Київський національний університет імені Тараса Шевченка

На правах рукопису

Поляков Сергій Анатолійович

УДК 004.655

Композиційна семантика ядра SQL-подібних мов

01.05.03 – математичне та програмне забезпечення обчислювальних машин і систем

Дисертація на здобуття наукового ступеня
кандидата фізико-математичних наук

Науковий керівник
Буй Дмитро Борисович,
доктор фіз.-мат. наук, старший науковий співробітник

Київ – 2011

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

D	універсум (універсальна множина)
$P(D)$	булеан (множина всіх підмножин) множини D
2^X	множина всіх скінченних підмножин множини X
$\varnothing$	порожня множина
$\varnothing_m$	порожня мультимножина
ε	порожнє бінарне відношення, порожній рядок
$U \mid X$	обмеження відношення U за множиною X
$U[X]$	повний образ множини X відносно бінарного відношення U
$\pi_i^2 U$	проекція бінарного відношення U за i-тою компонентою
$\approx$	бінарне відношення сумісності відношень (функцій)
dom$f$	область означеності функції f
true	логічне значення істини
false	логічне значення хиби
unknown,u	невідоме (третє) логічне значення тризначної логіки Кліні
$\wedge$	операція кон'юнкції
$\vee$	операція диз'юнкції
$\neg$	операція заперечення
$\otimes$	операція з'єднання (еквіз'єднання, inner natural join) таблиць на основі множин
$\otimes_m$	операція з'єднання (еквіз'єднання, inner natural join) таблиць на основі мультимножин
$\otimes_{A_1,\ldots,A_n}$	операція з'єднання таблиць за атрибутами
$\otimes_p$	операція з'єднання таблиць за предикатом
$f : A \xrightarrow{\sim} B$	часткова функція з множини A в множину B
$\simeq$	узагальнена рівність (сильна рівність Кліні)
$\cong$	відношення односхемнсті таблиць

$\cong_s$	відношення односхемнсті рядків
$\bigcup_t, \bigcap_t, \setminus_t$	об’єднання, перетин, різниця (односхемнх) таблиць
NULL, *null*	особливе, "невизначене" значення домену
N	множина натуральних чисел
N^+	множина натуральних чисел без 0
$\lvert X \rvert$	потужність множини (мультимножини) X
БНФ	форма Бекуса – Наура
СУБД	система управління базами даних

ВСТУП

Актуальність теми. Інформатизація суспільства зростає швидкими темпами, компанії, офіси, підприємства, навчальні та наукові заклади все ширше використовують інформаційні технології в своїй діяльності. В більшості випадків ядром, навколо якого будується інформаційна інфраструктура організації, виступають бази даних, як правило, реляційного типу. Реляційна модель, яка домінує над іншими моделями даних, тісно пов’язана з мовою SQL (Structured Query Language). Фактично, коли говориться про мову програмування для реляційних баз даних, то в якості такої мови мається на увазі саме SQL.

Мова SQL була створена в компанії IBM на початку 70-х років минулого століття і мала назву SEQUEL. Мова була спроектована для побудови запитів і маніпулювання даними в реляційній СУБД System R, розробленої в лабораторії IBM в Сан-Хосе. Пізніше її перейменували в SQL, тому що "SEQUEL" був торговою маркою авіаційної компанії Hawker Siddeley. Влітку 1979 р. Relational Software Inc, яка пізніше була перейменована в Oracle Corporation, випустила першу комерційну реалізацію SQL в СУБД Oracle V2, випередивши IBM на кілька тижнів. Компанія IBM, після тестування System R потенційними користувачами, почала розробку комерційного продукту на базі SQL. В 1979 р. була випущена СУБД System/38, в 1981 р. – СУБД SQL/DS, нарешті, в 1983 р. – відома СУБД DB/2, яка використовується і сьогодні.

Поступово SQL витіснив інші мови програмування баз даних і став стандартом де-факто.

Втілення мови різними компаніями, при відсутності стандартів значної частини мови, привело до появи чисельних діалектів SQL, тільки частково сумісних один з одним. Це, поряд зі складністю мови, вважається одним з головних її недоліків. Як правило, бази даних являються центром, навколо якого будується вся інша інформаційна структура організації. Тому зміна СУБД вимагає

перегляду та модифікації багатьох інших програм. При цьому треба враховувати, що багато запитів, вбудованих в програми, є динамічними, тобто компілюються тільки перед виконанням. Це приводить до того, що навіть синтаксичні помилки в запитах, особливо тих, які рідко виконуються, можуть бути виявлені тільки через деякий час після переходу на нову СУБД, не рідко досить значний.

Складність мови приводить до того, що навіть досвідчені програмісти не завжди розуміють, як буде інтерпретуватися той чи іншій запит. Особливо це стосується невизначених значень та трьохзначної логіки. В результаті запит, який коректно працював на початкових даних, може видавати невірний результат після зміни даних в таблицях. Враховуючи великі розміри сучасних баз даних, помітити такі помилки буває досить важко, і організація може довгий час використовувати невірно працюючий запит.

Для покращення становища актуальним є розробка семантики мови, яка б прояснювала "темні місця" SQL, розкривала його внутрішню логіку і була б надійною основою для подальшого розвитку SQL інтенсіональним шляхом. На сьогодні мова розвивається переважно екстенсіональним напрямком за рахунок додавання нових, часто досить складних, синтаксичних конструкцій в стандарт, які навіть провідні компанії не встигають своєчасно втілити в життя. А потім проходить ще багато часу, поки вони починають використовуватися широкими колами програмістів.

Реляційна алгебра, яка претендує на роль семантичного базису SQL, насправді таку роль не може виконувати, хоча б тому, що визначає таблиці через множини, а не мультимножини, як це має місце в SQL, не підтримує невизначені значення та трьохзначну логіку, не говорячи про те, що структура її операцій не відповідає логічній структурі SQL. Мається на увазі те, що SQL, як і всяка інша мова програмування, складається з трьох складових частин – структур даних, операцій і операторів (операцій над операціями), в той час як реляційна алгебра таке розділення не підтримує і підтримувати не може в силу своєї природи.

Виникає природна задача в розробці принципово нового типу алгебри, яка б задавала множину семантичних функцій SQL, базуючись на присутніх в мові

операторах та операціях, а також враховувала би перераховані вище властивості SQL.

Зв’язок роботи з науковими програмами, планами, темами. Дисертаційна робота є складовою частиною наукових робіт, які ведуться на кафедрі теорії та технології програмування факультету кібернетики Київського національного університету імені Тараса Шевченка (КНУ) при виконанні фундаментальних та прикладних тем: "Логіко-математичні та програмологічні засоби інформаційних технологій" (№ 0101U002163, 2001-2005 рр.), "Розробка конструктивних математичних формалізмів для інтелектуальних систем прийняття рішень, обробки знань, еталонування мов сучасних СУБД та CASE-засобів" (№ 0106U005856, 2006-2010 рр.), "Формальні специфікації програмних систем" (№ 08U002463, 2008-2009 рр.).

Мета і задачі дослідження. Метою дисертаційної роботи є побудова спеціальної програмної алгебри, носієм якої виступають функції на табличних даних, а в якості сигнатурних операцій використовуються композиції (оператори), які дозволяють будувати складні запити з атомарних функцій, а також огляд та класифікація основних типів моделей даних, до яких відносяться табличні, ієрархічні, мережні моделі, а також моделі, які базуються на багаторівневих таблицях та рекурсивних таблицях.

З огляду на мету в роботі ставляться такі задачі:

- створити нову семантичну модель таблиць, яка більш точно відповідає семантиці таблиць в реальних СУБД;

- надати визначення для композицій фільтрації, взяття повного образу, функції групування, операцій внутрішнього та зовнішнього з’єднання для нової семантичної моделі таблиць;

- побудувати нову семантику агрегатних функцій для коректної роботи з множинами (мультимножинами), які містять невизначені значення;

- побудувати операційну та денотаційну семантику рекурсивних запитів в SQL;

- побудувати класифікаційну схему для існуючих моделей даних на базі колекцій та відношень між елементами колекцій.

Предметом дослідження є мова SQL.

Об’єктом дослідження є реляційні бази даних.

Наукова новизна одержаних результатів. Вдосконалена та розширена композиційна SQL-орієнтована алгебра табличних функцій. Нове визначення таблиць за рахунок внесення схеми більш відповідає тому, як таблиці розуміються в реляційних базах даних. Це надало можливість більш точного та адекватного задання семантики операторів запитів мови SQL. Функції та композиції задані як на множинах так і на мультимножинах.

Вперше задана семантика рекурсивних запитів в SQL. Визначена як операційна, так і денотаційна семантика рекурсивних запитів, а також доведена їх еквівалентність при виконанні природних достатніх умов, таких, як дистрибутивність операції за першим аргументом та збереження порожньої таблиці за першим аргументом.

Досліджена композиційна структура операторів запитів та проведена класифікація операторів за їх синтаксичною структурою. Виділено три класи запитів, кожний з яких будується за чітко визначеною схемою.

Нове визначення композиції агрегації дозволяє більш точно задати семантику агрегатних функцій при роботі з невизначеними значеннями.

Проведено дослідження реляційної, ієрархічної, квазіреляційної та мережної моделі даних. Задана їх класифікація на основі поняття колекцій та відношень видів один до одного та один до багатьох на елементах колекцій. Побудована класифікаційна схема дозволила виявити та описати нові моделі даних, такі як рекурсивні таблиці, багаторівневі таблиці та таблиці з варіантними рядками.

Практичне значення одержаних результатів. Дисертація має теоретико-прикладну спрямованість. Результати роботи можуть використовуватись для задання семантики нових конструкцій SQL, розробці крос-процесорів, які транслюють запити з одного діалекту мови в іншій, заданні семантики нових мов запитів до реляційних, ієрархічних, мережевих та інших типів баз даних, при

побудові концептуальної моделі бази 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").

В Ніжинському державному університеті імені М.В. Гоголя на кафедрі прикладної математики та інформатики був прочитаний курс лекцій "Методологія проектування сучасних баз даних", підтриманий лабораторним практикумом.

В Кіровоградському державному педагогічному університеті імені Володимира Винниченка прочитаний курс лекцій "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.

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

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

- Всеукраїнська наукова конференція "Застосування обчислювальної техніки, математичного моделювання та математичних методів у наукових дослідженнях", Львів, 1997;

- The Fourth International Scientific Conference "Electronic Computers and Informatics’2000", September 28-29, 2000, Kosice – Herlany, Slovakia;

- Fifth International Conference "Information Theories & Applications", September, 1-15, 2000, Varna, Bulgaria;

- The Fourth International Conference "Theoretical and Applied Aspects of Program Systems Development" (TAAPSD’2007), Ukraine, Berdynsk, 4-9 September, 2007;

- The Fifth International Conference "Theoretical and Applied Aspects of Program Systems Development" (TAAPSD’2008), Ukraine, Chernihiv, Kyiv, 22-26 September, 2008;

- The Sixth International Conference "Theoretical and Applied Aspects of Program Systems Development" (TAAPSD’2009), Ukraine, Chernihiv, Kyiv, September 2009;

- Современные направления теоретических и прикладных исследований: международная конференция SWORD, 16-27 марта 2009 г., Одесса;

- VII міжнародна науково-практична конференція з програмування УкрПРОГ 2010, Київ, Україна, 2010 р.;

- Міжнародна наукова конференція "Computer Science and Engineering" CSE'2010, Кошице, Стара Любовна, Словаччина 2010 р.

Публікації. Результати дисертації опубліковані у 1 монографії [67] , 13 статтях в наукових журналах і збірниках наукових праць [8, 9, 10, 11, 12, 13, 14, 15, 46, 47, 48, 52, 53], 10 тезах конференцій [20, 45, 49, 50, 51, 54, 85, 86, 87, 104]; з них 13 статей опубліковані у фахових виданнях, затверджених ВАК України.

Структура і обсяг роботи. Дисертаційна робота складається з вступу, чотирьох розділів, висновків, списку використаних джерел (105 найменувань на 10 с.). Загальний обсяг дисертації становить 120 с., основний зміст викладено на 110 с. Праця містить 13 рис. та 13 табл.

РОЗДІЛ 1. ЗМІСТОВНА СЕМАНТИКА ЯДРА МОВИ SQL

Мета розділу полягає в огляді базових операторів мови SQL, формальна семантика яких буде описана в наступному розділі.

Об’єкт дослідження – оператори мови SQL, призначені для створення таблиць, оператори маніпулювання таблицями та оператори запитів на таблицях.

Основні результати розділу: огляд операторів SELECT, INSERT, UPDATE, DELETE. Розглянуті операції внутрішнього та зовнішнього з'єднання таблиць, агрегатні функції, підзапити та зв’язані (correlate) підзапити. Описаний оператор групування та розглянуті рекурсивні запити.

1.1. Базова форма оператора SELECT

При огляді мови SQL спираємось на [27, 89, 99, 101]. В найпростішому вигляді оператору SELECT потрібні тільки атрибути таблиці, які треба отримати і назва таблиці, звідки ці атрибути беруться:

SELECT < список атрибутів > FROM < таблиця >;

Звернемо увагу, що крапка з комою в кінці оператора не є обов’язковою і може бути опущена.

Атрибути виводяться в тому порядку, як вони задані в запиті (зліва-направо), порядок рядків невизначений, в різних реалізаціях СУБД він може бути різним. Зовнішній вигляд представлення особливого значення NULL теж може відрізнятися в різних СУБД.

Мова SQL не є чутливою до регістру букв, але прийнято набирати всі ключові слова мови в верхньому регістрі. Домовимося назву таблиці починати з великої літери, а назву атрибута з маленької, далі слідують маленькі букви в обох випадках. Якщо назва складається з декількох слів, то кожне наступне слово починається з великої літери. Наприклад ProjectEmployee буде назвою таблиці, а employeesId – назвою атрибута.

Будемо вважати, що кожний оператор SELECT генерує таблицю, хоча насправді, в реальних СУБД, таблиця (фізично) може і не створюватися.

Запити, які повертають всі рядки таблиці, використовуються не дуже часто. Набагато частіше треба повернути лише частину рядків, відфільтрувавши решту. Для цього в мові існує фраза WHERE, яка задає деяку умову. Тільки рядки, які задовольняють цієї умові, попадають в результат. Для фільтрування використовуються предикати, деякі з яких, а саме предикати на атомарних даних, приведені в табл. 1.1.

Таблиця 1.1 – Таблиця предикатів мови SQL

Предикат	**Назва**	**Приклад використання**
=	дорівнює	fld=5, fld1=fld2, 5=fld
<, >	менше, більше	55 < fld, 5+7>8, fld1<fld2
>=, <=	більше або дорівнює (не менше), менш або дорівнює (не більше)	4<=fld, fld1>=fld2
<>	не дорівнює	fld1<>fld2,5<>5, fld*4<>100, 4+fld<>55
BETWEEN	між двома значеннями, включаючи їх	fld BETWEEN 6 AND 44, fld BETWEEN '1/1/2000' AND '31/12/2000'
IS NULL	значенням є NULL	fld IS NULL
LIKE	порівняння строки з шаблоном	'abc' LIKE '_bc'
IN	належність множині (колекції)	x IN ('a', 'b', 'c')

Предикати, задані ключовими словами NOT BETWEEN, IS NOT NULL, NOT LIKE, NOT IN, розглядаються аналогічно як заперечення відповідних предикатів, наведених в табл. 1.1.

Як видно, SQL має стандартні предикати для порівняння значень: = (дорівнює), < (менше), > (більше), <= (менше або дорівнює), >= (більше або дорівнює), <> (не дорівнює).

Оператор обчислює значення предикату на кожному рядку таблиці. Ті рядки, на яких він істинний, попадають в результат. Всі інші рядки відфільтровуються. В фразі WHERE можуть використовуватись будь-які атрибути таблиці, не обов'язково тільки ті, які попадають в результуючу таблицю.

Предикати порівняння можуть використовуватися з будь-якими типами атомарних даних, а не тільки з числами[1].

Строка може порівнюватись не тільки з іншою строкою, але і з шаблоном за допомогою предикату LIKE. Шаблон складається з звичайних символів[2], але деякі з них мають спеціальне значення, так звані спеціальні символи (wild card).

Символ % означає, що на його місті може бути довільна підстрока, в тому числі, порожня. Символ _ означає, що на його місті може бути будь-який символ.

Без використання спеціальних символів предикат LIKE діє таким самим чином як і предикат рівності.

Предикат BETWEEN дозволяє задавати діапазон значень.

Предикат IN перевіряє, чи міститься значення в сукупності значень. Елементи колекції розділяються комами, а сама колекція береться в круглі дужки.

Для опису предиката IS NULL треба розглянути невизначені значення в SQL. В мові є одне спеціальне значення NULL, яке інтерпретується як невизначене. Воно може записуватись як значення атрибуту будь-якого типу даних, якщо "звичайне" значення в даному рядку невідоме. Якщо в предикатах хоча б одне із значень буде NULL, то результатом буде спеціальне значення unknown[3], а не істина чи хиба, як в інших випадках. Це означає, що значенням предикату x=NULL, завжди буде unknown незалежно від того, яке значення має x. Для перевірки значення x на рівність NULL використовується спеціальний предикат IS NULL або його заперечення IS NOT NULL для перевірки на те, що x визначений.

---

[1] Таким чином, всі типи атомарних даних лінійно впорядковані.

[2] Які в цьому контексті використовуються автонімно.

[3] Третє логічне значення.

Якщо невизначене значення з’являється в виразі, то результатом (значенням) такого виразу буде NULL. На жаль, в SQL це правило виконується не в усіх випадках. Та і взагалі кажучи, більшість колізій мови пов’язана саме з NULL-значеннями. Ця властивість мови є найбільш дискусійною і значна частина критики мови відноситься саме до NULL-значень.

Як було сказано вище, предикати можуть повертати значення unknown разом з true та false. Це означає, що в SQL використовується трьохзначна логіка, замість стандартної двозначної (булевої).

Атомарні предикати з’єднуються в складні вирази булевими операціями кон’юнкції, диз’юнкції та заперечення, розширеними на трьохзначну логіку. Наведемо таблиці істинності для цих операцій[4].

Таблиця 1.2 – Таблиця істинності кон’юнкції

AND	**true**	**false**	**unknown**
true	true	false	unknown
false	false	false	false
unknown	unknown	false	unknown

Таблиця 1.3 – Таблиця істинності диз’юнкції

OR	**true**	**false**	**unknown**
true	true	true	true
false	true	false	unknown
unknown	true	unknown	unknown

[4] Взагалі кажучи, для задання бінарних операцій кон’юнкції та диз’юнкції треба домовитись про відповідність між аргументами операцій та рядками (стовпчиками) таблиць.

Таблиця 1.4 – Таблиця істинності заперечення

NOT	
true	false
false	true
unknown	unknown

При застосуванні в умові фільтрації одночасно різних логічних операцій необхідно враховувати, що вони мають різний пріоритет. В SQL пріоритет операцій задається наступною таблицею.

Таблиця 1.5 – Таблиця пріоритетів логічних операцій

Пріоритет	**Операція**
найвищий	атомарні предикати
	NOT
	AND
найменший	OR

Виникає питання, яким чином діє умова фільтрації в фразі WHERE для трьохзначної логіки. Ця умова обчислюється для кожного рядка таблиці, потім тільки ті рядки, на яких значення буде істинним, попадають в результат. Всі інші рядки, на яких значення буде хибним або невизначеним (тобто рівним unknown), відфільтровуються.

Імена атрибутів, які з’являються в вихідній таблиці, не обов’язково співпадають з іменами відповідних атрибутів вхідної таблиці. Вони можуть перейменовуватися за допомогою функції перейменування, яка позначається ключовим словом AS. Такі імена часто називають псевдонімами (alias). Наприклад, оператор

SELECT deptCode AS code FROM Projects;

поверне таблицю, єдиний атрибут якої буде мати ім’я code.

В реляційних базах таблиці трактуються як мультимножини, тобто рядки можуть мати дублікати. Для вилучення дублікатів використовується ключове слово DISTINCT.

Запити дозволяють створювати так звані обчислювальні або похідні атрибути. Ці атрибути (точніше їхні значення) конструюються з (значень) атрибутів вхідної таблиці за допомогою спеціальних функцій та операцій. Імена таких атрибутів розраховуються СУБД автоматично в залежності від реалізації. Крім того, можна задавати псевдонім за допомогою ключового слова AS.

Мова містить стандартні арифметичні операції додавання, віднімання, добутку та ділення, що позначаються відповідно +, -, *, /. Крім того, в SQL входять спеціальні арифметичні функції:

ABS(N) – абсолютне значення N,

CEIL[ING](N) – найменше ціле число, яке більше ніж N,

EXP(N) – експонента N,

FLOOR(N) – найбільше ціле число, яке менше ніж N,

LN(N) – натуральний логарифм N,

MOD(N, D) – залишок від N, поділеного на D,

POWER(B, E) – B піднесене до степеня E,

SQRT(N) – квадратний корінь з N.

Для символьних рядків в SQL визначена одна операція, а саме конкатенація рядків, яка позначається через ||, а також кілька наступних функцій:

- функція знаходження підрядка SUBSTRING(<*source*> FROM <*start*> [FOR <*length*>]). Функція повертає підрядок рядку <*source*>, починаючи з позиції <*start*>. Довжина підрядка дорівнює <*length*>. Якщо довжина не задана, то функція повертає всі символи, починаючи з позиції <*start*> і до кінця рядка. Якщо <*source*> є порожнім рядком або позиція <*start*> виходить за межи рядка, то функція повертає порожній рядок (тобто рядок нульової довжини);

- функція вилучення небажаних символів, що знаходяться в початку або кінці рядка TRIM([[LEADING | TRAILING | BOTH] [<*trim characters*>] FROM] <*source*>). Функція повертає рядок <*source*>, з якого вилучені всі символи <*trim*

characters>, які стоять в початку або кінці рядка. Якщо вказане ключове слово LEADING, то вилучаються символи з початку рядка; якщо вказане ключове слово TRAILING, то вилучаються символи з кінця рядка, якщо вказане ключове слово BOTH, то вилучаються символи як з початку так і з кінця рядка; якщо ж ключове слово не задане, то за умовчанням вважається, що стоїть BOTH. Якщо аргумент <*trim characters*> не заданий, то вважається що він дорівнює одному пробілу;

- функції переводу рядка в ніжній або верхній регістр LOWER(<*source*>), UPPER(<*source*>). Функція UPPER повертає рядок <*source*>, переведений в верхній регістр. Функція LOWER повертає рядок <*source*>, переведений в нижній регістр. Функції змінюють тільки букви алфавіту, а решту символів залишають незмінною;

- функція пошуку входження підрядка в рядок POSITION(<*substring*> IN <*source*>). Функція шукає перше входження (зліва-направо) підрядка <*substring*> в рядок <*source*> і повертає позицію першого символу підрядка <*substring*>. Якщо підрядок не знайдений, то повертається 0;

- функція CHARACTER_LENGTH(<*source*>) повертає довжину рядка <*source*>.

В якості аргументів функцій можуть стояти довільні вирази, які відповідають типу аргументу.

Для даних типу дати, часу та інтервалу в SQL використовуються як арифметичні операції, так і спеціальні функції. Прикладами арифметичних операцій є:

Interval * Numeric, Interval / Numeric – повертає значення типу Interval;

Numeric * Interval – повертає значення типу Interval;

Datetime + Interval, Datetime – Interval – повертає значення типу Datetime;

Interval + Datetime – повертає значення типу Datetime;

Datetime – Datetime – повертає значення типу Interval;

Interval + Interval, Interval – Interval – повертає значення типу Interval.

Функція EXTRACT(<*field label*> FROM <*source*>) вибирає з дати, часу або інтервалу їх окремі елементи. Функція повертає числове значення окремого поля з

<*source*>. Поле, яке треба вибрати, задається аргументом <*field label*>. Цей аргумент може приймати значення YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, TIMEZONE_HOUR або TIMEZONE_MINUTE.

Бінарні рядки є послідовностями 0 і 1. В SQL до них можна застосовувати майже всі операції і функції, які застосовуються до символьних рядків. Їх можна з'єднувати операцією конкатенації, вирізати підрядок, вилучати символи, застосовувати предикат LIKE. Однією з відмін є те, що для порівняння бінарних строк можна використовувати тільки предикати = та <>. Предикати < та > не дозволяються.

При написанні запиту досить часто зустрічається ситуація, коли дані одного типу треба перетворити в інший. Наприклад, число в рядок або навпаки. В деяких випадках SQL виконує таке перетворення автоматично. Таке автоматичне перетворення називається неявним перетворенням типу (implicit type conversion). В інших випадках таке перетворення потрібно задавати явно. Для цього використовується функція CAST(<*source expression*> AS <*result type*>). Вона перетворює дані, вказані в аргументі <*source expression*>, в відповідні дані типу, який задається аргументом <*result type*>. В деяких випадках таке перетворення може привести до втрати частини інформації. Наприклад, перетворення числа з плаваючою точкою до цілого числа або перетворення з типу TIMESTAMP, який містить дату і час, до типу DATE, який містить тільки дату. Можлива і інша ситуація, коли дані перетворюються до більш інформативного типу. Наприклад, з типу DATE до типу TIMESTAMP. В такому випадку для відсутньої частини інформації буде задаватися значення за умовчанням. Так, для TIMESTAMP поле часу буде встановлено в нульове значення.

При інтерпретації особливого значення NULL діє загальний принцип: якщо один з аргументів функції (предикату) є невизначеним, то і результат те ж буде невизначеним, тобто повертається значення NULL (unknown).

Розглянуті предикати і функції використовуються не тільки для побудови обчислювальних атрибутів, а і в умові фільтрації, яка стоїть в фразі WHERE.

1.2. Агрегатні функції

В SQL існує спеціальний клас функцій для обчислень на атрибутах таблиць, які (функції) дозволяють знаходити різні підсумкові значення. Такі функції називаються агрегатними функціями. За їх допомогою знаходяться такі дані, як сума значень виразу по всім рядкам таблиці, середнє значення і т.д. В табл. 1.6 наведені основні агрегатні функції SQL.

Таблиця 1.6 – Основні агрегатні функції

Функція	**Опис**	**Тип даних**	**Чи ігнорує NULL**	**Чи впливає застосування DISTINCT**
AVG	Середнє значення колекції	Numeric	Так	Так
MAX	Найбільше значення колекції	Будь-який	Так	Ні
MIN	Найменше значення колекції	Будь-який	Так	Ні
SUM	Сума значень колекції	Numeric	Так	Так
COUNT(*)	Кількість рядків таблиці	Будь-який	Ні	Не дозволяється
COUNT	Кількість елементів колекції	Будь-який	Так	Так

Вказані функції є унарними та застосовуються до таблиць. Всі функції, за винятком останньої функції COUNT(*), залежать від одного параметру. Цім параметром може бути атрибут або вираз на рядку, тобто такий вираз, який розраховується на атрибутах одного рядку.

Функція AVG розраховує середнє значення на рядках таблиці. Якщо на деякому рядку вираз приймає значення NULL, то такий рядок ігнорується. Аналогічним чином функція SUM розраховує суму значень на рядках таблиці. Рядки, на яких вираз дорівнює NULL, вилучаються з розрахунку. Аргумент функції повинен належати одному з числових типів.

Функції MIN та MAX знаходять найменше та найбільше значення виразу на рядках таблиці. Значення NULL так само ігнорується. Ці функції визначені на любих типах даних, для яких допускається порівняння.

Існує два види функції COUNT. Перший вид, який записується як COUNT(*), розраховує кількість всіх рядків, включаючи дублікати. Рядок враховується в результаті, навіть якщо всі значення його атрибутів дорівнюють NULL. Другий вид функції обчислює значення виразу на рядку і потім підраховує кількість значень, які не дорівнюють NULL.

Агрегатні функції можуть використовуватися сумісно з фразою WHERE. В такому випадку рядки таблиці спочатку відфільтровуються за умовою фрази WHERE, а потім на них розраховуються агрегатні функції.

На порожній таблиці функції AVG, SUM, MIN, MAX повертають значення NULL. Так як функції ігнорують значення NULL, то вони повертають NULL, коли всі значення є невизначеними. Функція COUNT другого виду в цих випадках повертає нуль.

По умовчанню враховуються всі значення (відмінні від NULL), включаючи дублікати значень. Для виключення дублікатів значень, відмінних від NULL, в функції вказується ключове слово DISTINCT.

Використання DISTINCT має сенс для AVG[5], SUM та функції COUNT, яка обчислює значення виразу на рядку, тобто функції COUNT другого виду. Для MIN, MAX воно є допустимим, але не впливає на результат. Для функції COUNT(*), яка обчислює кількість рядків, використання DISTINCT не дозволяється.

1.3. Групування даних

Якщо в фразі SELECT зустрічаються агрегатні функції, то результатом буде таблиця, яка складається з одного рядка. Але часто виникає ситуація, коли треба виконати агрегування над декількома групами рядків. Для цього в SQL введений оператор GROUP BY, який дозволяє задати умову для розбиття таблиці на групи (підтаблиці).

[5] Якщо казати більш точно, то вилучення дублікатів не впливає на результат функції AVG тоді і тільки тоді, коли кількість дублікатів кожного значення однакова.

При виконанні цього оператора в першу чергу виконується групування рядків з однаковим значенням атрибутів, перерахованих в фразі GROUP BY. Далі на отриманих групах, що є підтаблицями, виконуються агрегатні функції, які знаходяться в фразі SELECT. Потім отримані рядки об’єднуються в результуючу таблицю. Таким чином, результуюча таблиця містить один рядок на кожну групу. Всі рядки, для яких атрибути групування містять невизначені значення, об’єднуються в одну окрему групу. Тобто при групуванні значення предикату NULL=NULL буде істинним, а значення предикату NULL=<визначене значення> буде хибним, що суперечить загальному правилу SQL.

