Київський національний університет

імені Тараса Шевченка

Анісімов А.В., Кулябко П.П.

Інформаційні системи та бази даних

Частина 1.

Навчально-методичний посібник

Київ

2017

УДК 681.3

Анісімов А.В., Кулябко П.П. Інформаційні системи та бази даних: Навчальний посібник для студентів факультету комп’ютерних наук та кібернетики. - Київ. – 2017. – 110 с.

Затверджено вченою радою

факультету

комп'ютерних наук та кібернетики,

протокол № 9 від 23 травня 2017 р.

2

Передмова.

Матеріал, викладений у цьому навчальному посібнику, умовно можна розділити на 5 розділів. Перший розділ присвячено основним поняттям та класифікаціям. У другому – розглядаються різні мовні системи реляційного підходу. Третій та четвертий розділи присвячено проблемам логічного проектування баз даних на основі реляційного підходу. П'ятий – це завдання для самостійної роботи. Слід відзначити, що наголос у посібнику робиться на практичні задачі, з більш глибоким висвітленням теоретичних аспектів можна ознайомитись в [1,2,3,4,5,6].

1. Вступ. Основні поняття.

1.1. Поняття інформаційної системи.

При самому загальному підході інформаційну систему (ІС) можна визначити як сукупність організаційних і технічних засобів для збереження та обробки інформації з метою забезпечення інформаційних потреб користувачів (абонентів). Таке визначення може бути задовільним тільки при самій узагальненій і неформальній точці зору і підлягає подальшому уточненню.

ІС здавна знаходять (в тому чи іншому вигляді) досить широке застосування в життєдіяльності людства. Це пов'язано з тим, що для існування цивілізації необхідним є обмін інформацією - передача знань, як між окремими членами і колективами суспільства, так і між різними поколіннями.

Найдавнішими і найпоширенішими ІС слід вважати бібліотеки. І, дійсно, здавна в бібліотеках збирають книжки (або їх аналоги), зберігають їх, дотримуючись певних правил, створюють каталоги різного призначення для полегшення доступу до книжкового фонду. Видаються спеціальні журнали та довідники, що інформують про нові надходження, ведеться облік видачі.

Ще один приклад. На великому сучасному підприємстві в тому чи іншому вигляді повинна існувати інформаційна система. Для забезпечення якісного управління потрібно знати (можливо з різним ступенем оперативності) об'єм запасів тієї чи іншої сировини, скільки і якої вироблено продукції, скільки споживається електроенергії, який цех що виробляє і що потребує, та багато іншої інформації, яка стосується виробничих питань. Крім цього, профспілкам необхідні відомості про потреби співробітників у соціально-побутовій, медично-оздоровчій сферах, тощо. Для обробки всіх таких даних потрібні певні організаційні і технічні засоби, тобто ІС.

До цього моменту мова йшла про інформаційні системи без врахування способу їх реалізації. Найстаріші (у моральному і у фізичному розумінні) системи повністю

базувалися на ручній праці. Пізніше їм на зміну прийшли різні механічні пристрої для обробки даних (наприклад, для сортування, копіювання, асоціативного пошуку, тощо). Наступним кроком стало впровадження автоматизованих інформаційних систем (АІС), тобто систем, де для забезпечення інформаційних потреб користувачів використовується ЕОМ зі своїми носіями інформації. В наш час - епоху інформаційного вибуху - розроблюється і впроваджується велика кількість самих різноманітних АІСів з дуже широким спектром використання.

1.2. Поняття бази даних.

Зауважимо, що в літературних джерелах існує багато описів поняття бази даних, що зовні значно відрізняються один від одного, акцентуючи увагу на тій чи іншій рисі об'єкта опису, але по своїй суті вони достатньо близькі.

База даних (БД) -- це сукупність взаємозв'язаних даних, що зберігаються разом. Основними та невід'ємними властивостями БД є такі:

- для даних допускається така мінімальна надлишковість, яка сприяє їх оптимальному використанню в одному чи кількох застосуваннях;
- незалежність даних від програм;
- для пошуку та модифікації даних використовуються спільні механізми;
- як правило, у складі БД існують засоби для підтримки її цілісності та захисту від неавторизованого доступу

Прокоментуємо додатково підкреслені слова та вирази у вищенаведеному описі, порівнюючи в основному з близьким попередником БД - файловими системами (ФС).

На відміну від файлових систем БД зорієнтована для підтримки даних для кількох застосувань. На практиці ця властивість інколи порушується. Часом таке порушення можна пояснити тим, що проект вводиться в дію поетапно, і у певний момент дійсно функціонує тільки одне застосування. Іноді відхід від вказаної властивості зумовлений іншими важливими причинами, але, на жаль, не є рідкістю просто помилка у виборі засобів для реалізації проекту і ситуація нагадує відоме прислів'я про стрілянину з гармати по горобцям.

Взаємозв'язаність даних полягає в тому, що доступ до певної групи даних якогось застосування загалом полегшує доступ до інших груп даних цього ж застосування. В умовах орієнтації БД на велику кількість застосувань виникає необхідність у підтримці значного числа різноманітних зв'язків між даними. Саме у розумінні тісного логічного

4

зв'язку використані слова про збереження разом даних. Розглянемо приклад, який ілюструє вказану властивість у порівнянні з простими ФС.

Приклад про мішок з різнокольоровими кульками.

Нехай у мішку є багато різнокольорових кульок, і завдання полягає в тому, щоб знайти кульку певного кольору, витягаючи їх по одній з мішка. Якщо операції витягання кульки з мішка повністю незалежні між собою, то цей приклад є аналогом технології роботи у рамках простих ФС з послідовним доступом. Якщо ж припустити, що кульки зв'язані між собою різнокольоровими мотузками, тоді пошук кульки можна пришвидчити, використавши на черговому кроці мотузку потрібного кольору. Цей варіант прикладу є аналогом технології роботи з взаємозв'язаними (мотузками) даними, що характерно для функціонування БД.

Вимога мінімізації надлишковості полягає у мінімальній кількості копій для одних і тих же даних з урахуванням орієнтації на кілька застосувань. Ці надлишкові копії використовуються для підтримки зв'язків між даними. Як приклад, розглянемо відомості, що зберігаються у відділі кадрів деякого підприємства про своїх співробітників. Користувачами цієї інформації виступають адміністрація, профспілкова організація та бухгалтерія підприємства. Адміністрацію цікавлять дані про кваліфікацію, професійний рівень і досвід роботи, профспілки використовують відомості соціально-побутового характеру, а бухгалтерія оброблює ті дані, що потрібні для нарахувань заробітної плати та підрахунку податків, інших нарахувань та відрахувань. Хоча інформація і різнорідна, але все ж має значну спільну частину. Всім користувачам потрібні службовий номер, прізвище, ім'я, по-батькові співробітника, його рік народження, дані про умови праці. Інформація про сімейний стан та склад сім'ї використовується бухгалтерією і профспілками. Якщо для зберігання даних застосувати технологію ФС, то можливі два крайні варіанти: а)незалежні один від одного файли, відсортовані згідно з потребами того чи іншого користувача, передбачають значну надлишковість даних; б)всі дані знаходяться у одному файлі, відсортованому так, як потрібно одному з користувачів (наприклад, адміністрації) - надлишковість при цьому практично відсутня, але зручно працювати тільки одному з користувачів. Концепція БД займає проміжне становище між вищеописаними крайніми позиціями.

Зайва надлишковість має кілька недоліків. По-перше, зберігання кількох копій веде до додаткових витрат пам'яті. По-друге, доводиться виконувати численні операції оновлення для кількох надлишкових копій. Крім того, оскільки різні копії даних можуть відповідати різним стадіям оновлення, то інформація, що зберігається в системі на певний час може стати суперечливою.

5

Про незалежність даних часто говорять, як про одну з основних властивостей БД. Під цим поняттям розуміється можливість зміни структури даних без зміни програм, що її використовують, а також рівень самоінтерпретованості даних. Міра незалежності даних тісно пов'язана з ступенем необхідної деталізації відомостей про організацію їх зберігання. Проілюструємо цю ситуацію дещо абстрагованим прикладом. Припустимо, що ви збираєтесь переглянути фільм у кінотеатрі, а для того, щоб прибути на місце плануєте скористатись послугами таксі. Поінформованість та досвід водія таксі відповідають мірі незалежності. У одному випадку Вам достатньо вказати лише назву фільму, а все інше зробить водій. У іншому випадку Вам потрібно буде визначити назву кінотеатру. Наступне зниження рівня - це адреса кінотеатру, а ще далі - вказівки по дорозі типу "їхати прямо, звернути наліво, а через 500 метрів - направо і т.п.". Аналогічно і користувачу при підвищенні ступеня незалежності даних треба менше задавати (і знати) "процедурної" інформації щодо доступу до даних. Зауважимо, що певний (хоч і досить низький) рівень незалежності мають сучасні ФС: при доступі до файлу достатньо вказати його ім'я, а інформація про треки та сектори непотрібна, але зміна розміру запису вимагає корекції всіх програм, що звертались до цього файлу.

Під цілісністю БД розуміють несуперечливість між собою даних, що в ній зберігаються. Наприклад, для кадрових відомостей рік народження співробітника не може бути більшим року призначення на посаду або поточного року. Щоб запобігти виникненню таких ситуацій при модифікації і поповненнях БД, співвідношення між даними контролюються спеціальними засобами підтримки цілісності БД. Специфікація подібних умов, що накладаються на дані і відслідковуються при будь-яких їх оновленнях, покладаються на спеціальну службу Адміністратора бази даних (АБД), а системи управління базами даних (СУБД) надають інструментальні засоби, які забезпечують службі АБД можливість виконання її функцій.

За критерієм виразової потужності інструментальні засоби специфікації умов цілісності можна підрозділити на такі групи:

1) порівняння поля запису (або атрибута) з константою або з іншим полем цього ж запису; приклад такої умови наводився вище;
2) порівняння поля запису з полем або кількома полями інших записів;
3) порівняння поля запису з множиною (підмножиною) значень полів всього файлу або навіть кількох файлів. При порівняннях використовуються відношення належності (неналежності) елемента множині, або застосовуються множинні функції типу суми, кількості, середнього арифметичного, тощо.

Приклад такої умови: заробітна плата певного службовця не може більш ніж у 5 разів перевищувати середнє арифметичне від заробітної плати його підлеглих.

Зауважимо також, що вищенаведений поділ на групи має в своїй основі не тільки виразову потужність, а і складність алгоритмів реалізації.

Оскільки однією з основних властивостей БД є орієнтація на широке коло застосувань, то природно передбачити засоби захисту від неавторизованого доступу (навмисного чи ненавмисного) користувачів до даних. З цією метою в БД встановлюється система паролів та ідентифікацій користувачів, а також розподіл даних і користувачів на групи з різноманітними взаємними правами.

1.3. Класифікація АІС.

В науковій літературі існує досить значне розмаїття щодо класифікації АІС. Різні автори в залежності від своїх задач та точок зору виділяють ті чи інші критерії і розподіляють пріоритети між ними. Зупинимось на одному з таких підходів, який на наш погляд, найбільш узгоджується з іншими темами цього курсу. Отже, АІС класифікуються:

1) *за призначенням* (фактографічні, документальні та змішані);
2) *за мовами* (замкнуті системи, системи з базовою мовою та змішані);
3) *за локалізацією* (локальні та розподілені);
4) з*а схемою додаткової обробки* (постобробка та попередня обробка);
5) *за структурами даних* (ієрархічні, мережаного типу , реляційні).

Розглянемо по черзі і детальніше кожний з критеріїв.

1) *за призначенням*.

Документальні системи зорієнтовані на обробку та зберігання документа (порівняно великої за розміром послідовності символів), внутрішню структуру якого система (майже) повністю ігнорує, тобто він неподільний (атомарний) з точки зору системи. Споживачем результатів пошуку виступає, як правило, кінцевий користувач.

Фактографічні системи оперують фактами (даними) різних типів, що зв'язані в системі в більш чи менш складні структури. Дані, що є результатом пошуку, можуть стати складовою частиною звітів або використовуються різноманітними обчислювальними процесами.

Змішані системи включають в себе в тих чи інших пропорціях риси обох вищеназваних варіантів. Переважну більшість сучасних систем для ПЕОМ слід віднести до категорії змішаних.

Звичайно, наведені описові характеристики не дають можливості чітко визначитись у випадку класифікації кожної конкретної ІС, але дозволяють зробити перші грубі припущення. Для більш точних класифікаційних оцінок необхідно враховувати додаткові властивості, що відносяться до пошукового процесу, а також до особливостей мов запитів, реалізованих в тій чи іншій системі. Оскільки подальші наші розгляди будуть стосуватися переважно фактографічних систем, тому зараз приділимо більше уваги саме документальним системам.

Дескрипторні або *документальні* АІС (ДАІС) історично були першими. Спочатку їх мовою була нічим не обмежена природна мова. Перші ДАІС були призначені для пошуку книг та документів у бібліотеках і великих сховищах, тому їх і почали називати документографічними.

Основним елементом інформаційного простору ДАІС була анотація або реферат книги, документа, явища чи об'єкта. Реферат повинен відображувати ті риси, які цікавлять користувача (як правило - людини). В ньому виділяються слова чи словосполучення, які в сукупності майже однозначно (в ідеалі точно) відповідають повному опису об'єкта, крім того, таких слів повинно бути відносно небагато. Їх називають *ключовими словами* або *дескрипторами*. Запит для ДАІС можна сформулювати8 у вигляді переліку дескрипторів, який на думку користувача характеризує потрібний реферат, а значить, і відповідний об'єкт. Алгоритм формування відповіді послідовно порівнює запит з кожним рефератом і вибирає такі, що пройшли порівняння. В таких системах запит називають пошуковим розпорядженням, а реферат - пощуковим образом.

2) *за мовами* (замкнуті системи, системи з базовою мовою та змішані);

Системи з базовою мовою передбачають взаємодію користувача з СУБД з середовища якоїсь іншої мови програмування, де і виконуються більшість постпошукових перетворень даних. Такий підхід зручний для розробки різного роду систем як надбудов над СУБД, бо дає можливість створювати високоефективні програми постпошукової обробки даних.

Замкнуті системи самостійно забезпечують користувача всіма необхідними засобами як для локалізації даних, так і для їх постпошукової чи передпошукової обробки. Недоліком таких систем є те, що в них відсутні (або малоефективні) засоби для розробки надбудов – проблемно-орієнтованих комплексів.

Змішані системи передбачають наявність обох можливостей двох попередніх підходів і є найбільш поширеними на сьогодні.

3) *за локалізацією* (локальні та розподілені);

Локальність передбачає розташування всього програмного забезпечення і даних на одному ізольованому комп'ютері, а *розподіленість* означає розташування системи на мережі комп'ютерів з певною стратегією рознесення даних.

4) з*а схемою додаткової обробки* (постобробка та попередня обробка);

Головним призначенням будь-якої системи баз даних є підтримка функцій локалізації даних, що зберігаються, але дуже важливою властивістю, що може значно підняти інтерфейсний рівень системи, є наявність постобробки даних після їх локалізації в базі даних, чи попередньої обробки.

5) *за структурами даних* (ієрархічні, мережаного типу , реляційні).

Структури даних, що підтримуються в системі бази даних, - це важливий фактор, що впливає, як на виразову потужність, так і на ефективність функціонування. Для систем з *ієрархічною* структурою базовою структурою даних є дерево; як правило, вони мають найвищу ефективність функціонування, але виразові можливості їх відносно низькі. Системи з структурами даних типу мережа мають значно кращі виразові можливості, але дещо програють у ефективності функціонування, точніше, від користувача вимагається значно вищий рівень кваліфікації для ефективної експлуатації таких систем. В останні десятиріччя найбільшого розповсюдження (особливо для персональних ЕОМ) зазнали СУБД *реляційного типу*, для яких характерно найпростіша структура даних (плоский файл), але одночасно суттєво підвищений рівень мов маніпулювання даними, що максимально употужнює виразові можливості та знижує ефективність функціонування, тому для таких систем потрібні потужні комп'ютери, і вони значно чутливіші (порівняно з попередниками) до росту об'ємів даних.

1.4. Класифікація запитів.

За складністю запити поділяються на найпростіші, прості та складні. Зауважимо, що множина складних запитів може бути класифікована більш детально. Для класифікації найпростіших запитів введемо основну форму:

$$\mathbf{A\,(\,O\,) = V,}$$

де A - ім'я атрибута або властивості, відносно якої формується запит, O - специфікація об'єкта запиту, V - значення, яке може набути атрибут об'єкта; замість знаку "=" може бути використаний будь-який із знаків бінарних предикатів, що визначені для значень даної властивості об'єкта, наприклад: {<, >, =>, ...}. Для ілюстрації запитів на прикладах використаємо таблицю, що відображає торгівельну діяльність деякої фірми: виторги по місяцям у кількох кіосках. Заголовки стовпчиків таблиці будуть такими: № кіоска, виторг за січень, виторг за лютий, ... , виторг за грудень, а рядки будуть заповнені значеннями сум виторгів для певного кіоска за місяць, назва якого вказана в заголовку відповідного стовпчика.

Підставляючи знаки питання у різні місця основної форми, отримаємо всі типи запитів (знак питання буде означати запит до тієї компоненти основної форми, на місці якої він стоїть). Будемо також брати до уваги такі властивості запитів як тип результату (одне значення чи множина) та режим відпрацювання, що характеризується кількістю звернень до зовнішньої пам'яті.

1. A(O)=? - по заданому атрибуту і об'єкту знайти відповідне значення.

Приклад: Який виторг у кіоска № 2 у березні.

Такий запит називають *прямим.* Його особливості: результатом пошуку завжди є одна клітина таблиці і відповідно одне значення (значення типу невизначено теж є значенням); для режиму відпрацювання такого запиту характерно одне звернення (у кращому випадку) до зовнішньої пам'яті, оскільки можлива пряма адресація потрібного запису.

2. A(?) = V - по заданому атрибуту та значенню знайти множину об'єктів.

Приклад: Які кіоски у березні мали виторг у 2000 грн.

Такий запит називають *інвертованим.* Його особливості: результатом пошуку є (взагалі кажучи) кілька клітин у стовпчику, що специфікує рядки (об'єкти) таблиці і відповідно множина значень; при відпрацюванні такого запиту потрібно виконати повний перебір записів або завчасно створити відповідні вторинні індекси.

3. ?(O)=V - знайти імена атрибутів, що мають вказане значення по специфікованому об'єкту.

Приклад: У якому місяці виторг кіоска № 2 дорівнював 2000 грн.

Такий запит теж часто називають *інвертованим (*але у широкому розумінні), оскільки його результатом є множина значень. Однак, він суттєво відрізняється від запиту попереднього типу тим, що діє над одним рядком таблиці, а при його відпрацюванні не потрібен перебір, тому по цій властивості він більше схожий на прямий запит.

Наступна група запитів має в своєму складі по два знаки ?, що у певному розумінні означає комбінування "чистих" попередніх запитів.

4. ?(O)= ? - знайти всі значення по специфікованому об'єкту разом з відповідними іменами атрибутів.

Приклад: Для кіоска № 2 по кожному місяцю знайти значення виторгу.

Результатом запиту є множина пар значень, але по одному рядку таблиці, тобто по одному об'єкту, тому перебір не потрібен.

5. A(?)= ? - знайти всі значення по вказаному атрибуту разом з специфікаціями відповідних об'єктів.

Приклад: Знайти значення виторгу за лютий місяць для всіх кіосків.

Результатом запиту є множина пар значень, крім того потрібен перебір рядків таблиці.

6. ?(?) = V - знайти специфікації об'єктів з іменами потрібних атрибутів, де мається вказане значення.

Приклад: Знайти номери кіосків та назви місяців, коли виторг дорівнював 2000 грн.

Результатом запиту є множина пар значень, при відпрацюванні потрібен перебір рядків таблиці.

Швидше для повноти розстановки знаків "?", аніж важливості для практики, наведемо останній тип найпростішого запиту.

7. ?(?) = ? - видати всі відомості.

Приклад: Видати всі значення таблиці.

Прості запити можна отримати з найпростіших (а також і простих) за допомогою логічних зв'язок ("або", "і", "ні"). Для побудови складних типів запитів необхідне використання кванторів існування та узагальнення.

1.5. Інформаційна модель концептуального рівня. ER – модель.

Інформаційна модель концептуального рівня (ІМКР) призначена для формального опису предметної області з урахуванням різноманітних точок зору на дані, які мають різні користувачі або задачі. Важливо зауважити, що ІМКР не залежить від конкретної СУБД, чи апаратної платформи, на якій реалізована база даних. Однією з найбільш вживаних модельних мов для опису ІМКР є ER-модель (**E**ntity-**R**elationship model), яка була запропонована Ченом (P.Chen) у 1976 році, та її модифікації.

Розглянемо базові засоби ER-моделі, основу концепції якої складають такі поняття як тип об’єкту, тип зв’язку, атрибут та ключ.

Об’єктом називають сутність, яка для даної предметної області має незалежне від інших існування. Ця сутність може бути реальним (чи віртуальним) об’єктом, або поняттям. Всю сукупність однотипних об’єктів називають *типом об’єкту*, а окремий об’єкт інколи отримує назву примірника об’єкту. Тип об’єкту має ім’я та список іменованих властивостей, які називаються *атрибутами*. В класичній ER-моделі кожен примірник об’єкту по кожному атрибуту міг мати тільки одне значення, яке належало його домену, або області значень відповідного атрибуту. При графічному представленні тип об’єкту зображується прямокутником, а атрибути розміщуються всередині, або, якщо їх багато, то поза прямокутником на сходинках.

Ключем об’єкту називається атрибут (або кілька атрибутів), значення якого однозначно специфікує примірник об’єкту за умови мінімальності. Умова мінімальності означає, що зі складу ключа не можна вилучити жодного атрибута без втрати властивості однозначної специфікації примірника об’єкту. Якщо ключ об’єкту складається з одного атрибута, то умова мінімальності виконується автоматично. Ключові атрибути на графіці підкреслюються, або відмічаються ‘*’.

Зв'язок асоціює один чи кілька примірників одного типу об'єкту з одним чи кількома примірниками інших типів об'єктів. Зв'язок має ім'я (бажано з семантичним навантаженням) і характеризується арністю та типом відображення. Бінарними називаються зв'язки, які асоціюють примірники двох типів об'єктів, відповідно тернарними називаються зв'язки, що підтримують три типи об'єктів і т.д. При графічному зображенні зв'язок позначається ромбом з ім'ям всередині.

