Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

АТЕСТАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Методи проектування програмного та інформаційного забезпечення об'єктно-орієнтованих баз даних

(тема)

Виконав:
студент II курсу, групи СПм-19-1
Гонтарєва Д. В.
(прізвище, ініціали)

Спеціальність
123 – Комп'ютерна інженерія
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Системне програмування
(повна назва освітньої програми)

Керівник: доц. Філімончук Т.В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2020 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 – Комп'ютерна інженерія
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20___ р.

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

студентові Гонтарєвій Дар'ї Владиславівні
(прізвище, ім'я, по батькові)

1. Тема роботи Методи проектування програмного та інформаційного забезпечення об'єктно-орієнтованих баз даних

затверджена наказом по університету від "30" жовтня 2020 р. № 1486 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 14 грудня 2020 р.

3. Вхідні дані до роботи Unified Modeling Language (UML);
Object Definition Language (ODL);
Object Query Language (OQL).

4. Перелік питань, що потрібно опрацювати в роботі Об'єктно-орієнтована парадигма;
Загальні поняття об'єктно-орієнтованих баз даних;
Об'єктно-орієнтовані системи управління базами даних;
Функціональна схема проектування програмного та інформаційного забезпечення об'єктно-орієнтованих баз даних;
Логічне моделювання об'єктно-орієнтованих баз даних;
Фізичне моделювання об'єктно-орієнтованих баз даних.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) Демонстраційні матеріали. Плакати – 14 арк. ф. А4

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Постановка задачі	3.11.2020	
2	Ознайомлення з літературними джерелами	4.11.2020-6.11.2020	
3	Визначення основних принципів ООП	9.11.2020-11.11.2020	
4	Визначення загальних понять ООБД	12.11.2020-13.11.2020	
5	Порівняння проблем, які виникають в	16.11.2020-18.11.2020	
6	реляційних та об'єктно-орієнтованих БД		
	Створення функціональної схеми	19.11.2020-25.11.2020	
7	Логічне моделювання ООБД	26.11.2020-30.11.2020	
8	Фізичне моделювання ООБД	1.12.2020-4.12.2020	
9	Оформлення пояснювальної записки	7.12.2020-11.12.2020	

Дата видачі завдання 2 листопада 2020 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Філімончук Т.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка атестаційної роботи: 78 с., 8 рис., 1 табл., 1 дод., 16 джерел.

ОБ’ЄКТНО-ОРІЄНТОВАНА ПАРАДИГМА, БАЗА ДАНИХ, ОБ’ЄКТНО-ОРІЄНТОВАНА БАЗА ДАНИХ, КЛАСС, АТРИБУТ, ОБ’ЄКТ, ПРОЕТУВАННЯ ОБ’ЄКТНО-ОРІЄНТОВАНОЇ БАЗИ ДАНИХ

Метою атестаційної роботи є дослідження методів проектування та інформаційного забезпечення об’єктно-орієнтованих баз даних, з’ясування того, як ведення об’єктно-орієнтованої бази даних вплине на процес розробки програмного забезпечення, які переваги та недоліки будуть виявлені.

У ході виконання атестаційної роботи було розглянуто що таке об’єктно-орієнтована парадигма програмування, які популярні об’єктно-орієнтовані системи управління базами даних існують, розглянуті основні етапи проектування об’єктно-орієнтованої бази даних, розроблена модель проектування програмного та інформаційного забезпечення, фізична модель ООБД.

ABSTRACT

Master`s thesis 78 pages, 8 figures, 1 table, 1 appendice, 16 sources.

OBJECT-ORIENTED PARADIGM, DATABASE, OBJECT-ORIENTED DATABASE, CLASS, ATTRIBUTE, OBJECT, OBJECT-ORIENTED DESIGN

The major goal of this thesis is to study the methods of design and information support of object-oriented databases, to find out how maintaining an object-oriented database will affect the software development process, what advantages and disadvantages will be identified.

During the attestation work it was considered what is an object-oriented programming paradigm, what popular object-oriented database management systems exist, the main stages of object-oriented database design are considered, a software and information design model is developed, a physical model OOBD.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – база даних

ООБД – об'єктно-орієнтована база даних

ООМП – об’єктно-орієнтована мова програмування

ООП – об’єктно-орієнтоване програмування

ООСУБД – об’єктно-орієнтована система управління базою даних

СУБД – система управління базою даних

API – Application Programming Language

ASCII – American standard code for information interchange

ODL – Object Definition Language

ODMG – Object Database Management Group

OQL – Object Query Language

SQL – Structured Query Language

UML – Unified Modeling Language

ВСУП

На сьогоднішній день комп'ютерні системи використовуються у все більш різноманітних областях та додатках. Вони стають поширенішими і виконують різноманітні задачі майже в усіх сферах нашого життя. Для того, щоб задовольнити всі наші потреби, комп'ютерні системи повинні зберігати великий обсяг інформації. Необхідно, щоб ця інформація зберігалася та була структурована таким чином, аби приносити користь і для самої системи, і, так само, для людей, які проектують та працюють з системою. Така структурована інформація називається даними і зазвичай її постійно зберігають в базі даних (БД).

Існують визначення БД, які отримано із міжнародних стандартів та національних стандартів, що були розроблені на основі міжнародних:

- база даних – це сукупність даних, які зберігаються у відповідності зі схемою даних, маніпулювання якими виконують відповідно до правил засобів моделювання даних [3];
- база даних – це сукупність даних, які організовані відповідно до концептуальної структури, що описує характеристики цих даних та взаємини між ними, та яка підтримує одну або більше областей застосування.

Існує безліч типів баз даних, навіть простий список на аркуші паперу можна вважати базою даних, але щоб бути доступною для комп'ютерної системи, вона повинна зберігатися в електронному вигляді. На ринку існують різні системи баз даних, від невеликих рішень для персональних комп'ютерів до величезних систем, які призначено для роботи на мейнфреймах чи кластерах комп'ютерів, які доступні майже для всіх через Інтернет.

Комп’ютерні системи, які використовуються в теперішній час, у більшості випадків розроблені в тій чи іншій формі об'єктно-орієнтованої мови програмування [1]. Часто це виходить дуже добре, бо реальний світ у багатьох випадках легко сприймається як об'єктно-орієнтована парадигма,

але, незважаючи на це, є проблема, яка виникає з розвитком цих комп’ютерних систем. Цією проблемою є бази даних.

Бази даних та системи програмного забезпечення використовують різні підходи. Найчастіше метод, який застосовується при проектуванні бази даних, не є об'єктно-орієнтованим [14]. Це призводить до кількох проблем з розробкою комп'ютерних систем, що використовуються базами даних. Деякі з цих проблем полягають в тому, що існують дві різні моделі однакових даних: одна для бази даних і одна для комп'ютерної системи, що застосовує її. Використання двох різних моделей для одних і тих самих даних, суперечить здоровому глузду. Об'єкти даних повинні бути переведені між двома моделями як при моделюванні та розробці системи, так і під час її роботи. Це трудомісткий процес, що використовує багато обчислювальних потужностей під час роботи.

Один з варіантів вирішення цих проблем – введення відносно нової методики розробки та впровадження бази даних: зробити базу даних також об'єктно-орієнтованою. Використовуючи об'єктно-орієнтовану базу даних, можна вирішити більшість існуючих проблем.

Метою атестаційної роботи є дослідження методів проектування та інформаційного забезпечення об’єктно-орієнтованих баз даних, з’ясування того, як ведення об’єктно-орієнтованої БД вплине на процес розробки програмного забезпечення, які переваги та недоліки будуть виявлені.

Найбільш центральні питання, що будуть розглянуті, стосуються того, що таке об’єктно-орієнтована парадигма програмування, які популярні об’єктно-орієнтовані системи управління базами даних існують. Будуть розглянуті основні етапи проектування об’єктно-орієнтованої бази даних, розроблена модель проектування програмного та інформаційного забезпечення, фізична модель ООБД. Буде з’ясовано, чи завжди використання об’єктно-орієнтованої бази даних позитивно впливає на комп’ютерну систему, чи існують випадки, коли цей вплив відсутній або можливо негативний.

1 ОСНОВНІ ПРИНЦИПИ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ

Об'єктно-орієнтоване програмування (ООП) – це методологія програмування, яка заснована на представленні програми у вигляді сукупності об'єктів, кожен з яких є екземпляром певного класу, а класи утворюють ієрархію наслідування [10].

Об'єктно-орієнтований підхід до програмування надає наступні переваги:

- зменшення складності програмного забезпечення;
- підвищення надійності програмного забезпечення;
- забезпечення можливості модифікації окремих компонентів програмного забезпечення без зміни інших його компонентів;
- забезпечення можливості повторного використання окремих компонентів програмного забезпечення.

ООП засноване на «чотирьох китах» – чотирьох найважливіших принципах, які надають об'єктам нові властивості. Цими принципами є інкапсуляція, унаслідування, поліморфізм та абстракція.

1.1 Концепції об’єктно-орієнтованого програмування

Одним з визначальних факторів при проектуванні компонентів програми є приховування внутрішніх даних компоненту та деталей його реалізації від інших компонентів програми та надання набору методів для взаємодії з ним (API). Цей принцип є одним з фундаментальних принципів ООП і називається інкапсуляцією [2].

Правильна інкапсуляція має велике значення з багатьох причин:

- вона сприяє повторному використанню компонентів: оскільки в цьому випадку компоненти взаємодіють між собою лише через їх API та нечутливі до змін внутрішньої структури, вони можуть використовуватись в

більш широкому контексті;

- інкапсуляція пришвидшує процес розробки: слабко пов'язані один з одним компоненти (тобто компоненти, чий код якомога менше звертається або використовує код інших компонентів) можуть розроблятися, тестуватися та доповнюватися незалежно;

- правильно інкапсульовані компоненти більш зрозумілі та легше налагоджуються, що спрощує підтримку програми.

Наслідування – це концепція ООП, згідно з якою абстрактний тип даних може успадковувати дані та функціональність деякого існуючого типу, сприяючи повторному використанню компонентів програмного забезпечення.

В об'єктно-орієнтованому програмуванні, починаючи з Simula 67, абстрактні типи даних називаються класами [9].

Суперклас (батьківський клас, предок або надклас) – це клас, від якого успадковуються інші класи. Суперкласом може бути підклас, базовий клас, абстрактний клас та інтерфейс.

Підклас (похідний клас, дочірній клас, клас нащадок, клас спадкоємець) – це клас, який успадкований від суперкласу або інтерфейсу. Підкласом може бути суперклас.

Базовий клас – це клас, що знаходиться на вершині ієрархії успадкування класів та в підставі дерева підкласів, тобто не є підкласом і не має наслідувань від інших суперкласів або інтерфейсів. Базовим класом може бути абстрактний клас та інтерфейс. Будь-який не базовий клас є підкласом.

Інтерфейс – це структура, яка визначає чистий інтерфейс класу, що складається з абстрактних методів. Інтерфейси беруть участь в ієрархії наслідувань класів та інтерфейсів [11].

Використовуючи наслідування, можна створити загальний клас, який буде визначати характеристики та поведінку, властиві певному набору пов'язаних об'єктів. В подальшому цей клас може унаслідуватися іншими, другорядними класами, кожен з яких додаватиме унікальні, властиві лише

йому характеристики та доповнюватиме або змінюватиме поведінку базового класу.

Наслідування є одним з найвагоміших принципів об'єктно-орієнтованого програмування, оскільки воно дозволяє створювати ієрархічні структури об'єктів. Застосування ієрархії класів робить керованими великі потоки інформації.

Наприклад, розглянемо опис першокурсника. «Першокурсник» – це частина загального класу «Студент». З іншого боку, студент – це частина більш загального класу-конструкції, який є частиною ще більш загального класу об'єктів, який можна назвати «Людина». У кожному разі породжений клас успадковує всі, пов'язані з батьківським класом, якості та додає до них свої власні визначальні характеристики. Без використання ієрархії класів, для кожного об'єкта довелося б задавати всі характеристики, які б вичерпно його визначали. Однак при використанні унаслідування можна описати об'єкт шляхом визначення того загального класу (або класів), до якого він належить, з тими спеціальними рисами, які роблять об'єкт унікальним.

Розглядаючи поліморфізм необхідно пам'ятати, що цей принцип нерозривно пов'язаний з іншим принципом ООП – наслідуванням, який допомагає реалізувати поліморфізм. Розглянемо абстрактний клас «Автомобіль», який наслідують два конкретних класи: «Спортивний автомобіль» та «Вантажний автомобіль».

І спортивні, і вантажні автомобілі володітимуть спільними характеристиками та матимуть можливість виконувати загальні для всіх автомобілів дії, вказані в абстрактному батьківському класі, але конкретна реалізація цих дій може бути різною.

Наприклад, загальна для всіх автомобілів дія «завестись» у спортивному автомобілі може бути реалізована шляхом натискання кнопки, а у вантажного – за допомогою ключа. Тобто: один результат – різні рішення. В цьому і полягає поліморфізм.

Більш точно, поліморфізм – це один з принципів ООП, який дозволяє

викликом перевизначеного методу через змінну батьківського класу отримати поведінку, яка буде відповідати реальному похідному класу, на який посилається ця змінна [2].

Відносно нещодавно абстракцію почали виділяти, як самостійний четвертий принцип. Абстракція – це теоретичний прийом дослідження, який дозволяє відсторонитися від деяких несуттєвих, у певному сенсі, властивостей явищ, які досліджуються, і виокремити суттєві та визначальні властивості [12].

Всі мови програмування пропонують користувачу деякі абстракції. Так мови групи асемблер є свого роду абстракцією відповідних мікропроцесорів, оскільки дозволяють відволіктися від деталей їх реалізації та звертатися до них через набір більш високорівневих інструкцій. Імперативні мови програмування, які з'явилися після асемблеру, наприклад Basic, Fortran, C, є більш високим рівнем абстракції над асемблерними мовами – вони дають можливість використовувати більш звичні для людини синтаксичні конструкції за рахунок наближення синтаксису до природних мов. Об'єктно-орієнтовані мови, такі як Java, виводять розробку на ще вищий рівень абстракції: об'єкти в ООП являють собою моделі понять оточуючого світу, такі як «Працівник», «Сервер», «Запис в щоденнику», і виділяють лише властивості цих понять, які необхідні в конкретному випадку для вирішення конкретної проблеми.

2 ЗАГАЛЬНІ ПОНЯТТЯ ОБ’ЄКТНО-ОРІЄНТОВАНИХ БАЗ ДАНИХ

Об'єктно-орієнтована база даних (ООБД) – це база даних, в якій дані моделюються у вигляді об'єктів, їх атрибутів, методів та класів.

Об'єктно-орієнтовані бази даних зазвичай рекомендовані для тих випадків, коли потрібна високопродуктивна обробка даних, що мають складну структуру.