Коли використовується оператор GROUP BY, то в фразі SELECT можуть записуватись тільки атрибути, які є в фразі GROUP BY (атрибути групування), агрегатні функції та константні вирази. Відмітимо, що агрегатні функції не обов’язково повинні бути присутніми в фразі SELECT. Якщо вони відсутні, то такий запит діє аналогічно оператору DISTINCT. Іншими словами, кожна група в вихідній таблиці перетворюється в один рядок, який містить значення атрибутів групування, що співпадають для всіх рядків цієї групи.

Після групування рядків утворені групи можуть бути додатково відфільтровані. Це робиться предикатом в фразі HAVING. Таким чином, запит може мати два предиката для фільтрування. Перший знаходиться в фразі WHERE і виконується на рядках проміжної таблиці, яка є значенням табличного виразу фрази FROM, а другий знаходиться в фразі HAVING і виконується після групування. В цьому предикаті використовуються тільки атрибути групування і агрегатні функції.

Допускається використовувати фразу HAVING без GROUP BY при наявності агрегатних функцій в SELECT. В цьому випадку вся таблиця трактується як одна група. Результуюча таблиця буде складатися з одного рядка, якщо предикат в HAVING є істинним, або буде порожньою в протилежному випадку.

1.4. Операції з’єднання таблиць

В попередніх підрозділах всі запити виконувались на одній таблиці. Але популярність SQL в значній мірі пояснюється його здатністю виконувати багатотабличні запити. В таких запитах дві або більше таблиці з’єднуються по деякій умові. Фактично такі запити моделюють зв’язки між таблицями, дозволяючи відображати в реляційний моделі більш складні моделі даних.

Для випадку декартова з’єднання (CROSS JOIN) при виконанні запиту кожний рядок першої таблиці з’єднується з кожним рядком другої таблиці. Таким чином, утворюється нова таблиця, кількість рядків в якій дорівнює добутку кількостей рядків першої та другої таблиць. Множина атрибутів нової таблиці утворюється об'єднанням множин атрибутів першої та другої таблиць. Очевидно, що кількість атрибутів результуючої таблиці дорівнює сумі кількостей атрибутів першої та другої таблиць. Потім рядки таблиці відфільтровуються за умовою, яка задана в фразі WHERE.

Вище описано, як таблиці з’єднуються на логічному рівні. На фізичному рівні процедура з’єднання може значно відрізнятися від описаної за рахунок оптимізації.

Якщо таблиці, які з’єднуються, мають однакові імена атрибутів, то для уникнення неоднозначності ці імена в запитах уточнюються за допомогою префіксів, в якості яких виступають імена таблиць. Префікс відділяється від імені атрибута крапкою.

Але можлива ситуація, коли таблиця з’єднується сама з собою. Префікси з іменем таблиці не дозволяють розв’язати неоднозначність. В такому випадку використовуються табличні аліаси або псевдоніми, коли таблиці дається додаткове ім’я, яке використовується в якості префікса для атрибутів цієї таблиці. Аліаси також використовуються для скорочення довгих імен таблиць.

В наступних версіях SQL були введені окремі оператори з’єднання, які дозволяють синтаксично відокремити ці логічно різні операції.

Оператор внутрішнього з’єднання INNER JOIN з’єднує дві таблиці по заданій умові. Умова з’єднання записується після ключового слова ON. В фразі ON може стояти предикат будь-якої складності, а не тільки атомарний предикат.

В результуючу таблицю попадають тільки ті рядки, які мають відповідні рядки в іншій таблиці. Для того, щоб отримати всі рядки однієї з таблиць, на яких виконується з’єднання, використовується з’єднання спеціального типу, яке називається зовнішнім з’єднанням.

Оператор LEFT OUTER JOIN з’єднує попарно рядки першої (лівої) та другої таблиць, на яких виконується умова, записана в фразі ON. Після цього в першій таблиці могли залишиться рядки, що не попали в результат, тобто для яких не існує жодного рядку другої таблиці, на якому би виконувалась ця умова. Такі рядки теж додаються в результат, при цьому в якості значень атрибутів другої таблиці використовується невизначене значення NULL.

Оператор RIGHT OUTER JOIN діє аналогічним чином, тільки додаткові рядки беруться не з першої, а з другої (правої) таблиці. Нарешті, FULL OUTER JOIN поєднує в собі перші два типи зовнішнього з’єднання. А саме, в результат додаються як рядки першої таблиці, для яких не існує відповідного рядка другої таблиці, так і рядки другої таблиці, для яких не існує відповідного рядка першої.

1.5. Теоретико-множинні операції на таблицях

Таблиці є мультимножинами рядків, тому над ними можуть виконуватися аналоги звичайних теоретико-множинних операцій об’єднання, перетину та різниці.

Операція UNION виконує об’єднання двох таблиць. Вона має наступний синтаксис

<лівий оператор SELECT> UNION [ALL | DISTINCT] <правий оператор SELECT>;

Оператори SELECT повинні повертати таблиці, які є сумісними. Це означає, що кількість атрибутів в таблицях співпадає, а атрибути, які знаходяться на однакових позиціях в обох операторах, повинні мати типи даних, що можуть бути

перетворені до однакового типу. Імена атрибутів двох таблиць можуть не співпадати. При об’єднанні сумісних таблиць атрибути "з’єднуються" по позиціям, які вони займають в фразі SELECT обох операторів, тобто перший атрибут першої таблиці "з’єднується" з першим атрибутом другої таблиці і т.д. Імена атрибутів нової таблиці беруться з першого (лівого) оператора SELECT[6].

Операція об’єднання може враховувати дублікати рядків чи не враховувати. Якщо вказане ключове слово ALL, то об’єднання враховує дублікати рядків; якщо ж вказане ключове слово DISTINCT, то дублікати вилучаються з об’єднання (тобто кількість дублікатів буде дорівнювати одиниці). Якщо нічого не вказано, то за умовчанням вважається, що стоїть DISTINCT.

Операція перетину рядків INTERSECT має наступний вигляд

<лівий оператор SELECT> INTERSECT [ALL | DISTINCT] <правий оператор SELECT>;

Таблиці, які перетинаються, повинні бути сумісними, як і для операції об’єднання. Як і для об’єднання, існує два види цієї операції – з врахуванням дублікатів та їх вилученням, в залежності від наявності ключових слів ALL або DISTINCT. При перетину з урахуванням дублікатів, кількість дублікатів рядку буде дорівнювати найменшій кількості дублікатів цього рядку в першій та другій таблицях.

За умовчанням перетин таблиць вилучає дублікати. В якості імен атрибутів беруться імена атрибутів першої таблиці.

Операція різниці рядків EXCEPT має наступний вигляд

<лівий оператор SELECT> EXCEPT [ALL | DISTINCT] <правий оператор SELECT>;

Операція повертає всі ті рядки першої таблиці, яких немає в другій таблиці. Операція, як і попередні, може виконуватися як з урахуванням дублікатів рядків так і без, в залежності від того, яке ключове слово (ALL або DISTINCT) вказане. При відніманні з урахуванням дублікатів якщо в першій таблиці знаходиться n

[6] Таким чином, комутативність об’єднання втрачається; асоціативність, як показує безпосередня перевірка, залишається.

дублікатів деякого рядка, а в другій таблиці знаходиться m дублікатів того ж самого рядка, то в результуючий таблиці буде n-m дублікатів цього рядка, якщо n>m; в протилежному випадку рядок в результат не включається.

1.6. Підзапити

В запиті можуть використовуватися підзапити. Існує три типа підзапитів. Підзапити першого типу повертають таблицю, яка складається з однієї колонки (схема таблиці одноатрибутна) та одного рядка. Така таблиця неявним чином перетворюється в атомарне (скалярне) значення, з якого вона складається. Підзапити такого типу можуть з'являтися в будь-яких атомарних предикатах SQL, включаючи BETWEEN та LIKE. При цьому типи даних повинні відповідати один одному або допускати перетворення до спільного типу даних.

Підзапити другого типу повертають таблицю, яка складається з однієї колонки (тобто, як і раніше схема одноатрибутна), але може мати декілька рядків. Такі підзапити використовуються в предикатах IN, ANY, ALL.

Предикат IN, як і раніше, перевіряє належність елемента колекції. Тільки колекція задається вкладеним оператором SELECT. Він повертає одноатрибутну таблицю, яка неявним чином перетворюється в колекцію атомарних значень.

Якщо колекція порожня, то предикат IN завжди хибний. Аналогічним чином діє операція NOT IN. На порожній множині предикат NOT IN завжди істинний.

Узагальненням предикатів IN та NOT IN є предикати ANY та ALL. Предикат ANY може використовуватися разом з будь-якою операцією порівняння (предикатом) в SQL. Він повертає істинне значення, якщо порівняння виконується хоча б для одного з елементів колекції. Якщо ж операція порівняння не виконується для жодного елементу або колекція є порожньою, то повертається значення хиби. В усіх інших випадках значення предикату є невизначеним (unknown).

Предикат IN еквівалентний предикату =ANY. Але предикат NOT IN не еквівалентний предикату <>ANY.

Іншім предикатом є ALL. Він повертає істинне значення, якщо для всіх елементів колекції виконується порівняння або колекція є порожньою. Якщо хоча б для одного елемента колекції порівняння не виконується, то предикат повертає значення хиби. В усіх інших випадках значення є невизначеним (unknown). Предикат NOT IN буде еквівалентний предикату <>ALL.

Підзапити третього типу повертають таблицю загального виду, без обмежень на кількість рядків та атрибутів. Такі запити використовуються в предикаті EXISTS, який перевіряє, чи існують рядки в таблиці (тобто перевіряє таблицю на непорожність). Він повертає істинне значення, якщо таблиця, яка будується підзапитом, не є порожньою (тобто містить хоча б один рядок). Якщо ж таблиця порожня, то повертається значення хиби. Значення цього предикату не може бути невизначеним.

1.7. Оператори модифікації таблиць

Мова SQL містить три оператора для модифікації даних: це INSERT, DELETE та UPDATE. Після створення таблиці її треба наповнити даними. Це робиться оператором INSERT. Він має наступний синтаксис

INSERT INTO <table name>

[(<attribute>,...,<attribute>)]

VALUES (<expression>,...,<expression>);

Для вилучення рядків використовується оператор DELETE, який має наступний синтаксис

DELETE FROM <table name> [[AS] <alias>]

[WHERE <condition>];

Для зміни існуючих рядків використовується оператор UPDATE, який має наступний синтаксис

UPDATE <table name> [[AS] <alias>]

SET <attribute>=<expression>,...,<attribute>=<expression>

[WHERE <condition>];

РОЗДІЛ 2. SQL АЛГЕБРА

Метою розділу є побудова алгебри функцій маніпулювання даними табличного типу. Носій алгебри складається з функцій маніпулювання таблицями, рядками та атомарними даними, сигнатура містить композиції фільтрації, взяття повного образу, агрегації та суперпозиції.

Об’єкт дослідження – оператори мови SQL, які складаються з операторів SELECT, INSERT, DELETE, UPDATE.

Основні результати розділу: виділені композиції, які дають змогу задати семантику оператора SELECT. До них входять композиції фільтрації, взяття повного образу, агрегації. Визначена семантика операцій з’єднання, агрегатних функцій, групування та предикатів ALL, ANY, IN, EXISTS. Побудовано два варіанти алгебри: для таблиць, станами яких виступають множини рядків, та для таблиць, станами яких виступають мультимножини рядків.

2.1. Загальна характеристика мови SQL

Традиційно вважається, що семантичним базисом для SQL виступає реляційна алгебра Кодда [90, 91, 92, 93, 94, 95, 96] або її більш вдосконалені варіанти, такі, як, наприклад, таблична алгебра, дослідженню якої присвячені праці [17, 28, 30, 38, 40, 41, 105]. Але в реляційних алгебрах не враховуються деякі важливі властивості SQL, такі, як наявність спеціального значення NULL, а також дублікатів рядків в таблицях. Крім того, при аналізі структури запитів з’ясовується, що треба розглядати не тільки і не стільки функції, а і структури більш високого порядку – функціонали (оператори) [56, 60, 61]. Все це приводить до необхідності введення до розгляду алгебри принципово іншого рівня ніж реляційна алгебра Кодда, а саме алгебри, сигнатура якої містить спеціалізовані композиції [19, 67]. Така алгебра базується на композиційному підходу в програмуванні [56, 60, 61, 62, 63, 64, 68, 69]. Вона описана в роботах [8, 9, 10, 11, 14, 15, 85, 86, 87]. Нижче буде розглянутий її удосконалений варіант. В першу чергу це стосується визначення таблиць, які тепер розглядаються як пари вигляду

(множина/мультимножина рядків, схема) на відміну від попереднього варіанту, коли таблиця розглядалась як множина (мультимножина) рядків, а схема виводилася зі (співпадаючих) схем рядків. Нове визначення дозволяє однозначно встановлювати схему для порожньої таблиці, що більш відповідає семантиці мови. В попередньому варіанті порожній таблиці відповідала будь-яка схема. Модифіковані визначення агрегатних функцій. В SQL використовуються два типа агрегатних функцій, в залежності від того, враховують вони NULL чи ні.

При розгляді мови SQL та її семантичних структур спираємось на [8, 10, 11, 16, 19, 21, 46, 88, 89].

Мова SQL складається з багатьох операторів. Оператори SQL діляться на підкласи. До головних відносяться *мова опису даних* (DDL – Data Definition Language, в стандартах SDL – Scheme Definition Language) і *мова маніпуляційних дій* (DML – Data Manipulation Language).

Мова DDL містить засоби для специфікації об’єктів баз даних: визначення, модифікація та знищення доменів, таблиць, індексів, зображень (view), тригерів та збережених процедур.

Мова DML включає *оператори маніпулювання даними* та *оператори завершення транзакцій*. У свою чергу *оператори маніпулювання даними* поділяються на *оператори вибірки даних* SELECT (за іншою термінологією *запити*) і *оператори оновлення даних*: *вставки* INSERT, *редагування* UPDATE і *вилучення* DELETE. Шляхом виконання операторів завершення транзакцій можна фіксувати в базі даних зміни, проведені окремими операторами маніпулювання, або відміняти ці зміни (відповідно оператори COMMIT і ROLLBACK).

Мова SQL займає особливе місце серед мов програмування. Ця мова не дозволяє розробляти всю програму самостійно, як це має місце, наприклад, для C++ або Java. Вона є, в першу чергу, мовою запитів до бази даних і при програмуванні повинна використовуватися з іншою мовою, яка називається базовою мовою (host language). Точніше кажучи, оператори мови SQL вбудовуються в базову мову загального призначення. Це потребує наявності спеціальних засобів для вбудування операторів SQL. Головна проблема полягає в

тому, що SQL орієнтований на роботу з мультимножинам (множинами), в той час як мови програмування загального призначення на таки типи даних не розраховані. Ця проблема отримала назву невідповідність входів (impedance mismatch). Для погодження входів введена спеціальна структура даних, яка називається *курсором*. *Курсор* це засіб, який дозволяє за допомогою спеціальних операторів отримати порядковий доступ до результату запиту. Фактично *курсор* можна уявляти як послідовність рядків таблиці, над якими виконуються таки операції, як

- отримати перший рядок;
- отримати останній рядок;
- отримати наступний рядок;
- отримати попередній рядок.

Для використання SQL з базовою мовою застосовуються декілька засобів. До них відносяться спеціальні процедурні розширення SQL, такі як вбудований SQL та динамічний SQL (embedded and dynamic SQL), модульній SQL (SQL client modules) та CLI інтерфейс (Call level Interface).

Вбудований SQL дозволяє записувати оператори звичайного SQL в текст програми на базовій мові. Команди вбудованого SQL починаються з спеціального префіксу EXEC SQL. Перед компіляцією текст програми обробляється препроцесором, який розпізнає команди SQL і передає їх на компіляцію до СУБД. Команди SQL заміняються на виклики функцій, скомпільованих СУБД. Після цього програма компілюється як звичайно. Динамічний SQL є розширенням вбудованого, він дозволяє формувати і виконувати команди SQL в процесі виконання програми. Вбудований та динамічний SQL вимагають додаткових зусиль для зв’язування змінних SQL зі змінними базової мови та необхідність транслювати оператори вбудованого та динамічного SQL в синтаксичні конструкції базової мови. Тому такій спосіб є менш зручним, порівняно з іншими, і використовується на сьогодні рідко.

На відміну від вбудованого SQL в модульному команди відокремлені від базової мови і знаходяться в спеціальному модулі. Модуль складається з об’яв

тимчасових таблиць, курсорів та процедур. Кожна процедура містить рівно один оператор SQL. Програма на базовій мові викликає процедури, записані в модулі. В такому вигляді модулі не є дуже поширеними та корисними і майже не використовуються. Але більшість СУБД мають свої процедурні розширення SQL. Такі розширення включають, поряд з операторами SQL, які використовуються через посередництво вбудованого та динамічного SQL, стандартні оператори циклювання та розгалуження, об'яви змінних та тимчасових таблиць і дозволяють писати складні запити та виконувати інші операції над базою даних, які неможливо або досить складно записати за допомогою тільки операторів SQL. Ці процедури зберігаються в базі даних і можуть бути викликані з базової мови.

І на кінець, CLI інтерфейс використовує бібліотеки функцій для звернення до СУБД з базової мови і виконання запитів та інших команд SQL. Оператори мови в цьому випадку передаються як параметри відповідних функцій. На сьогодні це є основним засобом виконання операторів SQL в середовищі інших мов програмування.

Семантичні структури операторів оновлення даних є похідними від семантичних структур запитів. Наприклад, семантика оператора редагування даних полягає в модифікації рядків таблиці, що задовольняють деякому предикату; цей предикат і функція, яка модифікує рядки, є спрощеними випадками відповідних фраз оператора SELECT.

Дамо стислу характеристику семантичних структур запитів SQL. Почнемо з структур даних. Таблиці – це сукупності рядків, причому допускається повторення рядків у таблиці; отже, можна говорити про дублікати (екземпляри) рядків у таблиці[7]. Для специфікації таблиць без дублікатів, тобто звичайних множин рядків, в операторі SELECT застосовується ключове слово DISTINCT. При визначенні таблиць для недопущення дублікатів треба задавати первинний ключ або вказувати атрибути таблиці, які повинні бути унікальними. Рядки задаються своїми атрибутами та їх значеннями; значення атрибутів вибираються з

[7] Аналогічна ситуація має місце для такого комбінаторного об'єкту як комбінація з повтореннями.

доменів. Елементи всіх доменів лінійно впорядковані, причому у випадку похідних доменів порядок успадковується. Всі домени містять виділений елемент NULL – невизначене значення, яке обробляється спеціальним чином.

Розглянемо структуру запитів SQL, для цього наведемо синтаксичні конструкції в термінах БНФ.

<специфікація курсору> ::= <запит> [<фраза order by>]

<запит> ::= <терм запиту>
| <запит2>

<запит2>::= <запит> <таблична операция> <терм запиту>

<таблична операция> ::= UNION [<специфікатор табл. операції >]
| INTERSECT [<специфікатор табл. операції >]
| EXCEPT [<специфікатор табл. операції >]

<специфікатор табл. операції > ::= ALL
| DISTINCT

<терм запиту> ::= <специфікація запиту> | (<запит>)

<специфікація запиту> ::= (SELECT [ALL | DISTINCT] <список вибірки>
<табличний вираз>)

<табличний вираз> ::= <фраза from> [<фраза where>] [<фраза group by>] [<фраза having>]

Змістовну семантику запитів дамо, проводячи їх природну типізацію на три типи. Запити першого типу мають вигляд

SELECT <список вибірки>
FROM <табличний вираз> [WHERE <умова>]
[ORDER BY <список впорядкування>];

причому агрегатні функції в списку вибірки не використовуються.

Рис. 2.1. Схема інтерпретації запитів, які не використовують агрегатні функції та групування

У цьому випадку семантика списку вибірки задається деякою функцією над рядками. Ця функція рядкам проміжної таблиці, яка є значенням табличного виразу фрази FROM після фільтрації за умовою фрази WHERE, ставить у відповідність рядки результуючої таблиці. Змістовна семантика всього запиту уточнюється схемою, наведеною на рис. 2.1. Запитам першого типу відповідають n-арні функції над таблицями. Таблиці-результати містять рядки, які будується за єдиним рядком проміжної таблиці.

Запити другого типу мають той самий вигляд, що і розглянуті запити першого типу. Єдина відмінність полягає в тому, що список вибірки містить агрегатні функції, а список упорядкування відсутній. Причому, якщо агрегатні функції використовуються хоча б в одному елементі списку вибірки, то й усі інші елементи повинні також використовувати агрегатні функції. Змістовна семантика таких запитів зображена на рис. 2.2.

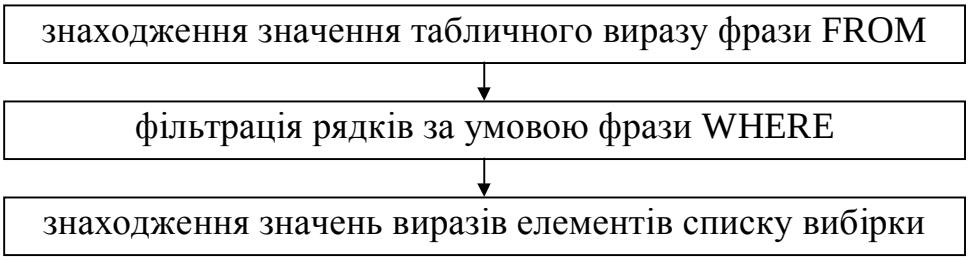

Рис. 2.2. Схема інтерпретації запитів без групування, які використовують агрегатні функції

Запитам другого типу відповідають n-арні функції над таблицями. Ці функції зіставляють вихідним таблицям однорядкові таблиці; єдиний рядок таблиць-результатів призначений для збереження значень агрегатних функцій (у загальному випадку значень виразів, які використовують агрегатні функції). Зауважимо, що внаслідок одноелементності таблиць-результатів список упорядкування в запитах цього типу зайвий.

Нарешті, запити третього типу використовують групування та агрегатні функції. Вони мають вигляд

SELECT < список вибірки >

FROM < табличний вираз > [WHERE < умова >]

GROUP BY < список групування > [HAVING < умова >]

[ORDER BY < список упорядкування >];

Змістовна семантика таких запитів зображена на рис. 2.3. Запитам третього типу відповідають n-арні функції над таблицями; але на відміну від запитів попередніх типів кожний рядок результуючої таблиці будується за підтаблицею проміжної таблиці; такі рядки призначені для зберігання значень агрегатних функцій на підтаблицях. Отже, групування передбачає використання агрегатних функцій.

Підкреслимо, що всі операції проводяться над таблицями, що в загальному випадку містять рядки з дублікатами.

Результати інтерпретації декількох операторів SELECT можна об’єднувати, перетинати, віднімати, якщо вони сумісні. Сумісність в найпростішому випадку означає односхемність таблиць; цю вимогу можна дещо послабити, вимагаючи однакову кількість і узгодженість доменів атрибутів таблиць, що об‘єднуються; для цього між атрибутами таблиць встановлюється бієкція згідно з порядком атрибутів, вказаних в фразах SELECT оператора запиту. Кількість дублікатів у результаті залежить від використання операцій об’єднання UNION DISTINCT (дублікати ігноруються) чи UNION ALL (дублікати враховуються). Аналогічно для операцій перетину та різниці таблиць.

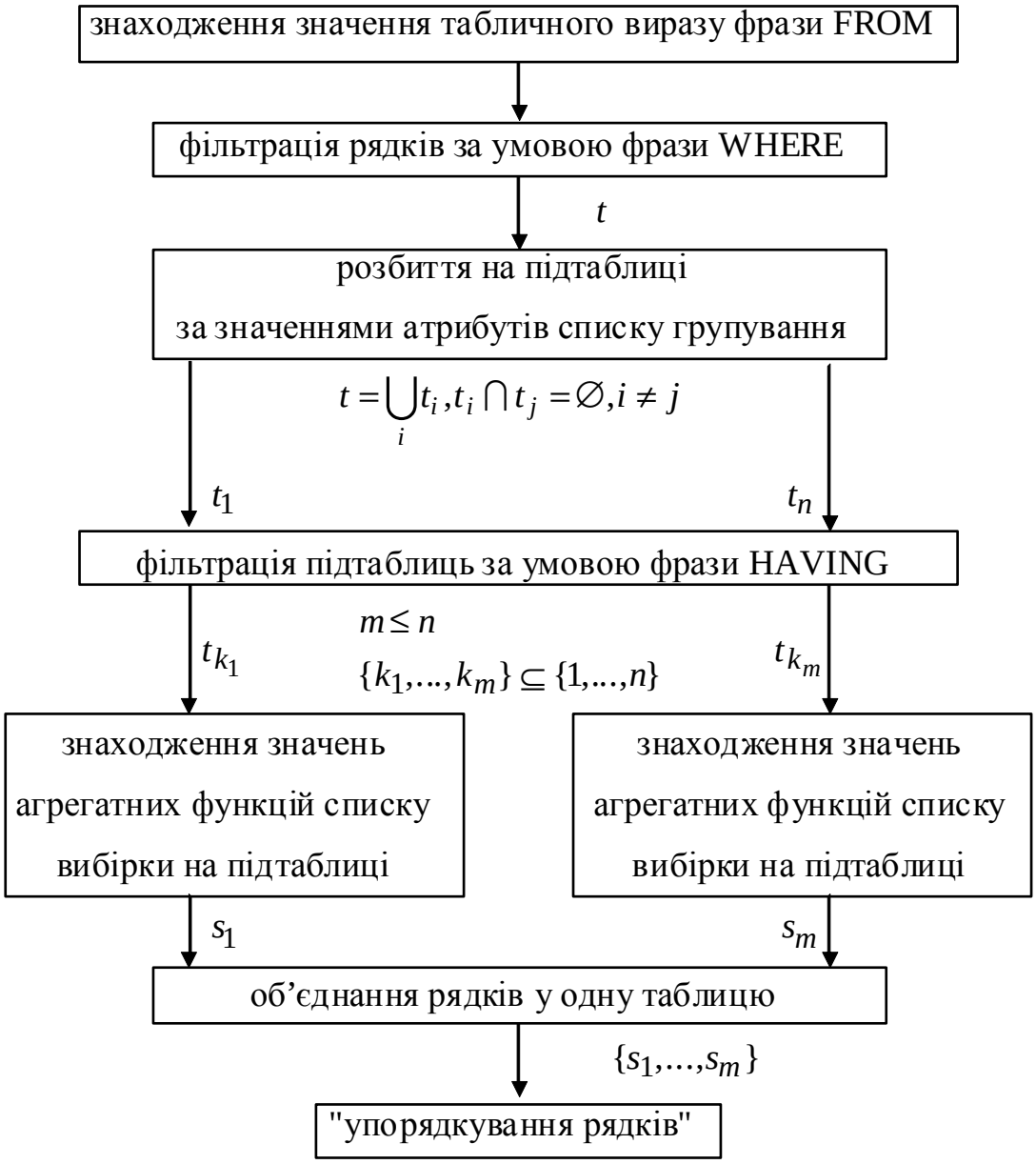

Рис. 2.3. Схема інтерпретації запитів загального вигляду з групуванням

Що стосується фрази ORDER BY, то вона призначена для "упорядкування" рядків результуючої таблиці; відповідне бінарне відношення на рядках є передпорядком і по суті будується лексикографічним добутком вихідних або інверсних порядків на доменах атрибутів фрази ORDER BY.

Отже, для аналізу семантичних структур SQL-подібних мов необхідно принаймні дослідити моделі реляційних структур даних та операції з'єднання, об'єднання, перетину, різниці, перейменування, а також механізми групування та упорядкування.

В розділі аналізуються існуючі моделі реляційних структур та обґрунтовується необхідність введення саме уточнень, розглянутих в роботі. Наводиться характеристика найсуттєвіших особливостей цих уточнень, які відрізняють їх від уточнень, що розглядаються в літературі; при аналізі будемо спиратися на [19, 59, 67].

Добре відомо, що моделі баз даних, зокрема реляційні, мають щонайменше два аспекти: інформаційний, пов’язаний з уточненням структур даних, і маніпуляційний, пов’язаний з уточненням операцій над такими структурами. Уточнення в сукупності вказаних аспектів зводиться до побудови відповідної алгебри структур даних.

Зрозуміло, що основне поняття, яке складає суть реляційного підходу і тому підлягає уточненню в першу чергу, – це поняття *реляції*. Як всяке інтуїтивне поняття реляція потенційно припускає багато уточнень. Спочатку як одне з таких уточнень Коддом було вибране класичне поняття скінченномісного відношення як підмножини декартова добутку множин, взятих у деякому порядку; іншими словами, відношенням є множина кортежів фіксованої довжини, тобто впорядкованих n-ок для n=1,2,... На початковому етапі таке уточнення обумовило успішність та популярність реляційного підходу в базах даних. Дійсно, змістовні поняття уточнювались і досліджувались за допомогою добре розвинутого формального апарату (теорія відношень, фрагменти мов логіки предикатів першого порядку та ін.) [6, 9, 13, 14, 15, 18, 25, 27, 47, 50, 51].

Разом з тим, традиційне теоретико-множинне уточнення реляції має ряд недоліків. Наприклад, інформаційний зміст реляції, взагалі кажучи, не залежить від порядку слідування компонент, а для відношення цей порядок суттєвий (зокрема, не завжди є сенс розглядати обернену реляцію, або за іншою термінологією інверсну, аналогічно з поняттям симетричності реляцій).

Використання стандартних імен 1,2,... для доступу до компонент елементів реляції (кортежам у даному випадку) обтяжливе, що, зокрема, призводить до появи в метамові, а не в мові-об’єкті довільних (так званих "мнемонічних") імен

як позначень стандартних. Крім того, при використанні тільки стандартних імен проблематика, пов’язана з перейменуванням, залишається взагалі поза розглядом.

