УДК 004.652

Терещенко В. М., Волошин С. Г.

ПІДХІД ЩОДО ПОБУДОВИ ОБ'ЄКТНО-РЕЛЯЦІЙНОЇ БАЗИ ДАННИХ

Роботу присвячено одному з підходів щодо реалізації віртуальної об'єктно-орієнтовної бази даних із властивістю наслідування даних. Цей підхід є динамічним і дозволяє під час роботи в системі редагувати чи додавати сутності. Проведено аналіз схем даних для опису та зберігання об'єктів і запропоновано узагальнювальну схему, яка поєднує декілька підходів. Формалізовано отриману об'єктно-реляційну модель. Запропоновано варіант реалізації мови запитів до об'єктно-орієнтовних даних.

Ключові слова: реляційні бази даних, об'єктно-орієнтовні бази даних, об'єктно-реляційна модель.

Вступ

Формулювання проблеми. При розробці інформаційних систем для опису об'єктів з реального світу зручно використовувати об'єктно-орієнтовний підхід (ООП). Найпоширенішим методом зберігання даних є використання реляційної системи управління базами даних (РСУБД). Проблема полягає в семантичному провалі між об'єктно-орієнтовним та реляційним підходами. Зокрема, важливою для опису даних є така властивість ООП, як наслідування, яка відсутня в реляційній моделі. Для подолання цього провалу зручно використовувати не реляційну, а об'єктну базу даних і описувати дані в рамках ООП. Об'єктно-орієнтовні СУБД (ООСУБД) або об'єктно-реляційні СУБД (ОРСУБД) не набули широкої популярності через відсутність єдиних стандартів. Застосування однієї з постреляційних СУБД вимагає використання специфічної можливості конкретної бази даних, а отже, позбавляє кінцеву систему універсальності. Тому зручно мати певну універсальну надбудову над РСУБД, яка б дозволяла використовувати ООП для опису та маніпулювання даними.

Аналіз останніх досліджень та публікацій. Для поєднання об'єктно-орієнтовного підходу та реляційної бази даних набув широкої популярності підхід під назвою ORM (англ. *object-relational mapping*, укр. *об'єктно-реляційна проекція*) – технологія програмування, яка пов'язує реляційну базу даних із концепціями об'єктно-орієнтовних мов програмування, створюючи «віртуальну об'єктну базу даних». Більшість публікацій присвячено саме цьому підходу. Так, Ляховец С. В. у роботі [1] використовує ідею об'єктно-реляційного адаптера для геоінформаційної системи, що, власне, не відрізняється від ORM. Деякі недоліки ORM-підходу описані в роботі [2], де автори як альтернативу пропонують використовувати метод функціональної декомпозиції для побудови системи.

Якщо не зважати на недоліки конкретних реалізацій ORM, то основними недоліками самого підходу є *відсутність динамічності та втрата швидкодії.* Зокрема, підхід полягає в генерації програмного коду, що вимагає кожного разу перекомпіляції самої програми і позбавляє гнучкості та динамічності саму реалізацію. Натомість у [3] автори пропонують використовувати метадані для динамічної генерації класу та опису проекції для ORM-систем. Проте це не дозволяє змінювати систему в реальному часі та вимагає перезапуску системи. Окрім того, дані об'єкту завантажуються повністю, що призводить до надлишкових операцій при одержанні частини атрибутів. У роботі [4] автор намагається позбутися цього недоліку частковим завантаженням атрибутів, пропонуючи два методи завантаження атрибутів: завантаження атрибутів одного об'єкту та завантаження атрибутів колекції об'єктів. Але питання автоматичного вибору методу залишається відкритим. У роботі [3] автори також вивчають питання наслідування для побудови об'єктно-реляційної проекції. Амбер та Скот [5] детально вивчають питання об'єктно-реляційної проекції та описують можливі реалізації наслідування. У роботі [6] продовжується вивчення проблеми та аналізуються можливі проблеми таких реалізацій. Натомість у роботі [7] запропоновано формалізацію об'єктно-реляційної моделі даних, розглянуто деякі особливості проектування баз даних з використанням об'єктно-реляційної моделі. В роботі [8] проведено аналіз мов запитів до об'єктної бази та проаналізовано їхні основні недоліки.

