Козловская Валентина Петровна

«Организация баз данных и знаний»

Конспект лекций, вторая редакция

2014

Конспект лекцій з дисципліни «Організація баз даних та знань» призначений для студентів ІІІ курсу денної та заочної форми навчання, що навчаються за спеціальністю «Інформаційні та управляючі системи та технології», рівень підготовки – бакалавр.

Автор конспекту – доцент кафедри інформаційних технологій, к.ф.-м.н. Козловська Валентина Петрівна.

Зміст

ВСТУП

Основні ідеї сучасної інформаційної технології базуються на концепції баз даних (БД). Відповідно до цієї концепції, основою інформаційної технології є дані, які повинні бути організовані в БД із метою адекватного відображення реального світу, що змінюється, і задоволення інформаційних потреб користувачів.

Збільшення обсягу й структурної складності даних, що зберігаються, розширення кола користувачів інформаційних систем (ІС) висунуло вимогу створення зручних загальносистемних засобів інтеграції даних, що зберігаються, і засобів керування цими даними. Це й привело до появи наприкінці 60-х років минулого століття перших промислових систем управління базами даних (СУБД) – спеціалізованих програмних засобів, призначених для організації й ведення БД.

Можна стверджувати, що поява баз даних стала найважливішим досягненням в галузі програмного забезпечення. Бази даних лежать в основі інформаційних систем, і це докорінно змінило характер роботи багатьох організацій. З моменту своєї появи технологія баз даних стала каталізатором багатьох значних досягнень в галузі програмного забезпечення.

За останні 40 років в галузі теорії систем баз даних було проведено ряд продуктивних досліджень. Отримані результати цілком можна вважати найбільш важливим досягненням інформатики за цей період. Зокрема, в останні роки розвиток технології баз даних призвів до створення досить потужних і зручних в експлуатації систем. Завдяки цьому системи баз даних стали доступні широкому колу користувачів. Але, на жаль, уявна простота таких систем сприяла тому, що користувачі стали самостійно створювати бази даних і додатки до них, не маючи достатніх знань про методи проектування ефективно працюючих систем, що часто приводило до непродуктивних витрат ресурсів і неякісних результатів. Викликана цим незадоволеність користувачів стала причиною виникнення відомої «кризи програмного забезпечення», або так званої «депресії програмного забезпечення», наслідки якої не усунуті й понині.

Тому метою даного курсу є навчання студентів теоретичним і практичним основам теорії баз даних, зокрема продуктивної методології проектування баз даних. Проектування баз даних складається із трьох стадій: концептуальної, логічної й фізичної. Перша стадія передбачає створення концептуальної моделі даних, що не залежить від яких-небудь фізичних характеристик. Ця частина проектування в даному курсі розглядається досить конспективно, оскільки вона найбільше повно розглядається в курсі «Проектування інформаційних систем», який вивчається у другому семестрі ІІІ курсу. У другій стадії, призначенням якої є створення логічної моделі даних, концептуальна модель піддається

доробці за допомогою видалення елементів, які не можуть бути реалізовані в системах з обраною моделлю даних – реляційних системах у більшості випадків в цей час. Реляційна модель даних досить повно розглядається в даному курсі. У третій стадії проектування логічна модель даних перетвориться у фізичний проект, призначений для реалізації в середовищі конкретної цільової системи управління базами даних (СУБД). Для придбання навичок роботи з конкретної СУБД у даному курсі розглядається робота з клієнт-серверною СУБД MS SQL Server 2000 (2008).

В 80-і роки завдяки досягненням в області штучного інтелекту з’являється багато систем, що базуються на використанні знань. Систему, що забезпечує створення, ведення й застосування баз знань, можна розглядати як інструментальну систему, яка називається системою управління базами знань (СУБЗ), або як прикладну систему з конкретною прикладною базою знань.

Існує тісний взаємозв'язок між технологією БД і систем баз даних (СБД) – з одного боку, і технологією систем баз знань (СБЗ) з іншої. Виникла тенденція «інтелектуалізації» систем БД. На зовнішньому рівні їхньої архітектури реалізуються різноманітні семантичні моделі даних, створюються «дружелюбні» інтерфейси для користувачів. Разом з тим, традиційні СУБД є необхідною складовою частиною інструментарію керування даними в СБЗ. Деякі аспекти роботи систем управління базами знань також розглядаються в даному курсі.

1 СИСТЕМИ БАЗ ДАНИХ. ОСНОВНІ ПОНЯТТЯ Й АРХІТЕКТУРА

1.1 Поняття інформації, даних, знань

У реальній дійсності різноманітні події, процеси, люди, предмети зв'язані з формуванням, передаванням, опрацюванням і споживанням інформації. Інформація присутня в мові людей, різних текстах, книжках, рисунках, кресленнях, колонках цифр, звукових та світлових сигналах, показах годинників, термометрів, енергетичних та силових імпульсах тощо. Ніщо живе у природі не може існувати і розвиватися, не зберігаючи свій генетичний код, не сприймаючи інформацію від навколишнього світу через органи чуття і не опрацьовуючи її за допомогою нервової системи. Реакція на повідомлення (тобто опрацювання інформації) є обов'язковою складовою частиною чуттєвого сприйняття реального світу.

Будь-які процеси суспільного життя – виробничі, господарські, керівні, науково-дослідні, суспільно-політичні, демографічні та інші – знаходять відображення в інформаційних процесах: люди отримують інформацію, запам'ятовують і нагромаджують її, опрацьовують інформацію, вирішуючи різноманітні завдання, передають інформацію телефоном, поштою, комп'ютерною мережею та іншими засобами зв'язку.

Проте, незважаючи на те, що з інформацією ми стикаємося щоденно, строгого і загальновизначеного означення інформації як наукової категорії не існує. Вона відноситься до так званих неозначуваних понять – понять, які не можна визначити через інші, простіші поняття, а необхідно вводити за допомогою пояснень, уточнень, прикладів.

В найширшому розумінні інформація трактується наступним чином.

Інформація – це відомості, пояснення, знання про всілякі об'єкти, явища, процеси реального світу (К.О. Соловйова).

Доволі розповсюдженим є погляд на інформацію як на ресурс, аналогічний енергетичним, матеріальним, трудовим і грошовим ресурсам. Ця точка зору відображається у наступному трактуванні інформації.

Інформація – нові відомості, які дозволяють поліпшити процеси, пов'язані з перетворенням речовин, енергії та самої інформації.

Інформацію не можна відокремити від процесу інформування. В основі цього процесу лежить взаємозалежність пари об'єктів – джерела і споживача інформації.

Джерелами інформації, насамперед, є природні об'єкти: люди, тварини, рослини, планети та ін. Разом із цим, у міру розвитку науки і техніки, джерелами інформації стають наукові експерименти, машини, апарати, технологічні процеси. Значний також перелік об'єктів, що є споживачами інформації: люди, тварини, рослини, різноманітні технічні пристрої тощо.

Погляд на інформацію з точки зору її споживачів окреслюється у наступному понятті інформації.

Інформація – це нові відомості, прийняті, зрозумілі і оцінені її користувачем як корисні.

Інформатика використовує своє власне тлумачення терміну «інформація», яке є значно вужчим, ніж розглянуті вище. Це зумовлено тим, що при визначенні цієї категорії, інформатика не може базуватися на таких поняттях як знання, відомості, усунення невизначеності. Засоби комп'ютерної техніки мають здатністю опрацьовувати інформацію автоматично, без участі людини, і про жодні знання або незнання тут мова йти не може. Ці засоби можуть працювати зі штучною, абстрактною і, навіть, із хибною інформацією, яка не має об'єктивного відображення ні у природі, ні у суспільстві.

Враховуючи зазначену специфіку комп'ютерної техніки, інформатика вилучає змістовні аспекти поняття інформації і використовує погляд на інформацію як на предмет певної діяльності. Тому з'являється наступне поняття інформації [11]:

Інформація – це відомості, які є об'єктом збирання, реєстрації, збереження, передавання і перетворення.

Основна маса інформації збирається, передається і опрацьовується у знаковій формі – числовій, текстовій, табличній, графічній і т.ін. Знакове подання інформації інформатика пов'язує з поняттям «дані».

Дані – це відомості, подані у певній знаковій формі, придатній для передавання, інтерпретації та опрацювання людиною або автоматичним пристроєм.

Взаємозв'язок між інформацією, даними і методами оперування ними характеризується низкою властивостей.

Інформація має динамічний характер. Вона не є статичним об'єктом, а динамічно змінюється й існує тільки у момент взаємодії даних і методів, тобто в момент протікання інформаційного процесу. Решту часу вона перебуває у стані даних.

Дані надають інформацію лише в момент їх використання. Це означає, що збережувані дані залишаються лише даними доти, поки їх не використано в тих чи інших методах деяким суб'єктом інформаційного процесу (людиною або автоматичним пристроєм). Якщо дані зовсім не використовуються, то говорять, що вони мають нульову інформативність, і вважають такі дані інформаційним шумом. Дані, що використовуються, називають інформативними.

Рівень інформативності даних залежить від ступеня адекватності методів, що застосовуються в інформаційних процесах.

Дані сприймаються їх отримувачем як певна змістовна інформація лише в тому випадку, коли в його «пам'яті» закладені поняття і моделі, що дозволяють зрозуміти зміст отриманих відомостей.

Коли дані опрацьовуються певними методами або евристиками, то на їх основі можна встановити залежності, послідовності чи висновки, які дозволяють здійснити підтримку процесу прийняття рішення. Такі дані називають знаннями. Рішення може прийматися як експертом предметної галузі (такі знання називають суб'єктизованими), так і комп'ютером (такі знання називають комп'ютеризованими).

Зв'язок між інформацією, даними та знаннями можемо простежити за рис. 1.1.

Рис. 1.1 – Зв'язок між інформацією, даними та знаннями.

Прикладом інформації є технічний паспорт автомобіля, даних – матриця споживання автомобілем бензину при різних швидкостях, знаннями – рекомендації щодо швидкості руху для різних типів доріг з метою оптимального споживання бензину.

У свою чергу дані подаються у формі:

- структури даних;
- баз даних;
- сховищ даних;
- просторів даних.

Знання подаються у вигляді:

- баз знань;
- дедуктивних баз даних;
- експертних систем тощо.

Форми подання даних утворюються як пари структур даних та функцій їх опрацювання.

1.2. Структури даних

У найперших прикладних розробленнях та мовах програмування (50-ті роки XX століття) використовувалися дані, призначені для розв’язування розрахункових задач.

Будь-які дані, що використовуються для моделювання фізичних об’єктів, явищ, процесів тощо, мають свої типи даних. Як правило, типи даних діляться на три групи:

- прості типи даних;
- структуровані типи даних;
- вказівникові типи даних.

Ранні мови програмування були призначені для розв’язування науково-технічних обчислювальних завдань. У цих мовах використовувалися тільки прості типи даних та масиви. Починаючи з кінця 60-х років XX століття починають інтенсивно використовувати для розв’язування так званих необчислювальних завдань, пов’язаних з опрацюванням різного роду документів. Для опису аналогічних подань даних в необчислювальних завданнях вводиться ряд нових понять.

Елемент даних (поле) – найменша одиниця поіменованих даних.

Запис – поіменована сукупність елементів даних (полів).

Екземпляр запису – поточне значення елементів запису.

Файл – поіменована сукупність всіх екземплярів записів заданого типу.

Дані, організовані у вигляді структур чи масивів, недоцільно застосовувати у задачах опрацювання [10], оскільки вони допускають надлишковість, неактуальність тощо.

Тому наступним кроком було створення баз даних.

1.3. База даних

Наведемо декілька найбільш поширених визначень бази даних (БД).

База даних – сукупність екземплярів різних типів записів і відношень між записами та елементами.

База даних – централізоване сховище даних, що забезпечує зберігання, доступ, первинне опрацювання і пошук інформації [12].

Базу даних можна визначити як сукупність взаємозв'язаних даних (прості чи складені типи), що зберігаються разом на одному носії та описують якусь предметну область за наявності такої мінімальної надмірності, яка допускає їх використання оптимальним чином для одного або декількох застосувань. Розрізняють ієрархічні, мережеві, реляційні, часові (темпоральні), постреляційні (об'єктно-орієнтовані, об’єктно-реляційні), розподілені та багатовимірні бази даних.

Використання бази даних припускає роботу з нею декількох прикладних програм (застосувань), що вирішують завдання різних користувачів.

Одним з перших фундаментальних підручників, де висловлені основні принципи побудови систем керування базами даних, є книга К.Дж. Дейта [13].

СУБД. Термін *база даних* позначає сукупність даних, призначених для спільного використання. Треба, однак, розрізняти терміни *документальна база даних* (сукупність довільних текстових документів) і *фактографічна база даних* (множина відомостей, що зберігаються в інформаційній системі й задовольняють фіксованій сукупності форматів). Поняття *система управління базами даних* (СУБД) відноситься до набору засобів програмного забезпечення, необхідних для використання фактографічних баз даних.

В фундаментальній книзі Т. Коннолі [9] наводиться наступне визначення бази даних:

База даних. Набір логічно зв'язаних даних (і опис цих даних), який спільно використовується, і призначений для задоволення інформаційних потреб організації.

Щоб глибше вникнути в суть цього поняття, розглянемо його визначення більш уважно. База даних – це єдине, велике сховище даних, що однократно визначається, а потім використовується одночасно багатьма користувачами – представниками різних підрозділів. Замість розрізнених файлів з надлишковими даними тут всі дані зібрані разом з мінімальною часткою надмірності. База даних уже не належить якому-небудь єдиному відділу, а є загальним корпоративним ресурсом. Причому база даних зберігає не тільки робочі дані цієї організації, але і їхній опис. Із цієї причини базу даних ще називають набором інтегрованих записів із самоописом. У сукупності опис даних називається *системним каталогом* (system catalog), або *словником даних* (data dictionary), а самі елементи опису прийняті називати *метаданими* (meta-data), тобто "даними про даний". Саме наявність самоопису даних у базі даних забезпечує в ній *незалежність програм від даних* (program-data independence).

Підхід, заснований на застосуванні баз даних, де визначення даних відділене від додатків, дуже схожий на підхід, що використовується при розробці сучасного програмного забезпечення, коли поряд із внутрішнім визначенням об'єкта існує його зовнішнє визначення. Користувачі об'єкта бачать тільки його зовнішнє визначення й не замислюються над тим, як він визначається й функціонує. Одна з переваг такого підходу, а саме *абстрагування даних* (data abstraction), полягає в тому, що можна змінити внутрішнє визначення об'єкта без яких-небудь наслідків для його користувачів, за умови, що зовнішнє визначення об'єкта залишається

незмінним. Аналогічним образом, у підході з використанням баз даних структура даних відділена від додатків і зберігається в базі даних. Додавання нових структур даних або зміна існуючих ніяк не впливає на додатки, за умови, що вони не залежать безпосередньо від компонентів, що змінюються. Наприклад, додавання нового поля в запис або створення нового файлу ніяк не вплине на роботу наявних додатків. Однак видалення поля з файлу, який використовується додатком, вплине на цей додаток, а тому його також буде потрібно відповідним чином модифікувати.

І, нарешті, потрібно пояснити останній термін з визначення бази даних, а саме поняття "логічно зв'язаний". При аналізі інформаційних потреб організації треба виділити сутності, атрибути й зв'язки. *Сутністю* (entity) називається окремий тип об'єкта (людина, місце або річ, поняття або подія), який потрібно представити в базі даних. *Атрибутом* (attribute) називається властивість, що описує деяку характеристику розглянутого об'єкта; *зв'язок* (relationship) – це те, що поєднує кілька сутностей.

Наприклад, на рис. 1.5 показана так звана діаграма "сутність-зв'язок", або ER-діаграма (Entity-Relationship – ER), для деякої частини навчального проекту *DreamHome*.

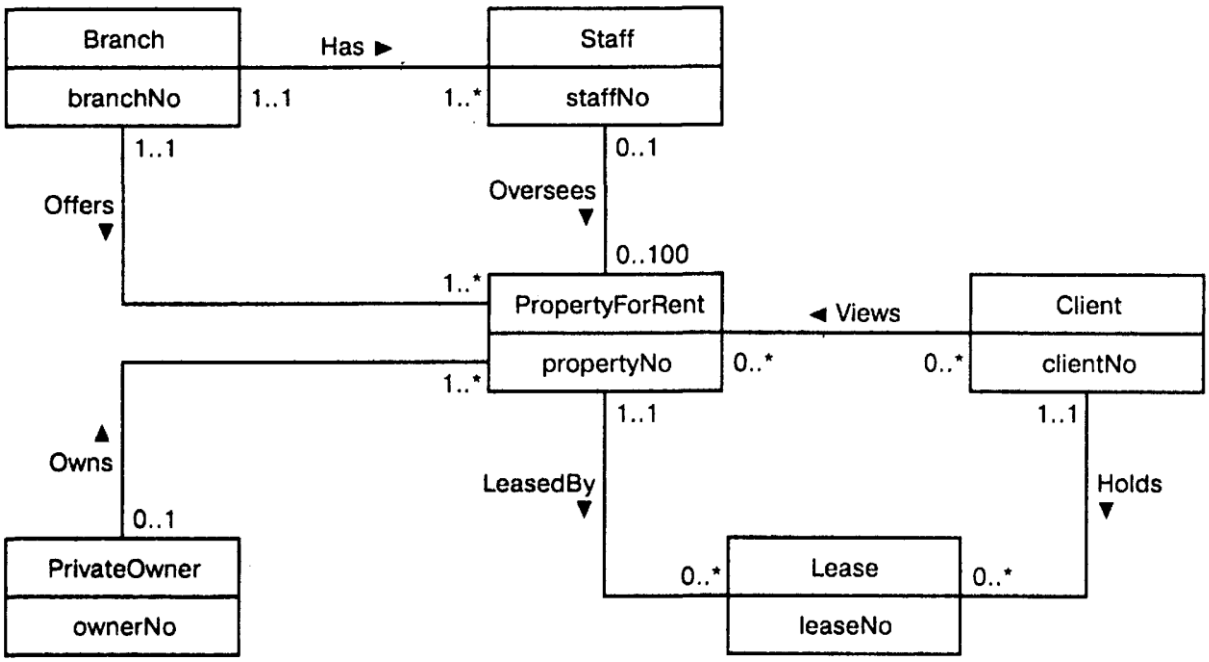


Рис. 1.2 Приклад діаграми "сутність-зв'язок"

Ця ER-діаграма складається з наступних компонентів:

- шести сутностей (які позначені прямокутниками): `Branch` (Відділення), `Staff` (Працівник), `PropertyForRent` (об'єкт, який здається в оренду), `Client` (Клієнт), `PrivateOwner` (Власник об'єкта нерухомості) і `Lease` (Договір оренди);

- семи зв'язків (які позначені стрілками): `Has` (Має), `Offers` (Пропонує), `Oversees` (Керує), `Views` (Оглядає), `Owns` (Володіє), `LeasedBy` (Здається в оренду) і `Holds` (Орендує);
- шести атрибутів, які відповідають кожній сутності: `branchNo` (Номер відділення), `staffNo` (Табельний номер працівника), `propertyNo` (Номер об'єкта, який здається в оренду), `clientNo` (Номер клієнта), `ownerNo` (Номер власника) і `leaseNo` (Номер договору оренди).

Подібна база даних представляє сутності, атрибути й логічні зв'язки між об'єктами. Інакше кажучи, база даних містить логічно зв'язані дані. Більш докладно модель типу "сутність-зв'язок" розглядається в дисципліні “Проектування ІС”.

1.4 Система управління базами даних – СУБД

СУБД. Програмне забезпечення, за допомогою якого користувачі можуть визначати, створювати й підтримувати базу даних, а також здійснювати до неї контрольований доступ.

СУБД – це програмне забезпечення, що взаємодіє із прикладними програмами користувача й базою даних і має перераховані нижче можливості.

- Дозволяє створити базу даних, що звичайно здійснюється за допомогою мови визначення даних (МВД або DDL — Data Definition Language). Мова DDL надає користувачам засоби вказівки типу даних і їхньої структури, а також засоби завдання обмежень для інформації, яка зберігається в базі даних.
- Дозволяє вставляти, обновляти, видаляти й витягати інформацію з бази даних, що звичайно здійснюється за допомогою мови маніпулювання даними (ММД або DML – Data Manipulation Language). Наявність централізованого сховища всіх даних і їхніх описів дозволяє використовувати мову DML як загальний інструмент організації запитів, яку іноді називають мовою запитів (query language). Наявність мови запитів дозволяє усунути властивим файловим системам обмеження, при яких користувачам доводиться мати справа тільки з фіксованим набором запитів або постійно зростаючою кількістю програм, що породжує інші, більше складні проблеми керування програмним забезпеченням. Найпоширенішим типом непроцедурної мови запитів є мова структурованих запитів (Structured Query Language – SQL), що у цей час визначається спеціальним стандартом і фактично є обов'язковою мовою для будь-

яких реляційних СУБД (SQL вимовляється або по буквах "S-Q-L", або як мнемонічне ім'я "SeeQuel".)

- Надає контрольований доступ до бази даних за допомогою перерахованих нижче засобів:
  - системи забезпечення захисту, що запобігає несанкціонований доступ до бази даних з боку користувачів;
  - системи підтримки цілісності даних, що забезпечує несуперечливий стан збережених даних;
  - системи керування паралельною роботою додатків, що контролює процеси їхнього спільного доступу до бази даних; при колективному режимі можливе використання загальних фізичних даних. Це вимагає підтримки узгодженості тих самих даних при роботі різних користувачів. Типові приклади неузгодженості виникають при неправильному керуванні одночасними модифікаціями. Такі проблеми, як втрата змін і видача помилкової інформації, розглядаються нижче (у розділах, присвячених захисту й цілісності). Прикладами можуть слугувати продаж більшого числа товарів, чим є в наявності, або продаж декількох квитків на одне місце. Гарна СУБД повинна забезпечувати механізми контролю цілісності для запобігання можливих неузгодженостей при використанні бази даних;
  - системи відновлення, що дозволяє відновити базу даних до попереднього несуперечливого стану, порушеного в результаті збоїв апаратного або програмного забезпечення;
  - доступного користувачам каталогу, що містить опис інформації, що зберегається в базі дані.

1.4.1 Стандарти СУБД

Сучасні СУБД розроблені з метою усунення недоліків файлових систем. Наступні принципи повинні дотримуватися при розробці СУБД:

1. Незалежність даних.
2. Універсальність. СУБД повинна мати потужні засоби підтримки концептуальної моделі даних для відображення логічних представлень користувачів.
3. Сумісність. СУБД повинна зберігати працездатність при розвитку програмного й апаратного забезпечення.
4. Ненадмірність даних. На відміну від файлових систем база даних повинна являти собою єдину сукупність інтегрованих даних.

5. Захист даних. СУБД повинна забезпечувати захист від несанкціонованого доступу.
6. Цілісність даних. СУБД повинна запобігати порушення бази даних користувачами.
7. Керування одночасною (паралельною) роботою. СУБД повинна охороняти базу даних від неузгодженостей у режимі колективного користування. Для забезпечення узгодженого стану бази даних всі запити користувачів (транзакції) повинні виконуватися в певному порядку.

Незалежність даних – найбільш важлива властивість, тому що вона впливає на наявність всіх інших властивостей, таких, як інтеграція даних, ненадмірність, колективне користування й можливість забезпечення захисту й цілісності.

До цих принципів необхідно додати наступне:

а) СУБД повинна бути універсальної, тобто вона повинна підтримувати різні моделі даних на єдиній логічній і фізичній основі. Це дозволяє забезпечити вимоги й переваги різних користувачів.

б) СУБД повинна підтримувати як централізовані, так і розподілені бази даних. У наш час широке застосування находить організація обчислювальних мереж і розподілена обробка даних. СУБД для територіально розосередженої організації, у якій важлива локальна автономність даних, повинна бути системою управління розподіленими базами даних (СУРБД). Подібні системи повинні вирішувати такі задачі, як розміщення даних, обробка розподілених запитів, керування одночасною роботою різних вузлів мережі, оптимізація виконання запиту з метою мінімізації обсягу даних, які передаються між окремими вузлами мережі.

1.5 Архітектура СУБД

1.5.1 Трирівнева архітектура ANSI-SPARC

Перша спроба створення стандартної термінології й загальної архітектури СУБД була почата в 1971 році групою DBTG. Вона була створена після конференції CODASYL (Conference on Data Systems and Languages — Конференція по мовах і системам даних), що пройшла в цьому ж році. Група DBTG визнала необхідність використання дворівневого підходу, побудованого на основі використання системного представлення, тобто схеми (schema), і представлення користувачів, тобто підсхем (subschema).

Концепції багаторівневої архітектури СУБД є основою сучасної технології баз даних. Ці ідеї зв'язують із опублікованим в 1975 р. звітами робочої групи по базах даних Комітету із плануванню стандартів Американського національного інституту стандартів (ANSI/X3/SPARC). У цій роботі була запропонована узагальнена трирівнева модель архітектури СУБД, що включає концептуальний, внутрішній і зовнішній рівні архітектури. Така модель описує архітектуру будь-якої системи з тим лише застереженням, що в кожній конкретній СУБД які-небудь компоненти або функції такої моделі можуть мати звироднілий характер.

Запропонована у звіті архітектурна модель не тільки дозволяє конструктивно описати погляд зсередини системи на процеси керування даними, що відбуваються в ній. Відповідно до такого підходу до архітектури вдається сформувати й більш диференційовану точку зору на функції персоналу адміністрування даними в системі баз даних.

Концептуальний рівень архітектури ANSI/SPARC служить для підтримки єдиного погляду на базу даних, загального для всіх її додатків і в цьому сенсі незалежного від них. Саме в середовище концептуального рівня при проектуванні бази даних відображається інфологічна модель предметної області системи бази даних. Представлення бази даних на концептуальному рівні називається *концептуальною базою даних*, а опис такого представлення – *концептуальною схемою бази даних*.

Механізми *внутрішнього рівня* архітектури служать для підтримки представлення бази даних у середовищі зберігання – внутрішньої бази даних або *бази даних, що зберігається*. Це – єдиний архітектурний рівень, де база даних у дійсності представлена повністю в "матеріалізованому" вигляді. На всіх інших рівнях у процесі виконання різних операцій над базою даних з'являються й зникають лише окремі екземпляри або множини екземплярів її об'єктів. Опис представлення бази даних на внутрішньому рівні архітектури називається *внутрішньою схемою*, або *схемою зберігання*.

Нарешті, користувачі бази даних мають справу з представленням бази даних на *зовнішньому рівні*, з так званими *зовнішніми базами даних*. Описи таких представлень називаються *зовнішніми схемами*. У системі бази даних може одночасно підтримуватися кілька зовнішніх схем для різних груп користувачів.

У цьому випадку для нас найбільш фундаментальним моментом у цих і наступних звітах дослідницьких груп є визначення трьох рівнів абстракції, тобто трьох різних рівнів опису елементів даних. Ці рівні формують трирівневу архітектуру, що охоплює зовнішні, концептуальний і внутрішній рівні, як показано на рис. 1.3. Рівень, на якому дані сприймаються користувачами, називається зовнішнім рівнем (external level), тоді як СУБД і операційна система сприймають дані на

внутрішньому рівні (internal level). Саме на внутрішньому рівні дані реально зберігаються. Концептуальний рівень (conceptual level) представлення даних призначений для відображення зовнішнього рівня на внутрішній і забезпечення необхідної незалежності друг від друга.

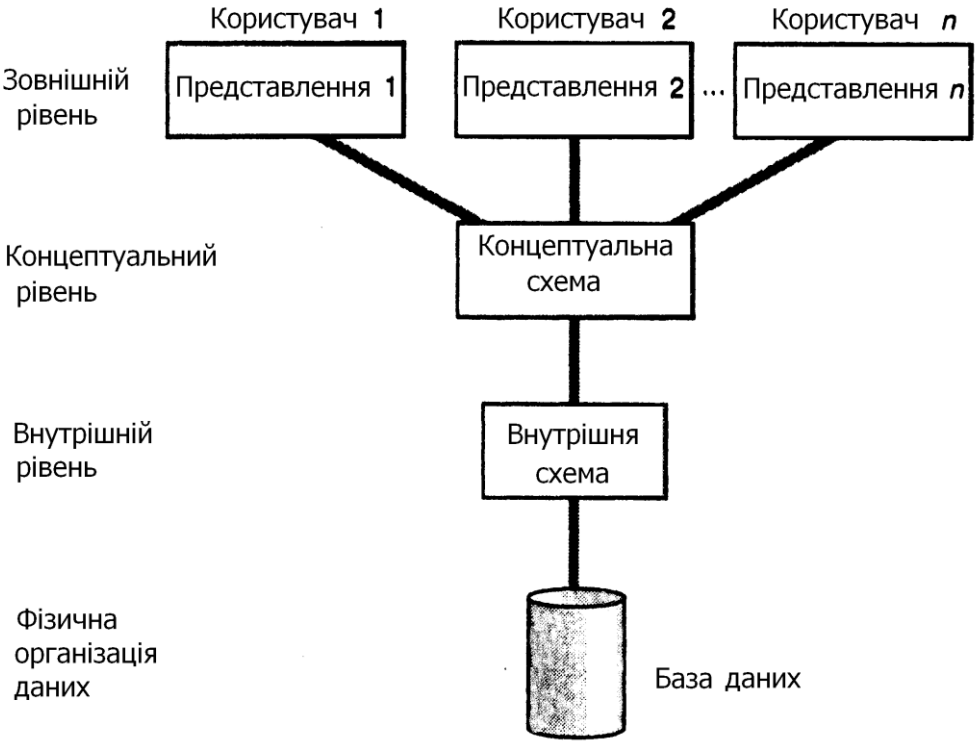


Рис. 1.3. Трирівнева архітектура ANSI-SPARC

Ціль трирівневої архітектури полягає у відділенні представлення користувачів бази даних від її фізичного представлення. Нижче перераховано кілька причин, по яких бажано виконати такий поділ.

- Кожний користувач повинен мати можливість звертатися до тих самих даних, реалізуючи своє власне представлення про них. Кожний користувач повинен мати можливість змінювати своє представлення про дані, причому ця зміна не повинне мати вплив на інших користувачів.
- Користувачі не повинні безпосередньо мати справу з такими подробицями фізичного зберігання даних у базі, як індексування й хешірування. Інакше кажучи, взаємодія користувача з базою не повинна залежати від особливостей зберігання в ній даних.
- Адміністратор бази даних (АБД) повинен мати можливість змінювати структуру зберігання даних у базі без здійснення впливу на представлення користувачів.

- Внутрішня структура бази даних не повинна залежати від таких змін фізичних аспектів зберігання інформації, як перехід на новий пристрій зберігання.
- АБД повинен мати можливість змінювати концептуальну структуру бази даних без якого-небудь впливу на всіх користувачів.

Зовнішній рівень

Зовнішній рівень. Представлення бази даних з погляду користувачів. Цей рівень описує ту частину бази даних, що відноситься до кожного користувача.

Зовнішній рівень складається з декількох різних зовнішніх представлень бази даних. Кожний користувач має справу з представленням "реального світу", вираженим у найбільш зручній для нього формі. Зовнішнє представлення містить тільки ті сутності, атрибути й зв'язки "реального світу", які цікаві користувачеві. Інші сутності, атрибути або зв'язки, які йому нецікаві, також можуть бути представлені в базі даних, але користувач може навіть не підозрювати про їхнє існування.

Крім цього, різні представлення можуть по-різному відображати ті самі дані. Наприклад, один користувач може переглядати дати у форматі (день, місяць, рік), а інший – у форматі (рік, місяць, день). Деякі представлення можуть включати *похідні дані*, або *дані, що обчислюваються*, які не зберігаються в базі даних, а створюються в міру потреби. Наприклад, у багатьох проектах для користувачів становлять інтерес дані про вік співробітників (пацієнтів, студентів). Однак навряд чи варто зберігати ці відомості в базі даних, оскільки в такому випадку їх довелося б щодня обновляти. Замість цього в базі даних зберігаються дати народження співробітників, а вік обчислюється засобами СУБД при виявленні відповідного посилання. Представлення можуть також включати комбіновані або похідні дані з декількох об'єктів. Більш докладно представлення розглядаються в розділі 5.6.

Концептуальний рівень

Концептуальний рівень. Узагальнююче представлення бази даних. Цей рівень описує те, які дані зберігаються в базі даних, а також зв'язки, що існують між ними.

Проміжним рівнем у трирівневій архітектурі є концептуальний рівень. Цей рівень містить логічну структуру всієї бази даних (з погляду АБД). Фактично це повне подання вимог до даних з боку організації, що не залежить від будь-яких міркувань щодо способу їхнього зберігання. На концептуальному рівні представлені наступні компоненти:

- всі сутності, їхні атрибути й зв'язки;
- обмеження, що накладаються на дані;
- семантична інформація про дані;
- інформація про міри забезпечення безпеки й підтримки цілісності даних.

Концептуальний рівень підтримує кожне зовнішнє представлення, у тому розумінні, що будь-які доступні користувачеві дані повинні міститися (або можуть бути обчислені) на цьому рівні. Однак цей рівень не містить ніяких відомостей про методи зберігання даних. Наприклад, опис сутності повинен містити відомості про типи даних атрибутів (числовий, дійсній або символьний) і їхній довжині (кількості значущих цифр або максимальній кількості символів), але не повинен включати відомостей про організацію зберігання даних, наприклад про обсяг зайнятого простору в байтах.

