Київський національний університет імені Тараса Шевченка

Кваліфікаційна наукова
праця на правах рукопису

Мохаммеда Карама Джасіма Мохаммеда

УДК 004.655

ДИСЕРТАЦІЯ

Розширення табличних алгебр множинним успадкуванням.

01.05.03 – Математичне та програмне забезпечення обчислювальних машин і систем

12 – Інформаційні технології

Подається на здобуття наукового ступеня кандидата фізико-математичних наук

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____

(підпис, ініціали та прізвище здобувача)

Науковий керівник Буй Дмитро Борисович,
доктор фіз.-мат. наук, професор

Київ – 2017

АНОТАЦІЯ

Мохаммед К. Д. Розширення табличних алгебр множинним успадкуванням. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня кандидата фізико-математичних наук за спеціальністю 01.05.03 «Математичне та програмне забезпечення обчислювальних машин і систем» (12 – Інформаційні технології). – Київський національний університет імені Тараса Шевченка, Київ, 2017.

У дисертаційній роботі ставилися дві мети. Однією метою було формальне завдання алгоритму множинного успадкування таблиць і вивчення його властивостей. Іншою метою було уточнення і подальший розвиток табличній алгебри семантичного функцій мови запитів SQL.

Хоча семантиці SQL присвячені численні теоретичні роботи, в них не приділяється достатньо уваги об'єктно-орієнтованим розширенням мови. Ці розширення були включені в стандарт і стали важливою частиною сучасних реляційних СУБД. Такі СУБД отримали назву об'єктно-реляційних СУБД. Однією з характеристичних особливостей об'єктно-реляційних СУБД є можливість успадкування таблиць. У них використовується одиночне успадкування таблиць. У той же час існує практична потреба в множинному успадкуванні таблиць. Це дозволило б більш адекватно моделювати предметну область і описувати сутності реального світу. При множинному успадкуванні виникає проблема конфлікту імен. Вона полягає в тому, що в батьківських таблицях різні колонки можуть мати однакові імена. В цьому випадку виникає питання, яку саме колонку включати в дочірню таблицю. Аналогічна проблема, свого часу, виникла в об'єктно-орієнтованих мовах програмування з множинним успадкуванням. Один з методів вирішення проблеми в об'єктно-орієнтованих мовах програмування полягає в лінеаризації батьківських об'єктів. У цьому методі на батьківських об'єктах задається відношення лінійного

порядку. Потім вибирається найменший об'єкт в сенсі цього порядку. Іншими словами, всі батьківські об'єкти шикуються в лінію. Атрибут того об'єкту, який розташовується лівіше інших об'єктів, що мають атрибути з такими ж іменами і вибирається в дочірній об'єкт. Були розроблені численні алгоритми лінеаризації. Найпоширенішим серед них виявився алгоритм MRO C3. Він і був обраний в якості основи для вирішення конфліктів імен при множині успадкуванні таблиць. Як виявилося, цей алгоритм був сформульований на напівформальному рівні, що унеможливлювало строге доведення його властивостей і коректності. У дисертації алгоритм MRO C3 був модифікований для застосування до таблиць і заданий строго, з використанням псевдокоду. Формально доведена його завершаємость.

Що стосується властивостей алгоритму, то були розглянуті такі важливі для лінеаризації властивості як монотонність і збереження локального порядку успадкування. Для цих властивостей також були відсутні математичні визначення, що унеможливлювало доказ їх наявності або відсутності у алгоритму. Для формального визначення властивостей алгоритму було введене поняття ієрархії успадкування таблиць, яке описує як одиночне, так і множинне успадкування таблиць. Сформульовано і доведено теореми про характерні ознаки одиночного успадкування.

На основі введених визначень була доведена монотонність алгоритму MRO C3. Модифікований для таблиць алгоритм не зберігає локальний порядок успадкування, що було показано на відповідному прикладі.

Що стосується семантики власне SQL, то було проведено уточнення табличній алгебри семантичних функцій SQL. Відзначимо, що дана алгебра має ряд переваг над іншими способами завдання семантики SQL. А саме, в табличній алгебри виділяється два типи функцій - функції на таблицях, і функції на функціях, які називаються операторами, як це прийнято в мовах програмування. Семантика складних виразів SQL в цій алгебрі природним чином будується з функцій на таблицях і операторів.

Було проведено порівняння результатів запитів мови SQL і результатів обчислень відповідних їм семантичних функцій табличній алгебри. Виявлено розбіжності в результатах виконання запитів, які містять операції зовнішнього та внутрішнього з'єднання і відповідним їм семантичним функціям на таблицях, які мають співпадаючі імена атрибутів. Також виявлено розбіжності при виконанні теоретико-множинних операцій на таблицях, пов'язані з тим, що в SQL операції об'єднання, перетину і різниці таблиць враховують порядок стовпців, заданий в запитах, а в семантичної алгебри враховуються імена атрибутів стовпців, а порядок не є важливим.

Запропоновано нові визначення для операцій внутрішнього і зовнішнього з'єднання і теоретико множинних операцій. В результаті, уточнена таблична алгебра семантичних функцій більш точно описує семантику ядра мови SQL і є більш зручною у використанні.

Проведена перевірка семантики оператора ORDER BY мови SQL. Виявлено, що семантичні функції задають частковий порядок на рядках, в той час як фраза SQL ORDER BY завжди повертає лінійний порядок на рядках таблиці. Сформульовано і доведено теорему, яка визначає умови, при яких семантичні функції задають точну семантику оператора ORDER BY.

Результати роботи були впроваджені у навчальний процес за спеціальністю "Інформатика" на факультеті кібернетики КНУ (нормативні курси "Прикладна логіка", "Композиційна семантика SQL-подібних мов"; спеціальний курс "Вступ до реляційних баз даних"), а також у Кіровоградському державному педагогічному університеті імені Володимира Винниченка (нормативний курс "Табличні алгебри та SQL подібні-мови").

Основні положення та висновки дисертаційного дослідження обговорювалися на наукових семінарах кафедри теорії та технології програмування КНУ, республіканському семінарі "Програмологія та її застосування".

Результати дисертаційного дослідження оприлюднені у доповідях і повідомленнях на Міжнародних та Всеукраїнських наукових конференціях, семінарах:

XI Міжнародна науково-практична конференція: ТЕОРЕТИЧНІ ТА ПРИКЛАДНІ АСПЕКТИ ПОБУДОВИ ПРОГРАМНИХ СИСТЕМ. – Київ, 15-17 грудня 2014 р.

I Международный научно-практический форум «Наука и бизнес»: – Черновцы, 2015.

13th International Scientific Conference on Informatics. – Poprad, Slovakia, 18-20 Nov., 2015.

XII Міжнародна науково-практична конференція ТЕОРЕТИЧНІ ТА ПРИКЛАДНІ АСПЕКТИ ПОБУДОВИ ПРОГРАМНИХ СИСТЕМ. – Київ, 23-26 грудня 2015 року.

XIII Міжнародна науково-практична конференція ТЕОРЕТИЧНІ ТА ПРИКЛАДНІ АСПЕКТИ ПОБУДОВИ ПРОГРАМНИХ СИСТЕМ. – Київ, 5-9 грудня 2016 року.

XII International Conference on Perspective Technologies and Methods in MEMS Design (MEMSTECH). – 2016.

Міжнародний науково-практичний семінар – "Комбінаторні конфігурації та їх застосування" – 2016.

Четвёртая конференция «КОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ В НАУКОЕМКИХ ТЕХНОЛОГИЯХ» (КМНТ-2016). – Харьков, 26-31 мая 2016 г

Головним результатом дисертації є розширення табличної алгебри семантичних функцій мови SQL одиночним та множинним успадкуванням таблиць з автоматичним розв’язанням конфліктів імен атрибутів за допомогою алгоритму лінеаризації, відомого під назвою MRO C3. Це розширення розв’язує

важливу задачу побудови математичної моделі об’єктно-реляційних баз даних, що має велике теоретичне та практичне значення для розробки перспективних комп’ютерних інформаційних систем.

Основні результати також включають в себе:

- Для більш точного опису семантики ядра мови SQL та більшої зручності у використанні замість старих визначень в табличній алгебрі семантичних функцій SQL запропоновані нові визначення операцій внутрішнього та зовнішнього з’єднання, а також теоретико множинних операцій.
- Доведена теорема, яка визначає умови, при яких семантичні функції задають точну семантику оператора ORDER BY. Це показало, що повна семантика ORDER BY може бути задана лише недетермінованими функціями.
- Доведені теореми про характеристичні ознаки одиночного успадкування. На їх базі розроблені алгоритми перевірки, чи є ієрархія успадкування одиночним чи множинним успадкуванням, та чи є ієрархія одиночного успадкування простою.
- Формально визначені такі важливі властивості методу лінеаризації ієрархії таблиць, як монотонність та збереження локального порядку успадкування, що дає можливість оцінити той чи інший метод лінеаризації та визначити, наскільки він є безпечним.
- Побудовано алгоритм MRO C3 для автоматичного вирішення конфлікту співпадаючих імен атрибутів при множинному успадкуванні таблиць. Це дозволило перевіряти властивості алгоритму на строгому математичному рівні. Зокрема доведено теореми про такі властивості алгоритму, як монотонність та його завершуваність. Алгоритм розроблювався на псевдокоді, що зробило його незалежним від мови програмування. Додатково він був реалізований на мові програмування Python. Проведене

тестування програмної реалізації також підтвердило коректність доведених теорем.

Ключові слова: реляційні бази даних, семантика SQL, об’єктно-реляційні бази даних, об'єктно-орієнтовані мови програмування, розв’язання конфлікту імен при множинному успадкуванні, алгоритми лінеаризації.

SUMMARY

Mohammed K. D. Extension of table algebras with multiple inheritance. Qualifying scientific work on the rights of manuscripts.

The dissertation for the degree of a candidate of physical and mathematical sciences in the specialty 01.05.03 "Mathematical and software of computing machines and systems" (12 - Information technologies). – Taras Shevchenko National University of Kyiv, Kyiv, 2017.

In the dissertation work there were set two goals. One goal was to formalize the algorithm of multiple inheritance of tables and study its properties. Another goal was to clarify and further develop the table algebra of the semantic functions of the language of SQL queries.

Although SQL semantics are devoted to numerous theoretical works, they do not pay enough attention to object-oriented language extensions. These extensions have been included in the standard and have become an important part of modern relational DBMS. Such DBMS’s are called object-relational DBMS. One of the characteristic features of object-relational DBMS is the possibility of inheritance of tables. They use a single inheritance of tables. At the same time, there is a practical need for multiple inheritance of tables. This would allow to have more adequately model of the subject area and describe the essence of the real world. When multiple inheritance is used there is a problem of name conflict. It consists in the fact that different columns in parents tables may have identical names. In this case, the

problem arises as to which column to include in the child table. A similar problem, at one time, arose in object-oriented programming languages with multiple inheritance. One of the methods for solving the problem in object-oriented programming languages is to linearize the parent objects. In this method, the relation of the linear order is established on the parent objects. Then the smallest object is selected in the sense of this order. In other words, all parent objects are lined up in a line. An attribute of an object that is left of the other objects with the same attributes names and is selected as a child object. Many linearization algorithms were developed. The most common among them was the MRO C3 algorithm. He was chosen as the basis for resolving the name conflicts with multiple inheritance tables. As it turned out, this algorithm was formulated at the semi-formal level, which made impossible the rigorous proof of its properties and correctness. In the dissertation, the MRO C3 algorithm has been modified to be applied to tables and is given strictly, using a pseudocode. Formally proved its completeness.

With regards to the properties of the algorithm, such important properties for the linearization as monotony and preservation of the local order of inheritance were considered. For these properties, there were also no mathematical definitions that made it impossible to prove their presence or absence in the algorithm. For the formal definition of the properties of the algorithm, the definition of the table inheritance hierarchy was introduced, which describes both the singular and the multiple inheritance of the tables. Theorems on the characteristic features of a single inheritance are formulated and proved.

On the basis of the introduced definitions, the monotony of the MRO C3 algorithm was proved. The modified for table inheritance algorithm does not preserve the local order of inheritance, which was shown in the corresponding example.

As for the semantics of SQL itself, a clarification of the table algebra of semantic SQL functions was done. Note that this algebra has several advantages over other methods of the task of SQL semantics. Namely, in table algebra there are two types of functions – functions on tables, and functions on functions that are called

operators, as is usially in programming languages. The semantics of complex expressions of SQL in this algebra is naturally constructed from functions on tables and operators.

A comparison was made between the results of SQL query requests and the results of calculations of the corresponding semantic functions of the table algebra. Differences in the results of queries that contain operations of the inner and outer joins and corresponding semantic functions of table algebra were found. Also, there are differences in the implementation of theoretical-set operations on tables associated with the fact that in SQL operations of union, intersection and table differences take into account the order of the columns given in queries, and in the semantic algebra, the names of column are taken into account, and their order is not is important.

New definitions for inner and outer joins operations and the theoretical-set operations are proposed. As a result, the updated table algebra of semantic functions more accurately describes the semantics of the core of the SQL language and is more convenient to use.

The semantics of ORDER BY operator SQL is checked. It is found that semantic functions specify a partial order on the rows, whereas the SQL operator ORDER BY always returns the linear order on the rows of the table. A theorem is formulated and proved, which defines the conditions under which semantic functions specify the exact semantics of the ORDER BY operator.

The results of the work were introduced into the educational process by the specialty "Informatics" at the Faculty of Cybernetics of the KNU (normative courses "Applied Logic", "Compositional Semantics of SQL-like Languages", special course "Introduction to Relational Databases"), as well as in the Kirovograd State Pedagogical Volodymyr Vynnychenko University (normative course "Table algebras and SQL-like-languages").

The main provisions and conclusions of the dissertation research were discussed at the scientific seminars of the Department of Theory and Programming Technology of the KNU, the republican seminar "Programmology and its application".

The results of the dissertation research were published in the reports and anouncements at the International and Ukrainian scientific conferences, seminars:

XI International Scientific and Practical Conference: THEORETICAL AND APPLIED ASPECTS OF THE DEVELOPMENT OF SOFTWARE SYSTEMS. - Kyiv, December 15-17, 2014

I the International Scientific and Practical Forum "Science and Business": - Chernivtsi, 2015.

13th International Scientific Conference on Informatics. - Poprad, Slovakia, Nov. 18-20, 2015.

XII International Scientific and Practical Conference THEORETICAL AND APPLIED ASPECTS OF THE PROCESSING OF SOFTWARE SYSTEMS. - Kyiv, December 23-26, 2015.

XIII International Scientific and Practical Conference THEORETICAL AND APPLIED ASPECTS OF SOFTWARE SYSTEMS. - Kyiv, December 5-9, 2016.

XII International Conference on Perspective Technologies and Methods in MEMS Design (MEMSTECH). - 2016

International scientific and practical seminar - "Combination configurations and their applications" - 2016.

Fourth Conference "COMPUTER SIMULATION IN SCIENTIFIC TECHNOLOGIES" (KMNT-2016). - Kharkiv, May 26-31, 2016

The main result of the dissertation is the extending of the table algebra of the semantic functions of the SQL language with single and multiple tables inheritance

with the automatic resolution of the attribute name conflicts using the linearization algorithm, known as the MRO C3. This extension solves the important task of constructing a mathematical model of object-relational databases that has great theoretical and practical significance for the development of promising computer information systems.

The main results also include:

• For a more precise description of the semantics of the core of the SQL language and the greater convenience of using instead of the old definitions in the table algebra of semantic SQL functions, new definitions of internal and external connection operations and the theory of multiple operations are proposed.

• A theorem is proved that defines the conditions under which semantic functions specify the exact semantics of the ORDER BY operator. This showed that the full semantics of ORDER BY can be given only by non-deterministic functions.

• Proved theorems on the characteristic features of a single inheritance. Used their theorems, algorithms for checking whether a hierarchy of inheritance is single or multiple inheritance is developed.

• Formally defined such important properties of the linearization method of the table hierarchy as monotony and preserving the local inheritance order, which makes it possible to evaluate one or another method of linearization and determine how safe it is.

• The MRO C3 algorithm is constructed to automatically resolve the conflict of matching attribute names when multiple tables are inherited. This allowed to check the properties of the algorithm at a strict mathematical level. In particular, theorems on the properties of the algorithm, such as monotony and its completeness, are proved. The algorithm was developed on a pseudocode, which made it independent of the programming language. Additionally, it was implemented in the Python

programming language. The conducted testing of software implementation also confirmed the correctness of the proved theorems.

Keywords: relational databases, SQL semantics, object-relational databases, object-oriented programming languages, resolving the name conflict with multiple inheritance, linearization algorithms.

Список публікацій здобувача

1. Мохаммед К.Д. Алгоритми CLOS та LOOPS лінеаризації класів в мовах програмування: формальна побудова / К.Д. Мохаммед, Д.Б. Буй, С.В. Компан, С.А. Поляков // Вісник Київського національного університету імені Тараса Шевченка. Серія: Фізико-математичні науки. – 2015. – Вип. 2. – С. 99-102. http://www.library.univ.kiev.ua/ukr/host/10.23.10.100/db/ftp/visnyk/fiz_mat_2_2015.pdf

2. Мохаммед К.Д. Відношення конфінальності, передпорядки та порядки, семантика фрази ORDER BY запитів SQL подібних мов[Електронний ресурс] / Д. Б. Буй, Н. Д. Кахута, О. В. Шишацька, Sunmade Fabunmi, К.Д. Мохаммед // Вісник Київського національного університету імені Тараса Шевченка. Серія: Фізико-математичні науки. – 2015. – Вип. 4. – С. 88-95. – Режим доступу: http://nbuv.gov.ua/UJRN/VKNU_fiz_mat_2015_4_16

3. Мохаммед К.Д. Логики частичных предикатов, индуцированные трехзначными логиками Клини / Д.Б. Буй, Е.В. Шишацкая, Ф. Санмейд, К.Д. Мохаммед // Штучний інтелект.– 2015. – №3-4. – С. 84-88. (Режим доступу: http://nbuv.gov.ua/UJRN/II_2015_3-4_10)

4. Мохаммед К.Д. Математические основания множественного наследования: рефлексивно-транзитивное замыкание. / Д.Б. Буй, Е.В.

Шишацкая, Ф. Санмейд, К.Д. Мохаммед // Вісник Харківського національного університету ім. В.Н. Каразіна. – 2016. – №28. - С .19-33 http://periodicals.karazin.ua/mia/article/viewFile/6549/6058

5. Мохаммед К.Д. Рефлексивно-транзитивные замыкания бинарных отношений. / К.Д. Мохаммед, Д.Б. Буй, Е.В. Шишацкая, Ф. Санмейд // Электротехнические и компьютерные системы. – 2016. – №22(98). – С.272-276. http://www.etks.opu.ua/?fetch=articles&with=info&id=806

6. Мохаммед К.Д. Характеристичні властивості одиночного успадкування класів. / К.Д. Мохаммед // Вісник Київського національного університету імені Тараса Шевченка. Серія: Фізико-математичні науки. – 2016. – Вип. 4. – С. 104-107.

7. Mohammed K. Join Operations in Relational Databases with Automatic Attributes Renaming / Poliakov S, Buy D, Mohammed K., Israa Jasim AL.kalafa // Scholars Journal of Engineering and Technology. – 2017. –№5(6) - P.254-257. http://saspublisher.com/sjet-56/

8. Мохаммед К.Д. Огляд типів рекомендаційних систем та використання баз даних для підвищення їх продуктивності / К.Д. Мохаммед, Буй Д.Б., Компан С.В., Поляков С.А. // XI Міжнародна науково-практична конференція: ТЕОРЕТИЧНІ ТА ПРИКЛАДНІ АСПЕКТИ ПОБУДОВИ ПРОГРАМНИХ СИСТЕМ. – Київ, 15-17 грудня 2014 р. - с 29-34- http://taapsd.at.ua/TAAPSD-2014.pdf

9. Мохаммед К.Д. Математические основания современных реляционных СУБД / К.Д. Мохаммед, Д. Б. Буй, , Н. Д. Кахута / I Международный научно-практический форум «Наука и бизнес»: Тезисы докладов. – Черновцы, 2015. – С. 52-55- http://library.krok.edu.ua/media/library/category/statti/kakhuta_0008.pdf

10. Mohammed K. Linearization algorithms CLOS and LOOPS of the classes in programming languages: the formal definitions / K. Mohammed, D. Buy, J. Karam,

S. Kompan, S. Polyakov.// 13th International Scientific Conference on Informatics. – Poprad, Slovakia, 18-20 Nov., 2015. – P. 63-66 (Print ISBN: 978-1-4673-9867-1, DOI 10.1109/Informatics.2015.7377809). http://ieeexplore.ieee.org/document/7377809/

11. Мохаммед К.Д. Реализация работы нестандартных типов данных в Framework Django на примере типа данных ltree / К.Д. Мохаммед, Д.Б.Буй, С.В.Компан, С.А. Поляков // XII Міжнародна науково-практична конференція ТЕОРЕТИЧНІ ТА ПРИКЛАДНІ АСПЕКТИ ПОБУДОВИ ПРОГРАМНИХ СИСТЕМ. – Київ, 23-26 грудня 2015 року. - с 26-32- http://taapsd.at.ua/

12. Мохаммед К.Д. Недетерминированные функции в SQL / К.Д. Мохаммед, Буй Д.Б., Поляков С.А. // XIII Міжнародна науково-практична конференція ТЕОРЕТИЧНІ ТА ПРИКЛАДНІ АСПЕКТИ ПОБУДОВИ ПРОГРАМНИХ СИСТЕМ. – Київ, 5-9 грудня 2016 року. - с 44-49-http://phd.isofts.kiev.ua/taapsd-2016/

13. Mohmmed K. Mathematical Foundations of Multiple Inheritance: Reflexive-transitive Closure of the Binary Relations / Dmytro Buy, Olena Shyshatska, Sunmade Fabunmi, Karam Mohammed // Perspective Technologies and Methods in MEMS Design (MEMSTECH), 2016 XII International Conference on Perspective Technologies and Methods in MEMS Design (MEMSTECH). – 2016. – С.192-195. (http://ieeexplore.ieee.org/document/7507540/)

14. Мохаммед К.Д. Математические основания множественного наследования: рефлексивно-транзитивное замыкание / К.Д. Мохаммед, Д.Б. Буй, Е.В. Шишацкая, Ф. Санмейд // Міжнародний науково-практичний семінар – "Комбінаторні конфігурації та їх застосування" – 2016. http://it-kntu.kr.ua/2016/04/18/results-cca-2016/#more-3782

15. Mohmmed K. Математические основы множественного наследования в ООП /K. Mohmmed, Буй Д.Б., Шишацкая Е.В., Fabunmi S. // Четвёртая

конференция «КОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ В НАУКОЕМКИХ ТЕХНОЛОГИЯХ» (КМНТ-2016). – Харьков, 26-31 мая 2016 г (http://www.dsmmph.org.ua/kmnt.html)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

$\varnothing$	порожня множина
$\overset{def}{\Leftrightarrow}$	за означенням
$\overset{def}{=}$	дорівнює за означенням
$\overline{O}$	доповнення множини O
π	відношення конфінальності на множинах $P \pi Q \overset{def}{\Leftrightarrow} \forall x(x \in P \Rightarrow \exists y(y \in Q \& x \le y))$
$\tilde{=}$	узагальнена рівність
$U\|X$	обмеження бінарного відношення U за множиною X
$U[X]$	повний образ множини X відносно бінарного відношення U
$[a,x)$	початковий відрізок ц. у. м.
$\tilde{\to}$	стрілка для запису часткових операцій
$U \circ V$	композиція бінарних відношень
$\operatorname{rang}\psi$	область значень операції ψ
π_i^n	проекція множини кортежів довжини n за i-тою компонентою
$\overline{1,n}$	множина натуральних чисел $\{1,...,n\}$
$(D \tilde{\to} D')$	сім'я часткових функцій з множини D у множину D'
$\approx$	бінарне відношення сумісності функцій
$\operatorname{dom} f$	область означеності функції f
true	істинне булеве значення
false	хибне булеве значення

$P(X)$	булеан множини X
2^X	множина всіх скінченних підмножин множини X
U^{-1}	бінарне відношення, обернене (інверсне) відношенню U
R	унарна композиція рекурсії для випадку одного параметра
$(D_1 \times ... \times D_n \times D \underset{m}{\overset{n+1}{\to}} D)$	підклас функцій класу $(D_1 \times ... \times D_n \times D \to D)$, монотонних за $n+1$-им аргументом; D – ч. в. м.
X^Y	множина всіх скінченних функціональних відношень на парі множин Y, X
$\underset{\Delta}{\overset{*}{\to}}$	транзитивне замикання відношення $\underset{\Delta}{\to}$
$X\|U$	приведення бінарного відношення U за множиною X
$\|A\|$	носій алгебри
$[\Sigma]_\Omega$	замикання множини Σ операціями сім'ї Ω
ε	порожня іменна множина (розд. 4); рядок порожньої схеми (розд. 5)
$\underline{\pi}$	рефлексивно-транзитивне замикання відношення $<$
(d_1, d_2)	іменна множина зі стандартними іменами $\{<1, d_2>, <2, d_2>\}$
$\bar{\Upsilon}$	об'єднання сумісних іменних множин
$\mathfrak{V}$	об'єднання s-множин
$\mathfrak{f}$	різниця s-множин
Ω_n	сигнатура програмних алгебр іменних функцій
$\Rightarrow v$	параметрична функція іменування, v – ім'я
$v \Rightarrow$	параметрична функція розіменування за іменем v
$\equiv$	іменний предикат рівності іменних даних
Υ_R, I_R, $\setminus_R$	відповідно об'єднання, перетин, різниця таблиць схеми R
$\otimes$	з'єднання, еквіз'єднання, внутрішнє природне з'єднання

$\div\limits_{R_2}^{R_1}$	ділення таблиць схеми R_1 на таблиці схеми R_2
$t_\varnothing$	порожня таблиця
t_ε	таблиця, що містить єдиний рядок ε порожньої схеми; $t_\varepsilon \stackrel{def}{=} \{\varepsilon\}$
$\cong$	відношення односхемності (таблиць), $t_1 \cong t_2 \stackrel{def}{\Leftrightarrow} \exists R(t_1 \in T(R) \& t_2 \in T(R))$
$\cong_S$	відношення односхемності рядків
$\cup_t, \cap_t, \setminus_t$	відповідно операції об‘єднання, перетину, різниці односхемних таблиць
Cj	операція декартова з‘єднання (cross join, Cartesian join)
$\underset{A_1,\ldots,A_n}{\otimes}$	операція внутрішнього з’єднанням за атрибутами $A_1,\ldots,A_n$, $n \geq 1$ (inner join using $A_1,\ldots,A_n$)
$\underset{p}{\otimes}$	операція внутрішнього з’єднанням за предикатом p (inner join on p)
$=_S$	тризначний предикат рівності мови SQL
$\times_m$	операція декартова з‘єднання мультимножин
$Orderby_A^{\eta}$	параметрична функція побудови у-таблиць
СУБД	система управління базою даних

ВСТУП

Перші реляційні СУБД були розроблені ще в 70-ті роки минулого століття. На початку 90-х років, до реляційних баз даних були застосовані ідеї об'єктно-орієнтованого підходу, що привело до появи об'єктно-реляційних систем управління базами даних (ОРБД), які поєднують функціональність реляційних СУБД з концепціями та ідеями взятими з об'єктно-орієнтованого програмування. В стандартах SQL:1999 - SQL:2016 в реляційну модель даних були додані численні об'єктно-орієнтовані риси. Це значно змінило модель даних, що лежить в основі останніх версій СУБД, базованих на цих стандартах. На сьогодні, незважаючи на існування інших моделей даних[80, 94, 121, 30], реляційні та об'єктно-реляційні бази даних залишаються основою інформаційних систем підприємств та організацій.

Семантиці SQL присвячені численні теоретичні роботи, починаючи, наприклад, з роботи в якій була побудована формальна модель запитів, названа E3VPC [36], та закінчуючи публікацією в 2017 році денотаційної семантики SQL, названої HoTTSQL [46] Потужність реляційних СУБД значною мірою пояснюється тим, що вони базуються на строгих математичних основах, викладених ще Кодом [12, 13]. В той же час, всі ці роботи не приділяють уваги об'єктно-орієнтованим властивостям СУБД, зосередившись, на структурах даних та операціях над ними.

На кафедрі теорії та технології програмування Київського національного університету імені Тараса Шевченка розвивається побудова семантики SQL на основі спеціальної програмної алгебри, названої табличною алгеброю семантичних функцій [63, 26, 60, 115, 50, 49, 53, 54, 55, 56, 97, 57, 94, 96, 99, 100, 93, 95], яка має ряд переваг над іншими підходами. А саме, в табличній алгебрі виділяється два типи функцій – функції на даних, зокрема таблицях, та функції на функціях, які називаються операторами, як це прийнято в інших

мовах програмування. Семантика складних виразів SQL в цій алгебрі природнім чином будується з функцій на даних і операторів.

Слід відмітити, що такий підхід до визначення семантики мов програмування був запропонований в роботах по системам алгоритмічних алгебр (САА) В.М. Глушкова, Г.Е. Цейтліна, Е.Л. Ющенко[65], а згодом в роботах по композиційній семантиці В.Н. Редька, Д.Б. Буя та М.С. Нікітченка [102, 106, 107, 108, 109, 110]. Ці праці стали теоретичним підґрунтям Київської школи програмування (http://www.computer-museum.ru/galglory/27.htm).

Потреба у розвитку теорії до рівня сучасних ОРБД, зокрема семантики запитів мови SQL, та семантики об'єктних розширень реляційної моделі даних, і визначає актуальність даної дисертаційної роботи в який розширюється таблична модель даних за рахунок множинного успадкування таблиць, та удосконалена таблична алгебра семантичних функцій SQL, зокрема перевизначені оператори внутрішнього та зовнішнього з'єднання.

Зв'язок роботи з науковими програмами, планами, темами. Дисертаційна робота є складовою частиною наукових робіт, проведених на кафедрі теорії та технології програмування факультету комп'ютерних наук та кібернетики Київського національного університету імені Тараса Шевченка при виконанні фундаментальної теми "Формальні специфікації та методи розробки надійних програмних систем" (№ 0111U007052, 2011-2015 рр.) і теми «Теорія і методи розробки інтелектуальних інформаційних технологій та систем» (№16КФ015-02).

Мета і задачі дисертаційного дослідження. *Метою* дисертаційної роботи є аналіз алгебри семантичних функцій ядра мови SQL, яка називається табличною алгеброю, на відповідність реалізаціям SQL в поширених СУБД, зокрема операцій з'єднання та теоретико-множинних операцій; знаходження розбіжностей, тобто випадків, коли результати обчислень семантичних функцій відрізняються від результатів, які повертають відповідні запити реалізовані в

мові SQL; модифікація алгебри семантичних функцій для уникнення таких розбіжностей; розширення реляційної моделі даних конструкцією одиночного і множинного успадкування таблиць.

З огляду на мету в роботі ставляться такі *задачі*:

провести перевірку табличної алгебри ядра мови SQL. Перевірку провести на версії SQL, втіленої в об'єктно-реляційної СУБД PostgreSQL, яка є однією з найбільш точних та повних реалізацій стандарту SQL. При цьому особу увагу приділити обчисленню запитів на таблицях, які мають співпадаючи імена стовбців, або які є пустими

перевизначити семантику операторів внутрішнього та зовнішнього з'єднання реляцій таким чином, щоб результати з'єднання для реляцій, імена атрибутів яких частково або повністю співпадають, повертали такі самі реляції як і в мові SQL, по можливості, усунути також інші розбіжності

перевірити семантичні функції, які задають семантику фрази ORDER BY. Дослідити властивості семантичних функцій, які відповідають за впорядкування рядків таблиць

дослідити алгоритми автоматичного вирішення конфліктів імен атрибутів при множинному успадкуванні, та застосувати ці алгоритми до успадкування таблиць

дослідити властивості методу лінеаризації, який використовується в алгоритмах множинного наслідування

Предметом дослідження є реляційна модель даних.

Об'єктом дослідження є реляційні бази даних та об'єктно-реляційні бази даних.

Наукова новизна одержаних результатів. Проведене порівняння результатів запитів ядра мови SQL та результатів обчислень відповідних їм

семантичних функцій табличної алгебри. Виявлені розбіжності в результатах виконання запитів, які містять операції зовнішнього та внутрішнього з'єднання та відповідним їм семантичним функціям на таблицях, які мають спільні імена атрибутів. Також виявлені розбіжності при виконанні теоретико-множинних операцій на таблицях, пов'язані з тим, що в SQL операції об'єднання, перетину та різниці таблиць враховують порядок стовбців заданий в запитах, а в семантичній алгебрі враховуються імена атрибутів стовбців, а порядок не є важливим.

Запропоновані нові визначення для операцій внутрішнього та зовнішнього з'єднання та теоретико множинних операцій, визначена нова форма оператору суперпозиції, більш зручна для задання семантики складних запитів. В результаті, поновлена алгебра семантичних функцій більш точно описує семантику ядра мови SQL та є більш зручною в використанні.

Проведена перевірка семантики фрази ORDER BY мови SQL. Виявлено, що семантичні функції задають частковий порядок на кортежах, в той час як фраза SQL ORDER BY завжди повертає лінійний порядок на рядках таблиці. Сформульована та доведена теорема, яка визначає умови, при яких семантичні функції задають точну семантику ORDER BY.

Введено поняття ієрархії успадкування, яке описує як одиночне так і множинне успадкування таблиць. Сформульовані та доведені теореми про характеристичні ознаки одиночного успадкування. Надані формальні математичні визначення для таких властивостей методу лінеаризації ієрархії таблиць, як монотонність та збереження локального порядку успадкування.

