УДК 683.1

Григорович А.Г., Григорович В.Г.*
Дрогобицький педагогічний ліцей
*Дрогобицький державний педагогічний університет імені Івана Франка,
кафедра інформаційних систем і технологій

НЕНОРМАЛІЗОВАНІ ВІДНОШЕННЯ: ІСТОРІЯ, КОНЦЕПЦІЇ ТА ТЕНДЕНЦІЇ РОЗВИТКУ.
ЧАСТИНА 3. РЕАЛІЗАЦІЯ НЕНОРМАЛІЗОВАНИХ ВІДНОШЕНЬ

© Григорович А.Г., Григорович В.Г., 2008

Проаналізовано історію, основні концепції та тенденції розвитку ненормалізованих відношень. Розглянуто модель Версо та реляційні операції над ненормалізованими відношеннями. Для розділених нормальних форм висвітлюються проблеми реструктурування ненормалізованих відношень та розглядаються моделі зберігання даних. Коротко охарактеризовані засоби реалізації опрацювання ненормалізованих відношень в промислових СУБД. Аналізується проблема ключів для ненормалізованих відношень. Запропоновано розширення моделі Сутність–Зв'язок для випадку складних та вкладених сутностей. Відзначаються застосування та перспективні напрямки досліджень ненормалізованих відношень.

In the article the authors analyze history, main conceptions and trends of nested relations research. There are examined Verso model and relational operations on nested relations. The authors focus upon the problems of restructuring of nested relations in partitioned normal forms, as well as data storage models. There are briefly characterized means of implementation of nested relations processing in industrial DBMS. The authors analyze the problem of keys of nested relations. They also propose the extension of the Entity-Relationship model for nested entities. The implementation and the perspective directions of researches of nested relation are singled out in this article.

Постановка задачі

У статті роглянуто досягнення та проаналізовано основні напрямки досліджень у галузі ненормалізованих відношень.

Аналіз досліджень і публікацій в галузі ненормалізованих відношень

Аналізуючи роботи, присвячені ненормалізованим відношенням, можна виділити такі напрямки та тематику публікацій:

- Поняття ненормалізованих відношень;
- Модель Версо;
- Реляційні операції над ненормалізованими відношеннями;
- Нормальні форми для вкладених відношень;
- Реструктурування ненормалізованих відношень;
- Моделі зберігання даних для ненормалізованих відношень;
- Семантика ключів для ненормалізованих відношень;
- Реалізація дій з ненормалізованими відношеннями в промислових СУБД;
- Застосування ненормалізованих відношень.

У цій статті розглядаються публікації за такими напрямками.

- Моделі зберігання даних для ненормалізованих відношень;
- Реалізація дій з ненормалізованими відношеннями у промислових СУБД;
- Застосування ненормалізованих відношень.

1. Частково нормалізована модель – оптимальна схема зберігання даних для ненормалізованих відношень

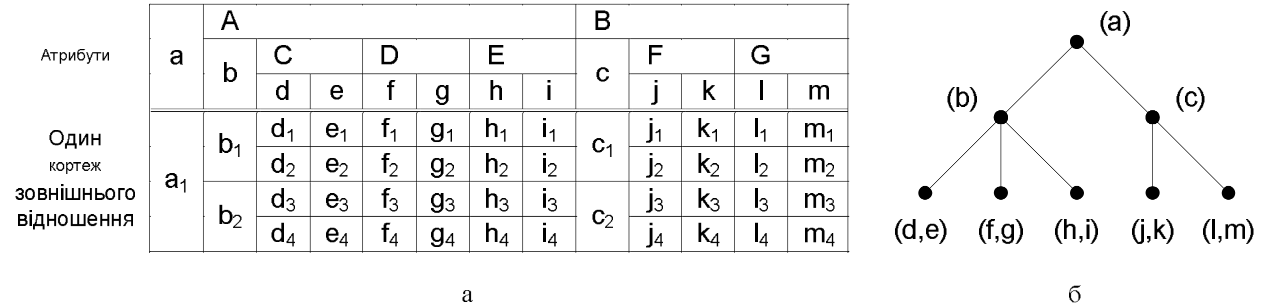
Структури зберігання даних для ненормалізованих відношень цікавили багатьох дослідників [1–7].

У роботі [8] вводиться частково нормалізована модель зберігання даних для ненормалізованих відношень. Ця модель використовує робочу інформацію системи баз даних для того, щоб отримати „кращу” (в значенні менших затрат на виконання запитів) модель зберігання даних для певного ненормалізованого відношення. Беручи за основу нормалізовану модель зберігання даних, схема ненормалізованого відношення графічно представляється у вигляді дерева, яке називається *деревом схеми* (*scheme tree*). Для представлення частково нормалізованої моделі зберігання даних, використовуючи робочу інформацію та здійснюючи серію сполучень вузлів дерева схеми, будується *приблизно оптимальне дерево схеми* (*near-optimum scheme tree*).

Доведено, що підхід [8], який використовує жадібний алгоритм, локалізує оптимальне дерево схеми в переважній більшості випадків. Показано також, що в невеликій кількості інших випадків, коли вказаний підхід локалізує „приблизно” оптимальне дерево схеми, відносна різниця між затратами на виконання запитів для побудованого дерева схеми та оптимального дерева схеми – дуже мала.

Огляд моделей зберігання даних

Як робочий приклад розглянемо ненормалізовану реляційну схему R, де R=(a, A, B), A=(b, C, D, E), B=(c, F, G), C=(d, e), D=(f, g), E=(h, i), F=(j, k) та G=(l, m). R, всеохоплююче відношення, називається *зовнішнім відношенням*. A, B, C, D, F та G – це *внутрішні відношення* (тобто відношення, які містяться в зовнішньому відношенні). *Атомарні атрибути* позначаються малими літерами латинського алфавіту. Схема відношення R та один кортеж із R зображені на рис. 1.



Атрибути	a	A							B				
		b	C		D		E		c	F		G	
			d	e	f	g	h	i		j	k	l	m
Один кортеж зовнішнього відношення	a_1	b_1	d_1	e_1	f_1	g_1	h_1	i_1	c_1	j_1	k_1	l_1	m_1
			d_2	e_2	f_2	g_2	h_2	i_2		j_2	k_2	l_2	m_2
		b_2	d_3	e_3	f_3	g_3	h_3	i_3	c_2	j_3	k_3	l_3	m_3
			d_4	e_4	f_4	g_4	h_4	i_4		j_4	k_4	l_4	m_4

Рис.1. Приклад дерева схеми для ненормалізованого відношення:
а – один кортеж із R; б – дерево схеми R

Згідно з літературними джерелами [8], моделі зберігання ненормалізованих відношень класифікуються такими п’ятьма способами [3–6, 8–10]: декомпонована, нормалізована, вирівняна, частково декомпонована, частково нормалізована. Для демонстрації кожної моделі використовуємо ненормалізоване відношення схеми R із попереднього прикладу. Припустимо, що для кожного зовнішнього чи внутрішнього відношення Φ існує *сурогатний ключовий атрибут* (*surrogate attribute*) $S_Φ$ – тобто, штучний ідентифікатор, який використовується як ідентифікатор для відношення Φ (подібно до „ідентифікатора кортежу” в 1NF-відношеннях).

Декомпонована модель зберігання (*Decomposed Storage Model, DSM*) [11], [6] використовує області пам’яті, які можуть переміщуватися (transposed storage). Кожний атомарний атрибут відношення разом із сурогатним ключем для ідентифікації запису формує бінарне відношення. Кожне бінарне відношення записується в окремому файлі.