У маніфесті ООБД пропонуються характеристики [8], яким має відповідати будь-яка база даних. Їх вибір заснований на 2 критеріях: система повинна бути об'єктно-орієнтованої та являти собою базу даних. Серед усіх характеристик можна відокремити ті, наявність яких обов’язкова:

- підтримка складних об'єктів: в системі повинна бути передбачена можливість створення складових об'єктів за рахунок застосування конструкторів складових об'єктів. Необхідно, щоб конструктори об'єктів були ортогональні, тобто будь-який конструктор можна було застосовувати до будь-якого об'єкту;

- підтримка індивідуальності об'єктів: всі об'єкти повинні мати унікальний ідентифікатор, який не залежить від значень їх атрибутів;

- підтримка інкапсуляції: коректна інкапсуляція досягається за рахунок того, що програмісти мають право доступу тільки до специфікації інтерфейсу методів, а дані та реалізація методів приховані всередині об'єктів;

- підтримка типів і класів: потрібно, щоб в ООБД підтримувалася хоча б невеликі відмінності між типами та класами. Термін «тип» в даному випадку більш відповідає поняттю абстрактного типу даних. У мовах програмування змінна оголошується із зазначенням її типу. Компілятор може використовувати цю інформацію для перевірки операцій, що виконуються із змінною, на сумісність з її типом, що дозволяє гарантувати коректність програмного забезпечення. З іншого боку клас є певним шаблоном для створення об'єктів та надає методи, які можуть застосовуватися до об'єктів;

- підтримка успадкування типів та класів від їх предків: підтип або підклас повинен успадковувати атрибути та методи від його супертипа або суперкласу, відповідно;

- перевантаження в поєднанні з повним зв'язуванням: методи повинні застосовуватися до об'єктів різних типів. Реалізація методу повинна залежати від типу об'єктів, до яких даний метод застосовується. Для забезпечення цієї функціональності зв'язування імен методів в системі не повинно виконуватися до часу виконання програми;

- обчислювальна повнота: мова маніпулювання даними повинна бути мовою програмування загального призначення;

- набір типів даних повинен бути розширюваним, тобто користувач повинен мати засоби створення нових типів даних на основі набору зумовлених системних типів. Більш того, між способами використання системних та типів даних, які призначені для користувача, не повинно бути ніяких відмінностей.

Раніше не було прийнято жодного стандарту для проектування об’єктно-орієнтованої моделі даних. Однак зараз існує незалежна група представників індустрії, відома під назвою Object Database Management Group (ODMG), яка працює над пропозиціями по такому стандарту [6].

Голова цього комітету Рік Каттелл пише у вступі до документації про стандарт моделі: «Ми працювали поза рамками традиційних органів по стандартизації, щоб якомога швидше отримати результат» [8]. Версія 2.0 стандарту ООБД даних була випущена в 1997 році, але не передана офіційно в органи стандартизації. Проте більшість основних виробників ООСУБД висловили бажання слідувати цим документом.

2.1 Об’єкт та літерал

Термінологія, що застосовується по відношенню до об’єктно-орієнтованої парадигми, в значній мірі залежить від конкретного

середовища, в якому відбувається робота. В стандарті проектування об’єктно-орієнтованих баз даних використовуються більшість термінів з тих, що давно відомі в об’єктно-орієнтованому програмуванні, але іноді в дещо іншому значенні. Через це краще почати вивчення стандарту проектування зі знайомства з точною термінологією, яка використовується в його документах.

Основними компонентами об’єктно-орієнтованої бази даних є об'єкт та літерал. Об'єкт – це конкретний екземпляр сутності, а літерал – це конкретне значення. Літерал – це конкретне значення, наприклад, слово «собака» або число 123. Літерал не має ідентифікатора, це не обов’язково одне значення. Це може бути і структура – набір взаємопов’язаних значень, що зберігаються під загальним ім’ям.

У об'єктів, як і в реляційній базі даних, є атрибути (властивості) та зв'язки з іншими об'єктами. Множина поточних значень всіх властивостей об’єкта складають його стан.

У всіх об'єктів та літералів є свої типи. Кожний тип має свій власний домен, який є спільним для всіх об'єктів та літералів даного типу. Типи також можуть володіти поведінкою. Якщо тип має деяку поведінку, то такою самою поведінкою будуть володіти всі об'єкти цього типу. На практиці тип може бути класом, з якого створюється об'єкт, інтерфейсом або простим типом даних (наприклад, цілим числом). Об'єкт можна уявити як екземпляр типу.

Будь-який елемент в об’єктно-орієнтованій базі даних побудовано з невеликої групи примітивних типів даних:

- boolean (приймає значення true або false);
- char (один символ в коді ASCII або UNICODE);
- short (ціле число зі знаком, зазвичай довжиною 8 або 16 біт);
- long (ціле число без знака, зазвичай довжиною 32 або 64 біти);
- float (число з плаваючою точкою одинарної точності);
- double (число з плаваючою точкою подвійної точності);
- octet (рівно 8 біт пам’яті);

- string (строка символів);
- enum (тип перерахування, в якому можливі значення явно задаються при об’яві типу);
- any (будь-який тип даних).

З цих типів даних можна будувати більш великі (як літерали, так і класи), які можуть використовуватися для створення ще більш великих та ін. Наприклад, клас Address (Адреса) складається з набора строк і може використовуватися як тип атрибуту в будь-якому класі, де потрібна адреса.

Дії, які може виконувати об’єкт, називають операціями. Операція може потребувати вхідні параметри та повертати значення будь-якого з відомих типів. Деякі дані містяться в схемі, яка записана на мові визначення об’єктів Object Definition Language (ODL). Це мова маніпулювання даними, яка визначена як частина стандарту.

2.2 Формування зовнішньої специфікації та реалізація

Одна з основних властивостей об’єктно-орієнтованої парадигми – це визначення відмінностей між відкритим інтерфейсом класу та його закритими елементами (інкапсуляція). В проекті стандарту об’єктно-орієнтованих баз даних ця відмінність знайшла відображення в термінах зовнішня специфікація типу та його реалізація.

До складу зовнішньої специфікації типу входять:

- властивості типу, доступні за допомогою того чи іншого механізму;
- виключення, які можуть ініціювати операції типу (у програмуванні виключенням називається помилка, що передбачувана. Наприклад, коли програма намагається відкрити файл, щоб прочитати дані, які в ньому зберігаються, може виникнути виключення (як зазвичай кажуть, «буде збуджено виняток»), якщо файл не знайдено. Тоді програма може перехопити виняток та зробити ті чи інші дії. В даному прикладі такою дією могла б стати видача повідомлення про те, що файл не знайдено. Включивши

обробку винятків в операції, що виконуються типами, програміст може спростити роботу та забезпечити більш суворий контроль цілісності даних);

- операції, які можна виконувати над екземпляром типу.

Зовнішня специфікація інтерфейсу не повинна залежати від реалізації. Вона містить тільки ту інформацію, яка необхідна для виклику операції над екземпляром типу, отримання значення властивостей та ідентифікації виключень. Автори стандарту називають це абстрактним описом, маючи на увазі, що деталі реалізації відсутні. Клас містить як абстрактні операції, так і абстрактний стан типу. Літерал зберігає тільки абстрактний стан.

Реалізація типу складається з двох частин. Представлення – це залежна від мови програмування структура даних, що містить властивості типу. Особливості реалізації є наслідком мовної прив'язки. Це означає, що внутрішнє уявлення типу буде різним у залежності від мови програмування, що використовується, і відповідно, у даного типу може бути декілька представлень.

Деталі операцій типу задаються набором методів. Для кожної операції у зовнішній специфікації повинен бути вказаний принаймні один метод. Однак тип може включати і такі методи, які не видно ззовні; зазвичай вони виконують допоміжні функції, необхідні іншим методам типу.

Методи кодуються на тій же мові програмування, яка використовувалася під час запису представлення типу. Якщо, наприклад, база даних підтримує програмування додатків на мовах C++, Java та SmallTalk, то знадобляться три реалізації кожного типу, по одній на кожній мові. Однак, як правило, в будь-якій прикладній програмі використовується тільки одна реалізація.

2.3 Організація наслідування інтерфейсів і класи

ООБД підтримує просте і множинне наслідування інтерфейсів. Правила успадкування дещо відрізняються від тих, які використовуються в

об'єктно-орієнтованих мовах програмування. Така суперечливість у визначеннях досить ускладнює розуміння принципів роботи конкретного об’єктно-орієнтованого середовища. Але є надія, що з часом з'явиться загальноприйнятий стандарт, в термінології якого не буде цих суперечностей.

У термінах стандарту проекту інтерфейс – це оголошення типу, з якого не можна створювати об'єкти (те ж саме, що абстрактний клас в більшості об'єктно-орієнтованих мов програмування). Клас – це тип, з якого можна створювати об'єкти, тобто еквівалент конкретного класу.

Наслідування, тобто відношення типу «є», може бути застосовано тільки до інтерфейсів. Оскільки інтерфейси (частини моделі ООБД) містять лише оголошення поведінки, то за допомогою цього механізму успадковується тільки поведінка. Інтерфейси, які успадковують інші інтерфейси, називаються супертипами, а інтерфейси-наслідники називаються підтипами.

Наприклад, якщо створюється об’єктно-орієнтована база даних для підприємства роздрібної торгівлі, то від інтерфейсу з ім'ям Item (Товар) можна зробити інтерфейс Furniture (Мебель), а від нього, в свою чергу, інтерфейси Chair (Стілець), Desk (Стіл) , Bad (Ліжко) та ін. У моделі бази даних це можна записати так, як наведено у прикладі 3.1.

```
interface Item {…};
interface Furniture : Item {…};
interface Chair : Furniture : Item {…};
interface Desk : Furniture : Item {…};
interface Bad : Furniture : Item {…};
```

Приклад 2.1 – Створення об’єктно-орієнтованої БД для підприємства роздрібної торгівлі

«Найнижчій» інтерфейс в ієрархії наслідування називається найспецифічнішим типом. Через те, що він успадковує поведінку всіх типів, які розташовані вище в ієрархії, він виявляється найповнішим. У прикладі 3.1

Chair, Desk та Bad – це найспецифічніші інтерфейси.

Одна з практичних переваг наслідування полягає в тому, що з підтипом можна взаємодіяти як з супертипом. Наприклад, прикладна програма може з однаковим успіхом оперувати стільцями, столами та ліжками, як меблями або навіть як товарами загального виду. Це спрощує розгляд підтипів як групи, якщо необхідно.

Підтипи можна спеціалізувати додаванням поведінки. Підтипи спеціалізованого підтипу наслідуватимуть додану до супертипу поведінку.

Враховуючи, що інтерфейси можна використовувати для створення об'єктів, до організації ієрархії успадкування застосовується кілька спеціальних правил:

- інтерфейси можуть успадковуватися від інших інтерфейсів;
- класи можуть успадковуватися від інтерфейсів;
- інтерфейси не можуть успадковуватися від інших класів;
- класи не можуть успадковуватися від інших класів.

На практиці це означає, що класи завжди знаходяться найнижче в ієрархії успадкування.

Модель об'єктної бази даних підтримує множинне наслідування. Однак в ній є неприпустимою перевантаження методів як частини наслідування. Така ситуація можлива, якщо два супертипа мають операції з однаковими іменами, але різними сигнатурами. Ніякі два інтерфейсу в ієрархії успадкування не можуть мати операції з однаковими іменами. Однак перевантаження всередині одного класу або інтерфейсу дозволене.

В моделі об'єктно-орієнтованої бази даних використовується відношення «розширює» для позначення одночасного наслідування стану та поведінки. Клас, який розширює інший клас, має доступ до всього стану і операціям свого супертипа, включаючи все те, що останній успадкував від своїх інтерфейсів.

Клас може одночасно успадковувати стан та поведінку не більше одного класу. Однак якщо ви будуєте ієрархію розширень, то класи внизу

ієрархії успадковують все, що їх супертипи успадкували від класів, які знаходяться в ієрархії вище. Як приклад розглянемо рисунок 2.1, на якому зображена ієрархія наслідування для університету.

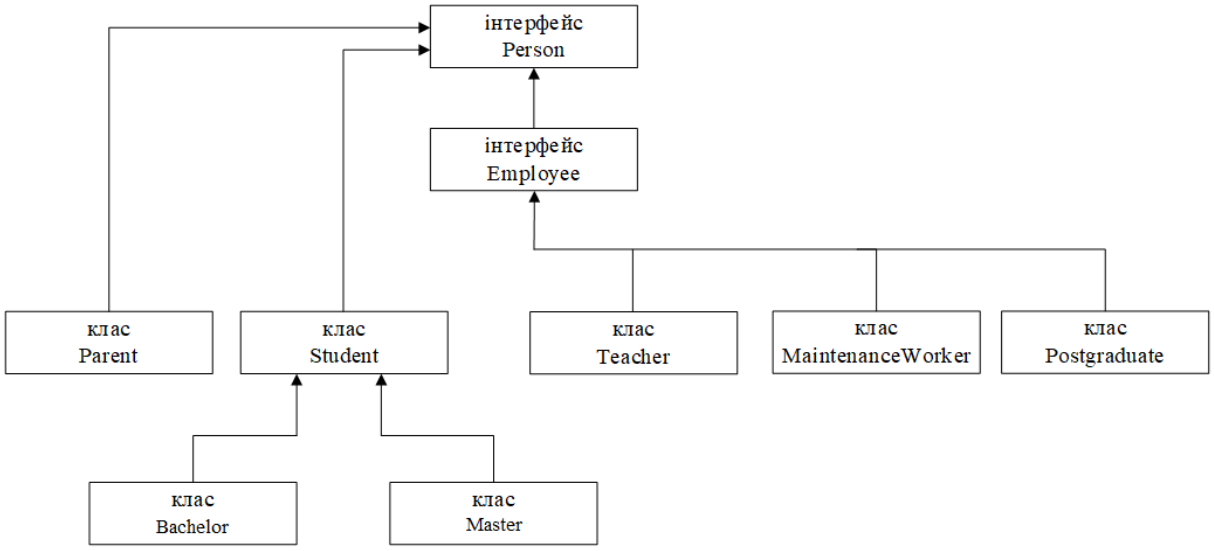


Рисунок 2.1 – Наслідування відповідно до стандарту ООБД

Інтерфейс Person (Людина) описує поведінку, яка властива всім людям, які представлено в базі даних. Однак вся множина людей розпадається на більш спеціалізовані категорії. Деякі, наприклад, є працівниками. Інтерфейс Employee, таким чином, містить поведінку, характерну для тих, хто працює в університеті. Інтерфейс Employee наслідує конкретні типи працівників, наприклад Teacher (Викладач), Postgraduate (Аспірант), MaintenanceWorker (Обслуговуючий персонал).

Клас Parent (Батьки) наслідує безпосередньо інтерфейс Person. Клас Student (Студенти) також унаслідується від вищезазначеного інтерфейсу. Класи Bachelor (Бакалавр) та Master (Магістр) розширюють клас Student. Саме з цих класів створюються об’єкти, які представляють студентів.

