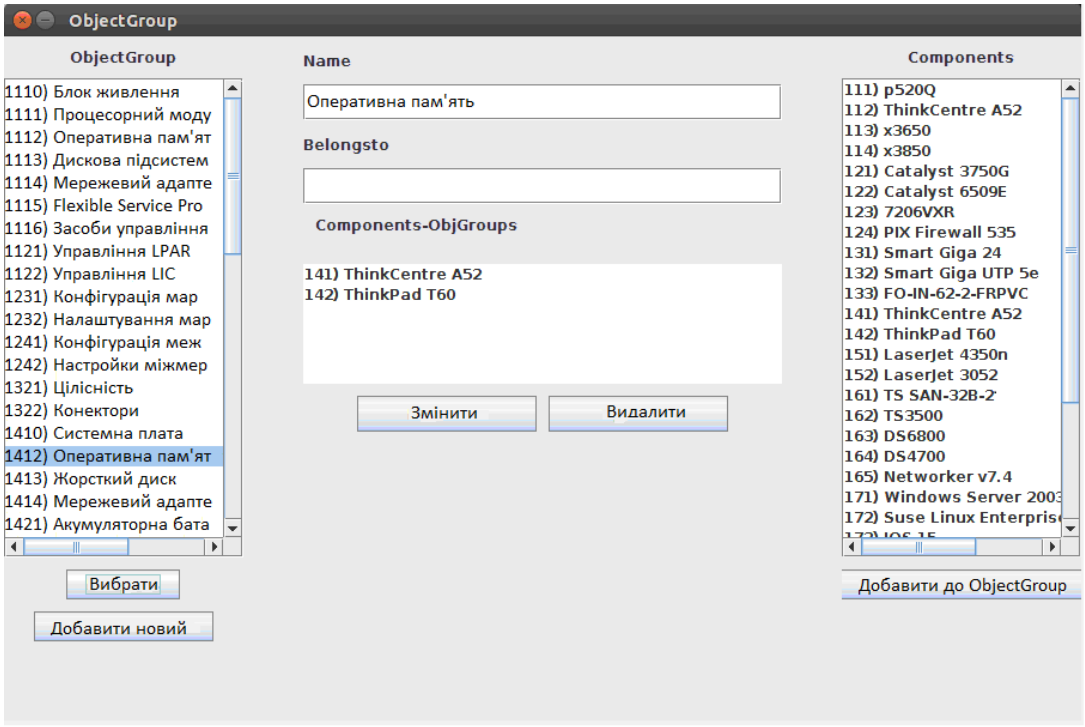

Рисунок 3.9 – Вікно ObjectGroup

Щоб додати новий ObjectGroup потрібно натиснути кнопку "Додати новий" і заповнити всі поля. Після чого необхідно натиснути на кнопку додати. Щоб до ObjectGroup додалися відносини Component-ObjectGroup потрібно вибрати Component справа в списку і натиснути на "Додати до ObjectGroup", після натискання кнопки в списку Component-ObjectGroup з'явиться нове вставлення, яке стосується форми введення.

Якщо ви хочете відмінити відношення Component-ObjectGroup потрібно натиснути на нього і з'явиться кнопка над списком "Прибрати Component-ObjectGroup", яку потрібно натиснути, щоб прибрати це відношення зі списку.

Щоб змінити вже існуючий ObjectGroup потрібно знайти цей ObjectGroup в списку ліворуч (список ObjectGroups) натиснути на знайдений ObjectGroup і натиснути на кнопку "Вибрати". Після натискання на кнопку "Вибрати" всі параметри даного ObjectGroup з'являться у відповідних полях, а відносини Component-ObjectGroup в списку. Після зміни параметрів ObjectGroup і його зв'язків потрібно натиснути на кнопку "Змінити".

Щоб видалити вже існуючий ObjectGroup потрібно знайти цей ObjectGroup в списку ліворуч (список ObjectGroups) натиснути на знайдений ObjectGroup і натиснути на кнопку "Вибрати". Після натискання на кнопку "Вибрати" потрібно натиснути на кнопку "Видалити", після чого з'явиться вікно підтвердження видалення запису де потрібно натиснути на кнопку "Yes". На рисунку 3.10 показаний скріншот графічного інтерфейсу, який відповідає за редагування Object.