Метою нашого дослідження є можливість застосування реляційної СУБД (РСУБД) для створення об'єктно-реляційного середовища, що дозволить динамічно описувати об'єктно-орієнтовні структури з можливістю наслідування.

1. Формулювання задачі

ОРСУБД – це РСУБД, що підтримує певні технології, які реалізують об'єктно-орієнтовний підхід. Відмінність від ООСУБД полягає в тому, що ОРСУБД – це РСУБД з певною надбудовою, яка дозволяє працювати з базою на вищому рівні абстракції. Прикладами таких СУБД можуть бути PostgreSQL, Oracle Database та MSSQL. Але, як правило, в таких базах об'єктно-орієнтовний підхід поширюється на типи даних, екземпляри яких (так звані persistent objects) в подальшому можуть бути збережені в базі.

При проектуванні бази даних, як правило, використовують семантичну модель даних, зокрема ER-модель (сутність-зв'язок). Об'єктно-орієнтовна модель спростила б перехід від проектування до безпосередньої реалізації бази та дозволила б проектувати схему паралельно із впровадженням. Це можливо, оскільки, на відміну від реляційних моделей, об'єктно-орієнтовні оперують спільними поняттями з ER-моделлю: сутність, атрибут і зв'язок.

Отже, задача полягає в тому, щоб на основі РСУБД розробити середовище з наступними основними властивостями:

- **Наслідування даних** – опис та наслідування спільних характеристик в спільному класі.
- **Універсальність** – за основу повинна підходити будь-яка або майже будь-яка РСУБД.
- **Динамічність** – опис під час роботи нових класів.
- **Швидкодія** – система повинна залишатися достатньо швидкодійною для практичного застосування.
- **Ґрунтування на метаданих** – ця властивість дасть значні переваги при застосуванні.
- **Існування механізму розподілу прав.**

Основою (ядром) такої системи може бути реляційна СУБД як найбільш розповсюджена і яка має потужну теоретичну базу та затверджені стандарти, а також об'єднує значну кількість спеціалістів. Але разом з тим не повинно бути прив'язки до конкретної СУБД, оскільки це значною мірою обмежить область застосування. Наслідування даних має велике значення при описі об'єктів предметної області, оскільки позбавляє необхідності постійного опису спільних характеристик та дає можливість маніпулювати узагальненими даними. Наприклад, при розробці системи документообігу це дозволяє описати спільний клас. У клас *універсальний документ* можна винести такі спільні поля, як дату та номер, а конкретний тип документу буде унаслідуватися та реалізовувати свої власні специфічні характеристики. І разом з тим це дозволить отримати єдиний реєстр документів усіх типів простим запитом. Оскільки процес розробки інформаційних систем є ітеративним та поетапним, то динамічність є значною перевагою і дозволяє вносити зміни паралельно з роботою. Це в свою чергу забезпечить оперативність розробки та виправлення помилок, що позитивно вплине на кінцевий результат. Створена система повинна мати достатню швидкодію для практичного застосування. Як правило, вузьким місцем таких систем є отримання набору даних згідно з певною умовою. Очевидно, що в цьому випадку еталоном є швидкодія РСУБД, на якій побудована система, а отже, чим ближче час виконання запиту до часу виконання звичайного SQL-запиту, тим краще. Існування метаданих дасть можливість автоматизувати наступні операції з даними: від автоматичної побудови відображення даних до організації фільтрування даних та системи розподілу прав.

Задача розподілу прав постає майже завжди при розробці багатокористувацьких систем. І якщо вже будується надбудова до РСУБД, то логічно було б включити сюди і систему розподілу прав. Тим більше що для включення такої підсистеми є всі складові: метадані та модуль генерації SQL-запитів. Отже, під час генерації запитів система, спираючись на метадані та використовуючи схему розподілу прав, може вносити свої корективи в кінцевий запит, що в свою чергу вплине на остаточний результат. Одну з моделей динамічного розподілу прав, яка б задовольняла вимоги розроблюваної системи, описано у роботі [9]. Як правило, ООСУБД мають 2 рівні: структура та поведінка. У нашій роботі зосереджено увагу на структурі об'єктної бази даних.