2.4 Робота з об’єктами

У стандарті ООБД існують характеристики об'єктів, переваги та недоліки які розглянемо нижче.

СУБД може організувати екстент (безперервна область в файлових системах) для кожного типу, що включає всі екземпляри, які належать цьому типу. Наприклад, екстент класу Guardian буде містити в собі всі об'єкти, що описують опікунів. Екстен можна проіндексувати для отримання швидкого доступу до його вмісту. Однак, стандарт проекту попереджає, що підтримка індексів пов'язана з великими накладними витратами та їх наявність факультативна.

Так само, як рядку в реляційній таблиці, типу можна приписати унікальний ідентифікатор, або ключ. Якщо ключ створено на основі тільки однієї властивості, він називається простим. Ключ, який складено з кількох властивостей, називається складовим. Однак, на відміну від реляційної моделі даних, від ключа не потрібно унікальності.

У кожного об'єкта є унікальний ідентифікатор об'єкта, який генерується СУБД. Його вміст залежить від конкретної СУБД, що залишає розробникам свободу у виборі фізичних методів доступу до об'єктів.

У кожного типу є ім'я, доступне з будь якої точки середовища, воно глобально для всієї бази даних. Відповідальність за відображення імені на внутрішнє уявлення типу лежить на СУБД.

Об'єкти можуть бути тимчасовими (transient) або стійкими (persistent). Тимчасові об'єкти існують тільки до тих пір, поки програма, яка створила їх працює. Вони використовуються або для непостійного зберігання, або для внутрішніх цілей у програмному забезпеченні. Стійкі об'єкти містяться в самій базі даних.

Згідно зі стандартом, СУБД повинна підтримувати чотири структурованих об’єкта: Date (дата), Interval (інтервал часу), Time (час), Timestamp (штамп часу).

Структура й функції цих об’єктів такі самі, як у однойменних типів даних, визначених в стандарті SQL-92.

Об’єкти-колекції, похідні від інтерфейсу Collection – це еквівалент контейнерних класів в стандарті проекту ООБД. Вони дозволяють зберігати в складі об'єкта кілька значень однієї й тієї ж самої властивості. В стандарті проекту згадані наступні об'єкти-коллекціі, або набори:

- множина (set) містить неупорядковану групу об'єктів одного і того ж самого типу, в якій не допускаються дублікати;
- комплект (bag) включає в себе неупорядковану групу об'єктів однакового типу, в якій допускаються дублікати;
- список (list) містить впорядковану групу об'єктів однакового типу;
- масив (array) включає в себе впорядковану групу об'єктів однакового типу, до елементів якого можна звертатися шляхом зазначення номера позиції. Цей масив аналогічний масиву, який використовується в мовах програмування, однак має розмір, що змінюється (не фіксований). Елементи можна вставляти та видаляти з будь-якої позиції;
- словник (dictionary) складається з впорядкованих ключів, з кожним з яких зіставлено одне значення. Наприклад, якщо є словник об'єктів класу book (книга), то можна взяти в якості ключа будь яку вибірку символів з назви книги та зв’язати з цим ключем повну назву. Тоді таким словником можна буде скористатися для швидкого пошуку назв книг.

В інтерфейсі Collection – це визначені методи (таблиця 3.1), які повертають інформацію про вміст набору об’єкта-колекції, або виконують певні операції з ним.

Таблиця 2.1 – Операції інтерфейсу Collection

Операція	Дія
1	2
invalidCollectionType	Збуджує виключення, якщо запропоновано некоректний тип набору.

Продовження таблиці 2.1

1	2
elementNotFound	Збуджує виключення, якщо зазначений елемент не входить в даний набір.
is_empty	Повертає true, якщо набір порожній, і false у противному випадку.
cardinality	Повертає число елементів в наборі.
is_ordered	Повертає true, якщо елементи набору впорядковані, і false у протилежному випадку.
allows_duplicates	Повертає true, якщо набір допускає дублікати, і false у протилежному випадку.
contains_element	Повертає true, якщо набір містить зазначений елемент, і false у протилежному випадку.
insert_element	Додає елемент в набір.
remove_element	Видаляє елемент з набору.
create_iterator	Створює об’єкт, який можна використовувати для перебору елементів в прямому порядку (від першого до останнього).
create_bidirectional_iterator	Створює об’єкт, який можна використовувати для перебору елементів в прямому і зворотному порядку (від першого до останнього і від останнього до першого).

Створення об'єктів в ООБД тісно пов’язано з мовою програмування, що використовується для написання додатку, яка ці об'єкти і формує. Кожна мовна прив'язка надає готові інтерфейси для будь-якого типу, в якому визначена операція створення нових екземплярів. Вона називається new та завжди повертає новий екземпляр типу. Для видалення об'єкта з бази даних потрібно скористатися операцією delete.

2.5 Представлення логічних зв’язків

В проекті стандарту об'єктно-орієнтованих баз даних зв'язки між типами представляються шляхом поміщення ідентифікаторів пов'язаних об'єктів всередину об'єкта, з яким вони пов'язані. Зв'язки визначаються між парою типів (як правило, класів), не допускаються зв'язки між трьома типами та більше.

Самі зв'язки не є об'єктами. У них немає імен та ідентифікаторів у базі даних. У кожного зв'язку є дві частини – типи на кожному її кінці, тому зв'язки завжди повинні оголошуватися парами. Сукупність зв'язків, що ведуть від одного типу до іншого, називаються шляхом навігації (traversal path). Саме вдовж цього шляху додаток може виконувати навігацію по базі даних.

СУБД повинна також виконувати каскадне видалення. Це означає, що якщо додаток знищує об'єкт, який пов'язано з іншими об'єктами, то зв'язки в останніх також повинні бути видалені. Пов'язані об'єкти можуть залишитися в базі даних, але з них слід прибрати ідентифікатори, які утворювали шлях навігації.

Для створення зв'язку виду «більшість до більшості» атрибут, що містить ідентифікатори пов'язаних об'єктів, оголошується як множинний. У ньому може зберігатися набір будь-якого з вищезазначених типів, хоча зазвичай для цієї мети застосовується множина.

2.6 Сфери використання об’єктно-орієнтованих баз даних

Об’єктно-орієнтовані бази даних сьогодні використовуються рідко, але в деяких випадках вони необхідні. Це відбувається тоді, коли дані дуже складні, в тому сенсі, що вони не є примітивними типами даних, наприклад, якщо існує потреба в зберіганні їх поведінки, то виникає потреба в об'єктно-орієнтованих базах даних.

На даний час об'єктно-орієнтовані бази даних використовуються організаціями для різних цілей.

EA Davis Company, видавці медичного словника «Taber's Cyclopeedic Medical Dictionary» та інших технічних довідкових матеріалів, використовують об'єктно-орієнтовану базу даних для управління змістом своїх публікацій. Компанія використовує XML та SGML для форматування своїх документів тому що ієрархічна природа цих документів більш точно відповідає об'єктно-орієнтованій моделі, ніж реляційної базі даних.

Echelon використовує об'єктно-орієнтовану базу даних в якості основи для свого програмного забезпечення мережі управління. Мережа управління – це мережа, яка відстежує або управляє процесом з мінімальним контролем з боку людини, наприклад, роботизовані системи на заводах, ліфти та клімат-контроль у висотних будівлях. Програмне забезпечення моделює кожен елемент, що контролюється, а також його поведінку.

Sales Media, розробник програмного забезпечення для індустрії страхування, перейшла на об'єктно-орієнтовану базу даних в якості основи для свого продукту Automated Agent. Комбінація об'єктно-орієнтованого внутрішнього інтерфейсу та об'єктно-орієнтованої мови розробки додатків спрощує для Sales Media швидку доставку оновлень для свого програмного забезпечення.

Компанія Adidas використовує об'єктно-орієнтовану базу даних для обробки свого каталогу компакт-дисків, контенту для свого веб-сайту та продажів в торгових точках. Консалтингова фірма, яка була найнята Adidas для розробки інформаційної системи, вирішила, що об'єктно-орієнтована база даних – кращий засіб обробляти великі обсяги мультимедійних даних (текст, аудіо, нерухомі зображення та відео). Навігаційна природа об'єктно-орієнтованої моделі даних забезпечує відмінну продуктивність для пошуку великих мультимедійних файлів.

Radio Computing Services надає програмне забезпечення для автоматизації радіостанцій. Їх оригінальна програма Selector підтримує вибір,

упорядкування та моніторинг пісень, відтворюваних станцією. Сьогодні компанія використовує об'єктно-орієнтовану базу даних в якості основного сховища даних для більш повного пакету автоматизації станцій, включаючи музичні уривки та рекламні ролики. Об'єктно-орієнтоване середовище економить час розробки, оскільки дозволяє програмістам однаково обробляти весь програмний матеріал, незалежно від того його типу.

Компанія Interface and Control Systems, використовує об'єктно-орієнтовані технології для створення бази даних, що підтримує систему стеження за польотом космічного апарату та керування їм. Для роботи даної системи потрібно аналізувати дуже великі обсяги даних. Об'єктно-орієнтована база здатна зробити це без помітного зниження продуктивності.

Французький національний центр космічних досліджень застосовує об'єктно-орієнтовані технології в авіакосмічній промисловості. Зокрема, там розроблена мультимедійна база даних, яка допомагає моделювати інтегровані системи, необхідні в проектуванні космічних апаратів. Фахівці, що працюють в цій організації, прийшли до висновку, що об'єктно-орієнтована база даних добре пристосована до середовища, в якому є велика кількість елементів, взаємодіючих між собою складним чином.

Компанія STERIA застосовує об'єктно-орієнтовану базу даних для підтримки комерційного додатка, призначеного для проектування та планування великих мереж передачі даних. Складність таких мереж настільки швидко зростає, що досить важко стежити, де яке обладнання і в якій конфігурації воно встановлено. Об'єктно-орієнтована база дозволяє проектувальнику моделювати взаємодії різних елементів мережі та постійно бути в курсі її структури.

Найбільша швейцарська компанія медичного страхування Swiss Social Christian має об'єктно-орієнтовану базу даних, що лежить в основі системи опрацювання звернень про страхові виплати. Компанія прийшла до рішення використовувати таку базу, оскільки правила прийняття рішень про те, яка

сума підлягає виплаті, дуже складні. Система здатна витримувати велику кількість звернень, приблизно до 18000 в день.

Французька електрична компанія Electricite de France користується об'єктно-орієнтованою базою для управління навантаженням на лінії електропередачі. База даних здатна намалювати карту ліній, розташованих в області відповідальності обслуговуючого підприємства. ООБД так само допомагає визначити, яке обладнання необхідно для прокладки нової лінії.

Після знайомства з усіма цими додатками можна сформулювати деякі загальні спостереження:

- багато додатків складаються з багатьох взаємодіючих частин, як, наприклад, космічні апарати або великі обчислювальні мережі. Кожна з цих частин має свою поведінку, яка залежить від поведінки інших;
- багато систем повинні обробляти великі обсяги даних;
- багато додатків здійснюють передбачуваний доступ до даних, через це навігаційна природа об'єктно-орієнтованих баз не є великим недоліком;
- у більшості додатків потреба в незапланованих запитах обмежена.

Сформульовані вище характеристики допомагають ідентифікувати додатки, в яких виправдано застосування об'єктно-орієнтованих баз даних. Однак, у випадках коли в середовищі є шаблони доступу, які важко передбачити, реляційна база, можливо, стане більш кращим вибором.

2.7 Порівняння проблем, які виникають в реляційних та об’єктно-орієнтованих базах даних

При використанні популярних нині реляційних баз даних у сучасних додатках виникають деякі центральні проблеми, які можна уникнути за допомогою використання ООБД.

UML в основному розробляється на принципах, які похідні від програмного забезпечення машинобудування. Однією з властивостей мови UML є те, що вона орієнтована на об'єкти, тому найкраще підходить для

моделювання об'єктно-орієнтованих систем та даних.

Реляційні бази даних базуються на математичній теорії множин. Це означає, що понять як об'єкти та класи не існує. Це є суть проблеми, відомої як невідповідність імпедансу.

Наступна проблема, яка буде розглянута, це подвійне визначення. Через різні теоретичні основи реляційних баз даних та об'єктно-орієнтованих систем існує потреба у двох позначеннях: одне для об’єктно-орієнтованої системи та одне для реляційної бази даних.

Сьогодні мова UML використовується в більшості випадків для моделювання об'єктно-орієнтованої системи. Одночасно є діаграми взаємозв'язку суб'єктів (ER-діаграми), які використовуються для моделювання реляційної бази даних. Це призводить до двох незручностей. По-перше, розробник повинен мати навички моделювання у двох позначеннях, UML та ER. По-друге, дані, що зберігаються в реляційних базах даних, майже завжди об'єктно-орієнтовані, тому що вони витягуються, генеруються або в деяких випадках використовуються в об'єктно-орієнтованій системі. Коли вони використовуються в системі, вони описуються та управляються класами та об’єктами. Ці класи та об'єкти повинні бути відображені в реляційній базі даних та змодельовані за допомогою ER.

Перетворення класу з UML діаграми до ER-діаграми є однією з двох найбільших проблем при використанні реляційної бази даних. Для цього існує множина патерів прив’язки. Більшість з них будуються поетапно. На першому кроці треба з’ясувати, яку інформацію слід зберігати і в якому класі вона розташована. Другим кроком є зіставлення класів із відношеннями, тобто таблицями, у реляційній моделі. Третій крок – це відображення атрибутів цих класів у стовпцях в таблиці реляційної моделі. Четвертий крок – це пошук унікальних ідентифікаторів, які можна використовувати як ключі в базі даних. Якщо жодного унікального атрибута немає, необхідно створити ключ, який буде генеруватися системою. П’ятий крок – це пошук

об’єктів, інкапсульованих іншими об’єктами, та їх протиставлення до взаємозв’язків у моделі реляційної бази даних.

Реалізація інтерфейсів у реляційній базі даних – це друга найбільша проблема при відображенні об’єктно-орієнтованих даних в реляційній моделі.

Незважаючи на те, що об’єктно-орієнтовані бази даних здаються новими і дуже вдало вирішують найбільші проблеми реляційних баз даних, вони все ще часто не використовується. Основною причиною цього є те, що розробники звикли користуватися реляційними базами даних і багато працювали з цією технологією. Проблеми, які з’являються при використані реляційних баз даних, були вирішені раніше, і ці рішення можуть бути реалізовані знову, якщо це необхідно.

У проектах з обмеженою економічністю немає часу на вивчення нової технології для використання в такій центральній частині комп'ютерної системи, як база даних. Якщо реалізація бази даних не вдається, розгортання проекту на стороні замовника може не тільки затриматися, але й взагалі не відбутися. Причиною того, що вартість впровадження об'єктно-орієнтованої бази даних є настільки висока, є відсутність стандартів у цій галузі.

