Козловская Валентина Петровна

«Организация баз данных и знаний»

Конспект лекций, первая редакция

2008

Конспект лекцій з дисципліни «Організація баз даних та знань» призначений для студентів ІІІ курсу денної та заочної форми навчання, що навчаються за напрямком «Комп’ютерні науки», 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 керування базами даних (СКБД) – спеціалізованих програмних засобів, призначених для організації й ведення БД.

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. Бази даних стали основою інформаційних систем і в корені змінили методи роботи багатьох організацій. 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 відомого “кризи програмного забезпечення”, або так званої “депресії програмного забезпечення”, наслідки якої не усунуті й понині.

Тому метою даного курсу в сукупності з курсом “Системний аналіз і проектування інформаційних систем” 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 саме вона найбільше повно розглядається в курсі “Системний аналіз і проектування інформаційних систем”. 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 керування базами даних. Для придбання навичок роботи з конкретної СКБД у даному курсі розглядається робота з локальною СКБД FoxPro (роботі із клієнт-серверними СКБД присвячений наступний курс - “Розробка прикладних систем баз даних”).

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 керування базами знань (СКБЗ), 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 СКБД є необхідною складовою частиною інструментарію керування даними в СБЗ. Деякі аспекти роботи систем керування базами знань також розглядаються в даному курсі.

1 Введення в бази даних

1.1 Основні поняття й терміни теорії баз даних і знань

Основні терміни, які використовуються в теорії баз даних і знань:

база даних (БД) — іменована сукупність даних, що відображає стан об'єктів і їхні відносини в розглянутій предметній області;

система керування базами даних (СКБД) — сукупність язикових і програмних засобів, призначених для створення, ведення й спільного застосування БД багатьма користувачами;

система бази даних (система БД або СБД) — СКБД із наповненою базою даних, яка керується її засобами;

банк даних (БнД) — заснована на технології БД система програмних, язикових, організаційних і технічних засобів, призначених для централізованого накопичення й колективного використання даних;

інформаційна система (ІС) — система, що реалізує автоматизований збір, обробку й маніпулювання даними й включає технічні засоби обробки даних, програмне забезпечення й відповідний персонал.

1.1.1 Визначення

Фізичні дані. Це дані, що зберігаються в пам'яті ЕОМ. При великому об'ємі даних використовується вторинна пам'ять на магнітних дисках, оптичних дисках і т.д.

Логічне представлення даних. Відповідає представленню фізичних даних користувачем. Розходження між фізичним і відповідним логічним представленням даних полягає в тому, що останнє відображає деякі важливі взаємозв'язки між елементами фізичних даних. Наприклад, у файлі можуть міститися деякі множини імен і грошових сум. Логічне представлення або інтерпретація цих даних може бути наступної: імена ідентифікують службовців, а грошові суми відповідають їхнім зарплатам.

Незалежність даних. Сучасні СКБД повинні забезпечувати незалежність користувача від організації фізичних даних. Зміни фізичної організації й/або параметрів запам'ятовувальних пристроїв сприймаються СКБД і не впливають на користувача або точніше, на прикладну програму. Зміна представлення користувача й/або додавання нового представлення також підтримуються СКБД і не вимагають витрат на реорганізацію й зміну механізму доступу до файлів фізичних даних. Можливість функціонування при змінах по обидва боки (тобто з боку користувача й фізичних даних) називається незалежністю даних. Відсутність незалежності даних у традиційних файлових системах приводить до

жорсткості системи й високої вартості підтримки існуючих програм і розвитку нових додатків.

Хоча термін незалежність даних має широке розповсюдження, часто говорять також про *логічну незалежність даних* і *фізичну незалежность даних.* Ці терміни приблизно позначають незалежність відповідних сторін багаторівневої (багатошарової) архітектури СКБД.

СКБД. 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 керування базами даних* (СКБД) відноситься до набору засобів програмного забезпечення, необхідних для використання фактографічних баз даних.

1.2 Порівняння СКБД із традиційними файловими системами

Для файлових систем характерний тісний зв'язок між фізичними даними й прикладною програмою. У них відсутні майже всі гнучкі засоби, що пропонуються СКБД. Отже, більшість необхідних властивостей повинна підтримувати програма користувача. Інакше кажучи, крім логіки прикладної задачі користувач повинен забезпечувати засоби логічного представлення даних, інтерпретувати операції над цим представленням, перекладати їх у примітивні файлові операції й стежити за підтримкою файлів, що містять фізичні дані.

При такому режимі роботи досягти незалежності даних було б неможливо. Будь-які зміни безпосередньо впливали б на прикладну програму. Тісний зв'язок між додатком користувача й фізичними даними не дозволив би організувати спільне використання даних декількома користувачами, тому що вони можуть представляти ці дані й маніпулювати ними по-різному. Це у свою чергу приводить до необхідності дублювання даних у різних додатках.

У наш час клієнтам, менеджерам і іншим співробітникам з кожним днем потрібно усе більше й більше інформації. У деяких галузях діяльності існують навіть правові норми, що регламентують оформлення щомісячних, щоквартальних і річних звітів. Очевидно, що ручна картотека зовсім не підходить для виконання роботи подібного типу. Файлові системи були розроблені у відповідь на потребу в одержанні більш ефективних способів доступу до даних. Однак замість організації централізованого сховища всіх даних підприємства був використаний децентралізований підхід, при якому співробітники кожного відділу за допомогою фахівців з обробки даних (ОД) працюють зі своїми власними даними й зберігають їх у своєму відділі. Щоб проілюструвати основні

принципи такого підходу, розглянемо його на прикладі навчального проекту *DreamHome*.

1.2.1 Підхід, що використовується у файлових системах

Співробітники відділу реалізації відповідають за продаж і здачу в оренду об'єктів нерухомості. Наприклад, якщо клієнт звертається у відділ реалізації компанії із пропозицією здати в оренду приналежний йому об'єкт нерухомості, то йому потрібно заповнити форму, подібну представленої на рис. 1.1. У ній вказуються такі відомості про об'єкт нерухомості, як адреса й кількість кімнат, а також інформація про власника.

Крім того, співробітники відділу реалізації обробляють запити від потенційних орендарів, кожний з яких повинен заповнити форму, подібну до форми, представленої на рис. 1.2.

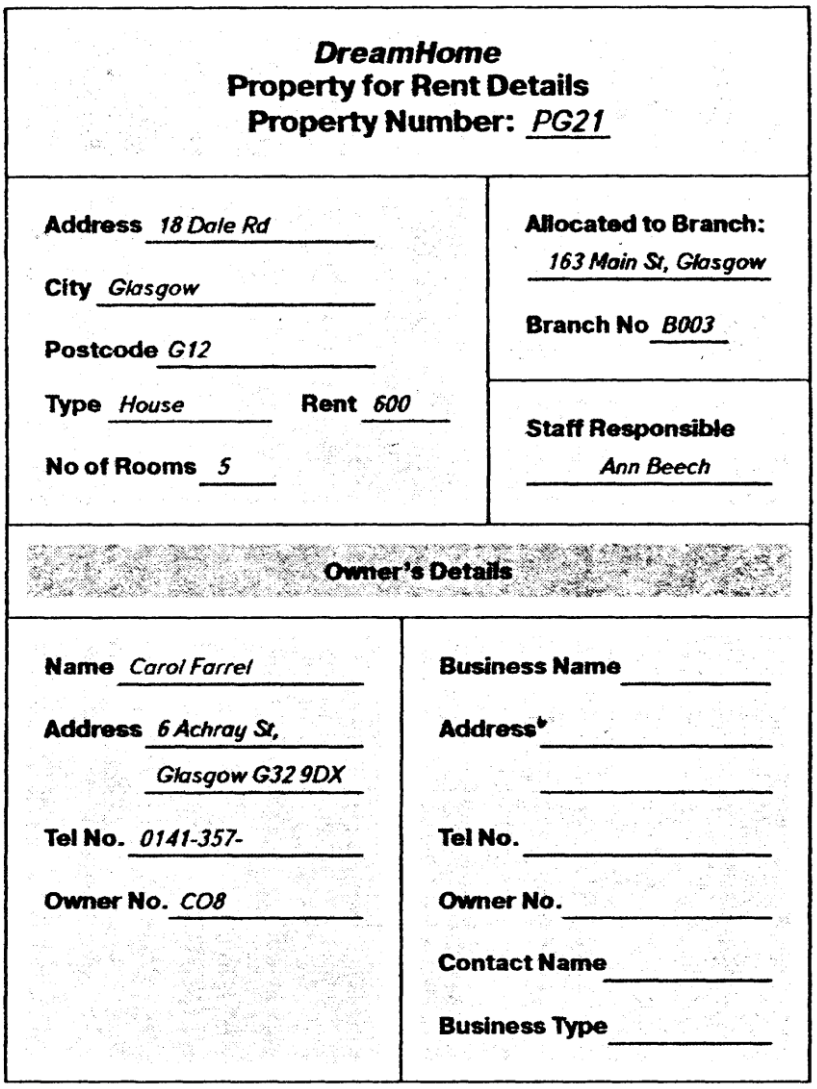


Рис. 1.1 Форми відділу реалізації: форма з відомостями про нерухомость, що здається у оренду

За допомогою співробітників відділу обробки даних (ОД) співробітники відділу реалізації створили інформаційну систему для керування даними про оренду нерухомості. Ця система складається із трьох показаних у табл. 1.1-1.3 файлів з даними про нерухомість (`PropertyForRent`), власника (`PrivateOwner`) і орендаря (`Client`). Для простоти викладу опустимо деталі, що відносяться до співробітників компанії, різним її відділенням і власникам нерухомості, що представляють собою юридичні особи.

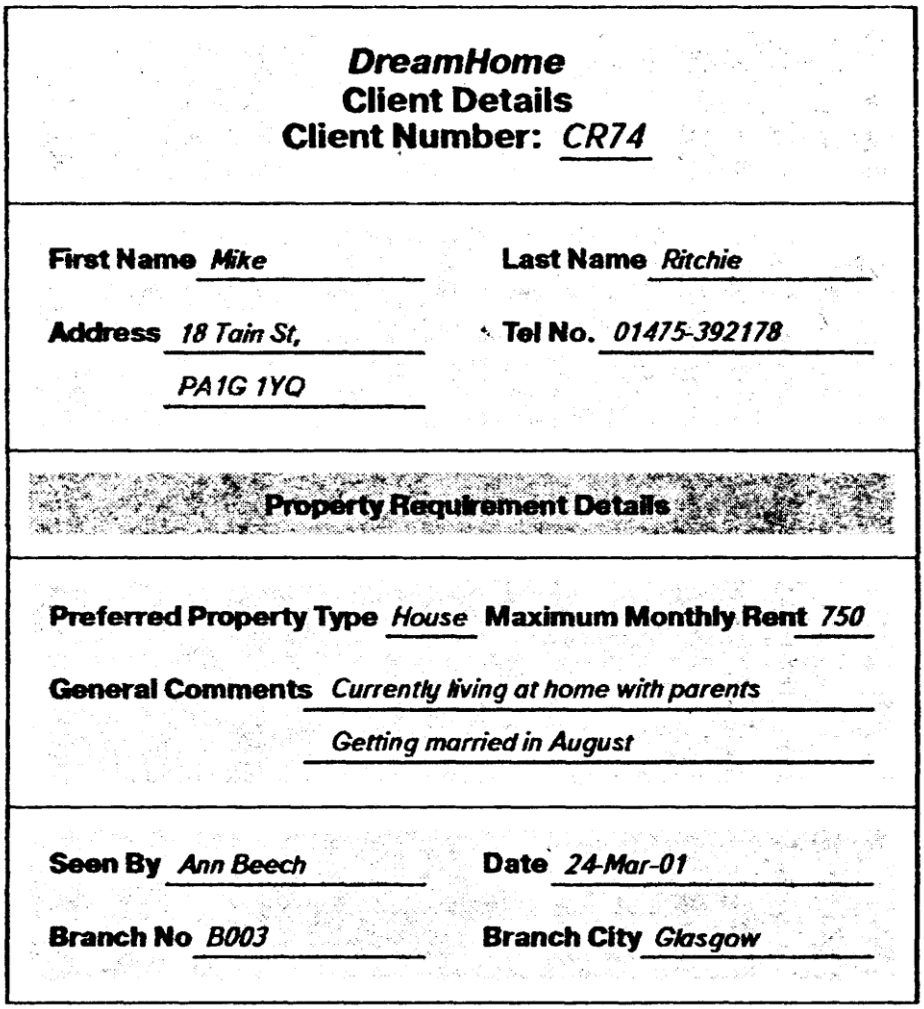


Рис. 1.2 Форми відділу реалізації: форма з інформацією про орендаря

У табл. 1.1 – 1.3 містяться наступні відомості для відділу реалізації: `propertyNo` (номер об'єкта), `street` (вулиця, на якій знаходиться об'єкт), `city` (місто), `postcode` (поштовий індекс), `type` (тип об'єкта – будинок, квартира), `rooms` (кількість кімнат), `rent` (орендна плата на місяць), `ownerNo` (номер власника), `fName` (ім'я), `lName` (прізвище), `address` (адреса), `telNo` (номер телефону), `clientNo` (номер клієнта – орендаря), `prefType` (найбільш бажаний тип орендного приміщення), `maxRent` (максимально припустимий розмір орендної плати).

Співробітники відділу контрактів відповідають за оформлення договорів про оренду нерухомості. Якщо клієнт згоден орендувати деякий об'єкт, що здається в оренду, то одним зі співробітників відділу реалізації заповнюється форма, показана на рис. 1.3. У ній вказуються всі необхідні відомості про орендаря й об'єкт нерухомості, що здається в оренду. Ця форма передається у відділ контрактів, співробітники якого присвоюють договору номер і вносять додаткові відомості про оплату й про тривалість оренди.

Таблиця 1.1 Інформація у файлі `PropertyForRent` відділу реалізації

PropertyNo	street	city	postcode	type	rooms	rent	ownerNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	C046
PL94	6 Argyll St	London	NW2	Flat	4	400	C087
PQ4	6 Lawrence St	Glasgow	G119QX	Flat	3	350	CO40
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	C093
PG21	18 Dale Rd	Glasgow	G12	House	5	600	C087
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	C093

Таблиця 1.2 Інформація у файлі `PrivateOwner` відділу реалізації

ownerNo	fName	lName	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	Murph	63 Well St, Glasgow G42	0141-943-1728
3093	Tony	Shaw	12 Park PI, Glasgow G4 OQR	0141-225-7025

Таблиця 1.3. Інформація у файлі `Client` відділу реалізації

clientNo	fName	lName	address	telNo	prefType	maxRent
CR76	John	Kay	56 High St, London SW1 4EH	0207-774-5632	Flat	425
CR56	Aline	Stewart	64 Fern Or, Glasgow G42 OBL	0141-848-1825	Flat	350
CR74	Mike	Ritchie	18 Tain St, Pal lYQ	01475-392178	House	750
CR62	Mary	Tregear	5 Tarbot Rd, Aberdeen AB9 3ST	01224-196720	Flat	600

За допомогою співробітників відділу ОД співробітники відділу контрактів створили для себе інформаційну систему обліку договорів оренди. Ця система складається із трьох файлів, представлених у табл. 1.4-1.6. Файли містять відомості про договори (Lease), об'єкти нерухомості (PropertyForRent) і орендарів (Client), які багато в чому подібні з даними в інформаційній системі відділу реалізації.

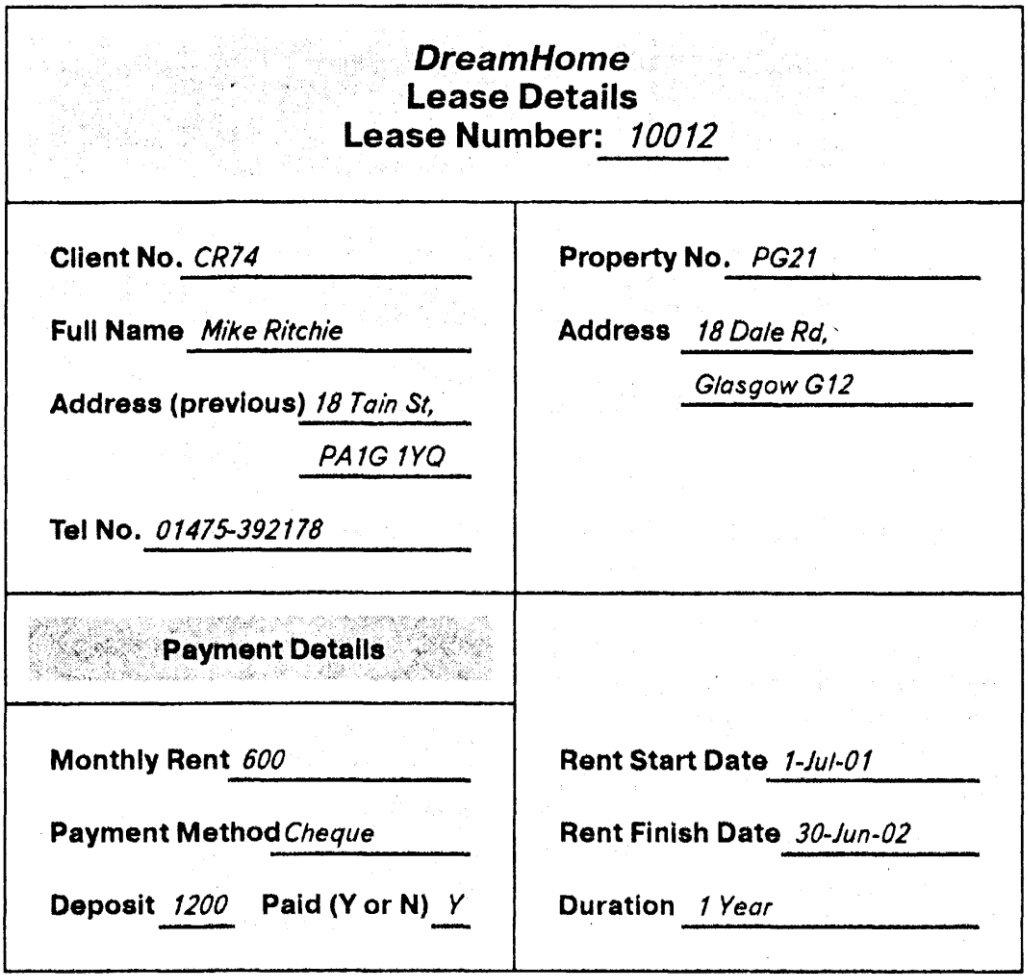


Рис. 1.3. Форма *Lease Details* — відомості про договор оренди

У табл.1.4 - 1.6 містяться наступні відомості для відділу контрактів, відсутні в табл.1.1 - 1.3: `leaseNo` (номер контракту), `paymentMethod` (спосіб оплати - готівка, чек, кредитна карта), `paid` (сплачено (Y), чи ні (N)), `rentStart` (дата початку оренди приміщення), `rentFinish` (дата кінця оренди приміщення), `duration` (термін оренди в місяцях) .

Таблиця 1.4. Інформація у файлі `Lease` відділу контрактів

leaseNo	propertyNo	clientNo	rent	paymentMethod	deposit	paid	rentStart	rentFinish	duration
10024	PA14	CR62	650	Visa	1300	Y	1-06-Ol	31-05-02	12
10075	PL94	CR76	460	Cash	800	N	l-08-01	31-01-02	6
10012	PG21	CR74	600	Cheque	1200	Y	l-07-01	30-06-02	12

Таблиця 1.5. Інформація у файлі `PropertyForRent` відділу контрактів

propertyNo	street	city	postcode	rent
PA14	16 Holhead	Aberdeen	AB7 5SU	650
PL94	6 Argyll St	London	NW2	400
PG21	18 Dale Rd	Glasgow	G12	600

Таблиця 1.6. Інформація у файлі `Client` відділу контрактів

clientNo	fName	lName	address	telNo
CR76	John	Kay	56 High St, London SW14EH	0171-774-5632
CR74	Mike	Ritchie	18 Tain St, PA1G 1YQ	01475-392178
CR62	Mary	Tregear	5 Tarbot Rd, Aberdeen AB9 3ST	01224-196720

У цілому ця ситуація схематично може бути представлена на рис. 1.4. На цій схемі показано, що кожний відділ звертається до своїх власних даних за допомогою спеціалізованих додатків. Набір додатків кожного відділу дозволяє вводити дані, працювати з файлами й генерувати деякий фіксований набір спеціалізованих звітів. Найважливішим є та обставина, що фізична структура й методи зберігання записів файлів з даними жорстко визначені в коді програм додатків.

Аналогічні приклади можна знайти й в інших відділах. Наприклад, у розрахунковому секторі бухгалтерії зберігаються відомості про зарплату кожного співробітника, які містяться у файлі наступного формату:

```
StaffSalary(staffNo, fName, lName, sex, salary,
            branchNo)
```

У відділі кадрів також зберігається інформація про персонал, але вона представлена у файлі іншого формату:

```
Staff(staffNo, fName, lName, position, sex, dateOfBirth,
      salary, branchNo)
```

Тут використовуються наступні описи співробітників компанії: `staffNo` (табельний номер співробітника), `fName`, `lName` (ім'я й

прізвище співробітника), `sex` (стать співробітника), `salary` (зарплата за рік), `branchNo` (номер відділення компанії, у якому працює співробітник), `position` (посада), `dateOfBirth` (дата народження).

Файли відділу реалізації

PropertyForRent (propertyNo, street, city, postcode, type, rooms, rent, ownerNo)
PrivateOwner (ownerNo, fName, lName, address, telNo)
Client (clientNo, fName, lName, address, telNo, prefType, maxRent)

Файли відділу контрактів

Lease (leaseNo, propertyNo, clientNo, rent, paymentMethod, deposit, paid, rentStart, rentFinish, duration)
PropertyForRent (propertyNo, street, city, postcode, rent)
Client (clientNo, fName, lName, address, telNo)

Рис. 1.4. Схема обробки даних у файловій системі

Цілком очевидно, що дуже велика кількість даних у відділах дублюється, і це дуже характерно для будь-яких файлових систем. Перш ніж почати обговорення недоліків цього підходу, було б корисно познайомитися з термінологією, яка використовується у файлових системах.

Файл є простим набором записів (record), які містять логічно зв'язані дані. Наприклад, файл `PropertyForRent`, вміст якого представлено в табл. 1.1, містить шість записів, по одному для кожного об'єкта нерухомості,який здається в оренду. Кожний запис містить логічно зв'язаний набір з одного або декількох полів (field), кожне з яких представляє деяку характеристику об'єкта, який ми моделюємо. З табл. 1.1 видно, що поля файлу `PropertyForRent` представляють характеристики об'єктів, що здаються в оренду, такі, як адреса, тип об'єкта й кількість кімнат.

1.2.2 Обмеження, властивим файловим системам

Такого короткого опису файлових систем цілком достатньо для того, щоб зрозуміти суть властивих їм обмежень, які перераховані в табл. 1.7.

Таблиця 1.7 Обмеження, властивим файловим системам

Обмеження
Розділення і ізоляція даних
Дублювання даних
Залежність від даних
Несумісність файлів
Фіксовані запити/швидке збільшення кількості додатків

Розділення і ізоляція даних

Коли дані ізольовані в окремих файлах, отримати доступ до них дуже важко. Наприклад, для створення списку всіх будинків, що відповідають вимогам потенційних орендарів, попередньо потрібно створити тимчасовий файл зі списком орендарів, що бажають орендувати нерухомість типу "будинок". Потім у файлі `PropertyForRent` потрібно здійснити пошук об'єктів нерухомості типу "будинок" з орендною платою нижче встановленого орендарем максимуму. Виконувати подібну обробку даних у файлових системах досить складно. Для витягу інформації, яка відповідає поставленим умовам, програміст повинен організувати синхронну обробку двох файлів. Труднощі істотно зростають, коли необхідно витягти дані більш ніж із двох файлів.

Дублювання даних

Через децентралізовану роботу з даними, яка проводиться в кожному відділі незалежно від інших відділів, у файловій системі фактично допускається безконтрольне дублювання даних, і це, у принципі, неминуче. Наприклад, на рис. 1.3 ясно видно, що у відділі реалізації й відділі контрактів дублюється інформація про об'єкти нерухомості й орендарях. Безконтрольне дублювання даних небажано по наступних причинах.

- Дублювання даних супроводжується неощадливою витратою ресурсів, оскільки на уведення надлишкових даних потрібно витрачати додатковий час і гроші.
- Більш того, для їхнього зберігання необхідно додаткове місце в зовнішній пам'яті, що пов'язане з додатковими накладними

витратами. У багатьох випадках дублювання даних можна уникнути за рахунок спільного використання файлів.

- Ще більш важливим є той факт, що дублювання даних може призвести до порушення їхньої цілісності. Інакше кажучи, дані в різних відділах можуть стати суперечливими. Наприклад, розглянемо описаний вище випадок дублювання даних у розрахунковому секторі бухгалтерії й відділі кадрів. Якщо співробітник переїде в інший будинок і зміну адреси буде зафіксовано тільки у відділі кадрів, то повідомлення про зарплату буде послане йому за старою, тобто помилковою, адресою. Більш серйозна проблема може виникнути, якщо якийсь співробітник одержить підвищення по службі з відповідним збільшенням заробітної плати. І знову ж, якщо ця зміна буде зафіксована тільки в інформації відділу кадрів, але не буде проведена у файлах розрахункового сектора, то співробітникові буде помилково нараховуватися колишня заробітна плата. При виникненні подібної помилки для її виправлення буде потрібно затратити додатковий час і кошти. Обидва ці приклади демонструють протиріччя, які можуть виникнути при дублюванні даних. Оскільки не існує автоматичного способу відновлення даних одночасно й у файлах відділу кадрів, і у файлах розрахункового сектора, неважко передбачити, що подібні протиріччя час від часу будуть неминуче виникати. Навіть якщо співробітники розрахункового сектора після одержання повідомлень про подібні зміни будуть негайно їх вносити, однаково існує ймовірність неправильного введення змінених даних.

Залежність від даних

Як уже згадувалося вище, фізична структура й спосіб зберігання записів файлів даних жорстко зафіксовані в коді додатків. Це значить, що змінити існуючу структуру даних досить складно. Наприклад, збільшення у файлі `PropertyForRent` довжини поля адреси з 40 до 41 символу здається зовсім незначною зміною його структури, але для втілення цієї зміни буде потрібно, як мінімум, створити одноразову програму спеціального призначення (тобто програму, що виконується тільки один раз), що перетворить уже існуючий файл `PropertyForRent` у новий формат. Вона повинна виконувати наступні дії:

- відкрити вихідний файл `PropertyForRent` для читання;
- відкрити тимчасовий файл із новою структурою запису;

- зчитати запис із вихідного файлу, перетворити дані в новий формат і записати їх у тимчасовий файл. Ці дії потрібно виконати для всіх записів вихідного файлу;
- видалити вихідний файл `PropertyForRent`;
- привласнити тимчасовому файлу ім'я `PropertyForRent`.

Крім цього, всі програми, що звертаються до файлу `PropertyForRent`, повинні бути змінені з метою відповідності новій структурі файлу. А таких програм може бути дуже багато. Отже, програміст повинен насамперед виявити всі програми, яким потрібна доробка, а потім їх перевірити і змінити. Зверніть увагу, що багато програм, яки підлягають зміненню, можуть звертатися до файлу `PropertyForRent`, але при цьому взагалі не використовувати поле адреси. Ясно, що виконання всіх цих дій вимагає великих витрат часу й може стати причиною появи помилок. Дана особливість файлових систем називається *залежністю програм від даних* (program-data dependence).

Несумісність форматів файлів

Оскільки структура файлів обумовлюється кодом додатків, вона також залежить від мови програмування цього додатка. Наприклад, структура файлу, створеного програмою мовою Pascal, може значно відрізнятися від структури файлу, створеного програмою мовою C/C++. Пряма несумісність таких файлів усклюднює процес їхньої спільної обробки.

Наприклад, припустимо, що співробітникам відділу контрактів потрібно знайти імена й адреси всіх власників, нерухомість яких у цей час здана в оренду. На жаль, у відділі контрактів немає відомостей про власників нерухомості, тому що вони зберігаються тільки у відділі реалізації. Однак у файлах відділу контрактів є поле `propertyNo` з номерами об'єктів нерухомості, які можна використати для пошуку відповідних номерів у файлі `PropertyForRent` відділу реалізації. Цей файл також містить поле `ownerNo` з номерами власників, які можна використати для пошуку відомостей про власників у файлі `PrivateOwner`. Припустимо, що програма відділу контрактів створена мовою Pascal, а програма відділу реалізації — мовою C. Тоді з метою пошуку відповідних номерів об'єктів нерухомості в полі `propertyNo` у двох файлах `PropertyForRent` програмістові буде потрібно створити програмне забезпечення, призначене для перетворення цих полів у деякий загальний формат. Не викликає сумніву, що цей процес може виявитися досить тривалим і дорогим.

Фіксовані запити/швидке збільшення кількості додатків

З погляду користувача можливості файлових систем набагато перевершують можливості ручних картотек. Відповідно зростають і вимоги до реалізації нових або модифікованих запитів. Однак файлові системи вимагають великих витрат праці програміста, оскільки всі необхідні запити й звіти повинні бути створені саме ним. У результаті події звичайно розвивалися по одному з наступних двох сценаріїв. По-перше, у багатьох організаціях типи запитів і звітів, що застосовувались, мали фіксовану форму, і не було ніяких інструментів створення незапланованих або довільних (ad hoc) запитів як до самим даних, так і до відомостей про те, які типи даних доступні.

По-друге, в інших організаціях спостерігалося швидке збільшення кількості файлів і додатків. В остаточному підсумку наступав момент, коли співробітники відділу обробки даних були просто не в змозі впоратися з усією роботою за допомогою наявних ресурсів. У цьому випадку навантаження на співробітників відділу настільки зростала, що неминуче наступав момент, коли програмне забезпечення було не в змозі адекватно відповідати запитам користувачів, ефективність його падала, а недостатність документування мала як наслідок додаткове ускладнення супроводу програм. При цьому часто ігнорувалися питання підтримки функціонування системи: не передбачалися заходи щодо забезпечення безпеки або цілісності даних; засоби відновлення у випадку збоїв апаратного або програмного забезпечення були вкрай обмежені або взагалі були відсутні. Доступ до файлів часто обмежувався вузьким колом користувачів, тобто не передбачалося їхнє спільне використання навіть співробітниками того самого відділу.