Внутрішній рівень

Внутрішній рівень. Фізичне представлення бази даних у комп'ютері. Цей рівень описує, як інформація зберігається в базі даних.

Внутрішній рівень описує фізичну реалізацію бази даних і призначений для досягнення оптимальної продуктивності й забезпечення ощадливого використання дискового простору. Він містить опис структур даних і організації окремих файлів, що використовуються для зберігання даних на запам'ятовувальних пристроях. На цьому рівні здійснюється взаємодія СУБД із методами доступу операційної системи (допоміжними функціями зберігання й витягу записів даних) з метою розміщення даних на запам'ятовувальних пристроях, створення індексів, витягу даних і т.д. На внутрішньому рівні зберігається наступна інформація:

- розподіл дискового простору для зберігання даних і індексів;
- опис подробиць збереження записів (із вказівкою реальних розмірів елементів даних, що зберігаються);
- відомості про розміщення записів;
- відомості про стиск даних і обраних методах їхнього шифрування.

Нижче внутрішнього рівня перебуває *фізичний рівень* (physical level), що контролюється операційною системою, але під керуванням СУБД. Однак функції СУБД і операційної системи на фізичному рівні не цілком чітко розділені й можуть варіюватися від системи до системи. В одних СУБД використовуються багато передбачених в даній операційній системі методів доступу, тоді як в інших застосовуються тільки самі основні й

реалізована власна файлова організація. Фізичний рівень доступу до даних нижче СУБД складається тільки з відомих операційній системі елементів (наприклад, покажчиків на те, як реалізован послідовний розподіл і чи зберігаються поля внутрішніх записів на диску у вигляді безперервної послідовності байтів).

Схеми, відображення й екземпляри

Загальний опис бази даних називається *схемою бази даних.* Існують три різних типи схем бази даних, які визначаються відповідно до рівнів абстракції трирівневою архітектури, як показано на рис. 1.3.

На найвищому рівні є кілька *зовнішніх схем* або *підсхем*, які відповідають різним представленням даних. На концептуальному рівні опис бази даних називають *концептуальною схемою*, а на найнижчому рівні абстракції — *внутрішньою схемою*.

Концептуальна схема описує всі сутності, атрибути й зв'язки між ними, із вказівкою необхідних обмежень підтримки цілісності даних. На нижньому рівні перебуває внутрішня схема, що є повним описом внутрішньої моделі даних. Вона містить визначення записів, що зберігаються, і методів подання, опису полів даних, відомості про індекси й обрані схеми хешірування. Для кожної бази даних існує тільки одна концептуальна й одна внутрішня схема.

СУБД відповідає за встановлення відповідності між цими трьома типами схем, а також за перевірку їхньої несуперечності. Інакше кажучи, СУБД повинна забезпечувати, щоб кожну зовнішню схему можна було вивести на основі концептуальної схеми. Для встановлення відповідності між будь-якими зовнішньою й внутрішньою схемами СУБД повинна використовувати інформацію з концептуальної схеми.

Концептуальна схема пов'язана із внутрішньою схемою за допомогою *концептуально внутрішнього відображення*. Воно дозволяє СУБД знайти фактичний запис або набір записів на фізичному пристрої зберігання, які утворять логічний запис у концептуальній схемі, з урахуванням будь-яких обмежень, встановлених для операцій, що виконуються над даним логічним записом. Воно також дозволяє виявити будь-які розходження в іменах об'єктів, іменах атрибутів, порядку проходження атрибутів, їхніх типах даних і т.д.

Нарешті, кожна зовнішня схема пов'язана з концептуальною схемою за допомогою *концептуально зовнішнього відображення*. З його допомогою СУБД може відображати імена представленя користувача на відповідну частину концептуальної схеми.

Важливо розрізняти опис бази даних і саму базу даних. Описом бази даних є *схема бази даних*. Схема створюється в процесі проектування бази даних, причому передбачається, що вона змінюється досить рідко.

Однак інформація, що втримується в базі даних, може мінятися часто – наприклад, щораз при вставці відомостей про нового співробітника або новий об'єкт нерухомості, що здається в оренду. Сукупність інформації, що зберігається в базі даних у будь-який певний момент часу, називається *станом бази даних*. Отже, тій самій схемі бази даних може відповідати безліч її різних станів. Схема бази даних іноді йменується *змістом* бази даних, а її стан – *деталізацією*.

Незалежність від даних

Основним призначенням трирівневої архітектури є забезпечення незалежності від даних, що означає, що зміни на нижніх рівнях не впливають на верхні рівні. Розрізняють два типи незалежності від даних: *логічну* й *фізичну*.

Логічна незалежність від даних. Логічна незалежність від даних означає повну захищеність зовнішніх схем від змін, внесених у концептуальну схему.

Такі зміни концептуальної схеми, як додавання або видалення нових сутностей, атрибутів або зв'язків, повинні здійснюватися без необхідності внесення змін у вже існуючі зовнішні схеми або коректування прикладних програм. Ясно, що користувачі, для яких ці зміни призначалися, повинні знати про їх, але дуже важливо, щоб інші користувачі навіть не підозрювали про це.

Фізична незалежність від даних. Фізична незалежність від даних означає захищеність концептуальної схеми від змін, внесених у внутрішню схему.

Такі зміни внутрішньої схеми, як використання різних файлових систем або структур зберігань, різних пристроїв зберігання, модифікація індексів або хешування, повинні здійснюватися без необхідності внесення змін у концептуальну або зовнішню схему. Користувачем можуть бути замічені зміни тільки в загальній продуктивності системи. У дійсності найпоширенішою причиною внесення змін у внутрішню схему є саме недостатня продуктивність виконання операцій. На рис. 1.4 показане місце перерахованих вище типів незалежності від даних у трирівневій архітектурі СУБД.

Прийняте в архітектурі ANSI-SPARC двоєтапне відображення може позначатися на ефективності роботи, але при цьому воно забезпечує більш високу незалежність від даних.

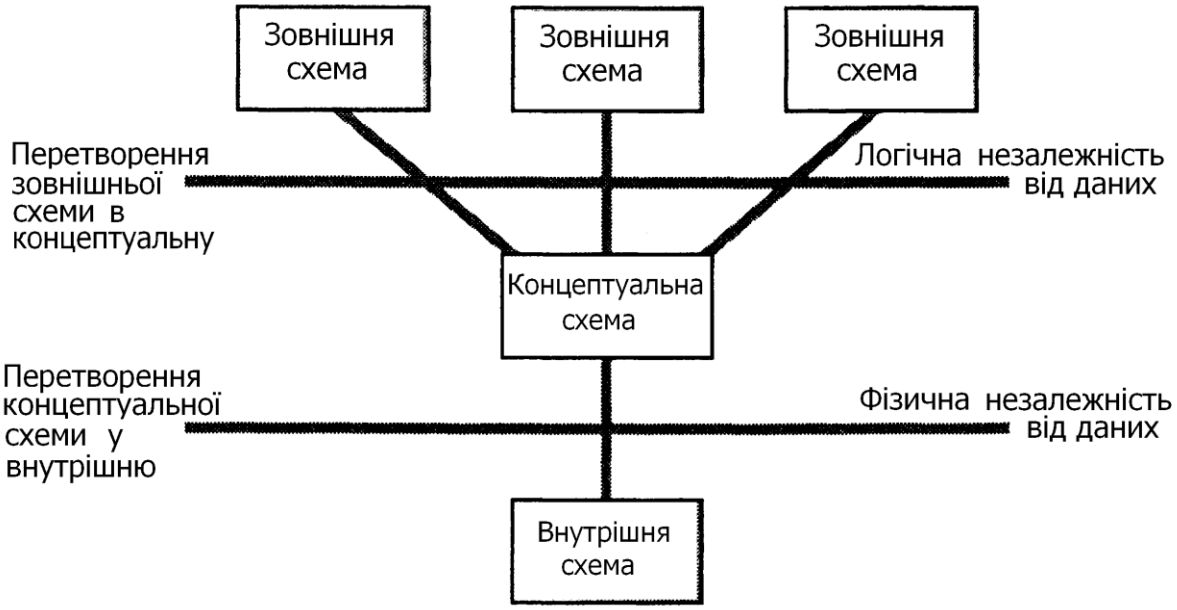


Рис. 1.4 Реалізація незалежності від даних у трирівневій архітектурі ANSI-SPARC

Для підвищення ефективності в моделі ANSI-SPARC допускається використання прямого відображення зовнішніх схем на внутрішні, без звертання до концептуальної схеми. Однак це знижує рівень незалежності від даних, оскільки при кожній зміні внутрішньої схеми буде потрібно внесення певних змін у зовнішню схему й всі залежні від неї прикладні програми.

1.5.2 Архітектура СУБД багатьох користувачів

По своїй суті бази даних повинні працювати в режимі з багатьма користувачами. Однак перші СУБД для персональних комп'ютерів були локальні, тобто СУБД обслуговувала тільки той комп'ютер, на якому сама розташовувалася. При цьому така СУБД могла працювати в мережі, і база даних могла розташовуватись на іншому комп'ютері. Для організації роботи багатьох користувачів з базами даних під керуванням подібних СУБД використовувалось кілька підходів [9], найчастіше використовувались СУБД, що базувались на *архітектурі з файловим сервером*.

В цих СУБД на кожному комп'ютері, що мав право працювати з базою даних, розташовувалася своя версія СУБД, а також необхідне програмне забезпечення для забезпечення захисту й цілісності бази даних.

На одному з комп’ютерів розташовувалась база даних – цей комп’ютер виступає у якості файлового серверу; інші комп'ютери виступають у ролі робочих станцій.

Для архітектури з файловим сервером можна відзначити наступні недоліки:

1. Проблеми з розробкою програмного забезпечення: кожний комп'ютер повинен бути забезпечений повною копією СУБД із усім програмним забезпеченням, що забезпечує захист і цілісність бази даних, але при цьому комп'ютери (робочі станції) можуть сильно відрізнятися друг від друга по своїх характеристиках. Розробка програмного забезпечення під самий "слабкий" комп'ютер приведе до неефективної роботи з базою даних більш сильних комп'ютерів.
2. Оскільки обробка даних ведеться тільки на робочих станціях, то буде пересилатися по мережі досить великий обсяг інформації. Наприклад, із двох таблиць, кожна з яких містить кілька тисяч записів, вибирається не більше десятка записів, причому в кожному записі вибираються не всі поля. Якби обробка проводилась на центральному комп'ютері, то необхідно було б передати тільки обрані дані. Однак, при роботі з файловим сервером іде звернення на передачу файлу (у цьому випадку двох файлів великого розміру), а вибірка відбувається на робочій станції. Природно, виходить великий обсяг мережного трафіка.
3. Управління одночасною роботою, захистом і забезпеченням цілісності ускладнюється, оскільки до тих самих файлів мають доступ кілька екземплярів СУБД. Крім того, ускладнюється робота адміністратора бази даних по забезпеченню захисту й цілісності бази даних, оскільки програмне забезпечення, що виконує ці функції, перебуває на всіх робочих станціях, і при його оновленні необхідно обновити всі його копії, і проконтролювати, що на всіх комп'ютерах, що є робочими станціями, це програмне забезпечення коректно працює.

У наш час файл-серверys СУБД вважається застарілою. До цього класу відносяться, наприклад, такі СУБД як Microsoft Access, Paradox, FoxPro, Visual FoxPro.

З метою усунення недоліків локальних СУБД була розроблена технологія "клієнт/сервер".

Технологія "клієнт/сервер"

У цій технології використовується принцип взаємодії програмних компонентів, при якому вони утворюють єдину систему. Як видно з назви, існує якийсь *клієнтський процес*, що вимагає певних ресурсів, і *серверний процес*, що надає ці ресурси. Ці процеси можуть бути на різних комп'ютерах, з'єднаних у мережу різної топології. Комп'ютер, що

забезпечує серверний процес, називається *сервером бази даних*. Звичайно на цьому комп'ютері перебуває база даних. Також база даних може перебувати на іншому комп'ютері, до якого прямо має доступ тільки сервер. Можлива схема побудови системи з архітектурою "клієнт/сервер" наведена на рис. 1.5.

Рис. 1.5 Загальна схема побудови систем з архітектурою "клієнт/сервер"

На сервері перебуває та частина програмного забезпечення (ПЗ) бази даних, що виконує функції забезпечення одночасної роботи, цілісності бази даних і її захисту. Цю частину програмного забезпечення називає *серверним ПЗ*, а часто просто *сервером*. Інша частина програмного забезпечення роботи з базою даних, називається *клієнтським ПЗ*. Вона виконує, в основному, наступні функції:

- керує інтерфейсом користувача;
- приймає й перевіряє синтаксис введеного користувачем запиту;
- виконує додаток;
- відображає отримані дані користувачеві.

Існує ще розподіл на «тонкий» і «товстий» клієнт. Тонкий клієнт управляє тільки інтерфейсом користувача, а вся обробка даних надається

серверу. Товстий клієнт може забезпечувати деякі функції збереження цілісності бази даних, наприклад, при уведенні й редагуванні даних перевіряти, щоб нові дані задовольняли обмеженням цілісності, прийнятим у цій базі даних.

Цей тип архітектури має наведені нижче переваги.

- Забезпечується більш широкий доступ до існуючих баз даних.
- Підвищується загальна продуктивність системи. Оскільки клієнти й сервер розташовуються на різних комп'ютерах, їх процесори здатні виконувати додатки паралельно. При цьому настроювання продуктивності комп'ютера з сервером спрощується, якщо на ньому виконується тільки робота з базою даних.
- Вартість апаратного забезпечення знижується. Достатньо потужний комп'ютер з великим пристроєм зберігання потрібен тільки серверу – для зберігання й керування базою даних.
- Зменшуються комунікаційні витрати. Додатки виконують частину операцій на клієнтських комп'ютерах і посилають через мережу тільки запити до бази даних, що дозволяє істотно скоротити обсяг даних, що пересилаються у мережі.
- Підвищується рівень несуперечності даних. Сервер може самостійно керувати перевіркою цілісності даних, оскільки всі обмеження визначаються і перевіряються тільки в одному місці. При цьому кожному додатку не потрібно виконувати власну перевірку.
- Ця архітектура досить природно відображається на архітектуру відкритих систем.

До недоліків клієнт-серверних СУБД можна віднести підвищені вимоги до сервера.

1.6 Функції СУБД

Основні функції СУБД перераховані з тих стандартах (розділ 1.4.1), яким повинне відповідати програмне забезпечення, щоб воно могло називатися системою керування базами даних. Розглянемо докладніше головні функції СУБД.

У свій час Кодд запропонував перелік з восьми служб, які повинні бути реалізовані в будь-який повномасштабної СУБД [7]. Нижче приводиться короткий опис кожної з них.

1.6.1 Зберігання, добування й відновлення даних

СУБД повинна надавати користувачам можливість зберігати, витягати й обновляти дані в базі даних.

Це сама фундаментальна функція СУБД.

1.6.2 Каталог, доступний кінцевим користувачам

СУБД повинна мати доступний кінцевим користувачам каталог, у якому зберігається опис елементів даних.

Ключовою особливістю архітектури ANSI-SPARC є наявність інтегрованого системного каталогу з даними про схеми, користувачів, додатки й т.д. Передбачається, що цей каталог доступний як користувачам, так і функціям СУБД. Системний каталог (або словник даних) є сховищем інформації, що описує дані в базі даних (по суті, це "дані про дані", або метадані). Залежно від типу СУБД, що використовується, кількість інформації й спосіб її застосування можуть змінюватися. Звичайно в системному каталозі зберігаються наступні відомості:

- імена, типи й розміри елементів даних;
- імена зв'язків;
- обмеження підтримки цілісності, що накладаються на дані;
- імена користувачів, яким надане право доступу до даних;
- зовнішня, концептуальна й внутрішня схеми й відображення між ними;
- статистичні дані, наприклад частота транзакцій і лічильники звертань до об'єктів бази даних.

Системний каталог дозволяє досягти певних переваг, перерахованих нижче:

- Інформація про дані може бути централізовано зібрана й збережена, що дозволить контролювати доступ до цих даних, як і до будь-якого іншого ресурсу.
- Можна визначити зміст даних, що допоможе іншим користувачам зрозуміти їхнє призначення.
- Спрощується спілкування, тому що є точне визначення змісту даних. У системному каталозі також можуть бути зазначені один або декілька користувачів, які є власниками даних або мають право доступу до них.

- Завдяки централізованому зберіганню надмірність і суперечливість опису окремих елементів даних можуть бути легко виявлені.
- Внесені в базу дані зміни можуть бути запротокольовані.
- Наслідки будь-яких змін можуть бути визначені ще до їхнього внесення, оскільки в системному каталозі зафіксовані всі існуючі елементи даних, установлені між ними зв'язки, а також всі їхні користувачі.
- Міри забезпечення безпеки можуть бути додатково посилені.
- З'являються нові можливості організації підтримки цілісності даних.
- Може виконуватися аудит інформації, що збегірається.

Деякі автори, крім системного каталогу, виділяють *каталог даних* (data directory), у якому перебуває інформація про місце й спосіб зберігання даних. У даному конспекті термін "системний каталог" застосовується відносно всієї інформації про зберігання даних.

1.6.3 Підтримка транзакцій

СУБД повинна мати механізм, що гарантує виконання або всіх операцій відновлення даної транзакції, або жодної з них.

Транзакція являє собою набір дій, які виконуються окремим користувачем або прикладною програмою з метою доступу або зміни вмісту бази даних.

Якщо під час виконання транзакції відбудеться збій, наприклад через вихід з ладу комп'ютера, база даних попадає в суперечливий стан, оскільки деякі зміни вже будуть внесені, а інші – ще ні. Тому всі часткові зміни повинні бути скасовані для повернення бази даних у колишній, несуперечливий стан.

1.6.4 Служби керування паралельною роботою

СУБД повинна мати механізм, що гарантує коректне відновлення бази даних при паралельному виконанні операцій відновлення багатьма користувачами.

Одна з основних цілей створення й використання СУБД полягає в тім щоб множина користувачів могла здійснювати паралельний доступ до даних, що спільно обробляються. Паралельний доступ порівняно просто організувати, якщо всі користувачі виконують тільки читання даних, оскільки в цьому випадку вони не можуть перешкодити один одному.

Однак коли два або більше користувачів одночасно одержують доступ до бази даних, конфлікт з небажаними наслідками легко може виникнути, наприклад, якщо хоча б один з них спробує обновити дані.

1.6.5 Служби відновлення

СУБД повинна надавати засоби відновлення бази даних на випадок якого-небудь її ушкодження або руйнування

Під час обговорення підтримки транзакцій згадувалося, що при збої транзакції база даних повинна бути повернута в несуперечливий стан. Подібний збій може відбутися в результаті виходу з ладу системи або запам'ятовуючого пристрою, помилки апаратного або програмного забезпечення, які можуть привести до останову СУБД.

Крім того, користувач може виявити помилку під час виконання транзакції й зажадати її скасування. У всіх цих випадках СУБД повинна надати механізм відновлення бази даних і повернення її до несуперечливого стану.

1.6.6 Служби контролю доступу до даних

СУБД повинна мати механізм, що гарантує можливість доступу до бази даних тільки санкціонованих користувачів.

Неважко привести приклади, коли потрібно сховати деякі збережені в базі даних відомості від інших користувачів. Наприклад, менеджерам відділень компаній можна було б надати всю інформацію, пов'язану із зарплатою співробітників, але бажано сховати її від інших користувачів.

Крім того, базу даних варто було б захистити від будь-якого несанкціонованого доступу. Термін *безпека* відноситься до захисту бази даних від навмисного або випадкового несанкціонованого доступу. Передбачається, що СУБД забезпечує механізми подібного захисту даних.

1.6.7 Підтримка обміну даними

СУБД повинна мати здатність до інтеграції з комунікаційним програмним забезпеченням

Більшість користувачів здійснюють доступ до бази даних за допомогою терміналів. Іноді ці термінали приєднані безпосередньо до комп'ютера із СУБД. В інших випадках термінали можуть перебувати на значній відстані й обмінюватися даними з комп'ютером, на якому розташовується СУБД, через мережу.

У кожному разі СУБД одержує запити у вигляді повідомлень обміну даними (communications messages) і аналогічним образом відповідає на них. Всі такі спроби передачі даних управляються диспетчером обміну даними (DEM – Data Exchange Manager). Хоча цей диспетчер не є частиною, яка властива СУБД, проте, щоб бути комерційно життєздатної, будь-яка СУБД повинна мати здатність інтеграції з різноманітними існуючими диспетчерами обміну даними.

1.6.8 Служби підтримки цілісності даних

СУБД повинна мати інструменти контролю за тим, щоб дані і їхні зміни відповідали заданим правилам.

Цілісність бази даних означає коректність і несуперечність збережених даних. Вона може розглядатися як ще один тип захисту бази даних. Крім того, що дане питання пов'язане із забезпеченням безпеки, воно має більш широкий зміст, оскільки цілісність пов'язана з якістю самих даних.

Цілісність звичайно виражається у вигляді обмежень або правил збереження несуперечності даних, які не повинні порушуватися в базі.

Розумно було б припустити, що, крім перерахованих вище восьми служб, СУБД повинна підтримувати ще дві служби.

1.6.9 Служби підтримки незалежності від даних

СУБД повинна мати інструменти підтримки незалежності програм від фактичної структури бази даних.

Поняття незалежності від даних уже розглядалося в розділі 1.5.1. Звичайно вона досягається за рахунок реалізації механізму підтримки представлень або підсхем. Фізична незалежність від даних досягається досить просто, тому що звичайно є кілька типів припустимих змін фізичних характеристик бази даних, які ніяк не впливають на представлення. Однак домогтися повної логічної незалежності від даних складніше. Як правило, система легко адаптується до додавання нового об'єкта, атрибута або зв'язку, але не до їхнього вилучення. У деяких системах взагалі забороняється вносити будь-які зміни у вже існуючі компоненти логічної структури.

1.6.10 Допоміжні служби

СУБД повинна надавати деякий набір різних допоміжних служб.

Допоміжні утиліти звичайно призначені для надання допомоги АБД в ефективному адмініструванні бази даних. Одні утиліти працюють на

зовнішньому рівні, а тому вони, у принципі, можуть бути створені самими АБД, тоді як інші функціонують на внутрішньому рівні системи й тому повинні бути надані самим розроблювачем СУБД. Нижче приводяться деякі приклади подібних утиліт.

- Утиліти імпортування, призначені для завантаження бази даних із плоских файлів, а також утиліти експортування, які служать для вивантаження бази даних у плоскі файли.
- Засоби моніторингу, призначені для відстеження характеристик функціонування й використання бази даних.
- Програми статистичного аналізу, що дозволяють оцінити продуктивність або ступінь використання бази даних.
- Інструменти реорганізації індексів, призначені для перебудови індексів у випадку їхнього переповнення.
- Інструменти зборки сміття й перерозподілу пам'яті для фізичного усунення вилучених записів із запам'ятовувальних пристроїв, об'єднання звільненого простору й перерозподілу пам'яті в міру необхідності.

1.7 Переваги й проблеми СУБД

СУБД мають як багатообіцяючі потенційні переваги, так і проблеми, які ми коротко розглянемо в цьому розділі.

Переваги систем керування базами даних перераховані нижче:

- Контроль за надмірністю даних
- Несуперечність даних
- Більше корисної інформації при тім же обсязі збережених даних
- Спільне використання даних
- Підтримка цілісності даних
- Підвищена безпека
- Застосування стандартів
- Підвищення ефективності з ростом масштабів системи
- Можливість знаходження компромісу при суперечливих вимогах
- Підвищення доступності даних і їхньої готовності до роботи
- Поліпшення показників продуктивності
- Спрощення супроводу системи за рахунок незалежності від даних
- Поліпшене керування паралельною роботою
- Розвинені служби резервного копіювання й відновлення

Контроль за надмірністю даних

Традиційні файлові системи неекономно витрачають зовнішню пам'ять, зберігаючи ті самі дані в декількох файлах. При використанні бази даних, навпаки, здійснюється спроба виключити надмірність даних за рахунок інтеграції файлів, що дозволяє виключити необхідність зберігання декількох копій того самого елемента інформації.

Але повністю надмірність інформації в базах даних не виключається, а лише обмежується її ступінь. В одних випадках ключові елементи даних необхідно дублювати для моделювання зв'язків, а в інших випадках деякі дані потрібно дублювати з міркувань підвищення продуктивності системи. Тому існує поняття *контрольованої надмірності даних* у БД.

Несуперечність даних

Усунення надмірності даних або контроль над нею дозволяє зменшити ризик виникнення суперечливих станів. Якщо елемент даних зберігається в базі тільки в одному екземплярі, то для зміни його значення буде потрібно виконати тільки одну операцію відновлення, при цьому нове значення стане доступним відразу всім користувачам бази даних. А якщо цей елемент даних з дозволу системи зберігається в базі даних у декількох екземплярах, то така система зможе стежити за тим, щоб копії не суперечили одна одній. На жаль, у багатьох сучасних СУБД такий спосіб забезпечення несуперечності даних не підтримується автоматично.

Більше корисної інформації при тім же обсязі збережених даних

Завдяки інтеграції робочих даних організації на основі тих же даних можна отримувати додаткову інформацію. Наприклад, у деканатах університету дані про результати сесії з відомостей заносяться в табличному вигляді у файл, що містить оцінки всіх студентів окремої групи за одну сесію. Однак, якщо необхідно одержати дані про всі оцінки окремо взятого студента за увесь час навчання, то доведеться створювати нові файли, в які треба скопіювати дані з множини інших файлів. Якщо створити базу даних з оцінками студентів, то з неї без дублювання інформації за допомогою представлень можна одержувати відомості про успішність студентів у різних видах: успішність групи в даному семестрі, успішність окремого студента за увесь час навчання, рейтинги всіх студентів за окремий семестр або за увесь час навчання й ін. Тепер на основі тих же даних користувачі зможуть одержувати більше інформації.

Спільне використання даних

Файли звичайно належать окремим особам або цілим відділам, які використовують їх у своїй роботі. У той же час база даних належить всій організації в цілому й може спільно використовуватися всіма

зареєстрованими користувачами. При такій організації роботи більша кількість користувачів може працювати з більшим обсягом даних.

Більше того, при цьому можна створювати нові додатки на основі вже існуючої в базі дані інформації й додавати в неї тільки ті дані, які в даний момент ще не зберігаються в ній, а не визначати заново вимоги до всіх даних, які необхідні новому додатку. Нові додатки можуть також використовувати такі функціональні можливості, що надаються типовими СУБД, як визначення структур даних і керування доступом до даних, організація паралельної обробки й забезпечення засобів копіювання/відновлення, виключивши необхідність реалізації цих функцій зі своєї сторони.

Підтримка цілісності даних

Цілісність бази даних означає коректність і несуперечність збережених у ній даних. Цілісність звичайно описується за допомогою обмежень, тобто правил підтримки несуперечності, які не повинні порушуватися в базі даних. Обмеження можна застосовувати до елементів даних усередині одного запису або до зв'язків між записами. Наприклад, обмеження цілісності може говорити, що зарплата співробітника не повинна перевищувати 10000 грн. або ж що в записі з даними про співробітника номер відділення, у якому він працює, повинен відповідати реально існуючому відділенню компанії. Таким чином, інтеграція даних дозволяє адміністраторові бази даних (АБД) задавати вимоги по підтримці цілісності даних, а СУБД застосовувати їх.

Підвищена безпека

Безпека бази даних полягає в захисті бази даних від несанкціонованого доступу з боку користувачів. Без залучення відповідних мір безпеки інтегровані дані стають більше уразливими, чим дані у файловій системі. Однак інтеграція дозволяє АБД визначити необхідну систему безпеки бази даних, а СУБД привести її в дію. Система забезпечення безпеки може бути виражена у формі імен і паролів для ідентифікації користувачів, які зареєстровані в цій базі даних. Доступ до даних з боку зареєстрованого користувача може бути обмежений тільки деякими операціями (витягом, вставкою, відновленням і видаленням).

Застосування стандартів

Інтеграція дозволяє АБД визначати й застосовувати необхідні стандарти. Наприклад, стандарти відділу й організації, державні й міжнародні стандарти можуть регламентувати формат даних при обміні ними між системами, угоди про імена, форму подання документації, процедури відновлення й правила доступу. Стандарти застосовуються вже на етапі проектування бази даних. Сучасні бази даних розробляються

командою програмістів, і без використання стандартів неможливе коректне стикування частин проекту, розроблених різними виконавцями.

Підвищення ефективності зі збільшенням масштабів системи

Комбінуючи всі робочі дані організації в одній базі даних і створюючи набір додатків, які працюють із одним джерелом даних, можна домогтися істотної економії коштів. У цьому випадку бюджет, що звичайно виділявся кожному відділу для розробки й підтримки їх власних файлових систем, можна об'єднати з бюджетами інших відділів (з більш низькою загальною вартістю), що дозволить домогтися підвищення ефективності при зростанні масштабів виробництва. Тепер об'єднаний бюджет можна буде використовувати для придбання обладнання в тій конфігурації, що більшою мірою відповідає потребам організації.

Можливість знаходження компромісу для суперечливих вимог

Потреби одних користувачів або відділів можуть суперечити потребам інших користувачів. Але оскільки база даних контролюється АБД, він може ухвалювати рішення щодо проектування й способу використання бази даних, при яких наявні ресурси всієї організації в цілому будуть використовуватися щонайкраще. Ці рішення забезпечують оптимальну продуктивність для найважливіших додатків, причому найчастіше за рахунок менш критичних.

Підвищення доступності даних і їхньої готовності до роботи

Дані, які перетинають кордони відділів, у результаті інтеграції стають безпосередньо доступними кінцевим користувачам. Потенційно це підвищує функціональність системи, що, наприклад, може бути використане для більш якісного обслуговування кінцевих користувачів або клієнтів організації.

У багатьох СУБД передбачені мови запитів або інструменти для створення звітів, які дозволяють користувачам уводити не передбачені заздалегідь запити й майже негайно отримувати необхідну інформацію на своїх терміналах, не вдаючись до допомоги програміста, який для витягу цієї інформації з бази даних повинен був би створити спеціальне програмне забезпечення.

При правильному проектуванні програмного забезпечення для бази даних (системи бази даних) програмісти повинні забезпечити користувачів практично всіма можливими запитами.

Поліпшення показників продуктивності

У СУБД передбачено багато стандартних функцій, які програміст звичайно повинен самостійно реалізувати в додатках для файлових систем.

Функції СУБД розглядаються в розділі 1.6. На базовому рівні СУБД забезпечує всі низкорівневі процедури роботи з файлами, що звичайно виконують додатки. Наявність цих процедур дозволяє програмістові сконцентруватися на розробці більш спеціальних, необхідних користувачам функцій, не піклуючись про подробиці їхнього втілення на більш низькому рівні.

Спрощення супроводу системи за рахунок незалежності від даних

У файлових системах опис даних і логіка доступу до даних вбудовані в кожний додаток, тому програми стають залежними від даних. У СУБД підхід іншої: опис даних відділені від додатків, а тому додатки захищені від змін в описах даних. Ця особливість називається *незалежністю від даних*. Наявність незалежності програм від даних значно спрощує обслуговування й супровід додатків, що працюють із базою даних.

Поліпшене керування паралельною роботою

У деяких файлових системах при одночасному доступі до того самого файлу двох користувачів може виникнути конфлікт двох запитів, результатом якого буде втрата інформації або втрата її цілісності. У свою чергу, у багатьох СУБД передбачена можливість паралельного доступу до бази даних і гарантується відсутність подібних проблем.

Розвинені служби резервного копіювання й відновлення

Відповідальність за забезпечення захисту даних від збоїв апаратного й програмного забезпечення у файлових системах покладається на користувача. Так, може знадобитися щоночі виконувати резервне копіювання даних. При цьому у випадку збоїв може бути відновлена резервна копія, але результати роботи, яка виконувалась після резервного копіювання, будуть втрачені, і дану роботу буде потрібно виконати заново. У сучасних СУБД передбачені засоби зниження ймовірності втрат інформації при виникненні різних збоїв.

Проблеми СУБД

Проблеми, що виникають при переході організації до підходу, пов'язаного із застосуванням баз даних, перераховані нижче:

- Складність
- Розмір
- Вартість СУБД
- Додаткові витрати на апаратне забезпечення
- Витрати на перетворення
- Продуктивність

- Більше серйозні наслідки при виході системи з ладу

Складність

Забезпечення функціональності, яку повинна мати кожна гарна СУБД, супроводжується значним ускладненням програмного забезпечення СУБД. Щоб скористатися всіма перевагами СУБД, проектувальники й розроблювачі баз даних, адміністратори даних і адміністратори баз даних, а також кінцеві користувачі повинні добре розуміти функціональні можливості СУБД. Нерозуміння принципів роботи системи може привести до невдалих результатів проектування, що буде мати самі серйозні наслідки для всієї організації.

Розмір

Складність і широта функціональних можливостей приводить до того, що СУБД стає надзвичайно складним програмним продуктом, що може зажадати багато місця на диску й мати потребу у великому об'ємі оперативної пам'яті для ефективної роботи.

Вартість СУБД

Залежно від наявного обчислювального середовища й необхідних функціональних можливостей вартість СУБД може змінюватися в дуже широких межах. Наприклад, локальна СУБД для персонального комп'ютера може коштувати близько 100 доларів. Однак велика СУБД для мейнфрейма, що обслуговує сотні користувачів, може бути надзвичайно дорогої: від 100 000 до 1 000 000 доларів. Крім того, варто врахувати щорічні витрати на супровід системи, які становлять деякий відсоток від її загальної вартості.