Існує кілька постачальників, які надають об'єктно-орієнтовані системи управління базами даних, але всі вони реалізовані по-різному. Це означає, що користувач повинен знати не тільки технологію одного додатка, але й декількох додатків, щоб мати можливість вибрати той, який найкраще підходить для поточної задачі. Ці дві проблеми призводять до того, що вигоди від впровадження об'єктно-орієнтованої бази даних у багатьох випадках менші за витрати, виміряні часом, який витрачено на вивчення нової технології.

Іншою проблемою об'єктно-орієнтованих баз даних може бути продуктивність. Багато об’єктно-орієнтованих баз даних реалізують стійкість завдяки доступності.

База даних представлена деревом з кореневим вузлом, тому через те,

що існують обмеження в оптимізації дерев, це може призвести до можливих проблем з продуктивністю.

2.8 Приклади об’єктно-орієнтованих систем управління базами даних

Система управління базами даних (СУБД) – це сукупність програмних та лінгвістичних засобів загального або спеціального призначення, що забезпечують управління створенням та використанням баз даних.

СУБД – це комплекс програм, що дозволяють створити базу даних та маніпулювати даними (вставляти, оновлювати, видаляти та обирати). Система забезпечує безпеку, надійність зберігання та цілісність даних, а також надає засоби для адміністрування БД.

Зараз ведеться дуже багато експериментальних та виробничих робіт в області об'єктно-орієнтованих СУБД. Найбільше університетських робіт, які в основному носять дослідницький характер. Але вже декілька років тому відзначалося існування щонайменше тринадцяти комерційно доступних систем ООБД. На жаль, з приводу цих систем практично не доступно жодних публікацій.

Проект ORION здійснювався з 1985 по 1989 р. фірмою MCC під керівництвом В. Кіма. Під назвою ORION насправді ховається сімейство трьох СУБД: ORION-1 – система, яка розрахована на одного користувача; ORION-1SX, призначена для використання в якості сервера в локальній мережі робочих станцій; ORION-2 – повністю розподілена об'єктно-орієнтована СУБД. Реалізація всіх систем проводилася з використанням мови Common Lisp на робочих станціях (і їх локальних мережах) Symbolics 3600 з операційною системою Genera 7.0 і SUN-3 в середовищі операційної системи UNIX. Опис реалізації ORION-2 поки не опубліковано, тому розглянемо тільки ORION-1 та ORION-1SX.

Основними функціональними компонентами системи є підсистеми управління пам'яттю, об'єктами та транзакціями. У ORION-1 всі компоненти

розташовуються на одній робочій станції; в ORION-1SX – рознесені між різними робочими станціями (зокрема, управління об'єктами відбувається на робочій станції-клієнті). Застосування в ORION-1SX для взаємодії клієнт-серверного механізму віддаленого виклику процедур дозволило використовувати в цій системі практично без переробки багато модулів ORION-1. Мережні взаємодії гуртувалися на стандартних засобах операційних систем.

В число функцій підсистеми управління пам'яттю входить розподіл зовнішньої пам'яті, переміщення сторінок з буферів оперативної пам'яті в зовнішню пам'ять та навпаки, пошук і розміщення об'єктів в буферах оперативної пам'яті (як прийнято в об'єктно-орієнтованих системах, підтримується два представлення об'єктів – дисковий та в оперативній пам'яті; при переміщенні об'єкта з буфера сторінок в буфер об'єктів та назад уявлення об'єкта змінюється). Крім того, ця підсистема відповідальна за підтримку допоміжних індексних структур, призначених для прискорення виконання запитів.

Підсистема управління об'єктами включає підкомпоненти обробки запитів, управління схемою та версіями об'єктів. Версії підтримуються тільки для об'єктів, при створенні яких така необхідність була явно вказана. Для схеми БД версії не підтримуються; при зміні схеми відстежується вплив цієї зміни на інші компоненти схеми та на існуючі об'єкти. При обробці запитів використовується техніка оптимізації, аналогічна тій, що застосовується в реляційних системах (тобто формується набір можливих планів виконання запиту, оцінюється вартість кожного з них та обирається для виконання найбільш дешевий).

Підсистема управління транзакціями забезпечує традиційну серіалізованість транзакцій, а також підтримує засоби журналізації змін та відновлення БД після збоїв. Для серіалізації транзакцій застосовується різновид двофазного протоколу синхронізаційних захоплень з різним ступенем гранулювання. Звичайно, при синхронізації враховується специфіка

ООБД, зокрема, наявність ієрархії класів. Журнал змін забезпечує відкати індивідуальних транзакцій тавідновлення БД після м'яких збоїв (архівні копії БД для відновлення після поломки дисків не підтримуються).

Проект O2 реалізується французькою компанією Altair, яка утворена спеціально для цілей проектування та реалізації об'єктно-орієнтованої СУБД. Початок проекту датується вереснем 1986 р і він був розрахований на п'ять років: три роки на прототипування та два роки на розробку промислового зразку. Після успішного завершення проекту для супроводу системи та її подальшого розвитку була організована нова чисто комерційна компанія O2.

Прототип системи функціонував в режимі клієнт-сервер в локальній мережі робочих станцій SUN с відповідним розподілом функцій між сервером і клієнтами.

Основними компонентами системи (не рахуючи розвиненого набору інтерфейсних засобів) є інтерпретатор запитів та підсистеми управління схемою, об'єктами та дисками. Управління дисками, тобто підтримка базового середовища постійного зберігання забезпечує система WiSS, яку розробники O2 перенесли в оточення операційної системи UNIX.

Найбільше функціональне навантаження несе компонент управління об'єктами. У число функцій цієї підсистеми входять:

- управління складними об'єктами, що включає створення та знищення об'єктів, вибірку об'єктів по іменах, підтримку зумовлених методів, підтримку об'єктів зі внутрішньою структурою-безліччю, списком та кортежем;
- управління передачею повідомлень між об'єктами;
- управління транзакціями;
- управління комунікаційним середовищем (на базі транспортних протоколів TCP/IP в локальній мережі Ethernet);
- відстеження довготривало збережених об'єктів;
- управління буферами оперативної пам'яті;
- управління кластеризацією об'єктів у зовнішній пам'яті;

- управління індексами.

Навіть наведений короткий опис особливостей двох об'єктно-орієнтованих СУБД показує прагматичність сучасного підходу до організації таких систем. Їх розробники не прагнуть до повного дотримання чистоти об'єктно-орієнтованого підходу та застосовують найбільш прості рішення проблем. Поки в співтоваристві розробників об'єктно-орієнтованих систем баз даних не видно роботи, яка могла б зіграти в цьому напрямку роль, аналогічну ролі System R по відношенню до реляційних систем.

3 ФУНКЦІОНАЛЬНА СХЕМА ПРОЕКТУВАННЯ ПРОГРАМНОГО ТА ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ ОБ'ЄКТНО-ОРІЄНТОВАНИХ БАЗ ДАНИХ

При створенні бази даних робота традиційно ділиться на три етапи; аналіз, розробка та реалізація.

Під час аналізу встановлюються більш глибокі знання про предметну область та дані, наявні в цій області. Перше, що повинен зробити розробник, це проаналізувати навколишнє середовище, в якому повинна знаходитися база даних. Саме це часто називають областю процесу розробки програмного забезпечення. При виконанні цього аналізу розробник повинен подивитися, які дані використовуються і як вони використовуються. Він також повинен подивитися, як працюють різні потоки даних через домен. Досвідчений розробник шукає шляхи вдосконалення обробки даних паралельно з аналізом потоків даних. На цьому етапі важливим є контакт з людьми, які працюють у межах предметної області, тобто працівниками підприємства. Вони мають безцінне знання даних, що використовуються в поточному домені.

Фазу проектування бази даних можна розділити на три етапи:

- створення концептуальної моделі;
- логічне проектування;
- створення фізичної моделі.

На першому етапі розробник повинен знайти засіб зафіксувати дані та потоки даних у концептуальній моделі. При описі концептуальної моделі традиційно використовуються ER-діаграми.

На другому етапі створюється логічна модель даних, наявних у предметній області, щодо конкретної архітектури бази даних. Логічна модель додає атрибути до сутності даних у концептуальній моделі та усуває всі взаємозв'язки «багато-до-багатьох». Відносини «багато-до-багатьох» – це відносини, які існують, наприклад, у наступній ситуації: багато студентів

проходять один курс. Один студент може пройти багато курсів, тому відносини між курсом та студентом – «багато-до-багатьох».

На третьому кроці відбувається побудова фізичної моделі бази даних. Вона додає ключові обмеження до атрибутів сутностей, які визначено в логічній моделі. На цьому кроці створюється опис реалізації бази даних: опис базових відносин, файлова організація та індекси, що використовуються для досягнення ефективного доступу до даних, а також будь-які пов’язані із цим заходи цілісності та безпеки.

Важливою перевагою об'єктно-орієнтованої структури є відсутність розриву між структурою бази даних та поведінковим аспектом об'єктів. Об'єктно-орієнтована модель складається з незалежних один від одного модулів, таким чином, можна використовувати один вдосконалений клас в самих різних базах даних. Можливість успадкування вносить в програму логічну структуру, пов'язуючи класи між собою за принципом «від загального до конкретного».

Функціональна схема – це схема, що роз'яснює певні процеси, які відбуваються у певних функціональних частинах виробу (устаткування) чи у виробі (устаткуванні) в цілому.

Функціональна схема містить інформацію про способи реалізації пристроєм заданих функцій. За такою схемою можна визначити, як здійснюються перетворення і які для цього необхідні функціональні елементи. Кожен функціональний елемент містить лише ті входи та виходи, які необхідні для його коректної роботи. Дана схема розробляється на основі структурної схеми для кожного блоку, в результаті з окремих функціональних елементів складається загальна функціональна схема об'єкту.

Також функціональні схеми можуть застосовуватися у програмуванні для візуалізації алгоритмів та спрощення обчислення їх складності, однак у цій сфері форма створення – довільна (точніше, обирається та, яка зручна автору). Функціональна схема проектування програмного та інформаційного

забезпечення об’єктно-орієнтованої бази даних наведена на рисунку 3.1.

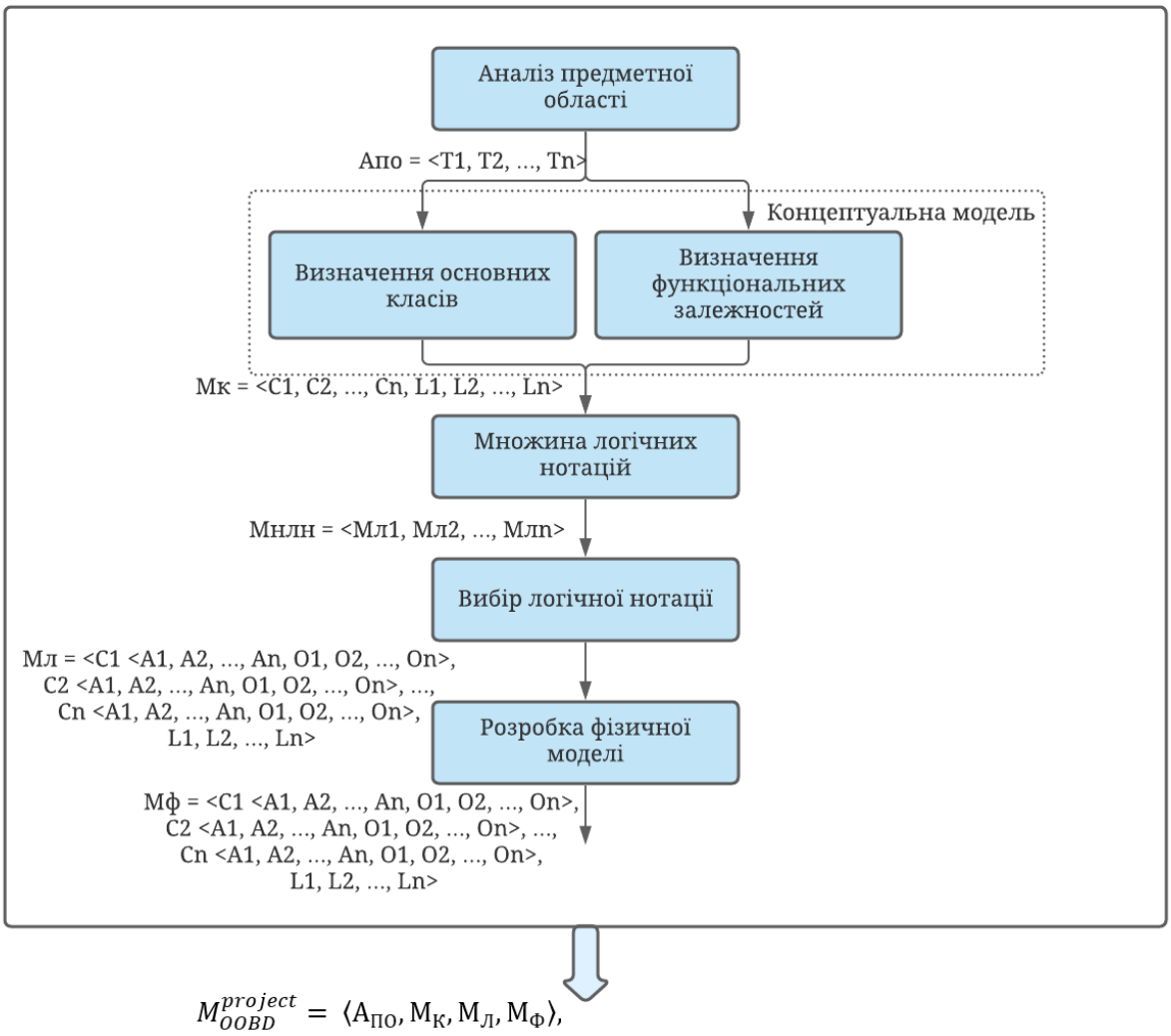


Рисунок 3.1 – Функціональна схема проектування програмного та інформаційного забезпечення об’єктно-орієнтованої бази даних

Для проектування програмного та інформаційного забезпечення об’єктно-орієнтованих баз даних пропонується використовувати процедуру, що складається з наступних етапів: аналіз предметної області, концептуальне проектування, логічне проектування, розробка фізичної моделі.

Формалізований метод проектування об’єктно-орієнтованої бази даних представляє собою формальну конструкцію виду:

$$M_{OOBD}^{project} = \langle A_{ПО}, M_К, M_Л, M_Ф \rangle,$$

де $А_{ПО}$ – аналіз предметної області;

$М_К$ – проектування концептуальної моделі об’єктно-орієнтованої бази даних;

$М_Л$ – проектування логічної моделі об’єктно-орієнтованої бази даних;

$М_Ф$ – проектування фізичної моделі об’єктно-орієнтованої бази даних.

Починаючи проектування об'єктно-орієнтованої бази даних, потрібно в першу чергу вивчити предметну область, в якій буде виконуватися робота БД, врахувати основні процеси, вимоги та побажання замовника та майбутніх користувачів. На цьому етапі відбувається визначення приблизного функціоналу програми та кордонів бази даних.