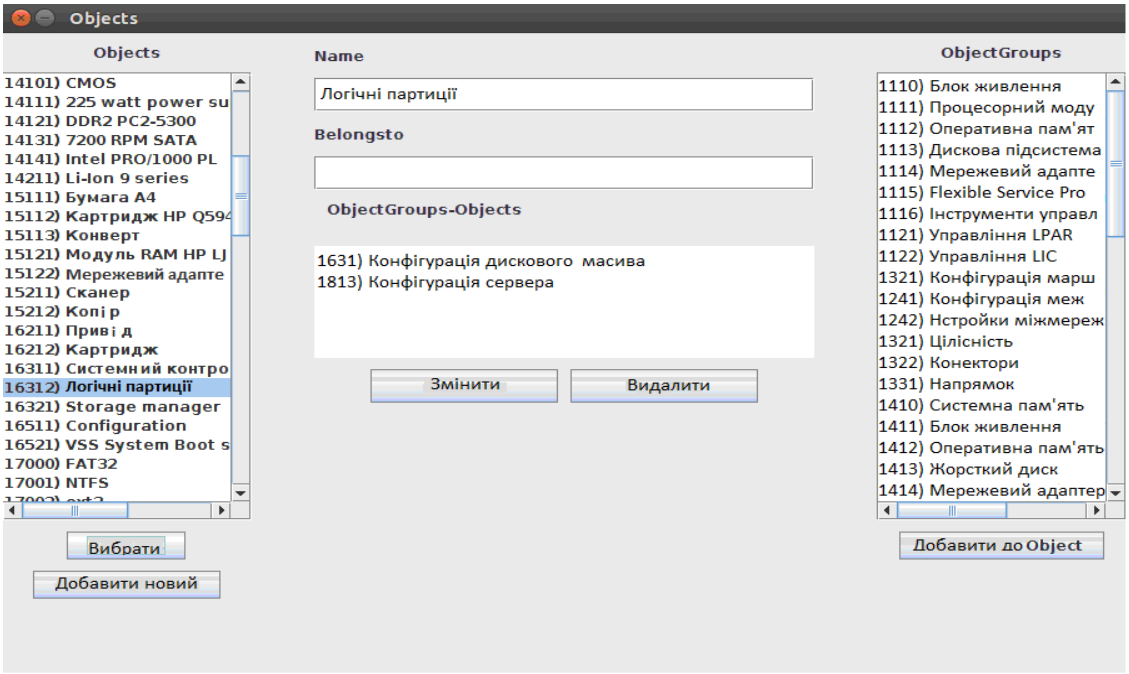

Рисунок 3.10 – Вікно Objects

Щоб додати новий Object потрібно натиснути кнопку "Додати новий" і заповнити всі поля. Після чого необхідно натиснути на кнопку додати. Щоб до Object додати відносини ObjectGroup-Object потрібно вибрати Component справа в списку і натиснути на "Додати до Object", після натискання кнопки в списку ObjectGroup-Object з'явиться нове ставлення, яке стосується форми введення.

Якщо ви хочете відмінити відношення ObjectGroup-Object потрібно натиснути на нього і з'явиться кнопка над списком "Прибрати ObjectGroup-Object", яку потрібно натиснути, щоб прибрати це відношення зі списку.

Щоб змінити вже існуючий Object потрібно знайти цей Object в списку

ліворуч (список Objects) натиснути на знайдений Object і натиснути на кнопку "Вибрати". Після натискання на кнопку "Вибрати" всі параметри даного Object з'являться у відповідних полях, а відносини ObjectGroup-Object в списку. Після зміни параметрів Object і його зв'язків потрібно натиснути на кнопку "Змінити".

Щоб видалити вже існуючий Object потрібно знайти цей Object в списку ліворуч (список Objects) натиснути на знайдений Object і натиснути на кнопку "Вибрати". Після натискання на кнопку "Вибрати" потрібно натиснути на кнопку "Видалити", після чого з'явиться вікно підтвердження видалення запису де потрібно натиснути на кнопку "Yes".

Щоб додати новий Incident потрібно натиснути кнопку "Додати новий" і заповнити всі поля. Після чого необхідно натиснути на кнопку додати. Щоб до Incident додати відносини Incident-Incident потрібно вибрати Incident зліва в списку і натиснути на "Додати до Incident", після натискання кнопки з'явиться вікно для введення параметрів відносини (issuedby, leadto, comment). На рисунку 3.11 нижче представлено діалогове вікно, в яке можна ввести коментар в поле Comment і відзначити значення параметрів issuedby і leadto як true або false.

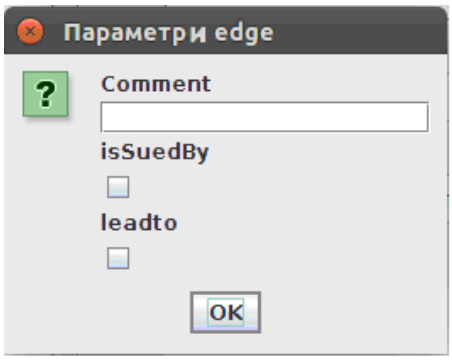

Рисунок 3.11 – Вікно параметры edge

Після введення параметрів в діалоговому вікні потрібно натиснути на кнопку "ОК" і в списку Incidents-Incidents, який відноситься до форми введення (під списком Objects-Incidents).

Щоб до Incident додати відносини Object-Incident потрібно вибрати Object справа в списку і натиснути на кнопку "Додати до Incident", після натискання

кнопки з'явиться вікно для введення параметрів відносини (issuedby, affect, comment). На рисунку 3.12 представлено діалогове вікно, в яке можна ввести коментар в поле Comment і відзначити значення параметрів issuedby і affect як true або false.

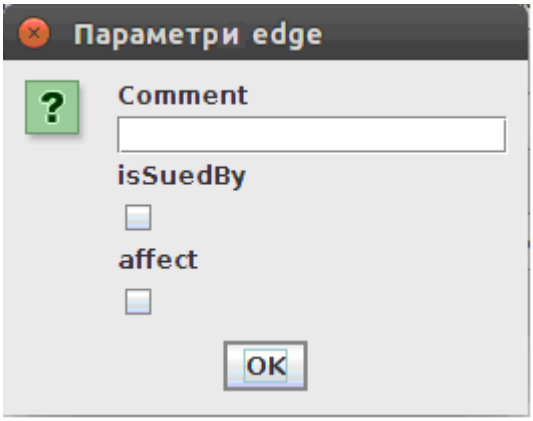

Рисунок 3.12 – Вікно параметры edge

Після введення параметрів в діалоговому вікні потрібно натиснути на кнопку "ОК" і в списку Object-Incident, який відноситься до форми введення (під полем Comment).

