НОВІ ЗАСОБИ КІБЕРНЕТИКИ, ІНФОРМАТИКИ, ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ ТА СИСТЕМНОГО АНАЛІЗУ

УДК 004.75

А.В. АНІСІМОВ
Київський національній університет імені Тараса Шевченка, Київ, Україна,
e-mail: *avatatan@gmail.com*.

І.О. ЗАВАДСЬКИЙ
Київський національній університет імені Тараса Шевченка, Київ, Україна,
e-mail: *ihorza@gmail.com*.

П.П. КУЛЯБКО
Київський національній університет імені Тараса Шевченка, Київ, Україна,
e-mail: *kpp1@ukr.net*.

РОЗШИРЕННЯ РЕЛЯЦІЙНОЇ АЛГЕБРИ З УРАХУВАННЯМ ПРОПОЗИЦІЙ DBTG CODASYL

Анотація. Досліджено розв'язання проблеми низької обчислювальної ефективності реляційної моделі даних. Запропоновано розширення реляційної алгебри за рахунок операцій над наборами даних — основної конструкції у пропозиціях DBTG CODASYL. Користувачу надається можливість вибирати спосіб реалізації зв'язків між даними залежно від вимог щодо швидкості їхнього оброблення: повільних але гнучких, що базуються на характерній для реляційних СКБД символічній адресації, чи швидких, але жорстких на прямих указівниках (відносна адресація), характерних для СКБД «дореляційних часів».

Ключові слова: реляційний підхід, реляційна алгебра, пропозиції DBTG CODASYL, набір даних, алгебра вибору Дрібаса.

ВСТУП

Як відомо, для реляційного підходу [1–3] передбачається максимальне спрощення структури даних до плоского файлу та значне підвищення рівня мови маніпулювання даними, що дає можливість створити для неї чітке математичне підґрунтя. Водночас перехід від більш складних структур до реляцій призводить (принаймні для простих запитів) до зниження ефективності оброблення, хоча б через те, що в реляційних базах застосовується переважно символічна адресація, а в БД зі складними структурами — відносна [4]. З розвитком технологій великих даних (Big Data) і хмарних обчислень (Cloud Computing) проблема недостатньої ефективності оброблення даних набуває особливого значення. Її розв'язання, принаймні часткове, може полягати у використанні різного типу зв'язків між даними відповідно до швидкості їхнього оброблення: від повільних, але гнучких, що характерно для реляційних СКБД, до швидких, але жорстких на прямих указівниках (або посиланнях), що характерно для СКБД «дореляційних часів».

У другій половині 60-х років у сфері оброблення даних було запропоновано перейти від роботи зі структурованими файлами до СКБД зі складними структурами даних, але у обох випадках одна операція читання в таких СКБД повертала в прикладну програму не більше одного запису. Модель DBTG CODASYL [4, 5]

має в своєму складі мову опису даних (DDL) та мову маніпулювання даними (DML). Мова DDL базується на записах у стилі мови КОБОЛ, що зв'язуються між собою за допомогою наборів даних (Data Sets, DS), які можуть забезпечити швидку локалізацію необхідних даних або знаходження місця, куди їх слід додавати. На відміну від простих файлів, кожен з яких мав лише один поточний стан (тобто позицію, звідки буде наступне читання під час звернення до файлу), DDL CODASYL давала можливість запам'ятовувати поточний стан для кожного оголошеного DS, завдяки чому програміст мав змогу швидко отримувати необхідні дані (за допомогою DML) різними шляхами, тобто використовуючи різні DS. Суттєвим недоліком DML CODASYL для користувачів (який було успішно подолано застосуванням реляційного підходу) є її рівень. Якщо порівнювати DML CODASYL, наприклад, з SQL, то це буде приблизно те саме, що порівнювати асемблер з Java. Зазвичай високорівневі мови програмування для більшості користувачів зручніші, однак слід враховувати ще такий фактор, як обчислювальна ефективність.

Мета запропонованої роботи — побудувати алгебру, носієм якої буде множина, що містить і реляційні відношення, і набори даних (структури на зразок наведених у [4, 5]), як доповнення до класичної реляційної алгебри чи до алгебри вибору Дрібаса [2]. Сигнатура запропонованої алгебри включатиме операції, подібні до реляційних, які, однак, виконуватимуться ефективно завдяки використанню відносної адресації в структурах даних.