Що стосується практичної частини, то розроблені алгоритми перевірки, чи є ієрархія успадкування одиночним успадкуванням чи множинним, та чи є ієрархія одиночного успадкування простою.

Надане формальне математичне визначення алгоритму MRO C3 для автоматичного вирішення конфлікту співпадаючих імен атрибутів при

множинному успадкуванні таблиць. Доведені теореми про монотонність алгоритму та його завершуваність.

Теоретичне і практичне значення одержаних результатів. Дисертація має теоретико-прикладну спрямованість. Результати роботи можуть використовуватись для задання семантики запитів мови SQL, для оптимізації запитів, досліджені властивостей мови SQL, доведенні коректності запитів, для еквівалентного перетворенні запитів та доведення еквівалентності запитів, проектуванні баз даних.

Результати роботи були впроваджені у навчальний процес за спеціальністю "Інформатика" на факультеті кібернетики КНУ (нормативні курси "Прикладна логіка", "Композиційна семантика SQL-подібних мов"; спеціальний курс "Вступ до реляційних баз даних"), а також у Кіровоградському державному педагогічному університеті імені Володимира Винниченка (нормативний курс "Табличні алгебри та SQL подібні-мови").

Особистий внесок здобувача. Всі результати, які складають суть дисертаційної роботи, отримані здобувачем самостійно. З праць, виконаних зі співавторами, на захист виносяться лише результати, отримані особисто здобувачем. У спільно виконаних роботах науковому керівнику Д.Б. Бую належить загальна постановка проблеми, обговорення та інтерпретація результатів.

Апробація результатів дослідження. Основні положення та висновки дисертаційного дослідження обговорювалися на наукових семінарах кафедри теорії та технології програмування КНУ, республіканському семінарі "Програмологія та її застосування".

Результати дисертаційного дослідження оприлюднені у доповідях і повідомленнях на Міжнародних та Всеукраїнських наукових конференціях, семінарах:

XI Міжнародна науково-практична конференція: ТЕОРЕТИЧНІ ТА ПРИКЛАДНІ АСПЕКТИ ПОБУДОВИ ПРОГРАМНИХ СИСТЕМ. – Київ, 15-17 грудня 2014 р.

I Международный научно-практический форум «Наука и бизнес»: – Черновцы, 2015.

13th International Scientific Conference on Informatics. – Poprad, Slovakia, 18-20 Nov., 2015.

XII Міжнародна науково-практична конференція ТЕОРЕТИЧНІ ТА ПРИКЛАДНІ АСПЕКТИ ПОБУДОВИ ПРОГРАМНИХ СИСТЕМ. – Київ, 23-26 грудня 2015 року.

XIII Міжнародна науково-практична конференція ТЕОРЕТИЧНІ ТА ПРИКЛАДНІ АСПЕКТИ ПОБУДОВИ ПРОГРАМНИХ СИСТЕМ. – Київ, 5-9 грудня 2016 року.

XII International Conference on Perspective Technologies and Methods in MEMS Design (MEMSTECH). – 2016.

Міжнародний науково-практичний семінар – "Комбінаторні конфігурації та їх застосування" – 2016.

Четвёртая конференция «КОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ В НАУКОЕМКИХ ТЕХНОЛОГИЯХ» (КМНТ-2016). – Харьков, 26-31 мая 2016 г

Публікації. Результати дисертації опубліковані у 7 статтях [81, 82, 83, 84, 89, 90, 32] в наукових журналах і збірниках наукових праць, 8 тезах конференцій [91, 87, 33, 88, 86, 35, 85, 34]; з них 5 статей опубліковані у

фахових виданнях, затверджених ВАК України, одна стаття опублікована в міжнародному журналі.

Структура та обсяг дисертації. Дисертаційна робота з анотації, переліку умовних позначень, вступу, чотирьох розділів, висновків, списку використаних джерел. Загальний обсяг дисертації становить 160 с, основний текст 110 с., список використаних джерел 121 найменування на 15 сторінках.

ЗМІСТОВНА СЕМАНТИКА МОВИ SQL

1.1 Огляд мови SQL.

В розділі надається змістовна семантика SQL[47, 75, 98, 101, 20, 29, 45, 19, 25, 44] на прикладі об’єктно-реляційної СУБД PostgreSQL [28, 42, 5, 37, 27]. Ця СУБД втілює об’єктні та реляційні оператори які є близькими до стандартів SQL [20, 29] ; вона використовується в багатьох крупних проектах, останнім часом її популярність зростає.

Мова SQL складається з багатьох операторів. На логічному рівні вони групуються в сукупності, які називаються теж мовами. Найбільш важливими є DDL (Data Definition Language) – мова визначення даних, DML (Data Manipulation Language) – мова маніпулювання даними, DCL (Data Control Laguage) – мова визначення привілей користувача. Далі, в розділі, будуть розглядатися DDL та DML, семантика яких визначається в наступних розділах.

Дані в реляційних базах даних зберігаються у спеціальних структурах, які в теорії баз даних називаються реляціями, а в СУБД – таблицями [24, 14, 15]. Хоча таблиці і реляції описують одну і туж саму сутність, вони мають деякі відмінності між собою, які будуть розглянуті в наступному розділі, коли буде надане визначення реляції. Ці відмінності пояснюються тим, що таблиці є програмною реалізацією чисто математичного поняття реляцій [24]. Взаємозв’язок між таблицями та реляціями можна пояснити за аналогією зв’язку між дійсними числами, які є математичною абстракцією, та числами з плаваючою точкою, які втілюють цю абстракцію в комп’ютерах. В даному розділі розглянемо семантику саме таблиць.

Таблиці в реляційних базах даних є формалізацією поняття звичайних таблиць. Вони складаються з рядків і стовпців, які мають імена. Перерахуємо основні властивості таблиць в реляційних баз даних[15, 74, 67, 68, 23, 4, 17]:

- Рядки в таблиці не впорядковані, тобто при виконані операцій, коли потрібен послідовний доступ до рядків таблиці, вони можуть перераховуватися в різному порядку. Така неоднозначність приводить до того, що в SQL операції, які залежать від порядку рядків є недетермінованими. Наприклад операція LIMIT, яка повертає n перших рядків або ORDER BY яка сортує рядки.

- Імена стовпців в таблиці є унікальними, а самі стовпці впорядковані. Таким чином до стовпця можна звертатися як по імені та і по порядковому номеру. Як правило, звернення виконується по імені, але є деякі операції в яких допускається звернення як по імені так і по номеру стовпця. В теоретико-множинних операціях об'єднання, перетину і різниці відповідність стовпців визначаються виключно по їх поряду розташування.

- Рядки можуть співпадати. Як правило, це є помилкою. Для заборони дублікатів рядків використовуються спеціальні механізми, такі як ключі або вимоги унікальності значень деяких стовпців.

- Значення кожного стовпця відносяться до одного типу даних, наприклад INTEGER або DATE. Крім того значення можуть бути невизначеними. Для цього використовується спеціальний символ NULL, який обробляється спеціальним чином.

- В PostgreSQL, як і в інших об'єктно-реляційних СУБД, в комірках таблиці значеннями можуть бути не тільки атомарні дані а і багатовимірні масиви однотипних даних, композитні типи даних, та слабоструктуровані дані у форматі JSON

Далі зображені приклади таблиць PostgreSQL. Таблиця 1.1 містить дані про дати народження людей. Для цього достатньо використовувати тільки атомарні типи даних. Таблиця 1.2 містить дані про взуття. Стовпчик footwear містить тип взуття, а стовпчик size містить масив розмірів наявного на складі

взуття. Таблиця 1.3 містить дані про забруднювачі повітря в містах. Вимірювальні станції мають різні набори датчиків для вимірювання забруднення. Стовпчик polluttion_measurements містить список забруднювачів, які вимірює дана станція, та їх величини. Дані зберігаються у JSON форматі. Звичайно всі ці дані можна записати у таблиці тільки з атомарними типами даних, але в такому випадку частина інформації буде дублюватися, або треба буде записувати дані у кілька зв'язаних таблиць, що ускладнить доступ до інформації.

Табл. 1.1 Приклад таблиці, яка складається тільки з атомарних даних

id	first_name	last_name	birthday
1	Andres	Jones	1950-01-05
2	Bianka	Brown	2001-04-18

Табл. 1.2 Приклад таблиці, яка містить масиви

id	footwear	size	price
100	shoe	[32, 34, 36]	150
200	sneakers	[34, 36, 38, 40, 41]	130

Табл. 1.3 Приклад таблиці, яка містить JSON дата

id	location	polluttion_measurements
22	London	{'CO': 10, 'N2O':11,

		‘PM’: 100}
564	Ottawa	{‘CO’: 11, ‘SO2’:12}

В базах даних крім таблиць використовуються і інші об’єкти, такі як індекси, послідовності, обмеження цілісності і т.д. Вони грають допоміжну роль і далі розглядатися не будуть.

Таблиці грають подвійну роль в реляційних базах даних. З одного боку вони використовуються для збереження об’єктів, які моделюють реальні або абстрактні об’єкти нашого світу. А з іншого боку вони використовуються для збереження зв’язків між об’єктами. Тобто в реляційної моделі даних об’єкти і зв’язки між ними представляються однаковим чином. Це значно полегшує маніпулювання зв’язками, яке робиться таким самим чином, як і маніпулювання зв’язками.

1.2 Опис бази даних інвойсів

В якості прикладу розглянемо базу даних інвойсів, яка буде використовуватися для пояснення семантики запитів SQL.

Вона складається з наступних таблиць:

- vendors – таблиця постачальників;
- vendor_contacts – таблиця контактів постачальників;
- invoices – таблиця рахунків-фактур
- invoice_line_items – таблиця окремих позицій рахунків-фактур;
- terms – таблиця термінів платежів;
- general_ledger_accounts – таблиця бухгалтерських рахунків;

Для створення таблиць використовується спеціальний оператор CREATE TABLE, який належить DDL.

Таблиця постачальників створюється наступним оператором:

```
create table vendors (
    vendor_id numeric not null primary key,
    vendor_name varchar(50) not null unique,
    vendor_address1 varchar(50),
    vendor_address2 varchar(50),
    vendor_city varchar(50) not null,
    vendor_state char(2) not null,
    vendor_zip_code varchar(20) not null,
    vendor_phone varchar(50),
    vendor_contact_last_name varchar(50),
    vendor_contact_first_name varchar(50),
    default_terms_id numeric not null references terms,
    default_account_number numeric not null references general_ledger_accounts
);
```

Стовпці містять наступну інформацію

vendor_id – унікальний ідентифікатор постачальника. Для того щоб відрізнити одного постачальника від іншого до таблиці додається спеціальний стовпчик, який містить штучні номера постачальників, які будемо називати ідентифікаторами постачальників. Обмеження primary key гарантує, що

ідентифікатори мають унікальні значення і не можуть бути невизначеним, тобто приймати значення NULL. На перший погляд цей стовпчик є зайвим, так як наступний стовпчик під назвою vendor_name теж містить унікальні значення. Але насправді це не так. Як говорилося вище, зв'язки в таблиці представляються таким же самим чином як і інша інформація. Тобто vendor_name повинен був би використовуватися одночасно і як назва компанії, і для посилання на рядок постачальника з інших таблиць. Такого подвійного використання слід уникати. Наприклад, якщо знадобиться перейменувати компанію, то теж саме треба буде зробити для всіх посилань на цю компанію з інших таблицю. Тому додається окремий стовпчик, єдине призначення якого – містить унікальні ідентифікатори рядків.

vendor_name – назва компанії або організації постачальника. Обмеження unique гарантує, що імена компаній будуть унікальними, а обмеження NOT NULL говорить, що назва компанії не може бути невизначеною.

vendor_address1, vendor_address2 – адреси компанії постачальника. Може бути кілька адресів, але не більше двох. Якщо у постачальника відсутня друга адреса, то замість неї записується невизначене значення. Перша адреса теж може бути невизначеною. Це будемо інтерпретувати як адреса компанії невідома. Звернем увагу на те що існує кілька видів невизначеності. А саме значення невідоме або значення відсутнє. Насправді видів невизначеності набагато більше. В SQL вони не розрізняються. Тобто, якщо друга адреса відсутня, то це може трактуватися і як значення є але воно невідоме.

vendor_city – місто, в якому знаходиться постачальник. Воно не може бути невизначеним.

vendor_state – штат, в якому знаходиться постачальник. Не може бути невизначеним

vendor_zip_code – поштовий код постачальника. Він не може бути невизначеним.

vendor_phone – телефон постачальника. Може бути невизначеним. Знову виникає питання, як трактувати невизначене значення.

vendor_contact_last_name – прізвище постачальника. Може бути невизначеним

vendor_contact_first_name – ім’я постачальника. Може бути невизначеним.

default_terms_id – термін постачання, який встановлюється за умовчанням. Це значення не може бути невизначеним. Терміни постачання є стандартними значеннями, які описані в таблиці terms. В стовпці знаходиться посилання на рядок таблиці terms у вигляді ідентифікатора відповідного рядка. Обмеження references terms гарантує, що всі ідентифікатори мають відповідні значення в таблиці terms.

default_account_number – посилання на банківський рахунок за умовчанням. Це значення не може бути невизначеним. Банківські рахунки знаходяться в таблиці general_ledger_accounts. Обмеження references general_ledger_accounts гарантує, що посилання буде містити коректні значення.

Таблиця інвойсів створюється наступним оператором:

```
create table invoices
(
    invoice_id numeric not null primary key,
    vendor_id numeric not null references vendors,
    invoice_number varchar(50) not null,
    invoice_date date not null,
    invoice_total numeric(9,2) not null,
```

```
    payment_total numeric(9,2) default 0,
    credit_total numeric(9,2) default 0,
    terms_id numeric not null references terms,
    invoice_due_date date not null,
    payment_date date
);
```

invoice_id – містить унікальний ідентифікатор рядка, який не може бути невизначеним

vendor_id – містить ідентифікатор постачальника з таблиці vendors

invoice_number – містить номер інвойсу. Кожен інвойс має унікальний номер, який складається з цифр, букв та символів – і / Не може бути невизначеним.

invoice_date – дата створення інвойсу. Не може бути невизначеною

invoice_total – сума інвойсу

payment_total – заплачена сума

credit_total – сума, яка взята в кредит. Після проведення платежу повинна виконуватися рівність credit_total+ payment_total= invoice_total

terms_id – посилання на термін постачання товару. Терміни постачання товару знаходяться в таблиці terms.

invoice_due_date – дата, до якої треба виконати платіж

payment_date – дата, коли було зроблено платіж. Якщо значення невизначене, то це трактується як те що платіж ще не був зроблений

В таблиці invoice_line_items знаходиться список товарів, який постачається по вказаному інвойсу. Таблиця створюється наступним оператором

```
create table invoice_line_items
(
    invoice_id numeric not null references invoices,
    invoice_sequence numeric not null,
    account_number numeric not null references general_ledger_accounts,
    line_item_amt numeric(9,2) not null,
    line_item_description varchar(100) not null,
    primary key (invoice_id, invoice_sequence)
);
```

invoice_id – ідентифікатор інвойсу в таблиці invoices. З кожним інвойсом може бути зв’язано кілька рядків з таблиці invoice_line_items

invoice_sequence – порядковий номер товару в інвойсі. Нумерація починається з одиниці

account_number – посилання на банківський рахунок в таблиці general_ledger_accounts

line_item_amt – сума по даному товару. Сума по усім товарам, які відносяться до даного інвойсу повинна дорівнювати значенню в стовпці invoice_total.

line_item_description – назва товару

Так як для кожного інвойсу може існувати кілька рядків в таблиці invoice_line_items то ідентифікатор не може використовуватися в якості

ідентифікаторів товарів. Тому використовується складений ідентифікатор, а саме invoice_id та invoice_sequence.

Таблиця бухгалтерських рахунків general_ledger_accounts створюється наступним оператором:

```
create table general_ledger_accounts
(
    account_number numeric not null primary key,
    account_description varchar(50) not null unique
);
```

account_number – номер рахунку. Він же використовується як унікальний ідентифікатор рахунку

account_description – змістовний опис рахунку

Таблиця термінів поставки товарів terms створюється наступним оператором:

```
 create table terms
(
    terms_id numeric not null primary key,
    terms_description varchar(50) not null,
    terms_due_days numeric not null
);
```

terms_id – ідентифікатор рядку

terms_description – опис терміну

terms_due_days – кількість днів, протягом яких товар треба поставити замовнику

Таблиця замовників vendor_contacts створюється наступним оператором:

```
create table vendor_contacts
(
    vendor_id numeric not null,
    last_name varchar(50) not null,
    first_name varchar(50) not null
);
```

vendor_id – ідентифікатор постачальника який працює з замовником

last_name – прізвище замовника

first_name – ім’я замовника

1.3 Базовий синтаксис оператору запитів SELECT

Базова форма оператору запитів SELECT, складається з звернення до однієї таблиці, фільтрації її рядків, проекції по стовпцям, перейменуванні стовпців, додаванні нових, обчислювальних стовпців та упорядкуванні результуючих рядків. Синтаксично оператор виглядає наступним чином:

```
SELECT <select list>
FROM <table>
[WHERE <search condition>]
[ORDER BY <order by list>]
```

Тут

<select list> – містить стовпці вихідної таблиці, вирази для обчислювальних стовпців та нові імена стовпців

<table> – задає ім’я вихідної таблиці, можливо з аліасом, який задає локальне ім’я таблиці. Аліас дійсний тільки в межах оператора SELECT

<search condition> – задає умову фільтрації рядків вихідної таблиці. Якщо умова відсутня, то всі рядки попадають в результат. Умова складається з однієї або більше операцій порівняння з’єднаних логічними операціями кон’юнкції та диз’юнкції. Зауважимо, що використовується трьохзначна логіка Кліні та Пріста. Якщо значенням умови на рядку буде True то рядок потрапляє в результуючу таблицю. В усіх інших випадках він не потрапляє в результуючу таблицю.

<order by list> – задає умову сортування рядків результуючої таблиці. Якщо умова не задана, то рядки виводяться в довільному порядку.

Розглянемо кілька прикладів.

Запит, який повертає всі рядки з таблиці інвойсів

SELECT * FROM invoices;

Частковий результат наведений в таблиці 1.1. Повний результат включає всі рядки таблиці повернені в довільному порядку.

Таблиця 1.1

invoice_id	vendor_id	invoice_number	invoice_date	invoice_total	payment_total	credit_total	terms_id	invoice_due_date	payment_date
1	34	QP58872	2014-02-25	116.54	116.54	0.00	4	2014-04-22	2014-04-11
2	34	Q545443	2014-03-14	1083.58	1083.58	0.00	4	2014-05-23	2014-05-14
3	110	P-0608	2014-04-11	20551.18	0.00	1200.00	5	2014-06-30	NULL
4	110	P-0259	2014-04-16	26881.40	26881.40	0.00	3	2014-05-16	2014-05-12
5	81	MABO148	2014-04-	936.93	936.93	0.00	3	2014-05-16	2014-05-

		9	16						13
……	……..	……	…….	…….	……	…….	…….	……..	……

Запит, який повертає три стовпця всіх рядків таблиці інвойсів:

SELECT invoice_number, invoice_date, invoice_total

FROM invoices

ORDER BY invoice_total;

Перші 5 рядків результату наведені в таблиці 1.2

Таблиця 1.2

invoice_number	invoice_date	invoice_total
24780512	2014-05-29	6.00
25022117	2014-05-24	6.00
24863706	2014-05-27	6.00
21-4748363	2014-05-09	9.95
21-4923721	2014-05-09	9.95
……	……	……

Запит, який повертає інвойс заданий своїм ідентифікатором

SELECT *

FROM invoices

WHERE invoice_id=17

Результат зображено в таблиці 1.3

Таблиця 1.3

invoice_id	vendor_id	invoice_number	invoice_date	invoice_total	payment_total	credit_total	terms_id	invoice_due_date	payment_date
17	90	97-1024A	2014-04-26	356.48	356.48	0.00	3	2014-06-09	2014-06-09

Стовпці результуючої таблиці задаються наступним чином

SELECT [ALL|DISTINCT] <column specification> [[AS] <result column>]

[, <column specification> [[AS] <result column>]]…

Тут <column specification> може бути

- ім’ям стовпця вихідної таблиці
- виразом на строкових типах даних, тобто виразом, який повертає строку
- арифметичним виразом
- скалярною функцією

В будь якому випадку, в якості результату повинно повертатися значення, яке може бути розташоване в однієї комірці таблиці.

Якщо треба повернути всі стовпці таблиці, то можна використовувати зірочку, замість перерахування всіх імен, наприклад SELECT * FROM invoices.

Наступний запит повертає для кожного інвойсу різницю між сумою інвойсу та зробленими платіжами, для інвойсів, по яким не повністю перераховані гроші:

SELECT invoice_number,

invoice_total - invoices.payment_total - invoices.credit_total AS balance

FROM invoices

WHERE invoice_total - invoices.payment_total - invoices.credit_total != 0

Результат приведено в таблиці 1.4

Таблиця 1.4

invoice_number	**balance**

P-0608	19351.18
989319-497	2312.2
989319-487	1927.54
97/553B	313.55
…….	…….

Наступний запит використовує вираз на строках, для конкатенації імені та призвища продавців кожного постачальника:

SELECT vendor_name,

vendor_contact_first_name || ' ' || vendor_contact_last_name AS vendor_contact

FROM vendors

Результат зображено в таблиці 1.5

Таблиця 1.5

vendor_name	**vendor_contact**
US Postal Service	Francesco Alberto
National Information Data Ctr	Ania Irvin
Register of Copyrights	Lukas Liana
Jobtrak	Kenzie Quinn
Newbrige Book Clubs	Michelle Marks
…….	……..

В таблиці рядки можуть співпадати. Для того, щоб співпадаючи рядки включалися в результат тільки один раз використовується ключове слово DISTINCT. Якщо треба повернути всі рядки, то використовується ключове слово ALL. За умовчанням вважається, що використовується ALL, тому його можна не вказувати.

Фраза WHERE використовується для фільтрації рядків таблиці. Предикат фільтрації задається як сукупність операцій порівняння , які з’єднані логічними функціями. В якості операцій порівняння використовуються стандартний набір

операцій =, <, >, <=, >=, <> та спеціальні операції LIKE, яка порівнює строку з шаблоном, та BETWEEN, яка перевіряє чи попадає значення в заданий діапазон.

Розглянемо кілька прикладів.

Постачальники, розташовані в штаті Каліфорнія вибираються наступною умовою: WHERE vendor_state = 'CA'

Непроплачені інвойси вибираються за допомогою наступного запиту WHERE invoice_total > payment_total + credit_total, або WHERE invoice_total - payment_total - credit_total <> 0.

Постачальники, чиї назви починаються на букви з A по L: WHERE vendor_name < 'M'

Інвойси, які були проплачені до 31 травня: WHERE payment_date <= '31-May-2014'

Прості вирази з порівнянням з'єднуються в більш складні вирази булевськими операціями кон'юнкції, диз'юнкції та заперечення, які позначаються AND, OR та NOT відповідно. Розглянемо синтаксис складених предикатів.

WHERE [NOT] <search condition 1> {AND|OR} [NOT] < search condition 2>…

Операція NOT має найвищий пріоритет, далі виконується операція AND, останньою іде операція OR. Порядок виконання операцій можна змінити за допомогою круглих дужок. Розглянемо кілька прикладів.

Знайти постачальників, які проживають в штаті New Jersey в місті Springfield: WHERE vendor_state='NJ' AND vendor_city='Springfield' Тут складна умова буде істинною коли виконуються одночасно дві прості умови

Знайти постачальників, які знаходяться в штаті New Jersey або в місті Pittsburg: WHERE vendor_state=’NJ’ OR vendor_city=’ Pittsburg’. Тут складна умова буде істиною, коли виконується хоча б одна з простих умов

Знайти постачальників, які не знаходяться в штаті New Jersey: WHERE NOT vendor_state=’NJ’ або WHERE vendor_state<>’NJ’

Розглянемо наступний запит:

```
SELECT  invoice_number, invoice_date, invoice_total
FROM invoices
WHERE invoice_date > '01-MAY-2014' OR invoice_total > 5000
    AND invoice_total - payment_total - credit_total > 0
ORDER BY invoice_number
```

Якщо розставити дужки згідно з пріоритетами булевських операцій, то умова фільтрації виглядатиме наступним чином

```
WHERE invoice_date > '01-MAY-2014' OR (invoice_total > 5000
    AND invoice_total - payment_total - credit_total > 0)
```

Результат зображено в таблиці 1.6

Таблиця 1.6

invoice_number	invoice_date	invoice_total
0-2058	2014-05-08	37966.19
0-2060	2014-05-08	23517.58
0-2436	2014-05-07	10976.06
1-200-5164	2014-05-07	63.40
1-202-2978	2014-05-06	33.00
…….	……..	……

Крім операцій порівняння існує ще кілька операцій, специфічних саме для SQL. Розглянемо найбільш важливі з них.

Операція IN перевіряє належність елемента сукупності значень. Вона має наступний синтаксис:

WHERE <test expression> [NOT] IN (<expression 1> [, <expression 2>…])

Наприклад, наступна операція перевіряє чи належить terms_id множені (1,2,3): terms_id IN (1,2,3). Такий предикат може бути записаний у еквівалентному вигляді через OR: terms_id=1 OR terms_id=2 OR terms_id=3. Предикат vendor_state NOT IN ('CA', 'NJ', 'OR') буде істинним, якщо vendor_state не знаходиться в заданій множені штатів. Він може бути записаний у еквівалентному вигляді через булевську операцію AND, а саме vendor_state <>'CA' AND vendor_state <> 'NJ' AND vendor_state <> 'OR'.

Операція BETWEEN перевіряє, чи знаходиться значення в заданому діапазоні. Вона має наступний синтаксис:

WHERE <test expression> [NOT] BETWEEN <begin expression> AND <end expression>. Розглянемо кілька прикладів.

Знайти всі інвойси, які були створені з 1 травня по 31 травня 2014 року:

SELECT *

FROM invoices

WHERE invoice_date BETWEEN '1-may-2014' AND '31-may-2014'

Результат показано в таблиці 1.7

Таблиця 1.7

invoice_id	vendor_id	invoice_number	invoice_date	invoice_total	payment_total	credit_total	terms_id	invoice_due_date	payment_date
23	121	97/465	2014-05-01	565.15	565.15	0.00	1	2014-05-14	2014-05-05

24	121	97/222	2014-05-01	1000.46	1000.46	0.00	3	2014-06-03	2014-05-25
25	123	4-342-8069	2014-05-01	10.00	10.00	0.00	4	2014-06-10	2014-05-27
26	123	4-327-7357	2014-05-01	162.75	162.75	0.00	3	2014-05-27	2014-05-21
27	123	4-321-2596	2014-05-01	10.00	10.00	0.00	2	2014-05-20	2014-05-11
……	……	……..	……..	……..	……	……..	……..	……	……

Запит може бути переписаний в еквівалентній формі з використанням булевської функції AND:

SELECT *

FROM invoices

WHERE invoice_date >= '1-may-2014' AND invoice_date <= '31-may-2014'

В наступному прикладі треба знайти всіх постачальників, поштовий індекс яких менший ніж 93600 або більший ніж 93799:

SELECT *

FROM vendors

WHERE vendor_zip_code BETWEEN '93600' AND '93799'

Результат показано в таблиці 1.8

Таблиця 1.8

vendor_id	**vendor_name**	**vendor_address1**	**vendor_address2**	**vendor_city**	**vendor_state**	**vendor_zip_code**	**vendor_phone**	**vendor_contact_last_name**	**vendor_contact_first_name**	**default_terms_id**	**default_account_number**
8	BFI Industries	PO Box 9369	NULL	Fresno	CA	93792	(559) 555-1551	Kaleigh	Erick	3	521
10	Robbins Mobile Lock	4669 N Fresno	NULL	Fresno	CA	93726	(559) 555-9375	Leigh	Bill	2	523

	And Key										
11	Bill Marvin Electric Inc	4583 E Home	NULL	Fresno	CA	93703	(559) 555-5106	Hostlery	Kaitlin	2	523
12	City Of Fresno	PO Box 2069	NULL	Fresno	CA	93718	(559) 555-9999	Mayte	Kendall	3	574
14	Expedata Inc	4420 N. First Street, Suite 108	NULL	Fresno	CA	93726	(559) 555-9586	Quintin	Marvin	3	589
….	…….	…….	…….	……	…….	…….	……..	……..	……..	……..	……..

Запит можна переписати в еквівалентному вигляді:

SELECT *

FROM vendors

WHERE vendor_zip_code <= '93600' OR vendor_zip_code >= '93799'

Для порівнянна строки з шаблоном використовується спеціальна операція LIKE, яка має наступний синтаксис:

WHERE <match expression> [NOT] LIKE <pattern>

де <match expression> – будь який символьний рядок, який порівнюється шаблоном <pattern>. Шаблон складається з звичайних символів, та метасимволів, до яких відносяться % та _. Символ % означає, що замість нього може стояти від нуля і більше будь яких символів. Символ _ означає, що замість нього може стояти один будь який символ. Наприклад шаблон ‘_арт’ відповідає символьним рядкам ‘жарт’, ‘гарт’, ‘*арт’ та іншим.

1.4. Сортування рядків

Рядки таблиць в реляційних базах даних не впорядковані. Але часто при виводі рядків виникає потреба виводити їх в певному порядку. Для цього використовується оператор ORDER BY. Він має наступний синтаксис:

ORDER BY <expression> [ASC|DESC] [, <expression> [ASC|DESC]…]

Кожний вираз <expression> повертає скалярне значення, розраховане на рядку, потім рядки впорядковуються згідно цьому значенню. Якщо вказане ключове слово ASC, то рядки впорядковуються в зростаючому порядку, якщо ж вказане ключове слово DESC, то рядки впорядковуються в убиваючому порядку. Якщо нічого не вказано, то рядки впорядковуються в зростаючому порядку, тобто ASC можна не вказувати. Якщо ж два рядки мають співпадаючи значення виразу <expression> то тоді порівнюються значення другого виразу <expression> і так далі. Може бути ситуація, коли значення всіх виразів для двох рядків однакові. Тоді порядок розміщення рядків невизначений, тобто вони можуть записуватися в довільному порядку. Таким чином, фраза ORDER BY не гарантує, що рядки будуть розміщені строго в одному порядку. Такі функції, результат яких є неоднозначним, називаються недетермінованими функціями [86].

Розглянемо кілька прикладів.

Вивести назви всіх постачальників за алфавітним порядком:

```
SELECT vendor_name
FROM vendors
ORDER BY vendor_name;
```

Так як назви постачальників є унікальними, то сортування буде повертати однозначний результат.

Вивести назви постачальників та їхні адреси з точністю до вулиці, та впорядкувати по штатам та містам:

```
SELECT vendor_name,
  vendor_city,
  vendor_state,
  vendor_zip_code
FROM vendors
ORDER BY vendor_state, vendor_city
```

Результат наведено в таблиці 1.9

Таблиця 1.9

vendor_name	vendor_city	vendor_state	vendor_zip_code
AT&T	Phoenix	AZ	85062
Wells Fargo Bank	Phoenix	AZ	85038
Computer Library	Phoenix	AZ	85023
Aztek Label	Anaheim	CA	92807
Blue Shield of California	Anaheim	CA	92850
…….	……..	……	……..

1.5 Операції зовнішнього та внутрішнього з’єднання

В запиті можна використовувати кілька таблиць. Таблиці, над якими виконується запит розташовуються в фразі FROM. Загальний синтаксис має наступний вигляд:

FROM <table 1> [<join type> <table2> [<join condition>]…]

Результатом виконання оператора FROM буде одна проміжна віртуальна таблиця, над якою вже виконуються всі інші частини запиту. Є три типи операцій з’єднання. Це внутрішнє з’єднання, зовнішнє з’єднання та декартіве з’єднання. Найбільш простим є декартіве з'єднання, вигляду: <table 1> CROSS JOIN <table 2>. Кожний рядок першої таблиці з’єднується з кожним рядком

другої таблиці. Якщо перша таблиця має n рядків, а друга таблиця має m рядків, то результуюча таблиця буде мати n*m рядків.

В якості прикладу з’єднаємо дві таблиці A та B

Таблиця 1.10. Таблиця A

a1	a2
11	a21
12	a22
13	a23

Таблиця 1.11. Таблиця B

b1	b2
11	b21
12	b22

Виконаємо над ними декартова з’єднання

SELECT * FROM A CROSS JOIN B

Результат наведено в таблиці 1.12

Таблиця 1.12

a1	a2	b1	b2
11	a21	11	b21
11	a21	12	b22
12	a22	11	b21
12	a22	12	b22
13	a23	11	b21
13	a23	12	b22

Внутрішнє з’єднання записується у вигляді <table 1> [INNER] JOIN <table 2> ON <boolean expression>. Воно може бути переписане через еквівалентний запит з використанням декартового з’єднання та фрази WHERE:

FROM <table 1> CROSS JOIN <table 2> WHERE < boolean expression >

Наприклад, для таблиць A і B з’єднання за умовою a1=b1 буде записане у вигляді SELECT * FROM A INNER JOIN B ON a1=b1 і поверне наступну таблицю

Таблиця 1.13

a1	a2	b1	b2
11	a21	11	b21
12	a22	12	b22

Він може бути переписаний через декартовий здобуток:

SELECT * FROM A CROSS JOIN B WHERE a1=b1

Оператори зовнішнього з’єднання мають наступний синтаксис:

<table 1> {LEFT|RIGHT|FULL} [OUTER] JOIN <table 2> ON <binary condition>

Як видно всього існує три оператора зовнішнього з’єднання. Результат дії операторів зовнішнього з’єднання включає в себе результат відповідного оператора внутрішнього з’єднання і деякі додаткові рядки. Для оператора лівостороннього зовнішнього з’єднання LEFT OUTER JOIN додатково додаються всі рядки лівої таблиці, які не потрапили в результат відповідного оператора внутрішнього з’єднання. Для оператора правостороннього зовнішнього з’єднання RIGHT OUTER JOIN додатково додаються всі рядки правої таблиці, які не потрапили в результат відповідного оператора внутрішнього з’єднання. А для оператора повного зовнішнього з’єднання FULL OUTER JOIN додаються як рядки лівої таблиці, так і рядки правої таблиці, які не потрапили в результат відповідного оператора внутрішнього з’єднання. При додаванні стовпці, які відсутні в правої або лівої таблиці заповнюються невизначеними значеннями. Наприклад, запит

SELECT * FROM A LEFT OUTER JOIN B ON a1=b1

повертає таблицю 1.14

Таблиця 1.14

a1	a2	b1	b2
11	a21	11	b21
12	a22	12	b22
13	a23	NULL	NULL

А запит

SELECT * FROM A RIGHT OUTER JOIN B ON a1=b1

повертає таблицю 1.15

Таблиця 1.15

a1	a2	b1	b2
11	a21	11	b21
12	a22	12	b22

Так як всі рядки таблиці B потрапили в результат відповідного внутрішнього з’єднання, то ніяких нових рядків не додається.