Якщо ви хочете відмінити відношення Incident-Incident потрібно натиснути на нього і з'явиться кнопка над списком "Прибрати Incident-Incident", щоб прибрати це відношення зі списку.

Якщо ви хочете відмінити відношення Object-Incident потрібно натиснути на нього і з'явиться кнопка над списком "Прибрати Object-Incident", щоб прибрати це відношення зі списку.

Щоб змінити вже існуючий Incident потрібно знайти цей Incident в списку ліворуч (список Incidents) натиснути знайдений Incident і натиснути на кнопку "Вибрати". Після натискання на кнопку "Вибрати" всі параметри даного Incident з'являться у відповідних полях, а відносини Incident-Incident і Object-Incident у відповідних списках. Після зміни параметрів Incident і його зв'язків потрібно натиснути на кнопку "Змінити".

Щоб видалити вже існуючий Incident потрібно знайти цей Incident в списку ліворуч (список Incidents) натиснути знайдений Incident і натиснути на кнопку "Вибрати" (рисунок 3.14). Після натискання на кнопку "Вибрати" потрібно натиснути на кнопку "Видалити", після чого з'явиться вікно підтвердження видалення запису де потрібно натиснути на кнопку "Yes". На рисунку 3.13 наведено дане вікно.

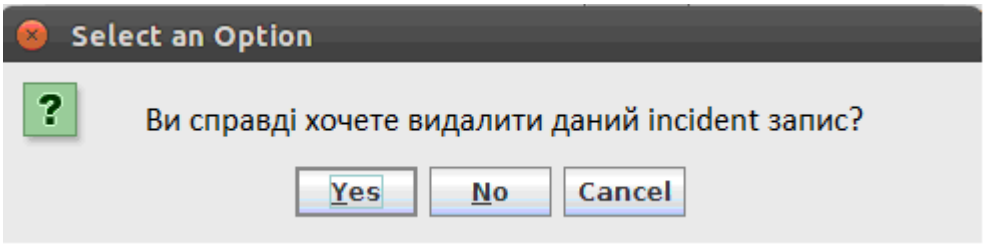

Рисунок 3.13 – Вікно підтвердження видалення

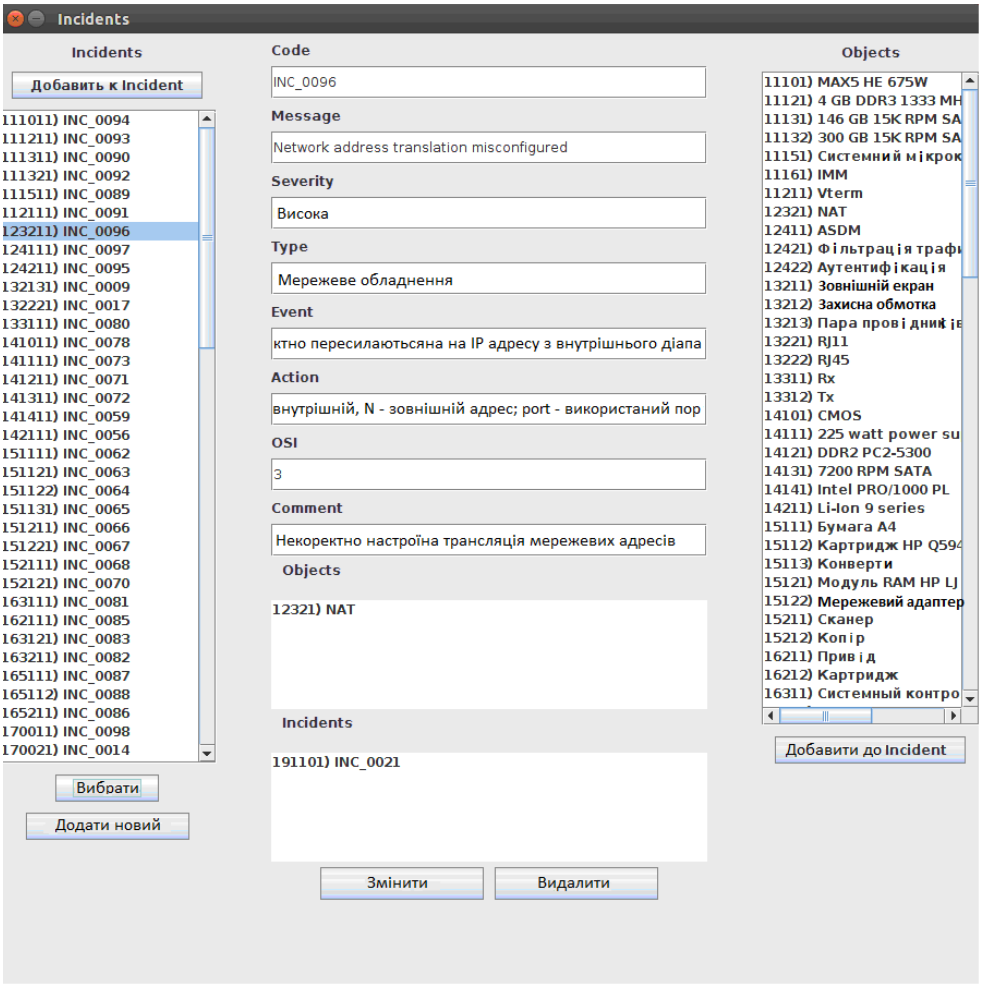

Рисунок 3.14 – Вікно Incidents

На рисунку 3.15 представлено вікно з меню. У цьому меню можна вибрати з яким з 6 можливих об'єктів (Enterprise, Subsystem, Component, ObjGroup, Object, Incident) буде проходити робота.