Наприклад, існує 13 бінарних відношень, і отже, – 13 файлів, які представляють схему R в DSM-моделі:

R_a=(S_R, a), A_b=(S_A, b), B_c=(S_B, c), C_d=(S_C, d), C_e=(S_C, e), D_f=(S_D, f), D_g=(S_D, g), E_h=(S_E, h), E_i=(S_E, i), F_j=(S_F, j), F_k=(S_F, k), G_l=(S_G, l), G_m=(S_G, m).

Нормалізована модель зберігання (*Normalized Storage Model, NSM*) [4], [5] декомпонує ненормалізоване відношення так, що атомарні атрибути кожного кортежу зовнішнього чи внутрішнього відношення формують запис файла; внутрішні відношення заданого рівня зв'язуються одне з одним за допомогою сурогатних ключів. Для того, щоб отримати внутрішнє відношення, можуть використовуватися індекси з'єднання [12–13].

NSM-представлення схеми R приводить до 8 файлів. Кожний файл містить атомарні атрибути відповідного зовнішнього чи внутрішнього відношення разом із його сурогатним ключем. Записи кожного файла мають такий формат:

R=(S_R, a), A=(S_A, b), B=(S_B, c), C=(S_C, d, e), D=(S_D, f, g), E=(S_E, h, i), F=(S_F, j, k), G=(S_G, l, m).

Вирівняна модель зберігання (*Flattened Storage Model, FSM*) [3, 5] авторами названа безпосередньою моделлю зберігання (Direct Storage Model) – ненормалізоване відношення зберігається безпосередньо в одному файлі. Кожний запис файлу представляє цілий кортеж зовнішнього відношення. Записи файла можуть бути згруповані за атомарними атрибутами кортежів зовнішнього відношення. Доступ до кортежів вкладених відношень оснований на інших атрибутах, ніж атрибути кортежів зовнішнього відношення, і здійснюється за допомогою вторинних індексів або послідовних сканувань.

Весь екземпляр ненормалізованого відношення зберігається в одному файлі, запис якого для схеми R має такий формат:

(S_R, a, {S_A, b, {S_C, d, e}, {S_D, f, g}, {S_E, h, i}}, {S_B, c, {S_F, j, k}, {S_G, l, m}}).

Можна вважати, що DSM та NSM являють собою часткові випадки наступної моделі.

Частково декомпонована модель зберігання (*Partial Decomposed Storage Model, P-DSM*) [9], [10], [5] – це суміш між DSM та NSM. Атомарні атрибути внутрішнього та зовнішнього відношення розділені так, що ті атомарні атрибути, до сукупності яких часто здійснюється доступ, записуються в одному і тому самому файлі. Кожний файл містить набір атомарних атрибутів і сурогатний ключ їхнього концептуального відношення.

P-DSM модель для R може бути виражена в кількох можливих формах. Одна із цих форм приводить до 10 файлів з такими форматами записів:

R=(S_a, a), A=(S_A, b), B=(S_B, c), C=(S_C, d, e), D_1=(S_D, f), D_2=(S_D, g), E=(S_E, h, i), F_1=(S_F, j), F_2=(S_F, k), G=(S_G, l, m).

Також можна розглядати моделі NSM та FSM як окремі випадки наступної моделі. **Частково нормалізована модель зберігання** (*Partial Normalized Storage Model, P-NSM*) [8] – у цій моделі ненормалізоване відношення вертикально розділяється на частини так, що ті підоб'єкти (тобто, внутрішні відношення), до сукупності яких часто здійснюється доступ, записуються в одному і тому самому файлі. Кожний файл містить атомарні атрибути внутрішнього та зовнішнього відношення та деякі його похідні відношення.

Одна із можливих P-NSM моделей для R складається із 6 файлів таких форматів:

R=(S_R, a, {S_A, b}), B=(S_B, c, {S_F, j, k}), C=(S_C, d, e), D=(S_D, f, g), E=(S_E, h, i), G=(S_G, l, m).

На рис. 2. показано зв'язок між моделями зберігання даних.

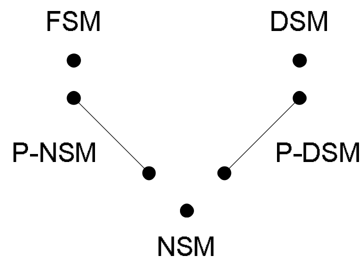


Рис. 2. Спектр різних моделей зберігання
для ненормалізованих відношень

Зміст частково нормалізованої моделі зберігання

Завдяки запису в один і той самий файл тих внутрішніх відношень, до сукупності яких часто здійснюється доступ, вдається зменшити кількість операцій вводу/виводу, необхідних для виконання запитів.

У загальному випадку, моделі зберігання для ненормалізованих відношень можуть класифікуватися на основі підтримки трьох різних типів запитів: (a) – запити, які маніпулюють цілими кортежами зовнішнього відношення, (b) – запити, які маніпулюють внутрішніми відношеннями, (c) – запити, які маніпулюють специфічними окремими компонентами кортежів відношень (зовнішніх та внутрішніх) та їхніми атомарними атрибутами на різних рівнях вкладеності. Якщо в системі переважають запити типу (a), то ясно, що модель FSM буде найкращим вибором для реалізації ненормалізованого відношення. Для запитів типу (b) очевидним вибором для реалізації ненормалізованого відношення буде модель NSM. Запити типу (c) утворюють загальний випадок для запитів до ненормалізованих відношень. Модель P-NSM забезпечує хорошу підтримку для тих систем ненормалізованих баз даних, в яких переважають запити типу (c). За допомогою використання відповідної технології та робочої інформації про систему модель NSM може бути перетворена до моделі P-NSM з кращим виконанням обчислень запитів.

Модель NSM графічно представляється у вигляді дерева схеми, кожний вузол якого являє собою файл, що містить атомарні атрибути внутрішнього чи зовнішнього відношення. У роботі [8] автори звертають особливу увагу на два параметри робочої інформації про систему баз даних. *Кількість запитів* (*query count*) – це перший параметр. Кожний вузол і кожне ребро в дереві схеми мають свою власну кількість (частоту) запитів – тобто, кількість запитів, які маніпулюють інформацією у цьому вузлі чи у вузлах, суміжних цьому ребру відповідно. Запропоновано алгоритм ASSIGN-F, який для даного набору запитів послідовно присвоює частоти вузлам та ребрам дерева схеми. Другим параметром є *вартість запиту* (затрати на виконання запиту, *query cost*). Вартість запиту для кожного вузла – це функція від розмірів файла, який представляється вузлом, та від типу використаного запиту. Кожний вузол може мати певну кількість різних вартостей запитів. Запити класифікуються відповідно до їх вартостей опрацювання дискової інформації. Кожна група запитів містить такі запити, які мають однакову вартість опрацювання дискової інформації.

Присвоєння частот дереву схеми

Введемо деякі поняття та означення:

Атомарне відношення R_A (внутрішнього чи зовнішнього) відношення R – це відношення, яке містить лише атомарні атрибути R. Зрозуміло, що для відношення S, яке перебуває у першій нормальній формі, відповідним атомарним відношенням буде саме S. Для ненормалізованого відношення схеми R=(a, b, B, C), B=(a_1, b_1), C=(a_2, b_2) атомарними відношеннями будуть R_A=(a, b), B_A=B, C_A=C відповідно.

Дерево схеми T з набором вузлів N_T ненормалізованого відношення представляє структуру цього відношення. Дерево схеми також є графічним представленням моделі NSM. Кожний вузол n_i із N_T представляє файл, який містить атомарне відношення даного (внутрішнього чи зовнішнього)

відношення. Кожне ребро в дереві схеми T представляє зв’язок між двома атомарними відношеннями, які суміжні до даного ребра.