З тієї ж самої причини запит SELECT * FROM A FULL OUTER JOIN B ON a1=b1 повертає такий же самий результат, що і лівосторонній запит.

Якщо в внутрішніх і зовнішніх запитах порівняння проводиться по стовпцях з однаковими іменами, то умову з’єднання можна записати в скороченій формі, використовуючи ключове слово USING <column list>. Тут <column list> містить список імен стовпців, розділених комами.

РОЗДІЛ 2. УТОЧНЕНА ТАБЛИЧНА АЛГЕБРА СЕМАНТИЧНИХ ФУНКЦІЙ ЗАПИТІВ ЯДРА МОВИ SQL

2.1. Математичні основи сучасних реляційних СУБД

Розглянемо деякі ключові аспекти математичних основ сучасних реляційних СУБД: мультимножини[], теоретико-множинні основи (конструкції повного образу, обмеження відношень, узагальненого декартова добутку, бінарне відношення сумісності, бінарне відношення конфінальності), некласичні логіки (сильна тризначна логіка Кліні), предпорядки і порядки [63].

Мультимножини. Добре відомо, що моделлю даних при реляційному підході до баз даних виступають, відповідно Кодду, реляції, що розуміються в логіко-математичному сенсі як підмножини декартового добутку відповідних доменів. Не входячи в дискусію щодо використання імен атрибутів (або просто атрибутів, подробиці див., наприклад в [114]), підкреслимо той факт, що в (канонічній) реляції її складові (кортежі), не можуть повторюватися. Разом з тим на практиці в SQL-подібних мовах таблиці, що є результатом інтерпретації оператора запитів SELECT, можуть містити повторювані рядки (дублікати, екземпляри рядків): кажучи точніше, наявність ключових слів DISTINCT, ALL управляє відсутністю або, відповідно, наявністю повторюваних рядків в результатах. Математичною моделлю сукупностей з повтореннями виступають мультимножини (див., наприклад, огляд [43]).

Таким чином, при даному підході моделлю даних повинні виступати не множини відповідних кортежів, а мультимножини таких кортежів. Основні результати по використанню мультимножин в реляційних базах даних можна знайти в огляді [114] і [43].

Теоретико-множинні основи. Одна з цілей моделювання даних полягає в уточненні маніпулювання ними. При реляційному підході маніпулювання

уточнюється розглядом класичних реляційних алгебр Кодда [12, 13] і табличних алгебр – їх сучасного аналога [63]. Теорія табличних алгебр носить більш-менш завершений характер, основні результати представлені в монографії [63]. Методика дослідження при цьому наступна. Спочатку встановлюються природні представлення сигнатурних операцій табличних алгебр в термінах загальнозначущих теоретико-множинних конструкцій: повного образу множини щодо відношення, обмеження відношення по множині, узагальненого декартова добутку, бінарного відношення сумісності бінарних відношень, бінарного відношення конфінальності множин. Потім досліджуються зазначені теоретико-множинні конструкції. Нарешті, властивості конструкцій, зважаючи на наявність представлень, природно переносяться на табличні алгебри [72]. Можна не сумніватися, що розроблений теоретико-множинний апарат має і інші успішні застосування.

Некласичні логіки. Використання спеціального значення NULL в SQL-подібних мовах і необхідність розширення предикатів на це спеціальне значення призвело до появи третього логічного значення UNKNOWN; в свою чергу використання пропозиційних зв'язок призвело до введення відповідної тризначною логіки (див., наприклад, [43, 13]). Виявилося, що при цьому була перевідкриття так звана сильна тризначна логіка Кліні, введена ним в теорії рекурсії [73]. Відзначимо три обставини. По-перше, для обґрунтування компактного завдання зазначеної логіки за допомогою трьохелементної ланцюга [8] треба використовувати класичний результат про зв'язок між гратами і абстрактними гратами (див., наприклад, [117]). По-друге, з огляду на використання в SQL-подібних мовах конструкцій ALL, ANY, які є власне кажучи кванторами загальності та існування відповідно, необхідно уточнити квантифікацію в сильній тризначній логікі Кліні [61, 83]. По-третє, сама сильна тризначна логіка Кліні виникає з класичної булевої логіки за допомогою конструкції поширення бінарної функції з елементів на множини елементів за допомогою повного образу [72, 61]. Нарешті відзначимо, що Кодд пропонує розглядати і спеціальну чотиризначну логіку [13].

Предпорядки і порядки. Необхідність розгляду зв'язку між предпорядками і порядками виникає при уточненні семантики фрази ORDER BY оператора SELECT в SQL-подібних мовах. Справа в тому, що бінарне відношення між рядками результуючої таблиці, індуковане зазначеної фразою, взагалі кажучи, є предпорядком (рефлексивним, транзитивним, але не антисиметричним відношенням), але не порядком [63]. Тут важливо, що передпорядок на множені індукує порядок на відповіднім фактор-множині (див., наприклад, [117]).

2.2. Визначення основних структур даних та операцій на них

Хоча реляційна алгебра Кодда [12] і лежить в основі реляційних та об'єктно-реляційних СУБД та мови запитів SQL, існує велика відстань між реляційною алгеброю та операторами SQL. Проблема полягає в тому, що немає прямого відображення між операторами SQL і операціями реляційної алгебри. Відносно легко по виразу реляційної алгебри написати еквівалентний вираз в мові SQL, але для запиту SQL побудувати еквівалентний вираз в реляційної алгебри важко, а часто навіть не можливо.

Пізніше з'явилися розширені версії реляційної алгебри, більш наближені до SQL. Це виразилося в тому, що до стовпців були додані імена, а сама алгебра була розширена новими операціями, такими як групування, агрегатні функції та обчислювальні стовбці, див. [17, 4, 23]. Тим не менше, навіть сучасна версія реляційної алгебри є дуже обмеженою для задання семантики SQL.

В розділі буде визначена спеціальна алгебра семантичних функцій мови SQL, названа табличною алгеброю, яка базується на інших принципах побудови запитів, ніж традиційна алгебра Кодда та її варіанти. А саме, запити будуть будуватися з базових функцій за допомогою спеціальних функцій вищого порядку, які називаються операторами, за аналогією з іншими мовами програмування, такими як Java, C++ і так далі [106]. Але якщо в традиційних мовах програмування в якості операторів використовуються оператори циклування, умовні оператори, та

оператор послідовного виконання, то в табличній алгебрі введені спеціальні оператори, призначені для роботи з множинами. До таких операторів відносяться фільтрація за предикатом, побудова повного образу, оператори з'єднання та суперпозиція за іменами. Перший варіант табличної алгебри був описаний в [63]. В розділі приводиться оновлена версія, в якій надані нові, більш зручні, визначення операторів, які більш точно задають семантику запитів SQL. В першу чергу це стосується операторів зовнішнього та внутрішнього з'єднання та теоретико-множинних операцій. Перейдемо до базових визначень.

Нехай задані множина імен атрибутів Atr та універсальний домен Dom, який містить множину атомарних даних. Накладемо вимоги, що ці множини є непустими і в множині атомарних даних існує спеціальний елемент $null$, який трактується як невизначене значення. *Схемою* будемо називати будь яку скінченну підмножину імен атрибутів $S \subseteq Atr$. Множину всіх схем будемо позначати через Sch

Рядком, який задовольняє схемі S , або просто рядком схеми S будемо називати скінченну множину пар $\{(a_i, d_i) | a_i \in S,\ d_i \in Dom, i = 1..n\}$ таку, що $a_i \neq a_j$, якщо $i \neq j$, де $i, j = 1..n$. Такі пари назвемо атрибутами, перший компонент пари є ім'я атрибуту, а другий – значення атрибуту. Іноді, коли це не викликає неоднозначності, під атрибутом буде розумітися тільки його ім'я. Позначимо операцію проекції рядку r по його першій компоненті через $pr_1(r)$, а схему рядку r будемо записувати як $S(r)$. Функція проекції по першій компоненті є всюдивизначеною, тому для довільного рядка завжди можна отримати його схему. Рядки також визначаються як скінченні відображення з множини імен атрибутів в множину атомарних значень $Atr \to Dom$.

Під *таблицею* схеми S будемо розуміти пару (t, S), де t є скінченою множиною рядків схеми S, яка називається *станом таблиці або просто станом*. Відмітимо, що так як схема таблиці співпадає зі схемами рядків то, в більшості випадків, схема таблиці може бути розрахована з її стану. Єдиним виключенням є випадок, коли стан таблиці є пустою множиною. В попередньої версії табличної алгебри, таблицям не приписувалася її схема. Це приводило до того, що в деяких запитах схему результуючої таблиці не можна було отримати у випадку обчислення запиту на пустих таблицях, що відрізняється від семантики SQL, де схема результату може бути завжди обчислена.

Множину всіх рядків (таблиць) схеми S будемо позначати як $Rec(S)$ (відповідно $Tbl(S)$), а множину всіх рядків (таблиць) – *Rec* (відповідно *Tbl*). Так $Rec(S) \stackrel{\text{def}}{=} \{r | pr_1(r) = S\}$, $Tbl(S) \stackrel{\text{def}}{=} \{(t, S) | t \in 2^{Rec(S)}\}$, $Rec \stackrel{\text{def}}{=} \bigcup_{S \subseteq Atr} Rec(S)$, $Tbl \stackrel{\text{def}}{=} \bigcup_{S \subseteq Atr} Tbl(S)$.

Запис вигляду $r(a)$, $r \in Rec, a \in Atr$ означає значення атрибуту з іменем a на рядку r. Домовимося схему рядку r позначати $S(r)$

Схема таблиці може бути пустою $\emptyset$. Для такої схеми існує рівно один рядок який позначимо як ε, і два стана – $\{\varepsilon\}$ та $\emptyset$. Домовимося позначати рядки через $r, r_1, r_2, \ldots$, таблиці – $T, T_1, T_2, \ldots$, стани таблиць через $t, t_1, t_2, \ldots$, а схеми через $S, S_1, S_2, \ldots$

Бінарне відношення *сумісності рядків* $r_1 \approx r_2$ визначається за формулою $\forall a(a \in S(r_1) \& a \in S(r_2) \Longrightarrow r_1(a) = r_2(a))$. Іншими словами два рядки є сумісними якщо значення атрибутів з співпадаючими іменами теж співпадають.

Рядки, схеми яких не перетинаються завжди є сумісними. Бінарна операція об'єднання сумісних рядків $\bar{\cup}: Rec \times Rec \longrightarrow Rec$ визначається за формулою $r_1 \bar{\cup} r_2 = \{(a,d)|(a,d) \in r_1 \vee (a,d) \in r_2\}$. Її областю визначення буде множина $dom \bar{\cup} \stackrel{def}{=} \{(r_1, r_2)|r_1, r_2 \in Rec \,\&\, r_1 \approx r_2\}$

Разом з простими іменами атрибутів будуть використовуватися складені імена вигляду $a_1.a_2 \ldots a_n$, $a_i \in Atr, i = 1..n$. Множину всіх складених атрибутів позначимо Atr^+. Такі імена будуть використовуватися для того, щоб не порушувати вимогу унікальності імен атрибутів при об'єднані рядків. Відмітимо, що $Atr \subseteq Atr^+$. Далі визначимо спеціальну функцію $a \Rightarrow: Rec \longrightarrow Rec$ параметризовану по імені атрибута a. Вона визначається за формулою $a \Rightarrow (r) = \{(a.a_i, d_i)|(a_i, d_i) \in r\}$. Таку функцію назвемо *функцією префіксації*, а ім'я атрибуту a будемо називати *префіксом*.

Протилежна функція, яка вилучає першу компоненту складених атрибутів, називається функцією *депрефіксації* і визначається за формулою $\Leftarrow (r) = \{(b,d)|(a.b,d) \in r \,\&\, a \in Atr \,\&\, b \in Atr^+\}$. Вона визначена на множені рядків, імена атрибутів яких складаються, принаймні з двох компонент, причому, якщо імена деяких атрибутів співпадають з точністю до перших компонент, то їх значення повинні теж співпадати. В протилежному випадку функція буде невизначеною. Тобто депрефіксація є частковою функцією на відміну від префіксації. Має місце наступна рівність: $r = \Leftarrow (a \Rightarrow (r))$

Операція *об'єднання рядків з префіксацією* $\cup_{a_1,a_2}: Rec \times Rec \longrightarrow Rec$, $a_1 \neq a_2$ є операцією параметризованою по іменам префіксів a_1 і a_2. Вона визначається наступним чином: $r_1 \cup_{a_1,a_2} r_2 = \{r_1' \cup r_2' | r_1' = (a_1 \Rightarrow r_1) \& r_2' = (a_2 \Rightarrow r_2)\}$

Позначимо через e пусте ім'я. Тоді складене ім'я $e.a$ будемо вважати співпадаючим з іменем a. Це означає, що, наприклад, $r_1 \cup_{e,a_2} r_2 = \{r_1' \cup r_2' \,|\, r_1' = r_1 \& r_2' = (a_2 \Rightarrow r_2)\} = \{r_1 \cup r_2' \,|\, r_2' = (a_2 \Rightarrow r_2)\}$.

Якщо в якості області визначеності розглядати множини рядків, імена атрибутів яких є простими, то операція буде всюди визначеною.

Аналогічним чином визначимо префіксацію для схем $S_1 \cup_{a_1,a_2} S_2 = a_1 \Rightarrow (S_1) \cup a_2 \Rightarrow (S_2)$, де $a \Rightarrow (S) \stackrel{\text{def}}{=} \{a.a_i | a_i \in S\}$

Будемо говорити, що рядок належить таблиці, якщо він належить стану цієї таблиці: $r \in T \stackrel{\text{def}}{=} \exists t \exists S (T = (t, S) \wedge r \in t)$.

2.3 Теоретико-множинні операції на таблицях

В підрозділі задані нові версії теоретико-множинних операцій на реляціях. На відміну від попередніх визначень, відповідність стовбців двох таблиць визначається за позицією, а не за ім'ям, таким самим чином, як це робиться в SQL.

Будемо говорити, що таблиці T_1 і T_2 є *однотипними* якщо потужності їхніх схем співпадають. Позначимо таке відношення через $T_1 \cong T_2$. Відношення *однотипності* рядків визначається аналогічним чином: $r_1 \cong r_2 \stackrel{\text{def}}{=} |S(r_1)| = |S(r_2)|$

Позначимо через $REN_B^A(t)$ унарну функцію перейменування атрибутів. Тут A і B – послідовності імен атрибутів однакової довжини. Функція заміняє в рядку r всі імена атрибутів з послідовності A на відповідні імена з послідовності B. Функція є частковою, так як після перейменування деякі атрибути можуть отримати однакові імена. Для таблиць ця функція буде

перейменовувати кожний рядок, тобто $REN_B^A((t,S)) = (t',S')$, де $t' = \{r'|r' = REN_B^A(r)\&r \in t\}, S' = (S\backslash A) \cup B$

Позначимо через T_1 таблицю (t_1,S_1), а через T_2 іншу таблицю (t_2,S_2), однотипну першій, тобто $|S_1| = |S_2|$. Зафіксуємо послідовності, які складаються з імен атрибутів схеми S_1 та S_2, та позначимо їх через $< S_1 >$ та $< S_2 >$ відповідно.

Тоді теоретико-множинні операції задаються наступним чином:

Об'єднання таблиць $T_1 \Cup_{S_2}^{S_1} T_2 = (t_1 \cup \text{REN}_{S_2}^{S_1}(t_2), S_1)$. Областю визначеня операції $\Cup_{S_2}^{S_1}$ буде множина пар однотипних таблиць, схема першої з яких співпадає з S_1, а схема другої співпадає з S_2: $\text{dom}\ T_1 \Cup_{S_2}^{S_1} T_2 = \{(T_1,T_2)|T_1 \cong T_2\ \&\ S(T_1) = S_1 \& S(T_2) = S_2\}$.

Домовимося операції вигляду $\Cup_{S_1}^{S_1}$ записувати як $\Cup$. Така операція діє як обмеження звичайної теоретико-множинної операції об'єднання на випадок, коли схеми таблиць співпадають.

Перетин таблиць: $T_1 \Cap_{S_2}^{S_1} T_2 = (t_1 \cap \text{REN}_{S_2}^{S_1}(t_2), S_1)$. Область визначення задається аналогічно, як і для об'єднання.

Різниця таблиць: $T_1 \backslash\backslash_{S_2}^{S_1} T_2 = (t_1 \backslash \text{REN}_{S_2}^{S_1}(t_2), S_1)$. Область визначення задається аналогічно, як і для об'єднання.

Як випливає з означення, на відміну від звичайних теоретико-множинних операцій, ці операції не є симетричними.

Розглянемо кілька прикладів. Нехай надані дві таблиці з однаковою кількістю колонок.

Таблиця 2.1 T1

A1	A2	A3	B1	B2
a11	a21	a31	b11	b21
a12	a22	a32	b12	b22
a13	a23	a33	b13	b23

Таблиця 2.2 T2

B1	B2	B3	A1	A2
b_11	b_21	b_31	a_11	a_21
b_12	b_22	b_32	a_12	a_22
b_13	b_23	b_33	a_13	a_23
b_14	b_24	b_34	a_14	a_24

Треба їх об’єднати. А саме стовпчик A1 таблиці T1 з стовпчиком B1 таблиці T2, стовпчик A2 таблиці T1 з стовпчиком B2 таблиці T2, стовпчик A3 з стовпчиком B3, стовпчик B1 з стовпчиком A1, стовпчик B2 з стовпчиком A2

Це робиться наступною операцією $T1 \cup_{B1,B2,B3,A1,A2}^{A1,A2,A3,B1,B2} T2$. Результат зображено в таблиці 2.3

Таблиця 2.2 Результат об’єднання таблиці T1 з T2

A1	A2	A3	B1	B2
a11	a21	a31	b11	b21
a12	a22	a32	b12	b22
a13	a23	a33	b13	b23
b_11	b_21	b_31	a_11	a_21
b_12	b_22	b_32	a_12	a_22
b_13	b_23	b_33	a_13	a_23
b_14	b_24	b_34	a_14	a_24

2.4 Операції з'єднання таблиць

В підрозділі визначаються операції декартова з'єднання, природнього з'єднання, та з'єднання за атрибутами.

Операція *декартова з'єднання* $\times_{a_1,a_2}$ є бінарною функцією типу $\times_{a_1,a_2}: Tbl \times Tbl \longrightarrow Tbl$ яка параметризована по префіксах атрибутів a_1, a_2, де $a_1 \neq a_2$.

Позначимо через T_1 таблицю (t_1, S_1) та через T_2 таблицю (t_2, S_2). Тоді декартове з'єднання визначається по формулі $\times_{a_1,a_2}(T_1, T_2) = (t_3, S_1 \cup_{a_1,a_2} S_2)$, де $t_3 = \{r_1 \cup_{a_1,a_2} r_2 | r_1 \in t_1 \& r_2 \in t_2\}$

Операція *природнього з'єднання або еквіз'єднання* $\otimes$ є всюди визначеною бінарною функцією типу $\otimes: Tbl \times Tbl \to Tbl$. Позначимо через T_1 таблицю (t_1, S_1) і через T_2 таблицю (t_2, S_2). Тоді природне з'єднання

визначається по формулі $\otimes(T_1, T_2) = (t_3, S_1 \cup S_2)$, де $t_3 = \{r_1 \bar{\cup} r_2 | r_1 \in t_1 \& r_2 \in t_2 \& r_1 \approx r_2\}$,

Наступною операцією є *з'єднання за атрибутами* $\otimes^B_{a_1,a_2}$ яке має тип $Tbl \times Tbl \to Tbl$, де B є множиною імен атрибутів, a_1, a_2 є префіксами імен атрибутів, такими що $a_1 \neq a_2$. З'єднання за атрибутами є бінарною функцією з областю визначення $dom \otimes^B_{a_1,a_2} \stackrel{\text{def}}{=} \{(T_1, T_2) | B \subseteq S(T_1) \cap S(T_2)\}$. Вона визначається за формулою $\otimes^B_{a_1,a_2}(T_1, T_2) = (t_3, S_1 \cup_{a_1,a_2} S_2)$, де $t_3 = \{r_1 \cup_{a_1,a_2} r_2 | r_1 \in t_1 \& r_2 \in t_2 \& r_1(B) = r_2(B)\}$

2.5 Визначення операторів на таблицях

В підрозділі визначаються оператори (функціонали), які будують запити. Задаються чотири оператора. Це оператор фільтрації , який відповідає фразі WHERE оператора SELECT, оператор відображення який відповідає фразі SELECT, оператори внутрішнього та зовнішнього з'єднання, які відповідають фразі FROM. Характерною рисою цих операторів є те що вони розповсюджують функції на рядках на реляції. Крім того вводиться допоміжний оператор суперпозиції, який будує нові функції з існуючих.

Нехай $p: Rec \to Bool$ буде, взагалі кажучи частковим, предикатом, визначеним на рядках, який використовує тризначну логіку Кліні. Позначимо булевські значення через $true, false, unknown$. Множину всіх предикатів позначимо через P.

Назвемо множину функцій типу $Tbl^n \to Tbl$, $n = 0,1,2 \ldots$ n-арними табличними функціями. Позначимо її як TF^n. Тоді множина всіх табличних

функцій TF будується як об'єднання n-арних табличних функцій $TF = \cup_{i=0,1,2\ldots} TF^i$

Оператор суперпозиції Λ

Нехай задані послідовність імен $B = (b_1, b_2, \ldots, b_n)$, і послідовність даних $D = (d_1, d_2, \ldots, d_n)$, причому $|B| = |D|$. Тоді операція побудови номінативної множини з послідовності даних та імен $B \twoheadrightarrow D$ визначається за формулою $B \twoheadrightarrow D = \{b_1 : d_1, b_2 : d_2, \ldots, b_n : d_n\}$.

Нехай задані номінативна множина $M = \{b_1 : d_1, b_2 : d_2, \ldots, b_n : d_n\}$, та послідовність імен $B' = (\mathrm{b}_{i_1}, \mathrm{b}_{i_2}, \ldots, \mathrm{b}_{i_k})$, така що всі імена з цієї послідовності належать множині імен з M. Тоді операція побудови впорядкованого зрізу з номінативної множини $\mathrm{M}|B'$ визначається як операція, що будує послідовність $(d_{i_1}, d_{i_2}, \ldots, d_{i_k})$, таку, що $(b_{i_p} : d_{i_p}) \in M, p = 1..k$.

Вираз $(B \twoheadrightarrow D)|B'$ будує зріз послідовності D.

Нехай задані послідовність імен $B = (b_1, b_2, \ldots, b_n)$, послідовності B_1, $B_2, \ldots B_m$, які містять імена з B і послідовність даних $D = (d_1, d_2, \ldots, d_n)$.

Оператор суперпозиції Λ визначається як функціонал типу $TF^n \to TF$, де $n = 0,1,2, \ldots$, по формулі:

$\Lambda(B) : g\big(f_1(B_1), f_2(B_2), \ldots, f_m(B_m)\big)(D) = g(f_1(B \twoheadrightarrow D)|B_1, \ldots, f_m(B \twoheadrightarrow D)|B_m)$, де $B, B_1, B_2, \ldots, B_m$ є параметрами оператору, а $g, f_1, f_2, \ldots, f_m$ – табличні функції.

Оператор фільтрації Ф

Нехай T є таблицею (t, S). Оператор фільтрації визначається як функціонал типу $\Phi: P \to TF^1$ що будує унарну табличну функцію з предиката на рядках за наступною формулою:

$\Phi(p)(T) = (t', S)$, де $t' = \{r | r \in t \& p(t) = true\}$, а схема S співпадає зі схемою таблиці T. Про такі оператори будемо говорити, що вони зберігають схему таблиці.

Оператор відображення М

Функцією на рядках будемо називати часткову функцію f_S типу $Rec \to Rec(S)$, де $Rec(S)$ – множина рядків схеми S, параметризовану по схемі $S \subseteq Atr$, таку що для будь якого вхідного рядку вона повертає рядок схеми S або результат є невизначеним. Позначимо множину функцій на рядках як RF.

Нехай $f_{S'}$ є функією на рядках і T є таблицею (t, S). Тоді оператор відображення визначається як функціонал типу $RF \to TF^1$ який будує унарну табличну функцію з функції на рядках, за формулою:

$\mathrm{M}(f_{S'})(T) = (t', S')$, де $t' = \{r' | r \in t \& r' = f_{S'}(r)\}$

Оператор внутрішнього з'єднання Θ_{a_1, a_2}.

Оператор внутрішнього з'єднання за предикатом p визначається як оператор Θ_{a_1, a_2} типу $P \to TF$, параметризований по префіксам імен атрибутів a_1, a_2, де $a_1 \neq a_2$. Оператор визначається за формулою

$\Theta_{a_1,a_2}(p)(T_1,T_2) = (t_3, S_1 \cup_{a_1,a_2} S_2)$, де

$t_3 = \{r | r = (r_1 \cup_{a_1,a_2} r_2) \& r_1 \in t_1 \& r_2 \in t_2 \& p(r) = true\}$

Позначимо через $State(T)$ стан таблиці T. Нехай $dom\ p$ є областю визначення предиката p. Тоді область визначення побудованої функції буде $dom\ \Theta_{a_1,a_2}(p) \stackrel{def}{=} \{(T_1,T_2) | State(T_1 \times_{a_1,a_2} T_2) \subseteq dom\ p\}$.

Операції декартового з'єднання, природнього з'єднання та з'єднання за атрибутами є похідними від оператора з'єднання та деякого предикату.

Оператор лівостороннього зовнішнього з'єднання $\overleftarrow{\Theta}_{a_1,a_2}$.

Оператор лівостороннього зовнішнього з'єднання за предикатом p визначається як оператор $\overleftarrow{\Theta}_{a_1,a_2}$типу $P \to TF$, параметризований по префіксам імен атрибутів a_1, a_2, де $a_1 \neq a_2$.

Визначимо допоміжний оператор лівостороннього доповнення

$\ltimes_{a_1,a_2}(p)(T_1,T_2) = (t_3, S_3)$, де $S_3 = S_1$ і

$t_3 = \{r_1 | r_1 \in t_1 \& \forall r_2 (r_2 \in t_2 \Rightarrow p(r_1 \cup_{a_1,a_2} r_2) \neq true)\}$

Введемо операцію побудови таблиці зі схемою S, невизначеної на кожному атрибуті. Вона будується за формулою $Null(S) = (t_{Null}, S)$, і має тип $Sch \to Tbl$. Нехай схема S дорівнює $\{a_1, a_2, \dots, a_n\}$, тоді $t_{Null} = \{\{a_1 : null, a_2 : null, \dots, a_n : null\}\}$, тобто таблиця складається з одного рядку з невизначеними значеннями.

Тоді оператор лівостороннього зовнішнього з'єднання визначається за формулою:

$$\overleftarrow{\Theta}_{a_1,a_2}(p)(T_1,T_2) = \Theta_{a_1,a_2}(p)(T_1,T_2) \Cup (\ltimes_{a_1,a_2}(p)(T_1,T_2) \times_{a_1,a_2} Null(S(T_2)))$$

Аналогічним чином визначаються оператори правостороннього зовнішнього з'єднання $\overrightarrow{\Theta}_{a_1,a_2}$ та повного зовнішнього з'єднання $\overleftrightarrow{\Theta}_{a_1,a_2}$

Введені оператори відносяться до одного з наступних типів:

Оператори типу $P \to TF$. До цього типу відносяться оператори фільтрації та оператори зовнішнього та внутрішнього з'єднання за предикатом.

Оператори типу $RF \to TF$. Єдиний оператор цього типу це відображення

Оператори типу $TF^n \to TF$. До цього типу відноситься оператор суперпозиції.

Розглянемо приклад з'єднання таблиць. Нехай є дві таблиці 2.4 і 2.5. Треба виконати операцію внутрішнього з'єднання за всіх рядків для яких значення колонки A2 першої таблиці більше ніж значення колонки A2 другої таблиці.

Таке з'єднання виконується наступним оператором: $\Theta_{t_1,t_2}(t_1.A2 > t_2.A2)(T_1,T_2)$. Результат зображений в таблиці 2.5

Таблиця 2.4 T1

A1	A2	A3
a11	21	a31

a12	22	a32
a13	23	a33

Таблиця 2.5 T2

B1	B2	A2
b_11	b_21	21
b_12	b_22	22
b_13	b_23	23
b_14	b_24	24

Таблиця 2.6 Результат внутрішнього з’єднання таблиць

t_1.A1	t_1.A2	t_1.A3	t_2.B1	t_2.B2	t_2.A2
a12	22	a32	b_12	b_22	22
a13	23	a33	b_12	b_22	22
a13	23	a33	b_13	b_23	23

Таблична алгебри семантичних функцій SQL є багатосортною [77]. Її носій складається з множини предикатів PF, множини функцій на рядках RF і множини табличних функцій TF. Сигнатура складається з операторів

фільтрації, відображення, групи операторів з'єднання за предикатом, та суперпозиції. Позначимо таку алгебру як $(PF, RF, TF, (\Phi, \mathrm{M}, \Theta, \Lambda))$.

Різні версії SQL відрізняються множиною предикатів і функцій на рядках. Тому, задаючи різні множини предикатів та функцій на рядках, отримуємо версії табличної алгебри, які задають семантику SQL, втілених в різних СУБД.

Задана алгебра складається з функцій, визначених на даних і з функцій біль високого порядку, які називаються операторами. Такий підхід дозволяє описати семантику запитів SQL більш точно ніж реляційна алгебра. Таблична алгебра може бути розширена операторами групування та іншими операторами.

2.6. Відношення конфінальності, передпорядки та порядки

Підрозділ присвячений властивостям відношення конфінальності – бінарного відношення на булеані множини, індукованого бінарним відношенням на множини; [72, 73]

Відношення конфінальності та коініціальності: загальні властивості

Поняття конфінальності (точніше кажучи, поняття конфінальної підмножини) є класичним поняттям теорії множин (див., наприклад, [62, гл. III, § 1, п. 7, с. 146; 48, § 2, с. 22]); в теорії табличних алгебр це поняття природно використовується при явному введенні відношення порядку на таблицях, що відповідає комутативній напівгрупі таблиць за з'єднанням ([63, підрозділ 2.8, с. 56-58]). Основні властивості відношення конфінальності наведені в роботах [60, 58, 71, 70].

Зауважимо в класичних роботах по суті вводиться поняття конфінальної підмножини, яке природно можна розповсюдити на довільні множини (див. далі). Крім того, основний смисл введення поняття конфінальної підмножини полягав в тому, що задача знаходження супремуму множини зводиться до аналогічної задачі для конфінальної підмножини; більш того, не важно

показати, що супремуми множини та її конфінальної підмножини існують або не існують одночасно, причому у випадку існування збігаються (див. далі). Перейдемо до точних означень.

Зафіксуємо бінарне відношення $\leq$ на універсумі D. Це відношення індукує бінарне відношення конфінальності $\prec$ на булеані універсуму $P(D)$:

$$X \prec Y \overset{def}{\Longleftrightarrow} \forall x(x \in X \Rightarrow \exists y(y \in Y \& x \leq y)).$$

Зауважимо, що за термінологією [92, розд. I, § 2, с. 39] відношення $X \pi Y$ означає, що множина Y конфінальна множині X. Безпосередньо перевіряється, що відношення конфінальності має найменшим елементом порожню множину $\varnothing$, тобто виконується $\emptyset \prec X$ для всіх множин X. Крім того, порожня множина є і мінімальним елементом в тому розумінні, що виконується імплікація $X \prec \emptyset \Rightarrow X = \emptyset$. Остання імплікація легко перевіряється від супротивного.

Що стосується властивостей рефлексивності та транзитивності, то мають місце дві наступні леми про зв‘язок вказаних властивостей вихідного відношення на універсумі $\leq$ та індукованого відношення конфінальності $\prec$.

Лема 2.1 (про рефлексивність відношення конфінальності). Відношення конфінальності $\prec$ рефлексивне тоді і тільки тоді, коли відношення $\leq$ рефлексивне. □

Доведення. Ідея подальших доведень полягає в тому, що на одноелементних множинах (сінглітонах) відношення конфінальності по суті збігається з початковим відношенням.