Стаття складається з трьох розділів. У розд. 1 описано мову визначення даних, тобто правила, за якими описується основна множина узагальненої алгебри, у розд. 2 — мову маніпулювання даними, що визначає сигнатуру алгебри, а в розд. 3 наведено кілька прикладів застосування цієї мови, що обґрунтовують її ефективність.

1. МОВА ВИЗНАЧЕННЯ ДАНИХ

Набором даних (DS) називається множина пар записів вигляду (o, m), де o — запис-власник (OWNER), а m — запис-член (MEMBER). Усі записи-власники мають бути однотипними, так само як і всі записи-члени мають належати до одного певного типу (можливо, іншого, ніж записи-власники). Терміни «запис» і «однотипний» тлумачимо в реляційному розумінні. Під записом розуміємо набір пар {ім'я атрибута: значення}. Два записи вважаємо однотипними, якщо між їхніми атрибутами існує бієкція, за якої збігаються й імена атрибутів, і типи значень. Таким чином, усі записи структурно схожі на кортежі реляцій, що є спрощенням порівняно з пропозицією DBTG CODASYL [4, 5].

У загальному випадку кожному запису-власнику може відповідати будь-яка кількість записів-членів, і навпаки. Цим наведене означення набору даних відрізняється від означення DBTG CODASYL, де вимагалося, щоб кожному запису-члену відповідав лише один запис-власник. Однак для досягнення основної мети (забезпечення швидкого виконання запитів) будемо послуговуватися базовою структурою [4] зберігання даних — дворівневим деревом. Так само як і в [4], набір даних зберігатиметься у вигляді множини дворівневих дерев, коренем кожного з яких буде певний запис-власник, а листками — підпорядковані йому записи-члени. Відмінність від пропозиції DBTG CODASYL полягає в тому, що у запропонованій структурі на один листок можуть посилатися корені кількох дерев.

Таким чином, набір даних реалізує зв'язок «багато–до–багатьох» [6] із забезпеченням швидкого доступу лише в одному напрямку: від записів-власників до підлеглих записів-членів. Зазвичай під час проєктування бази даних виникає потреба реалізовувати й зв'язки інших типів: «один–до–багатьох» зі швидким доступом від екземплярів сутності з боку «один» до екземплярів сутності з боку

«багато», «багато–до–багатьох» зі швидким доступом в обох напрямках, «один–до–одного», слабкі сутності [3] тощо. Наведемо BNF-нотацію визначення набору даних, яка дає змогу імплементувати найрізноманітніші зв'язки, де {…} означає повторення 0 або більше разів, а […] — повторення 0 або 1 раз:

<Визначення набору даних> ::= SET <Ім'я набору даних> IS
OWNER [SINGLE] <Ім'я сутності>
MEMBER [SINGLE] <Ім'я сутності> [UNIQUE (<Список атрибутів>)]
[ATTRIBUTE <Ім'я атрибута> <Тип атрибута> {, <Ім'я атрибута> <Тип атрибута>}]
[REVERSE <Ім'я оберненого набору даних>];

Набір даних призначено для моделювання зв'язків між сутностями (реляціями або іншими наборами даних) з підтримкою механізмів швидкого доступу від об'єктів сутності, позначеної OWNER, до об'єктів підпорядкованої сутності, позначеної MEMBER. Специфікатор SINGLE для певної сутності означає, що не більше одного об'єкта цієї сутності може посилатися на кожен об'єкт пов'язаної сутності. Для створення слабких сутностей призначено специфікатор UNIQUE. Кортеж значень атрибутів, указаних після цього специфікатора, не може повторюватися в множині об'єктів, підпорядкованих певному об'єкту-власнику. Набір даних може мати власні атрибути, перелік означень яких записують після ключового слова ATTRIBUTE.

За допомогою ключового слова REVERSE може бути оголошено симетричний набір даних, у якому підтримуватимуться механізми швидкого доступу від об'єктів сутності, вказаної в початковому наборі як MEMBER, до об'єктів сутності, вказаної як OWNER. Узгодженість даних у симетричних наборах має забезпечувати СКБД. У разі моделювання зв'язків «один–до–одного» і «один–до–багатьох» такий підхід має переваги порівняно з класичними мережевими СКБД, де симетричні набори є незалежними і їхнє узгодження має забезпечуватися користувачем або розробником БД.