Частоти запитів для вузлів та ребер дерева схеми залежать від відношення між вузлами, введеними до запиту.

Для запиту Q вузол n називається *вузлом запиту* (*query node*), якщо запит звертається до цього вузла. N_Q – це набір всіх вузлів запиту, які стосуються запиту Q. Ті вузли, до яких запит Q не звертається, називаються *вузлами поза запитом* (*nonquery nodes*). N_T-N_Q – це набір вузлів поза запитом відносно запиту Q.

Вилучимо всі вузли поза запитом із дерева схеми T. Набір дерев, які залишилися після цього, формує *ліс запиту* (*query forest*) F_Q відносно запиту Q. Якщо кардинальність F_Q дорівнює одиниці (тобто, $|F_Q|=1$), то результуюче дерево називається *деревом запиту* (*query tree*) T_Q.

Наприклад, нехай дерево схеми T з набором вузлів N_T – таке, як зображено на рис. 3, a. Припустимо, що для деякого запиту Q набором вузлів запиту буде $N_Q=\{n_1, n_2, n_3, n_5, n_8\}$. Тоді набір вузлів поза запитом – це $\{n_4, n_6, n_7\}$. Якщо із дерева схеми T вилучити всі вузли поза запитом, то вузли, які залишаться, сформують ліс запиту F_Q, який складається із одного дерева, зображеного на рис. 3, b. Припустимо тепер, що для іншого запиту Q′ набором вузлів запиту буде $N_{Q'}=\{n_2, n_3, n_4, n_6, n_8\}$. Тоді набір вузлів поза запитом – це $\{n_1, n_5, n_7\}$. Якщо із дерева схеми T вилучити всі вузли поза запитом, то вузли, які залишаться, сформують ліс запиту $F_{Q'}$, зображений на рис. 3, c. $F_{Q'}$ складається із двох різних дерев, тобто $|F_{Q'}|=2$.

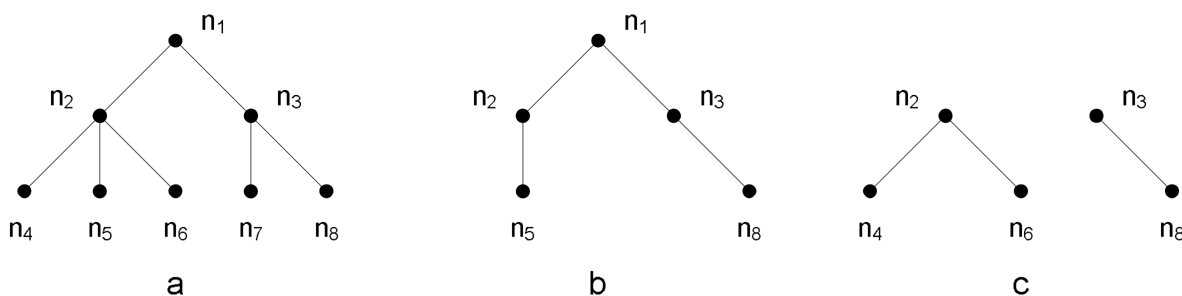


Рис. 3. Приклад дерева схеми та ліси різних запитів: a – дерево схеми T;
b – ліс запиту F_Q; c – ліс запиту $F_{Q'}$

Надалі розглядаються лише такі запити, для яких $|F_Q|=1$.

Доступ до файла F, який представляється вузлом n дерева схеми, може здійснюватися запитами, які стосуються лише атрибутів вузла n, вузла n та інших вузлів n_i, суміжних до вузла n.

Завдання полягає в тому, щоб взяти дерево схеми і перетворити його на нове дерево з кращими якостями. Перед тим, як підрахувати кількість звертань до файла, потрібно присвоїти частоти вузлам та ребрам того дерева схеми, яке підлягає модифікації.

Частота f_j вузла n_j для запиту типу q означає кількість тих запитів типу q, які стосуються лише атрибутів вузла n_j. Частота fx_i ребра e_i, інцидентного вузлам n та n_j, для запиту типу q означає кількість тих запитів типу q, які стосуються атрибутів вузлів n та n_j. Припустимо, що існує L ребер, інцидентних вузлу n із T; та що e_i, $1\le i\le L$ – ребро, інцидентне вузлу n. Кількість запитів $A_{n,q}$ до файла F відносно до запитів типу q означає кількість запитів типу q, які стосуються атрибутів вузла n. Отже, $A_{n,q}$ визначається як

$$A_{n,q} := \left(\sum_{i=1}^{L} fx_i \right) + f_n .$$

Алгоритм ASSIGN-F присвоює частоти вузлам і ребрам дерева T:

```
Алгоритм ASSIGN-F (T, S)
  Вхід:   (а) Дерево схеми T з деякими частотами вузлів та ребер
              (можливо, всі частоти дорівнюють нулю), та
          (б) послідовність S запитів, таких, що
              для кожного запиту Q із S: |F_Q|=1
  Вихід:      Нова частота, присвоєна ребрам та вузлам T.
    begin
      { fx_i означає частоту ребра e_i; f_j означає частоту вузла n_j }
      for each запиту Q in S do
      begin
        for each ребра e_i in T_Q do fx_i := fx_i + 1;
        for each вузла n_j in T_Q do f_j := f_j + 1 – d(n_j);
                                      { d(n_j) означає степінь вузла n_j }
      end;
    end.
```

Наприклад, нехай дерево схеми T_Q запиту Q – таке, як зображено на рис. 4.

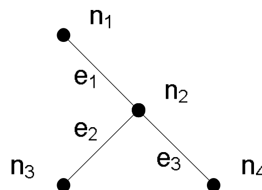


Рис. 4. Дерево T_Q

Припустимо, що до врахування запиту Q частоти мали значення: f_1=3, f_2=2, f_3=5, f_4=2 та fx_1=1, fx_2=2, fx_3=4. Згідно з алгоритмом ASSIGN-F, частота кожного ребра e_i буде збільшена на 1, і після зміни стане fx_1=2, fx_2=3, fx_3=5. Частота кожного вузла n_j буде збільшена на 1-d(n_j) (тут d(n_j) – степінь вузла n_j). Тобто, після зміни, частоти вузлів стануть такими: f_1=3, f_2=0, f_3=5, f_4=2.

Отже, кожного разу, як тільки запит Q звертається до деяких атрибутів, що зберігаються у файлі F, кількість запитів до файла збільшується на 1.

Розрахунок вартості дерева схеми

Вартість дерева схеми обчислюється за допомогою використання вартостей запитів та частот вузлів і ребер дерева схеми. Залежно від типів запитів, вартість дерева схеми в моделі NSM може бути змінена за допомогою зменшення глибини дерева схеми шляхом *операції сполучення* (операції зливання, *merge operation*) двох або більшої кількості вузлів – і це приводить до P-NSM представлення. Тоді дерево схеми з найменшою вартістю може бути знайдене шляхом перегляду всіх можливих дерев, отриманих із заданого дерева схеми за допомогою операції сполучення вузлів.

Вартість запиту $C_{n,q}$ вузла n *відносно типу запитів* q – це кількість доступів до диску, необхідних для виконання одного запиту типу q до файла F, представленого вузлом n. Кожний вузол може мати деяку кількість різних вартостей запитів відповідно до типів запитів, які

розглядаються в конкретній системі баз даних. Всі запити групуються згідно з вартістю опрацювання дискової інформації, – таким чином, що до кожної групи входять ті запити, які мають однакову вартість опрацювання дискової інформації. Вартість опрацювання дискової інформації представляється в порядковій нотації як функція розмірів відповідного файла. Вартості запитів для різних типів запитів можуть бути K_1.1, K_2.*log*N, K_3.N, …, і т.д., де N – це розмір файла, який використовується для виконання запиту, а K_i – це константи.