2. Підходи щодо структури бази даних

Існує декілька підходів щодо проекції ієрархічної структури класів на реляційну модель [5]:

- Проекція ієрархічної структури на одну таблицю.
- Проекція кожного класу на власну таблицю.
- Проекція кожного конкретного класу на таблицю.
- Проекція класів на загальну структуру таблиць.

Перший варіант полягає в тому, що створюється одна спільна таблиця для всіх класів. Створюються всі можливі атрибути всіх класів і атрибут, який вказує на клас. Такий підхід виправдовує себе лише при невеликій кількості класів,

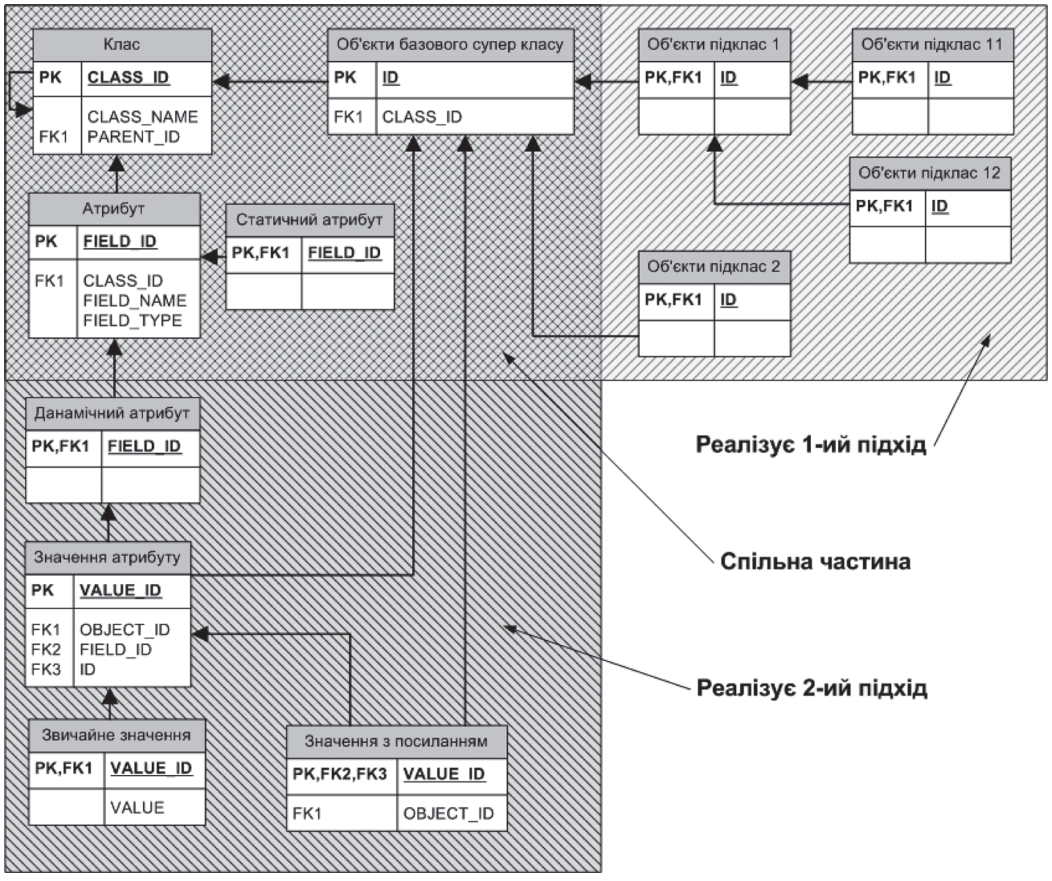
і при такому підході ми отримаємо дуже розріджені дані. Для більшості інформаційних систем такий варіант не підходить, оскільки кількість класів часто досягає сотень та тисяч.