Розглянемо приклад предметної області. Спочатку наведемо її текстовий опис та ER-модель [6] (рис. 1):

— лікарня (**Hosp**ital), у лікарні є відділення (**Dep**artment), у відділеннях — палати (**Ch**amber), у палатах перебувають пацієнти (**Pat**ient);

— у відділеннях працюють лікарі (**Ph**ysician);

— кожен лікар відповідає за кілька палат (можливо, що якийсь лікар не відповідає за жодну палату);

— кожним відділенням завідує один із лікарів, що працюють у ньому;

— лікарі консультують пацієнтів, причому один лікар може консультувати декількох пацієнтів, а один пацієнт може консультуватися у кількох лікарів.

Зазначимо, що виокремлені жирним шрифтом частини імен об'єктів будуть використовуватись як імена реляцій чи записів для скорочення тексту. Атрибути первинних ключів на ER-моделі позначено *. Як бачимо, **Dep**artment і **Ch**amber є слабкими сутностями зі складеними ключами, тобто номери відділень можуть повторюватися, але не в межах однієї лікарні, так само як можуть повторюватися номери палат, але не в межах одного відділення.

Наведемо опис цієї предметної області згідно з реляційним підходом:

Hospital(h#*, Hname, Hadress, rank); **Dep**artment(d#*, h#*, Dname, Dmng);

Chamber(c#*, d#*, h#*, Cname, qnt);

Physician(p#*, d#, h#, Pname, spec, age, IsHeadOfDep);

Patient(t#*, c#, d#, h#, Tname, diagn, temp); **Con**sulting(r#*, p#, t#, data);

Тепер опишемо цю предметну область у термінах запропонованого далі підходу. У табл. 1 наведено означення відношень, що зображені на ER-моделі прямокутниками (див. рис. 1). Вони виконуються оператором Record Name, синтаксис якого достатньо очевидний.

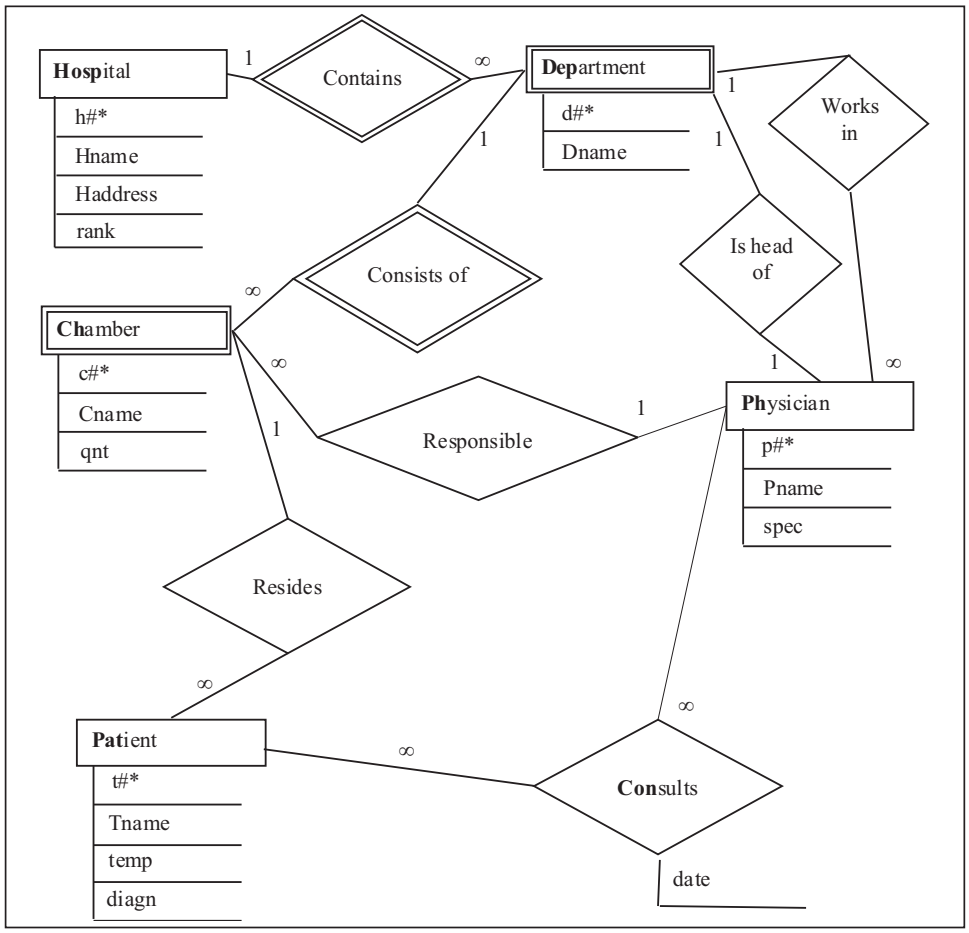

Рис. 1. ER-модель предметної області «Лікарня»