Що стосується уточнень операцій, то за винятком теоретико-множинних, вихідні математичні операції зазнали суттєвої модифікації при перенесенні у реляційні бази даних. Навіть при розгляді бінарних теоретико-множинних операцій (об’єднання, перетину, різниці) доводиться обмежувати їх на відношення однакової арності між одними і тими ж доменами.

Результатом традиційного декартова добутку відношень є (універсальне) бінарне відношення, яке складають всі можливі пари кортежів відношень-аргументів, а результатом декартова добутку відношень в реляційній моделі (розширеного декартова добутку за термінологією [5] або геометричного добутку згідно з [48]) – відношення арності n+m, де n,m – арності відношень-аргументів, яке складають кортежі, що отримуються конкатенацією вихідних кортежів. Ця, на перший погляд, невелика різниця в означеннях призводить до суттєвих розбіжностей властивостей цих двох операцій добутку відношень: так розширений декартів добуток асоціативний, а вихідний класичний декартів добуток ні. Аналогічно класична операція проектування відношення за компонентою дає множину, а операція проектування відношення в реляційній моделі – відношення меншої арності, ніж відношення-аргумент.

Зазначимо характерний факт: операція добутку (за іншою термінологією композиція, послідовне застосування або згортка де Моргана) бінарних відношень, що складає ядро алгебри відношень (див., наприклад, [52]), не грає такої центральної ролі в реляційній моделі. Аналогами операції добутку виступають різноманітні параметричні операції з’єднання. Так, операція добутку відношень моделюється за допомогою операцій θ-з’єднання та проекції, де параметр θ – це предикат перевірки рівності специфікованих компонент кортежів.

Крім того, багато операцій над відношеннями, що традиційно розглядались в математиці, взагалі не ввійшли в реляційну модель. Це стосується операцій інверсії, різних замикань та інших [43]. Саме з цього факту, наприклад, випливає відоме твердження про неповноту реляційної алгебри Кодда: операція

рефлексивно-транзитивного замикання непохідна відносно сигнатурних операцій див., наприклад, [43].

Враховуючи прикладний аспект, зазначимо, що сучасні реляційні (табличні) СУБД в розумінні вказаного вище уточнення структур даних такими не є, оскільки допускають, наприклад, довільні імена компонент та дублікати рядків таблиць.

Отже, існують передумови, як теоретичного, так і практичного характеру, для інших "розширювальних" інтерпретацій реляцій. Починаючи з перших робіт Кодда і часто в неявній формі (подробиці див., наприклад, в [18, 50, 51]), такі спроби робилися в рамках базового поняття відношення.

По-перше, вводилися довільні імена (так звані імена атрибутів або просто атрибути) замість стандартних, і реляція уточнювалася як трійка, яку складають відношення, множина імен і бієкція між цими іменами і стандартними. Таким чином, формально довільні імена в уточненнях реляцій з’являються, але вони не несуть ніякого семантичного навантаження.

По-друге, на множині відношень вводилося відношення еквівалентності, яке індукується перестановками компонент, і реляція уточнювалась як клас еквівалентності (так званий зв’язок – від англ. relationship). Змістовно кажучи, реляція при цьому є відношенням з точністю до перестановки компонент (стовпчиків). При цьому довільні імена компонент (атрибути) взагалі відсутні в моделях реляцій, а з’являються, як і раніше, лише в метамові, і потенційна можливість маніпуляцій з ними (наприклад обчислення імен з подальшим засиланням і/або вибором за ними денотатів) залишається нереалізованою. Але розгляд таких маніпуляцій з іменами складає одну з провідних тенденцій у розвитку сучасних СУБД.

При вищезазначених уточненнях реляцій операції над ними вводяться досить громіздко (наприклад, згадувана еквівалентність в класі відношень повинна бути додатково стабільною відносно можливих операцій над відношеннями-представниками класів еквівалентностей, тобто бути конгруенцією).

При решті уточнень під реляцією розуміється множина функціональних відношень з однаковою областю означення: відношення в розумінні [10, 11, 26], функціональні дані [49], позиційні множини [44, 65].

Уточнення реляції, запропоноване в [42] і модифіковане у цьому розділі, найбільш близьке до згаданих уточнень у вигляді функціональних даних та позиційних множин. При такому уточненні розглядається один універсальний домен. Це дозволяє дослідити загальні властивості табличних маніпуляцій, не вводячи до розгляду несуттєві деталі, та спростити міркування.

Незважаючи на те, що традиційно під реляцією розуміється скінченна множина (мультимножина) рядків, і ця обставина врахована у подальших означеннях, як правило, математичні твердження про стандартні властивості уточнень реляційних операцій залишаються вірними і для нескінченних реляцій. Підкреслимо, що означення операцій над уточненнями реляцій інваріантні відносно потужності аргументів; крім того, можна піти далі й розглядати нескінченні схеми таблиць.

2.2. Основні операції на таблицях

У цьому підрозділі будується програмна алгебра, орієнтована на моделювання семантичних структур операторів маніпулювання даними SQL-подібних мов.

Уточнимо реляції в термінах іменних множин [17, 60]. Зафіксуємо наступні дві множини: $\boldsymbol{A}$, елементи якої назвемо *атрибутами*, і D – *універсальний домен*. Будемо вважати, що універсальному домену належить спеціальний елемент *null*, який інтерпретується як невизначене значення. Він виступає семантикою ключового слова NULL в мові SQL. Далі ці множини гратимуть роль множин імен та денотатів відповідно. Довільну (скінченну) множину атрибутів $R \subseteq \boldsymbol{A}$ назвемо *схемою*.

Рядком схеми R називається іменна множина на парі R, D (тобто множина пар (атрибут, значення)), проекція якої за першою компонентою рівна R. *Таблицею схеми* R називається пара $(\bar{t}, R)$, де $\bar{t}$ – (скінченна) множина рядків

схеми R, яка називається *станом таблиці*. Відмітимо, що в більшості випадків схема таблиці може бути відновлена по множині рядків. Але для порожньої множини рядків це неможливе.

Множину всіх рядків (таблиць) схеми R позначимо $S(R)$ (відповідно $T(R)$), а множину всіх рядків (таблиць) – S (відповідно T). Таким чином, $S \stackrel{def}{=} \bigcup_{R \subseteq \boldsymbol{A}} S(R)$, $T(R) \stackrel{def}{=} \{(\bar{t}, R) \mid \bar{t} \in 2^{S(R)}\}$, $T \stackrel{def}{=} \bigcup_{R \subseteq \boldsymbol{A}} T(R)$. Схема може бути порожньою; при цьому існує єдиний рядок схеми $\varnothing$, який позначається ε. Рядки будемо позначати $s, s_1, s_2, \ldots$, таблиці – $t, t_1, t_2, \ldots$, стани таблиць – $\bar{t}, \bar{t}_1, \bar{t}_2, \ldots$ а схеми – $R, R_1, R_2, \ldots$.

Введемо деякі означення. Проекцію рядка за першою компонентою будемо позначати $\pi_1^2 s$. Відношення односхемності таблиць визначається як $t_1 \cong t_2 \stackrel{def}{\Leftrightarrow} \exists R(t_1 \in T(R) \,\&\, t_2 \in T(R))$. Відношення односхемності рядків $\cong_S$ визначається аналогічно відношенню односхемності таблиць. Бінарне відношення сумісності рядків $s_1 \approx s_2 \stackrel{def}{\Leftrightarrow} \underset{A \in R_1 \cap R_2}{\&} s_1(A) = s_2(A)$ (рядки, що не мають спільних атрибутів, тобто $R_1 \cap R_2 = \varnothing$, вважаються сумісними). Часткова бінарна операція об'єднання сумісних рядків визначається так: $\overline{\cup}: S \times S \stackrel{\sim}{\to} S$, $\mathrm{dom}\overline{\cup} \stackrel{def}{=} \{(s_1, s_2) \mid s_1 \approx s_2\}$, $s_1 \overline{\cup} s_2 \stackrel{def}{=} s_1 \cup s_2$.

Будемо говорити, що рядок належить таблиці, якщо рядок належить стану таблиці.

2.2.1. Теоретико-множинні операції

Об'єднання, перетин і різниця таблиць $\cup_t, \cap_t, \setminus_t : T \times T \stackrel{\sim}{\to} T$. Областю означеності цих операцій є множина пар односхемних таблиць $\{(t_1, t_2) \mid t_1 \cong t_2\}$, а стан результуючої таблиці знаходиться, відповідно, як теоретико-множинний перетин, об'єднання і різниця станів таблиць-аргументів. Очевидно, що схема результуючої таблиці збігається зі схемами вихідних таблиць.

2.2.2. Операції внутрішнього з'єднання

Під *декартовим з'єднанням* (*cross join, Cartesian join*) розуміється часткова операція вигляду $Cj: T \times T \tilde{\to} T$,

$$\mathrm{dom}\, Cj \stackrel{def}{=} \{(t_1, t_2) \mid \exists R_1 \exists R_2 (t_1 \in T(R_1) \,\&\, t_2 \in T(R_2) \,\&\, R_1 \cap R_2 = \varnothing)\},$$

Стан таблиці визначається формулою $\{s \mid \exists s_1 \exists s_2 (s_1 \in \bar{t}_1 \,\&\, s_2 \in \bar{t}_2 \,\&\, s = s_1 \cup s_2)\}$.

Схема результуючої таблиці будується шляхом об'єднання схем вихідних таблиць.

Під *внутрішнім природним з'єднанням* (*inner natural join*), а за іншою термінологією *з'єднанням* або *еквіз'єднанням*, розуміється всюди визначена операція вигляду $\otimes: T \times T \to T$. Стан таблиці визначається формулою $\{s \mid \exists s_1 \exists s_2 (s_1 \in \bar{t}_1 \,\&\, s_2 \in \bar{t}_2 \,\&\, s = s_1 \cup s_2 \,\&\, s_1 \approx s_2)\}$.

Схема результуючої таблиці, як і в попередньому випадку, будується шляхом об'єднання схем вихідних таблиць.

Отже, з'єднання табличних алгебр у SQL-подібних мовах з'являється під назвою внутрішнього природного з'єднання; з'єднання докладно вивчається у [41 підрозд. 2.7-2.10].

Під *внутрішнім з'єднанням за атрибутами* $A_1, \ldots, A_n$, $n \geq 1$ (*inner join using* $A_1, \ldots, A_n$) розуміється часткова параметрична операція вигляду $\underset{A_1,\ldots,A_n}{\otimes} : T \times T \tilde{\to} T$,

$$\mathrm{dom} \underset{A_1,\ldots,A_n}{\otimes} \stackrel{def}{=} \{(t_1, t_2) \mid \exists R_1 \exists R_2 (t_1 \in T(R_1) \,\&\, t_2 \in T(R_2) \,\&\, R_1 \cap R_2 = \{A_1, \ldots, A_n\})\}.$$ Стан таблиці визначається формулою $\{s \mid \exists s_1 \exists s_2 (s_1 \in \bar{t}_1 \,\&\, s_2 \in \bar{t}_2 \,\&\, s = s_1 \cup s_2 \,\&\, \overset{n}{\underset{i=1}{\&}} s_1(A_i) = s_2(A_i))\}$.

Схема результуючої таблиці, як і в попередньому випадку, будується шляхом об'єднання схем вихідних таблиць.

Нехай $p: S \times S \tilde{\to} \{true, false\}$ – взагалі кажучи, частковий бінарний предикат на множині всіх рядків S, такий, що виконується імплікація

$$\forall s_1 \forall s_2 ((s_1, s_2) \in \mathrm{dom}\, p \;\&\; p(s_1, s_2) \tilde{=} true \Rightarrow s_1 \approx s_2).$$

Тут і далі $\tilde{=}$ позначає узагальнену рівність, що використовується з огляду на частковість функцій.

Під *внутрішнім з'єднанням за предикатом* p (*inner join on* p) розуміється, взагалі кажучи, часткова операція вигляду $\underset{p}{\otimes}: T \times T \tilde{\to} T$,

$\mathrm{dom} \underset{p}{\otimes} \overset{def}{=} \{((\bar{t}_1, R_1), (\bar{t}_2, R_2)) \mid \bar{t}_1 \times \bar{t}_2 \subseteq \mathrm{dom}\, p\}$. Стан таблиці визначається формулою $\{s \mid \exists s_1 \exists s_2 (s_1 \in \bar{t}_1 \;\&\; s_2 \in \bar{t}_2 \;\&\; s = s_1 \cup s_2 \;\&\; p(s_1, s_2) \tilde{=} true)\}$.

Схема результуючої таблиці, як і раніше, будується шляхом об'єднання схем вихідних таблиць.

Зауважимо, що в мові SQL декартове з'єднання – це тотальна операція, тотальність досягається попереднім перейменуванням спільних атрибутів. При з'єднанні за атрибутами $A_1, \ldots, A_n$ перетин схем таблиць-аргументів включає перелічені атрибути. Спільні атрибути таблиць-аргументів, які відрізняються від атрибутів $A_1, \ldots, A_n$, перед з'єднанням також перейменовуються.

Маючи наведене вище уточнення операцій внутрішнього з'єднання, можна зробити два висновки. По-перше, операція з'єднання $\otimes$ займає особливе місце, оскільки вона є розширенням довільної іншої операції внутрішнього з'єднання у наступному розумінні:

$$t_1 Cj t_2 = t_1 \otimes t_2,\ t_1 \underset{A_1, \ldots, A_n}{\otimes} t_2 = t_1 \otimes t_2,\ \overline{t_1 \underset{p}{\otimes} t_2} \subseteq \bar{t}_1 \otimes \bar{t}_2,$$

за умови визначення значень операцій у лівих частинах цих виразів[8].

---

[8] Отже, з'єднання є розширенням декартова з'єднання та внутрішнього з'єднання за атрибутами (за предикатом) у стандартному розумінні: $Cj \subseteq \otimes$, $\underset{A_1, \ldots, A_n}{\otimes} \subseteq \otimes$, $\otimes_p \subseteq \otimes$.

По-друге, параметрична операція з'єднання за предикатом також займає особливе місце, бо решта операцій є операціями з'єднання за відповідними нижченаведеними конкретними предикатами:

$$p_{Cj}: S \times S \overset{\sim}{\to} \{true\},\ \mathrm{dom}\, p_{Cj} \overset{def}{=} \{(s_1, s_2) \mid \pi_1^2 s_1 \cap \pi_1^2 s_2 = \varnothing\},\ p_{Cj}(s_1, s_2) = true;$$

$$p_{\otimes}: S \times S \to \{true, false\},\ p_{\otimes}(s_1, s_2) = true \overset{def}{\Leftrightarrow} s_1 \approx s_2;$$

$$p_{A_1,\ldots,A_n}: S \times S \overset{\sim}{\to} \{true, false\},\ \mathrm{dom} p_{A_1,\ldots,A_n} \overset{def}{=} \{(s_1, s_2) \mid \pi_1^2 s_1 \cap \pi_1^2 s_2 = \{A_1,\ldots,A_n\}\},$$

$$p_{A_1,\ldots,A_n}(s_1, s_2) \overset{\sim}{=} true \overset{def}{\Leftrightarrow} s_1 \approx s_2 \overset{def}{\Leftrightarrow} \&_{i=1}^{n} s_1(A_i) = s_2(A_i).$$

Іншими словами, p_{Cj} – частковий константний предикат, істинний на рядках зі схемами, які не перетинаються; $p_{\otimes}$ – тотальний предикат, істинний на сумісних рядках; нарешті, $p_{A_1,\ldots,A_n}$ – частковий предикат, істинний на рядках, які мають однакові значення всіх атрибутів $A_1,\ldots,A_n$; причому предикат визначений тільки на рядках, спільними атрибутами яких є саме вказані атрибути $A_1,\ldots,A_n$.

2.2.3. Операції зовнішнього з'єднання

Всі операції зовнішнього з'єднання підпорядковані своїм операціям внутрішнього з'єднання і задаються за однією логічною схемою. Опишемо цю схему.

Нехай $\varphi: T \times T \overset{\sim}{\to} T$ – деяка часткова операція на множині таблиць, причому виконується включення $\varphi(t_1, t_2) \subseteq t_1 \otimes t_2$ для всіх $t_1, t_2 \in \mathrm{dom}\varphi$. Нагадаємо, що операції внутрішнього з'єднання Cj, $\otimes$, $\underset{A_1,\ldots A_n}{\otimes}$, $\underset{p}{\otimes}$ саме такі.

Зафіксуємо таблиці t_1, t_2 з області означеності операції φ. Тоді перша (ліва в інфіксному записі) таблиця припускає наступне розбиття: $t_1 = (t_1 \underset{\varphi}{\cap} t_2) \cup_t (t_1 \underset{\varphi}{-} t_2)$, де стан перетину $t_1 \underset{\varphi}{\cap} t_2$ визначається виразом

$\{s_1 \mid s_1 \in \bar{t}_1 \,\&\, \exists s_2 (s_2 \in \bar{t}_2 \,\&\, s_1 \cup s_2 \in \varphi(t_1, t_2))\}$, а стан різниці $t_1 \underset{\varphi}{-} t_2$ – виразом $\{s_1 \mid s_1 \in \bar{t}_1 \,\&\, \forall s_2 (s_2 \in \bar{t}_2 \Rightarrow s_1 \cup s_2 \notin \overline{\varphi(t_1, t_2)})\}$.

Кажучи змістовно, рядки з підтаблиці $t_1 \underset{\varphi}{\cap} t_2$ використовуються в формуванні результату з'єднання, а рядки з підтаблиці $t_1 \underset{\varphi}{-} t_2$ не використовуються, оскільки їх продовжень в результаті з'єднання немає. Зауважимо, що за термінологією [16, п. 3.4.1, с. 130] таблиця $t_1 \underset{\varphi}{\cap} t_2$ називається *напівз'єднанням таблиць* t_1, t_2. Операції лівого, правого і повного зовнішнього з'єднання як раз і призначені для врахування рядків таблиць-аргументів, які не попали в результат вихідного внутрішнього з'єднання. Але при цьому виникає питання: як розширити рядки до надсхеми. Для цього в SQL використовується особливий елемент універсального домену *null*.

Нижче через s_R^{null} позначимо наступний константний рядок $s_R^{null}: R \to \{null\}$. Задамо безпосередньо операції зовнішнього з'єднання, індуковані операцією внутрішнього з'єднання φ. Для цього знадобляться наступні природні з'єднання, в яких R_1, R_2 – схеми таблиць t_1, t_2 відповідно: $(t_1 \underset{\varphi}{-} t_2) \otimes (\{s_{R_2 \setminus R_1}^{null}\}, R_2 \setminus R_1)$, $(\{s_{R_1 \setminus R_2}^{null}\}, R_1 \setminus R_2) \otimes (t_2 \underset{\varphi}{-} t_1)$. Стани результуючих таблиць розраховуються відповідно виразами

$$\overline{(t_1 \underset{\varphi}{-} t_2)} \otimes \{s_{R_2 \setminus R_1}^{null}\} = \{s_1 \cup s_{R_2 \setminus R_1}^{null} \mid s_1 \in \overline{t_1 \underset{\varphi}{-} t_2}\}$$

$$\{s_{R_1 \setminus R_2}^{null}\} \otimes \overline{(t_2 \underset{\varphi}{-} t_1)} = \{s_{R_1 \setminus R_2}^{null} \cup s_2 \mid s_2 \in \overline{t_2 \underset{\varphi}{-} t_1}\}$$

Схеми таблиць-результатів отримуються шляхом об'єднання схем вихідних таблиць.

Під *зовнішнім лівим з'єднанням, індукованим операцією* φ (*outer left join*), розуміється операція вигляду $\varphi_l: T \times T \overset{\sim}{\to} T$, $\mathrm{dom}\varphi_l \overset{def}{=} \mathrm{dom}\varphi$, стан результуючої таблиці $\varphi_l(t_1, t_2)$ розраховується формулою $\overline{\varphi(t_1, t_2)} \cup (\overline{(t_1 \underset{\varphi}{-} t_2)} \otimes \{s_{R_2 \setminus R_1}^{null}\})$.

Під *зовнішнім правим з'єднанням* (*outer right join*) розуміється операція вигляду $\varphi_r : T \times T \tilde{\to} T$, $\mathrm{dom}\varphi_r \overset{def}{=} \mathrm{dom}\varphi$, стан результуючої таблиці $\varphi_r(t_1, t_2)$ розраховується формулою $\overline{\varphi(t_1, t_2)} \cup (\{s^{null}_{R_1 \setminus R_2}\} \otimes \overline{(t_2 -_{\varphi} t_1)})$.

Під *повним зовнішнім з'єднанням* (*outer full join*) розуміється операція вигляду $\varphi_f : T \times T \tilde{\to} T$, $\mathrm{dom}\varphi_f \overset{def}{=} \mathrm{dom}\varphi$, стан результуючої таблиці розраховується формулою $\overline{\varphi(t_1, t_2)} \cup (\overline{(t_1 -_{\varphi} t_2)} \otimes \{s^{null}_{R_2 \setminus R_1}\}) \cup (\{s^{null}_{R_1 \setminus R_2}\} \otimes \overline{(t_2 -_{\varphi} t_1)})$.

Під *зовнішнім з'єднанням об'єднанням* (*union join*) розуміється операція вигляду $\varphi_\cup : T \times T \tilde{\to} T$, $\mathrm{dom}\varphi_\cup \overset{def}{=} \mathrm{dom}\varphi$, стан результуючої таблиці розраховується формулою $(\overline{(t_1 -_{\varphi} t_2)} \otimes \{s^{null}_{R_2 \setminus R_1}\}) \cup (\{s^{null}_{R_1 \setminus R_2}\} \otimes \overline{(t_2 -_{\varphi} t_1)})$.

Схеми результуючих таблиці, як і раніше, будується шляхом об'єднання схем вихідних таблиць.

Таким чином, ліве з'єднання призначене для врахування рядків першої (лівої) таблиці, які не беруть участі у внутрішньому з'єднанні, праве з'єднання – для врахування рядків другої (правої) таблиці, які не беруть участі у внутрішньому з'єднанні, повне з'єднання – для врахування рядків обох таблиць-аргументів, які не беруть участь у внутрішньому з'єднанні. Нарешті, з'єднання об'єднанням, на відміну від попередніх операцій, не поповнює результат внутрішнього з'єднання, а будує лише відповідним чином розширені рядки обох таблиць-аргументів, які не беруть участі у внутрішньому з'єднанні.

2.2.4. Структура операцій з'єднання

Повернемося до операцій внутрішнього з'єднання SQL. Для декартового з'єднання зовнішні з'єднання не має сенсу вводити, тому що ліве, праве і повне з'єднання збігаються з ним, а з'єднання об'єднанням завжди будує порожню таблицю (тому, що всі рядки таблиць-аргументів беруть участь у формуванні результату декартова з'єднання). Крім того, у сучасному SQL, включаючи останні версії, зовнішнє з'єднання об'єднанням підтримується тільки для природного

з'єднання (аналізувались доступні джерела). Структура сім'ї операцій з'єднання SQL-подібних мов наведена в табл. 2.1.

Зауважимо, що попередні конструкції треба модифікувати для врахування особливої ролі елемента *null* при визначенні сумісних рядків: рядки сумісні, якщо спільні атрибути мають однакові значення у рядках, причому значення відрізняються від *null*. Тієї ж мети можна досягти, розглядаючи наступне уточнення рівності у тризначній логіці:

$$d_1 =_S d_2 \overset{def}{=} \begin{cases} true, \text{якщо } d_1 = d_2 \ \& \ d_1 \neq null \ \& \ d_2 \neq null, \\ false, \text{якщо } d_1 \neq d_2 \ \& \ d_1 \neq null \ \& \ d_2 \neq null, \\ unknown, \text{якщо } d_1 = null \vee d_2 = null; \end{cases}$$

для всіх $d_1, d_2 \in D$; тут *unknown* – невизначене логічне значення. Тоді *відношення сумісності*, якому відповідає вже тризначний предикат, модифікується так: $s_1 \approx s_2 \overset{def}{\Leftrightarrow} \underset{A \in R_1 \cap R_2}{\&} s_1(A) =_S s_2(A)$, де R_i – схема рядка s_i, $i = 1,2$. Зауважимо, що у SQL-подібних мовах рівність уточнюється саме як предикат $=_S$. Крім того, наведене означення є проявом загальної ситуації при розширенні предикатів на *null*-значення: предикати зберігають *null*-значення, тобто якщо хоча б один аргумент дорівнює *null*, то результатом є невизначене булеве значення *unknown*.

Аналогічно функції при розширенні на *null*-значення зберігають його: якщо хоча б один аргумент збігається з *null*, то результат є *null*. Отже, ситуація подібна природним розширенням часткових функцій [40], але у SQL розширені функції розглядаються як часткові (наприклад, ділення).

Табл. 2.1 – Структура операцій з’єднання

Внутрішні з’єднання	**Предикат операції з’єднання за предикатом**	**Зовнішні з’єднання**			
		ліве	**праве**	**повне**	**об’єднанням**
декартове *Cj*	$p_{Cj}(s_1, s_2) = true$	–	–	–	–
природне (еквіз’єднання) $\otimes$	$s_1 \approx s_2$	+	+	+	+
з’єднання за атрибутами $\underset{A_1,\ldots,A_n}{\otimes}$	$\&_{i=1}^{n} s_1(A_i) = s_2(A_i)$	+	+	+	–
з’єднання за предикатом $\underset{p}{\otimes}$	p	+	+	+	–

2.3. Основні композиції

З точки зору композиційного підходу семантиками програм мов маніпулювання даними є функції, визначені на таблицях, значеннями яких (функцій) виступають знову таблиці, зокрема впорядковані. Такі функції будуються з атомарних (вихідних) функцій за допомогою спеціальних засобів, що називаються композиціями, котрі є спеціальними операціями над функціями.

Специфіку реляційних мов складає те, що ці мови орієнтовані на роботу з множинами (мультимножинами) у вигляді таблиць на відміну від універсальних мов типу Pascal або C, що орієнтовані на обробку даних по елементах. Тому композиції, виділені і досліджені в цих мовах, такі як циклювання, послідовне застосування, розгалуження та інші, неадекватні для опису семантики реляційних мов[9].

[9] Відмітимо роботу [4], присвячену заданню композиційної семантики мов реляційних баз даних; в ній використовувалися загальнозначні композиції, що призвело до громіздких конструкцій.

Останні вимагають спеціальних композицій, суть яких полягає в перенесенні рядкових функцій (функцій, які працюють з елементами) на таблиці (множини та мультимножини).

Розглянемо основні композиції, що використовуються в SQL-подібних мовах. Ці композиції будуть задаватися на багатомісних функціях, аргументи і значення яких належать одній з множин – D,S,T. Іншими словами, позначаючи множини сім'ї $\{D,S,T\}$ як $\Delta,\Delta_1,...$, будемо розглядати часткові функції вигляду $f^{(n)}:\Delta_1\times...\times\Delta_n \tilde{\to} \Delta$, $n\geq 1$.

Певна річ, семантиками операторів запитів є функції вигляду $f^{(n)}:T\times...\times T \tilde{\to} T$, $n\geq 1$, тобто багатомісні операції над таблицями; але в процесі побудови таких функцій будуть використовуватись функції, що визначені на універсальному домені, рядках, таблицях.

Предикатами назвемо функції вигляду $p^{(n)}:\Delta_1\times...\times\Delta_n \tilde{\to} \{true, false, u\}$, $n\geq 1$. Тут і далі для спрощення позначень третє логічне значення позначається як u. Використання тризначної логіки викликане тим, що в SQL задіяна саме така логіка. Елементи множин $\Delta,\Delta_1,...$ позначимо літерою δ з індексами. При записі рівностей значень часткових функцій, як і раніше, будемо використовувати узагальнену рівність $\tilde{=}$.

Композиція фільтрації Fl предикату вигляду $p^{(n+1)}:\Delta_1\times...\times\Delta_n\times S \tilde{\to} \{true, false, u\}$ ставить у відповідність функцію $f^{(n+1)}:\Delta_1\times...\times\Delta_n\times T \tilde{\to} T$, $n\geq 0$, довільне значення якої задається узагальненою рівністю

$$f^{(n+1)}(\delta_1,...,\delta_n,t)\tilde{=}(\{s|p^{(n+1)}(\delta_1,...,\delta_n,s)\tilde{=}true \,\&\, s\in\bar{t}\},R) \qquad (2.1)$$

де R – схема таблиці t, для всіх $\delta_i\in\Delta_i$, $i=1,...,n$; $t\in T$. У випадку, коли хоча б одне значення вигляду $p^{(n+1)}(\delta_1,...,\delta_n,s)$, де $s\in\bar{t}$, невизначене, то і значення в лівій частині рівності (2.1) покладається невизначеним.

Функція зберігає схему таблиці t, тобто результуюча таблиця має ту ж саму схему.

Композиція взяття повного образу Im функції вигляду $f_R^{n+1}:\Delta_1\times...\times\Delta_n\times S \tilde{\to} S(R)$, параметрізованої по схемі R результуючої множини рядків, ставить у відповідність функцію вигляду $g^{(n+1)}:\Delta_1\times...\times\Delta_n\times T \tilde{\to} T$, $n\geq 0$, довільне значення якої задається узагальненою рівністю

$$g^{(n+1)}(\delta_1,...,\delta_n,t)\tilde{=}(\{f^{(n+1)}(\delta_1,...,\delta_n,s)\mid s\in\bar{t}\},R) \qquad (2.2)$$

для всіх $\delta_i\in\Delta_i$, $i=1,...,n$; $t\in T$.

Схема результуючої таблиці R співпадає з параметром R функції f. З означення функції випливає, що всі рядки результату мають однакову схему, тобто множина цих рядків буде станом таблиці.

При цьому значення $g^{(n+1)}(\delta_1,\ldots,\delta_n,t)$ покладається визначеним, тільки якщо усі значення вигляду $f^{(n+1)}(\delta_1,\ldots,\delta_n,s)$, де $s\in\bar{t}$, визначені. Як і у випадку композиції фільтрації це уточнення диктується алгоритмічною природою інтерпретації операторів SQL. В реалізації, як правило, видається повідомлення про помилку. Що стосується схеми результуючої таблиці, то вона дорівнює параметру функції R.