У кожному разі, подібна організація роботи із часом зживає себе, і потрібно шукати інші рішення.

1.3 Системи з використанням баз даних

Всі перераховані вище обмеження файлових систем є наслідком двох факторів.

1. Визначення даних міститься усередині додатків, а не зберігається окремо й незалежно від них.
2. Крім додатків не передбачено ніяких інших інструментів доступу до даних і їхній обробки.

Для підвищення ефективності роботи необхідно використати новий підхід, а саме *базу даних* (database) і *систему керування базами даних*, або СКБД (Database Management System - DBMS). У цьому розділі

представлене формальне визначення цих термінів. Компоненти середовища СКБД розглянуті в наступному розділі.

1.3.1 База 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.

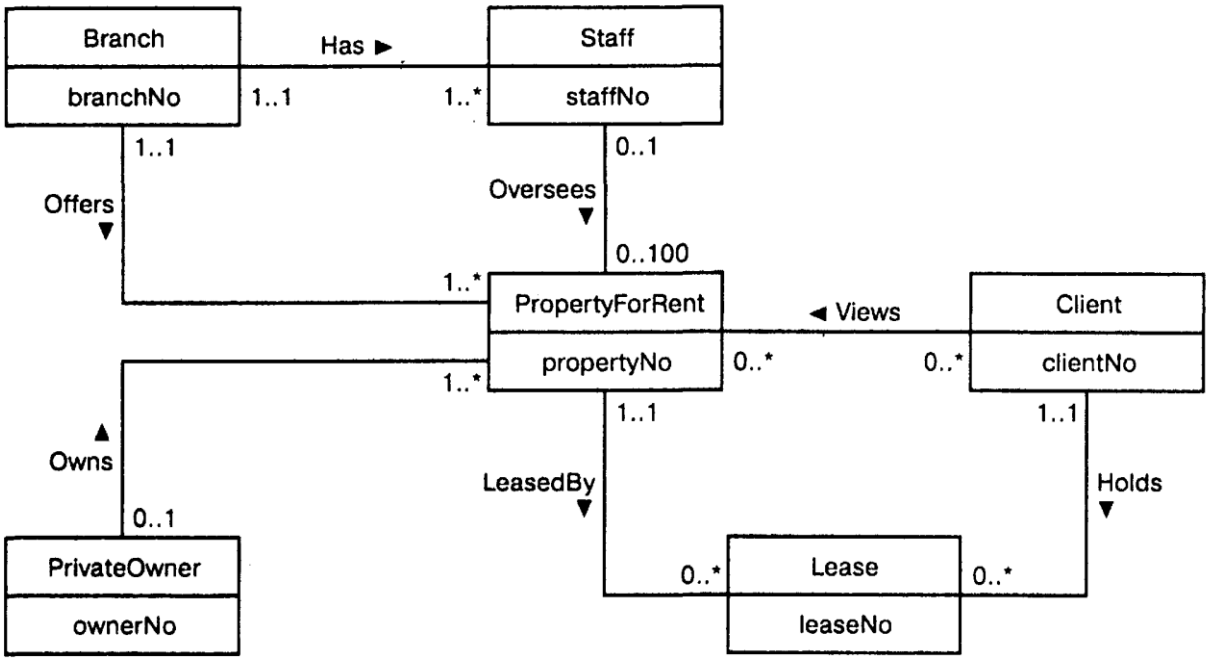


Рис. 1.5 Приклад діаграми "сутність-зв'язок"

Наприклад, на рис. 1.5 показана так звана діаграма "сутність-зв'язок", або ER-діаграма (Entity-Relationship - ER), для деякої частини навчального проекту *DreamHome*. 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 “Системний аналіз і проектування інформаційних систем”.

1.4 Система керування базами даних - СКБД

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

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

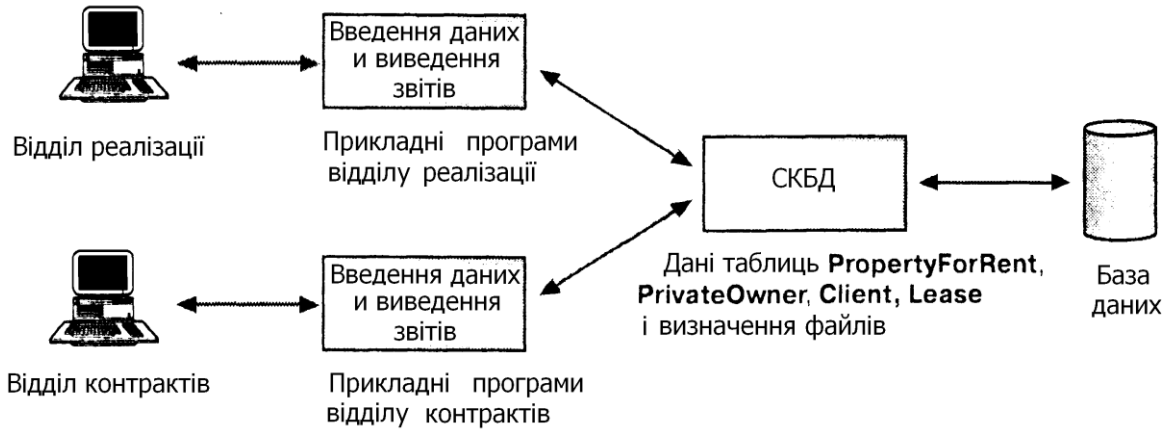
- Дозволяє створити базу даних, що звичайно здійснюється за допомогою мови визначення даних (МВД або DDL — Data Definition Language). Мова DDL надає користувачам засоби вказівки типу даних і їхньої структури, а також засоби завдання обмежень для інформації, яка зберігається в базі даних.
- Дозволяє вставляти, 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 СКБД (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 СКБД 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.

На рис. 1.6 показаний приклад реалізації підходу із застосуванням бази даних замість розглянутого раніше варіанта з використанням файлової системи (див. рис. 1.3).




```
PropertyForRent (propertyNo, street, city, postcode, type,
                 rooms, rent, ownerNo)
PrivateOwner (ownerNo, fName, lName, address, telNo)
Client (clientNo, fName, lName, address, telNo, prefType,
        maxRent)
Lease (leaseNo, propertyNo, clientNo, paymentMethod,
       deposit, paid, rentStart, rentFinish)
```

Рис. 1.6 Схема обробки даних за допомогою СКБД

У новому варіанті відділ реалізації й відділ контрактів використовують власні додатки для доступу до загальної бази даних, організованої за допомогою СКБД. Набір додатків кожного відділу забезпечує уведення й коректування даних, а також генерацію необхідних звітів. Але на відміну від варіанта з файловою системою фізична структура й спосіб зберігання даних контролюються за допомогою СКБД.

1.4.1 Стандарти СКБД

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

1. **Незалежність даних.**
2. **Універсальність**. СКБД повинна мати потужні засоби підтримки концептуальної моделі даних для відображення логічних представлень користувачів.
3. **Сумісність**. СКБД 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**. СКБД повинна забезпечувати захист від несанкціонованого доступу.
6. **Цілісність даних**. СКБД повинна запобігати порушення бази даних користувачами.
7. **Керування одночасною (паралельною) роботою**. СКБД 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) СКБД 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) СКБД повинна підтримувати як централізовані, так і розподілені бази даних. У наш час широке застосування находить організація обчислювальних мереж і розподілена обробка даних. СКБД для територіально розосередженої організації, у якій важлива локальна автономність даних, повинна бути системою керування

розподіленими базами даних (СКРБД). 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.

1.4.2 Переваги й недоліки СКБД

СКБД мають як багатообіцяючи потенційні переваги, так і недоліки, 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

Як уже говорилося в розділі 1.1, 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. Причини введення в базу контрольованої надмірності даних стануть зрозумілі при читанні наступних розділів.

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, те така система зможе стежити за тим, щоб копії не суперечили одна одній. На жаль, у багатьох сучасних СКБД 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 СКБД, 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 СКБД застосовувати їх. Роль і завдання АБД розглядаються далі в розділі 2.3.

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 СКБД 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). Наприклад, АБД компанії із продажу й оренді нерухомості може бути надане право доступу до всіх даних у базі даних, менеджерові відділення компанії - до всіх даних, які належать до його відділення, а інспекторові відділу реалізації - лише до всіх даних про нерухомість, у результаті чого він не буде мати доступу до конфіденційних даних, таких, як зарплата співробітників.

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. Наприклад, вона може складатися з одного потужного комп'ютера або мережі з невеликих комп'ютерів.

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 СКБД 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. Наприклад, менеджер відділення компанії може отримати перелік всіх квартир, які здаються в оренду з місячною орендною платою понад 1000 грн., якщо введе на своєму терміналі наступний оператор SQL:

```
SELECT  *
FROM PropertyForRent
WHERE type='Flat' AND rent>1000;
```

Докладніше запити мовою SQL будуть розглянуті нижче. Тут же потрібно відзначити, що навіть для такого простого запиту менеджер повинен знати хоча б основи мови SQL. При правильному проектуванні програмного забезпечення для бази даних (системи бази даних) програмісти повинні забезпечити користувачів практично всіма можливими запитами. Тобто, у цьому випадку, у клієнтському додатку менеджера (структура СКБД "клієнт/сервер" розглядається в розділі 2.9) повинен бути пункт меню "Пропозиції по оренді", при виконанні якого менеджерові потрібно буде тільки вибрати зі списку можливий об'єкт для оренди (квартира, будинок, офіс і т.д.), указати при необхідності верхню й/або нижню межу орендної плати, а також, можливо, указати деякі необов'язкові параметри для вибірки даних, наприклад, район міста.

Поліпшення показників продуктивності

У СКБД передбачено багато стандартних функцій, які програміст звичайно повинен самостійно реалізувати в додатках для файлових систем. Функції СКБД розглядаються в розділі 2.7. На базовому рівні СКБД 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.

У багатьох СКБД передбачене також середовище розробки четвертого покоління з інструментами, що спрощують створення додатків баз даних. Результатом є підвищення продуктивності роботи програмістів і скорочення часу розробки нових додатків (з відповідною економією засобів).

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. Для зміни структури даних (наприклад, для збільшення довжини поля з адресою з 40 символів до 41 символу) або для зміни способу зберігання даних на диску може знадобитися істотно перетворити всі програми, на які ці зміни здатні вплинути.

У СКБД підхід іншої: опис даних відділені від додатків, а тому додатки захищені від змін в описах даних. Ця особливість називається *незалежністю від даних*. Докладніше мова про неї піде в розділі 2.3.5. 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 СКБД передбачена можливість паралельного доступу до бази даних і гарантується відсутність подібних проблем. Більше докладно керування паралельним доступом до даних обговорюється в главі 13.

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 СКБД передбачені засоби зниження ймовірності втрат інформації при виникненні різних збоїв.

Недоліки СКБД

Недоліки підходу, пов'язаного із застосуванням баз даних, перераховані нижче:

- Складність
- Розмір
- Вартість СКБД
- 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 СКБД, супроводжується значним ускладненням програмного забезпечення СКБД. Щоб скористатися всіма перевагами СКБД, проектувальники й розроблювачі баз даних, адміністратори даних і адміністратори баз даних, а також кінцеві користувачі повинні добре розуміти функціональні можливості СКБД. 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 СКБД 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 СКБД

Залежно від наявного обчислювального середовища й необхідних функціональних можливостей вартість СКБД може змінюватися в дуже широких межах. Наприклад, локальна СКБД для персонального комп'ютера може коштувати близько 100 доларів. Однак велика СКБД 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 СКБД 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 СКБД. Придбання іншого додаткового апаратного забезпечення призведе до подальшого зростання витрат.

Витрати на перетворення

У деяких ситуаціях вартість СКБД і додаткового апаратного забезпечення може виявитися несуттєвої в порівнянні з вартістю перетворення існуючих додатків для роботи з новою СКБД 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.

Все це є однією з основних причин, через яку деякі організації залишаються прихильниками колишніх систем і не бажають переходити до більш сучасних технологій керування базами даних. Термін “традиційна система” іноді використовується для позначення застарілих і, як правило, не найкращих систем.

Продуктивність

Звичайно файлова система створюється для деяких спеціалізованих додатків, наприклад для оформлення рахунків, а тому її продуктивність може бути досить висока. Однак СКБД 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 СКБД, вихід з ладу одного з її компонентів може привести до повного припинення всієї роботи організації.

2 СЕРЕДОВИЩЕ БАЗИ ДАНИХ

Основна мета *системи керування базами даних* (далі - просто СКБД) полягає в тім, щоб запропонувати користувачеві абстрактне представлення даних, сховавши конкретні особливості зберігання й керування ними. Отже, відправною точкою при проектуванні бази даних повинен бути абстрактний і загальний опис інформаційних потреб організації, які повинні знайти своє відображення в базі даних, що створюється. У цій і наступній главах поняття "організація" використовується в широкому сенсі, позначаючи або деяку організацію в цілому, або тільки її частину. Наприклад, у навчальному проекті *DreamHome* становить інтерес моделювання наступних понять:

- *сутності* "реального світу", такі як `Staff` (Працівник), `PropertyForRent` (Орендована власність), `PrivateOwner` (Власник власності) і `Client` (Клієнт);
- *атрибути*, що описують властивості або якості кожної сутності (наприклад, сутність `Staff` має атрибути `name` (Ім'я), `position` (Посада) і `salary` (Зарплата));
- *зв'язки* між цими сутностями (наприклад, `Staff` *Manages* `PropertyForRent`).

Більш того, оскільки база даних є загальним ресурсом, те кожному користувачеві може знадобитися своє, відмінне від інших представлення про характеристики інформації, що зберігає в базі даних. Для задоволення цих потреб архітектура більшості сучасних комерційних СКБД у тім або іншому ступені будується на базі так званої архітектури ANSI-SPARC. У цьому розділі ми обговоримо різні архітектурні й функціональні характеристики СКБД.

2.1 Компоненти середовища СКБД

Як показано на рис. 2.1, у середовищі СКБД можна виділити наступні п'ять основних компонентів: апаратне й програмне забезпечення, дані, процедури й користувачів.

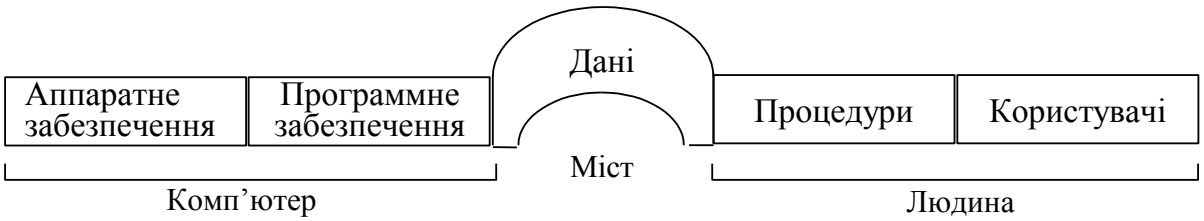


Рис. 2.1. Середовище СКБД

Апаратне забезпечення

Для роботи СКБД і додатків необхідно деяке апаратне забезпечення. Воно може варіюватися в дуже широких межах - від єдиного персонального комп'ютера або одного мейнфрейма до мережі з багатьох комп'ютерів. Апаратне забезпечення, що використовується, залежить від вимог даної організації й типу СКБД. Одні СКБД призначені для роботи тільки з конкретними типами операційних систем або обладнання, інші можуть працювати із широким колом апаратного забезпечення й різних операційних систем.

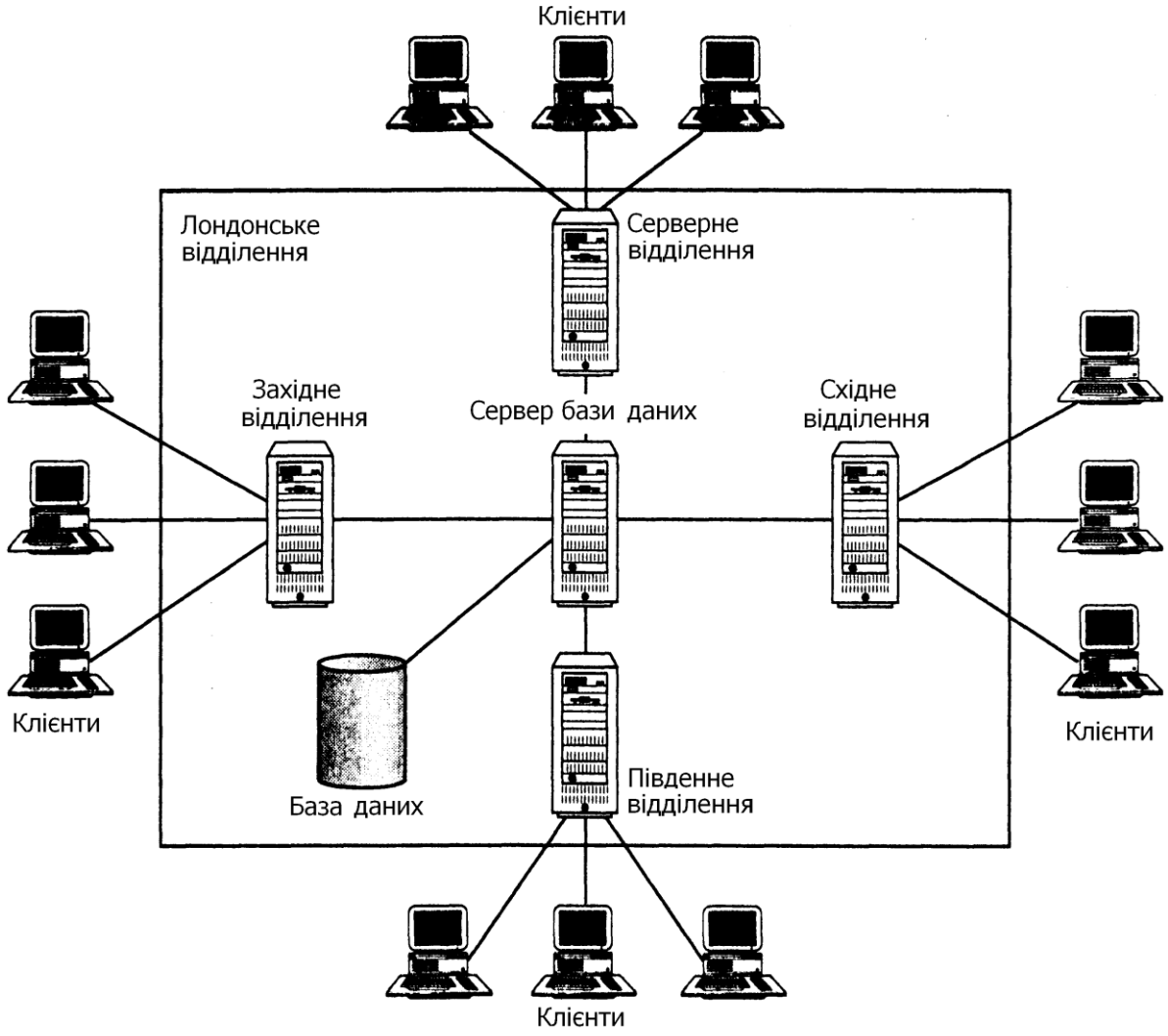


Рис. 2.2 Конфігурація апаратного забезпечення для навчального проекту DreamHome

Для роботи СКБД звичайно потрібно деякий мінімум оперативної й дискової пам'яті, але такої мінімальної конфігурації може виявитися зовсім недостатньо для досягнення прийнятної продуктивності системи. На рис. 2.2 показана спрощена схема конфігурації апаратного

забезпечення для навчального проекту *DreamHome*. Вона складається з мережі міні-комп'ютерів із центральним комп'ютером у Лондоні. На центральному комп'ютері працює серверна частина СКБД (backend), що обслуговує й контролює доступ до бази даних. На схемі також показано кілька комп'ютерів, розташованих в інших регіонах. На них працюють клієнтські частини СКБД (frontend), які здійснюють взаємодію з користувачами.

Подібна архітектура зветься клієнт/сервер (client-server), де сервером є комп'ютер із серверною частиною СКБД, а клієнтами - комп'ютери із клієнтськими частинами СКБД. Більш докладно ця архітектура розглядається в розділі 2.9.

Програмне забезпечення

Цей компонент охоплює програмне забезпечення самої СКБД і прикладних програм, разом з операційною системою, включаючи й мережне програмне забезпечення, якщо СКБД використовується в мережі. Звичайно додатки створюються на мовах третього покоління, таких як C, C++, Java, Visual Basic, COBOL, Fortran, Ada або Pascal, або на мовах четвертого покоління, таких як SQL, оператори яких впроваджуються в програми на мовах третього покоління.

Втім, СКБД може мати свої власні інструменти четвертого покоління, призначені для швидкої розробки додатків з використанням вбудованих непроцедурних мов запитів, генераторів звітів, форм, графічних зображень і навіть повномасштабних додатків. Використання інструментів четвертого покоління дозволяє істотно підвищити продуктивність системи й сприяє створенню більш зручних для обслуговування програм.

До інструментів четвертого покоління відносять декларативні мови (третє покоління – процедурні й объектно-ориентированные мови програмування), які дозволяють програмістові вказувати, *що* потрібно одержати, але не вказувати *як*.

Дані

Імовірно, найважливішим компонентом середовища СКБД (з погляду кінцевих користувачів) є дані. На рис. 2.1 показано, що дані відіграють роль мосту між комп'ютером і людиною. База даних містить як робочі дані, так і метадані, тобто "дані про дані".

Структура бази даних називається схемою (schema). Показана на рис. 1.6 схема бази даних складається із чотирьох файлів, або таблиць (table): `PropertyForRent` (об'єкт, що здається в оренду), `PrivateOwner` (Власник власності), `Client` (Клієнт) і `Lease` (Договір

оренди). Таблиця PropertyForRent має вісім полів, або атрибутів: propertyNo (Номер об'єкта), street (Вулиця), city (Місто), postcode (Поштовий індекс), type (Тип об'єкта), rooms (Кількість кімнат), rent (Щомісячна орендна плата) і ownerNo (Номер власника). Атрибут ownerNo моделює зв'язок між таблицею PropertyForRent і таблицею PrivateOwner, тобто якийсь власник володіє (Owns) якоїсь нерухомістю, що здається в оренду — як показано на діаграмі "сутність-зв'язок", представленої на рис. 1.5.

В системному каталозі зберігаються відомості, які докладно розглядаються в розділі 2.10.

Процедури

До процедур відносяться інструкції й правила, які повинні враховуватися при проектуванні й використанні бази даних. Користувачам і обслуговуючому персоналу бази даних необхідно надати документацію, що містить докладний опис процедур використання й супроводу даної системи, включаючи інструкції про правила виконання наведених нижче дій.

- Реєстрація в СКБД.
- Використання окремого інструмента СКБД або додатка.
- Запуск і останов СКБД.
- Створення резервних копій СКБД.
- Обробка збоїв апаратного й програмного забезпечення, включаючи процедури ідентифікації компонента, що вийшов з ладу, виправлення компонента, що відмовив (наприклад, за допомогою виклику фахівця з ремонту апаратного забезпечення), а також відновлення бази даних після усунення несправності.
- Зміна структури таблиці, реорганізація бази даних, розміщеної на декількох дисках, способи поліпшення продуктивності й методи архівування даних на вторинних пристроях зберігання.

Користувачі

Останнім, ще не розглянутим нами компонентом середовища СКБД є користувачі системи. Цей компонент докладно обговорюється в розділі 2.3.

2.2 Розробка бази даних - зміна принципів проектування

Дотепер за замовчуванням передбачалося, що дані в базі мають деяку структуру. Наприклад, на рис. 1.6 показані чотири таблиці: PropertyForRent, PrivateOwner, Client і Lease. Але як була

отримана така структура? Відповідь на це питання досить проста: структура бази даних визначається під час її проектування. Однак сам процес проектування бази даних може виявитися надзвичайно складним. Для створення системи, що задовольняла б інформаційним потребам деякої організації, необхідно використати підхід, що зовсім відрізняється від методів розробки звичайних файлових систем, у яких вся робота полягає в розробці додатків, що задовольняють потребам окремих підрозділів.

Для успішної реалізації системи на основі бази даних необхідно подумати насамперед про дані і лише потім про додатки. Така зміна підходу цілком може розцінюватися як зміна принципів проектування. Щоб система повністю задовольняла запитам користувачів, необхідно дуже уважно поставитися до процесу проектування бази даних. Погано спроектована база даних буде породжувати помилки, здатні привести до прийняття неправильних рішень, які спричинять самі серйозні наслідки для даної організації. З іншого боку, добре спроектована база даних дозволить створити систему, що поставляє коректну інформацію, що може успішно використовуватись для прийняття правильних і ефективних рішень.

Деякі аспекти проектування баз даних розглядаються в даному курсі (розділи 2.4, 7), але більш докладно проблеми проектування баз даних розглядаються в курсі «Системний аналіз і проектування інформаційних систем».

2.3 Розподіл обов'язків у системах з базами даних

У цьому розділі ми розглянемо згаданий вище п'ятий компонент середовища СКБД – її користувачів. Серед них можна виділити чотири різні групи: адміністратори даних і баз даних, розроблювачі баз даних, прикладні програмісти й кінцеві користувачі.

2.3.1 Адміністратори даних і адміністратори баз даних

База даних і СКБД є корпоративними ресурсами, якими необхідно керувіти так само, як і будь-якими іншими ресурсами. Звичайне керування даними й базою даних передбачає керування й контроль за СКБД і поміщеними в неї даними. Адміністратор даних, або АД (Data Administrator - DA), відповідає за керування даними, включаючи планування бази даних, розробку й супровід стандартів, прикладних алгоритмів і ділових процедур, а також за концептуальне й логічне проектування бази даних. АД консультує й дає свої рекомендації керівництву вищої ланки, контролюючи відповідність загального напрямку розвитку бази даних установленим корпоративним цілям.

Адміністратор бази даних, або АБД (Database Administrator - DBA), відповідає за фізичну реалізацію бази даних, включаючи фізичне проектування й втілення проекту, за забезпечення безпеки й цілісності даних, за супровід операційної системи, а також за забезпечення максимальної продуктивності додатків і користувачів. У порівнянні з АД обов'язки АБД носять більш технічний характер, і для нього необхідне знання конкретної СКБД і системного оточення. В одних організаціях між цими ролями не робиться відмінностей, а в інших важливість корпоративних ресурсів відбита саме у виділенні окремих груп персоналу із зазначеним колом обов'язків.

2.3.2 Розроблювачі баз даних

У проектуванні великих баз даних беруть участь розроблювачі двох різних типів: *розроблювачі логічної бази даних* і *розроблювачі фізичної бази даних*. Розроблювач логічної бази даних займається ідентифікацією даних (тобто сутностей і їхніх атрибутів), зв'язків між даними, і встановлює обмеження, що накладають на дані, що зберігаються. Розроблювач логічної бази даних повинен мати всебічне й повне розуміння структури даних організації і її діловий регламент. Діловий регламент описує основні вимоги до системи з погляду організації. Нижче наведений приклад типового ділового регламенту для проекту *DreamHome*.

- Будь-який співробітник не може відповідати одночасно більш ніж за сто об'єктів нерухомості, які продаються або здаються в оренду.
- Будь-який співробітник не має права продавати або здавати в оренду свою власну нерухомість.
- Довірена особа не може виступати одночасно і як покупець, і як продавець нерухомості.

Для ефективної роботи розроблювач логічної бази даних повинен якомога раніше притягнути всіх передбачуваних користувачів бази даних до процесу створення моделі даних. Робота *розроблювача логічної бази* даних розподіляється на два етапи:

- Концептуальне проектування бази даних, що зовсім не залежить від таких деталей її втілення, як конкретна цільова СКБД, додатки, мови програмування або будь-які інші фізичні характеристики.
- Логічне проектування бази даних, що проводиться з урахуванням особливостей обраної моделі даних: реляционной, мережний, ієрархічної або объектно-ориентированной.

Розроблювач фізичної бази даних одержує готову логічну модель даних і займається її фізичною реалізацією, у тому числі:

- перетворенням логічної моделі даних у набір таблиць і обмежень цілісності даних;
- вибором конкретних структур зберігання й методів доступу до даних, що забезпечують необхідний рівень продуктивності при роботі з базою даних;
- проектуванням будь-яких необхідних мір захисту даних.

Багато етапів фізичного проектування бази даних у значній мірі залежать від обраної цільової СКБД, а тому може існувати кілька різних способів втілення необхідної схеми. Отже, розроблювач фізичної бази даних повинен розбиратися у функціональних можливостях цільової СКБД і розуміти переваги й недоліки кожного можливого варіанта реалізації. Розроблювач фізичної бази даних повинен уміти вибрати найбільш підходящу стратегію зберігання даних з урахуванням всіх існуючих особливостей їхнього використання.

Якщо концептуальне й логічне проектування бази даних, відповідає на запитання "що?", то фізичне проектування відповідає на запитання "як?". Для рішення цих задач потрібні різні навички роботи, якими найчастіше володіють різні люди. Методологія концептуального проектування бази даних докладно розглядається в курсі “Системний аналіз і проектування інформаційних систем”. У цьому ж курсі також розглядаються аспекти логічного й фізичного проектування. Але деякі аспекти логічного й фізичного проектування розглядаються також у даному курсі.

2.3.3 Прикладні програмісти

Відразу після створення бази даних потрібно приступитися до розробки додатків, що надають користувачам необхідні їм функціональні можливості. Саме цю роботу й виконують прикладні програмісти. Звичайно прикладні програмісти працюють на основі специфікацій, створених системними аналітиками. Як правило, кожна програма містить деякі оператори, що вимагають від СКБД виконання певних дій з базою даних - наприклад, таких як витяг, вставка, відновлення або видалення даних. Як уже згадувалося в попередньому розділі, ці програми можуть створюватися на різних мовах програмування третього або четвертого покоління.

2.3.4 Користувачі

Користувачі є клієнтами бази даних - вона проектується, створюється й підтримується для того, щоб обслуговувати їхні інформаційні потреби. Користувачів можна класифікувати по способу використання ними системи.