Таблиця 1. Означення реляцій у предметній області «Лікарня»

Record Name is **Hosp** { H# CHAR(5), Hname CHAR(25), Hadress CHAR(30), rank CHAR(7) };	Record Name is **Dep** { D# CHAR(5), Dname CHAR(25) };
Record Name is **Ch** { C# CHAR(5), Cname CHAR(25), Qnt integer };	Record Name is **Ph** { P# CHAR(5), Pname CHAR(25), spec CHAR(20), age integer };
Record Name is **Pat** { T# CHAR(5), Tname CHAR(25), diag CHAR(40), temp float };	

Зауважимо, що у записах не використовуються зовнішні ключі та інші атрибути, призначені для моделювання зв'язків у реляційній моделі (атрибут IsHeadOfDep у реляції **Ph**), оскільки зв'язки моделюватимуться через набори даних, означення яких наведено у табл. 2.

Таблиця 2. Моделювання зв'язків у предметній області «Лікарня»

1. У лікарні є відділення (1–∞). Set HD is OWNER SINGLE **Hosp** MEMBER **Dep** UNIQUE d# REVERSE DH;	2. У відділеннях є палати (1–∞). Set DC is OWNER SINGLE **Dep** MEMBER **Ch** UNIQUE c#;
3. У палатах є пацієнти (1–∞). Set CP is OWNER SINGLE **Ch** MEMBER **Pat**;	4. Лікар працює у відділенні (1–∞). Set DP is OWNER SINGLE **Dep** MEMBER **Ph**;

Для симетричних зв'язків (∞–∞ і 1–1) за допомогою ключового слова REVERSE вказано симетричні набори даних, між якими є зв'язок. Це означає, що механізми швидкого доступу підтримуються в обох напрямках. Крім того, якщо виникає потреба у швидкому виконанні запитів певного типу, слово REVERSE можна застосувати і для набору, що реалізує зв'язок «один–до–багатьох», як це зроблено у п. 1 табл. 2. (Створений за його допомогою набір даних DH використано в запиті 2 у розд. 3.)

На відміну від DBTG CODASYL, у запропонованій моделі даних надано можливість одному наборові бути власником або членом іншого набору, що зручно для ієрархічних побудов. Наприклад, замість перших двох наборів даних із табл. 2 можна визначити «багаторівневі» набори, наведені в табл. 3. Зауважимо, що як член або власник нового набору даних можна використовувати лише раніше описаний набір.

Таблиця 3. Набори даних як члени інших наборів даних

Семантика способів подання перших двох зв'язків у табл. 2 і 3 еквівалентна. Однак під час моделювання предметної області їх не можна використовувати разом, оскільки створення наборів зі специфікатором SINGLE визначає не просто зв'язок між елементами даних, але й обмеження цілісності (кожному екземпляру однієї сутності може відповідати не більше одного екземпляра іншої сутності). Дублювання подібних наборів може призвести до порушення такого обмеження, навіть якщо в кожному наборі даних окремо воно виконується.

Зауважимо також, що використання одних наборів даних як власників або членів інших наборів дає можливість моделювати зв'язки арності більше ніж 2. Так, наприклад, оголошення набору даних без специфікаторів SINGLE як члена іншого набору даних без цього специфікатора фактично означає створення тер-

нарного зв'язку «багато–до–багатьох–до–багатьох», однак зі швидким доступом лише в одному з трьох можливих напрямків. Реалізація всіх типів зв'язків великої арності та передбачення всіх можливих шляхів швидкого доступу потребує певної модифікації синтаксису і в цій статті не розглядатиметься.

2. МОВА МАНІПУЛЮВАННЯ ДАНИМИ

Аргументами багатьох операцій вибірки даних можуть бути не довільні, а лише однотипні набори. Наведемо означення цього поняття.

Два набори даних є однотипними, якщо однотипні і їхні записи-власники, і всі записи-члени з обох наборів. Множина однотипних наборів даних формує тип набору даних. Деякі набори даних мають лише один запис OWNER і порожню множину записів MEMBER. Можуть бути також записи поза будь-яким набором даних, тоді вважаємо, що такі записи є MEMBER деякого DS, власником якого є спеціальний запис SYSTEM. З однотипних записів із власником SYSTEM можуть формуватися класичні реляційні відношення, над якими означені всі операції як реляційної алгебри Е. Кодда, так і реляційної алгебри В. Дрібаса. Тож запропонована модель даних може розглядатися як розширення реляційної моделі, а мова вибору даних — як розширення реляційних алгебр Кодда чи Дрібаса. І хоча щодо аргументів операцій вживаємо термін «набір даних», ними можуть бути також і реляційні відношення.