Наступний підхід схожий на попередній, але для кожного батьківського класу створюється власна таблиця. Проте за умови необхідності мати одного спільного батька, структура зводиться до першого варіанту, а отже, не підходить.

Далі розглянемо підхід, який полягає в тому, що на кожен конкретний клас створюється власна таблиця, яка містить атрибути лише конкретного класу. Наслідування реалізується у вигляді зв'язку 1 до 1 між таблицею предка та нащадка. Екземплярами класу є записи в таблиці. Такий підхід відповідає вимогам, але має певні недоліки. Для створення нового класу необхідно змінювати схему бази, що може не кожна СУБД під час активної роботи з базою. До переваг такого підходу слід віднести достатню швидкодію при отримані табличного набору даних.

Останній підхід є найбільш універсальним. Його суть полягає в тому, що для збереження об'єктів необхідно лише декілька таблиць: в одній зберігаються заголовки об'єктів, в іншій – значення атрибутів. Це позбавляє необхідності вносити зміни в схему бази та надає динамічності системі. Але такий підхід має значний недолік – часові та системні витрати на отримання звичної таблиці з даними. Це обумовлено тим, що для кожного атрибуту, що потрапляє в результат, необхідно виконувати операцію з'єднання в SQL-запиті. Така структура даних більше підходить для систем, в яких треба динамічно описувати велику кількість різнотипних об'єктів, наприклад для організації бази знань. Проте такий підхід має право на існування і надасть системі додаткової гнучкості.

Ми пропонуємо узагальнену схему, яка поєднує переваги двох попередніх (мал. 1). В її основі лежать метадані, які описують класи та їхні атрибути. Атрибути поділяються на два типи: статичні та динамічні. Від типу залежить спосіб зберігання та спосіб маніпуляції ними. Всі класи системи успадковані від єдиного батька. Це дозволяє мати єдиний унікальний нумератор всіх об'єктів та винести всі спільні атрибути в одну таблицю. Для статичних полів створюється таблиця зі зв'язком 1 до 1 між таблицею батька та нащадка. Значення всіх динамічних атрибутів зберігаються в одній таблиці «Значення атрибутів». Залежно від типу значення динамічного поля дані зберігаються у двох таблицях: таблиця для простих значень і таблиця для значень із



Мал. 1. Схема реалізації об'єктної СУБД в реляційній СУБД

посиланням. Це забезпечує цілісність даних на рівні схеми СУБД. Оскільки всі класи мають спільного предка, то зовнішній ключ для значення з посиланням буде вказувати на таблицю базового суперкласу. Попередні два варіанти – це поодинокі випадки запропонованого підходу (мал. 1). Існування динамічних полів дозволяє вносити зміни в систему не лише фахівцям, а й звичайним користувачам. Для роботи з базою розробнику необхідна або SQL-подібна мова, або спеціалізоване програмне API. Це дозволить приховати складну структуру бази даних від розробника та дозволить маніпулювати об'єктами предметної області.

3. Динамічна об'єктно-орієнтовна надбудова над РСУБД

Формалізація об'єктно-реляційної моделі

На сьогодні існує декілька підходів щодо побудови об'єктно-реляційної моделі даних (ОРМД). Один із них запропоновано в роботі [7], де описано складну структуру доменів та відношень. Ми пропонуємо розширити множину доменів, множину значень посилань на інші об'єкти. Вводимо поняття наслідування даних. Формально реляційну модель даних (РМД), запропоновану Коддом у 70-х роках [10], можна подати у вигляді:

$$M = \langle D, R, O, I \rangle, \qquad (1)$$

де $D = \{D_1, D_2, ..., D_n\}$ – множина доменів, $R = \{R_1, R_2, ..., R_m\}$ – множина відношень; O – операції визначені над відношеннями; I – обмеження цілісності.

У роботі [7] запропоновано наступну формалізацію об'єктно-реляційної моделі:

$$M_{OR} = \langle OE, OS \rangle, \qquad (2)$$

де OE – об'єктні сутності (object entity), OS – операційна специфіка (object specification).