Почнемо з необхідності. Нехай відношення конфінальності $\prec$ рефлексивне, покажемо від супротивного, що тоді і початкове відношення $\leq$

рефлексивне. Дійсно, нехай відношення $\leq$ не є рефлексивним, тоді існує елемент універсуму x, такий, що $x \not\leq x$. Але відношення конфінальності рефлексивне, тобто виконується $\{x\} \prec \{x\}$, звідки очевидно випливає, що $x \leq x$. Прийшли до суперечності. ■

Достатність. Нехай відношення $\leq$ рефлексивне, покажемо, що відношення конфінальності $\prec$ також рефлексивне. Дійсно, розглянемо довільну множину X. Якщо ця множина порожня, то, як зазначалося вище, має місце $\emptyset \prec \emptyset$. Нехай ця множина не порожня. Тоді розглянемо довільній елемент $x \in X$. Залишається врахувати, що $x \leq x$ (тобто в якості елемента множини X, більшого за x, можна взяти той самий елемент x); таким чином, $X \prec X$. ■ □

Лема 2.2 (про транзитивність відношення конфінальності). Відношення конфінальності $\prec$ транзитивне тоді і тільки тоді, коли відношення $\leq$ транзитивне. □

Почнемо з необхідності. Нехай відношення конфінальності $\prec$ транзитивне, покажемо від супротивного, що тоді і початкове відношення $\leq$ транзитивне. Дійсно, нехай відношення $\leq$ не є транзитивним, тоді існують елементи універсуму x, y, z, такі, що виконується $x \leq y$, $y \leq z$, але $x \not\leq z$. Очевидно, що $\{x\} \prec \{y\}$ та $\{y\} \prec \{z\}$. З транзитивності відношення конфінальності випливає, що тоді $\{x\} \prec \{z\}$, звідки $x \leq z$; прийшли до суперечності. ■

Достатність. Нехай відношення $\leq$ транзитивне, покажемо, що відношення конфінальності $\prec$ також транзитивне. Дійсно, розглянемо довільні множини X, Y, Z, такі, що $X \prec Y$, $Y \prec Z$, та покажемо, що тоді $X \prec Z$.

Випадок порожньої множини X тривіальний, тому розглянемо довільний елемент $x \in X$. Оскільки $X \prec Y$то знайдеться елемент $y \in Y$, такий, що $x \leq y$ В свою чергу, оскільки $Y \prec Z$, то для елемента $y \in Y$ знайдеться елемент z$\in Z$, такий, що $y \leq z$. З транзитивності початкового відношення випливає, що $x \leq z$. Таким чином, для довільного елемента множини X в множині Z знайшовся елемент, його більший, а це і означає виконання $X \prec Z$. ■ □

Нагадаймо, що рефлексивне і транзитивне бінарне відношення називається відношенням передпорядку (див., наприклад, [62, , гл. III, § 1, п. 2, с. 138;92, гл. I, § 2, с. 37]).

З лем 2.1-2.2 безпосередньо випливає наступний наслідок.

Наслідок 2.1 (критерій бути передпорядком для відношення конфінальності). Відношення конфінальності $\prec$ є передпорядком тоді і тільки тоді, коли відношення $\leq$ є передпорядком. □

Розглянемо тепер випадок, коли початкове відношення $\leq$ є порядком (взагалі кажучи, частковим). З наслідку 1 випливає, що відношення конфінальності $\prec$ є тільки передпорядком. Дійсно, розглянемо наступний простий приклад.

Нехай $< N, \leq >$ – множина натуральних чисел з стандартним порядком; N', N'' – множини парних та непарних натуральних чисел відповідно. Очевидно, що $N' \prec N''$ та навпаки $N'' \prec N'$, але $N' \neq N''$.

Таким чином, задача полягає у пошуку умов того, щоб відношення конфінальності було порядком.

Далі будемо вважати, що початкове відношення $\leq$ є порядком. Крім того будемо розглядати відношення конфінальності не на всьому булеані, а на дискретних множинах.

Множину X назвемо дискретною, якщо частково впорядкована множина $< X, \leq >$ (за індукованим порядком) дискретна в звичайному розумінні [118, § 1, с. 8], тобто для всіх $x_1, x_2 \in X$ виконується імплікація $x_1 \leq x_2 \Rightarrow x_1 = x_2$.

Найпростіший приклад дискретної множини – сінглітон.

Твердження 2.1 (про відношення конфінальності на дискретних множинах). Відношення конфінальності (індуковане частковим порядком) частково впорядковує сім'ю дискретних множин. □

Доведення. Очевидно, достатньо встановити антисиметричність відношення на дискретних множинах. Нехай $X \pi Y$ та $Y \pi X$, причому множини X, Y дискретні. Треба показати, що ці множини рівні.

Нехай спочатку $X = \emptyset$, тоді, з огляду на мінімальність порожньої множини, маємо рівність $Y = \emptyset$.

Нехай тепер множина X не порожня і розглянемо довільний елемент $x \in X$. Тоді знайдуться елементи $y \in Y$ і $z \in X$, такі, що $x \leq y \leq z$; звідки, враховуючи транзитивність відношення $\leq$, отримуємо $x \leq z$. Але x, z –

елементи дискретної множини X, тобто $X=Z$. Таким чином, $x \le y \le x$, звідки $x=y$ з огляду на анти симетричність порядку $\le$. Отже, $x \in Y$, тобто включення $X \subseteq Y$ доведене. Обернене включення $Y \subseteq X$ встановлюється повністю аналогічно (просто множини X, Y міняються ролями). □

Незважаючи на начебто штучність твердження 2.1, зауважимо, що два природних часткових порядку на таблицях (які є дискретними множинами відносно порядку включення рядків в розумінні теоретико-множинного включення графіків рядків) будуються саме за наведеною логічною схемою [115; 63, підрозділ 2.6, с. 47-48, підрозділ 2.8, с. 56-58, 62]. Це питання ще буде далі розглянуто більш детально.

Наведене доведення можна знайти в [63, підрозділ 1.3, с. 25], крім явного розгляду випадку порожніх множин.

Вказані властивості відношення конфінальності переносяться на випадок відношення коініціальності $\sqsubset$ на булеані універсуму $P(D)$, яке також індукується початковим відношенням:

$$X \sqsubset Y \overset{def}{\Longleftrightarrow} \forall x(x \in X \Rightarrow \exists y(y \in Y \& y \le x)).$$

Відповідні аналоги наведемо без доведення, які проводяться повністю аналогічно випадку конфінальності. Зауважимо, що порожня множина є найменшим та мінімальним елементом в розумінні $\emptyset \sqsubset X$ та імплікації $X \sqsubset \emptyset \Rightarrow X = \emptyset$.

Лема 2.1' (про рефлексивність відношення коініціальності). Відношення коініціальності $\sqsubset$ рефлексивне тоді і тільки тоді, коли відношення $\le$ рефлексивне. □

Лема 2.2' (про транзитивність відношення коініціальності). Відношення коініціальності $\sqsubset$ транзитивне тоді і тільки тоді, коли відношення $\leq$ транзитивне. □

Наслідок 2.1' (критерій передпорядку для відношення коініціальності). Відношення коініціальності $\sqsubset$ є передпорядком тоді і тільки тоді, коли відношення $\leq$ є передпорядком. □

Твердження 2.1' (про відношення коініціальності на дискретних множинах). Відношення коініціальності (індуковане частковим порядком) частково впорядковує сім'ю дискретних множин. □

Застосуємо отримані результати (твердження 1 та 1 ') для побудови природних впорядкувань таблиць. Всі неозначувані тут поняття та позначення для таблиць використовуються в розумінні монографії [63].

В якості універсального домену розглянемо множину всіх рядків S, впорядкованих за включенням їхніх графіків порядком $\subseteq$. На булеані $P(S)$ розглянемо за розглянутою схемою відношення конфінальності та коініціальності:

$$X \prec Y \overset{def}{\Longleftrightarrow} \forall s(s \in X \Rightarrow \exists s'(s' \in Y \& s \leq s')),$$

$$X \sqsubset Y \overset{def}{\Longleftrightarrow} \forall s(s \in X \Rightarrow \exists s'(s' \in Y \& s' \leq s)).$$

Оскільки теоретико-множинне відношення включення $\subseteq$ є порядеом, то введені відношення конфінальності та коініціальності згідно з наслідками 1 та 1' є передпоряками.

Лема 2.3 (про дискретність таблиць). Довільна множина односхемних рядків, впорядкована відношенням теоретико-множинного включення $\subseteq$ є

дискретною. Зокрема кожна таблиця, впорядкована аналогічним чином, є дискретною. □

Доведення. Нехай X довільна множина рядків, причому всі рядки мають однакову схему (нагадаймо, в даному випадку схема це область визначення рядка як функції). Розглянемо два довільних рядки цієї множини s, s', такі, що, наприклад, $s \subseteq s'$. Неважко зрозуміти, що так як $doms = doms'$, то $s = s'$. Більш формально доведення можна провести з використанням загальної властивості обмеження $f \subseteq g \,\&\, domf = domg \Rightarrow f = g$ (див. [71, підрозділ 1.4, твердження 1.11, п. 10]). ■

Зрозуміло, що встановлена властивість прямо переноситься на таблиці, оскільки, згідно з означенням, таблиця є скінченною множиною одно схемних рядків. ■ □

З цієї леми та тверджень 2.1 та 2.1’ безпосередньо випливає наступний наслідок.

Теорема 2.1 (про відношення конфінальності та коініціальності на таблицях, індуковані теоретико-множинним включенням рядків). Відношення конфінальності $\prec$ та коініціальності $\sqsubset$ частково впорядковують множину таблиць. □

Зауважимо, що відношення конфінальності $\prec$ природньо зв‘язане з операцією проекції табличних алгебр [63, підрозділ 2.6, с. 47-48], а відношення коініціальності з операцією з‘єднання цих же алгебр [63, підрозділ 2.8, с. 56-58].

Відношення конфінальності та ініціальності: знаходження точних граней

Всюди далі на універсамі розглядається частковий порядок $\leq$.

Наведемо загальні властивості відношення конфінальності, позначаючи верхній (нижній) конус множини X як X^{Δ} (відповідно X^{∇}), а супремум (інфімум) цієї ж множини як $\bigvee X$ (відповідно $\bigwedge X$).

Твердження 2.2 (про верхні конуси конфінальних множин). Для довільних множин виконується імплікація $X \pi Y \Rightarrow Y^{\Delta} \subseteq X^{\Delta}$. □

Доведення очевидне, бо кожна верхня грань множини Y є і верхньою гранню множини X . □

Наслідок 2.2 (про супремуми конфінальних множин). Для довільних множин за умови існування їхніх супремумів виконується імплікація $X \pi Y \Rightarrow \bigvee X \le \bigvee Y$. □

Доведення випливає з попереднього твердження та означення супремумів, як найменших елементів відповідних верхніх конусів. □

В літературі поняття конфінальності найчастіше використовується для конфінальних підмножин в наступному розумінні: множину Y назвемо конфінальною підмножиною множини X, якщо, по-перше, $Y \subseteq X$ та, по-друге, $X \pi Y$.

Основна властивість конфінальних підмножин сформульована в наступному твердженні.

Твердження 2.3 (властивості конфінальних підмножин). Якщо Y конфінальна підмножина множини X, то верхні конуси цих множин співпадають – $X^{\Delta} = Y^{\Delta}$; тобто виконується узагальнена рівність $\bigvee X \simeq \bigvee Y$. □

Вище через $\simeq$ позначена узагальнена рівність (по іншій термінології сильна рівність Кліні): вираз $\varphi \simeq \phi$ означає, що або обидві частини (одночасно)

невизначені, або визначені та мають рівні значення (див., наприклад, [70, вступ, с. 11].)

Доведення. Нехай $X \pi Y$ та $Y \subseteq X$, тоді за твердженням 2 $Y^{\Delta} \subseteq X^{\Delta}$. Крім того, за загальними властивостями конусів виконується включення $Y^{\Delta} \supseteq X^{\Delta}$ (див., наприклад, [76]). Значить, верхні конуси множин X, Y співпадають; і тому вони одночасно мають або не мають найменші елементи – супремуми відповідних множин. Звідси і випливає узагальнена рівність з формулювання твердження. □

Отже, питання про супремум множини еквівалентне тому самому питанню для її конфінальної підмножини.

Зауважимо, що незважаючи на простоту, природність та потужність властивості конфінальних підмножин (твердження 3), в явному вигляді ця властивість відсутня в літературі (див., наприклад, [118, § 1; 92, розд. 1, § 2;]).

На останок додамо, що згідно з принципом двоїстості (див., наприклад, [118, § 1, с. 10]) при переході до оберненого (інверсного) порядку всі твердження для конфінальності переходять в твердження для коініціальності з заміною верхніх конусів на нижні, супремумів на інфінуми. Наведемо відповідні означення та аналоги вищенаведених тверджень.

Твердження 2.2' (про нижні конуси коініціальних множин). Для довільних множин виконується імплікація $X \sqsubset Y \Rightarrow Y^{\nabla} \subseteq X^{\nabla}$. □

Наслідок 2.2' (про інфінуми коініціальних множин). Для довільних множин за умови існування їхніх інфінумів виконується імплікація $X \sqsubset Y \Rightarrow \prod X \leq \prod Y$. □

Множину Y назвемо коініціальною підмножиною множини X, якщо по-перше, $Y \subseteq X$ та, по друге, $X \sqsubset Y$.

Твердження 2.3' (властивості коініціальних підмножин). Якщо Y – коініціальна підмножина множини X, то нижні конуси цих множин співпадають – $X^{\nabla} = Y^{\nabla}$; тобто виконується узагальнена рівність $\prod X = \prod Y$. □

Передпорядки та порядки

Всюди далі будемо вважати, що вихідне відношення $\leq$ є передпорядком.В літературі добре відома конструкція побудови по передпорядку порядку на відповідній фактор-множині (див. наприклад, [92, гл. I, § 2, с. 37; 118, § 1, с. 7, теорема 3]). Наведемо цю конструкцію розгорнуто. На універсумі введемо наступне відношення $x \sim y \overset{def}{\Leftrightarrow} x \leq y \,\&\, y \leq x.$

Лема 2.4 (про еквівалентність, індуковану передпорядком). Відношення $\sim$ є еквівалентністю. □

Доведення. Рефлексивність відношення $\sim$ випливає з рефлективності відношення $\leq$. ■

Транзитивність відношення $\sim$ випливає з транзитивності відношення $\leq$. Дійсно, нехай $x \sim y$ та $y \sim z$, покажемо, що тоді $x \sim z$. Очевидно, $x \leq y$ та $y \leq z$, тобто $x \leq z$. Аналогічно встановлюється, що $z \leq x$; отже, $x \sim z$. ■

Симетричність відношення $\sim$ випливає з його означення. ■ □

Лема 2.5 (про устрій суміжних класів еквівалентності, індукованої передпорядком). Якщо x, y довільні елементи одного суміжного класу еквівалентності $\sim$, то виконується $x \leq y$ та $y \leq x$. □

Доведення. Нехай x, y належать до одного суміжного класу еквівалентності з представником z : $x, y \in [z]$. Тоді $x \sim z$ та $y \sim z$. Отже $x \leq z$ та $z \leq y$, звідки $x \leq y$. зворотна нерівність $y \leq x$ встановлюється повністю аналогічно. □

На фактор-множині $D/\sim$ (множині всіх суміжних класів по еквівалентності $\sim$) розглянемо наступне бінарне відношення $\eta \ll \theta \overset{def}{\Leftrightarrow} \exists x \exists y (x \in \eta \& y \in \theta \& x \leq y)$, де η, θ суміжні класи.

Зауважимо, що в літературі відношення $\ll$ вводиться через представників суміжних класів наступним чином $[x] \ll [y] \overset{def}{\Leftrightarrow} x \leq y$. Вада цього шляху полягає в тому, що треба показувати коректність цього означення (незалежність від представників), Це робиться з використанням леми 4.

Твердження 2.4 (про порядок на фактор-множині, індукований передпорядком). Відношення $\ll$ на фактор множині є $D/\sim$ є порядком. □

Доведення. Рефлексивність відношення $\ll$ випливає з рефлексивності передпорядку $\leq$. ■

Встановимо транзитивність відношення $\ll$. Нехай $\eta \ll \theta$ та $\theta \ll \vartheta$, покажемо, що тоді $\eta \ll \vartheta$. Дійсно з означення відношення $\ll$ випливає існування таких елементів $x \in \eta, y_1 \in \theta$, $y_2 \in \theta$, $z \in \vartheta$, що має місце $x \leq y_1$, $y_2 \leq z$. Але елементи y_1, y_2 належать одному суміжному класу, значить згідно з лемою 4 виконується нерівність $y_1 \leq y_2$. З транзитивності вихідного перед

порядку випливає $x \leq z$ і залишається скористатися означенням відношення $\ll$.

■

Нарешті встановимо антисиметричність відношення $\ll$. Нехай для суміжних класів η, θ виконуються нерівності $\eta \ll \theta$ та $\theta \ll \eta$.

Покажемо, що ці суміжні класи співпадають. Дійсно, від супротивного: нехай вони різні. З означення відношення $\ll$ випливає існування таких елементів $x_1 \in \eta, y_1 \in \theta$, $y_2 \in \theta$, $x_2 \in \eta$ що має місце $x_1 \leq y_1$, $y_2 \leq x_2$. Але елементи x_1, x_2 належать одному суміжному класу і так само для елементів y_1, y_2. За лемою 4 маємо нерівності $y_1 \leq y_2$ та $x_2 \leq x_1$ (див рис. 1, на якому направлена стрілка йде від меншого елемента до більшого). Отже маємо ланцюжок нерівностей $y_1 \leq y_2 \leq x_2 \leq x_1$, звідки з огляду на транзитивність вихідного перед порядку випливає нерівність $y_1 \leq x_1$. З врахуванням нерівності $x_1 \leq y_1$ маємо, що $x_1 \sim y_1$. Таким чином, ці елементи повинні належати одному суміжному класу; прийшли до суперечності. ■ □

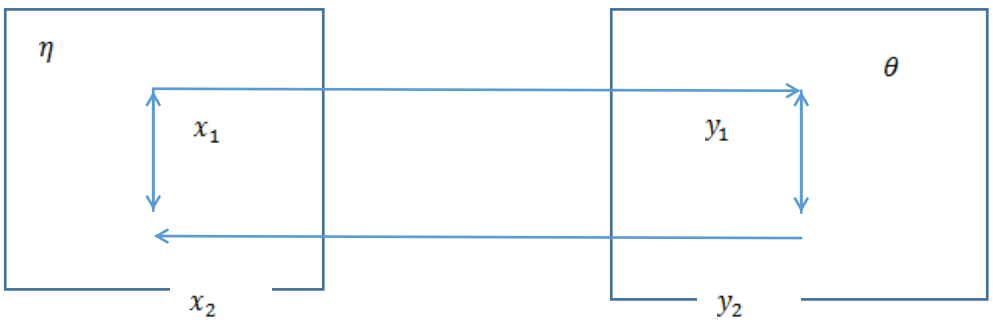

Рис. 2.1. До доведення твердження 2.4 (антисиметричність відношення $\ll$)

В термінах розглянутої фактор-множини наведемо простий, але корисний критерій того, що перед порядок є порядком.

Твердження 2.5 (критерій передпорядку бути порядком). Передпорядок $\leq$ є порядком тоді і тільки тоді, коли всі суміжні класи фактор-множини $D/\sim$ є сінглітонами. □

Доведення. Почнемо з необхідності. Нехай перед порядок є порядком (тобто він є антисиметричним), покажемо від супротивного, що вказана фактор-множина містить тільки сінглітони. Отже, нехай існує суміжний клас, що містить два різних елемента x, y. Тоді за лемою 4 $x \leq y$ та навпаки $y \leq x$. Але це суперечить анти симетричності відношення $\leq$. ■

Достатність встановимо також від супротивного. Нехай фактор-множина містить тільки сінглітони, але пере порядок не є порядком. Оскільки відношення $\leq$ є рефлексивним та транзитивним, то тоді порушується властивість антисиметричності, тобто існують різні елементи x, y, такі. Що $x \leq y$ та навпаки $y \leq x$. Звідси за означенням еквівалентності $x \sim y$, тобто ці елементи належать одному суміжному класу. Значить, знайшовся суміжний клас (наприклад, з представником x), який не є сінглітоном. ■ □

2.7 Семантика фрази ORDER BY запитів SQL-подібних мов

Семантика фрази ORDER BY запитів SQL-подібних мов була розглянута в монографії [63, підрозділ 3.5, с. 159-165], де було показано, що бінарне відношення, яке уточнює вказану фразу, є в загальному випадку передпорядком, а не порядком на рядках таблиці.

Далі буде наведений критерій того, що відповідне бінарне відношення є не тільки передпорядком, а і порядком (при цьому будемо спиратися на критерій, сформульований у твердженні 5.)

Нагадаймо основні означення з [63] щодо впорядкування рядків таблиць. Нехай t таблиця схеми R, таблиця розглядається як множина рядків s тієї ж

схеми R; в свою чергу рядок схеми R є функцією вигляду $s: R \to D$, де D універсальна множина, яка вважається лінійно впорядкованою з порядком $\leq$. Нехай $< A_1, \dots, A_n >$ – кортеж атрибутів схеми R (тобто $A_1, \dots, A_n \in R$). На рядках таблиці вводиться наступне параметризоване за кортежем атрибутів $< A_1, \dots, A_n >$ бінарне відношення

$$s \preccurlyeq s' \overset{def}{\Leftrightarrow} s(A_1) < s'(A_1) \vee \big(s(A_1) = s'(A_1) \& s(A_2) < s'(A_2)\big) \vee \dots \vee \big(\big(s(A_1) = s'(A_1) \wedge \dots \wedge s(A_{n-1}) = s'(A_{n-1})\big) \& s(A_n) < s'(A_n)\big) \vee \big(\big(s(A_1) = s'(A_1) \& \dots \& s(A_{n-1}) = s'(A_{n-1})\big) \& s(A_n) \leq s'(A_n)\big)$$

.

В наведеному означенні $s(A)$ – значення атрибуту A в рядку s, а $<$ – строга нерівність, тобто $d < d' \overset{def}{\Leftrightarrow} d \leq d' \& d \neq d'$.

Змістовна схема введеного відношення така: спочатку порівнюються значення (в рядках, що порівнюються) першого атрибуту (A_1); якщо ці значення рівні, то переходимо до порівняння значень другого атрибуту (A_2), якщо значення і першого і другого атрибутів рівні, то переходимо до порівняння значень третього атрибуту і т.д.

Якщо ж казати точно, то мова по суті йде про лексикографічний добуток згідно з порядком атрибутів кортежу параметра (див., наприклад, [118, § 2, с. 28]).

Лема 2.6. Відношення $\preccurlyeq$ на рядках таблиці є передпорядком. □

Доведення. Рефлексивність відношення $\preccurlyeq$ випливає з його означення та рефлексивності відношення $\leq$ на універсальному домені■.

Транзитивність відношення □ випливає з його означення та транзитивності відношення $\leq$ на універсальному домені. ■□

Отже відношення на рядках таблиці є передпорядком, можна навести прості приклади, коли воно не буде антисиметричним (див. [63, підрозділ 3.5, с. 159-165]). Тому виникає природнє питання про пошук критерію, коли цей передпорядок переходить в порядок.

Нам знадобиться наступна лема про устрій суміжних класів рядків для еквівалентності, індукованою передпорядком $\preccurlyeq$. Нижче (параметричне) відношення еквівалентності на рядках, індуковане предпорядком, $\sim$ вводиться так само як і для загального випадку: $s \sim s' \overset{def}{\Leftrightarrow} s \preccurlyeq s' \& s' \preccurlyeq s$.

Лема 2.7 (про устрій суміжних класів еквівалентності $\sim$ на рядках таблиці). Для довільних рядків s, s' виконується еквівалентність

$$s \sim s' \Leftrightarrow s(A_1) = s'(A_1) \& \ldots \& s(A_n) = s'(A_n).$$

Іншими словами два рядка належать одному суміжному класу тоді і тільки тоді, коли в цих рядках співпадають значення всіх атрибутів з кортежа-параметра. □

Доведення. Достатність тривіальна, а необхідність доводиться безпосередньо. □

Нижче поняття ключа (точніше кажучи, первинного ключа, primary key) таблиці використовується в стандартному розумінні: множина атрибутів X називається ключем таблиці t, якщо виконується наступне твердження

$$\forall s \forall s' (s, s' \in t \& s|X = s'|X \Rightarrow s = s').$$

В цьому означенні $s|X$ – обмеження рядка s (як функції) на множину атрибутів X.

Змістовно кажучи, множина атрибутів X називається ключем таблиці t, якщо з співпадіння значень всіх ключових атрибутів в довільних двох рядках таблиці випливає спів падіння самих рядків (в еквівалентній формі: якщо два довільних різних рядка таблиці розрізняються значенням хоча б одного з ключових атрибутів).

Теорема 2.2 (критерій передпорядку на рядках $\preccurlyeq$ бути порядком). Параметричне відношення передпорядку на рядках буде порядком тоді і тільки тоді, коли множина атрибутів $\{A_1, \dots, A_n\}$ містить ключ. □

Доведення. Достатність. Нехай множина атрибутів $\{A_1, \dots, A_n\}$ містить ключ (в якості підмножини). Тоді, очевидно, що і сама множина атрибутів $\{A_1, \dots, A_n\}$ буде ключем. З леми 6 та означення ключа випливає, що довільний суміжний клас еквівалентності $\sim$ буде одноелементним. Залишається застосувати критерій з твердження 2.5. ■

Необхідність. Нехай відношення передпорядку $\preccurlyeq$ є порядком, тоді, з огляду на критерій твердження 2.5 кожний суміжний клас фактор множини $t/\sim$ ‘ є сінглітоном. Але з цього випливає, що множина атрибутів $\{A_1, \dots, A_n\}$ є ключем. ■ □

РОЗДІЛ 3. МАТЕМАТИЧНІ ОСНОВИ АЛГОРИТМІВ ЛІНЕАРИЗАЦІЇ

3.1 Основні властивості операції лінеаризації

Розділ присвячений математичним основам алгоритмів лінеаризації – одного з методів розв‘язання конфлікту імен, що виникає при множинному успадкуванні в об’єктно-реляційних базах даних, та об‘єктно-орієнтованих мовах програмування. Основним об‘єктом дослідження є рефлексивно-транзитивне замикання бінарного відношення. Встановлені основні властивості цього замикання: критерій бути частковим порядком, замикання є оператором замикання відносно теоретико-множинного включення, наведені три неявні представлення замикання, виходячи з його властивостей, та як найменшого розв‘язку певного характеристичного рівняння (при цьому встановлена структура множини всіх розв’язків такого рівняння).

Незважаючи на те, що алгоритми лінеаризації були запропоновані ще на початку 90-х років XX століття, вони (алгоритми), строго кажучи, на сьогоднішній день не мають формального математичного обґрунтування, тобто алгоритми формально не верифіковані.

Теорія бінарних відносин займає важливе місце в математиці взагалі, і в дискретної математики, зокрема; ми згадаємо тільки не дуже відомі фундаментальні класичні роботи [116, 64] (відзначимо, що [116] - переклад, який вийшов з друку в 1963 р, оригінальної роботи, надрукованій в Бюлетені французького математичного товариства в 1948 р), які не втратили і зараз свою актуальність.

У розділі отримані такі основні результати: встановлено властивості двох допоміжних операцій (інверсії бінарного відносини і добутки бінарних відносин), а також рефлексивно-транзитивного замикання бінарного відношення. Для зазначеного замикання: (1) встановлений критерій бути

частковим порядком; (2) замикання є оператором замикання щодо теоретико-множинного включення; (3) замикання є найменшим (щодо теоретико-множинного включення) рефлексивним і транзитивним відношенням, що містить вихідне відношення; (4) замикання є множиною, породженою діагоналлю на універсуме в побудованої часткової алгебри всіх пар; (5) замикання U^* є найменшим розв'язком параметричного характеристичного рівняння $X = \Delta_D \cup X \cup U \circ X$, де Δ_D – діагональ на універсумі, а $\circ$ – добудок бінарних відносин (а також найменшим розв'язком більш простого параметричного рівняння $X = \Delta_D \cup U \circ X$), (6) встановлено структуру множини всіх рішень зазначеного рівняння. Відзначимо, що в перерахованих вище результати (3) – (5) мова йде про неявні (денотативні) завдання рефлексивно-транзитивного замикання.

Всі стандартні поняття для бінарних відносин розуміються в сенсі [92], для замкнутості викладу наведені основні визначення.

Зафіксуємо універсум D, елементи якого позначимо $x, y, z, \ldots$; бінарні відносини на D позначимо $U, V, \ldots$ Усюди далі символ ■ позначає кінець формулювання твердження і доказу, а символ □ - кінець логічної частини доказу.

3.2 Допоміжні операції інверсії і добутку бінарних відносин

У наступному твердженні наведені потрібні далі властивості інверсії бінарного відношення, яке вводяться стандартно: $U^{-1} = \{\langle y, x\rangle \mid \langle x, y\rangle \in U\}$. Нижче $U \circ V = \{\langle x, y\rangle \mid \exists z (\langle x, z\rangle \in U \wedge \langle z, y\rangle \in V)\}$ – звичайний добуток відносин U і V (в зазначеному порядку).

Пропозиція 3.1 (властивості інверсії). Виконуються наступні твердження:

1) $U \subseteq V \Rightarrow U^{-1} \subseteq V^{-1}$ (монотонність інверсії відносно теоретико-множинного включення);

2) $\Delta_D^{-1} = \Delta_D$, где $\Delta_D = \{\langle x, x\rangle | x \in D\}$ – діагональ на D (діагональ – нерухома точка для інверсії[1]);

3) $\left(\bigcup_{i\in I} U_i\right)^{-1} = \bigcup_{i\in I} U_i^{-1}$ (дистрибутивність інверсії відносно об'єднання);

4) $(U \circ V)^{-1} = V^{-1} \circ U^{-1}$ (інверсія добутку відношень);

5) $(U^{-1})^{-1} = U$ (інверсія інверсії). ■

Доведення проводиться безпосередньо. ■

Відмітимо, що інверсія є дистрибутивною і відносно перетину.

В наступному твердженні наведені необхідні далі властивості добутку відношень.

Пропозиція 3.2 (властивості добутку). Виконуються наступні твердження:

1) $U \subseteq V \wedge U' \subseteq V' \Rightarrow U \circ U' \subseteq V \circ V'$ (монотонність відносно теоретико-множинного включення по кожному аргументу[2]);

---

[1] Це твердження можна підсилити: всі нерухомі точки інверсії і тільки вони мають вигляд піддиагоналей – Δ_L, де $L \subseteq D$.

[2] А, значить, і по сукупності аргументів.

2) $\left(\bigcup_{i\in I} U_i\right) \circ \left(\bigcup_{j\in J} V_j\right) = \bigcup_{i\in I, j\in J} U_i \circ V_j$ (дистрибутивність відносно об'єднання по кожному аргументу);

3) $\left(\bigcap_{i\in I} U_i\right) \circ \left(\bigcap_{j\in J} V_j\right) \subseteq \bigcap_{i\in I, j\in J} U_i \circ V_j$ (верхня оцінка добутку перетинів[3]);

4) $U \circ \Delta_D = \Delta_D \circ U = U$ (діагональ Δ_D – права (ліва) одиниця по добутку).

■

Доведення проводиться безпосередньо. ■

3.3. Допоміжні результати: рефлексивність, антисиметричність і транзитивність

Рефлексивність бінарного відношення розуміється стандартно:

U – рефлексивно $\Leftrightarrow \forall x \langle x, x\rangle \in U$.

Очевидно, що U – рефлексивно $\Leftrightarrow \Delta_D \subseteq U \Leftrightarrow \Delta_D \subseteq U^{-1}$.

Перша еквівалентність очевидна, а друга випливає з очевидною еквівалентності $\Delta_D \subseteq U \Leftrightarrow \Delta_D \subseteq U^{-1}$ (треба застосувати монотонність інверсії (п. 1 пропозиції 3.1), а також пп. 2, 5 пропозиції 3.1).

Нижче поняття антисиметричності відносин розуміється стандартно:

U – антисиметричне $\Leftrightarrow \forall xy \langle x, y\rangle \in U \wedge \langle y, x\rangle \in U \Rightarrow x = y$.

Нижче наведений критерій антисиметричності без використання предметних змінних.

[3] Іншими словами, добуток не є, взагалі кажучи, дистрибутивним відносно перетинів.

Пропозиція 3.3 (критерій антисиметричності). Виконується еквівалентність:

U – антисиметричне $\Leftrightarrow U \cap U^{-1} \subseteq \Delta_D$. ■

Доведення. Необхідність. Нехай U – антисиметричне. Розглянемо $\langle x, y\rangle \in U \cap U^{-1}$, тоді $\langle x, y\rangle \in U$ і $\langle x, y\rangle \in U^{-1}$. Таким чином, $\langle x, y\rangle \in U$ и $\langle y, x\rangle \in U$, тобто, враховуючи антисиметричність U, $x = y$, отже, $\langle x, y\rangle \in \Delta_D$. □

Достатність. Нехай $\langle x, y\rangle \in U$ і $\langle y, x\rangle \in U$, тобто $\langle x, y\rangle \in U$, а $\langle x, y\rangle \in U^{-1}$. Звідки $\langle x, y\rangle \in U \cap U^{-1} \subseteq \Delta_D$. □■

Прості приклади показують, що замінити включення на рівність в правій частині еквівалентності з пропозиції 3.3 не можна.

Включення з правої частини еквівалентності пропозиції 3.3 переходить в рівність в точності для рефлексивних і антисиметричних відносин.

Наслідок 3.1 (критерій антисиметричності і рефлексивності). Виконується еквівалентність:

U – антисиметричне і рефлексивне $\Leftrightarrow U \cap U^{-1} = \Delta_D$. ■

Доведення. Необхідність. Нехай U – антисиметричне і рефлексивне. З пропозицією 3.3 $U \cap U^{-1} \subseteq \Delta_D$. Залишається показати зворотне включення. З огляду на рефлексивність U виконуються включення $\Delta_D \subseteq U$, $\Delta_D \subseteq U^{-1}$, з яких і випливає потрібне включення $\Delta_D \subseteq U \cap U^{-1}$. □

Достатність. Нехай $U \cap U^{-1} = \Delta_D$. З огляду на пропозицію 3.3 U антисиметричне. Крім того, очевидно, що $\Delta_D \subseteq U$,(і також $\Delta_D \subseteq U^{-1} \Delta_D \subseteq U^{-1}$) тобто U є рефлексивним. □■

Переходимо до транзитивності відношень, яка розуміється в стандартному сенсі:

U – транзитивне $\Leftrightarrow \forall xyz(\langle x,y\rangle \in U \wedge \langle y,z\rangle \in U \Rightarrow \langle x,z\rangle \in U)$.