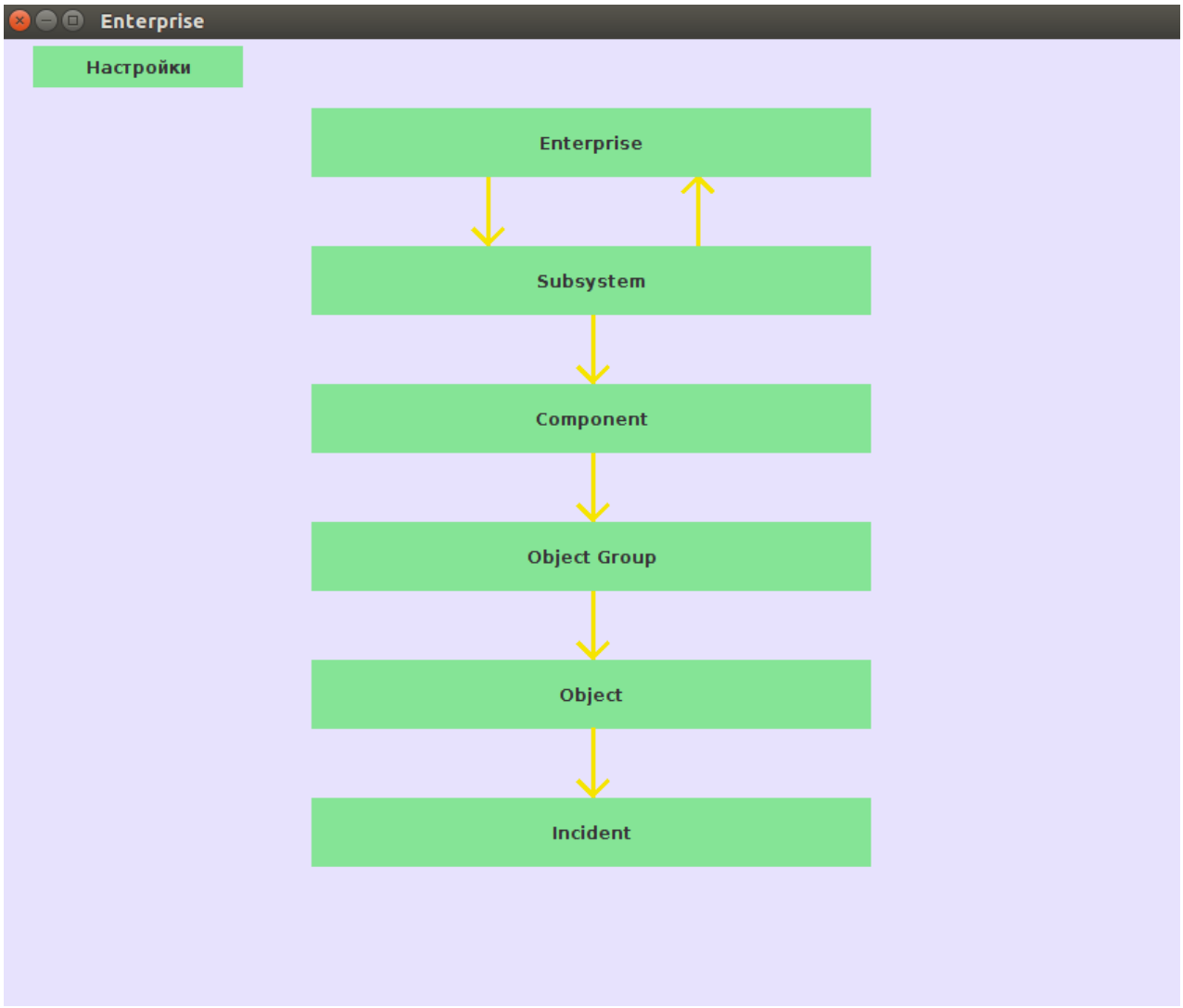

Рисунок 3.15 – Головне меню

Стрілками в меню відзначені відносини між об'єктами. Це буде корисно для користувачів, які тільки почали знайомиться з програмою. У цьому меню також є кнопка, яка викликає вікно налаштувань. На рисунку 3.16 показано як виглядає вікно налаштувань.