Згідно з типом відображення зв'язки поділяються на три групи. Зв'язки з типом відображення 1:1 (один до одного) - одному примірнику одного типу об'єкту ставлять у відповідність точно один примірник іншого типу об'єкту. Прикладом такого зв'язку є зв'язок між об'єктами завідувач відділу та відділ. У графічному представленні такий зв'язок має вигляд, показаний на мал. 1.5.1:

Мал. 1.5.1

Зв'язки з типом відображення 1:n (1-∞)(один до багатьох) одному примірнику одного типу об'єкту ставлять у відповідність кілька примірників іншого типу об'єкту. Прикладом такого зв'язку є зв'язок між об'єктами співробітник та відділ. У графічному представленні такий зв'язок має вигляд, показаний на мал. 1.5.2:

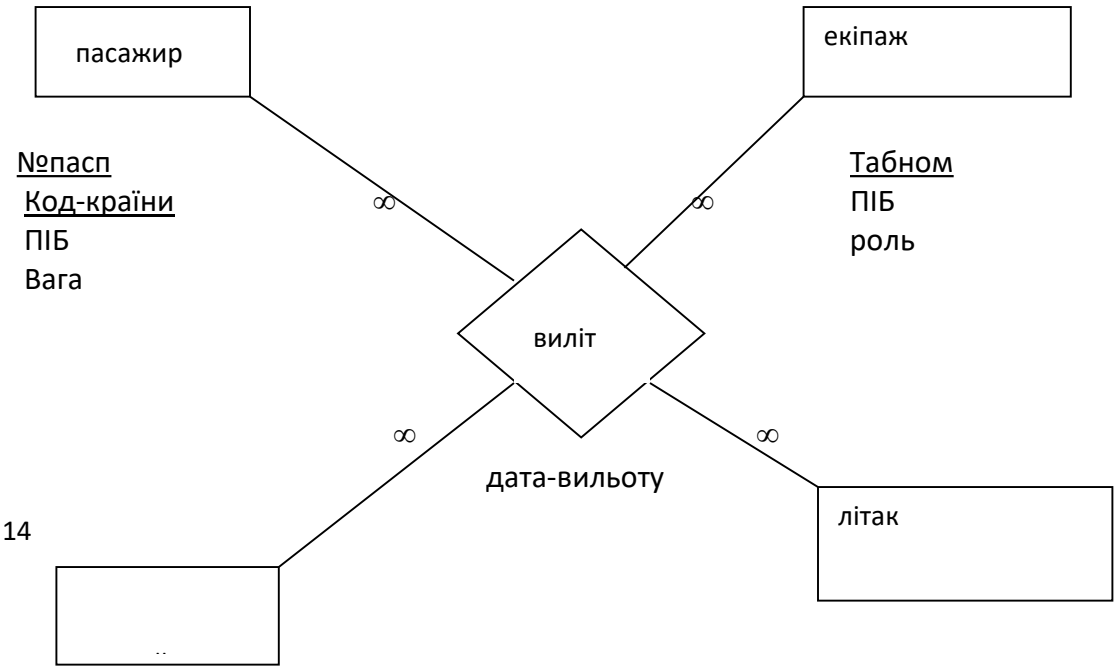
Мал. 1.5.2.

Зв’язки з типом відображення m:n (∞ - ∞) (багато до багато) багатьом примірникам одного типу об’єкту ставлять у відповідність багато примірників іншого типу об’єкту. Прикладом такого зв’язку є зв’язок між об’єктами співробітник та тема. У графічному представленні такий зв’язок має вигляд, показаний на мал. 1.5.3:

Мал. 1.5.3.

Зв’язки можуть мати власні атрибути, тобто атрибути, які належать саме зв’язку, а не якомусь з асоційованих об’єктів. Наприклад, співробітники працюють над темами по етапам або від однієї дати до іншої, тоді зв’язок «виконує» повинен мати власний атрибут № етапу або дві дати (початок та завершення).

Розглянемо більший приклад, який є фрагментом предметної області «Диспетчерська аеропорту».



№квитка

рейс
№рейсу
Час-вильоту
Час-польоту
Пункт-призн
Режим

№борту
тип-літака
дата-ТО

Зв’язок «Виліт» зображений як 4-х-арний з типом відображення «багато-багато-багато-багато». Між пасажиром і рейсом міг би бути інший тип відображення, наприклад, багато пасажирів на один рейс, але це означало б, що пасажир не може забронювати собі кілька квитків на майбутні польоти, а також, що не зберігається архів польотів. Тип об’єкту «Екіпаж» можна теж розуміти по-різному; на малюнку зображений варіант, примірник об’єкту «Екіпаж» - це один член команди, разом з тим можливий варіант, коли примірник відповідає бригаді – таке доцільно робити в тих випадках, коли склад бригади стабільний.

Атрибути «Дата-вильоту» та «№квитка» є атрибутами зв’язку «Виліт». Зауважимо також, що один 4-х-арний зв’язок міг би бути заміненим на кілька бінарних, але це дещо б змінило семантику представленої предметної області.

Тепер розглянемо процес переходу від ER-моделі до реляційних таблиць. Відзначимо, що будемо акцентуватись саме на логічному проектуванні, залишаючи поза увагою розміри реляцій, особливостей їх розміщення, а також частоти запитів до груп певних атрибутів.

1) Кожен об’єкт може бути представлений однією реляцією з відповідними атрибутами.
2) Об’єкти, поєднані зв’язком 1 – 1, можуть бути об’єднані в 1 реляцію. Наприклад, «відділ» можна з’єднати з «начальником відділу».
3) Об’єкти, поєднані зв’язком 1 - ∞, можуть бути представлені 2 реляціями, причому реляція, до якої спрямована ∞, доповнюється ключовими атрибутами

іншої реляції для забезпечення зв'язку. Наприклад, в реляцію «Співробітник» вноситься «№відділу» з реляції «Відділ».

4) Об'єкти, поєднані зв'язком ∞ - ∞, можуть бути представлені 3 реляціями, дві для об'єктів та одна для зв'язку. В зв'язкову реляцію додаються ключові атрибути двох інших реляцій (для забезпечення зв'язку), а також власні атрибути зв'язку, якщо вони є. Інколи в зв'язкову реляцію вводиться спеціальний службовий атрибут (зміст – номер рядка), який стане ключем; без службового атрибуту ключем зв'язкової реляції буде сукупність ключових атрибутів двох інших реляцій; при наявності службового атрибуту ключові атрибути інших реляцій при наявності службового атрибуту ключові атрибути інших реляцій отримують назву «чужі» ключі. Наприклад, окрім реляцій «Співробітник» та «Тема» нам потрібна ще реляція «Виконує», до складу якої будуть обов'язково входити атрибути «Таб-номер-співробітника» та «№теми», додатково ще, можливо, буде потрібен атрибут «№етапу», і ця трійка утворить ключ.

Завдання для самостійної роботи. Розробити та намалювати ER-модель для обраної самостійно предметної області. По побудованій ER-моделі розробити потрібну кількість реляцій в середовищі деякої реляційної СУБД.

2. Реляційна модель баз даних. Мови запитів.

Теоретичні основи реляційної моделі баз даних були закладені Е.Коддом на початку 70-х років і спочатку дійсно мали чисто теоретичний характер, тобто це була теорія відношень, але проінтерпретована у відповідністю з проблематикою баз даних. На відміну від поширених на той час систем з ієрархічними чи мережаними типами структур даних реляційний підхід запропонував гранично спрощені структури даних – *реляції* чи таблиці, але значно употужнив мови маніпулювання даними та мови запитів.

Введемо поняття реляції формально.

Нехай V – основний алфавіт, тобто деяка скінченна множина. ω - деякий виділений елемент; $\omega \notin V$. Забігаючи наперед, зазначимо, що ω буде означати *невизначено*.

Позначимо через $D_1', D_2', \dots D_n' \subseteq V^*$; де V^* - множина всіх слів в алфавіті V.

Домени визначаються як $D_i = D_i' \cup \{\omega\}$;

Множину імен атрибутів позначимо Ω . І введемо однозначне, але не обов'язково ін'єктивне відображення N: $\Omega \to \{D_1, D_2, \dots D_n\}$, яке іменує домени. Кожен домен може мати кілька імен, позначимо $\Omega_i = N^{-1}(D_i)$ – множину всіх імен домена D_i. Очевидно, що ці множини мають такі властивості:

$$\Omega_i \cap \Omega_j = \varnothing; \qquad \cup_{(i=1\div n)} \Omega_i = \Omega;$$

В класичному розумінні відношення визначається як підмножина декартового добутку, але для реляційної моделі в таблиці суттєвим є не місце розташування стовпчика, а його ім'я, тому формальне визначення таблиці(реляції) тут дещо складніше.

Атрибутом називається пара (A,D_i) – де $A \in \Omega_i$, тобто атрибут визначається як іменований домен, причому у різних атрибутів домени можуть бути однакові, а імена відрізнятися.

Нехай $\Re \subseteq (A_{i1},D_1)\otimes(A_{i2},D_2)\otimes\dots\otimes(A_{iк},D_к)$ – відношення над декартовим добутком атрибутів. Введемо над множиною так визначених відношень розбиття на класи еквівалентності наступним чином: до одного класу віднесемо ті відношення, які відрізняються тільки порядком компонент у декартовому добутку. Представник такого класу називається *реляційною таблицею* або *реляцією* $R((A_{i1},D_1),(A_{i2},D_2),\dots,(A_{iк},D_к))$.

Схемою реляції $R(A_1,A_2,\dots,A_к)$ називають відповідну реляційну таблицю без даних, тобто без наповнення. Сукупність схем реляцій називають *схемою бази даних*.

Розглянемо два приклади опису схем бази даних.

1. Перший приклад відноситься до предметної області, що пов'язана з постачальниками (П), деталями (Д), отримувачами (О) та поставками (ОПД). Спочатку опишемо домени:

```
CREATE DOMAIN C   CHAR(3) DEFAULT ' ';

CREATE DOMAIN N   INTEGER DEFAULT 0;
```

CREATE DOMAIN STR CHAR(20) DEFAULT ‘ ‘;

CREATE DOMAIN COLOR (черв, оранж, жовт, зел, блак, син, фіол);

У наведеному описі після ключових слів CREATE DOMAIN задається власне ім’я домена, а далі – його тип.

Тепер введемо реляції:

CREATE TABLE П (кп C, прізв STR, статус N, місто STR,
PRIMARY KEY(кп));
{кп – код постачальника}

CREATE TABLE Д (кд C, назва STR, колір COLOR, вага REAL,
місто STR, PRIMARY KEY(кд));
{кд – код деталі}

CREATE TABLE О (ко C, прізв STR, місто STR,
PRIMARY KEY(ко)); {ко – код отримувача}

CREATE TABLE ОПД (кп C, кд C, ко C, кільк N, ціна REAL,
PRIMARY KEY(кп, кд, ко));

Реляція поставка (ОПД), окрім ключових атрибутів, включає до свого складу також атрибути: кількість деталей в поставці та ціна однієї деталі. Зауважимо, що при такому описі ціна однієї деталі залежить не тільки від самої деталі, а і від постачальника та отримувача.

2. Другий приклад описує схему бази даних для предметної області – розклад занять в учбових закладах. Спосіб запису досить часто застосовується в публікаціях.

Л(кл, прізв, орг, тел, наук_ступ); /*Лектор*/
П(кп, назва, тип, год, контр); /*Навчальний предмет*/
Г(кг, фак, каф, курс, к_чол); /* Група*/
Р(кл, кп,кг, день, ауд, пара); /*Розклад*/

Підкреслені атрибути утворюють ключ реляції. Такий спосіб опису схеми бази даних використовується в тих випадках, коли тип даних для доменів атрибутів не має першочергового значення.

2.1. Реляційна алгебра.

2.1.1. Реляційна алгебра Кодда.

Реляційна алгебра є алгеброю в строгому математичному розумінні цього слова; елементи її основної множини – це реляції, а сигнатура складається з 8-ми операцій.

Теоретико-множинні операції $\cup$, $\cap$, $\setminus$ – частково визначені (визначені тільки для сумісних реляцій з однаковою структурою).

Реляції $R_1(A_1, \dots, A_n)$ і $R_2(B_1, \dots, B_k)$ називаються *сумісними*, якщо:

1. у них однакова кількість атрибутів, тобто $k = n$;
2. кожному атрибуту першої реляції можна поставити у взаємно однозначну відповідність атрибут другої реляції, тобто існує таке бієктивне відображення $S:\{1, \dots, k\} \to \{1, \dots, k\}$, що

$$N(A_i) = N(B_{S(i)}),\ i = \overline{1,k}.$$

Результуюча реляція має ту ж саму структуру, що і 1-ша реляція-операнд.

$R_1 \cup R_2$: в результуючу реляцію попадають всі кортежі першої і другої реляції без дублів.

$R_1 \cap R_2$: в результуючу реляцію попадають кортежі, присутні як в першій, так і в другій реляції.

$R_1 \setminus R_2$: в результуючу реляцію попадають кортежі з R_1, яких немає в R_2.

Деякі домени можуть бути нескінченними, тому в результаті операції доповнення можна отримати або нескінченну реляцію, або реляцію з дуже великою кількістю кортежів, тому замість операції доповнення була взята операція різниці.

19

Операція *декартового добутку* теж відноситься до теоретико-множинних операцій, але на відміну від попередніх є всюди визначеною.

$R_1 \otimes R_2$ – визначається для будь-яких реляцій.

$R_1 \otimes R_2 = \{(r_1, \dots, r_k, r_{k+1}, \dots, r_{k+n}) \mid (r_1, \dots, r_k) \in R_1, (r_{k+1}, \dots, r_{k+n}) \in R_2\}$.

Ця операція може різко збільшити об'єм результату.

Реляція $S(B_1, \dots, B_n)$ *узгоджена* з $R(A_1, \dots, A_k)$, якщо

$$\exists\, f : \Omega_S \to \Omega_R,\ N(f(B_i)) = N(A_i),\ i = 1,n$$

Проекцією реляції R на схему S відносно узгодження f будемо називати реляцію

$T = R[S]$ або $T = \{(r[A_{i1}], r[A_{i2}], \dots, r[A_{in}])\}$, тобто результуюча реляція має схему, запозичену від реляції S, але наповнена відповідними значеннями з реляції R.

$T = R[A_{i1}, A_{i2}, \dots, A_{in},]$ – такий вигляд має операція проекції у спрощеному і більш вживаному варіанті.

Реляція R несе на собі і схему, і дані, а реляція S - тільки схему.

Атрибути S повинні бути підмножиною (по доменам) множини атрибутів R.

Посилаючись на реляцію П (постачальник), запит "знайти прізвища всіх постачальників" може бути виражений так:

П[прізв].

Якщо список атрибутів не є ключем попередньої реляції, то внаслідок проекції можуть утворитись дублі, які повинні бути вилучені за визначенням реляції, бо реляція - це множина кортежів. Операція вилучення дублів пов'язана з сортуванням, тому є досить складною в сенсі виконання.

Операція *θ - з'єднання (θ - join)*.

Введемо поняття *θ* - порівнюваності для кортежів.

На двох реляціях $R(A_1, \dots, A_k)$ і $S(B_1, \dots, B_n)$ візьмемо два кортежі з

$D_1 \otimes D_2 \otimes \dots \otimes D_m$: $r_1 = (d_1, d_2, \dots, d_m)$, $r_2 = (d_1', d_2', \dots, d_m')$, а $\theta = \{=, \neq, <, \leq, >, \geq, \dots\}$;

Два кортежі r_1 і r_2 називаються *θ - порівнюваними*, якщо:

$r_1 \theta r_2 \Leftrightarrow d_i \theta d_i'$, $i = 1,m$ $(m \leq k, m \leq n)$

Нехай в реляції R виділений список атрибутів $M_1 = \{A_1, \dots, A_m\}$, а в реляції S список атрибутів $M_2 = \{B_1, \dots, B_m\}$, причому $N(A_i) = N(B_i)$, $i = 1,m$

Операція θ - з'єднання реляції R з реляцією S по спискам M_1 і M_2 :

$R[M_1 \theta M_2]S = \{t \in R \otimes S \mid t[M_1] \theta t[M_2]\}$.

Приклад. Назвемо порівняння з '=' *еквіз'єднанням*.

Розглянемо приклади; нехай задані 2 реляції

R	A_1	A_2	A_3
	A	1	1
	A	2	1
	B	1	2
	B	2	5
	C	3	3

S	B_1	B_2
	2	U
	3	V
	4	U

1) $R_1 \leftarrow R[A_3 = B_1]S$

R_1	A_1	A_2	A_3	B_1	B_2
	B	1	2	2	U

C	3	3	3	V

2) $R_2 \leftarrow R[A_2 > B_1]S$

R_2	A_1	A_2	A_3	B_1	B_2
	C	3	3	2	U

Операція еквіз'єднання називається *природним з'єднанням*, якщо в результат проходить лише один стовпчик з двох.

Наприклад: $R_1 \leftarrow R[A_3 \circ B_1]S$ – в результуючій таблиці не було б атрибута B_1.

θ- обмеження (*θ-restriction)* реляції R по M_1 і M_2:

$R[M_1 \theta M_2] = \{r \mid r \subseteq R \ \& \ r[M_1] \ \theta \ r[M_2]\}$.

3) $R_3 \leftarrow R[A_2 = A_3]$

R_3	A_1	A_2	A_3
	A	1	1
	C	3	3

Тепер перейдемо до розгляду операції ділення, але спочатку деякі додаткові визначення. Нехай задана реляція $R(A_1, \dots, A_k)$ і список атрибутів на ній $M_1 = \{A_1, \dots, A_n\}$, а також доповнюючий список атрибутів $M_2 = \{A_{n+1}, \dots, A_k\}$.

Нехай є $C_1 = D_1 \otimes \dots \otimes D_n$; $C_2 = D_{n+1} \otimes \dots \otimes D_k$; де $D_i = N(A_i)$, i = 1,k

Якщо візьмемо деякий елемент $x \in C_1$, то *образом x* по реляції R буде називатись множина підкортежів : $im_R(x) = \{y \in C_2 \mid (x, y) \in R\}$.

Розглянемо тепер схеми реляцій $R(\Omega_R)$, $S(\Omega_S)$, а також їх списки атрибутів $A= \{A_1, \dots, A_n\}$ і $B = \{ B_1, \dots, B_n \}$. Нехай $R[A_1, \dots, A_n]$ і $S[B_1, \dots, B_n]$ – сумісні, тоді операція $R[A \div B]S$ називається *діленням* R на S по спискам атрибутів A та B.

Результатом є множина підкортежів по атрибутам, що доповнюють список A реляції R та задовольняють наступним умовам.

$$R[A \div B]S = \{r[A_{n+1}, \dots, A_k] \mid r \in R \ \& \ S[B_1, \dots, B_n] \subseteq im_R(r[A_{n+1}, \dots, A_k])\}$$

Приклад використання операції ділення.

Знайти прізвища програмістів, які знають мови А і Ф одночасно.

R[мова ÷ мв]S

R	Мова	прізв
	А	І
	А	П
	ПЛ	С
	Ф	Ф
	Ф	І
	Ф	С
	ПЛ	П
	П	Ф
	П	І

S	мв
	А
	Ф

Образ І буде {А, Ф, П} – тому І проходить в результат; а П, С, Ф – не проходять в результат.

Зауважимо, що такий же результат можна було б отримати за допомогою 2-х операцій по відбору тих програмістів, які знають Ф чи А з наступною операцією перетину, але це можливо тільки в тому випадку, коли дільник наперед відомий, і не проходить, якщо дільник є результатом попередніх операцій.

Відзначимо також, що список з 8 операцій є надлишковим, бо об'єднання можна виразити через перетин та різницю і перетин – через об'єднання та різницю. Операція θ - з'єднання виражається через декартів добуток і θ-обмеження, але з практичної точки зору такий підхід є не економним, бо декартів добуток може різко збільшити об'єм проміжного результату. Операція ділення також є

23

надлишковою, тобто може бути виражена через інші операції: як це зробити – залишаємо для самостійної роботи.

Приклади використання реляційної алгебри на практиці. Будемо використовувати вкладеність операцій, позначаючи це за допомогою круглих дужок.

1. Знайти прізвища постачальників, що живуть в Одесі.

(П[місто = ‘Одеса’])[прізв] → П1

Зауважимо, що з теоретичної точки зору наведений запис є некоректним, бо в операції θ-обмеження на місці константи ‘Одеса’ повинно знаходитись ім’я атрибуту. Для теоретичної чистоти потрібно ввести додаткову реляцію Т з єдиним атрибутом (наприклад ‘мст’) та єдиним значення в ньому ‘Одеса’, тоді б розв’язок вищенаведеного завдання виглядав би так: (П[місто = мст]Т)[прізв] → П1, тобто це вже була б операція θ-з’єднання. Але, оскільки після теоретичного означення реляційної алгебри була запропонована ціла низка мов запитів (з реалізацією), де допускалась така «вільність» (бо це зручно у практичному використанні), то ми теж надалі будемо користуватись такими послабленнями.

2. Знайти ціни поставок деталей з кодом Д1, що виконують постачальники з кодом S1.

(ОПД[КП = ‘S1’ & КД = ‘Д1’])[ціна] → ОП2

Наведений приклад теж є відходом від теоретичної чистоти задля практичної зручності. Для теоретичного чистого випадку потрібно було б ввести ще одну реляцію з двома атрибутами та одним кортежем:

ОПД1	КП	КД
	S1	Д1

Тоді теоретично чистий запис виглядав би так:

(ОПД[КП,КД = КП,КД]ОПД1)[ціна] → ОП2

Легко бачити, що окрім констант в попередньому запису маємо ще одне послаблення: безпосередньо порівнюються пари атрибутів і використовується логічна зв'язка &. Такий підхід (теж з реалізованих мов запитів) дає можливість комбінувати в одній умові диз'юнкцію та кон'юнкцію, а також в різних парах порівнянь використовувати різні типи порівняння, наприклад '=' і '>'. Надалі ми теж будемо використовувати такі послаблення.

3. Знайти прізвища постачальників, які постачають принаймні одну червону деталь.

(Д[колір = 'червоний'])[КД] → Д3

(ОПД[КД = КД]Д3)[КП] → П4

(П4[КП = КП]П)[прізвище] → Res

3-1. Знайти прізвища постачальників, які постачають хоча б одну деталь з тих, що постачають постачальники, які постачають принаймні одну червону деталь.