Приклади визначення семантики низки представницьких запитів за допомогою введених композицій, а також підстановки можна знайти в [41, підрозд. 3.3].

2.4. Операції на мультимножинах та табличні функції

2.4.1. Мультимножини

Визначення таблиці як множини рядків однієї схеми припускає, що кожний елемент таблиці може зустрічатися точно один раз, що, взагалі кажучи, не має

місце у сучасних СУБД. Так, навіть якщо у початковій таблиці[10] кожний елемент зустрічається один раз, то після виконання проекції або об'єднання можуть з'явитися дублікати, причому кількість дублікатів суттєво впливає на остаточний результат. Тому подальше уточнення таблиць буде проведене на основі поняття мультимножини, тобто сукупності з дублікатами [6, 21].

Для уточнення цього поняття позначимо $N \stackrel{def}{=} \{0,1,2,...\}$ – множина натуральних чисел з нулем, покладемо $N^+ \stackrel{def}{=} \{1,2,...\}$. Зафіксуємо деяку множину U.

Під *мультимножиною* α з *основою* U будемо розуміти відображення вигляду $\alpha: U \to N^+$.

Змістовно пара $(d,n) \in \alpha$ означає, що елемент d має $n \geq 1$ дублікатів у мультимножині α. Зауважимо, що мультимножини близькі до комбінаторного поняття комбінацій з повтореннями [53, ч. II, § 1, п. 5, с. 172]; цікаво зазначити, що поняття мультимножини з'являється і у теорії λ-числення [87, розд. 12, с. 310].

Мультимножини будемо позначати грецькими літерами $\alpha, \beta, \ldots$ можливо з індексами. Мультимножини, областю значень яких є порожня множина або сінглтон $\{1\}$, назвемо 1-*мультимножинами*. Очевидно, що для непорожніх 1-мультимножин число дублікатів елементів основи рівне 1, тобто ці мультимножини є аналогами звичайних множин.

Перейдемо тепер до задання таблиць на основі мультимножин. Під *таблицею* в цьому підрозділі будемо розуміти пару $(\bar{t}, R)$, де $\bar{t}$ є (скінченна) мультимножина, основою якої виступає множина рядків схеми R. Схема R називається схемою таблиці. Мультимножину $\bar{t}$ будемо називати станом таблиці. Як і раніше, множину всіх таблиць схеми R позначимо $T(R)$.

---

[10] Можна навести загальне твердження: якщо таблиця має ключ (KEY), зокрема, створена оператором CREATE TABLE, то кількість дублікатів кожного рядка такої таблиці дорівнює 1.

Для введення операцій над таблицями у вказаному розумінні попередньо розглянемо аналоги основних теоретико-множинних операцій над мультимножинами. Зафіксуємо D – універсум елементів основ мультимножин, тоді булеан $P(D)$ є універсумом основ мультимножин. Діючи аналогічно, наприклад, [20, 23], під *характеристичною функцією мультимножини* α розуміємо функцію вигляду $\chi_\alpha : D \to N$, значення якої задається наступною кусковою схемою для всіх $d \in D$:

$$\chi_\alpha(d) \overset{def}{=} \begin{cases} \alpha(d), \text{ якщо } d \in \operatorname{dom}\alpha, \\ 0, \text{ інакше.} \end{cases}$$

Очевидно, що між мультимножинами і їх характеристичними функціями існує бієкція, тому операції над мультимножинами зручно вводити у термінах їхніх характеристичних функцій. Зокрема, характеристичні функції 1-мультимножин і тільки вони мають областю значень підмножину множини $\{0,1\}$.

Перейдемо безпосередньо до операцій над мультимножинами. Нижче теоретико-числові функції sg і $\dot{-}$ (сигнум і зрізана різниця) розуміються в звичайному смислі (див., наприклад, [23, розд. I, § 2, с. 37]).

Операція $\bigcup_1$ мультимножинам α, β зіставляє мультимножину, значення характеристичної функції якої на аргументі d задається виразом

$$sg(\chi_\alpha(d) + \chi_\beta(d)). \tag{2.3}$$

Зазначимо, що цю ж функцію можна задати і виразами $sg\,(\max(\chi_\alpha(d), \chi_\beta(d)))$, $\max(sg\,\chi_\alpha(d), sg\,\chi_\beta(d))$.

Операція $\bigcap_1$ мультимножинам α, β зіставляє мультимножину, значення характеристичної функції якої на аргументі d задається виразом

$$sg(\min(\chi_\alpha(d), \chi_\beta(d))). \tag{2.4}$$

Операція $\setminus_1$ мультимножинам α, β зіставляє мультимножину, значення характеристичної функції якої на аргументі d задається виразом

$$sg\,(\chi_\alpha(d)) \dot{-} sg(\,\chi_\beta(d)). \tag{2.5}$$

Очевидно, що операції $\bigcup_1, \bigcap_1, \setminus_1$ будують 1-мультимножину, основа якої отримується відповідно теоретико-множинним об'єднанням, перетином і різницею основ вихідних мультимножин.

Операція $\bigcup_{All}$ мультимножинам α, β зіставляє мультимножину, значення характеристичної функції якої на аргументі d задається виразом

$$\chi_\alpha(d) + \chi_\beta(d). \tag{2.6}$$

Операція $\bigcap_{All}$ мультимножинам α, β зіставляє мультимножину, значення характеристичної функції якої на аргументі d задається виразом

$$\min(\chi_\alpha(d), \chi_\beta(d)). \tag{2.7}$$

Операція $\setminus_{All}$ мультимножинам α, β зіставляє мультимножину, значення характеристичної функції якої на аргументі d задається виразом

$$\chi_\alpha(d) \dot{-} \chi_\beta(d). \tag{2.8}$$

Неважко перевірити, що основами мультимножин $\alpha \bigcup_{All} \beta$, $\alpha \bigcap_{All} \beta$ є, відповідно, множини $A \cup B$, $A \cap B$, де A і B – основи мультимножин α, β. Для визначення основи мультимножини $\alpha \setminus_{All} \beta$ введемо наступну множину $C^{>}(\alpha,\beta) \stackrel{def}{=} \{d \mid d \in A \cap B \,\&\, \alpha(d) > \beta(d)\}$. Тоді основа мультимножини $\alpha \setminus_{All} \beta$ задається виразом $(A \setminus B) \cup C^{>}(\alpha,\beta)$, що перевіряється безпосередньо. Кількість дублікатів для трьох останніх операцій задається наступними кусковими схемами:

$$(\alpha \bigcup_{All} \beta)(d) = \begin{cases} \alpha(d), & \text{якщо } d \in A \setminus B, \\ \beta(d), & \text{якщо } d \in B \setminus A, \\ \alpha(d) + \beta(d), & \text{якщо } d \in A \cap B, \end{cases}$$

де $d \in A \cup B$;

$$(\alpha \bigcap_{All} \beta)(d) = \min(\alpha(d), \beta(d)),$$

де $d \in A \cap B$;

$$(\alpha \setminus_{All} \beta)(d) = \begin{cases} \alpha(d), & \text{якщо } d \in A \setminus B, \\ \alpha(d) - \beta(d), & \text{якщо } d \in C^{>}(\alpha,\beta), \end{cases}$$

де $d \in (A \setminus B) \cup C^{>}(\alpha,\beta)$.

Зауважимо, що основу мультимножини $\alpha \setminus_{All} \beta$ можна задати виразом $\{d \mid d \in A \& (d \in B \Rightarrow \alpha(d) > \beta(d))\}$. Зазначимо також зв'язок між рівностями (2.3) і (2.6), (2.4) і (2.7), (2.5) і (2.8): рівності (2.6) – (2.8) отримуються з рівностей (2.3) – (2.5) вилученням операції sg, яка, по суті, потрібна лише для побудови 1-мультимножин.

Операція *Dist* мультимножині α зіставляє мультимножину, значення характеристичної функції якої на аргументі d задається виразом $sg\,\chi_\alpha(d)$. Очевидно, що мультимножина $Dist(\alpha)$ є 1-мультимножиною, основа якої збігається з основою вихідної мультимножини. Неважко перевірити виконання для всіх мультимножин α наступної рівності $Dist(\alpha) = \alpha \bigcup_1 \alpha$; отже, операція $Dist$ є похідною відносно операції $\bigcup_1$.

Операція декартова з'єднання мультимножин $\otimes_m$ мультимножинам α, β з основами A і B відповідно зіставляє мультимножину, основою якої є множина $A \times B$, а характеристична функція задається рівністю $\chi(d_1, d_2) \stackrel{def}{=} \chi_\alpha(d_1) \cdot \chi_\beta(d_2)$, $d_1, d_2 \in D$[11].

Введемо аналог повного образу для мультимножин. *Повним образом мультимножини* α відносно функції $f: D \stackrel{\sim}{\to} D$ назвемо мультимножину $f[\alpha]$, значення характеристичної функції якої на аргументі d задається рівністю $\sum_{d' \in f^{-1}(d)} \chi_\alpha(d')$ у припущенні, що сума порожньої множини доданків дорівнює нулю. Очевидно, що основою мультимножини $f[\alpha]$ є повний образ $f[U]$, де U – основа мультимножини α. Легко перевірити, що якщо функція f ін'єктивна, то кількість дублікатів елементів основи при побудові повного образу не змінюється.

Надалі домовимося мультимножину $\{(a_1, n_1), ..., (a_k, n_k)\}$ записувати у вигляді $\{a_1^{n_1}, ..., a_k^{n_k}\}$. Розглянемо низку прикладів. Нехай задані дві мультимножини

[11] Природно, універсум покладається замкненим відносно операції побудови впорядкованих пар.

$\alpha \stackrel{def}{=} \{ab^2, ac^3, cb^1\}$ і $\beta \stackrel{def}{=} \{ab^3, ac^1, da^5\}$, основи яких складаються з слів у алфавіті літер a, b, c, d. Тоді

$$\alpha \cup_1 \beta = \{ab^1, ac^1, cb^1, da^1\},\ \alpha \cap_1 \beta = \{ab^1, ac^1\},\ \alpha \setminus_1 \beta = \{cb^1\},$$

$$\alpha \cup_{All} \beta = \{ab^5, ac^4, cb^1, da^5\},\ \alpha \cap_{All} \beta = \{ab^2, ac^1\},\ \alpha \setminus_{All} \beta = \{ac^2, cb^1\},$$

$$\alpha \otimes_m \beta = \{(ab, ab)^6, (ab, ac)^2, (ab, da)^{10}, (ac, ab)^9,$$

$$(ac, ac)^3, (ac, da)^{15}, (cb, ab)^3, (cb, ac)^1, (cb, da)^5\}.$$

Нехай функція E повертає першу літеру слова-аргумента. Тоді $E[\alpha] = \{a^5, c^1\}$, $E[\beta] = \{a^4, d^5\}$.

2.4.2. Функції над таблицями

Перейдемо до функцій над таблицями, станами яких є мультимножини, основами останніх є множини односхемних рядків. Зауважимо, що маніпуляції над мультимножинами назвемо операціями, а над таблицями – функціями.

Функції об'єднання, перетину та різниці таблиць вводяться як обмеження однойменних операцій над мультимножинами $\cup_1$, $\cap_1$, $\setminus_1$, $\cup_{All}$, $\cap_{All}$, $\setminus_{All}$ на множину пар односхемних таблиць. Стан нової таблиці розраховується відповідною однойменною операцією на мультимножинах, а в якості схеми береться схема вихідних таблиць.

Позначати такі операції на таблицях будемо аналогічно операціям на мультимножинах з індексом T зверху. Наприклад, $\cup_1^T : T \times T \nrightarrow T$, $\mathrm{dom}\cup_1^T \stackrel{def}{=} \{(t_1, t_2) \mid \exists R(t_1 \in T(R) \,\&\, t_2 \in T(R))\}$, стан таблиці розраховується формулою $\overline{t_1 \cup_1^T t_2} \stackrel{def}{=} \bar{t}_1 \cup_1 \bar{t}_2$, а в якості схеми береться схема вихідних таблиць.

Функція декартова з'єднання таблиць Cj^T вводиться за допомогою операції декартова з'єднання мультимножин та взяття повного образу відносно операції об'єднання сумісних рядків $\bar{\cup}$. Це часткова функція декартова з'єднання $Cj^T : T \times T \nrightarrow T$, стан таблиці задається рівністю $\overline{t_1 Cj^T t_2} \stackrel{def}{=} \bar{\cup}[\bar{t}_1 \otimes_m \bar{t}_2]$, а схема

отримується об'єднанням схем вихідних таблиць. Областю означеності є множина пар таблиць, схеми яких не перетинаються.

Функція вилучення дублікатів $Dist^T : T \to T$ отримується обмеженням операції $Dist$ на множину станів таблиць, схема таблиці залишається незмінною.

Таким чином, визначення таблиць як мультимножин рядків дозволяє задати низку операцій, які використовуються у мові SQL, але невиразні за допомогою зображення таблиць як множин рядків. Відплатою за це є об'єктивне ускладнення означень таблиць, функцій і композицій, оскільки тепер необхідно визначати не тільки елементи, які попадають у результат, але і кількість їх дублікатів.

2.4.3. Композиції

Розглянемо задання композицій на випадок таблиць, стани яких розуміються як мультимножини. Елементи станів таблиці – пари (s,i), де i – кількість дублікатів рядка s, будемо записувати у вигляді s^i.

Композиція фільтрації Fl предикату вигляду $p^{(n+1)} : \Delta_1 \times ... \times \Delta_n \times S \tilde{\to} \{true, false, u\}$ ставить у відповідність функцію вигляду $f^{(n+1)} : \Delta_1 \times ... \times \Delta_n \times T \tilde{\to} T$, $n \geq 0$. Стан нової таблиці задається рівністю $f^{(n+1)}(\delta_1,...,\delta_n,t) \tilde{=} \{s^i \mid p(\delta_1,...,\delta_n,s) \tilde{=} true \& s^i \in \bar{t}\}$. Як і раніше значення у лівій частині останньої рівності покладається невизначеним, якщо хоча б одне значення вигляду $p(\delta_1,...,\delta_n,s)$, де $s^i \in \bar{t}$, невизначене. Схема таблиці співпадає зі схемою вихідної таблиці.

Таким чином, всі дублікати одного рядка або одночасно попадають в результат, або не попадають.

Композиція взяття повного образу Im функції вигляду $f_R^{n+1} : \Delta_1 \times ... \times \Delta_n \times S \tilde{\to} S(R)$, параметризованої по схемі результуючих рядків R, ставить у відповідність функцію вигляду $g_R^{n+1} : \Delta_1 \times ... \times \Delta_n \times T \tilde{\to} T(R)$, $n \geq 0$. Для її завдання позначимо як $F_{R,\delta_1,...,\delta_n} : T \tilde{\to} T(R)$ унарну функцію, що отримується з вихідної функції f фіксацією перших n аргументів елементами $\delta_1, \delta_2, ..., \delta_n$

відповідно. Тоді довільне значення функції g обчислюється як таблиця, станом якої є повний образ мультимножини – $\overline{g_R^{n+1}(\delta_1,...,\delta_n,t)} \tilde{=} F_{R,\delta_1,...,\delta_n}[\bar{t}]$ для всіх $\delta_i \in \Delta_i$, $i=1,...,n$; $t \in T$;. схема ж результату дорівнює параметру функції R. При цьому значення $g_R^{n+1}(\delta_1,...,\delta_n,t)$ покладається невизначеним, якщо хоча б одне значення $f_R^{n+1}(\delta_1,...,\delta_n,s)$, де s належить основі стану таблиці t, невизначене. Як і у випадку композиції повного образу для таблиць, стани яких уточнюються як множини, остання вимога диктується алгоритмічною природою інтерпретації операторів SQL.

Зазначимо, що має місце наступна рівність $\overline{g_R^{n+1}(\delta_1,...,\delta_n,t)} \simeq \bigcup_{All}_{s^i \in \bar{t}} \{ f_R^{n+1}(\delta_1,...,\delta_n,s)^i \}$. Неважко перевірити, що операція $\bigcup_{All}$ комутативна і асоціативна, тому припускає природне коректне розширення на сім'ї мультимножин. Крім того, для забезпечення коректності визначення останньої композиції на вихідну функцію потрібно накладати обмеження стабільності відношення односхемності рядків відносно цієї функції (аналогічно випадку станів таблиць як множин рядків).

2.5. Упорядкування таблиць

Цей підрозділ присвячений наступному уточненню таблиць, яке дозволяє задати семантику фрази ORDER BY. Суть уточнення полягає у розгляді станів таблиць як множин (мультимножин) з бінарним відношенням, яке індукується зазначеною фразою. Отже, стани таблиць уточнюються як моделі [22, розд. I, § 2, п. 2.2, с. 47], носіями яких є стани таблиць в попередньому розумінні, а сигнатури містять єдиний бінарний предикатний символ. Розглянемо його інтерпретацію. Для визначеності припустимо, що таблиці зображаються парами виду $(\bar{t}, R)$, де $\bar{t}$ – як і раніше, множина односхемних рядків, яка називається станом таблиці, а R – схема таблиці.

Нехай $\vec{A} \stackrel{def}{=} < A_1,...,A_n >$, $n = 1,2,...$ – кортеж попарно різних атрибутів. Покладемо $W \stackrel{def}{=} \{A_1,...,A_n\}$; сім'ю всіх рядків, множина атрибутів яких точно збігається з W, позначимо $S_W^=$, а сім'ю всіх рядків, множина атрибутів яких є надмножиною множини W, – $S_W^\geq$. Таким чином, $S_W^= \subseteq S_W^\geq$ та можна говорити про значення атрибуту $A \in W$ в рядках сім'ї $S_W^\geq$.

Зафіксуємо на універсальному домені D порядок $\leq$. Зазначимо, що в реалізації цей порядок є, як правило, лінійним, і, більш того, цілком впорядковує універсальний домен. На множині $S_W^=$ розглянемо наступне бінарне відношення, яке параметризоване за кортежем $\vec{A}$

$$s_1 \leq_{\vec{A}} s_2 \stackrel{def}{\Leftrightarrow} s_1(A_1) < s_2(A_1) \vee s_1(A_1) = s_2(A_1) \,\&\, s_1(A_2) < s_2(A_2) \vee ... \vee$$

$$\vee \mathop{\&}_{i=1}^{n-1}(s_1(A_i) = s_2(A_i)) \,\&\, s_1(A_n) < s_2(A_n) \vee \mathop{\&}_{i=1}^{n}(s_1(A_i) = s_2(A_i)).$$

Зазначимо, що це відношення по суті будується лексикографічним добутком вихідного порядку $\leq$ з врахуванням порядку співмножників згідно з кортежем $\vec{A}$.

Зазначимо також, що ключове слово ASC (від англ. ascending – за зростанням) фрази ORDER BY відповідає вихідному порядку, а ключове слово DESC (від англ. descending – за спаданням) – інверсному. Таким чином, співмножники лексикографічного атрибуту дорівнюють або вихідному порядку, або інверсному до нього.

Очевидно, що відношення $\leq_{\vec{A}}$ є порядком на сім'ї рядків $S_W^=$. Розширимо це відношення на множину рядків $S_W^\geq$. Через $\uparrow W: S_W^\geq \to S_W^=$, $\uparrow W(s) = s|W$ позначимо функцію обмеження рядків на відповідну множину атрибутів. Тепер на множині рядків $S_W^\geq$ розглянемо наступне бінарне відношення, параметризоване за кортежем $\vec{A}$: $s_1 \preceq_{\vec{A}} s_2 \stackrel{def}{\Leftrightarrow} \uparrow W(s_1) \leq_{\vec{A}} \uparrow W(s_2)$.

Очевидно, що рефлексивність і транзитивність порядку $\leq_{\vec{A}}$ успадковується відношенням $\preceq_{\vec{A}}$; що ж до антисиметричності, то з огляду на неін'єктивність

функції обмеження, відношення $\preceq_{\bar{A}}$ цієї властивості не має. Отже, воно є лише передпорядком на сім'ї рядків $S_W^{\geq}$. Разом з цим вказаний передпорядок індукує порядок на фактор-множині сім'ї рядків $S_W^{\geq}$ за ядерною еквівалентністю функції обмеження $\uparrow W$: $[s_1] \trianglelefteq_{\bar{A}} [s_2] \overset{def}{\Leftrightarrow} s_1 \preceq_{\bar{A}} s_2$, де $[s_i]$ – клас ядерної еквівалентності з представником s_i. Дійсно, рефлексивність і транзитивність так введеного відношення очевидні, а його антисиметричність випливає з еквівалентності $s_1 \preceq_{\bar{A}} s_2 \;\&\; s_2 \preceq_{\bar{A}} s_1 \Leftrightarrow \uparrow W(s_1) = \uparrow W(s_2)$.

У виникненні порядку на фактор-множині немає нічого несподіваного; конструкція побудови порядку на фактор-множині за передпорядком на вихідній множині добре відома (див., наприклад, [45, § 1, теорема 3]); для відношення $\trianglelefteq_{\bar{A}}$ потрібно лише врахувати вказану раніше еквівалентність, яка є характеристикою ядра обмеження.

Повернемося до таблиць. Під *у-таблицею* ("упорядкованою" таблицею) будемо розуміти трійку вигляду $(\bar{t}, R, \preceq_{A_1,\ldots,A_n})$, де $\bar{t}$ – (скінченна) множина односхемних рядків, R – схема цих рядків, тобто схема таблиці, а атрибути $A_1,\ldots,A_n$ належать R.

Розглянемо більш докладно питання про порядок рядків в у-таблицях. Як і раніше для компактності запису покладемо $\vec{A} \overset{def}{=} < A_1,\ldots,A_n >$, $W \overset{def}{=} \{A_1,\ldots,A_n\}$. Нехай s_1, s_2 – різні рядки у-таблиці $(\bar{t}, R, \preceq_{A_1,\ldots,A_n})$, причому порівнювані відносно $\preceq_{\bar{A}}$: нехай для визначеності $s_1 \preceq_{\bar{A}} s_2$. Маємо рівно дві можливості:

$\uparrow W(s_1) = \uparrow W(s_2)$, тобто значення атрибутів $A_1,\ldots,A_n$ збігаються у вказаних рядках; тоді впорядкування рядків цілком залежить від реалізації;

$\uparrow W(s_1) <_{\bar{A}} \uparrow W(s_2)$, тобто рядки розрізняються значенням хоча б одного атрибута з $A_1,\ldots,A_n$. Тоді рядок s_1 передує рядку s_2.

Таким чином, у частковому випадку, коли схема таблиці збігається з множиною атрибутів W, відношення $\preceq_{\bar{A}}$ впорядковує рядки цієї таблиці.

Можна сформувати і більш сильне твердження: якщо множина атрибутів W містить (первинний) ключ таблиці, то відношення $\preceq_{\bar{A}}$ є порядком на цій таблиці.

Так визначені у-таблиці призначаються лише для впорядкування результатів запитів; ці таблиці є елементами областей значень, а не областей означення функцій, що є семантиками запитів.

Нарешті, для задання композиційної семантики запитів з фразою ORDER BY треба ввести функцію, яка перетворює таблиці в у-таблиці. Такою є параметризована за кортежем попарно різних атрибутів $\vec{A}$ функція $Orderby_{\vec{A}}$, яка зіставляє таблицям $(\bar{t}, R)$, схеми яких містять атрибути $A_1, \ldots, A_n$, у-таблиці вигляду $(\bar{t}, R, \preceq_{\bar{A}})$.

Приклади еталонування відповідних запитів наведені в [41, підрозд. 3.5].

2.6. Семантика агрегатних функцій

2.6.1. Задання агрегатних функцій

Почнемо з розгляду агрегатних функцій на множинах; далі перейдемо і до мультимножин. Характеристична властивість таких функцій полягає в тому, що їхніми аргументами є скінченні множини або пари, одна з компонент яких є скінченою множиною, а друга має такий самий тип, що і елементи множини, а значеннями – елементи універсального домену або булевські значення.

У мові SQL явно виділені наступні агрегатні функції:

SUM – сума елементів скінченної числової множини, які не є *null*-значеннями;

AVG – середнє арифметичне значення елементів скінченної числової множини, які не є *null*-значеннями;

MIN – найменший з елементів скінченної лінійно впорядкованої множини, відмінних від *null*;

MAX – найбільший з елементів скінченної лінійно впорядкованої множини, відмінних від *null*;

COUNT – кількість елементів скінченної множини, відмінних від *null*;

COUNT(*) – кількість рядків в таблиці.

Крім того, з огляду на зазначену вище характеристичну властивість функцій-семантик агрегатних функцій, наступні мовні конструкції є агрегатними функціями:

EXISTS – перевірка множини (таблиці) на непорожність;

IN – перевірка належності елемента (рядка) множині (таблиці);

ALL – квантор загальності за скінченною множиною;

ANY – квантор існування за скінченною множиною.

Зауважимо, що у логічних конструкціях ALL, ANY використовується тризначна логіка SQL.

2.6.2.Формальні означення агрегатних функцій SQL-подібних мов

Нехай Num – числова підмножина універсального домену, замкнена відносно звичайного додавання. Покладемо, що $null \in Num$.

Сума елементів множини $Sum{:}2^{Num} \to Num$. Значення задаються кусковою схемою для всіх $\Lambda \in 2^{Num}$:

$$Sum(\Lambda) \overset{def}{=} \begin{cases} null, \text{ якщо } \Lambda \setminus \{null\} = \varnothing, \\ \sum\limits_{\lambda \in \Lambda \setminus \{null\}} \lambda \text{ , якщо } \Lambda \setminus \{nul\} \neq \varnothing. \end{cases}$$

Таким чином, $Sum(\varnothing) = Sum(\{null\}) = null$ і $Sum(\Lambda) = Sum(\Lambda \setminus \{null\})$.

Нехай $\leq$ – лінійний порядок на універсальному домені D. *Найбільший елемент множини* $Max{:}2^{D} \to D$. Значення задаються наступною кусковою схемою для всіх $\Lambda \in 2^{D}$ – скінченних ланцюгів:

$$Max(\Lambda) \overset{def}{=} \begin{cases} null, \text{ якщо } \Lambda \setminus \{null\} = \varnothing, \\ \text{найбільший елемент } \Lambda \setminus \{null\}, \text{ якщо } \Lambda \setminus \{null\} \neq \varnothing. \end{cases}$$

Таким чином, $Max(\varnothing) = Max(\{null\}) = null$, $Max(\Lambda) = Max(\Lambda \setminus \{null\})$.

Найменший елемент множини $Min{:}2^{D} \to D$ задається повністю аналогічно функції Max, лише вихідний порядок змінюється на обернений.

Зазначимо, що функції Max, Min задаються в термінах найбільшого або найменшого елемента ланцюга елементів, відмінних від $null$; тому порівнянність

особливого елемента *null* з рештою елементів універсального домену в даному випадку неістотна. Ця обставина істотна при впорядкуванні рядків (результуючих) таблиць (див. підрозділ 2.5); зазначимо, що в конкретних реалізаціях SQL *null* може бути як найменшим, так і найбільшим елементом.

Кількість елементів множини, відмінних від null, $Count{:}2^D \to N$. Значення задаються рівністю $Count(\Lambda) \stackrel{def}{=} |\Lambda \setminus \{null\}|$, де $|X|$ – потужність (кількість елементів) множини X. Таким чином, $Count(\varnothing) = Count(\{null\}) = 0$, $Count(\Lambda) = Count(\Lambda \setminus \{null\})$.

Припустимо, що числова підмножина *Num* універсального домену замкнена відносно звичайного ділення, причому операція ділення $/ : Num \times Num \xrightarrow{\sim} Num$ зберігає *null*.

Середнє арифметичне значення $Avg{:}2^{Num} \to Num$ вводиться як похідна функція відносно ділення і агрегатних функцій *Sum*, *Count*: $Avg(\Lambda) \stackrel{def}{=} Sum(\Lambda) / Count(\Lambda)$, де $\Lambda \in 2^D$.

Таким чином, з означення випливають рівності

$$Avg(\varnothing) = Sum(\varnothing) / Count(\varnothing) = null / 0 = null;$$

$$Avg(\{null\}) = Sum(\{null\}) / Count(\{null\}) = null / 0 = null;$$

$Avg(\Lambda \cup \{null\}) = Sum(\Lambda \cup \{null\}) / Count(\Lambda \cup \{null\}) = \frac{1}{|\Lambda|} \sum_{\lambda \in \Lambda} \lambda$ – середнє арифметичне значення чисел множини Λ; причому $null \notin \Lambda$, $\Lambda \neq \varnothing$. Зазначимо, що з означення функції *Avg* і властивостей функцій *Sum*, *Count* випливає рівність $Avg(\Lambda) = Avg(\Lambda \setminus \{null\})$; зауважимо, що для функцій *Sum*, *Count* аналогічна рівність теж виконується.

Кількість рядків таблиці $Count(*){:}T \to N$. Значення задаються рівністю $Count(*)(t) \stackrel{def}{=} |\bar{t}|$, де $\bar{t}$ – стан таблиці t.

Перевірка таблиці (як множини рядків) *на непорожність* $Exists{:}T \to \{true, false\}$ задається кусковою схемою для всіх таблиць t:

$$Exists(t) \stackrel{def}{=} \begin{cases} true, \text{якщо } \bar{t} \neq \varnothing, \\ false, \text{якщо } \bar{t} = \varnothing. \end{cases}$$

У подальших конструкціях $Bool \stackrel{def}{=} \{true, false, u\}$ – множина значень тризначної логіки, де u є скороченням від *unknown*. Нехай $\theta: D_1 \times D_2 \to Bool$ – деякий предикат на універсальному домені, який є параметром в наступних двох конструкціях, $D_1, D_2 \subseteq D$. Вважається, що $null \in D_1$, $null \in D_2$, а предикат θ зберігає значення *null*.