- **Рядові користувачі**. Ці користувачі звичайно навіть і не підозрюють про наявність СКБД. Вони звертаються до бази даних за допомогою спеціальних додатків, що дозволяють у максимальному ступені спростити операції, що ними виконуються. Такі користувачі ініціюють виконання операцій бази даних, уводячи найпростіші команди або вибираючи команди меню. Це значить, що таким користувачам не потрібно нічого знати про базу даних або СКБД. Наприклад, щоб довідатися ціну товару, касир у супермаркеті використовує сканер для зчитування нанесеного на нього штрих-коду. У результаті цієї найпростішої дії спеціальна програма не тільки зчитує штрих-код, але й вибирає на основі його значення ціну товару з бази даних, а також зменшує значення в іншім полі бази даних, що позначає залишок таких товарів на складі, після чого вибиває ціну й загальну вартість на касовому апарату.
- **Досвідчені користувачі.** З іншої сторони спектра перебувають досвідчені кінцеві користувачі, які знайомі зі структурою бази даних і можливостями СКБД. Для виконання необхідних операцій вони можуть використовувати таку мову запитів високого рівня, як SQL. А деякі досвідчені користувачі можуть навіть створювати власні прикладні програми.

2.4 Трирівнева архітектура ANSI-SPARC

Перша спроба створення стандартної термінології й загальної архітектури СКБД 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).

Подібні термінологія й архітектура були запропоновані в 1975 році Комітетом планування стандартів і норм SPARC (Standards Planning and Requirements Committee) Національного інституту стандартизації США (American National Standard Institute — ANSI), ANSI/X3/SPARC. Комітет

ANSI/SPARC визнав необхідність використання трирівневого підходу до створення системного каталогу. У цих матеріалах відбиті пропозиції, які були зроблені організаціями Guide and Share, що складаються з користувачів продуктів корпорації IBM, і опубліковані за кілька років до цього. Основну увагу в них було сконцентровано на необхідності втілення незалежного рівня для ізоляції програм від особливостей подання даних на більш низькому рівні. Хоча модель ANSI/SPARC не стала стандартом, вона усе ще являє собою основу для розуміння деяких функціональних особливостей СКБД.

Концепції багаторівневої архітектури СКБД 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 СКБД, що включає концептуальний, внутрішній і зовнішній рівні архітектури. Така модель описує архітектуру будь-якої системи з тим лише застереженням, що в кожній конкретній СКБД 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. 2.3. Рівень, на якому дані сприймаються користувачами, називається зовнішнім рівнем (external level), тоді як СКБД 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. 2.3. 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.
- Користувачи не повинні безпосередньо мати справу з такими подробицями фізичного зберігання даних у базі, як індексування й хешірування (див. розділ 3). 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.

2.4.1. 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 СКБД при виявленні відповідного посилання. Представлення можуть також включати комбіновані або похідні дані з декількох об'єктів. Більш докладно представлення розглядаються в розділі 5.6.

2.4.2. 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.

2.4.3 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 СКБД 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 СКБД. Однак функції СКБД і операційної системи на фізичному рівні не цілком чітко розділені й можуть варіюватися від системи до системи. В одних СКБД 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 СКБД 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).

2.4.4 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. 2.3.

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.

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

Концептуальна схема пов'язана із внутрішньою схемою за допомогою *концептуально внутрішнього відображення*. Воно дозволяє СКБД 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 СКБД може відображати імена представленя користувача на відповідну частину концептуальної схеми.

Приклади різних рівнів наведені на рисунку 2.4. На рисунку показані два різних зовнішніх представлення інформації про персонал: одне складається з табельного номера співробітника (`sNo`), його імені (`fName`) і прізвища (`lName`), віку (`age`), суми зарплати за рік (`salary`). Інше представлення включає табельний номер співробітника (`staffNo`), прізвище (`lName`) і номер відділення компанії, у якому він працює (`branchNo`).

Ці зовнішні представлення зливаються воєдино в одному концептуальному представленні. Особливістю даного процесу злиття є те, що поле віку співробітника (`age`) перетвориться в поле дати його народження (`DOB`). СКБД підтримує концептуально зовнішнє відображення. Наприклад, поле `sNo` з першого зовнішнього представлення відображається на поле `staffNo` у записі концептуального представлення.

Потім концептуальний рівень відображається на внутрішній рівень, що містить фізичний опис структури запису концептуального представлення. На цьому рівні визначення структури формулюється мовою високого рівня. Ця структура містить покажчик (`next`), що дозволяє фізично зв'язати всі записи про співробітників у єдиний ланцюжок. Зверніть увагу, що порядок полів на внутрішньому рівні відрізняється від порядку атрибутів, прийнятого на концептуальному рівні. Таким є механізм, за допомогою якого СКБД здійснює *концептуально внутрішнє відображення*.

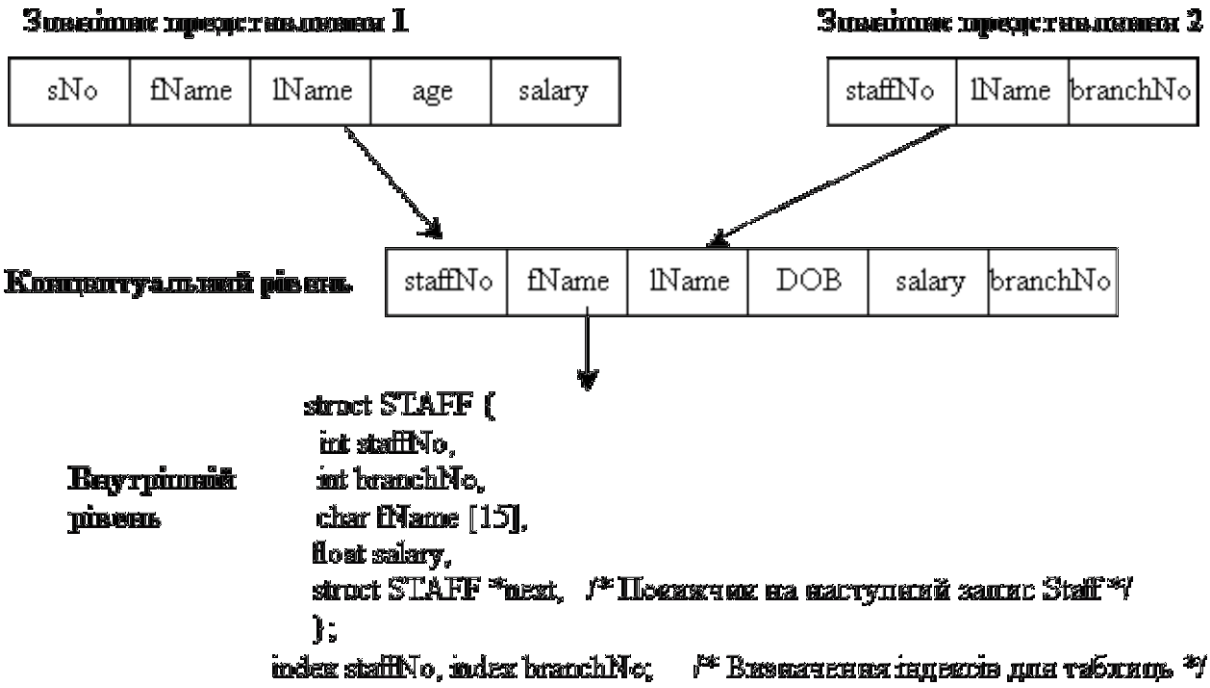


Рис. 2.4 Розходження між трьома рівнями представлення даних

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*.

2.4.5 Незалежність від даних

Основним призначенням трехуровневой 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 хешірування, 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. 2.5 показане місце перерахованих вище типів незалежності від даних у трирівневій архітектурі СКБД.

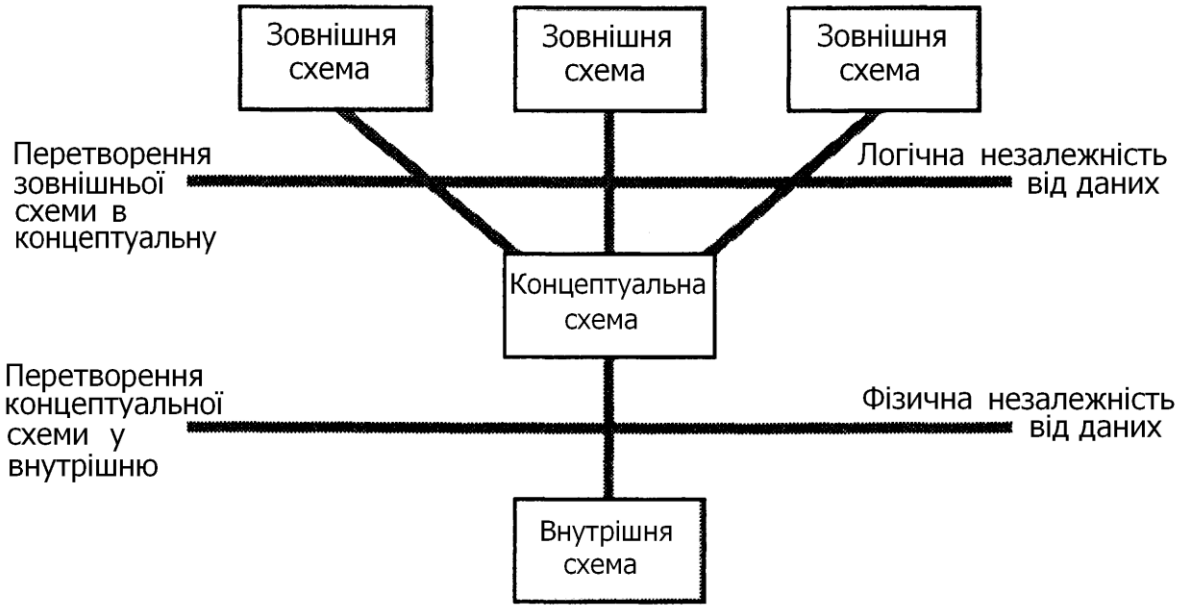


Рис. 2.5 Реалізація незалежності від даних у трирівневій архітектурі ANSI-SPARC

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

2.5 Мови баз даних

Внутрішня мова СКБД 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 СКБД 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 СКБД, 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.

2.5.1 Мова визначення даних - DDL

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 СКБД. Перед доступом до реальних даних СКБД 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 СКБД. 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.

2.5.2 Мова керування даними - DML

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 СКБД 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 СКБД 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. СКБД 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 СКБД 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 СКБД — більша. Більш докладно мова SQL розглядається в главах 9 і 11.

2.5.3 Мови 4GL

Абревіатура 4GL являє собою скорочений англійський варіант написання терміна мова четвертого покоління (Fourth-Generation Language). Чіткого визначення цього поняття не існує, хоча, по суті, мова йде про деякий стенографічний варіант мови програмування. Якщо для організації деякої операції з даними мовою 3-го покоління буде потрібно написати сотні рядків коду, то для реалізації цієї ж операції мовою 4-го покоління досить 10-20 рядків.

У той час як мови 3-го покоління є процедурними, мови 4GL виступають як непроцедурні, оскільки користувач визначає, *що* повинне бути зроблене, але не повідомляє, *як саме* повинен бути досягнуть бажаний результат.

2.6 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 СКБД. 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 СКБД 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 СКБД;
- внутрішня модель даних, що відображає концептуальну схему певним чином, що підходить для обраної цільовий СКБД.

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.

2.7 Функції СКБД

Основні функції СКБД 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 СКБД.

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

2.7.1 Зберігання, добування й відновлення даних

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

Це сама фундаментальна функція СКБД. З обговорення, проведеного в розділі 1.2, ясно, що спосіб реалізації цієї функції в СКБД повинен дозволяти приховувати від кінцевого користувача внутрішні деталі фізичної реалізації системи (наприклад, файлову організацію або структури зберігання, які використовуються).

2.7.2 Каталог, доступний кінцевим користувачам

СКБД 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 СКБД. 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 СКБД, 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 (див. розділ 2.4, 2.6);
- 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.

2.7.3 Підтримка транзакцій

СКБД 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.

Прикладами простих транзакцій у проекті *DreamHome* може служити додавання в базу даних відомостей про нового співробітника, відновлення відомостей про зарплату деякого співробітника або видалення відомостей про деякий об'єкт нерухомості.

Прикладом більше складної транзакції може бути видалення відомостей про співробітника з бази даних і передача відповідальності за всі об'єкти нерухомості, якими він курирує, іншому співробітникові. У цьому випадку в базу даних буде потрібно внести відразу кілька змін.

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. Більш докладно обробка транзакцій розглядається в розділі 13.

2.7.4 Служби керування паралельною роботою

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

Одна з основних цілей створення й використання СКБД полягає в тім щоб безліч користувачів могло 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.

Розглянемо транзакції T_1 і T_2, які паралельно виконуються, і послідовність виконання яких представлена в табл. 2.1.

Таблиця 2.1 Приклад виникнення проблеми "загубленого відновлення"

Час	Транзакція T_1	Транзакція T_2	Стовпець bal_x
t_1		read (bal_x)	100
t_2	read(bal_x)	$bal_x = bal_x + 100$	100
t_3	$bal_x = bal_x - 10$	write (bal_x)	200
t_4	write (bal_x)		90
t_5			90

У транзакції T_1 10 доларів знімається з рахунку, залишок якого зберігається в стовпці bal_x, а в транзакції T_2 100 доларів додаються на цей же рахунок. Якби транзакції виконувалися послідовно, тобто одна за іншою, без чергування окремих операцій, то в остаточному підсумку залишок на рахунку дорівнював би 190 доларам, незалежно від порядку виконання транзакцій.

У протилежному випадку може виникнути наступна ситуація. Допустимо, що транзакції T_1 і T_2 стартували практично одночасно й прочитали вихідне значення залишку, яке дорівнює 100 доларів. Потім транзакція T_2 збільшує це значення на 100 доларів (до 200 доларів) і зберігає отримане значення в базі даних. Негайно після цього транзакція

T_1 зменшує свою копію зчитаного вихідного значення на 10 доларів (до 90 доларів) і зберігає отримане значення в базі даних замість тільки що записаного результату попереднього відновлення, внаслідок чого виникає ефект "втрати" 100 доларів.

СКБД повинна гарантувати, що при одночасному доступі до бази даних багатьох користувачів подібних конфліктів не відбудеться. Більш докладно це питання розглядається в розділі 13.

2.7.5 Служби відновлення

СКБД 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 СКБД.

Крім того, користувач може виявити помилку під час виконання транзакції й зажадати її скасування. У всіх цих випадках СКБД повинна надати механізм відновлення бази даних і повернення її до несуперечливого стану. Це питання більш докладно розглядається в розділі 13.

2.7.6 Служби контролю доступу до даних

СКБД 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 СКБД забезпечує механізми подібного захисту даних. Більш докладно заходи щодо забезпечення безпеки розглядаються в главі 13.

2.7.7 Підтримка обміну даними

СКБД 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 СКБД. В інших випадках термінали можуть перебувати на значній відстані й обмінюватися даними з комп'ютером, на якому розташовується СКБД, через мережу.

У кожному разі СКБД 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 СКБД, проте, щоб бути комерційно життєздатної, будь-яка СКБД повинна мати здатність інтеграції з різноманітними існуючими диспетчерами обміну даними.

Навіть СКБД для персональних комп'ютерів повинні підтримувати роботу в локальній мережі, щоб замість декількох розрізнених баз даних для кожного окремого користувача можна було б установити одну централізовану базу даних і використовувати її як загальний ресурс для всіх існуючих користувачів. При цьому передбачається, що не база даних повинна бути розподілена в мережі, а віддалені користувачі повинні мати можливість доступу до централізованої бази даних. Така організація роботи називається *розподіленою обробкою*.

2.7.8 Служби підтримки цілісності даних

СКБД 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. Наприклад, можна вказати, що співробітник не може відповідати одночасно більш ніж за сто об'єктів нерухомості. У цьому випадку при спробі закріпити черговий об'єкт нерухомості за деяким співробітником СКБД повинна перевірити, чи не перевищений установлений ліміт, і у випадку виявлення подібного перевищення заборонити закріплення нового об'єкта за даним співробітником.

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

2.7.9 Служби підтримки незалежності від даних

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

Поняття незалежності від даних уже розглядалося в розділі 2.4.5. 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.

2.7.10 Допоміжні служби

СКБД 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 СКБД. 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.

2.8 Компоненти СКБД

СКБД є досить складним видом програмного забезпечення, призначеним для надання перерахованих у попередньому розділі служб.

Компонентну структуру СКБД практично неможливо узагальнити, оскільки вона дуже сильно розрізняється в різних системах. Однак при вивченні систем баз даних корисно уявляти собі її узагальнену структуру у вигляді набору з декількох компонентів і певних зв'язків між ними. У цьому розділі ми розглянемо одну з можливих архітектур СКБД.

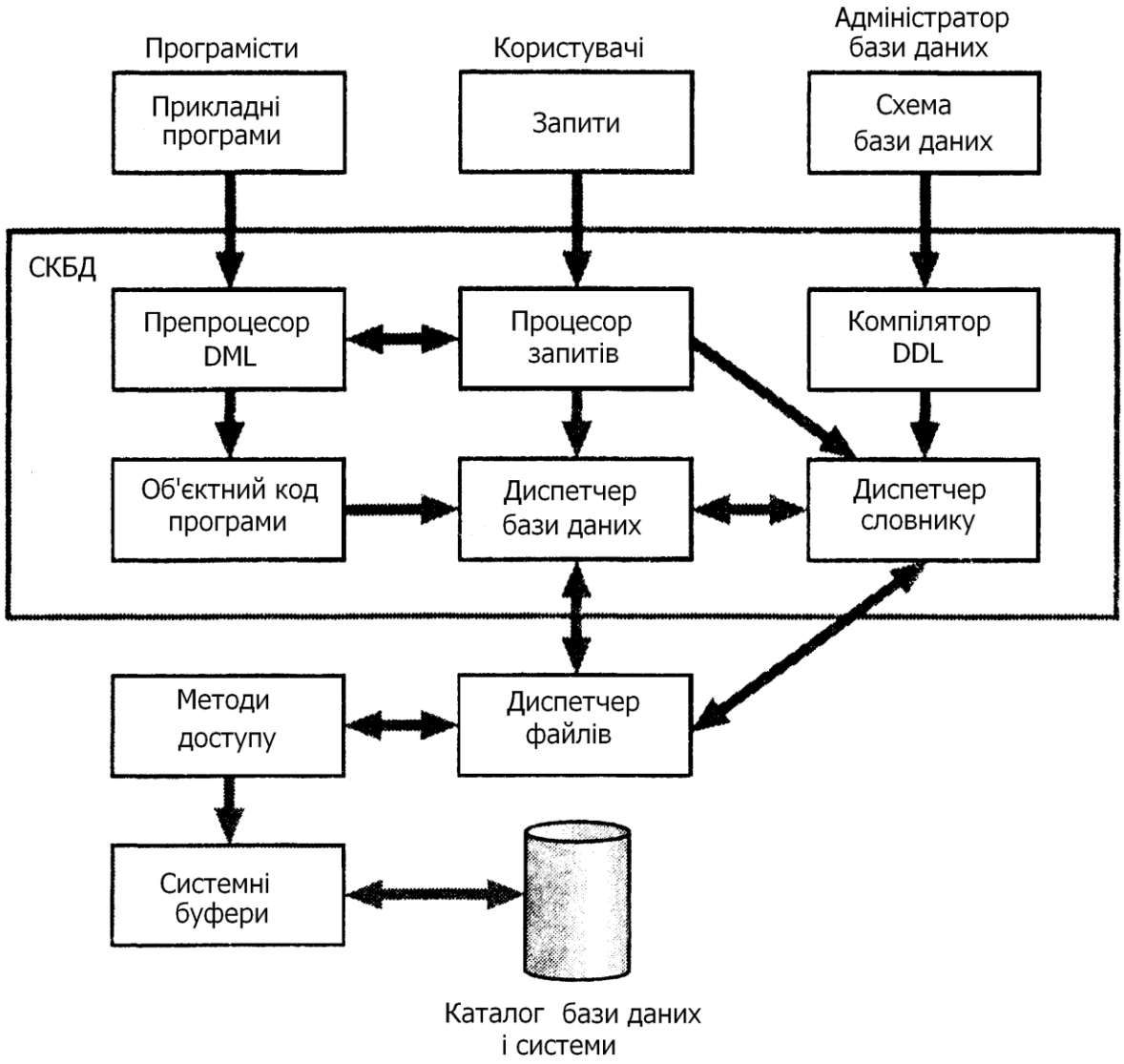
СКБД складається з декількох програмних компонентів (модулів), кожний з яких призначений для виконання специфічної операції. Як уже говорилося вище, деякі функції СКБД можуть підтримуватися операційною системою, що використовується. Однак у кожному разі операційна система надає тільки базові служби, і СКБД завжди являє собою надбудову над ними. Таким чином, при проектуванні СКБД варто враховувати особливості інтерфейсу між СКБД, що створюється, і конкретною операційною системою.

Основні програмні компоненти середовища СКБД представлені на рис. 2.7. На цій схемі також показано, як СКБД взаємодіє з іншими програмними компонентами, наприклад з такими, як запити користувачів й методи доступу (тобто методи керування файлами, що використовуються при збереженні й добуванні записів з даними).

На рис. 2.7 показані наступні програмні компоненти середовища СКБД.

- **Процесор запитів.** Це основний компонент СКБД, що перетворить запити в послідовність низькорівневих команд для диспетчера бази даних.
- **Диспетчер бази даних**. Цей компонент взаємодіє із запущеними користувачами прикладними програмами й запитами. Диспетчер бази даних приймає запити й перевіряє зовнішні й концептуальні схеми для визначення тих концептуальних записів, які необхідні для задоволення вимог запиту. Потім диспетчер бази даних викликає диспетчер файлів для виконання запиту, що надійшов. Компоненти диспетчера бази даних показані на рис. 2.8.
- **Диспетчер файлів**. Маніпулює призначеними для зберігання даних файлами й відповідає за розподіл доступного дискового простору. Він створює й підтримує список структур і індексів, визначених у внутрішній схемі. Якщо використовуються хешировані файли, то в його обов'язок входить і виклик функцій хешірування для генерації адрес записів (індекси й хешірування розглядаються в розділі 3). Однак диспетчер файлів не управляє фізичним введенням і виводом даних безпосередньо, а лише передає запити відповідним методам

доступу, які зчитують дані в системні буфери або записують їх відтіля на диск (або в кеш).



Мал. 2.7 Основні компоненти типової системи керування базами даних

- **Препроцесор мови DML**. Цей модуль перетворює впроваджені в прикладні програми DML-оператори у виклики стандартних функцій базової мови. Для генерації відповідного коду препроцесор мови DML повинен взаємодіяти із процесором запитів.
- **Компілятор мови DDL**. Компілятор мови DDL перетворить DDL-команди в набір таблиць, що містять метадані. Потім ці таблиці зберігаються в системному каталозі, а керуюча інформація – у заголовках файлів з даними.

- **Диспетчер словника.** Диспетчер словника управляє доступом до системного каталогу й забезпечує роботу з ним. Системний каталог доступний більшості компонентів СКБД.

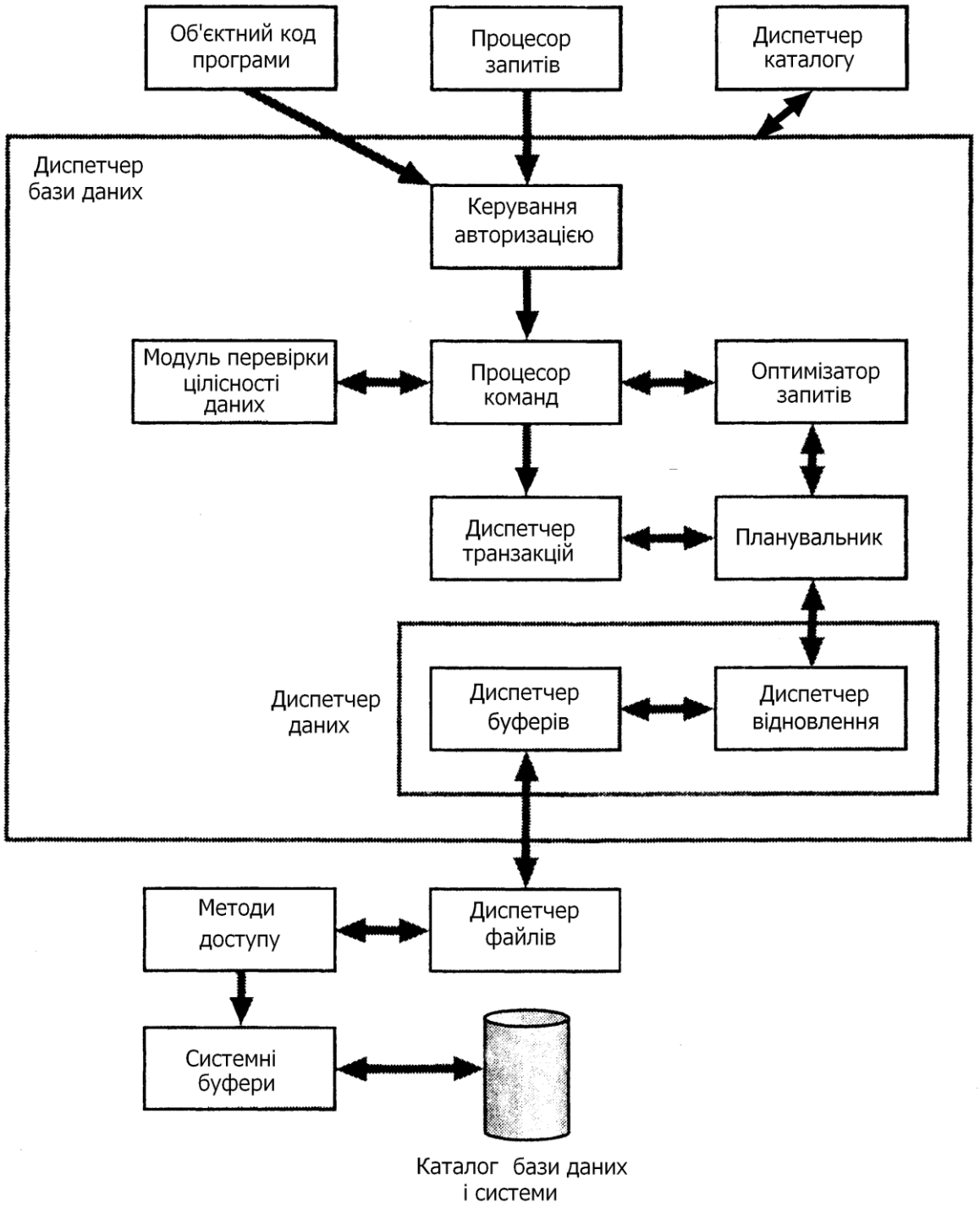


Рис. 2.8 Компоненти диспетчера бази даних

Нижче перераховані основні програмні компоненти, що входять до складу диспетчера бази даних, зображені на рисунку 2.8:

- **Модуль контролю прав доступу (керування авторизацією)**. Цей модуль перевіряє наявність у даного користувача повноважень для виконання викликаної операції.
- **Процесор команд**. Після перевірки повноважень користувача керування передається процесору команд для виконання викликаної операції.
- **Засоби контролю цілісності**. У випадку операцій, які змінюють вміст бази даних, засоби контролю цілісності виконують перевірку того, чи задовольняє викликана операція всім установленим обмеженням підтримки цілісності даних (наприклад, вимогам, установленим для ключів).
- **Оптимізатор запитів**. Цей модуль визначає оптимальну стратегію виконання запиту.
- **Диспетчер транзакцій**. Цей модуль здійснює необхідну обробку операцій, що надходять у процесі виконання транзакцій.

2.9 Архітектура СКБД багатьох користувачів

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

По-перше, на кожному комп'ютері, що мав право працювати з базою даних, розташовувалася своя версія СКБД, а також необхідне програмне забезпечення для забезпечення захисту й цілісності бази даних; така архітектура називається *архітектурою з файловим сервером* – як файловий сервер виступає комп'ютер, на якому розташовується база даних; інші комп'ютери виступають у ролі робочих станцій.

По-друге, робота з базою даних могла бути організована в *режимі телеобробки*, при якому один комп'ютер з єдиним процесором був з'єднаний з декількома терміналами. При цьому вся обробка виконувалася за допомогою єдиного комп'ютера, а приєднані до нього термінали користувачів були “неінтелектуальними” пристроями, не здатними функціонувати самостійно.

Обидві ці архітектури мають свої недоліки. Для архітектури з файловим сервером можна відзначити наступні недоліки:

1. Проблеми з розробкою програмного забезпечення: кожний комп'ютер повинен бути забезпечений повною копією СКБД 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 СКБД. 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.

Основним недоліком архітектури з телеобробкою є її низька продуктивність (фактично з базою даних працює тільки один комп'ютер). Крім того, оскільки термінали можуть тільки одержувати або передавати дані для бази даних, і не можуть виконувати функції звичайного персонального комп'ютера, то необхідне дублювання апаратного забезпечення: для роботи з базою даних у підрозділі повинен бути термінал, а для іншої комп'ютерної роботи – звичайний персональний комп'ютер.

З метою усунення недоліків цих двох підходів 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. 2.9.

Рис. 2.9 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 зпрощується, якщо на ньому виконується тільки робота з базою даних.
- Вартість апаратного забезпечення знижується. Достатньо потужний комп'ютер з великим пристроєм збегігання потрібен тільки серверу – для зберігання й керування базою даних.
- Зменшуються комунікаційні витрати. Додатки виконують частину операцій на клієнтских 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 цілістності даних, оскільки всі обмеження визначаються і перевіряються тільки в одному місці. При цьому кожному додатку не потрібно виконувати власну перевірку.
- Ця архітектура досить природньо відображається на архітектуру відкритих систем.

У даному курсі робота із СКБД розглядається на прикладі локальної СКБД FoxPro, що при роботі в мережі працює, звичайно, за технологією файлового сервера. Роботі із клієнт-серверними СКБД присвячений наступний курс – “Розробка прикладних систем баз даних”

2.10 Системні каталоги

У розділі 2.7 згадано, що СКБД повинна мати доступний кінцевому користувачеві системний каталог або словник даних. У цьому розділі ми познайомимося із системними каталогами більш докладно.

Системний каталог. Сховище даних, які описують інформацію, що зберігає в базі даних, тобто метадані, або "дані про дані".