(Д[колір = 'червоний'])[КД] → Д3

(ОПД[КД = КД]Д3)[КП] → П4

((ОПД[КП, КД])[КП ° КП]П4)[КД] → Д4

((РПД[КП, КД])[КД ° КД]Д4) → П5

(П5[КП = КП]П)[прізвище] → Res1

4. Знайти прізвища постачальників, які постачають принаймні одну не червону деталь.

(Д[колір ≠ 'червоний'])[КД] → Д3

(ОПД[КД = КД]Д3)[КП] → П4

(П4[КП = КП]П)[прізвище] → Res

5. Знайти номери одержувачів, які не одержують жодної червоної деталі від постачальників з міста N.

Задача розв'язується методом від супротивного, тобто спочатку розбивається на під задачі, перша з яких протилежна за змістом, а потім відніманням отримуємо потрібний результат. Спочатку знаходимо коди одержувачів, які одержують хоча б одну червону деталь, а потім від множини кодів всіх одержувачів віднімаємо множину кодів раніше знайдених одержувачів.

Зверніть увагу на змістовну відмінність поточного та попереднього запитів.

(П[місто = 'N'])[КП] → R1

(Д[колір = 'червоний'])[КД] → R2

(((R1[КП = КП]ОПД)[КД, КО])[КД = КД]R2)[КО] → R3

(О[КО]\R3[КО]) → R4

6. Знайти прізвища постачальників, що постачають *всі* деталі.

(ОПД[КП, КД])[КД ÷ КД](Д[КД]) → R1

(R1[КП = КП]П)[прізвище] → Res

Цей запит на множинне порівняння, бо нам треба порівняти множину кодів деталей, яку постачає кожен постачальник, з множиною кодів всіх деталей; Д[КД] – це дільник, а ділене повинно мати такий атрибут як у дільника, а також атрибут, по якому формується результат. Важливо уважно слідкувати за складом атрибутів діленого та дільника.

7. Знайти прізвища постачальників, які постачають хоча б одну деталь з тих, які постачає Іванов.

((П[прізвище = 'Іванов'][КП])[КП = КП]ОПД)[КД] → R1

(R1[КД = КД]ОПД)[КП] → R2

(R2[КП = КП]П)[прізвище] → Res

7-1. Знайти прізвища постачальників, які постачають принаймні всі ті деталі, які постачає Іванов.

((П[прізвище = ‘Іванов’][КП])[КП = КП]ОПД)[КД] → R1

(ОПД[КП, КД])[КД ÷ КД]R1 → R2

(R2[КП = КП]П)[прізвище] → Res

7-2. Знайти прізвища постачальників, які постачають точно ті деталі, які постачає Іванов.

((П[прізвище = ‘Іванов’][КП])[КП = КП]ОПД)[КД] → R1

(ОПД[КП, КД])[КД ÷ КД]R1 → R2 /* КП, які постачають принаймні всі деталі */

((ОПД[КП, КД])[КД ≠ КД]R1)[КП] → R3

/* КП, які постачають хоча б одну деталь не з R1 */

R2\R3 → R4

(R4[КП = КП]П)[прізвище] → Res

8. Знайти прізвища постачальників, які постачають тільки ті деталі, які постачає Іванов (але не обов’язково всі).

((П[прізвище = ‘Іванов’][КП])[КП = КП]ОПД)[КД] → R1

((ОПД[КП, КД])[КД = КД]R1)[КП] → R2 /* хоча б одну з R1*/

((ОПД[КП, КД])[КД ≠ КД]R1)[КП] → R3/* хоча б одну не з R1*/

R2\R3 → R4

(R4[КП = КП]П)[прізвище] → Res

9. Знайти прізвища постачальників, які постачають принаймні всі червоні та зелені деталі.

(Д[колір = ‘червоний’ ∨ колір = ‘зелений’])[КД] → R1

(ОПД[КП, КД])[КД ÷ КД]R1 → R2

(R2[КП = КП](П[КП, прізвище]))[прізвище] → R3

9-1. Знайти прізвища постачальників, які постачають принаймні одну червону та одну зелену деталь.

(Д[колір = ‘червоний’])[КД] → Д1

(Д[колір = ‘зелений’])[КД] → Д2

((ОПД[КП, КД])[КД ° КД]Д1)[КП] → П1

((ОПД[КП, КД])[КД ° КД]Д2)[КП] → П2

П1 ∩ П2 → П3

10. Знайти прізвища постачальників, які постачають одну і ту ж саму деталь всім одержувачам.

((ОПД[КП, КД, КО])[КО ÷ КО](О[КО]))[КП] → R1

(R1[КП = КП](П[КП, прізвище]))[прізвище] → R2

У цьому запиті головним є перший рядок; на перший погляд здається, що атрибут КД є надлишковим, адже завдяки останній проекції [КП], він зникає. Але тут важливо чітко згадати точне визначення операції ділення – образ для КО формується по підкортежу з двох атрибутів КП, КД, а не по одному КП, тому і з’являється цей ефект «тої самої» деталі.

11. Знайти прізвища постачальників, які постачають деталі принаймні всіх тих кольорів, що і постачальник Іванов.

((П[прізвище = ‘Іванов’][КП])[КП = КП]ОПД)[КД] → R1

(Д[КД=КД] R1)[колір] → R2 /* множина кольорів деталей Іванова - дільник*/

(Д[КД=КД]ОПД)[КП, колір] → R3 /* формування діленого */

Запити-дії на поповнення реляцій.

До реляції Д додати новий кортеж

Д ∪ {<'Д7', 'гайка', 'сірий', 20, 'Одеса'>} → Д

До реляції П додати новий кортеж

П ∪ {<'S3', 'Сидорів', 7, 'Донецьк'>} → П

Щоб зміна мала ефект, її треба в щось зберегти, можна і в початкову реляцію.

Завдання для самостійної роботи. Виразити в термінах реляційної алгебри Кодда запити з розділів 5.1, 5.2, 5.3.

2.1.2. Алгебра вибору (Дрібаса)

Реляційна алгебра Дрібаса [5] була запропонована суттєво пізніше реляційної алгебри Кодда і тому мала можливість врахувати і подолати слабкості останньої (з прагматичної точки зору). Тут, як і в алгебрі Кодда, маємо основну множину, елементами якої є реляції. Сигнатура операцій складається з двох груп: теоретико – множинні операції (∪, ∩, \, ⊗, […]-операція проекції) та чотирьох операцій фільтрації (інколи кажуть, що 3, бо $F_{\exists}$ та $F_{\forall}$ дуже схожі за складністю та способом реалізації). Теоретико-множинні операції мають ту ж саму назву, такий же синтаксис і таку ж семантику, як і в алгебрі Кодда. Розглянемо більш детально операції фільтрації.

1) $F_{\beta}(R, \beta)$.

Операція β-фільтрації в якості операндів має реляцію R та умову β, яка є ДНФ від атрибутів реляції та констант; а результатом фільтрації реляції R через фільтр β буде реляція тієї ж структури, що і R, наповнена тільки тими кортежами з R, які задовольняють умові β. Відзначимо, що ця операція практично співпадає з операцією θ-обмеження алгебри Кодда (з урахуванням раніше розглянутих нами поправок).

2) $F_{\exists}(R1, R2, \beta)$.

Вхідними операндами цієї операцій є дві реляції R1, R2 та умова β, яка є ДНФ від атрибутів обох реляцій та констант. Результуюча реляція (по структурі співпадає з R1) складається з кортежів реляції R1, для яких в реляції R2 існує хоча б один кортеж, що задовольняє умову β. Умова β перевіряється до першого виконання.

3) $F_{\forall}(R1, R2, \beta)$.

Аналогічно попередньому, вхідними операндами цієї операцій є дві реляції R1, R2 та умова β, яка є ДНФ від атрибутів обох реляцій та констант. Результат містить тільки ті кортежі з R1, для яких умова β виконується для всіх кортежів з реляції R2. Умова β перевіряється до першого невиконання.

4) $F_S(R1, R2, A, B_a\ \theta\ C)$ – фільтрація з множинним порівнянням.

Вхідними параметрами цієї операції є реляції R1 та R2; A, B_a – списки атрибутів з R1, а C – список атрибутів з R2; $B_a\ \theta\ C$ – умова множинного порівняння, де θ – бінарний предикат множинного порівняння: $\subseteq, \subset, \supseteq, \supset, =, \neq$. Результат по структурі співпадає з R1 і є частиною кортежів з цієї реляції.

a) кортежі реляції R1 групуються по значенням атрибутів списку A; $\{R1_a\}a \in A$
b) R2 проектується по C
c) з груп $\{R1_a\}$ вибираються ті кортежі, для яких виконується умова R1[B] θ R2[C]
d) в результуючу реляцію проходять тільки ті кортежі з R1, які задовольняють вказану умову.

Розглянемо кілька прикладів.

1. Знайти прізвища постачальників, що живуть в Одесі.

$(F_{\beta}(П, \text{місто} = \text{'Одеса'}))\ [\text{прізвище}] \rightarrow П1$

2. Знайти прізвища постачальників, які постачають принаймні одну червону деталь.

$F_{\exists}(ОПД, (F_{\beta}(Д, \text{колір} = \text{'черв'}), ОПД.КД=Д.КД)\ [КП] \rightarrow П1$

$F_{\exists}(П, П1, П.КП=П1.КП)\ [\text{прізвище}] \rightarrow П2$

3. Знайти прізвища постачальників, які не постачають жодної червоної деталі.

$F_{\exists}$(ОПД, (F_{β}(Д, колір = ‘черв’), ОПД.КД=Д.КД) [КП] $\rightarrow$ П1

(П[КП]) \ П1 $\rightarrow$ П2

$F_{\exists}$(П, П1, П.КП=П1.КП) [прізвище] $\rightarrow$ П2

4. Знайти прізвища постачальників, які постачають принаймні всі червоні деталі.

(F_{β}(Д, колір = ‘черв’)) [КД] $\rightarrow$ Д1

F_S (ОПД, Д1, ОПД.КП,ОПД[КД] $\supseteq$ Д1[КД]) [КП] $\rightarrow$ П2

$F_{\exists}$(П, П2, П.КП=П2.КП) [прізвище] $\rightarrow$ П3

5. Знайти прізвища постачальників, які постачають тільки червоні деталі.

(F_{β}(Д, колір = ‘черв’)) [КД] $\rightarrow$ Д1

F_S (ОПД, Д1, ОПД.КП,ОПД[КД] $\subseteq$ Д1[КД]) [КП] $\rightarrow$ П2

$F_{\exists}$(П, П2, П.КП=П2.КП) [прізвище] $\rightarrow$ П3

2.2. Реляційне числення. Мова ALPHA.

Реляційне числення (РЧ) відіграє дуже важливу роль в усій структурі реляційного підходу, фактично це наріжна брила всієї реляційної будівлі. З теоретичної точки зору РЧ є аналогом численням предикатів 1-го порядку і, таким чином, створює чітке та математично строге підґрунтя для вище розташованої надбудови. З прагматичної точки зору РЧ є основою для мов запитів дуже високого рівня.

Для досить великого числа баз даних та інформаційних систем, які були створені в дореляційну епоху, не існувало основного критерію стосовно дуже простого і водночас дуже важливого питання: наскільки потужною є та множина запитів, яку може підтримувати та чи інша система. Здавалося, що вирішити цю проблему неможливо, адже запити представляються у термінах тої чи іншої природної мови (неформальна система), а їх підтримка має здійснюватися на основі строгої формальної мови запитів у середовищі тої чи іншої СУБД. Спираючись на РЧ як аналог числення предикатів 1-го порядку Кодд

запропонував поняття *реляційної повноти*, тобто мова запитів є реляційно повною, якщо вона дає можливість описати певну множину формул (стосовно даних, які зберігаються в базі), інтерпретація яких дає можливість одержати будь-які потрібні дані з бази у вигляді відповідної реляції. Він це виразив як тезу про повноту, пізніше більшість фахівців стали називати її ***тезою Кодда:*** реляційне числення є реляційно повним.

Формально довести цю тезу неможливо; вона схожа з тезами Т'юринга та Чьорча з курсу «Теорія алгоритмів». Але були спроби навести контр-приклад, наприклад, бралася формула предикатів 2-го порядку (яка не була формулою 1-го порядку), і намагалися висловити її словами англійської (були спроби і на основі деяких інших природних мов) мови. Отриманий результат самі автори визнавали негативним, бо фраза не відповідала нормативам англійської мови.

Далі ми розглянемо власне реляційне числення у формальному викладі, мову *ALPHA* на його основі з низкою прикладів та алгоритм редукції Кодда, за допомогою якого обґрунтовується реляційна повнота реляційної алгебри.

2.2.1. Реляційне числення

Для запису формул РЧ потрібен алфавіт. Введемо його наступним чином. Символ ';' будемо використовувати як розділовий знак.

$\mathbf{V_1 = \{a; r; P; \Delta\};}$ - підалфавіт основних символів.

$\mathbf{V_2 = \{\vee; \neg; \&; \exists; \forall\};}$ - підалфавіт кванторів та логічних зв'язок.

$\mathbf{V_3 = \{\neq; \leq; \geq; =; >; <\}}$ – підалфавіт бінарних відношень порівняння

$\mathbf{V_4 = \{[;]; (;); , ; : \}}$ – підалфавіт допоміжних знаків.

$\mathbf{V = V_1 \cup V_2 \cup V_3 \cup V_4;}$ - основний алфавіт РЧ.

Індекси – це слова виду $\Delta\ldots\Delta$ **,** тобто 0 або багато таких символів.

Константи – це слова виду a∆…∆**,** або, використовуючи скорочене позначення a_m **,** де m – це кількість символів ∆**,** m=0,1,2…; якщо m=0, то записують просто a, тобто a_0=a.

Змінні - – це слова виду r∆…∆**,** з аналогічними скороченими позначеннями.

Предикати – це слова виду P∆…∆**,** з аналогічними скороченими позначеннями. Забігаючи наперед, зауважимо, що кожному предикату ставиться у відповідність певна реляція.

Зрізи – це слова виду r∆…∆[∆…∆**]** з скороченим позначенням $r_i[m]$.

Терми значень – це слова виду P_ir_j. Поєднання предиката із змінною означає, що ця змінна буде пробігати значення (кортежі) реляції, відповідної предикату, це дуже схоже на оголошення змінної певного типу, де типом є реляція.

Терми з'єднань – це слова виду λΘμ або λΘα, де λ, μ – зрізи, α – константа, Θ – бінарне відношення порівняння.

Наведемо приклади термів з'єднань.

$$r_1[1] \le r_2[3];\ \ r_1[3] = a_5;$$

Правильно побудовані формули (ППФ) означимо аксіоматично.

1) Всі терми значень і терми з'єднань є ППФ. Всі їх змінні вільні.
2) Якщо F – ППФ, то ¬F – ППФ; змінні залишаються вільними.
3) Якщо F_1 і F_2 – ППФ і їх спільні змінні вільні у кожній з формул, то $F_1 \vee F_2$ і F_1 & F_2 – ППФ, а їх змінні вільні у $F_1 \vee F_2$ і F_1 & F_2, якщо вони вільні принаймні у одній формулі.
4) Якщо F – ППФ і r – вільна у ній змінна, то ∃ r(F) і ∀r(F) – ППФ, а r – зв'язана змінна.
5) Інших ППФ немає.

Зауважимо, що побудова ППФ цілком узгоджена з відповідною конструкцією числення предикатів 1-го порядку.

Розглянемо кілька прикладів ППФ.

$P_1r_2 \vee (r_2[1] < r_3[2])$; $\forall r_2((P_1r_1 \& P_2r_2) \vee (r_2[1] \neq r_2[2]))$

Безквантorна ППФ F називається ***правильно визначеною (пв) над*** змінною r, якщо виконані такі умови:

1) r єдина вільна змінна в F;
2) всі терми в F є термами значень;
3) ¬ входить в F тільки після &, або взагалі відсутнє.

Остання умова пов'язана з тим, що у реляційному підході замість доповнення використовують різницю.

Нехай F – ППФ пв над r, а G - ППФ і має r як вільну змінну, але не має термів значень виду P_ir, тоді будемо використовувати наступні скорочення:

$$\exists\, r(F\&G) \rightarrow \exists F(G)$$

$$\forall\, r(\neg F \vee G) \rightarrow \forall F(G)$$

ППФ W називається ***формулою з областю визначення (ФОВ)***, якщо виконуються наступні вимоги:

1) $W = U_1 \& U_2 \& \ldots \& U_n \& V$, $n \geq 1$.
2) U_i для всіх i є пв над r $(i \neq j \Rightarrow r_i \neq r_j)$
3) V – або пуста, або ППФ, яка задовольняє умовам:
    a. квантори входять у V тільки у вигляді ∃F(G) або ∀F(G);
    b. кожна вільна змінна з V є вільною змінною в одній з U_i ;
    c. у V немає термів значень з вільними змінними.

Приклад: $P_1r_1 \& P_2r_2 \& \exists P_3r_3(r_3[2] = r_1[1])$

Слово $(t_1,\ldots, t_k):W$ називається *простим α-виразом*, якщо виконуються умови:

1) W - ФОВ;
2) $t_1,\ldots, t_k$ - послідовність змінних чи зрізів;

3) $\{t_1,\ldots, t_k\}$ = {множина вільних змінних з W}

Частину α-виразу $(t_1,\ldots, t_k)$ будемо називати *цільовим списком*.

α-вираз визначимо аксіоматично.

1) Простий α-вираз є α-вираз;
2) Якщо $t = (t_1,\ldots, t_k)$ і $t:W_1$, $t:W_2$ - α-вирази, тоді $t:(W_1 \vee W_2)$, $t:(W_1 \& W_2)$, $t:(W_1 \& \neg W_2)$ є α-виразами;
3) Будь-який α-вираз є наслідком пунктів 1, 2.

В основі багатьох операторів мови ALPHA лежать α-вирази.

2.2.2. Мова ALPHA

На відміну від родини мов на основі реляційної алгебри, де задається процедура знаходження потрібної реляції, у мові ALPHA описуються властивості даних, які треба знайти, тобто це непроцедурна мова, її рівень суттєво вищий.

Розглянемо низку прикладів.

1. Знайти всі відомості про всіх постачальників.

GET W(П.КП, П.прізвище, П.статус, П.місто):

У наведеному операторі відсутній предикат, за замовчуванням це означає тотожну істину; слово GET вказує на пошуковий запит, тобто ЗНАЙТИ; W специфікує область вводу-виводу, куди спрямовується результат, іншими словами це – ідентифікатор результуючої реляції, а цільовий список задає її структуру.

GET W(П)

У тому випадку, коли в результат виводяться всі атрибути деякої реляції, їх можна не перелічувати, а просто вказати ім’я реляції

2. Знайти коди постачальників з статусом більше 20, що живуть в місті N

GET W(П.КП): П.статус > 20 & П.місто = ‘N’

У цьому випадку предикат не пустий, тому будуть відібрані ті рядочки реляції П, які задовольняють предикату, а потім за допомогою цільового списку П.КП на вихід надходять тільки коди постачальників.

GET W(П.КП, П.статус): П.місто = ‘N’ DOWN П.статус

Підфраза від DOWN не належить до предикату і означає впорядкування за спаданням по П.статус результуючої множини кортежів. Аналогічним чином можна використати і ключове слово UP.

3. Знайти прізвища постачальників, що постачають деталь Д3.

RANGE ОПД X

GET W(П.прізвище): ∃X (X.КП = П.КП & X.КД = Д3)

Цей запит складніший за попередні, тому розберемо його детальніше. В якості константи заданий код деталі, це поле присутнє в реляції ОПД, ОПД.КД=Д3, але для того, щоб перейти до прізвищ постачальників (в реляції П), необхідно використати перехід по КП. У цільовому списку є тільки П (можна використати як вільну змінну), отже ОПД має з’явитись як зв’язана змінна, тобто під квантором. Але яким? Для з’ясування цього питання переформулюємо запит так, щоб зазвучали квантори: «Знайти прізвища таких постачальників, для яких **існує** поставка деталі з кодом Д3. Таким чином треба використати ∃.

Оператор RANGE ОПД X – це оператор декларації змінної, де ОПД – це тип, а X – змінна. Вільні змінні, тобто ті, що входять у цільовий список, можна не декларувати, бо діє правило замовчування, згідно з яким для вільних змінних в якості імен можна використовувати імена реляцій; змінні ж, що входять під квантор, обов’язково декларуються.

4. Знайти прізвища постачальників, які постачають принаймні одну червону деталь.

Перший варіант:

Згідно з 1-им варіантом запит реалізується двома операторами:

1) Знайти код КП такий, що для нього **існує** деталь червоного кольору.
2) Знайти прізвища постачальників таких, що для них **існують** поставки заданих (W) деталей.

RANGE Д X

GET W(ОПД.КП): ∃X (X.КД = ОПД.КД & X.колір = червоний)

RANGE W Y

GET W2(П.прізвище): ∃Y (Y.КП = П.КП)

Другий варіант:

Знайти прізвища постачальників таких, що для них **існують** поставки з кодом КП такі, що для них **існують** деталі червоного кольору.

RANGE Д X

RANGE ОПД Y

GET W(П.прізвище): ∃X ∃Y (Y.КП = П.КП & X.КД = Y.КД & X.колір = червоний)

Перший варіант реалізується в 2 оператора, а другий – одним, але з складнішим предикатом. Що краще? Залежить від властивостей відповідного компілятора, якщо у нього є потужний оптимізатор вихідного коду, то 2-ий варіант кращий для виконання.

5. Знайти прізвища тих постачальників, які постачають принаймні одну деталь, яку постачає постачальник П5.

RANGE ОПД X, Y

GET W(П.прізвище): ∃X ∃Y (X.КП = П.КП & X.КД = Y.КД & Y.КП = П5)

Переформульований варіант: Знайти прізвища тих постачальників таких, що для них існують поставки з кодами КП такі, що для них існують поставки деталей (КД) від постачальника з кодом П5.

Зауважимо, що в даному прикладі для однієї і тієї ж реляції оголошуються 2 змінні; це пов'язано з тим, що при запису предикату у нас немає можливості вказувати порядок дій, і якщо нам потрібно використати деяку реляцію кілька разів, необхідно оголосити відповідну кількість змінних саме для цієї реляції.

Зауважимо також, що подібно до числення предикатів 1-го порядку мові Альфа порядок однойменних кванторів несуттєвий, а різних кванторів – суттєвий, бо різнойменні квантори некомутативні.

6. Знайти прізвища тих постачальників, які **не** постачають деталь Д1.