Розглянемо дерево схеми T з набором вузлів N_T, тип запитів q, та файл F, представлений вузлом n із T. *Вартість файла* F *відносно типу запитів* q визначається як кількість запитів до F для типу запитів q, помножена на вартість запиту для файла F відносно типу запитів q. Для набору QT запитів, які належать до різних типів, *вартість файла* F *відносно* QT визначається так:

$$C_{n,QT} = \sum_{q \in QT} (A_{n,q} \times C_{n,q})$$

де $A_{n,q}$ та $C_{n,q}$ – означають відповідно кількість запитів та вартість запитів файла F відносно q.

Розглянемо дерево схеми T з набором вузлів N_T та набір типів запитів QT. *Вартість* E_T *дерева схеми* T *відносно* QT визначається як

$$E_T = \sum_{n \in N_T} C_{n,QT} .$$

Залежно від типів запитів вартість дерева схеми змінюється із зміною глибини дерева схеми. Глибина дерева схеми зменшується у разі сполучення (злиття) в один вузол двох або більшої кількості вузлів.

Алгоритм GREEDY-MERGE отримання оптимального дерева схеми

У роботі [8] також наводиться алгоритм жадібного сполучення GREEDY-MERGE, на вході якого задаються: початкове NSM-дерево схеми; типи запитів; вартості запитів, пов'язані з кожним вузлом; частоти для кожного вузла та кожного ребра дерева схеми; і використовується жадібний алгоритм для перетворення NSM-дерева схеми у P-NSM-дерево схеми з меншою вартістю дерева схеми. Алгоритм GREEDY-MERGE працює поперек порядку рівнів, і починається з найнижчого рівня. На кожному кроці перевіряється підмножина вузлів даного рівня на злиття з їхнім батьківським вузлом. Після перевірки вузлів поточного рівня алгоритм переходить вгору до наступного вищого рівня і повторює перевірки злиттям на новому рівні. Для великої кількості типів вартостей запитів алгоритм GREEDY-MERGE знаходить дерево схеми з найменшою вартістю без необхідності перегляду всіх можливих дерев, отриманих із заданого дерева схеми.

Підхід, запропонований в [8], для знаходження приблизно оптимального дерева схеми полягає у знаходженні приблизно оптимального піддерева для кожного дворівневого піддерева в дереві схеми T. Для заданого дворівневого піддерева алгоритм починається із обчислення вартостей піддерев, отриманих після сполучення одного із „дочірніх" вузлів з „батьківським" вузлом. Очевидно, що кількість таких піддерев збігається із кількістю „дочірніх" вузлів. Потім порівнюються вартості початкового піддерева та отриманих піддерев, і те піддерево, вартість якого найменша, вибирається як нове. Після цього нове піддерево знову досліджується на злиття двох вузлів в один – злиття, яке приводить до побудови піддерева з ще меншою вартістю, – і т.д., доти, поки більше не залишиться можливостей вдосконалити (зменшити) вартість дворівневого дерева схеми.

Кількість піддерев, які перевіряються алгоритмом GREEDY-MERGE, становить $O(|N_T|^2)$:

```
Алгоритм GREEDY-MERGE (T, T′)
  Вхід:      Дерево схеми T з деякими частотами та вартостями
             вузлів та ребер із T.
  Вихід:     „Приблизно оптимальне” дерево схеми T′.
    begin
      for i from найвищого рівня T to 1 do
      begin
        for each дворівневого піддерева ST з листами на рівні i do
        begin
          repeat
            Обчислити вартість піддерева ST;
            for each листа j, 1≤j≤k in ST do
            begin
              Отримати піддерево ST_j шляхом злиття j
              з його батьківським вузлом;
              Обчислити вартість ST_j;
            end;
            Нехай ST_t – означає найменшу із вартостей ST_j, 1≤j≤k;
            if вартість(ST_t) < вартість(ST)
              then замінити ST на ST_t та помітити „можна вдосконалити”
              else помітити „не можна вдосконалити”;
          until існує мітка „не можна вдосконалити”;
        end;
      end;
    end.
```

Розглянемо роботу алгоритму GREEDY-MERGE на прикладі:
Нехай дерево схеми T – таке, як зображено на рис. 5, a.

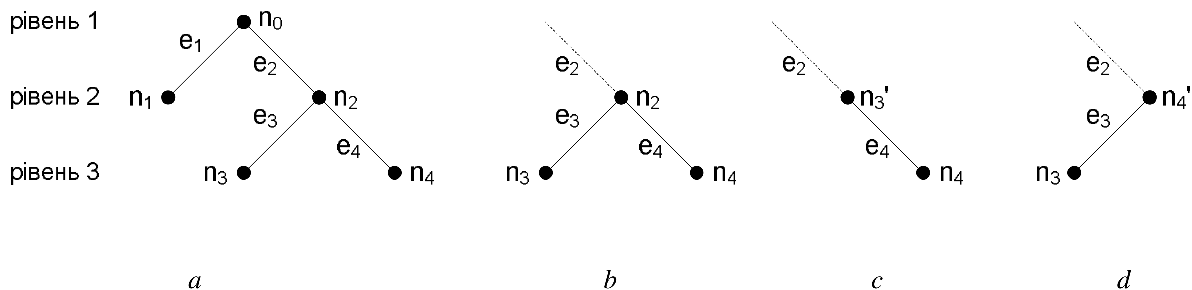


Рис. 5. Приклад роботи алгоритму GREEDY-MERGE: a – дерево схеми T; b – піддерево T_1; c – піддерево T_2; d – піддерево T_3

Для простоти припустимо, що в системі опрацьовуються лише запити одного типу, тобто, у формулах розрахунку вартостей не будемо враховувати різні типи запитів. Кожний вузол n_i, $0 \le i \le 4$ має частоту f_i та вартість запиту C_i; кожне ребро e_j, $1 \le j \le 4$ має частоту fx_i.

Алгоритм GREEDY-MERGE починає роботу з вузлів, розміщених на рівні 3. У цьому прикладі це піддерево T_1, що містить такі вузли разом з їх батьківським вузлом n_2 (рис. 5, b).

Вартість дерева схеми для піддерева T_1 становить

$$E_{T_1} = C_2\left(f_2 + \sum_{i=2}^{4} fx_i\right) + \sum_{i=3}^{4} C_i\left(f_i + fx_i\right).$$

Альтернативами цьому піддереву є піддерева від злиття вузла n_3 з вузлом n_2 (піддерево T_2, Рис. 5. c) та злиття вузла n_4 з вузлом n_2 (піддерево T_3, Рис. 5. d). Наприклад, новий вузол n_3' із T_2 має частоту $f_3'=f_2+f_3+fx_3$ та вартість запиту C_3', яка є вартістю запиту файла, отриманого шляхом злиття файлів, представлених вузлами n_2 та n_3.

Вартості дерев схеми для піддерев T_2 та T_3 будуть такими:

$$E_{T_2} = C_3'\left(f_2 + f_3 + \sum_{i=2}^{4} fx_i\right) + C_4\left(f_4 + fx_4\right);$$

$$E_{T_3} = C_4'\left(f_2 + f_4 + \sum_{i=2}^{4} fx_i\right) + C_3\left(f_3 + fx_3\right),$$

де C_4' – це вартість запиту нового вузла n_4', частота вузла n_4' становить $f_4'=f_2+f_4+fx_4$.