Наступним етапом є концептуальне проектування БД, інакше кажучи, конструювання моделі, що не залежить від будь-яких фізичних особливостей реалізації. Це дозволяє описати високорівневу модель бази даних та схему її роботи, незалежно від будь-яких деталей реалізації, як, наприклад, мова програмування, тип СУБД, різноманітні питання, що стосуються апаратної та програмних частин, продуктивності.

Після того, як концептуальне проектування було завершено, настає логічне проектування БД, тобто конструювання інформаційної моделі на основі конкретно виділених, існуючих типів даних, але без урахування специфіки СУБД та інших фізичних особливостей реалізації. Узагальнено, логічне проектування являє собою перетворення концептуальної моделі в логічну, при цьому враховуючи тип СУБД. Надалі логічна модель послужить основою для фізичної моделі. Логічна модель дозволяє розробнику всебічно вивчити поведінку БД та конкретних даних, прийняти деякі рішення щодо вибору засобів фізичної її реалізації.

3.1 Логічне моделювання об’єктно-орієнтованої бази даних

Першою формалізованою та загальновизнаною нотацією даних була реляційна модель Коада-Йодана. У цій моделі, як і у всіх наступних,

виділялися три аспекти – структурний, цілісний та маніпуляційний. Структури даних в реляційній моделі ґрунтуються на плоских нормалізованих відносинах, обмеження цілісності виражаються за допомогою засобів логіки першого порядку і, нарешті, маніпулювання даними здійснюється на основі реляційної алгебри або еквівалентного їй реляційного числення. Як відзначають багато дослідників, своїм успіхом реляційна модель даних багато в чому зобов'язана тому, що спиралася на суворий математичний апарат теорії множин, відносин та логіки першого порядку. Розробники будь-якої конкретної реляційної системи вважали своїм обов'язком показати відповідність своєї конкретної моделі даних загальній реляційній моделі, яка виступала в якості міри «реляційної» системи.

Основні труднощі об'єктно-орієнтованого моделювання даних є наслідком того, що такого розвиненого математичного апарату, на який могла б спиратися загальна об'єктно-орієнтована модель даних, не існує. У великій мірі тому досі немає базової об'єктно-орієнтованої моделі. З іншого боку, у одній із робіт Майера стверджується, що загальна об'єктно-орієнтована модель даних в класичному сенсі не може бути визначена через непридатність класичного поняття моделі даних до об’єктно-орієнтованої парадигми [10].

Не наводячи доказів на користь цього твердження Майера, але і не заперечуючи його, Бєєрі [13] пропонує в загальних рисах формальну основу ООБД, вона далеко не повна та не є моделлю даних в традиційному сенсі, але дозволяє дослідникам та розробникам систем ООБД принаймні говорити на одній мові (якщо, звичайно, пропозиції Бєєрі будуть розвинені та отримають підтримку).

По-перше, слідуючи практиці багатьох ООБД, пропонується виділити два рівня моделювання об'єктів: нижній (структурний) та верхній (поведінковий). На структурному рівні підтримуються складні об'єкти, їх ідентифікація та різновиди зв'язку «isa». База даних – це набір елементів даних, пов'язаних відносинами «входить в клас» або «є атрибутом». Таким

чином, БД може розглядатися як орієнтований граф.

Важливим аспектом є чіткий поділ схеми БД та самої БД. В якості первинних концепцій схемного рівня ООБД виступають типи та класи. Відзначається, що в усіх системах, що використовують тільки одне поняття (або тип, або клас) це поняття неминуче перевантажено: тип передбачає наявність певної кількості значень, що визначається структурою даних цього типу. Клас також передбачає наявність множини об'єктів, але ця множина визначається користувачем. Таким чином, типи та класи грають різну роль, і для строгості та недвозначності потрібні одночасне підтримання обох понять.

Бєєр не представляє повної формальної моделі структурного рівня ООБД, але висловлює впевненість, що поточного рівня розуміння досить, щоб формалізувати таку модель. Що ж стосується поведінкового рівня, запропоновано тільки загальний підхід до необхідного для цього логічного апарату.

Важливим, хоча і недостатньо обґрунтованим припущенням Беєра є те, що двох традиційних рівнів – схеми та даних для ООБД недостатньо. Для точного визначення ООБД потрібно рівень, вміст якого має визначати види об'єктів та зв'язків, допустимих на схемному рівні БД. Цей рівень повинен грати для ООБД таку ж роль, яку відіграє структурна частина реляційної моделі даних для схем реляційних баз даних.

Є достатня кількість інших публікацій, що відносяться до теми об'єктно-орієнтованих моделей даних, але вони чи зачіпають досить приватні питання, або використовують занадто серйозний для цього огляду математичний апарат. Для ілюстрації поточного стану справ слід розглянути особливості деяких моделей даних. Як приклад буде використовуватися проста БД про деталі та постачальників для компанії, що виробляє взуття та головні убори. Класи Footwear (Взуття) та Headgear (Головні убори) – це підкласи узагальненого класу MerchandiseItem (Товар). Крім цього, БД містить класи, що описують джерела сировини та матеріал, які необхідні для виробництва конкретних виробів.

3.1.1 Модель Коада-Йордана

Пітер Коад та Едвард Йордан є авторами основоположних робіт по об'єктно-орієнтованому аналізу та проектуванню. Найбільш широко використовується запропонована ними модель для уявлення класів та зв'язків між ними.

На рисунку 3.2 зображена ER-діаграма простої бази даних в нотації Коада-Йордана. Для прикладу включені кілька атрибутів та методів, але в теперішньому вигляді діаграма показана неповно.

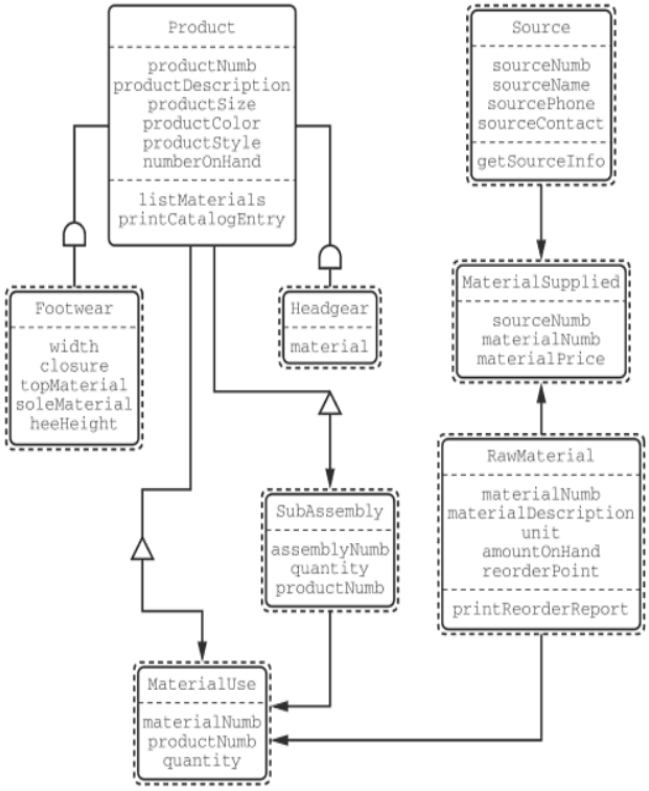


Рисунок 3.2 – Проект об’єктно-орієнтованої бази даних в нотації Коада

Ось деякі відмінні риси цієї нотації:

- клас відображається заокругленим прямокутником;

- прямокутники класів, з яких створюються об'єкти, обводяться пунктиром. У прямокутника, що представляє абстрактний клас, є лише одна

суцільна рамка;

- ім'я класу пишеться у верхній частині прямокутника, який його позначає;

- в середній частині прямокутника розташовуються атрибути, які визначені в класі;

- в нижній частині прямокутника розташовуються методи, які встановлені в класі;

- символ ⌓ представляє зв'язок типу «є» (клас Footwear є виробом, так само як і клас Headgear);

- символ △ відповідає зв'язку «ціле частина» (клас SubAssembly – це частина виробу);

- лінія без стрілки позначає жодного або один примірник;

- лінія зі стрілкою позначає жодного, один або більше примірників.

У нотації Коада є дві переваги. По-перше, вона добре підтримує спеціальні види зв'язків між класами, характерні для об'єктно-орієнтованого середовища. По друге, в ній чітко виділені абстрактні та конкретні класи. Однак, ця модель не дозволяє розрізнити обов'язкові та необов'язкові зв'язки та не залишає місця для документування семантики зв'язків. Відсутня також вбудована можливість представлення агрегованих класів, тому вони і не показані на рисунку 3.2.

3.1.2 Модель Шлеєра-Меллора

Альтернативою нотації Коада-Йордана є модель Шлеєра-Меллора. Класи в ній зображуються прямокутниками, що містять такі ж секції для імені, атрибутів та методів, що і в нотації Коада-Йордана (рисунок 3.3).

Зв'язок типу «є» в нотації Шлеєра-Меллора представлено вертикальною лінією, яка перекреслена горизонтальним штрихом. Зв'язки в базі даних зображені ромбиками, з яких виходять лінії зі стрілками. Одиночна стрілка означає нуль або один екземпляр, а подвійна стрілка –

один або більше. З кожним зв'язком асоційована мітка, яка пояснює її семантику.

Модель Шлеєра-Меллора дає більше інформації про зв'язки в базі даних, ніж модель Коада. Однак в ній немає явної підтримки зв'язку «ціле частина». До того ж вона теж не дозволяє розрізнити обов'язкові та факультатив зв'язки і не має простих засобів для подання агрегатів.

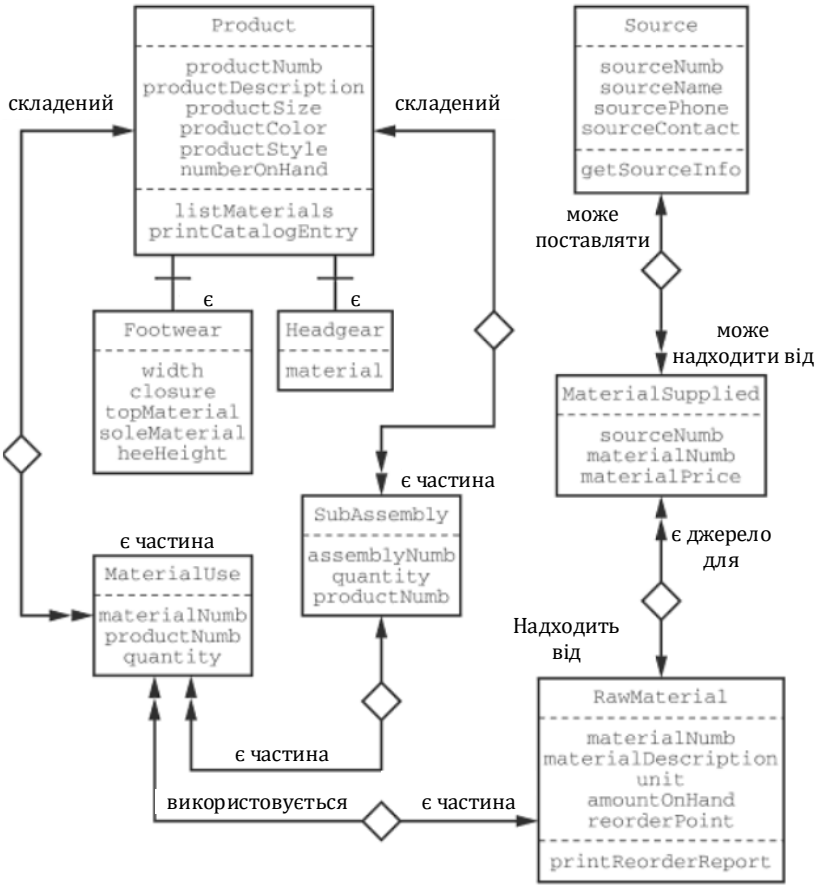


Рисунок 3.3 – Проект об’єктно-орієнтованої бази даних в моделі Шлеєра-Меллера

3.1.3 Модель Рамбо

Методика Object Modeling Technique (OMT – методика моделювання об'єктів) була розроблена групою вчених під керівництвом Дж. Рамбо. Він розглядає значимість моделювання за допомогою компонентів реального

світу, які називаються об'єктами. Цей підхід в значній мірі засновано на традиційних структурних методах та передбачає використання потужної системи позначень, яка разом з тим є достатньо складною та докладною. Складність нотації частково пояснюються тим, що деякі з підтримуваних нею засобів призначені для автоматичної генерації початкового коду. Запропонована ним модель OMT отримала широке визнання.

Приклад нотації OMT представлено на рисунку 3.4. Як і раніше, класи позначені прямокутниками, в які вписується ім'я, атрибути та методи класу. Зв'язок типу «є» представлений символом △, що протилежно його використанню в нотації Коада-Йордана.

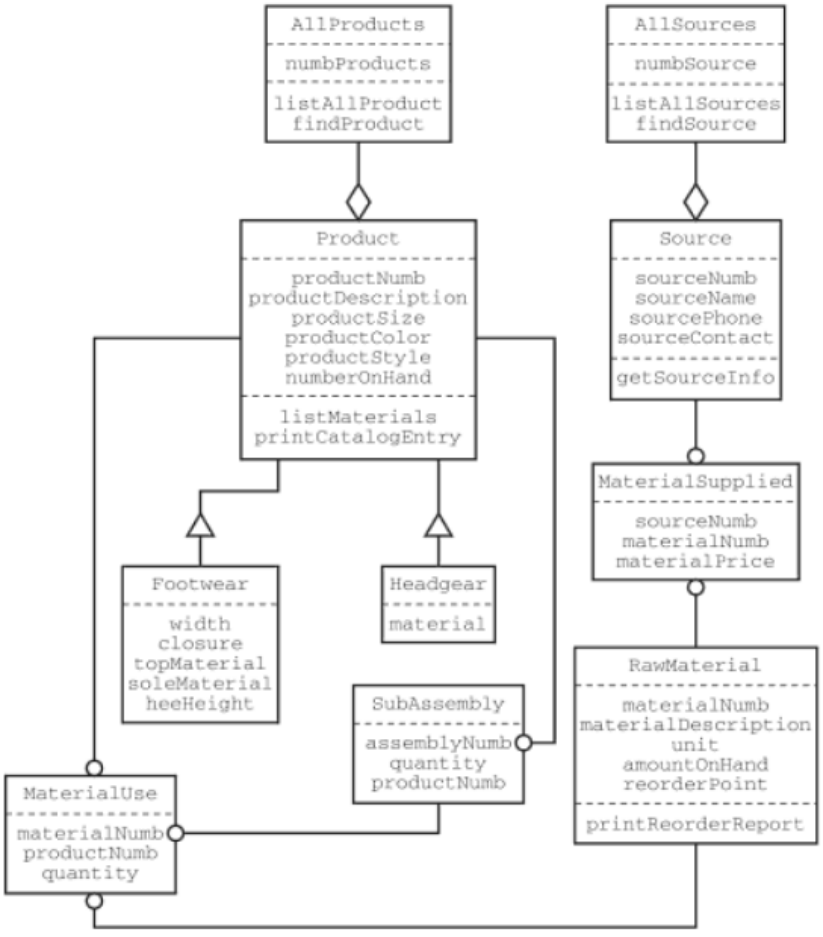
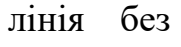
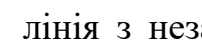


Рисунок 3.4 – Проект об’єктно-орієнтованої бази даних в нотації OMT

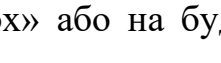
Види зв'язків, що називаються асоціаціями, позначаються символами на кінцях стрілок:

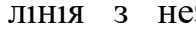
- лінія без стрілок () представляє «рівно один» та використовується на кінці «один» обов'язкового зв'язку «один до багатьох»;

- лінія з незафарбованим колом на кінці () означає «нуль або один». Вона застосовується на кінці «один» необов’язкового зв'язку «один до багатьох»;

- лінією з зафарбованим колом на кінці () зображується «нуль або більше». Вона вживається на кінці «багатьох» зв'язку «один до багатьох» або на будь-якому кінці зв'язку «багато до багатьох». В нотації OMT немає спеціального символу для позначення зв'язку «один або більше»;

- лінія з незафарбованим ромбом на кінці () показує обов’язкове агрегування, наприклад, входження об'єктів класів Product та Source до відповідних класів агрегатів;

- лінія з незафарбованими ромбом та колом на кінці () означає необов'язкове агрегування;

- лінія з незафарбованим ромбом та зафарбованим колом на кінці () показує множинне агрегування;

- якщо кратність зв'язку фіксована та заздалегідь відома, то її можна вказати, помістивши над лінією ціле число (). Припустимо, наприклад, що у кожного службовця є рівно два службових телефони, що зберігаються в об'єкті Employee (в кабінеті та мобільний). Тоді зв'язок між об'єктами, що описують службовця і номер телефону , буде представлена символом 2.