Такі запити зручніше всього реалізовувати методом від супротивного, тобто спочатку виписується запит з точністю до навпаки, а потім до нього попереду дописується заперечення (¬).

RANGE ОПД X

GET W(П.прізвище): ¬(∃X(X.КП = П.КП & X.КД = Д1))

Або:

GET W(П.прізвище): ∀X(X.КП ≠ П.КП ∨ X.КД ≠ Д1)

Другий варіант запиту отримали в результаті переходу до пренексної форми, опустити символ заперечення в глибину з використанням правил де Моргана. Легко бачити, що проговорити такий запит (навіть уже записаний) українською (чи якоюсь іншою природною) мовою набагато складніше.

7. Знайти прізвища постачальників, які постачають всі деталі.

Переформулюємо запит так, щоб зазвучали квантори: знайти прізвища постачальників таких, що для **всіх** деталей **існують** поставки.

RANGE Д X

RANGE ОПД Y

GET W(П.прізвище): ∀X ∃Y(Y.КП = П.КП & Y.КД = X.КД)

8. Знайти прізвища тих постачальників, що постачають принаймні всі ті деталі, що і Петренко.

Запити такого виду при реалізації зручно декомпонувати на 2 підзапити:

1) Знайти коди деталей, для яких існують поставки такі, що для них існує постачальник Петренко ->

   ∃X ∃Z (X.КП = Z.КП & X.КД = Y.КД & Z.прізвище = Петренко);

   відзначимо, що Y.КД виступає для цього підзапиту як вільна змінна.
2) Знайти прізвища постачальників (а в підзапиті П.КП), для яких існує поставка деталі Y.КД (в підзапиті виступає як константа)
3) Потім порівнюємо множини деталей – множина деталей Петренка є підмножиною деталей шуканих постачальників, а підмножина у численні представляється імплікацією (⇒); оскільки йдеться про множинне порівняння, то попереду ставимо ∀Y.

RANGE ОПД X, X1

RANGE Д Y

RANGE П Z

GET W(П.прізвище): ∀Y (∃X ∃Z (X.КП = Z.КП & X.КД = Y.КД & Z.прізвище = Петренко) ⇒ ∃X1 (X1.КП = П.КП & X1.КД = Y.КД))

Зауважимо, що якби в запиті було **тільки тих** деталей, а не **принаймні всіх**, то все, що треба було б зробити, так це спрямувати імплікацію у протилежному напрямку, а для випадку **тих і тільки тих** – замість імплікації використати тотожність, тобто двонаправлену стрілку.

9. Знайти прізвища тих постачальників, що постачають деталі принаймні всіх тих кольорів, що і Петренко.

Загалом цей запит реалізується по тій же схемі, що і попередній, але зв'язок здійснюється не по кодам деталей, а по їх кольорам, тобто по неключовим атрибутам.

RANGE ОПД X, X1

RANGE Д Y,Y1

RANGE П Z

GET W(П.прізвище): ∀Y (∃X ∃Z∃ Y1 (X.КП = Z.КП & X.КД = Y1.КД &

Y.колір = Y1. колір & Z.прізвище = Петренко) ⇒

∃X1∃ Y2 (X1.КП = П.КП & Y.колір= Y2.колір & X1.КД = Y2.КД))

10. Знайти прізвища постачальників, які постачають одну і ту ж саму деталь всім одержувачам

Переформулюємо запит так, щоб зазвучали квантори: знайти прізвища постачальників таких, що для них існує деталь така, що для всіх одержувачів існують її поставки.

RANGE O X

RANGE Д Y

RANGE ОПД Z

GET W(О.КО): ∃Y ∀X ∃Z (Z.КП = П.КП & Z.КД = Y.КД & Z.КО = X.КО)

11. Знайти коди одержувачів, потреби яких повністю задовольняє постачальник з кодом П7.

RANGE ОПД X

GET W(О.КО): ∀X (X.КО = О.КО ⇒ X.КП = П7)

Щодо цього запиту потрібно зауважити, що потрібні одержувачі це ті, яким ідуть поставки саме від П7.

12. Знайти номери одержувачів, які використовують тільки ті деталі, що постачаються від поставника П7.

RANGE ОПД X

RANGE Д Y

RANGE ОПД Z

GET W(О.КО): ∀Y (∃Z (Z.КД = Y.КД & Z.КО = О.КО) ⇒

∃X(X.КД = Y.КД & X.КП = П7))

В цьому запиті на відміну від попереднього несуттєво від кого саме здійснюються поставки, а зв'язок виконується по деталям.

13. Знайти коди деталей, які постачаються всім одержувачам з міста 'N'.

RANGE ОПД X

RANGE О Y

GET W(Д.КД): ∀Y (Y.місто = 'N' ⇒ ∃X (X.КД = Д.КД & X.КО = Y.КО))

Запити дії вимагають до себе більш відповідального ставлення, ніж пошукові запити. Дійсно, якщо користувач помилився з пошуковим запитом, то він отримає неправильний результат, але сама база даних від цього не постраждає. Якщо ж користувач виконав некоректний запит дії, то база даних може змінитися так, що відновити її буде неможливо.

Ще одне важливе зауваження. Мова ALPHA була запропонована у ті часи, коли дружній інтерфейс був не на часі. Всі взаємодії з базою здійснювались тільки з середовища прикладних програм, а запити дії виконувалися в трифазному режимі, власне такий режим підтримується і зараз, але прихований під покривалом дружнього інтерфейсу.

14. Змінити колір деталі Д6 на червоний.

HOLD W(Д.КД, Д.колір): Д.КД = Д6

W.колір = червоний

UPDATE W

Це оператор оновлення бази даних, але, як вже відзначалося, здійснюється у три фази. На першому кроці виконується оператор HOLD – здійснюється пошук, але залишається вказівник на тому місці, звідки дані отримані (цим він і відрізняється від GET). Потім над даними, які були відібрані до середовища прикладної програми, виконуються певні дії у цьому середовищі, і нарешті, на останньому кроці здійснюється власне оновлення (UPDATE).

Подібним же чином використовуються: DELETE W – вилучення даних та PUT W – внесення даних.

Мова ALPHA має кілька вбудованих функцій. Розглянемо їх на прикладах.

15. Знайти загальну кількість постачальників.

GET W (COUNT (П.КП))

Функція COUNT підраховує кількість значень для поля будь-якого типу.

Функція TOTAL додає всі відібрані значення для деякого поля; дійсна лише для полів з арифметикою.

Функції MAX і MIN знаходять максимальне чи мінімальне значення для множини відібраних значень для деякого поля; дійсні лише для даних, де заданий порядок, зокрема для чисел.

Функція AVERAGE підраховує середнє арифметичне значення; використовуються лише по арифметичним полям.

16. Знайти коди постачальників з найбільшим статусом.

GET W (П.КП): TOP(1, П.статус)

Функція TOP упорядковує вивід у відповідності зі спаданням по другому параметру, перший параметр скільки кортежів вивідної послідовності потрібно взяти, а цільовий список вказує, які саме поля беруться з послідовності вивідних кортежів; аналогічним

чином використовується функція BOTTOM, але упорядкування здійснюється за зростанням по другому параметру.

Розглянемо ще одну групу функцій, так звані і-функції.

17. Для кожної деталі, що постачається, знайти її код та загальний об'єм поставки.

GET W(ОПД, КД, iTOTAL(ОПД, КД, кількість))

Перший параметр функції iTOTAL в даному випадку – 'ОПД' – це реляція, в якій відбувається групування по атрибуту 'КД'(який заданий як 2-ий параметр) і для кожної групи кортежів підраховується сума (тобто функція TOTAL) по атрибуту 'кількість', тобто по третьому параметру; аналогічним чином використовуються функції iCOUNT, iMAX, iMIN, iAVERAGE.

18. Знайти коди деталей, що постачаються більш ніж одним постачальником.

GET W(ОПД.КД):iCOUNT(ОПД, КД, КП) > 1

Тут функція iCOUNT задіяна не в цільовому списку, а в умові відбору.

2.2.3. Алгоритм редукції Кодда

Алгоритм редукції Кодда (АРК) має як теоретичне, так і практичне значення. З теоретичної точки зору АРК обґрунтовує реляційну повноту реляційної алгебри, а з практичної – показує як можна виразити будь-яку формулу РЧ через послідовність операцій реляційної алгебри.

Розглянемо 2 варіанти.

I. Нехай формула РЧ має вигляд: $(t_1,\ldots t_k)$: U_1 & …& U_n; де U_i – ППФ над r_i;

Для всіх $U_i(P_i) \rightarrow R_i \Rightarrow$ (формуємо) $R_1 \otimes \ldots \otimes R_n$ далі проекція по $(t_1,\ldots t_k)$

II. Нехай формула РЧ має вигляд: $(t_1,\ldots t_k)$: U_1 & …& U_p & V, де p – це кількість вільних змінних, а q – кількість зв'язаних змінних.

Позначимо відповідність $P_j \leftrightarrow R_j$, де n_j – це арність реляції R_j .

Нехай μ_j = {if j>1 then $\sum(i=1;j-1)n_i$ else 0}. μ_j – це номери стовпчиків у загальному декартовому добутку задіяних реляцій.

1) Будуємо S_i з U_i $\rightarrow S_i$ шляхом таких замін ($P_ir_i \rightarrow R_i$; $\vee$, &, $\neg$& $\rightarrow \cup,\cap,\setminus$)
2) Нехай $\Omega(S_i) \rightarrow$ об'єднання всіх R_j , сумісних з S_i далі X_i = {if (r_i – вільна змінна, або зв'язана $\exists$) then S_i else /*$\forall$*/ $\Omega(S_i)$}, тоді $S = X_1 \otimes \ldots \otimes X_n$;
3) Перетворення $V \rightarrow V^1$ шляхом вилучення кванторів; якщо $V^1 = \varnothing$, то T_{p+q} = S інакше T_{p+q} = перетворення V^1 { $\vee \rightarrow \cup$; &$\rightarrow \cap$; $r_i[m] \Theta r_k[l] \rightarrow S [(m+ \mu_j)\Theta (l+ \mu_k)]$; $(r_i[m] \Theta a) \rightarrow (S [(m+ \mu_j) \Theta a])$}
4) for j:=p+q to p+1 do $T_{p+q} \rightarrow T_p$:(T_{j-1}={if ($\exists$) then $T_j[1,2,\ldots, \mu_j]$ else/*$\forall$*/$T_j[E_j \div D_j]S_j$}, де $E_j=(\mu_j+1, \mu_j+2, \ldots, \mu_j+n_j)$, а D_j= (1, 2, …, n_j);
5) $T = T_p[C_1, C_2,\ldots, C_k]$, де C_h = {if ($t_h = r_j$) then (μ_j+1, μ_j+2, …, μ_j+n_j) else /* $t_h = r_j[m]$ */ (μ_j+m).

Розглянемо приклад. Нехай R1(A1,A2,A3); R2(B1,B2); R3(C1,C2,C3); і всі реляції несумісні, а також задана формула РЧ

$(r_1[2],r_1[3])$: P_1r_1 & $\forall P_2r_2$ $\exists P_3r_3(r_1[1]=r_3[1]$ & $r_3[3] = r_2[1])$

$S_i = R_i = \Omega(S_i)$, i=1,2,3; $\mu_1 = 0$; $\mu_2 = 3$; $\mu_3 = 5$;

$S = R1 \otimes R2 \otimes R3$; $(r_1[1]=r_3[1]$ & $r_3[3] = r_2[1]) \rightarrow S[1=6] \cap S[4=8] = T_3$;

$T_2 = T_3[1,2,3,4,5]$; $T_1 = T_2[(4,5) \div (1,2)]$ R2; $T = T_1[2,3]$;

Завдання для самостійної роботи. Виразити в термінах мови ALPHA запити з розділів 5.1, 5.2, 5.3.

2.4. Теорія відображень, мова SQL.

2.4.1. Теорія відображень як математична основа мови SQL.

Відомості про мову SEQUEL (попередник SQL) були опубліковані у 1973 році. На відміну від реляційної алгебри та реляційного числення, які на момент своєї появи носили чисто теоретичний характер, ця мова зразу позиціонувалась як вхідна мова СУБД System

R. Схожість же цих мовних компонентів реляційного підходу полягала в тому, що всі вони базувались на добре досліджених математичних теоріях:

реляційна алгебра → теорія відношень;

реляційне числення → числення предикатів 1-го порядку;

SEQUEL → теорія відображень.

Попередником мови SEQUEL вважається мова SQUARE, основний оператор якої будує образ для деякого аргументу на базі відношення, заданого у вигляді таблиці. Наприклад, нехай задана таблиця R(A,B).

R	A	B
	a_1	b_1
	a_1	b_2
	a_1	b_3
	a_2	b_3

${}_{B}R_{A}$ ({'a_1'}) = { b_1, b_2, b3} – це означає, що шукаємо образ для елемента a_1.

У більш загальному вигляді ${}_{B}R_{A}$ (S) = {b | b ϵ B, s ϵ S: (s,b) ϵ R(A,B)},

також можна використовувати суперпозицію ${}_{B}R_{A}$ (${}_{A}P_{C}$(…)) на довільну глибину.

Досить цікавою є історія з назвою мови та її розвитком. SEQUEL був задуманий, як мова функціональної парадигми і такою була його перша версія. Але через деякий час була випущена наступна версія SEQUEL2, де з'явилися нові засоби, зручніші для більш модного в той час процедурного стилю. На рубежі 70-х і 80-х років минулого століття була змінена назва мови на SQL з фінансово-організаційних міркувань. Тоді ж мова була поширена на слабенькі в ті часи персональні комп'ютери, і з неї були вилучені найбільш складні в сенсі виконання засоби, зокрема можливість безпосереднього множинного порівняння. У наступні роки ці засоби так і не були відновлені в складі мови SQL,

оскільки розробники були зорієнтовані на залучення до використання мови якомога ширшого кола користувачів. В зв'язку з цим ми спочатку розглянемо приклади на множинне порівняння у відповідності з правилами SEQUEL2 (розглядаючи це як своєрідне технічне завдання), а потім опишемо кілька методик переходу до сучасних версій мови.

2.4.2. Запити в стилі класичного SEQUEL2.

Головний пошуковий оператор мови SEQUEL2 – SELECT, як правило, складається з трьох підфраз SELECT, FROM і WHERE. Легко бачити, що структурно він дуже схожий на відповідний оператор мови SQUARE, схожим чином, підфраза SELECT задає вихідні дані, FROM визначає таблицю, а WHERE відповідальне за вхід, який подається у вигляді умови.

1. Знайти всі відомості про всіх постачальників.

```
SELECT *          SELECT П.КП, П.Прізвище, П.Статус, П.Місто
FROM П;           FROM П;
```


Цей запит поданий у двох варіантах: у варіанті зліва список всіх полів замінюється *, бо так компактніше. Зауважимо також, що кожен атрибут має префікс (через крапку) у вигляді імені таблиці; у даному контексті такий префікс не є необхідним, бо все відбувається в межах однієї таблиці, але краще виробити звичку завжди його проставляти. Відсутність підфрази WHERE означає, що умова тотожно істинна. Традиційно (але необов'язково) підфрази записують на окремих рядках одна під одною.

2. Знайти коди постачальників з статусом, більш 20, що живуть у місті N.

```
SELECT П.КП
FROM П
WHERE П.місто = 'N' AND П.статус > 20;
```

У реалізації 2-ого запиту присутня підфраза WHERE, тому з таблиці П відбираються тільки ті кортежі, які задовольняють умові.

3. Знайти прізвища постачальників, які постачають деталь Д1.

```
SELECT Прізвище
FROM П
WHERE П.КП IN
    (SELECT ОПД.КП
    FROM ОПД
    WHERE ОПД.КД = ‘Д1’);
```

Цей запит реалізується через 2 SELECT-форми, вбудовані одна в другу і з’єднані бінарним предикатом IN (тобто елемент КП належить множині). Нижній SELECT-оператор знаходить множину КП, які постачають деталь Д1. А верхній – знаходить прізвища постачальників. Наявні дужки не є необхідними, бо їх відсутність не веде до неоднозначності. Запис SELECT-операторів «сходинкою» є комфортною традицією, а не жорсткою вимогою.

4. Знайти прізвища постачальників, які постачають принаймні одну червону деталь.


```
SELECT П.Прізвище
FROM П
WHERE П.КП IN
    SELECT ОПД.КП
    FROM ОПД
    WHERE ОПД.КД IN
    SELECT Д.КД
        FROM Д
        WHERE Д.колір = ‘червоний’;
```

Дужки обов’язково ставляться, якщо може виникнути неоднозначність. Зверніть, будь-ласка, увагу на те, як сходинковий запис покращує читання програми.

5. Знайти прізвища постачальників, які постачають принаймні одну деталь, яку постачає постачальник Іванчук.

```
SELECT П.Прізвище
FROM П
WHERE П.КП IN
        SELECT ОПД.КП
        FROM ОПД
        WHERE ОПД.КД IN
        SELECT ОПД.КД
                FROM ОПД
                WHERE ОПД.КП IN
                        SELECT П.КП
                        FROM П
                        WHERE П.прізвище = ‘Іванчук’;
```

6. Знайти прізвища постачальників, які не постачають деталь Д1.

	Варіант з інверсним ходом.
SELECT П.Прізвище FROM П WHERE П.КП NOT IN SELECT ОПД.КП FROM ОПД WHERE ОПД.КД = ‘Д1’;	SELECT П.Прізвище FROM П WHERE ‘Д1’ NOT IN SELECT ОПД.КД FROM ОПД WHERE ОПД.КП = П.КП;

6-ий запит зручно реалізовувати (лівий стовпчик) методом від супротивного, вбудований SELECT-оператор знаходить КП протилежні тим, що треба, а потім, за рахунок NOT IN відшукуються потрібні прізвища. Розглянемо ще один варіант (правий стовпчик): з точки зору ефективності, він гірший за попередній, але в ньому присутня техніка (у відносно спрощеному вигляді), яка буде потрібною у подальших прикладах. У відповідності з функціональною парадигмою всі компоненти програми повинні бути

функціями, тобто всі вхідні данні задаються при виклику функції, а результат з'являється при завершенні, ніякі глобальні змінні чи параметри середовища не повинні впливати на виконання функції; іншими словами, функція діє як труба, данні завантажили на вході, а результат отримали на виході. За таких умов можлива *інверсія*, а саме данні завантажуються на виході, а результат отримуємо на вході. У мові SEQUEL (у SQL - також) підфраза SELECT оператора SELECT є виходом, а підфраза WHERE – входом, а у наведеному прикладі (нижній SELECT-оператор) - все навпаки, тому маємо інверсію. Семантика тут така: для кожного ОПД.КП (з підфрази WHERE) формується множина ОПД.КД (підфраза SELECT), стосовно якої і здійснюється перевірка належності до неї константи 'Д1'. Якщо перевірка повернула значення TRUE, то значення ОПД.КП через місточок = П.КП передається у охоплюючий SELECT-оператор; при значенні перевірки FALSE переходимо до наступного значення ОПД.КП. Зауважимо, що у підфразі WHERE у виразі ОПД.КП=П.КП префікс ОПД не є обов'язковим, бо для найближчого SELECT-оператора саме таблиця ОПД є базовою, а от префікс П є обов'язковим, бо потрібне посилання на охоплюючий SELECT-оператор.

7. Знайти номери тих деталей, що постачаються кількома постачальниками.

```
SELECT DISTINCT X.КД
FROM ОПД X
WHERE X.КД IN
    SELECT ОПД.КД
    FROM ОПД
    WHERE ОПД.КП ≠ X.КП;
```

В реалізації цього запиту знову використаний метод інверсного ходу, але тут дещо складніше. Спочатку уточнимо деякі деталі. Модифікатор DISTINCT (протилежний йому ALL використовується за замовчуванням) означає, що вивід буде здійснюватися без дублів. У мові SEQUEL в аналогічній ролі використовувався модифікатор UNIQUE.

У підфразі FROM окрім ОПД записаний ще X – це *аліас* і, хоча інколи це пояснюють як синонім, більш точним тлумаченням буде таке: X – це змінна по типу даних ОПД,

тобто змінна, значеннями якої будуть кортежі таблиці ОПД, при відсутності явного оголошення змінної мовний процесор вводить свою змінну, невідому користувачу, і використовує її замість імені таблиці. У даному прикладі використання змінної необхідно, бо потрібно розрізняти два використання таблиці ОПД у різних SELECT-операторах.

Тепер про семантику цього прикладу. У верхньому SELECT-операторі пробігаються кортежі таблиці ОПД і для кожного кортежа перевіряється, чи належить Х.КД множині ОПД.КД, яка формується для кожного ОПД.КП, якщо умова про належність виконується, а КП різні, то КД з верхнього SELECT-оператора проходить на вихід як результат.

8. Для кожної деталі, що постачається, знайти її код і назви всіх міст, де живуть її постачальники.

```
SELECT DISTINCT ОПД.КД, П.місто
FROM ОПД, П
WHERE ОПД.КП = П.КП;
```


У підфразі FROM записано дві таблиці через кому – це означає декартів добуток згаданих таблиць; взагалі кажучи, на цьому місці може бути табличний вираз, у тому числі і SELECT-оператор, який формує таблицю. Умова у підфразі WHERE означає поставки деталі постачальником.

У сучасних версіях SQL цей запит виглядав би так:

```
SELECT DISTINCT ОПД.КД, П.місто
FROM ОПД INNER JOIN П ON ОПД.КП = П.КП;
```

Операція INNER JOIN означає операцію природного з’єднання і реалізується набагато ефективніше порівняно з декартовим добутком, бо таблиця створюється віртуально на вказівниках.

9. Знайти прізвища постачальників, які постачають всі деталі.

```
SELECT П.Прізвище
FROM П
```

```
WHERE        (SELECT ОПД.КД
             FROM ОПД
             WHERE ОПД.КП = П.КП)
              =
             (SELECT Д.КД
             FROM Д);
```

Останній SELECT-оператор формує множину всіх кодів деталей з таблиці Д; далі ця множина порівнюється з множиною ОПД.КД другого SELECT-оператора для кожного ОПД.КП і, якщо множини рівні, то КП через місточок П.КП передається у охоплюючий SELECT-оператор. Такого безпосереднього порівняння множин у сучасних версіях мови SQL немає; те, як це можна моделювати, розглянемо пізніше.