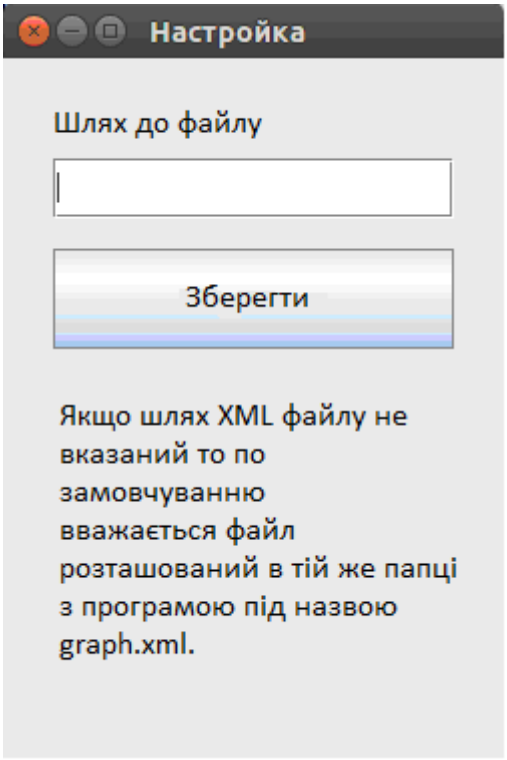

Рисунок 3.16 – Вікно налаштувань

У цьому вікні налаштувань можна ввести шлях до файлу, з яким буде проходити робота. Якщо файл не прописаний в програмі, то за замовчуванням за замовчуванням вважається файл, що знаходиться в тій же папці з програмою з назвою graph.xml.

4 ТЕХНІКО-ЕКОНОМІЧНИЙ РОЗДІЛ

Метою техніко – економічного розділу дипломної роботи є здійснення економічних розрахунків, спрямованих на визначення економічної ефективності програмного модуля редагування структурованих даних заданих у вигляді графа та прийняття рішення про його подальший розвиток і впровадження або ж недоцільність проведення відповідної розробки. Для проведення даного дослідження необхідно провести ряд розразунків.

4.1 Розрахунок витрат на розробку програмного модуля

Витрати на розробку і впровадження програмного модуля для маршрутизації запитів у комп'ютерній мережі (K) включають:

$$K = K_1 + K_2,$$

де K_1 - витрати на розробку апаратного та програмного забезпечення грн.;

K_2 - витрати на відлагодження і дослідну експлуатацію програми рішення задачі на комп'ютері, грн.

Витрати на розробку апаратних та програмних засобів включають:

- витрати на оплату праці розробників ($B_{ОП}$);
- витрати на відрахування у спеціальні державні фонди ($B_ф$);
- витрати на матеріали та комплектуючі ($П_в$);
- накладні витрати (H);
- інші витрати ($I_в$)
- витрати на використання комп'ютерної техніки($B_{КТ}$) .

Витрати на оплату праці включають заробітну плату (ЗП) всіх категорій працівників, безпосередньо зайнятих на всіх етапах проектування. Розмір ЗП обчислюється на основі трудоємності відповідних робіт у людино-днях та середньої ЗП відповідних категорій працівників.

У розробці проектного рішення задіяні наступні спеціалісти - розробники, а саме: керівник проекту; студент-дипломант; консультант техніко-економічного розділу (таблиця 4.1).

Таблиця 4.1 - Вихідні дані для розрахунку витрат на оплату праці

№п/п	Посада виконавців	Місячний оклад, грн.
1	Керівник ДП, викладач	6026
2	Консультант техніко-економічного розділу, доцент	6026
3	Студент	1100

Витрати на оплату праці розробників проекту визначаються за наступною формулою:

$$B_{ОП} = \sum_{i=1}^{N} \sum_{j=1}^{M} n_{ij} \cdot t_{ij} \cdot C_{ij} \ , \qquad (4.1)$$

де n_{ij} – чисельність розробників i-ої спеціальності j-го тарифного розряду, осіб;

t_{ij} – затрачений час на розробку проекту співробітником i-ої спеціальності j-го тарифного розряду, год;

C_{ij} – годинна ставка працівника i-ої спеціальності j-го тарифного розряду, грн.,

Середньогодинна ставка працівника може бути розрахована за такою формулою:

$$C_{ij} = \frac{C_{ij}^{0}(1+h)}{РЧ_i}, \quad (4.2)$$

де C_{ij} – основна місячна заробітна плата розробника *i*-ої спеціальності *j*-го тарифного розряду, грн.;

h – коефіцієнт, що визначає розмір додаткової заробітної плати (при умові наявності доплат);

$РЧ_i$ - місячний фонд робочого часу працівника *i*-ої спеціальності *j*-го тарифного розряду, год. (приймаємо 168 год.).

Коефіцієнт *h*, який визначає розмір додаткової заробітної плати, для керівника та консультанта техніко-економічного розділу дорівнює 0,47.

Результати розрахунку записують до таблиці 4.2.

Таблиця 4.2 - Розрахунок витрат на оплату праці

№ п/п	Посада виконавців	Час розробки, год	Погодинна заробітна плата, грн/год.	Витрати на розробку, грн
1	Керівник ДП, доцент	16	88,6	1417,6
2	Консультант техніко-економічного розділу, доцент	2	88,6	177,2
3	Студент	144	6,55	943,2
Разом				2538

Відрахування на соціальні заходи. Величну відрахувань у спеціальні державні фонди визначають у відсотковому співвідношенні від суми основної та додаткової заробітних плат. Згідно діючого нормативного законодавства сума відрахувань у спеціальні державні фонди складає 20,5% від суми заробітної плати: $B_Ф = \frac{20,5}{100} \cdot 2538 = 520,29$ грн.

Загальна сума витрат на матеріальні ресурси (B_M) визначається за формулою:

$$B_M = \sum_{i=1}^{n} K_i \cdot Ц_i, \qquad (4.3)$$

де K_i - витрата *i*-го типу матеріалу, натуральні одиниці вимірювання;

$Ц_i$ - ціна за одиницю *i*-го типу матеріалу, грн.;

i - тип матеріального ресурсу;

n - кількість типів матеріальних ресурсів.