Квантор загальності відносно предиката θ $All_\theta: D_1 \times 2^{D_2} \to Bool$. Значення задаються рівністю (тут і надалі в тризначній логіці) $All_\theta(d, D') \stackrel{def}{=} \forall d'(d' \in D' \Rightarrow \theta(d, d'))$; при цьому використовується звичайна угода відносно порожньої множини – $All_\theta(d, \varnothing) \stackrel{def}{=} true$.

Квантор існування відносно предиката θ $Any_\theta: D_1 \times 2^{D_2} \to Bool$. Значення задаються рівністю $Any_\theta(d, D') \stackrel{def}{=} \exists d'(d' \in D' \wedge \theta(d, d'))$; при цьому використовується звичайна угода відносно порожньої множини – $Any_\theta(d, \varnothing) \stackrel{def}{=} false$.

Таблиця 2.2 – Визначення функцій *All, Any*

Аргумент	**Значення функції *All***	**Значення функції *Any***
$\{true, false, u\}$	*false*	*true*
$\{true, false\}$	*false*	*true*
$\{false, u\}$	*false*	*u*
$\{true, u\}$	*u*	*true*
$\{true\}$	*true*	*true*
$\{false\}$	*false*	*false*
$\{u\}$	*u*	*u*
$\varnothing$	*true*	*false*

Останні дві функції припускають природні описи, які отримуються

розповсюдженням булевський операцій $\vee$ і $\wedge$ на скінченні множини булевських значень. Як і звичайно в логіці диз'юнкція відповідає квантору існування, а кон'юнкція – квантору загальності. Ці розповсюдження уточнюються функціями $Any{:}2^{Bool} \to Bool$ і $All{:}2^{Bool} \to Bool$, які з огляду на скінченність множини 2^{Bool} задамо таблично згідно з табл. 2.2.

Зауважимо, що операції $\vee$, $\wedge$ є комутативними та асоціативними; саме цей факт дозволяє здійснити їх коректне розповсюдження на скінченні множини.

Неважко перевірити наступні рівності: $All(B) = \underset{b\in B}{\wedge} b, Any(B) = \underset{b\in B}{\vee} b$, де $B \subseteq Bool$. Що стосується наведених вище параметричних конструкцій, то вони в термінах функцій All, Any задаються так:

$$All_\theta(d, D') = All(\{\theta(d,d')|d' \in D'\}),\ Any_\theta(d, D') = Any(\{\theta(d,d')|d' \in D'\}).$$

Зауважимо, що виконуються наступні рівності, які випливають з означень та збереження значення $null$ предикатом θ:

$$All(\{u\}) = Any(\{u\}) = u;\ All_\theta(null, D') = Any_\theta(null, D') = u,\ D' \neq \varnothing;$$

$$All_\theta(d, \{null\}) = Any_\theta(d, \{null\}) = u,\ d \in D_1;$$

$$All_\theta(d_1, \{d_2\}) = Any_\theta(d_1, \{d_2\}) = \theta(d_1, d_2),\ d_1 \in D_1,\ d_2 \in D_2.$$

Перевірка належності елемента множині $In{:}D \times 2^D \to Bool$. У SQL ця агрегатна функція вводиться як частковий випадок конструкції Any_θ, де предикат θ є рівність $=_S{:}D \times D \to Bool$, яка зберігає значення $null$ (ця рівність вводилася наприкінці підрозділу 2.2). Таким чином, In є $Any_{=_S}$. Звідси, зокрема, випливають рівності $In(d, \varnothing) = false$; $In(null, \varnothing) = false$; $In(null, D') = u$, $D' \neq \varnothing$; $In(d, \{null\}) = u$; $In(null, \{null\}) = u$.

Зробимо одне загальне зауваження про тризначну (сильну) логіку SQL. Неважко перевірити, що алгебра $\langle Bool; \wedge, \vee \rangle$ є абстрактною решіткою за термінологією [38, розд. II, § 5, с. 130], тобто її сигнатурні операції ідемпотентні, комутативні, асоціативні та задовольняють двом законам поглинання. Але між абстрактними решітками та решітками (тобто частково впорядкованими множинами, в яких для довільних двох елементів існують обидві точні грані)

існує добре відомий зв'язок [38, розд. II, § 5, с. 130-131; 45, § 4, теорема 2, с. 60]. Відповідний частковий порядок $\leq$ на множині *Bool* легко відновлюється за логічними операціями ($a \leq b \overset{def}{\Leftrightarrow} a \vee b = b \Leftrightarrow a \wedge b = a$); він зображений на рис. 2.4.

$$false \longrightarrow u \longrightarrow true$$

Рис. 2.4. Лінійний порядок решітки $\langle Bool, \leq \rangle$, який відповідає абстрактній решітці $\langle Bool; \wedge, \vee \rangle$ тризначної логіки SQL

Зауважимо, що це лінійний порядок з найменшим елементом *false* і найбільшим елементом *true*. Отже, операції тризначної логіки SQL повністю визначені порядком – $a \vee b = \coprod\{a,b\}$, $a \wedge b = \prod\{a,b\}$; аналогічно для операцій квантифікації та функцій *Any*, *All* – $Any(B) = \coprod B$, $All(B) = \prod B$ (див. табл. 2.2). □

2.6.3. Композиція агрегування

У цьому пункті, виходячи з означень, наведемо характеристичні рівності для агрегатних функцій, які дозволяють задавати ці функції індуктивно за потужністю аргументу. Таке індуктивне завдання і буде лежати в основі спеціальної композиції агрегування, призначеної для розширення бінарних функцій на скінченні множини.

Функція *Sum*. Розглянемо наступне розширення звичайного додавання, яке не зберігає значення *null*. Нехай $\oplus: Num \times Num \to Num$, а значення задаються кусковою схемою для всіх $d_1, d_2 \in Num$:

$$d_1 \oplus d_2 \overset{def}{=} \begin{cases} d_1 + d_2, d_1, d_2 \neq null, \\ d_2, d_1 = null, \\ d_1, d_2 = null, \\ null, d_1 = d_2 = null. \end{cases}$$

Безпосередньо перевіряється, що операція $\oplus$ комутативна і асоціативна; має праву і ліву одиниці *null*. Аналізуючи всі можливі випадки, встановлюється

рівності, які зв'язують значення функції Sum на множинах потужності $n+1$ і n, $n=0,1,2,...$

$$Sum(\varnothing)=null,\ Sum(\{d\}\bigcup D')=d\oplus Sum(D'), \tag{2.9}$$

де $D'\in 2^{Num}$, $d\in Num$, $d\notin D'$. З (2.9) випливає наступна рівність

$$\begin{gathered}Sum(\{d_1,...,d_n\})=d_1\oplus d_2\oplus...\oplus d_n\oplus Sum(\varnothing)=\\ =d_1\oplus d_2\oplus...\oplus d_n\oplus null,\end{gathered} \tag{2.9'}$$

де $n=1,2,...$ і всі елементи d_i попарно різні. Звичайно, оскільки операція $\oplus$ комутативна і асоціативна, порядок запису елементів множини-аргументу неістотний.

Функція Max. Розглянемо наступне розширення функції взяття найбільшого з двох відмінних від $null$ елементів, яке не зберігає $null$. Нехай $\max: D\times D\to D$, а значення задаються кусковою схемою для всіх $d_1,d_2\in D$ (як і раніше припускається, що універсальний домен D лінійно впорядкований):

$$\max(d_1,d_2)\overset{def}{=}\begin{cases}\text{найбільший з } d_1,d_2\ ,\ d_1,d_2\neq null,\\ d_1,\ d_2=null,\\ d_2,\ d_1=null,\\ null,\ d_1,d_2=null.\end{cases}$$

Безпосередньо перевіряється, що операція max є ідемпотентною, комутативною, асоціативною, причому має праву і ліву одиницю $null$. Безпосередньою перевіркою встановлюється наступна рівність, яка пов'язує значення функції Max на множинах потужності $n+1$ і n, $n=0,1,2,...$

$$Max(\varnothing)=null,\ Max(\{d\}\bigcup D')=\max(d,Max(D')), \tag{2.10}$$

де $d\in D$, $D'\in 2^D$. З (2.10) у свою чергу випливає рівність

$$\begin{gathered}Max(\{d_1,...,d_n\})=\max(d_1,\max(d_2,...,\max(d_n,Max(\varnothing))...))=\\ =\max(d_1,\max(d_2,...,\max(d_n,null)...)),\end{gathered} \tag{2.10'}$$

де $n=1,2,...$ Зазначимо, що через ідемпотентність операції max вимога щодо попарної відмінності елементів d_i зайва. Як і раніше, зважаючи на комутативність і асоціативність операції max, порядок запису елементів множини-аргументу неістотний.

Функція Min. Повністю аналогічно випадку функції Max з переходом до оберненого порядку слід розглядати наступну функцію $\min: D \times D \to D$

$$\min(d_1, d_2) \overset{def}{=} \begin{cases} \text{найменший з } d_1, d_2, & d_1, d_2 \neq null, \\ d_1, & d_2 = null, \\ d_2, & d_1 = null, \\ null, & d_1, d_2 = null; \end{cases}$$

де $d_1, d_2 \in D$. Для функції Min виконуються наступні рівності

$$Min(\varnothing) = null, \; Min(\{d\} \cup D') = \min(d, Min(D')), \tag{2.11}$$

$$Min(\{d_1, ..., d_n\}) = \min(d_1, \min(d_2, ..., \min(d_n, null)...)), \tag{2.11'}$$

де $d \in D$, $D' \in 2^D$, $n = 1,2,...$

Функція $Count$. Розглянемо наступну функцію $sm: D \times N \to N$, яка не зберігає значення $null$:

$$sm(d, n) \overset{def}{=} \begin{cases} n, & \text{якщо } d = null, \\ n+1, & \text{якщо } d \neq null; \end{cases}$$

де $d \in D$, $n \in N$. Зазначимо, що функція sm близька до унарної функції слідування теорії рекурсії ($s(x) = x + 1$). Неважко встановити наступну рівність

$$Count(\varnothing) = 0, \; Count(\{d\} \cup D') = sm(d, Count(D')), \tag{2.12}$$

де $d \in D$, $D' \in 2^D$, $d \notin D'$. З останньої формули випливає рівність

$$\begin{aligned} Count(\{d_1, ..., d_n\}) &= sm(d_1, sm(d_2, ..., sm(d_n, Count(\varnothing))...)) = \\ &= sm(d_1, sm(d_2, ..., sm(d_n, 0)...)), \end{aligned} \tag{2.12'}$$

де $n = 1,2,...$, а елементи d_i попарно різні. Очевидно, що порядок запису елементів d_i у множині-аргументі на результат не впливає.

Функція $Count(*)$. Цей випадок аналогічний попередньому; слід розглянути лише наступну модифікацію функції sm – $sm': S \times N \to N$, $sm'(s, n) \overset{def}{=} n + 1$, де $s \in S$, $n \in N$. Рівності (2.12)-(2.12') приймають вигляд

$$Count(\varnothing) = 0, \; Count(*)(\{s\} \cup \bar{t}) = sm'(s, Count(*)(\bar{t})), \tag{2.13}$$

де $s \in S$, $t \in T$, $s \notin \bar{t}$; причому рядок s та стан таблиці $\bar{t}$ односхемні. Нехай таблиця t має стан $\{s_1, ..., s_n\}$. Тоді

$$Count(*)(t) = sm'(s_1, sm'(s_2,..., sm'(s_n, Count(*)(\varnothing))...)) = \\ = sm'(s_1, sm'(s_2,..., sm'(s_n, 0)...)), \tag{2.13'}$$

де $n = 1,2,...$, а рядки s_i попарно різні.

Функції $All_\theta, Any_\theta, All, Any$. Неважко перевірити істинність наступних рівностей, де $\theta: D_1 \times D_2 \to Bool$ предикат, як і раніше.

$$All_\theta(d', \varnothing) = true,\ All_\theta(d', \{d\} \cup D') = \theta(d', d) \wedge All_\theta(d', D'), \tag{2.14}$$

$$Any_\theta(d', \varnothing) = false,\ Any_\theta(d', \{d\} \cup D') = \theta(d', d) \vee Any_\theta(d', D'), \tag{2.15}$$

де $d' \in D_1,\ d \in D_2,\ D' \in 2^{D_2}$. Зокрема, виконуються рівності

$$All_\theta(d', \{d_1,...,d_n\}) = \bigwedge_{i=1}^{n} \theta(d', d_i) \wedge true = \bigwedge_{i=1}^{n} \theta(d', d_i), \tag{2.14'}$$

$$Any_\theta(d', \{d_1,...,d_n\}) = \bigvee_{i=1}^{n} \theta(d', d_i) \vee false = \bigvee_{i=1}^{n} \theta(d', d_i). \tag{2.15'}$$

Для функцій Any і All маємо аналогічні рівності

$$All(\{b\} \cup B) = b \wedge All(B), \tag{2.16}$$

$$Any(\{b\} \cup B) = b \vee Any(B), \tag{2.17}$$

де $b \in Bool$, $B \subseteq Bool$. Зокрема, виконуються рівності

$$All(\{b_1,...,b_n\}) = b_1 \wedge ... \wedge b_n \wedge true = b_1 \wedge ... \wedge b_n, \tag{2.16'}$$

$$Any(\{b_1,...,b_n\}) = b_1 \vee ... \vee b_n \vee false = b_1 \vee ... \vee b_n, \tag{2.17'}$$

де $n = 1,2,...$, $b_i \in Bool$ (і не обов'язково попарно різні).

Функція $Exists$. Очевидно, що виконуються рівності

$$Exists(\varnothing) = false,\ Exists(\{s\} \cup \bar{t}) = true, \tag{2.18}$$

де $s \in S$, $t \in T$, причому рядок s і таблиця t односхемні.

Проаналізуємо отримані рівності (2.9) – (2.18). З них випливає, що значення кожної з розглянутих агрегатних функцій F на множині $\{d_1,...,d_n\}$ задається термом вигляду $\varphi(d_1, \varphi(d_2,..., \varphi(d_n, F(\varnothing))...)$ для відповідної бінарної функції φ (див. рис. 2.5). Зауважимо, що агрегатні функції In, All_θ, Any_θ мають неістотні

відмінності від цієї загальної схеми, враховані в наступному загальному означенні композиції агрегування; ці відмінності пов'язані з введенням параметра θ.

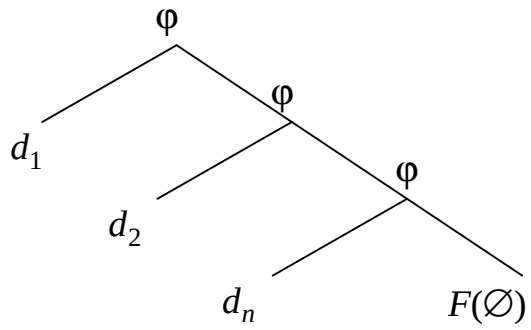

Рис. 2.5. Терм для знаходження значення агрегатної функції $F(\{d_1,...,d_n\})$

Зафіксуємо функцію ψ вибору елемента з підмножин універсального домену, тобто для непорожніх множин D' виконується $\psi(D') \in D'$.

Композиція агрегування *Agr* (у випадку множин) функції вигляду $f: D_1 \times ... \times D_k \times D' \times D'' \to D''$, $k = 0,1,2,...$ і елементу $d_0 \in D''$ ставить у відповідність функцію вигляду $F: D_1 \times ... \times D_k \times 2^{D'} \to D''$, значення якої задаються індукцією за числом елементів скінченних множин:

$$F(d_1,...,d_k,\varnothing) = d_0,$$

$$F(d_1,...,d_k,D''') = f(d_1,...,d_k,\psi(D'''),F(d_1,...,d_k,D''' \setminus \{\psi(D''')\})),$$

де $d_1 \in D_1$, ..., $d_k \in D_k$, $D''' \in 2^{D'}$, $|D'''| \geq 1$.

Іншими словами, оскільки функція вибору індукує зображення скінченних множин у вигляді послідовностей без повторень, то має місце рівність $F(...,\{d_1',...,d_n'\}) = f(...,d_1', f(...,d_2',..., f(...,d_n',d_0)...)$ для $n \geq 1$.

Таблиця 2.3 – Визначення агрегатних функцій композицією агрегування

Агрегатна функція	***n*-Арна функція**	**Значення на порожній множині**
$Sum: 2^{Num} \to Num$	$\oplus: Num \times Num \to Num$	$Sum(\varnothing) = null$
$Max: 2^{D} \to D$	$\max: D \times D \to D$	$Max(\varnothing) = null$
$Min: 2^{D} \to D$	$\min: D \times D \to D$	$Min(\varnothing) = null$

Агрегатна функція	***n*-Арна функція**	**Значення на порожній множині**
$Count: 2^D \to N$	$sm: N \times D \to N$	$Count(\varnothing) = 0$
$Count(*): T \to N$	$sm': N \times S \to N$	$Count(*)(\varnothing) = 0$
$Exists: T \to \{true, false\}$	$f_{Ex}: S \times \{true, false\} \to$ $\to \{true, false\},$ $f_{Ex}(s,b) = true$	$Exists(\varnothing) = false$
$In: D \times 2^D \to Bool$	$f_{In}: D \times D \times Bool \to Bool,$ $f_{In}(d_1,d_2,b) = (d_1 =_S d_2) \vee b$	$In(d,\varnothing) = false$
$All_\theta: D_1 \times 2^{D_2} \to Bool$	$f_\theta: D_1 \times D_2 \times Bool \to Bool$ $f_\theta(d_1,d_2,b) = \theta(d_1,d_2) \wedge b$	$All_\theta(d,\varnothing) = true$
$Any_\theta: D_1 \times 2^{D_2} \to Bool$	$f_\theta: D_1 \times D_2 \times Bool \to Bool$ $f_\theta(d_1,d_2,b) = \theta(d_1,d_2) \vee b$	$Any_\theta(d,\varnothing) = false$
$All: 2^{Bool} \to Bool$	$\wedge: Bool \times Bool \to Bool$	$All(\varnothing) = true$
$Any: 2^{Bool} \to Bool$	$\vee: Bool \times Bool \to Bool$	$Any(\varnothing) = false$

Всі формули (2.9'), (2.10') (2.17') є частковими випадками останньої формули при відповідних k, f і d_0. Конкретні вихідні n-арні функції і виділені елементи для значень на порожній множині для всіх розглянутих агрегатних функцій вказані у табл. 2.3. Зауважимо, що композиція агрегування близька до конструкції згортки функціональних програм [3, 54].

2.6.4. Завдання агрегатних функцій для мультимножин

У цілому для мультимножин агрегатні функції задаються аналогічно випадку множин.

Нижче $O(\alpha) \stackrel{def}{=} \mathrm{dom}\,\alpha$ – основа мультимножини α, а $2_m^{D'} \stackrel{def}{=} \{\alpha | O(\alpha) \in 2^{D'}\}$ – сім'я всіх мультимножин, основи яких є скінченними підмножинами множини D'; тут $D' \subseteq D$ – підмножина універсального домену. Під *потужністю*

скінченної мультимножини α будемо розуміти суму дублікатів елементів її основи $|\alpha| \stackrel{def}{=} \sum_{d \in O(a)} \alpha(d)$; за означенням покладемо $|\varnothing| = 0$.

Розглянемо означення тільки функції знаходження суми. Точні означення всіх інших агрегатних функцій для мультимножин наведені у [41, підрозд. 3.8].

Сума елементів основи мультимножини, які відрізняються від значення *null*. Це функція вигляду $Sum_m : 2_m^{Num} \to Num$, її значення задаються кусковою схемою

$$Sum_m(\alpha) \stackrel{def}{=} \begin{cases} null, \text{ якщо } O(\alpha) \setminus \{null\} = \varnothing, \\ \sum_{d \in O(\alpha) \setminus \{null\}} d \cdot \alpha(d), \text{ якщо } O(\alpha) \setminus \{null\} \neq \varnothing, \end{cases}$$

де $\alpha \in 2_m^{Num}$. Таким чином, $Sum_m(\{d_1^{k_1}, \ldots, d_n^{k_n}\}) = \sum_{i=1}^{n} d_i k_i$, в припущенні, що всі елементи основи d_i відрізняються від елемента *null*.

Для агрегатних функцій виконуються аналоги співвідношень (2.9) – (2.18), які зв'язують значення функцій на мультимножинах потужностей $n+1$ та n, $n = 0,1,2,\ldots$ Ці аналоги отримуються заміною теоретико-множинного об'єднання $\cup$ на операцію об'єднання мультимножин $\cup_{All}$ та одноелементних множин на 1-мультимножини з одноелементною основою. Наведемо аналог тільки для суми:

$$Sum_m(\{d^1\} \cup_{All} \alpha) = d \oplus Sum_m(\alpha), \qquad (2.19)$$

де $\alpha \in 2_m^{Num}$, $d \in Num$. Зауважимо, що елемент d може і належати основі мультимножини α.

Ця рівність випливає з рівності $\alpha = \{d^1\} \cup_{All} (\alpha \setminus_{All} \{d^1\})$, де $d \in O(\alpha)$, яка перевіряється безпосередньо (і є аналогом теоретико множинної рівності $A = \{a\} \cup (A \setminus \{a\})$, де $a \in A$).

Співвідношення типу (2.19) приводять до означення композиції агрегування для мультимножин, яке отримується з відповідного означення заміною теоретико-множинної операції різниці $\setminus$ на операцію над мультимножинами $\setminus_{All}$ та одноелементних множин на 1-мультимножини з одноелементною основою.

Функція вибору ψ розуміється як і раніше. Вона індукує функцію вибору елемента основи мультимножини: $\psi_m(\alpha) \stackrel{def}{=} \psi(O(\alpha))$.

Композиція агрегування Agr_m (випадок мультимножин) функції вигляду $f: D_1 \times ... \times D_k \times D' \times D'' \to D''$, $k = 0,1,2,...$ та елементу $d_0 \in D''$ ставить у відповідність функцію вигляду $F_m: D_1 \times ... \times D_k \times 2_m^{D'} \to D''$, значення якої задаються індукцією за потужністю скінченних мультимножин:

$$F_m(d_1,...,d_k,\varnothing) = d_0,$$

$$F_m(d_1,...,d_k,\alpha) = f(d_1,...,d_k,\psi_m(\alpha),F_m(d_1,...,d_k,\alpha \setminus_{All} \{\psi_m(\alpha)^1\})),$$

де $d_1 \in D_1$, ..., $d_k \in D_k$, $\alpha \in 2_m^{D'}$, $|\alpha| \geq 1$.

Іншими словами, оскільки функція вибору індукує зображення скінченних мультимножин у вигляді послідовностей з повтореннями, то виконується рівність для $n \geq 1$, $k_1, \ldots, k_n \geq 1$:

$$F_m(...,\{d_1^{k_1},...,d_n^{k_n}\}) = \underbrace{f(...,d_1,f(...,d_1,...f(...,d_1}_{k_1},...\underbrace{f(...,d_n,...f(...,d_n}_{k_n},d_0)...).$$

Зауважимо, що з очевидними модифікаціями заповнення табл. 2.3 залишається вірним і для мультимножин.

Наприкінці зробимо декілька загальних висновків щодо агрегатних функцій. Композиція агрегування параметризована за функціями вибору елемента множини, тобто за процедурою зображення скінченних множин (мультимножин) у вигляді послідовностей без повторень (з повтореннями). Але аналіз табл. 2.3 показує, що всі агрегатні функції, наведені в цій таблиці, насправді інваріантні відносно вказаних зображень. Причина полягає або в комутативності та асоціативності вихідних функцій ($\oplus$, max, min, $\wedge$, $\vee$) або в специфічному вигляді вихідних функцій (sm, sm', f_{Ex}, f_{In}, f_θ для кванторів).

Скінченномісні функції табл. 2.3 розпадаються на дві групи: такі, які не зберігають *null*-значення ($\oplus$, max, min, sm, f_θ для кванторів існування та загальності, f_{In}, $\wedge$, $\vee$) і такі, які зберігають *null*-значення (sm', f_{Ex}). Зазначимо, що функції sm' і f_{Ex}, які відповідають агрегатним функціям *Count*(*) і *Exists*,

тривіально зберігають значення *null*, оскільки їх аргументи відмінні від цього особливого елемента.

Зазначений вище факт дозволяє зробити висновок про два різні підходи в SQL при розширенні звичайних функцій на *null*-значення аргументів: при розширенні на рядки таблиць функції завжди зберігають значення *null*, а при розширенні на скінченні множини, які містять *null* (або, змістовно кажучи, на стовпчики таблиць), функції, як правило, не зберігають це особливе значення (*null* може виступати як одиниця, наприклад, для функцій $\oplus$, $\min$, $\max$).

Експерименти, проведені для СУБД MySql, MS SQL Server, DB2, INTERBASE, FIREBIRD, показали коректність розглянутої семантичної моделі ядра мови SQL.

2.7. Семантика групування

Нехай $W \stackrel{def}{=} \{A_1,...,A_n\}$ – множина, яка складається з n атрибутів, $n \geq 1$. Змістовно кажучи, ця множина відповідає атрибутам фрази GROUP BY. Як і раніше $S_W^{\geq}$ позначає сім'ю всіх рядків, схеми яких містять всі атрибути множини W. На цій сім'ї розглянемо бінарне відношення, параметризоване за множиною W: $s_1 \underset{W}{\approx} s_2 \stackrel{def}{\Leftrightarrow} \underset{i=1}{\overset{n}{\&}} s_1(A_i) = s_2(A_i)$.

Очевидно, що відношення $\underset{W}{\approx}$ – еквівалентність; більш того, це ядерна еквівалентність для функції обмеження рядків за множиною атрибутів W. Таким чином, на відміну від відношення сумісності, у термінах якого вводиться операція з'єднання таблиць (підрозділ 2.2), при групуванні рядків значення *null* у різних рядках вважаються рівними. Ця особливість проявляється у тому, що в означенні відношення $\underset{W}{\approx}$ використовується стандартна рівність, а не її модифікація $=_S$.

Еквівалентність $\underset{W}{\approx}$ індукує розбиття множини рядків $S_W^{\geq}$, яке складається з усіх суміжних класів за відношенням $\underset{W}{\approx}$ – $[s]_{\underset{W}{\approx}}$, $s \in S_W^{\geq}$. Значить, це відношення

індукує і розбиття стану довільної таблиці t, схема якої містить всі атрибути з множини W:

Якщо $t=(\bar{t},R)$, де $\bar{t}$ –стан таблиці, а R – її схема, причому стан уточнюється як множина односхемних рядків, то розбиття має вигляд $(\bar{t},R)|_{\underset{W}{\approx}} \overset{def}{=} \{(\bar{t} \cap [s]_{\underset{W}{\approx}},R) \mid s \in \bar{t}\}$;

Якщо $t=(\bar{t},R)$, де стан $\bar{t}$ уточнюється як мультимножина, основою якої є множина односхемних рядків, то розбиття має вигляд $(\bar{t},R)|_{\underset{W}{\approx}} \overset{def}{=} \{(\bar{t}\,|[s]_{\underset{W}{\approx}},R) \mid s \in \mathrm{O}(\bar{t})\}$, де $\mathrm{O}(\bar{t})$ – основа мультимножини $\bar{t}$.

Таким чином, у першому випадку розбиття складається з перетинів стану вихідної таблиці з суміжними класами, а у другому – з обмежень стану вихідної таблиці за суміжними класами.

Формальним уточненням таблиць з групуванням виступають скінченні множини односхемних таблиць. Зазначимо, що в SQL такі множини таблиць є розбиттями таблиць за еквівалентностями вигляду $\underset{W}{\approx}$. Сім'ю всіх таких таблиць з групуванням позначимо T_g. Поява нових моделей структур даних призводить до необхідності введення нових функцій – параметричних *функцій групування* $Gr_W: T \overset{\sim}{\to} T_g$, де $\mathrm{dom}\,Gr_W \overset{def}{=} \{t \mid \exists R(t \in T(R) \& W \subseteq R)\}$, $Gr_W(t) \overset{def}{=} t|_{\underset{W}{\approx}}$.

Розгляд агрегатних функцій і таблиць з групуванням призводить не тільки до розширення класу функцій, які розглядаються, але і до розповсюдження композицій фільтрації і повного образу на нові функції.

Розглянемо множину функцій вигляду $f^{(n)}: X_1 \times ... \times X_n \overset{\sim}{\to} X$, де $n \geq 1$, $X, X_1, ..., X_n \in \{D, 2^D, S, T, T_g, Bool\}$; тут випадок $X = Bool$, як і раніше, відповідає предикатам.

Якщо дублікати рядків у таблицях ігноруються, тобто стани таблиць уточнюються як множини односхемних рядків, визначення композицій фільтрації і повного образу розширюються наступним чином.

Для спрощення ситуації наступні означення сформулюємо в термінах станів таблиць. Нижче $\overline{T},\overline{T_g}$ – множини станів, індуковані відповідними множинами таблиць.

Композиція фільтрації Fl предикату $p^{(n+1)}: X_1\times...\times X_n\times\overline{T}\xrightarrow{\sim} Bool$ ставить у відповідність функцію вигляду $f^{(n+1)}: X_1\times...\times X_n\times\overline{T_g}\xrightarrow{\sim}\overline{T_g}$, $n\geq 0$, довільне значення якої знаходиться за формулою

$$f^{(n+1)}(x_1,...,x_n,\bar{t_g})\simeq\{\bar{t}\,|\,\bar{t}\in\bar{t_g}\ \&\ p^{(n+1)}(x_1,...,x_n,\bar{t})\simeq true\}. \qquad (2.20)$$

У випадку, коли хоча б одне значення вигляду $p^{(n+1)}(x_1,...,x_n,\bar{t})$, де $\bar{t}\in\bar{t_g}$, у правій частині рівності (2.20) невизначене, то і ліва частина цієї рівності покладається невизначеною.