10. Знайти прізвища постачальників, які постачають принаймні всі ті деталі, що і постачальник з номером П127.

```
SELECT П.Прізвище
FROM П
WHERE П.КП IN
      SELECT X.КП
      FROM ОПД X
      WHERE        (SELECT ОПД.КД
                   FROM ОПД
                   WHERE ОПД.КП = X.КП)
                     CONTAINS     /*⊇*/
                   (SELECT ОПД.КД
                         FROM ОПД
                         WHERE ОПД.КП = ‘127’);
```


51

Слово CONTAINS означає, що остання множина є підмножиною попередньої, тобто його значення подібне значенню символа $\supseteq$. Такого Крім нього, також для порівняння множин будемо використовувати э, &, $\subseteq$.

Цей же запит розглянемо з використанням засобів, які з'явилися у *SEQUEL 2.*

```
SELECT П.Прізвище
FROM П
WHERE П.КП IN
    SELECT ОПД.КП
    FROM ОПД
    GROUP BY ОПД.КП
    HAVING SET(ОПД.КД)
      CONTAINS
        (SELECT ОПД.КД
        FROM ОПД
        WHERE ОПД.КП = '127');
```


Підфраза GROUP BY ОПД.КП здійснює групування кортежів таблиці ОПД по атрибуту ОПД.КП; а у наступній підфразі функція SET(ОПД.КД) формує множину кодів деталей у межах кожної з підгруп. Відзначимо, що ключове слово HAVING має той же сенс, що і WHERE, але додатково попереджає мовний процесор про наступні «важкі» дії з множинами. Відзначимо також, що GROUP BY впливає не тільки на те, що після HAVING, але і на попередню підфразу SELECT, а саме поля після SELECT повинні бути ті ж самі, що і на GROUP BY, або узгоджені з ними.

11. Знайти коди постачальників, що не постачають у поточний момент ніяких деталей.

```
(SELECT КП
FROM П) MINUS /*EXCEPT*/
    (SELECT КП
    FROM ОПД);
```

Віднімання двох таблиць з однаковою структурою; у мові SEQUEL був MINUS, а у останньому стандарті SQL – EXCEPT, а в реалізаціях зустрічаються обидва випадки.

12. Знайти коди постачальників, що постачають деталь Д1та Д2.

```
SELECT DISTINCT ОПД.КП
FROM ОПД
GROUP BY ОПД.КП
HAVING SET(ОПД.КД) CONTAINS (‘Д1’, ‘Д2’);
```

Друга таблиця представлена константами.

13. Знайти коди постачальників, що мають такі ж значення статусу і міста, що і постачальник П127.

```
SELECT П.КП
FROM П
WHERE < П.статус, П.місто> IN
    SELECT П.статус, П.місто
    FROM П
    WHERE КП = ‘П127’;
```

Підкортеж представлений константами.

14. Знайти коди деталей, які постачаються для всіх одержувачів у місті N.

```
SELECT DISTINCT X.КД
FROM ОПД X
WHERE    (SELECT ОПД.КО
          FROM ОПД
          WHERE ОПД.КД = X.КД)
          CONTAINS
          (SELECT O.КО
          FROM O
```


53

```
        WHERE О.місто = ‘N’);
        АБО
SELECT DISTINCT ОПД.КД
FROM ОПД
GROUP BY ОПД.КД
HAVING SET(ОПД.КО)
    CONTAINS
    (SELECT О.КО
    FROM О
    WHERE О.місто = ‘N’);
```

15. Знайти пари прізвищ постачальників, які проживають в одному і тому ж місті та постачають однакові множини деталей.

```
SELECT X.Прізвище, Y.Прізвище
FROM П X INNER JOIN П Y ON X.місто = Y.місто
WHERE
    (SELECT ОПД.КД
     FROM ОПД
     WHERE ОПД.КП=X.КП)
        =
    (SELECT ОПД.КД
     FROM ОПД
     WHERE ОПД.КП= Y.КП);
```

Вводиться два аліаса (тобто дві змінні) для однієї і тієї ж таблиці.

54

2.4.3. Використання вбудованих функцій.

Відзначимо, що в різних версіях мови SQL використовується досить велика кількість вбудованих функцій; ми розглянемо найбільш вживані з них.

16. Знайти загальну кількість постачальників.

```
SELECT COUNT(*)
FROM П;
```

Функція COUNT обчислює кількість елементів у множині; вона всюди - визначена. У стандарті мови SQL аргументу функції може передувати модифікатор DISTINCT, тоді перед підрахунком вилучаються дублі, але у деяких версіях мови SQL (наприклад, SQL ACCESS) використання модифікаторів на цьому місці заборонено.

17. Знайти загальну кількість деталей з номером Д123.

```
SELECT SUM(ОПД.кількість)
FROM ОПД
WHERE ОПД.КД = ‘Д123’;
```

Функція SUM обчислює суму елементів у множині; визначена тільки для даних, для яких задані арифметичні дії; з дублями ситуація аналогічна COUNT.

18. Знайти прізвище постачальників з максимальним статусом.

```
SELECT П.Прізвище
FROM П
WHERE П.статус =
    SELECT MAX(П.статус)
    FROM П;
```

Функція MAX обчислює максимальний елемент у множині; визначена тільки для даних, для яких існує відношення порядку. Аналогічно використовуються функції MIN та AVERAGE.

19. Для кожної деталі, що постачається, знайти її код та кількість постачальників.

```
SELECT ОПД.КД, COUNT(ОПД.КП)
FROM ОПД
```

```
GROUP BY ОПД.КД;
```

Дія GROUP BY поширюється вгору на підфразу SELECT, як наслідок функція COUNT виконується не по всій таблиці, а по групі для конкретної деталі.

20. Знайти номери деталей, які постачаються більше ніж одним постачальником.

```
SELECT ОПД.КД
FROM ОПД GROUP BY ОПД.КД
HAVING COUNT(ОПД.КП) > 1;
```

Аналогічно попередньому прикладу функція COUNT виконується по групі, а не по всій таблиці.

2.4.4. Запити дії.

На відміну від пошукових запитів запити дії змінюють стан бази даних, тому виконувати їх мають право лише ті користувачі, у яких для цього є повноваження.

21. В таблиці Д знайти кортеж з кодом деталі 321 і замінити її колір на жовтий.

```
UPDATE Д
SET Д.колір = ‘жовтий’
WHERE Д.КД = ‘321’;
```


22. Збільшити кількість деталей з кодом 137, що постачаються постачальником з кодом 12, на 10.

```
UPDATE ОПД
SET ОПД.кількість = ОПД.кількість + 10
WHERE ОПД.КП = ‘12’ AND ОПД.КД = ‘137’;
```

23. Внесення нових кортежів.

```
INSERT INTO Д VALUES   (…, …, …, …);
```

Оператор INSERT має кілька синтаксичних форм, зокрема вищенаведений варіант передбачає, що кортеж, позначений в дужках, повинен мати всі значення, передбачені структурою таблиці. Наступний варіант, орієнтований тільки на ті значення атрибутів, які перелічені в списку після Д.

```
INSERT INTO Д (КД,назва) VALUES (‘Д39’,’гайка’);
```

В обох випадках більшість СУБД одним таким оператором може ввести один кортеж.

```
INSERT INTO Д
    SELECT *
    FROM Z
    WHERE колір = ‘червоний’;
```

Такий варіант оператора INSERT дозволяє ввести багато кортежів з допоміжної таблиці Z; списки полів після Д та після SELECT повинні бути узгоджені. Принагідно зауважимо, що для коректного виконання оператора INSERT необхідно, щоб заповнювана таблиця була попередньо створена, не можна вводити дані до неіснуючої таблиці. Створення таблиць здійснюється за допомогою оператора CREATE TABLE; більш детально зараз ми його розглядати не будемо (як і CREATE DOMAIN – створити новий тип), оскільки .у більшості СУБД існують високо рівневі засоби (з графікою та віконним інтерфейсом) для задання такого оператора.

24. Вилучити постачальника з кодом 013.


```
DELETE П
WHERE П.КП = ‘013’;
```

Відзначимо, що оператор DELETE без фрази WHERE (тобто умова тотожно істинна) призводить до повного вилучення всіх записів із заданої таблиці, таблиця спустошується, але продовжує існувати. При необхідності знищення таблиці можна використати оператор DROP.

2.4.5. SQL ACCESS.

Розглянемо приклади.

25. Знайти статус постачальника і ціну поставок, для яких максимальна ціна менше середньої ціни по таблиці.

```
SELECT DISTINCTROW П.статус, MAX(ОПД.ціна)
FROM П INNER JOIN ОПД ON П.КП = ОПД.КП
```

```
GROUP BY П.статус
HAVING ((MAX(ОПД.ціна)) <
        SELECT AVG(ОПД.ціна)
        FROM ОПД);
```

1) DISTINCTROW – призначений для відображення унікальних значень після виконання SELECT. Якщо у FROM стоїть лише одна таблиця, то DISTINCTROW та DISTINCT не відрізняються, але якщо у FROM табличний вираз (зокрема INNER JOIN), тоді DISTINCTROW відповідальне за відсутність дублів рядків на рівні підфрази FROM, а DISTINCT – за відсутність дублів на рівні підфрази SELECT.
2) INNER JOIN ON – природне з'єднання таблиць.
3) GROUP BY – утворює групи по однакових значеннях (в даному випадку – однаковому значенню статусу).
4) SELECT AVG – обчислюється середнє значення по таблиці.

Кожне поле, яке вказане в SELECT, може бути супроводжене аліасом, наприклад: 58

```
SELECT IN AS ALIAS.
```

Також можуть бути використані модифікатори:

- TOP 5 – взяти перші 5 значень.
- TOP 5 PERCENT – взяти перші 5% значень.

26. Знайти прізвища постачальників, які щось постачають.

```
SELECT DISTINCT П.прізвище
FROM П
WHERE EXISTS    (SELECT *
                 FROM ОПД
                 WHERE ОПД.КП = П.КП);
```

У цьому прикладі, окрім інверсного ходу використана функція EXISTS, на вході якої деяка множина, а на виході бульовське значення. Якщо множина порожня, то EXISTS повертає значення FALSE, інакше TRUE.

27. Знайти назву і вагу деталей, для яких вага більша ніж у кожної червоної деталі.

```
SELECT DISTINCTROW Д.назва, Д.вага
FROM Д
WHERE ((Д.вага) > ALL
                    (SELECT Д.вага
                    FROM Д
                    WHERE колір = ‘червоний’));
```

Єдиний новий елемент у цьому запиті – це модифікатор ALL. Справа в тому, що вбудований SELECT повертає в якості значення множину, а зліва від значка > знаходиться одиничне значення, сенс модифікатора ALL в тому, що ((Д.вага) зліва повинна бути більшою від всіх значень множини, заданої вбудованим оператором SELECT. Якби було б достатньо для успішного порівняння тільки одного значення з множини, то слід було б використати SOME чи ANY.

28. Знайти прізвища тих постачальників, які постачають деталі принаймні тієї ж кількості кольорів, що і Іванчук.

```
SELECT П.прізвище
FROM П  INNER JOIN  ( ОПД  INNER JOIN  Д  ON Д.КД=ОПД.КД )
                     ON  П.КП=ОПД.КП
GROUP BY П.прізвище
HAVING COUNT (Д.колір) =
                    SELECT COUNT (Д.колір)
                    FROM  Д  INNER JOIN  ( ОПД  INNER JOIN П ON П.КП=ОПД.КП)
                                        ON ОПД.КД=Д.КД
                     WHERE П.прізвище=”Іванчук”;
```

59

Відзначимо, що операція INNER JOIN є бінарною, тому доцільно брати в дужки відповідні підвирази. Також зауважимо, що вбудований SELECT повертає значення функції COUNT.

29. Вивести прізвища всіх постачальників у порядку зменшення їх статусу.

```
SELECT П.прізвище
FROM П
ORDER BY П.статус DESC;
```

Підфраза ORDER BY упорядковує вивід, при цьому DESC означає спадний порядок, а ASC (значення за замовчуванням) – зростаючий.

30. Знайти прізвища постачальників, які нічого не постачають.

```
SELECT П.прізвище
FROM П LEFT JOIN ОПД ON П.КП=ОПД.КП
WHERE П.КП IS NULL;
```

Операція LEFT JOIN на відміну від INNER JOIN – це зовнішнє з'єднання з головною таблицею зліва; при такому з'єднанні в результуючу таблицю заносяться всі рядки з головної таблиці і доповнюються значеннями з другої таблиці там, де існує відповідне значення ОПД.КП, а інші рядки доповнюються значеннями NULL. Оскільки у мові SQL діє тернарна логіка (третє значення NULL), то порівняння виду П.КП= NULL дасть значення NULL при будь-якому значенні П.КП, тому використана спеціальна функція IS NULL, яка повертає значення TRUE, якщо аргумент (тобто П.КП) – NULL, інакше FALSE.

Загальний порядок підфраз у операторі SELECT такий:

SELECT список елементів виводу

FROM табличний вираз

WHERE проста умова

GROUP BY список полів для групування

HAVING умова з множинними (агрегатними) функціями

ORDER BY упорядкування виводу.

60

2.4.6. Методики перетворення запитів з прямими множинними порівняннями.

Розглянемо запит, який раніше вже розглядався: знайти прізвища постачальників, які постачають всі існуючі деталі. Напишемо цей запит у двох варіантах:

Інверсний хід	З використанням GROUP BY
SELECT П.прізвище FROM П WHERE (SELECT ОПД.КД FROM ОПД WHERE ОПД.КП = П.КП) CONTAINS (SELECT Д.КД FROM Д);	SELECT П.прізвище FROM П WHERE П.КП IN SELECT ОПД.КП FROM ОПД GROUP BY ОПД.КД HAVING SET(ОПД.КД) CONTAINS (SELECT Д.КД FROM Д);

І методика (назвемо її **кількісною**) є найбільш ефективною серед інших, але, на жаль, не універсальною. ЇЇ суть полягає в тому, що всі деталі – це найбільша кількість деталей, яку може постачати деякий постачальник, тому, взявши за основу правий варіант (з використанням GROUP BY), замінимо слово SET на слово COUNT, додамо COUNT в останній оператор SELECT, а CONTAINS замінимо на >= (в даному прикладі можна просто на =) і отримаємо варіант сумісний з сучасними версіями мови SQL.

```
SELECT П.прізвище
FROM П
WHERE П.КП IN
    SELECT ОПД.КП
    FROM ОПД
    GROUP BY ОПД.КД
```


61

```
HAVING COUNT(ОПД.КД)
        =
    (SELECT COUNT(Д.КД)
    FROM Д);
```

II методика (умовна назва **«не-не»**) полягає у тому, що початковий запит переформульовується на рівні природної мови на наступний «знайти прізвища постачальників таких, що для них **не** існує деталей, яких би вони **не** постачали».

```
SELECT П.прізвище
FROM П
WHERE NOT EXISTS
        (SELECT Д.КД
         FROM Д
         WHERE NOT EXISTS
            (SELECT *
             FROM ОПД
             WHERE ОПД.КП = П.КП
                    AND ОПД.КД = Д.КД)) ;
```


III методика (умовна назва **«не-ні»**) полягає у тому, що спочатку запит реалізується в термінах мови SEQUEL (інверсний хід); цей текст надалі буде використовуватись як формалізоване технічне завдання. Для зручності введемо деякі позначення:

нехай

A	SELECT ОПД.КД FROM ОПД WHERE ОПД.КП = П.КП	B	SELECT Д.КД FROM Д

тоді нижню частину запиту ми можемо представити у вигляді $A \supseteq B$, або $B \setminus A = \varnothing$.

Таким чином, можемо записати

```
SELECT П.прізвище
FROM П
WHERE NOT EXISTS  /*для  = ∅ */
        ((SELECT Д.КД
          FROM Д)
        EXCEPT
        (SELECT ОПД.КД
         FROM ОПД
         WHERE ОПД.КП = П.КП));
```

або для випадку, коли у певній версії SQL немає операції різниці

```
SELECT П.прізвище
FROM П
WHERE NOT EXISTS  /*для  = ∅ */
        (SELECT Д.КД
         FROM Д
         WHERE Д.КД
            NOT IN
          (SELECT ОПД.КД
           FROM ОПД
          WHERE ОПД.КП = П.КП));
```


Стосовно інших типів множинного порівняння, легко бачити, що A=B ↔ { B \ A= ∅ & A\B = ∅}; A ⊃ B ↔ { B \ A= ∅ & A\B ≠ ∅}

Порівняємо ефективність виконання програм, отриманих в результаті застосування згаданих методик. Звичайно, найефективніший результат одержується по першій методиці, але вона не є універсальною. Друга та третя методики по ефективності

63

приблизно однакові, точніше у деяких випадках лідирує друга, а в інших – третя. Але третя методика дозволяє діяти більш формально.

Завдання для самостійної роботи. Виразити в термінах мови SQL та реалізувати в середовищі деякої СУБД запити з розділів 5.1, 5.2, 5.3, 5.4.

2.5. Мова Query-By-Example(QBE)

Мова QBE була запропонована M.Zloof (IBM Res. Lab.) у 1975 році публікації, а реалізація 1977 р. Семантична основа цієї мови як і у SQL – теорія відображень. Досить оригінальним був (як на той час) інтерфейс мови. Справа у тому, що у більшості випадків в ті часи користувачі спілкувалися з комп'ютером, вводячи рядки тексту, а M.Zloof запропонував табличний інтерфейс або двовимірний, тобто на екрані дисплея система малювала одну чи кілька таблиць, а користувач їх заповнював; результат теж подавався у вигляді таблиці.

2.5.1. Приклади запитів на мові QBE.

1. Знайти коди всіх деталей, що постачаються.

ОПД	КП	КД	КО	кількість	ціна
		P.x			

На екран дисплея система виводить структуру таблиці ОПД, потім користувач у потрібному полі, у даному випадку КД, записував P.x, де P. є функція виводу (P. скорочення від print), підкреслений ідентифікатор – змінна. Взагалі то, Zloof у своїй роботі не вживав термін «змінна», замість цього він використовував слово "example", тобто зразок, пояснюючи це так: потрібно набрати текст, який користувач очікує отримати, але обов'язково його підкреслити, і зовсім несуттєво: є таке дане в таблиці чи ні.

2. Знайти всі відомості про всіх постачальників.

П	КП	прізвище	статус	місто
P.				

Для скорочення набору замість запису Р. у всі поля таблиці, його можна записати тільки в поле назви таблиці.

3. Знайти коди постачальників з міста N та зі статусом більше 20.

П	КП	прізвище	статус	місто
	Р.х		>20	N

Розташування на одному рядочку 2-х і більше елементарних умов означає, що вони зв'язані між собою за допомогою кон'юнкції. Відсутність значка порівняння означає, що використаний значок «=» за замовчуванням.

4. Знайти коди постачальників з міста N або зі статусом більше 20.

П	КП	прізвище	статус	місто
	Р.х			N
	Р.у		>20	

Розташування елементарних умов на 2-х (і більше) рядках з виводом на різні змінні (Р.х і Р.у) означає, що ці умови зв'язані між собою за допомогою диз'юнкції. Відбувається начебто об'єднання двох вивідних потоків.

5. Знайти коди тих постачальників, які постачають деталі Д1 і Д2.

ОПД	КП	КД	…
	Р.х	Д1	
	х	Д2	

У цьому запиті елементарні умови розташовані на 2-х рядках, але вивід виконується через одну і ту саму змінну; вивідні потоки перетинаються, а не об'єднуються.

6. Знайти прізвища постачальників, що постачають деталь Д7.

65

П	КП	прізвище	статус	місто
	x	P.n		

ОПД	КП	КД	КО
	x	Д7	

Цей запит виконується на 2-х таблицях; спочатку в таблиці ОПД із змінною x зв'язуються коди постачальників, які постачають деталь Д7, а потім через таблицю П знаходимо прізвища цих постачальників.

7. Знайти прізвища постачальників, що постачають принаймні одну деталь червоного кольору.

П	КП	прізвище	статус	місто
	x	P.n		

ОПД	КП	КД	КО
	x	d	

Д	КД	назва	колір	вага	місто
	d		червоний		

Запит про червону деталь потребує 3-х таблиць, в іншому він схожий на попередній.

8. Знайти прізвища постачальників, що не постачають деталь Д7.

П	КП	прізвище	статус	місто
	x	P.n		

ОПД	КП	КД	КО
¬	x	Д7	

Запит від супротивного: реалізується як і в інших мовах, спочатку записуємо запит протилежного змісту, потім визначаємо, що нам потрібна протилежна (щодо x) множина і ставимо значок ¬ (в деяких версіях NOT) в поле імені таблиці, в якій x використовується як вихідна змінна.

9. Знайти прізвища постачальників, що постачають принаймні одну деталь, яку постачає постачальник з кодом П5.

П	КП	прізвище	статус	місто
	x	P.n		

ОПД	КП	КД	КО
	x	z	
	П5	z	

В цьому запиті ілюструється повторне використання таблиці ОПД, друга таблиця ОПД не малюється, а в існуючу додається рядок.

10. Знайти прізвища постачальників, що постачають всі деталі.

ОПД	КП	КД	...
	y	ALL.x	

Д	КД	...
	ALL.x	

П	КП	прізвище	...
	y	P.z	

11. Знайти коди постачальників, що постачають принаймні всі ті деталі які постачає постачальник П5.

ОПД	КП	КД	...
	P.x	[ALL.y,.]	
	П5	ALL.y	

Без П5

Блок умови
x!=П5

Використовується конструкція з ключовим словом ALL, яка в другому рядку таблиці ОПД формує множину КД, що відповідають П5 (важливий нюанс: у подальшому ця множина буде використовуватись як одне ціле, а не поелементно). У першому рядку таблиці ОПД для кожного КП (який виводиться) формується образ з КД і порівнюється з множиною ALL.y; наявність крапки означає порівняння не по =, а по $\supseteq$. Квадратні дужки виконують допоміжну роль, показуючи, що крапка використовується разом з ALL.y, тобто множина КД, які постачає П5 і, можливо, ще щось. Блок умови дозволяє задавати додаткову умову, у даному випадку – вивід без П5.

12. Знайти коди постачальників з найбільшим статусом.

П	КП	прізвище	статус	...
	P.x		c	
¬			> c	

У цьому запиті ілюструється можливість для QBE відпрацьовувати також і інверсним ходом; перший рядок зв'язує з c деяку множину статусів, на яку накладається умова">c", що означає статуси, для яких є більший статус. Нарешті заперечення дає максимальний елемент.

2.5.2. Використання вбудованих функцій.

13. Підрахувати скільки всього є постачальників.

П	КП	...
	P.COUNT.ALL.x	

Підрахунком займається функція COUNT, а ALL формує множину, по якій працює COUNT.