Додаткові витрати на апаратне забезпечення

Для задоволення вимог, які висуваються до дискових накопичувачів з боку СУБД і бази даних, може знадобитися придбати додаткові пристрої зберігання інформації. Більш того, для досягнення необхідної продуктивності може знадобитися могутніший комп'ютер, що, можливо, буде працювати тільки із СУБД. Придбання іншого додаткового апаратного забезпечення призведе до подальшого зростання витрат.

Витрати на перетворення

У деяких ситуаціях вартість СУБД і додаткового апаратного забезпечення може виявитися несуттєвої в порівнянні з вартістю перетворення існуючих додатків для роботи з новою СУБД і новим апаратним забезпеченням. Ці витрати включають також вартість підготовки персоналу для роботи з новою системою, а також оплату

послуг фахівців, які будуть надавати допомогу в перетворенні й запуску нової системи.

Продуктивність

Звичайно файлова система створюється для деяких спеціалізованих додатків, наприклад для оформлення рахунків, а тому її продуктивність може бути досить висока. Однак СУБД призначені для рішення більш загальних завдань і обслуговування відразу декількох додатків, а не якогось одного з них. У результаті багато додатків у новому середовищі будуть працювати не так швидко, як колись.

Більш серйозні наслідки при виході системи з ладу

Централізація ресурсів підвищує уразливість системи. Оскільки робота всіх користувачів і додатків залежить від готовності до роботи СУБД, вихід з ладу одного з її компонентів може привести до повного припинення всієї роботи організації.

2 МОДЕЛІ ДАНИХ

Вище вже згадувалося, що схема створюється за допомогою деякої мови визначення даних. У дійсності вона створюється на основі мови визначення даних конкретної цільової СУБД. На жаль, це мова відносно низького рівня; з її допомогою важко описати вимоги до даних у масштабі всієї організації так, щоб створена схема була доступна розумінню користувачів самих різних категорій. Що нам дійсно потрібно, так це опис схеми на якомусь, більше високому рівні. Цей опис будемо називати *моделлю даних*.

Кожна СУБД підтримує ту або іншу модель даних. Модель даних – це засіб для визначення логічного представлення фізичних даних, що відносяться до деякого додатка.

Модель даних. Інтегрований набір понять для опису й обробки даних, зв'язків між ними й обмежень, що накладають на дані в деякій організації.

Модель є представленням "реального світу" об'єктів і подій, а також існуючих між ними зв'язків. Це деяка абстракція, у якій акцент робиться на найважливіших і невід'ємних аспектах діяльності організації, а всі другорядні властивості ігноруються. Таким чином, можна сказати, що модель даних представляє саму організацію.

Модель повинна відбивати основні концепції, представлені в такому вигляді, що дозволить проектувальникам і користувачам бази даних обмінюватися конкретними й недвозначними думками про ролі тих або інших даних в організації. Модель даних можна розглядати як сполучення трьох зазначених нижче компонентів.

- Структурна частина, тобто набір правил, по яких може бути побудована база даних.
- Керуюча частина, що визначає типи припустимих операцій з даними (сюди відносяться операції відновлення й витягу даних, а також операції зміни структури бази даних).
- Набір (необов'язковий) обмежень підтримки цілісності даних, що гарантують коректність даних, що використовуються.

Мета побудови моделі даних полягає в представленні даних у зрозумілому виді. Якщо таке представлення можливо, то модель даних можна легко застосувати при проектуванні бази даних. Для відображення архітектури ANSI-SPARC, яка обговорювалась в розділі 2.4, можна визначити наступні три зв'язані моделі даних:

- зовнішня модель даних, що відображає представлення кожного існуючого в організації типу користувачів, що іноді називають предметною областю (Universe of Discourse – UoD);

- концептуальна модель даних, що відображає логічне (або узагальнене) представлення про дані, незалежне від типу обраної СУБД;
- внутрішня модель даних, що відображає концептуальну схему певним чином, що підходить для обраної цільовий СУБД.

У літературі запропоновано й опубліковано досить багато моделей даних. Вони підрозділяються на три категорії: *об'єктні* (object-based) моделі даних, *моделі даних на основі записів* (record-based) і *фізичні моделі даних*. Перші дві використовуються для опису даних на концептуальному й зовнішньому рівнях, а остання – на внутрішньому рівні.

Модель даних визначає правила породження припустимих для даної СУБД видів структур даних, можливі операції над такими структурами, а також класи обмежень цілісності даних, що можуть представлятися засобами цієї системи.

По суті, модель даних, яка підтримується механізмами СУБД, повністю визначає множину конкретних баз даних, які можуть бути створені засобами цієї системи, а також способи модифікації стану бази даних з метою відображення тих змін, які відбуваються в предметній області.

У наш час розроблено значну кількість різноманітних моделей даних. Така ситуація є не просто результатом абстрактних теоретичних вишукувань їхніх авторів, а наслідком реальних потреб практики обробки даних. У кожному випадку розробки системи баз даних потрібно мати можливість вибору способу "бачення" предметної області, адекватного потребам її користувачів.

У більшості комерційних СУБД використовуються моделі, що вважаються класичними: реляційна модель і різновиди графових моделей даних – мережна модель даних CODASYL і ієрархічна модель даних. Всі вони одержали свою назву по видах структур даних, які розглядаються у них.

Ієрархічні моделі дозволяють будувати бази даних з ієрархічною деревоподібною структурою, а мережні – бази даних, структура яких представляється графом загального виду. У таких моделях даних передбачаються характерні для подібного роду структур операції навігації й маніпулювання даними. Принципове значення при цьому має та обставина, що передбачається одночасна обробка тільки одиночних об'єктів даних з бази даних.

Апарат *навігації* в графових моделях служить для установки тих об'єктів даних, до яких буде застосовуватися чергова операція маніпулювання даними. Такі об'єкти називаються *поточними*. Механізми доступу до даних і навігації вздовж структури даних у таких моделях

досить складні, особливо в мережній моделі, і істотно опираються на концепцію поточного стану механізмів доступу.

В 70-х роках почали активно проводитися теоретичні дослідження реляційної моделі даних. Були створені перші СУБД. які її підтримують. З появою персональних ЕОМ реляційні системи стали домінувати на ринку програмного забезпечення систем баз даних.

Відповідно до *реляційної моделі даних* база даних представляється у вигляді сукупності *таблиць*, над якими можуть виконуватися операції, що формулюються в термінах реляційної алгебри або реляційного вирахування.

На відмінність від ієрархічних і мережних моделей даних у реляційнії моделі операції над об'єктами бази даних мають теоретико-множинний характер. Це дає можливість користувачам формулювати їхні запити більш компактно, у термінах більших агрегатів даних.

Однак такий підхід породжує складні проблеми, пов'язані із забезпеченням досить високого рівня продуктивності СУБД цього класу, які доводиться вирішувати їхнім розроблювачам. Інша проблема виникає, коли потрібно забезпечити інтерфейс СУБД, що підтримує реляційну модель даних, із традиційними мовами програмування. Вона полягає в невідповідності структур даних моделі й мов такого типу, орієнтованих на "позаписну" обробку. Для її рішення доводиться поповнювати модель даних спеціальними узгодженими типами об'єктів.

2.1 Ієрархічні системи

Основним типом логічної структури, яка підтримується ієрархічними СУБД, є *ієрархія*, або *деревоподібна структура*. Ієрархічна структура визначається на відповідних *типах записів* (або сегментах) відповідно до схеми прикладної бази даних або *дереву визначень*. На цьому дереві типи записів є вузлами, а дуги представляють зв'язки вихідний – породжений між вузлами дерева різних рівнів. Якщо різниця рівнів двох зв'язаних вузлів дорівнює одиниці, то зв'язок є безпосереднім (без проміжних вузлів). Крім того, будь-які дві вершини дерева, що належать одній гілці, транзитивно зв'язані одна з одною. На рис. 4.1 показана ієрархія п'яти типів записів, скорочений варіант вихідної схеми й приклад завантаженої бази даних.

На рис. 2.1*в* наведено кілька екземплярів ієрархічних записів бази даних, що відповідають дереву визначень. Екземпляр дерева бази даних не обов'язково повинен містити всі свої сегменти. (Прикладами є перше дерево $\mathbf{R_1}$ і передостаннє дерево $\mathbf{R_{n-1}}$). При необхідності можна додавати або видаляти екземпляри типів записів відповідно до вимог додатка. Припустимо, що типи записів A, B, C, D і E інтерпретуються як типи

записів «Курс», «Попередня реєстрація», «Семестр», «Викладач» і «Студент». Інформація про деякий курс може бути відсутньою (наприклад, $\mathbf{R_{n-1}}$), або для курсу може бути відома лише попередня реєстрація ($\mathbf{R_1}$) (*Зауваження*. У західних університетах студенти самі вибирають курси, які будуть слухати. Рішення про те, що деякий курс буде читатися, приймається після реєстрації студентів, які бажають слухати цей курс).

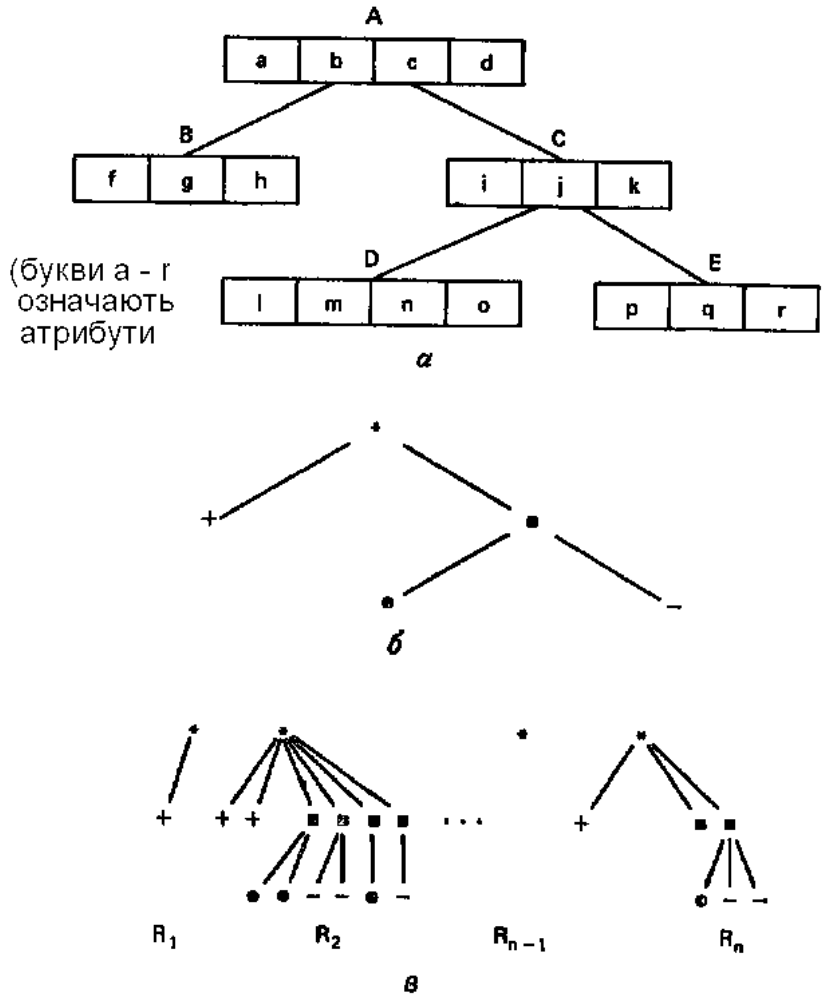


Рис. 2.1. а) ієрархія п'яти типів записів; б) скорочений варіант дерева визначень; в) завантажена база даних

Порядок зберігання записів у фізичній базі даних залежить від особливостей реалізації, що визначають спосіб відображення логічних дерев у фізичні структури файлів конкретної системи. Найбільш простий спосіб складається в лінеаризації дерева, при цьому екземпляри типів записів зберігаються послідовно: у порядку обходу дерева зверху вниз зліва направо.

В ієрархічних СУБД спосіб реалізації операції пошуку орієнтований на деревоподібну структуру. У зв'язку із цим пошук починається з кореня

й триває в напрямку породжених вузлів. Якщо в конкретній реалізації крім послідовного перегляду всього дерева відсутній який-небудь спосіб прямого доступу до конкретного типу запису, то положення вузла в дереві має важливе значення для доступу до нього.

Іншою проблемою ієрархій є неможливість зберігання в базі даних породженого вузла без відповідного вихідного, тобто в цьому випадку необхідно ввести порожній вихідний вузол. Відповідно видалення даного вихідного вузла тягне видалення всіх породжених вузлів (піддерев), пов'язаних з ним. Ці обмеження створюють проблеми для деяких додатків. Таким чином, проектування схеми в ряді випадків може виявитися складним завданням.

2.2 Мережні системи

Мережні СУБД використовують мережну модель даних. З погляду теорії графів мережній моделі відповідає довільний граф (який можливо має цикли й петлі). У вузлах графа містяться типи записів, а ребра інтерпретуються як зв'язки між типами записів. На рис. 2.2 показана мережна схема й відповідна цій схемі база даних.

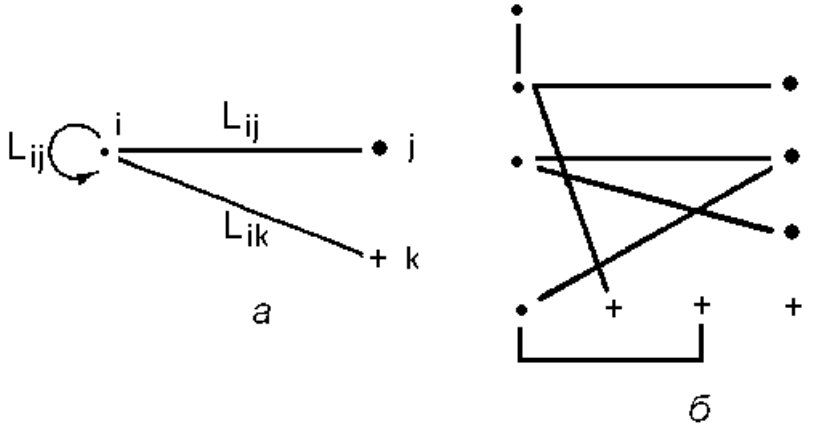


Рис. 2.2 Структура мережної бази даних.
а – схема мережі, 6 – екземпляр запису мережної БД

Наведене визначення є занадто загальним для цілей реалізації, тому що в ньому відсутні обмеження на способи з'єднання типів записів у схемі бази даних. Робоча група по базах даних (РГБД) асоціації КОДАСИЛ (CODASYL) сформулювала деякі обмеження на мережну модель даних.

Схема мережної бази даних моделі РГБД складається з множини типів записів, які можуть бути власниками або членами типів наборів, обумовлених у схемі. Тип набору визначає зв'язок між типами запису-члена й запису-власника в такий спосіб:

а) Тип набору визначає відображення 1:N між типом запису-власника й типами записів-членів набору.

б) Екземпляр типу запису-члена може брати участь тільки в одному екземплярі даного типу набору.

в) Тип запису-власника в типі набору не може збігатися з типом запису-члена (це обмеження було знято).

У пункті а) є одна відмінність від строго ієрархічної системи. Воно полягає в тому, що допускаються записи-члени, що не беруть участь у наборах. Це відповідає породженим вузлам дерева, що не мають вихідних вузлів. Таким чином, замість повної функціональної залежності «члени – власник» допускається часткова функціональна залежність.

Крім того, у мережній моделі РГБД можна визначити декілька типів наборів між двома типами записів, так що між ними можуть бути задані різні відносини на відміну від єдиного відношення вихідний-породжений, припустимого в ієрархічних структурах.

Існують різні способи перетворення загальних мережних структур у структури РГБД із урахуванням обмежень а) – в).

2.3 Реляційна модель даних

Ієрархічні й мережні системи досить громіздкі й вимагають великого мистецтва від проектувальника. Реляційні моделі були відповіддю на вимогу часу до спрощення баз даних. У деякому змісті реляційні системи були поверненням до файлових систем але на принципово іншому рівні.

Реляційні СУБД використовують реляційну модель даних, за якою структура даних представляється у вигляді прямокутних таблиць, що складаються зі стовпців і рядків. Прямокутні таблиці реляційної бази даних називаються відношеннями. Відношення (таблиця) описує деякий об'єкт реального миру (наприклад, відношення ЛІКАР, ПАЦІЄНТ) або взаємозв'язок між об'єктами (наприклад відношення ЛІКУЄ містить відомості про те, які лікарі яких пацієнтів лікують).

Реляційні системи використовують такі математичні апарати, як теорію множин, реляційну алгебру й реляційне вирахування. Широке поширення реляційних систем почалося в середині 80-х років XX століття у зв'язку з появою порівняно дешевих і досить продуктивних персональних комп'ютерів, для яких стали розроблятися реляційні СУБД різними виробниками програмного забезпечення.

3 РЕЛЯЦІЙНІ СИСТЕМИ

Теоретичні основи реляційної моделі баз даних були закладені Коддом на початку 70-х років XX століття.

Структура даних у реляційних системах простіша, ніж в ієрархічних або мережних системах. Маніпулювання даними є також більш простим, і має більше можливостей. Крім того, як було зазначено в попередньому розділі, теорія реляційних баз даних опирається на потужні математичні апарати.

3.1 Короткий огляд історії реляційної моделі

Реляційна модель уперше була запропонована Э. Ф. Коддом (E. F. Codd) в 1970 році в його основній статті "Реляційна модель даних для великих банків даних, що спільно використовуються " [4]. У наш час публікацію цієї статті прийнято вважати поворотним пунктом в історії розвитку систем баз даних, хоча варто помітити, що ще раніше була запропонована модель, заснована на множинах [2]. Мети створення реляційної моделі формулювалися в такий спосіб.

- Забезпечення більш високого ступеня незалежності від даних. Прикладні програми не повинні залежати від змін внутрішнього представлення даних, зокрема від змін організації файлів, переупорядкування записів і шляхів доступу.
- Створення міцного фундаменту для рішення семантичних питань, а також проблем несуперечності й надмірності даних. Зокрема, у статті Кодда вводиться поняття нормалізованих відношень, тобто відношень без груп, що повторюються. (Процес нормалізації обговорюється в главі 8.)
- Розширення мов керування даними за рахунок включення операцій над множинами.

Хоча інтерес до реляційної моделі обумовлений декількома різними причинами, все-таки найбільш значні дослідження були проведені в рамках трьох проектів з дуже різною долею [11].

Перший з них розроблявся наприкінці 1970-х років у дослідницькій лабораторії корпорації IBM у місті Сан-Хосе, штат Каліфорнія, під керівництвом Астрахана (Astrahan), результатом чого стало створення системи за назвою "System R", що була прототипом дійсної реляційної СУБД. Цей проект був задуманий з метою одержання реальних доказів можливості практичного застосування реляційної моделі за допомогою реалізації структур даних і операцій, що передбачаються нею. Цей проект також зарекомендував себе як найважливіше джерело інформації про такі

проблеми реалізації, як керування транзакціями, керування паралельною роботою, технологія відновлення, оптимізація запитів, забезпечення безпеки й цілісності даних, урахування людського фактора й розробка інтерфейсу користувача. Виконання проекту стимулювало публікацію багатьох науково-дослідних статей і створення інших прототипів реляційних СУБД. Зокрема, робота над проектом System R дала поштовх проведенню наступних найважливіших розробок:

- створення мови структурованих запитів SQL (цю назву вимовляють або по буквах "S-Q-L", або (іноді) за допомогою мнемонічного імені "See-Quel")» яка з тих пор придбала статус формального стандарту ISO (International Organization for Standardization) і в цей час є фактичним стандартом мови реляційних СУБД;
- створення різних комерційних реляційних СУБД, які вперше з'явилися на ринку наприкінці 1970-х і початку 1980-х років, таких як DB2 і SQL/DS корпорації IBM, а також Oracle корпорації Oracle Corporation.

Другим проектом, що зіграв помітну роль у розробці реляційної моделі даних, був проект INGRES (INteractive GRaphics REtrieval System), робота над яким проводилася в Каліфорнійському університеті (місто Беркли) майже в той же самий час, що й над проектом System R. Проект INGRES включав розробку прототипу реляційної СУБД із концентрацією основної уваги на тих же загальних цілях, що й проект System R. Ці дослідження привели до появи академічної версії INGRES, що внесла істотний вклад у загальне визнання реляційної моделі даних.

Третім проектом була система Peterlee Relational Test Vehicle наукового центра корпорації IBM. Цей проект був більш теоретичним, чим проекти System R і INGRES. Його результати мали дуже велике й навіть принципове значення, особливо в таких галозях, як обробка запитів і оптимізація, а також функціональні розширення системи.

Комерційні системи на основі реляційної моделі даних почали з'являтися наприкінці 1970-х – початку 1980-х років. У цей час існує кілька сотень типів різних реляційних СУБД, як для мейнфреймів, так і для персональних комп'ютерів, хоча багато хто з них не повністю відповідають точному визначенню реляційної моделі даних. Прикладами реляційних СУБД для персональних комп'ютерів є СУБД Access і FoxPro фірми Microsoft, Paradox фірми Corel Corporation, InterBase і BDE фірми Borland, а також R:Base фірми R:Base Technologies.

Завдяки популярності реляційної моделі багато нереляційних систем тепер забезпечуються реляційним інтерфейсом користувача, незалежно від базової моделі, що використовується. Основна мережна СУБД, система

IDMS фірми Computer Associates, тепер називається CA-IDMS/SQL і підтримує реляційне представлення даних. Іншими СУБД для мейнфреймів, у яких підтримуються деякі реляційні компоненти, є Model 204 фірми Computer Corporation of America і AD ABAS фірми Software AG.

Крім того, пізніше були запропоновані деякі розширення реляційної моделі даних, призначені для найбільш повного й точного вираження змісту даних [6], для підтримки обє'ктно-орієнтованих понять, а також для підтримки дедуктивних можливостей.

3.2 Термінологія, що використовується

Реляційна модель заснована на математичному понятті *відношення*, фізичним представленням якого є *таблиця*. Справа в тому, що Кодд, будучи досвідченим математиком, широко використовував математичну термінологію, особливо з теорії множин і логіки предикатів. У цьому розділі пояснюється термінологія, яка використовується в реляційній моделі, а також її основні структурні поняття.

Хоча реляційні СУБД практично витиснули СУБД іншої структури й на світовому ринку програмного забезпечення, і на вітчизняному також, проте існує різночитання в самій термінології реляційних бах даних. Як уже було сказано вище, реляційна база даних складається з таблиць, які називають *відношеннями* (по-англійському *relation*). Іноді в літературі пропонують називати таблиці *реляціями*, по аналогу з англомовним звучанням. Але такі назви скоріше забруднюють мову, чим сприяють розумінню теорії баз даних. Деякі автори пропонують називати таблиці *реляційними відношенями*. Це припустима назва, хоча вона якоюсь мірою є тавтологією (масло масляне). У даному курсі лекцій таблиці реляційної бази даних називаються просто *відношеннями*.

Столбцы таблиці називаються *атрибутами відношення*; але часто в різних реляционных СУБД атрибути відношення називають *полями* (fields). Термін атрибут (властивість, характеристика) має цілком певний зміст. Наприклад, відношення ПАЦІЄНТ може мати атрибути: ПРІЗВИЩЕ. ІМ'Я, ПО_БАТЬКОВІ, ДАТА_НАРОДЖЕННЯ, СТАТЬ й т.д.

Рядки таблиці називаються *кортежами відношення* або *записами* (records).

Можна розглядати відношення як набір зв'язків між *n* атрибутами, тобто як *n*-арний атрибутний зв'язок. У відповідне представлення даних включаються тільки комбінації зв'язків атрибутів, які мають зміст і/або припустимі. Нижче наведений ряд формальних визначень:

а) Доменом називається сукупність однотипних значень даних. Прикладами є домен грошових сум, домен імен, домен цілих чисел і т.д.

б) Термін атрибут був уведений вище для позначення властивості об'єкта. Кілька атрибутів можуть одержувати значення з одного домена. Таким чином, значення домена по-різному інтерпретуються в різних атрибутах, що використовуються при описі інформаційної структури. Наприклад, атрибути «зарплата» і «комісійні» об'єкта (відношення) «службовці» одержують значення з домена грошових сум.

в) Нехай є n доменов D1, D2, ..., Dn. (необов'язково різних). Відношення R визначається як множина упорядкованих n-ок (кортежів), що є підмножиною декартова добутку доменов. Домен (множина) D, представлений у кортежах і-м елементом. Вимогу впорядкованості елементів кортежу можна усунути, ідентифікуючи в кортежі кожне входження домена унікальним ім'ям атрибута.

3.3 Структура реляційних даних

Відношення. Плоска таблиця, що складається з рядків і стовпців

У будь-який реляційній СУБД передбачається, що користувач сприймає базу даних як набір таблиць. Однак варто підкреслити, що це сприйняття відноситься тільки до логічної структури бази даних, тобто до зовнішнього й до концептуального рівням архітектури ANSI-SPARC, що розглядалася нами в розділі 2.4. Подібне сприйняття не відноситься до фізичної структури бази даних, що може бути реалізована за допомогою різних структур зберігання.

Атрибут. Іменований стовпець відношення

У реляційній моделі відношення використовуються для зберігання інформації про об'єкти, представлені у базі даних. Відношення звичайно має вигляд двовимірної таблиці, у якій рядки відповідають окремим записам, а стовпці – атрибутам. При цьому атрибути можуть розташовуватися в будь-якому порядку – незалежно від їх переупорядкування відношення буде залишатися тим самим, а тому мати той же зміст.

Наприклад, інформація про відділення компанії може бути представлена відношенням Branch, що включає стовпці з атрибутами `branchNo` (Номер відділення), `street` (Вулиця), `city` (Місто) і `postcode` (Поштовий індекс). Інформація про працівників компанії може

бути представлена відношенням Staff (Персонал), що включає стовпці з атрибутами staffNo (Табельний номер співробітника), fName (Ім'я), lName (Прізвище), position (Посада), sex (Стать), DOB (Дата народження), salary (Зарплата), branchNo (Номер відділення).

На рис. 3.1 показані приклади відношень Branch і Staff. Як видно із цього приклада, кожний стовпець містить значення того самого атрибута — наприклад, стовпець branchNo містить тільки номера існуючих відділень компанії.

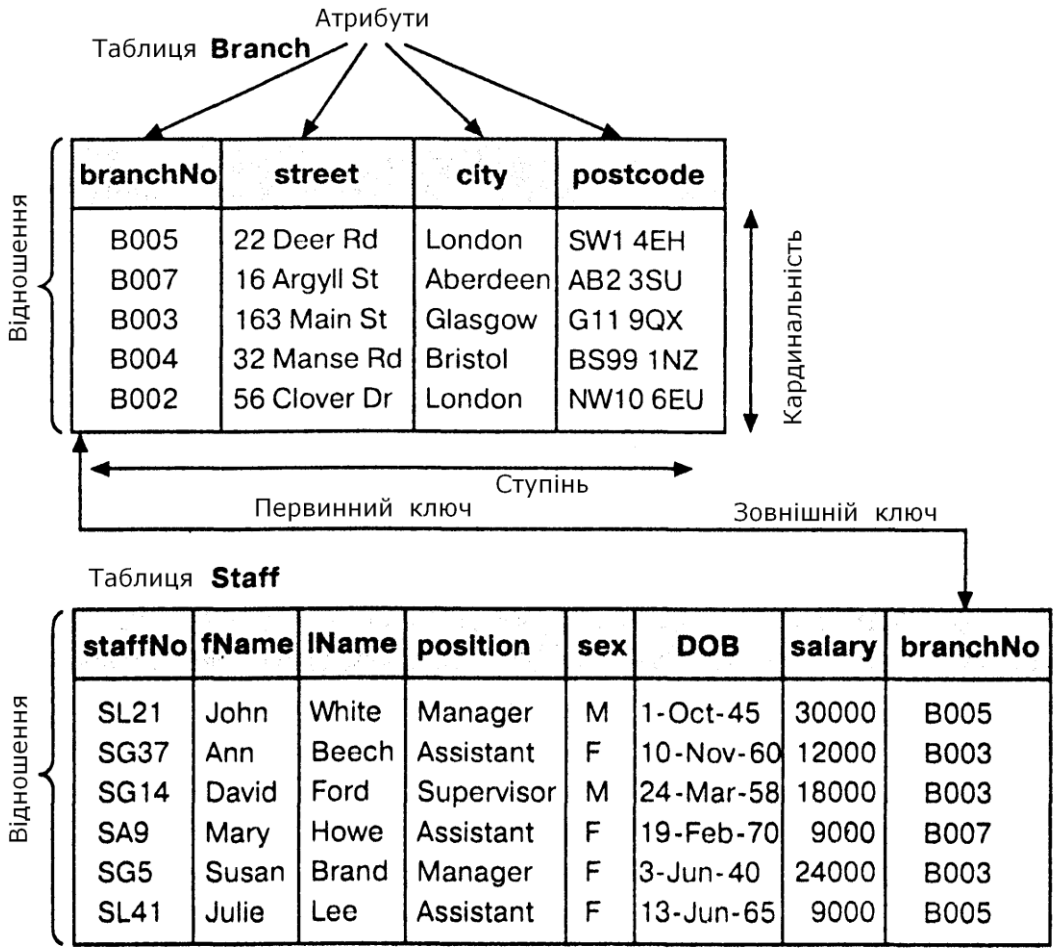


Рис. 3.1. Структура відношень Branch і Staff

Домен. Набір припустимих значень одного або декількох атрибутів.

Домени являють собою надзвичайно потужний компонент реляційної моделі. Кожний атрибут реляційної бази даних визначається на деякому домені. Домени можуть відрізнятися для кожного з атрибутів, але два й більше атрибути можуть визначатися на тому самому домені. У табл. 5.1 представлені домени для деяких атрибутів відношень Branch і Staff. Зверніть увагу, що в будь-який момент часу в доменах можуть існувати значення, які реально не представлені значеннями відповідного атрибута.

Поняття домена має велике значення, оскільки завдяки йому користувач може централізовано визначати зміст і джерело значень, які можуть одержувати атрибути. У результаті при виконанні реляційної операції системі доступно більше інформації, що дозволяє уникнути в ній семантично некоректних операцій.

Наприклад, безглуздо порівнювати назву вулиці з номером телефону, навіть якщо для обох цих атрибутів визначеннями доменів є символьні рядки. З іншого боку, помісячна орендна плата об'єкта нерухомості й кількість місяців, протягом яких він здавався в оренду, належать різним доменам (перший атрибут має грошовий тип, а другий – числовий). Однак множення значень із цих доменів є припустимою операцією. Як випливає із цих двох прикладів, забезпечити повну реалізацію поняття домена зовсім непросто, тому в багатьох реляційних СУБД вони підтримуються не повністю, а лише частково.

Таблиця 3.1 Домени деяких атрибутів відношень `Branch` і `Staff`

Атрибут	Ім'я домену	Вміст домену	Визначення домену
branchNo	BranchNumbers	Множина всіх припустимих номерів відділень компанії	Символьний; розмір 4, діапазон 'B001' - 'B999'
street	StreetNames	Множина всіх назв вулиць у Великобританії	Символьний; розмір 25
city	CityNames	Множина всіх назв міст у Великобританії	Символьний; розмір 15
postcode	Postcodes	Множина всіх поштових індексів у Великобританії	Символьний; розмір 8
sex	Sex	Позначення статі людини	Символьний; розмір 1, значення 'M' або 'F'
DOB	DatesOfBirth	Всі можливі значення дати народження працівника компанії	Дата; діапазон від 1-01-1930,формат dd-mm-yyyy
salary	Salaries	Всі можливі значення річної заробітної плати працівника компанії	Грошовий; 7 цифр, діапазон 6000.00-40000.00

Кортеж. Рядок відношення.

Елементами відношення є *кортежі*, або рядки, таблиці. Кортежі можуть розташовуватися в будь-якому порядку, при цьому відношення буде залишатися тим же самим, а виходить, і мати той же зміст.

Опис структури відношення разом зі специфікацією доменів і будь-якими іншими обмеженнями можливих значень атрибутів іноді називають

його *заголовком* (*змістом* або *інтенсіоналом* (intension)). Звичайно воно є фіксованим, доти, поки зміст відношення не зміниться за рахунок додавання до нього додаткових атрибутів. Кортежі називаються *розширенням* або *екстенсіоналом* (extension), *станом* (state) або *тілом відношення*, що згодом змінюється.

Ступінь. Ступінь відношення визначається кількістю атрибутів, що воно містить.

Відношення `Branch`, показане на рис. 5.1, має чотири атрибути, і, отже, його *ступінь* дорівнює чотирьом. Це значить, що кожний рядок таблиці є чотириелементним кортежем, тобто кортежем, що містить чотири значення. Відношення тільки з одним атрибутом має ступінь 1 і називається *унарным* (unary) відношенням (або одноелементним кортежем). Відношення із двома атрибутами називається *бінарним* (binary), відношення із трьома атрибутами — *тернарным* (ternary), а для відношення з більшою кількістю атрибутів використовується термін *n*-арное (*n*-ary). Визначення ступеня відношення є частиною заголовка відношення.

Кардинальність (або потужність). Кількість кортежів, що міститься у відношенні.

Кількість кортежів, що міститься у відношенні, називаються *кардинальністю відношення*. Ця характеристика змінюється при кожному додаванні або видаленні кортежів. Кардинальність є властивістю тіла відношення й визначається поточним станом відношення в довільно взятий момент.