Композиція взяття повного образу Im функції вигляду $f^{(n+1)}: X_1\times...\times X_n\times S\xrightarrow{\sim} D$ ставить у відповідність функцію вигляду $g^{(n+1)}: X_1\times...\times X_n\times\overline{T}\xrightarrow{\sim} 2^D$, $n\geq 0$, довільне значення якої знаходиться наступним чином

$$g^{(n+1)}(x_1,...,x_n,\bar{t})\tilde{=}\{f^{(n+1)}(x_1,...,x_n,s)\,|\,s\in\bar{t}\}. \qquad (2.21)$$

Композиція взяття повного образу Im функції вигляду $f_R^{n+1}: X_1\times...\times X_n\times\overline{T}\xrightarrow{\sim} S(R)$ ставить у відповідність функцію вигляду $g^{(n+1)}: X_1\times...\times X_n\times\overline{T_g}\xrightarrow{\sim}\overline{T}$, $n\geq 0$, довільне значення якої знаходиться за формулою

$$g^{(n+1)}(x_1,...,x_n,\bar{t_g})\tilde{=}\{f_R^{n+1}(x_1,...,x_n,\bar{t})\,|\,\bar{t}\in\bar{t_g}\}. \qquad (2.22)$$

У рівностях (2.21), (2.22) двох останніх означень для визначеності лівих частин використовуються домовленості щодо визначеності, аналогічні першому означенню. Зауважимо, що перше означення композиції Im потрібне для розповсюдження агрегатних функцій на таблиці, а друге – для побудови таблиці-результату за проміжною таблицею з групуванням (див. рис. 2.3).

На закінчення підрозділу розглянемо більш детально структуру відношень $\underset{W}{\approx}$. Далі замість $\underset{\{A\}}{\approx}$, $S^{\geq}_{\{A\}}$ будемо писати $\underset{A}{\approx}$, $S^{\geq}_{A}$. Неважко перевірити рівність $\underset{W}{\approx} = \bigcap\limits_{i=1}^{n} \underset{A_i}{\approx}$, де $W \stackrel{def}{=} \{A_1,...,A_n\}$. Отже, еквівалентність $\underset{W}{\approx}$ (на $S^{\geq}_{W}$) є теоретико-множинним перетином всіх еквівалентностей $\underset{A}{\approx}$ (на $S^{\geq}_{A}$), де $A \in W$. Це зображення призводить до того, що суміжні класи за еквівалентністю $\underset{W}{\approx}$ є теоретико-множинним перетином суміжних класів за еквівалентностями $\underset{A_1}{\approx},\ldots,\underset{A_n}{\approx}$; тобто виконуються наступні рівності для всіх рядків $s \in S^{\geq}_{W}$, $s_1 \in S^{\geq}_{A_1}$, ..., $s_n \in S^{\geq}_{A_n}$:

$$[s]_{\underset{A_1}{\approx}} \cap \ldots \cap [s]_{\underset{A_n}{\approx}} = [s]_{\underset{W}{\approx}}; \; [s_1]_{\underset{A_1}{\approx}} \cap \ldots \cap [s_n]_{\underset{A_n}{\approx}} = \begin{cases} \varnothing, \text{ якщо } \bigcap\limits_{i=1}^{n} [s_i]_{\underset{A_i}{\approx}} = \varnothing, \\ [s']_{\underset{W}{\approx}}, \text{ інакше;} \end{cases}$$

де $s' \in \bigcap\limits_{i=1}^{n} [s_i]_{\underset{A_i}{\approx}} \neq \varnothing$ – довільний фіксований елемент.

Встановлена структура суміжних класів за відношенням вигляду $\underset{W}{\approx}$ уточнює розбиття вихідної таблиці на підтаблиці (групування за першим атрибутом), кожної підтаблиці на свої підтаблиці (групування за другим атрибутом) і аналогічні подальші розбиття для решти атрибутів. Наведені означення неважко модифікувати для врахування дублікатів рядків таблиць, тобто на випадок мультимножин.

Приклади задання семантики запитів з групуванням та з агрегатними функціями можна знайти в [41, підрозд. 3.9].

Вирази фрази HAVING та списку вибірки задають предикати та функції, визначені на таблицях – підтаблицях проміжної таблиці. Атрибути, які відсутні в списку групування, розрізняються своїми значеннями у рядках підтаблиць. Тому такі атрибути можуть входити в означені вирази лише разом з агрегатними функціями. На використання атрибутів групування ніяких обмежень не накладається.

2.8. Оператори оновлення даних

Для опису семантики операторів оновлення даних мови SQL (INSERT, UPDATE, DELETE) не потрібні нові конструкції. Обґрунтуємо цю тезу, розглядаючи уточнення станів таблиць у вигляді множин односхемних рядків.

Нехай функція sel_p задає семантику оператора вибірки рядків таблиці, які задовольняють предикату p: $sel_p(t) \tilde{=} (\{s \mid s \in \bar{t} \ \& \ p(s) \tilde{=} true\}, R)$, де $\bar{t}$ – стан таблиці t, а R – її схема. Тоді семантика оператора *вилучення* рядків таблиці, які задовольняють предикату p, задається функцією $del_p(t) \tilde{=} t \setminus_t sel_p(t)$.

Нехай функція $sel_{p,f}$ задає семантику оператора вибірки рядків таблиці, які задовольняють предикату p, з наступною модифікацією вибраних рядків згідно з функцією f: $sel_{p,f}(t) \simeq (\{f(s) \mid s \in \bar{t} \ \& \ p(s) \tilde{=} true\}, R)$, де $\bar{t}$ – стан таблиці t, а R – її схема. Функції вигляду sel_p і $sel_{p,f}$ будуються за допомогою композицій фільтрації і повного образу (див. підрозділ 2.3). В цих позначеннях семантика оператора *модифікації* рядків таблиці, які задовольняють предикату p, згідно з функцією f задається такою функцією: $update_{p,f}(t) \tilde{=} del_p(t) \bigcup_t sel_{p,f}(t)$.

Нарешті, семантика оператора *поповнення* рядків таблиці рядками іншої таблиці задається просто об’єднанням таблиць $insert(t_1, t_2) = t_1 \bigcup_t t_2$.

Отже, дійсно, функції, які задають семантику операторів оновлення, похідні від функцій, які задають семантику запитів.

2.9. Визначення семантики операторів маніпулювання даними

В попередніх підрозділах була побудована програмна алгебра, орієнтована на визначення семантики операторів маніпулювання даними SQL-подібних мов. Носій цієї алгебри складається з часткових багатомісних функцій вигляду $f^{(n)}: X_1 \times ... \times X_n \rightharpoonup X$, де $n \geq 1$, $X, X_1, ..., X_n \in \{D, 2^D, S, T, T_g, Bool\}$. Сигнатура містить композиції фільтрації, взяття повного образу, агрегування та стандартну підстановку. Основний результат розділу полягає в наступній тезі. В її

формулюванні фігурує множина базових функцій та предикатів Σ, яку формують такі елементи:

1. теоретико-множинні операції та функції об‘єднання, перетину та різниці таблиць[12]; функція вилучення дублікатів;

2. параметрична функція побудови упорядкованих таблиць $Orderby_{\bar{A}}$;

3. предикати та функції, потрібні для побудови агрегатних функцій композицією агрегування згідно з табл. 2.3;

4. параметрична функція групування Gr_W;

5. операції та функції внутрішнього і зовнішнього з‘єднання;

6. операції кон‘юнкції, диз‘юнкції та заперечення тризначної сильної логіки Кліні;

7. операція об‘єднання сумісних рядків $\bar{\cup}$, параметричні функції іменування та розіменування рядків за атрибутами;

8. вихідні операції та предикати універсального домену (арифметика, рівність, порядок та т.і.) чи сім’ї доменів.

Теза. Замикання множини базових функцій та предикатів Σ композиціями фільтрації, взяття повного образу, агрегування та підстановки містить функції, які є семантиками всіх операторів маніпулювання даними SQL-подібних мов за винятком рекурсивних запитів. □

Теза обґрунтовується конструкціями розд. 2, а також низкою представницьких прикладів еталонування запитів, наведених у [41, підрозд. 3.3-3.5, 3.9].

[12] Нагадаємо, що відповідні маніпуляції над таблицями називаються операціями чи функціями в залежності від уточнення станів таблиць у вигляді множин односхемних рядків чи мультимножин, основами яких виступають множини односхемних рядків.

РОЗДІЛ 3. СЕМАНТИКА РЕКУРСИВНИХ ЗАПИТІВ

Метою розділу є дослідження рекурсивних запитів в SQL, які задаються через CTE вирази (Common Table Expression) у їх рекурсивному вигляді, виявлення структури таких запитів та задання їх семантики. Крім того, побудована програмна алгебра введенням нової композиції рекурсії, яка задає множину семантичних функцій SQL, включаючи рекурсивні запити.

Об’єкт дослідження – CTE вирази мови SQL у їх рекурсивній формі.

Основні результати розділу: досліджена структура рекурсивних запитів, побудована операційна та денотаційна семантика цих запитів, доведена еквівалентність операційної та денотаційної семантики. Побудована відповідна програмна алгебра.

3.1. Загальні зауваження

Мова SQL надає потужні та елегантні засоби для маніпулювання табличними структурами даних, які є мультимножинами рядків [67]. Операції SQL в більшості випадків виявляються набагато ефективнішими при виконанні запитів над таблицями, ніж виконання тих же самих запитів засобами традиційних мов програмування. При цьому SQL-запити простіші в написанні. Це пояснюється високим рівнем абстракції операцій SQL, орієнтованих на роботу з мультимножинами на відміну від традиційних мов програмування, які потребують поелементної обробки колекцій[13], зокрема мультимножин. Наприклад, в SQL існують аналоги стандартних теоретико-множинних операцій, таких як операції перетину, об’єднання та різниці множин.

Разом з тим, платою за такий високий рівень операцій є більш обмежені функціональні можливості мови SQL в порівнянні з традиційними мовами програмування. Існують класи запитів до табличних структур даних, які не можуть бути реалізовані засобами тільки SQL. В той же час такі запити

[13] Тут і далі під колекціями розуміються множини, мультимножини та послідовності елементів.

реалізуються традиційними мовами (програмування). Одним із прикладів є запити до ієрархічних структур даних [105, 103, 88]. На сьогодні розроблено декілька методів відображення ієрархічних структур у таблиці. Головне питання полягає в наявності операцій високого рівня для маніпулювання ієрархічними даними, зокрема, для побудови так званих рекурсивних запитів. В перших версіях SQL ці можливості були відсутні. Тому маніпулювання такими даними програмувалось засобами процедурних розширень SQL і мало чим відрізнялось від програмування на класичних мовах програмування. Тільки в стандарті SQL:99 було введено розширення, яке допускає рекурсивні запити [103]. В подальшому це розширення мови було втілено в таких провідних СУБД як DB2 та MS SQL Server. В той же час Oracle не підтримав стандарт і має своє розширення для написання рекурсивних запитів [84].

3.2. Рекурсивні запити

Рекурсивні запити використовуються, зокрема, для роботи з ієрархічними структурами даних. Таблиці не підтримують ієрархічні структури явним чином, але вони можуть відображатися в табличні структури даних. Ієрархічні дані представляються в таблицях кількома методами. Найбільш поширеним є використання так званих списків суміжності, які і будуть розглянуті далі. Розглянемо таку класичну ієрархічну структуру як бюрократична організація, що будується за принципом керівник-підлеглий. Кожний підлеглий має тільки одного керівника і кожний керівник має багато підлеглих. Визначимо таблицю співробітників Employees.

Таблиця 3.1 – Таблиця співробітників Employees

employeeId	managerId	lastName	firstName	title	birthDate	hireDate	city	region	address

Перші два атрибути цієї таблиці employeeId і managerId використовуються для задання ієрархічних зв’язків на рядках. Атрибут employeeId містить унікальний ідентифікатор працівника і є первинним ключем таблиці. Атрибут managerId містить ідентифікатор керівника цього співробітника і є зовнішнім

ключем, який посилається на атрибут employeeId в тій же самій таблиці. Якщо у співробітника немає керівника, тобто він є керівником вищого рівня, то атрибут managerId має особливе значення NULL. Інші атрибути таблиці містять персональну інформацію про співробітника: його ім’я (firstName) та прізвище (lastName), посаду (title), дату народження (birthDate), дату найму в організацію (hireDate), місто (city) та область (region), де він працює, адресу (address). Фактично атрибути employeeId і managerId утворюють зв’язок типу один до багатьох на рядках однієї таблиці.

Такі ієрархічні відношення ще називають відношеннями предок-нащадок. Наведений вище приклад списку суміжності демонструє найпростішій або базовий тип ієрархії, коли кожний нащадок має тільки одного предка. В загальному випадку у кожного нащадка може існувати декілька предків.

В теорії графів списку суміжності з кількістю нащадків не більше ніж n відповідає орієнтований ациклічний граф з валентністю вершин не більше n. Стандартно валентністю вершини називається кількість ребер, інцидентних вершині [31]. Але в даному розділі під валентністю буде розумітись кількість тільки вхідних ребер, інцидентних вершині. Для наведеного вище приклада валентність вершин не перевищує одиниці.

Наступним прикладом є генеалогічне дерево, яке містить родинні зв’язки між людьми. У кожній людини є батько та матір. Таким чином, кожний нащадок має двох предків (точніше кажучи, не більше двох предків). Для завдання ієрархічних зв’язків такого типу використовуються так звані дуальні списки суміжності. В теорії графів їм відповідають ациклічні орієнтовані графи з валентністю вершин не більш ніж два. Якщо валентність вершини дорівнює двом, то існує інформація про батько і матір людини; якщо валентність дорівнює одиниці, то інформація про одного з батьків відсутня; якщо ж, нарешті, валентність вершини дорівнює нулю, то відсутня інформація про обох батьків. З огляду на скінченність в будь-якому випадку в такому графі повинна бути хоча б одна вершина з нульовою валентністю.

Таблиця, яка задає генеалогічне дерево, наведена нижче. Перші три атрибути використовуються для встановлення дуальних зв’язків між рядками таблиці.

Таблиця 3.2 – Таблиця родинних зв’язків FamilyTree

personId	patherId	motherId	lastName	firstName	birthDate	city	region	address

Атрибут personId містить, як і раніше, унікальний ідентифікатор людини, тобто є первинним ключем. Атрибути patherId та motherId посилаються на батька та матір людини відповідно. Кожний з них є зовнішнім ключем та посилається на один і той самий атрибут personId. Інші атрибути таблиці містять, як і раніше, прізвище (lastName) та ім’я (firstName) людини, дату народження (birthDate), місто (city), область (region) та адресу проживання (address) відповідно.

Далі розглянемо так звані навігаційні запити, тобто запити, в умові фільтрації яких використовується зв’язок предок/нащадок. Наприклад, потрібно знайти всі вузли, які є нащадками заданого вузла. Найпростішими навігаційними запитами є такі, коли треба знайти всіх синів вказаного вузла. Наприклад, треба знайти всіх працівників, керівником яких є співробітник з ідентифікаційним кодом "010101".

```
SELECT Employees.employeeId, Employees.lastName, Employees.firstName,
Employees.title, Manager.lastName, Manager.firstName
FROM Employees INNER JOIN Employees Manager
        ON Employees.managerId = Manager.employeeId
WHERE Manager.employeeId = '010101';
```

Звернемо увагу, що таблиця Employees поєднується сама з собою. Для того, щоб уникнути колізії імен, другий екземпляр таблиці перейменовується за допомогою псевдоніма Manager.

В приведеному прикладі треба перейти на один рівень нижче від заданого вузла. Аналогічним чином будуються запити, коли треба перейти на будь-яку наперед задану кількість рівнів.

Але коли кількість рівнів, які треба обійти, невідома заздалегідь, то запит не може бути побудований стандартними засобами оператора SELECT. Наприклад, нехай потрібно знайти всіх нащадків якої-небудь людини. Природно, людина

вважається дитиною деякої особи, якщо її patherId чи motherId дорівнює personId цієї особи.

Для цього можна використати курсори SQL та стандартні ітеративні методи обходу генеалогічного дерева, наприклад, обхід в глибину. Але коли даних багато, використання курсорів не є ефективним.

Більш ефективним є метод, коли створюється допоміжна тимчасова таблиця. На першому кроці туди записується тільки один рядок з тією людиною, нащадки якої шукаються. Далі виконується ітеративна процедура поповнення таблиці. На і-му кроці ітерації в таблицю додаються всі нащадки, які знаходяться на і-му рівні глибини відносно свого спільного предка. Це робиться одним оператором SELECT. Процес закінчується, коли оператор SELECT повертає порожню таблицю.

Кількість операторів SELECT дорівнює кількості рівнів в ієрархічній структурі. Так, якщо є мільйон вузлів, розташованих на ста рівнів, то буде виконано лише сто операторів. Тобто множинно-орієнтований підхід діє набагато ефективніше попереднього процедурно-орієнтованого. Але він все ще вимагає створення процедур і виконання кількох операторів. Якщо процедура виконується не на сервері, а на комп'ютері клієнта, то кожний раз потрібно звернення до серверу, що знижує продуктивність (мова йде про клієнт-серверну архітектуру).

В стандарті SQL:99 був введений новий оператор – так званий загальний табличний вираз CTE, який у своїй рекурсивній формі дозволяє побудувати результат без необхідності звернення до процедурного коду.

Вирази CTE в своїй нерекурсивній формі подібні звичайним представленням (view) зі сферою дії, обмеженою одним запитом. На них можна дивитися як на підзапити, які відокремлені від основного запита та мають своє ім'я. Такий підзапит можна використовувати в декількох місцях основного запиту. Він починається з ключового слова WITH, за яким йде ім'я підзапиту та перелік параметрів. Після CTE виразу починається головній запит, який має посилання на CTE (виклик CTE). Далі приведена загальна структура запиту з CTE.

```
/*CTE*/
WITH CTEName (parameters)
AS (Simple Subquery)

/* головний запит*/
SELECT
FROM CTEName;
```

Рекурсивна форма CTE має більш складний вигляд. Вона складається з двох операторів SELECT, які поєднані операцією UNION ALL.

Розглянемо приклад. Нехай родинне дерево (дерево нащадків) будується для людини, ідентифікатор якої personId дорівнює 10. Після фрази WITH вказується назва таблиці, яка будується за допомогою CTE, та список її атрибутів. Атрибут level є обчислюваним і містить рівень, на якому розташовані нащадки відповідного рівня людини, для якої будується запит. Так, сама людина має рівень 1, її діти мають рівень 2, онуки – 3 і т.д.

Перший оператор SELECT задає ініціальну таблицю. Для даного приклада вона буде складатися тільки з одного рядка, який відповідає вказаній людині.

Другий оператор SELECT формує результуючу таблицю рекурсивним (в іншій термінології рекурентним або індуктивним) чином.

```
WITH Tree( lastName, firstName, personID, level)
AS ( /* начальний стан таблиці */
SELECT lastName, firstName, personID, 1
FROM FamilyTree A
WHERE personID = 10
/* рекурсивний виклик */
UNION ALL
SELECT Node.lastName, Node.firstName, Node.personID, T.level + 1
FROM FamilyTree Node
JOIN Tree T
ON Node.motherID = T.personID OR Node.fatherID = T.personID
```

```
)
/*головний запит */
SELECT personID, lastName, firstName, level
FROM Tree;
```

3.3. Операційна семантика рекурсивних запитів

Перейдемо тепер до опису семантики оператора CTE. Засоби задання семантики об’єктів різноманітних предметних областей можна на відповідному рівні абстракції розділити на явні (конотативні, операційні) та неявні (денотативні). Явні засоби характеризуються наявністю (ефективних) процедур задання; що ж до неявних засобів, то тут об’єкти визначаються за допомогою специфікації властивостей об’єктів.

Денотативні (денотаційні) засоби задання є досить потужними та зручними в застосуваннях; в математиці та інформатиці ці засоби виступають у вигляді так званого методу нерухомої точки [16, 40, 19]. Кажучи більше точно, об’єкти задаються як найменші, відносно певного порядку, рішення рівнянь вигляду $x = F(x)$, $x \in D$, де D – (частково впорядкована) множина певного класу.

Нижче буде задана як операційна, так і денотаційна семантика рекурсивних запитів, а також доведена їх еквівалентність при виконанні деяких умов.

Нехай задані унарна функція $g : T \to T$ та бінарна функція $f : T \times T \to T$. CTE вираз уточнюється за допомогою бінарної композиції рекурсії C, яка функціям g, f ставить у відповідність нову унарну часткову функцію $C(g, f)$; значення останньої функції на аргументі t знаходиться наступним чином. Будуємо послідовність таблиць $t_1, t_2, t_3, \ldots, t_i, \ldots$, де $t_1 = g(t)$, $t_2 = f(t_1, t)$, $t_3 = f(t_2, t)$, … $t_i = f(t_{i-1}, t)$, …

Якщо на деякому кроці $n+1$, де $n = 1,2,...$, отримується порожня таблиця, причому усі попередні таблиці послідовності не є порожніми, то процес обчислень закінчується з результатом $t_1 \cup_{All} t_2 \cup_{All} \ldots \cup_{All} t_n$.

В іншому випадку (тобто всі таблиці $t_i, i = 1,2,...$ непорожні) значення $C(g, f)(t)$ покладається невизначеним. Таким чином,

$$C(g, f)(t) \simeq \begin{cases} \bigcup_{i=1,2,...} {}_{All} t_i, \text{ якщо } \exists n(\bar{t}_{n+1} = \varnothing_m \wedge \forall i \le n(\bar{t}_i \neq \varnothing_m)), \\ \text{невизначене в іншому випадку}. \end{cases} \quad (3.1)$$

Вище операція $\cup_{All}$ – операція об'єднання мультимножин з врахуванням дублікатів [67, підрозд. 3.4, с.153; розд. 2].[14] Нижченаведена діаграма ілюструє процедуру знаходження значень функції, яка будується композицією рекурсії.

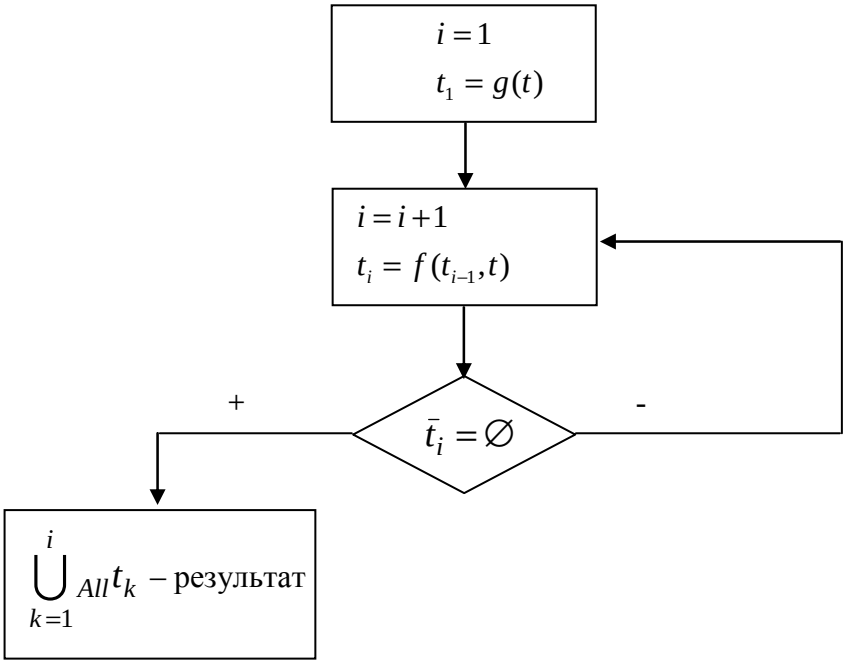

Рис. 3.1. Блок-схема процедури знаходження значень вигляду $C(g, f)(t)$

Вказана процедура знаходження значень функції $C(g, f)$ модифікується очевидним чином на випадок часткових функцій.

Підкреслимо ще раз, що операція об'єднання $\cup_{All}$ є об'єднанням мультимножин з врахуванням дублікатів. Характеристична властивість цього об'єднання полягає в тому, що коли обидві таблиці-аргументи не є порожніми, то

[14] За іншою термінологією ця операція називається операцією додавання мультимножин [6].

результат відрізняється від аргументів (навіть якщо один аргумент "вкладається" в інший).[15]

Вкажемо другий спосіб обчислення значення $C(g, f)(t)$, який в деяких випадках є більш зручним. А саме: будуються дві послідовності таблиць $t_1^{'}, t_2^{'}, \ldots$ та $t_1, t_2, \ldots$: $t_1^{'} = g(t), t_1 = g(t)$, $t_2^{'} = f(t_1^{'}, t), t_2 = t_1 \cup_{All} t_2^{'}$, … $t_i^{'} = f(t_{i-1}^{'}, t), t_i = t_{i-1} \cup_{All} t_i^{'}$, …

Якщо на деякому кроці $n+1$, де $n = 1,2,\ldots$, виконується рівність $t_n = t_{n+1}$, то процес обчислень закінчується, і таблиця t_n покладається рівним результату. В протилежному випадку (тобто для всіх $n = 1,2,\ldots$, виконується нерівність $t_n \neq t_{n+1}$) значення $C(g, f)(t)$ невизначено.

Змістовно кажучи, значення нової функції визначено тоді і тільки тоді, коли послідовність $t_i, i = 1,2,\ldots$ стабілізується. Нижченаведена діаграма ілюструє описану процедуру знаходження значень функції, яка будується композицією рекурсії (рис. 3.2).

В вищенаведеному прикладі побудови генеалогічного дерева функція g задається першим оператором SELECT

```
SELECT lastName, firstName, personID, 1
FROM FamilyTree A
WHERE personID = 10;
```

Тут таблиця FamilyTree виступає аргументом. Функція f задається другим оператором SELECT

```
SELECT Node.lastName, Node.firstName, Node.personID, T.level + 1
FROM FamilyTree Node
JOIN Tree T
```

---

[15] Точніше кажучи, мова йде про наступне бінарне відношення над мультимножинами $\alpha \lhd \beta \overset{def}{\Leftrightarrow} O(\alpha) \subseteq O(\beta) \wedge \forall x(x \in O(\alpha) \Rightarrow \alpha(x) \leq \beta(x))$. Тут $O(\alpha), O(\beta)$ – основи мультимножин α, β відповідно [67, підрозд. 3.4, с. 151-155]. Це бінарне відношення є частковим порядком, воно перетворює сім'ю мультимножин в рішітку (подробиці дивись в [21, 6]).

ON Node.motherID = T.personID OR Node.fatherID = T.personID;

В цьому запиті таблиці FamilyTree та Tree виступають аргументами.

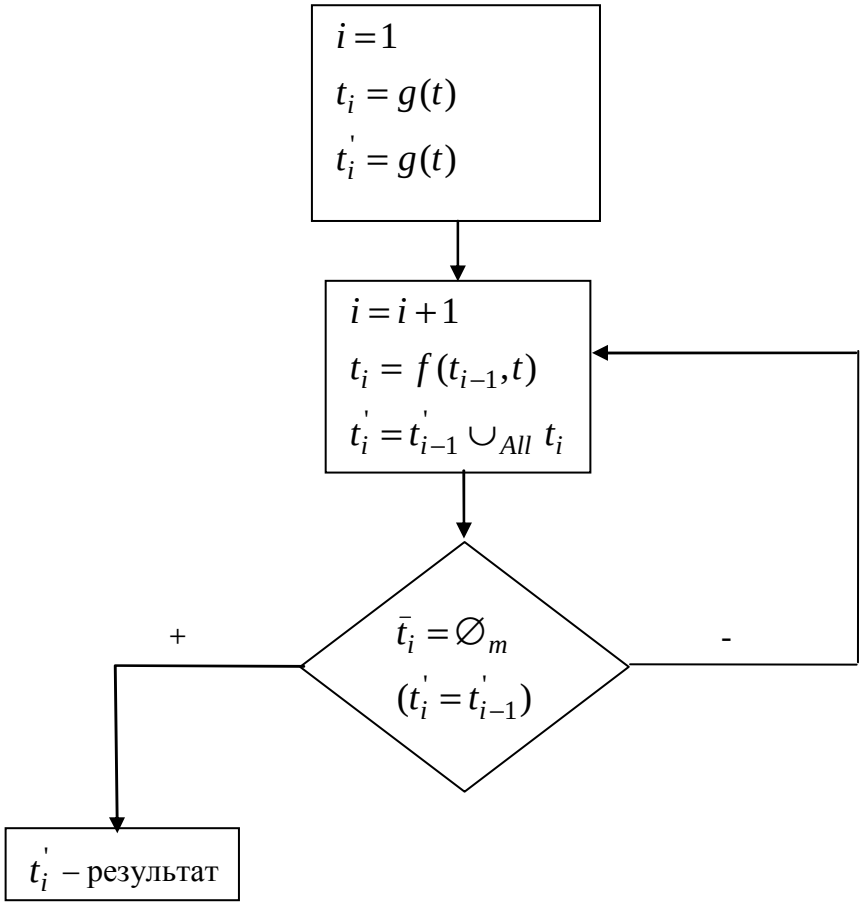

Рис. 3.2. Блок схема процедури знаходження значень вигляду $C(g, f)(t)$

За умови, що таблична операція f зберігає порожню таблицю за першим аргументом (тобто $f((\varnothing_m, R), t) = \varnothing_m$ для всіх таблиць t), неважко довести наступний критерій скінченності.

Лема 3.1 Має місце еквівалентність: стан об'єднання $\bigcup_{i=1,2,...}{}_{All}\, t_i$ – скінченний (тобто вказане об'єднання є таблицею) $\Leftrightarrow \exists i(\bar{t}_i = \varnothing_m)$.