14. Підрахувати кількість постачальників, які щось постачають.

ОПД	КП	...
	P.COUNT.U.ALL.x	

В таблиці ОПД (оскільки КП сам по собі не є ключем) можуть зустрічатися повторні значення по полю КП; наявність модифікатора U (тобто unique) виключає дублі.

15. Підрахувати кількість постачальників які постачають деталь Д1.

ОПД	КП	КД	...
	P.COUNT.U.ALL.x	Д1	

Загалом дуже схоже на попередній випадок, але з додатковою умовою.

16. Знайти коди деталей, в яких більше одного постачальника.

ОПД	КП	КД	...
	ALL.x	P.y	

Блок умови
COUNT.ALL.x>1

Більш складна умова порівняно з попереднім прикладом.

Окрім COUNT, використовуються також функції SUM, MAX, MIN, AVERAGE.

2.5.3. Запити дії.

Запити дії на відміну від раніше розглянутих пошукових запитів означають зміну наповнення таблиць.

17. Змінити колір червоних деталей на жовтий.

Д	КД	...	колір	...
	x		червоний	
UPDATE	x		жовтий	

Першим рядком формується множина КД, а в другому рядку функція UPDATE здійснює заміну.

18. Збільшити для постачальника з кодом П2 кількість деталей на 5.

ОПД	КП	КД	...	кількість
	П2			x
UPDATE	П2			x+5

Перший рядок знаходить множину кількостей для П2, а в другому – до кожного такого значення додається 5.

19. Вилучити запис з кодом постачальника П2.

П	КП	...
DELETE	П2	

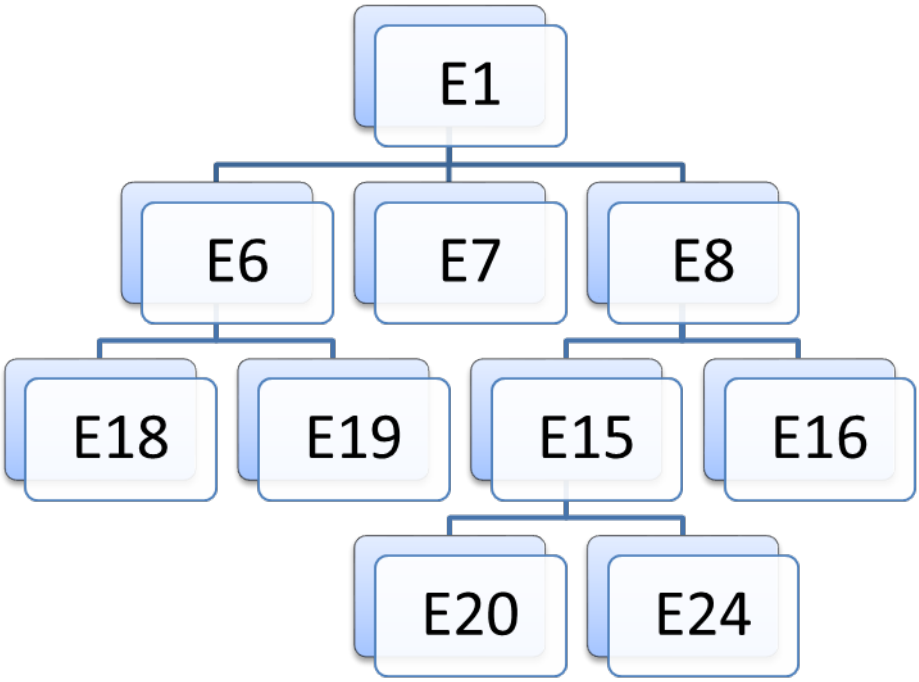
20. Ввести до таблиці П постачальника з кодом П37, прізвищем Сидорчук з Житомира, який має такий же статус, як і постачальник з кодом П12.

П	КП	прізвище...	статус	місто
INSERT	П37	Сидорчук	x	Житомир
	П12		x	

2.5.4. Ієрархічні запити в QBE.

Ієрархічні запити, тобто запити на структурі типу дерево, займають особливу позицію не тільки в QBE, але і в реляційному підході загалом. Справа в тому, що всі мови реляційного типу орієнтуються на роботу з таблицями, чи інакше плоскими файлами, а більш складні структури потрібно моделювати.

Нехай задана структура підлеглості «керівник-підлеглий», яка зображена на малюнку.



Цю структуру можна представити у вигляді табл. RS

RS	MGS	EMP
	E1	E6
	E1	E7
	E1	E8
	E6	E18
	E6	E19
	E8	E15
	E8	E16
	E15	E20
	E15	E24

Таке представлення є найбільш вживаним, хоча, якщо відомо, що дерево строго бінарне, то доцільніше було б використати таблицю з трьома полями (один керівник і два підлеглих. Розглянемо деякі запити над цією таблицею.

1. Знайти підлеглого у E8 на 1-му рівні.

RS	MGS	EMP
	E8	P.x

Змін поки що немає.

2. Знайти підлеглого у E8 на 2-му рівні.

RS	MGS	EMP
	E8	y
	y	P.x

Такий варіант теж змін не вводить.

RS	MGS	EMP
	E8	P.x(2L)

Але наступний варіант того ж самого запиту демонструє зміни (L означає Level), які принаймні зменшують кількість рядків для запису запиту, особливо, якщо б йшлося не про 2 рівень, а, припустимо, про 10 чи 20.

3. Знайти начальників на 2-му рівні відносно E20.

RS	MGS	EMP
	P.M(2L)	E20

Третій запит подібний попередньому, тільки шукає начальників, а не підлеглих. Принагідно зауважимо, що подібні запити ми можемо розгорнути у звичайні багаторядкові запити, користуючись, наприклад, технікою препроцесора, тобто використовуючи просто текстові перетворення.

4. Для E8 знайти всіх підлеглих з вказанням рівня підлеглості.

RS	MGS	EMP
	E8	P.x(6L)

Цей запит принципово відрізняється від попереднього, бо у нього підкресленим є номер рівня, тобто рівень став змінною і його значення потрібно знаходити, препроцесорної техніки тут явно недостатньо, бо ми не знаємо значення рівня поки запит не відпрацює.

В наступних запитах використовуються вбудовані функції по ієрархії.

5. Для E8 знайти вузли з максимальним ієрархічним шляхом.

RS	MGS	EMP
	E8	P.x(MAX.6L)

6. Знайти всі листові вершини дерева (співробітники, що не мають підлеглих) з коренем E8.

RS	MGS	EMP
	E8	P.x(LAST.L)

7. Знайти на якому рівні E20 є підлеглим у E1.

RS	MGS	EMP
	E1	P.E20(7L)

Є ще кілька типів цікавих запитів.

8. Видати імена таблиць, де використовується атрибут КП.

P.	КП

9. Знайти імена таблиць та стовпчиків, де серед даних є “LONDON”.

P.	P.
	LONDON

2.5.5. QBE Paradox та DataBase Desktop.

Найбільш близький до класичного варіант мови QBE належить СУБД Paradox та компоненті DataBase Desktop з інструментальних середовищ Delphi та C++ Builder (Borland). Власне йдеться про один і той же різновид мови, але у випадку DataBase Desktop він дещо урізаний. Розглянемо кілька пошукових запитів.

1. Знайти коди постачальників з міста N та зі статусом більше 20.

П	КП	прізвище	статус	місто
	√		>20	N

Легко бачити, що вся відмінність від класики в тому, що замість P.x використано значок √ (Check). У кожному полі таблиці зліва знаходиться невеличке віконечко, схоже на Checkbox; якщо клацнути правою клавішею миші по цьому віконечку, то відкриється випадаюче меню зі значками:

Вигляд значка	Назва	Коментар
√	Check	Відмітка полів, які виводяться
$\surd_{+}$	CheckPlus	Вивід значень полів з дублями
$\surd_{\blacktriangledown}$	CheckDescending	Вивід значень полів у спадному порядку
$\surd_{G}$	GroupBy	Групування кортежів таблиці за значеннями відміченого поля; виводу немає.

Структура та назва результуючої таблиці:

ANSWER	КП
	….

Результуюча таблиця завжди має назву ANSWER і складається з полів, які в запиті були відмічені вивідними √; назви полів зберігаються; якщо потрібно змінити назву поля, використовують фразу AS ідентифікатор.

Принагідно зауважимо, що результуюча таблиця зберігає свою назву до виконання наступного пошукового запиту (або до кінця сесії), тому при необхідності збереження результатів пошуку на більш тривалий час таблицю ANSWER треба перейменувати.

2. Знайти коди постачальників з міста N або зі статусом більше 20.

П	КП	прізвище	статус	місто
	√		>20	
	√			N

Диз'юнкція по умовам над різними полями виконується на 2-х рядках; якщо ж 2 і більше умов над одним полем, то використовується зв'язка OR. Наприклад, умова для поля статус могла б виглядати так: >20 OR 15. Кон'юнкція по умовам над одним полем позначається комою. Наприклад, >20, <=50. Для заперечення використовується логічна зв'язка NOT.

Доповненням щодо класики є розширення можливостей при роботі з текстовими рядками. Так, окрім перевірки на рівність, використовується предикат LIKE. Він повертає значення TRUE, якщо значення в полі узгоджене зі зразком – параметром предиката. У зразку, окрім звичайних літер латинського чи кириличного алфавітів, можуть зустрічатися "@"(означає один будь-який символ) та ".."(означає багато різних символів). У випадку ".." LIKE повертає значення TRUE, якщо співпадання значення в полі зі зразком більше ніж на 2/3 символів.

3. Знайти прізвища постачальників, які постачають принаймні одну деталь червоного кольору.

П	КП	прізвище	статус	місто
	x	√		

ОПД	КП	КД	КО
	x	d	

Д	КД	назва	колір	вага	місто
	d		"червоний"		

У випадку використання змінних, вони не підкреслюються, але відзначаються червоним кольором. Щоб задати змінну, потрібно мишкою зафіксувати поле, де буде розташована змінна і натиснути клавішу F5, після чого набрати ідентифікатор змінної(він буде червоного кольору); режим набору ідентифікатора змінної вимикається при повторному натисненні клавіші F5 або при наборі символу, який не може входити до складу ідентифікатора, наприклад, кома або пропуск. Для змінних не можна використовувати OR. Відзначимо також, що одне слово латиницею (якщо воно не ключове) в лапки можна не брати, в іншому випадку лапки необхідні.

75

4. Знайти прізвища постачальників, що не постачають деталь Д7.

П	КП	прізвище	статус	місто
	x!	√		

ОПД	КП	КД	КО
	x, count=0	Д7	

Запит від супротивного: реалізується на основі зовнішнього з'єднання; x! означає, що поле КП з таблиці П є головним, а таблиця ОПД доповнююча – вираз «x, count=0» фіксує значення типу "blank", які подібні до "NULL" в мові SQL.

5. Знайти прізвища постачальників, які постачають всі червоні деталі.

ОПД	КП	КД	...
	$\sqrt{_G}$ y	every x	

Д	КД	...	колір
set	x		"червоний"

П	КП	прізвище	...
	y	√	

В таблиці Д формується множина кодів червоних деталей; у таблиці ОПД по КП проводиться групування і для кожного y створюється образ по КД, який порівнюється у теоретико-множинному сенсі з x. Предикат порівняння визначається по ключовому слову.

Ключове слово	Значення
EVERY	$G \supseteq S$
ONLY	$G \subseteq S$
EXACTLY	$G = S$
NO	G та S не співпадають по жодному елементу

Символ G означає образ, який утворився при групуванні, а символ S – це множина, утворена інструкцією set. Відзначимо також, що для групування замість $\sqrt{_G}$ можна було б використати √, але тоді б у таблицю ANSWER попало б і поле КП – а в завданні говорилося лише про прізвища постачальників.

6. Знайти коди постачальників, які постачають тільки деталь Д1.

ОПД	КП	КД	...
	√	only “Д1”	

Розглянемо використання вбудованих функцій.

7. Для кожного постачальника (КП) знайти загальну кількість деталей, які він постачає.

ОПД	КП	Кількість	...
	√	calc sum	

Функція calc (скорочення від calculate) запускає обчислення наступної функції sum з подальшим виведенням результату, але оскільки в полі КП стоїть символ √ , то виконується групування і виводиться значення коду постачальника і сумарної кількості деталей для цього постачальника. Відзначимо також, що сумування здійснюється з врахуванням дублів (так замовчування працює для sum), щоб отримати суму без дублів, треба було б набрати calc sum unique. А для функції count все навпаки, за замовчуванням дублі не враховуються, для їх врахування потрібен модифікатор all.

Виконання запитів дії для функцій INSERT та DELETE задається подібно до класичного варіанту, але виконання запиту може відбуватися у двох режимах: звичайний (за замовчуванням) і fast. Режим fast вмикається для одного запиту на закладці Property. При режимі fast зміни вносяться в ту ж таблицю, для якої вони записувались, а при звичайному – в спеціальні таблиці INSERTED та DELETED відповідно. Таким чином, користувач має можливість перевірити результат змін до того як вони будуть внесені в основні таблиці.

Стосовно операції оновлення, то замість слова UPDATE у полі імені таблиці використовується слово changeto в тому полі, значення якого буде змінюватись.

8. Змінити деталі червоного кольору на жовтий колір.

Д	КД	колір

	x	Red
	x	changeto yellow

9. Збільшити кількість деталей, які постачаються постачальником П2, на 5.

ОПД	КП	КД	...	кількість
	“П2”			x, changeto x+5

Подібно до запитів INSERT та DELETE для цього типу запитів також існує 2 режими. При звичайному (за замовчуванням) режимі зміни вносяться в спеціальну таблицю CHANGED, а при режимі fast – в основну таблицю.

3. Залежності та нормальні форми реляцій.

3.1. Функціональні залежності (ФЗ). Квазіключ, ключ, суперключ, «чужі ключі».

Згадаємо поняття функціональної залежності. Нехай задано бінарне відношення

$R \subseteq D_1 \otimes D_2$, тоді будемо говорити, що D_1 **функціонально визначає** D_2 (а D_2 **функціонально залежить** від D_1), якщо $d_1 \in D_1$ і $|im_R\ d_1| = 1$.

Розглянемо тепер випадок n-арного відношення

$R \subseteq D_1 \otimes D_2 \otimes \ldots \otimes D_n$, $n \geq 2$; нехай M_1, M_2 – списки атрибутів відношення R, тоді

$\forall r_1 \in R[M_1]$, $\forall r_2 \in R[M_2]$ розглянемо відношення τ_R:

$r_1\ \tau_R\ r_2 \leftrightarrow \exists\ r \in R$, де $r_1 = r[M_1]$ & $r_2 = r[M_2]$

Таким чином, n-арне відношення зводиться до бінарного, а для бінарного відношення поняття функціональної залежності ми визначили. Отже, якщо τ_R є функціональним, то ми можемо говорити, що M_1 функціонально визначає M_2:

$R.M_1 \rightarrow RM_2$; а якщо до того ж τ_R є взаємно-однозначним, то

$R.M_1 \leftrightarrow R.M_2$;

Якщо Ω_R – це множина імен атрибутів деякої реляції R, то $\mathscr{B}(\Omega_R)$ – це булеан, тобто множина всіх підмножин множини Ω_R.

Пару $(\mathscr{B}(\Omega_R), f_R)$ будемо називати **структурою** функціональних залежностей реляції R, де f_R – множина функціональних залежностей для реляції R. Дещо пізніше ми переконаємося, що ця пара дійсно є структурою у строго математичному сенсі.

Розглянемо тепер поняття квазіключа (candidate of key), первинного ключа чи ключа (primary key), суперключа (super-key), «чужого» ключа (foreign key). Поняття ключа відповідно до реляційного підходу є дуже близьким до поняття ключа в ER-моделі, але не тотожне.

Квазіключем $K \subseteq \Omega_R$ (деякої реляції R) називається список атрибутів, який задовольняє умовам :

1) $\forall M \subseteq \Omega_R$, $R.K \rightarrow R.M$

2) $\forall K' \subset K$ (власної підмножини), $\exists M \subseteq \Omega_R$: $\neg(R.K' \rightarrow R.M)$

Інколи в літературі зустрічається дещо інше визначення квазіключа, але воно еквівалентне даному:

замість умови 1) задають умову 1′) $R.K \rightarrow R.\Omega_R$, а замість умови 2) задають умову

2′) $\forall K' \subset K$, $K'' = K \backslash K''$: $\neg(R.K' \rightarrow R.K'')$

Розглянемо тепер приклад реляції:

A_1	A_2	A_3	A_4	A_5	A_6	A_7
Тріска	08611	жива	200	400	140	60
Тріска	08611	заморожена	30	50	20	10
Тріска	08611	консерв.	100	75	25	75
Судак	08612	жива	100	150	34	66

Судак	08612	заморожена	150	200	75	75
Судак	08612	консерв.	100	40	35	65

де A_1 – назва продукції, A_2 – шифр, A_3 – стан, A_4 – фактично одержано, A_5 – замовлено, A_6 – одержано 1-го сорту, A_7 – одержано 2-го сорту.

Також для цієї реляції задана така структура функціональних залежностей:

$R.A_1 \leftrightarrow R.A_2$; $R.(A_6,A_7) \rightarrow R.A_4$; $R.(A_4,A_6) \rightarrow R.A_7$; $R.(A_4,A_7) \rightarrow R.A_6$

На основі описаної структури функціональних залежностей може бути створено кілька квазіключів. Атрибути A_5 і A_3 не входять до структури функціональних залежностей, тому A_5 і A_3 повинні обов'язково входити у кожен квазіключ. З A_4, A_6, A_7 будь-які два повинні ввійти у кожен квазіключ. З A_1 і A_2 – один ввійде до складу квазіключа. Отже, випишемо деякі квазіключі: $A_1A_3A_5A_4A_6$, $A_2A_3A_5A_4A_6$, $A_1A_3A_5A_4A_7$,…

Тепер перед розробником бази даних постає задача – обрати з множини квазіключів один первинний ключ; на цей вибір вже не впливають функціональні залежності, але це не означає, що він повністю довільний. Інші причини стають вагомими. Наприклад, по A_1 розмір поля змінюється в широкому діапазоні, а по A_2 розмір фіксований, отже з точки зору ефективності обробки A_2 має перевагу. З іншого боку, A_1 має вищу семантичну навантаженість. Важливими факторами також можуть бути стійкість до помилок певних атрибутів (їх доменів) при ручному вводі даних.

3.2. Первинні та вторинні атрибути. 1-а нормальна форма (1НФ); Функціонально повна залежність (ФПЗ), 2НФ, 2НФп; Теорема Хіза.

Атрибути, які входять до складу хоча б одного квазіключа називаються **первинними**.

Атрибути, які не входять до складу жодного квазіключа називаються **вторинними**.

Кажуть, що реляція знаходиться в першій нормальній формі (**1НФ**), якщо всі її атрибути атомарні (тобто неподільні). Оскільки всі таблиці, з якими ми мали справу до

цього часу, були в 1НФ, то поговоримо про реляції не в 1НФ. Приклад: відомість на заробітну плату. Деякі стовпчики цієї таблиці є складеними (наприклад, «Нараховано», «Утримано»), тобто в їх складі в свою чергу знаходяться інші стовпчики. Процес перетворення такої таблиці до 1НФ називається нормалізацією до 1НФ, або просто **нормалізацією**.

M_2 **функціонально повно залежить** (ФПЗ) від M_1, якщо

1. $R.M_1 \rightarrow R.M_2$

2. $\forall$ $A \subset M_1$ (власної підмножини M_1) $\exists$ $B \subseteq M_2$: така, що $\neg$(R.A→R.B)

Кажуть, що реляція знаходиться в 2-ій нормальній формі (**2НФ**), якщо вона знаходиться в першій нормальній формі і кожний вторинний атрибут функціонально повно залежить від кожного квазіключа.

На перший погляд може скластися враження, що наведене визначення дуже далеке від практичних потреб, але це не так. Розглянемо приклад таблиці Т:

Т	КП	КД	місто	КП→ місто – це єдина залежність.

Квазіключем і ключем є (КП, КД), а місто - вторинний атрибут. Маємо залежність від КП і КД. Оскільки є залежність КП→ місто, то реляція не знаходиться в 2НФ. А тепер проблеми з практики: нехай в таблиці знаходиться інформація, що постачальник П1 з міста О постачає деталь Д1. В якийсь момент він припиняє поставку, тоді треба вилучати відповідний кортеж, але якщо це був єдиний кортеж з постачальником П1, то ми втратимо інформацію про те, що П1 живе в місті О. Через деякий час П1 відновив поставки деталі Д7 і відповідний кортеж має бути занесений в таблицю, але в ньому немає даних про місто постачальника П1. Проблеми такого типу були відомі задовго до появи реляційного підходу, вирішувались інтуїтивним способом і називались відповідно **аномаліями** вилучення та вставки. Була також аналогічна проблема – аномалія оновлення.

Вирішення цих проблем полягає у декомпозиції реляції Т на Т1=Т[КП,місто] і Т2=Т[КД,КП]; у такому випадку згадані проблеми зникають. Відзначимо також, що реляція Т може бути відновлена з 2-х реляцій без втрати інформації за допомогою операції природного з'єднання по атрибуту КП. Дійсно, оскільки КП і КД утворюють ключ реляції Т, то в Т2 ми будемо мати точно таку ж кількість кортежів, що і в початковій реляції Т (вилучення дублів неможливе), а доповнення значеннями міста (3-й атрибут) здійснюється з Т1 однозначно, бо має місце функціональна залежність КП→ місто. Реляції Т1 і Т2 знаходяться в 2НФ, оскільки ключ з Т1 складається з одного атрибута, а реляція Т2 не має вторинних атрибутів.

Достатні умови знаходження реляції в 2НФ.

1) всі атрибути первинні;
2) кожен квазіключ має один атрибут.

Теорема Хіза (Heath I.Y.):

Якщо $R.M_1 \rightarrow R.M_2$, тоді R можна декомпонувати на дві реляції з можливістю відновлення без втрат. Природне з'єднання відбувається по атрибутам M_1.

$$R = (R[M_1, M_2]) [M_1 ° M_2] R[\Omega_R \setminus (M_2 \setminus M_1)]$$

$$\Omega_R \setminus (M_2 \setminus M_1) = M_1 \cup \neg(M_2)$$

Ідея доведення – по суті та ж сама, що і у попередньому прикладі.