І нарешті, ми підійшли до визначення самої реляційної бази даних.

Реляційна база даних. Набір нормалізованих відношень, які розрізняються по іменах.

Реляционная база даних складається з відношень, структура яких визначається за допомогою особливих методів, що називаються *нормалізацією* (normalization). Обговорення цього питання буде продовжено в главі 8.

3.3.1 Альтернативна термінологія

Термінологія, яка використовується в реляційній моделі, часом може привести до плутанини, оскільки крім запропонованих двох наборів термінів існує ще один — третій. Відношення в ньому називається *файлом* (file), кортежі — *записами* (records), а атрибути — *полями* (fields). Ця термінологія заснована на тім факті, що фізично реляційна СУБД може зберігати кожне відношення в окремому файлі. У табл. 5.2 показані відповідності, що існують між трьома згаданими вище групами термінів.

Таблиця 3.2. Альтернативні варіанти термінів у реляційній моделі

Офіційні терміни	Альтернативний варіант 1	Альтернативний варіант 2
Відношення	Таблиця	Файл
Кортеж	Рядок	Запис
Атрибут	Стовпець	Поле

3.3.2 Математичні відношення

Для розуміння дійсного сенсу терміна *відношення* розглянемо кілька математичних понять. Допустимо, існують дві множини, $\mathbf{D_1}$ і $\mathbf{D_2}$, де $\mathbf{D_1}$**={2,4}** і $\mathbf{D_2}$**={1,3,5}**. *Декартовим добутком* цих двох множин (позначається як $\mathbf{D_1 \times D_2}$) називається набір із всіх можливих упорядкованих пар, у яких першим іде елемент множини $\mathbf{D_1}$, а другим — елемент множини $\mathbf{D_2}$. Альтернативний спосіб вираження цього добутку полягає в пошуку всіх комбінацій елементів, у яких першим іде елемент множини $\mathbf{D_1}$, а другим – елемент множини $\mathbf{D_2}$. У даному прикладі одержимо наступний результат:

$D_1 \times D_2 = \{(2,1), (2,3), (2,5), (4,1), (4,3), (4,5)\}$

Будь-яка підмножина цього декартова добутку є відношенням. Наприклад, у ньому можна виділити відношення **R**, показане нижче.

$R = \{(2,1),(4,1)\}$

Для визначення тих можливих пар, які будуть входити у відношення, можна задати деякі умови їхньої вибірки. Наприклад, якщо звернути увагу на те, що відношення **R** містить всі можливі пари, у яких другий елемент дорівнює 1, то визначення відношення **R** можна сформулювати в такий спосіб:

$R = \{(x,y) | x \in D_1, y \in D_2 \text{ і } y = 1\}$

На основі тих же множин можна сформувати інше відношення, **S**, у якому перший елемент завжди повинен бути у два рази більше другого. Тоді визначення відношення S можна сформулювати так:

$S = \{(x,y) | x \in D_1, y \in D_2 \text{ і } x = 2y\}$

У цьому прикладі заданій умові відповідає тільки одна можлива пара даного декартова добутку:

$S = \{(2,1)\}$

Поняття відношення можна легко поширити й на три множини. Нехай є три множини — $\mathbf{D_1}$, $\mathbf{D_2}$ і $\mathbf{D_3}$. Декартовмй добуток $\mathbf{D_1 \times D_2 \times D_3}$ цих трьох множин є набором, що складається із всіх можливих упорядкованих трійок елементів, у яких першим іде елемент множини $\mathbf{D_1}$, другим – елемент множини $\mathbf{D_2}$, а третім — елемент множини $\mathbf{D_3}$. Будь-яка підмножина цього декартова добутку є відношенням. Розглянемо наступний приклад трьох множин і обчислимо їх декартовий добуток:

$D_1 = \{(1,3)\}$, $D_2 = \{(2,4)\}$, $D_3 = \{(5,6)\}$

$D_1 \times D_2 \times D_3 = \{(1,2,5), (1,2,6), (1,4,5), (1,4,6), (3,2,5), (3,2,6), (3,4,5), (3,4,6)\}$

Будь-яка підмножина з наведених вище впорядкованих трійок елементів є відношенням. Збільшуючи кількість множин, можна дати узагальнене визначення відношення на ***n*** доменах. Нехай є п множин $\mathbf{D_1}$, $\mathbf{D_2}$, ..., $\mathbf{D_n}$. Декартовий добуток для цих ***n*** множин можна визначити в такий спосіб:

$D_1 \times D_2 \times \ldots \times D_n = \{(d_1, d_2, \ldots, d_n) \mid d_1 \in D_1, d_2 \in D_2, \ldots, d_n \in D_n\}$

Будь-яка множина ***n***-арних кортежів цього декартова добутку є відношенням ***n*** множин. Зверніть увагу на те, що для визначення цих відношення необхідно вказати множини, або *домени,* з яких вибираються значення.

3.3.3 Відношення в базі даних

Використовуючи зазначені концепції в контексті бази даних, ми одержимо наступне визначення реляційної схеми.

Реляційна схема. Іменоване відношення, що визначене на основі множини пар атрибутів і імен доменів.

Наприклад, для атрибутів $\mathbf{A_1, A_2, ..., A_n}$ з доменами $\mathbf{D_l, D_2, ..., D_n}$ реляційною схемою буде множина $\mathbf{\{A_1:D_l, A_2:D_2, \ldots, A_n:D_n\}}$. Відношення **R**, задане реляційною схемою **S**, є множиною відображень імен атрибутів на відповідні їм домени. Таким чином, відношення **R** є множиною таких ***n***-арних кортежів:

$\mathbf{\{A_1:d_l, A_2:d_2, ..., A_n:d_n\}}$, де $\mathbf{d_1 \in D_1, d_2 \in D_2, ..., d_n \in D_n}$.

Кожний елемент ***n***-арного кортежу складається з атрибута й значення цього атрибута. Звичайно при запису відношення у вигляді таблиці імена атрибутів перераховуються в заголовках стовпців, а кортежі утворюють рядки формату $\mathbf{(d_l, d_2, \ldots, d_n)}$, де кожне значення береться з відповідного домена. Таким чином, у реляційній моделі відношення можна представити як довільну підмножину декартова добутку доменів

атрибутів, тоді як таблиця – це всього лише фізичне представлення такого відношення.

У прикладі, показаному на рис. 3.1, відношення `Branch` має атрибути `branchNo`, `street`, `city` і `postcode` з відповідними їм доменами. Відношення `Branch` являє собою довільну підмножину декартова добутку доменів або довільну множину чотириелементних кортежів, у яких першим іде елемент із домену `BranchNumbers`, другим — елемент із домену `StreetNames` і т.д. Наприклад, один із чотириелементних кортежів може мати такий вигляд:

```
{(B005,22 Deer Rd, London, SW1 4EH)}
```

Цей же кортеж можна записати в більш коректній формі:

```
{(branchNo: B005, street: 22 Deer Rd, city: London,
postcode: SW1 4EH)}
```

Ця конструкція називається *екземпляром віднішення.* Таблиця `Branch` являє собою зручний спосіб запису всіх чотириелементних кортежів, що утворять відношення в деякий заданий момент часу. Це зауваження пояснює, чому рядки таблиці в реляційної моделі називаються кортежами. Таким чином, схему має не тільки відношення, але й реляційна база даних.

Якщо $\mathbf{R_1}$, $\mathbf{R_2}$, ..., $\mathbf{R_n}$ — набір реляційних схем, то схему реляційної бази даних (або просто реляційну схему) **R** можна представити в такий спосіб:

$$R = \{R_1, R_2, ..., R_n)$$

3.3.4 Властивості відношень

Відношення має наступні характеристики.

- Відношення має ім'я, що відрізняється від імен всіх інших відношень у реляційній схемі.
- Кожний осередок відношення містить тільки одне елементарне (неподільне) значення.
- Кожний атрибут має унікальне ім'я.
- Значення атрибута беруться з того самого домена.
- Кожний кортеж є унікальним, тобто дублікатів кортежів бути не може.
- Порядок проходження атрибутів не має значення.

- Теоретично порядок проходження кортежів у відношенні не має значення. (Але практично цей порядок може істотно вплинути на ефективність доступу до них.)

Для ілюстрації змісту цих обмежень знову розглянемо відношення `Branch`, показане на рис. 5.1. Оскільки кожний осередок повинен містити тільки одне значення, не допускається зберігання в одному і тому ж осередку двох поштових індексів того самого відділення компанії. Інакше кажучи, відношення не можуть містити груп, що повторюються. Про відношення, що має таку властивість, говорять, що воно *нормалізовано*, або перебуває в *першій нормальній формі*. (Більш докладно нормальні форми розглядаються в главі 8.)

Імена стовпців, які звичайно вказують над стовпцями, відповідають атрибутам відношення. Значення атрибута `branchNo` беруться з домена `BranchNumbers` — розміщення в цьому стовпці інших значень, наприклад поштового індексу, не допускається. У відношенні не повинне бути кортежів, які повторюються. Наприклад, рядок: `(B005, 22 Deer Rd, London, SW1 4EH)`, – може бути представлена у відношенні тільки один раз.

Стовпці можна міняти місцями за умови, що ім'я атрибута переміщається разом з його значеннями. Таблиця усе ще буде представляти те ж відношення, якщо атрибут `city` розташований у ній перед атрибутом `postcode`, хоча для зручності сприйняття розумніше було б розташовувати окремі частини адреси у звичайному порядку. Крім того, рядки також можна міняти місцями довільним образом (наприклад, перемістити рядок відділення B05 на місце рядка відділення B04); саме відношення при цьому залишиться колишнім.

Більша частина властивостей реляційних відношень походить від властивостей математичних відношень.

- При обчисленні декартова добутку множин із простими однозначними елементами (наприклад, числовими значеннями) кожний елемент у кожному кортежі має єдине значення. Аналогічно, кожний осередок відношення містить тільки одне значення. Однак математичне відношення не має потреби в нормалізації. Кодд запропонував заборонити застосування груп, що повторюються, з метою спрощення реляційної моделі даних.
- Набір можливих значень для даної позиції відношення визначається множиною, або доменом, на якому визначається ця позиція. У таблиці всі значення в кожному стовпці повинні походити від того самого домена, визначеного для даного атрибута.

- У множини немає елементів, що повторюються. Аналогічно, відношення не може містити кортежі-дублікати.
- Оскільки відношення є множиною, то порядок елементів не має значення. Отже, порядок кортежів у відношенні несуттєвий.

Однак у математичному відношенні порядок проходження елементів у кортежі має значення. Наприклад, припустима впорядкована пара значень (1, 2) зовсім відмінна від упорядкованої пари (2, 1). Це твердження невірно для відношень у реляційній моделі, де спеціально обмовлюється, що порядок атрибутів несуттєвий.

Справа в тому, що заголовки стовпців однозначно визначають, до якого саме атрибута відноситься дане значення. Наслідком цього факту є положення про те, що порядок проходження заголовків стовпців у заголовку відношення несуттєвий. Однак, якщо структура відношення вже визначене, то порядок елементів у кортежах тіла відношення повинен відповідати порядку імен атрибутів.

3.3.5 Реляційні ключі

Як зазначено вище, у відношенні не повинно бути кортежів, що повторюються. Тому необхідно мати можливість унікальної ідентифікації кожного окремого кортежу відношення за значеннями одного або декількох атрибутів (які називаються *реляційними ключами*). У цьому розділі описується термінологія, що використовується для позначення реляційних ключів.

Суперключ (superkey). Атрибут або множина атрибутівів, що єдиним образом ідентифікує кортеж даного відношення.

Суперключ однозначно позначає кожний кортеж у відношенні. Але суперключ може містити додаткові атрибути, які необов'язкові для унікальної ідентифікації кортежу, тому нас будуть цікавити суперключі, що складаються тільки з тих атрибутів, які дійсно необхідні для унікальної ідентифікації кортежів.

Потенційний ключ. Суперключ, що не містить підмножини, що також є суперключем даного відношення.

Потенційний ключ `K` для даного відношення `R` має дві властивості.

- Унікальність. У кожному кортежі відношення `R` значення ключа `K` єдиним образом ідентифікують цей кортеж.
- Неприводимість. Ніяка припустима підмножина ключа `K` не має властивості унікальності.

Відношення може мати кілька потенційних ключів. Якщо ключ складається з декількох атрибутів, то він називається *складеним ключем*. Розглянемо відношення `Branch`, показане на рис. 5.1. Конкретне значення атрибута `city` може визначати відразу кілька відділень компанії (наприклад, у Лондоні розташовано два відділення). Тому даний атрибут не може бути обраний як потенційний ключ. З іншого боку, оскільки в навчальному проекті *DreamHome* для кожного відділення компанії задається його унікальний номер, кожне значення цього номера (атрибут `branchNo`) може визначати не більше одного кортежу (у відношенні `Branch`), тому `branchNo` є потенційним ключем. Аналогічно, атрибут `postcode` також є потенційним ключем цього відношення.

Тепер розглянемо відношення `Viewing` (Огляд), що містить інформацію про ознайомчи огляди об'єктів нерухомості потенційними орендарями. Це відношення включає атрибути номера орендаря (`clientNo`), номера об'єкта нерухомості (`propertyNo`), дати перегляду (`viewDate`) і, можливо, коментарі (`comment`). По заданому номеру орендаря (`clientNo`) можна вибрати відомості про декілька оглядів різних об'єктів нерухомості. Аналогічно, по заданому номеру об'єкта нерухомості (`propertyNo`) можна вибрати відомості про декілька орендарів, які оглядали цей об'єкт нерухомості. Отже, сам по собі номер орендаря (`clientNo`) або номер об'єкта нерухомості (`propertyNo`) не може бути використаний як потенційний ключ. Однак комбінація атрибутів `clientNo` і `propertyNo` ідентифікує не більше одного кортежу. Якщо допустити можливість того, що орендар може оглядати той самий об'єкт кілька разів, то до даного складеного ключа буде потрібно додати дату (`viewDate`). Однак у даному прикладі передбачається, що цього не потрібно.

Зверніть увагу на те, що будь-який конкретний набір кортежів відношення не можна використати для доказу того, що якийсь атрибут або комбінація атрибутів є потенційним ключем. Той факт, що в деякий момент часу не існує значень-дублікатів, зовсім не означає, що їх не може бути взагалі. Однак наявність значень-дублікатів у конкретному існуючому наборі кортежів цілком може бути використана для демонстрації того, що деяка комбінація атрибутів не може бути потенційним ключем.

Для ідентифікації потенційного ключа потрібно знати зміст атрибутів, що використовуються, в "реальному світі"; тільки це дозволить обґрунтовано ухвалити рішення щодо можливості існування значень-дублікатів. Тільки виходячи з подібної семантичної інформації можна гарантувати, що деяка комбінація атрибутів є потенційним ключем відношення.

Наприклад, на підставі представлених на рис. 3.1 даних можна вирішити, що підходящим потенційним ключем для відношення Staff цілком може бути атрибут lName, що містить прізвище співробітника. Однак, хоча в цей момент в організації є тільки один співробітник із прізвищем White, при зарахуванні в організацію нового співробітника із прізвищем White атрибут lName уже не можна буде використати як потенційний ключ.

Первинний ключ. Потенційний ключ, що обраний для унікальної ідентифікації кортежів усередині відношення.

Оскільки відношення не містить кортежів-дублікатів, завжди можна унікальним образом ідентифікувати кожний його рядок. Це значить, що відношення завжди має первинний ключ. У найгіршому разі вся множина атрибутів може використовуватись як первинний ключ, але звичайно, щоб розрізнити кортежі, досить використати трохи меншу підмножину атрибутів.

Потенційні ключі, які не обрані як первинний ключ, називаються *альтернативними ключами*. Якщо у відношенні Branch вибрати як первинний ключ атрибут branchNo, то альтернативним ключем цього відношення буде атрибут postcode. У відношенні Viewing є тільки один потенційний ключ, що складається з атрибутів clientNo і propertyNo, тому дане сполучення атрибутів автоматично утворить його первинний ключ.

Зовнішній ключ. Атрибут або множина атрибутів усередині відношення, що відповідає потенційному ключу якогось (може бути, того ж самого) відношення.

Якщо якийсь атрибут присутній у декількох відношеннях, то його наявність звичайно відбиває певний зв'язок між кортежами цих відношень. Наприклад, атрибут branchNo навмисно включений у відношення Branch і Staff для встановлення зв'язку між відомостями про відділення компанії й відомостями про співробітників, які працюють у кожному з відділень. У відношенні Branch атрибут branchNo є первинним ключем, а у відношенні Staff він уведений для встановлення відповідності між відомостями про співробітників і відомостями про ті відділення компанії, у яких вони працюють.

У відношенні Staff атрибут branchNo є зовнішнім ключем. У такому випадку говорять, що атрибут branchNo у відношенні Staff посилається на первинний ключ, тобто на атрибут branchNo, у *базовому відношенні* (Branch). (Базове відношення іноді називають *цільовим*

відношенням.) Як буде показано в наступній главі, ці загальні атрибути відіграють важливу роль у маніпулюванні даними.

3.3.6 Представлення схем у реляційній базі даних

Реляційна база даних може складатися з довільної кількості нормалізованих відношень. Реляційні схеми для тієї частини навчального проекту *DreamHome*, у якій міститься й обробляється інформація про оренду власності, виглядають так:

```
Branch (branchNo, street, city, postcode)

Staff (staffNo, fName, lName, position, sex, DOB,
       salary, branchNo)

PropertyForRent (propertyNo, street, city, postcode,
         type, rooms, rent, ownerNo, staffNo,
         branchNo)

Client (clientNo, fName, IName, telNo, prefType,
          maxRent)

PrivateOwner (ownerNo, fName, lName, address, telNo)

Viewing (clientNo, propertyNo, viewDate, comment)

Registration (clientNo, branchNo, staffNo, dateJoined)
```

Загальноприйняте позначення реляційної схеми включає ім'я відношення, за яким (у дужках) розташовуються імена атрибутів. При цьому первинний ключ (звичайно) підкреслюється.

Концептуальною моделлю, або *концептуальною схемою*, називається множина всіх реляційних схем бази даних. У табл. 5.3-5.9 показаний приклад визначення реляційної схеми.

Таблиця 3.3. Приклад деякого поточного стану бази даних навчального проекту *DreamHome*. Таблиця `Branch`

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	Gil 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

Таблиця 3.4. Приклад деякого поточного стану бази даних навчального проекту *DreamHome*. Таблиця `Staff`

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	l-10-1945	30000	B005
SG37	Ann	Beech	Assistant	F	10-11-1960	12000	B003
SG14	David	Ford	Supervisor	M	24-03-1958	18000	B003
SA9	Mary	Howe	Assistant	F	19-02-1970	9000	B007
SG5	Susan	Brand	Manager	F	3-06-1940	24000	B003
SL41	Julie	Lee	Assistant	F	13-06-1965	9000	B005

Таблиця 3.5. Приклад деякого поточного стану бази даних навчального проекту *DreamHome*. Таблиця `PropertyForRent`

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberde	AB7 5SU	House	6	650	C046	SA9	B007
PL94	6 Argyll St	London	NW2	Flat	4	400	CO87	SL41	B005
PG4	6 Lawrence St	Glasgo	Gil 9QX	Flat	3	350	C040		B003
PG36	2 Manor Rd	Glasgo	G32 4QX	Flat	3	375	C093	SG37	B003
PG21	18 Dale Rd	Glasgo	G12	House	5	600	CO87	SG37	B003
PG16	5 Novar Dr	Glasgo	G12 9AX	Flat	4	450	C093	SG14	B003

Таблиця 3.6. Приклад деякого поточного стану бази даних навчального проекту *DreamHome*. Таблиця `Client`

clientNo	fName	lName	telNo	prefType	maxRent
CR76	John	Kay	0207-774-5632	Flat	425
3056	Aline	Stewart	0141-848-1825	Flat	350
CR74	Mike	Ritchie	01475-392178	House	750
CR62	Mary	Tregear	01224-196720	Flat	600

Таблиця 3.7. Приклад деякого поточного стану бази даних навчального проекту *DreamHome*. Таблиця `PrivateOwner`

ownerNo	fName	IName	address	telNo
3046	Joe	Keogh	2 Fergus Dr, Aberdeen AB2 7SX	01224-861212
3087	Carol	Farrel	6 Achray St, Glasgow G32 9DX	0141-357-7419
3040	Tina	Murphy	63 Well St, Glasgow G42	0141-943-1728
3093	Tony	Shaw	12 Park PI, Glasgow G4 0QR	0141-225-7025

Таблиця 3.8. Приклад деякого поточного стану бази даних навчального проекту *DreamHome*. Таблиця `Viewing`

clientNo	propertyNo	viewDate	comment
CR56	PA14	24-05-2001	too small
CR76	PG4	20-04-2001	too remote
CR56	PG4	26-05-2001	
CR62	PA14	14-05-2001	no dining room
CR56	PG36	28-04-2001	

Таблиця 3. 9. Приклад деякого поточного стану бази даних навчального проекту *DreamHome*. Таблиця `Registration`

clientNo	branchNo	staffNo	dateJoined
CR76	B005	SL41	2-01-2001
CR56	B003	SG37	ll-04-2000
CR74	B003	SG37	16-11-1999
CR62	B007	SA9	7-03-2000

3.4 Реляційна цілісність

У попередньому розділі була розглянута структурна частина реляційної моделі даних. Як згадувалося в розділі 2.6, модель даних має дві інші частини: керуючу частину, що визначає типи припустимих операцій з даними, і набір обмежень цілісності, які гарантують коректність даних. У цьому розділі розглядаються реляційні обмеження цілісності, а в наступному – реляційні операції керування даними.

Оскільки кожний атрибут пов'язаний з деяким доменом, для множини припустимих значень кожного атрибута відношення визначаються так звані *обмеження домена.* Крім цього, задаються два важливі *правила цілісності,* які, по суті, є обмеженнями для всіх припустимих станів бази даних. Ці два основні правила реляційної моделі називаються *цілісністю сутностей* і *посилальною цілісністю.* Однак, перш ніж приступитися до вивчення цих правил, варто розглянути поняття порожніх значень.

3.4.1 Порожні значення

Порожнє значення. Указує, що значення атрибута в даний момент невідомо або неприйнятно для цього кортежу.

Порожнє значення (яке умовно позначається як NULL) варто розглядати як логічну величину "невідомо". Інакше кажучи, або це

значення не входить в область визначення деякого кортежу, або ніяке значення ще не задане. Ключове слово NULL являє собою спосіб обробки неповних або незвичайних даних. Однак NULL не слід розуміти як нульове чисельне значення або заповнений пробілами текстовий рядок. Нулі й пробіли являють собою деякі значення, тоді як ключове слово NULL позначає відсутність якого-небудь значення. Тому NULL варто розглядати інакше, чим інші значення. Деякі автори використають термін "значення NULL", але насправді NULL не є значенням, а лише позначає його відсутність, тому термін "значення NULL" можна використовувати лише з певними застереженнями.

Наприклад, у представленому в табл. 5.8 відношенні `Viewing` атрибут `Comment` може не мати певного значення доти, поки потенційний орендар не відвідає даний об'єкт нерухомості й не повідомить свою думку агентству. У противному випадку для представлення цього стану без використання ключового слова NULL буде потрібно або ввести якісь фіктивні дані, або створити додаткові атрибути, які можуть бути безглуздими для користувачів. У даному прикладі можна спробувати представити відсутній коментар за допомогою значення -1. Крім того, можна додати у відношення `Viewing` ще один атрибут `hasCommentBeenSupplied` (чи є коментар) зі значенням Y (Yes), якщо коментар є, і N (No) — у противному випадку. Однак обидва цих підходу можуть тільки заплутати користувача.

Використання NULL може викликати проблеми на етапі реалізації. Труднощі виникають через те, що реляційна модель заснована на вирахуванні предикатів першого порядку, що має двозначну, або булєву, логіку, тобто припустимими є тільки два значення: істина й неправда. Застосування NULL означає, що доведеться вести роботу з логікою більш високого порядку, наприклад тризначною або навіть чотиризначною [9], [10].

Використання поняття NULL у реляційній моделі є спірним питанням. Кодд [10] розглядає поняття NULL як складову частину цієї моделі, а інші фахівці вважають цей підхід неправильним, думаючи, що проблема відсутньої інформації ще не до кінця зрозуміла.

Тепер можна приступитися до вивчення реляційних обмежень цілісності.

3.4.2 Цілісність сутностей

Перше обмеження цілісності стосується первинних ключів базових відношень. Тут базове відношення визначається як відношення, що відповідає деякій сутності в концептуальній схемі (див. розділ 2.1)..

Цілісність сутностей. У базовому відношенні жоден атрибут первинного ключа не може містити відсутні значення, які позначаються як NULL.

За визначенням, первинний ключ – це мінімальний ідентифікатор, що використовується для унікальної ідентифікації кортежів. Це значить, що ніяка підмножина первинного ключа не може бути достатньою для унікальної ідентифікації кортежів. Якщо допустити присутність NULL у будь-якій частині первинного ключа, це рівносильно твердженню, що не всі його атрибути необхідні для унікальної ідентифікації кортежів, що суперечить визначенню первинного ключа.

Наприклад, оскільки `branchNo` є первинним ключем відношення `Branch`, то не можна допустити вставку у відношення `Branch` кортежу, що містить NULL в атрибуті `branchNo`. У якості ще одного приклада розглянемо складений первинний ключ відношення `Viewing`, що складається з атрибутів номера орендаря (`clientNo`) і номера об'єкта власності – (`propertyNo`). У відношення `Viewing` не допускається вставка кортежів, що містять NULL в атрибуті `clientNo` або `propertyNo`, або й у тім і в іншому атрибуті.

Якщо розглянути це правило більш уважно, то можна помітити декілька його незвичайних властивостей. По-перше, чому воно застосовується до первинних ключів, але не використовується до потенційних ключів, які також однозначно визначають кортежі? По-друге, чому воно обмежується тільки базовими відношеннями? Наприклад, використовуючи дані відношення `Viewing` (див. табл. 5.8), розглянемо наступний запит: "Виконати вибірку всіх коментарів за результатами огляду". Результатом цього запиту є унарне відношення з єдиного атрибута `comment`. За визначенням, цей атрибут повинен бути первинним ключем, але серед його значень міститься NULL (що відповідає оглядам об'єктів PG36 і PG4 орендарем CR56). Оскільки це відношення не є базовим, реляційна модель допускає присутність NULL у його первинному ключі.

3.4.3 Посилальна цілісність

Друге обмеження цілісності стосується зовнішніх ключів.

Посилальна цілісність. Якщо у відношенні існує зовнішній ключ, то значення зовнішнього ключа повинне або відповідати значенню потенційного ключа деякого кортежу в його базовому відношенні, або зовнішній ключ повинен повністю складатися зі значень NULL.

Наприклад, атрибут `branchNo` у відношенні `Staff` є зовнішнім ключем, що посилається на атрибут `branchNo` базового відношення

Branch. Система повинна запобігати будь-які спроби створити запис із інформацією про співробітника відділення B025 доти, поки у відношенні Branch не буде створений запис, що містить відомості про відділення компанії з номером B025. Однак уважається припустимим створення запису з інформацією про нового співробітника із вказівкою NULL замість номера відділення, у якому цей співробітник працює. Така ситуація може мати місце в тому випадку, коли співробітник зарахований у штат компанії, але ще не приписаний до якогось конкретного відділення.

3.4.4 Корпоративні обмеження цілісності

Корпоративні обмеження цілісності. Додаткові правила підтримки цілісності даних, обумовлені користувачами або адміністраторами бази даних.

Користувачі самі можуть вказувати додаткові обмеження, яким повинні задовольняти дані. Наприклад, якщо в одному відділенні не може працювати більше 20 співробітників, то користувач може вказати це як правило, а СУБД повинна стежити за його виконанням. У цьому випадку у відношення Staff не можна буде додати рядок з відомостями про нового співробітника деякого відділення, якщо в даному відділенні компанії вже налічується 20 співробітників. На жаль, рівень підтримки реляційної цілісності в різних системах істотно відрізняється, і корпоративні обмеження цілісностізвичайно підтримуються у серверних додатках бази даних, а не засобами самої СУБД.

3.5 Представлення

У розділі 2 при розгляді трирівневої архітектури ANSI-SPARC зовнішнє представлення описувалось як структура бази даних з погляду окремого користувача. У реляційній моделі слово "представлення" (view) має трохи інше значення. Воно характеризує не всю зовнішню модель користувача, а лише якесь *віртуальне* або *похідне відношення* (virtual relation), яке використовується користувачем, тобто відношення, що насправді не існує, але динамічно відтворюється на підставі одного або декількох *базових відношень* (відношень, що реально існують у базі даних).

Таким чином, зовнішня модель може складатися одночасно як з базових (на концептуальному рівні) відношень, так і з *представлень*, відтворених на основі цих базових відношень. У цьому розділі розглядаються саме такі віртуальні відношення, або представлення, що є типовим елементом реляційних систем.

В розділах 1, 2 описані можливості СУБД і їхні переваги й недоліки в порівнянні з файловими системами. У зв'язку з наявністю зазначених

вище функціональних можливостей СУБД є надзвичайно корисним інструментом. Але оскільки для кінцевих користувачів не має значення, наскільки проста або складна внутрішня організація системи, можна почути заперечення, що СУБД утрудняє роботу, надаючи користувачам набагато більшу кількість даних, чим їм дійсно потрібно.

Як показано на рис. 1.6, у підході, заснованому на використанні баз даних, необхідні співробітникам відділу контрактів докладні відомості про об'єкти нерухомості організовані трохи інакше, чим у варіанті з файловою системою, представленому на рис. 1.4. Тепер у базі даних містяться також відомості про тип нерухомості, число кімнат і про власника об'єкта, які не завжди потрібні співробітникам компанії. Для рішення проблеми "усунення" зайвих даних у СУБД передбачений механізм створення *представлень* (view), що дозволяє будь-якому користувачеві мати свій власний "образ" бази даних (представлення можна розглядати як деяку підмножину бази даних). Наприклад, можна організувати представлення, у якому співробітникам відділу контрактів будуть доступні тільки ті дані, які необхідні для оформлення договорів оренди.

Крім спрощення роботи за рахунок надання користувачам тільки дійсно потрібних їм даних, представлення володіють декількома іншими достоїнствами.

- Забезпечують додатковий рівень безпеки. Представлення можуть створюватися з метою виключення тих даних, які не повинні бачити деякі користувачі. Наприклад, можна створити деяке представлення, що дозволить менеджерам відділень і співробітникам розрахункового сектора бухгалтерії переглядати всі дані про персонал, включаючи відомості про їхню зарплату. У той же час для організації доступу до даних інших користувачів можна створити ще одне представлення, з якого всі відомості про зарплату будуть виключені.
- Надають механізм настроювання зовнішнього інтерфейсу бази даних. Наприклад, співробітники відділу контрактів можуть працювати з полем Monthlyrent (Щомісячна орендна плата), використовуючи для нього більш коротке й просте ім'я – rent.
- Дозволяють зберігати зовнішній інтерфейс бази даних несуперечливим і незмінним навіть при внесенні змін у її структуру – наприклад, при додаванні або видаленні полів, зміні зв'язків, розбивці файлів, їхньої реорганізації або перейменуванні. Якщо у файл додаються або з нього видаляються поля, які використовуються в деякому представленні, то всі ці зміни ніяк не відіб'ються на даному представленні. Таким чином, представлення забезпечує повну

незалежність програм від реальної структури даних, що дозволяє усунути найважливіший недолік файлових систем.

Наведені вище міркування мали досить загальний характер. У дійсності реальний обсяг функціональних можливостей залежить від конкретної СУБД. Наприклад, у СУБД для персонального комп'ютера може не підтримуватися паралельний спільний доступ, а керування режимом захисту, підтримкою цілісності даних і відновленням буде присутнім тільки в дуже обмеженому ступені. Однак сучасні потужні СУБД з багатьма користувачами пропонують всі перераховані вище функціональні можливості й багато чого іншого.

Сучасні системи являють собою надзвичайно складне програмне забезпечення, що складається з мільйонів рядків коду й багатьох томів документації. Такий результат прагнення одержати програмне забезпечення, що могло б задовольняти вимогам усе більш загального характеру. Більше того, у цей час використання СУБД припускає майже стовідсоткову надійність і готовність навіть при збоях в апаратному й програмному забезпеченні. Програмне забезпечення СУБД постійно вдосконалюється й повинне усе більше й більше розширюватися, щоб задовольняти все новим вимогам користувачів. Наприклад, у деяких додатках тепер потрібно зберігати графіки, відео, звук і т.д. Для охвата цієї частини ринку СУБД повинна розвиватися, причому згодом їй, імовірно, буде потрібно виконувати якісь нові функції, а тому функціональна частина СУБД ніколи не буде незмінною.

3.5.1 Термінологія

Ті відношення, з якими ми мали справу дотепер, називаються базовими відношеннями.

Базове відношення. Іменоване відношення, що відповідають сутності в концептуальній схемі, кортежі якого фізично зберігаються в базі даних.

Поняття представлення визначається на основі базових відношень.

Представлення. Динамічний результат одної або декількох реляційних операцій над базовими відношеннями з метою створення деякого іншого відношення. Представлення є *віртуальним відношенням*, що реально в базі даних не існує, але створюється на вимогу окремого користувача в момент надходження цієї вимоги.