3.1.4 Модель Буча

Модель Буча є ще однією варіацією на тему ER-діаграмм для уявлення класів та їх асоціацій. Як видно з рисунку 3.5, класи відображаються закругленими прямокутниками з трьома звичайними секціями: ім'я, атрибути та методи.

Асоціації між класами маркуються різноманітними позначеннями на

лініях, що з’єднують їх:

- лінія без символу на кінці (—— 1 n ——) позначає асоціацію, кратність якої задається числами. На рисунку 3.5 всі асоціації мають кратність 1:n, тобто є обов'язковими зв’язками «один до багатьох»;
- лінія з одиночної стрілкою на кінці (——►) зображує наслідування. Як видно з рисунку 3.4, стрілка спрямована від виробничого класу до базового;
- лінія з зафарбованим колом на кінці (——●) показує асоціацію типу «є». На рисунку 3.5 вона використовується для уявлення зв'язків «ціле частина»;
- бліда лінія з однією стрілкою (——►) позначає метаклас, тобто клас класів. Такий зв'язок застосовується для асоціювання класів Product і Source з класами, які їх агрегують.

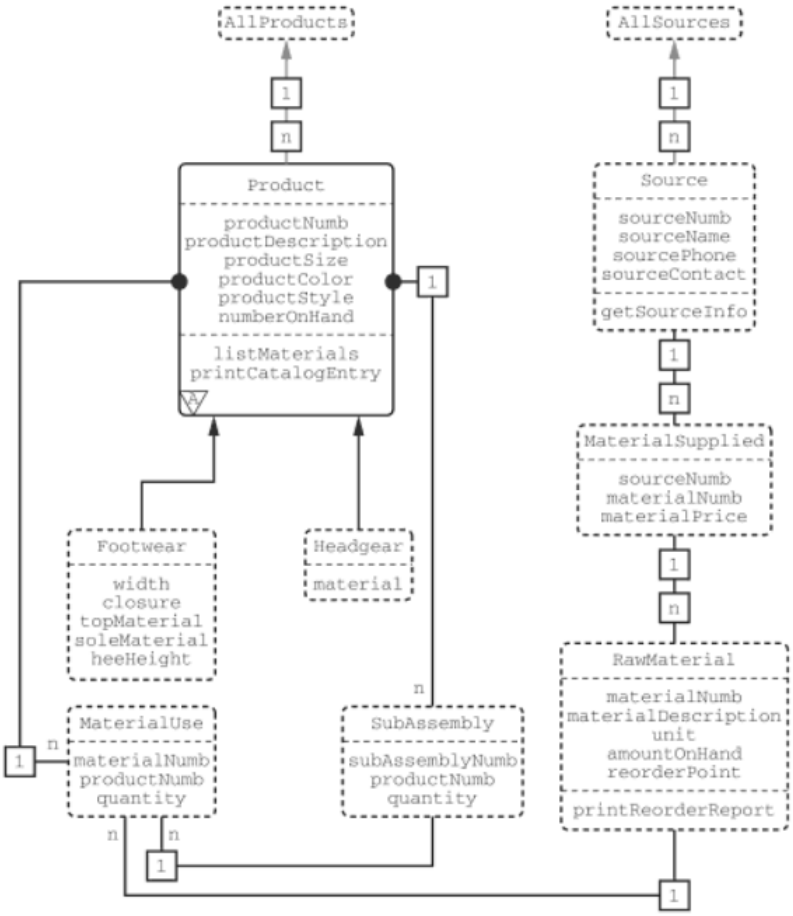


Рисунок 3.5 – Проект об’єктно-орієнтованої бази даних в нотації Буча

3.1.5 Уніфікована мова моделювання (UML)

До ролі стандарту для зображення моделей даних та інших елементів проекту системи найближче наблизилась уніфікована мова моделювання (Unified Modeling Language, UML). У її нотації об'єднані багато елементів, які розглядалися в попередній методиках. А крім того, ця мова має можливості, яких немає в інших системах. Приклад моделі UML наведено на рисунку 3.6.

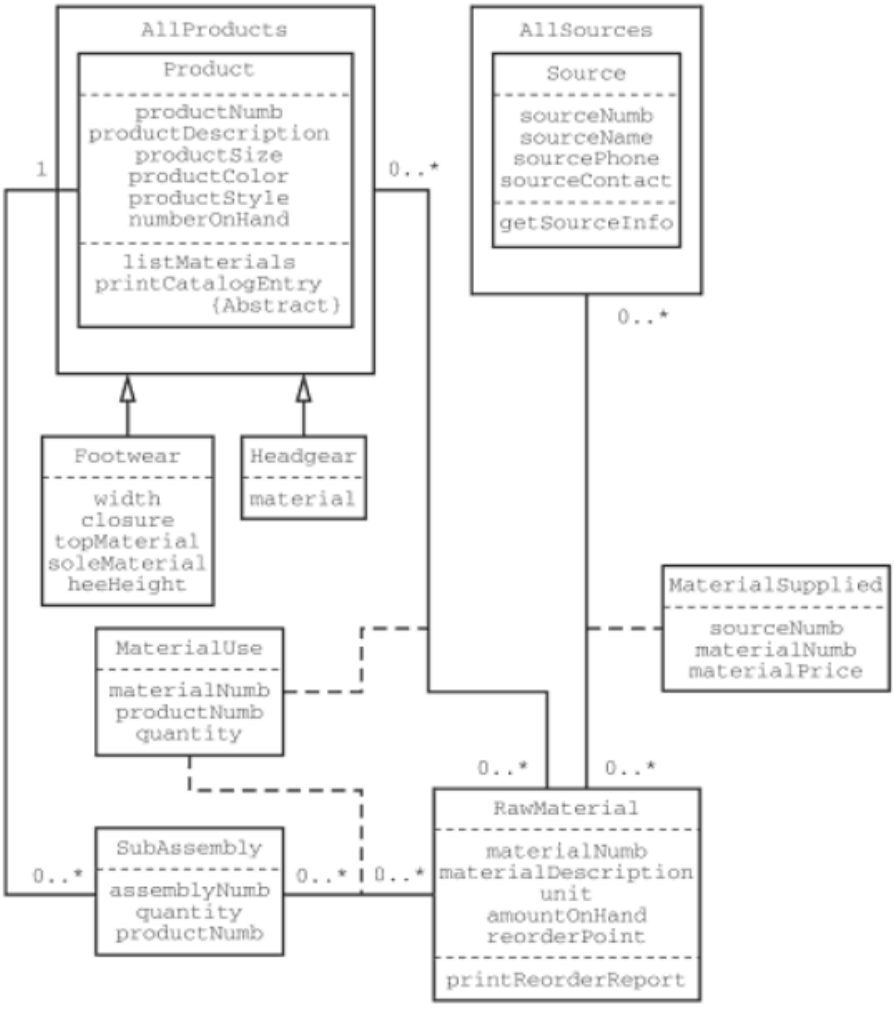


Рисунок 3.6 – Проект об’єктно-орієнтованої бази даних в нотації UML

Варто виділити наступні особливості:

- регулярний клас відображається прямокутником, розділеним на ті ж три секції, що і раніше (ім'я, атрибути, методи);

- клас-агрегат зображено прямокутником, в який записано його ім’я і прямокутник класу, об’єкти якого агрегуються. Наприклад, на рисунку 3.6 класи Product та Sourse знаходяться відповідно всередині агрегуючих класів AllProducts і AllSources;
- асоціації зображуються лініями, на кінцях яких немає ніяких символів. При цьому кратність один записується просто як «1». Якщо кратність може бути нуль або один, то це виглядає як «0..1». Якщо кратність дорівнює нулю або більше, то пишеться «0 .. *», а кратність один або більше позначається як «1 .. *»;
- наслідування представлено лінією с незафарбованою стрілкою, яка направлена в бік батьківського класу. На рисунку 3.5 класи Footwear та Headgear мають такі стрілки, які вказують на клас Product;
- замість терміна «композиційна сутність» реляційної бази даних в ООБД застосовується термін клас-асоціація. Такі класи приєднуються до відповідного зв'язку пунктирною лінією. На рисунку 3.6 кожен з класів MaterialSupplied та MaterialUse прикріплений, щонайменше, до одного зв'язку «більшість до більшості» пунктирною лінією.

3.2 Фізична модель об’єктно-орієнтованої бази даних

Одна з причин, за допомогою якої реляційні бази даних отримали велику популярність, полягає в існуванні стандартної мови маніпулювання даними (SQL). Через це в стандарті проекту ООБД визначені дві мови: Object Definition Language (ODL) це – мова визначення об’єктів, та Object Query Language (OQL) – мова запиту об’єктів. В ООБД разом вони будуть грати ту саму роль, яка відведена SQL в реляційних базах.

ODL використовується для об’явлення структури класів, включаючи сигнатури операцій та властивості. Однак реалізація операцій не входить в специфікацію ODL, оскільки для неї потрібна конкретна мова програмування. Синтаксис її схожий на SQL та доповнений концепціями успадкування, інкапсуляції та поліморфізму.

3.2.1 Сигнатура опису інтерфейсу та класу

Синтаксис ODL для оголошення інтерфейсів та класів аналогічний синтаксису розповсюджених об’єктно-орієнтованих мов програмування C++ та Java, хоча і не співпадає з ними повністю. Обмеження в ініціалізації інтерфейсу та класу скопійовані з мови C++ (приклад 3.1).

```
class ім’я_класа
{
      //Тут визначаються елементи класу.
};

interface ім’я_інтерфейсу
{
      //Тут записуються елементи інтерфейсу.
};
```

Приклад 3.1 – Сигнатура опису інтерфейсу та класу

Оголошення починається з ключового слова class або interface та означає, що саме ініціалізується. За ним розміщується ім’я класу або інтерфейсу, яке за стандартом завжди пишеться з великої літери.

Якщо клас реалізує один або декілька інтерфейсів, то імена останніх відокремлюються двокрапкою від імені класу (приклад 3.2).

```
class ім’я_класу : ім’я_інтерфейсу
{
      //Тут визначаються елементи класу.
};
```

Приклад 3.2 – Реалізація класом інтерфейсу

Якщо клас наслідується від іншого класу (розширює його), то вноситься наступне редагування (приклад 3.3).

```
class ім’я_класу extends ім’я_суперкласу
{
      //Тут визначаються елементи класу.
};
```

Приклад 3.3 – Клас наслідується від іншого класу

Таким чином, класи та інтерфейси для бази даних про університет (рисунок 2.1) виглядають так, як наведено у прикладі 3.4.

```
Interface Person {};
Interface Employee : Person {};
class Parent : Person (extent parents) {};
class Student : Person {};
class Bachelor extends Student (extent bachelors) {};
class Master extends Student (extent masters) {};
class Teacher : Employee (extent teachers) {};
class MaintenanceWorker : Employee (extent maintenanceworkers) {};
class Postgraduate : Employee (extent postgraduates) {};
```

Приклад 3.4 – Класи та інтерфейси бази даних «Університет»

3.3.2 Оголошення атрибутів та завдання зв’язків

При оголошені атрибутів вказуються його домен та ім’я. Домен може належати будь-якому із класів примітивних типів, які були розглянуто в третьому розділі, або будь-якому оголошеному в базі даних, але не інтерфейсу, оскільки з останніх не можна створити об’єктів. Синтаксис оголошення атрибутів наведено у прикладі 3.5.

```
attribute домен ім’я_атрибута;
```

Приклад 3.5 – Синтаксис оголошення атрибута

В базі даних університету потрібні два додаткових класи (адреса та номер телефону), які будуть використовуватися в якості доменів. Часткове оголошення цих класів показано у прикладі 3.6.

```
class Address
{     attribute string city;
      attribute string street;
      attribute string house;
      attribute apartment;    };

class Phone
{     attribute string code;
      attribute string number;      };
```

Приклад 3.6 – Часткове оголошення класів адреси та номера телефона

Атрибути оголошуються за можливістю на самому верхньому рівні ієрархії, щоб запобігти повторення даних. Після додавання всіх атрибутів та доменів в БД університету отримаємо класи, які наведено у прикладі 3.7.

```
Interface Person
{     attribute string id; //Ідентифікатор
      attribute string firstName; //Ім'я
      attribute string lastName; //Прізвище
      attribute string birthday; //Дата народження
      attribute string gender; //Стать
      attribute Address homeAddress; // Домашня адреса
      attribute Phone cellPhone; //Номер телефону                };

Interface Employee : Person
{     attribute Date dateHired; //Дата прийому на роботу
      attribute float salary; //Заробітня платня
      attribute integer workbookNumber; //Номер трудової книжки   };

class Parent : Person (extent parents)
{     attribute string workAddress; //Місце роботи
      attribute string employer; //Посада
      attribute Phone cellPhone; //Робочий номер телефону         };

class Student : Person
{     attribute string studentIdNumber; //Номер студентського квитка
      attribute string gradeBookNumber; //Номер залікової книжки  };

class Bachelor extends Student (extent bachelors)
{     attribute integer currentGrade; //Теперішній середній бал   };

class Master extends Student (extent masters)
{     attribute integer practiceGrade; //Оцінка за практику
      attribute string topicOfSertificationWork; //Тема прботи    };

class Teacher : Employee (extent teachers)
{     attribute string faculty; //Факультет
      attribute string department; //Кафедра
      attribute string email; //Адреса електронної пошти          };

class MaintenanceWorker : Employee (extent maintenanceworkers)
{     attribute string post; //Посада
      attribute string timetable; //Графік роботи                };

class Postgraduate : Employee (extent postgraduates)
```

```
{     attribute string faculty; //Факультет
      attribute string department; //Кафедра
      attribute string email; //Адреса електронної пошти          };

class Address
{     attribute string city;
      attribute string street;
      attribute string house;
      attribute apartment;                                        };

class Phone
{     attribute string code;
      attribute string number;                                    };
```

Приклад 3.7 – Класи з оголошенням атрибутів бази даних «Університет»

З приводу оголошення атрибутів є ще два зауваження:

- якщо в атрибуті потрібно зберігати декілька значень, то після attribute повинно йти ключове слово, яке визначає тип набору, наприклад set або bag, а за ним розміщується в кутових дужках тип даних елементів набора. Наприклад, в строчці attribute set <Phone> phoneNumbers; оголошена множина об’єктів класу Phone, і вона повідомляє про те, що атрибут phoneNumbers може містити декілька таких об’єктів. Доречи, це не те саме, що зберігання зв’язків в об’єктах Phone. В атрибуті знаходяться не ідентифікатори, а самі об’єкти Phone. Такий підхід використовується, коли точно відомо, що ніколи не доведеться шукати об’єкти, які зберігаються, або отримувати до них доступ поза контекстом їх батьків;
- щоб асоціювати з класом або інтерфейсом ключ, необхідно задати ключове слово key, за яким вказати список атрибутів, його складові. Оскільки кожний конкретний клас в базі даних університету реалізують інтерфейс Person, то достатньо зробити оголошення ключа тільки в цьому інтерфейсі і воно розповсюдиться вниз за ієрархією.