Беремо реляцію $R[\Omega_R \setminus (M_2 \setminus M_1)]$ (вона має ту ж саму кількість кортежів, що і R) і доповнюємо її елементами з M_2, які однозначно визначаються на основі функціональної залежності від M_1, в результаті отримуємо повну реляцію R.

Процедура отримання реляцій у 2НФ наступна: потрібно дослідити залежності заданої реляції і визначити, чи є порушення 2НФ. Якщо порушень немає, то процедура завершена і ми маємо 2НФ. Якщо ж порушення є, то потрібно взяти залежність, яка створює це порушення, і застосувати теорему Хіза. В результаті отримаємо дві реляції, стосовно яких знову застосуємо попередню процедуру. Оскільки кількість як

залежностей, так і атрибутів у початковій реляції скінченна, то рано чи пізно вийдемо на 2НФ. Цей процес називається декомпозицією реляції до 2НФ, або нормалізацією (2НФ). Основна проблема полягає в тому, що таких порушуючих 2НФ залежностей може бути кілька, а вибір тої чи іншої призводить до різних кінцевих результатів.

Оптимальною декомпозицією вважається та, у якої найменше реляцій. Тому для отримання оптимальної декомпозиції потрібно отримати всі, а потім вибрати серед них оптимальну. Очевидно, що на практиці застосовувати алгоритми переборного типу не є достатньо ефективним рішенням, тому використовують методики, про які мова піде пізніше.

Кажуть, що реляція знаходиться в 2-ій нормальній формі посиленій (**2НФп**)(або у 2-ій нормальній формі *Бойса-Кодда* (Boyce-Codd)), якщо вона знаходиться в першій нормальній формі і **кожен** атрибут функціонально повно залежить від кожного квазіключа. Іншими словами знімається вимога вторинності атрибута.

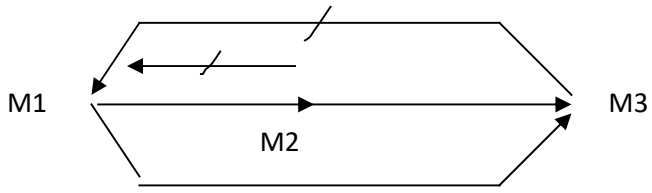
3.3. Транзитивна функціональна залежність в сенсі реляційного підходу, 3НФ, 3НФп чи нормальна форма Бойса-Кодда. Декомпозиція реляцій до 3НФ.

Нехай M_1 і M_2, $M_3 \subseteq \Omega_R$, $M_1 \neq M_2$, $M_3 \not\subset M_2$

M_3 **транзитивно** залежить від M_1, якщо $R.M_1 \rightarrow R.M_2$ & $R.M_2 \rightarrow R.M_3$ & $\neg(R.M_2 \rightarrow R.M_1)$.

Відзначимо, що $\neg(R.M_3 \rightarrow R.M_1)$ є наслідком.

Зобразимо це графічно:



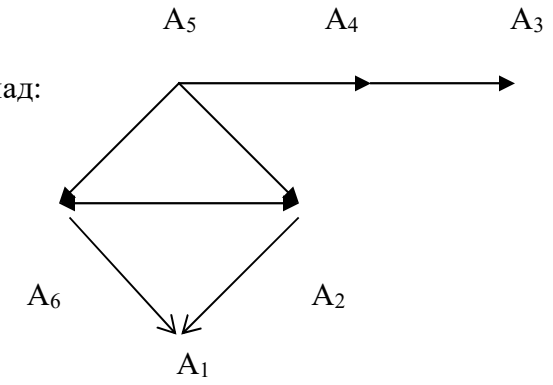
Зауважимо, що транзитивна залежність у сенсі реляційного підходу відрізняється від поняття транзитивної залежності у класичному сенсі тим, що вимагається

$\neg(R.M_2 \rightarrow R.M_1)$.

Кажуть, що реляція знаходиться в **3НФ**, якщо вона в 2НФ і не має транзитивної залежності вторинних атрибутів від кожного квазіключа.

Кажуть, що реляція знаходиться в **3НФп**, якщо вона в 2 НФп і не має транзитивної залежності кожного атрибута від кожного квазіключа.

Приклад:



A_1 – шифр міністерства,

A_2 – шифр головного управління,

A_3 – шифр області,

A_4 – шифр району,

A_6 – шифр підприємства,

A_7 – шифр галузі.

$\{ A_5 \rightarrow A_6 \rightarrow A_1 \}$

$\{ A_5 \rightarrow A_2 \rightarrow A_1 \}$ A_5 є ключем ієрархічної структури.

$\{ A_5 \rightarrow A_4 \rightarrow A_3 \}$

Маємо транзитивну залежність $A_5 \rightarrow A_4 \rightarrow A_3$, причому це транзитивна залежність як в класичному, так і в реляційному сенсі. Залежності $A_5 \rightarrow A_6 \rightarrow A_1$ як і $A_5 \rightarrow A_2 \rightarrow A_1$ - також транзитивні в обох сенсах. А от залежність $A_6 \rightarrow A_2 \rightarrow A_1$ (чи $A_2 \rightarrow A_6 \rightarrow A_1$) є транзитивною в класичному сенсі, але не транзитивною в реляційному. Такі структури в логічному проектуванні часто називають «трикутником» (жаргонна назва).

В даному прикладі існують 2 транзитивні залежності і тільки один первинний атрибут, тому ця реляція не знаходиться в 3 НФ. Для декомпозиції заданої реляції до 3НФ скористаємося теоремою Хіза. Відсікаємо $A_4 \rightarrow A_3$. У відповідністю з теоремою Хіза отримаємо $R1(\underline{A_4},A_3)$ і $R^1(\underline{A_5},A_6,A_1,A_2,A_4)$. Отримали 2 реляції: R1 знаходиться в 3НФ, а R^1 – ні, продовжуємо декомпонувати R^1, наприклад, по залежності $A_5 \rightarrow A_6 \rightarrow A_1$. Отримуємо: $R2(\underline{A_5},A_6)$ і $R^2(\underline{A_6},A_2,A_1)$ – обидві знаходяться в 3НФ. Отже результат декомпозиції $R1(\underline{A_4},A_3)$; $R2(\underline{A_5},A_6)$; $R3(\underline{A_6},A_2,A_1) = R^2$. Методика, яка була задіяна для цього прикладу, характерна для ієрархічних структур: за допомогою теореми Хіза потрібно відсікати листові вершини і рухатися вгору до кореня.

Достатньою умовою знаходження реляції в 3НФ є структура типу "сонечко": в центрі знаходяться ключові атрибути, а на променях, довжиною в одну залежність – вторинні атрибути.

3.4. Багатозначна залежність. 4НФ. Залежність по з'єднанню без втрат. 5НФ.

4НФ базується не на функціональних, а на багатозначних залежностях, тому спочатку поговоримо про них.

КВП	Курс	Викладачі	Підручник
	програмування	Іванчук	Pascal
	програмування	Іванчук	C
	програмування	Сидоренко	Pascal
	програмування	Сидоренко	C

Розглянемо таблицю КВП. Курс програмування ведуть два викладачі і для курсу використовується два підручники. Легко бачити, що функціональних залежностей між атрибутами цієї таблиці немає; разом з тим якісь залежності є – певний курс можуть вести не всі викладачі, а тільки деяка їх підмножина, подібні ж стосунки між курсом та підручниками, а також викладачами та підручниками. Залежності такого виду будемо називати багатозначними (multivalued).

Перейдемо до формального визначення поняття багатозначної залежності.

85

Нехай X і У списки атрибутів реляції R.

Визначимо поняття узагальненого образу

$$X,У \subset \Omega_R: im_R(X,Y)=\{y \mid z \in R \ \& \ z[X]=x \ \& \ z[Y]=y\}$$

Визначимо Z, як $Z = \Omega_R \setminus (X \cup Y)$.

$X \rightarrow\rightarrow У$ (У **багатозначно залежить** від X – списка атрибутів), якщо

$\forall (x,z) \in XZ, im_R(XZ, Y) = im_R(X, Y)$.

Розглянемо ще одну таблицю ВС. Між її атрибутами знову таки немає функціональної залежності.

ВС	КЛ	КП	Рік	З/п
	ПТ	геометрія	1979	180
	ПТ	алгебра	1979	180
	ПТ	геометрія	1980	200
	ПТ	алгебра	1980	200
	СД	математика1	1979	250
	СД	математика2	1979	250
	СД	математика1	1980	270
	СД	математика2	1980	270

Проаналізуємо цю реляцію щодо наявності багатозначних залежностей. Нехай X – це КЛ, а У – КП, тоді Z – це (рік, з/п). Узагальнений образ {ПТ, 1979, 180} є {геометрія, алгебра}, узагальнений образ {ПТ} - це теж { геометрія, алгебра}. Крім того, узагальнений образ {ПТ, 1980, 200} теж є {геометрія, алгебра}. Аналогічний результат отримаємо також і для значення «СД». Таким чином, наповнення таблиця не заперечує того, що $КЛ \rightarrow\rightarrow КП$. Подібним же способом ми можемо проаналізувати наповнення і для $КЛ \rightarrow\rightarrow$(рік, з/п). Відзначимо, що на основі даних в таблиці можемо зробити висновок про те, що наповнення таблиці заперечує чи не заперечує наявність тої чи іншої багатозначної залежності, але не можемо стверджувати, що така залежність існує. Наявність

86

багатозначної залежності (наприклад, КЛ$\rightarrow\rightarrow$КП) випливає з аналізу предметної області, наприклад, відомо що КП відповідає чітко визначена підмножина КЛ.

Функціональна залежність є частковим випадком багатозначної залежності, тобто, якщо має місце функціональна залежність, то є і багатозначна; зворотне твердження, взагалі кажучи, невірне.

Багатозначна залежність називається **тривіальною**, якщо вона дублюється функціональною, інакше вона є **нетривіальною**.

Реляція знаходиться в **4НФ,** якщо вона знаходиться в 3НФ і не має нетривіальної багатозначної залежності, або

4НФ: $R \Leftrightarrow A\rightarrow\rightarrow B \Rightarrow A\rightarrow \Omega_R$ +3 НФ (тобто А - квазіключ)

Реляцію ВС можна декомпонувати на реляції в 4НФ. ВС1(КЛ,КП) і ВС2(КЛ, рік, з/п), тоді отримаємо 2 таблиці по 4 рядки у кожній, але довжина рядків менша.

Теорема Фейджіна (Fagin). (ця теорема дуже схожа на теорему Heath).

Якщо в реляції R(A,B,C) є залежність A$\rightarrow\rightarrow$B, то реляція може бути декомпонована на 2 реляції R(A,B) і R(A,C) без втрат даних, тобто R може бути відновлена природним з'єднанням по А. (А, В,С – це списки атрибутів).

Розглянемо тепер ще один вид залежностей, які узагальнюють багатозначні залежності. Легко бачити, що в теоремах Heath і Fagin з існування залежності *випливає* можливість декомпозиції без втрат даних, але обернене твердження, взагалі кажучи, невірно.

Для реляції R(A,B,C) A ~» B $\Leftrightarrow$ можлива декомпозиція на 2 реляції R(A,B) і R(A,C) без втрат даних. Такі залежності будемо називати залежностями **по з'єднанню без втрат**.

Багатозначні залежності є частковим випадком залежностей по з'єднанню без втрат, тобто якщо має місце багатозначна залежність, то є і залежність по з'єднанню без втрат; зворотне твердження, взагалі кажучи, невірне.

Залежність по з'єднанню без втрат називається **тривіальною**, якщо вона дублюється багатозначною, інакше вона є **нетривіальною**.

Реляція знаходиться в **5НФ**, якщо вона знаходиться в 4НФ і не має нетривіальних залежностей по з'єднанню без втрат.

Функціональна залежність $\Rightarrow$ багатозначна залежність $\Rightarrow$ залежність по з'єднанню без втрат.

4. Логічне проектування баз даних

4.1. Аксіоми Армстронга.

Для чого потрібні аксіоми Армстронга?

Є принаймні дві причини:

1) множина функціональних залежностей реляційної БД є структурою, точніше верхньою напіврешіткою;

2) існують мінімальні структури функціональних залежностей, які є суттєво зручнішими при декомпозиції реляцій до 3НФ чи 3НФп.

При створенні структури функціональних залежностей для деякої предметної області групі розробників БД доводиться протягом довгого часу спілкуватися з експертами відповідної галузі, причому далеко не завжди цей процес здійснюється у потрібному формально-математичному стилі. Тому бажана процедура мінімізації.

Властивості (система R):

R_1: $M_1 \supseteq M_2 \Rightarrow M_1 \rightarrow M_2$ - проективність

R_2: $M_1 \rightarrow M_2 \,\&\, M_2 \rightarrow M_3 \Rightarrow M_1 \rightarrow M_3$ - транзитивність

R_3: $M_1 \to M_2 \Rightarrow (M_1 \cup M_3) \to (M_2 \cup M_3)$ – монотонність

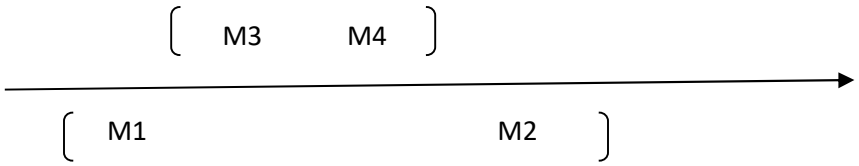
Система R – це 3 властивості, які виконуються для будь-якої множини функціональних залежностей.

Визначення часткового порядку для пар множин

$(M_1, M_2) \text{ » } (M_3, M_4) \Leftrightarrow M_1 \subseteq M_3 \ \& \ M_2 \supseteq M_4$

Розглянемо приклад на відрізках, який хоч і є досить далеким аналогом введеного означення часткового порядку, але зручний для запам'ятовування.

Приклад:



Списки атрибутів M_1 ототожнюємо з точкою M1 на дійсній осі, M_2 – з M2, M_3 – з M3, M_4 – з M4. Те, що $M_1 \subseteq M_3$ перетворюється в M3 ≥ M1, а $M_2 \supseteq M_4$ – в M2 ≥ M4. Таким чином, відрізок [M3, M4] включається у відрізок [M1, M2]. Відрізок більший той, який включає в себе менший. Аналогічним способом вводиться частковий порядок на парах множин. Після введення часткового порядку розглянемо систему P. Тут вже йдеться про будь-які множини, а символ «→» означає просто деяке відношення. Система P задає верхню напіврешітку.

Система аксіом Армстронга (система P):

P_1: $M_1 \to M_1$ – рефлексивність;

P_2: $P2 \equiv R_2$ - теж транзитивність;

P_3: $M_1 \to M_2 \ \& \ (M_1, M_2) \text{ » } (M_3, M_4) \Rightarrow M_3 \to M_4$

Якщо якась властивість (залежність) задана на $[M_1, M_2]$, то вона задана і на $[M_3, M_4]$

P_4: $M_1 \to M_2 \ \& \ M_3 \to M_4 \Rightarrow (M_1 \cup M_3) \to (M_2 \cup M_4)$

Твердження: системи R і P еквівалентні:

Доведення:

$P \Leftarrow R$:

P_1: $M_1 \supseteq M_1 \Rightarrow M_1 \to M_1$

P_2: $P_2 \equiv R_2$

P_3: $M_1 \to M_2$ & (M_1, M_1) » (M_1, M_2)

$M_1 \subset M_3 \Rightarrow M_3 \to M_1$
$M_2 \supseteq M_4 \Rightarrow M_2 \to M_4$
$\Rightarrow M_3 \to M_4$

P_4: $M_1 \to M_2$ | $M_1 \cup M_3 \to M_2 \cup M_3$ | $M_1 \cup M_3 \to M_4 \cup M_2$

$M_3 \to M_4$ $M_3 \cup M_2 \to M_4 \cup M_2$

$R \Rightarrow P$:

R_1: $M_1 \supseteq M_2$ & (M_1, M_1) » (M_1, M_2) & $M_1 \to M_1$(на основі P_1і P_3) $\Rightarrow$ $M_1 \to M_2$

R_2: $P_2 \equiv R_2$

R_3: $M_1 \to M_2$ & $M_3 \to M_3$ $\Rightarrow (M_1 \cup M_3) \to (M_2 \cup M_3)$

Таким чином, тепер ми можемо стверджувати, що множина функціональних залежностей є структурою чи точніше верхньою напіврешіткою.

Система D:

D_1: $M_1 \to M_2$ & $M_1 \to M_3 \Rightarrow M_1 \to M_2 \cup M_3$ – об'єднання — випливає з P_4

D_2: $M_1 \to M_2$ & $M_3 \cup M_2 \to M_4 \Rightarrow M_1 \cup M_3 \to M_4$ – псевдо-транзитивність

D_3: $M_1 \to M_2$ & $M_3 \subseteq M_2 \Rightarrow M_1 \to M_3$ – декомпозиція

Система D зручніша для практичних потреб, а її пункти можна вивести з P чи R.

4.2. Еквівалентність структур ФЗ. Мінімальна структура ФЗ.

Кажуть, що структура $\mathbf{F}^+$ є **замиканням** структури залежностей F, якщо будь-який елемент з F^+ можна отримати з F, користуючись системою аксіом Армстронга або її еквівалентами.

Кажуть, що структура ФЗ G **накриває** структуру ФЗ F, якщо $F^+ \subseteq G^+$.

Дві структури F і G називаються **еквівалентними**, якщо їх замикання рівні.

F~G, якщо $F^+ = G^+$.

Твердження:

Кожна структура функціональних залежностей F накривається деякою структурою G, де кожна права частина має не більше одного атрибуту. 91

Твердження випливає з властивості D_3.

Розглянемо приклад, на якому буде продемонстровано переходу до структур з одноатрибутними правими частинами:

Нехай задана залежність $A \to BCD$, еквівалентна їй система залежностей з одноатрибутними правими частинами така.

$A \to D$

$A \to B$

$A \to C$

Означення.

Кажуть, що структура функціональних залежностей F ***мінімальна***, якщо виконуються 3 властивості:

- Кожна права частина має один атрибут
- Ні для якої функціональної залежності не буде виконуватись

  Якщо $(X \to A) \in F$, то $(F \setminus \{X \to A\}) \sim F$

Це антинадлишковість залежностей, тобто з структури F не можна вилучити жодної залежності, не втративши еквівалентності з F.

- Антинадлишковість лівої частини

  Ні для якої функціональної залежності не буде виконуватись

  Якщо $(X \to A) \in F \;\&\; Z \subset A$, то $((F \setminus \{X \to A\}) \cup \{Z \to A\}) \sim F$

тобто з лівої частини залежності не можна вилучити жодного атрибута, не втративши еквівалентності з F.

Теорема:

Для кожної функціональної залежності F існує F′- мінімальна & F ~ F′

Доведення: доведемо дещо більше, а саме **як** побудувати F′. Рушимо по умовам мінімальності.

(1) попереднє твердження про одноатрибутні праві частини;
(2) досліджуємо кожну залежність на антинадлишковість (кожна літера – один атрибут). Нехай задана структура функціональних залежностей

$A \to B$; $A \to C$;

$B \to A$; $B \to C$; $C \to A$

Відкинемо $A \to B$, але отримати цю залежність з інших (на основі аксіом Армстронга) не можемо, отже вона не є надлишковою.

Візьмемо наступну залежність $A \to C$

$A \to C$: $A \to B$ & $B \to C \Rightarrow A \to C$ отже надлишкова

$B \to A$: $B \to C$ & $C \to A$ отже надлишкова

$B \to C$: $B \to A$ & $A \to C$ отже надлишкова

Залежність $C \to A$ не надлишкова, бо атрибут C зустрічається зліва тільки один раз.

Тепер аналізуємо надлишкові разом, бо вони взаємопов'язані, отже щоб отримати мінімальну структуру залежностей треба відкинути одну з надлишкових залежностей.

(3) антинадлишковість лівої частини

Розглянемо приклад

$AB \to C$; $A \to B$; $B \to A$ – структура F. Оскільки друга та третя залежності зліва мають по одному атрибуту, то треба перевірити тільки $AB \to C$. Для цього викреслимо A і отримаємо структуру G :

$B \to C$; $A \to B$; $B \to A$

Чи еквівалентні структури F та G?

Накриття в один бік виконується завжди, тобто якщо маємо $B \to C$, то також маємо і $AB \to C$, а от накриття в інший бік необхідно спеціально досліджувати

Чи можемо ми, починаючи з $B \to C$ в межах структури G отримати залежність $AB \to C$

$$\left.\begin{array}{l} B \to A \\ B \to AB \to C \end{array}\right\} \Rightarrow AB \to B \to C$$

Отже, A можна викреслити.

Тепер перевіримо, чи можна викреслити B (аналогічно). Відповідь – так.

Отже, атрибути A та B можна викреслити, але тільки один з них.

Власне доведення полягає в тому, щоб пройтись по всім залежностям структури і перевірити:

1) чи є кожна залежність одноатрибутною, якщо ні – то перетворити;
2) перевірити на надлишковість всі залежності та відкинути надлишкові;
3) перевірити на надлишковість лівої частини всі залежності з більш ніж одним атрибутом зліва, при потребі відкинути надлишкові атрибути.

4.3. Процедура мінімізації.

Нехай задана множина атрибутів X та структура функціональних залежностей F. Для цієї множини X по структурі F та аксіомам Армстронга можна побудувати замикання X^+ - множину атрибутів, які або належать множині X, або є залежними від X по структурі F та аксіомам Армстронга.

Нехай задано множину атрибутів X деякої реляції. Побудуємо її замикання по структурі F.

$X^0 = X$

$X^1 = X^0 \cup$ {атрибути, які можуть бути отримані з X^0 за один крок}

...

$X^{i+1} = X^i \cup$ { атрибути, які можуть бути отримані з X^0 за і кроків}

Якщо $X^к = X^{к+1} = X^+$, то процес завершується, тобто настала стабілізація. Якщо ж на деякому кроці $X^к$ зрівнюється з усією множиною атрибутів, то процес можна завершувати достроково.

Лема. Функціональна залежність $(X \rightarrow У) \in F^+$ тоді і тільки тоді, коли $У \subseteq X_F^+$.

Коментар: відповідь на питання $(X \rightarrow У) \in F^+$ має складність – експоненту, а відповідь на еквівалентне питання $У \subseteq X_F^+$ має степеневу складність.

Доведення. Нехай У= $A_1 \ldots A_к$, нехай також У $\subseteq X_F^+$, тоді $X \rightarrow A_i$ для всіх і за визначенням X_F^+, а це в свою чергу означає $X \rightarrow$У за властивістю Д1.