В свою чергу кожну об'єктну сутність можна подати у вигляді $OE = \langle D', R' \rangle$, де D', R' – сукупність доменів та відношень відповідно. Але, на відміну від РМД, сукупність доменів та відношень можуть мати складну структуру. За аналогією до попередніх підходів (1), (2), запропоновану об'єктно-реляційну модель можна подати у вигляді:

$$M_{OR} = \langle C, O, OS' \rangle, \qquad (3)$$

де $C = \{c_B, c_U, c_1, c_2, ..., c_k\}$ – множина класів, c_B – базовий суперклас, c_U – клас користувачів системи; $O = \{o_1, o_2, ..., o_m\}$ – множина об'єктів або об'єктні сутності, множина об'єктів ділиться на підмножини об'єктів певного класу O_i: $\bigcup_1^k O_i = O$; OS' – операційна специфіка.

В свою чергу множина об'єктів є множиною кортежів у відповідній реляції класу. Реляції визначені на множині доменів $D'' = \{D''_1, D''_2, ..., D'_n\}$. Домен – це підмножина певного типу. Окрім стандартних типів (число, дата, рядок), домен може бути підмножиною множини об'єктів певного класу O_i: $\bigcup_1^k O_i = O$. Інакше кажучи, основу моделі складають об'єкти та класи. Класи визначають структуру даних та складаються з атрибутів. Значення атрибуту може бути як звичайне, основане на типі (число, дата, рядок), так і значенням-посиланням на інший об'єкт. Об'єкти є екземплярами класу. Класи можуть наслідувати атрибути іншого класу. Дозволяється наслідування лише від одного класу. Глибина ієрархії наслідування класів обмежена певними технічними обмеженнями конкретної РСУБД, на якій ґрунтується система. В системі наперед визначена певна множина системних класів. Кожен клас повинен бути нащадком якогось класу, окрім базового суперкласу. Базовий клас знаходиться в основі ієрархії та містить системні атрибути, серед яких унікальний ідентифікатор об'єкту. Особливістю моделі є те, що атрибути поділяються на два види: статичні та динамічні. Вид атрибуту впливає на спосіб зберігання даних у базі та на спосіб маніпуляції цими даними.

Введемо наступну формалізацію основних елементів. Поняття класу, об'єкту й успадкування будемо визначати аналогічно до цього поняття в ООП, але з ухилом на структурний аспект [11].

Поняття класу. Клас визначає абстрактні характеристики певної сутності. Характеристики визначаються у вигляді атрибутів. Атрибути можуть мати як простий тип (число, дата, рядок), так і посилатися на екземпляр певного класу.

Поняття об'єкту або об'єктної сутності. Об'єкт або Об'єктна сутність – окремий екземпляр класу. Сукупність значень атрибутів окремого об'єкту називається станом. Власне об'єкт є кортежем у відношенні визначеного на множині доменів. Множини доменів визначені в атрибутах.

Поняття наслідування або успадкування. В деяких випадках клас може мати підкласи, спеціалізовані версії класу. Підкласи успадковують атрибути своїх батьківських класів і можуть вводити свої власні. Клас, від якого успадковується інший клас, є **суперкласом**. Безпосереднім суперкласом може бути лише 1 клас.

Базовий суперклас. Клас, який не успадковується від жодного класу і є безпосереднім чи опосередкованим суперкласом для будь-якого іншого класу.

$c_B = \langle A_B \rangle$ – базовий суперклас, $A_B = \{ID, Name, Note, CreateDate, State, Owner\}$ – атрибути базового суперкласу.

Клас користувачів. Окремий системний клас, який визначає користувачів системи.

$c_U = \langle c_B, A_U \rangle$ – клас користувачів системи, є нащадком базового, $A_U = \{Login, Password\}$.

Решта класів має такий вигляд: $c_i = \langle cs, A_i \rangle$ – клас, де $cs \in C$ – суперклас, $A_i = \{a_{i1}, a_{i2}, ..., a_{ij}\}$ – множина атрибутів класу, кожен атрибут має тип і визначає домен D_{ij}, що обмежує допустиму множину значень.

Всі атрибути діляться на два види: *SA – статичні* та *DA – динамічні,*

$$SA \cup DA = A,\ SA \cap DA = \varnothing,$$

де $A = \bigcup_{1}^{k} A_1$.