Таблиця 4.3 - Зведені розрахунки матеріальних витрат

№ п/п	Найменування матеріальних ресурсів	Од. виміру	Факт. витрачено матеріалів	Ціна за одиницю, грн.	Сума, грн	Транспортні витрати (10% від суми)	Загальна сума, грн
	Допоміжна література	шт	1	600	600	60	660
	Папір (формат А4)	уп	2	80	160	16	176
	Ручка кулькова	шт	2	10	20	2	22
	Олівець простий	шт	2	10	20	2	22
	Диски CD-R	шт	2	15	30	3	33
	Зошит, 96 арк	шт	1	50	50	5	55
	Тонер для принтера	уп	1	90	90	9	99
	Канцелярскі маркери (червоний, зелений)	шт	2	20	40	4	44
	Р а з о м						1111,00

Витрати на використання комп'ютерної техніки ($В_{КТ}$) включають витрати на амортизацію комп'ютерної техніки, витрати на користування програмним забезпеченням, витрати на електроенергію, що споживається комп'ютером. За даними обчислювального центру ТНЕУ для комп'ютера типу IBM PC/ATX вартість години роботи становить 6 грн. Середній щоденний час роботи на комп'ютері – 2 години. Розрахунок витрат на використання комп'ютерної техніки приведений в таблиці 4.4.

Таблиця 4.4- Розрахунок витрат на використання комп'ютерної техніки

№ п/п	Назва етапів робіт, при виконанні яких використовується комп'ютер	Час використання комп'ютера, год.	Витрати на використання комп'ютера грн.
1	Проведення досліджень та оформлення їх результатів	60	360
2	Оформлення техніко-економічного розділу	8	48
3	Оформлення ДП	12	72
Разом		80	480

Накладні витрати проектних організацій включають три групи видатків: витрати на управління, загальногосподарські витрати, невиробничі витрати. Вони розраховуються за встановленими відсотками до витрат на оплату праці. Середньостатистичний відсоток накладних витрат приймемо 150% від заробітної плати: $H = 1{,}5 \cdot 1860{,}4 = 2790{,}6$ (грн).

Інші витрати є витратами, які не враховані в попередніх статтях. Вони становлять 10% від заробітної плати: $I_B = 1860{,}4 \cdot 0{,}1 = 186{,}04$ (грн).

Витрати на розробку програмного забезпечення складають:

$$K_1 = B_{ОП} + B_Ф + B_M + H + I_B + B_{КТ},$$

$$K_1 = 1860{,}4 + 381{,}38 + 1111{,}00 + 2790{,}6 + 186{,}04 + 480{,}00 = 6809{,}42 \text{ (грн)}.$$

Витрати на відлагодження і дослідну експлуатацію програмного продукту визначаємо за формулою:

$$K_2 = S_{м.г.} \cdot t_{від} \qquad (4.4)$$

де $S_{м.г.}$ - вартість однієї машино-години роботи ПК, грн./год;

$t_{від}$ - комп'ютерний час, витрачений на відлагодження і дослідну експлуатацію створеного програмного продукту, год.

Загальна кількість днів роботи на комп'ютері дорівнює 30 днів. Середній щоденний час роботи на комп'ютері – 2 години. Вартість години роботи комп'ютера дорівнює 6 грн., тому K_2= 6·60 = 360 грн.

4.2 Визначення експлуатаційних витрат

Для оцінки економічної ефективності розроблювальної системи моніторингу слід порівняти її з аналогом, тобто існуючим програмним забезпеченням ідентичного функціонального призначення.

Експлуатаційні одноразові витрати по програмному забезпеченню і аналогу включають вартість підготовки даних і вартість роботи комп'ютера (за час дії програми):

$$E_П = E_{1П} + E_{2П},$$

де $E_П$ - одноразові експлуатаційні витрати на ПЗ (аналог), грн.;

$E_{1П}$ - вартість підготовки даних для експлуатації ПЗ (аналогу), грн.;

$E_{2П}$ - вартість роботи комп'ютера для виконання проектного рішення (аналогу), грн.

Річні експлуатаційні витрати $B_{ЕП}$ визначаються за формулою:

$$B_{ЕП} = E_П * N_П,$$

де $N_П$ - періодичність експлуатації ПЗ (аналогу), раз/рік.

Вартість підготовки даних для роботи на комп'ютері визначається за формулою:

$$E_{1П} = \sum_{l=1}^{n} n_i t_i c_i ,$$

де *i* - категорії працівників, які приймають участь у підготовці даних (i=1,2,...n);

n_i - кількість працівників *i*-ої категорії, осіб.;

t_i - трудомісткість роботи співробітників *i*-ої категорії по підготовці даних, год.;

c_i - середнього годинна ставка працівника *i*-ої категорії з врахуванням додаткової заробітної плати, що знаходиться із співвідношення:

$$c_i = \frac{c_i^0 (1+b)}{m},$$

де c_i^0 - основна місячна заробітна плата працівника *i*-ої категорії, грн.;

b - коефіцієнт, який враховує додаткову заробітну плату (приймемо 0,57);

m - кількість робочих годин у місяці, год.

Для роботи з даними як для проектного рішення так і аналогу потрібен один працівник, основна місячна заробітна плата якого складає: *c* = 3723 грн. Тоді:

$$c_1 = \frac{3723(1+0{,}57)}{22*8} = 33{,}21 \text{грн/год}$$

Трудомісткість підготовки даних для проектного рішення складає 1 год., для аналога 1,5 год.