3.3. Денотаційна семантика рекурсивних запитів

Перейдемо до задання денотаційної семантики. Для цього розглянемо наступне рівняння з параметром t відносно змінної x (область значень – T).

$$x = g(t) \bigcup_{All} f(x, t). \qquad (3.2)$$

Для подальшого розгляду припустимо, що операція f є дистрибутивною за першим аргументом відносно розповсюдження табличної операції $\bigcup_{All}$, тобто $f(\bigcup_{k\in K} {}_{All}\, t_k^{'}, t^{''}) = \bigcup_{k\in K} {}_{All}\, f(t_k^{'}, t^{''})$ для довільних таблиць $t_k^{'}, t^{''}$ та множини індексів K, а також зберігає порожню таблицю за першим аргументом.

Теорема 3.1. Об'єднання $\bigcup_{i=1,2,...} {}_{All}\, t_i$, де $t_1 \overset{def}{=} g(t)$, $t_i \overset{def}{=} f(t_{i-1}, t)$ в припущенні скінченності стану є розв'язком рівняння (3.2).

Доведення. Нехай стан вказаного об'єднання скінченний. Тоді згідно з лемою 3.1 існує таке k, що $\bar{t}_k = \varnothing_m$. Тоді, з властивості збереження порожньої таблиці операцією f випливає, що всі $\bar{t}_m = \varnothing_m$, де $m \geq k$. Таким чином, маємо рівність $\bigcup_{i=1,2,...} {}_{All}\, t_i = \bigcup_{i=1}^{k-1} {}_{All}\, t_i$. Підставимо праву частину останньої рівності в праву частину рівняння (3.2), отримуємо вираз $g(t) \bigcup_{All} f(\bigcup_{i=1}^{k-1} {}_{All}\, t_i, t)$.

В силу дистрибутивності за першим аргументом функції f отримуємо об'єднання вигляду $g(t) \cup_{All} f(t_1, t) \cup_{All} f(t_2, t) \cup_{All} \ldots \cup_{All} f(t_{k-1}, t)$.

Використовуючи означення t_i $(t_i = f(t_{i-1}, t))$, останнє об'єднання приймає вигляд $t_1 \cup_{All} t_2 \cup_{All} \ldots \cup_{All} t_k$.

Враховуючи рівність $\bar{t}_k = \varnothing_m$, отримуємо, що права частина рівняння (3.2) при розглянутій підстановці дорівнює $\bigcup_{i=1}^{k-1} {}_{All}\, t_i$, тобто збігається з лівою частиною цього рівняння. Теорема доведена.

Таким чином, якщо операція f є дистрибутивною за першим аргументом відносно розповсюдження табличної операції $\bigcup_{All}$, а також зберігає порожню таблицю за першим аргументом і результат композиції $C(g, f)$ визначений на таблиці t (тобто є таблицею, станом якої є скінченна мультимножина), то об'єднання з правої частини кускової схеми (3.1) є рішенням рівняння (3.2).

Відкритим залишається питання о наявності інших рішень рівняння (3.2), а також загальне питання опису множини всіх розв’язків.

В розділі 3 була визначена операційна та денотаційна семантика рекурсивних запитів в SQL-подібних мовах. Наявність відповідного уточнення – композиції рекурсії – дозволяє задати семантику CTE виразів мови SQL.

Збагачення композицією рекурсії програмної алгебри, орієнтованої на визначення семантики операторів маніпулювання даними SQL-подібних мов, дає змогу задати семантику CTE виразів, зокрема, в рекурсивній формі.

Носій збагаченої алгебри складається, як і раніше, з часткових багатомісних функцій вигляду $f^{(n)}: X_1 \times ... \times X_n \xrightarrow{\sim} X$, де $n \geq 1$, $X, X_1, ..., X_n \in \{D, 2^D, S, T, T_g, Bool\}$. Сигнатура містить композиції фільтрації, взяття повного образу, агрегування, стандартну підстановку та нову композицію рекурсії. Множина базових функцій та предикатів Σ залишається незмінною.

Теза (розширений варіант). замикання множини базових функцій та предикатів Σ композиціями фільтрації, взяття повного образу, агрегування, підстановки та рекурсії містить функції, які є семантиками всіх операторів маніпулювання даними SQL-подібних мов, включаючи CTE вирази в їх рекурсивній формі.

Теза обґрунтовується конструкціями розділу 2, 3, а також низкою представницьких прикладів еталонування запитів, наведених у [41, підрозд. 3.3-3.5, 3.9].

РОЗДІЛ 4. КЛАСИФІКАЦІЯ ТА ОГЛЯД МОДЕЛЕЙ ДАНИХ

Мета розділу полягає в класифікації та огляді моделей даних, спираючись на єдиний концептуальний базис – колекції та відношення на колекціях. Розглядаються моделі даних табличного, ієрархічного, мережного типів, моделі даних на базі багаторівневих та рекурсивних таблиць.

Об’єкт дослідження – моделі даних.

Основні результати розділу: розглянуті та описані табличні моделі даних, моделі на базі багаторівневих таблиць, ієрархічні моделі, моделі даних на базі рекурсивних таблиць та мережні моделі. Їх визначення проводиться на основі поняття колекції та відношень на них. Поняття відношення на множинах розширене на мультимножини та послідовності. Визначення зв’язків типу один до одного та один до багатьох розширено на випадок мультимножин та послідовностей.

4.1. Основні поняття

Модель даних відноситься до фундаментальних понять в теорії баз даних. Це поняття згадувалось ще в революційній статті Е. Кодда, присвяченій реляційним базам даних [90], але без деталізації. Пізніше Кодд розглянув це поняття більш детально в статті "Моделі даних в керуванні базами даних" в 1981 р. [91]. Згідно з визначенням Кодда модель даних складається з трьох компонентів:

a) сукупність типів об’єктів даних, що утворюють базові будівні блоки для будь-якої бази даних, яка відповідає моделі;

b) сукупність загальних правил цілісності, що обмежують множину екземплярів тих типів об’єктів, які на законній підставі можуть з’явитись в будь-якій базі даних, яка відповідає моделі;

c) сукупність операцій, які застосовуються к екземплярам об’єктів для вибірки та інших дій.

Ці компоненти не є взаємно незалежними. Сукупність правил цілісності та операцій багато в чому залежить від типів об’єктів даних. Таким чином, перший компонент відіграє центральну роль в моделях даних. Інші два компонента є підпорядкованими і можуть навіть не включатись в поняття моделі даних. Наприклад в підручнику А.С.Маркова [41] модель даних визначається як трійка $\{D, R, A\}$, де

D – задана множина (носій структури);

R – скінченна сукупність відносин, в яких знаходяться елементи множини;

A – обмежувальні умови, які накладаються на відносини (аксіоми структури).

Тобто D та R складають перший компонент із визначення Кодда, а A є другим компонентом – обмеженнями цілісності. В той же час Дж. Мартин в своїй монографії [42] відносить до моделі даних лише сукупність даних та зв’язків між ними. Зв’язки відіграють важливу роль в моделі даних; і хоча в визначенні Кодда вони явним чином не вказані, в інших визначеннях зв’язки, як правило, фігурують як окрема сутність. Так Дж. Мартин відзначає: "Если бы назначением базы данных было только хранение данных, то структура ее была бы простой. Причина же в ее сложности объясняется тем, что она должна обеспечивать еще и связи между различными элементами данных" [42, с. 68].

Далі модель даних буде задаватися через

– тип атомарних даних та тип колекцій даних;

– тип допустимих зв’язків на колекціях даних;

– множину операцій.

Множина операцій є необов’язковим компонентом і може бути відсутня для деяких моделей даних.

Відмітимо, що потрібно розрізняти терміни "модель даних" і "модель бази даних". Останній задає множину допустимих станів якоїсь конкретної бази даних, в той час як перший задає множину всіх допустимих моделей баз даних.

Дані об’єднуються в рядки. Кожний рядок може розглядатися як множина пар (ім’я, значення) або як функція, яка відображає множину імен в множину

значень. Рядок є аналогом поняття структури або запису в мовах програмування. В той же час, на відміну від цих понять, він не вимагає порядку на атрибутах (іменах). Тобто два рядка дорівнюють один одному, якщо для кожного атрибуту одного рядка існує атрибут з таким самим ім'ям в іншому рядку, значення яких співпадають, і навпаки. Два рядка будемо називати однотипними, якщо множини їх атрибутів співпадають.

Однотипні рядки групуються у множини, мультимножини та послідовності. Для узагальненої назви таких сукупностей будемо використовувати термін "колекція". Тобто колекція відіграє роль метатипу, значеннями або спеціалізаціями якого виступають множини, мультимножини та послідовності, . Слід відмітити, що на сьогодні не існує загальноприйнятого набору типів, які включаються до колекції. Так, іноді до колекції можуть входити списки, множини, мультимножини, асоціативні масиви, дерева, графи [97] або множини, класи, сім'ї та мультимножини [98].

Перейдемо до зв'язків між колекціями. Зв'язки будуть задаватися за допомогою відношень. Для множин відношення розуміються в стандартному сенсі як підмножини декартового добутку множин.

Для послідовностей та мультимножин такий спосіб не підходить внаслідок наявності дублікатів. Наприклад, нехай є дві мультимножини $\{a,a\}$ та $\{b,b\}$. Тоді їх декартовим з'єднанням[16] буде мультимножина $\{(a,b),(a,b),(a,b),(a,b)\}$. В ній відсутня інформація про те, які саме екземпляри елементів a та b входять до декартового з'єднання. Тому відношення, яке складається з першого екземпляру елемента a і першого екземпляру елемента b, виглядає так само, як і відношення, яке складається з другого екземпляру елемента a і другого екземпляру елемента b, а саме $\{(a,b)\}$. Як буде показано нижче, ця інформація є важливою при маніпулюванні з відношеннями. Така ж сама ситуація має місце і для послідовностей.

[16] Нагадаємо, що для мультимножин операція, аналогічна декартовому добутку, називається декартовим з'єднанням ([67]; розділ 2).

Введемо операцію розширеного декартового з'єднання. Для наведених вище мультимножин, розширеним декартовим з'єднанням буде множина вигляду $\{((a,1),(b,1)),((a,1),(b,2)),((a,2),(b,1)),((a,2),(b,2))\}$, другий компонент елементів якої враховує, який саме екземпляр входить до елементів з'єднання. Тоді відношення, яке складається з першого екземпляру елемента a і першого екземпляру елемента b, задається як $\{((a,1),(b,1))\}$, а відношення, яке складається з другого екземпляру елемента a і другого екземпляру елемента b задається як $\{((a,2),(b,2))\}$. Для послідовностей будемо враховувати місце (позицію), яке займає елемент в послідовності. Наприклад для послідовностей (a,b,a) і (c) розширене декартове з'єднання $\{((a,1),(c,1)),((b,2),(c,1)),((a,3),(c,1))\}$ означає, що в парі $((a,1),(c,1))$ елемент a знаходиться на першій позиції в вихідній послідовності, а в парі $((a,3),(c,1))$ елемент a знаходиться на третій позиції в вихідній множині. На перший погляд може здатися, що не має значення, який саме екземпляр елемента мультимножини використовується, або з якої позиції в вихідній послідовності він береться. Насправді, це може не мати значення, коли колекція використовується тільки в одному відношення. Якщо ж деяка колекція бере участь у кількох відношеннях, то вміння розрізняти окремі екземпляри одного і того ж елементу має принципове значення. Наприклад, нехай є мультимножина $\{a,a,b,c\}$, на якій задані відношення $\{((b,1),(a,1)),((a,1)(c,1))\}$ та $\{((b,1),(a,1)),((a,2)(c,1))\}$. В транзитивному замиканні першого відношення з'явиться новий елемент $((b,1),(c,1))$, в той час як в транзитивному замиканні другого відношення цей елемент буде відсутній, див. рис. 4.1.

Рис. 4.1. Приклад відношень на мультимножинах

Перейдемо до зв'язків, під якими будемо розуміти відношення з накладеними на них обмеженнями спеціального виду. Зв'язок виду $1-1$ (один до одного) є бінарним відношенням на колекціях A та B, для якого кожний елемент колекції A находиться у відношенні рівно з одним елементом колекції B та навпаки. Таке відношення задає бієкцію між A та B.

Зв'язок виду $1-m$ (один до багатьох) є бінарним відношенням на колекціях A та B, для якого кожний елемент колекції A знаходиться у відношенні з одним та більше елементами колекції B (але кількість таких елементів скінченна), а кожний елемент колекції B знаходиться у відношенні рівно з одним елементом колекції A[17]. Такий зв'язок задає сюр'єктивну функцію, яка відображає B на A.

Зв'язок виду $1-0..m$ є бінарним відношенням на колекціях A та B, для якого кожний елемент колекції A знаходиться у відношенні з довільною (скінченною) кількістю елементів колекції B, в тому числі нульовою, а кожний елемент колекції B знаходиться у відношенні рівно з одним елементом колекції A. Такий зв'язок задає функцію, яка відображає B в A та, взагалі кажучи, не є сюр'єкцією.

Зв'язок виду $0..1-0..m$ є бінарним відношенням на колекціях A та B, для якого кожний елемент колекції A знаходиться у відношенні з довільною (скінченною) кількістю елементів колекції B, в тому числі нульовою, а кожний елемент колекції B знаходиться у відношенні не більш ніж з одним елементом колекції A. Такий зв'язок задає часткову функцію, яка відображає B в A.

Поряд з бінарними зв'язками, будуть використовуватись і n-арні зв'язки виду один до багатьох, $n=3,4,...$. Під n-арним зв'язком виду один до багатьох, будемо розуміти множину $n-1$ бінарних відношень виду один до багатьох $\{C_0R_1C_1, C_0R_2C_2, ..., C_0R_{n-1}C_{n-1}\}$, в яких колекція C_0 однакова для всіх

[17] Точніше говорячи, встановлюється зв'язок виду $1-m$ на коллекціях A та B в напрямку від A до B.

зв'язків[18]. Зауважимо, що в цьому розділі R, R_1, R_2 позначають бінарні відношення, а не схеми таблиць, як в попередніх.

Насправді існує набагато більше видів зв'язків (дивись, наприклад, [72]). Тут розглянуті лише ті види, які будуть використовуватись для характеристики моделей даних. Поняття зв'язку дуже близько до поняття відношення, складається враження, що використання двох різних термінів зумовлене скоріше історичними причинами ніж об'єктивною необхідністю. В подальшому ці терміни іноді будуть використовуватися як синоніми.

Для характеристики моделей даних потрібно буде розглядати не окремі бінарні відношення, а сукупності таких відношень, які утворюють так звані дерева відношень (або реляційні дерева). Поняття реляційного дерева введемо індуктивним чином.

Бінарне відношення виду один до одного або один до багатьох C_1RC_2 є реляційним деревом з коренем C_1RC_2 та кореневою колекцією C_1.

*Я*кщо $RT_1, RT_2, ..., RT_n$ – реляційні дерева з спільною кореневою колекцією C_1, а C_0RC_1 – бінарне відношення виду один до одного або один до багатьох і колекція C_0 не входить до будь-якого дерева $RT_1, RT_2, ..., RT_n$, то $(C_0RC_1, (RT_1, RT_2, ..., RT_n))$ є реляційним деревом з коренем C_0RC_1 та кореневою колекцією C_0.

В реляційному дереві елементи колекцій, в свою чергу, утворюють дерева, дуги яких задаються відношеннями реляційного дерева. Будемо вважати, що дуги є орієнтованими, і проходять від лівого елементу пари, яка належить відношенню, до правого. Так, дерево, яке складається з одного відношення виду один до одного, визначає ліс, зображений на рис. 4.2.

---

[18] Моделювання n-арного зв'язку $n-1$ бінарними зв'язками основане на очевидному ін'єктивному відображенні $(a_1, a_2, ..., a_n) \mapsto ((a_1, a_2), (a_1, a_3), ..., (a_1, a_n))$.

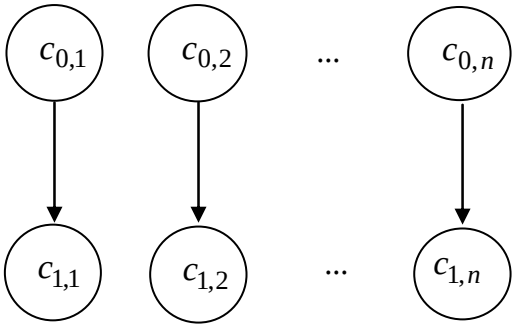

Рис. 4.2. Дерево, яке складається з одного відношення виду один до одного

Дерево, яке складається з одного відношення виду один до багатьох, визначає ліс, зображений на рис. 4.3.

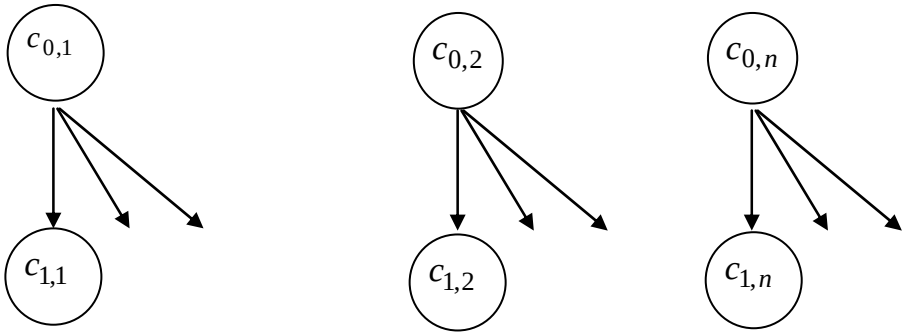

Рис. 4.3. Дерево, яке складається з одного відношення виду один до багатьох

Розглянемо тепер випадок бінарного відношення C_0RC_0 виду один до багатьох, заданого на одній і той же самій колекції C_0. Якщо граф, який відповідає цьому відношенню, є ациклічним і кожна вершина має не більше ніж одну вхідну дугу , то він утворює ліс на елементах колекції C_0. Якщо ж існує рівно одна вершина, яка не має вхідних дуг, то це буде дерево.

Відповідний ліс або дерево може бути побудовані ітеративним чином. Спочатку шукається множина кореневих вершин. Це будуть вершини, які не мають вхідних дуг. Так як граф є ациклічним, то існує принаймні одна така вершина. Для кожної вершини будується відповідне дерево ітеративним чином. На першому кроці знайдемо всіх синів кореневої вершини. Далі повторюємо процедуру для кожного із синів. Так як граф ациклічний, то процедура

закінчиться через скінченну кількість кроків. В результаті отримаємо дерево для цієї вершини.

4.2 Моделі даних першого рівня (табличні моделі)

Розглянемо моделі, які взагалі не містять відношень. Такі моделі назвемо моделями першого рівня. Вони складаються тільки з колекцій однотипних рядків та операцій на колекціях. Нехай рядки визначені тільки на атомарних даних, тобто значеннями атрибутів виступають тільки атомарні дані.

Якщо в якості колекції виступає множина, то отримаємо реляції (таблиці) з реляційної алгебри Кодда, або з табличної алгебри [67, 90]. В якості операцій – операції реляційної алгебри або операції табличної алгебри.

Якщо в якості колекції виступає мультимножина, то отримаємо таблиці, які використовуються в реляційних базах даних. На відміну від таблиць реляційної алгебри вони можуть мати дублікати рядків. В якості операцій використовуються операції SQL-алгебри, які були описані в попередніх розділах.

Якщо в якості колекції виступає послідовність, то така модель задає курсори реляційних баз даних. В якості операцій виступають навігаційні операції: взяти перший рядок, взяти останній рядок, перейти до наступного/попереднього рядка, перевірити чи рядок є останнім/першим, змінити рядок, вилучити рядок та інші.

Відсутність зв’язків не свідчить про те, що такі моделі є найслабкішими та найменш виразними. Насправді зв’язки в таких моделях присутні, але вони задаються в неявному вигляді за допомогою додаткових атрибутів у рядках. Таким чином, такі моделі є однорідними, і для маніпулювання зі зв’язками використовуються такі ж самі операції, як і для маніпулювання з іншими атрибутами рядків. Така однорідність моделі обумовлює її високу гнучкість і на сьогодні табличні моделі є найбільш поширеним типом комерційних баз даних.

Зворотною стороною медалі є порівняно невисока продуктивність таких баз даних, що пред’являє високі вимоги до оптимізації фізичної структури бази

та оптимізації запитів. Крім того, така однорідність приховує інформацію про структуру предметної області і буває дуже важко розібратися, в яких атрибутах зберігаються власне дані, а в яких – інформація про зв'язки між ними.

4.3 Моделі даних другого рівня (багаторівневі моделі)

Перейдемо до моделей, в яких присутні зв'язки виду один до одного. Рядки, як і раніше, містять тільки атомарні дані. Така модель має одну досить цікаву інтерпретацію, яку не вдалося знайти в доступній літературі. А саме, нехай потрібно працювати з рядками, що мають багаторівневі заголовки, наприклад, такими, які зображені в табл. 4.1 і які містять інформацію про людину.

Таблиця 4.1 – Приклад багаторівневої таблиці

Ідентифікатор	**ПІБ**			**Адреса**				**Рік народження**
	Прізвище	**Ім'я**	**По батькові**	**Місто**	**Вулиця**	**Дім**	**Кв.**	

Така складна таблиця, очевидно, може бути відображена у звичайну, однорівневу, таблицю. Але при цьому буде втрачена інформація про логічні групи атрибутів, що ускладнює розуміння моделі бази даних і може привести до помилок при маніпулюванні з даними. В той же час таблиця може бути розділена на три окремі частини, між якими встановлюється зв'язок один до одного, як зображено на рис. 4.4. Такі багаторівневі таблиці складаються з сукупності багаторівневих рядків, кожний з яких, в свою чергу, складається з одного рядка першого рівня (кореневого рядка) та довільної кількості рядків другого рівня. В свою чергу, рядки другого рівня можуть містити в собі рядки третього рівня і так далі. Кожний такий багаторівневий рядок у графічному вигляді зображається як дерево, вершини якого є рядками першого, другого та

ін. рівнів, а іменовані дуги задають зв’язки між рядками. Сукупність багаторівневих рядків (багаторівнева таблиця) утворює ліс.

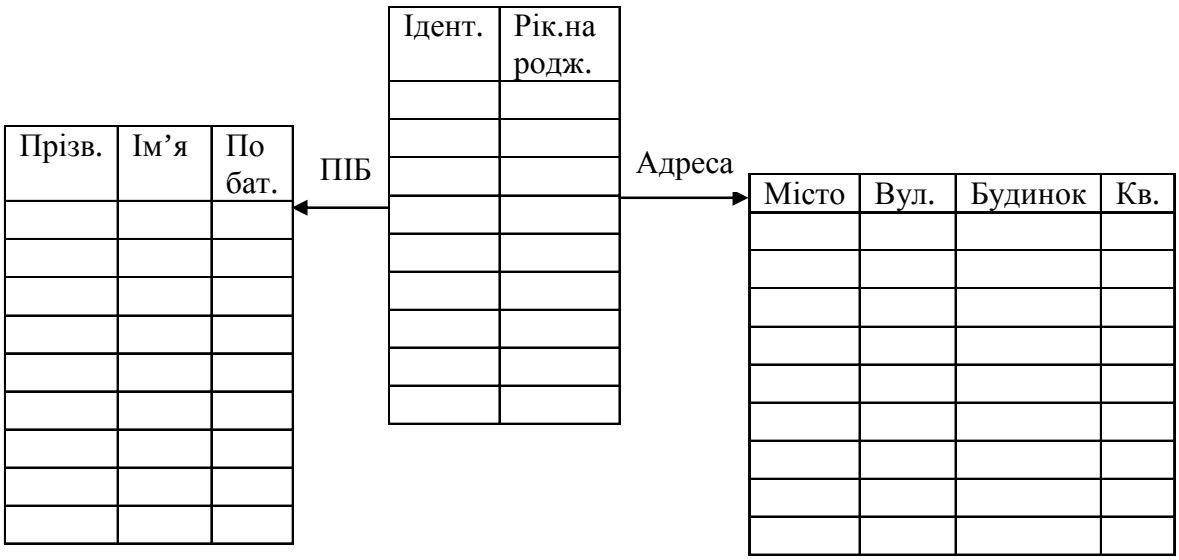

Рис. 4.4. Приклад зв’язку типу один до одного 1–1

В якості типу колекції може виступати множина або мультимножина. Аналогічно моделям першого рівня послідовність таких рядків може розглядатись як аналог курсору для випадку багаторівневих таблиць.

Множина операції для багаторівневих таблиць на сьогодні відсутня.

Перейдемо до іншої інтерпретації моделей даних другого рівня, а саме до випадку зв’язків виду $1-0..1$. Такими моделями зручно відображати записи з варіантами, які використовуються в багатьох сучасних мовах програмування. В них запис складається з фіксованої частини, яка присутня завжди, та варіантної части, яка вибирається з одного з заданих видів. Наприклад, для транспортного засобу, в залежності від того вантажний він чи легковий, знадобляться додаткові атрибути, які описують кількість пасажирських місць для легкового транспорту або вантажопідйомність для вантажного. Модель бази даних для такої таблиці з варіантами приведена на рис. 4.5.

Моделі такого виду в літературі не зустрічаються і на сьогодні вивчені дуже мало. Так, для них відсутня множина допустимих операцій, не говорячи вже про підтримку таких моделей на рівні СУБД. В той же час такі моделі є

досить поширеними і сумісними з реляційними базами знизу до гори, тобто реляційні моделі вкладаються в багаторівневі моделі.

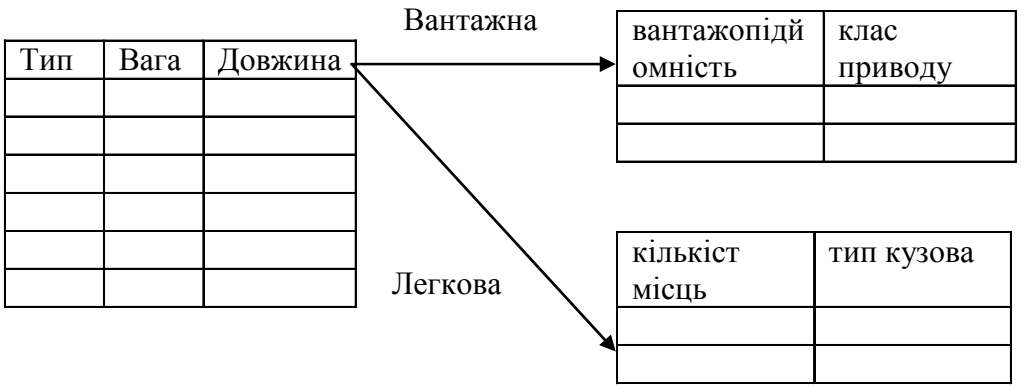

Рис. 4.5. Приклад зв’язку типу 1–0..1

4.4. Моделі даних третього рівня (ієрархічні моделі даних)

Перейдемо до більш відомого типу моделей даних – до ієрархічних моделей. Історично цей тип баз даних з’явився першим. В 1968 р. була закінчена розробка ієрархічної СУБД IMS, яку компанія IBM розробляла в рамках космічної програми "Аполлон" для того, щоб зберігати список деталей ракети Saturn V (так званий BOM – Bill Of Materials) [102]. І хоча СУБД з’явилася дуже пізно для того, щоб значним чином бути використаною в програмі "Аполлон", сама система стала дуже популярною і використовується навіть в наш час (хоча вона і має значно менш поширення ніж реляційні СУБД).

В ієрархічній базі даних в якості елементів колекцій використовуються рядки з атомарними значеннями, а зв’язки мають вид один до багатьох. Причому зв’язки мають утворювати реляційне дерево (див. підрозділ 4.1).

Якщо в якості типу колекції береться послідовність, то отримуємо структуру даних, яка використовується в СУБД IMS. Для маніпулювання даними використовується мова DL/I (Data Language Interface). Операції цієї мови складаються з навігаційних операцій для переміщення по вузлам дерева та операцій модифікації дерева. До навігаційних відносяться операції:

- GET UNIQUE (вибрати унікальний) для пошуку рядку в послідовності за деяким критерієм;

- GET NEXT (вибрати наступний) для доступу до наступного рядка;

- GET NEXT WITHIN PARENT (вибрати наступний синовній) для переходу до першого рядку синовньої колекції;

- GET HOLD UNIQUE (вибрати унікальний і зберегти) для пошуку рядку в послідовності за деяким критерієм та зберігання для наступного вилучення або модифікації;

- GET HOLD NEXT (вибрати наступний і зберегти) для доступу до наступного рядка та зберігання для наступного вилучення або модифікації;

- GET HOLD NEXT WITHIN PARENT (вибрати наступний синовній і зберегти) для переходу до першого рядку синовньої колекції та зберігання для наступного вилучення або модифікації;

Для зміни даних після виконання останніх трьох операцій, використовуються операції:

- REPLACE для оновлення рядка;

- DELETE для вилучення рядка;

- INSERT для додавання рядка.

Недоліком навігаційних моделей є наявність порядку на рядках. Як правило, такий порядок не є необхідним для збереження інформації, а при підготовці звітів може бути потрібно упорядковувати рядки різним чином в залежності від типу звіту. Навігаційні операції є більш низького рівня ніж теоретико-множинні операції, які використовуються в моделях даних на основі множин та мультимножин.

Прикладом моделі даних на основі множин рядків є квазіреляції [3]. На неформальному рівні ієрархічна структура може бути представлена як таблиця, в яку вкладаються інші таблиці. Більш точно така структура задається через ієрархічні таблиці, яки визначаються рекурсивним чином як пара (стан, схема). Спочатку задамо ієрархічну схему.

1) Множина атрибутів $\{A_1, A_2, ..., A_n\}$ є ієрархічною схемою s листового типу.

2) Множина виду $\{h_1, h_2, ..., h_n\}$, де h_i є атрибутом, який будемо називати листовим, або парою (A_i, s_i), A_i – атрибут, а s_i – ієрархічна схема, є ієрархічною схемою.