Нижче наведено критерій транзитивності без використання предметних змінних. Далі $U^2 = U \circ U$.

Пропозиція 3.4 (критерій транзитивності). Виконується еквівалентність:

U – транзитивне $\Leftrightarrow U^2 \subseteq U$. ■

Доведення проводиться безпосереднє. ■

Відзначимо, що прості приклади показують, що замінити в правій частині еквівалентності з пропозиції 3.4 включення на рівність, взагалі кажучи, не можна.

Дійсно, нехай $U = \{\langle x,y\rangle, \langle y,z\rangle, \langle x,z\rangle\}$, де елементи x, y, z попарно різні (див. рис. 3.1).

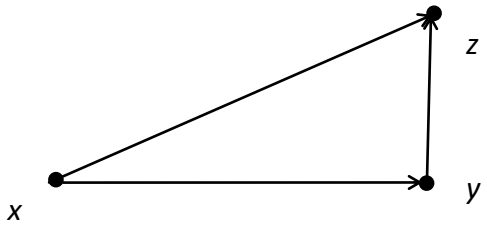

Рис. 1. Приклад транзитивного відношення, такого, що $U \subsetneq U^2$

Очевидно, що $U^2 = \{\langle x,z\rangle\}$, тобто U – транзитивне; при цьому $U \subsetneq U^2$, таким чином, рівність $U = U^2$ не виконується.

Включення з правої частини еквівалентності пропозиції 3.4 переходить в рівність для рефлексивних відносин.

Наслідок 3.2 (критерій транзитивності і рефлексивності). Виконується еквівалентність:

U – транзитивне і рефлексивне $\Leftrightarrow U^2 = U \wedge \Delta_D \subseteq U$. ■

Доведення. Необхідність. Нехай U – транзитивне і рефлексивне. За визначенням рефлексивності $\Delta_D \subseteq U$. З огляду на пропозицію 3.4 $U^2 \subseteq U$ і залишається встановити зворотне включення. Дійсно, $U \subseteq U$ і $\Delta_D \subseteq U$, звідки в силу монотонності добутку (п. 1 пропозиції 3.2) і п. 4 пропозиції 3.2 (Δ_D-одиниця за добутком) отримуємо включення $U \circ \Delta_D = U \subseteq U^2$. □■

Достатність випливає із визначення рефлексивності і пропозиції 3.4 (критерій транзитивності). □■

Нижче наведено критерій часткового порядку (рефлексивного, транзитивного і антисиметричного відношення) без використання предметних змінних.

Пропозиція 3.5 (критерій часткового порядку). Виконується еквівалентність:

U – частковий порядок $\Leftrightarrow U \cap U^{-1} \subseteq \Delta_D \wedge \Delta_D \subseteq U \wedge U^2 \subseteq U$. ■

Доказ випливає з визначення рефлексивності і критеріїв антисиметричності (пропозиція 3.3) і транзитивності (пропозиція 3.4). ■

Спростимо праву частину еквівалентності пропозиції 3.5, враховуючи рефлексивність часткового порядку.

Наслідок 3.3 (критерій часткового порядку). Виконується еквівалентність:

U – частковий порядок $\Leftrightarrow U \cap U^{-1} = \Delta_D \wedge U^2 = U$. ■

Доведення. Необхідність випливає з критеріїв антисиметричності і рефлексивності (наслідок 3.1) і транзитивності і рефлексивності (наслідок 3.2). □

Достатність також випливає з цих же критеріїв. □■

3.4. Рефлексивно-транзитивне замикання бінарного відношення

Переходимо до основного поняття роботи – рефлексивно-транзитивному замиканню відношення U, яке має наступне явне визначення:

$$U^* = \bigcup_{i=0,1,2,\ldots} U^i, \text{ де } U^0 = \Delta_D,\ U^1 = U,\ U^{i+1} = U \circ U^i,\ i = 1,2,\ldots$$

Пропозиція 3.6 (властивості рефлексивності і транзитивності рефлексивно-транзитивного замикання). U^* - рефлексивне і транзитивне. ■

Доведення. Рефлексивність очевидна ($U^0 = \Delta_D$), перевіримо транзитивність U^*. Для цього з огляду на пропозицію 3.4 (критерій транзитивності) досить перевірити включення $U^* \circ U^* \subseteq U^*$ (через рефлексивність U^* насправді буде доведена рівність). Маємо ланцюжок рівностей:

$$U^{*2} = \left(\bigcup_{i=0,1,\ldots} U^i\right) \circ \left(\bigcup_{j=0,1,\ldots} U^j\right) = \bigcup_{i=0,1,\ldots;\, j=0,1,\ldots} U^{i+j} = \bigcup_{i=0,1,\ldots} U^i \cup \bigcup_{i=0,1,\ldots;\, j=1,2,\ldots} U^{i+j} = U^*.$$

Тут скористалися дистрибутивністю добутку щодо об'єднань (п. 2 пропозиції 3.2) і очевидним включенням $\bigcup_{i=0,1,\ldots;j=1,2,\ldots} U^{i+j} \subseteq U^*$. ■

Таким чином, щоб бути частковим порядком рефлексивно-транзитивному замиканню необхідно і достатньо бути антисиметричним. Відповідний критерій

в термінах вихідного (замикаємого) відношення без використання предметних змінних наведено в наступній пропозиції.

Пропозиція 3.7 (критерій антисиметричності для рефлексивно-транзитивного замикання). Виконується еквівалентність:

$$U^* \text{ – антисиметричне} \Leftrightarrow \forall i, j = 1,2,... U^i \cap \left(U^{-1}\right)^{\curvearrowright} \subseteq \Delta_D . \blacksquare$$

Доведення. Необхідність. Нехай U^* – антисиметричне, тоді за критерієм антисиметричності (пропозиція 3.3) $U^* \cap \left(U^*\right)^{\succ 1} \subseteq \Delta_D$. Розпишемо ліву частину цього включення:

$$U^* \cap \left(U^*\right)^{\succ 1} = \bigcup_{i=0,1,...} U^i \cap \left(\bigcup_{i=0,1,...} U^i \right)^{-1} = \bigcup_{i=0,1,...} U^i \cap \bigcup_{i=0,1,...} \left(U^{-1}\right)^{\curvearrowright} = \bigcup_{i=0,1,...;\, j=0,1,..} U^i \cap \left(U^{-1}\right)^{\curvearrowright} .$$

У зазначених переходах, поряд з загальнозначимими теоретико-множинними рівностями, скористалися дистрибутивністю інверсії щодо об'єднань (другий перехід) і інверсією добутку (останній перехід) - пп. 3 і 4 пропозиції 3.1. Отже, маємо:

$\bigcup_{i=0,1,...;\, j=0,1,..} U^i \cap \left(U^{-1}\right)^{\curvearrowright} = U^* \cap \left(U^*\right)^{\succ 1} \subseteq \Delta_D$. Звідки і випливає, що $U^i \cap \left(U^{-1}\right)^{\curvearrowright} \subseteq \Delta_D$ для всіх $i, j = 1,2,...$ □

Достатність. З огляду на критерій антисиметричності (пропозиція 3.3) досить встановити включення $U^* \cap \left(U^*\right)^{\succ 1} \subseteq \Delta_D$.

Використаємо представлення лівої частини цього включення, наведене вище при доказі необхідності. Маємо:

$$U^* \cap \left(U^*\right)^{-1} = \bigcup_{i=0,1,\ldots; j=0,1,\ldots} U^i \cap \left(U^{-1}\right)^j =$$

$$= \bigcup_{i=1,2,\ldots; j=1,2,\ldots} U^i \cap \left(U^{-1}\right)^j \cup \bigcup_{j=0,1,\ldots; i=0} \Delta_D \cap \left(U^{-1}\right)^j \cup \bigcup_{i=0,1,\ldots; j=0} U^i \cap \Delta_D \subseteq$$

$$\subseteq \Delta_D \cup \Delta_D \cup \Delta_D = \Delta_D.$$

□■

Відзначимо, що в правій частині еквівалентності пропозиції 3.7 можна розширити область дії зв'язаних змінних $i, j = 0,1,2,\ldots$

Тепер можна сформулювати один з основних результатів.

Теорема 3.1 (критерій часткового порядку для рефлексивно-транзитивного замикання). Виконується еквівалентність:

$$U^* \text{ – частковий порядок} \Leftrightarrow \forall i, j = 1,2,\ldots U^i \cap \left(U^{-1}\right)^j \subseteq \Delta_D. \blacksquare$$

Доведення. Необхідність випливає з критерію антисиметричності для рефлексивно-транзитивного замикання (пропозиція 3.7), а достатність – з того ж критерію і пропозиції 3.6. ■

Саме цей результат важливий для алгоритмів лінеаризації, коли по ієрархії класів, що задовольняє по суті умові відсутності циклів у відповідному індуцированому графі на класах, будується частковий порядок на множині класів (вимога $\forall i, j = 1,2,\ldots U^i \cap \left(U^{-1}\right)^j \subseteq \Delta_D$ і говорить по суті про відсутність циклів).

3.5 Рефлексивно-транзитивне замикання бінарного відношення: характеристичні денотатівні властивості

Переходимо до властивостей рефлексивно-транзитивного замикання, важливим для внутрішньої проблематики теорії бінарних відносин. Будуть

представлені два завдання рефлексивно транзитивного відношення в термінах його властивостей.

3.5.1 Рефлексивно-транзитивне замикання як найменше рефлексивне і транзитивне відношення, що містить вихідне відношення

Розглянемо наступне сімейство відносин для відносини U:

$$F_U = \{V \mid V - \text{рефлексивне и транзитивне} \wedge U \subseteq V\}.$$

Це сімейство непорожнє: воно містить, наприклад U^*, (з огляду на пропозицію 3.6) і універсальне відношення $D \times D$ на домені.

Пропозиція 3.8 (властивість замкнутості F_U щодо перетинів). Сімейство F_U замкнуто щодо довільних перетинів. ■

Доведення. Нехай $\hat{V} = \bigcap_{i \in I} V_i$, де $V_i \in F_U$ для всіх індексів $i \in I$. Покажемо, що для перетину виконується приналежність $\hat{V} \in F_U$.

Рефлексивність $\hat{V}$ і включення $U \subseteq \hat{V}$ очевидні. □

Покажемо транзитивність $\hat{V}$. З огляду на критерій транзитивності (пропозиція 3.4) для цього достатньо перевірити включення $\hat{V} \circ \hat{V} \subseteq \hat{V}$. Дійсно, маємо ланцюжок рівностей і включень:

$$\hat{V} \circ \hat{V} = \left(\bigcap_{i \in I} V_i\right) \circ \left(\bigcap_{j \in I} V_j\right) \subseteq \bigcap_{i \in I;\, j \in I} V_i \circ V_j \subseteq \bigcap_{i \in I} V_i \circ V_i = \bigcap_{i \in I} V_i = \hat{V}. \cdot$$

У цих переходах використовували верхню оцінку добутку перетинів (другий перехід, п. 3 пропозиції 3.2) і рівність $V_i^2 = V_i$ (передостанній перехід;

використовували те, що відношення V_i рефлексивне і транзитивне, далі по слідству 3.2). □ ■

З пропозиції 3.8 випливає, що сімейство є муровскім по термінології [51] і центрованим по термінології [76].

Теорема 3.2. Виконується рівність $U^* = \bigcap_{V \in F_U} V$. ■

Доведення. Позначимо $V_0 = \bigcap_{V \in F_U} V$. Так як $U^* \in F_U$, то $V_0 \subseteq U^*$ і залишається показати зворотне включення. □

Зафіксуємо довільне $V \in F_U$ і індукцією по $k = 0,1,...$ перевіримо включення $U^k \subseteq V$.

Базис індукції. $U^0 = \Delta_D \subseteq V$ приймаючи до уваги рефлексивність V. □

Індуктивний крок. Нехай $U^k \subseteq V$, домножимо обидві частини включення на U справа; тоді, з огляду на монотонність добутку (п.1 пропозиції 3.2), включення збережеться: $U^k \circ U = U^{k+1} \subseteq V \circ U$. Залишається врахувати включення $V \circ U \subseteq V \circ V = V$ (використана та ж монотонність добутку, включення $U \subseteq V$, властивості рефлексивності і транзитивності V і відповідний критерій - наслідок 3.2). □

Отже, $U^k \subseteq V$ для всіх $k = 0,1,2,...$, тобто $U^* \subseteq V$. Звідси через довільність $V \in F_U$ і випливає необхідне включення $U^* \subseteq \bigcap_{V \in F_U} V = V_0$. □ ■

Таким чином, замикання U^* – найменше (щодо теоретико-множинного включення) рефлексивне і транзитивне відношення, що містить відношення U. Це і є перше денотативне завдання рефлексивно-транзитивного замикання.

Звідси випливає наступний наслідок.

Наслідок 3.4 (опис нерухомих точок оператора $U\,\alpha\,U^*$). Має місце:

U – рефлексивне и транзитивне$\Leftrightarrow U^* = U$. ■

Доведення. Необхідність випливає з теореми 2, а достатність - з пропозиції 3.6. ■

Пропозиція 3.9 (властивості оператора $U\,\alpha\,U^*$). Виконуються наступні твердження:

1) $U \subseteq V \Rightarrow U^* \subseteq V^*$ (монотонність відносно теоретико-множинного включення);

2) $U \subseteq U^*$ (возрастаїмість відносно теоретико-множинного включення);

3) $(U^*)^* = U^*$ (идемпотентність). ■

Доведення. П. 1 випливає з монотонності добутку (п. 1 пропозиції 3.2) і теоретико-множинного об'єднання. □

П. 2 очевидний ($U^1 = U$). □

Для перевірки п. 3 треба застосувати пропозицію 3.6 (U^* - рефлексивне і транзитивне) і наслідок 3.4 (про нерухомих точках оператора $U\,\alpha\,U^*$). □■

Нижче поняття оператора замикання використовується в сенсі [118]. Наступний наслідок є просто коротким формулюванням попередньої пропозиції 3.9.

Наслідок 3.5. Оператор $U\,\alpha\,U^*$ є оператором замикання відносно теоретико-множинного включення. ■

Доведення безпосередньо випливає з пропозиції 3.9. ■

3.5.2 Рефлексивно-транзитивне замикання як подалгебра, яка породжена діагоналлю в спеціальній алгебри всіх пар

Розглянемо наступну унарну параметричну операцію над парами:

$$\xrightarrow[x,y]{}: D\times D \xrightarrow{\sim} D\times D,\ dom \xrightarrow[x,y]{} = D\times D,\ \langle z,x\rangle \xrightarrow[x,y]{} = \langle z,y\rangle$$

(використовуємо постфіксний форму записи). Усюди далі $U\xrightarrow[x,y]{}$ - повний образ відношення U щодо операції $\xrightarrow[x,y]{}$ (також використовуємо постфіксну форму записи, щодо загальнозначущої теоретико-множинної конструкції повного образу множини щодо відношення, див. працю [59]).

Основні властивості цієї операції наведені в наступній пропозиції.

Пропозиція 3.10. Виконуються наступні твердження:

1) $\langle z,x\rangle\in U\wedge\langle x,y\rangle\in V\Rightarrow\langle z,x\rangle\xrightarrow[x,y]{}\in U\circ V$;

2) $\langle x,y\rangle\in V\Rightarrow U\xrightarrow[x,y]{}\subseteq U\circ V$;

3) $\langle x,y\rangle\in U\circ V\Rightarrow\exists z\langle x,z\rangle\in U\wedge\langle z,y\rangle\in V\wedge\langle x,y\rangle=\langle x,z\rangle\xrightarrow[z,y]{}$;

4) $U\circ V=\bigcup\limits_{\langle x,y\rangle\in V}U\xrightarrow[x,y]{}$.■

Доведення. П. 1 випливає з визначень операції $\xrightarrow[x,y]{}$ і добутку відносин. □П. 2 випливає з п. 1 і визначення повного образу. □

П. 3 випливає з визначень операції $\xrightarrow[x,y]{}$ і добутку відносин. □

П. 4. Включення правій частині в ліву випливає з доведеного п. 2 пропозиції 3.10, а зворотне включення - з уже доведеного в п. 3 цієї ж пропозиції. ■

Наслідок 3.6 (властивості операції $\xrightarrow[x,y]{}$). Виконуються наступні твердження:

1) $U^{k+1} = \bigcup_{\langle x,y\rangle \in U} U^k \xrightarrow[x,y]{}$, для всіх $k = 0,1,2,\ldots$;

2) $U^k \xrightarrow[x,y]{} \subseteq U^{k+1}$, для всіх $\langle x,y\rangle \in U$ и $k = 0,1,2,\ldots$;

3) U^* замкнуто відносно будь-якої операції виду $\xrightarrow[x,y]{}$, де $\langle x,y\rangle \in U$.■

Доведення. П. 1. З огляду на п. 4 пропозиції 3.10 і визначення добутку відношень маємо ланцюжок рівностей: $U^{k+1} = U^k \circ U = \bigcup_{\langle x,y\rangle \in U} U^k \xrightarrow[x,y]{}$.□

П. 2 випливає з п. 1. □

П. 3. Нехай $\langle x,y\rangle \in U$, маємо ланцюжок рівностей і включень:

$$U^* \xrightarrow[x,y]{} = \left[\bigcup_{k=0,1,\ldots} U^k\right] \xrightarrow[x,y]{} = \bigcup_{k=0,1,\ldots} U^k \xrightarrow[x,y]{} \subseteq \bigcup_{k=0,1,\ldots} U^{k+1} = \bigcup_{k=1,2,\ldots} U^k \subseteq U^*.$$

У другому переході була використана дистрибутивність повного образу щодо об'єднань (див. [59]), а в третьому - доведений п. 2 наслідку 3.6. Отже, для повного образу $U^* \xrightarrow[x,y]{} \subseteq U^*$, що і означає замкнутість U^* щодо операції $\xrightarrow[x,y]{}$, де $\langle x,y\rangle \in U$. ■

Зафіксуємо відношення U і розглянемо часткову параметричну алгебру всіх пар $A_U = \langle D \times D, \Omega_U \rangle$, де $\Omega_U = \left\{ \xrightarrow[x,y]{} \middle| \langle x,y\rangle \in U \right\}$ – сигнатура.

Далі $\Delta_D \xrightarrow[]{}_{\Omega_U}$ – подалгебра алгебри A_U, яка породжена діагоналлю Δ_D (з приводу подалгебр і породжуючих сукупностей див., наприклад, [гл. I, § 1, п. 1.3, 78]).

Тепер можна сформулювати твердження про друге денотативне представлення рефлексивно-транзитивного замикання.

Теорема 3.3 (рефлексивно-транзитивне замикання як замикання в алгебрі пар). Виконується рівність $U^* = \left[\Delta_D \right]_{\Omega_U}$. ■

Доведення. В силу п. 3 наслідку 3.6 виконується $\left[\Delta_D \right]_{\Omega_U} \subseteq U^*$ (адже носій даної подалгебри – найменша підмножина, що містить діагональ і замкнута щодо всіх сигнатурних операцій, а U^* містить діагональ і замкнуто щодо всіх сигнатурних операцій в силу згаданого п. 3 слідства 3.6) і залишається встановити зворотне включення. □

Для цього індукцією по $k = 0,1,2,...$ покажемо, що $U^k \subseteq \left[\Delta_D \right]_{\Omega_U}$.

Базис очевидний, так, як $U^0 = \Delta_D$. □

Індуктивний крок. В силу п. 1 слідства 3.6, індуктивного припущення і монотонності повного образу (див. [59]) маємо ланцюжок: $U^{k+1} = \bigcup_{\langle x,y\rangle \in U} \left[U^k\right]_{x,y} \subseteq \bigcup_{\langle x,y\rangle \in U} \left[\Delta_D \right]_{\Omega_U} \xrightarrow[x,y]{} \subseteq \left[\Delta_D \right]_{\Omega_U}$. В останньому переході скористалися очевидними властивостями замкнутості (щодо операцій $\xrightarrow[x,y]{}$ для $\langle x, y\rangle \in U$) замикання $\left[\Delta_D \right]_{\Omega_U}$. ■

Таким чином, рефлексивно-транзитивне замикання U^* породжується діагоналлю в алгебрі пар виду A_U. Це і є друге денотативне завдання рефлексивно-транзитивного замикання.

3.6 Рефлексивно-транзитивне замикання бінарного відношення як найменше рішення характеристичного рівняння

Дамо третю денотативну характеристику рефлексивно-транзитивного замикання, а саме як найменшого рішення відповідного природного рівняння.

Зафіксуємо відношення U і розглянемо параметричне рівняння щодо змінної $X \in 2^{D \times D}$:

$$X = \Delta_D \cup X \cup U \circ X . \qquad (3.1)$$

Теорема 3.4 (рефлексивно-транзитивне замикання як найменше рішення рівняння). Рефлексивно-транзитивне замикання U^* є найменшим (стосовно теоретико-множинного включення) розв'язком рівняння (3.1). ■

Доведення. Покажемо спочатку, що U^* є розв'язком рівняння (3.1). Маємо ланцюжок рівностей (у другому переході скористалися дистрибутивністю добутку щодо об'єднань (п. 2 пропозиції 3.2)):

$$\Delta_D \cup U^* \cup U \circ U^* = \Delta_D \cup \bigcup_{i=0,1,\ldots} U^i \cup U \circ \bigcup_{i=0,1,\ldots} U^i = \Delta_D \cup \bigcup_{i=0,1,\ldots} U^i \cup \bigcup_{i=0,1,\ldots} U^{i+1} =$$
$$= \Delta_D \cup \bigcup_{i=0,1,\ldots} U^i \cup \bigcup_{i=1,2,\ldots} U^i = \bigcup_{i=0,1,\ldots} U^i = U^*. \qquad \square$$

Нехай тепер V – довільне рішення рівняння (3.1). Очевидно, що тоді виконуються включення

$$\Delta_D \subseteq V , \; U \circ V \subseteq V . \qquad (3.2)$$

Індукцією по $k = 0,1,2,\ldots$ покажемо, що виконується включення $U^k \subseteq V$.

Базис. $U^0 = \Delta_D$, залишається застосувати перше включення з (3.2). □

Індуктивний крок. Нехай $U^k \subseteq V$, домножимо обидві частини включення на U зліва. В силу монотонності добутку включення збережеться: $U \circ U^k = U^{k+1} \subseteq U \circ V \subseteq V$ (використовували друге включення з (3.2)). □

Ітак, $U^k \subseteq V$ для всіх $k = 0,1,2,...$, тобто $U^* = \bigcup_{k=0,1,...} U^k \subseteq V$. □■

Аналогічно, як в теоремі 3.4, можна показати, що рефлексивно-транзитивне замикання є найменшим (відносно теоретико-множинного включення) розв'язком більш простого в порівнянні з (3.1) рівняння $X = \Delta_D \cup U \circ X$.

Далі встановимо структуру множини всіх рішень рівняння (3.1), для чого знадобиться лема, яка узагальнює друге включення з (3.2).

Лемма 1 (властивості рішення рівняння (3.1)). Якщо V – рішення рівняння (3.1), то виконується включення

$$U^k \circ V' \subseteq V \qquad (3.3)$$

для всіх $k = 0,1,2,...$ і $V' \subseteq V$. ■

Доведення проведемо індукцією по $k = 0,1,2,...$

Базис проводиться очевидним чином з використанням того, що Δ_D – одиниця за добутком (п. 4 пропозиції 3.2). □

Індуктивний крок проводиться аналогічно доведенню теореми 3.4 (індуктивний крок для встановлення включення $U^k \subseteq V$) з використанням другого включення з (3.2). □ ■

Теорема 3.5 (структура множини всіх рішень рівняння (3.1)). Відношення виду $U^* \cup \bigcup_{k=0,1,...} U^k \circ V$ для будь-якого відношення V є розв'язком рівняння (3.1). Будь-яке рішення має зазначений вид (для відповідного відношення V). ■

Доведення. Нехай V – довільне відношення. Покажемо, що відношення виду, зазначеного в теоремі 3.5, є розв'язком рівняння (3.1). Маємо ланцюжок рівностей:

$$\Delta_D \cup \left(U^* \cup \bigcup_{k=0,1,\ldots} U^k \circ V \right) \cup U \circ \left(U^* \cup \bigcup_{k=0,1,\ldots} U^k \circ V \right) =$$

$$= \Delta_D \cup U^* \cup \bigcup_{k=0,1,\ldots} U^k \circ V \cup \bigcup_{k=0,1,\ldots} U^{k+1} \cup \bigcup_{k=0,1,\ldots} U^{k+1} \circ V =$$

$$= \Delta_D \cup U^* \cup \bigcup_{k=0,1,\ldots} U^k \circ V \cup \bigcup_{k=1,2,\ldots} U^k \cup \bigcup_{k=1,2,\ldots} U^k \circ V = U^* \cup \bigcup_{k=0,1,\ldots} U^k \circ V.$$

У другому переході скористалися дистрибутивністю добутку щодо об'єднань (п. 2 пропозиції 3.2). □

Нехай тепер $\hat{V}$ – довільне рішення рівняння (3.1). Покажемо, що воно має вигляд, зазначений в теоремі, що доводиться; тобто знайдемо відношення V, таке, що $\hat{V} = U^* \cup \bigcup_{k=0,1,\ldots} U^k \circ V$. Покладемо $V = \hat{V} \setminus U^*$ і встановимо необхідну рівність. Маємо ланцюжок рівностей (врахуємо, що U^* найменше рішення рівняння (3.1) відповідно до теореми 3.4):

$$U^* \cup \bigcup_{k=0,1,\ldots} U^k \circ \left(\hat{V} \setminus U^* \right) = U^* \cup \hat{V} \setminus U^* \cup \bigcup_{k=1,2,\ldots} U^k \circ \left(\hat{V} \setminus U^* \right) = \hat{V} \cup \bigcup_{k=1,2,\ldots} U^k \circ \left(\hat{V} \setminus U^* \right).$$

Таким чином, виконуються два включення $\hat{V} \subseteq \hat{V} \cup \bigcup_{k=1,2,\ldots} U^k \circ \left(\hat{V} \setminus U^* \right) \subseteq \hat{V}$ (останнє включення випливає з включення (3.3) в лемі 3.1), з яких і випливає потрібне представлення рішення $\hat{V}$. □ ■

Відзначимо, що замість рівняння (3.1) можна розглядати рівняння вигляду $X = \Delta_D \cup X \cup X \circ U$. При цьому аналоги теорем 3.4-3.5 мають місце. Відзначимо також, що відносно проста структура множини всіх рішень рівняння (3.1) (теорема 3.5), ймовірно, пов'язана з зростанням оператора, відповідного правій частині цього рівняння.

3.7 Основні результати і висновки

На закінчення коротко сформулюємо основні результати і висновки, які вони дозволяють зробити. У розділі на основі властивостей інверсії і добутку

бінарних відносин (пропозиції 3.1, 3.2) встановлені такі властивості рефлексивно-транзитивного замикання бінарного відношення:

1) рефлексивно-транзитивне замикання відношення U є частковим порядком тоді і тільки тоді, коли $\forall i, j = 1,2,... U^i \cap \left(U^{-1}\right)^j \subseteq \Delta_D$ (теорема 3.1);

2) оператор побудови по відношенню його рефлексивно-транзитивного замикання є монотонним, зростаючим (щодо теоретико-множинного включення) і ідемпотентним (пропозиція 3.9);

3) рефлексивно-транзитивне замикання відношення є найменшим (по теоретико-множинному включенню) рефлексивним і транзитивним відношенням, що містить вихідне відношення (теорема 3.2);

4) рефлексивно-транзитивне замикання відношення породжується діагоналлю в часткової параметричної алгебри пар виду A_U (теорема 3.3);

5) рефлексивно-транзитивне замикання U^* є найменшим (по теоретико-множинному включенню) розв'язком параметричного рівняння $X = \Delta_D \cup X \cup U \circ X$ (теорема 3.4);

6) всі рішення рівняння з попереднього пункту мають вигляд $U^* \cup \bigcup\limits_{k=0,1,...} U^k \circ V$ для відповідних відношень V; будь-яке відношення такого виду є розв'язком зазначеного рівняння (теорема 3.5).

Із зазначених результатів випливають висновки:

1) рефлексивно-транзитивне замикання відношення є частковим порядком тоді і тільки тоді, коли граф зв'язків цього відносини є ациклічним;

2) оператор побудови по відношенню його рефлексивно-транзитивного замикання є оператором замикання щодо теоретико-множинного включення (наслідок 3.5)

3) поряд з явним завданням рефлексивно-транзитивного замикання є і три неявних (денотативних): по-перше, як найменшого рефлексивного і транзитивного відношення, що містить вихідне відношення (теорема 3.2); по-друге, як замикання діагоналі в побудованої алгебри пар (теорема 3.3); по-третє, як найменшого рішення рівняння (теорема 3.4) (відзначимо, що оскільки ліва частина характеристичного рівняння вирішувана щодо змінної, то, по суті, мова йде про найменшу нерухому точку оператора, відповідного правій частині цього рівняння, – X a $\Delta_D \cup X \cup U$ o X).

РОЗДІЛ 4. МНОЖИННЕ УСПАДКУВАННЯ ТАБЛИЦЬ З ВИКОРИСТАННЯМ АЛГОРИТМІВ ЛІНЕАРИЗАЦІЇ

4.1 Основні визначення та поняття успадкування таблиць

Класи складаються з методів, полей, властивостей та інших сутностей. Таблиці складаються з колонок. Далі будемо розглядати абстракцію цих сутностей, яку назвемо класом. Клас складається з атрибутів класу які не будемо розділяти на окремі типи. Важливо тільки те, що вони мають унікальні імена в межах одного класу, хоча ці імена можуть і повторюватися в інших класах. Позначимо через C_1, C_2, C_3 класи, множину яких позначимо через Cls. Класи також будемо позначати і іншими буквами латинського алфавіту. Множину всіх атрибутів позначимо через Atr, а окремі атрибути – через $a_1, a_2, \ldots$ Тоді клас буде визначатися як пара (C, atr), де C – ім'я класу, а atr – множина атрибутів $a_1, a_2, \ldots, a_n$. Такий клас будемо записувати у вигляді $C(a_1, a_2, \ldots, a_n)$, або просто C, якщо відомо, про які атрибути йде мова, або конкретний набор атрибутів неважливий.

Між класами існує відношення успадкування, коли один клас успадковує атрибути іншого класу, або, навіть, сукупності класів. Якщо клас C_1 успадковується від класа C_2 то домовимося таку пару записувати у вигляді $C_1 \to C_2$, а, якщо деякий клас C_1 успадковується від кількох класів $C_{21}, C_{22}, \ldots C_{2n}$, то будемо це записувати у вигляді $C_1 \to (C_{21}, C_{22}, \ldots, C_{2n})$ де $C_{2i} \neq C_{2j}, i \neq j$, та $C_1 \neq C_{2i}, i = 1..n$. Порядок, в якому класи $C_{21}, C_{22}, \ldots, C_{2n}$ перераховуються є важливим, тобто, якщо поміняти місцями, наприклад, перший

і другий класи $C_1 \to (C_{22}, C_{21}, \dots, C_{2n})$ то ми отримуємо інше успадкування. Якщо клас успадковується рівно від одного класу, то назвемо це одиночним успадкуванням, якщо ж він успадковується більш ніж від одного класу, то назвемо це множинним успадкуванням. Клас, який успадковується від інших класів, називається нащадком, а клас, від якого успадковуються інші класи, називається предком. Наприклад, в успадкуванні $C_1 \to C_2$ клас C_1 є нащадком класу C_2, а C_2 є предком класу C_1. Так як класи не можуть успадковуватися від самих себе то відношення успадкування є антирефлексивним.

Підмножина відношення успадкування утворює ієрархію класів. Наприклад $\{C \to (A, B),\ E \to (B, D)\}$ задає ієрархію, яка показана на діаграмі, намальованій на рис.4.1

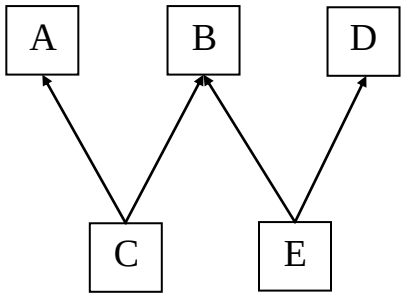

Рис.4.1 Приклад ієрархії класів

Тут клас C є потомком класів A і B, а клас E є потомком класів B, D. Відповідно клас A є предком класу C, клас B є предком класів C, E а клас D є предком класу E.

Але не будь яка підмножина відношення успадкування утворює ієрархію класів. Крім того, в ієрархію можуть включатись одиночні класи, які не є предками, або потомками інших класів. Прикладом підмножини успадкування

яка не утворює ієрархію класів є множина $\{A \to B,\ B \to C,\ C \to A\}$ яка зображена на діаграмі, на рис.4.2

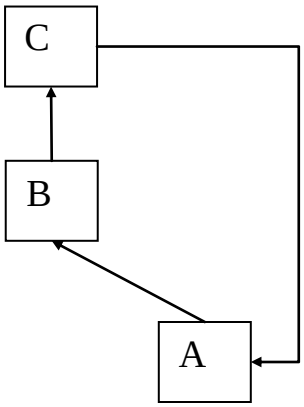

Рис. 4.2 Приклад циклів в успадкуванні

Тобто наслідування не може утворювати цикли. Наступна підмножина теж не утворює ієрархію класів: $\{A \to B,\ A \to C\}$, діаграма цього успадкування зображена на рис.4.3

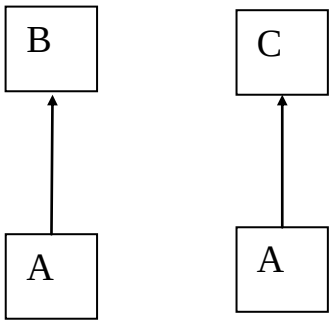

Рис. 4.3 Приклад успадкування, яке не утворює ієрархії класів

В цій множині втрачається порядок, в якому перераховуються предки класу A. Для того щоб зробити з цього ієрархію класів, підмножину треба переписти у вигляді $\{A \to (B, C)\}$, або $\{A \to (C, B)\}$.