З погляду користувача представлення є відношенням, що постійно існує й з яким можна працювати точно так само, як з базовим відношенням. Однак представлення не завжди зберігається в базі даних

так, як базові відношення (хоча його визначення зберігається в системному каталозі). Вміст представлення визначається як результат виконання запиту до одному або декількох базовим відношень. Будь-які операції над представленням автоматично транслюються в операції над відношеннями, на підставі яких воно було створено. Представлення мають *динамічний* характер, тобто зміни в базових відношеннях, які можуть вплинути на вміст представлення, негайно відбиваються на вмісті цього представлення. Якщо користувачі вносять у представлення деякі припустимі зміни, останні негайно заносяться в базові відношення представлення. У даному розділі описується призначення представлень і коротко розглядаються обмеження на відновлення даних, здійснювані через представлення.

3.5.2 Призначення представлень

Механізм представлень може використовуватись з кількох причин.

- Він надає потужний і гнучкий механізм захисту, що дозволяє сховати деякі частини бази даних від певних користувачів. Користувач не буде мати відомостей про існування яких-небудь атрибутів або кортежів, відсутніх у доступні йому представленнях.
- Він дозволяє організувати доступ користувачів до даних найбільш зручним для них образом, тому ті самі дані в той самий час можуть розглядатися різними користувачами зовсім різними способами.
- Він дозволяє спрощувати складні операції з базовими відношеннями. Наприклад, якщо представлення буде визначено на основі з'єднання двох відношень, то користувач зможе виконувати над ним прості унарні операції вибірки й проекції, які будуть автоматично перетворені засобами СУБД в еквівалентні операції з виконанням з'єднання базових відношень.

Представлення варто проектувати, вибираючи такий спосіб підтримки зовнішньої моделі, який був би найбільш зручний користувачеві. Нижче наведені приклади застосування подібного підходу на практиці.

- При роботі із записами відношення `Branch` користувачам, крім різних атрибутів із записів цього відношення, може знадобитися вибирати імена й інші відомості про відповідних менеджерів. Подібне представлення створюється шляхом з'єднання

відношення Branch і Staff з наступною його проекцією за значенням атрибута Manager.

- Більшість користувачів при роботі із записами відношення Staff не повинні мати доступ до атрибута salary.
- Атрибути можуть перейменовуватися, так що користувач, що звик використовувати замість номерів відділень (атрибут branchNo) їхню повну назву (Branch Number), зможе використовувати відповідний текст як заголовок стовпця.
- Кожному співробітникові може бути надане право переглядати записи з характеристиками тільки тих об'єктів нерухомості, за які він відповідає.

Всі ці приклади демонструють певний ступінь логічної незалежності від даних, що досягає за рахунок використання представлень (див. розділ 2.4.5). Однак насправді представлення дозволяють домогтися й більш важливого типу логічної незалежності від даних, пов'язаної із захистом користувачів від реорганізацій концептуальної схеми. Наприклад, якщо у відношення буде доданий новий атрибут, то користувачі не будуть навіть підозрювати про його існування, поки визначення їхніх представлень не включають цей атрибут. Якщо існуюче відношення реорганізоване або розбите на частині, то представлення, яке його використовує, може бути перевизначене так, щоб користувачі могли продовжувати працювати з даними в колишньому форматі.

3.5.3 Оновлення представлень

Всі оновлення даних у базовому відношенні повинні бути негайно відбиті у всіх представленнях, пов'язаних із цим базовим відношенням. Аналогічно, при оновленні даних у представленні внесені зміни повинні бути відбиті в його базовому відношенні. Але на типи змін, які можуть бути виконані за допомогою представлень, накладаються певні обмеження. Нижче перераховані умови, які використовуються в більшості існуючих систем для перевірки допустимості оновлення даних через деяке представлення.

- Оновлення допускаються через представлення, що визначено на основі простого запиту до єдиного базового відношення й містить первинний або потенційний ключ цього базового відношення.
- Оновлення не допускаються в будь-яких представленнях, визначених на основі декількох базових відношень.

- Оновлення не допускаються в будь-яких представленнях, що включають виконання операцій агрегування або групування.

Представлення прийнято ділити на наступні класи: *теоретично неоновлювані*, *теоретично оновлювані* й *частково оновлювані*.

Приклад неузгодженності бази даних, яка може виникнути при наявності оновлення представлень, наведено у розділі 12.2.2, де додається кортеж у представлення, створеного з'єднанням двух базових відношень.

У додатках до баз даних можна знайти приклади роботи, яка схожа з оновлення представлень, визначених на декількох базових відношеннях. Однак для узгодженності бази даних під час виконання таких операцій програмісти, що розробляють такі додатки, дуже уважно відстежують, щоб оновлення проводились лише для операцій екві-з'єднання (найчастіше – природного з'єднання), а не тета-з'єднання (див. розділ 6) базових відношень. Крім того, з'єднуватись в такому представленні повинні повні базові відношення, а не їх проекції на деяку підмножину атрибутів.

4 РЕЛЯЦІЙНА АЛГЕБРА Й РЕЛЯЦІЙНЕ ВИРАХУВАННЯ

Реляційна алгебра й реляційне вирахування являють собою формальні, а не дружні користувачеві мови. У реляційних базах даних вони використовувались як основа для розробки інших мов керування даними більш високого рівня.

4.1 Реляційна алгебра

Реляційна алгебра — це теоретична мова операцій, що дозволяють створювати на основі одного або декількох відношень інше відношення без зміни самих вихідних відношень. Таким чином, обидва операнди й результат є відношеннями, тому результати однієї операції можуть застосовуватися в іншій операції. Це дозволяє створювати вкладені вирази реляційної алгебри (за аналогією з тим, як створюються вкладені арифметичні вирази), але при будь-якій глибині вкладеності результатом є відношення. Така властивість називається *замкнутістю.* Вона підкреслює те, що застосування будь-якої кількості операцій реляційної алгебри до відношень не приводить до створення інших об'єктів, крім відношень, точно так само, як результатами арифметичних операцій із числами є тільки числа.

Реляційна алгебра є мовою послідовного використання відношень, у якій всі кортежі, можливо, навіть узяті з різних відношень, обробляються однією командою, без організації циклів. Для команд реляційної алгебри запропоновано кілька варіантів синтаксису. Нижче ми скористаємося загальноприйнятими символічними позначеннями для цих команд і представимо їх у неформальному виді.

Існує кілька варіантів вибору операцій, які включаються в реляційну алгебру. Спочатку Кодд [5] запропонував вісім операцій, але згодом до них були додані й деякі інші. П'ять основних операцій реляційної алгебри, а саме *вибірка* (selection), *проекція* (projection), *декартовий добуток* (cartesian product), *об'єднання* (union) і *різниця множин* (set difference), виконують більшість дій по добуванню даних, які можуть представляти для нас інтерес. На підставі п'яти основних операцій можна також вивести додаткові операції, такі як операції *з'єднання* (join), *перетинання* (intersection) і *розподілу* (division), які можуть бути виражені в термінах п'яти основних операцій. Результати цих операцій схематично показані на рис. 4.1.

Операції вибірки й проекції є *унарними,* оскільки вони працюють із одним відношенням. Інші операції працюють із парами відношень, і тому їх називають *бінарними* операціями. У наведених нижче визначеннях

R і **S** – це два відношення, визначені на атрибутах **A =(a_1, a_2, . . . ,a_n)** і **B= (b_1, b_2,. . ., b_m)** відповідно.

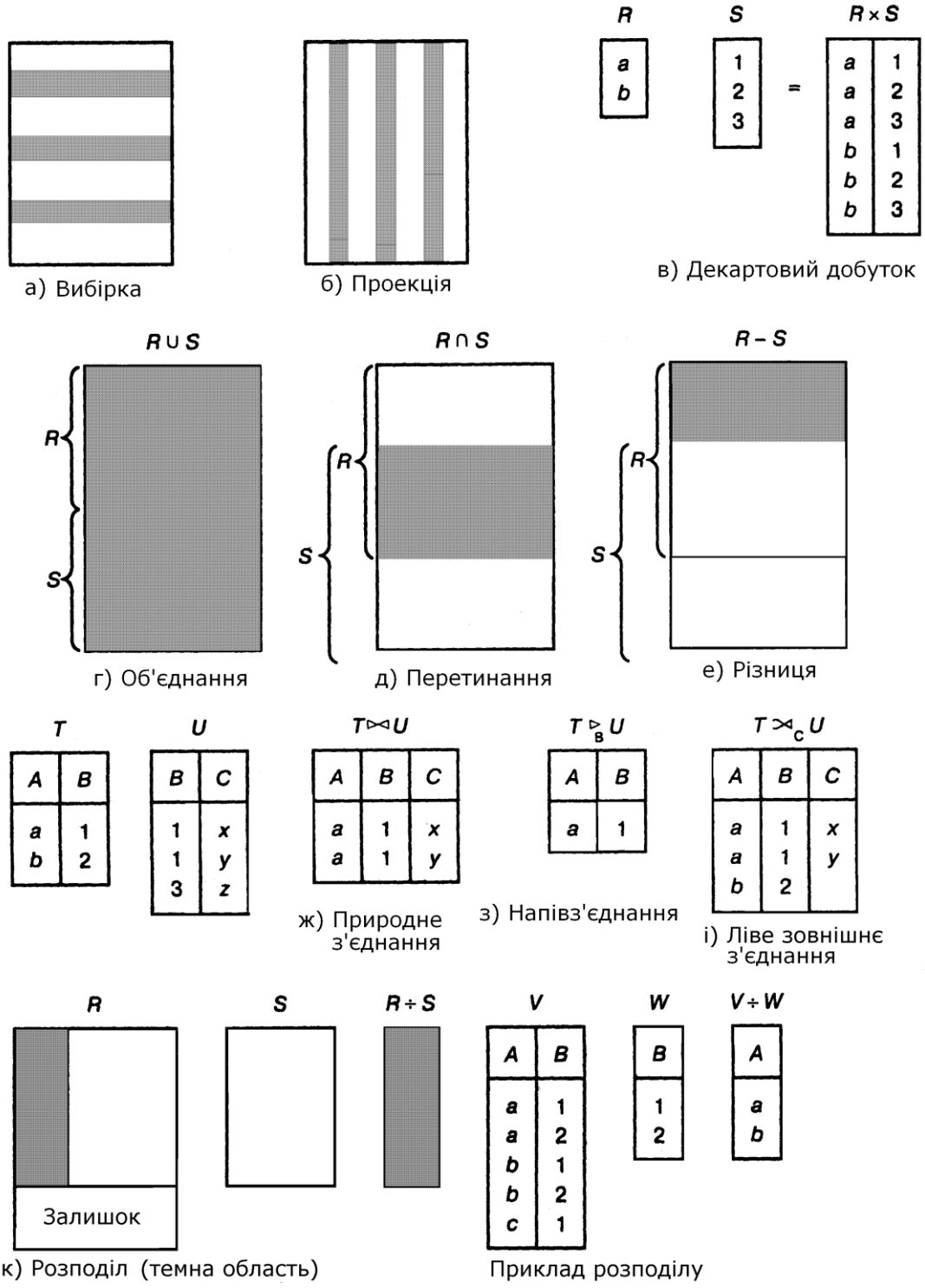


Рис. 6.1 Схематичне подання результатів операцій реляційної алгебри

П'ять операцій є основними: проекція, об'єднання, різниця, декартовий добуток і вибірка. Інші операції, що часто використовуються:

перетинання, з'єднання й розподіл – можна виразити через п'ять основних операцій.

У прикладах розглядаються відношення `Staff`, `Branch`, `PropertyForRent`. Також використовуються наступні схематичні відношення:

$P(D_1, D_2, D_3)$		
1	11	*x*
2	11	*y*
3	11	*z*
4	12	*x*

$Q(D_4, D_5)$	
x	1
x	2
y	1

$R(M, P, Q, T)$			
x	101	5	*a*
y	105	3	*a*
z	500	9	*a*
w	50	1	*b*
w	10	2	*b*
w	300	4	*b*

$S(A, B)$	
5	*a*
10	*b*
15	*c*
2	*d*
6	*a*
1	*b*

Опис кожного відношення складається з *інтенсіонала* відношення. Під описом наведене деяке заповнення кортежів відношення (*екстенсіонал* відношення).

4.1.1 Унарні операції реляційної алгебри

Вибірка (або обмеження)

$\sigma_{предикат}(R)$ Операція вибірки застосовується до одного відношення R і визначає результуюче відношення, що містить тільки ті кортежі (рядки) з відношення R, які задовольняють заданій умові *(предикату)*

Приклад 4.1 на операцію вибірки.

Відберіть тільки ті кортежі з відношення **R**, *у яких значення атрибута* **P** *перевищує 100.*

$$\sigma_{P>100}(R) = \begin{bmatrix} x & 101 & 5 & a \\ y & 105 & 3 & a \\ z & 500 & 9 & a \\ \sout{w} & \sout{50} & \sout{1} & \sout{b} \\ \sout{w} & \sout{10} & \sout{2} & \sout{b} \\ w & 300 & 4 & b \end{bmatrix} = \begin{bmatrix} x & 101 & 5 & a \\ y & 105 & 3 & a \\ z & 500 & 9 & a \\ w & 300 & 4 & b \end{bmatrix}$$

Приклад 4.2 на операцію вибірки

Складіть список всіх співробітників із зарплатою, що перевищує 10000 фунтів стерлінгів.

$\sigma_{salary>10000}(Staff)$

Тут вихідним відношенням є відношення `Staff`, а предикатом – вираз `salary>10000`. Операція вибірки визначає нове відношення, що містить тільки ті кортежі відношення `Staff`, у яких значення атрибута `salary` перевищує 10000 фунтів стерлінгів. Результат виконання цієї операції показаний у табл. 4.1. Більш складні предикати можуть бути створені за допомогою логічних операцій ∧ (AND), ∨ (OR) і ~ (NOT).

Таблиця 4.1. Результат виконання операції вибірки з відношення `Staff` кортежів з атрибутом `salary` > 10000

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-10-1945	30000	B005
SG37	Ann	Beech	Assistant	F	10-11-1960	12000	B003
SG14	David	Ford	Superviso	M	24-03-1958	18000	B003
SG5	Susan	Brand	Manager	F	3-J06-1940	24000	B003

Проекція

$\pi_{a1,...an}(R)$. Операція проекції застосовується до одного відношення R і визначає нове відношення, що містить вертикальну підмножину відношення R, яке створюється за допомогою витягу значень зазначених атрибутів і виключення з результату рядків-дублікатів.

Приклад 4.3 на операцію проекції

Знайти проекцію відношення R на атрибути M й T:

$$\pi_{M,T}(R) = \begin{bmatrix} x & a \\ y & a \\ z & a \\ w & b \\ \sout{w} & \sout{b} \\ \sout{w} & \sout{b} \end{bmatrix} = \begin{bmatrix} x & a \\ y & a \\ z & a \\ w & b \end{bmatrix}$$

Приклад 4.4 на операцію проекції

Створіть відомість зарплати всіх співробітників компанії із вказівкою тільки атрибутів `staffNo, fName, lName і salary`

$$\pi_{\text{staffNo, fName, lName, salary}}(\text{Staff})$$

У цьому прикладі операція проекції визначає нове відношення, що буде містити тільки атрибути staffNo, fName, lName і salary відношення Staff, розміщені в зазначеному порядку.

4.1.2 Операції з множинами

Операції вибірки й проекції передбачають добування інформації тільки з одного відношення. Проте на практиці часто виникає необхідність одержати дані за допомогою декількох відношень. Нижче в цьому розділі розглядаються бінарні операції реляційної алгебри, починаючи з операцій об'єднання, обчислення різниці, перетинання й декартова добутку.

Об'єднання

R U S. Об'єднання двох відношень R і S визначає нове відношення, що включає всі кортежі, що містяться тільки в R, тільки в S, одночасно в R і S, причому всі дублікати кортежів виключені. При цьому відношення R і S повинні бути сумісними за об'єднанням.

Якщо **R** і **S** включають, відповідно, ***I*** і ***J*** кортежів, то об'єднання цих відношень можна одержати, зібравши всі кортежі в одне відношення, що може містити не більше (***I*** + ***J***) кортежів. Об'єднання можливо, тільки якщо схеми двох відношень збігаються, тобто складаються з однакової кількості атрибутів, причому кожна пара відповідних атрибутів має однаковий домен. Інакше кажучи, відношення повинні бути *сумісними за об'єднанням.* Відзначимо, що у визначенні сумісності за об'єднанням не зазначено, що атрибути повинні мати однакові імена. У деяких випадках для одержання двох сумісних за об'єднанням відношеннь може бути використана операція проекції.

Приклад 4.5 на операцію об'єднання

*Створіть відношення, що поєднує відношення **S** і проекцію відношення **R** на атрибути **Q, T**:*

$$\pi_{Q,T}(R) \cup S = \begin{bmatrix} 5 & a \\ 3 & a \\ 9 & a \\ 1 & b \\ 2 & b \\ 4 & b \end{bmatrix} \cup \begin{bmatrix} \sout{5} & \sout{a} \\ 10 & b \\ 15 & c \\ 2 & d \\ 6 & a \\ \sout{1} & \sout{b} \end{bmatrix} = \begin{bmatrix} 5 & a \\ 3 & a \\ 9 & a \\ 1 & b \\ 2 & b \\ 4 & b \\ 10 & b \\ 15 & c \\ 2 & d \\ 6 & a \end{bmatrix}$$

Приклад 4.6 Операція об'єднання

Створіть список всіх міст, у яких є відділення компанії або об'єкт нерухомості.

$\pi_{city}(Branch) \cup \pi_{city}(PropertyForRent)$

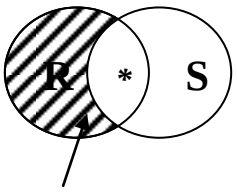
Для створення сумісних за об'єднанням відношень спочатку варто застосувати операцію проекції, щоб виділити з відношеннь Branch і PropertyForRent стовпці з атрибутами city, видаляючи в разі необхідності дублікати. Потім для комбінування отриманих проміжних відношень потрібно використати операцію об'єднання. Результат виконання всіх цих дій наведений у табл. 4.2.

Таблиця 4.2. Об'єднання, засноване на атрибуті city відношеннь Branch і PropertyForRent

city
London
Aberdeen
Glasgow
Bristol

Різниця

R – S. Різниця двох відношень R і S складається з кортежів, які є у відношенні R, але відсутні у відношенні S. Причому, відношення R і S повинні бути сумісні за об'єднанням.



Подання діаграмою Венна

R-S

На діаграмі Венна заштрихована частина являє собою різницю R – S, а зірочкою (*) відзначена область, що позначає перетинання $R \mid S$.

Приклад 4.7 операція різниці

$$\pi_{Q,T}(R) - S = \begin{bmatrix} \underline{5 \quad a} \\ 3 \quad a \\ 9 \quad a \\ \underline{1 \quad b} \\ 2 \quad b \\ 4 \quad b \end{bmatrix} - \begin{bmatrix} \underline{5 \quad a} \\ 10 \quad b \\ 15 \quad c \\ 2 \quad d \\ 6 \quad a \\ \underline{1 \quad b} \end{bmatrix} = \begin{bmatrix} 3 \quad a \\ 9 \quad a \\ 2 \quad b \\ 4 \quad b \end{bmatrix}$$

Приклад 4.8 операція різниці з відношеннями Branch і PropertyForRent

Створіть список всіх міст, у яких є відділення компанії, але немає об'єктів нерухомості, які здаються в оренду.

$\pi_{city}(Branch) - \pi_{city}(PropertyForRent)$

У цьому випадку аналогічно прикладу 4.5 потрібно створити сумісні за об'єднанням відношення Branch і PropertyForRent, виконавши їхню проекцію по атрибуту city. Потім для комбінування отриманих нових відношень потрібно використати операцію різниці.

Перетинання

R ∩ S. Перетинання двох відношень R і S складається з кортежів, які є як у відношенні R, так і у відношенні S. Причому, відношення R і S повинні бути сумісні за об'єднанням.

Перетинання **R ∩ S = R – (R – S)**, що відповідає області, відзначеною зірочкою на діаграмі Венна для операції різниці.

Приклад 4.9 операція перетинання з відношеннями Branch і PropertyForRent

Створіть список всіх міст, у яких є відділення компанії, а також щонайменше один об'єкт нерухомості, що здається в оренду.

$\pi_{city}(Branch) \cap \pi_{city}(PropertyForRent)$

Приклад 4.10 операція перетинання

Знайти $\pi_{Q,T}(R) \cap S$:

$$\pi_{Q,T}(R) \cap S = \begin{bmatrix} \underline{5 & a} \\ 3 & a \\ 9 & a \\ \underline{1 & b} \\ 2 & b \\ 4 & b \end{bmatrix} - \begin{bmatrix} \underline{5 & a} \\ 10 & b \\ 15 & c \\ 2 & d \\ 6 & a \\ \underline{1 & b} \end{bmatrix} = \begin{bmatrix} 5 & a \\ 1 & b \end{bmatrix}$$

Декартовий добуток

R×S. Операція декартового добутку визначає нове відношення, що є результатом конкатенації (зчеплення) кожного кортежу з відношення R з кожним кортежем з відношення S.

Операція декартового добутку застосовується для множення двох відношень. Множенням двох відношень називається створення іншого відношення, що складається із всіх можливих пар кортежів обох відношень. Отже, якщо одне відношення має ***I*** кортежів і ***N*** атрибутів, а інше — ***J*** кортежів і ***M*** атрибутів, то їх декартовий добуток буде містити (***I*** х ***J***) кортежів і ***(N+M)*** атрибутів. Вихідні відношення можуть містити атрибути з однаковими іменами. У такому випадку імена атрибутів будуть містити назви відношень у вигляді префіксів для забезпечення

унікальності імен атрибутів у відношенні, отриманому як результат виконання операції декартового добутку.

Таким чином,

Ступінь (R × S) = Ступінь (R) + Ступінь(**S**),

Потужність (R × S) = Потужність (R) × Потужність (S) .

Звідси випливає, що результуюче відношення може мати дуже більші розміри. (На практиці звичайно використовується обмежений варіант цієї операції, який називається *з'єднанням.)*

Приклад 4.11 на операцію декартового добутку.

Знайти $\pi_{M,T}(R) \times (\pi_{Q,T}(R) \cap S)$

$$\pi_{M,T}(R) = \begin{bmatrix} x & a \\ y & a \\ z & a \\ w & b \end{bmatrix} \quad \text{(див. приклад 4.3)}$$

$$\pi_{Q,T}(R) \cap S = \begin{bmatrix} 5 & a \\ 1 & b \end{bmatrix} \quad \text{(див. приклад 4.10)}$$

Тому $\pi_{M,T}(R) \times (\pi_{Q,T}(R) \cap S) =$

$$= \begin{bmatrix} x & a \\ y & a \\ z & a \\ w & b \end{bmatrix} \times \begin{bmatrix} 5 & a \\ 1 & b \end{bmatrix} = \begin{bmatrix} x & a & 5 & a \\ x & a & 1 & b \\ y & a & 5 & a \\ y & a & 1 & b \\ z & a & 5 & a \\ z & a & 1 & b \\ w & b & 5 & a \\ w & b & 1 & b \end{bmatrix}$$

Ступінь результуючого відношення дорівнює 4 (тобто 2+2), а потужність (кардинальність) 8 (тобто 2x4).

Приклад 4.12 Операція декартового добутку

Створіть список всіх орендарів, які оглядали об'єкти нерухомості, із вказівкою зроблених ними коментарів.

Імена орендарів зберігаються у відношенні Client, а відомості про виконані ними огляди – у відношенні Viewing. Щоб одержати список орендарів і коментарі про оглянуту ними нерухомість, необхідно об'єднати ці два відношення:

$\pi_{clientNo,\ fName,\ lName}(Client) \times \pi_{clientNo,\ propertyNo,\ comment}(Viewing)$

Результати виконання цієї операції показані в табл. 4.3 (частина записів). У такому вигляді це відношення містить більше інформації, ніж необхідно. Наприклад, перший кортеж цього відношення містить різні значення атрибута clientNo.

Таблиця 4.3. Декартовий добуток скороченого варіанта відношень Client і Viewing (використані тільки деякі атрибути)

Client. clientNo	**fName**	**lName**	**Viewing. clientNo**	**propertyNo**	**comment**
CR76	John	Kay	CR56	PA14	Too small (Занадто мала)
CR76	John	Kay	CR76	PG4	Too remote (Занадто далеко)
CR76	John	Kay	CR56	PG4	
CR76	John	Kay	CR62	PA14	No Dining room (Немає окремої їдальні)
CR76	John	Kay	CR56	PG36	
…					

Для одержання шуканого списку необхідно для цього відношення зробити операцію вибірки з добуванням тих кортежів, для яких виконується рівність `Client.clientNo=Viewing.clientNo`. Повністю ця операція виглядає так, як показано нижче.

$\sigma_{Client.clientNo\ =Viewing.clientNo}((\pi_{clientNo,\ fName,\ lName}(Client))\ x$

$(\pi_{clientNo,\ propertyNo,\ comment}(Viewing)))$

Результат виконання цієї операції показаний у табл. 4.4. Як ми незабаром побачимо, комбінація декартового добутку й вибірки може бути зведена до однієї операції *з'єднання.*

Таблиця 4.4 Обмежений декартовий добуток скороченого варіанта відношень Client і Viewing (використані тільки деякі атрибути)

Client. clientNo	**fName**	**lName**	**Viewing. clientNo**	**propertyNo**	**comment**
CR76	John	Kay	CR76	PG4	Too remote (Занадто далеко)
CR56	Aline	Stewart	CR56	PA14	Too small (Занадто мала)
CR56	Aline	Stewart	CR56	PG4	
CR56	Aline	Stewart	CR56	PG36	
CR62	Mary	Tregear	CR62	PA14	No dining room (Немає окремої їдальні)

4.1.3 Декомпозиція складних операцій

Операції реляційної алгебри можуть в остаточному підсумку стати надзвичайно складними. Для спрощення такі операції можна розбити на ряд менших операцій реляційної алгебри й привласнити імена результатам проміжних виразів. Для присвоювання імен результатам операції реляційної алгебри використовується операція присвоювання, позначена стрілкою вліво (←). Дія, яка виконується при цьому, аналогічно операції присвоювання в мові програмування; у цьому випадку результат виразу праворуч від знака операції "←" привласнюється виразу, що перебуває ліворуч від знака операції. Зокрема, операції, що виконувались у попередньому прикладі, можна представити в такий спосіб:

$$\text{TempViewing(clientNo, propertyNo, comment)} \leftarrow \pi_{\text{clientNo, propertyNo, comment}}(\text{Viewing})$$

$$\text{TempClient(clientNo,fName,lName)} \leftarrow \pi_{\text{clientNo, fName, lName}}(\text{Client})$$

$$\text{Comment(clientNo, fName, IName, vclientNo, propertyNo, comment)} \leftarrow \text{TempClient x TempViewing}$$

$$\text{Result} \leftarrow \sigma_{\text{clientNo = vclientNo}}(\text{Comment})$$

Ще один варіант складається у використанні операції перейменування **ρ** (позначається грецькою буквою "ро"), що дозволяє привласнити ім'я результатам операції реляційної алгебри. Операція перейменування дозволяє також задати довільне ім'я для кожного з атрибутів нового відношення.

$\rho_S(E)$ або **$\rho_{S(a_1,\ldots,a_n)}(E)$** Операція перейменування дозволяє привласнити нове ім'я S виразу E, а також додатково перейменувати атрибути як $a_1, \ldots, a_n$

4.1.4 Операції з'єднання

Як правило, користувачів цікавить лише деяка частина всіх комбінацій кортежів декартова добутку, що задовольняє заданій умові. Тому замість декартова добутку звичайно використовується одна з найважливіших операцій реляційної алгебри – операція з'єднання. У результаті її виконання на базі двох вихідних відношень створюється деяке нове відношення.

Операція з'єднання є похідною від операції декартова добутку, тому що вона еквівалентна операції вибірки з декартова добутку двох операндів-відношень тих кортежів, які задовольняють умові, зазначеному в предикаті з'єднання як формула вибірки. З погляду ефективної реалізації в реляційних СУБД ця операція є однією із самих складних і часто

виявляється однією з основних причин, що викликають проблеми із продуктивністю, властиві всім реляційним системам.

Нижче перераховані різні типи операцій з'єднання, які трохи відрізняються друг від друга й можуть бути в тім або іншому ступені корисні.

- Тета-з'єднання (theta join).
- З'єднання за еквівалентністю або екві-з'єднання (equijoin), що є одним з видів тета-з'єднання.
- Природне з'єднання (natural join).
- Зовнішнє з'єднання (outer join).
- Напівз'єднання (semijoin).

Тета-з'єднання

$R \rhd\lhd_F S$. Операція тета-з'єднання визначає відношення, що містить кортежі з декартова добутку відношень R і S, що задовольняють предикату F. Предикат F має вигляд: $R.a_i\ \theta\ S.b_j$, де замість θ може бути зазначена одна з операцій порівняння (<, <=, >, >=, = або ~=).

Позначення тета-з'єднання можна переписати на основі базових операцій вибірки й декартова добутку, як показано нижче.

$$R \rhd\lhd_F S = \sigma_F(R \times S)$$

Так само як і у випадку з декартовим добутком, ступенем тета-з'єднання називається сума ступенів операндів-відношень R і S. Якщо предикат F містить тільки операцію порівняння за рівністю (=), то з'єднання називається з'єднанням за еквівалентністю (equi-join).

Приклад 4.13 Тета-з'єднання $R \rhd\lhd_{R.Q>S.A} S$

При виконанні з'єднання необхідно для кожного кортежу з ***R*** взяти значення атрибута ***Q*** і зрівняти його зі значенням атрибута ***A*** з кожного кортежу ***S***.

R (*M*,	*P*,	*Q*,	*T*)	***S*** (*A*,	*B*)
x	101	5	*a*	5	*a*
y	105	3	*a*	10	*b*
z	500	9	*a*	15	*c*
w	50	1	*b*	2	*d*
w	10	2	*b*	6	*a*
w	300	4	*b*	1	*b*

Як видно з екстенсіоналів відношень ***R*** і ***S***, перший кортеж відношення ***R*** зв'яжеться з 4-м і 6-м кортежем відношення ***S***; 2-й кортеж

відношення ***R*** зв'яжеться також з 4-м і 6-м кортежем відношення ***S***; 3-й кортеж відношення ***R*** зв'яжеться також з 1-м, 4-м, 5-м і 6-м кортежем відношення ***S;*** і т.д. Слід зазначити, що кортеж <*w* 50 1 *b*> відношення ***R*** не ввійде в результуюче відношення. У результаті одержимо:

x	101	5	*a*	2	*d*
x	101	5	*a*	1	*b*
y	105	3	*a*	2	*d*
y	105	3	*a*	1	*b*
z	500	9	*a*	5	*a*
z	500	9	*a*	2	*d*
z	500	9	*a*	6	*a*
z	500	9	*a*	1	*b*
w	10	2	*b*	1	*b*
w	300	4	*b*	2	*d*
w	300	4	*b*	1	*b*

Ще раз звернемося до запиту, що розглядався в прикладі 4.12 (див. приклад 4.14).

Приклад 4.14. Операція з'єднання за еквівалентністю (екві-з'єднання).

Створіть список всіх орендарів, які оглядали об'єкт нерухомості, із вказівкою їхніх імен і зроблених ними коментарів.

У прикладі 4.12 для одержання цього списку використовувались операції декартового добутку й вибірки. Однак той же самий результат можна одержати за допомогою операції з'єднання за еквівалентністю.

$(\pi_{clientNo,fName,Lname}(Client)) \bowtie (\pi_{clientNo,propertyNo,commtnt}(Viewing))$

або

$TempViewing(clientNo, propertyNo, comment) \leftarrow \pi_{clientNo,\ propertyNo,\ comment}(Viewing)$

$TempClient(clientNo,fName,lName) \leftarrow \pi_{clientNo,\ fName,\ lName}(Client)$

$Result \leftarrow TempClient \bowtie \pi_{TempClient.clientNo=TempViewing.clientNo}\ TempViewing$

Природне з'єднання

R ⋈ S. Природним з'єднанням називається з'єднання за еквівалентністю двох відношень R і S, виконане по всіх загальних атрибутах X, з результатів якого виключається по одному екземпляру загального атрибута.

Ступенем природного з'єднання називається сума ступенів операндів-відношень R і S за винятком кількості атрибутів X.

Приклад 4.15. Природне з'єднання $P \rhd\lhd \rho_{Q(D_3,D_5)}(Q)$

У цьому випадку кожний кортеж відношення P буде з'єднуватися з тим кортежем відношення Q, у якому значення атрибута D_4 дорівнює значенню атрибута D_3 у відношенні P. При цьому в підсумковому відношенні не буде атрибута D_4. Для виконання природного з'єднання необхідна наявність *загальних* атрибутів. Тому до виконання операції природного з'єднання спочатку виконується операція перейменування ρ, що атрибуту D_4 відношення Q привласнює ім'я D_3:

P(D_1,	D_2,	D_3)	***Q***(D_4,	D_5)
1	11	*x*	*x*	1
2	11	*y*	*x*	2
3	11	*z*	*y*	1
4	12	*x*		

Таким чином, 1-й і 4-й кортежі відношення **P** з'єднається кожний з 1-м і 2-м кортежами відношення **Q**; 2-й кортеж відношення **P** — з 3-м кортежем відношення **Q**; 3-й кортеж відношення **P** не ввійде в результуюче відношення:

$$\begin{bmatrix} 1 & 11 & x & x & 1 \\ 1 & 11 & x & x & 2 \\ 2 & 11 & y & y & 1 \\ 4 & 12 & x & x & 1 \\ 4 & 12 & x & x & 2 \end{bmatrix} = \begin{bmatrix} 1 & 11 & x & 1 \\ 1 & 11 & x & 2 \\ 2 & 11 & y & 1 \\ 4 & 12 & x & 1 \\ 4 & 12 & x & 2 \end{bmatrix}$$

Приклад 4.16. Операція природного з'єднання

Створіть список всіх орендарів, які оглядали об'єкти нерухомості, із вказівкою їхніх імен і зроблених ними коментарів.