На основі трьох формул обчислення вартостей E_{T_1}, E_{T_2} та E_{T_3}, алгоритм GREEDY-MERGE вибирає те піддерево, яке має найменшу вартість. У випадку, коли найменшою вартістю є E_{T_1}, алгоритм GREEDY-MERGE переходить вгору до вищого рівня і повторює процедуру. З іншого боку, коли найменшою вартістю є E_{T_2} чи E_{T_3}, алгоритм GREEDY-MERGE бере піддерево, отримане на останньому кроці (тобто, T_2 чи T_3) і повторює процедуру.

Обґрунтування частково нормалізованої моделі зберігання

У роботі [8] наведено оцінку верхньої границі кількості піддерев, які перевіряються алгоритмом GREEDY-MERGE: для дерева схеми T з набором вузлів N_T, алгоритм GREEDY-MERGE опрацьовує, в найгіршому випадку, менш ніж $|N_T|(|N_T|-1)/2$ різних дворівневих піддерев дерева схеми T.

Показано також, що для тих типів запитів, які мають однакову постійну вартість запиту, оптимальне дерево схеми – це дерево, у якому всі вузли злиті в один вузол; в цьому випадку найкращою моделлю зберігання даних буде модель FSM.

Доведено, що коли всі вузли мають однакову функцію вартості запиту, алгоритм GREEDY-MERGE завжди знаходить оптимальне дерево схеми для практично всіх функцій вартості запиту, за винятком logN, де N – це розмір файла. Для випадку, коли вартість запиту $C_{n,q}=log$N, доведено, що алгоритм GREEDY-MERGE знаходить оптимальне піддерево для кожного дворівневого дерева в дереві схеми.

Експериментально досліджений ефект застосування алгоритму GREEDY-MERGE до дерева схеми T, коли лише один тип запитів введено до робочої інформації про систему баз даних. Отриманий результат показує, що алгоритм GREEDY-MERGE майже завжди локалізує „приблизно" оптимальне дворівневе піддерево.

Автори [8] наводять результати попередньо проведених експериментів, які показують, що більш ніж для 25 000 випадків у 95,2% із них алгоритм GREEDY-MERGE продукує P-NSM-структури з оптимальними деревами схеми. Для тих випадків, коли отримано "приблизно" оптимальне дерево схеми, середній і максимальний процент помилок становлять 1,255% та 5,744% відповідно.

2. Реалізація дій з ненормалізованими відношеннями в промислових СУБД

2.1. Об'єктно-реляційні особливості та робота з вкладеними таблицями в Oracle

Визначення об'єктних типів, що містять вкладені відношення

Oracle дає змогу визначати типи подібно до типів SQL. Наприклад, визначення точки, заданої двома координатами, виглядатиме так:

```
CREATE TYPE PointType AS OBJECT (
        x NUMBER,
        y NUMBER );
```

Об'єктний тип можна використовувати подібно будь-якому іншому типу в подальших визначеннях об'єктних типів або типів-таблиць. Наприклад, ми можемо визначити тип лінії:

```
CREATE TYPE LineType AS OBJECT (
        end1 PointType,
        end2 PointType  );
```

Після цього можна створити відношення, яке містить лінії типу LineType:

```
CREATE TABLE Lines (
        lineID INT,
        line   LineType );
```

Конструювання об'єктних величин

Подібно С++, Oracle забезпечує вбудовані конструктори для значень оголошеного типу, які носять ім'я типу. Тому значення типу `PointType` формується словом PointType і списком відповідних значень, які беруться в дужки. Наприклад, щоб створити в `Lines` відрізок із значенням 27 для ключа `lineID`, який проходить від початку координат до точки (3; 4), запишемо:

```
INSERT INTO Lines
        VALUES(27, LineType(
                     PointType(0.0, 0.0),
                     PointType(3.0, 4.0)  )  );
```

Тобто ми створили два значення типу `PointType`, які використовуються, щоб створити значення типу `LineType` і це значення використовується з цілим 27, щоб створити кортеж для `Lines`.

Вкладені таблиці

Надзвичайно потужною можливістю використання об'єктних типів в Oracle є те, що стовпчикові типи (*column types*), тобто типи атрибутів, можуть бути типом-таблицею (*table-type*) [14, 15]. Тобто значення атрибутів в одному кортежі може бути повним відношенням, як зображено на рис.6, де відношення з схемою (a, b) має значення b, яке, своєю чергою, є відношенням зі схемою (x, y, z).

Для того, щоб отримати відношення як тип деякого атрибуту, необхідно відразу визначити тип, використовуючи означення `AS TABLE OF`. Наприклад:

```
CREATE TYPE PolygonType AS TABLE OF PointType;
```

Наведене визначення вказує, що тип `PolygonType` – це відношення, чиї кортежі мають тип `PointType`, тобто вони мають два дійсні компоненти x та y.

Тепер можна визначати відношення, один із стовпців якого має значення, що представляють багатокутники; тобто вони – набори точок. Можливе наступне визначення, в якому багатокутники представлені назвою та набором точок:

```
CREATE TABLE Polygons (
    name   VARCHAR2(20),
    points PolygonType )
    NESTED TABLE points STORE AS PointsTable;
```

a	b		
-	x	y	z
	-	-	-
	-	-	-
	-	-	-
-	x	y	z
	-	-	-
-	x	y	z
	-	-	-
	-	-	-

Рис. 6. Схема ненормалізованого відношення R(a, b(x,y,z))

При цьому «елементарні» відношення, які представляють певні багатокутники, збережені не безпосередньо як значення атрибутів `points`, а в одинарній таблиці `PointsTable`, яка повинна бути описана після записаних в дужках списку атрибутів таблиці `Polygons` і яка використовується для збереження відношень типу `PolygonType`.

Для вставки рядків у відношення, що мають один або декілька стовпців, які є вкладеними відношеннями (наприклад, у відношення `Polygons`), використовують конструктор типу для вкладеного відношення (у нашому прикладі `PolygonType`), щоб охопити значення вкладеного відношення. Значення вкладеного відношення представлене списком значень відповідного типу; в нашому прикладі тип `PointType` представлений конструктором типу того самого імені.

За допомогою наступної конструкції вставляється багатокутник “`square`”, який складається з чотирьох точок і зображає одиничний квадрат:

```
INSERT INTO Polygons VALUES(
'square', PolygonType(PointType(0.0, 0.0), PointType(0.0,
1.0),PointType(1.0, 0.0), PointType(1.0, 1.0) ) );
```

Отримати точки цього квадрата можна за допомогою запиту

```
SELECT points
  FROM Polygons
  WHERE name = 'square';
```

Також можна отримати детальну інформацію із вкладеного відношення за допомогою запиту `FROM`, використовуючи ключове слово `THE`, застосоване до підзапиту, результатом якого є відношення. Якщо вищенаведений запит повертає ціле вкладене відношення, то результатом наступного будуть ті точки квадрату, які знаходяться на головній діагоналі (тобто x = y):

```
SELECT ss.x
FROM THE(SELECT points
         FROM Polygons
         WHERE name = 'square'
        ) ss
WHERE ss.x = ss.y;
```

У цьому запиті вкладене відношення отримало додатковий параметр ss, який використовується в запитах SELECT та WHERE так, ніби це звичайне відношення.

Комбінація вкладених відношень та вказівників (References).

У випадку створення вкладених таблиць, чиї кортежі фактично є вказівниками на кортежі іншої таблиці (це зустрічається при нормалізації даних), виникає проблема, пов'язана з тим, що атрибут вкладеної таблиці не має ніякої назви. У таких випадках в СУБД Oracle використовується ім'я COLUMN_VALUE.