Наведемо операції вибірки даних, аргументами і результатами яких є набори даних.

Теоретико-множинні операції: об'єднання $\cup$, перетин $\cap$ і різниця $\setminus$. Як і у випадку реляційних алгебр, ці операції є частково визначеними, тобто лише для однотипних DS. Вони виконуються над множинами пар (o, m), з яких складаються DS.

Декартів добуток: DS1 $\otimes$ DS2 -> DS3. Ця операція є всюди визначеною і вимоги однотипності наборів немає. Отже, DS3.OWNER = DS1.OWNER $\otimes$ $\otimes$ DS2.OWNER, записи-члени теж перемножуються між собою, але лише в межах підпорядкованості своїм OWNER. Інакше кажучи, DS3 складається з усіх таких пар $((o_1, o_2), (m_1, m_2))$, що $(o_1, m_1) \in$ DS1, а $(o_2, m_2) \in$DS2.

Проєкція: PROJECT (DS1, AList). У результаті цієї операції отримуємо набір даних, у якому зберігаються підпорядкованості DS1, але набори атрибутів і власника, і члена обмежуються лише зазначеними в списку AList полями. Якщо у власника та члена є однойменні поля, то пріоритет мають поля власника, а для зазначення поля члена використовують специфікацію MEMBER.<ім'я поля>. Якщо членом або власником набору даних є інший набір даних, то його поля теж можна вказувати в списку. У разі збігу імен пріоритет матимуть імена з більш зовнішнього набору, а для доступу до менш пріоритетних ідентифікаторів можна використовувати послідовності специфікаторів MEMBER і OWNER, розділених крапками. Символ * позначає список посилань на всі поля. Ці правила доступу до імен атрибутів діють і для всіх розглянутих далі операцій.

Якщо в результаті проєкції буде отримано кілька однакових записів власників або членів, то дублікати записів вилучають.

Може скластися враження, що операція PROJECT не є алгебричною, оскільки як аргумент використовується не лише DS, а й список, але насправді це лише зручніший запис порівняно з формально алгебраїчним PROJECT (DS1, DS2) -> DS3.

Несуттєво, які дані містяться в DS2, оскільки цей набір даних потрібен лише для формування списку полів, за якими виконується проєкція. Далі теж будуть мати місце відступи від класичного алгебричного стилю для зручності. Такий прийом використовував і Е. Кодд, коли вводив операції реляційної алгебри.

Бета-фільтрація: BFilter (DS1, *умова*) -> DS2. Ця операція фільтрації є узагальненням операції бета-фільтрації алгебри вибору Дрібаса [2]. Тут DS2 має такий самий тип, що і DS1, та заповнений тими даними з DS1, що задовольняють *умову*.

Остання є формулою числення висловлювань, у якій можуть фігурувати поля як з OWNER, так і з MEMBER, константи, а також деякі унарні предикати. Наприклад, EmptyMember (NotEmptyMember) повертає значення FALSE (TRUE), якщо для поточного примірника OWNER існує хоча б один примірник MEMBER. Виконання *умови* перевіряється для кожного елемента набору даних.

Фільтрація за множинним порівнянням (ця операція є видозміненням однойменної операції алгебри Дрібаса): SetFilter (DS1, AList, BList, θ, *умова*) -> DS2. У цій найбільш загальній операції фільтрації AList і BList — списки атрибутів DS1, що не перетинаються (вони можуть бути атрибутами як власника, так і члена); θ — бінарний предикат множинного порівняння $\subseteq$, $\subset$, $\supseteq$, $\supset$, $=$, $\neq$; *умова* — логічний вираз над атрибутами DS1. У результаті отримаємо набір даних DS2, однотипний з PROJECT(DS1, AList).

Порядок виконання операції такий:

— з DS1 вибираються елементи, що задовольняють *умову*, які потім проєктуються на BList, у такий спосіб отримуємо допоміжний набір даних DS3;

— множина елементів DS1 групується за значеннями атрибутів списку AList;

— з груп $\{DS1_{AList}\}$ вибираються ті примірники, для яких виконується умова {BList} θ DS3 (тут {BList} — множина кортежів, що складаються з атрибутів BList і відповідають групі).