У прикладі 4.14 для складання цього списку використовувалось з'єднання за еквівалентністю, але в ньому були присутні два атрибути `clientNo`. Для видалення одного із цих атрибутів можна скористатися операцією природного з'єднання.

$(\pi_{clientNo,fName,Lname}(Client)) \rhd\lhd (\pi_{clientNo,propertyNo,commtnt}(Viewing))$

або

$TempViewing(clientNo, propertyNo, comment) \leftarrow \Pi_{clientNo,\ propertyNo,\ comment}(Viewing)$

$TempClient(clientNo,fName,lName) \leftarrow \Pi_{clientNo,\ fName,\ lName}(Client)$

$Result \leftarrow TempClient \rhd\lhd TempViewing$

Результат цих операцій показаний у табл. 4.5.

Таблиця 4.5. Природне з'єднання скороченого варіанта відношень `Client` і `Viewing` (використані тільки деякі атрибути)

clientN	fName	lName	propertyNo	comment
CR76	John	Kay	PG4	Too remote (Занадто далеко)
CR56	Aline	Stewart	PA14	Too small (Занадто мала)
CR56	Aline	Stewart	PG4	
CR56	Aline	Stewart	PG36	
CR62	Mary	Tregear	PA14	No dining room (Немає окремої їдальні)

Композиція

Композицією називається з'єднання за еквівалентністю двох відношень R і S, виконане по всіх загальних атрибутах x, з результатів якого виключається по два екземпляра загального атрибута.

Зовнішнє з'єднання

При з'єднанні двох відношень часто виникає така ситуація, що для кортежу одного відношення не існує відповідного кортежу в іншім відношенні. Інакше кажучи, у стовпцях з'єднання виявляються незбіжні значення. Але іноді може знадобитися, щоб рядок з одного відношення був представлений у результатах з'єднання, навіть якщо в іншому відношенні немає співпадаючого значення. Ця мета може бути досягнута за допомогою зовнішнього з'єднання.

$R \supset\!\!\triangleleft S$. Лівим зовнішнім з'єднанням називається з'єднання, при якому в результуюче відношення включаються також кортежі відношення R, що не мають співпадаючих значень у загальних стовпцях (атрибутах) відношення S.

Для позначення відсутніх значень у другому відношенні використовується значення NULL. Зовнішнє з'єднання застосовується в реляційних СУБД все частіше, до того ж у цей час для його виконання передбачена операція, що включена в новий стандарт SQL (див. розділ 10). Перевагою зовнішнього з'єднання є те, що після його виконання зберігається вихідна інформація, тобто зовнішнє з'єднання зберігає кортежі, які були б виключені при використанні інших типів з'єднання.

Приклад 4.17. Зовнішнє з'єднання

Створіть звіт про хід проведення оглядів об'єктів нерухомості.

У цьому випадку необхідно створити відношення, що складається як з переліку оглянутих клієнтами об'єктів нерухомості (із приведенням їхніх коментарів із цього приводу), так і переліку об'єктів нерухомості, які ще не оглядалися. Це можна зробити за допомогою наступного зовнішнього з'єднання.

$(\pi_{clientNo,\ street,\ city}(PropertyForRent)) \supset\!\bowtie Viewing$

Результат цієї операції показаний у табл. 4.6.

Таблиця 4.6. Ліве зовнішнє з'єднання відношень `PropertyForRent` і `Viewing`

propertyNo	street	city	clientNo	viewDate	comment
PA14	16 Holhead	Aberdeen	CR56	24-05-01	Too small (Занадто мала)
PA14	16 Holhead	Aberdeen	CR62	14-05-01	No dining room (Немає окремої їдальні)
PL94	6 Argyll St	London	NULL	NULL	NULL
PG4	6 Lawrence St	Glasgow	CR76	20-04-01	Too remote (Занадто далеко)
PG4	6 Lawrence St	Glasgow	CR56	26-05-01	
PG36	2 Manor Rd	Glasgow	CR56	28-04-01	
PG21	18 Dale Rd	Glasgow	NULL	NULL	NULL
PG16	5 Novar Dr	Glasgow	NULL	NULL	NULL

Строго кажучи, у прикладі 4.17 показане ліве зовнішнє з'єднання, оскільки в результуючому відношенні втримуються всі кортежі лівого відношення. Існує також праве зовнішнє з'єднання, яке називається так тому, що в результуючому відношенні втримуються всі кортежі правого відношення. Крім того, існує й повне зовнішнє з'єднання, у результуюче відношення якого містяться всі кортежі з обох відношень і в якому для позначення незбіжних значень кортежів використовуються значення NULL.

Напівз'єднання

$R \triangleright_F S$. Операція напівз'єднання визначає відношення, що містить ті кортежі відношення R, які входять у з'єднання відношень R і S.

Перевага напівз'єднання полягає в тім, що воно дозволяє скоротити кількість кортежів, які потрібно обробити для одержання з'єднання. Це особливо корисно при обчисленні з'єднань у розподілених системах. Операцію напівз'єднання можна сформулювати й за допомогою операцій проекції й з'єднання:

$$R \triangleright_F S = \Pi_A (R \bowtie_F S)$$

Тут **A** – це набір всіх атрибутів у відношенні **R**. У дійсності це – тета-напівз'єднання, причому слід зазначити, що існують також напівз'єднання за еквівалентністю й природні напівз'єднання.

Приклад 4.18. Операція напівз'єднання

Створіть звіт, що містить повну інформацію про всіх співробітників, що працюють у відділенні компанії, розташованому в місті 'Glasgow'.

Якщо нас цікавлять тільки атрибути відношення `Staff`, то ми можемо використати наступну операцію напівз'єднання, що приводить до створення відношення, наведеного в табл. 4.7.

$$Staff \triangleright_{staff.branchNo=branch.branchNo\ and\ branch.city='Glasgow'} Branch$$

Таблиця 4.7. Результат напівз'єднання відношень Staff і Branch

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SG37	Ann	Beech	Assistant	F	10-11-60	12000	B003
SG14	David	Ford	Supervisor	M	24-03-58	18000	B003
SG5	Susan	Brand	Manager	F	3-06-40	24000	B003

4.1.5 Розподіл

Операція розподілу може застосовуватися у випадку запитів особливого типу, які досить часто зустрічаються в додатках баз даних. Припустимо, що відношення R визначене на множині атрибутів A, а відношення S – на множині атрибутів B, причому $B \subseteq A$ (тобто B є підмножиною A). Нехай C = A – B, тобто C є множиною атрибутів відношення R, які не є атрибутами відношення S. Тоді визначення операції розподілу буде виглядати в такий спосіб.

R ÷ S. Результатом операції розподілу є набір кортежів відношення R, визначених на множині атрибутів C, які відповідають комбінації всіх кортежів відношення S.

Цю операцію можна представити й за допомогою інших основних операцій:

$$T_1 \leftarrow \pi_C(R)$$

$$T_2 \leftarrow \pi_C((S \times T_1) – R)$$

$$T \leftarrow T_1 – T_2$$

4.1.6 Короткий перелік операцій реляційної алгебри

Операція	Позначення	Область застосування
Проекція	$\pi_{a_1,...a_n}(R)$	Визначає нове відношення, що містить вертикальну підмножину відношення R, що створюється за допомогою витягу значень зазначених атрибутів і виключення з результату рядків-дублікатів
Вибірка	$\sigma_{предикат}(R)$	Визначає результуюче відношення, що містить тільки ті кортежі (рядки) з відношення R, які задовольняють заданій умові (предикату)
Об'єднання	$R \cup S$	Визначає нове відношення, що включає всі кортежі, що містяться тільки в R, тільки в S, одночасно в R і S, причому всі дублікати кортежів виключені. При цьому відношення R і S повинні бути сумісними по об'єднанню
Різниця	$R - S$	Різниця двох відношень R і S складається з кортежів, які є у відношенні R, але відсутні у відношенні S. Причому відношення R і S повинні бути сумісними за об'єднанням
Перетинання	$R \cap S$	Визначає відношення, що містить кортежі, що є присутніми як у відношенні R, так і у відношенні S. Відношення R і S повинні бути сумісними за об'єднанням
Декартовий добуток	$R \times S$	Визначає нове відношення, що є результатом конкатенації (тобто зчеплення) кожного кортежу з відношення R з кожним кортежем з відношення S
Тета-з'єднання	$R \rhd\lhd_F S$	Визначає відношення, що містить кортежі з декартова добутку відношень R і S, що задовольняють предикату F
З'єднання за еквівалентністю	$R \rhd\lhd_F S$	Визначає відношення, що містить кортежі з декартова добутку відношень R і S, що задовольняють предикату F (предикат повинен передбачати тільки порівняння на рівність)
Природне з'єднання	$R \rhd\lhd S$	Природним з'єднанням називається з'єднання за еквівалентністю двох відношень R і S, виконане по всіх загальних атрибутах x, з результатів якого виключається по одному екземплярі кожного загального атрибута
(Ліве) зовнішнє з'єднання	$R \supset\lhd S$	З'єднання, при якому кортежі відношення R, що не мають співпадаючих значень у загальних стовпцях відношення S, також включаються в результуюче відношення
Напівз'єднання	$R \rhd_F S$	Визначає відношення, що містить кортежі відношення R, які входять у з'єднання відношень R і S
Розподіл	$R \div S$	Визначає відношення, що складається з множини кортежів відношень R, які визначені на атрибуті C, що відповідає комбінації всіх кортежів відношення S, де C — множина атрибутів, наявних у відношенні R, але відсутніх у відношенні S

4.2. Реляційне вирахування

У виразах реляційної алгебри завжди явно задається якийсь порядок, а також мається на увазі якась стратегія обчислення запиту. У реляційному вирахуванні не існує ніякого опису процедури обчислення запиту, оскільки в запиті реляційного вирахування вказується, *що*, а не *як* варто витягти.

Реляційне вирахування не має нічого спільного з диференціальним або інтегральним вирахуванням, а його назва відбулася від тієї частини символьної логіки, що називається вирахуванням предикатів. У контексті баз даних воно існує у двох формах: у формі запропонованого Коддом [5] *реляційного вирахування кортежів* і у формі запропонованого Лакруа й Пиро *реляційного вирахування доменов*.

Ми розглянемо реляційне вирахування кортежів.

У логіку першого порядку (або теорії вирахування предикатів) під *предикатом* мається на увазі булєва функція з параметрами. Після підстановки значень замість параметрів функція стає виразом, що називається *судженням*, що може бути істинним або помилковим. Наприклад, речення "Джон Уайт є співробітником даної організації" і "Джон Уайт має більш високу зарплату, чим Енн Бич" є судженнями, оскільки можна визначити їхню істинність або помилковість. У першому випадку функція "є співробітником даної організації" має один параметр ("Джон Уайт"), а в другому випадку функція "має більш високу зарплату, чим" має два параметри ("Джон Уайт" і "Енн Бич").

Якщо предикат містить змінну, наприклад у вигляді "x є співробітником цієї організації", то в цієї змінної повинна бути відповідна *область визначення*. При підстановці замість змінної x одних значень із її області визначення дане судження може виявитися істинним, а при підстановці інших – помилковим. Наприклад, якщо областю визначення є всі люди й ми підставимо замість змінної x значення "Джон Уайт", те судження "Джон Уайт є співробітником даної організації" буде істинним. Якщо ж замість змінної x підставити ім'я іншої людини, що не є співробітником даної організації, то судження стане помилковим.

Якщо **P** – предикат, то множина всіх значень змінної **x**, при яких судження **P** стає істинним, можна символічно записати в такий спосіб:

$$\{ x \mid P(x) \}$$

Предикати можуть з'єднуватися за допомогою логічних операцій $\wedge$ (AND), $\vee$ (OR) и $\sim$ (NOT) з утворенням складених предикатів.

4.2.1. Реляційне вирахування кортежів

У реляційному вирахуванні кортежів завдання полягає в знаходженні таких кортежів, для яких предикат є істинним. Це вирахування засноване

на *змінних кортежу* або *кортежних змінних*. Змінними кортежу є такі змінні, областю визначення яких служить зазначене відношення. Такими є змінні, для яких припустимими значеннями можуть бути тільки кортежі даного відношення. (Поняття "область визначення" у цьому випадку ставиться не до діапазону значень, що використовується, а до домену, у якому визначені ці значення.)

Наприклад, для вказівки відношення `Staff` в якості області визначення змінної кортежу S використовується наступна форма запису:

Staff(S)

Крім того, запит "знайти множину всіх кортежів S, для яких F(S) є істинним " можна записати в такий спосіб:

{ S | F(S) }

Тут предикат F називається формулою (у математичній логіці, правильно побудованою формулою — Well-Formed Formula, або скорочено WFF). Наприклад, запит "вибрати атрибути `staffNo`, `fName`, `lName`, `position`, `sex`, `DOB`, `salary` і `branchNo` для всіх співробітників, які одержують зарплату більше 10000 фунтів стерлінгів" можна записати в такий спосіб:

{ S | Staff(S) ∧ S.salary > 10000 }

Тут вираз S.salary означає значення атрибута salary для кортежу S. Для вибірки одного певного атрибута (наприклад, salary), можна сформулювати цей запит інакше:

{ S.salary | Staff(S) ∧ S.salary > 10000}

Квантори існування й спільності

Для вказівки кількості екземплярів, до яких повинен бути застосований предикат, у формулах можуть використовуватися два типи кванторів. Квантор існування (∃), або так званий символ "існує", використовується у формулі, що повинна бути істинною хоча б для одного екземпляра, наприклад:

Staff(S) ∧ (∃B)(Branch(B) ∧ (B.branchNo=S.branchNo) ∧ (B.city='London')

Цей вираз означає, що у відношенні Branch існує кортеж, що має таке ж значення атрибута branchNo, що й значення атрибута branchNo у поточному кортежі S з відношення Staff, а атрибут city з кортежу B має значення 'London'. Квантор спільності (∀), або так званий символ "для всіх", використовується у виразах, які відносяться до всіх екземплярів, наприклад:

$$(\forall B)\ (B.city \neq 'Paris')$$

Цей вираз означає, що в жодному кортежі відношення Branch значення атрибута city не дорівнює 'Paris'. Відносно логічних операцій можуть застосовуватися наступні правила еквівалентності:

$$(\exists X)(F(X)) \equiv \sim(\forall X)(\sim(F(X)))$$

$$(\forall X)(F(X)) \equiv \sim(\exists X)(\sim(F(X)))$$

$$(\exists X)(F_1(X) \equiv F_2(X)) \equiv \sim(\forall X)(-(F\hat{}X)) \vee \sim(F_2(X)))$$

$$(\forall X)(F_1(X) \equiv F_2(X)) \equiv \sim(\exists X)(-(F\hat{}X)) \vee \sim(F_2(X)))$$

Тому наведену вище формулу можна представити в такий спосіб:

$$\sim(\exists B)(B.city = 'Paris')$$

У такому вигляді вона означає, що в Парижі немає відділень компанії.

Змінні кортежу називаються *вільними змінними*, якщо вони не кваліфікуються кванторами $\forall$ або $\exists$; у противному випадку вони називаються *зв'язаними змінними*. У виразі, складеному за правилами реляційного вирахування, вільні змінні можуть перебувати тільки ліворуч від знака вертикальної риси (|). Наприклад, у наступному запиті єдиною вільною змінною є S і в процесі обчислення цього виразу послідовно відбувається зв'язування змінної S с кожним кортежем відношення Staff.

$$\{S.fName, S.lName \mid Staff(S) \wedge (\exists B)\ (Branch(B) \wedge (B.branchNo = S.branchNo) \wedge (B.\ city\ = 'London')\}$$

Вирази й формули

Аналогічно тому, як не всі можливі послідовності букв алфавіту утворять правильно побудовані слова, так і в реляційному вирахуванні не кожна послідовність формул є припустимою. Припустимими формулами можуть бути тільки недвозначні й небезглузді послідовності. Вираз в реляційному вирахуванні кортежів має наступну загальну форму:

$$\{ S_1.a_1, S_2.a_2, ..., S_n.a_n \mid F(S_1, S_2, ..., S_m) \};\ m \geq n$$

Тут $S_1, S_2, ..., S_n, ..., S_m$ — змінні кортежу,

a_i — атрибути відношення, у якому визначене значення змінної S_i,

F — формула.

Формула (такою вважається тільки правильно побудована формула) складається з одного або декількох елементарних виразів, які можуть мати одну з наступних форм.

- $R(S_i)$, де S_i — змінна кортежу, а R — відношення.

- $S_i.a_1\ \theta\ S_j.a_2$, де S_i і S_j — змінні кортежі, a_1 — атрибут відношення, у якому визначене значення змінної S_i, a_2 — атрибут відношення, у якому визначене значення змінної S_j, і θ — одна з операцій порівняння ($<$, $\leq$, $>$, $\geq$, $=$, $\neq$); атрибути a_1 і a_2 повинні мати області визначення, для порівняння елементів яких застосування знака операції θ є припустимим.
- $S_i.a_1\ \theta\ c$, де S_i— змінна кортежу, a_1 — атрибут відношення, у якому визначене значення змінної S_i, c — константа з області визначення атрибута a_1 і θ – одна з операцій порівняння.

Формули рекурсивно будуються з елементарних виразів на основі наступних правил.

- Будь-який елементарний вираз розглядається як формула.
- Якщо вирази F_1 і F_2 є формулами, то вирази, отримані в результаті їх кон'юнкції $(F_1 \wedge F_2)$, диз'юнкції $(F_1 \vee F_2)$ і заперечення $(\sim F_1)$, також є формулами.
- Якщо вираз F є формулою з вільної змінної X, то вирази $(\exists X)(F)$ і $(\forall X)(F)$ також є формулами.

Приклад 6.18. Реляційне вирахування кортежів

А. Створіть список всіх менеджерів, зарплата яких перевищує 25000 фунтів стерлінгів.

$$\{S.fName, S.lName \mid Staff(S) \wedge (S.position = 'Manager') \wedge (S.salary > 25000)\}$$

Б. Створіть список всіх співробітників, які відповідають за роботу з об'єктами нерухомості в Глазго.

$$\{ S \mid Staff(S) \wedge (\exists P)\ (PropertyForRent(P) \wedge (P.staffNo = S.staffNo) \wedge (P.city = 'Glasgow')\}$$

5 МОВИ БАЗ ДАНИХ

Внутрішня мова СУБД для роботи з даними складається із двох частин: *мови визначення даних* (Data Definition Language,— DDL) і *мови маніпулювання даними* (Data Manipulation Language — DML). Мова DDL використовується для визначення схеми бази даних, а мова DML — для читання й відновлення даних, що зберігаються у базі.

Ці мови називаються *підмовами даних*, оскільки в них відсутні конструкції для виконання всіх обчислювальних операцій, які звичайно використовуються у мовах програмування високого рівня, такі як умовні оператори або оператори циклу. У багатьох СУБД передбачена можливість *впровадження* операторів підмови даних у програми, які написані на таких мовах програмування високого рівня, як COBOL, Fortran, Pascal, Ada, C, C++, Java або Visual Basic. У цьому випадку мову високого рівня прийнято називати *базовою мовою* (host language).

Перед компіляцією файлу програми базовою мовою, що містять впроваджені оператори підмови даних, такі оператори видаляються й заміняються викликами функцій. Потім цей попередньо оброблений файл звичайним образом компілюється із включенням результатів в об'єктний модуль, що компонується з бібліотекою, що містить функції СУБД, які викликаються в програмі. Після цього отриманий програмний текст готовий до виконання. Крім механізму впровадження, для більшості підмов даних надаються також засоби інтерактивного виконання операторів, що вводяться користувачем безпосередньо з термінала.

Мова DDL. Описова мова, що дозволяє АБД або користувачеві описати й іменувати сутності й атрибути, необхідні для роботи деякого додатка, а також зв'язки, наявні між різними сутностями, крім того, указати обмеження цілісності й захисту.

Схема бази даних складається з набору визначень, виражених спеціальною мовою визначення даних – DDL. Мова DDL використовується як для визначення нової схеми, так і для модифікації вже існуючої. Ця мова не можна використати для керування даними.

Результатом компіляції DDL-операторів є набір таблиць, збережений в особливих файлах, що називаються *системним каталогом*. У системному каталозі інтегровані *метадані* — тобто дані, які описують об'єкти бази даних, а також дозволяють спростити спосіб доступу до них і керування ними. Метадані включають визначення записів, елементів даних, а також інші об'єкти, що представляють інтерес для користувачів або необхідні для роботи СУБД. Перед доступом до реальних даних СУБД звичайно звертається до системного каталогу. Для позначення системного каталогу також використовуються терміни *словник даних* і *каталог даних*,

хоча перший з них (словник даних) звичайно відноситься до програмного забезпечення більше загального типу, чим просто каталог СУБД. Системні каталоги більш докладно обговорюються в розділі 2.10.

Теоретично для кожної схеми в трирівневій архітектурі можна було б виділити кілька різних мов DDL, а саме мова DDL зовнішніх схем, мова DDL концептуальної схеми й мова DDL внутрішньої схеми. Однак на практиці існує одна загальна мова DDL, що дозволяє задавати специфікації, як мінімум, для зовнішньої й концептуальної схем.

Мова DML. Мова, що містить набір операторів для підтримки основних операцій маніпулювання даними, що втримувалися в базі.

До операцій керування даними відносяться:

- вставка в базу даних нових відомостей;
- модифікація відомостей, збережених у базі даних;
- витяг відомостей, що містяться в базі даних;
- видалення відомостей з бази даних.

Таким чином, одна з основних функцій СУБД полягає в підтримці мови маніпулювання даними, за допомогою якого користувач може створювати вирази для виконання перерахованих вище операцій з даними. Поняття маніпулювання даними застосовно як до зовнішнього й концептуального рівнів, так і до внутрішнього рівня. Однак на внутрішньому рівні для цього необхідно визначити дуже складні процедури низького рівня, що дозволяють виконувати доступ до даних досить ефективно. На більше високих рівнях, навпаки, акцент переноситься убік більшої простоти використання, і основні зусилля направляються на забезпечення ефективної взаємодії користувача із системою.

Частина непроцедурної мови DML, що відповідає за витяг (добування) даних, називається *мовою запитів*. Мову запитів можна визначити як високорівневу вузькоспеціалізовану мову, призначену для задоволення різних вимог по вибірці інформації з бази даних. У цьому змісті термін "запит" зарезервований для позначення оператора витягу даних, вираженого за допомогою мови запитів. Терміни "мова запитів" і "мова керування даними" часто використовуються як синоніми, хоча з технічної точки зору це некоректно.

Мови DML мають різні базові конструкції добування даних. Існують два типи мов DML: *процедурний* і *непроцедурний*. Основне розходження між ними полягає в тім, що процедурні мови вказують те, як можна одержати результат оператора мови DML, тоді як непроцедурні мови описують те, який результат буде отриманий. Як правило, у процедурних

мовах записи розглядаються окремо, тоді як непроцедурні мови оперують із цілими наборами записів.

Процедурні мови DML

Процедурна мова DML. Мова, що дозволяє повідомити системі про те, які дані необхідні, і точно вказати, як їх можна витягти.

За допомогою процедурної мови DML користувач, а точніше – програміст, указує на те, які дані йому необхідні і як їх можна одержати. Це значить, що користувач повинен визначити всі операції доступу до даних (здійснювані за допомогою виклику відповідних процедур), які повинні бути виконані для одержання необхідної інформації.

Звичайно така процедурна мова DML дозволяє витягти запис, обробити його й, залежно від отриманих результатів, витягти інший запис, що повинен бути підданий аналогічній обробці, і т.д. Подібний процес добування даних триває доти, поки не будуть витягнуті всі дані, що запитуються. Звичайно оператори процедурної мови DML вбудовуються в програму мовою програмування високого рівня, що містить конструкції для забезпечення циклічної обробки й переходу до інших ділянок коду. Мови DML мережних і ієрархічних СУБД звичайно є процедурними.

Непроцедурні мови DML

Непроцедурна мова DML. Мова, що дозволяє вказати лише те, які дані потрібні, але не те, як їх варто витягати.

Непроцедурні мови DML дозволяють визначити весь набір необхідних даних за допомогою одного оператора вибірки або відновлення. За допомогою непроцедурних мов DML користувач указує, які данные йому потрібні, без визначення способу їхнього одержання. СУБД транслює вирази мовою DML у процедуру (або набір процедур), що забезпечує маніпулювання викликаним набором записів.

Такий підхід звільняє користувача від необхідності знати подробиці внутрішньої реалізації структур даних і особливості алгоритмів, що використовуються для добування й можливого перетворення даних. У результаті робота користувача стає деякою мірою незалежної від даних.

Непроцедурні мови часто також називають декларативними мовами. Реляційні СУБД у тій або іншій формі звичайно включають підтримку непроцедурних мов маніпулювання даними — найчастіше це мова структурованих запитів SQL (Structured Query Language) або мова запитів за зразком QBE (Query-by-Example). Непроцедурні мови звичайно простіше зрозуміти й використати, чим процедурні мови DML, оскільки користувачем виконується менша частина роботи, а СУБД — більша. Більш докладно мова SQL розглядається в главах 9 і 11.

5.1 Мова запитів SQL

Вибірка даних – найбільш розповсюджена операція під час роботи з базою даних. Вона здійснюється запитами до БД – командою мови SQL SELECT. Команда SELECT отримує записи з бази даних та дозволяє вибірку одного чи більше рядків або стовпців з однієї або декількох таблиць. Результатом дії будь-якої команди SELECT є нова таблиця, тому можливі багаторівневі запити, в яких вкладені запити (запити нижнього рівня), повертають таблицю, що використовується у запиті вищого рівня.

Взаємодія з більшістю реляційних баз даних, зокрема з СУБД SQL Server, здійснюється за допомогою мови SQL або його діалекту. SQL – це, фактично, стандарт для операцій з базами даних. Мова SQL є найбільш поширеною мовою управління базами даних типу клієнт/сервер. Основна перевага SQL полягає в тому, що вона уніфікована: стандартний набір інструкцій SQL можна використати у будь-якій системі управління базами даних, сумісною з SQL.

Перший американський стандарт SQL був зареєстрований в 1986 р. як ANSI X3.135-1986. Стандартом версії MS SQL Server 2000 є ANSI X3.135-1992, широко відомий як SQL-92.

SQL є мовою реляційних баз даних, а не мовою системного програмування. SQL – це мова, орієнтована на роботу з множинами. Таким чином, ANSI SQL не включає ні засобів управління виконанням програми (розгалуження і цикли), ні засобів для створення форм або звітів. Функції управління реалізуються в мовах програмування, наприклад, мовах об’єктно-орієнтованого програмування, які застосовуються для створення застосувань до баз даних. Проте у версії мови SQL деяких СУБД додані оператори розгалуження та циклу, що дозволяє використовувати ці мови у якості процедурної мови, наприклад, для розробки серверного програмного забезпечення у вигляді процедур, що зберігаються на сервері (збережені процедури, розділ 5.2).

Наприклад, в мову Transact-SQL, що використовується в Microsoft SQL Server, додані два оператори (IF ELSE і WHILE). Також Transact-SQL додає до базової мови SQL ключові слова, що дозволяють формувати інструкції вибірки, а також збереження даних і маніпуляцій над ними. При розробці діалекту SQL, вживаного в SQL Server, фірма Microsoft додала до нього додаткові можливості і розширення, що роблять і багато інших розробників систем управління базами даних.

В порівнянні з іншими діалектами мови SQL, синтаксис Transact-SQL найбільш близький до стандарту. Цим він відрізняється, наприклад, від діалекту SQL-PLUS, вживаного для реляційних баз даних Oracle, який має найбільш специфічний додатковий синтаксис. Використання

синтаксису, характерного тільки для конкретного діалекту, не дозволяє переносити набори команд SQL, що зберігаються на диску, і може викликати великі труднощі при міжсистемному перенесенні.

Мову Transact-SQL можна характеризувати як оптимальну і строгу. Крім того, є достатній набір доповнень до базової мови SQL, що дозволяє формувати функціональні запити. Таким чином, подібно до усіх інших мов програмування – а SQL являється мовою програмування баз даних – Transact-SQL є набором інструкцій (чи операторів), правильне застосування яких дозволяє виконати поставлене завдання.

Хоча мова SQL і є стандартом мови для реляційних баз даних, існує багато «діалектів» мови SQL, тобто мова SQL конкретної СКБД має деякі, властиві саме їй, відмінності. Тому розглянемо головну команду мови SQL – команду SELECT – в контексті СКБД MS SQL Server 2000.

5.1.1 Синтаксис команди SELECT

Для створення запиту на вибірку даних використовується команда SELECT. Повний синтаксис цієї команди досить складний, але можна перелічити основні розділи команди, а саме:

SELECT *select_list*
[INTO *new_table*]
FROM *table_source*
[WHERE *search_condition*]
[GROUP BY *group_by_expression*]
[HAVING *search_condition*]
[ORDER BY *order_expression* [ASC | DESC]]

Розділ SELECT (обов'язковий): SELECT *select_list* – він вказує стовпці таблиці (*select_list*), яка повертається запитом.

Розділ FROM (обов'язковий): FROM *table_source* – він вказує з яких таблиць (*table_source*): базових відношень, представлень, віртуальних таблиць, – отримуються дані.

Необов'язкові розділи:

INTO *new_table*

Створює таблицю і записує до неї результат запиту (без цього розділу результат запиту є віртуальною таблицею).

WHERE *search_condition*

Містить умови пошуку рядків таблиць, які повинні брати участь у запиті.

GROUP BY *group_by_expression*

Містить перелік стовпців таблиці, за якими буде проводитись групування вибраних даних для отримання характеристик цих груп:

кількості записів в групі, середнього значення, суми, максимуму або мінімуму якогось поля (чи виразу з полів).

HAVING *search_condition*

Містить умови відбору рядків результуючої таблиць, коли в запиті використовується групування; в цьому розділі, на відміну від розділу WHERE, звичайно записуються умови для груп; в одному запиті (з групуванням) можуть бути умови в розділі WHERE (на записи в таблицях, наприклад, будинки з кількістю поверхів не менше 9), та умови на групи в розділі HAVING (наприклад, групування виконується по вулицям, та вибираються вулиці, на яких не менше 10 багатоповерхових будинків).

ORDER BY *order_expression*

Містить перелік стовпців результуючої таблиці, за якими виконується упорядкування даних. Сортування може проводитись по декількох стовпцях (багаторівневе упорядкування).

Розглянемо докладніше деякі розділи команди SELECT.

Розділ SELECT

Вказує стовпці таблиці, яка повертається запитом.
Синтаксис команди:

```
SELECT [ ALL | DISTINCT ]
  [ TOP n [ PERCENT ] [ WITH TIES ] ]
  < select_list >
```

Параметри команди:

ALL

Вказує, що однакові (дублюючі) рядки повертають запитом. За замовчуванням (коли не вказано ALL або DISTINCT) використовується ALL. Однакові рядки можуть з'явитися в результату запиту з таблиць, в яких немає дублюючих записів. Наприклад, з бази даних підприємства обирається наступна інформація про працівників: прізвище, ім'я, посада. Зрозуміло, що на підприємстві можуть виявитись працівники з однаковими іменем та прізвищем на однаковій посаді. В них є характеристики, що відрізняють їх один від одного, принаймні такі, як ідентифікаційний номер та серія і номер паспорту. Тому в базі даних немає дублюючих записів, але в запиті вони з'являються за рахунок того, що вибирається часткова інформація про працівників.

DISTINCT

Вказує, що однакові (дублюючі) рядки не повертають запитом.

TOP *n* [PERCENT]

Вказує, що повертаються тільки перші *n* рядків з таблиці, що є результатом запиту. *n* є ціле число у діапазоні між 0 та 4294967295. Якщо є ключове слово PERCENT, то повертається *n* відсотків записів з результуючої таблиці. Наприклад, в результаті запиту без слова TOP повинна була повернутись таблиця з 40 рядків. Якщо додати до запиту: TOP 10 – повернеться таблиця з 10 рядків. Якщо додати до запиту: TOP 10 PERCENT – повернеться таблиця з 4 рядків. Параметр звичайно застосовують в запитах з розділом ORDER BY, тобто тих, що повертають упорядковані дані. Наприклад, якщо запит повертає прізвище, ім'я та зарплатню працівників підприємства, і є упорядкування за убуванням зарплатні, то наявність в запиті опції: TOP 10 – дозволить вибрати 10 працівників з самою високою зарплатнею.

WITH TIES

Ці ключові слова параметра TOP вказують, що в результат запиту потрібно включити всі записи з однаковим значенням атрибуту упорядкування, якщо хоч один запис входить до кількості TOP *n* (або TOP *n* PERCENT). Наприклад, є 8 працівників з самою високою зарплатнею, і 4 працівника з однаковою зарплатнею дещо нижчою, але більшою за решту працівників. Тоді запит з параметром: TOP 10 – поверне 10 записів; а запит з параметром: TOP 10 WITH TIES – поверне 12 записів, тобто всіх 4 працівників з однаковою зарплатнею.