Змінимо відношення, що описує багатокутники, щоб мати вкладені таблиці вказівників. Спочатку ми створюємо новий тип, який є вкладеною таблицею вказівників до точок:

```
CREATE TYPE PolygonRefType AS TABLE OF REF PointType;
```

Після цього створюємо нове відношення, подібне до Polygons, але з точками, збереженими як вкладена таблиця вказівників:

```
CREATE TABLE PolygonsRef (
    name   VARCHAR2(20),
    pointsRef PolygonRefType)
    NESTED TABLE pointsRef STORE AS PointsRefTable;
```

Необхідно пам'ятати, що точки повинні бути безпосередньо збережені в деякому відношенні типу PointType. У цьому випадку при конструюванні запиту на знаходження точок, що лежать на головній діагоналі, необхідно використати COLUMN_VALUE, щоб звернутися до стовпця вкладеної таблиці. Запит набуде вигляду:

```
SELECT ss.COLUMN_VALUE.x
FROM THE(SELECT pointsRef
         FROM PolygonsRef
         WHERE name = 'square'
        ) ss
WHERE ss.COLUMN_VALUE.x = ss.COLUMN_VALUE.y;
```

2.2. Реалізація вкладених таблиць засобами SQL3 та SQL4

Для створення вкладених таблиць, в яких стовбець однієї таблиці фактично містить іншу таблицю, застосовують колекції. Колекції – це конструктори типів, які використовуються для визначення колекцій інших типів. Колекції використовують для збереження кількох значень в одному стовпці таблиці. У результаті може бути створена єдина таблиця, яка містить декілька рівнів вкладеності типу «головний/підпорядкований». Отже, колекції дають змогу гнучкіше проектувати фізичну структуру бази даних.

У стандарті SQL3 введено параметризований тип колекції ARRAY, а в стандарті SQL4 передбачено додаткове введення параметризованих типів колекцій LIST, SET та MULTISET [16]. У будь-якому випадку параметр, який називається *елементом типу*, може мати попередньо визначений тип, тип користувача UDT (User-Defined Types), стрічковий тип або бути колекцією, але не може бути вказівниковим типом, або типом UDT, який містить вказівниковий тип. Крім того, кожна колекція повинна бути однорідною, тобто всі її елементи повинні мати однаковий тип, або бути породженими з ієрархії одного типу. Типи колекцій мають такі визначення:

ARRAY (масив). Одномірний масив, для якого вказана максимальна кількість елементів;

LIST (список). Впорядкована колекція, в якій допускається наявність дублікатів;

SET (множина). Невпорядкована колекція, в якій не допускається наявність дублікатів;

MULTISET (мультимножина). Невпорядкована колекція, в якій допускається наявність дублікатів.

Ці типи аналогічні типам, визначеним в стандарті ODMG 3.0 [16, с.1012-1036], за винятком того, що ім'я Bag замінено типом даних MULTISET мови SQL. Для цих колекцій передбачено такий набір операції (приклади, які їх демонструють, наведено нижче):

Дві множини SET використовуються як операнди і повертають результат типу SET;

Дві мультимножини MULTISET використовуються як операнди і повертають результат типу MULTISET;

Мультимножина MULTISET використовується як операнд і повертає результат типу SET;

Будь-яка колекція використовується як операнд і повертає результат, який дорівнює кількості елементів в цій колекції;

Два списки LIST використовуються як операнди і повертають результат типу LIST;

Два списки LIST використовуються як операнди і повертають результат типу INTEGER;

Список LIST та одне або два цілі числа використовуються як операнди і повертають результат типу LIST.

Масив ARRAY та одне ціле число використовуються як операнди та повертають елемент масиву ARRAY;

Два масиви ARRAY використовуються як операнди і виконується конкатенація цих двох масивів в єдиний масив ARRAY у вказаному порядку.

Приклади 1–3 ілюструють можливості роботи з колекціями в SQL.

Приклад 1. **Використання колекції типу SET**

Необхідно доповнити таблицю Staff, щоб в ній можна було зберігати дані про кількість членів сім'ї (next of kin), а потім найти імена та прізвища членів сім'ї особи з іменем John White.

Колекція типу даних SET дає змогу ввести до таблиці Staff стовпець так:

```
NextOfKin SET (PersonType)
```

У такому випадку запит набуде вигляду:

```
SELECT n.fName, n.lName
FROM Staff s, TABLE (s.NextOfKin) n
WHERE s.lName = 'White' AND s.fName = 'John';
```

Приклад 2. **Використання функції COUNT при роботі з колекцією типу SET**

Необхідно визначити кількість членів сім'ї кожного співробітника. Оскільки NextOfKin є полем, що має тип множини, то для визначення шуканого значення можна використати агрегуючу функцію COUNT.

```
SELECT staffNO, lName, COUNT (nextOfKin)
FROM Staff;
```

Приклад 3. **Використання колекції типу ARRAY**

Якщо передбачене обмеження, згідно з яким допускається зберігання відомостей не більше ніж про трьох членів сім'ї, то цей стовпчик можна реалізувати у вигляді даних типу ARRAY.

```
NextOfKin PersonType ARRAY(3)
```

У такому випадку запит, що передбачає вибірку даних тільки про першого члена сім'ї, виглядатиме так:

```
SELECT s.NextOfKin[1].fName, s.nextOfKin[1].lName
FROM Staff s
WHERE s.lName = 'White' AND s.fName = 'John';
```

Оскільки масив не можна використовувати як вказівку на таблицю, то в списку вибірки оператора SELECT необхідно використовувати повну форму запису.

2.3. Вкладені зв'язки в XML

XML – скорочене позначення стандартної розширюваної мови розмітки (eXtensible Markup Language). У мові XML дані подаються у вигляді рядків тексту, який передбачає введення *розмітки*, призначеної для опису властивостей даних. Застосування розмітки дає змогу доповнювати текст інформацією, що стосується його змісту або форми.

У XML можна використовувати два підходи для представлення реляційних даних:

Відношення типу «один-до-багатьох» представленні як окремі таблиці рядків, пов'язані спільними стовпчиками. Ці спільні стовпчики визначаються в колекціях як ключі, і між ними встановлюється відношення.

Вкладені відношення, представлені ієрархічно, тобто батьківські елементи містять вкладені дочірні елементи. У наступному прикладі показані замовлення `Orders` покупця, вкладеного в елемент `Customers` [17]. Тобто, дані про покупців та всі їхні замовлення групуються в окрему ієрархію з `Customers`.

```
<CustomerOrders>
  <Customers>
    <CustomerID>ALFKI</CustomerID>
    <Orders>
      <OrderID>10643</OrderID>
        <OrderDetails>
      <CustomerID>ALFKI</CustomerID>
      <OrderDate>1997-08-25</OrderDate>
    </Orders>
    <Orders>
      <OrderID>10692</OrderID>
      <CustomerID>ALFKI</CustomerID>
      <OrderDate>1997-10-03</OrderDate>
    </Orders>
    <CompanyName>Alfreds Futterkiste</CompanyName>
  </Customers>
  <Customers>
    <CustomerID>ANATR</CustomerID>
    <Orders>
      <OrderID>10308</OrderID>
      <CustomerID>ANATR</CustomerID>
      <OrderDate>1996-09-18</OrderDate>
    </Orders>
    <CompanyName>Ana Trujillo Emparedados y helados</CompanyName>
  </Customers>
</CustomerOrders>
```

2.3. Особливості СУБД UniVerse фірми IBM

Фірма IBM пропонує дві постреляційні СУБД: UniVerse та UniData. Фактично, обидві СУБД використовують одні і ті самі технології та подібні за своїми властивостями і інструментами. Ці СУБД в своїй основі використовують модель даних та набір процесорів, запропонованих фірмою Pick Systems.