Системний каталог СКБД є одним з фундаментальних компонентів системи. Більшість перерахованих в розділі 2.8 програмних компонентів будуються на використанні даних, що зберігаються в системному каталозі. Наприклад, модуль контролю прав доступу використовує системний каталог для перевірки наявності в користувача повноважень, необхідних для виконання запитаних їм операцій. Для проведення подібної перевірки системний каталог повинен включати наступні компоненти:

- імена користувачів, для яких дозволений доступ до бази даних;
- імена елементів даних у базі даних;
- елементи даних, до яких кожний користувач має право доступу, і дозволені типи доступу до них – для вставки, відновлення, видалення або читання.

Іншим прикладом можуть служити засоби перевірки цілісності даних, які використовують системний каталог для перевірки того, чи задовольняє запитана операція всім установленим обмеженням підтримки цілісності даних. Для виконання цієї перевірки в системному каталозі повинні зберігатися такі відомості:

- імена елементів даних з бази даних;
- типи й розміри елементів даних;
- обмеження, установлені для кожного з елементів даних.

Як уже згадувалося раніше, термін "словник даних" часто використовується для програмного забезпечення більш загального типу, чим просто каталог СКБД. Система словника даних може бути або *пасивної*, або *активної*.

Активна система завжди узгоджується зі структурою бази даних, оскільки вона автоматично підтримується цією системою. Пасивна система може суперечити стану бази даних через зміни, що ініціюються користувачами. Якщо словник даних є частиною бази даних, то він називається *інтегрованим словником даних*.

Автономний словник даних має свою власну спеціалізовану СКБД. Його переважно використовують на початкових етапах проектування бази

даних для деякої організації, коли потрібно відкласти на якийсь час прив'язку до конкретного СКБД. Однак недолік цього підходу полягає в тім, що після вибору СКБД і втілення бази даних автономний словник даних значно сутужніше підтримувати узгодженим зі станом бази даних. Цю проблему можна було б звести до мінімуму, якщо перетворити словник, що використовувався при проектуванні даних, безпосередньо в каталог СКБД.

3 ПОШУК ДАНИХ

Розглянемо проблему пошуку деякої одиниці інформації у великому обсязі даних. Як ми побачимо, деякі методи організації даних дозволяють виконати процес пошуку більш ефективно.

3.1 Основні методи пошуку

Перш ніж розглядати конкретні методи пошуку, визначимо деякі терміни. *Таблиця* або *файл* є деякою групою елементів, кожний з яких називається *записом*.

З кожним записом асоціюється деякий *ключ*, що використовується для того, щоб відрізнити один запис від іншої. Відповідність між записом і ключем може бути простою або складною. У найпростішому випадку ключ є деяким полем усередині запису, що розташовується з деяким конкретним зміщенням від початку запису. Такий ключ називається *внутрішнім ключем* або *вбудованим ключем*. В інших випадках ключем є відносна позиція запису усередині файлу або є деяка окрема таблиця ключів, що містить покажчики на записі. Такі ключі називаються *зовнішніми ключами*.

Для кожного файлу є принаймні один набір ключів (можливо й кілька таких наборів), які є унікальними (тобто ніякі два записи у файлі не мають однакового значення ключа). Такий ключ називається *потенційним ключем*. Один з потенційних ключів обирається в якості *первинного* ключа. Наприклад, якщо файл зберігається як деякий масив, то індекс деякого елемента в цьому масиві є унікальним зовнішнім ключем для цього елемента.

Однак оскільки будь-яке поле запису може служити як ключ у якому-небудь конкретному додатку, то ключі не завжди повинні бути унікальними. Наприклад, якщо в деякому файлі із прізвищами й адресами назва міста використовується як ключ для деякого пошуку, то він, можливо, не буде унікальним, тому що у файлі можуть бути два записи з назвою того самого міста. Такий ключ називається *вторинним ключем*.

Деякі з розглянутих алгоритмів припускають наявність унікальних ключів, а інші дозволяють використовувати ключі, що повторюються. При адаптації деякого алгоритму для конкретного додатка програміст повинен знати, чи будуть ключі унікальними, і переконатися, що даний алгоритм на це розрахований.

Друге визначення зовнішнього ключа – це вторинний ключ, що служить для зв'язку з іншою (базовою) таблицею, і містить значення первинного ключа базової таблиці.

Алгоритмом пошуку є деякий алгоритм, що сприймає деякий аргумент `a` й намагається локалізувати деякий запис, ключ якої дорівнює `a`. Даний алгоритм може повернути весь даний запис або найчастіше може повернути деякий покажчик на цей запис. Однак можливо, що пошук деякого конкретного аргументу в таблиці є невдалим, тобто в даній таблиці не існує запису із цим аргументом як ключ. У цьому випадку такий алгоритм може повернути деякий спеціальний «порожній запис» або деякий порожній покажчик. Якщо пошук закінчився невдало, то часто буває бажано додати деякий новий запис із цим аргументом як ключ. Алгоритм, що виконує цю функцію, називається *алгоритмом пошуку і вставки*.

Зауважте, що нічого не було сказано про те, як організована таблиця або файл. Це може бути масив записів, зв'язаний список, дерево або навіть граф. Оскільки різні методи пошуку можуть відповідати різним організаціям таблиць, то таблиця часто будується, виходити з міркувань конкретного методу пошуку. Така таблиця може повністю розташовуватися в оперативній пам'яті, або в допоміжній пам'яті, або там і там. Ясно, що для різних організацій таблиць необхідні різні методи пошуку.

Методи пошуку, при яких вся таблиця постійно перебуває в оперативній пам'яті, називаються *методами внутрішнього пошуку*, а ті методи, для яких більша частина таблиці зберігається в допоміжній пам'яті (такий, як диск), називаються *методами зовнішнього пошуку*. Розглянемо внутрішній пошук.

3.2 Послідовний пошук

Найпростішою формою пошуку є послідовний пошук. Цей пошук застосовується до таблиці, що організована або як масив, або як зв'язаний список. Припустимо, що `k` є деякий масив з `n` ключів, а `r` — деякий масив записів, такий що `k[i]` є ключем для `r[i]`. Припустимо також, що `key` є деяким аргументом пошуку. Ми хочемо встановити в змінну `search` найменше ціле число `i`, таке що `k[i] =key`, якщо таке `i` існує, і `0` у протилежному випадку. Алгоритм виконання цих дій наступний (C/C++):

```
search=0; i=1;
While ((i<=n)&&(k[i]!=key)) i++;
if (i<=n) search=i;
```

Даний алгоритм перевіряє по черзі кожний ключ. При знаходженні ключа, що збігається з аргументом пошуку, видається його індекс (який виступає в ролі деякого покажчика на запис). Якщо збіг не знайдений, то видається 0.

Цей алгоритм може бути просто модифікований для того, щоб додавати деякий запис `rec` із ключем `key` у таблицю, якщо ключа `key` немає в даній таблиці: замість короткого оператору `if` в останньому рядку потрібно написати повний оператор `if`:

```
if (i<=n) search=i;
else { n++; r[n]=rec;  k[n]=key; search=n;}
```

Для того, щоб використовувати послідовний пошук із вставкою для деякого масиву, для цього масиву попередньо повинне бути виділене достатньо пам'яті.

Представлення таблиці у вигляді деякого зв'язаного списку має ті переваги, що розмір такої таблиці може бути в міру необхідності динамічно збільшений. Іншою перевагою представлення таблиці у вигляді зв'язаного списку, а не масиву, є те, що зі зв'язаного списку простіше видалити запис. Видалення елемента з масиву вимагає переміщення в середньому половини його елементів.

3.3 Пошук в упорядкованій таблиці

Якщо таблиця впорядкована за значенням ключа записів, яке зменшується, або за значенням, що збільшується, то існує декілька методів, які можуть бути використані для збільшення ефективності пошуку. Це особливо справедливо для таблиць із фіксованим розміром.

Однією явно очевидною перевагою пошуку у відсортованому файлі перед пошуком у невідсортованому файлі є той випадок, коли запис із ключем, рівним аргументу, відсутній у файлі. У цьому випадку для виявлення такої ситуації в невідсортованому файлі необхідно виконати `n` порівнянь. У випадку відсортованого файлу, припускаючи, що значення аргументу ключа рівномірно розподілені в діапазоні ключів у файлі, необхідно виконати (у середньому) тільки `(n+1)/2` порівнянь. Це відбувається через те, що ми знаємо, що запис із деяким заданим ключем відсутній у файлі, який упорядкований за зростанням значення ключа, коли у файлі зустрічається ключ, що більше, ніж значення аргументу.

Як правило, записи в базі даних зберігаються в неупорядкованому вигляді. Фізичне сортування записів для реальної БД із великою кількістю інформації вимагає досить великого часу і є неефективною з кількох причин:

1. Інформація в БД постійно обновляється: записи редагуються, додаються, видаляються; при бажанні мати увесь час упорядковану базу даних довелося б після кожної операції зміни записів в БД заново впорядковувати базу, що зробило б роботу з такою базою даних практично неможливої.

2. Найчастіше запис у базі даних має декілька полів, і пошук може знадобитися не тільки по одному із цих полів. Але неможливо виконати таке сортування, щоб одночасно були впорядковані записи по всіх полях. Отже, упорядкування записів у таблицях втрачає зміст.

3.4 Індексно-послідовний пошук

Для збільшення ефективності пошуку в упорядкованому файлі існує метод, що приводить до збільшення необхідного простору. Цей метод називається *індексно-послідовним методом пошуку*. На додаток до файлу заводиться деяка допоміжна таблиця, іка називається *індексом*. Кожний елемент індексу складається із ключа `kindex` і покажчика на запис у файлі, що відповідає цьому ключу `kindex`. Елементи в індексі повинні бути відсортовані за цим ключем.

Дійсною перевагою індексно-послідовного методу є те, що елементи в таблиці можуть бути перевірені послідовно, якщо доступ повинен бути здійснений до всіх записів у файлі, а для доступу до деякого конкретного елемента час пошуку сильно скорочено. Послідовний пошук виконується по меншому індексі, а не по великій таблиці. Коли знайдений правильний індекс, другий послідовний пошук виконується по невеликій частині записів самої таблиці.

Індекс може бути представлений і у вигляді зв'язаного списку, і у вигляді масиву. Використання зв'язаного списку приводить до трохи більших накладних витрат по просторі для покажчиків, хоча вставки й видалення можуть бути виконані простіше.

Якщо таблиця є такою великою, що навіть використання індексу не дає достатньої ефективності (або через те, що індекс великий, щоб зменшити послідовний пошук по таблиці, або через те, що індекс усічений, так що сусідні ключі в індексі перебувають далеко друг від друга в таблиці), то може бути використаний індекс другого рівня. Індекс другого рівня діє як індекс для первинного індексу, що вказує на елементи в послідовній таблиці.

Видалення з індексно-послідовної таблиці можуть бути зроблені найбільш простим способом – за допомогою відмітки вилучених записів прапором. При послідовному пошуку по таблиці вилучені записи ігноруються. Відзначимо, що якщо деякий елемент вилучений, але його ключ перебуває в індексі, нічого не треба робити з індексом, а треба відмітити прапором даний елемент первісної таблиці.

Вставка в індексно-послідовну таблицю є більше важкою, оскільки між двома вже існуючими елементами таблиці може не бути вільного місця, що приводить до необхідності пересувати велику кількість

елементів таблиці. Однак, якщо деякий недалеко розташований елемент у таблиці був відзначений прапором як вилучений, то необхідно пересунути тільки кілька елементів, і поверх вилученого елемента може бути записана нова інформація. Це у свою чергу може привести до необхідності зміни індексу, якщо елемент, на який указував деякий елемент індексу, був пересунутий.

Індексно-послідовний пошук можна застосовувати й до неупорядкованої таблиці. Для кожного поля, по якому можливий пошук, створюється індексний файл у вигляді впорядкованої таблиці всього із двома полями: значення ключа (поля, по якому ведеться пошук) і номер запису у вихідній таблиці, що відповідає даному значенню ключа. Тут також можливий багаторівневий ключ: ключ другого рівня містить зкорочений ключ першого рівня й номер запису, що відповідає цьому ключу в індексній таблиці першого рівня.

Зкорочений ключ можна одержувати різними способами залежно від типу даних, по яких виконується пошук. Якщо ключ першого рівня key1 числовий, то зкорочений ключ key2 також числовий, наприклад, key2=key1/100 або key2=key1/1000 (розподіл націло). Якщо ключ key1 типу дати, то ключ key2 може бути числовим, і являти собою, наприклад, рік з дати. Якщо ключ key1 є рядком, то ключ key2 може бути підрядком цього рядка.

3.5 Бінарний пошук

Досить ефективним методом пошуку є пошук за індексом, що організований у вигляді впорядкованого *збалансованого бінарного дерева*. *Упорядкованим* дерево буде в тому випадку, якщо для кожного його вузла всі ліві підвузли містять менші значення, ніж в вузлі, а всі праві підвузли – більші значення. Процедура запису деякого набору даних у вигляді впорядкованого дерева досить проста, але складніше одержати *збалансоване дерево*, тобто дерево, у якого всі гілки однієї довжини: на i-ом рівні міститься 2^i вузлів (нульовий рівень — корінь дерева).

Пошук у такому дереві здійснюється за принципом розподілу відрізка навпіл і в середньому потрібно $\log_2 n$ порівнянь для пошуку елемента.

Додавання й видалення елементів в упорядковане дерево не викликає складностей. Для вставки нового елемента виконується пошук елемента, ліворуч або праворуч від якого потрібно вставити новий елемент. Пошук проводиться від кореня: якщо новий елемент менше поточного елемента дерева, то переходимо від поточного елемента до його лівого підвузла, якщо більше – до правого. Якщо немає того

підвузла, на який потрібно перейти, то новий елемент і стає цим підвузлом.

При видаленні вузла дерева, його місце займає його лівий підвузол, тобто ліве піддерево піднімається на один рівень. Правий підвузол вилученого вузла (з усім своїм поддеревом) приєднується праворуч до самого правого елемента лівого піддерева.

Як видно з наведеного алгоритму додавання й видалення елементів у бінарне дерево, ці процедури порушують збалансованість дерева. Пошук по незбалансованому дереву виконується довше, ніж по збалансованому. Однак, операції редагування, додавання й видалення записів є нормальними операціями по роботі з базою даних. Отже, періодично повинна проводиться процедура балансування бінарного дерева.

3.6 Хеширування

Нехай шуканий запис зберігається в деякій таблиці, і перш, ніж буде знайдений необхідний запис, необхідно організувати перегляд деякої кількості ключів. Організація файлу (послідовна, індексно-послідовна, у вигляді бінарного дерева й т.д.) і порядок, у якому вставляються ключі, визначають те число ключів, яке неодхідно буде перевірити до одержання потрібного ключа. Очевидно, що ефективними методами пошуку є ті методи, які мінімізують число цих порівнянь. В ідеалі ми б хотіли мати таку організацію таблиці, при якій не було б непотрібно ніяких порівнянь.

Якщо кожний ключ повинен бути витягнутий за один доступ, то положення запису усередині такої таблиці може залежати тільки від даного ключа. Воно не може залежати від розташування інших ключів, як це має місце в дереві. Найбільш ефективним способом організації такої таблиці є масив, тобто кожний запис зберігається по деякому конкретному зсуву стосовно базової адреси таблиці. Якщо ключі записів є цілими числами, то самі ключі можуть використовуватись як індекси в масиві.

Розглянемо деякий приклад такої системи. Припустимо, що якась фірма-виробник має файл із найменуваннями виробів фірми, що складається із 100 позицій, причому кожна позиція має унікальний номер із двох цифр. У цій ситуації номера виробів є ключами, які використовуються як індекси в даному масиві.

На жаль, однак, така система не завжди має практичний сенс. Наприклад, припустимо, що фірма має деякий файл виробів, що складає із більше 100 пунктів, і ключ кожного запису є номером виробу із семи цифр. Для застосування прямої індексації з використанням повного семизначного ключа потрібен був би масив з 100 млн. елементів. Ясно, що це привело б до втрати великого простору, оскільки неймовірно, що яка-небудь фірма може мати більше чим кілька тисяч найменувань виробів.

Тому необхідeн деякий метод перетворення ключа в яке-небудь ціле число усередині обмеженого діапазону. В ідеалі в те саме число не повинні перетворюватися два різних ключі. На жаль, такого ідеального методу звичайно не існує. Спробуємо розробити методи, які наближаються до ідеальних, і визначити, які дії треба застосувати, коли ідеальний випадок не досягається.

Розглянемо знову приклад з файлом найменувань виробів фірми, у якому кожний запис задається ключем із семизначного номера виробу. Припустимо, що фірма має менш 1000 найменувань виробів і що для кожного виробу використовується тільки один запис. Тоді для зберігання всього файлу буде досить масиву з 1000 елементів. Цей масив індексується цілим числом у діапазоні від 0 до 999 включно. Як індекс запису про вироб у цьому масиві використовуються три останні цифри номера виробу. Відзначимо, що два ключі, які розташовані близькі один до одного як числа (такі як 4618396 і 4618996), можуть розташовуватися далі друг від друга в цій таблиці, чим два ключі, які значно розрізняються як числа (такі як 0000991 і 9846995).

Функція, що трансформує ключ у деякий індекс у таблиці, називається *хеш-функцией*. Якщо `h` є деякою хеш-функцией, а `key` – деякий ключ, то `h(key)` називається *значенням хеш-функции від ключа* `key` і є індексом, по якому повинен бути поміщений запис із ключем `key`.

Цей метод має один недолік. Припустимо, що існують два ключі `k1` і `k2` такі, що `h(k1)=h(k2)`. Коли запис із ключем `k1` уводиться в таблицю, вона вставляється в позицію `h(k1)`. Але коли хеширується ключ `k2`, потрібна позиція є тією же позицією, у якій зберігається запис із ключем `k1`. Ясно, що два записи не можуть займати ту саму позицію.

Така ситуація називається *колізією при хешируванні* або *зіткненням*. Слід зазначити, однак, що гарною хеш-функцией є така функція, що мінімізує кількість колізій й розподіляє записи рівномірно по всій таблиці. Тому й бажано мати масив з розміром більше, ніж число реальних записів. Чим більше діапазон хеш-функции, тим менш імовірно, що два ключі дадуть однакове значення хеш-функции. Звичайно, при цьому виникає компроміс між часом і простором. Наявність порожніх місць у масиві неефективно з погляду використання простору, але при цьому зменшується необхідність вирішення колізій при хешировании, що, отже, є більше ефективним у сенсі витрат часу.

4 МОДЕЛІ ДАНИХ

Кожна СКБД підтримує ту або іншу *модель даних*. Модель даних визначає правила породження припустимих для даної СКБД 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 СКБД, 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 СКБД 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 СКБД. 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 реляційній 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 СКБД цього класу, які доводиться вирішувати їхнім розроблювачам. Інша проблема виникає, коли потрібно забезпечити інтерфейс СКБД, 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.

4.1 Ієрархічні системи

Основним типом логічної структури, яка підтримується ієрархічними СКБД, 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. 4.1 *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).

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.

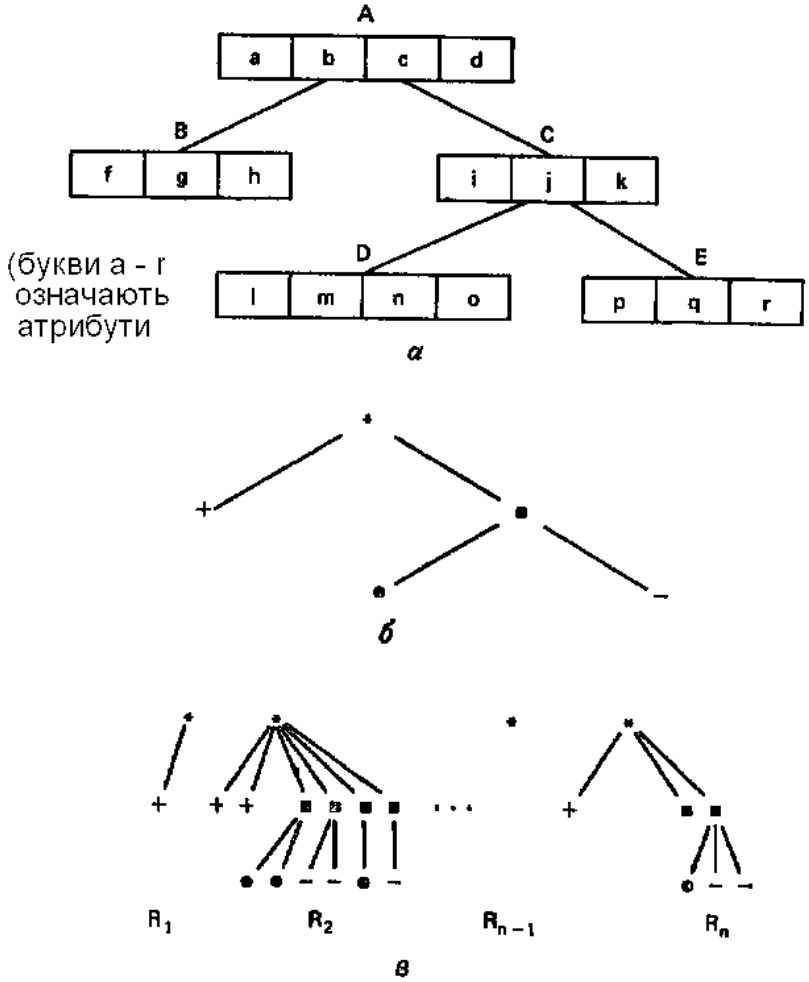


Рис. 4.1. а) ієрархія п'яти типів записів; б) скорочений варіант дерева визначень; в) завантажена база даних

4.1.1 Деякі властивості ієрархічних систем

Ієрархічна структура реалізує відображення N:1 між породженим і вихідним типами записів. Відображення розглядаються в розділі 7.

Це відображення повністю функціонально, тому що дерево не може містити породжений вузол без вихідного вузла (за винятком кореня). Отже, відображення 1:1 і 1:N можуть безпосередньо представлятися ієрархічними структурами. Однак для представлення відображення типу M:N необхідне дублювання дерев.

Наприклад, при побудові ієрархічної схеми для зв'язку ПОСТАЧАЛЬНИКИ–ДЕТАЛІ, якщо один постачальник може поставляти кілька деталей, і одну деталь можуть поставляти декілька постачальників, можливі обидва варіанти, наведені на рис. 4.2.

Рис 4.2. а) дві можливі схеми; б) завантажена база даних (по першій схемі); в) завантажена база даних (по другій схемі)

В ієрархічних СКБД 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.

У базі даних «ПОСТАЧАЛЬНИКИ-ДЕТАЛІ» є відображення M:N між типами записів. Якщо не відомий тип запису, що найчастіше використовується, то з метою загальності й гнучкості необхідно при наявності ресурсів пам'яті зберігати обидва варіанти (рис. 4.2 б і 4.2 в). У противному випадку повинен бути обраний один з варіантів, що може викликати труднощі.

Так, у випадку 4.2 б набагато легше відповістити на запитання про всі деталі, що поставляє даний постачальник, чим на питання про всіх постачальників, що поставляють дану деталь (що легко зробити без перегляду всіх листів у випадку 4.2, б). Отже, в ієрархічних СКБД реалізація складних зв'язків вимагає дублювання даних і/або більших витрат на пошук.

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.

4.2 Мережні системи

Мережні СКБД 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. 4.3 показана мережна схема й відповідна цій схемі база даних.

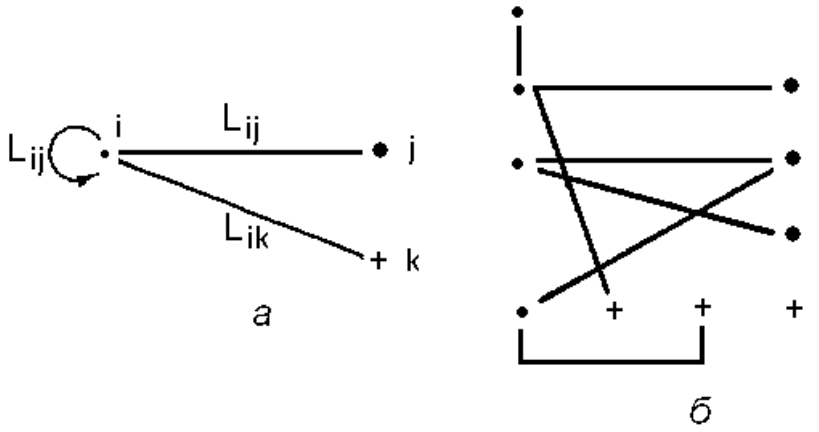


Рис. 4.3 Структура мережної бази даних.
а – схема мережі, 6 – екземпляр запису мережної БД

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.

4.2.1 Мережна модель даних РГБД CODASYL

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).

Для перетворення загальної мережної структури, показаної на рис. 4.2 а, можна ввести запис-зв'язку. Таким чином, зв'язок M:N, прикладом якого є зв'язок між постачальниками й деталями, перетвориться в сукупність зв'язків 1:N (тобто ієрархії), що задовольняють обмеженню а). Екземпляри запису-зв'язки можна вибирати так, щоб задовольнялося обмеження б). Крім того, записи-зв'язки можуть містити додаткову інформацію (наприклад, у записі-зв'язці може зберігатися величина поставки). Ці перетворення показані на рис. 4.4.

Из рис. 4.4 видно, що для перетворення загальної структури в структуру з обмеженнями необхідно ввести кілька типів наборів і типів записів. Це ще один приклад, що показує необхідність дублювання для представлення складних і потужних структур за допомогою примітивних і/або обмежених структур.

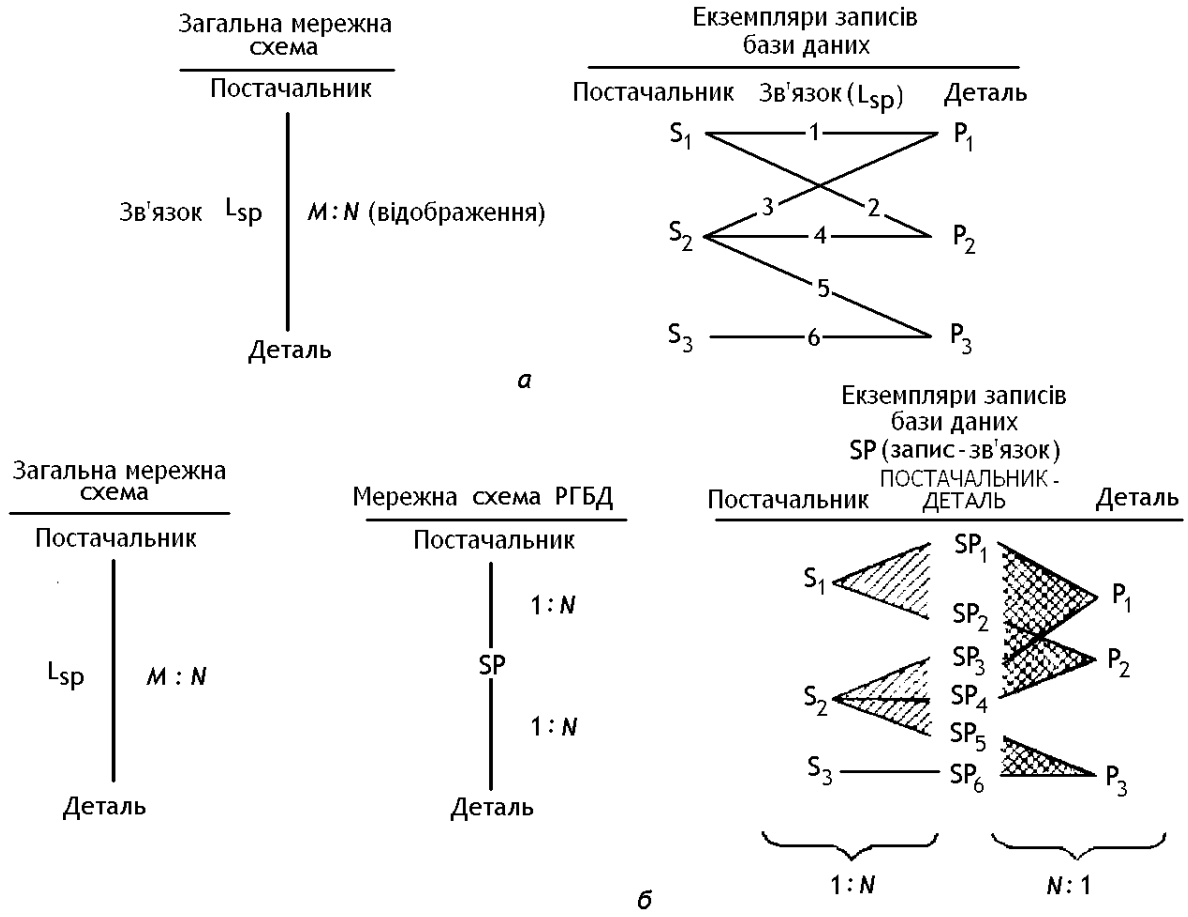


Рис. 4.4 Перетворення в мережну модель РГБД:
а – загальна мережна схема; б – мережна модель РГБД

Що стосується обмеження в), то розглянемо зв'язок «керує», у якому начальник може керувати сам собою. У термінах пропозицій РГБД тип запису-члена тут повинен збігатися з типом запису-власника, що неприпустимо. У цьому випадку за аналогією з рішенням для обмежень а) і б), розглянутий зв'язок можна відобразити, або вводячи різні типи записів для різних ролей того самого типу запису, або записи-зв'язки й два типи наборів.

4.3 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 СКБД використовують реляційну модель даних, за якою структура даних представляється у вигляді прямокутних таблиць, що складаються зі стовпців і рядків. Прямокутні таблиці реляционной бази даних називаються відношенями. 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 СКБД різними виробниками програмного забезпечення.

5 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.

5.1 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 СКБД. 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 СКБД. 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 СКБД;
- створення різних комерційних реляційних СКБД, 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 СКБД 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 СКБД, як для мейнфреймів, так і для персональних комп'ютерів, хоча багато хто з них не повністю відповідають точному визначенню реляційної моделі даних. Прикладами реляційних СКБД для персональних комп'ютерів є СКБД 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 СКБД, система IDMS фірми Computer Associates, тепер називається CA-

IDMS/SQL і підтримує реляційне представлення даних. Іншими СКБД 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.