Зауважимо, що в загальному випадку ця операція є суто реляційною і зв'язки між записами-власниками та записами-членами DS1 неможливо використати для підвищення ефективності власне множинного порівняння. Однак якщо AList = DS1.OWNER, BList = DS1.MEMBER, то множинне порівняння можна істотно пришвидчити завдяки структурованості набору даних. Цей підхід реалізовано в наступних двох операціях, які є окремими випадками попередньої.

Фільтрація за належністю: OnlyFilter (DS1, *умова*) -> DS2. Тут DS2 має той самий тип, що і DS1.OWNER, і заповнений записами з DS1.OWNER, які мають зв'язки лише з тими записами-членами, що відповідають *умові*. Тобто операція еквівалентна SetFilter (DS1, DS1.OWNER, DS1.MEMBER, $\subseteq$, *умова*). Ефективний алгоритм її виконання може бути таким: спочатку позначають записи-члени, які не задовольняють *умові*, а потім для кожного запису-власника перевіряють його підпорядковані записи, доки не буде знайдено серед них позначеного. Якщо серед підпорядкованих записів позначеного не знайдено, то відповідний запис-власник додають до результату.

Фільтрація за узагальненням: EveryFilter (DS1, *умова*) -> DS2. Тут DS2 має той самий тип, що і DS1.OWNER, і заповнений записами з DS1.OWNER, які мають зв'язки з усіма тими записами-членами, що відповідають *умові.* Операція еквівалентна SetFilter (DS1, DS1.OWNER, DS1.MEMBER, $\supseteq$, *умова*). Для ефективного виконання цієї операції можна спочатку застосувати до DS1 бета-фільтр за заданою умовою, а потім вибрати в отриманому наборі тих власників, у яких кількість підпорядкованих членів дорівнює кількості членів взагалі.

З'єднання: Join (DS1, DS2, *умова*) -> DS3. Множина записів-власників DS3 збігається з множиною записів-власників DS1, а множина записів-членів DS3 — із множиною записів-членів DS2. Зв'язки в DS3 встановлюються між тими власниками й членами, які задовольняють *умову*, що є формулою над атрибутами DS1 і DS2.

Природне з'єднання: Join * (DS1, DS2) -> DS3. У результаті цієї операції для таких пар елементів DS1 і DS2, що DS1.MEMBER = DS2.OWNER, буде створено екземпляр набору DS3, причому DS3.OWNER = DS1.OWNER, а DS3.MEMBER = = DS2, тобто членом набору даних DS3 є набір даних DS2, а власником — власник набору DS1 і при цьому підпорядкованості з набору DS1 трансформуються у підпорядкованості між власником набору DS1 і елементами набору DS2.

Наведемо приклади операцій додавання даних (повний опис цих операцій, а також операцій оновлення і вилучення виходить за межі завдань цієї статті):

— AddElement (DS, REC1, REC2) додає до DS елемент із записом-власником REC1 і записом-членом REC2;

— AttachMember (DS, REC, [*умова*]) додає запис-член REC і підпорядковує його всім власникам у DS, які задовольняють *умову*. На місці REC може бути відношення, сумісне з DS.MEMBER, тоді до всіх власників, що задовольняють *умову*, додаються всі кортежі цього відношення;

— Compose (REL1, REL2, *умова*) -> DS1. Тут REL1 та REL2 — реляційні відношення, з яких формується DS1, причому REL1 стає власником набору даних, а REL2 — його членом. Утворюються всі можливі підпорядкованості між REL1 і REL2, що задовольняють *умову*;

— Compose1 (REL1, [OList], [MList]) -> DS1. Операція створює набір даних DS1 на основі реляційного відношення REL1. Останнє групується за атрибутами зі списку OList, які стають атрибутами власника, у той час як атрибути зі списку MList стають атрибутами члена. Для кожного кортежу (OList) створюється запис-власник, якому підпорядковуються всі записи-члени, що належать до відповідної групи. Квадратні дужки в цьому випадку — це саме символи квадратних дужок;

— REVERSE (DS1) -> DS2. Операція створює набір даних DS2, обернений до DS1. Записи-власники DS1 стають записами-членами DS2, а записи-члени DS1 — записами-власниками DS2. При цьому $(o, m) \in$ DS2 тоді й тільки тоді, коли $(m, o) \in$ DS1.