Ієрархічним рядком назвемо рядок, значеннями будь-якого атрибуту в якому є або атомарне дане, або множина ієрархічних рядків.

Будемо говорити, що ієрархічний рядок відповідає ієрархічній схемі $\{h_1, h_2, ..., h_n\}$, якщо множина атрибутів рядка співпадає з множиною атрибутів схеми, а значеннями атрибутів в ньому виступають або атомарні дані для листових атрибутів, або множини (можливо порожні) рядків для нелистових атрибутів, які відповідають підсхемі даного атрибуту. Порожня множина відповідає будь-якій схемі.

Станом ієрархічної таблиці схеми s назвемо множину ієрархічних рядків, які відповідають даній схемі.

Аналогічним чином можуть бути визначені ієрархічні таблиці на мультимножинах.

На сьогодні не існує досить потужного набору операцій для ієрархічних баз даних, аналогічного реляційній алгебрі. Відповідно відсутні комерційні реалізації ієрархічних моделей на базі колекцій типу множина або мультимножина. Таким чином, потенціал цієї моделі даних ще не розкрито.

Іншим різновидом ієрархічних моделей є модель з бінарним відношенням один до багатьох, яке визначене на одній і той самій колекції і задає ліс на елементах цього відношення. Така модель даних згадується у Мартіна [42]. Розглянемо її для випадку, коли типом колекції є множина рядків з атомарними значеннями. Такі структури даних будемо називати рекурсивними таблицями. Кожний рядок такої таблиці містить множину звичайних атрибутів, значеннями яких є атомарні дані, а також один спеціальний атрибут, ім’я якого співпадає з ім’ям заданого зв’язку. Значенням атрибуту виступає таблиця з такою ж самою схемою, і так далі. Глибина вкладеності необмежена. На рис. 4.6 наведений

приклад такої таблиці для бази деталей, які в свою чергу складаються з інших деталей і так далі. Зв’язок має назву "Складається з".

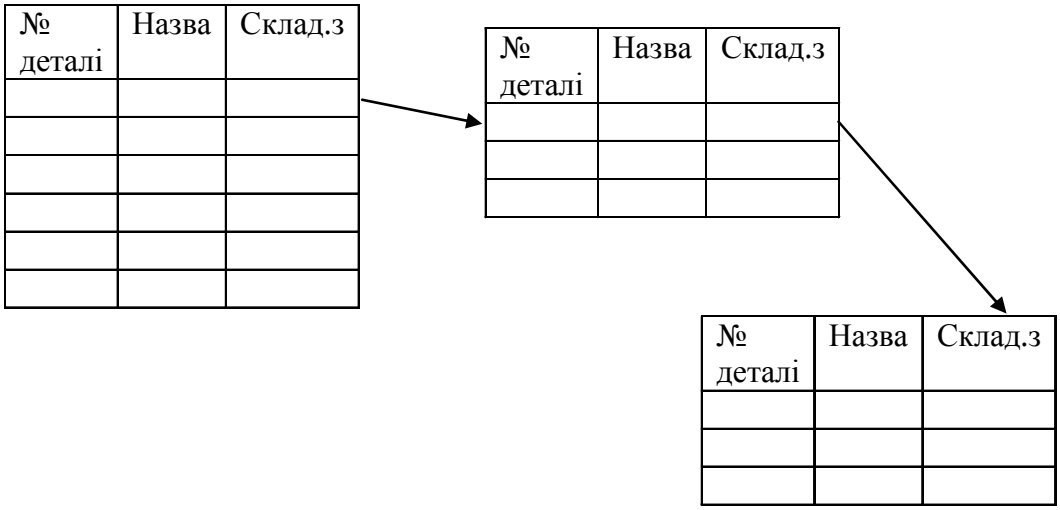

Рис. 4.6. Приклад рекурсивної таблиці

Схемою рекурсивної таблиці назвемо пару $(\{A_1, A_2, ..., A_n\}, A_{n+1})$, де $\{A_1, A_2, ..., A_n\}$ – множина атрибутів, A_{n+1} – спеціальний, виділений, атрибут, який будемо називати табличним атрибутом.

Рядок r відповідає схемі $s = (\{A_1, A_2, ..., A_n\}, A_{n+1})$, якщо множина атрибутів рядка співпадає з $\{A_1, A_2, ..., A_n, A_{n+1}\}$, значеннями атрибутів $\{A_1, A_2, ..., A_n\}$ є атомарні дані, а значенням табличного атрибуту A_{n+1} є порожня множина або множина рядків, які в свою чергу відповідають схемі s.

Рекурсивною таблицею назвемо пару (t, s), де s – схема рекурсивної таблиці, а t – множина рядків, які відповідають цієї схемі.

Для послідовностей та мультимножин рекурсивні таблиці визначаються аналогічним чином.

Множина операцій для рекурсивних таблиць на сьогодні не визначена, також як і сама модель даних дуже рідко зустрічається в літературі. Дивись, наприклад, вже згадану роботу Мартіна [42].

Слід відмітити, що хоча рекурсивні таблиці та ієрархічні таблиці не існують як самостійні об’єкти у відповідних СУБД, вони можуть бути

промодельовані у реляційних базах даних. Для роботи з ними використовуються звичайні оператори мови SQL, в тому числі рекурсивна форма оператора SELECT для роботи з рекурсивними таблицями, які в такому випадку моделюються списками суміжності. Але при такому моделюванні втрачається інформація про логічну структуру предметної області. До того ж при відображенні ієрархічних таблиць в реляційні виконується декомпозиція ієрархічної таблиці на кілька стандартних таблиць, при запитах необхідно, навпаки, виконати зворотну операцію з'єднання, яка є одною з найбільш ресурсоміських операцій в мові. Таким чином, розробка самостійних ієрархічних моделей, в першу чергу адекватних операцій високого рівня на ієрархічних структурах даних, виглядає перспективним напрямком розвитку СУБД.

4.5. Моделі даних четвертого рівня (мережні моделі даних)

Мережна модель даних з'явилась майже в той же самий період, що і ієрархічна модель даних. В жовтні 1969 р. комітет DBTS (Data Base Task Group) більш відомий під своєю первісною назвою CODASYL (Conference on Data Systems Languages) опублікував специфікацію мови для мережної моделі даних. В цієї моделі в якості колекцій виступають послідовності рядків, значеннями атрибутів яких можуть бути як атомарні дані, так і більш складні типи даних, наприклад, аналоги масивів. Між колекціями задається довільна кількість n-арних зв'язків виду один до багатьох, точніше кажучи, зв'язків виду $1-0..M$. Для кожного зв'язку існує один тип колекції, який називається власником, і один або більше типів колекцій, які називаються членами зв'язку. Особливим випадком є зв'язок з відсутнім власником. Вважається, що власником такого типу зв'язку є віртуальна колекція, яка називається "Система". Такий зв'язок по суті дозволяє задавати колекції, на яких не задано жодного зв'язку.

Зв'язки можуть задаватися між довільними колекціями, за винятком зв'язків на одній і тій самій колекції, тобто петлі не є допустимими. В той же час інші види циклів можуть задаватися. Зв'язок, разом з типами колекцій, на

яких він заданий, називається набором (set). Зв’язок без власника називається сингулярним набором.

До основних операцій, які виконуються над даними, відносяться наступні [82].

- Оператор READY. Перший оператор в програмі, його функції аналогічні оператору OPEN при роботі з файлом.

- Оператор FINISH. Останній оператор в програмі, його функції аналогічні оператору CLOSE при роботі з файлом.

- Оператор FIND. Основний оператор маніпулювання даними. Використовується для пошуку даних. Має кілька видів.

- Оператор STORE. Додає новий рядок в базу.

- Оператор GET. Використовується для переміщення знайденого рядку з бази в прикладну програму.

- Оператор ERASE. Використовується для вилучення рядка з бази. Попередньо рядок повинен бути знайдений оператором FIND.

- Оператор MODIFY. Змінює рядок в базі на новий рядок, який знаходиться в оперативній пам’яті. Як правило, перед цим виконується послідовність операторів FIND, GET, MOVE. Оператор MOVE змінює атрибути рядка в оперативній пам’яті.

- Оператори CONNECT та DISCONNECT. Використовуються для включення або виключення рядку до деякого зв’язку. Рядок може також включатися до зв’язку при виконанні оператора STORE.

- Оператор ORDER. Дозволяє задати порядок на рядках деякого набору. Порядок може бути тимчасовим (тільки на час виконання програми) або постійним, тобто виконується сортування рядків.

- Оператор ACCEPT. Використовується для читання індикаторів поточного стану.

- Оператор IF. За його допомогою виконуються перевірка над останнім знайденим рядком, чи входить він до деякого набору.

- Оператор USE. Задає реакцію програми на виняткові ситуації, які виникають під час роботи з базою.

- Оператори KEEP, FREE, REMONITOR. Ці оператори пов’язані з вирішенням проблем, які виникають при одночасному доступі програм до одного і того ж рядка, і є аналогами блокування записів в файлах.

Модель даних CODASYL була втілена в багатьох СУБД. Найбільш відомі це Honeywell's Integrated Data Store (IDS/2), Cullinet's Integrated Database Management System IDMS, Univac's DMS-1100, Digital Equipment Corporation's DBMS32 та Cullinane Database Systems. Компанія Cullinet продана компанії Computer Associates, яка до 2007 р. підтримувала цей продукт.

Неважко перевизначити структуру даних типу набора для множин і мультимножин. Але проблема, як і для ієрархічних моделей даних, полягає в заданні операцій, які б дозволяли виконувати нетривіальні запити на таких мережних базах.

4.6. Класифікація моделей даних

Наприкінці розділу наведемо таблицю, яка містить перелік розглянутих моделей даних і їх характеристики.

Таблиця 4.2 – Класифікаційна таблиця моделей даних

Назва	**Тип колекції**	**Тип зв’язку**	**Множина операцій**
Табличні моделі			
Реляційна	множина	відсутній	реляційна алгебра
комерційні СУД	мультимножина	відсутній	SQL алгебра
Курсори	послідовність	відсутній	навігація по курсору
Багаторівневі таблиці			
Багаторівневі таблиці	послідовність	1–1	
Багаторівневі	множина	1–1	

таблиці			
Багаторівневі таблиці	мультимножина	1–1	
Ієрархічні моделі			
Ієрархічна	послідовність	1–0..M, утворює реляційне дерево	операції мови DL/1 СУБД IMS
Ієрархічні таблиці	множина	1–0..M, утворює реляційне дерево	
Ієрархічні таблиці	мультимножина	1–0..M, утворює реляційне дерево	
Рекурсивні таблиці	послідовність	1–0..M на одній і той самій колекції	
Рекурсивні таблиці	множина	1–0..M на одній і той самій колекції	
Рекурсивні таблиці	мультимножина	1–0..M на одній і той самій колекції	
Мережеві моделі			
КОДАСИЛ	послідовність	1–0..M	навігаційні операції КОДАСИЛ
	множина	1–0..M	
	мультимножина	1–0..M	

Основні результати розділу 4 наступні.

1. Надане визначення моделі даних, яке містить найбільш суттєві риси існуючих визначень. Показано, що визначення зв'язку між таблицями на множинах не може бути безпосередньо застосовано до мультимножин та послідовностей без втрати суттєвої інформації. Надане визначення зв’язку на випадок мультимножин та послідовностей, яке зберігає інформацію про те, які саме екземпляри входять в зв’язок.

2. Побудовані реляційні дерева, які є композицією бінарних зв’язків в деревоподібні структури та використовуються для задання ієрархічних та рекурсивних моделей даних.

3. Розглянуті моделі першого рівня (табличні), до яких відносяться таблиці на основі множин, мультимножин та послідовностей.

4. Розглянуті нові моделі другого рівня (багаторівневі таблиці, таблиці на рядках з варіантами).

5. Розглянуті моделі третього рівня (ієрархічні моделі), до яких відносяться ієрархічна модель СУБД IMS, квазіреляції та нова модель (рекурсивні таблиці), опис якої не вдалося знайти в доступній літературі. Ієрархічні структури даних будуються на основі реляційних дерев.

6. Розглянуті моделі четвертого рівня (мережні моделі), до яких відносяться мережна модель даних, специфікація якої була опублікована комітетом CODASYL, та моделі, типами колекцій в яких виступають множини та мультимножини.

ВИСНОВКИ

Головним результатом роботи є побудова семантичної моделі (програмної алгебри) композиційного типу, яка задає множину семантичних функцій, інваріантних відносно різних діалектів мови SQL. Ця модель розв’язує важливу задачу розробки адекватної семантики операторів маніпулювання даними мови SQL та уніфікації роботи з різними діалектами мови, що має істотне значення для теорії та практики побудови сучасних СУБД.

Основні результати наступні.

Побудована нова семантична модель таблиць, завдяки чому функції та оператори семантичної моделі мають таку ж семантику для випадку порожніх таблиць, як і в SQL. Для нової семантики таблиць визначені і математично обґрунтовані композиції фільтрації, взяття повного образу та функції групування, а також операції внутрішнього та зовнішнього з’єднання.

Для агрегатних функцій побудована нова композиція агрегування та множина базових функцій, семантика роботи яких з NULL значеннями, як показало тестування на поширених СУБД, співпадає з наявними реалізаціями мови SQL.

Вперше надана операційна та денотаційна семантика рекурсивних запитів в SQL.

Доведена еквівалентність операційної та денотаційної семантик рекурсивних запитів.

Розроблена класифікаційна схема для моделей даних на базі поняття колекції (включає множини, мультимножини, послідовності), яка дозволила знайти нові моделі даних, такі як рекурсивні таблиці, багаторівневі таблиці, таблиці на рядках з варіантами. Вирішена задача визначення n-арних відношень на мультимножинах та послідовностях. Показано що структури даних всіх розглянутих моделей даних можуть бути трансльовані в реляційну модель.

Побудована у дисертації семантична модель (програмна алгебра) враховує такі суттєві особливості реальних СУБД як наявність дублікатів рядків таблиць, упорядкування та групування рядків, зовнішні з‘єднання та агрегатні функції, які не моделюються засобами реляційних та табличних алгебр і потребують застосування більш виразних моделей.

З результатів роботи випливають наступні висновки.

Агрегатні функції та предикати SQL-подібних мов задаються за однією логічною схемою.

Ординарні багатомісні функції по-різному розширюються на рядки та стовпчики таблиць (в першому випадку вони зберігають *Null* значення, в другому – ні), при з‘єднанні таблиць невизначені значення інтерпретуються як різні, при групування рядків – як рівні.

Збагачення композицією рекурсії програмної алгебри, орієнтованої на визначення семантики операторів маніпулювання даними SQL-подібних мов, дає змогу задати семантику CTE виразів в рекурсивній формі.

Спираючись на поняття колекцій рядків та зв’язків між рядками, побудована класифікаційна схема, в яку вкладаються широко відомі моделі даних, такі, як реляційна, ієрархічна, квазіреляційна та мережна. Крім того, ця схема надає опис нових моделей даних, наприклад, рекурсивних таблиць та таблиць з варіантними рядками.

Для класифікації моделей даних недостатньо використовувати тільки бінарні зв’язки між колекціями, а необхідно розглядати композиції об’єднання бінарних зв’язків у більш складні структури, які, по суті, є n-арними зв’язками спеціального вигляду.

Розширення реляційних даних новими структурами, виявленими в процесі аналізу та класифікації моделей даних, здатне значно підняти рівень абстракції даних, які задаються в сучасних базах даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Андон Ф., Резниченко В. Язык запросов SQL. Учебный курс / Ф. Андон, В. Резниченко. – СПб.: Питер, Киев: Издательская группа BHV, 2006. – 416 с.

2. Барендрегт Х. Ламбда-исчисление. Его синтаксис и семантика / Х. Барендрегт. – Москва: Мир, 1985. – 606 с.

3. Басараб І.А. Композиційні бази даних / І. А. Басараб, М. С. Нікітченко, В. Н. Редько. – К.:Либідь, 1992. – 192 с.

4. Басараб И. А. Композиционный подход к построению языков манипулирования данными / И. А. Басараб, Б. В. Губский // Программирование. – 1988. – № 6. – С. 59-70.

5. Беренсон Х. Критика уровней изолированности в стандарте ANSI SQL / Х. Беренсон, Ф. Бернштейн, Д. Грэй и др. // СУБД. – 1996. – № 2. – С. 45-60.

6. Богатырёва Ю.А. Мультимножества: библиография, решетка мультимножеств / Ю.А. Богатырёва // Theoretical and Applied Aspects of Program Systems Development: international conference, December 8-10, 2009. – Kyiv. – 2009. – С. 13-20.

7. Боуман Дж. С. Практическое руководство по SQL / Дж. С. Боуман, С. Л. Эмерсон, М.Дарновски. – Киев: Диалектика, 1997. – 320 с.

8. 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.

9. 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.

10. 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.

11. 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.

12. Буй Д.Б. 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.

13. Буй Д.Б. Композиційні структури SQL-подібних мов: фільтрація та повний образ / Д. Б Буй, С. А. Поляков // Вісник Львівського університету. Сер. механіко-математична. – 1998. – Вип. 50 "Задачі та методи прикладної математики". – С. 24-26.

14. Буй Д. Б. Композиційна семантика SQL-подібних мов: мультимножини, рядки, впорядковані таблиці / Д. Б Буй, С. А. Поляков // Вісник Київського університету. Сер. фіз.-мат. науки. – 1999. – Вип. 2. – С. 183-194.

15. Буй Д. Б. Композиційна семантика SQL-подібних мов: табличні структури даних, композиції, приклади / Д. Б Буй, С. А. Поляков // Вісник Київського університету. Сер. фіз.-мат. науки. – 1999. – Вип. 1. – С. 130-140.

16. Буй Д. Б. Неподвижные точки и операторы замыкания: программологические аспекты / Д. Б. Буй, В. Н. Редько // Кибернетика. – 1995. – № 1. – С. 113-121.

17. Буй Д. Б. Про різні способи задання іменних даних / Д. Б. Буй, Ю. Й. Брона // Вісник Київського університету. Сер. фіз.-мат. науки. – 1994. – С. 186-194.

18. Буй Д. Б. Теоретико-множинні конструкції в теорії реляційних баз даних / Д. Б. Буй, Ю. Й. Брона // Вісник Київського університету. Сер. фіз.-мат. науки. – 1996. – Вип. 1. – С. 216-224.

19. 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.

20. Буй Д. Б. Три замечания о трехзначной логике Клини / Д. Б. Буй, С. А. Поляков, Е. В. Шишацкая // The Fourth International Conference “Theoretical and Applied Aspects of Program Systems Development (TAAPSD’2007). Abstracts (Ukraine, Berdynsk, 4-9 September, 2007). – Київ: Пульсари, 2007. – С. 47-51.

21. Буй Д.Б. Решітка мультимножин / Д.Б. Буй, Ю.О. Богатирьова // Современные направления теоретических и прикладных исследований: международная конференция SWORD, 16-27 марта 2009 г., Одесса: Черноморье. – 2009. – Т.2. – С. 49-52.

22. Бэкус Дж. Алгебра функциональных программ: мышление функционального уровня, линейные уравнения и обобщенные определения / Дж. Бэкус // Математическая логика в программировании: Сб. стат. – Москва: Мир, 1991. – С. 8-53.

23. Горбатов В. А. Основы дискретной математики / В. А. Горбатов. – Москва: Высшая школа, 1986. – 311 с.

24. Горчинская О. Ю. Теоретический аспект построения реляционных моделей баз данных / О. Ю. Горчинская // Автоматика и телемеханика. – 1983. – № 1. – С. 5-25.

25. Грабер М. Справочное руководство по SQL / М. Грабер. – Москва: ЛОРИ, 1997. – 291 с.

26. Грей П. Логика, алгебра и базы данных / П. Грей. – Москва: Машиностроение, 1989. – 360 с.

27. Дейт К. Руководство по реляционной СУБД DB2 / К. Дейт. – Москва: Финансы и Статистика, 1988. – 320 с.

28. Дейт К. Введение в системы баз данных / К. Дейт. – Киев: Диалектика, 1998. – 781 с.

29. Дейт К. Введение в системы баз данных / К. Дейт. – Москва: Наука, 1980. – 464 с.

30. Джексон Г. Проектирование реляционных баз данных для использования с микроЭВМ / Г. Джексон. – Москва: Мир, 1991. – 252 с.

31. Емеличев В. А. Лекции по теории графов / В. А. Емеличев, О. И. Мельников, В. И. Сарванов, Р. И. Тышкевич. – Москва: Наука, 1990. – 384 с.

32. Калиниченко Л. А. Методы и средства интеграции неоднородных баз данных / Л. А. Калиниченко. – Москва: Наука, 1983. – 424 с.

33. Коннолли Т. Базы данных: проектирование, реализация и сопровождение. Теория и практика / Т. Коннолли, К. Бегг, А. Страчан. – Москва: Издательский дом "Вильямс", 2000. – 1120 с.

34. Кузин Л. Т. Основы кибернетики / Л. Т. Кузин. – Москва: Энергия, 1979. – Т.2: Основы кибернетических моделей. – 584 с.

35. Кузнецов С. Д. Стандарты языка реляционных баз данных SQL: краткий обзор / С. Д. Кузнецов // СУБД. – 1996. – № 2. – С. 6-36.

36. Куратовський К. Топология / К. Куратовський. Т. 1. – Москва: Мир, 1966. – 594 с.

37. Ладани Х. SQL. Энциклопедия пользователя / Х. Ладани. – Киев: ДиаСофт, 1998. – 624 с.

38. Мальцев А. И. Алгебраические системы / А. И. Мальцев. – Москва: Наука, 1964. – 392 с.

39. Мальцев А. И. Алгоритмы и рекурсивные функции / А. И. Мальцев. – Москва: Наука, 1986. – 368 с.

40. Манна З. Теория неподвижной точки программ / З. Манна // Кибернет. сб. Нов. сер. – Москва: Мир, 1978. – Вып. 15. – С. 38-100.

41. Марков А. С. Базы данных. Введение в теорию и методологию / А. С. Марков, К. Ю. Литовский. – Москва: Финансы и статистика, 2006. – 512 с.

42. Мартин Дж. Организация баз данных в вычислительных системах / Дж. Мартин. – Москва: Мир, 1980. – 660 с.

43. Мейер Д. Теория реляционных баз данных / Д. Мейер. – Москва: Мир, 1987. – 608 с.

44. Плоткин Б. И. Универсальная алгебра, алгебраическая логика и базы данных / Б. И. Плоткин. – Москва: Наука, 1981. – 248 с.

45. 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.

46. 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.

47. Поляков С.А. Огляд стандартів мови SQL / С. А. Поляков // Вісник Київського університету. Сер. фіз.-мат. науки. – 2008. – Вип. 1. – С. 132-137.

48. 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.

49. 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.

50. 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.

51. Поляков С. А. Побудова рекурсивних запитів в SQL / С. А. Поляков // International Conference "Theoretical and Applied Aspects of Program Systems Development (TAAPSD’2009)". Abstracts (Ukraine, Chernihiv, Kyiv, September 2009). Volume 2. – Київ: Пульсари, 2009. – С. 204-209.

52. Поляков С.А. Алгебра табличних функцій / С.А. Поляков // Вісник Київського університету. Сер.: фіз.-мат. науки. – 1996. – Вип. 2. – С. 150-155.

53. Поляков С.А. Композиційна семантика SQL-подібних мов / С.А. Поляков // Вісник Київського університету. Сер.: фіз.-мат. науки. – 1997. – Вип. 3. – С. 205-211.

54. Поляков С.А. Композиційні структури SQL-подібних мов: фільтрація та повний образ / С.А. Поляков, Д.Б. Буй // Вісник Львівського університету. Задачі та методи прикладної математики. Серія механіко-математична – 1998. – Вип. 50. – С. 24-27.

55. Редько В. Н. Взаимная непроизводность и выразительная сила операций реляционных алгебр / В. Н. Редько, Ю. И. Брона, Д. Б. Буй // Доповіді НАН України. Математика. Природознавство. Технічні науки. – 1996. – № 11. – С. 84-88.

56. Редько В. Н. Дескриптологические основания программирования / В. Н. Редько // Кибернетика и системный анализ. – 2002. – № 1. – С. 3-19.

57. 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.

58. 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.

59. 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.

60. Редько В. Н. Композиции программ и композиционное программирование / В. Н. Редько // Программирование. – 1978. – № 5. – С. 3-24.

61. Редько В. Н. Композиционная структура программологии / В. Н. Редько // Кибернетика и системный анализ. – 1998. – № 4. – С. 47-66.

62. Редько В. Н. Основания композиционного программирования / В. Н. Редько // Программирование. – 1979. – № 3. – С. 3-13.

63. Редько В. Н. Основания программологии / В. Н. Редько // Кибернетика и системный анализ. – 2000. – № 1. – С. 35-57.

64. Редько В. Н. Програмологія: ретроспективи та перспективи / В. Н. Редько // Вісник. Кібернетика (Київський національний університет імені Тараса Шевченка). – 2000. –Вип. 1. – С. 34-57.

65. 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.

66. 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.

67. Редько В. Н. 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. – 196 с.

68. Редько В. Н. Семантические структуры программ / В. Н. Редько // Программирование. – 1981. – № 1. – С. 3-19.

69. Редько В. Н., Буй Д. Б. , Загорский С. П. Манипуляционный аспект баз данных: композиционный подход / В. Н. Редько, Д. Б. Буй, С. П. Загорский // Кибернетика. – 1989. – № 6. – С. 105-113.

70. Риге Ж. Бинарные отношения, замыкания, соответствия Галуа / Ж. Риге // Киб. сб.: Сб. переводов. – Москва: ИЛ, 1963. – Вып. 7. – С. 129-185.

71. Сибли Э. Теория моделей данных и процессирование позиционных множеств / Э. Сибли, Т. Хардгрейв // Модели данных и системы баз данных: Тр. совмест. советско-американского семинара, Москва, 14-23 ноября 1977 г. – Москва. – 1979. – С. 31-90.

72. Сільвейструк Л.М. Формалізація моделі „сутність-зв’язок”: типи сутностей, типи зв’язків та їх обмеження: дисертація кандидата фізико-математичних наук: 01.05.03 – математичне та програмне забезпечення обчислювальних машин і систем / Сільвейструк Людмила Миколаївна. – Київ, 2009. – 173 с.

73. Скорняков Л. А. Элементы теории структур / Л. А. Скорняков. – Москва: Наука, 1982. – 158 с.

74. Смит Д. М. Абстракции баз данных: агрегация и обобщение / Д. М. Смит, Д. К. Смит // СУБД. – 1996. – № 2. – С. 141-160.

75. Ульман Дж. Основы систем баз данных / Дж. Ульман. – Москва: Финансы и статистика, 1983. – 334 с.

76. Успенский В. А. Лекции о вычислимых функциях / В. А. Успенский. – Москва: Гос. изд-во физ.-мат. лит-ры, 1960. – 492 с.

77. Цаленко М. Ш. Моделирование семантики в базах данных / М. Ш. Цаленко. – Москва: Наука, 1989. – 287 с.

78. Цаленко М. Ш. Реляционные модели баз данных (обзор) / М. Ш. Цаленко // Алгоритмы и организация решений экономических задач. – Москва: Статистика, 1977. – Вып. 9. – С. 18-36; Вып. 10. – С. 16-29.

79. Цаленко М. Ш. Семантические и математические модели баз данных / М. Ш. Цаленко // Итоги науки и техники. Информатика. Т.9. – Москва: ВИНИТИ. – 1985. – 207 с.

80. Шрейдер Ю. А. Алгебра бинарных отношений // Маркус С. Теоретико-множественные модели языков / Ю. А. Шрейдер. – Москва: Наука, 1970. – С. 300-330.

81. Яблонский С. В. Введение в дискретную математику / С. В. Яблонский. – Москва: Наука, 1986. – 384 с.

82. Язык описания данных КОДАСИЛ / Пер. с англ. под ред. М. Р. Когаловского и Г. К. Столяров. – М.:Статистика, 1981. – 183 с.

83. Backus J. Can Programming be Liberated from the von Neumann Style? A Functional Style and its Algebra of Programs / J. Backus // Comm. of the ACM. – 1978. – V. 21, N. 8. – P. 613-641.

84. Beaulieu A. Mastering Oracle SQL [2nd edition] / A. Beaulieu, S. Mishra. – Sebastopol: "O'Relly Media Inc.", 2004. – 492 p.

85. 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.

86. 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.

87. 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.

88. Celko J. Joe Celko's Trees and hierarchies in SQL for smarties / J. Celko. – San Francisco: "Morgan Kaufmann Publishers", 2004. – 238 p.

89. Chamberlin D. SEQUEL: a Structured English Query Language / Donald D. Chamberlin, Raymond F. Boyce // SIGMOD Workshop, Vol 1. – 1974. – P. 249-264.

90. Codd E.F. A relational model of data for large shared data banks / E. F. Codd //Comm. ACM. – V. 13, N 6. – 1970. – P. 377-387. (Есть русский пер.: Е.Ф. Кодд. Реляционная модель данных для больших совместно используемых банков данных //СУБД, 1/95, с. 145-160.)

91. 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.

92. 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.

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

94. 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.

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

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

97. Collection (computing) [електронний ресурс]. - режим доступу: http://en.wikipedia.org/wiki/Collection_(computing)

98. Collection (mathematics) [електронний ресурс]. - режим доступу: http://en.wikipedia.org/wiki/Collection_(mathematics)

99. 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.

100. Hardgrave W. T. The Relational Model. A Reformulation of Some Mathematical Aspects / W. T. Hardgrave. – Report of Departament of Inc. Syst. Management, Univ. of Maryland, October 1997.

101. Melton J. SQL:1999 Understanding Relational Language Components / Jim Melton and Alan Simon. – Academic Press, 2002. – 895 p.

102. 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.

103. Neilesen P. SQL Server 2005 Bible / P. Neilesen. – Indiana: "Wiley Publishing Inc.", – 2007. – 1293 p.

104. 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.

105. 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.