Сформулюємо умови, за яких підмножина відношення успадкування утворює ієрархію класів. Нехай є дві пари наслідування класів $A \to (B_1, B_2, \dots, B_k)$ і $C \to (D_1, D_2, \dots, D_l)$. Якщо існує таке B_i, що $C = B_i$, то назвемо такі пари зв'язаними. Послідовність пар наслідування $I_1, I_2, \dots, I_m$ утворює ланцюг успадкування, якщо будь яких два послідовних елемента є зв'язаними, тобто I_j, I_{j+1}, j=1..m-1 є зв'язаними. Якщо, крім того, I_m зв'язана з I_1 то такий ланцюг успадкування утворює цикл. Підмножина відношення наслідування називається ациклічною, якщо будь який ланцюг успадкування з цієї підмножини не утворює цикл.

Для того, щоб уникнути неоднозначності при перерахуванні предків деякого класу, достатньо вимагати, щоб вибрана підмножина відношення наслідування була функціональною, тобто для будь яких пар $a \to b$ і $c \to d$, з цієї підмножини, $a \neq b$.

Таким чином ієрархією класів назвемо пару (S, I) де S – множина класів, а I - підмножина відношення наслідування, яка є ациклічною, функціональною і яка утворюється з класів з множини S. Якщо в S є класи, які не приймають участі в I, тобто не є предками або потомками інших класів, то такі класи будемо називати класами без успадкування.

В будь який ієрархії можна виділити чотири види класів:

- класи які є тільки предками інших класів

- класи, які є тільки потомками інших класів
- класи, які є і предками і потомками інших класів
- класи, які не є ні предками, ні потомками інших класів, тобто класи без наслідування.

Далі, для деякого класу $C(a_1, a_2, \dots, a_k)$ звернення до атрибуту a_i будемо записувати у вигляді $C.a_i$. У випадку, коли клас входить до деякої ієрархії класів, це звернення трактується у розширеному сенсі. А саме, якщо атрибут a відсутній безпосередньо в класі C, то він шукається в предках цього класу. Наприклад, нехай існує наступна ієрархія класів $(\{A(a_1, a_2), B(b_1, b_2), \mathrm{C}(\mathrm{c}_1)\},\ A \to (B, C))$, тоді при зверненні $A.c_1$ атрибут c_1 спочатку шукається між атрибутами класу A, потім між атрибутами класів B і C. Але може бути ситуація, коли атрибут c_1 зустрічається в обох класах B і C. Тоді виникає питання, який з цих двох атрибутів вибрати. В різних мовах програмування така неоднозначність вирішується по різному. Так, можна явним чином вказати необхідний атрибут, наприклад $A.C.c_1$, як це робиться в C++. Але для мов з динамічною типізацією такий метод не є задовільненим, тому що на етапі компіляції система може не знати, від яких класів успадковується клас, в якому виконується звернення до атрибуту. Можна відмовитися взагалі від множинного успадкування, як це робиться, наприклад, в мові Java. Тоді пошук відповідного атрибуту виконується тривіальним чином, коли вибирається атрибут з першого, найбільш спеціалізованого класу. Але заборонити проблему, це не є теж саме що її розв'язати. Тому використовується спеціальний метод, який називається лінеаризацією [41, 40, 31, 22, 38]. Введемо кілька означень. Нехай в ієрархії наслідування приймають участь тільки ті пари наслідування, в яких в якості предка використовується тільки один клас, тобто права частина складається тільки з одного елемента. Такі ієрархії будемо

називати ієрархіями одиночного наслідування класів. Якщо ж є хоча б одна пара з кількістю предків більших за одиницю, то будемо називати її ієрархією множинного наслідування класів. Якщо в ієрархії одиночного наслідування з всіх пар наслідування можна утворити один ланцюг, який містить всі класи, то таку ієрархію назвемо простою ієрархією одиночного наслідування класів.

За аналогією з деревами, клас, який не має жодного потомка будемо називати листом, а клас, який не має жодного предка – коренем. Відмітимо, що для ієрархії одиночного наслідування класів лист має не більш ніж одного предка.

4.2 Одиночне успадкування таблиць

Теорема 4.1: Непуста ієрархія одиночного успадкування класів буде простою тоді і тільки тоді, коли в ній існує рівно один лист.

Доведення: Доведемо необхідність. За визначенням простою ієрархією одиночного наслідування називається ієрархія всі елементи якої можуть утворити один ланцюг. Тоді перший елемент цього ланцюга буде листом ієрархії. Так як всі інші елементи входять в ланцюг то вони не можуть бути листом.

Достатність. В іншу сторону будемо доводити від противного. Припустимо, що виконуються умови теореми, тобто в ієрархії існує рівно один лист, але не можна побудувати ланцюг, який містить всі класи. Візьмемо початкову множину ланцюгів, таку що кожний ланцюг складається з одного класу, і всі класи входять до цієї множини. Потім почнемо з’єднувати ті ланцюги, кінцевий клас яких має предком початковий клас інших ланцюгів, поки це можливо. За припущенням, не існує одного ланцюга, який містить всі класи, тому в результаті отримаємо кілька ланцюгів, причому, один клас, за побудовою, входить тільки в один ланцюг. Так як ті ланцюги, що залишилися, не можна з’єднувати, то це означає, що початок одного ланцюга не може мати потомком кінець іншого ланцюга. Крім того, початок ланцюга не може мати

потомком класи які входять в іншій ланцюг але не є останніми, так як це порушує умову одиночного успадкування. Таким чином перші елементи ланцюгів повинні бути листами. Отримали протиріччя. Теорема доведена.

Для ієрархії одиночного наслідування, яка не є простою, не існує одного ланцюга успадкування, який містить всі класи. Виникає питання, яка мінімальна кількість ланцюгів, що містять всі класи. Відповідь дає наступна теорема

Теорема 4.2. Якщо в ієрархії класів з одиночним успадкуванням існує k листів, то тоді мінімальна потужність множини ланцюгів успадкування які містять всі класи, дорівнює k.

Доведення. Позначимо множину листів через $L = \{L_1, L_2, \dots, L_m\}$, а множину всіх інших класів через $U = \{U_1, U_2, \dots, U_n\}$. Об'єднання $L \cup U$ співпадає з множиною всіх класів ієрархії. Утворимо з елементів L ланцюги успадкування довжиною один. Далі, ітеративно повторюємо наступний крок, поки це можливо: до кожного ланцюгу додаємо предка останнього елемента в ланцюзі, якщо такий предок існує в U і вилучаємо його з U. Зупиняємося, коли ніяких нових елементів до ланцюгів не можна додати.

Так як множина класів, яка утворює ієрархію, скінченна то і виконання цього кроку закінчиться через скінченну кількість ітерацій. Стверджується, що після завершення множина U буде пустою.

Дійсно, припустимо противне, а саме, що множина U є не пустою, але ніяких нових елементів додати до ланцюгів неможливо. Нехай U' - один з таких класів. За визначенням він повинен мати потомка, якого позначимо U''. Цей потомок не може знаходитися в будь якому ланцюзі, тому що тоді і U' повинен

був би бути доданим до цього ланцюга. Тому U'' знаходиться в множені U. З аналогічних міркувань потомок U'' теж повинен знаходитися в множені U. І так далі. Але множина U скінченна, тому деякий потомок не може мати ще одного потомка. Отримали протиріччя.

Побудована множина ланцюгів має потужність k. Так як всі ланцюги починаються з листів, то неможливо побудувати множину меншої потужності

Має місце і аналогічна теорема для коренів.

Теорема 4.3. Якщо в ієрархії класів з одиночним успадкуванням існує k коренів, то тоді мінімальна потужність множини всіх ланцюгів успадкування, які містять всі класи ієрархії не може бути менша ніж k .

Доведення. Дійсно, кожний ланцюг закінчується коренем, або класом, який входить в інший ланцюг. Так як всі класи містяться в множині цих ланцюгів, причому деякі класи можуть включатися в кілька ланцюгів, то загальна кількість ланцюгів не менша ніж кількість коренів.

З теорем 4.2 і 4.3 випливає наступний наслідок

Наслідок 4.1. В ієрархії одиночного наслідування кількість листів завжди більша або дорівнює кількості коренів.

Доведення. За теоремою 4.2 мінімальна кількість ланцюгів які містять всі класи дорівнює кількості листів, а за теоремою 4.3 ця кількість ланцюгів більша або дорівнює кількості коренів. Звідки впливає, що кількість листів завжди більша, або дорівнює кількості коренів. Наслідок доведений.

Перейдемо до опису деяких алгоритмів на ієрархіях одиночного наслідування. В алгоритмах будуть використовуватися такі структури даних, як множина, список та словник. Тут словник розуміється як номінативна множина, тобто множина пар (ім’я, значення), імена яких є унікальними. Якщо позначити

множину імен як Nm, а множину даних як Dat то тоді словник можна визначити як скінченну функцію типу $Nm \to Dat$. Позначимо множину всіх словників як Dic. Операція [] є бінарною операцією типу $Dic \times Nm \to Dat$, яка визначається за формулою $dic[n] = \begin{cases} d, (n,d) \in dic \\ null, \nexists d(n,d) \in dic \end{cases}$, де $dic \in Dic$, $n \in Nm$, $null$ – спеціальне значення, яке трактується як невизначенне значення

Прикладом словника є {book:’ House of Lords and Commons: Poems’, author:’ Ishion Hutchinson’, year:2016}. Словники можуть бути вкладеними.

Для списків операція з’єднання списків буде позначатися через +, а доступ до елементу списку позначимо через [].

Представимо множину класів як словник, ім’ям якого виступають імена класів, а значеннями є множина атрибутів відповідного класу. Відношення успадкування для цих класів теж будемо задавати, як словник, іменами елементів якого є імена класів, а значеннями – список класів предків. Для ієрархії з одиночним успадкуванням, множина предків складається рівно з одного елемента. Для запису алгоритмів буде використовуватися псевдокод, подібний до того, який був описаний в книжці Кормена.

Вкладені оператори будуть записуватися з відступом.

Цикл for <name>, <value> in <dictionary> виконує циклування по словнику. В змінну <name> записується ім’я поточного елементу, а в змінну <value> записується його значення

Цикл for <value> in <collection> виконує циклування по списку або множені. В змінну <value> записується поточне значення

Цикл for <index>=<start number> to <end number> виконує циклування по діапазону чисел

Умовний оператор має вигляд

if <умова>
    <оператори>
elseif <умова>
    <оператори>
…
else
    <оператори>

Наступний алгоритм визначає, чи є ієрархія успадкування ієрархією одиночного успадкування.

Вхід: множина класів set_cls, множина наслідування класів inherit_set

Вихід: True, якщо це ієрархія одиночного успадкування, False в противному випадку

```
def is_single_inheritance(set_cls: Type(dictionary of classes),
                          inherit_set: Type(dictionary of class inheritens))
    for clas_name, set_of_parent_cls in inherit_set
        if size(set_of_parent_cls) > 1
            return False
    return True
```

Наведемо допоміжний алгоритм, який знаходить кількість потомків вказаного класу:

Вхід: множина класів set_cls, множина наслідування класів inherit_set, ім’я класу cls_name

Вихід: True, якщо це ієрархія одиночного успадкування, False в противному випадку

```
def calc_child(set_cls, inherit_set, cls_name)
    child_cnt = 0
```

```
for cls, set_of_parent in inherit_set:
    if cls_name in set_of_parent:
        child_cnt ++
return child_cnt
```

Наступний алгоритм визначає, чи є ієрархія одиночного успадкування простою.

Вхід: множина класів set_cls, множина наслідування класів inherit_set

Вихід: True, якщо ієрархія одиночного наслідування є простою, False в протилежному випадку

```
def is_simple(set_cls, inherit_set)
    leaf_cnt = 0
    for cls_name, attr_set in set_cls
        child_cnt = calc_child(set_cls, inherit_set, cls_name)
        if child_cnt = 0:
            leaf_cnt ++
        if leaf_cnt > 1:
            return False
    return True
```

Далі розглянемо алгоритм, який для ієрархії одиночного успадкування, та класу з цієї ієрархії будує ланцюг максимальної довжини, який починається з вказаного класу

Вхід: множина класів set_cls, множина успадкування класів inherit_set, ім’я класу, починаючи з якого треба побудувати ланцюг name_cls

Вихід: список класів chain_cls, які утворюють ланцюг, максимальної довжини, який починається з вказаного класу.

Алгоритм побудови ланцюга

```
def build_chain(set_cls, inherit_set, name_cls)
    chain_cls = [name_cls]
    next_cls = inherit_set[name_cls]
    while next_cls is not null:
        chain_cls = chain_cls + next_cls
        next_cls = inherit_set[next_cls]
```

Для одиночного успадкування лінеаризація для деякого класу полягає в побудові ланцюга успадкування максимальної довжини, який починається з цього класу. Потім, зліва-направо шукається перший клас, який містить потрібний атрибут. Цей атрибут і буде використовуватися. Якщо ієрархія є простою, то для неї можна відразу побудувати ланцюг успадкування максимальної довжини і шукати потрібний атрибут починаючи з вказаного класу

Розглянемо приклад. Нехай є ієрархія одиночного наслідування {{A(a1, a2, a3), B(b1, b2), C(c1, c2, c3), D(c1), E(c1)}, {A→(B), B→(C), D→(B)}} Ця ієрархія зображена на рис 4.4

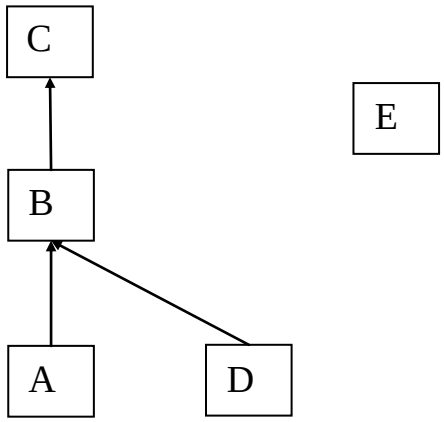

Рис. 4.4 Ієрархія одиночного успадкування

Треба знайти клас, який використовується при зверненні до атрибуту A.c1. Для цього будуємо ланцюг класів максимальної довжини починаючи з A: (A, B, C). Потім шукаємо перший клас, який містить атрибут a1. Таким класом буде C. Значить виклик A.c1 буде замінено на C.c1

Таким чином лінеаризація для одиночного успадкування розміщує класи в порядку від більш спеціалізованого до менш спеціалізованого.

4.3 Множинне успадкування таблиць та класів

Перейдемо до множинного успадкування класів. Ієрархія успадкування класів буде множинним успадкуванням, якщо принаймні в одній парі успадкування $C \rightarrow X$ множина предків X буде мати потужність більше одиниці.

Наприклад наступна ієрархія успадкування класів {{A,B,C,D,E,F,G, H},{A→(B,C,D), B→(E,F), E→(G)}} задає множинне успадкування, яке графічно зображене на рис. 4.5

Для вирішення проблеми вибору найбільш підходящого атрибуту з множини атрибутів з однаковими іменами, застосовується такий же самий підхід, як і для одиночного успадкування, а саме метод лінеаризації, коли рухаємося від більш спеціалізованого класу к менш спеціалізованому, або, від найближчого до того, який знаходиться на більший відстані, якщо під відстанню розуміти довжину ланцюга успадкування.

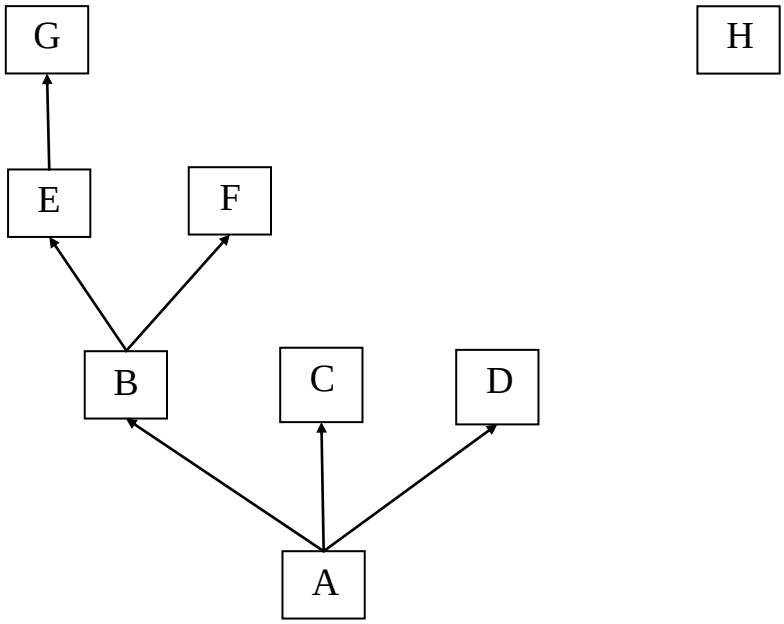

Рис. 4.5 Приклад множинного успадкування класів.

Проблема полягає в тому, що при множинному успадкуванні може бути кілька класів, які знаходяться на однаковій відстані. Так для класу A, зображеному на рис. 4.5 класи B, C, D знаходяться на однаковій відстані, і класи E, F теж знаходяться на однаковій відстані. Тому, для лінеаризації множинного успадкування треба вводити додаткові критерії порівняння класів, крім довжини ланцюгу успадкування. Одним із таких критеріїв є локальний лінійний порядок, заданий на предках класів. А саме, якщо в ієрархії класів існує пара успадкування $A \to (A_1, A_2, \dots, A_n)$, то будимо говорити, що A_i менший за A_j відносно класу A, якщо $i \leq j$ і записувати це у вигляді $A_i \leq_A A_j$. Тобто відношення $\leq_A$ задає лінійний порядок на предках класу A, який назвемо локальним порядком успадкування класів. Алгоритм лінеаризації може враховувати цій локальний порядок успадкування класів. В такому випадку будемо говорити, що лінеаризація зберігає локальний порядок успадкування класів.

Відмітимо, що в цієї ж самій ієрархії може існувати інший клас B для якого виконується протилежна нерівність $A_j \leq A_i$. Зрозуміло, що в такому випадку неможливо зберегти локальний порядок успадкування класів.

Іншою корисною властивістю є монотонність. Змістовно кажучи, монотонність означає, що лінеаризація не порушує відносний порядок слідування класів при переході від деякого класу до будь якого його нащадку. Іншими словами, якщо в лінеаризації деякого класу A клас B знаходиться ліворуч від класу C то і в лінеаризації будь якого нащадку класу A ця властивість зберігається.

Дамо кілька визначень. Операцією лінеаризації будемо називати функцію, яка для ієрархії класів і деякого класу з цієї ієрархії, повертає список класів, який починається з класу, переданого в якості параметру, та всіх його предків, які розміщуються в списку без повторень. Позначимо операцію лінеаризації через L. Отриманий список будемо називати лінеаризацією класу.

Якщо в списку l елемент x знаходиться ліворуч від y то будемо позначати це через $x \prec_l y$. Будемо говорити, що список l_1 включається в список l_2 зі збереженням порядку, якщо всі елементи l_1 входять до списку l_2, причому для будь яких двох елементів x_1 x_2 які входять до l_1 таких що $x_1 \prec_{l_1} x_2$ має місце $x_1 \prec_{l_2} x_2$. Таке включення позначимо через $l_1 \sqsubseteq l_2$.

Операція лінеаризації називається монотонною, якщо для будь якої ієрархії класів I_C , довільного класу A який входить до цієї ієрархії, та будь якого його потомка B, виконується $L(I_C, A) \sqsubseteq L(I_C, B)$.

Монотонність є дуже важливою властивістю лінеаризації. Якщо вона не виконується, то це може приводити до помилок при проектуванні системи класів, які важко виявити.

Таким чином, добре спроектована операція лінеаризації повинна зберігати локальний порядок успадкування та бути монотонною. Легко перевірити, що ці властивості виконуються при лінеаризації ієрархій класів з одинарним успадкуванням. Але для множинного успадкування важко розробити операцію лінеаризації, яка б задовольняла би вказаним властивостям. Далі розглянемо один з найбільш поширених алгоритмів лінеаризації, який називається MRO C4.

Вхід: ієрархія класів I_C, яка задається як словник, кожний елемент якого має вигляд <клас>:<список предків класу>, та клас A. Відмітимо, що список предків класу може бути і порожнім.

Вихід: лінеаризація A відносно I_C, яка задається у вигляді списку класів

Нехай предками класу A в I_C є послідовність класів $[B_1, B_2, \dots, B_k]$

Тоді алгоритм лінеаризації задається наступним чином:

$$L(I_C, A) = [A] + Merge([L(I_C, B_1), L(I_C, B_2), \dots, L(I_C, B_k)])$$

Тут + означає операцію конкатенації списків, причому, якщо один із операндів дорівнює Null, то і результат теж буде Null. Квадратні дужки позначають операцію побудови списку з елементів. Якщо один з елементів Null, то і результат теж буде Null.

Визначимо функцію Merge. Позначимо через $ll = [l_1, l_2, \dots, l_k]$ список лінеаризацій класів, де l_i – лінеаризація деякого класу. Тоді

def *Merge(ll)*

```
    lin = []
    while not IsEmpty(ll)
        item = FindItem(ll)
        if item is Null
            return Null
        lin = lin + [item]
    return lin

def IsEmpty(ll)
    if ll є пустим списком
        return True
    elseif l_i є пустим списком для всіх l_i з ll
        return True
    else
        return False

def FindItem(ll)
    for i=0 to Len(ll) - 1
        first_item = Head(ll[i])
        if first_item not in Tail(l_i)) для всіх l_i з ll
            вилучити first_item з l_i для всіх l_i з ll
            return first_item
    return Null
```

В алгоритмі лінеаризації, функція *FindItem* визначає порядок в якому елементи вибираються з списків лінеаризації батьківських класів. Нехай $ll = [l_1, l_2, \dots, l_k]$ – список списків лінеаризації. Тоді має місце наступна лема.

Лема 4.1 Якщо значення функції *FindItem* визначене на ll, тобто не є значенням Null, то воно буде мінімальним елементом для кожного списку $l_i \in ll$, відносно порядку $\leq_{l_i}$, $i = 1..k$

Доведення: Так як функція повертає перший елемент A з деякого списку, то він буде мінімальним для цього списку. Переконаємося, що він буде мінімальним і для усіх інших списків. Це забезпечує умова

if *first_item* not in *Tail*(l_i)) для всіх l_i з *ll*

яка, перевіряє, що якщо елемент A належить іншим спискам, то він може бути тільки першим елементом в них. Доведення завершене.

Відповідь на питання про монотонність алгоритму дає наступна теорема.

Теорема 4.4 Алгоритм MRO C3 є монотонним

Доведення: Нехай надана ієрархія класів I_C, клас A, та послідовність його предків $(B_1, B_2, \dots, B_k)$. Через $L_A, L_{B_1}, L_{B_2}, \dots, L_{B_k}$ позначимо лінеаризації $L(I_C, A), L(I_C, B_1), L(I_C, B_2), \dots, L(I_C, B_k)$ відповідно. Треба довести, що $L_{B_i} \sqsubseteq L_A$, $i = 1..k$.

Якщо довжина L_{B_i} дорівнює одиниці, то ця вимога виконується автоматично.

Розглянемо ті списки, довжина яких строго більша за одиницю. Тоді для довільних класів E та F, якщо $E \leq_{L_{B_i}} F$ то повинно бути $E \leq_{L_A} F$. Елементи додаються в результуючий список в тому порядку, в якому вони вибираються з списків лінеаризації батьківських класів функцією *FindItem*. Але, як було доведено в лемі 4.1 функція кожний раз повертає мінімальний елемент відносно всіх списків лінеаризації. Тому вона спочатку завжди поверне елемент E, а

потім F, або невизначене значення. Але в останньому випадку і результат роботи алгоритму теж буде невизначеним. Теорема доведена.

Що стосується збереження локального порядку успадкування, то для цього алгоритму ця властивість, взагалі кажучи, не виконується. Дійсно, розглянемо наступний приклад: $I_C = \{\{A, B, C\}, \{A \to (B, C),\ C \to (B)\}\}$, діаграма якого зображена на рис 4.6

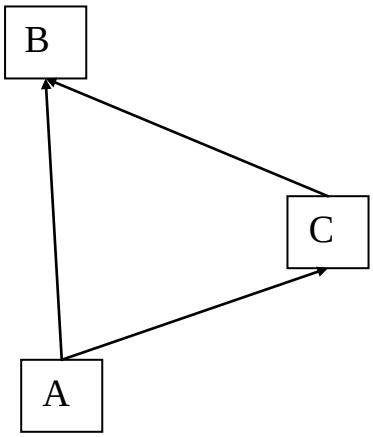

Рис. 4.6 Приклад ієрархії успадкування, яка не зберігає локальний порядок успадкування

Застосуємо алгоритм лінеаризації до класу A

$L(I_C,A) = [A] + Merge([L(I_C,B), L(I_C,C)])$

$lin = []$

$L(I_C,B) = [B]$

$L(I_C,C) = [C, B]$

$ll = [[B], [C, B]]$

$FindItem(ll) = C,\ ll = [[B], [B]]$

$lin = [C]$

$FindItem([[B], [B]]) = B,\ ll = [[], []]$

lin = [C, B]

$Merge$([$L(I_C,B)$, $L(I_C,C)$]) = [C, B]

$L(I_C,A)$ = [A, C, B]

Таким чином локальний порядок успадкування для A порушується

Твердження 4.1. Функція *Merge* завжди завершується

Доведення: В функції використовується один цикл while, тому доведення того, що функція завершується зводиться до доведення того що цій цикл завершується. Умова закінчення циклу є повернення функцією *IsEmpty*(*ll*) значення True, яке означає що $ll = [[\ \ ], [\ \ ], \dots, [\ \ ]]$, або $ll = []$, тобто ll є пустим списком, або списком пустих списків.

Якщо початкове значення ll є пустим списком, або списком пустих списків, то цикл не виконується і функція завершується. Нехай $ll = [l_1, l_2, \dots, l_n]$ і принаймні деякі з l_i не є пустими. Тоді цикл починає виконуватися і викликається функція *FindItem*(*ll*). Якщо вона повертає Null то умовний оператор if *item* is Null завершує цикл. Залишається розглянути випадок, коли *FindItem*(*ll*) повертає не Null. Тоді, як доведено в лемі 4.1 це буде перший елемент одного з списків $l_1, l_2, \dots, L_n$. Але це можливе тільки в тому випадку, коли виконується умовний оператор

if *first_item* not in *Tail*(l_i)) для всіх l_i з *ll*

вилучити *first_item* з l_i для всіх l_i з *ll*

return *first_item*

Перед поверненням в ньому елемент який повертається вилучається зі всіх списків лінеаризації. Тобто довжина принаймні одного списку в *ll* скоротиться на одиницю.

Таким чином, на кожному кроці або *FindItem* повертає Null і цикл завершується, або *FindItem* повертає не Null і, тоді довжина принаймні одного списку з *ll* зменшується на одиницю. В будь якому випадку рано чи пізно цикл завершиться. Максимальна кількість ітерацій дорівнює $|l_1| + |l_2| + \cdots + |l_n|$ кроків. Доведення завершене.

Твердження 4.2 Функція лінеаризації *L*() на коректних вхідних даних завжди завершується.

Доведення: Якщо клас який лінеаризується не має батьків, то функція *Merge*() викликається на пустому списку, і, як було показано в попередньому твердженні, вона завершується. Якщо список батьків не пустий, то функція лінеаризації рекурсивне викликається для кожного батька і так далі. Фактично функція викликається для кожного класу, який знаходиться в одному з ланцюгів, які починаються в заданому класі. причому для деяких класів вона може викликатися кілька разів, якщо клас знаходиться в кількох ланцюгів. Так як ієрархія класів ациклічна, то і рекурсивні виклики теж закінчяться. Всі інші функції, які виконуються в алгоритмі теж завершуються. Тому алгоритм теж завершується на коректних вхідних даних. Доведення закінчене.

Наведемо реалізацію алгоритму MRO C3 на мові програмування Python:

```
def linearization(ic, a):
    parents = ic.get (a, [])
    lin_tail = merge ([linearization(ic, p) for p in parents])
    if lin_tail is None:
        return None
    return [a] + lin_tail



def merge(ll):
    lin = []
```

```
    while not is_empty(ll):
        item = find_item(ll)
        if item is None:
            return None
        lin = lin + [item]
    return lin


def is_empty(ll):
    for i in range(0, len(ll)):
        if len(ll[i]) != 0:
            return False

    return True


def find_item(ll):
    for i in range (0, len (ll)):
        if len (ll[i]) == 0:
            continue
        first = ll[i][0]
        is_good = True
        for j in range (0, len (ll)):
            if first in ll[j][1:]:
                is_good = False
                break
        if is_good:
            for k in range (0, len (ll)):
                if len (ll[k]) > 0 and first in ll[k]:
                    ll[k].remove (first)
```

return first

return None

4.4 Приклади множинного успадкування таблиць

Приклад 1. Нехай задана наступна ієрархія таблиць: I_C={{T, F, E, D, C, B, A}, {F→T, E→T, D→T, C→(F, D), B→(E, D), A→(C, B)}}.

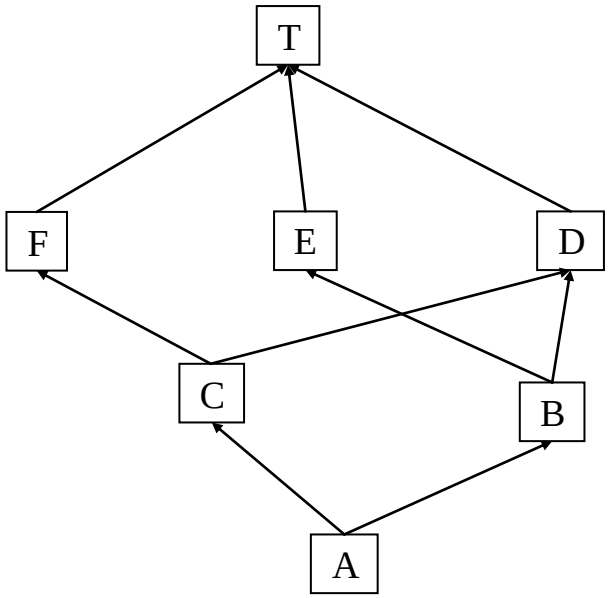

Рис 4.7

Для таблиць F, E та D лінеаризація будується тривіальним чином. Побудуємо лінеаризацію для C та F.

L(C) = [C] + Merge ([[F, T], [D, T]])

Виконуємо Merge([[F,T], [D, T]])

На початку циклу маємо ll = [[F,T], [D, T]], lin = []. Умова циклу виконується. В циклі викликає FindItem([[F,T], [D, T]]) яка повертає F

Після першого кроку будемо мати lin = [F], ll = [[T], [D, T]]

На другому кроці FindItem([[T], [D, T]]) повертає D, відповідно lin = [F, D], ll = [[T], [T]]

На третьому кроці FindItem([[T], [T]]) повертає T, відповідно lin = [F, D, T], ll = [[], []]

Так як IsEmpty([[], []]) = True то цикл завершується і Merge повертає [F, D, T].

Таким чином L(C) = [C] + Merge([[F,T], [D, T]]) = [C] + [F, D, T] = [C, F, D, T]

Аналогічним чином обчислюємо L(B) = [B] + Merge([[E, T], [D, T]])

Отримуємо L(B)=[B, E, D, T]

Нарешті обчислимо L(A) = [A] + Merge([[C, F, D, T], [B, E, D, T]])

На початку циклу маємо ll = [[C, F, D, T], [B, E, D, T]], lin = []. Умова циклу виконується. В циклі викликається функція FindItem([[C, F, D, T], [B, E, D, T]]), яка повертає C. Після першого кроку циклу змінні мають наступні значення: ll = [[F, D, T], [B, E, D, T]], lin = [C]. При виконанні другого кроку циклу функція FindItem([[F, D, T], [B, E, D, T]]) поверне значення F, а змінні отримають значення ll = [[D, T], [B, E, D, T]], lin = [C, F]. На третьому кроці функція FindItem([[D, T], [B, E, D, T]]) поверне значення B, а змінні отримають значення ll = [[D, T], [E, D, T]], lin = [C, F, B]. На четвертому кроці функція FindItem([[D, T], [E, D, T]]) поверне значення E, а змінні отримають значення ll = [[D, T], [D, T]], lin = [C, F, B, E]. Після п’ятого і шостого кроків змінні отримають значення ll = [[], []], lin = [C, F, B, E, D, T]. Умова циклу стає хибною і функція Merge повертає значення [C, F, B, E, D, T]. Результат роботи алгоритму буде L(A) = [A] + Merge([[C, F, D, T], [B, E, D, T]]) = [A] + [C, F, B, E, D, T] = [A, C, F, B, E, D, T].

Приклад 2. Нехай задана наступна ієрархія таблиць: I_C={{T, F, E, D, C, B, A}, {F→T, E→T, D→T, C→(D, F), B→(E, D), A→(B, C)}}. Вона відрізняється

від попереднього прикладу тільки локальним порядком успадкування для C, B, A.