< select_list >

Перелік стовпців таблиці, яка повертаються запитом. В переліку окремі пункти вибору (імена атрибутів базових таблиць, представлень, віртуальних таблиць, або виразів для вирахування значення в стовпці) відокремлюються один від одного комою.

Опція < select_list > має наступний синтаксис:

```
< select_list > ::=
    {    *
       | { table_name | view_name | table_alias }.*
       |    { column_name | expression | IDENTITYCOL | ROWGUIDCOL }
          [ [ AS ] column_alias ]
       | column_alias = expression
    }   [ ,...n ]
```

Параметри:

*

Вказує, що повертаються всі поля всіх таблиць, що є в розділі FROM. Тобто команда: SELECT * FROM *table_name* – поверне просто всю таблицю з іменем *table_name,* як вона є.

{*table_name* | *view_name* | *table_alias* }**.***

Обмежує зону дії параметру * одною вказаною таблицею. Тут в якості імені таблиці може використовуватись або ім'я базового відношення БД (*table_name*), або ім'я представлення (*view_name*), або псевдонім базової таблиці, представлення, віртуальної таблиці (*table_alias*). Псевдоніми визначаються в розділі FROM (розглядається нижче). Наприклад, з бази даних `City` потрібно вибрати інформацію про всі будинки на всіх вулицях, але для кожного будинку вказати назву вулиці. В таблиці `House` є тільки ідентифікатор вулиці, але назва вулиці є в таблиці `Street`. Отже, наступний запит вибере потрібну інформацію:

```
SELECT name_street, House.*
FROM Street, House
WHERE Street.id_street=House.id_street
```

column_name

Ім'я стовпця в таблиці, що повертається запитом. Потрібно кваліфікувати ім'я стовпця (уточнювати іменем таблиці), якщо існують стовпці з дублюючими іменами в таблицях, що перелічені в розділі `FROM`. Наприклад, атрибут `id_street` є як в таблиці `Street`, так і в таблиці `House`. Якщо запит вибирає цей атрибут, і в розділі `FROM` є обидві таблиці, то запис імені стовпці просто `id_street` викликає помилку: `Ambiguous column name 'id_street'`. Щоб уникнути подібної помилки, треба написати `Street.id_street` (або `House.id_street`).

expression

Вираз, що обраховує значення в стовпці. Це може бути ім'я атрибуту таблиці, константа, функція, будь-яка комбінація атрибутів, констант, і функцій у вигляді виразу, або вкладений запит (підзапит).

IDENTITYCOL

Повертає атрибут, що описаний в команді створення таблиці з властивістю IDENTITY. Якщо в розділі FROM є декілька таблиць, що мають такі атрибути, необхідно кваліфікувати поле, наприклад: `Street.IDENTITYCOL`

column_alias

Альтернативне ім'я стовпця (псевдонім стовпця) в результуючій таблиці. Наприклад, можна написати:

```
SELECT name_stree AS "Назва вулиці"
```

Якщо псевдонім стовпці є ідентифікатором (тобто містить тільки літери, цифри та знак підкреслення й починається є літери або підкреслення), то необов'язково включати його в обмежувачі. У наведеному прикладі псевдонім стовпця повинен бути в обмежувачах (подвійних лапках), тому що в ньому є символ пробілу (пропуску).

Псевдоніми стовпців потрібні в тих випадках, коли для обчислення значення в стовпці використовується вираз. Якщо такому стовпцю не надати псевдоніма, то його ім'я буде: `(No column name)`.

column_alias може використовуватись в розділі ORDER BY, якщо виконується упорядкування за вказаним стовпцем. Але в розділах WHERE, GROUP BY, HAVING заборонено використовувати псевдоніми стовпців. Ключове слово AS перед псевдонімом стовпця можна пропускати.

column_alias = *expression*

Аналогічно запису: *expression* AS *column_alias*

[**,**...*n*]

В розділі SELECT можна через кому записати декілька стовпців результуючої таблиці за описаними вище правилами.

Розділ FROM команди SELECT

Вказує таблиці, представлення, віртуальні таблиці (що вираховуються вкладеним запитом) та з'єднані таблиці, що використовуються у командах DELETE, SELECT, UPDATE.

Синтаксис команди:

```
FROM { < table_source > } [ ,...n ]

< table_source > ::=
   table_name [ [ AS ] table_alias ]
   | view_name [ [ AS ] table_alias ]
   | derived_table [ AS ] table_alias
   | < joined_table >

< joined_table > ::=
   < table_source > < join_type > < table_source > ON < search_condition >
   | < table_source > CROSS JOIN < table_source >


< join_type > ::=
   [ INNER | { { LEFT | RIGHT | FULL } [ OUTER] } ]
   [ < join_hint > ]
   JOIN
```

Параметри команди:

table_name

Ім’я таблиці.

[AS] *table_alias*

Псевдонім таблиці, під яким вона буде фігурувати у інших розділах команди SELECT (DELETE, UPDATE).

< joined_table >

Результуючий набір, що є декартовим добутком двох або більше таблиць. Як відомо, добуток таблиць R і S визначає нове відношення, що є результатом конкатенації (тобто зчеплення) кожного кортежу з відношення R з кожним кортежем з відношення S. Деякі види з’єднання таблиць R і S визначають відношення, що містять деяку підмножину декартового добутку таблиць R і S; але це не зовсім вірно щодо зовнішніх з’єднань. Розділ FROM дозволяє реалізувати як різні види з’єднань таблиць (з опцією <table_source> <join_type> <table_source> ON <search_condition>) так і операцію реляційної алгебри *перетинання* (з опцією < table_source > CROSS JOIN < table_source >).

< join_type >

Вказує тип операції з’єднання.

INNER

Вказує, що повертаються всі пари рядків, які відповідають умові з’єднання. Виключаються рядки обох таблиць, які на відповідають умові з’єднання. Цей тип зв’язку є типом зв’язку за замовчуванням, тобто конструкція:

```
table_name1 JOIN table_name2 ON search_condition
```

еквівалентна конструкції:

```
table_name1 INNER JOIN table_name2 ON search_condition
```

FULL [OUTER]

Вказує, що повертаються всі рядки обох таблиць. При цьому, ті рядки, що відповідають умові з’єднання таблиць, повертаються зв’язаними один з одним (як для зв’язку INNER JOIN). Але повертаються і додаткові рядки: всі рядки з обох таблиць повинні бути повернуті. Якщо для рядка з лівої таблиці (ліворуч від ключового слова FULL) немає відповідного рядка в правій таблиці, він повертається доповнений значеннями NULL для всіх атрибутів правої таблиці. Аналогічно, якщо для рядка з правої таблиці (праворуч від ключового слова FULL) немає відповідного рядка в

лівій таблиці, він повертається доповнений значеннями NULL для всіх атрибутів лівої таблиці.

LEFT [OUTER]

На відміну від `FULL JOIN` не повертаються рядки правої таблиці, які не мають пари в лівій таблиці. Тобто повертаються всі рядки лівої таблиці, або зв'язані з рядками правої таблиці (якщо вони відповідають умові з'єднання), або доповнені значеннями NULL для всіх атрибутів правої таблиці (якщо вони не відповідають умові з'єднання).

RIGHT [OUTER]

На відміну від `FULL JOIN` не повертаються рядки лівої таблиці, які не мають пари в правій таблиці. Тобто повертаються всі рядки правої таблиці, або зв'язані з рядками лівої таблиці (якщо вони відповідають умові з'єднання), або доповнені значеннями NULL для всіх атрибутів лівої таблиці (якщо вони не відповідають умові з'єднання).

JOIN

Вказує, що визначений оператор зв'язку використовується до заданих таблиць або представлень.

ON <search_condition>

Вказує умову зв'язку таблиць. Умова повинна бути предикатом, в якому використовуються атрибути таблиць та операції порівняння. Наприклад, якщо в запиті використовуються таблиці `Street` і `House` (БД `City`), то умова зв'язку буде виглядати так:

```
ON Street.id_street=House.id_Street
```

Якщо в запиті використовуються таблиці `Flat` і `House` (БД `City`), то умова зв'язку буде виглядати так:

```
ON Flat.id_street=House.id_Street AND
    Flat.id_house=House.id_house
```

Розділ WHERE

Вказує умову, якій повинні відповідати рядки, що повертаються запитом.

Синтаксис команди:

WHERE < search_condition >

Параметри:

<search_condition>

Умова, якій повинні відповідати рядки, що повертаються запитом. Умова є комбінацією одного чи більше предикатів з використанням логічних операторів AND, OR, NOT. В цьому розділу крім умови пошуку рядків (умови фільтру), можна також включати умови зв'язків між таблицями, тоді в розділі FROM таблиці лише перелічуються через кому.

Формат запису умови:

```
< search_condition > ::=
  {    [ NOT ] < predicate > | ( < search_condition > )
     [ { AND | OR } [ NOT ] { < predicate > | ( < search_condition > ) } ]
  }

< predicate > ::=
  {    expression { = | < > | ! = | > | > = | ! > | < | < = | ! < } expression
     | string_expression [ NOT ] LIKE string_expression
     | expression [ NOT ] BETWEEN expression AND expression
     | expression IS [ NOT ] NULL
     | expression [ NOT ] IN ( subquery | expression [ ,...n ] )
     | expression { = | < > | ! = | > | > = | ! > | < | < = | ! < }
        { ALL | SOME | ANY} ( subquery )
  }
```

Параметри:

NOT

Інвертує значення логічного виразу: `NOT TRUE=FALSE, NOT FALSE=TRUE.`

AND

Комбінує дві умови та обраховує TRUE, якщо результат обох умов є TRUE.

OR

Комбінує дві умови та обраховує TRUE, якщо результат хоча б одної з умов є TRUE.

< predicate >

Вираз, який повертає значення TRUE, FALSE, або UNKNOWN.

expression

Атрибут таблиці, константа, функція, скалярний підзапит (тобто той, що повертає одне значення), або будь-яка комбінація атрибутів, констант, функцій, зв'язаних операторами.

Оператори порівняння (= , < > , ! = , > , > = , ! > , < , < = , ! <) докладно тут не розглядаються, тому що вони вже використовувались в інших дисциплінах на молодших курсах.

string_expression

Рядок символів та шаблонів символів.

[NOT] LIKE

Вказує, що символьний рядок праворуч містить шаблон символів для порівняння.

[NOT] BETWEEN

Вказує діапазон значень. Ключове слово AND відділяє початкове значення діапазону від кінцевого значення.

IS [NOT] NULL

Вказує пошук порожніх значень (NULL), або не порожніх значень (при наявності слова NOT).

[NOT] IN **(** *subquery* | *expression* [**,***...n*] **)**

Вказує на пошук рядків таблиць, для яких значення виразу ліворуч від ключового слова IN, приймає значення, яке є в множині значень, записаних у круглих дужках праворуч від ключового слова IN (або немає в множині значень, з словом NOT). Множина значень задається або переліком виразів (через кому), або вкладеним запитом, який повертає один стовпець значень.

ALL

Використовується з оператором порівняння та вкладеним запитом. Повертає TRUE для предикату, якщо всі значення, в множині значень, що повертаються вкладеним запитом, задовольняють оператору порівняння, або FALSE, якщо не всі значення з множини задовольняють оператору порівняння, або якщо вкладений запит не повертає жодного рядка. Вкладений запит повинен повертати таблицю з одного стовпця.

{ SOME | ANY }

Використовується з оператором порівняння та вкладеним запитом. Повертає FALSE для предикату, якщо всі значення, в множині значень, що повертаються вкладеним запитом, не задовольняють

оператору порівняння, або якщо вкладений запит не повертає жодного рядка, та повертає TRUE, якщо деякі значення з множини задовольняють оператору порівняння, Вкладений запит повинен повертати таблицю з одного стовпця.

Приклади правильних предикатів:

```
rooms>3
YEAR(DATE())- YEAR(birthday)>=50
name_street LIKE '%басов%'
count_flat IS NOT NULL
id_house IN (SELECT id_house FROM House WHERE
count_floor>5)
count_flat>ANY (SELECT count_flat FROM House H, Street S
                WHERE H.id_street=S.id_street AND
                    name_street='Дерібасовська')
```

Розділ GROUP BY

Вказує групи, для яких розраховуються значення за допомогою агрегатних функцій: суму, кількість записів, середнє значення, максимальне значення, мінімальне значення. Якщо є розділ GROUP BY, в розділі SELECT крім виразів з агрегатними функціями можуть бути тільки стовпці, перелічені в розділі GROUP BY. Якщо в команді SELECT є розділ GROUP BY і немає розділу ORDER BY, упорядкування рядків результуючого набору проводиться за пунктами групування.

Синтаксис команди:

GROUP BY [ALL] *group_by_expression* [**,***...n*]

Параметри:

ALL

Вказує, що всі групи повинні бути представлені у результуючому наборі, навіть якщо в них немає жодного запису. Коли вказаний параметр ALL, для групи без записів (для якої умова фільтру в розділі WHERE не знайде жодного запису) функція COUNT повертає значення арифметичного нуля, інші агрегатні функції (MAX, MIN, SUM, AVG) повертають значення NULL. Якщо параметр не вказаний, група без записів не включається в результат запиту.

group_by_expression

Вказує вираз, за яким групуються запити. Найчастіше вираз складається з імені стовпця, за яким проводиться групування, але це може бути неагрегатний вираз з використанням атрибутів стовпців.

Якщо в розділі SELECT для цього виразу був наданий псевдонім стовпця, то в розділі GROUP BY повинен використовуватись вираз, а не псевдонім.

[,...*n*]

Групування, як і упорядкування, може бути багаторівневим.

Приклади використання розділу GROUP BY.

Вирахувати кількість будинків на кожній вулиці, при цьому включити в результат вулиці, на яких немає будинків (якщо в цьому запиті описати зв'язок між таблицями в розділі FROM, то вулиці без будинків не будуть вибрані не зважаючи на параметр ALL):

```
SELECT name_street, COUNT(*) AS kol_vo
FROM street s, house h WHERE s.id_street=h.id_street
GROUP BY ALL name_street
```

Вирахувати загальну кількість кімнат по всіх квартирах будинку, та середню площу квартири для кожного будинку на кожній вулиці, при цьому виключити з результату будинки, в яких немає квартир:

```
SELECT name_street+' д.'+h.number AS house_N,
          SUM(rooms) as kol_vo, AVG(all_square) as AVG_Square
FROM street s, house h,flat f
WHERE s.id_street=h.id_street AND h.id_street=f.id_street AND
      f.number=h.number
GROUP BY name_street+' д.'+h.number
```

Розділ HAVING

Вказує умову пошуку для груп або агрегатних функцій.

Синтаксис команди:

HAVING < search_condition >

Параметр:

< search_condition >

Вказує умову пошуку для груп з використанням агрегатних функцій. Наприклад, якщо в останньому запиті потрібно вибрати тільки будинки з загальною кількістю кімнат не менше 20, то після розділу: GROUP BY … – буде розділ HAVING:
HAVING COUNT(*)>=20

Приклад: вирахувати кількість будинків на кожній вулиці, при цьому включити в результат тільки ті вулиці, на яких більше 5 будинків:

```
SELECT name_street, COUNT(*) AS kol_vo
FROM street s, house h WHERE s.id_street=h.id_street
GROUP BY ALL name_street
```

HAVING COUNT(*)>=5

В одному запиті можуть бути як розділ WHERE, так і розділ HAVING. Звичайно в розділі WHERE записуються умови пошуку даних в таблицях бази даних, а в розділі HAVING – умови пошуку груп даних, отриманих в результату виконання запиту.

Розділ ORDER BY

Вказує сортування результуючого набору. Розділ ORDER BY є неприпустимим у представленнях та вкладених запитах; але може використовуватись в них, якщо в них вказаний параметр TOP.

Синтаксис команди:

ORDER BY { *order_by_expression* [ASC | DESC] } [**,**...*n*]

Параметри:

order_by_expression

Вказує стовпець, по якому проводиться упорядкування. Стовпець сортування вказується по імені, псевдоніму або виразу, що використовується для його обчислення. Також можна вказати ціле число (починаючи з 1), яке відповідає позиції стовпця упорядкування, рахуючи зліва направо.

Припускається багаторівневе упорядкування. Послідовність стовпців сортування у розділі ORDER BY визначає організацію сортування у результуючому набору. Можна сортувати за атрибутами таблиць, які не вказані в розділі SELECT. Не можна сортувати за стовпцями з даними типу **ntext**, **text**, або **image.**

ASC

Вказує, що значення в зазначеному стовпці повинні упорядковуватись в зростаючому (ascending) порядку від найменшого значення до найбільшого. Цей порядок обирається за замовчуванням.

DESC

Вказує, що значення в зазначеному стовпці повинні упорядковуватись в порядку убування (descending) від найбільшого значення до найменшого.

5.1.2 Вибірка даних командою SELECT

Для прикладів використання команди SELECT будемо використовувати базу даних PERSONAL (Персонал підприємства), що має наступну схему:

Otdel (id_otd, name_otd)

Post (id_post, name_post, post_money, days, count_post)

Persona (id_man, PassNo, surname, name1, name2, birthdate, married, children, education)

Worker (id_man, id_otd, id_post, main_work, date_work, stavka)

Vacation (id_man, Year_vac, date_vac)

Тут таблиця Otdel містить опис відділів підприємства; таблиця Post – опис посад, що є на підприємстві; Persona містить особові дані працівників підприємства; Worker – це аналог трудової книжки працівника: хто в якому відділі працює, на якій посаді, сумісник чи основний працівник, з якої дати прийнятий на роботу, на яку частину ставки; таблиця Vacation містить дані про те, коли який працівник йде у щорічну відпустку, та рік, за який ця відпустка призначена.

Логічна схема бази даних наведена на рис.5.1.

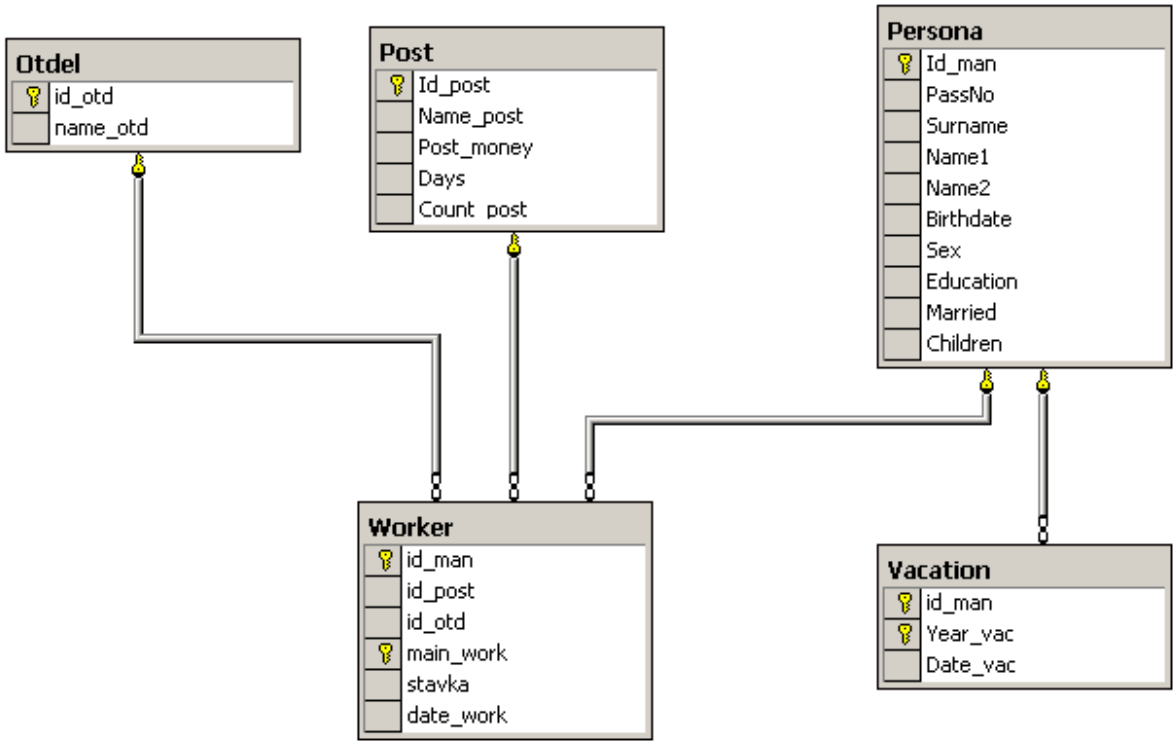


Рис.5.1 Логічна схема бази даних PERSONAL

Запит з використанням полів, що обчислюються

1. Для працівників з вищою освітою, які працюють не на повну ставку, вибрати: прізвище, ім’я, вік, назву посади, зарплату (зарплата розраховується за формулою post_money*Stavka).

Атрибути: прізвище, ім’я та дата народження (за яким розраховується вік), – належать таблиці Persona; атрибут посада – таблиці Post; зв'язок між ними можна встановити тільки через таблицю Worker: таблиця Persona зв’язується з таблицею Worker по ключу Id_man, таблиця Worker зв’язується з таблицею Post по ключу Id_post. Встановимо зв'язок в розділі FROM; в розділі WHERE будуть умови фільтру (вища освіта та неповна ставка):

```
SELECT Surname as Фамилия, Name1 as Имя,
       YEAR(GETDATE())-YEAR(birthdate) as Возраст,
       Name_post Должность,
       STR(post_money*Stavka,7,2)as Зарплата
FROM persona p INNER JOIN worker w ON p.id_man=w.id_man
    INNER Join post  ON w.id_post=post.id_post
WHERE Education ='высшее' and stavka<1
```

Результат виконання команди видно на рисунку 5.2.

	Фамилия	Имя	Возраст	Должность	Зарплата
1	Антонова	Ольга	36	Секретарь-референт	2000.00
2	Ильин	Антон	28	Бухгалтер	2000.00
3	Петров	Олег	40	Секретарь-референт	2000.00
4	Антонюк	Ольга	36	Менеджер	2625.00
5	Иванникова	Ольга	52	Прибиральниця	1000.00
6	Петренко	Семен	40	Секретарь	1250.00
7	Алексеев	Ольга	36	Менеджер	2625.00
8	Климов	Антон	46	Менеджер	2625.00

Рисунок 4 – Результат команди SELECT

Мова SQL, як і інші мови програмування, має велику бібліотеку функцій, які можна використовувати у запитах. Одними з широко використовуваних функцій є функції перетворення типів даних.

Однієї з основних характеристик стовпця є *тип даних* (data type). Тип даних визначає діапазон значень, які можна буде зберігати в стовпці.

Функції **CAST** і **CONVERT**

Конвертують вираз одного типу даних в інший. Функції виконують аналогічні дії, але CONVERT має трохи більше можливостей.

Синтаксис

CAST **(** *expression* AS *data_type* **)**

CONVERT **(** *data_type* [**(** *length* **)**] **,** *expression* [**,** *style*] **)**

Аргументи:

expression

Будь-який припустимий для синтаксису Microsoft SQL Server вираз.

data_type

Це тип даних, в який потрібно конвертувати *expression.* Це потрібен бути системний тип даних; типи даних, що визначені користувачем, не припустимі.

length

Довжина символьного рядка. Необов'язковий параметр, що використовується при перетворенні у типи: **nchar**, **nvarchar**, **char**, **varchar**, **binary**, або **varbinary.**

style

Це є стиль завдання формату дати/часу під час перетворення типів **datetime** або **smalldatetime** в символьні типи (**nchar**, **nvarchar**, **char**, **varchar**, **nchar**, або **nvarchar**), або формат символьного рядка під час перетворення типів **float**, **real**, **money**, або **smallmoney** в символьні типи (**nchar**, **nvarchar**, **char**, **varchar**, **nchar**, або **nvarchar**).

У таблиці 3 перелічені основні формати запису дати. Повний перелік форматів запису дати/часу можна отримати у файлі допомоги (Help) програми Query Analyzer SQL Server.

Таблиця 3 – Нумерація стилів формату дати/часу

№ стилю без століття (yy)	**№ стилю зі століттям (yyyy)**	**Вхідне/вихідне значення**
1	101	mm/dd/yy
2	102	yy.mm.dd
3	103	dd/mm/yy
4	104	dd.mm.yy
5	105	dd-mm-yy
8	108	hh:mm:ss
14	114	hh:mi:ss:mmm(24h)
	20 або 120	yyyy-mm-dd hh:mi:ss(24h)
	21 або 121	yyyy-mm-dd hh:mi:ss.mmm(24h)

У таблиці 3 третій стовпчик називається “Вхідне/вихідне значення”, тобто можна конвертувати символьний рядок у дату/час (вхідне значення символьного рядка для обраного стилю запису дати/часу повинного відповідати формату запису, представленому у 3-му стовпчику таблиці 3), або конвертувати дату/час у символьний рядок – тоді 3-й стовпчик таблиці 3 демонструє, як буде виглядати символьний рядок з записом дати/часу.

Зауваження: Якщо використовується дата без століття, то SQL Server інтерпретує двоцифровий запис року (yy) як рік двадцятого століття, якщо він знаходиться в діапазоні від 50 до 99, і як рік 21 століття – якщо двоцифрове число в діапазоні від 0 до 49.

Таким чином функція:

```
CONVERT(datetime,'15/11/40',3)
```

поверне дату: 15 листопаду 2040 року; а функція:

```
CONVERT(datetime,'07.20.70',1)
```

поверне дату: 20 липня 1970 року.

Для впевненості правильності завдання дати краще завжди використовувати формат дати з століттям.

Для перетворення числових типів в рядкові (символьні) можна також використовувати функцію STR.

Функція **STR**

Повертає символьні дані, конвертовані з числових даних.

Синтаксис

STR **(** *float_expression* [**,** *length* [**,** *decimal*]] **)**

Аргументи:

float_expression

Вираз приблизного числового типу даних з десятковою крапкою.

length

Загальна довжина рядка символів, що містить десяткову крапку, знак числа (для від’ємних чисел), цифри та пробіли (пропуски). За замовчуванням length = 10.

decimal

Кількість цифр дробової частини числа, тобто кількість цифр праворуч десяткової крапки.

Тип значення, що повертається – **char**

Запит з використанням функцій CASE та CONVERT

2. Для працівників відділу «Адміністрація», які ще не пішли у відпустку поточного року, вибрати: прізвище з ініціалами, назву посади, сезон виходу у відпустку: зима, весна, літо, осінь, – і дату виходу у відпустку у форматі dd/mm/yyyy:

```
SELECT RTRIM(Surname)+' '+LEFT(Name1,1)+'.'+LEFT(Name2,1)+'.'
as [Фамилия И.О.],     Name_post Должность,
[Сезон отпуска]= case
     when month(date_vac) between 3 and 4 then 'весна'
     when month(date_vac) between 6 and 8 then 'лето'
     when month(date_vac) between 9 and 11 then 'осень'
     else 'зима' end,
[Начало отпуска]=convert(char(10),date_vac,103)
FROM persona p INNER JOIN worker w ON p.id_man=w.id_man
    INNER Join post  ON w.id_post=post.id_post
    INNER JOIN vacation v ON p.id_man=v.id_man
    INNER JOIN Otdel ON otdel.id_otdel=w.id_otdel
WHERE  name_otdel='Администрация' and year_vac=year(getdate())
       and date_vac>getdate()
```

Результат виконання команди видно на рисунку 5.3.

	Фамилия И.О.	Должность	Сезон отпуска	Начало отпуска
1	Иванов И.И.	Директор	лето	15/07/2012
2	Сорокина С.П.	Секретарь	лето	01/06/2012
3	Иванова И.И.	Зам.директора	лето	25/07/2012
4	Воробьева С.П.	Системный администратор	осень	02/10/2012
5	Иванникова О.И.	Прибиральниця	весна	25/04/2012

Рисунок 5.3 – Результат команди SELECT з функцією convert

Використання вкладених запитів

3. Для жінок, які не працюють за сумісництвом (не входять в множину працівників-сумісників), вибрати: прізвище з ініціалами, кількість дітей, освіту, відділ, зарплату (зарплата розраховується за формулою `post_money*Stavka`).

```
     select ФИО=surname+'
'+LEFT(Name1,1)+'.'+LEFT(Name2,1)+'.',
      "Кол-во детей"=Children, Образование=Education,
      Отдел=Name_otdel, Зарплата=Post_money*stavka
     FROM persona p INNER JOIN worker w ON p.id_man=w.id_man
```

```
    INNER Join post  ON w.id_post=post.id_post
INNER JOIN Otdel ON w.Id_Otdel=Otdel.Id_otdel
WHERE Sex='ж' and w.Id_man NOT IN
(SELECT Id_man FROM Worker WHERE Main_work=0)
```

Результат виконання команди видно на рисунку 6.

	ФИО	Кол-во детей	Образование	Отдел	Зарплата
1	Петрова О.И.	2	высшее	Бухгалтерия	6000.0000
2	Зимина А.Л.	0	н/высшее	Бухгалтерия	4000.0000
3	Сидорова И.С.	0	н/высшее	Маркетинга	2625.0000
4	Сорокина С.П.	0	среднее	Администрация	2500.0000
5	Иванова И.И.	1	высшее	Администрация	8000.0000
6	Замятина А.Л.	0	н/высшее	Бухгалтерия	4000.0000
7	Антонюк О.П.	2	высшее	Маркетинга	2625.0000
8	Лисицына А.Л.	0	н/высшее	Маркетинга	3500.0000
9	Алексеев О.П.	2	высшее	Маркетинга	2625.0000
10	Ольгина С.П.	0	среднее	Производства	4000.0000
11	Хрущ И.И.	1	высшее	Производства	5000.0000
12	Соколова Л.Л.	0	н/высшее	Производства	4000.0000
13	Медведь И.С.	0	н/высшее	Маркетинга	3500.0000
14	Семенов О.П.	0	высшее	Маркетинга	4000.0000
15	Хрущева С.П.	2	среднее	Производства	3000.0000

Рисунок 6 – Результат команди SELECT

Агрегатні функції

Крім пошуку інформації, яка зберігається в таблицях, за допомогою запитів також можна отримувати деяку статистичну інформацію: максимальне значення деякого атрибута (або вираза з використанням атрибута), мінімальне, середнє значення. Ці можливості надають агрегатні функції. До агрегатних функцій відносяться функції COUNT, SUM, AVG, MAX, MIN. Зазвичай агрегатні функції використовуються у запитах з групуванням і обчислюють кількість записів в групі, суму деякого атрибута для групи, середнє, мінімальне або максимальне значення для групи деякого атрибута. Агрегатні функції можна використовувати не тільки для одного атрибута, но і для виразу з атрибутом, наприклад, для розрахунку середнього (мінімального, максимального) віку групи осіб за їх датою народження.

Приклади запитів з групуванням

1. Для кожного відділу розрахувати кількість чоловіків та кількість жінок. Вибрати: назву відділу, стать, кількість осіб цієї статі в цьому відділі.

```
SELECT Name_otdel as Отдел, sex as Пол, count(*)  Количество
FROM Persona p INNER JOIN worker w ON p.id_man=w.id_man
     INNER JOIN Otdel ON w.Id_Otdel=Otdel.Id_otdel
GROUP BY Name_otdel, sex
```

В цьому запиті є два рівня групування: за статтю та за відділом. Якщо поміняти місцями атрибути в розділі `GROUP BY` (тобто написати `GROUP BY sex, Name_otdel`), то розрахунки залишаться тими самими, але зміниться порядок виводу записів в результаті запиту: або для кожного відділу виводиться спочатку кількість жінок, потім – кількість чоловіків; або спочатку виводиться кількість жінок для кожного відділу, а потім – кількість чоловіків для кожного відділу). Спробуйте виконати цей запит з обома порядками групування та отримайте результат.

2. Для кожного відділу вибрати: назву відділу, кількість працюючих робітників, середній вік працівників, мінімальну зарплатню з урахуванням ставки працівника. Включаються відділи, де є працівники з дітьми.

```
  SELECT Отдел=Name_otdel, [Количество сотрудников]=
count(*),
    AVG(YEAR(GETDATE())-YEAR(birthdate)) as [Средний
Возраст],
  STR(MIN(post_money*Stavka),7,2)) as MIN_Зарплата
  FROM Persona p, Worker w , Post, otdel
  WHERE p.id_man=w.id_man AND w.id_post=Post.id_post AND
    Otdel.id_otdel=w.id_otdel AND w.id_otdel IN
  (SELECT id_otdel FROM Persona p, Worker w
    WHERE p.id_man=w.id_man AND children>0)
  GROUP BY Name_otdel
```

Результат виконання команди видно на рисунку 7.

	Отдел	Количество сотрудников	Средний Возраст	MIN_Зарплата
1	Администрация	8	41	2000.00
2	Бухгалтерия	8	35	1000.00
3	Маркетинга	15	36	1000.00
4	Производства	15	38	1100.00

Рисунок 7 – Результат команди SELECT

6 ТЕОРІЯ НОРМАЛІЗАЦІЇ РЕЛЯЦІЙНОЇ МОДЕЛІ ДАНИХ

У реляційних базах даних схема містить як структурну, так і семантичну інформацію. Структурна інформація пов'язана з оголошенням відношень, а семантична виражається множиною відомих функціональних залежностей між атрибутами відношень, оголошених у схемі. Однак деякі функціональні залежності можуть бути небажаними через побічні ефекти або аномалії, які вони викликають при модифікації бази даних. У зв'язку із цим виникає питання про коректність представленої схеми. Коректної вважається схема, у якій відсутні небажані функціональні залежності.