5.2 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 СКБД практично витиснули СКБД 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 СКБД 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.

5.3 Структура реляційних даних

Відношення. Плоска таблиця, що складається з рядків і стовпців

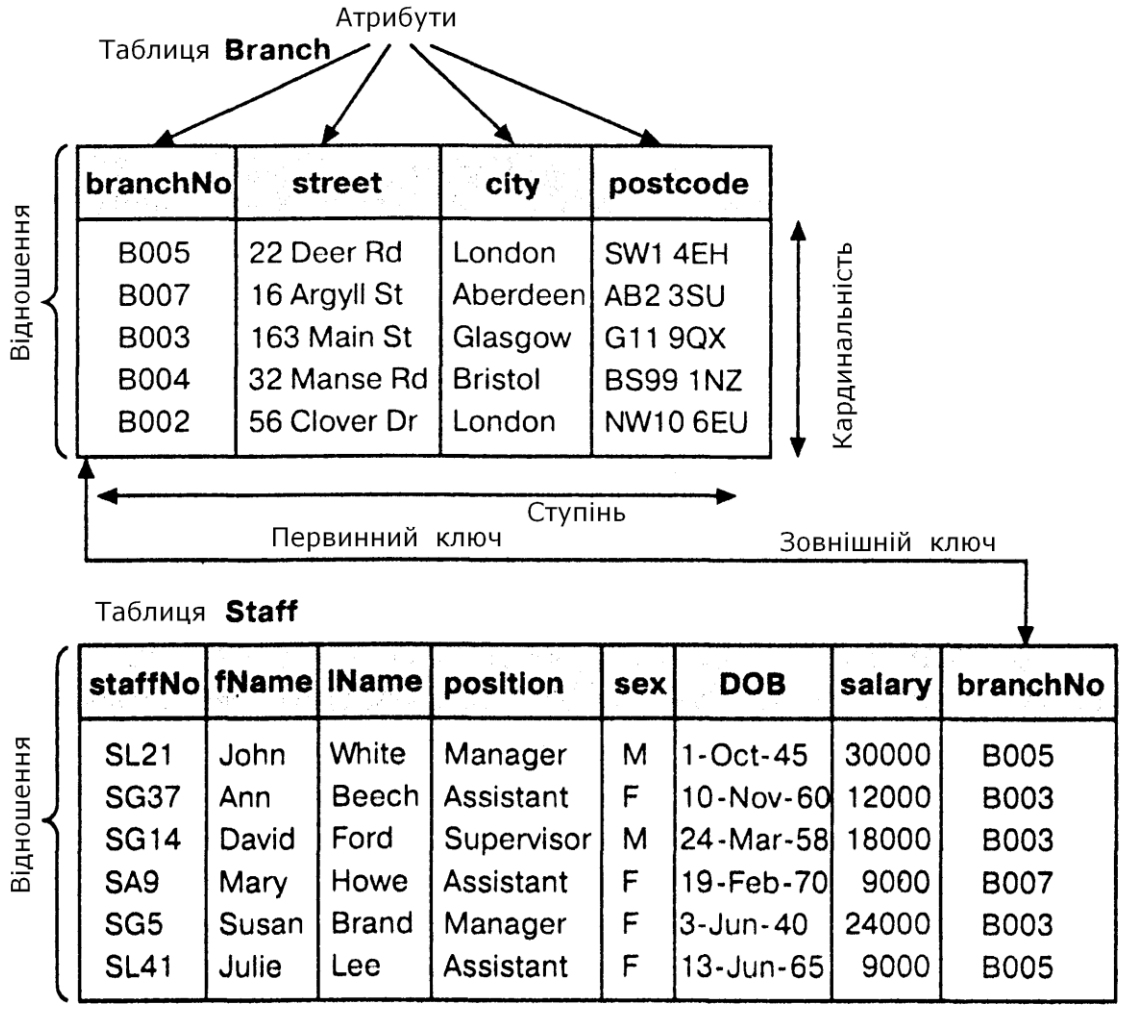
У будь-який реляційній СКБД передбачається, що користувач сприймає базу даних як набір таблиць. Однак варто підкреслити, що це сприйняття відноситься тільки до логічної структури бази даних, тобто до зовнішнього й до концептуального рівням архітектури ANSI-SPARC, що розглядалася нами в розділі 2.4. Подібне 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. 5.1 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.



Branch

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

Рис. 5.1. 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 СКБД вони підтримуються не повністю, а лише частково.

Таблиця 5.1 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.

5.3.1 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 СКБД може зберігати кожне відношення в окремому файлі. У табл. 5.2 показані відповідності, що існують між трьома згаданими вище групами термінів.

Таблиця 5.2. Альтернативні варіанти термінів у реляційній моделі

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

5.3.2 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 $\mathbf{D_1, D_2, ..., D_n}$. 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.

5.3.3 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. 5.1, 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 віднщення.* 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 $\mathbf{R_l, R_2, ..., R_n}$ — набір реляційних схем, то схему реляційної бази даних (або просто реляційну схему) **R** можна представити в такий спосіб:

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

6.3.4 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.

5.3.5 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. 5.1 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.

5.3.6 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 5.3. 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 5.4. 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 5.5. 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 5.6. 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 5.7. 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 5.8. 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 5. 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

5.4 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.

5.4.1 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.

5.4.2 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.

5.5.3 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.

5.5.4 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 СКБД 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 СКБД.

5.6 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 СКБД і їхні переваги й недоліки в порівнянні з файловими системами. У зв'язку з наявністю зазначених вище функціональних можливостей СКБД є надзвичайно корисним інструментом. Але оскільки для кінцевих користувачів не має значення, наскільки проста або складна внутрішня організація системи, можна почути заперечення, що СКБД 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 СКБД 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 СКБД. Наприклад, у СКБД 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 СКБД 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 СКБД припускає майже стовідсоткову надійність і готовність навіть при збоях в апаратному й програмному забезпеченні. Програмне забезпечення СКБД 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 СКБД повинна розвиватися, причому згодом їй, імовірно, буде потрібно виконувати якісь нові функції, а тому функціональна частина СКБД ніколи не буде незмінною.

5.6.1 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.

5.6.2 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 СКБД 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.

5.6.2 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.

6 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.

6.1 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. 6.1.

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 $\mathbf{A = (a_1, a_2, \ldots, a_n)}$ і $\mathbf{B = (b_1, b_2, \ldots, b_m)}$ відповідно.

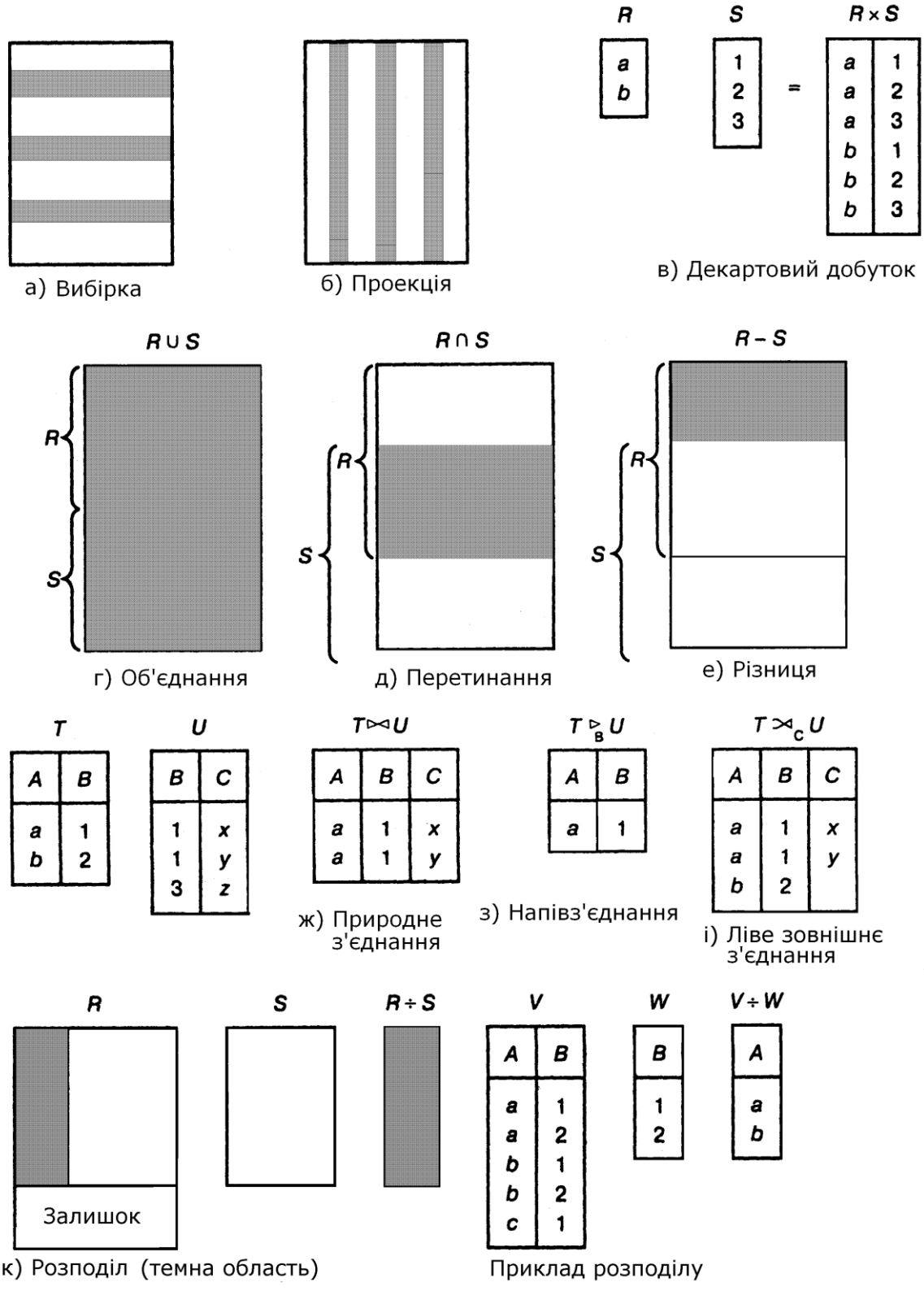


Рис. 6.1 Схематичне подання результатів операцій реляційної алгебри

П'ять операцій є основними: проекція, об'єднання, різниця, декартовий добуток і селекція (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).

6.1.1 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 6.1 на операцію вибірки (селекції)

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 6.2 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. 6.1. Більш складні предикати можуть бути створені за допомогою логічних операцій ∧ (AND), ∨ (OR) і ~ (NOT).

Таблиця 6.1. 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

$\Pi a_1,...a_n(R)$. 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 6.3 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 6.3 на операцію проекції

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

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

У цьому прикладі операція проекції визначає нове відношення, що буде містити тільки атрибути `staffNo`, `fName`, `lName` і `salary` відношення `Staff`, розміщені в зазначеному порядку. Результат виконання цієї операції показаний у табл. 6.2.

Таблиця 6.2. Проекція відношення `Staff` по атрибутах `staffNo`, `fName`, `lName` і `salary`

staffNo	fName	lName	salary
SL21	John	White	30000
SG37	Ann	Beech	12000
SG14	David	Ford	18000
SA9	Mary	Howe	9000
SG5	Susan	Brand	24000
SL41	Julie	Lee	9000

6.1.2 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.

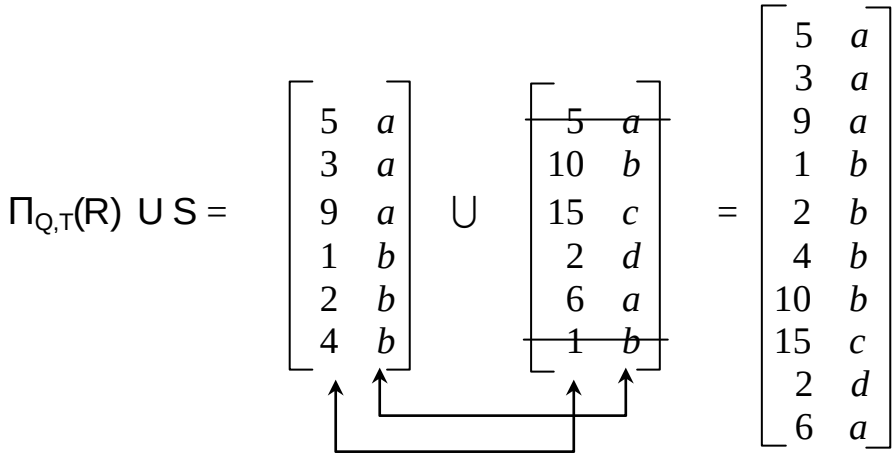
Об'єднання

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 6.5 на операцію об'єднання

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



Приклад 6.6 Операція об'єднання

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

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

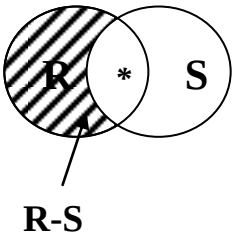
Для створення сумісних за об'єднанням відношень спочатку варто застосувати операцію проекції, щоб виділити з відношеннь `Branch` і `PropertyForRent` стовпці з атрибутами `city`, видаляючи в разі необхідності дублікати. Потім для комбінування отриманих проміжних відношень потрібно використати операцію об'єднання. Результат виконання всіх цих дій наведений у табл. 6.3.

Таблиця 6.3. 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

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

Приклад 6.7 операція різниці

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

Приклад 6.8 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 6.5 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. Результат виконання цих операцій показаний у табл. 6.4.

Таблиця 6.4. Різниця, заснована на атрибуті `city` відношень `Branch` і `PropertyForRent`

city
Bristol

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 6.9 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.

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

Таблиця 6.5. Перетинання, засноване на атрибуті `city` відношень `Branch` і `PropertyForRent`

city
Aberdeen
London
Glasgow

Приклад 6.10 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*** x ***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 6.11 на операцію декартова добутку.

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

$$\Pi_{M,T}(R) = \begin{bmatrix} x & a \\ y & a \\ z & a \\ w & b \end{bmatrix}$$ (див. приклад 6.3)

$$\Pi_{Q,T}(R) \cap S = \begin{bmatrix} 5 & a \\ 1 & b \end{bmatrix}$$ (див. приклад 6.10)

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 6.12 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. 6.6. У такому вигляді це відношення містить більше інформації, ніж необхідно. Наприклад, перший кортеж цього відношення містить різні значення атрибута `clientNo`.

Таблиця 6.6. 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	
CR56	Aline	Stewart	CR56	PA14	Too small (Занадто мала)
CR56	Aline	Stewart	CR76	PG4	Too remote (Занадто далеко)
CR56	Aline	Stewart	CR56	PG4	
CR56	Aline	Stewart	CR62	PA14	No Dining room (Немає окремої їдальні)
CR56	Aline	Stewart	CR56	PG36	
CR74	Mike	Ritchie	CR56	PA14	Too small (Занадто мала)
CR74	Mike	Ritchie	CR76	PG4	Too remote (Занадто далеко)
CR74	Mike	Ritchie	CR56	PG4	
CR74	Mike	Ritchie	CR62	PA14	No Dining room (Немає окремої їдальні)
CR74	Mike	Ritchie	CR56	PG36	
CR62	Mary	Tregear	CR56	PA14	Too small (Занадто мала)
CR62	Mary	Tregear	CR76	PG4	Too remote (Занадто далеко)
CR62	Mary	Tregear	CR56	PG4	
CR62	Mary	Tregear	CR62	PA14	No Dining room (Немає окремої їдальні)
CR62	Mary	Tregear	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.

$$\sigma_{Client.client0=Viewing.clientNo}((\Pi_{clientNo,\ fName,\ lName}(Client)) \times (\Pi_{clientNo,\ propertyNo,\ comment}(Viewing)))$$

Результат виконання цієї операції показаний у табл. 6.7. Як ми незабаром побачимо, комбінація декартового добутку й вибірки може бути зведена до однієї операції *з'єднання.*

Таблиця 6.7 Обмежене декартово 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)

6.1.3 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:

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

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

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

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

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

6.1.4 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 СКБД 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 6.13 Тета-з'єднання $R \triangleright\triangleleft_{R.Q>S.A} S$

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 6.12 (див. приклад 6.14).

Приклад 6.14. 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 6.12 для одержання цього списку використовувались операції декартова добутку й вибірки. Однак той же самий результат можна одержати за допомогою операції з'єднання за еквівалентністю.

$(\Pi_{clientNo,fName,Lname}(Client))$
$\bowtie_{Client.clientNo=Viewing.clientNo} (\Pi_{clientNo,propertyNo,commtnt}(Viewing))$

або

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

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

$Result \leftarrow TempClient \bowtie_{TempClient.clientNo=TempViewing.clientNo} TempViewing$

Природне з'єднання

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

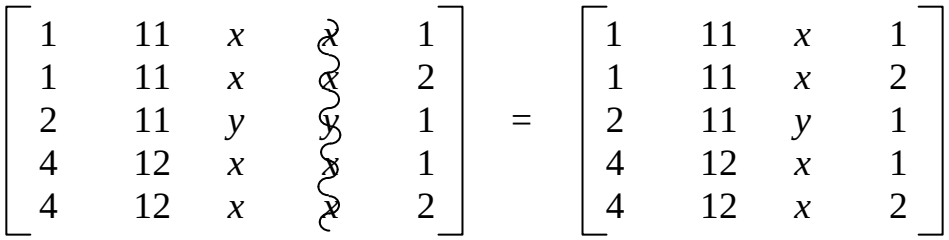
Ступенем природного з'єднання називається сума ступенів операндів-відношень R і S за винятком кількості атрибутів х.

Приклад 6.15. Природне з'єднання $P \bowtie \rho_{Q(D_3,D_5)}(Q)$

У цьому випадку кожний кортеж відношення Р буде з'єднуватися з тим кортежем відношення Q, у якому значення атрибута D_4 дорівнює значенню атрибута D_3 у відношенні Р. 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:



Приклад 6.16. Операція природного з'єднання

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

У прикладі 6.14 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.

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

або

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

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

`Result ← TempClient ⋈ TempViewing`

Результат цих операцій показаний у табл. 6.8.

Таблиця 6.8. 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 $\supset\!\bowtie$ 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 СКБД 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 6.17. 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. 6.9.

Таблиця 6.9. 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)

property No	street	city	client No	viewDate	comment
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 6.17 показане ліве зовнішнє з'єднання, оскільки в результуючому відношенні втримуються всі кортежі лівого відношення. Існує також праве зовнішнє з'єднання, яке називаэться 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 6.18. 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. 6.10.

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

Таблиця 6.10. 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

6.1.5 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:

$$T_1 \leftarrow \Pi_C(R)$$

$$T_2 \leftarrow \Pi_C((S \times T_1) - R)$$

$$T \leftarrow T_1 - T_2$$

Приклад 6.17. Операція розподілу відношень

Створіть список всіх орендарів, які оглядали об'єкти нерухомості із трьома кімнатами.

Для рішення поставленї задачі спочатку треба за допомогою операції вибірки виконати пошук всіх трикімнатних об'єктів нерухомості, а потім за допомогою операції проекції одержати відношення, що містить номери тільки цих об'єктів нерухомості. Після цього потрібно застосувати наведену нижче операцію розподілу й одержати нове відношення, як показано в табл. 6.11-6.13.

$$(\Pi_{clientNo,\ propertyNo}(Viewing)) \div (\Pi_{propertyNo}(\sigma_{room=3}(PropertyForRrent)))$$

Таблиця 6.11. Результат застосування операції проекції $\Pi_{\texttt{clientNo,propertyNo}}(\texttt{Viewing})$ до відношення `Viewing`

clientNo	propertyNo
CR56	PA14
CR76	PG4
CR56	PG4
CR62	PA14
CR56	PG36

Таблиця 6.12. Результат застосування до відношення `PropertyForRrent` операції вибірки $\Pi_{\text{propertyNo}}(\sigma_{\text{room=3}}(\text{PropertyForRrent}))$

clientNo
PG4
PG36

Таблиця 6.13. Результат застосування операції розподілу до результатів двох попередніх операцій

clientNo
CR56

6.1.6 Короткий перелік операцій реляційної алгебри

Основні відомості про операції реляційної алгебри наведені в табл. 6.14.

Таблиця 6.14 Операції 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	$\sigma_{\text{предикат}}(R)$	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

Операція	Позначення	Область застосування
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 ті 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:

$$(\forall B)\ (B.city \neq \text{'Paris'})$$

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:

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

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

Змінні кортежу називаються *вільними змінними*, якщо вони не кваліфікуються кванторами ∀ або ∃; 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 єдиної вільної змінної є S і в процесі обчислення цього виразу послідовно відбувається зв'язування змінної S с кожним кортежем відношення Staff.

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

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.

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

Атрибут `staffNo` у відношенні `PropertyForRent` містить табельний номер того співробітника, що відповідає за роботу з даним об'єктом нерухомості. Цей запит можна сформулювати інакше: "Для всіх співробітників, дані про які потрібно привести в списку, у відношенні `PropertyForRent` є кортежі, що відповідають цим співробітникам,

причому значення атрибута city у кожному такому кортежі дорівнює 'Glasgow'".

Зверніть увагу, що в такому формулюванні запиту немає ніякої вказівки на стратегію виконання запиту — СКБД надається свобода вибору операцій для виконання даного завдання, а також порядку виконання цих операцій. Еквівалентне формулювання даного запиту в реляційній алгебрі буде виглядати так: "Вибрати такі кортежі відношення PropertyForRent, у яких значення атрибута city дорівнює 'Glasgow', і виконати їхнє об'єднання з відношенням Staff. Як бачите, тут порядок виконання операцій задається неявно, але цілком однозначно.

В. Створіть список всіх співробітників, які в цей момент не працюють із об'єктами нерухомості.

{S.fName,S.lName|Staff(S) ∧ (~(∃P)(PropertyForRent(P) ∧ (S.staffNo=P.staffNo)))}

Використовуючи правила еквівалентності логічних операцій, цей вираз можна переписати так:

{S.fName,S.lName|Staff(S) ∧ ((∀P)(~PropertyForRent(P)v ~(S.staffNo=P.staffNo)))}

Г. Створіть список всіх клієнтів, що оглядали об'єкти нерухомості в місті Глазго.

{C.fName,C.lName | Client(C) ∧ ((∃V)(∃P)(Viewing(V) ∧ PropertyForRent(P) ∧ (C. clientNo=V.clientNo) ∧ (V.propertyNo=P.propertyNo) ∧ (P.city='Glasgow')))}

При відповіді на цей запит варто враховувати, що формулювання "клієнти, які оглядали будь-який об'єкт нерухомості в місті 'Glasgow'", можна замінити формулюванням "клієнти, для яких у відношенні Viewing є дані про огляд ними об'єктів нерухомості в місті 'Glasgow'".

Д. Створіть список всіх міст, у яких є відділення компанії, але немає орендованих об'єктів нерухомості.

{ B. city | Branch(B) ∧ (~(∃P)(PropertyForRent(P) ∧ (B. city=P.city)))}

Порівняєте цей вираз з еквівалентним виразом реляційної алгебри, наведеним у прикладі 6.8.

Е. Створіть список всіх міст, у яких є відділення компанії, а також є щонайменше один орендований об'єкт нерухомості.

$$\{B.city \mid Branch(B) \wedge ((\exists P)(PropertyForRent(P) \wedge (B.city=P.city)))\}$$

Порівняєте цей вираз з еквівалентним виразом реляційної алгебри, наведеним у прикладі 6.9.

Ж. Створіть список всіх міст, у яких є або відділення компанії, або хоча б один орендований об'єкт нерухомості.

У прикладі 6.6 для подання цього запиту засобами реляційної алгебри застосовувалася операція об'єднання. Але, на жаль, даний запит не може бути представлений у тій версії реляційного вирахування, що розглядається в даному курсі.

7 ПРОЕКТУВАННЯ БАЗ ДАНИХ

7.1 Проблема проектування баз даних

Проектування без даних являє собою тривалий і трудомісткий процес. Проектування великих баз даних, що включають 500 і більше елементів, звичайно триває кілька місяців. Дотепер проектування баз даних залишається скоріше мистецтвом, ніж наукою.

В останні роки розвиваються CASE-засоби для автоматизації процесу проектування баз даних. Слід зазначити, що ці засоби, безумовно, прискорюють роботу проектувальника, але потребують від останнього досить глибоких знань по теорії баз даних, глибокого пророблення вихідного матеріалу, системного аналізу предметної області, для якої проектується база даних. Докладно про ці засоби розповідається в курсі «Системний аналіз і проектування інформаційних систем».

Найбільш важливими аспектами баз даних є цілісність і узгодженість даних; не повинне бути випадкових втрат або руйнувань даних, і, крім того, дані, що повторюються, повинні відповідати одному рівню відновлення для того, щоб користувач одержував ті дані, які йому необхідні. До важливих критеріїв якості баз даних відносяться забезпечення захисту й таємності даних: дані повинні бути захищені від несанкціонованого доступу. База даних повинна мати здатність до розширення й можливістю забезпечення до даних вимог, що змінюються,.

Один з основних наслідків поганої структури бази даних – неприпустимо великі витрати на реструктуризацію існуючої бази даних, яку необхідно виконувати відразу ж після виявлення її неспроможності. Тому проектувальник повинен усе робити правильно з першого разу! Зміни фізичної структури (методу доступу, розміру блоку, груп наборів даних, використовуваних покажчиків і т.д.) звичайно в меншому ступені торкаються готового проекту. У цих випадках прикладні програми не змінюються, хоча реорганізацію бази даних, як правило, виконувати доводиться. Зміни деяких параметрів проектування, таких, наприклад, як правила включення, видалення й зміни сегментів, розміру бази даних, фізичне розміщення індексів і т.д., не приводять до зміни прикладних програм і реорганізації баз даних. Однак логіка прикладних програм може бути чутлива до кожного з перерахованих параметрів, і тому для забезпечення необхідних функцій при зміні бази даних потрібна зміна прикладних програм.

Звичайно проектуванню ефективних баз даних заважає наступне:

- недостатньо глибокий аналіз вимог до даних (у тому числі семантики імен і взаємозв'язків даних);

- більша тривалість процесу структурування, що робить цей процес стомлюючим і важко здійсненним при ручній обробці;
- труднощі, пов'язані з обліком при проектуванні продуктивності системи;
- часові обмеження, обумовлені використовуваними технічними засобами.

7.2 Багаторівневе проектування баз даних

У відповідність із багаторівневою архітектурою баз даних, розглянутої в розділі 2.4, при проектуванні баз даних також розрізняють три рівні або три етапи проектування бази даних: концептуальний, логічний і фізичний.

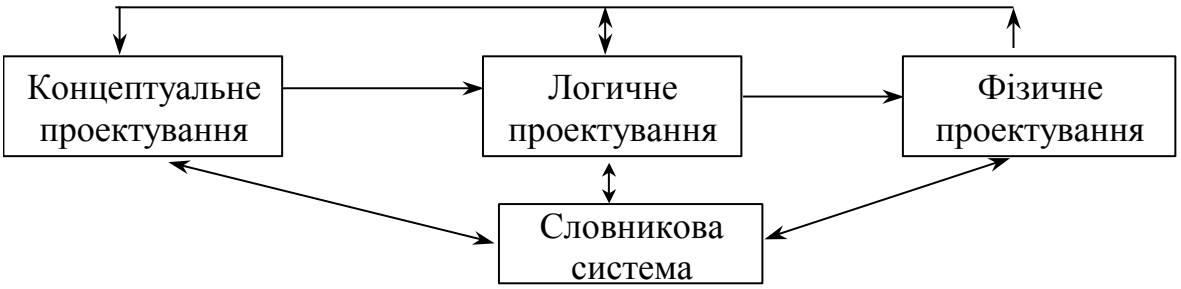


Рис. 7.1 Зв'язок між етапами проектування БД

На рисунку 7.1 представлена сукупність процедур проектування бази даних. Процес включає три самостійних етапи.

На етапі *концептуального проектування* здійснюються збір, аналіз і редагування вимог до даних. У процесі *логічного проектування* вимоги до даних перетворяться в структури (наприклад, у вигляді сегментів і ієрархій або таблиць) системи керування базами даних, що використовується. На етапі *фізичного проектування* вирішуються питання, пов'язані із продуктивністю системи, визначаються структури зберігання даних і методи доступу.

Кожний етап проектування розглядається як сукупність ітеративних процедур, у результаті виконання яких одержують модель. Ми будемо говорити про створення концептуальної моделі на етапі концептуального проектування, логічної моделі на етапі логічного проектування й фізичної моделі на етапі фізичного проектування. Весь процес характеризується як ітеративний. Результати проектування, отримані на етапі логічного або фізичного проектування, можуть потребувати зміни або повторення ітерацій для забезпечення необхідних властивостей системи, що проектується.

На рисунку 7.1 визначена також взаємодія між етапами проектування й словниковою системою. Однак розглянуті процедури проектування можуть використатися незалежно від словникової системи, якщо вона, наприклад, відсутня у проектувальника.

Незважаючи на те, що багато процедур процесу проектування автоматизовані, передбачається активна участь у ньому людини. Ітеративні процедури вимагають наявності діалогу між проектувальниками й кінцевими користувачами, а добре продумана автоматизація сприяє розумінню виникаючих при проектуванні проблем, які обговорюються й вирішуються в процесі діалогу.

Концептуальне проектування інваріантно стосовно структури бази даних, і тому методи концептуального проектування можуть бути застосовані для проектування баз даних, що керуються будь-якою СКБД. Процес логічного проектування в основному також інваріантний до СКБД, що використовується, але отримана на етапі логічного проектування результуюча модель даних повинна бути орієнтована на конкретні структури.

7.3 Зв'язки, відображення й асоціації

Завдання проектувальника на етапі концептуального проектування полягають у визначенні локальних представлень для всіх функцій, які будуть використовувти розроблену базу даних.

Існують два основних джерела одержання локальних представлень: функціональні вимоги й внутрісистемні вимоги.

Функціональні вимоги являють собою вимоги до даних кожної функції, що буде використовувати базу даних. У процесі визначення цих вимог елементи даних кожної функції ідентифікуються за допомогою різного типу асоціацій між парами зв'язаних елементів.