L(A) = [A] + Merge(L(B), L(C))

L(B) = [B] + Merge([L(E), L(D)]) = [B] + Merge([E, T], [D, T])

L(C) = [C] + Merge([L(D), L(F)]) = [C] + Merge([[D, T], [F, T]])

Обчислимо Merge([[E, T], [D, T]]). На початку циклу ll = [[E, T], [D, T]], lin=[]. Умова циклу істина, отже виконуємо перший крок циклу. Функція FindItem([[E, T], [D, T]]) поверне значення E. Нові значення змінних будуть ll = [[T], [D, T]], lin=[E]. Умова циклу істинна, тому виконуємо другий крок. Функція FindItem([[T], [D, T]]) поверне значення D. Значення змінних стануть ll = [[T], [T]], lin=[E, D]. Умова циклу виконується, отже знову викликається функція FindItem([[T], [T]]) яка поверне значення T. Нові значення змінних будуть ll = [[], []], lin=[E, D, T]. Умова циклу хибна, результатом буде список лінеаризації [E, D, T]. Маємо L(B) = [B, E, D, T].

Аналогічним чином обчислимо Merge([[D, T], [F, T]]). Результатом буде список лінеаризації [D, F, T]. Маємо L(C) = [C, D, F, T]

Нарешті розрахуємо L(A) = [A] + Merge(L(B), L(C)) = [A] + Merge([[B, E, D, T], [C, D, F, T]]). Виконуємо функцію Merge([[B, E, D, T], [C, D, F, T]]). Початковим станом змінних буде ll = [[B, E, D, T], [C, D, F, T]], lin=[]. Умова циклу виконується, тому викликаємо FindItem([[B, E, D, T], [C, D, F, T]]), яка поверне B. Новим станом змінних буде ll = [[E, D, T], [C, D, F, T]], lin=[B]. На наступному кроці циклу FindItem([[E, D, T], [C, D, F, T]]) поверне E. Змінні приймуть значення ll = [[D, T], [C, D, F, T]], lin=[B, E]. Так як ll не є пустим списком, то цикл продовжується. Функція FindItem([[D, T], [C, D, F, T]]) не може вибрати D так як він існує в середині другого списку. Тому вибирається C. Стан змінних становиться ll = [[D, T], [D, F, T]], lin=[B, E, C]. Умова циклу (not IsEmpty(ll)) виконується, тому переходимо до наступного кроку циклу. На

наступному кроці FindItem([[D, T], [D, F, T]]) поверне D, так як воно знаходиться на початку всіх списків. Стан змінних становиться ll = [[T], [F, T]], lin=[B, E, C, D]. Умова циклу (not IsEmpty(ll)) виконується, тому переходимо до наступного кроку циклу. На наступному кроці циклу функція FindItem([[T], [F, T]]) поверне F. Після цього новим станом змінних буде ll = [[T], [T]], lin=[B, E, C, D, F]. Умова циклу (not IsEmpty(ll)) виконується, тому переходимо до наступного кроку циклу. На останньому кроці функція FindItem([[T], [T]]) поверне T і станом змінних буде ll = [[], []], lin=[B, E, C, D, F, T]. Умова циклу (not IsEmpty(ll)) становиться хибною, тому функція виходить з циклу і повертає список лінеаризації [B, E, C, D, F, T]. Кінцевим результатом обчислення буде:

L(A) = [A] + Merge(L(B), L(C)) = [A] + [B, E, C, D, F, T] = [A, B, E, C, D, F, T]

Приклад 3. Нехай задана наступна ієрархія таблиць I_C={{A, B, C, D, E, F, G}, {A→(B, C, D), B→E, C→E, D→G, E→F, E→G}}, яка зображена на рис 4.8. Треба знайти лінеаризацію для A.

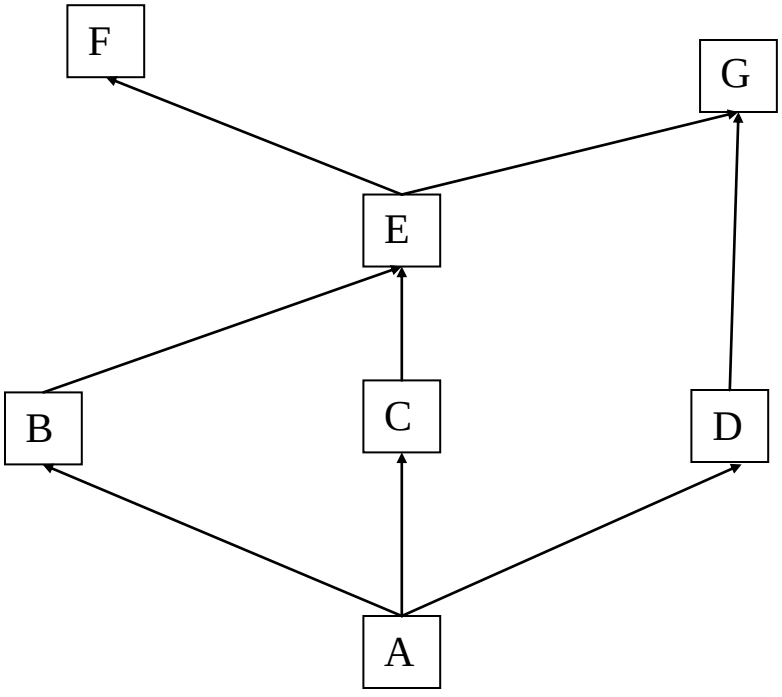

Рис. 4.8

L(A) = [A] + Merge (L (B), L(C), L (D))

L (B) = [B] + Merge (L (E)) = [B] + Merge ([E, F, G]) = [B, E, F, G]

L(C) = [C] + Merge (L (E)) = [C] + Merge ([E, F, G]) = [C, E, F, G]

L (D) = [D] + Merge (L (G]) = [D, G]

Тоді L(A) = [A] + Merge ([B, E, F, G], [C, E, F, G], [D, G])

На початку виконання функції Merge змінні мають наступний стан:

ll = [[B, E, F, G], [C, E, F, G], [D, G]], lin=[]. Так як (not IsEmpty([[B, E, F, G], [C, E, F, G], [D, G]])) є істинним, починаємо виконувати перший крок циклу. Функція FindItem([[B, E, F, G], [C, E, F, G], [D, G]]) поверне B. Змінні отримають нові значення ll = [[E, F, G], [C, E, F, G], [D, G]], lin=[B]. Умова циклу виконується, тому переходимо до другого кроку. Функція FindItem([[E, F, G], [C, E, F, G], [D, G]]) не може повернути E, тому що E зустрічається у нутрі іншого списку. Буде повернуте C. Новим станом змінних буде ll = [[E, F, G], [E, F, G], [D, G]], lin=[B, C]. Умова (not IsEmpty([[E, F, G], [E, F, G], [D, G]])) є істинною, тому переходимо до наступного кроку циклу. Функція FindItem([[E, F, G], [E, F, G], [D, G]]) поверне E. Змінні отримають наступні значення ll = [[F, G], [F, G], [D, G]], lin=[B, C, E]. Умова (not IsEmpty([[F, G], [F, G], [D, G]])) буде істинною, тому переходимо до наступного кроку циклу. Функція FindItem([[F, G], [F, G], [D, G]]) поверне F. Змінні отримають наступні значення ll = [[G], [G], [D, G]], lin=[B, C, E, F]. Умова циклу (not IsEmpty([[G], [G], [D, G]])) буде істинною, тому переходимо до наступного кроку циклу. Функція FindItem([[G], [G], [D, G]]) поверне D. Змінні отримають наступні значення ll = [[G], [G], [G]], lin=[B, C, E, F, D]. Умова (not IsEmpty([[G], [G], [G]])) знову буде істинною, тому переходимо до наступного кроку циклу. Функція FindItem([[G], [G], [G]]) поверне G. Змінні отримають наступні значення ll = [[], [], []], lin=[B, C, E, F, D G]. Умова (not IsEmpty([[], [], []])) буде хибною, цикл завершено. Функція повертає значення [B, C, E, F, D, G].

Тоді L(A) = [A] + Merge ([B, E, F, G], [C, E, F, G], [D, G]) = [A] + [B, C, E, F, D, G] = [A, B, C, E, F, D, G]

Приклад 4. Нехай задана наступна ієрархія таблиць I_C={{A, B, C, D, E, F, G}, {A→(B, C, D), B→(F, E), C→(F, E, G), D→(G, E)}, яка зображена на рис 4.9. Треба знайти лінеаризацію для A. Тобто обчислити вираз

L (A) = [A] + Merge ([L(B), L(C), L(D))

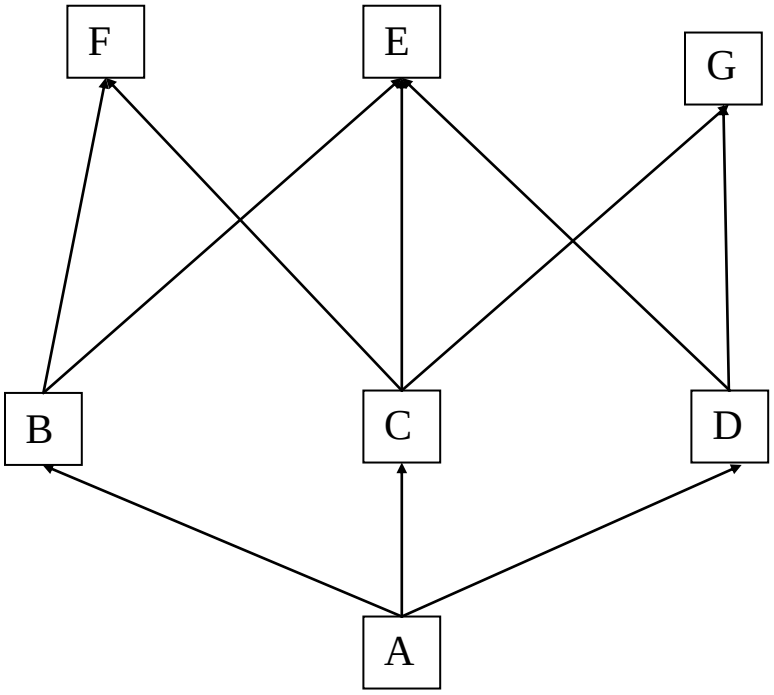

Рис. 4.9

Маємо

L(B) = [B] + Merge([L(F), L(E)]) = [B] + Merge([F, E]) = [B, F, E]

L(C) = [C] + Merge([L(F), L(E), L(G)]) = [C] + Merge([F, E, G]) = [C, F, E, G]

L(D) = [D] + Merge([L(G), L(E)]) = [D] + Merge([G, E]) = [D, G, E]

Тоді

L (A) = [A] + Merge ([L(B), L(C), L(D)) = [A] + Merge([B, F, E], [C, F, E, G], [D, G, E])

Розрахуємо Merge([B, F, E], [C, F, E, G], [D, G, E]).

Початковим станом буде ll = [[B, F, E], [C, F, E, G], [D, G, E]], lin=[]

Так як (not IsEmpty([[B, F, E], [C, F, E, G], [D, G, E]])) повертає істинну, то починаємо виконувати перший крок циклу. Функція FindItem([[B, F, E], [C, F, E, G], [D, G, E]]) поверне B. Новий стан змінних буде ll = [[F, E], [C, F, E, G], [D, G, E]], lin=[B]. Вираз (not IsEmpty([[F, E], [C, F, E, G], [D, G, E]])) повертає істинну. Виконуємо наступний крок циклу. Функція FindItem([[F, E], [C, F, E, G], [D, G, E]]) поверне C. Змінні будуть мати наступні значення ll = [[F, E], [F, E, G], [D, G, E]], lin=[B, C]. Вираз (not IsEmpty([[F, E], [F, E, G], [D, G, E]])) повертає істинну. Виконуємо наступний крок циклу. Функція FindItem([[F, E], [F, E, G], [D, G, E]]) поверне F. Змінні приймуть наступні значення ll = [[E], [E, G], [D, G, E]], lin=[B, C, F]. Вираз (not IsEmpty([[E], [E, G], [D, G, E]])) повертає істинну. Виконуємо наступний крок циклу. Функція FindItem([[E], [E, G], [D, G, E]]) поверне D, так як E знаходиться в середині останнього списку. Змінні приймуть значення ll = [[E], [E, G], [G, E]], lin=[B, C, F, D]. Вираз (not IsEmpty([[E], [E, G], [G, E]])) повертає істинну. Виконуємо наступний крок циклу. Функція FindItem([[E], [E, G], [G, E]]) не може повернути E так як воно знаходиться у нутрі останнього списку, але вона також не може повернути G, так як воно знаходиться у нутрі другого списку. Тому вона повертає NULL. Згідно з алгоритмом, в такому випадку функція Merge теж повертає NULL. Результатом обчислення всього виразу L (A) теж буде NULL, що означає, що лінеаризація неможлива.

В даному випадку це сталося, тому, що в успадкуваннях C→(F, E, G), D→(G, E) порядок слідування E та G відрізняється. Якщо б було D→(E, G) то тоді лінеаризація була би можливою. А саме, функція FindItem([[E], [E, G], [E, G]]) повернула би E. Тоді змінні мали би наступні значення ll = [[], [, G], [G]],

lin=[B, C, F, D, E]. На наступному кроці циклу функція FindItem([[], [G], [G]]) повернула би G. Змінні мали би наступні значення ll = [[], [], []], lin=[B, C, F, D, E, G]. Вираз (not IsEmpty([[], [], []])) став би хибним і Merge повернула би список [B, C, F, D, E, G]. Тоді значення всього виразу було би L (A) = [A] + Merge ([L(B), L(C), L(D)) = [A] + Merge([B, F, E], [C, F, E, G], [D, E , G]) = [A] + [B, C, F, D, E, G] = [A, B, C, F, D, E, G]

ВИСНОВКИ

Головним результатом дисертації є розширення табличної алгебри семантичних функцій мови SQL одиночним та множинним успадкуванням таблиць з автоматичним розв'язанням конфліктів імен атрибутів за допомогою алгоритму лінеаризації, відомого під назвою MRO C3. Це розширення розв'язує важливу задачу побудови математичної моделі об'єктно-реляційних баз даних, що має велике теоретичне та практичне значення для розробки перспективних комп'ютерних інформаційних систем.

Основні результати включають в себе:

1. Для більш точного опису семантики ядра мови SQL та більшої зручності у використанні замість старих визначень в табличній алгебрі семантичних функцій SQL запропоновані нові визначення операцій внутрішнього та зовнішнього з'єднання, а також теоретико множинних операцій.

2. Доведена теорема, яка визначає умови, при яких семантичні функції задають точну семантику оператора ORDER BY. Це показало, що повна семантика ORDER BY може бути задана лише недетермінованими функціями.

3. Доведені теореми про характеристичні ознаки одиночного успадкування. На їх базі розроблені алгоритми перевірки, чи є ієрархія

успадкування одиночним чи множинним успадкуванням, та чи є ієрархія одиночного успадкування простою.

4. Формально визначені такі важливі властивості методу лінеаризації ієрархії таблиць, як монотонність та збереження локального порядку успадкування, що дає можливість оцінити той чи інший метод лінеаризації та визначити, наскільки він є безпечним.

5. Побудовано алгоритм MRO C3 для автоматичного вирішення конфлікту співпадаючих імен атрибутів при множинному успадкуванні таблиць. Це дозволило перевіряти властивості алгоритму на строгому математичному рівні. Зокрема доведено теореми про такі властивості алгоритму, як монотонність та його завершуваність. Алгоритм розроблювався на псевдокоді, що зробило його незалежним від мови програмування. Додатково він був реалізований на мові програмування Python. Проведене тестування програмної реалізації також підтвердило коректність доведених теорем.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Brona J. Compositional Approach to the Semantisc of SQL / J. Brona, D. Buy, S. Zagorsky, S. Polyakov // Fifth International Conference "Information Theories & Applications" (September, 1-15, 2000, Varna, Bulgaria). Abstracts. – Sofia: FOI-COMMERE. – 2000. – P. 33-34.

2. Brona J. Compositional Approach to the Semantisc of SQL / J. Brona, D. Buy, S. Zagorsky, S. Polyakov // Information Theories & Applications. – 2001. – V. 8, N. 3. – P. 133-142.

3. Brona J. Compositional Semantisc of SQL / J. Brona, D. Buy, S. Zagorsky, S. Polyakov // Proc. of the Fourth International Scientific Conference "Electronic Computers and Informations’2000". – Kosice, 2000. – P. 287-292.

4. C.J.Date SQL and Relational Theory. – O'Reilly Media, Inc., 584 P., 3rd edition, 2015

5. Chauhan C. PostgreSQL Cookbook. / Chitij Chauhan. – PACKT publishing, 2015. – 286 p.

6. Codd E. F. A Data Base Sublanguage Founded on the Relational Calculus / E. F. Codd // Proc. of 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control. – N.-Y.: ACM. – 1971. – P. 35-68.

7. Codd E. F. Further Normalization of Data Base Relational Model / E. F. Codd // Data Base Systems. – N.-Y.: Prentice-Hall. – 1972. – P. 33-64.

8. Codd E. F. Normalized Data Base Structure: A Brief Tutorial / E. F. Codd // Proc. of 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control. – N.-Y.: ACM. – 1971. – P. 1-17.

9. Codd E. F. Relational Completeness of Data Base Sublanguages / E. F. Codd // Data Base Systems. – N.-Y.: Prentice-Hall. – 1972. – P. 65-93.

10. Codd E. F. Relational Database: A Practical Foundation for Productivity / E. F. Codd // Comm. of ACM. – V. 25, N 2. – 1982. – P. 109-117.

11. Codd E.F. Data Models in Database Management / E. F. Codd. Proc. Workshop in Data Abstraction, Databases, and Conceptual Modelling (Michael L. Brodie and Stephen N. Zilles, eds.), Pingree Park, Colo. (June 1980): ACM SIGART Newsletter No. 74 (January 1981); ACM SIGMOD Record 11(2), February 1981; ACM SIGPLAN Notices 16(1), January 1981.

12. Codd E. F. A Relational Model of Data for Large Shared Data Banks / E. F. Codd // Comm. of ACM. – 1970.– Vol. 13. – № 6. – P. 377-387.

13. Codd E. F. The Relational Model for Database Management: Version 2 / E. F. Codd. – Addison-Wesley, 1990. – 541 p.

14. Conolly T. Database Systems. Apractical approach to design, implementation and management. / T. Connolly, C. Begg. – PEARSON, 2015. – 1442 p.

15. Coronel C. Database Systems. Design, Implementation, and Management. / C. Coronel, S. Morris. – CENGAGE Learning, 2016. – 818 p.

16. D. Buy, J. Karam, S. Kompan, S. Polyakov. Linearization algorithms CLOS and LOOPS of the classes in programming languages: the formal definitions // 13th International Scientific Conference on Informatics. – Poprad, Slovakia, 18-20 Nov., 2015. – P. 63-66 (Print ISBN: 978-1-4673-9867-1, DOI 10.1109/Informatics.2015.7377809).

17. D.Maier. The Theory of Relational Databases, Computer Press, Rockville, Md., 637 P., 1983

18. Date C. J. Database in Depth: Relational Theory for Practitioners / C. J. Date. – California: O‟ Reilly, 2005. – 208 p.

19. Date C. SQL and Relational Theory. / C.J. Date. – O'Reilly, 2015. – 584 p.

20. Eisenberg A. SQL:2003 has been published / Andrew Eisenberg, Jim Melton, Krishna Kulkarni, Jan-Eike Michels, Fred Zemke // SIGMOD Record, Vol. 33, No. 1. – March 2004. – P. 7-15.

21. Elmasri R. Fundamental of Database Systems: [3rd Edition] / R. Elmasri, S. Navathe. – Addison-Wesley, 2000. – 893 p

22. Guido van Rossum. Method Resolution Order – [Электронный ресурс]. – Режим доступа: http://python-history.blogspot.com/2010/06/method-resolution-order.html.

23. H.Garcia-Molina, J.D.Ullman, J.Widom Database Systems. Pearson Education Inc., 2nd edition, 1203 P., 2009

24. Haan L. Applied Mathematics for Database Professionals. / L.de Haan, T.Koppelaars. – APRESS, 2007. – 406 p.

25. Harrington J. SQL Clearly Explained. / Jan.L. Harrington. – Elsevier, 2010. – 481 p.

26. J. Brona, D. Buy, S. Zagorsky, S.Poliakov Compositional Semantisc of SQL. Proc. of the Fourth International Scientific Conference "Electronic Computers and Informations'2000". – Kosice, 2000. – P. 287-292.

27. Kaur M. PostgreSQL Development Essentials. / M.Kaur, B.Shaik. – PACKT publishing, 2016. – 205 p.

28. Learning PostgreSQL. / S.Juba, A.Vannahme, A.Volkov. – PACKT publishing, 2015. – 464 p.

29. Melton J. SQL:1999 Understanding Relational Language Components / Jim Melton and Alan Simon. – Academic Press, 2002. – 895 p.

30. Meltz D. An Introduction to IMS: Your Complete Guide to IBM's Information Management System / D. Meltz, R. Long, M. Harrington, R. Hain, G. Nicholls. – IBM Press, 2004. – 592 p.

31. Michele Simionato. The Python 2.3 Method Resolution Order. – [Электронный ресурс]. – Режим доступа: https://www.python.org/download/releases/2.3/mro/.

32. Mohammed K. Join Operations in Relational Databases with Automatic Attributes Renaming / Poliakov S, Buy D, Mohammed K., Israa Jasim AL.kalafa // Scholars Journal of Engineering and Technology. – 2017. –№5(6) - P.254-257. http://saspublisher.com/sjet-56/

33. Mohammed K. Linearization algorithms CLOS and LOOPS of the classes in programming languages: the formal definitions / K. Mohammed, D. Buy, J. Karam,

S. Kompan, S. Polyakov.// 13th International Scientific Conference on Informatics. – Poprad, Slovakia, 18-20 Nov., 2015. – P. 63-66 (Print ISBN: 978-1-4673-9867-1, DOI 10.1109/Informatics.2015.7377809). http://ieeexplore.ieee.org/document/7377809/

34. Mohmmed K. Математические основы множественного наследования в ООП /K. Mohmmed, Буй Д.Б., Шишацкая Е.В., Fabunmi S. // Четвёртая конференция «КОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ В НАУКОЕМКИХ ТЕХНОЛОГИЯХ» (КМНТ-2016). – Харьков, 26-31 мая 2016 г (http://www.dsmmph.org.ua/kmnt.html)

35. Mohmmed K. Mathematical Foundations of Multiple Inheritance: Reflexive-transitive Closure of the Binary Relations / Dmytro Buy, Olena Shyshatska, Sunmade Fabunmi, Karam Mohammed // Perspective Technologies and Methods in MEMS Design (MEMSTECH), 2016 XII International Conference on Perspective Technologies and Methods in MEMS Design (MEMSTECH). – 2016. – C.192-195. (http://ieeexplore.ieee.org/document/7507540/)

36. Negri M. Formal semantics of SQL queries./ M. Negri, G. Pelagatti, L. Sbattella / ACM Trans. Database Syst. 16(3), 513-534 (1991)

37. Obe R. PostgreSQL Up & Running. / R. Jbe, L.Hsu. – O'Relly, 2015. – 231 p.

38. Par Gaël Pegliasco. Python Tutorial: Understanding Python MRO – Class search path. – [Электронный ресурс]. – Режим доступа: http://makina-corpus.com/blog/metier/2014/python-tutorial-understanding-python-mro-class-search-path.

39. Polyakov S. Recursive queries in SQL and their generalization – systems of recursive queries / S. Polyakov, D. Buy // Proceeding of CSE 2010 International Scientific Conference on Computer Science and Engineering, September 20-22, 2010, Koice – Stara Lubovna, Slovakia. – P. 252-257.

40. R. Ducournau, M. Habib, M. Huchard, M.L. Mugnier. Monotonic conflict resolution mechanisms for inheritance // Proceeding OOPSLA '92 conference proceedings on Object-oriented programming systems, languages, and applications. – 1992. – P.16-24.

41. R. Ducournau, M. Habib, M. Huchard, M.L. Mugnier. Proposal for a Monotonic Multiple Inheritance Linearization // OOPSLA '94 Proceedings of the ninth annual conference on Object-oriented programming systems, language, and applications. – 1994. – P. 164-175.

42. Schonig H. Mastering PostgreSQL 9.6. /Hans-Jurgen Schonig. – PACKT publishing, 2017. – 881 p.

43. Silbeschatz A. Database System Concepts: [6th Edition] / A. Silbeschatz, H. Korth, S. Sudarshan. – McGraw-Hill, 2011. – 1376 p.

44. Viescas J. Effective SQL. /J.L.Viescas, D.J.Steele, B.G.Clothier. – Addison-Wesley, 2017. – 546 p.

45. Viescas J.L. SQL Queries for Mere Mortals: a hands-on guide to data manipulation in SQL [2nd edition] / J.L. Viescas, M.J. Hernandez. – Massachusetts: "Addison-Wesley", 2007. – 631 p.

46. Weitz K. HoTTSQL: proving query rewrites with univalent SQL semantics./ Shumo Chu, Konstantin Weitz, Alvin Cheung, Dan Suciu / Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, P. 510-524 (Barcelona, Spain — June 18 - 23, 2017)

47. Андон Ф., Резниченко В. Язык запросов SQL. Учебный курс / Ф. Андон, В. Резниченко. – СПб.: Питер, Киев: Издательская группа BHV, 2006. – 416 с.

48. Архангельский А.В. Канторовская теория множеств.˜ М.: Издательство Московского университета, 1988.˜ 111 с.

49. Басараб И. А. Композиционный подход к построению языков манипулирования данными / И. А. Басараб, Б. В. Губский // Программирование. – 1988. – № 6. – С. 59-70.

50. Басараб І.А. Композиційні бази даних / І. А. Басараб, М. С. Нікітченко, В. Н. Редько. – К.:Либідь, 1992. – 192 с.

51. Биркгоф Г. Теория решеток. – Москва: Наука, 1984. – 568 с.

52. Боуман Дж. С. Практическое руководство по SQL / Дж. С. Боуман, С. Л. Эмерсон, М.Дарновски. – Киев: Диалектика, 1997. – 320 с.

53. Брона Ю. Й. Композиційна семантика SQL-подібних мов: агрегатні функції / Ю. Й. Брона, Д. Б Буй., С. П. Загорський, С. А. Поляков // Вісник Київського університету. Сер. фіз.-мат. науки. – 2000. – Вип. 1. – С. 178-192.

54. Брона Ю. Й. Композиційна семантика SQL-подібних мов: групування, маніпулювання даними, приклади / Ю. Й. Брона, Д. Б. Буй, С. П. Загорський, С. А. Поляков // Вісник Київського університету. Сер. фіз.-мат. науки. – 2000. – Вип. 2. – С. 177-185.

55. Брона Ю. Й. Композиційна семантика SQL-подібних мов: операції з’єднання / Ю. Й. Брона , Д. Б Буй., С. А. Поляков // Вісник Київського університету. Сер. фіз.-мат. науки. – 1999. – Вип. 4. – С. 100-104.

56. Брона Ю. Й. Композиційна семантика агрегатних функцій SQL-подібних мов / Ю. Й. Брона, Д. Б. Буй, С. П. Загорський, С. А. Поляков // Проблемы программирования. – 2000. – № 1-2. – С. 554-565.

57. Буй Д. Б. Теорія програмних алгебр композиційного типу та її застосування: дисертація доктора фізико-математичних наук: 01.05.03 – математичне та програмне забезпечення обчислювальних машин і систем / Д. Б. Буй. – Київ, 2002. – 365 с.

58. Буй Д.Б. Властивості відношення конфінальності та устрій множини часткових функцій / Д.Б. Буй, Н.Д. Кахута // Вісник Київського університету. Сер.: фіз.-мат. науки. – 2006. – Вип. 2. – С. 125-135

59. Буй Д.Б. Властивості теоретико-множинних конструкцій повного образу та обмеження / Д.Б.Буй, Н.Д.Кахута // Вісник Київського університету. Серія: Фізико-математичні науки. – 2005. – Вип. 2. – С. 232-240.

60. Буй Д.Б. Теория програмних алгебр композиційного типу та її застосування: дисертація доктора фізико-математичних наук: 01.05.03 – математичне та програмне забезпечення обчислювальних машин та систем / Буй Дмитро Борисович. – Київ, 202. – 365 с.

61. Буй Д. Трехзначные логики Клини и трехэлементные цепи / Д. Буй, Е. Шишацкая // International Book Series «Information Science & Computing» №. 1. Supplement to the International Journal «Information Technologies & Knowledge». – 2008.– V. 2. – P. 165-172.

62. Бурбаки Н. Теория множеств. М.: Мир, 1965. 455 с.

63. В. Н. Редько Реляційні бази даних: табличні алгебри та SQL-подібні мови / В. Н. Редько, Ю. Й. Брона, Д. Б. Буй, С. А. Поляков. – Київ: Видавничий дім «Академперіодика», 2001. – 198 с.

64. Вагнер В.В. Теория отношений и алгебра частичных отображений // Теория полугрупп и ее приложения (Сборник статей). – Саратов: Изд-во Саратовского ун-та, 1965. – Вып. 1. – С. 3-178.

65. Глушков В.М. Алгебра. Языки. Программирование./В.М.Глушков, Г.Е.Цейтлин, Е.Л.Ющенко. – Киев : Наукова думка, 1978. — 318 с.

66. Грабер М. Справочное руководство по SQL / М. Грабер. – Москва: ЛОРИ, 1997. – 291 с.

67. Дейт К. Введение в системы баз данных / К. Дейт. – Киев: Диалектика, 1998. – 781 с.

68. Дейт К. Руководство по реляционной СУБД DB2 / К. Дейт. – Москва: Финансы и Статистика, 1988. – 320 с.

69. Емеличев В. А. Лекции по теории графов / В. А. Емеличев, О. И. Мельников, В. И. Сарванов, Р. И. Тышкевич. – Москва: Наука, 1990. – 384 с.

70. Катленд Н. Вычислимость. Введение в теорию рекурсивных функций / Н. Катленд– М.: Мир, 1983. – 256 с.

71. Кахута Н.Д. Застосування теоретико-множинних конструкцій повного образу, обмеження, конфінальності та сумісності в табличних базах даних: дисертація кандидата фізико-математичних наук: 01.05.03 – математичне та програмне забезпечення обчислювальних машин і систем / Кахута Надія Дмитрівна. – Київ, 2010. – 94 с.

72. Кахута Н. Д. Застосування теоретико-множинних конструкцій повного образу, обмеження, конфінальності та сумісності в табличних базах даних: дисертація кандидата фізико-математичних наук: 01.05.03 – математичне та програмне забезпечення обчислювальних машин і систем / Н.Д. Кахута. – Київ, 2010. – 116 с.

73. Клини С. К. Введение в метаматематику / С. К. Клини. – М.: ИЛ, 1957. – 526 с.

74. Коннолли Т. Базы данных: проектирование, реализация и сопровождение. Теория и практика / Т. Коннолли, К. Бегг, А. Страчан. – Москва: Издательский дом "Вильямс", 2000. – 1120 с.

75. Кузнецов С. Д. Стандарты языка реляционных баз данных SQL: краткий обзор / С. Д. Кузнецов // СУБД. – 1996. – № 2. – С. 6-36.

76. Куратовский К., Мостовский А. Теория множеств. – Москва: Мир, 1970. – 416 с.

77. Мальцев А. И. Алгебраические системы / А. И. Мальцев. – Москва: Наука, 1964. – 392 с.

78. Мальцев А.И. Алгоритмы и рекурсивные функции. – Москва: Наука, 1986. – 368 с.

79. Марков А. С. Базы данных. Введение в теорию и методологию / А. С. Марков, К. Ю. Литовский. – Москва: Финансы и статистика, 2006. – 512 с.

80. Мартин Дж. Организация баз данных в вычислительных системах / Дж. Мартин. – Москва: Мир, 1980. – 660 с.