Ядро СУБД UniVerse оптимізовано за продуктивністю для підтримки двох моделей даних: реляційної (традиційний доступ через SQL) та моделі зберігання бізнес-об'єктів (доступ за допомогою внутрішніх засобів UniVerse). Доступ до даних може здійснюватися в інтерактивному

режимі з використанням процесорів з оболонки uvsh, через стандартний інтерфейс ODBC або через відповідний програмний інтерфейс (API).

Методи доступу СУБД UniVerse дають змогу реалізувати обидва підходи. Для кожного з них існує відповідний набір інструментів. Для першого – стандартний SQL, включно з ODBC-драйвером і відповідний API – UniVerse Calling Interface (UCI), для другого – засоби доступу до бази даних через процесори оболонки UniVerse, інтерфейс доступу до об'єктів, реалізація якого для Windows включає Active-X компоненти (UniVerse Objects) і інтерфейс низького рівня InterCall [18].

UniVerse підтримує багатовимірні масиви та вкладені таблиці, але динамічна нормалізація даних поширюється тільки на вкладені таблиці. Вкладені таблиці мають наперед визначені вкладені відношення та обмеження цілісності: вкладена таблиця не може існувати без запису, до якого вона відноситься і при видаленні запису головної таблиці разом з нею видаляються і всі її вкладені таблиці.

Для реалізації другої моделі вибрана запис-орієнтована система доступу до даних. З одного боку, записи є рядками в таблиці, і над ними можна виконувати відповідні операції вибірки/оновлення тощо, а з іншого боку – кожний запис може бути набором інформації, яка містить вкладені таблиці та динамічні масиви даних.

Нижче наведено типовий приклад доступу до записів UniVerse через UniVerse Objects в кодах Visual Basic:

```
... Set UVCommand = ObjSession.Command
Set InvList = ObjSession.SelectList(0)
Set InvTable = ObjSession.OpenFile("INVOICES")
UVCommand.Text = "SELECT INVOICES IF NAME LIKE JOHN..."
UVCommand.Execute
InvTable.RecordId = InvList.Next
InvTable.Read
Invoice = InvTable.Record
Invoice.Field(ACCEPTED) = DateNow()
InvTable.Record = Invoice
InvTable.Write ...
```

Доступ до запису здійснюється за первинним ключем. Вибірка списків ключів може здійснюватися через оператори SQL або спеціальними засобами UniVerse (процесор RetrieVe, B-tree індекси, послідовна вибірка).

При звертанні до запису як до об'єкта вкладені таблиці трактуються як двовимірні або тривимірні динамічні масиви. Динамічні масиви дають змогу додавати та знищувати рядки, стовпці, змінювати розмірність без перевизначення. Така гнучкість представлення даних дає змогу зберігати в записах складні бізнес-об'єкти з множинними наборами параметрів, додавати властивості об'єктів. При цьому всі зміни об'єктів після їх запису відразу відображаються на загальній картині даних, представленій у вигляді набору таблиць. Для забезпечення колективного доступу до бізнес-об'єктів застосовується механізм блокування, який забезпечує різний рівень обмеження доступу до даних.

2.4. Особливості реалізації ненормалізованих відношень в інших СУБД

System 2000 та ADABAS – дві ієрархічні системи баз даних [19], які використовуються для реалізації ненормалізованих відношень. Сегменти ієрархічних записів вставляються в один файл. Всі ієрархічні відношення виражені за допомогою іншого файла. В системі OASIS [20] екземпляр кортежу разом із його похідними розміщується в одному компактному записі змінної довжини.

Dadam та інші використовують технологію відокремлення структурної інформації від даних [3], де кожний кортеж зовнішнього відношення зберігається в одному записі і для кожного кортежу існує запис в каталозі, який називається *міні-каталогом* (*mini-directory, MD*). Цей міні-каталог використовується для опрацювання розміщення кожного кортежу.

У системі IMS [21] відношення разом з усіма його внутрішніми відношеннями зберігаються в одному файлі. Кожний запис верхнього рівня у файлі містить атомарні атрибути кортежу в зовнішньому відношенні.

Останнім часом для нетрадиційних областей застосування були розроблені деякі нові системи управління базами даних, які підтримують ненормалізовані відношення. Прикладами таких систем є EXODUS [22] та POSTGRES [4]. В системі EXODUS основною одиницею збережених даних є об'єкт зберігання (цілий кортеж). Розмір об'єкта зберігання може збільшуватися або зменшуватися без накладання додаткових обмежень при виконанні вставки або вилучення. Отже, система підтримує вставку та вилучення нових частин об'єкта зберігання в будь-яке місце всередині об'єкта. В системі POSTGRES для підтримки ненормалізованих відношень визначено новий тип даних, який називається POSTQUEL. Поле типу POSTQUEL містить послідовність команд для отримання даних із інших відношень, які представляють підоб'єкти. Всі відношення зберігаються як динамічні області даних (*кучі*, *heaps*) у межах необов'язкової колекції вторинних індексів.

3. Застосування ненормалізованих відношень

Існує цілий клас бізнес-задач (так звані обліково-аналітичні задачі), в яких звернення до даних зручно здійснювати за допомогою так званих «бізнес-об'єктів» – аналогів реальних документів. У фінансовій сфері прикладами бізнес-об'єктів можуть бути рахунки-фактури, меморіальні ордери, накладні. З інформаційного погляду бізнес-об'єкти являють собою набори інформації, іноді досить складні, які містять елементи, що повторюються, але зв'язані в єдине ціле.

Існує два підходи при проектуванні інформаційної системи з таких бізнес-об'єктів.

За першим підходом необхідно нормалізувати інформацію, тобто розкласти дані, що містяться в бізнес-об'єктах, на таблиці, які містять неподільні «атомарні» елементи інформації. При цьому необхідно розв'язати низку проблем: встановити зв'язки між цими таблицями, при визначенні відношень «багато-до-багатьох» створити проміжні таблиці перехресних посилань, встановити обмеження цілісності тощо.

При проектуванні системи, яка містить декілька бізнес-об'єктів, отримаємо набір однорангових таблиць, які можна легко зобразити у вигляді ER-діаграми. При цьому бізнес-об'єкти «розмиваються» на фоні таблиць, оскільки таблиці не мають ієрархії, а бізнес-об'єкти ієрархічні за своєю суттю. Наприклад, рядки товарів в рахунку-фактурі не можуть існувати і не мають змісту без самого рахунку. При зверненні до бізнес-об'єкта проектувальник визначає в операторах SQL операції «JOIN» для вибірки інформації з кількох таблиць.

При використанні другого підходу операції вибірки/доступу до даних здійснюються безпосередньо над бізнес-об'єктами. Нормалізація даних відбувається динамічно, в оперативній пам'яті, тоді, коли це дійсно необхідно: наприклад, коли здійснюється групування рахунків-фактур за типом товару.

Цей підхід забезпечує максимальну ефективність при роботі з бізнес-об'єктами, але потребує від СУБД підтримки складних структур, таких як вкладені таблиці або масиви, а також можливості динамічної реалізації таких структур. Таблиці в такій системі отримують певну ієрархію: виділяються «головні» та «допоміжні» таблиці.