Концептуальне проектування розглядається в курсі “Системний аналіз и проектування інформаційних систем”. В даному курсі ми розглянемо деякі аспекти логічного проектування реляційних систем, а саме процесс *нормалізації*. Для проведення процесу нормалізації і перевірки результату цього процесу на відсутність помилок необхідне знання всіх зв'язків між елементами бази даних.

7.3.1 Зв'язки

При проектуванні бази даних необхідно розглядати не тільки дані, які повинні в них зберігатися, але й зв'язки між ними. В одних СКБД (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.

7.3.2 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. 7.2 показаний приклад відображення 1:1 для елементів даних НОМЕР_СЛУЖБОВЦЯ й ІДЕНТИФІКАЦІЙНИЙ_КОД. Кожний з них унікально ідентифікує іншої.

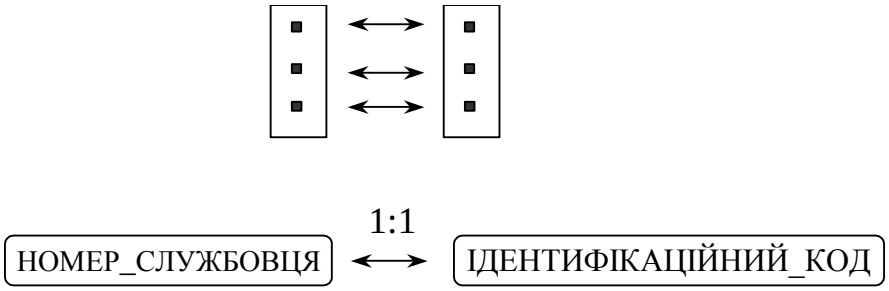


Рис. 7.2 Відображення 1:1.

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. 7.3 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).

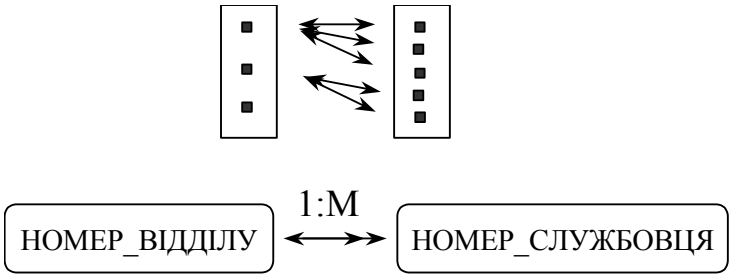


Рис. 7.3 Відображення 1 : M.

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.

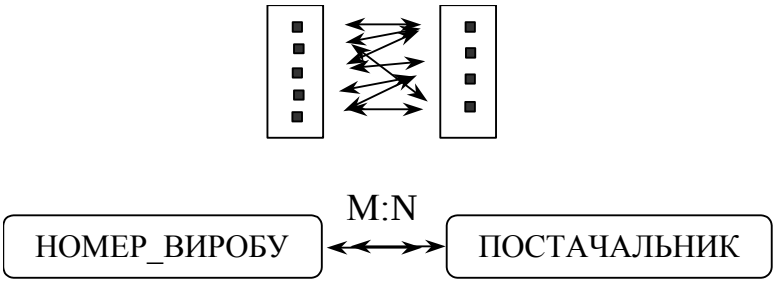


Рис. 7.4 Відображення M : N.

7.3.3 Асоціації

Відображення описують двосторонні зв'язки між парами зв'язаних елементів даних. Однобічні зв'язки між парами зв'язаних елементів даних можна представляти за допомогою асоціацій. Асоціації між парою зв'язаних елементів, визначені в обидва боки, являють собою відображення.

Часто доцільно визначати взаємозв'язки елементів даних за допомогою однобічних асоціацій, а іноді за допомогою відображень; процес структурування елементів даних повинен ураховувати цю особливість. Наприклад, для одного користувача може не мати значення характер інверсних асоціацій, і їхній довільний вибір може привести до протиріч із реальними інформаційними потребами інших користувачів. Асоціації використовуються також для визначення зв'язку ключ-атрибут,

оскільки в більшості випадків зв'язки від атрибутів до ключів не визначаються. Існують три типи асоціацій: асоціація типу 1 (проста), типу М (складна), типу С (умовна).

Асоціація типу 1 (проста)

Екземпляр елемента даних, від якого спрямований зв'язок, ідентифікує один і тільки один екземпляр елемента даних, до якого зв'язок спрямований. Ідентифікація є унікальною (атомарної) і визначає функціональну залежність. Приклади асоціації типу 1 наведені на рис. 7.5: між елементами даних НОМЕР_СЛУЖБОВЦЯ й ІДЕНТИФІКАЦІЙНИЙ_КОД і між елементами даних НОМЕР_СЛУЖБОВЦЯ й НОМЕР_ВІДДІЛУ. Для службовця визначений тільки один ідентифікаційний код, і він працює тільки в одному відділі. Зв'язку у зворотному напрямку не розглядаються.

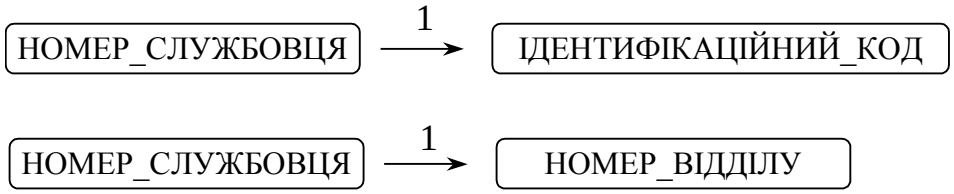


Рис. 7.5 Асоціація 1

Асоціація типу М (складна)

Екземпляр елемента даних, від якого спрямований зв'язок, ідентифікує деяке число (нуль, один або декілька) екземплярів елемента даних, до якого спрямована зв'язок.

Ідентифікація не обов'язково є унікальною і являє собою багатозначну залежність. На рис. 7.6 наведені приклади асоціації типу М між елементами даних НОМЕР_ВІДДІЛУ й НОМЕР_СЛУЖБОВЦЯ й НОМЕР_ВИРОБУ й ПОСТАЧАЛЬНИК. У даному відділі можуть працювати багато службовців, і даний виріб може поставлятися багатьма постачальниками. Зв'язки у зворотному напрямку не розглядаються.

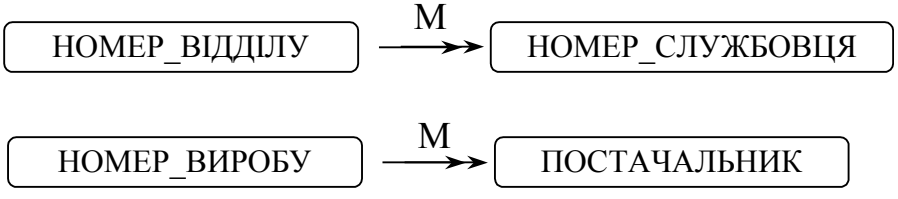


Рис. 7.6 Асоціація М

Асоціація типу С (умовна)

Для даного екземпляра елемента даних, від якого спрямований зв'язок, може не існувати відповідного екземпляра елемента даних, до якого зв'язок спрямований, але якщо вона існує, то відноситься до єдиного

екземпляра елемента даних. Ідентифікація, якщо існує, є унікальною. Прикладом асоціації типу С можуть служити асоціації для елементів даних НОМЕР_СЛУЖБОВЦЯ й ДАТА_ЗВІЛЬНЕННЯ й для елементів даних ЛІКАРНЯНЕ_ЛІЖКО й ХВОРИЙ. Службовець може звільнитися чи ні з даного відділу, але якщо звільнився, то дата звільнення буде тільки одна. На лікарняному ліжку може перебувати, а може й не перебувати хворий, але якщо він є, то обов'язково один на ліжку.

7.4 Основні математичні поняття

Нехай є *множини* А и В. Відношення А **R** У вказує на зв'язок між окремими елементами цих множин. Розрізняють рефлексивні відношення А **R** А (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:М) і багато-до-багатьох (М: N).

Таким образом, зв'язок – це відповідність або відображення між елементами двох (або більше) множин. У таблиці 7.1 наведені деякі приклади сутностей, їхніх властивостей і відповідного подання за допомогою типів записів і атрибутів.

Таблиця 7.1 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. 7.7 показані різні зв'язки між екземплярами різних множин у порядку, що відповідає пунктам а) - д) наведеного вище списку.

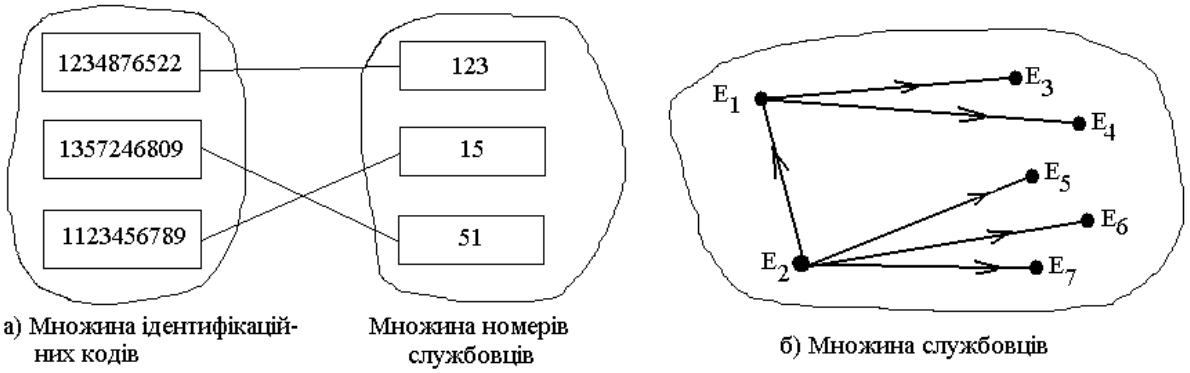


Рис. 7.7 Зв'язки між множинами сутностей: а – відповідність значення; б – зв'язок «керує»;

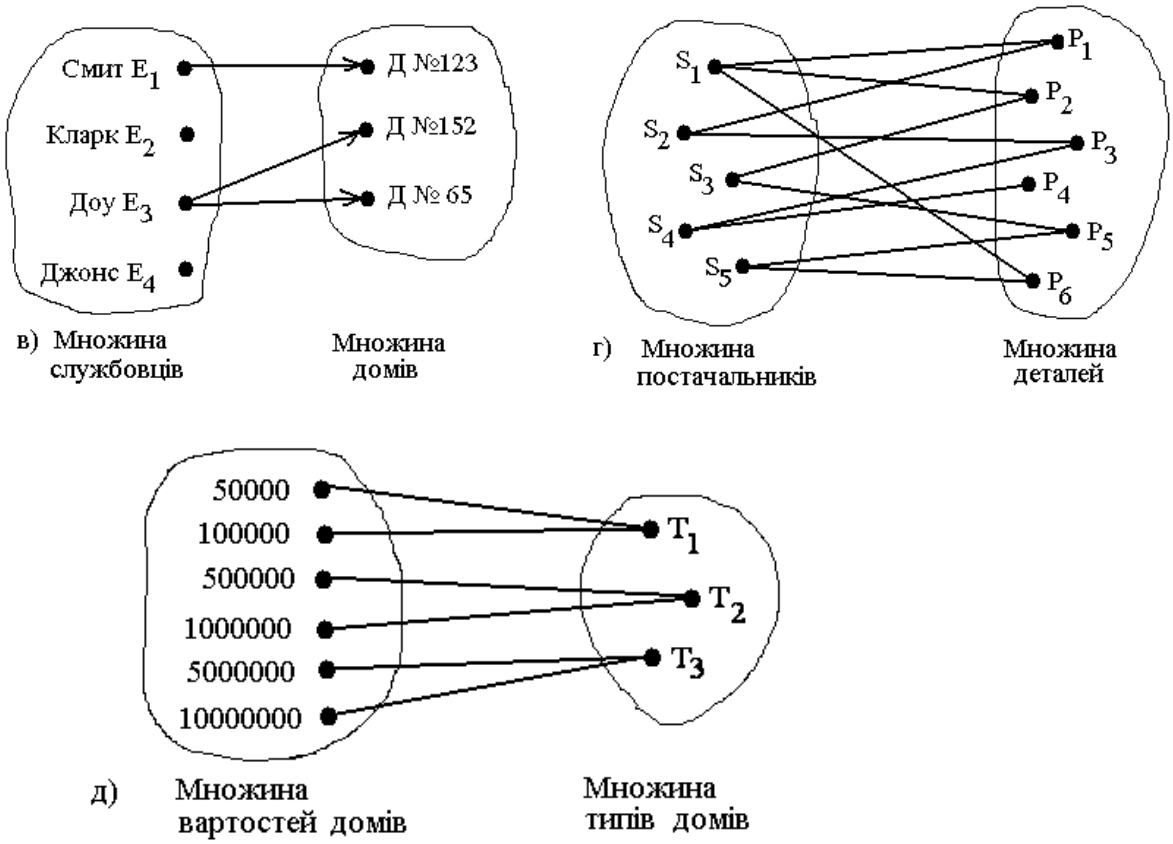


Рис. 7.7 Зв'язки між множинами сутностей: в – зв'язок «володіє»; г –- зв'язок «поставляє»; д – зв'язок між типами будинків і їхньою вартістю

Досліджуючи типи зв'язків на рис. 7.7, можна зробити наступні зауваження:

1. На рис. 7.7, *а* й *д* представлені зв'язки між множинами атрибутів, які називаються також *міжатрибутними* зв'язками. Ці зв'язки мають внутрісутнісний тип.
2. На рис. 7.7, 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. 7.7, а наведений приклад відображення 1:1 між відповідними множинами. Це приклад взаємно однозначного відображення.

4. На рис. 7.7, у показано, що не всі службовці володіють будинками, а деякі власники можуть мати більше одного будинку.
5. З рис. 7.7, 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. 7.7, 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. 7.7, 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. 7.6, то можна зробити наступні виводи:

На рис. 7.7,*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. 7.7, *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.

7.5 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 "$\subseteq$" 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 $A,E \to D$, якщо $A \to B$ й $B \to C,D$? (Передбачається, що E не є підмножиною A)

Рішення

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

2. Внаслідок проективности $A \to D$.

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

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→→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 $X \to\to Y$, тоді $X \to\to U - X - Y$. Це правило не має аналога для функціональних залежностей.

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

8 ТЕОРІЯ НОРМАЛЬНИХ ФОРМ

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.

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 $A \to B$ 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 непервнннних 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.

8.1 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

8.2 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.

8.3 Третя нормальна форма

Розглянемо транзитивну залежність наступного типу:

Якщо $A \rightarrow B$, B не візначає 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)

8.4 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#)

8.5 Четверта нормальна форма (4НФ)

Ця нормальна форма виключає нефункціональні багатозначні залежності. Як видно з наведеного нижче приклада, при проектуванні бази даних подібних залежностей можна уникнути, якщо додержуватися правила: варення - окремо, цвяхи - окремо. Адже справді, наявність дітей у професора, їхня кількість, а тим більше їхні імена не визначають курси, що він читає в університеті. І навпаки.

Проілюструємо нефункціональні багатозначні залежності на прикладі відношення ПРОФЕСОР (ІД#, **ДІТИ, КУРСИ,** ПОСАДА), що містить дані про дітей професора, курсах, що він читає, і його посаді. Між професором і дітьми є зв'язок типу 1 : М, а між професором і курсами зв'язок M:N, якщо припустити, що деякі курси можуть читати декілька викладачів. Нехай екстенсіонал відношення має такий вигляд:

ІД#	ДІТИ	КУРСИ	ПОСАДА
525-111	Джон	ДО410	Доцент
525-111	Кэт	ДО412	Доцент
525-111	Джон	ДО412	Доцент
525-111	Кэт	ДО410	Доцент
340-055	Джек	ДО410	Асистент

Якщо оголошується багатозначна залежність атрибутів ДІТИ або КУРСИ від атрибута ІД# (номер викладача), кожному значенню атрибута ІД# повинна відповідати фіксована множина значенні атрибутів ДІТИ або КУРСИ відповідно. Інакше кажучи, можлива зміна значення цих атрибутів у будь-якому рядку відношення. Заміна значення атрибута КУРСИ в кортежі <525-111 **Кэт** ДО412 Доцент> дасть кортеж <525-111 Кэт ДО410 Доцент>. Заміна значення атрибута ДІТИ на Джон дасть кортеж <525-111 Джон ДО412 Доцент>. (Порядок заміни відповідає порядку попереднього твердження.) Обидва отриманих кортежу вже є у відношенні. Таким чином, інші значення кортежів ніяк не зв'язані зі значеннями багатозначних атрибутів. Отже, має місце ІД# →→ДІТИ й ІД# →→КУРСИ. Для наявності у відношенні багатозначної залежності необхідно мати мінімум три атрибути: ключ і незалежні атрибути, яких не може бути менше двох (щоб бути незалежними один від одного!).

Можна перевірити правило доповнення (правила виводу для багатозначних залежностей, розділ 7.5) на розглянутому нами прикладі. Якщо врахувати, що функціональна залежність є багатозначної, можна вивести зв'язок між атрибутами ІД# і ПОСАДА.

Відношення перебуває *в 4НФ*, якщо воно перебуває в НФБК, але в ньому відсутні багатозначні залежності, які не є функціональними.

Для іншого визначення 4НФ дамо визначення тривіальної багатозначної залежності. Для атрибутів А і В відношення залежність $X \to\to U-X-Y$ (де U – множина всіх атрибутів відношення) є тривіальною. Також тривіальною є залежність $X \to\to 0$.

Відношення перебуває *в 4НФ*, якщо у відношенні для будь-якої нетривіальної багатозначної залежності *Y* від *X* (тобто $X \to\to Y$) *X* обов'язково містив ключ відношення.

Наступні відношення перебувають в 4НФ:

*R*1(ІД#,**ДІТИ**)

R2 (ІД#, **КУРСИ)**

*R*3 **(**ІД#, ПОСАДА)

Четверта нормальна форма показує, що відношення може перебувати в НФБК і проте можуть існувати деякі аномалії, особливо при відновленнях. Наприклад, якщо в професора з'явиться ще одна дитина, у відношення необхідно додати не один кортеж, а стільки, скільки професор читає курсів. (Аналогічна ситуація виникає з появою нового курсу, що читає професор.) Ці численні модифікації необхідні для збереження незалежності між всіма можливими значеннями атрибутів.

8.6 П'ята нормальна форма 5НФ (проекція/з'єднання)

Той факт, що відношення може бути відновлене без втрат з'єднанням деяких його проекцій, відомий як *залежність за з'єднанням*.

Відношення перебуває в 5НФ тоді й тільки тоді, коли будь-яка залежність за з'єднанням в R визначається можливими ключами R.

Інакше кажучи:

Відношення перебуває в 5НФ тоді й тільки тоді, коли кожна проекція R містить не менш одного можливого ключа й принаймні один непервинний атрибут.

Розходження 5НФ і 4НФ можна показати на прикладі. Нехай є відношення:

R_1 (П#, Д#, ОТД)			R_2(П#, Д#)		R_3(Д#, ОТД)		R_4(П#, ОТД)	
П1	Д1	А	П1	Д1	Д1	А	П1	А
П1	Д1	В	П2	Д1	Д1	В	П1	В
П2	Д1	А	П2	Д2	Д2	А	П2	А
П2	Д2	В	П3	Д1	Д2	В	П2	В
П3	Д1	А	П3	Д2			П3	А
П3	Д1	В					П3	В
П3	Д2	А						
П3	Д2	В						

У відношенні R_1 відсутні незалежні багатозначні залежності, і воно складається тільки з первинних атрибутів (є «повністю ключовим»); отже, воно перебуває в 4НФ. Відношення R_2, R_3 і R_4 перебувають в 5НФ, тому що R_1 задовольняє залежності по з'єднанню R_2, R_3 і R_4. Перевага схеми з (R_2, R_3 і R_4) над R_1 полягає в тому, що вона усуває надмірність, а разом з нею аномалії відновлення.

8.7 З'єднання без втрат і залежності, що зберігаються

З'єднання без втрат мають велике значення, тому що з'єднання із втратами не дозволяють відновити вихідну схему по декомпозиції. Таким чином, із всіх можливих розкладань схем повинні використовуватися тільки ті розкладання, які мають властивість з'єднання без втрат. Як вказувалося вище, еквівалентність схем вимагає *збереження залежностей* і відсутності втрат.

Нехай у схемі R є множина функціональних залежностей. Говорять, що схема R розкладена без втрат на відношення R_1, R_2, ..., R_k с збереженням функціональних залежностей, якщо для кожного кортежу r з R r може бути відновлений з'єднанням його проекцій, тобто $r = r\,[R_1] * r\,[R_2] * \ldots * r[R_k]$ ($*$ позначає природне з'єднання).

Умова відсутності втрат при з'єднанні. Якщо R_1, R_2 є результатом розкладання R зі збереженням множини функціональних залежностей, це розкладання забезпечує з'єднання без втрат зі збереженням функціональних залежностей тоді й тільки тоді, коли

$R_1 \cap R_2 \rightarrow$(або $\rightarrow\!\!\rightarrow$) $R_1 - R_2$

або

$R_1 \cap R_2 \rightarrow$(або $\rightarrow\!\!\rightarrow$) $R_2 - R_1$

Тут операції перетинання ($\cap$) і різниці (—) визначені над множинами атрибутів відношення.

З'єднання R_1 і R_2 в відношення R буде без втрат зі збереженням функціональних залежностей тоді й тільки тоді, коли перетинання множин атрибутів відношень R_1 і R_2 визначають

(або багатозначно визначають) різницю множин атрибутів R_1 □-R_2 або R_2□-R_1.

Приклад. Нехай задана схема відношення СЛУЖБОВЦІ і функціональні залежності, що встановлюють, що кожний службовець може працювати лише в одному відділі й жити лише в одному місті:

СЛУЖБОВЦІ (**СЛ**#, ВІДД, МІСТО);
(СЛ# → ВІДД і СЛ# → МІСТО – властивість ключа).

РОЗКЛАДАННЯ1: E_1**(СЛ#,** ВІДД); E_2**(СЛ** #, МІСТО).
РОЗКЛАДАННЯ2: E_1**(СЛ#,** ВІДД); E_2(ВІДД, МІСТО).

Для першого розкладання:

$E_1 \cap E_2$= СЛ#; $E_1 - E_2$ = ВІДД; $E_2 - E_1$= МІСТО.

Оскільки СЛ# → ВІДД, то $E_1 \cap E_2 \rightarrow E_1 - E_2$ (як видно із заданих функціональних залежностей): таким чином, розкладання є розкладанням без втрат.

Для другого розкладання

$E_1 \cap E_2$= ВІДД $E_1 - E_2$== СЛ# $E_2 - E_1$= МІСТО

Тому що ВІДД не визначає СЛ# і МІСТО, то розкладання є розкладанням із втратами.

Для розкладань, що складаються більш ніж із двох відношеннь, можна використати метод табло. Не вдаючись у подробиці, цей метод можна представити в такий спосіб.

Дана множина функціональних залежностей, схема й відношення, отримані в результаті розкладання. Процедура складається в побудові таблиці, рядками якої є розкладені відношення, а стовпями – список атрибутів (A_1, ..., A_n) цих відношенні без повторень. Таблиця заповнюється хрестиками, якщо елемент рядка i у стовпці j відповідає атрибуту A_j, відношення R_i, у противному випадку в таблиці ставиться 0. Після заповнення таблиці проводиться ітеративний перегляд всіх функціональних залежностей ($X \rightarrow Y$). Якщо для атрибутів з X найдуться рядки, де у відповідних місцях знаходяться хрестики, то нулі цих рядків, що відповідають стовпцям атрибутів з Y, заміняються на хрестик. Якщо в результаті з'явиться рядок таблиці, повністю заповнений хрестиками, то дане з'єднання – без втрат; у противному випадку – це з'єднання із втратами.

Приклад. Нехай задані відношення $R(A, B, C, D)$ і функціональні залежності A→ C, B →C, C,D→B, C→D. Схема після розкладання має вигляд:

$R_1(A, B)$, $R_2(B, D)$, $R_3(A, B, C)$, $R_4(B, C, D)$.

Таблиця буде виглядати так:

	A	*B*	*C*	*D*
R_1	x	x	0	0
R_2	0	x	0	x
R_3	x	x	x	0
R_4	0	x	x	x

Розглянемо $A \rightarrow C$

	A	*B*	*C*	*D*
R_1	x	x	(x)	0
R_2	0	x	0	x
R_3	x	x	x	0
R_4	0	x	x	x

Далі $B \rightarrow C$

	A	*B*	*C*	*D*
R_1	x	x	x	0
R_2	0	x	(x)	x
R_3	x	x	x	0
R_4	0	x	x	x

Залежність $C,D \rightarrow B$ нічого не змінює.

Далі $C \rightarrow D$

	A	*B*	*C*	*D*
R_1	x	x	x	(x)
R_2	0	x	x	x
R_3	x	x	x	(x)
R_4	0	x	x	x

Є рядок з усіма хрестиками; отже, дане розкладання без втрат.

9 МОВНІ ЗАСОБИ СКБД

Є численні приклади мов СКБД, що поєднують можливості опису даних і маніпулювання даними в єдиних синтаксичних рамках. Досить популярним серед мов такого роду є реляційна мова SQL, розроблена у дослідницькій лабораторії фірми IBM Corp. у Сан-Хосе й реалізована у середині 70-х років в експериментальної реляційній СКБД System R.

У цей час SQL досить широко розповсюджена. Прийняті стандарти SQL як реляційної мови, розроблені Американським національним інститутом стандартів (ANSI) і Міжнародною організацією по стандартизації (ISO). Вони використовуються в багатьох СКБД. Іншими прикладами мов цього класу можуть служити мова Quel системи Ingres, створеної в Каліфорнійському університеті, мовні засоби більшості СКБД для персональних ЕОМ, наприклад мова dBase сімейства СКБД фірми Ashton-Tate і сумісних з ними систем, мова СКБД R:Base фірми Microrim.

Деякі СКБД мають у своєму розпорядженні такі мови, які не тільки реалізують функції визначення даних і маніпулювання даними, але й мають керуючі структури й інші засоби, властиві традиційним мовам програмування. Завдяки цьому вони можуть використовуватись як функціонально повний інструментальний засіб для створення прикладних систем або для формулювання запитів користувачів до бази даних. Такі мови називають *автономними*. Як приклад можна привести мову dBase, побудовану у стилі структурного програмування. Автономні непроцедурні мови високого рівня називають *мовами запитів*. Вони призначені для кінцевих користувачів систем баз даних.

Хоча автономні мови сучасних СКБД і мають розвинені функціональні можливості, існує багато додатків, для реалізації яких цих можливостей виявляється недостатньо. Досить поширені випадки, коли модель даних, що підтримується системою, виявляється занадто бідною засобами моделювання семантики предметної області; в ній недостатньо розвинені структуроутворюючі й/або операціональні засоби моделі.

У таких ситуаціях вже на стадії створення СКБД передбачається, що її мова буде використовуватис в сполученні зі звичайними мовами програмування, засобами яких буде заповнювати функціональна неповнота даної системи. Мова програмування, по відношенню до мови СКБД, виступає в ролі мови, що включає, і прикладні системи реалізуються на такій об'єднаній мові.

Реалізація інтерфейсів мови, що включає, заснована на використанні області зв'язку між прикладною програмою й СКБД, доступним обом взаємодіючим сторонам. Для цієї мети структура області зв'язку повинна бути описана в програмі в термінах її мови програмування. Через область зв'язку здійснюється обмін даними між програмою й СКБД при виконанні

операцій маніпулювання даними за допомогою операторів DML, передаються програмі сигнали зворотного зв'язка СКБД – повідомлення про помилки й коди завершення операцій. У системі ADABAS через область зв'язку програма передає й самі команди для СКБД.

У процесі еволюції СКБД і мов програмування стало ясно, що має серйозні недоліки підхід, пов'язаний з використанням інтерфейсів мов, що включають, і інтерфейсу користувача. Цей підхід змушує мати справа з комбінацією двох зовсім різних з концептуальної точки зору інструментальних засобів.

Дві інструментальні мови використовуються у одному програмного додатку, які розроблені незалежно друг від друга і базуються на різних системах понять, мають різні функціональні механізми, а іноді й "раіномасштабні" технології. Така ситуація породжує проблему, яка називається *невідповідністю імпедансу*.

Із цією проблемою доводиться зіштовхуватися, наприклад, при використанні інтерфейсу мови, що включає, реляційній СКБД для якої-небудь із традиційних мов програмування. У зв'язку з тим, що традиційні мови програмування не мають у своєму розпорядженні можливостей обробки даних на теоретико-множинному рівні, виникає неузгодженість СКБД і мови програмування по структурах даних.

Це змушує розроблювачів СКБД доповнювати реалізовану модель даних неприродними для неї об'єктами й операціями над ними для синхронізації алгоритмів обробки даних у прикладній програмі й механізмів керування даними в СКБД. Така неузгодженість не тільки створює дискомфорт для розроблювачів прикладних систем. Наслідком її часто буває й недостатньо висока продуктивність усього комплексу, що не піддається оптимізації саме в силу його неоднорідності.

Аналогічна ситуація виникає при спробах сполучення сучасних мов програмування, що володіють можливостями абстрактних типів даних і інших нових засобів, із сьогоднішніми комерційними СКБД. Тут виявляється *невідповідність компетенції* – вимоги до СКБД із боку прикладної програми, написаної на такій мові, не можуть бути задоволені.

Спроби подолання зазначених труднощів привели до створення нового інструментального напрямку, пов'язаного зі створенням систем програмування баз даних. Їхньою основою повинні стати цілісні мови програмування нового типу, що з'єднують на єдиній концептуальній основі як можливості новітніх мов програмування, так і функції, властиві язиковим засобам традиційних СКБД. Використання систем програмування баз даних відкриває принципово нові архітектурні підходи до реалізації СКБД. На цьому шляху можливо, крім того, побудова таких систем, які підтримують у єдиному операційному середовищі різні моделі даних відповідно до потреб користувачів.

10 РЕЛЯЦІЙНА СКБД FOXPRO

Ця СКБД виявилася досить удалою: вона сполучила програмні можливості мови Clipper і можливості інтерактивної обробки баз даних, які представляють такі системи, як, наприклад, dBaseIII PLUS, dBaseIV. Після того, як даний продукт став належати Microsoft, були розроблені й удосконалені версії Visual FoxPro під Windows. При цьому були збережені практично всі програмні можливості FoxPro, схожі з можливостями Clipper, і додалися нові можливості, наприклад, додалися команди мови SQL.