Таблиця 4.5- Розрахунок витрат на підготовку даних та реалізацію проектного рішення на комп'ютері

№	Час роботи співробітників, год.	Середньогодинна заробітна плата, грн./год.	Витрати , грн.
Проектне рішення			
1	1	33,21	33,21
Аналог			
2	1,5	33,21	66,42

Витрати на експлуатацію комп'ютера визначається за формулою:

$$E_{2П} = t * S_{МГ}$$

де t - витрати машинного часу для реалізації рішення (аналогу), год.;

$S_{МГ}$ - вартість однієї години роботи комп'ютера, грн./год.

Далі:

$E_{2П}$ = 1* 6 = 6 грн.; $E_{2А}$=1,5*6 = 9 грн.

$E_{П}$=33,21+6=39,21 грн. ; $E_{А}$=66,42+9 = 75,42 грн.

$B_{ЕП}$= 39,21*252=9880,92 грн. ; $B_{ЕА}$=75,42 *252=19005,84 грн.

Накладні витрати пов'язані з обслуговуванням виробництва, утриманням апарату управління підприємства (фірми) та створення необхідних умов праці.

В залежності від організаційно-правової форми діяльності господарюючого суб'єкта, накладні витрати можуть становити 60–100 % від суми основної та додаткової заробітної плати працівників.

$$H_B = 0,7 * B_{ОП}, \qquad (4.5)$$

де H_B – накладні витрати.

H_B=0,7*5845,11=4091,58 грн.

Результати проведених розрахунків зведемо у таблицю 4.6.

Таблиця 4.6 - Кошторис витрат

№ п/п	Найменування витрат	Сума витрат, грн.
1	2	3
1	Витрати на оплату праці	1860,4

Продовження таблиці 4.6

1	2	3
2	Відрахування у спеціальні державні фонди	381,38
3	Витрати на матеріали та комплектуючі	1111,00
4	Накладні витрати на розробку	2790,6
5	Інші витрати	186,04
6	Витрати на відлагодження і дослідну експлуатацію програмного продукту	360
7	Накладні витрати експлуатацію	4091,58
8	Річні експлуатаційні витрати	19005,84
Разом		29786,09

Договірна ціна ($Ц_Д$) для проектних рішень розраховується за формулою:

$$Ц_Д = B_{КС} \cdot \left(1 + \frac{p}{100}\right), \quad (4.6)$$

де $B_{КС}$ – кошторисна вартість, грн.;

p - середній рівень рентабельності, % (приймаємо 26% за погодженням з керівником): $Ц_Д = 29786,09 \cdot (1+0,26) = 37530,47$ грн.

4.3 Визначення економічної ефективності і терміну окупності капітальних вкладень

Економічна ефективність ($E_ф$) полягає у відношенні результату виробництва до затрачених ресурсів:

$$E_ф = \frac{П}{B_{КС}}, \quad (4.7)$$

де $П$ – прибуток, грн.;

$B_{КС}$ – кошторисна вартість, грн..

$$E_{ф}=7812,27 \text{ грн.} / 29786,09 \text{ грн.} = 0,25.$$

Поряд із економічною ефективністю розраховують термін окупності капітальних вкладень (Tp):

$$T_P = \frac{1}{E_P} \ . \qquad (4.8)$$

Тобто: T_P =1/0,25 = 4р.

Прийнятним вважається термін окупності, близький до 7 років.

Розраховані економічні показники проекту занесемо до таблиці 4.7.

Таблиця 4.7 - Економічні показники розробки

№ п/п	Показник	Значення
1.	Собівартість, грн.	29786,09
2.	Плановий прибуток, грн.	7744,38в
3.	Ціна, грн.	37530,47
4.	Економічна ефективність	0,25
5.	Термін окупності, рік	4

Враховуючи основі економічні показники з таблиці 4.7, можна зробити висновок, що при економічній ефективності 0,25 та терміні окупності 4 роки проводити роботи по впровадженню даного програмного модуля є доцільним та економічно вигідним.

ВИСНОВКИ

1. У цій випускній кваліфікаційній роботі було проведено аналіз існуючого програмного забезпечення для редагування структурованих даних і на його основі було вирішено розробити програмне забезпечення, яке дає можливість редагувати опис інформаційної системи, яка представлена у вигляді графа.

2. Спроектовано графічний інтерфейс програмного забезпечення і спроектована схема бази даних. До початку розробки були обрані всі необхідні інструменти і технології.

3. Після розробки даного програмного продукту було проведено його тестування. В результаті було повністю виконано технічне завдання по випускній кваліфікаційної роботі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Schloegel K. Partitioninng for High Performance Scientific Simulations.// K. Schloegel, G. Karypis, V. Kumar / Minneapolis, Minnesota: Dept. of Computer Science and Engineering, University of Minnesota, 1999.– P. 198-207.

2. Big Data (биг дата) - обработка больших объемов данных [Електронний ресурс] // DIS - Дистрибуция и внедрение инновационных продуктов и решений для корпоративного сектора от лидеров мирового ИТ-рынка: [сайт]. [2014]. URL: http://www.disgroup.ru/solutions/data_management/big_data/ (дата звернення: 05.03.2018).

3. David Kotz's papers (by topic) and research summaries // Dartmouth. Department of Computer Science. 1998. URL: http://www1.cs.dartmouth.edu/~dfk/papers/kotz:encyc1.pdf (дата звернення: 01.04.2018).