Навпаки, нехай $X \rightarrow$У, тоді $X \rightarrow A_i$ для всіх і за властивістю Д3, а це в свою чергу означає, що У $\subseteq X_F^+$ за визначенням X_F^+.

Розглянемо приклад побудови замикання атрибутів BD^+ на структурі:

$AB \rightarrow C$

$C \rightarrow A$

$BC \rightarrow D$

$ACD \rightarrow B$

$D \rightarrow EG$

$BE \rightarrow C$

$CG \rightarrow BD$

$CE \rightarrow AG$

Побудуємо замикання 2-х атрибутів: BD^+

$X^0 = \{B, D\}$

$X^1 = \{B, D, E, G\}$- на підставі $D \rightarrow EG$

$X^2 = \{B, D, E, G, C\}$ - на підставі $BE \rightarrow C$

$X^3 = \{B, D, E, G, C, A\}$ – на підставі $C \rightarrow A$

Отже, всі атрибути побудовані і $X^3 = X^+$

Можемо отримати ще один результат - оскільки BD^+ включає до свого складу всі атрибути, то BD може бути квазіключем, потрібно тільки перевірити всі його підмножини: чи не є вони квазіключами.

$B^+ = \{B\} \Rightarrow$ B не може бути квазіключем

$D^+ = \{DEG\}$

Отже, BD – це один з квазіключів. У даному прикладі досить легко побудувати ще кілька квазіключів. Оскільки є BC→D, то CD, BC – також є квазіключами. Оскільки є AB→C, то AB – теж є квазіключем. Легко бачити, що BE і CG – квазіключі. Таким чином, можемо зробити висновок, що всі атрибути реляції, яка розглядається є первинними.

Мінімізуємо дану структуру:

Перший крок мінімізації полягає у перетворенні заданої структури до еквівалентної їй, але з одноатрибутними правими частинами; для зручності посилань залежності отриманої структури перенумеруємо.

1) AB→C

2) C→A

3) BC→D

4) ACD→B

5) D→E

6) D→G

7) BE→C

8) CG→B

9) CG→D

10) CE→A

11) CE→G

Другий крок процедури мінімізації полягає у перевірці кожної залежності на надлишковість. Беремо першу залежність і відкидаємо її, на тому що залишилося будуємо AB^+. Якщо отримаємо C, то ця залежність була надлишковою, якщо ні – не надлишковою, або обов'язковою. Потім перша залежність повертається до списку, а друга відкидається і

т.д. Окрім того, для надлишкових залежностей визначаємо залежності, при наявності яких досліджувана залежність буде надлишковою (це так звані опорні залежності)

1) AB^+ = {AB} – обов'язкова;

2) C^+ = {C} – обов'язкова;

3) BC^+ = {BCA} – обов'язкова;

4) ACD^+ = {ACDEGB} – залежність надлишкова, якщо присутні 6 і 8 залежності; скорочений запис (6,8);

5) D^+ = {D, G}– обов'язкова;

6) D^+ = {D, E}– обов'язкова;

7) BE^+ = {B, E}– обов'язкова;

8) CG^+ = {C, G, D, A, E, B} – надлишкова, якщо присутні 4, 2 і 9 залежності, (2,4,9), але друга – обов'язкова, тому (4,9);

9) CG^+ = {C, G, B, D, A} — надлишкова, якщо присутні 8 і 3, тобто (8);

10) CE^+ = {C, E, A, …} - надлишкова при наявності 2, тобто ( ) – можемо зразу вилучити;

11) CE^+ = {C, E, A}– обов'язкова.

Розглянемо окремо тільки надлишкові залежності, їх опорні залежності подаємо в дужках.

4) ACD^+ = {ACDEGB} – (8)

8) CG^+ = {C, G, D, A, E, B} – (4,9)

9) CG^+ = {C, G, B, D, A} — (8)

Далі можливі 2 варіанти: викреслюємо 4 і 9, або тільки 8. Загалом вибір варіанту довільний, точніше не залежить від залежностей. З огляду на бажаність дещо продемонструвати на 3-ому кроці вибираємо до викреслення 8-му залежність.

Друга фаза алгоритму мінімізації завершена.

Наступний крок стосується тільки тих залежностей, де в лівій частині більше одного атрибута.

Беремо 1) AB→C

Викреслюємо атрибут A і будуємо замикання по B(що залишилось) на основі початкової структури $B_F^+ = \{B\}$, оскільки в замиканні C(права частина) не з'явилось, то A - обов'язковий атрибут.

Далі A повертаємо і викреслюємо B та будуємо замикання по A

$A^+ = \{A\}$, оскільки в замиканні C(права частина) не з'явилось, то B - обов'язковий атрибут.

Беремо 3) BC→D

$C^+ = \{C, A\}$ $B^+ = \{B\}$

Оскільки в замиканнях D не з'явилось, то обидва атрибута лівої частини є обов'язковими.

Беремо 4) ACD→B

Відкидаємо A і будуємо замикання по залишку

$CD^+ = \{C, D, E, G, \underline{A}, B\}$ – символ A надлишковий

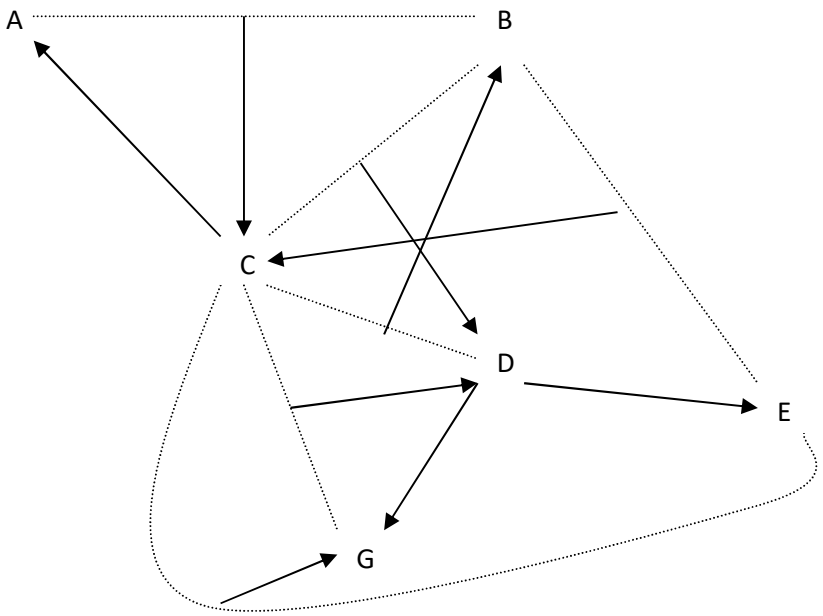
$AD^+ = \{ADEG\}$ $AC^+ = \{AC\}$

Якби трапилася ситуація, що можна відкинути ще один символ, то треба перевірити, чи можна вилучити комбінацію з кількох атрибутів.

Побудова декомпозиції початкової реляції до 3 НФ.

Спочатку побудуємо графічне представлення структури залежностей. Це допоміжна дія, яка потрібна буде для визначення порядку для застосування теореми Хіза.

98



B, C, D – первинні атрибути, які належать деякому квазіключу.

BD, CD – квазіключі.

AB^+ = {ABCD…} – комбінуємо BD чи CD, що дає можливість отримати всі

$\Rightarrow$ AB – квазіключ

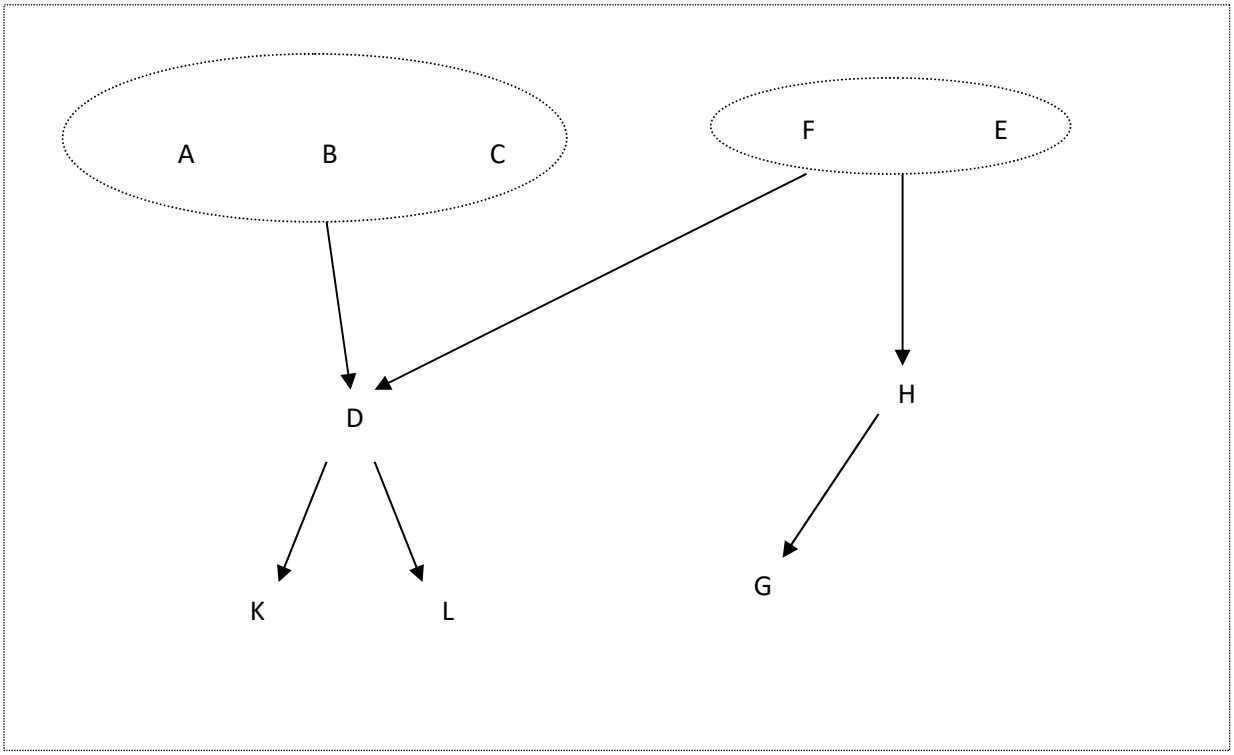
BE^+ = {BECA} – теж квазіключ

CG^+ - теж квазіключ

$\Rightarrow$ всі атрибути первинні $\Rightarrow$ реляція знаходиться в 3НФ. Тому в даному прикладі декомпозиція зайва.

Розглянемо інший приклад. Нижче структура задана у вигляді графічного представлення. Для того щоб отримати 3 НФ треба відсікати ієрархічні компоненти. Початкова реляція

R(ABCDEFGHKL).



Обираємо дві «найнижчі по ієрархії» залежності D→KL і H→G та виконуємо декомпозицію, користуючись теоремою Хіза: (спочатку D→KL)

R_1 (D, K, L) $\tilde{R}_1$ (ABCDFEHG)

R_2 (H, G) $\tilde{R}_2$ (ABCDEFH)

Реляції R_1 та R_2 вже знаходяться в 3НФ, а $\tilde{R}_2$ – ні; отже виконуємо ще одну декомпозицію, наприклад, по залежності FE→ DH

R_3 (F, E, D, H) $\tilde{R}_3$ (ABCEF)

Оскільки обидві реляції знаходяться в 3НФ, то процес декомпозиції завершено, а його результат: R_1 (D, K, L), R_2 (H, G), R_3 (F, E, D, H), R_4= $\tilde{R}_3$ (ABCEF).

Зауважимо, що в цьому прикладі ми проігнорували залежність ABC→D, бо використання ще і її призвело б до появи ще однієї реляції, а вторинний атрибут D мав би надлишковий характер.

4.3. Алгоритм перевірки з’єднання без втрат.

З теоретичної точки зору алгоритм перевірки з’єднання без втрат (АПЗБВ) є зайвим, бо фактично це є низка застосувань теореми Heath у зворотному порядку. Але на практиці він має досить широке застосування, оскільки досить часто вносяться проміжні зміни в логічний проект.

Входом для алгоритму є: початкова реляція $R(A_1,A_2,\ldots,A_n)$; сукупність реляцій, які були отримані в результаті декомпозиції$\{R_1,R_2,\ldots,R_k\}$, та структура функціональних залежностей F, а на виході маємо булевське значення TRUE чи FALSE. TRUE означає, що декомпозиція виконана коректно і втрати неможливі, а FALSE означає, що декомпозиція виконана некоректно і втрати *можливі*, але не обов’язкові. Все функціонування алгоритму відбувається на таблиці з n стовпчиків з іменами $A_1,A_2,\ldots,A_n$ та к рядочків, відмічених іменами реляцій $R_1,R_2,\ldots,R_k$. Виконання алгоритму складається з двох фаз: I-фаза ініціалізації таблиці, II-фаза перегляд та модифікації таблиці.

T	A_1	A_2	…	A_n
R_1				
R_2				
…	…	…	…	…
R_k				

I-фаза: якщо $A_j \in R_i$, тоді $T[i,j] := a_j$, інакше $T[i,j] := b_{ij}$; зауважимо, що заповнення клітинок таблиці виконується двома типами значень: типу a і типу b, при цьому значення типу a індексуються тільки одним індексом, що означає їх співпадання для одного стовпчика.

II-фаза: складається з трьох вкладених один в одного циклів. Перший (самий зовнішній) цикл забезпечує перебігання по всім залежностям структури F, він завершується при стабілізації вмісту таблиці, тобто протягом одного проходу циклу не вноситься змін в

клітинки таблиці. Другий цикл виконується для кожної залежності і кожний його прохід здійснюється для того чи іншого рядка таблиці: ведеться пошук рядків, у яких для досліджуваної залежності співпадають підкортежі для лівої частини залежності; якщо однакових підкортежів немає, то 2-ий цикл завершується і переходимо до наступної залежності; якщо співпадання є, то запам'ятовуємо номери відповідних рядків (виділяємо їх) і проводимо для виділених рядків ототожнення по значенням підкортежів (3-ій цикл), які відносяться до правої частини залежності: у напрямку згори вниз, якщо у стовпчику є тільки символи типу b; на символ типу a, якщо такий символ зустрівся хоча б в одному виділеному рядку стовпчика.

Після завершення 1-ого циклу аналізуємо таблицю: якщо є хоча б один рядок, який складається виключно з символів типу a, то вихід алгоритму TRUE, інакше – FALSE. Можливий достроковий вихід з циклів, тобто дострокове завершення алгоритму: якщо в якийсь момент отримаємо рядок таблиці, який повністю складається з символів типу a, то можна завершувати алгоритм з результатом TRUE, оскільки символи типу a мають пріоритет і не можуть бути змінені в подальшому.

Розглянемо приклади.

1) Нехай задано R(S,A,I,P); {R1(S,A); R2(S,I,P)}; F = {S→A; SI→ P}

	S	A	I	P
R1	a_1	a_2	b_{13}	b_{14}
R2	a_1	b_{22}	a_3	a_4

Після першої ж перевірки стовпчика S (для залежності S→A) отримаємо, що потрібно замінити b_{22} на a_2, а тоді 2-ий рядок складається з символів типу a, отже вихід алгоритму TRUE.

2)

Нехай задано R(A1,A2,A3,A4,A5);

102

{R1(A1,A4); R2(A1,A2); R3(A2,A5); R4(A3,A4,A5); R5(A1,A5)}

F ={A1→ A3; A2→ A3; A3→ A4; A4A5→ A3; A3A5→ A1}

Залишаємо для самостійного виконання.

Завдання для самостійної роботи. Виконати завдання 5.5.

5. Завдання для самостійної роботи.

Завдання для самостійної роботи будуть полягати у формулюванні запитів розділів 5.1. – 5.4. у термінах певних мов над схемою бази даних, яка задана у розділі 2.

Л(кл, прізв, орг, тел, наук_ступ);
П(кп, назва, тип, год, контр);
Г(кг, фак, каф, курс, к_чол);
Р(кл, кп,кг, день, ауд, пара);

5.1. Відносно прості запити.

1) Знайти прізвища та телефони лекторів з науковим ступенем доктора, які проводять другу пару у середу.
2) Знайти прізвища та організації лекторів, які читають с\к (тип) на 3-ому курсі.
3) Знайти назви предметів, які читаються *не* на факультеті кібернетика у середу.
4) Знайти назви предметів, які читаються на 4-ому курсі по кафедрі інформатики.
5) В які дні та в яких аудиторіях Іванчук проводить заняття на 2-ій парі.
6) Знайти коди груп та курси, у яких заняття проводяться не у п’ятницю.

5.2. Запити від супротивного

1) Знайти коди груп та курси, у яких Іванчук *не* проводить заняття у середу.
2) Знайти назви предметів, які Іванчук *не* читає в аудиторії 205.
3) Знайти назви предметів, які *не* читаються на факультеті кібернетика у середу.
4) Знайти прізвища та телефони лекторів з науковим ступенем доктора, які *не* проводять другу пару у середу.
5) Знайти коди груп та курси, у яких заняття *не* проводяться у п’ятницю.

103

5.3. Запити з множинним порівнянням

1) Визначити аудиторії в яких займаються усі групи 1-го курсу географічного факультету.

2) Знайти прізвища лекторів з наук. ступенем доктора, які проводять всі заняття принаймні у ті дні, що і Іванчук.

3) Визначити прізвища викладачів, що ведуть заняття в ті і тільки в ті дні, що і викладач Іванчук.

4) Визначити назви тих організацій, викладачі яких читають тільки на факультеті кібернетики.

5) Визначити назви предметів, що читаються усіма лекторами з телефоном 111-11-11.

5.4. Запити з вбудованими функціями

1) Знайти назви предметів, що читаються більш ніж двома викладачами з інституту кібернетики.

2) Знайти кількість груп та загальну кількість студентів у них, де учбовий процес повністю забезпечують викладачі Іванчук та Петренко.

3) Знайти прізвища викладачів, що читають на 1-ому курсі *тільки* ті предмети, які мають максимальну кількість годин.

4) Знайти загальну кількість студентів у групах факультету кібернетики, де ведуть заняття виключно 5 викладачів з університету.

5) Знайти назви предметів типу н\к, які на 4-ому курсі читає більш ніж один викладач з наук. ступенем доктора.

5.5. Завдання для мінімізації

У кожному завданні цього розділу задається реляція зі списком атрибутів і структура функціональних залежностей до неї. Потрібно здійснити мінімізацію структури ФЗ; у відповідності з отриманим результатом виконати декомпозицію реляції до сукупності реляцій у 3НФ, а потім перевірити її алгоритмом перевірки з'єднання без втрат.

1) R(A,B,C,D,E,F,G,H); { A->BC; C->DE; FB->DC;D->EFG; F->CE;}

2) R(A,B,C,D,E,F,G,H); { A->BCF; FB->CD; C->AE; AB->FG;}

3) R(A,B,C,D,E,F,G); { A->BC; BC->DF; D->BE; DE->B;}

4) R(K,L,M,N,O,P,Q);{ P->MNO; K->LM; M->NO; LP->N; N->OP;}

5) R(S,T,Q,U,Z,Y,X); { Z->ST; ST->Q; Q->U; QU->SY; Z->US;}

Література.

1. Пасічник В.В., Резніченко В.А. Організація баз даних та баз знань. – К.:Видавнича група BHV, 2006. – 384 с.

2. Дейт К. Введение в системы баз данных. 6-е изд. – К.: Диалектика. 1998. – 784 с.

3. Мартин Дж. Организация баз данных в вычислительных системах. Мир. 1980.

4. Ульман Дж. Основы систем баз данных. М. Финансы и статистика. 1983.- 334 с.

5. Дрибас В.П. Реляционные модели баз данных. Изд-во БГУ. Минск. 1982.

6. Конноли Т., Бегг К., Страчан А. Базы данных: проектирование, реализация и сопровождение. Теория и практика. Издательский дом «Вильямс». 2000. – 1120 с.

Зміст

Передмова. 3

1. Вступ. Основні поняття. 3

1.1. Поняття інформаційної системи. 3

1.2. Поняття бази даних. 4

1.3. Класифікація АІС. 7

1.4. Класифікація запитів. 9

1.5. Інформаційна модель концептуального рівня. ER – модель. 12

2. Реляційна модель баз даних. Мови запитів. 16

2.1. Реляційна алгебра. 19
2.1.1. Реляційна алгебра Кодда. 19
2.1.2. Алгебра вибору (Дрібаса) 29
2.2. Реляційне числення. Мова ALPHA. 31
2.2.1. Реляційне числення. 32
2.2.2. Мова ALPHA 35
2.2.3. Алгоритм редукції Кодда 43
2.4. Теорія відображень, мова SQL. 44
2.4.1. Теорія відображень як математична основа мови SQL. 44
2.4.2. Запити в стилі класичного SEQUEL2. 46
2.4.3. Використання вбудованих функцій. 55
2.4.4. Запити дії. 56
2.4.5. SQL ACCESS. 57
2.4.6. Методики перетворення запитів з прямими множинними порівняннями. 61
2.5. Мова Query-By-Example(QBE) 64
2.5.1. Приклади запитів на мові QBE. 64
2.5.2. Використання вбудованих функцій. 68
2.5.3. Запити дії. 69
2.5.4. Ієрархічні запити в QBE. 70
2.5.5. QBE Paradox та DataBase Desktop. 73
3. Залежності та нормальні форми реляцій. 78
3.1. Функціональні залежності (ФЗ). Квазіключ, ключ, суперключ, «чужі ключі». 78
3.2. Первинні та вторинні атрибути. 1-а нормальна форма (1НФ); Функціонально повна залежність (ФПЗ), 2НФ, 2НФп; Теорема Хіза. 80
3.3. Транзитивна функціональна залежність в сенсі реляційного підходу, 3НФ, 3НФп чи нормальна форма Бойса-Кодда. Декомпозиція реляцій до 3НФ. 83

106

3.4. Багатозначна залежність. 4НФ. Залежність по з’єднанню без втрат. 5НФ. 85

4. Логічне проектування баз даних 88

4.1. Аксіоми Армстронга. 88

4.2. Еквівалентність структур ФЗ. Мінімальна структура ФЗ. 91

4.3. Процедура мінімізації. 94

4.3. Алгоритм перевірки з’єднання без втрат. 101

5. Завдання для самостійної роботи. 103

5.1. Відносно прості запити. 103

5.2. Запити від супротивного 103

5.3. Запити з множинним порівнянням 104

5.4. Запити з вбудованими функціями 104

5.5. Завдання для мінімізації 104

Література. 105

107