Зв’язки відрізняються від атрибутів тим, що в них зберігаються ідентифікатори об’єктів, а не самі данні. В оголошені зв’язку повинен буди вказаний інший його кінець (звернення). Загальний синтаксис кінця «один» наведено у прикладі 3.8.

```
relationship пов’язаний_клас ім’я_зв’язку
      inverse пов’язаний_клас :: ім’я_зв’язку_в_пов’язаному_класі;
```

Приклад 3.8 – Загальний синтаксис кінця «один»

Наприклад, так як в студента є лише один куратор, зв’язок оголошується так, як наведено у прикладі 3.9.

```
relationship Teacher curator
      inverse Teacher :: student;
```

Приклад 3.9 – Оголошення зв’язку «Куратор-студент» з кінця «один»

Зв’язок на кінці «більшість» ініціюється трохи інакше (приклад 3.10).

```
relationship тип_набора<пов’язаний_клас> ім’я_зв’язку
      inverse пов’язаний_клас :: ім’я_зв’язку_в_пов’язаному_класі;
```

Приклад 3.10 – Загальний синтаксис кінця «більшість»

Оскільки один вчитель-куратор має цілу групу студентів, то зворотній кінець зв’язку «Куратор-студент» наведено у прикладі 3.11.

```
relationship set <Student> student
      inverse Student ::curator;
```

Приклад 3.11 – Оголошення зв’язку «Куратор-студент» з кінця «один»

У прикладі 3.12 представлений проект бази даних з додаванням зв’язків. Додавання зв’язків, як і атрибутів, починається з самого верхнього рівня ієрархії наслідування.

```
Interface Person
{     attribute string id; //Ідентифікатор
      attribute string firstName; //Ім’я
      attribute string lastName; //Прізвище
      attribute string birthday; //Дата народження
```

```
      attribute string gender; //Стать
      attribute Address homeAddress; // Домашня адреса
      attribute Phone cellPhone; //Номер телефону
      key id;                                                    };

Interface Employee : Person
{     attribute Date dateHired; //Дата прийому на роботу
      attribute float salary; //Заробітня платня
      attribute integer workbookNumber; //Номер трудової книжки   };

class Parent : Person (extent parents)
{     attribute string workAddress; //Місце роботи
      attribute string employer; //Посада
      attribute Phone cellPhone; //Робочий номер телефону
      relationship set<Student> parent_of
            inverse Parent :: children_of;                       };

class Student : Person
{     attribute string studentIdNumber; //Номер студентського квитка
      attribute string gradeBookNumber; //Номер залікової книжки
      relationship set <Parent> student_of
            inverse Parent :: parent_of;
      relationship Teacher curator
            inverse Teacher :: student;                          };

class Bachelor extends Student (extent bachelors)
{     attribute integer currentGrade; //Теперішній середній бал   };

class Master extends Student (extent masters)
{     attribute integer practiceGrade; //Оцінка за практику
      attribute string topicOfSertificationWork; //Тема роботи    };

class Teacher : Employee (extent teachers)
{     attribute string faculty; //Факультет
      attribute string department; //Кафедра
      attribute string email; //Адреса електронної пошти
      relationship set <Student> student
            inverse Student ::curator;                           };

class MaintenanceWorker : Employee (extent maintenanceworkers)
{     attribute string post; //Посада
      attribute string timetable; //Графік роботи                };


class Postgraduate : Employee (extent postgraduates)
{     attribute string faculty; //Факультет
      attribute string department; //Кафедра
      attribute string email; //Адреса електронної пошти          };

class Address
{     attribute string city;
      attribute string street;
      attribute string house;
      attribute apartment;                                       };

class Phone
{     attribute string code;
      attribute string number;                                   };
```

Приклад 3.12 – Проект бази даних «Університет» з додаванням зв’язків

3.3.3 Додавання сигнатур операцій з параметрами та винятків

Останній елемент оголошення класу – це сигнатура операції. В кожній сигнатурі повинен бути заданий тип значення, що повертається (або void, якщо операція не повертає значення), ім’я та список параметрів. Додатково в сигнатурі можуть бути перераховані виключення, які збуджує операція. Таким чином, синтаксична конструкція наведена у прикладі 3.13.

```
тип_повертаємого_значення ім’я_операції (список_параметрів)
      raises (список_виключень)
```

Приклад 3.13 – Сигнатура оголошення операції

Список параметрів містить ініціалізацію одного або декількох значень, які або передаються операції при її визові, або повертаються нею по завершенню. Параметри бувають трьох видів:

- параметр виду in використовується тільки для передачі вхідних значень. Якщо під час виконання операції значення параметра модифікується, то це станеться непомітно для об'єкта, який викликав операцію;
- параметр виду out повідомляє адресу деякої області пам'яті. Під час виконання операція розміщує значення вихідного параметра за цією адресою, так що об'єкт, який викликав операцію, може отримати до нього доступ;
- параметр виду inout містить значення, яке передається операції при її виклику, при цьому його зміни в ході виконання операції буде видно об'єкту по завершенні процедури.

Елементи списку параметрів розділяються комами. Для кожного елемента вказується вид, тип даних та ім’я (приклад 3.14).

```
вид_параметру тип_даних_параметра ім’я_параметра
```

Приклад 3.14 – Сигнатура оголошення параметру

Наприклад, операція пошуку студента за темою атестаційної роботи повинна на вході отримувати значення теми атестаційної роботи (приклад 3.15).

```
in string topicOfSertificationWork
```

Приклад 3.15 – Оголошення вхідного параметру, який містить тему атестаційної роботи

Якщо у операції немає параметрів, то список буде порожній. В якості параметрів можна передавати наступні елементи:

- базові типи даних, наприклад цілі числа та строки;
- масиви, тобто іменовані контейнери, які містять декілька значень одного й того самого типа. Для оголошення параметру-масива після імені в квадратних дужках задається максимальна кількість елементів в масиві. Наприклад, запис integer values [5] оголошує один параметр, в якому зберігається десять цілих значень. Якщо масив вказано у якості значення, що повертається, то задавати число елементів не потрібно. Наприклад, запис integer[] ім’я_операції каже про те, що операція повертає масив цілих значень;
- структури – іменовані контейнери, які містять декілька значень різних типів;
- об’єкти.

Можна також використовувати структури та масиви, які містять об’єкти.

На перший погляд може здатися, що наявність у операції та значення, що повертається, і вихідних параметрів – це надмірність. Однак, повернення

значень в кожному з цих випадків здійснюється по-різному.

Будь-яка операція може передати один елемент даних (об'єкт, структуру або базовий тип) в якості значення, що повертається. Зазвичай в мові програмування для цієї мети використовується оператор return. Якщо об'єкт викликає операцію, що повертає значення, то він спочатку резервує область пам'яті, в яку буде поміщено значення (приклад 4.16).

```
float theSalaryPay;
```

Приклад 3.16 – Резервування області пам’яті в яку буде надходити інформація про заробітну платню викладача

Після цього викликана операція, що належить другому об’єкту, записує значення, що повертається, в цю область (приклад 4.17).

```
theGrossPay = theEmployee.computeGrossPay ();
```

Приклад 3.17 – Запис інформації про заробітну платню викладача

Якщо викликана операція зустрічає оператор return, вона копіює значення, що повертається, в область, яка зарезервована для цього об’єктом.

З вихідними параметрами інша справа. Коли об'єкт викликає операцію, що має вихідні параметри, мова програмування, яку використовують, спочатку виділяє для них пам'ять. Операції передаються їх адреси в оперативній пам'яті, які потім використовуються всередині операції так, якби вони були в ній же і оголошені. Операція розміщує значення вихідних параметрів за вказаними адресами, тому після завершення процедури об'єкт, що викликав її, може їх використовувати. Операція маніпулює адресами пам'яті безпосередньо, без копіювання (приклад 3.18).

```
void getAddress (out string city, out string street, out string house, out string apartment)
```

Приклад 3.18 – Операція, яка містить вихідні параметри

Значення, яке повертається, не вказано, про що свідчить ключове слово void. Замість цього є чотири складові адреси (місто, вулиця, будинок, квартира), які повертаються через вихідні параметри.

Значна перевага вихідних параметрів полягає в тому, що їх число не обмежене. Якщо операція повинна повернути об'єкту більш ніж одне значення, то єдиний спосіб це зробити – застосувати вихідні параметри.

Виключення – це передбачувана помилка, яка може бути оброблена операцією. У мовах програмування говориться, що операція збуджує (raise) виняток. Тому в оголошенні операції в ООБД можуть бути присутні імена винятків. Припускаючи, що імена даються зрозумілі, програміст, який кодує реалізацію операцій, знає, які виключення можуть бути в даному випадку, і враховує їх. Наприклад, будь-якій операції, «яка чекає» на вході дату, необхідно перевірити, чи потрапляє дата в розумний діапазон. Малоймовірно, що учнем університету може бути людина, яка народилася після 2010 року (приклад 3.19).

```
void setBirthDate (in Date iBirthDate) raises (invalidDate);
```

Приклад 3.19 – Операція, яка містить виключення

Якщо передбачається кілька винятків, то їх імена розділяються комами.

3.3.4 Кінцева схема бази даних

Кінцева схема бази даних університету приведена у прикладі 3.20. Можливо, під час написання прикладних програм, які працюють з цією

базою, з’явиться необхідність в додаткових елементах.

```
      Interface Person
      {     attribute string id; //Ідентифікатор
            attribute string firstName; //Ім’я
            attribute string lastName; //Прізвище
            attribute string birthday; //Дата народження
            attribute string gender; //Стать
            attribute Address homeAddress; // Домашня адреса
            attribute Phone cellPhone; //Номер телефона
            key id;
            string getId();
            string getFirstName();
            string getLastName();
            string getBirthday();
            string getGender();
            void getHomeAddress(out string oCity, out string oStreet, out
string oHouse, out string oApartment);
            void getCellPhone(out string oCode, out string oNumber);
            void changeName (in string iFirst, in string iLast);
            void changeHomeAddress (in string iCity, in string iStreet, in
string iHouse, in string iApartment);
            void changeCellPhone (in string iCode, in string iNumber);
            void setBirthday (in date iBirthday)
                  raises (invalidate);
            void setGender (in string iGender);                          };

      Interface Employee : Person
      {     attribute Date dateHired; //Дата прийому на роботу
            attribute float salary; //Заробітня платня
            attribute integer workbookNumber; //Номер трудової книжки
            date getDateHired();
            float getSalary();
            integer getWorkbookNumber();
            void setDateHired (in date iDateHired)
                  raises (invalidDate);
            void setSalary (in float iSalary)
                  raises (invalidSalary);
            void setWorkbookNumber(in integer iWorkbookNumber);          };

      class Parent : Person (extent parents)
      {     attribute string workAddress; //Місце роботи
            attribute string employer; //Посада
            attribute Phone cellPhone; //Робочий номер телефона
            relationship set<Student> parent_of
                  inverse Parent :: children_of;
            void changeWorkAddress(in string iCity, in string iStreet, in
string iHouse, in string iApartment);
            void setEmployer(in string iEmployer);
            void changeCellPhone(in string iCode, in string iNumber);
            void getWorkAddress(out string oCity, out string oStreet, out
      string oHouse, out string oApartment);
            string getEmployer();
            void getCellPhone(out string oCode, out string oNumber);
            boolean addStudent (in Student iStudent);
            boolean removeStudent (in Student iStudent);
                  raises (noSuchStudent);
            Student[] getStudents();                                     };

      class Student : Person
      {     attribute string studentIdNumber; //Номер студентського квитка
            attribute string gradeBookNumber; //Номер залікової книжки
```

```
            relationship set <Parent> student_of
                  inverse Parent :: parent_of;
            relationship Teacher curator
                  inverse Teacher :: student;
            void setStudentIdNumber(in string iStudentIdNumber)
                  raises(invalidStudentNumber);
            void setGradeBookNumber(in string iGradeBookNumber)
                  raises(invalidGradeBookNumber);
            boolean addCurator(in Teacher iTeacher)
                  raises (noSuchTeacher);
            string getStudentIdNumber();
            string getGradeBookNumber();                                    };

      class Bachelor extends Student (extent bachelors)
      {     attribute integer currentGrade; //Теперішній середній бал
            void setCurrentGrade (in string iCurrentGrade);
            integer getCurrentGrade();                                      };

      class Master extends Student (extent masters)
      {     attribute integer practiceGrade; //Оцінка за практику
            attribute string topicOfSertificationWork; //Тема работи
            void setPracticeGrade (in string iPracticeGrade);
            void        setTopicOfSertificationWork        (in        string
iTopicOfSertificationWork);
            integer getPracticeGrade();
            string getTopicOfSertificationWork();                           };

      class Teacher : Employee (extent teachers)
      {     attribute string faculty; //Факультет
            attribute string department; //Кафедра
            attribute string email; //Адреса електронної пошти
            relationship set <Student> student
                  inverse Student ::curator;
            void setFaculty(in string iFaculty);
            void setDepartment(in string iDepartment);
            void changeEmail(in string iEmail);
            string getFaculty();
            string getDepartment();
            string getEmail();
            boolean addStudent (in Student iStudent);
            boolean removeStudent (in Student iStudent);
                  raises (noSuchStudent);
            Student[] getStudents();                                        };

      class MaintenanceWorker : Employee (extent maintenanceworkers)
      {     attribute string post; //Посада
            attribute string timetable; //Графік роботи
            void setPost(in string iPost);
            void setTimetable(in string iTimetable);
            string getPost();
            string getTimetable();                                          };

      class Postgraduate : Employee (extent postgraduates)
      {     attribute string faculty; //Факультет
            attribute string department; //Кафедра
            attribute string email; //Адреса електронної пошти
            void setFaculty(in string iFaculty);
            void setDepartment(in string iDepartment);
            void changeEmail(in string iEmail);
            string getFaculty();
            string getDepartment();
            string getEmail();                                              };

      class Address
```

```
{     attribute string city;
      attribute string street;
      attribute string house;
      attribute apartment;
      void changeHomeAddress(in string iCity, in string iStreet, in
string iHouse, in string iApartment);                              };

class Phone
{     attribute string code;
      attribute string number;
      void changeCellPhone(in string iCode, in string iNumber);   };
```

Приклад 3.20 – Кінцева схема бази даних «Університет»

4 ОПТИМІЗАЦІЯ ЗАПИТІВ ДО ОБ’ЄКТНО-ОРІЄНТОВАНИХ БАЗ ДАНИХ

Перш, ніж перейти до розгляду конкретних проблем та рішень в області оптимізації запитів, розглянемо типовий для сучасних СУБД шлях обробки запиту. Обробка запиту, що надійшов в систему представленим на деякій мові запитів, складається з етапів або фаз.