Для прискореного багаторазового виконання параметризованих запитів інколи доцільно зберігати результати підзапитів у вигляді перманентних структур у БД. Для позначення цієї ситуації перед виразом узагальненої алгебри записуватимемо вказівку Save.

3. ПРИКЛАДИ ВИБІРКОВИХ ЗАПИТІВ

Розглянемо декілька прикладів запитів за предметною областю, яка була описана раніше. Кожен запит буде поданий у трьох виглядах: українською мовою, мовою SEQUEL, та засобами запропонованої алгебри.

Запит 1. Знайти назви відділень у лікарнях з Rank = 5:

Select Dname from **Dep** Where **Dep**.h# in (select **Hosp**.h# from **Hosp** where **Hosp**.Rank = 5);	BFilter(HD, Rank = 5 & NotEmptyMember) -> DS1; Project(DS1,[Dname]) -> res;

Запит 2. Знайти назви лікарень, що мають відділення «xxx»:

Select Hname from **Hosp** Where **Hosp**.h# in (Select h# from **Dep** where Dname = «xxx»);	BFilter(**DH**, Dname = «xxx») -> DS1; Project(DS1, [Hname]) -> res;

Запити 1 і 2 є симетричними. Такі запити в системах ієрархічного типу завжди швидко виконуються в напрямку підпорядкованості і довго — в протилежному напрямку, а в реляційних системах — з однаковою достатньо низькою швидкістю в обох напрямках. Однак у запропонованому варіанті можна використати «двобічний» набір **DH**, що забезпечить швидке виконання запиту в обох напрямках.

Запит 3. Знайти прізвища пацієнтів з температурою вищою 37° у лікарні № 1, відділенні № 3, палаті № 6:

Select t.Tname from **Pat** t Where t.temp > 37 AND t.c# = 6 AND t.d# = 3 AND t.h# = 1	BFilter(HDCP, temp > 37 & h# = 1 & d# = 3 & c# = 6) -> DS1 Project(DS1, Tname) -> res;

Запит 4. Знайти назви палат у лікарні № 1, де немає пацієнтів з температурою вищою 37°:

Select c. Cname from **Ch** c where c.c# NOT IN (Select p.c# from **Pat** p where p.temp > 37);	OnlyFilter(CP, not(temp > 37)) -> DS1; Project(DS1, Cname) -> res;

За допомогою операції OnlyFilter спочатку знаходимо всіх пацієнтів з температурою вищою 37° (тобто всі записи-члени, для яких не виконується умова temp > 37), а потім у кожній палаті перевіряємо показники температури всіх пацієнтів, поки не знайдемо пацієнта з отриманої множини чи не перевіримо її всю. У середньому цей метод працюватиме швидше за реляційний запит, у якому для кожної палати потрібно перевіряти всю знайдену множину пацієнтів.

Запит 5. Визначити номери відділень, яких немає в лікарні «yyy»:

Select d# From **Dep** Where d# not in (Select d# From **Dep** inner join **Hosp** on **Dep**.h# = **Hosp**.h# Where Hname = «yyy»)	Compose (**Hosp**, **Dep**, TRUE) -> DS1; Save HD\DS1 -> DS2; BFilter(DS2,Hname = «yyy») -> DS3; Project (DS3, d#) -> res;

У DS1 буде встановлено зв'язки між усіма лікарнями та відділеннями, у DS2 — між відділеннями та тими лікарнями, яким ці відділення не належать. Завдяки цьому допоміжному набору даних запит виконуватиметься дуже швидко: достатньо лише знайти лікарню «yyy» та всі відділення, підпорядковані їй у DS2.

Запит 6. Знайти прізвища лікарів, які консультують принаймні всіх пацієнтів з палати «zzz» відділення «xxx» лікарні «yyy»:

Select Pname from **Ph** Where (Select t# from **Con** Where **Con**.p# = **Ph**.p#) ⊇ (Select t# from ((**Pat** inner join **Ch** ON **Pat.**c# = **Ch**.c#) inner join **Dep** ON **Ch**.d# = **Dep**.d#) inner join **Hosp** ON **Hosp**.h# = **Dep**.h# where **Ch**. Cname = «zzz» AND **Dep**.Dname = «xxx» AND **Hosp**.Hname = «yyy»);	1. Join(PhPat,HDCP,PhPat.t# = HDCP.t#) -> DS1 EveryFilter (DS1, Hname = «yyy» & Dname = «xxx» & Cname= «zzz») -> res;
	2. Reverse (CP) -> PC; Reverse (DC) -> CD; Save (Join*(PhPat,Join*(PC,(Join*(CD,DH)))) -> DS1; EveryFilter(DS1, Hname = «yyy» & Dname = «xxx» & Cname = «zzz») -> res;