4. Введение в OpenMP: параллельное программирование на C++ [Електронний ресурс] // Intel. Developer Zone.: [сайт]. [2011]. URL: https://software.intel.com/ru-ru/blogs/2011/11/21/openmp-c (дата звернення: 03.04.2018).

5. Overview [Електронний ресурс] // Apache Hama: [сайт]. [2016]. URL: https://hama.apache.org/index.html (дата обращения:05.05.2018).

6. Message Passing Interface (MPI) [Електронний ресурс] // High Performance Computing: [сайт]. [2016]. URL: https:// computing.llnl.gov/tutorials/mpi (дата обращения: 04.04.2018).

7. Pace M.F. BSP vs MapReduce // Procedia Computer Science. Sep 2012. – Т. 1. – № 1.– P. 246 – 255.

8. Dean J. The 6th Conference on Symposium on Operating Systems Design & Implementation // J. Dean, S. Ghemawat/ MapReduce: Simplified Data Processing on Large Clusters. New York. –2004. –Т. 6. –P. 137–150.

9. About [Електронний ресурс] // Apache Hadoop: [сайт]. [2016]. URL: http://hadoop.apache.org/ (дата звернення: 16.03.2018).

10. Anjani A.C.P. An Efficient Framework for Large Scale Data Analytics // A.C.P. Anjani, M. Sridhar, D. Spark/ International Journal of Scientific & Engineering Research.– Т. 7 –№ 2, Feb 2016.– P. 401-405.

11. Stanford Large Network Dataset Collection [Електронний ресурс] // Stanford Network Analysis Project: [сайт]. [2016]. URL: http:/ /snap.stanford.edu/data/ (дата звернення: 07.02.2018).

12. Брюс Э. Философия Java 4-е изд.: СПб.:Питер, 2016.–383 с.

13. Программирование на JAVA, пакет java.awt. [Електронний ресурс] // URL: http://kufas.ru/programming_java196.htm (дата звернення 09.04.2018).

14. Введение в библиотеку Swing. [Електронний ресурс] // URL: http://math.sgu.ru/sites/chairs/prinf/materials/java/lesson8.htm (дата звернення 06.04.2018).

15. Библиотека SWT (Standard Widget Toolkit). [Електронний ресурс] // URL: http://java-online.ru/libs-swt.xhtml (дата звернення 06.04.2018).

16. Язык XML - практическое введение. [Електронний ресурс] // URL: http://www.codenet.ru/webmast/xml/ (дата зверненняя 07.04.2018).

17. Parsing XML using DOM, SAX and StAX Parser in Java. [Електронний ресурс] // URL: https://sanaulla.info/2013/05/23/parsing-xml- using-dom-sax-and-stax-parser-in-java/ (дата звернення 10.04.2018).

18. Java DOM XML Parser. [Електронний ресурс] // URL: http://javism.blogspot.ru/2012/01/java-dom-xml-parser-dom-java-xml.html (дата звернення 04.04.2018).

19. Паттерн Singleton (одиночка, синглет). [Електронний ресурс] // URL: http://cpp-reference.ru/patterns/creational-patterns/singleton/ (дата звернення 04.04.2018).

20. Модульное тестирование с jUnit4. [Електронний ресурс] // URL: http://www.quizful.net/post/jUnit4_common_information (дата звернення 04.04.2018).

21. SQLite — замечательная встраиваемая БД (часть 1). [Електронний ресурс] // URL: https://habrahabr.ru/post/149356/ (дата звернення 09.04.2018).

22. Емеличев В.А. Лекции по теории графов / В.А. Емеличев и др. - М.: Наука, 1990. - 384 с.

23. Мельников О.И. Занимательные задачи по теории графов / О.И. Мельников. - М.: Мир, 2001. - 195 с.

24. Майника Э. Алгоритмы оптимизации на сетях и графах. – М.: Мир, 1981.– 239 с.

25. Уилсон, Р. Введение в теорию графов / Р. Уилсон. - М.: Книга по Требованию, 2012. - 207 с.

26. Емеличев В. А. Лекции по теории графов/ Емеличев В. А. и др. — М.: Наука, Гл. ред. физ.- мат. лит., 1980. — 336 с.

27. Оре О. Теория графов. — 2-е издание. М.: Наука, Гл. ред. физ.-мат. лит., 1990. — 384 с.

28. Трохимчук Р.М. Збірник задач з теорії графів. Навчальний посібник для студентів факультету кібернетики. К.: РВЦ “Київський університет”, 1998. — 43 с.

29. Харари Ф. Теория графов. М.: Мир, 1973. — 300 с.

30. Методичні рекомендації до виконання дипломного проекту з освітньо-кваліфікаційного рівня “Бакалавр” напряму підготовки 6.050102 «Комп’ютерна інженерія» фахового спрямування «Комп’ютерні системи та мережі» / О.М. Березький, Л.О.Дубчак, Р.Б. Трембач, Г.М. Мельник, Ю.М. Батько, С.В. Івасьєв / Під ред. О.М. Березького. - Тернопіль: ТНЕУ, 2016.–65 с.

31. Методичні вказівки до написання техніко-економічного розділу для дипломних проектів на здобуття освітньо-кваліфікаційного рівня “Бакалавр” напряму підготовки 6.050102 «Комп’ютерна інженерія» / І.Р.Паздрій. - Тернопіль: ТНЕУ, 2015.– 36 с.