81. Мохаммед К.Д. Алгоритми CLOS та LOOPS лінеарізації класів в мовах програмування: формальна побудова / К.Д. Мохаммед, Д.Б. Буй, С.В. Компан, С.А. Поляков // Вісник Київського національного університету імені Тараса Шевченка. Серія: Фізико-математичні науки. – 2015. – Вип. 2. – С. 99-102. http://www.library.univ.kiev.ua/ukr/host/10.23.10.100/db/ftp/visnyk/fiz_mat_2_2015.pdf

82. Мохаммед К.Д. Відношення конфінальності, передпорядки та порядки, семантика фрази ORDER BY запитів SQLподібних мов[Електронний ресурс] / Д. Б. Буй, Н. Д. Кахута, О. В. Шишацька, Sunmade Fabunmi, К.Д. Мохаммед // Вісник Київського національного університету імені Тараса Шевченка. Серія: Фізико-математичні науки. – 2015. – Вип. 4. – С. 88-95. – Режим доступу: http://nbuv.gov.ua/UJRN/VKNU_fiz_mat_2015_4_16

83. Мохаммед К.Д. Логики частичных предикатов, индуцированные трехзначными логиками Клини / Д.Б. Буй, Е.В. Шишацкая, Ф. Санмейд, К.Д. Мохаммед // Штучний інтелект.– 2015. – №3-4. – С. 84-88. (Режим доступу: http://nbuv.gov.ua/UJRN/II_2015_3-4_10)

84. Мохаммед К.Д. Математические основания множественного наследования: рефлексивно-транзитивное замыкание. / Д.Б. Буй, Е.В. Шишацкая, Ф. Санмейд, К.Д. Мохаммед // Вісник Харківського національного університету ім. В.Н. Каразіна. – 2016. – №28. - С .19-33 http://periodicals.karazin.ua/mia/article/viewFile/6549/6058

85. Мохаммед К.Д. Математические основания множественного наследования: рефлексивно-транзитивное замыкание / К.Д. Мохаммед, Д.Б. Буй, Е.В. Шишацкая, Ф. Санмейд // Міжнародний науково-практичний семінар – "Комбінаторні конфігурації та їх застосування" – 2016. http://it-kntu.kr.ua/2016/04/18/results-cca-2016/#more-3782

86. Мохаммед К.Д. Недетерминированные функции в SQL / К.Д. Мохаммед, Буй Д.Б., Поляков С.А. // XIII Міжнародна науково-практична конференція ТЕОРЕТИЧНІ ТА ПРИКЛАДНІ АСПЕКТИ ПОБУДОВИ ПРОГРАМНИХ СИСТЕМ. – Київ, 5-9 грудня 2016 рокАу. - с 44-49- http://phd.isofts.kiev.ua/taapsd-2016/

87. Мохаммед К.Д. Огляд типів рекомендаційних систем та використання баз даних для підвищення їх продуктивності / К.Д. Мохаммед, Буй Д.Б., Компан С.В., Поляков С.А. // XI Міжнародна науково-практична конференція: ТЕОРЕТИЧНІ ТА ПРИКЛАДНІ АСПЕКТИ ПОБУДОВИ ПРОГРАМНИХ СИСТЕМ. – Київ, 15-17 грудня 2014 р. - с 29-34- http://taapsd.at.ua/TAAPSD-2014.pdf

88. Мохаммед К.Д. Реализация работы нестандартных типов данных в Framework Django на примере типа данных ltree / К.Д. Мохаммед, Д.Б.Буй, С.В.Компан, С.А. Поляков // XII Міжнародна науково-практична конференція ТЕОРЕТИЧНІ ТА ПРИКЛАДНІ АСПЕКТИ ПОБУДОВИ ПРОГРАМНИХ СИСТЕМ. – Київ, 23-26 грудня 2015 року. - с 26-32- http://taapsd.at.ua/

89. Мохаммед К.Д. Рефлексивно-транзитивные замыкания бинарных отношений. / К.Д. Мохаммед, Д.Б. Буй, Е.В. Шишацкая, Ф. Санмейд // Электротехнические и компьютерные системы. – 2016. – №22(98). – С.272-276. http://www.etks.opu.ua/?fetch=articles&with=info&id=806

90. Мохаммед К.Д. Характерестичні властивості одиночного успадкування класів. / К.Д. Мохаммед // Вісник Київського національного університету імені Тараса Шевченка. Серія: Фізико-математичні науки. – 2016. – Вип. 4. – С. 104-107.

91. Мохаммед К.Д. Математические основания современных реляционных СУБД / К.Д. Мохаммед, Д. Б. Буй, , Н. Д. Кахута / I Международный научно-практический форум «Наука и бизнес»: Тезисы докладов. – Черновцы, 2015. – С. 52-55- http://library.krok.edu.ua/media/library/category/statti/kakhuta_0008.pdf

92. Общая алгебра. Т. 1 / О.В. Мельников, В.Н. Ремесленников, В.А. Романьков и др. Под общей редакцией Л.А. Скорнякова. – Москва: Наука, 1990. – 592 с.

93. Поляков С.А. Алгебра табличних функцій / С.А. Поляков // Вісник Київського університету. Сер.: фіз.-мат. науки. – 1996. – Вип. 2. – С. 150-155.

94. Поляков С.А. Ієрархічні дані в SQL / С. А. Поляков // Современные направления теоретических и прикладных исследований: международная конференция SWORD, 16-27 марта 2009 г., Одесса: Черноморье. – 2009. – Т. 2. – С. 52-59.

95. Поляков С.А. Композиційна семантика SQL-подібних мов / С.А. Поляков // Вісник Київського університету. Сер.: фіз.-мат. науки. – 1997. – Вип. 3. – С. 205-211.

96. Поляков С.А. Композиційна семантика рекурсивних виразів та їхніх узагальнень в SQL-подібних мовах / С.А. Поляков, Д.Б.Буй // Наукові записки НаУКМА . Сер. Комп'ютерні науки. – 2010. – Том 112 . – С. 21-25.

97. Поляков С.А. Композиційна семантика рекурсивних запитів в SQL-подібних мовах / Д. Б. Буй, С. А. Поляков // Вісник Київського університету. Сер. фіз.-мат. науки. – 2010. – Вип. 1. – С. 45-56.

98. Поляков С.А. Огляд стандартів мови SQL / С. А. Поляков // Вісник Київського університету. Сер. фіз.-мат. науки. – 2008. – Вип. 1. – С. 132-137.

99. Поляков С.А. Рекурсивні запити в SQL-подібних мовах: приклади, змістовна і формальна семантика. / С.А. Поляков, Д.Б.Буй // Проблеми програмування. – 2010. – №2-3 (Спеціальний випуск). – С.434-439.

100. Поляков С.А. Семантика языков запросов в реляционных базах данных: SQL / С.А. Поляков // XI Международная конференция "Проблемы теоретической кибернетики". – Ульяновск: СВНЦ. – 1996. – С. 24.

101. Поляков С.А. Типи даних в стандарті SQL: 2003 / С. А. Поляков, Д. Б. Буй // International Conference "Theoretical and Applied Aspects of Program Systems Development (TAAPSD'2008)". Abstracts (Ukraine, Chernihiv, Kyiv, 22-26 September, 2008). Volume 2. – Київ: Пульсари, 2008. – С. 53-57.

102. Редько В. Н. Дескриптологические основания программирования / В. Н. Редько // Кибернетика и системный анализ. – 2002. – № 1. – С. 3-19.

103. Редько В. Н. Информационный аспект Case-технологий: основные соотношения в табличных алгебрах / В. Н. Редько, Ю. И. Брона, Д. Б. Буй // Проблемы программирования. – 1997. – Вып. 1. – С. 5-11.

104. Редько В. Н. К основаниям теории реляционных баз данных: табличные алгебры / В. Н. Редько, Ю. И. Брона, Д. Б. Буй; Киев. ун-т. – Киев, 1996. – 105 с. – Рус. – Деп. В ГНТБ Украины 11.11.96, № 3189УК-96.

105. Редько В. Н. К основаниям теории реляционных моделей баз данных / В. Н. Редько, Д. Б. Буй // Кибернетика и системный анализ. – 1996. – № 4. – С. 3-13.

106. Редько В. Н. Композиции программ и композиционное программирование / В. Н. Редько // Программирование. – 1978. – № 5. – С. 3-24.

107. Редько В. Н. Композиционная структура программологии / В. Н. Редько // Кибернетика и системный анализ. – 1998. – № 4. – С. 47-66.

108. Редько В. Н. Основания композиционного программирования / В. Н. Редько // Программирование. – 1979. – № 3. – С. 3-13.

109. Редько В. Н. Основания программологии / В. Н. Редько // Кибернетика и системный анализ. – 2000. – № 1. – С. 35-57.

110. Редько В. Н. Програмологія: ретроспективи та перспективи / В. Н. Редько // Вісник. Кібернетика (Київський національний університет імені Тараса Шевченка). – 2000. –Вип. 1. – С. 34-57.

111. Редько В. Н. Реляционные алгебры: операции деления и переименования / В. Н. Редько, Ю. И. Брона, Д. Б. Буй // Кибернетика и системный анализ. – 1997. – № 5. – С. 3-15.

112. Редько В. Н. Реляционные алгебры: операции проекции и соединения / В. Н. Редько, Ю. И. Брона, Д. Б. Буй // Кибернетика и системный анализ. – 1997. – № 4. – С. 89-100.

113. Редько В. Н. Семантические структуры программ / В. Н. Редько // Программирование. – 1981. – № 1. – С. 3-19.

114. Редько В. Н. Современное состояние теории мультимножеств с сущностной точки зрения / В. Н. Редько, Д. Б. Буй, Ю. А. Гришко // Кибернетика и системный анализ. – 2015. – № 1. – С. 171-178.

115. Редько В.Н. Реляционные алгебры: операции проекции и соединения / В.Н. Редько, Ю.И. Брона, Д.Б. Буй // Кибернетика и системный анализ. – 1997. – №4. – С. 89-100.

116. Риге Ж. Бинарные отношения, замыкания, соответствия Галуа // Кибернетический сборник: Сборник переводов. – Москва: Иностранная литература, 1963. – Вып. 7. – С. 129-185.

117. Скорняков Л. А. Элементы теории структур. / Л. А. Скорняков. – М.: Наука, 1970. – 160 с.

118. Скорняков Л.А. Элементы теории структур. – Москва: Наука, 1982. – 159 с.

119. Ульман Дж. Основы систем баз данных / Дж. Ульман. – Москва: Финансы и статистика, 1983. – 334 с.

120. Яблонский С. В. Введение в дискретную математику / С. В. Яблонский. – Москва: Наука, 1986. – 384 с.

121. Язык описания данных КОДАСИЛ / Пер. с англ. под ред. М. Р. Когаловского и Г. К. Столяров. – М.:Статистика, 1981. – 183 с.

Додаток 1 База даних інвойсів

Таблиця 1. general_ledger_accounts

account_number	account_description
100	Cash
110	Accounts Receivable
120	Book Inventory
150	Furniture
160	Computer Equipment
162	Capitalized Lease
167	Software
170	Other Equipment
181	Book Development
200	Accounts Payable
205	Royalties Payable
221	401K Employee Contributions
230	Sales Taxes Payable
234	Medicare Taxes Payable
235	Income Taxes Payable
237	State Payroll Taxes Payable
238	Employee FICA Taxes Payable
239	Employer FICA Taxes Payable
241	Employer FUTA Taxes Payable
242	Employee SDI Taxes Payable
243	Employer UCI Taxes Payable
251	IBM Credit Corporation Payable
280	Capital Stock
290	Retained Earnings
300	Retail Sales
301	College Sales
302	Trade Sales
306	Consignment Sales
310	Compositing Revenue
394	Book Club Royalties
400	Book Printing Costs
403	Book Production Costs
500	Salaries and Wages
505	FICA
506	FUTA
507	UCI

508	Medicare
510	Group Insurance

Таблиця 2. **invoices**

invoice_id	vendor_id	invoice_number	invoice_date	invoice_total	payment_total	credit_total	terms_id	invoice_due_date	payment_date
1	34	QP58872	0001-02-25 BC	116.54	116.54	0.00	4	0001-04-22 BC	0001-04-11 BC
2	34	Q545443	0001-03-14 BC	1083.58	1083.58	0.00	4	0001-05-23 BC	0001-05-14 BC
3	110	P-0608	0001-04-11 BC	20551.18	0.00	1200.00	5	0001-06-30 BC	NULL
4	110	P-0259	0001-04-16 BC	26881.40	26881.40	0.00	3	0001-05-16 BC	0001-05-12 BC
5	81	MABO1489	0001-04-16 BC	936.93	936.93	0.00	3	0001-05-16 BC	0001-05-13 BC
6	122	989319-497	0001-04-17 BC	2312.20	0.00	0.00	4	0001-06-26 BC	NULL
7	82	C73-24	0001-04-17 BC	600.00	600.00	0.00	2	0001-05-10 BC	0001-05-05 BC
8	122	989319-487	0001-04-18 BC	1927.54	0.00	0.00	4	0001-06-19 BC	NULL
9	122	989319-477	0001-04-19 BC	2184.11	2184.11	0.00	4	0001-06-12 BC	0001-06-07 BC
10	122	989319-467	0001-04-24 BC	2318.03	2318.03	0.00	4	0001-06-05 BC	0001-05-29 BC
11	122	989319-457	0001-04-24 BC	3813.33	3813.33	0.00	3	0001-05-29 BC	0001-05-20 BC
12	122	989319-447	0001-04-24 BC	3689.99	3689.99	0.00	3	0001-05-22 BC	0001-05-12 BC
13	122	989319-437	0001-04-24 BC	2765.36	2765.36	0.00	2	0001-05-15 BC	0001-05-03 BC
14	122	989319-427	0001-04-25 BC	2115.81	2115.81	0.00	1	0001-05-08 BC	0001-05-01 BC
15	121	97/553B	0001-04-26 BC	313.55	0.00	0.00	4	0001-07-09 BC	NULL

Таблиця 3. invoice_line_items

invoice_id	invoice_sequence	account_number	line_item_amt	line_item_description
1	1	572	116.54	MVS Online Library
2	1	572	1083.58	MSDN
3	1	120	20551.18	CICS Part 2

4	1	120	26881.40	MVS JCL
5	1	527	936.93	Quarterly Maintenance
6	1	553	2312.20	Freight
7	1	541	600.00	Trade advertising
8	1	553	1927.54	Freight
9	1	553	2184.11	Freight
10	1	553	2318.03	Freight
11	1	553	3813.33	Freight
12	1	553	3689.99	Freight
13	1	553	2765.36	Freight
14	1	553	2115.81	Freight
15	1	540	313.55	Card revision
16	1	553	2051.59	Freight
17	1	523	356.48	Network wiring
18	1	540	904.14	DB2 Card decks
19	1	536	1197.00	MC Bouncebacks
19	2	540	765.13	SCMD Flyer
20	1	536	639.77	Card deck
21	1	536	601.95	Card deck revision
22	1	536	953.10	Crash Course revision
23	1	541	565.15	Crash Course Ad
24	1	540	1000.46	Crash Course Cover
25	1	553	10.00	Freight
26	1	553	162.75	International shipment
27	1	553	10.00	Address correction
28	1	553	27.00	Freight
29	1	553	13.75	Freight
30	1	570	17.50	Supplies
31	1	553	144.70	Int'l shipment
32	1	523	95.00	Telephone service
33	1	553	33.00	Freight
34	1	120	10976.06	VSAM for the Cobol Programmer
35	1	553	63.40	Freight
36	1	120	23517.58	DB2 Part 1
37	1	120	37966.19	CICS Desk Reference
38	1	553	61.50	Freight
39	1	553	158.00	Int'l shipment
40	1	553	26.75	Freight
41	1	553	23.50	Freight
42	1	572	9.95	Monthly access fee
43	1	572	9.95	Monthly access fee
44	1	553	52.25	Freight
45	1	553	109.50	Freight
46	1	553	42.75	Freight
47	1	553	36.00	Freight

48	1	553	111.00	Freight
49	1	553	53.25	Freight
50	1	553	53.75	Freight
51	1	553	108.50	Freight
52	1	553	38.75	Freight
53	1	553	15.50	Freight
54	1	553	127.75	Freight
55	1	553	241.00	Int'l shipment
56	1	553	129.00	Freight
57	1	553	40.75	Freight
58	1	553	60.00	Freight
59	1	553	104.00	Freight
60	1	553	67.00	Freight
61	1	553	147.25	Freight
62	1	553	172.50	Freight
63	1	553	108.25	Freight
64	1	553	138.75	Freight
65	1	553	127.75	Freight
66	1	553	739.20	Freight
67	1	553	10.00	Address correction
68	1	553	30.75	Freight
69	1	510	116.00	Health Insurance
70	1	553	67.92	Freight
71	1	553	59.97	Freight
72	1	553	26.25	Freight
73	1	553	22.57	Freight
74	1	553	31.95	Freight
75	1	553	42.67	Freight
76	1	553	44.44	Freight
77	1	553	40.20	Freight
78	1	553	400.00	Freight
79	1	553	42.50	Freight
80	1	591	220.00	Form 571-L
81	1	541	1575.00	Catalog ad
82	1	553	6.00	Freight out
83	1	553	25.67	Freight out
84	1	552	6.00	Freight out
85	1	553	6.00	Freight
86	1	540	207.78	Prospect list
87	1	536	2184.50	PC card deck
88	1	536	2433.00	Card deck
89	1	120	6940.25	OS Utilities
90	1	523	450.00	Back office additions
91	1	536	90.36	Card deck advertising
92	1	536	175.00	Card deck advertising

93	1	548	7125.34	Web site design
94	1	569	503.20	Bronco lease
95	1	235	1600.00	Income Tax
96	1	520	1750.00	Warehouse lease
97	1	520	4901.26	Office lease
98	1	522	46.21	Telephone (Line 1)
99	1	522	39.77	Telephone (Line 2)
100	1	580	50.00	DiCicco's
100	2	540	75.60	Kinko's
100	3	570	58.40	Office Max
100	4	546	478.00	Publishers Marketing
101	1	221	1367.50	401K Contributions
102	1	574	856.92	Property Taxes
103	1	522	19.67	Telephone (Line 3)
104	1	552	290.00	International pkg.
105	1	522	32.70	Telephone (line 4)
106	1	522	16.33	Telephone (line 5)
107	1	522	16.33	Telephone (line 6)
108	1	589	16.62	Propane-forklift
109	1	580	41.80	Coffee
110	1	540	85.31	Book copy
111	1	510	224.00	Health Insurance
112	1	510	224.00	Health Insurance
113	1	540	21842.00	Book repro
114	1	541	579.42	Catalog ad

Таблиця 4. vendors

vendor_id	vendor_name	vendor_address1	vendor_address2	vendor_city	vendor_state	vendor_zip_code	vendor_phone	vendor_contact_last_name	vendor_contact_first_name	default_terms_id	default_account_number
1	US Postal Service	Attn: Supt. Window Services	PO Box 7005	Madison	WI	53707	(800) 555-1205	Alberto	Francesco	1	552
2	National Information Data Ctr	PO Box 96621	NULL	Washington	DC	20090	(301) 555-8950	Irvin	Ania	3	540
3	Register of	Library Of	NULL	Washingt	DC	20559	NULL	Liana	Lukas	3	403

	Copyrights	Congress		on							
4	Jobtrak	1990 Westwood Blvd Ste 260	NULL	Los Angeles	CA	90025	(800) 555-8725	Quinn	Kenzie	3	572
5	Newbrige Book Clubs	3000 Cindel Drive	NULL	Washington	NJ	07882	(800) 555-9980	Marks	Michelle	4	394
6	California Chamber Of Commerce	3255 Ramos Cir	NULL	Sacramento	CA	95827	(916) 555-6670	Mauro	Anton	3	572
7	Towne Advertiser's Mailing Svcs	Kevin Minder	3441 W Macarthur Blvd	Santa Ana	CA	92704	NULL	Maegen	Ted	3	540
8	BFI Industries	PO Box 9369	NULL	Fresno	CA	93792	(559) 555-1551	Kaleigh	Erick	3	521
9	Pacific Gas & Electric	Box 52001	NULL	San Francisco	CA	94152	(800) 555-6081	Anthoni	Kaitlyn	3	521
10	Robbins Mobile Lock And Key	4669 N Fresno	NULL	Fresno	CA	93726	(559) 555-9375	Leigh	Bill	2	523
11	Bill Marvin Electric Inc	4583 E Home	NULL	Fresno	CA	93703	(559) 555-5106	Hostlery	Kaitlin	2	523
12	City Of Fresno	PO Box 2069	NULL	Fresno	CA	93718	(559) 555-9999	Mayte	Kendall	3	574
13	Golden Eagle Insurance Co	PO Box 85826	NULL	San Diego	CA	92186	NULL	Blanca	Korah	3	590
14	Expedata Inc	4420 N. First	NULL	Fresno	CA	93726	(559) 555-	Quintin	Marvin	3	589

		Street, Suite 108					9586				
15	ASC Signs	1528 N Sierra Vista	NULL	Fresno	CA	93703	NULL	Darien	Elisabeth	1	546
16	Internal Revenue Service	NULL	NULL	Fresno	CA	93888	NULL	Aileen	Joan	1	235
17	Blanchard & Johnson Associates	27371 Valderas	NULL	Mission Viejo	CA	92691	(214) 555-3647	Keeton	Gonzalo	3	540
18	Fresno Photoengraving Company	1952 "H" Street	P.O. Box 1952	Fresno	CA	93718	(559) 555-3005	Chaddick	Derek	3	403
19	Crown Printing	1730 "H" St	NULL	Fresno	CA	93721	(559) 555-7473	Randrup	Leann	2	400
20	Diversified Printing & Pub	2632 Saturn St	NULL	Brea	CA	92621	(714) 555-4541	Lane	Vanesa	3	400
21	The Library Ltd	7700 Forsyth	NULL	St Louis	MO	63105	(314) 555-8834	Marques	Malia	3	540
22	Micro Center	1555 W Lane Ave	NULL	Columbus	OH	43221	(614) 555-4435	Evan	Emily	2	160
23	Yale Industrial Trucks - Fresno	3711 W Franklin	NULL	Fresno	CA	93706	(559) 555-2993	Alexis	Alexandro	3	532
24	Zee Medical Service Co	4221 W Sierra Madre #104	NULL	Washington	IA	52353	NULL	Hallie	Juliana	3	570

25	California Data Marketing	2818 E Hamilton	NULL	Fresno	CA	93721	(559) 555-3801	Jonessen	Moises	4	540
26	Small Press	121 E Front St - 4th Floor	NULL	Traverse City	MI	49684	NULL	Colette	Dusty	3	540
27	Rich Advertising	12 Daniel Road	NULL	Fairfield	NJ	07004	(201) 555-9742	Neil	Ingrid	3	540
29	Vision Envelope & Printing	PO Box 3100	NULL	Gardena	CA	90247	(310) 555-7062	Raven	Jamari	3	551
30	Costco	Fresno Warehouse	4500 W Shaw	Fresno	CA	93711	NULL	Jaquan	Aaron	3	570
31	Enterprise Communications Inc	1483 Chain Bridge Rd, Ste 202	NULL	Mclean	VA	22101	(770) 555-9558	Lawrence	Eileen	2	536
32	RR Bowker	PO Box 31	NULL	East Brunswick	NJ	08810	(800) 555-8110	Essence	Marjorie	3	532
33	Nielson	Ohio Valley Litho Division	Location #0470	Cincinnati	OH	45264	NULL	Brooklynn	Keely	2	541
34	IBM	PO Box 61000	NULL	San Francisco	CA	94161	(800) 555-4426	Camron	Trentin	1	160
35	Cal State Termite	PO Box 956	NULL	Selma	CA	93662	(559) 555-1534	Hunter	Demetrius	2	523
36	Graylift	PO Box 2808	NULL	Fresno	CA	93745	(559) 555-6621	Sydney	Deangelo	3	532
37	Blue Cross	PO Box 9061	NULL	Oxnard	CA	93031	(800) 555-0912	Eliana	Nikolas	3	510
38	Venture Comm	60 Madison Ave	NULL	New York	NY	10010	(212) 555-4800	Neftaly	Thalia	3	540

	unicati ons Int'l										
39	Custo m Printin g Compa ny	PO Box 7028	NULL	St Loui s	MO	63177	(301) 555- 1494	Myles	Harley	3	540
40	Nat Assoc of Colleg e Stores	500 East Lorain Street	NULL	Ober lin	OH	44074	NUL L	Bernard	Lucy	3	572
41	Shield s Design	415 E Olive Ave	NULL	Fres no	CA	93728	(559) 555- 8060	Kerry	Rowan	2	403
42	Opam p Techni cal Books	1033 N Sycam ore Ave.	NULL	Los Ang eles	CA	90038	(213) 555- 4322	Paris	Gideon	3	572
43	Capital Resour ce Credit	PO Box 39046	NULL	Min neap olis	MN	55439	(612) 555- 0057	Maxwell	Jayda	3	589
44	Courie r Compa nies, Inc	PO Box 5317	NULL	Bost on	MA	02206	(508) 555- 6351	Antavius	Troy	4	400
45	Naylor Public ations Inc	PO Box 40513	NULL	Jack sonv ille	FL	32231	(800) 555- 6041	Gerald	Kristofer	3	572
46	Open Horizo ns Publis hing	Book Market ing Update	PO Box 205	Fairf ield	IA	52556	(515) 555- 6130	Damien	Deborah	2	540
47	Baker & Taylor Books	Five Lakepo inte Plaza, Ste 500	2709 Water Ridge Parkwa y	Char lotte	NC	28217	(704) 555- 3500	Bernardo	Brittnee	3	572
48	Fresno Count y Tax Collect	PO Box 1192	NULL	Fres no	CA	93715	(559) 555- 3482	Brenton	Kila	3	574

	or										
49	Mcgraw Hill Companies	PO Box 87373	NULL	Chicago	IL	60680	(614) 555-3663	Holbrooke	Rashad	3	572
50	Publishers Weekly	Box 1979	NULL	Marion	OH	43305	(800) 555-1669	Carrollton	Priscilla	3	572
51	Blue Shield of California	PO Box 7021	NULL	Anaheim	CA	92850	(415) 555-5103	Smith	Kylie	3	510
52	Aztek Label	Accounts Payable	1150 N Tustin Ave	Anaheim	CA	92807	(714) 555-9000	Griffin	Brian	3	551
53	Gary McKeighan Insurance	3649 W Beechwood Ave #101	NULL	Fresno	CA	93711	(559) 555-2420	Jair	Caitlin	3	590
54	Ph Photographic Services	2384 E Gettysburg	NULL	Fresno	CA	93726	(559) 555-0765	Cheyenne	Kaylea	3	540
55	Quality Education Data	PO Box 95857	NULL	Chicago	IL	60694	(800) 555-5811	Misael	Kayle	2	540
56	Springhouse Corp	PO Box 7247-7051	NULL	Philadelphia	PA	19170	(215) 555-8700	Maeve	Clarence	3	523
57	The Windows Deck	117 W Michelotorena Top Floor	NULL	Santa Barbara	CA	93101	(800) 555-3353	Wood	Liam	3	536
58	Fresno Rack & Shelving Inc	4718 N Bendel Ave	NULL	Fresno	CA	93722	NULL	Baylee	Dakota	2	523

Таблиця 5. vendor_contacts

vendor_id	last_name	first_name
5	Davison	Michelle
12	Mayteh	Kendall
17	Onandonga	Bruce
44	Antavius	Anthony
76	Bradlee	Danny
94	Suscipe	Reynaldo
101	O'Sullivan	Geraldine
123	Bucket	Charles

Таблиця 6. terms

terms_id	terms_description	terms_due_days
1	Net due 10 days	10
2	Net due 20 days	20
3	Net due 30 days	30
4	Net due 60 days	60
5	Net due 90 days	90

Додаток 2. Список публікацій здобувача

1. Мохаммед К.Д. Алгоритми CLOS та LOOPS лінеарізації класів в мовах програмування: формальна побудова / К.Д. Мохаммед, Д.Б. Буй, С.В. Компан, С.А. Поляков // Вісник Київського національного університету імені Тараса Шевченка. Серія: Фізико-математичні науки. – 2015. – Вип. 2. – С. 99-102. http://www.library.univ.kiev.ua/ukr/host/10.23.10.100/db/ftp/visnyk/fiz_mat_2_2015.pdf

2. Мохаммед К.Д. Відношення конфінальності, передпорядки та порядки, семантика фрази ORDER BY запитів SQLподібних мов[Електронний ресурс] / Д. Б. Буй, Н. Д. Кахута, О. В. Шишацька, Sunmade Fabunmi, К.Д. Мохаммед // Вісник Київського національного університету імені Тараса Шевченка. Серія: Фізико-математичні науки. – 2015. – Вип. 4. – С. 88-95. – Режим доступу: http://nbuv.gov.ua/UJRN/VKNU_fiz_mat_2015_4_16

3. Мохаммед К.Д. Логики частичных предикатов, индуцированные трехзначными логиками Клини / Д.Б. Буй, Е.В. Шишацкая, Ф. Санмейд, К.Д.

Мохаммед // Штучний інтелект.– 2015. – №3-4. – С. 84-88. (Режим доступу: http://nbuv.gov.ua/UJRN/II_2015_3-4_10)

4. Мохаммед К.Д. Математические основания множественного наследования: рефлексивно-транзитивное замыкание. / Д.Б. Буй, Е.В. Шишацкая, Ф. Санмейд, К.Д. Мохаммед // Вісник Харківського національного університету ім. В.Н. Каразіна. – 2016. – №28. - С .19-33 http://periodicals.karazin.ua/mia/article/viewFile/6549/6058

5. Мохаммед К.Д. Рефлексивно-транзитивные замыкания бинарных отношений. / К.Д. Мохаммед, Д.Б. Буй, Е.В. Шишацкая, Ф. Санмейд // Электротехнические и компьютерные системы. – 2016. – №22(98). – С.272-276. http://www.etks.opu.ua/?fetch=articles&with=info&id=806

6. Мохаммед К.Д. Характерестичні властивості одиночного успадкування класів. / К.Д. Мохаммед // Вісник Київського національного університету імені Тараса Шевченка. Серія: Фізико-математичні науки. – 2016. – Вип. 4. – С. 104-107.

7. Mohammed K. Join Operations in Relational Databases with Automatic Attributes Renaming / Poliakov S, Buy D, Mohammed K., Israa Jasim AL.kalafa // Scholars Journal of Engineering and Technology. – 2017. –№5(6) - P.254-257. http://saspublisher.com/sjet-56/

8. Мохаммед К.Д. Огляд типів рекомендаційних систем та використання баз даних для підвищення їх продуктивності / К.Д. Мохаммед, Буй Д.Б., Компан С.В., Поляков С.А. // XI Міжнародна науково-практична конференція: ТЕОРЕТИЧНІ ТА ПРИКЛАДНІ АСПЕКТИ ПОБУДОВИ ПРОГРАМНИХ СИСТЕМ. – Київ, 15-17 грудня 2014 р. - с 29-34- http://taapsd.at.ua/TAAPSD-2014.pdf

9. Мохаммед К.Д. Математические основания современных реляционных СУБД / К.Д. Мохаммед, Д. Б. Буй, , Н. Д. Кахута / I Международный научно-

практический форум «Наука и бизнес»: Тезисы докладов. – Черновцы, 2015. – С. 52-55- http://library.krok.edu.ua/media/library/category/statti/kakhuta_0008.pdf

10. Mohammed K. Linearization algorithms CLOS and LOOPS of the classes in programming languages: the formal definitions / K. Mohammed, D. Buy, J. Karam, S. Kompan, S. Polyakov.// 13th International Scientific Conference on Informatics. – Poprad, Slovakia, 18-20 Nov., 2015. – P. 63-66 (Print ISBN: 978-1-4673-9867-1, DOI 10.1109/Informatics.2015.7377809). http://ieeexplore.ieee.org/document/7377809/

11. Мохаммед К.Д. Реализация работы нестандартных типов данных в Framework Django на примере типа данных ltree / К.Д. Мохаммед, Д.Б.Буй, С.В.Компан, С.А. Поляков // XII Міжнародна науково-практична конференція ТЕОРЕТИЧНІ ТА ПРИКЛАДНІ АСПЕКТИ ПОБУДОВИ ПРОГРАМНИХ СИСТЕМ. – Київ, 23-26 грудня 2015 року. - с 26-32- http://taapsd.at.ua/

12. Мохаммед К.Д. Недетерминированные функции в SQL / К.Д. Мохаммед, Буй Д.Б., Поляков С.А. // XIII Міжнародна науково-практична конференція ТЕОРЕТИЧНІ ТА ПРИКЛАДНІ АСПЕКТИ ПОБУДОВИ ПРОГРАМНИХ СИСТЕМ. – Київ, 5-9 грудня 2016 рокАу. - с 44-49- http://phd.isofts.kiev.ua/taapsd-2016/

13. Mohmmed K. Mathematical Foundations of Multiple Inheritance: Reflexive-transitive Closure of the Binary Relations / Dmytro Buy, Olena Shyshatska, Sunmade Fabunmi, Karam Mohammed // Perspective Technologies and Methods in MEMS Design (MEMSTECH), 2016 XII International Conference on Perspective Technologies and Methods in MEMS Design (MEMSTECH). – 2016. – С.192-195. (http://ieeexplore.ieee.org/document/7507540/)

14. Мохаммед К.Д. Математические основания множественного наследования: рефлексивно-транзитивное замыкание / К.Д. Мохаммед, Д.Б. Буй, Е.В. Шишацкая, Ф. Санмейд // Міжнародний науково-практичний семінар

– "Комбінаторні конфігурації та їх застосування" – 2016. http://it-kntu.kr.ua/2016/04/18/results-cca-2016/#more-3782

15. Mohmmed K. Математические основы множественного наследования в ООП /K. Mohmmed, Буй Д.Б., Шишацкая Е.В., Fabunmi S. // Четвёртая конференция «КОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ В НАУКОЕМКИХ ТЕХНОЛОГИЯХ» (КМНТ-2016). – Харьков, 26-31 мая 2016 г (http://www.dsmmph.org.ua/kmnt.html)

Додаток 3. Відомості про апробацію

Результати дисертаційного дослідження оприлюднені у доповідях і повідомленнях на Міжнародних та Всеукраїнських наукових конференціях, семінарах:

XI Міжнародна науково-практична конференція: ТЕОРЕТИЧНІ ТА ПРИКЛАДНІ АСПЕКТИ ПОБУДОВИ ПРОГРАМНИХ СИСТЕМ. – Київ, 15-17 грудня 2014 р.

I Международный научно-практический форум «Наука и бизнес»: – Черновцы, 2015.

13th International Scientific Conference on Informatics. – Poprad, Slovakia, 18-20 Nov., 2015.

XII Міжнародна науково-практична конференція ТЕОРЕТИЧНІ ТА ПРИКЛАДНІ АСПЕКТИ ПОБУДОВИ ПРОГРАМНИХ СИСТЕМ. – Київ, 23-26 грудня 2015 року.

XIII Міжнародна науково-практична конференція ТЕОРЕТИЧНІ ТА ПРИКЛАДНІ АСПЕКТИ ПОБУДОВИ ПРОГРАМНИХ СИСТЕМ. – Київ, 5-9 грудня 2016 року.

XII International Conference on Perspective Technologies and Methods in MEMS Design (MEMSTECH). – 2016.

Міжнародний науково-практичний семінар – "Комбінаторні конфігурації та їх застосування" – 2016.

Четвёртая конференция «КОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ В НАУКОЕМКИХ ТЕХНОЛОГИЯХ» (КМНТ-2016). – Харьков, 26-31 мая 2016 г