Так само, як і в Clipper, відношення (таблиці) реляційної СКБД FoxPro записуються в.DBF файли. В FoxPro використаються ті ж основні типи даних, що й в Clipper, а саме:

Character – символьні рядки;

Numeric – числовий тип;

Date – тип дата (8 байт);

Logical – логічний тип (1 байт);

Memo – текстові поля максимального припустимого розміру 64 Кб (зберігаються на диску окремо в.dbt файлах).

Але є й нові типи даних.

Багато дій в FoxPro можна виконати мишею, як у будь-якому іншому Windows-додатку, але можна писати команди в командному вікні.

Мова FoxPro складається з команд, які можуть виконуватись по одній, або бути записані в програмі. Якщо команди виконуються по одній, вони пишуться у командному вікні Visual FoxPro. У будь якій формі задавання команд (безпосередньо або через програму), кожна *окрема команда* повинна бути записана *у окремому рядку.* Якщо рядок з командою занадто довгий, наприклад, команда має багато аргументів, допустимо записувати команду в декілька рядків, але тоді в кожному рядку команди, крім останнього, необхідно поставити символ продовження команди. В FoxPro таким символов є символ “;” (крапка з комою). Команди, які описується в даному розділі, не містять у собі символу “;” наприкінці кожного, крім останнього, рядка, тому що тут показаний *формат запису* команди з переліком можливих аргументів, а не сама команда, як вона виглядає під час виклику.

10.1 Створення бази даних і таблиць в FoxPro

Створити базу даних можна стандартними засобами, наявними в будь-якому Windows-додатку: або за допомогою пункту меню File/New або кнопкою (New). У кожному разі з'являється діалогове вікно з перерахуванням всіх видів документів, які можна створити в FoxPro. Із

цього списку потрібно вибрати базу даних (Database). Можна також створити базу даних командою в командному вікні:

CREATE DATABASE DBname

Створити нову таблицю також можна стандартними засобами, наявними в будь-якому Windows-додатку: за допомогою пункту меню File/New або кнопкою (New). У цьому випадку зі списку всіх видів документів, які можна створити в FoxPro, потрібно вибрати таблицю (Table).

При створенні таблиці за допомогою команди, записаної в командному вікні (Command Window), діалогове вікно для вибору виду створюваного документа не з'являється, тому що в команді конкретно вказується, якого типу документ створюється. Для створення таблиці потрібно написати команду:

CREATE TABLE FileName

Коли ім'я таблиці зазначене тим або іншому способу, відкривається вікно **Дизайнера таблиць** (Table Designer).

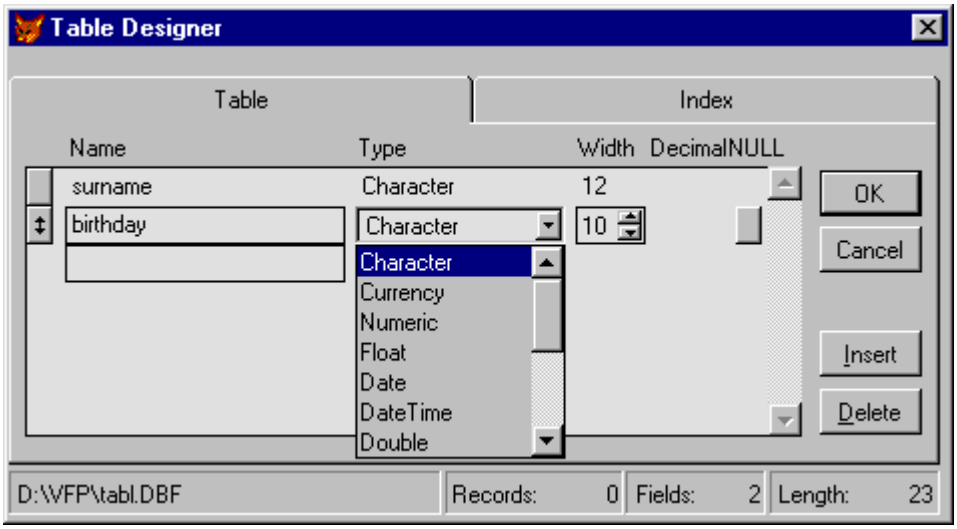


Рис. 10.1 Вікно **Table Designer**

У цьому вікні є дві вкладки: Table і Index. Для створення структури таблиці БД (завдання її атрибутів) використовується перша вкладка. Відразу після відкриття вікна **Table Designer** на першій вкладці в колонці Name є віконце, у якому потрібно записати ім'я першого атрибута (поля), потім указати для нього тип даних (колонка Type), для типів Character і Numeric указати кількість позицій (колонка Width), для типу Numeric можна ще задати кількість цифр після коми (Decimal), у колонці NULL потрібно поставити відмітку, якщо для цього атрибута припустимі значення NULL. У колонці Type для вибору типу даних є список з можливими типами.

Коли зазначені всі необхідні атрибути, потрібно нажати або клавішу **Enter** або кнопку у вікні **Ok**. FoxPro створить задану структуру таблиці БД

і запитає: “Input data records now? (Уводити запису з даними зараз?)”. Якщо відповісти “так”, то з'явиться вікно пункту меню **View/Edit**, у якому можна вводити й редагувати дані (див. рисунок 10.2).

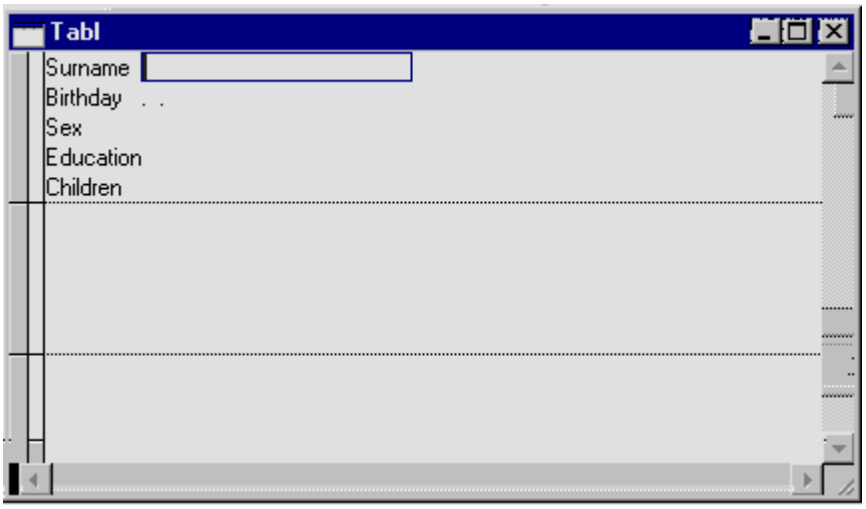


Рис.10.2 Вікно редагування й додавання записів у таблицю

При уведенні нової таблиці це вікно відкривається в режимі додавання записів; якщо відкрити це вікно для вже наявної таблиці (або через меню, або командою Edit), те для можливості додавання записів у таблицю потрібно поставити галочку біля пункту меню **View/Append Mode**.

Якщо з'являється необхідність змінити структуру таблиці (кількість, ширину або назви атрибутів (полів)), то потрібно для *відкритої таблиці* виконати команду MODIFY STRUCTURE. Крім того, якщо для відкритої таблиці виконаний пункт меню **View/Browse** (або команда BROWSE), які виводять на екран зміст таблиці БД, і вікно із цією таблицею активно, тоді в меню є пункт **Table**, що дозволяє обробляти таблицю. За допомогою пунктів меню **Table/Properties…/Modify** можна також змінити структуру таблиці.

10.2 Відкриття таблиці БД

Відкрити таблицю можна або засобами Windows-вікна (кнопка Open або пункт меню **File/Open** — і зі списку можливих документів вибрати спочатку таблицю (Table), потім знайти серед наявних .DBF файлів потрібний), або можна використати команду USE, що надає більше можливостей, чим пункт меню або кнопка:

```
USE [TableName|?]
[IN nWorkArea | cTableAlias] [AGAIN]
[NOREQUERY [nDataSessionNumber]]
[NODATA]  [INDEX IndexFileList | ?
[ORDER [nIndexNumber | IDXFileName
| [TAG] TagName [OF CDXFileName]
[ASCENDING | DESCENDING]]]]
[ALIAS cTableAlias] [EXCLUSIVE] [SHARED] [NOUPDATE]
```

Аргументи, що найбільше використовуються:

`TableName` – визначає ім'я таблицы, що відкривається. Оскільки пробіли істотні в іменах файлів в Windows 95 і Windows NT, бажано уникати використання сторонніх пробілів в `TableName` в Visual FoxPro.

`?` – з'являється діалогове вікно USE, у якому можна вибрати таблицу, що відкривається.

`IN nWorkArea` – визначає робочу область, у якій відкривається таблиця. Можнае закрити таблицю в зазначеній робочій області, написавши USE з опцією IN і з номером робочої області.

Опція IN підтримує нуль як номер робочої області. Включення 0 відкриває таблицю в самою низкою (з найменшим номером) доступної робочої області. Наприклад, якщо таблиці відкриті в робочих областях з 1 до 10, що наступна команда відкриває таблицю клієнта в робочій області 11:

USE customer IN 0

`IN cTableAlias` – визначає, що таблиця відкривається в робочій області таблиці, що є в цей час відкритої. Псевдонім відкритої таблиці визначений через `cTableAlias`. Якщо опустити `nWorkArea` і `cTableAlias`, таблиця відкривається в поточній робочій області.

`ORDER [TAG TagName] [OF CDXFileName]` – визначає хазяїна, що управляє TAG-ами в складеному `.CDX` індексному файлу. Назва TAG-а може бути від структурного складеного індексного файлу або будь-якого відкритого складеного індексного файлу.

`EXCLUSIVE` – відкриває таблицю для виняткового використання у мережі. Якщо виконана команда `SET EXCLUSIVE` з опцією `ON`, і в команді не вказано ні опцію `EXCLUSIVE`, ні опцію `SHARED`, тоді за замовчуванням таблиця відкривається для виняткового використання.

`SHARED` – відкриває таблицю для загальнодоступного використання у мережі. `SHARED` дозволяє відкрити таблицю для загальнодоступного використання навіть коли виконана команда `SET EXCLUSIVE` з опцією `ON`.

NOUPDATE – запобігає зміни в таблиці й в її структурі (таблиця відкривається тільки для читання).

Зауваження: Якщо команда USE написана без імені таблиці, і файл таблиці відкритий у поточній робочій області, таблиця закривається. Також, таблиця закривається, коли інша таблиця відкривається в тієї ж самій робочій області. Не можливо відкрити більше однієї таблиці в одній робочій області одночасно.

10.3 Зміна робочої області

Ці дії виконує команда SELECT:

```
SELECT nWorkArea | cTableAlias
```

Аргументи:

NWorkArea – указує номер обираної робочої області; якщо вказати 0, то вибереться вільна область із найменшим номером.

CTableAlias – вибирається робоча область, що містить відкриту таблицю з іменем CTableAlias.

10.4 Зона дії команди

Мова Clipper розроблялась як мова, яка передбачає по-записну обробку записів в таблиці. Мова FoxPro зберігає всі можливості, притаманні мові Clipper. Тому в ній присутні як конструкції мови SQL, призначені для обробки цілої таблиці, так і команди по-записної обробки рядків таблиці (кортежів відношення).

У багатьох командах, що працюють із кортежами відношень, вказується, до яких кортежів треба застосовувати дану команду. У записі таких команд зону дії позначають словом <scope>. Зона дії може приймати наступні значення:

ALL – всі кортежі відношення;

REST – залишок кортежів (починаючи з поточного);

NEXT <N> – наступні N кортежів;

RECORD <N> – кортеж під заданим номером N.

Якщо не вказати зону дії, то деякі команди мають на увазі один поточний кортеж, інших – всі кортежі.

10.5 Умови дії команди

У командах обробки кортежів відношення крім зони дії можуть ще вказуватися умови, які визначають, до яких кортежів застосовується дана команда. Ці умови можуть мати вигляд:

```
FOR <expL>   або  WHILE <expL>
```

Тут <expL> – логічний вираз (відповідно, <expN> - числовий вираз, <expC> – символьний, <expD> – вираз типу дата).

Якщо задана FOR умова, то вибираються у заданій зоні дії всі кортежі, що задовольняють умові. Якщо задана WHILE умова, то пошук кортеж виконується, поки умова вірна; як тільки зустрінеться кортеж, для якого умова виявиться невірною, пошук припиняється.

10.6 Відмітка записів для видалення

Команда DELETE
Синтаксис:

```
DELETE [Scope] [FOR <exp1>] [WHILE <exp2>]
     [IN nWorkArea | cTableAlias]
     [NOOPTIMIZE]
```

<scope> – задає зону дії команди; за замовчуванням обробляються всі кортежі.

WHILE <exp1> і FOR <exp2> – задають умови, доки кортежі розглядати (WHILE) і які кортежі (FOR).

IN nWorkArea – визначає робочу область таблиці, у якій записи позначаються для видалення.

IN cTableAlias – визначає псевдонім таблиці, у якій записи позначаються для видалення. Якщо опущені nWorkArea і cTableAlias, записи позначаються для видалення в таблиці, що перебуває в поточній робочій області.

Зоною дії команди (scope) за замовчуванням є один поточний запис, якщо не задані умови WHILE і/або FOR. Якщо умови задані, то зоною дії за замовчуванням є весь файл.

Після команди DELETE кортежі тільки позначаються на видалення, але не стираються з файлу. Фізично стирає позначені на видалення записи команда PACK (без параметрів). Після дії команди PACK записи не можна відновити.

10.7 Відновлення записів, позначених на видалення

Команда RECALL
Синтаксис:

```
RECALL  [<scope>] [WHILE <exp>] [FOR <exp>]
```

Аргументи команди мають той же зміст і ті ж значення, що й у команді DELETE.

10.8 Видалення й відновлення записів через меню

Для того, щоб позначити на видалення, відновити позначені на видалення записи, фізично стерти позначені на видалення записи засобами меню Visual FoxPro потрібно, щоб було активним вікно з таблицею, для якої виконуються ці дії. Тоді у меню є пункт **Table**, у якому є підпункти:

Delete Records…—для позначки записів, що видаляються;

Recall Records…—для зняття позначки про видалення;

Remove Deleted Records — для фізичного видалення записів з диска.

Якщо нажати на перші зазначені два підпункти, то з'явиться віконце із трьома кнопками ліворуч: `Scope...`, `For...`, `While...` По кнопці `Scope` можна вибрати одну із чотирьох зон дії команди (`ALL`, `NEXT`, `RECORDS`, `REST`). При натисканні кнопок `For` або `While` у віконці **Expression** потрібно написати логічний вираз для записів, що будуть обиратися. При цьому можна не тільки вибирати функції для виразу у віконцях “String”, “Math”, “Date” і знаки логічних операцій у віконці “Logical”, але й замість заміни вручну в обраних функціях слів типу ExpL (ExpN, ExpD), можна у віконці Expression виділити ці слова, а потім двічі клацнути нижче по тому полю таблиці, що повинне стояти на цьому місці (це поле повинно мати той самий тип, що вказаний для відповідного виразу).

При натисканні пункту **Remove Deleted Records** з'являється запит на видалення записів з файлу. Якщо нажати “Yes”, то відзначені на видалення записи зітруться з диска.

10.9 Додавання нового запису в поточну таблицю бази даних

Ці дії виконує команда `APPEND BLANK`.

`APPEND BLANK` є командою бази даних, що додає новий запис у кінець поточного файлу бази даних і потім встановлює його як поточний запис. Нові значення полів перетворяться в порожні значення для кожного типу даних: символьні поля позначаються пробілами; цифрові поля – нулем; логічні поля позначаються як (.F.), поля дати встановлюються `CTOD` (“”);. поля МЕМО залишаються порожніми.

При роботі в мережі, коли до поточного файлу бази даних звертаються й інші користувачі, `APPEND BLANK` намагається закрити доступ стороннім до знову записаної інформації. Ця команда використовується, коли таблицю бази даних обробляють у програмному режимі.

10.10 Установка формату запису дати

Команда `SET CENTURE` указує, потрібно чи в запис дати включати сторіччя.

Синтаксис:

SET CENTURE ON | OFF | <EXPL>

ON або <exp>=.T. – дозволяють уведення й вивід цифр, що відповідають сторіччю в датах;

OFF або <exp>=.F. – забороняють уведення й вивід цифр, що відповідають сторіччю в датах.

Команда SET DATE задає порядок чисел, що визначають день(dd), місяць(mm) і рік(yy) у даті.

Синтаксис:

```
SET DATE [TO] AMERICAN | ANSI | BRITISH | FRENCH |
              GERMAN | ITALIAN | JAPAN | USA
```

Формат дат:

AMERICAN	mm/dd/yy
ANSI	yy.mm.dd
BRITISH	dd/mm/yy
FRENCH	dd/mm/yy
GERMAN	dd.mm.yy
ITALIAN	dd-mm-yy
JAPAN	yy/mm/dd
USA	mm-dd-yy

10.11 Призначення нових значень атрибутам (полям) кортежів.

Команда REPLACE

Синтаксис:

```
REPLACE <idField> WITH <exp>[, <idField2> WITH
       <exp2>…]
       [<scope>] [WHILE<expL>] [FOR<expL>]
```

Аргументи:

<idField> – являє собою ім'я поля, якому призначається нове значення. Якщо псевдонім таблиці передує <idField>, то призначення відбувається в визначеній псевдонімом робочої області.

<exp> – це значення, що призначається.

Зміст і значення інших аргументів, як і в командах вище.

По пункті меню **Table/Replace Field…**також можна заміняти значення, але тільки для одного поля (атрибута) за один раз використання пункта меню.

10.12 Функції FoxPro

10.12.1 Функції для роботи з датами

DATE() – функція без аргументу, що повертає поточну дату.

YEAR(<expD>) – функція з аргументом типу дата, що повертає рік з дати (тобто число).

MONTH(<expD>) – функція з аргументом типу дата, що повертає номер місяця з дати.

CMONTH(<expD>) - функція з аргументом типу дата, що повертає назву (по-англійському) місяця з дати (тобто символьний рядок).

DAY(<expD>) – функція з аргументом типу дата, що повертає номер дня з дати.

DOW(<expD>) – перетворює величину типу дата в відповідній цій даті номер дня тижня; номери починаються з неділі (1).

CDOW(<expD>) – перетворює величину типу дата в англійську назву дня тижня, який відповідає цій даті.

10.12.2 Функції перетворення даних

CTOD(<expC>) – перетворює символьний вираз в дане типу дата; при цьому символьний вираз повинне містити дату, записану у вигляді, якій встановлений командою SET DATE. Наприклад, якщо встановлено формат дати FRENCH, сторіччя придушується, і потрібно записати дату 15 липня 1986 р, те функція буде мати вигляд: CTOD(”15/07/86”).

DTOC(<expD>) – зворотна до CTOD, перетворює вираз типу дата в символьне дане. Вигляд рядка з датою буде визначатися командами
SET DATE і SET CENTURE.

DTOS(<expD>) – функція, що перетворює дату в рядок виду “YYYYMMDD”; використовується для індексних виразів, особливо для складних індексів, що містять і дати й символьні вирази.

STR(<expN>[,<nLength>[,<nDecimals>]]) – перетворює числовий вираз <expN> у символьний рядок. <nLength> – довжина символьного рядка, що повертається, включаючи цілу й дробову частину, десяткову крапку й знак; <nDecimals> - число знаків після крапки (в дробовій частині). Якщо <nLength>

менше числа цифр у цілій частині <expN>, то STR поверне послідовність символів “*” замість цифр; якщо <nLength> зазначений, а <nDecimals> ні, тоді число округляється до цілого. Якщо не зазначені ні <nLength>, ні <nDecimals> – використовується формат запису за замовчуванням.

VAL(<expC>) – зворотна до STR функція перетворення рядка в число; при цьому рядок може містити символи, які не можна представити у вигляді числа. Функція VAL вираховує <expC> до другої десяткової крапки або першого нецифрового символу або до кінця виразу. Наприклад:

```
VAL(”12a13”)=12.00
VAL(”1.34.56”)=1.34
VAL(”a45”)=0
```

10.12.3 Функції для роботи з рядками тексту

ALLTRIM(<expC>) – видаляє провідні й хвостові пробіли з рядка <expC>. В деяких версіях FoxPro ця функція має назву TRIM.

RTRIM(<expC>) – видаляє хвостові пробіли з рядка <expC>.

LTRIM(<expC>) – видаляє провідні пробіли з рядка <expC>.

LEFT(<expC>,<nCount>) – повертає підрядок з рядка <expC>; підрядок містить <nCount> символів, починаючи із крайнього лівого символу <expC>.

RIGHT(<expC>,<nCount>) – повертає підрядок з рядка <expC>; підрядок містить <nCount> крайніх правих символів з <expC>.

SUBSTR(<expC>,<nStart>[,<nCount>]) – повертає підрядок з рядка <expC>, починаючи із символу з номером <nStart> (якщо <nCount> не зазначено, то вибираються всі символи від <nStart> до кінця рядка).

Наприклад:

Name=”Пржевальський”

? SUBSTR(Name,1,4) Результат: Прже

? SUBSTR(Name,15) Результат: Порожній рядок

LEN(<expC>) – функція, що повертає довжину рядка (кількість символів у рядку).

SPACE(<nCount>) – функція, що повертає рядок, що містить <nCount> пробілів.

REPLICATE(<expC>,<nCount>) – повертає рядок <expC>, повторений <nCount> разів.

LOWER(<expC>) – перетворює латинські заголовні букви в рядку в прописні; інші символи залишаються без змін.

UPPER(<expC>) – перетворює латинські прописні букви в рядку в заголовні; інші символи залишаються без змін.

10.12.4 Умовна функція

IF(<expL>,<exp1>,<exp2>) – якщо значення вираз <expL> істинний, тоді повертає значення виразу <exp1>, інакше – <exp2>. При цьому вирази <exp1>,<exp2> можуть бути різних типів. Наприклад, точний вік людини з урахуванням того, чи був у неї цього року день народження вираховується наступним виразом:

```
YEAR(DATE())-YEAR(Birthday)-
IF((MONTH(DATE())>MONTH(Birthday)).OR.
((MONTH(DATE())=MONTH(Birthday)).AND.(DAY(DATE())>DAY(Birthday))),0
,1)
```

10.13 Створення індексних файлів

Як правило, записи в БД утримуються в невідсортованому порядку – так, як їх уводили. Фізичне сортування даних на диску вимагають часу (а БД можуть мати дуже багато записів), і звичайно не ефективне, тому що додавання або редагування даних може порушити порядок. Крім того, один раз потрібно, щоб дані були впорядковані за одним атрибутом, інший раз – за іншим, третій раз – і зовсім за комбінацією декількох атрибутів.

Тому фізичне сортування записів у БД звичайно не виконують, а створюють так звані індексні файли. Індексні файли не міняють порядок розташування записів у фізичній БД, але міняють порядок доступу до них і порядок їхнього виводу. Для будь-якої БД можна написати скільки завгодно індексних файлів.

Індексні файли можуть бути різними: одиночними, складовими, структурними складовими. Зручніше за все користуватися структурним складовим файлом: цей файл має ім'я таке ж, як ім'я таблиці БД (для якої створюються індекси) і розширення .CDX, але ні його ім'я, ні розширення не потрібно вказувати при створенні індексних файлів, потрібно лише в команді INDEX указати ім'я TAG-а (відмітки), під яким у структурний

складовий індексний файл записується даний індекс. Структурний складовий індексний файл гарний ще тим, що не треба думати про відновлення індексів, які в ньому втримуються – при редагуванні таблиць цей файл відкритий завжди й автоматично обновлюється.

Команда для створення індексів:

```
INDEX ON eExpression TO IDXFileName | TagName [OF CDXFileName]
     [FOR lExpression]
     [ASCENDING | DESCENDING]
     [UNIQUE | CANDIDATE]
     [ADDITIVE]
```

Основні аргументи:

`eExpression` – визначає індексне вираження, що може включати назви атрибутів (полів) поточної таблиці. Ключ індексу, заснований на індексному виразі створюється в індексному файлі для кожного запису (кортежу) у таблиці. Visual FoxPro використає ці ключі, щоб відображати й звертатися до записів у таблиці. Вираз може бути типів Character, Numeric, Date. Якщо буде намагання будувати індекс із ключем, що може мати змінну довжину, ключ буде доповнюватися пробілами.

Другим аргументом може бути або `TO IDXFileName` або `TAG TagName [OF CDXFileName]`, але краще писати `TAG` і без `OF CDXFileName` для включення індексу в структурний складений індексний файл.

`TAG TagName` – задає в складовому індексному файлі ім'я (відмітку) для даного індексу.

`ASCENDING` або `DESCENDING` – за зростанням або убуванням значень ключового виразу; за замовчуванням буде `ASCENDING`.

Також індекс можна створити й за допомогою пункту меню. Коли відкрите й активне вікно таблиці бази даних, і в меню є пункт **Table**, потрібно виконати пункти **Table/Properties…,**і у вікні, що з'явилося, нажати **кнопку…**, те відкриється вікно **Table Designer**, у якому є дві сторінки (вкладки): Table і Index. Для створення й зміни індексів потрібно перейти на другу сторінку (дивись рис.10.3).

У першій колонці (**Name**) потрібно написати ім'я індексу (TAG-а); у другий (**Type**) вибирається тип індексу — найчастіше Regular; у третій (**Expression**) записується ключовий вираз, за яким відбувається індексування, при цьому можна клацнути мишою по кнопочці [...] ліворуч від віконця для ключового вираження, і відкриється вікно **Expression Builder**, що дозволяє одержати допомогу при написанні виразу:

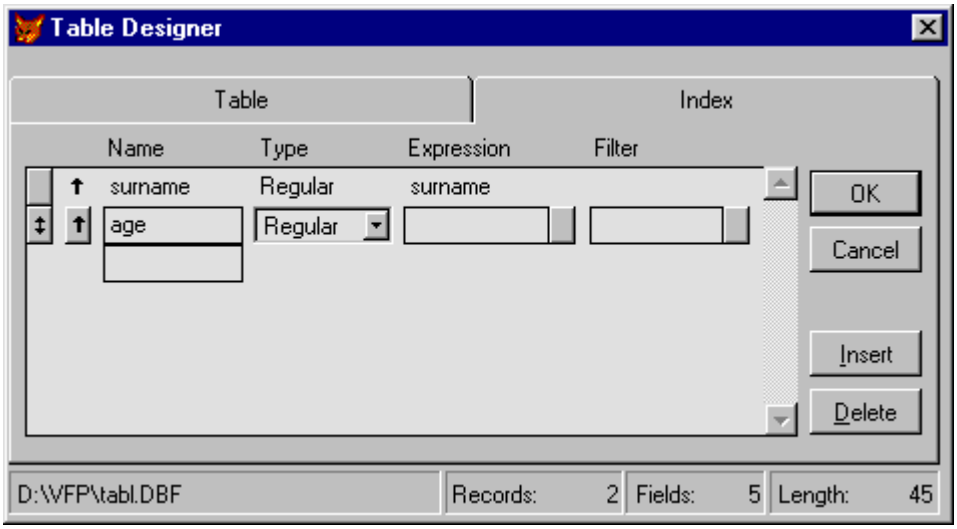


Рис.10.3 Вікно **Table Designer/Index**

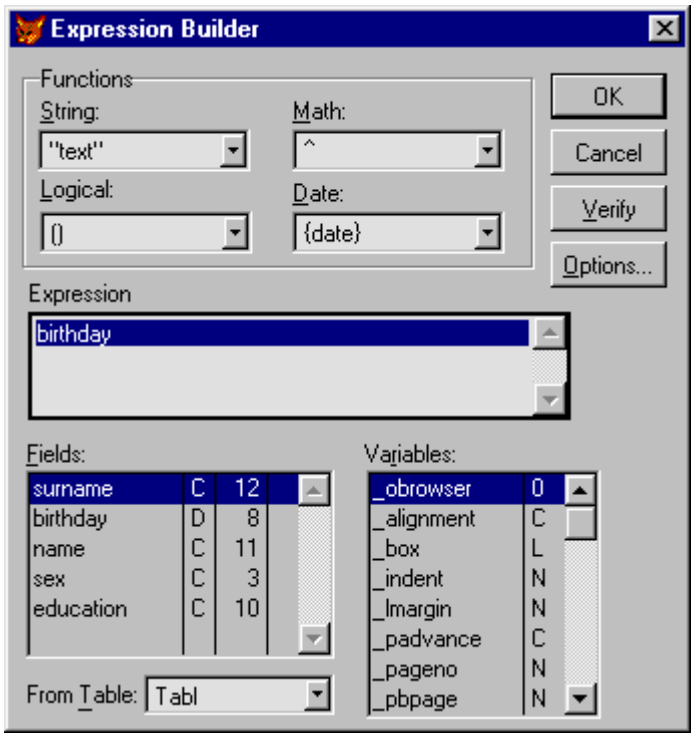


Рис.10.4 Вікно Expression Builder

У цьому вікні у віконці **Fields** можна вибирати поля (атрибути), що включаються в індексний вираз; у віконцях області «Functions» можна вибрати функції, необхідні (можливо) для індексного виразу. Якщо пишеться складовий індекс, що включає кілька атрибутів різних типів, то необхідно привести атрибути до загального типу. Загальним типом для всіх типів звичайно є символьний (рядковий) тип. Для приведення одного типу даних в інший використовують функції перетворення типів. які описані в розділі 10.12.2.

10.14 Установка зв'язків між відношенями

Команда SET RELATION

Установка зв'язку між двома робочими областями (основною і підлеглою) за значенням ключа або номеру запису.

Зв'язки можуть бути встановлени між таблицями, якщо вони мають спільні домени (атрибути). До встановлення зв'язку підлегла (дочірня) таблиці повинна мати відкритий індекс, створений з спільних атрибутів таблиць. Основна (батьківська) таблиця може мати будь-який відкритий індекс, або не мати його.