У протилежному випадку доводиться прибігати до процедури, яка називається декомпозицією (розкладанням), при якій дана множина відношень заміняється іншою множиною відношень (число їх зростає), що є проекціями перших. Ціль цієї процедури — усунути небажані функціональні залежності (а отже, і аномалії), що становить суть процесу нормалізації. Інакше кажучи, нормалізація — це покроковий оборотний процес заміни даної схеми (або сукупності відношень) іншою схемою, у якій відношення мають більш просту й регулярну структуру.

У теорії нормальних форм визначаються різні нормальні форми, які обмежують типи припустимих функціональних залежностей відношень. Як уже було сказано, для приведення відношення до якої-небудь нормальної форми прибігають до декомпозиції. При цьому ми зіштовхуємося із проблемою оборотності, тобто можливості відновлення вихідної схеми. Це означає, що декомпозиція повинна зберігати еквівалентність схем при заміні однієї схеми на іншу.

Для забезпечення еквівалентності схем необхідна декомпозиція, що гарантує відсутність втрат і зберігає залежності. Декомпозиція без втрат гарантує оборотність, тобто одержання вихідної множини відношень шляхом застосування послідовності природних з'єднань над їхніми проекціями. При цьому в результуючому відношенні не повинні з'являтися кортежі, що раніше були відсутніми, що є наслідком помилкового з'єднання. Збереження залежностей має на увазі виконання вихідної множини функціональних залежностей на відношеннях нової схеми.

При проектуванні бази даних необхідно розглядати не тільки дані, які повинні в них зберігатися, але й зв'язки між ними. В одних СУБД (ієрархічні, мережні) зв'язки є частиною самої структури бази даних і містяться в ній у вигляді покажчиків у зв'язаних списках; в інших (реляційних) зв'язки задаються процедурно, як деякі залежності між файлами-відношеннями.

Для розгляду нормальних форм реляційної бази даних необхідно розглянути поняття зв’язків у базі даних та функціональних залежностей.

6.1 Зв’язки та відображення

Відображення – традиційний засіб для визначення характеру взаємозв'язків між парами зв'язаних елементів даних. Нижче наведений короткий перелік типів відображення.

Відображення 1:1

За допомогою відображення 1:1 представляють такий тип зв'язку, коли один екземпляр елемента, від якого спрямований зв'язок, ідентифікує один і тільки один екземпляр елемента, до якого спрямована зв'язок, і навпаки. Ідентифікація унікальна в обох напрямках. На рис. 6.1 показаний приклад відображення 1:1 для елементів даних НОМЕР_СЛУЖБОВЦЯ й ІДЕНТИФІКАЦІЙНИЙ_КОД. Кожний з них унікально ідентифікує іншої.

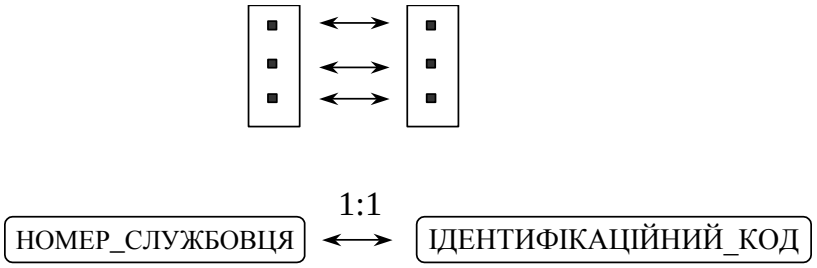


Рис. 6.1 Відображення 1:1.

Відображення 1 :М

Екземпляр елемента даних, від якого спрямований зв'язок, ідентифікує деяке число (нуль, один або декілька) екземплярів елемента даних, до якого спрямована зв'язок, причому ідентифікація в даному напрямку не обов'язково є унікальною. Однак у зворотному напрямку будь-який екземпляр елемента, до якого спрямований зв'язок, ідентифікує один і тільки один екземпляр елемента, від якого спрямована зв'язок.

У прикладі на рис. 6.2 елементи НОМЕР_ВІДДІЛУ й НОМЕР_СЛУЖБОВЦЯ зв'язані між собою відображенням 1:М. У даному відділі звичайно працює багато службовців, але кожний службовець працює тільки в одному відділі (якщо не дозволене внутрішнє сумісництво).

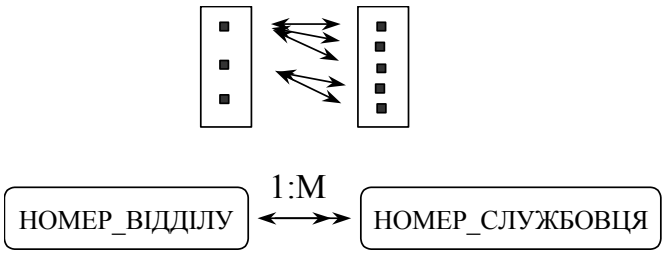


Рис. 6.2 Відображення 1 : М.

Відображення M:1

Це відображення аналогічно відображенню 1:M. Взаємозв'язок між елементами даних є асоціативною.

Відображення M:N

Екземпляр елемента даних, від якого спрямований зв'язок, ідентифікує деяке число екземплярів елемента даних, до якого спрямований зв'язок, і навпаки, тобто ідентифікація є неунікальною в обох напрямках. На рис. 7.4 такими елементами даних є НОМЕР_ВИРОБУ й ПОСТАЧАЛЬНИК. Даний виріб може поставлятися багатьма постачальниками, і даний постачальник може поставляти багато виробів.

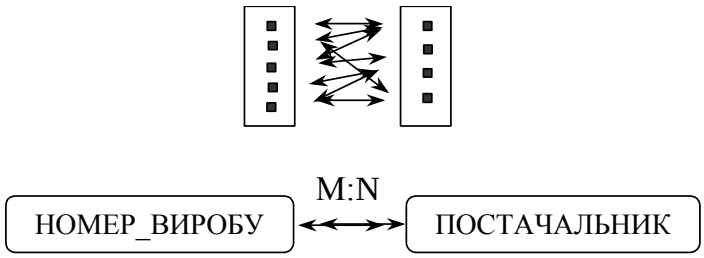


Рис. 6.3 Відображення M : N.

6.2 Основні математичні поняття

Нехай є *множини* A и B. Відношення A **R** У вказує на зв'язок між окремими елементами цих множин. Розрізняють рефлексивні відношення A **R** A (зв'язки між елементами той самої множини), транзитивні (опосередковані зв'язки) і т.д. На практиці використовується деяка значуща інтерпретація зв'язків між множинами й кардинальними числами цих зв'язків (тобто кількості елементів в екземплярі зв'язку). Множини можуть відповідати атрибутам або типам записів. Зв'язки можуть бути функціональними, тобто такими, що задовольняють визначенню математичної функції. Кардинальні числа зв'язків використовуються також для визначення типу відображення між парами множин. Вище говорилося, що існують відображення один-до-одного (1:1), один-до-багатьох (1:M) і багато-до-багатьох (M: N).

Таким образом, зв'язок – це відповідність або відображення між елементами двох (або більше) множин. У таблиці 6.1 наведені деякі приклади сутностей, їхніх властивостей і відповідного подання за допомогою типів записів і атрибутів.

Таблиця 6.1 Приклади сутностей і їхніх властивостей

Область понять		Область інформаційних моделей
Сутності	Властивості	Типи записів (список атрибутів)
Службовці	номер службовця, ідентифікаційний код, ім'я, зарплата	СЛУЖБОВЦІ (НОМЕР_СЛУЖБОВЦЯ, ІДЕНТИФІКАЦІЙНИЙ_КОД, ІМ'Я, ЗАРПЛАТА)
Будинки	номер ділянки, тип, вартість	БУДИНКИ (НОМЕР_ДІЛЯНКИ, ТИП, ВАРТІСТЬ)
Постачальники	Номер постачальника, ім'я, місто	ПОСТАЧАЛЬНИК (НОМЕР_ПОСТАЧАЛЬНИКА, ІМ'Я_ПОСТАЧАЛЬНИКА, МІСТО)
Деталі	Номер деталі, опис деталі, наявність деталей	ДЕТАЛІ (НОМЕР_ДЕТАЛІ, ОПИС_ДЕТАЛІ, НАЯВНІСТЬ_ДЕТАЛЕЙ)

У цих прикладах є різні типи множин. Один з них — це множина аналогічних сутностей, такі, як всі люди, що працюють в організації й називаних СЛУЖБОВЦІ. Аналогічно БУДИНКИ, ПОСТАЧАЛЬНИКИ, ДЕТАЛІ являють собою множини подібних сутностей. Кожна сутність, у свою чергу, представляється своїми властивостями. У результаті кожній множині сутностей тут відповідає кілька множин, що містять значення відповідних властивостей. Інакше кажучи, кожна множина сутностей представляється типом запису, а тип запису характеризується відповідними атрибутами. Екземпляр якого-небудь типу запису відповідає одиничному представникові сутності, такому, як окремий службовець, будинок, постачальник або деталь (так само, як у файловій системі запис файлу службовців відповідає окремій особі). У фізичному наборі даних зберігаються значення, що відповідають множині атрибутів, що становлять збережений екземпляр типу запису. Приведемо приклади деяких зв'язків:

а) кожному номеру службовця відповідає єдиний ідентифікаційний код, і навпаки;

б) деякі службовці мають підлеглих;

в) службовці можуть мати будинки;

г) постачальники поставляють (продають) деталі;

д) тип будинку службовця можна визначити по його вартості.

Подальші міркування будуються на прикладі цих зв'язків. Для більше глибокого розуміння розглянутих понять будуть використані діаграми. На рис. 6.4 показані різні зв'язки між екземплярами різних множин у порядку, що відповідає пунктам а) – д) наведеного вище списку.

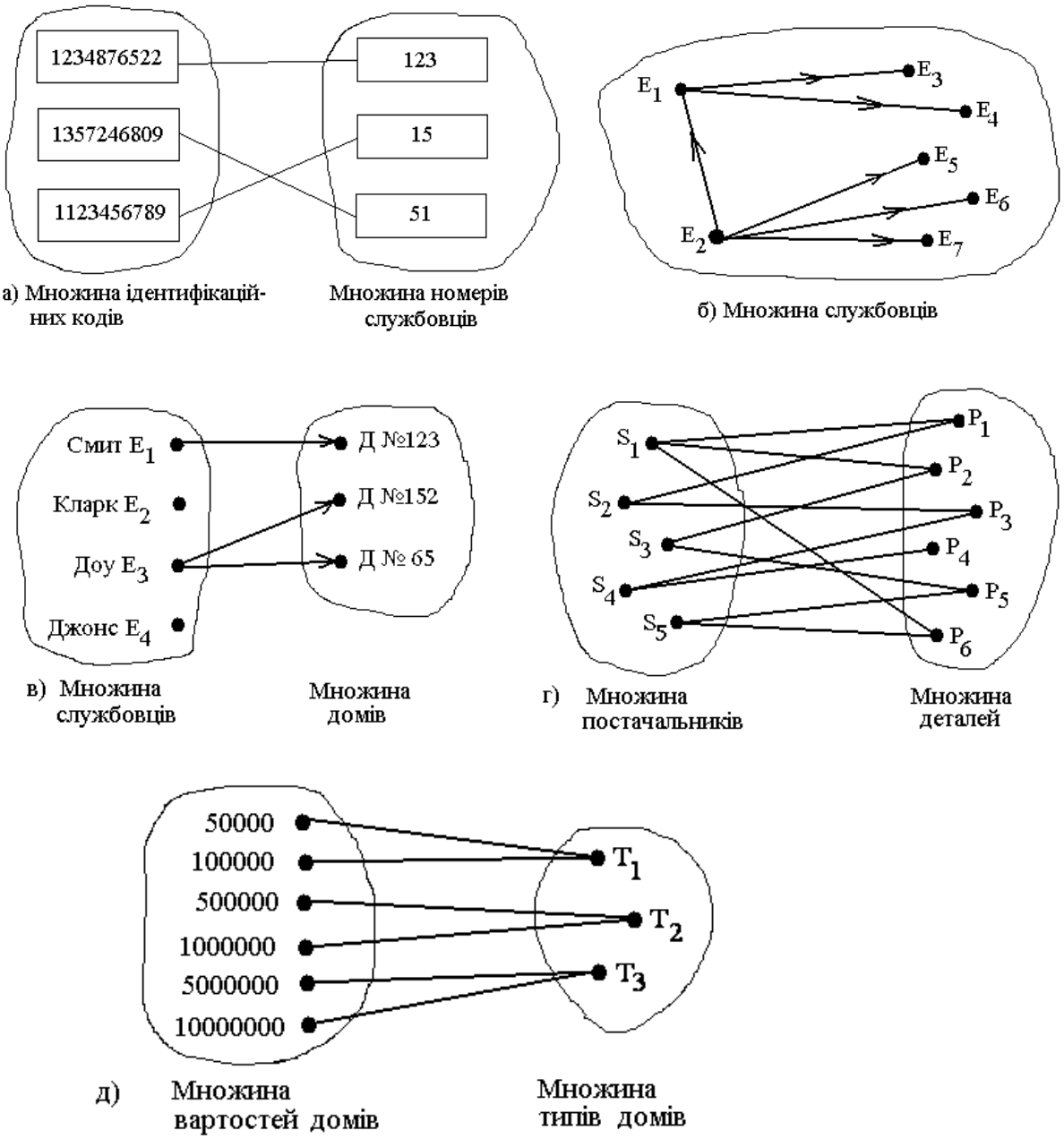


Рис. 6.4 Зв'язки між множинами сутностей: а – відповідність значення; б – зв'язок «керує»; в – зв'язок «володіє»; г –- зв'язок «поставляє»; д – зв'язок між типами будинків і їхньою вартістю

Досліджуючи типи зв'язків на рис. 6.4, можна зробити наступні зауваження:

1. На рис. 6.4, *а* й *д* представлені зв'язки між множинами атрибутів, які називаються також *міжатрибутними* зв'язками. Ці зв'язки мають внутрісутнісний тип.
2. На рис. 6.4, б, в и г представлені зв'язки між сутностями (*міжсутнісні зв'язку*). Можливі зв'язки, що охоплюють більше

двох сутностей, як у прикладі «постачальники поставляють деталі для проектів, і проекти використовують деталі». Зв'язки між сутностями називаються також «асоціаціями». Особливий випадок зв'язків між сутностями представлений на рис. 7.7, б, де зображений приклад зв'язку «керує», що включає єдиний тип сутності, тобто зв'язок між елементами той самої множини сутностей.

3. На рис. 6.4, а наведений приклад відображення 1:1 між відповідними множинами. Це приклад взаємно однозначного відображення.
4. На рис. 6.4, у показано, що не всі службовці володіють будинками, а деякі власники можуть мати більше одного будинку.
5. З рис. 6.4, б видно, що деякі службовці можуть керувати іншими службовцями й що кожний службовець має керівника (у E_2 є підлеглий E_1, що також є керівником). Виключення становить президент компанії, що керує сам собою (або керується радою директорів!).
6. З рис. 6.4, м видно, що всі постачальники є «активними», тобто всі вони поставляють деталі. При цьому кожний постачальник може поставляти кілька деталей, а деякі деталі можуть поставлятися декількома постачальниками.
7. На рис. 6.4, д показано, що вартість будинку визначає його тип.

Тепер повернемося до властивостей зв'язків. Насамперед кілька зауважень про математичні *функції*.

Нехай D і R множини. Математична функція $f: D \to R$ є відображення множини D на множину R. Множина D називається областю визначення, а множина R — областю значень функції f.

Функція є підмножиною декартова добутку ($D \times R$) множин D і R, причому, якщо $[x,y]$ і $[x,z]$ упорядковані пари в f, тоді y и z повинні бути рівні один одному, тобто елемент із D завжди відображається в єдину точку в R. Функція f називається *всюди визначеною* або *повною,* якщо $\{x \mid [x,y] \in f\} = D$, тобто кожному елементу з D відповідає деякий елемент в R. Функція f називається *часткової,* якщо f визначена на деякій підмножині D. Якщо досліджувати зв'язки на рис. 6.4, то можна зробити наступні виводи:

На рис. 6.4,*а* зв'язок між атрибутами ІДЕНТИФІКАЦІЙНИЙ_КОД і НОМЕР_СЛУЖБОВЦЯ є функціональним й усюди визначеним. Дана

властивість виконується для обох напрямках зв'язку. Таким чином, кожне значення першого атрибута відображається на єдине значення другого, і навпаки. Це властивість відображення 1:1.

На рис. 6.4, *в* властивістю функціональності володіє тільки відображення «керований», зворотне до відображення «керує», тому що відображення множини службовців на множину керівників ставиться до типу M : 1.

На рис. 7.7, *д* показане всюди визначене функціональне відображення вартості будинків на їхні типи.

Для всіх функціональних зв'язків справедливо, що атрибут (або сутність), що є областю визначення, однозначно визначає атрибут (або сутність) області значень. (Наприклад, «якщо ви назвете вартість будинку, те можна однозначно визначити його тип».) Говорять, що атрибут області визначення визначає атрибут області значень, або інакше — що останній залежить від першого. Це приводить до поняття функціональної залежності в теорії баз даних.

6.3 Функціональні залежності

Функціональні залежності відіграють велику роль у проектуванні баз даних. Армстронг виділив деякі властивості функціональних залежностей і сформулював їх у вигляді аксіом. Ці аксіоми називаються також правилами виводу, тому що, використовуючи їх, можна вивести або одержати з відомих функціональних залежностей ряд інших. Це важливо при проектуванні й дослідженні баз даних. Були сформульовані наступні правила виводу (символ "→" читається «визначає», символ "⊆" читається «є підмножиною»):

Основні правила

1. Рефлексівність. Нехай задана множина X и $Y \subseteq X$, тоді $X \to X$ та $X \to Y$. Це тривіальні функціональні залежності, що означають, що множина визначає будь-яка свою підмножину.

2. Транзитивність. Якщо $X \to Y$ й $Y \to Z$, то $X \to Z$.

3. Додатковість. Якщо $X \to Y$ і $X \subseteq W$, то $W \to Y$.

Наслідки з основних правил

1. Аддитивність (об'єднання). Якщо $X \to Y$ і $W \to Z$, то $X,W \to Y,Z$, або якщо $X \to Y$ і $X \to Z$, тоді $X \to Y,Z$.

2. Проективность (декомпозиція). Якщо $X \to Y,Z$, тоді $X \to Y$ і $X \to Z$.

3. Псевдотранзитивность. Якщо $X \to Y$ і $Y,W \to Z$, то $X,W \to Z$.

Приклад

Чому $A,E \rightarrow D$, якщо $A \rightarrow B$ й $B \rightarrow C,D$? (Передбачається, що E не є підмножиною A)

Рішення

1. На підставі правила транзитивності $A \rightarrow C,D$.

2. Внаслідок проективности $A \rightarrow D$.

3. На підставі додатковості $A,E \rightarrow D$.

Функціональні залежності виражають семантику (тобто сенс) бази даних. У будь-який момент безпосередньо доступна тільки семантика, виражена заданими функціональними залежностями. Однак за допомогою правил виводу можна одержати додаткову інформацію або знання про базу даних, які не сформульовані явно й не очевидні з доступної інформації.

Багатозначні залежності. Дотепер мова йшла лише про функціональні залежності. У відношення існують і інші залежності. Одним з видів залежностей є багатозначні залежності даного атрибута `B` від іншого атрибута `A` в відношенні `R`, що містить і інші атрибути. Говорять, що `A` багатозначно визначає `B` у `R` (або що `B` багатозначне залежить від `A`), позначаючи зазначену залежність $A \twoheadrightarrow B$, якщо кожному значенню `A` відповідає множина (можливо, порожня) значень `B`, ніяк не пов'язаних з іншими атрибутами `R`.

Аксіомы (правила виводу) для багатозначних залежностей. Введення багатозначних залежностей приводить до розширення розглянутої вище множини правил виводу. Припустимо, що X, Y і Z є атрибутами відношення R, а U позначає множину всіх атрибутів R. Двома найбільш важливими правилами для багатозначних залежностей є наступні:

1. **Доповнення**. Якщо $X \twoheadrightarrow Y$, тоді $X \twoheadrightarrow U-X-Y$. Це правило не має аналога для функціональних залежностей.

2. **Транзитивність**. Якщо $X \twoheadrightarrow Y$ і $Y \twoheadrightarrow Z$, то $X \twoheadrightarrow Z-Y$. Це більш обмежений варіант транзитивності в порівнянні із правилом для функціональних залежностей.

7 ТЕОРІЯ НОРМАЛЬНИХ ФОРМ

Забезпечення відсутності втрат і збереження залежностей при декомпозиції вимагає знання всіх можливих функціональних залежностей, наявних у даній схемі. Спочатку відома лише їхня підмножина, але можна одержати всі інші, користуючись розглянутими вище правилами виводу функціональних залежностей.

Атрибут, що входить у ключ, називається *первинним;* у противному випадку він називається *непервинним.* Функціональна залежність $A \rightarrow B$ називається повною *функціональною залежністю,* якщо B залежить від всієї групи атрибутів A, а не від її частини (підмножини). Наприклад, якщо $A = A_1, A_2, \ldots, A_k$ і $A_1, A_2 \rightarrow B$, тоді функціональна залежність B від A неповна.

Нижче ми розглянемо нормальні форми від першої до п'ятої, включаючи нормальну форму Бойса-Кодда. Для позначення нормальних форм використовуються скорочення 1НФ, 2НФ, ЗНФ, НФБК, 4НФ, 5НФ. Перша (1НФ), друга (2НФ) і третя (ЗНФ) нормальні форми обмежують залежність непервнинних атрибутів від ключів. Нормальна форма Бойса-Кодда (НФБК) обмежує також залежність первинних атрибутів. Четверта нормальна форма (4НФ) формулює обмеження на види багатозначних залежностей, які обговорюються нижче. П'ята нормальна форма (5НФ) уводить інші типи залежностей, які називаються залежностями з'єднання.

Рівень нормалізації відношення залежить від його семантики й не може бути однозначно визначений з даних, що містяться в поточний момент у базі даних. Це означає, що семантика повинна бути задана за допомогою функціональних залежностей. Як правило, при проектуванні бази даних потрібно, щоб база даних була принаймні в третій нормальній формі або НФБК.

Як видно з наведених нижче прикладів, при грамотному проектуванні й виборі коректних відношень, які є відображенням якоїсь конкретної сутності, а не їхньої сукупності, багатозначні залежності в таких відношеннях практично виключаються. Тому нормальні форми з багатозначними залежностями використовуються рідко. Коли між сутностями дійсно присутній зв'язок багато-до-багатьох, при проектуванні бази даних вводять сутності-зв'язки, наприклад, між сутностями ПОСТАЧАЛЬНИК і ДЕТАЛЬ буде сутність- зв'язка ПОСТАЧАЄ, між сутностями ЛІКАР і ПАЦІЄНТ буде сутність- зв'язка ЛІКУЄ.

7.1 Перша нормальна форма (1НФ)

Відношення перебуває в першій нормальній формі, якщо значення всіх його атрибутів прості (атомарні), тобто значення атрибута не повинне бути множиною або групою (масивом). Ненормалізованому відношенню

відповідає багаторівнева таблиця (ієрархія) на відміну від однорідної табличної структури нормалізованого відношення. Таким чином, без першої нормальної форми немає реляційної бази даних. У наш час розвиваються так звані постреляційні бази даних, для яких припустимими є дані типу масив.

Приклад. Є таблиця РЕЙСИ, у якій зберігається інформація про польоти літаків.

РЕЙСИ (НОМЕР, ПУНКТ_ВІДПРАВЛЕННЯ,
ПУНКТ_ПРИЗНАЧЕННЯ, РОЗКЛАД)

Тут у РОЗКЛАД записується інформація про день тижня (коли є рейс) і час вильоту:

РОЗКЛАД (ДЕНЬ, ЧАСУ-ВИЛЬОТУ)

НЕХАЙ є наступні дані про рейси:

TW101	Чикаго	Фінікс	пн 9.40
			вт 9.40
			пт 10.30
TW800	Фінікс	Нью-Йорк	пн 7.30
			чт 7.30
			пт 7.30

Для перетворення цього ненормалізованого відношення в 1НФ необхідно в складеному відношенні РЕЙСИ замінити відношення РОЗКЛАД відповідними атрибутами:

РЕЙС (НОМЕР, ПУНКТ_ВІДПРАВЛЕННЯ,
ПУНКТ_ПРИЗНАЧЕННЯ,
ДЕНЬ, ЧАСУ-ВИЛЬОТУ)

TW101	Чикаго	Фінікс	пн	9.40
TW101	Чикаго	Фінікс	вт	9.40
TW101	Чикаго	Фінікс	пт	10.30
TW800	Фінікс	Нью-Йорк	пн	7.30
TW800	Фінікс	Нью-Йорк	чт	7.30
TW800	Фінікс	Нью-Йорк	пт	7.30

7.2 Друга нормальна форма (2НФ)

Нехай є відношення ПОСТАВКИ, що містить дані про постачальників (які ідентифікуються номером П#), товарах, що вони поставляють, і цінах товарів:

ПОСТАВКИ(**П#, ТОВАР,** ЦІНА)

Припустимо, що постачальник може поставляти різні товари, а той самий товар можуть поставляти різні постачальники. Таким чином, ключ відношення (виділений напівжирним шрифтом) буде складатися з

атрибутів П# і ТОВАР. Нехай відомо, що ціна будь-якого товару зафіксована (тобто всі постачальники поставляють товар по однієї й тій же ціні). Семантика відношення включає наступні залежності:

П#,ТОВАР→ ЦІНА (за визначенням ключа)
ТОВАР→ЦІНА

Можна відзначити неповну функціональну залежність атрибута ЦІНА від ключа. Це приводить до наступних аномалій:

Аномалія включення. Якщо в постачальника з'являється новий товар, інформація про товар і його ціну не зможе зберігатися в базі даних доти, поки постачальник не почне поставляти його.

Аномалія видалення. Якщо поставки деякого товару припиняються, з бази даних доведеться видалити відомості про товар і його ціну, навіть якщо він є в наявності в постачальників.

Аномалія відновлення. При зміні ціни товару необхідний повний перегляд відношення з метою знайти всі поставки товару, щоб зміну ціни було відбито для всіх постачальників. Таким чином, зміна значення атрибута одного об'єкта тягне необхідність змін у декількох кортежах відношення: у противному випадку база даних виявиться неузгодженою. Причиною цих аномалій є неповна функціональна залежність атрибута ЦІНА від ключа, що обумовлено об'єднанням у відношенні ПОСТАВКИ двох семантичних фактів в одній структурі. Розкладання відношення ПОСТАВКИ на два відношення усуває неповну функціональну залежність.

Відношення перебуває *в другій нормальній формі*, якщо воно перебуває в 1НФ і кожний непервинний атрибут функціонально повно залежить від ключа (ключів).

Наступне розкладання приводить до відношення в 2НФ:

ПОСТАВКИ (**П#, ТОВАР)**
ЦІНА_ТОВАРУ **(ТОВАР**, ЦІНА)

Ціну товару конкретної поставки можна визначити шляхом з'єднання двох відношень по атрибуту ТОВАР. Зміна ціни товару викличе модифікацію лише одного кортежу другого відношення. Зберігання ціни товару у відношення ПОСТАВКИ буде необхідним, якщо у кожного постачальника своя ціна того ж самого товару.

7.3 Третя нормальна форма

Розглянемо транзитивну залежність наступного типу:

Якщо $A \rightarrow B$, B не визначає A (B не є ключем) і $B \rightarrow C$, то $A \rightarrow C$.

Нехай є відношення ЗБЕРІГАННЯ (ФІРМА, СКЛАД, ОБ'ЄМ), що містить інформацію про фірми, що одержують товари зі складів, і об'ємах цих складів. У відношенні є функціональні залежності:

ФІРМА→СКЛАД (фірма одержує товари тільки з одного складу)
СКЛАД→ ОБ'ЄМ

Аномалії. Якщо на даний момент відсутня фірма, що одержує товар зі складу, то в базу даних не можна ввести інформацію про об'єм складу (*аномалія включення*). Якщо остання фірма перестає одержувати товар зі складу, дані про склад і його об'єм не можна зберегти в базі даних (*аномалія видалення*). Якщо об'єм складу змінюється, необхідний перегляд усього відношення й зміна кортежів для всіх фірм, зв'язаних зі складом (*аномалія відновлення*). Транзитивна залежність (аналогічно неповної функціональної залежності в попередньому прикладі) викликана наявністю у відношенні двох семантичних різних фактів.

Перетворення відношення в ЗНФ усуває розглянуті аномалії.

Відношення перебуває *в ЗНФ*, якщо воно перебуває в 2НФ і в ньому відсутні транзитивні залежності непервинних атрибутів від ключа (ключів).

Наступне розкладання приводить до відношень у ЗНФ:

ЗБЕРІГАННЯ (**ФІРМА**,СКЛАД)
С_ ОБ'ЄМ (**СКЛАД**, ОБ'ЄМ)

7.4 Нормальна форма Бойса – Кодда (НФБК)

Нехай є відношення ПРОЕКТ (**Д#, ПР#,** П#), що відбиває використання в проектах деталей, що поставляють постачальниками. У проекті використовується кілька деталей, але кожна деталь проекту поставляється тільки одним постачальником. Кожний постачальник обслуговує тільки один проект, але проекти можуть забезпечуватися декількома постачальниками (різних деталей). Деталі, проекти, постачальники ідентифікуються відповідними номерами Д#, ПР#, П#. У відношенні присутні наступні функціональні залежності:

Д #, П Р # → П # (по визначенню ключа)
П# → ПР#

Розглянуте відношення перебуває в ЗНФ, тому що в ньому відсутні неповні функціональні залежності й транзитивні залежності непервинних атрибутів від ключів; при цьому, однак, спостерігаються наступні аномалії.

Аномалії. Факт поставки постачальником деталей для проекту не може бути занесений у базу даних доти, поки в проекті дійсно не почнуть використовутись ці деталі (*аномалія включення*). Якщо останній з типів деталей, що поставляються постачальником для проекту, використаний, дані про постачальника будуть також вилучені з бази даних (*аномалія видалення*). Якщо змінюється постачальник деякого типу деталей для проекту, необхідний перегляд відношення для зміни всіх кортежів, що містять ці деталі (*аномалія відновлення*).

Розкладання вихідного відношення на відношення в НФБК усуває перераховані аномалії.

Відношення перебуває *в НФБК*, якщо воно перебуває в ЗНФ і в ньому відсутні залежності первинних атрибутів від непервинних.

Є еквівалентне визначення НФБК:

Відношення перебуває *в НФБК*, якщо всі детермінанти (тобто домени функціональних залежностей) є можливими ключами.

Для цього необхідно усунути в даному відношенні залежність П# → ПР#. Наступне розкладання приводить до відношень у НФБК:

ПРОЕКТ_ДЕТАЛЬ (**Д#,ПР#**)
ПОСТАВКИ (**П#**,ПР#)

Деякі автори розглядають нормальні форми більш високого порядку: четверту та п’яту, – які формулюються для багатозначних залежностей.

Але проблеми з багатозначними залежностями виникають у проектувальників БД, які погано представляють собі предметну область, для якої вони проектують базу даних. В цьому випадку можлива ситуація, коли в одній таблиці намагаються зберігати дані, не пов’язані між собою. Наприклад, в таблиці, що описує викладачів університету, зберігати дані про дисципліні, які веде викладач, а також імена та дати народження дітей викладача. Зрозуміло, що ці дані ніяк не пов’язані між собою і повинні зберігатись у двох окремих таблицях.

ПЕРЕЛІК ПОСИЛАНЬ

1. American National Standards Institute (1975). ANSI/X3/SPARC Study Group on Data Base Management Systems. Interim Report, FDT. *ACM SIGMOD Bulletin,* 7(2).
2. Childs D.L. (1968). Feasibility of a set-theoretical data structure. In Proc. Fall Joint Computer Conference, 557-564.
3. Codd E.F. (1970). A relational model of data for large shared data banks. Comm. ACM, 13(6), 377-387.
4. Codd E.F. (1972a). Relational completeness of data base sublanguages. In Data Base Systems, Courant Comput. Set. Symp 6th (R. Rustin, ed.), 65-98. Englewood Cliffs, NJ: Prentice-Hall.
5. Codd E.F. (1982). The 1981 ACM Turing Award Lecture: Relational database: A practical foundation for productivity. Comm. ACM, 25(2), 109-117.
6. Codd E.F. (1979). Extending the data base relational model to capture more meaning. ACM Trans. Database Systems, 4(4), 397-434.
7. Codd E.F. (1986). Missing information (applicable and inapplicable) in relational databases. ACM SIGMOD Record, 15(4).
8. Codd E.F. (1990). The Relational Model for Database Management Version 2. Reading, MA: Addison-Wesley.
9. Конноли, Томас, Бегг, Каролин. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. 3-е издание.: Пер. с англ. – М.: Издательский дом "Вильям", 2003. – 1440 с.: ил..
10. Когаловский М.Р. Абстракции и модели в системах баз данных // М.: Журнал СУБД. – Издательский дом «Открытые системы». – №4-5, 1998. – С.56-61.
11. Пасічник В.В., Шаховська Н.Б. Сховища даних: Навчальний посібник. – Львів: «Магнолія 2006», 2008. – 496 с.
12. Ульман Дж.Д. Введение в системы баз данных / Джеффри Д. Ульман, Дженнифер Уидом [пер. с англ.]. – Издательство «Лори», 2000. – 374 с.
13. Дейт К. Дж. Введение в системы баз даннях, 6-е издание: Пер. с англ. – К.; М.; СПб.: Издательский дом «Вильямс», 2000. – 848 с.