На множині класів визначено функцію $y = Super(x)$; $x, y \in C$, що повертає суперклас для вказано класу.

$$Super(c_B) = \varnothing,\ Super(c) \neq \varnothing,\ \forall c \in C\ \&\ c \neq c_B$$

Множина всіх суперкласів певного класу – це множина класів, від яких успадковуються атрибути безпосередньо чи опосередковано. Цю множину можна визначити таким чином:

$$SUPER(c) = \{Super(c)\} \cup SUPER(Super(c)),$$
$$SUPER(c) \subseteq C,\ c \in C$$

Загальна множина атрибутів класу складається з безпосередніх своїх атрибутів та успадкованих. Нехай множина безпосередніх атрибутів класу $ATTR(c) = A$, $c \in C$, тоді множина всіх атрибутів:

$$ALLATTR(c) = ATTR(c) \cup ALLATTR(Super(c)),$$
$$ALLATTR(\varnothing) = \varnothing$$

Операційну специфіку нашої моделі *OS*" будемо визначати аналогічно до моделі (2) [7], а саме: визначимо дві операції: *EVOLVE* (розгортання) та *CONVOLVE* (згортання). Ці операції дозволять подати об'єктні дані в реляційному вигляді та навпаки.

У класичній РМД визначено наступні операції:
- Вибірка (SELECT),
- Проекція (PROJECT),
- Добуток (PRODUCT),
- Об'єднання (UNION),
- Перетин (INTERSECT),
- Віднімання (DIFFERENCE),
- Об'єднання (JOIN).

Для застосування цих операції над множинами об'єктів певного класу необхідно на першому етапі виконати операції *EVOLVE*, а після виконання операції реляційної алгебри застосувати операцію *CONVOLVE*.

При застосуванні операції *EVOLVE* (розгортання об'єктів в реляційний вигляд) у результуючу множину атрибутів реляції потрапляють як безпосередні атрибути класу, так і успадковані. Також до множини атрибутів потрапляють атрибути класів, на які посилаються власні атрибути. Наприклад, маємо такі класи:

Базовий клас *= {*
ID : ***Identity****;*
Name : ***String****;*
Note: ***String****;*
CreateDate: ***Date****;*
State : ***Integer****;*
Owner : ***Користувач****}*

Користувач *:* ***Базовий клас*** *= {*
Login : ***String****;*
Password: ***String****; }*

Місто *:* ***Базовий клас*** *= {*
Код телефону : ***String****;*
}
Вуз *:* ***Базовий клас*** *= {*
Місто: ***Місто****;*
}

Розгорнутий вигляд об'єктів класу *Вуз* буде мати наступні атрибути:

БазовийКлас.ID,
БазовийКлас.Name,
БазовийКлас.Note,
БазовийКлас.CreateDate,
БазовийКлас.State,
Місто.Код Телефону
Місто.БазовийКлас.Name,

Звісно, операцію *EVOLVE* можна оптимізовувати так, що в результаті розгортання будуть лише ті атрибути, які необхідні для виконання операції. Типово атрибути формуються з безпосередніх атрибутів класу. Операція згортання *CONVOLVE* може виконуватись некоректно, проте не завжди в цьому є потреба. Звичним для сприйняття інформації є таблиця – візуальне відображення реляційних даних, але, крім візуального відображення, необхідно виконувати редагування даних, де без об'єктно-орієнтовного виду не обійтися. Для цього введемо операцію *CONVOLVE_BY_ID*, яка буде повертати об'єкт за унікальним ідентифікатором. Таке перетворення можливо виконати завжди.

Мова запитів. Вже існує декілька мов запитів і програмних інтерфейсів для об'єктно-реляційних баз даних. Це, наприклад, OQL [12], JDOQL [13], db4o SODA [14]. Власне, мова запитів для об'єктних даних (ObQL – object query

language) дуже схожа на мову запитів SQL і є абстрактною надбудовою. Відмінність полягає в тому, що OQL оперує поняттям «об'єкт», а не «реляція». Відмінність у синтаксисі полягає в існуванні комплексних атрибутів.