На першій фазі запит, поданий на мові запитів, піддається лексичному та синтаксичному аналізу. При цьому виробляється його внутрішнє подання, що відбиває структуру запиту і містить інформацію, яка характеризує об'єкти бази даних, згадані в запиті (класи, атрибути та константи). Інформація про збережені в базі даних об'єкти обирається з екстентів бази даних. Внутрішнє представлення запиту використовується та перетворюється на наступних стадіях обробки запиту. У цій роботі не будуть розглядатися питання синтаксичного аналізу запиту, оскільки вони не пов'язані з оптимізацією. Зауважимо лише, що істотним є вибір внутрішнього уявлення, яке має бути досить зручним для подальших оптимізаційних перетворень.

На другій фазі запит піддається логічній оптимізації. При цьому можуть застосовуватися різні перетворення, які вдосконалюють початкове уявлення запиту. Серед цих перетворень можуть бути еквівалентні перетворення, після проведення яких виходить внутрішнє уявлення, семантично еквівалентний початковому (наприклад, приведення запиту до деякої канонічної формі). Перетворення можуть бути і семантичними, коли подання не є семантично еквівалентним початковому, але гарантується, що результат виконання перетвореного запиту збігається з результатом запиту в початковій формі при дотриманні обмежень цілісності, існуючих в базі даних.

Третій етап обробки запиту полягає у виборі на основі інформації, якою володіє оптимізатор, набору альтернативних процедурних планів виконання даного запиту відповідно до свого внутрішнього уявлення, яке

було отримано на другій фазі. Крім того, для кожного плану оцінюється передбачувана вартість виконання запиту за цим планом. При оцінках використовується статистична інформація про стан бази даних, доступний оптимізатору. З отриманих альтернативних планів вибирається найбільш дешевий, і саме його внутрішнє подання тепер відповідає запиту, що обробляється.

На четвертому етапі по внутрішньому поданню найбільш оптимального плану виконання запиту формується представлення плану, яке виконується. Воно може бути програмою в машинних кодах, якщо система орієнтована на компіляцію запитів в машинні коди, або бути машинно-незалежним, але більш зручним для інтерпретації. В даному випадку це не принципово, оскільки четверта фаза обробки запиту вже не пов'язана з оптимізацією.

Нарешті, на останньому, п'ятому етапі обробки запиту відбувається його реальне виконання відповідно до виконуваних планом запитів. Це або виконання відповідної підпрограми, або виклик інтерпретатора з передачею йому для інтерпретації виконуваного плану.

Зауважимо, що для нашого розгляду несуттєвий поділ процесу обробки запиту на підготовчу (що включає фази 1-4) та виконавчу (фазу 5) частини, але деякі методи оптимізації (і навіть підходи до оптимізації) досить сильно залежать від загальної організації обробки запиту. При відриві в часі процесу компіляції від реального виконання запиту оптимізатор має менше і менш достовірну інформацією, ніж в тому випадку, коли етап компіляції тісно прив'язаний з етапом виконання.

4.1 Логічна оптимізація

При класичному підході до організації оптимізаторів запитів на етапі логічної оптимізації виробляються деякі еквівалентні перетворення внутрішнього подання запиту, які «покращують» початкове внутрішнє

уявлення по деяким фіксованим в оптимізаторі планам. При цьому «покращення» носить досить умовний характер, тобто тісно пов'язано зі специфікою загальної організації оптимізатора, зокрема, з тим, як влаштована третя фаза обробки запиту. Тому досить важко привести повну характеристику та класифікацію методів логічної оптимізації.

Очевидний клас логічних перетворень складають перетворення, пов'язані з приведенням предикатів, які задає умова вибірки в даному запиті, до канонічного подання. Маються на увазі предикати, що містять операції порівняння простих значень. У загальному випадку такий предикат має вигляд «арифметичний вираз або арифметичний вираз», де арифметичні вирази лівої та правої частин в загальному випадку містять імена атрибутів класів та константи.

Якщо предикат включає в точності два атрибути різних класів (або двох різних атрибутів одного класу), то його канонічне уявлення може мати, наприклад, вид «ім'я поля або арифметичний вираз», де арифметичне вираз в правій частині містить тільки константи та друге ім'я поля.

Нарешті, для розглянутих предикатів більш загального вигляду має сенс приведення предиката до канонічного виду «арифметичний вираз або константний арифметичний вираз», де вирази правої та лівої частин також приведені до канонічного вигляду, наприклад, у виразах повністю розкриті дужки та зроблено впорядкування. Такі перетворення мають сенс для того, щоб в подальшому можна було зробити пошук загальних арифметичних виразів в різних предикатах запиту. Така робота може бути виправдана, оскільки при реальному виконанні запиту обчислення арифметичних виразів буде проводитися при вибірці кожного об’єкту, тобто потенційно дуже велике число раз.

Звичайно при приведенні предикатів до канонічного вигляду можна та потрібно робити обчислення константних виразів там, де це можливо, і позбавлятися від логічних заперечень.

4.2 Проблеми виконання та оптимізації запитів

Публікації, що стосуються оптимізації запитів до ООБД, практично відсутні. Це свідчить про недостатню розвиненість будь-яких оригінальних підходів. В ООБД використовується той самий підхід до оптимізації запитів, який використовувався в реляційних системах: формується набір альтернативних планів, оцінюється вартість кожного з них та обирається план з найменшою вартістю [7]. Можливість застосування такого підходу в СУБД ORION, O2 та в інших спирається на те, що об'єкти в цих системах в не повному обсязі інкапсульовані. Поряд з методами, в об'єктах видно і деякі атрибути, і якщо умова вибірки задана через ці атрибути, оптимізатор запитів, якому відомі внутрішня структура об'єктів і набір існуючих індексів, отримує можливість вибору. Якщо ж умову вибірки можна формулювати тільки через методи, при підходах ORION і O2 єдиним можливим способом вибірки об'єктів класу буде послідовний перегляд всіх об'єктів цього класу.

Основна проблема з оптимізацією запитів до ООБД пов'язана з розширюваністю набору типів в такий БД. Кожен новий тип вводить власну алгебру, невідому оптимізатору запитів. Наприклад, оптимізатор не має інформації про можливу комутативність двох операцій типу та ін. Можливому вирішенню проблеми оптимізації могло б сприяти формальне визначення алгебраїчних властивостей операцій типу при його розробці. Приблизно такий підхід застосовується в оптимізаторах, заснованих на наборі правил, що застосовуються в розширюваних СУБД.

Якщо в системі для програмування методів застосовується мова досить високого рівня, то при компіляції запиту могло б проводитися часткове обчислення методів, які викликаються, з подальшим перетворенням запиту до виду, коли умови визначені на атрибутах збережених класів. Після цього можна було б виконувати звичайну оптимізацію запиту. Слід зауважити, що це не було б порушенням інкапсуляції об'єктів, оскільки оптимізатор є частиною системи, для якої внутрішня організація об'єктів БД відкрита.

ВИСНОВКИ

Об'єктно-орієнтована база даних – це база даних, в якій дані моделюються у вигляді об'єктів, їх атрибутів, методів та класів.

Для того, щоб спроектувати об'єктно-орієнтовану базу даних, необхідно в першу чергу вивчити предметну область, в якій буде виконуватися робота БД, врахувати основні процеси, вимоги та побажання замовника та майбутніх користувачів. Наступним етапом є концептуальне проектування БД, інакше кажучи, конструювання моделі, що не залежить від будь-яких фізичних особливостей реалізації. Після того, як концептуальне проектування було завершено, настає логічне проектування БД, тобто конструювання інформаційної моделі на основі конкретно виділених, існуючих типів даних. Надалі логічна модель послужить основою для фізичної моделі. Логічна модель дозволяє розробнику всебічно вивчити поведінку БД та конкретних даних, прийняти деякі рішення щодо вибору засобів фізичної її реалізації.

Першою формалізованою та загальновизнаною моделлю даних була реляційна модель Коада. До ролі стандарту для зображення логічних моделей даних найближче наблизилась уніфікована мова моделювання (Unified Modeling Language, UML). У її нотації об'єднані багато елементів, які розглядалися в багатьох методиках.

Одна з причин, за допомогою якої реляційні бази даних отримали велику популярність, полягає в існуванні стандартної мови маніпулювання даними (SQL). Через це в стандарті проекту ООБД визначені дві мови: Object Definition Language (ODL) це – мова визначення об’єктів, та Object Query Language (OQL) – це мова запиту об’єктів. В ООБД разом вони будуть грати ту саму роль, яка відведена SQL в реляційних базах.

СУБД – це комплекс програм, що дозволяють створити базу даних та маніпулювати даними (вставляти, оновлювати, видаляти та обирати).

Система забезпечує безпеку, надійність зберігання та цілісність даних, а також надає засоби для адміністрування БД.

Зараз ведеться дуже багато експериментальних та виробничих робіт в області об'єктно-орієнтованих СУБД. Найбільше університетських робіт, які в основному носять дослідницький характер. Але вже декілька років тому відзначалося існування щонайменше тринадцяти комерційно доступних систем ООБД. На жаль, з приводу цих систем практично не доступно жодних публікацій.

Підводячи підсумки, можна відзначити, що об'єктно-орієнтовані бази даних дуже зручні в реалізації інформаційних структур практично будь-якої компанії або підприємства. Головною задачею стає виділення основних сутностей та їх атрибутів, а також сутнісних відносин, що вимагає ознайомлення з об'єктною областю застосування БД. Також необхідно пам'ятати, що на етапах логічного та фізичного проектування всі рішення необхідно погоджувати з замовником.

Сучасні об'єктно-орієнтовані мови програмування дозволяють з легкістю створювати класи, які реалізують суті, а об’єктно-орієнтовані системи управління базами даних, як правило, вже включають в себе необхідні бібліотеки. При знанні певних правил побудови об'єктно-орієнтованих баз даних та дотриманні зазначених порад, побудова будь-якої ООБД зводиться до послідовного виконання виділених етапів, які забезпечують успішне завершення створення необхідної бази даних.

Якщо є необхідність зберігати поведінку або дані дуже складні, тобто складаються з декількох примітивних типів даних, – слід розглянути можливість використання об'єктно-орієнтованої бази даних. В іншому випадку, в більшості випадків переважніше використовувати реляційну базу даних.

Наведене вище може бути використано як орієнтир при виборі архітектури бази даних.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. S. L. Pfleeger. Software Engineering: theory and practice (Second Edition) –Upper Saddle River – 2001 – 716 p.

2. L. Cardelli, P. Wegner. On Understanding Types, Data Abstraction, and Polymorphism – ACM Computing Surveys. – 1985. – 522 p.

3. Connolly T., Begg C. Database Systems Third Edition – Addison Wesley. – 2002. – 576 p.

4. C. J. Date. SQL and Relational Theory: How to Write Accurate SQL Code 2nd edition – O'reilly Media, Inc. – 2012. – 428 p.

5. Embley, David W. Object Database Development – Addison Wesley. – 1998. – 887 p.

6. 1. Connolly T., Begg C. Database Systems A Practical Approach to Design, Implementation, and Management: Global Edition – Pearson Education, 2014. – 1440 p.

7. 2. Elmasri R., Navathe S.B. Fundamentals of Database Systems – Addison Wesley, 2016. – 1272 p.

8. Atkinson M., Bansilhon F., DeWitt D., Dittrich K., Maier D., Zdonik S.. The Object-Oriented Database System Manifesto : 1st Int. Conf. Deductive and Object-Oriented Databases, Kyoto, Japan, Dec. 4-6, 1989.

9. Dietrich W., Urban D. Fundamentals of Object Databases: Object-oriented and Object-relational Design – Morgan & Claypool Publishers. – 2011. – 151 p.

10. Б. Мейер «Почувствуй класс. Учимся программировать хорошо с объектами и контрактами» – Национальный открытый ун-т "ИНТУИТ". – 2011. – 775 с.

11. Г. Буч. Объектно-ориентированный анализ и проектирование с примерами приложений – И. Д. Вильямс. – 2008. – 718 с.

12. М. Вайсфельд «Объектно-ориентированное мышление» – Питер. – 2014. – 255 с.

13. Харрингтон Д. Проектирование объектно-ориентированных баз данных – ДМК Пресс, 2001. – 272 с.

14. Новиков Б. А., Горшкова Е. А. Основы технологий баз данных. – ДМК Пресс. – 2019. – 240 с.

15. Філімончук Т.В., Гонтарєва Д.В. Об’єктно-орієнтовані бази даних. Проблеми інформатизації: Тези доповідей восьмої міжнародної науково-технічної конференції. Том 2, секція 4. Черкаси: ЧДТУ; Харків: НТУ «ХПІ»; Баку: ВАЗС АР; Бельсько-Бяла: УТіГН. – 2020. – с. 52.

16. Філімончук Т.В., Мартовицький В.О., Гонтарєва Д.В. Транзакції і блокування, рівні ізольованості транзакцій. Системи управління, навігації і зв’язку: збірник наукових праць. П.: Національний університет «Полтавська політехніка імені Юрія Кондратюка». – 2020. Вип. 4 (62), – 73-76.