Сьогодні ці можливості реалізовані в багатьох СУБД, які існують на ринку і широко використовуються. Наприклад, постреляційна система управління базами даних Adabas компанії Software AG Adabas прекрасно працює в системах, які передбачають обробку великих об'ємів даних, а також велику кількість одночасно працюючих користувачів (OLTP: Online Transaction Processing). Adabas відрізняється від реляційних СУБД наступним:

- відношення можуть зберігатися як вкладені відношення/таблиці, що зумовлює зменшення використання ресурсів ЕОМ порівняно з традиційними реляційними СУБД;

o підтримуються ієрархічні поля з можливістю мати до 200 екземплярів значень такого поля всередині одного запису.

Продукція і широкий діапазон послуг фірми Software AG знаходять різноманітні сфери застосування [23]. Це промисловість, торгівля, транспорт, зв'язок, медицина, банки і страхування, поліція і військові системи, державне і муніципальне управління, громадські організації і багато інших сфер людської діяльності. На основі продуктів Software AG автоматизується робота банків Chase Manhattan, Dresdner Bank і Deutsche Bank, поліцейських служб і казначейства Великобританії, страхової компанії Prudential Assurance, апарату Білого дому США, японських компаній Japan Air Lines, Nissan і Nomura Securities, а також багатьох інших. Інформаційно-пошукова система SPHINX, також створена на основі виробів фірми, вирішує задачу оперативного доступу до даних світових інформаційних агентств в телекомпанії ZDF.

В Україні, Росії, СНД і країнах колишнього СРСР продукція Software AG має двадцятирічний досвід використання. Серед організацій, в яких були реалізовані проекти Software AG, Міністерство зв'язку України, Югтокобанк України, Адміністрація Президента Російської Федерації, РАО "Газпром", ГК "Росвооружение", Аерофлот, аеропорт "Шереметьево", ГВЦ Міністерства шляхів сполучення, Міністерство закордонних справ, Департамент морського флоту РФ, Державна Центральна наукова медична бібліотека, РНЦ "Курчатовський інститут", видавництво "Преса", Чебоксарський завод промислових тракторів, морський завод Мурманська "Севморпуть", Омський шинний завод, Волзький трубний завод, Володимирський хімічний завод, концерн OTIS і багато інших.

Висновки

Ця робота входить до циклу статей, присвячених огляду досягнень та аналізу основних напрямків досліджень в галузі ненормалізованих відношень.

У статті розглядаються моделі зберігання даних: декомпонована, нормалізована, вирівняна, частково декомпонована та частково нормалізована. Описано алгоритм ASSIGN-F присвоєння частоти вузлам і ребрам дерева схем, здійснено розрахунок вартості дерева схеми, розкрито роботу алгоритму GREEDY-MERGE отримання оптимального дерева схеми для великої кількості типів вартостей запитів без необхідності перегляду всіх можливих дерев, отриманих із заданого дерева схеми.

Розглянуто об'єктно-реляційні особливості та реалізацію дій з ненормалізованими відношеннями в стандартах SQL3, SQL4, XML та промислових СУБД Oracle, UniVerse фірми IBM та інших. Висвітлені можливості дають змогу стверджувати, що для забезпечення підтримки операцій над ненормалізованими відношеннями вже реалізовано відповідні розширення функцій систем управління базами даних. Багато існуючих СУБД містять інструментальні засоби для опрацювання ненормалізованих відношень, завдяки яким розв'язується проблема адекватного представлення даних в інформаційних системах.

Наведено приклади бізнес-задач, які потребують для адекватного представлення даних використання вкладених відношень і обґрунтовано необхідність такого використання. На прикладі постреляційної системи управління базами даних Adabas компанії Software AG проілюстровано можливості ненормалізованих відношень при реалізації таких бізнес-задач в різноманітних сферах людської діяльності. Особливо багатообіцяючим є застосування ненормалізованих відношень при дослідженні моделей просторових даних, – це дасть змогу на рівні моделі даних реалізувати масштабування: залежно від масштабу композитний об'єкт може розглядатися як „простий" або як такий, що має певну просторову структуру.

1. Abiteboul, S. and Bidoit, N. Non First Normal Form Relations to Represent Hierarchically Organized Data. In Proc. 3rd ACM SIGACT-SIGMOD, Waterloo, 1984, p. 191-198. 2. Hammer, M.,

Niamir, B. A Heuristic Approach to Attribute Partitioning. Proc ACM SIGMOD Int. Conf. on Management of Data. May 1979. 3. Dadam, P., Kuesprt, K., Andersen, F., Blanken, H., Erbe, R., Guenauer, J., Lum, V., Pistor, P., Walch, G. A DMBS Prototype to Support Extended NF^2 Relations: an Integrated View on Flat Tables and Hierarchies. Proc. ACM SIGMOD Int. Conf. on the Management of Data. May 1986. 4. Stonebraker, M., Rove, L.A. The Design of POSTRGES. Proc. CAN SIGMOD Int. Conf. on Management of Data, May 1986. 5. Valduriez, P., Khoshafian, S., Copeland, G. Implementation Techniques of Complex Objects. Int. Conf. on VLDB, Aug. 1986. 6. Khoshafian, S., Copeland, G., Jagodits, T., Boral, H., Valduriez, P. A Query Processing Strategy for the Decomposed Storage Model. 3rd Int. Conf. on Data Engineering. Feb. 1987. 7. Despande, A., Van Gucht, D. An Implementation for Nested Relational Databases. Technical Report, Computer Science Dept., Indiana University, Feb. 1988. 8. Hafez, A., Ozsoyoglu, G. The Partial Normalized Storage Model of Nested Relations. Proceedings of the 14th VLDB Conference. Los Angeles, California, 1988, p. 100-111. www.vldb.org/conf/1988/P100.PDF 9. Hoffer, J,A., Severance, D.G. The Use of Cluster Analysis in Physical Database Design. Proc. 2nd Int. Conf. on VLDB, 1975. 10. Navathe, S., Ceri, S., Wiederhold, G., Jinglie, D. Vertical Partitioning Algorithms for Database Design. ACM Trans. on Database Systems. Vol. 8, No. 4, Dec. 1984. 11. Copeland, G., Khoshafian, S. A Decomposed Storage Model. ACM SIGMOD Int. Conf. on Management of data, May 1985. 12. Valduriez, P., Boral, H. Evaluation of Recursive Queries Using Join Indices. Proc. of First Int. Conf. on Expert Systems, Apr. 1986. 13. Valduriez, P. Join Indices. ACM Trans. on Database Systems, Vol. 12, No. 2, June 1987. 14. Фейерштейн С., Прибыл Б. Oracle PL/SQL для профессионалов 3-е изд. – Спб.: Питер, 2003. С. 799 – 857. 15. Jeff Ullman. Object-Relational Features of Oracle. www.db.stanford.edu/~ullman/fcdb/oracle/or-objects.html 16. Коннолли Т., Бегг К. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. 3-е издание. : Пер. с англ. – М.:Издательский дом «Вильямс», 2003. 17. http://msdn.microsoft.com/library/rus/default.asp?url=/library/rus/vbcon/html/vbconnestedrelationshipsinxml.asp 18. http://www.arkcom.ru/CUV9.html 19. Olle, T.W. Introduction to 'Feature Analysis of Generalized Data Base Management systems', CACM, Vol. 14. No 5, May 1971. 20. Wiederhold, G. Database Design (2nd ed.) McGraw-Hill, 1983. 21. McGee, W.C. The Information Management System IMS/VS Part 1: General structure and Operation. IBM Syst. J., Vol. 16., No. 2, 1977. 22. Carey, M.J., DeWitt, D.J., Richardson, J.E., Shekita, E. Object and File Management in the EXODUS Extensible Database System. Int. Conf. on VLDB, Aug. 1986. 23. http://www.softwareag.ru/Solutions.