Твердження 1. *Мова запитів до об'єктних даних (ObQL) зводиться до SQL.*

Для доведення твердження достатньо показати реалізацію особливостей основних операцій ObQL через SQL. Покажемо це на прикладах наступних операцій.

Операція вибірки (обмеження селекції). Операції вибірки в ObQL складається з кількох операцій: з'єднання, перейменування атрибутів та вибірка.

Приклад: Нехай маємо такі класи: Базовий -> Люди -> Працівники

ObQL:

```
SELECT * FROM Працівники
```

SQL:

```
SELECT *, FROM Працівники
INNER JOIN Люди ON Люди.ІД = Працівники.ІД
INNER JOIN Базовий ON Базовий.ІД = Працівники.ІД
```

SQL при наявності динамічних полів:

```
SELECT
   *
  , ДинПоле1.Значення AS НазваПоля1
  , ДинПоле2.Значення AS НазваПоля2
…….
  , ДинПолеN.Значення AS НазваПоляN
FROM Працівники
INNER JOIN Люди ON Люди.ІД = Працівники.ІД
INNER JOIN Базовий ON Базовий.ІД = Працівники.ІД
INNER/LEFT JOIN Значення поля AS ДинПоле1 ON Базовий.ІД = ДинПоле1.Об'єкт_ІД AND Значення поля.НазваПоля = 'ДинПоле1'
INNER/LEFT JOIN Значення поля AS ДинПоле2 ON Базовий.ІД = ДинПоле2.Об'єкт_ІД AND Значення поля.НазваПоля = 'ДинПоле2'
……..
INNER/LEFT JOIN Значення поля AS ДинПолеN ON Базовий.ІД = ДинПолеN.Об'єкт_ІД AND Значення поля.НазваПоля = 'ДинПолеN'
```

Комплексні атрибути

Приклад:

```
SELECT Компанія.Територія.Код FROM Компанія
```

Запит перетворюється в такий SQL:

```
SELECT Територія.Код FROM Компанія
INNER JOIN Територія ON Територія.ІД = Компанія.Територія_ІД
```

SQL при наявності динамічних полів:

```
SELECT ДинПоле1.Значення AS Код FROM Компанія
INNER/LEFT JOIN Значення поля AS ДинПоле1 ON Компанія. Територія_ІД = ДинПоле1.Об'єкт_ІД AND Значення поля.НазваПоля = 'Код'
```

4. Висновки

У роботі розглянуто проблеми поєднання об'єктно-орієнтовної та реляційної моделей. Проаналізовано останні дослідження з цієї тематики. Показано недоліки існуючих рішень, зокрема ORM. Зроблено огляд існуючих схем для зберігання об'єктно-орієнтовних ієрархічних даних та проаналізовано їхні переваги та недоліки. Запропоновано оригінальну узагальнювальну схему та показано її переваги. Формалізовано запропоновану об'єктно-реляційну модель та модифікацію мови запитів SQL для зручної роботи з об'єктними даними.

1. Ляховец С. В. Сопряжение объектно-ориентированной ГИС с реляционной базой данных / Ляховец С. В. // Сб. науч. трудов 9-й Междунар. научной конф. «Теория и техника передачи, приема и обработки информации». – Харьков : ХНУРЭ, 2003. – С. 268–269.
2. Бойко Ю. В. Формування логіки внутрішньої міжрівневої взаємодії в багатоланковій системі / Бойко Ю. В. Погорілий С. Д., Коваленко О. В. // Проблеми програмування. – 2008. – № 2–3 (спец. вип.). – С. 91–96.
3. Чупрынин А. Д. Использование метаданных для динамического расширения моделей предметных областей в программных системах / А. Д. Чупрынин, И. Б. Гавсиевич, В. В. Казимир // Вісник Хмельницького національного університету. – 2009. – № 3. – С. 171–179.
4. Suchal J. Transparently Mapping Objects to Relational Databases with AspectJ / Ján Suchal // 3rd Student Research Conference in Informatics and Information Technologies Bratislava, April 18, 2007 : Proceedings in Informatics and Information Technologies. – Bratislava : STU v Bratislave FIIT, 2007. – S. 33–40. – ISBN 978-80-227-2631-3.
5. Scott A. Mapping Objects to Relational Databases: O/R Mapping in Detail [Електронний ресурс] / Ambler Scott. – Режим доступу : http://www.ambysoft.com/essays/mappingObjects.html. – Назва з екрана.
6. Somma R. The O/R Problem: Mapping Between Relational and Object-Oriented Methodologies, Strategic Planning for Database Systems [Електронний ресурс] / Ryan Somma. – Режим доступу : http://ideonexus.com/wp-content/uploads/2010/01/RyanSomma-CIS515Project.pdf. – Назва з екрана.
7. Касаткина Н. В. Об одном подходе к построению объектно-реляционной модели данных / Касаткина Н. В., Танянский С. С., Чапланова Е. Б. // Збірник наукових праць Військового інституту Київського національного університету імені Тараса Шевченка. – 2009. – Вип. 20. – С. 20–24.