У правому стовпці таблиці із запитом 6 наведено два способи реалізації запиту. Оскільки обернених наборів даних PC (палати підпорядковуються пацієнтам) і CD (відділення підпорядковуються палатам) на етапі проєктування БД не було створено, то в способі 1 обернений набір даних DS1 конструюється на основі реляційного підходу, через рівність значень атрибутів PhPat.t# і HDCP.t#. Однак «реляційного» зв'язування можна уникнути, що продемонстровано в способі 2.

Запит 7. Знайти прізвища пацієнтів, яких консультують лише ті лікарі, які консультують пацієнта «ppp»:

Select Tname From **Pat** Where t# Not in (Select t# From **Con** Where p# Not in (Select p# From **Con**,**Pat** Where **Con**.t# = **Pat**.t# and **Pat**.Tname = «ppp»))	Save (Join*(PatPh, PhPat)) -> DS1; OnlyFilter(DS1,MEMBER.Tname = «ppp») -> res;

Тут оператор Save Join* конструює набір даних з однойменними атрибутами Tname у власника та члена. Тому для доступу до атрибута члена необхідно використати специфікатор MEMBER. Власне оператор OnlyFilter, принцип дії якого описано в коментарі до запиту 4, виконується значно швидше за відповідні реляційні конструкції.

Запит 8. Знайти такі спеціальності, за якими всі лікарі консультують пацієнтів з діагнозом «ddd»:

Select spec From **Ph** Group by spec Having Set(p#) ⊆ (Select p# From **Con** Where t# in (Select t# From **Pat** Where diag = «ddd»))	1. SetFilter(PhPat,spec,p#,⊆,diag = «ddd») -> res;
	2. Save Project(PhPat,spec,diag) -> DS1; OnlyFilter(DS1,diag = «ddd») -> res;

Цей запит мовою запропонованої алгебри також реалізовано у двох варіантах. Перший потребує групування і виконуватиметься так само довго, як і реляційний запит (за належної оптимізації останнього), однак має значно компактніший запис. У другому варіанті операція OnlyFilter виконуватиметься значно швидше за описаним раніше алгоритмом, однак необхідно попередньо зберегти допоміжний набір даних.

ВИСНОВКИ

Запропоноване розширення до класичної реляційної алгебри дає можливість підвищити ефективність виконання запитів і скорочує розрив між DML CODASYL та SQL.

З наведених прикладів видно, що більшість запитів у запропонованій алгебрі виконується швидше і виглядає компактніше.

СПИСОК ЛІТЕРАТУРИ

1. Codd E.F. A relational model of data for large shared data banks. *Communications of the ACM*. 1970. Vol. 13, N 6. P. 377–387. https://dl.acm.org/doi/10.1145/362384.362685.
2. Дрибас В.П. Реляционные модели баз данных. Минск: Изд-во БГУ, 1982. 192 с.
3. Дейт К.Дж. Введение в системы баз данных. 8-е изд. Москва: Издательский дом «Вильямс», 2005. 1328 с.
4. Olle T.W. The CODASYL approach to data base management. NY: John Wiley&Sons, 1978. 281 p.
5. CODASYL Database Task Group Report. NY: ACM, 1971. 487 p.
6. Chen P.P. The Entity-Relationship model — Toward a unified view of data. *ACM Trans. Database Systems.* 1976. Vol. 1, N 1. P. 9–36.

A.V. Anisimov, I.O. Zavadskiy, P.P. Kuliabko

EXTENSION OF THE RELATIONAL ALGEBRA ON THE BASIS OF DBTG CODASYL PROPOSALS

Abstract. The problem of low computational efficiency of the relational algebra is investigated. A certain extension of the relational algebra with the help of operations over data sets (the basic construction of DBTG CODASYL propositions) is proposed. The user is given a choice to implement the links between data, depending on the requirements for their processing speed, a slow but flexible option based on symbolic addressing (which is typical for relational databases) or fast but hard on direct pointers (relative addressing), which is typical for DBMS of the pre-relational era.

Keywords: relational approach, relational algebra, DBTG CODASYL proposals, data sets, Dribas selective algebra.

Надійшла до редакції 31.08.2021