8. Cook W. Native Queries for Persistent Objects [Електронний ресурс] / William R. Cook, Carl Rosenberger // Dr. Dobb's Journal (DDJ), February 2006. – Режим доступу : http://www.drdobbs.com/cpp/184406432. – Назва з екрана.
9. Терещенко В. Розподіл прав доступу в облікових системах / Терещенко В., Волошин С. // Вісник Київського університету. Серія : Фізико-математичні науки. – 2008. – Вип. 3. – С. 180–184.
10. Codd E. A Relational Model of Data for Large Shared Data Banks / E. F. Codd // Communications of the ACM. – June, 1970. – Volume 13, Number 6. – P. 377–387.
11. Armstrong D. The Quarks of Object-Oriented Development / Debrah J. Armstrong // Communications of the ACM. – Fabruary, 2006. – Volume 49, Issue 2. – P. 123–128. – ISSN:0001-0782.
12. Cattell R. G. G. The Object Data Standard ODMG 3.0. / R. G. G. Cattell, D. K. Barry, M. Berler, J. Eastman, D. Jordan, C. Russell, O. Schadow, T. Stanienda, and F. Velez, ed. – Morgan Kaufmann; 1 edition, 2000. – 280 p.
13. Jordan D. Java Data Objects (JDO) / David Jordan, Craig Russell. – O'Reilly Media; 1 edition, 2003. – 380 p.
14. DB4OBJECTS [Електронний ресурс]. – Режим доступу : http://www.db4o.com. – Назва з екрана.
15. Терещенко В. Аналіз сучасних програмних платформ систем автоматизації / Терещенко В., Волошин С. // Proceeding of the International Conference on Computer and Information Technologies 2009, IEEE. – Lviv, 2009. – С. 151–154.
16. Atkinson Malcom. The Object-Oriented Database System Manifesto / Malcom Atkinson, Francois Bancilhon, David DeWitt, Klaus Dittrich, David Maier, Stanley Zdonik // Carnegie Mellon School of Computer Science, 1995.
17. Atwood J. Object-Relational Mapping is the Vietnam of Computer Science. Coding Horror, Programming and Human Factors [Електронний ресурс] / Atwood, Jeff. – Режим доступу : http://www.codinghorror.com/blog/2006/06/object-relational-mapping-is-the-vietnam-of-computer-science.html. – Назва з екрана.
18. ORM [Електронний ресурс]. – Режим доступу : URL : http://ru.wikipedia.org/wiki/ORM/ – Назва з екрана.
19. Hitting the Relational Wall: An Objectivity White Paper. – Objectivity, Inc., 2005.

V. Tereshchenko, S. Voloshyn

THE APPROACH FOR IMPLEMENTING OBJECT-RELATIONAL DATABASE

This paper is devoted to the implementation of a virtual object-relational database over relational database. Particular attention is paid to the data inheritance. The approach is dynamic and allows real-time modify entities. In article analyzed the existing schemas to describe object-oriented data and proposed a generalized scheme, which combines several approaches.

Keywords: object-relational database, ORM, object-relational mapping.