Синтаксис команди:

```
SET RELATION TO
     [eExpression1 INTO nWorkArea1 | cTableAlias1
     [, eExpression2 ...]
          [IN nWorkArea | cTableAlias]
     [ADDITIVE]]
```

Аргументи:

`TO eExpression` – це вираз, який використовується для виконання команди SEEK у підлеглій робочій області, щоразу, коли покажчик запису переміщається в основній робочій області. Для цього підлегла робоча область повинна мати відкритий індекс. Команда SEEK переміщає покажчик поточного запису на запис, який повинен стати наступним поточним записом *у відповідності до ключа індексування таблиці.*

`INTO nWorkArea2 | cTableAlias2` – визначає підлеглу робочу область і може задаватися або номером робочої області (`nWorkArea2`), або іменем таблиці в ній (`cTableAlias2`).

`ADDITIVE` – додає нові зв'язки до існуючих зв'язків у поточній робочій області. Якщо ця опція не визначена, що існуючі зв'язки в поточній робочій області звільняються до того, як будуть установлені нові підлеглі зв'язки.

`SET RELATION TO` без аргументів звільняє всі зв'язки, визначені в поточній робочій області.

Для установки зв'язку між таблицями за допомогою меню, підлеглу таблицю також потрібно проиндексировать необхідним образом, зробити індекс, створений з спіних атрибутів, активним, потім виконати пункт меню **Window/View Window**. З'явиться вікно **View**, у якому будуть всі

відкриті таблиці (у віконці **Aliases**). У нових версіях FoxPro вікно **View** називається **Data session**.

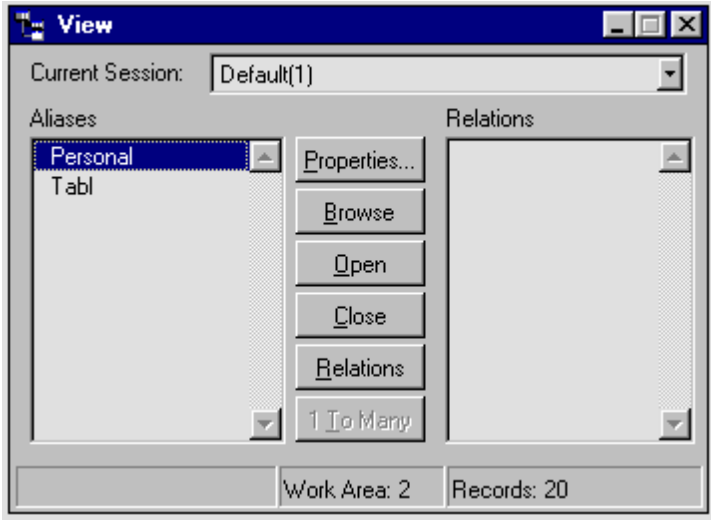


Рис. 10.5 Вікно View

Клацнувши один раз по імені таблиці у віконці **Aliases,** можна переглянути, які є в неї індекси (клацнувши по кнопці **Properties…**), і встановити зв'язок за допомогою кнопки **Relations**. При цьому потрібно спочатку клацнути по імені батьківської таблиці, потім по кнопці **Relations**, потім по імені дочірньої таблиці. Якщо зв'язок можливий (тобто дочірня таблиця проіндексована по спільних атрибутах, по яких встановлюється зв'язок, і цей індекс обраний у якості активного), то зв'язок встановиться, і його буде видно в правому віконці (**Relations**).

10.15 Звіти в FoxPro

Створити форму звіту можна або командою:

```
CREATE REPORT FileName
```

або пунктом меню **File/New/Report**.

У кожному разі відкриється вікно **Report Designer**:

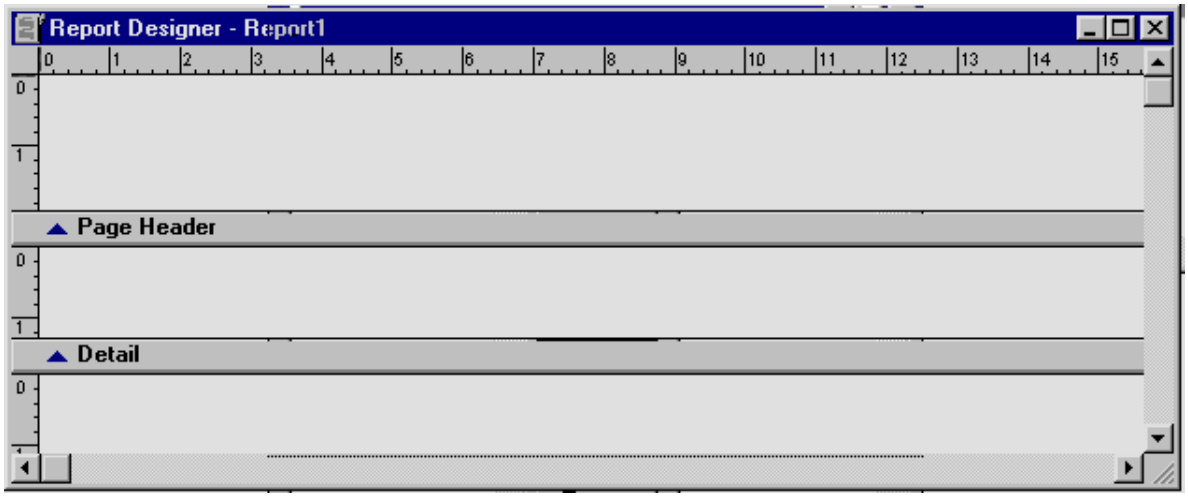


Рис.10.6 Вікно дизайнера звітів (**Report Designer**)

У верхнім його віконці (**Page Header**) потрібно писати заголовки колонок звіту, а в другому (**Detail**) - вирази для вмісту колонок.

Разом з вікном Дизайнера Звітів зазвичай відкривається панель **Report Control**. Якщо її немає на екрані, то потрібно поставити галочку в пункті меню **View/Report Controls Toolbar.** На цій панелі є кнопки:

 — за допомогою її пишуться заголовки звіту;

 – за допомогою її пишуться вирази для вмісту колонок звіту.

Також на панелі **Report Control** є кнопки для малювання ліній, прямокутників, закруглених прямокутників. Їх можна використати для оформлення зовнішнього вигляду звіту.

Якщо у звіт включаються дані з декількох таблиць, то таблиці потрібно зв'язати, і у виразах для вмісту колонок обов'язково використовувати розширені імена атрибутів (полів) таблиць, тобто імені атрибуту повинен передувати еліас (ім'я, псевдонім) таблиці, наприклад, `personal.surname`. У виразах можна використовувати функції FoxPro.

Для запису виразів вмісту колонок звіту потрібно нажати кнопку  на панелі **Report Control** і потім провести мишею в області **Detail** для виділення прямокутного простору, що відводиться для даних цієї колонки звіту. Після цього відкривається вікно **Report Expression**, у якому пишеться вираз для колонки звіту:

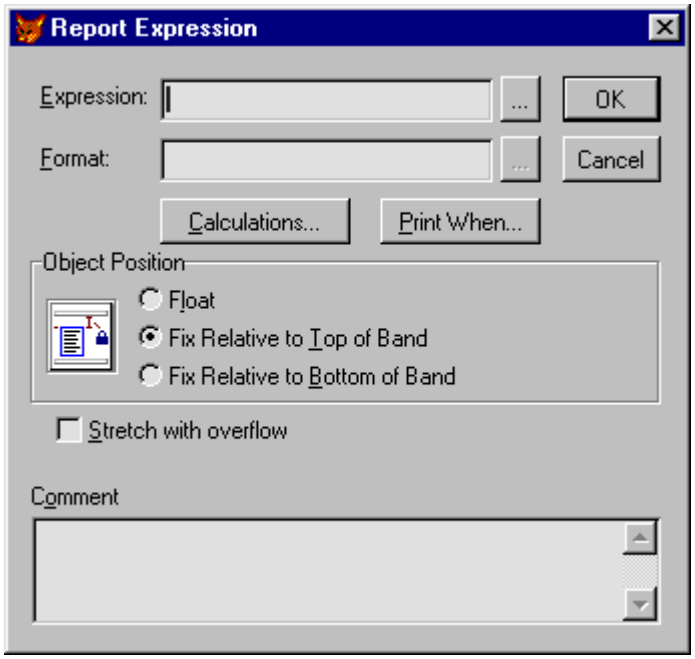


Рис.10.7 Вікно **Report Expression**

У верхнім віконці можна писати вираз для вмісту колонки. Але це не дуже зручно. Краще нажати на кнопочку ▪ поруч із цим віконцем, щоб відкрилося вікно **Expression Builder**, зображене на рис. 10.4.

У цьому вікні зручніше писати вирази; крім того, після написання виразу його можна перевірити, якщо нажати на кнопку **Verify**. Якщо вираз вірний, то внизу робочої області екрана Visual FoxPro (у нижньому лівому куті) з'явиться повідомлення: Expression is valid. Якщо вираз містить помилки, то після натискання на кнопку **Verify**, буде видане повідомлення про помилку.

10.15.1 Складання звітів по таблицях БД

Спочатку, як описано у попередньому розділі, створюється (і зберігається на диску у файлах з розширенням .FRM та .FRX) форма звіту, у якій вказуються назви колонок звіту, і зміст, що виводиться в цих колонках. Потім за цією формою можна безпосередньо виводити звіт командою:

```
    REPORT FORM FileName1 | ?
     [ENVIRONMENT] [Scope] [FOR lExpression1] [WHILE
lExpression2]
     [HEADING cHeadingText] [NOEJECT] [NOCONSOLE] [NOOPTIMIZE]
     [PDSETUP] [PLAIN] [PREVIEW [NOWAIT]]
     [TO PRINTER| TO FILE FileName2 [ASCII]]
     [NAME ObjectName]   [SUMMARY]
```

Як видно із синтаксису запису команди, всі її параметри необов'язкові, крім імені файлу форми звіту (`FileName1`). Параметри `Scope, For, While` уже розглядалися. Параметр `HEADING` задає заголовок, що буде виводитися на кожній сторінці звіту. Але якщо є слово `PLAIN`, те воно відміняє `HEADING` і заголовок буде тільки на першій сторінці. `PREVIEW` – попередній перегляд (без виводу).

Якщо вказати `TO PRINTER` або `TO FILE FileName2`, то вивід звіту буде йти або на принтер, або в зазначений файл; але на екран звіт все-таки виведеться, якщо не написати `NOCONSOLE`.

При виводі звіту у файл бажано написати `ASCII`, щоб у файл виводився тільки текст звіту, але не виводилася службова інформація. Текстовий файл виходить у форматі “Текст Windows” і його можна переглянути й роздрукувати в будь-якому текстовому редакторі “під Windows”.

11 СЕЛЕКЦІЯ В РЕЛЯЦІЙНИХ СИСТЕМАХ

Реляційне вирахування використовується для теоретичного формулювання запитів і є альтернативою реляційної алгебри. Реляційнее вирахування й реляційна алгебра розглядалися в розділі 6. Як там говорилося, користувачі звичайно не використовують мови програмування, у яких потрібно прямо використовувати операції реляційної алгебри, або реляційного вирахування. Проте, ці формули використовують мови запитів, основним з яких є мова SQL.

Реляційна алгебра – процедурна, у той час як реляційне вирахування –декларативно. Це означає, що в реляційній алгебрі користувач сам повинен задати послідовність операцій для одержання відповіді. Відповідь (результат) записується в нове відношення. У реляційному вирахуванні користувач лише формулює запит, залишаючи вибір способу одержання відповіді на розсуд системи. Більшість запитів реляційної алгебри реалізується за допомогою відомих операцій *селекції, з'єднання* й *проекції.* В мові запитів SQL присутні і елементи реляційної алгебри, особливо, коли йде мова про операції *з'єднання* та *об'єднання*. Але вона більш нагадує реляційне вирахування, особливо формою запису предікатів та тим, що є декларативною.

У подальших прикладах використається варіант наступної медичної бази даних:

ЛІКАРНЯ (НОМ_Б, НАЗВА, АДРЕСА, КІЛЬКІСТЬ_ЛІЖОК)
ПАЛАТА (**ПЛТ#**, НАЗВА, КІЛЬКІСТЬ_ЛІЖОК)
ЛІКАР (НСС, ІМ'Я, СПЕЦІАЛЬНІСТЬ)
ЛІКУЄ (НСС, ПЦТ#, ПЛТ#, ДАТА)
ПЕРСОНАЛ (**СЛЖ#**, ІМ'Я, ПОСАДА, ЗАРПЛАТА)
ПАЦІЄНТ (ПЦТ#, ІМ'Я, АДРЕСА, РІК_НАРОДЖЕННЯ)

Не всі СКБД підтримують імена таблиць та їх атрибутів, записаних кірилицею (тим більше з апострофами). Тому ці назви в прикладах наведено тільки для полегшення зрозуміння. Якщо розглядати працюючу базу данних, то ці імена, наібільш вірогідно, записані і ній латиницею.

11.1 Приклади запитів мовою SEQUEL (SQL)

Розглянемо реляційну мову запитів високого рівня SQL, що спочатку називався SEQUEL (Structured English Query Language). Попередником мови SEQUEL була більш строга з погляду математики мова SQUARE. Мова SQL прийнята як мова даних програмних виробів фірми IBM і DML реляційної СКБД SYSTEM R фірми IBM.

Мова SQL має блокову структуру. Блок SQL утворять вираження SELECT FROM WHERE (два останніх ключових слова необов'язкові). Запит може представлятися як одноблоковою програмою, так і декількома

вкладеними блоками. Умови усередині блоку й вкладеність блоків відповідають зв'язкам усередині відношення й між відношенями. Залежно від свого типу вкладеність має на увазі операцію з'єднання або розподілу.

Звичайно блок SELECT FROM WHERE використовується в такий спосіб:

• SELECT список атрибутів і (або) агрегатних функцій від атрибутів
• FROM ім'я {імена} відношення (відношень)
• WHERE умови

Умови можуть бути булєвым предикатом, що складається із простих умов, які можуть охоплювати інші блоки. Зазвичай умови після WHERE є умовами зв'язку (між відношеннями) й/або умовами фільтру (вибірки кортежів з відношень).

Приклади. (У прикладах використовується медична база даних)

Проста селекція

Запит 1: *Знайти, прізвища пацієнтів, що народилися після 1937 р.*

```
SELECT ІМ'Я
FROM ПАЦІЄНТ
WHERE РІК_НАРОДЖЕННЯ > 1937
```

Селекція, що використає з'єднання

Запит 2: *Знайти прізвища пацієнтів, що лікуються в лікаря із НСС = 995-42-7515.*

```
SELECT ІМ'Я
FROM ПАЦІЄНТ
WHERE ПЦТ# =
        SELECT ПЦТ#
                FROM ЛІКУЄ
                WHERE НСС ==”995-42-7515”
```

або

```
        SELECT ІМ'Я
        FROM ПАЦІЄНТ, ЛІКУЄ
        WHERE ПАЦІЄНТ. ПЦТ# = ЛІКУЄ. ПЦТ#
              AND ЛІКУЄ.НСС =”995-42-7515”
```

або

```
      SELECT ІМ'Я
      FROM ПАЦІЄНТ
      WHERE ПЦТ# = ANY ( SELECT ПЦТ#
                         FROM ЛІКУЄ
                         WHERE НСС =”995-42-7515”)
```

або

```
SELECT ІМ'Я
FROM ПАЦІЄНТ
WHERE ПЦТ# IN     ( SELECT ПЦТ#
                      FROM ЛІКУЄ
                      WHERE НСС =”993-42-7515”)
```

або

```
SELECT ІМ'Я
FROM ПАЦІЄНТ
WHERE EXISTS
            (SELECT *
            FROM ЛІКУЄ
             WHERE ПЦТ # = ПАЦІЄНТ. ПЦТ # AND
                   НСС =”995-42-7515”)
```

Із прикладів видно, що по мірі розвитку мови до неї додавалися різні синтаксичні конструкції. Вибір конкретної конструкції залежить від користувача. Стиль деяких з наведених прикладів тяжіє до стилю реляційної алгебри, стиль інших більш близький до стилю реляційного вирахування. Для окремих висловлень зміст легше засвоюється при перегляді запиту зверху вниз, для інших – знизу нагору. Основна ідея всіх варіантів – виділити проміжну підмножину з одного відношення, а потім з'єднати цю підмножину з іншим відношенням, що містить вихідні дані. Ключове слово SELECT * має на увазі вибірку всього кортежу або множини кортежів (а не окремих атрибутів з них), а SELECT UNIQUE викликає усунення дублікатів у проекції результуючого відношення.

Селекція, що включає порівняння множин (розподіл)

Запит 3: *Знайти прізвища лікарів, всі пацієнти яких народилися до 1935 р.*

```
SELECT ІМ'Я
FROM ЛІКАР
WHERE НСС=
      SELECT НСС
      FROM ЛІКУЄ
      WHERE ПЦТ# = ALL
                  SELECT ПЦТ#
                  FROM ПАЦІЄНТ
                  WHERE РІК_НАРОДЖЕННЯ < 1935
```

У цьому запиті другий рівень вкладеності відповідає квантору існування (тобто з'єднанню з відношенням ЛІКАР), а самий нижній рівень вкладеності – квантору загальності (тобто розподілу відношення ЛІКУЄ). Цей запит можна сформулювати іншим образом:

```
SELECT ІМ'Я
FROM ЛІКАР
WHERE НСС =
        SELECT НСС
        FROM ЛІКУЄ
        WHERE SET (ПЦТ#) CONTAINS
                        SELECT ПЦТ#
                        FROM ПАЦІЄНТ
                        WHERE РІК_НАРОДЖЕННЯ<
                        1935
```

За реченням CONTAINS замість блоку SELECT може міститися множина відомих констант, якщо їх небагато. Наприклад, якщо відомо, що шукану множину пацієнтів становлять пацієнти з номерами “0129” і “2560”, те остання частина запиту може виглядати в такий спосіб;

```
WHERE SET (ПЦТ#) CONTAINS (“0129”, “2560”)
```

Селекція з угрупованням. Це спосіб розбивки відношення за значеннями атрибута, зазначеного в реченні GROUP BY, і умов у речення WHERE.

Запит 4: *Знайти прізвища лікарів, що лікують більше п'яти пацієнтів.*

```
SELECT ІМ'Я
FROM ЛІКАР
WHERE НСС=
        SELECT НСС
        FROM ЛІКУЄ GROUP BY НС
        WHERE COUNT (ПЦТ#) > 5
```

Запит 5: *Знайти номери пацієнтів, що лікуються більш ніж в одного лікаря.*

```
SELECT ПЦТ#
FROM ЛІКУЄ GROUP BY ПЦТ#
WHERE COUNT (НСС)> 1
```

Селекція з кореляцією. Кореляція подібна до угруповання в тому розумінні, що використовується єдине відношення й вільні змінні, як у реляційному вирахуванні. При кореляції, однак, кортежі співвідносяться, а не групуються, так що її можна розглядати як операцію із двома курсорами: у той час як ведучий (зовнішній) курсор указує на поточний кортеж, внутрішній курсор переглядає всі кортежі відношення, порівнюючи кожного з них із зовнішнім поточним кортежем. Операція закінчується, коли провідний курсор пробіжить всі кортежі відношення.

Запит 6: Знайти всі пари прізвищ лікарів, що мають однакові спеціальності.

```
SELECT Р1. ІМ'Я, Р2.ІМ'Я
FROM ЛІКАР Р1, ЛІКАР Р2
WHERE Р1.СПЕЦІАЛЬНІСТЬ = Р2.СПЕЦІАЛЬНІСТЬ
AND   Р1.ІМ'Я < Р2.ІМ'Я
```

або

```
В1: SELECT ІМ'Я, В2.ІМ'Я
    FROM ЛІКАР
    WHERE СПЕЦІАЛЬНІСТЬ=
    В2:    SELECT СПЕЦІАЛЬНІСТЬ
           FROM ЛІКАР
           WHERE ІМ'Я> В1.ІМ'Я
```

У першому випадку додаткова умова після **AND** робить неможливими варіанти зі збігом і перестановкою прізвищ. У другому випадку використовуються *мітки блоків,* що подібно використанню двох курсорів.

Селекція зі скалярною квантифікацією

Запит 7: *Знайти прізвища службовців, що заробляють більше будь-якої медсестри.*

```
SELECT ІМ'Я
FROM ПЕРСОНАЛ
WHERE ЗАРПЛАТА > ALL
        SELECT ЗАРПЛАТА
        FROM ПЕРСОНАЛ
        WHERE ПОСАДА=МЕДСЕСТРА
```

При порівнянні зарплат можна використати наступні варіанти речення **WHERE**:

```
WHERE ЗАРПЛАТА> ANY                  або  WHERE ЗАРПЛАТА >
      SELECT ЗАРПЛАТА                       SELECT MAX
(ЗАРПЛАТА)
```

Якщо в запиті необхідні прізвища службовців, що заробляють більше, ніж яка-небудь із медсестер, порівняння зарплат буде наступним:

```
WHERE ЗАРПЛАТА > SOME          або  WHERE ЗАРПЛАТА >
      SELECT ЗАРПЛАТА                 SELECT MIN
(ЗАРПЛАТА)
```

Після оператора **WHERE** в умовах можуть використовутись агрегатні функції (тобто SUM, COUNT, MAX, MIN, AVERAGE (AVR)).

Селекція з бінарними операціями над безлічами

Запит 8. *Знайти прізвища лікарів, що не лікують пацієнта під номером О155.*

SELECT UNIQUE ІМ'Я
FROM ЛІКАР
DIFFERENCE
SELECT ІМ'Я
FROM ЛІКАР
WHERE НСС =
SELECT НСС
FROM ЛІКУЄ
WHERE ПЦТ# = “0155”

або

SELECT ІМ'Я
FROM ЛІКАР
WHERE “0155” ≠**ALL**
(SELECT ПЦТ#
FROM ЛІКУЄ
WHERE НСС = ЛІКАР.НСС)

У мові можна також використовувати й теоретико-множинні операції ОБ'ЄДНАННЯ й ПЕРЕТИНАННЯ. У другому варіанті застосована негативна форма квантора загальності.

Відновлення. Зразки запитів для відновлення й видалення кортежів відношення наведені нижче. Для відновлення:

UPDATE відношення
SET атрибут-значення
WHERE будь-який запит на **SQL**

Для видалення:

DELETE відношення
WHERE будь-який запит на **SQL**

Тут речення **WHERE** визначають множину кортежів, які будуть видалятися або змінювати значення своїх атрибутів.

При додаванні запису у таблицю на відміну від видалення в запиті повинне бути зазначене значення кортежу, що додається. Мова SQL дозволяє виконувати всі необхідні операції над таблицями бази даних, такі як, наприклад, створення таблиці; додавання записів у таблицю; модифікація структури таблиці, і т.д. Деякі SQL-сервери мають розвинений інтерфейс для зручної роботи в них користувачів різної кваліфікації. Але подібні програмні продукти коштують досить дорого. Вільно розповсюджувані SQL-сервери не мають зручного інтерфейсу, і для роботи з ними потрібний досвід роботи з комп'ютером у режимі командного рядка. Тому іноді зручніше користуватися не SQL-серверами, а іншими реляційнми СКБД, які мають розвитий інтерфейс і використають можливості мови SQL.

В СКБД Visual FoxPro дії для додавання, видалення, редагування записів таблиць (кортежів відношень) зручніше виконувати візуальними засобами, тобто за допомогою інтерфейсу користувача, який надає ця СКБД. Тому в даному курсі команди SQL для виконання цих дій розглядаються лише конспективно. Доклано вони будуть розглядатися у наступному курсі “Розробка прикладних баз даних”, який присвячений роботі з базами даних у клієнт/серверних СКБД.

11.1 SQL запити в FoxPro

В FoxPro весь SQL-запит повинен бути однією командою. Це означає, що весь запит потрібно написати в одному рядку, або при записі в кілька рядків ставити “;” наприкінці кожного рядка, крім останньої. Якщо не зазначено, що запит повинен бути записаний у таблицю (відношення), то обрані запитом кортежі містяться у віртуальній таблиці у вікні з ім'енем **Query**.

Наприклад: запит 2: *Знайти прізвища пацієнтів, що лікуються в лікаря із НСС = 995-42-7515,* – буде виглядати в FoxPro у такий спосіб:

```
SELECT ІМ'Я;
FROM ПАЦІЄНТ, ЛІКУЄ;
WHERE ПАЦІЄНТ. ПЦТ# = ЛІКУЄ. ПЦТ# ;
      AND ЛІКУЄ.НСС =”995-42-7515”
```

Правда, це не зовсім запит мовою SQL-FoxPro, тому що не підтримує імена таблиць і атрибутів, які написані кірілецею. Тому для виконання подібного запиту потрібно було б створити базу даних з іменами таблиць і атрибутів, написаних латиницею, а потім написати із цими іменами аналогічний запит.

Після оператора WHERE в умовах можуть використовуватись агрегатні функції (тобто SUM, COUNT, MAX, MIN, AVG (від AVERAGE - середнє)).

У наведеному прикладі з обраних записів не створювалася таблиця на диску. У цьому випадку буде створена віртуальна таблиці й вона буде виведена у вікні з назвою Query. Якщо обрані запитом кортежі потрібно записати на диск у вигляді деякої тимчасової таблиці, то після рядків SELECT … FROM… WHERE… потрібно написати INTO TABLE <ім'я таблиці>**.** Крім того, як при виводі у вікно запиту (**Query**), так і при виводі в нову таблицю, обрані SQL-запитом кортежі можуть бути впорядковані по будь-якому стовпці створеної таблиці. Для цього потрібно дописати в запит речення ORDER:

```
ORDER BY cAtrName | nColumnNumber  [ASC | DESC ]
```

Якщо зазначено слово **DESC**, то запису в запиті впорядковуються за убуванням значень у стовпці. Якщо є слово **ASC**, чи ні ніякого — то за

зростанням . Після опції ORDER BY може стояти або ім'я або номер того стовпця нової таблиці (одержуваної запитом), значення в якому потрібно впорядкувати.

Наведу приклади SQL-запитів для БД із двох таблиць: PERSONAL і STATES.

Структура таблиці PERSONAL

Атрибут	Тип	Призначення
CODE	числовий	табельний номер співробітника
SURNAME	символьний	прізвище співробітника
NAME	символьний	ім'я співробітника
NAME2	символьний	по батькові співробітника
SEX	символьний	стать співробітника (“жін”, “чол”)
BIRTHDAY	дата	дата народження співробітника
CHILDREN	числовий	кількість дітей у співробітника
EDUCATION	символьний	отримана освіта (середня, вища, н/вища)
OTPUSK	дата	дата початку чергової відпустки
COD_POST	числовий	код посади, на якій працює співробітник

Структура таблиці STATES

Атрибут	Тип	Призначення
COD_POST	числовий	код посади
POST	символьний	назва посади
MONEY	числовий	заробітня платня за посадою
DAYS	числовий	кількість днів відпустки

Нехай у нову таблицю потрібно вивести записи про всіх співробітників, які йдуть у відпустку влітку й мають зарплату більше 500 гривень. У новій таблиці повинні бути три колонки: у першій прізвище й ім'я співробітника (в однієй), у другій – номер місяця відпустки, у третій – відпускні виплати. Кортежі повинні виводитися в нову таблицю в порядку місяців відпустки. Нехай відпускні виплати розраховуються за формулою: money*days/30. Нову таблицю з обраними кортежами назвемо NEW. Тоді запит буде мати вигляд:

```
SELECT trim(a.surname)+" "+ trim(a.name) as;
 ”Прізвище Ім'я”,  month(a.otpusk) AS ”Місяць”,;
 b.money*b.days/30 AS ”Відпускні”;
FROM personal a, states b;
WHERE a.code= b.code AND b.money>500;
AND month(a.otpusk)<9 AND month(a.otpusk)>5;
ORDER BY 2 INTO TABLE NEW
```

Тут для зручності запису застосовуються локальні псевдоніми таблиць (по-англійському alias) (у нашому прикладі це `a, b`) замість довгих назв таблиць. При цьому обов'язково в реченні FROM локальний псевдонім повинен бути зазначеним після імені таблиці (personal a).

У даному запиті показано, як установлюється зв'язок між таблицями в SQL-запитах. Це для нашого приклада робить умову `a.code=b.code`. При цьому не потрібно індексувати й зв'язувати таблиці, більше того, для виконання SQL-запиту не обов'язково навіть відкривати таблиці (якщо вони перебувають у поточному для FoxPro каталозі, і він може їх знайти). Але індексування таблиць використовується оптимізатором запитів, тому для прискорення роботи СКБД рекондується створювати індекси за тими виразами, які використовуються для зв'язку між таблицями, а також які використовуються для упорядкування кортежів в таблицях.

Для створення запитів існує **Дизайнер Запитів**. Але грамотно з ним працювати можна тільки одержавши досить великий досвід. Справа в тому, що досить легко створити запит за допомогою дизайнера запитів; але, якщо виявиться, що запит записаний з помилками, відредагувати такий запит буде складніше, ніж створити новий.

Простіше писати SQL-запити в програмному вікні. Для цього потрібно створити програму; для цього потрібно або в командному вікні написати команду

MODIFY COMMAND

або виконати пункт меню **File/New/Program**. Або нажати кнопку на панелі інструментів, на якій зображений чистий аркуш, і знов-таки вибрати у вікні, що з'являється**,** пункт **Program.**

Після написання в програмному вікні запиту, його потрібно зберегти на диск (указавши ім'я файлу). По цьому імені файлу потім запит посилається на виконання командою:

DO <ім'я файлу>

Якщо програмне вікно із запитом закрили, і потрібно його знову відкрити, то шукати файл із запитом потрібно буде серед програмних файлів (з розширенням .PRG).

Якщо в запиті є помилки, він не буде виконаний, і FoxPro виведе віконце з повідомленням про помилку.

Наступний приклад: у нову таблицю з іменем TEMP вибрати записи про всіх співробітників з дітьми за умови, що вік співробітника на кінець року досягне 60 років, а освіта співробітника “середня”. У нову таблицю вибираються колонки із прізвищем співробітника, кількістю дітей, віком, посадою, зарплатою. Записи повинні бути впорядковані за убуванням зарплати:

```
SELECT a.surname, a.children, year(date())-year(a.birthday) ;
       AS age,; b.post, b.money ;
FROM personal a, states b;
WHERE a.code=b.code AND (a.children>0 AND ;
year(date())-year(a.birthday)>59 AND  a.education=”середня”);
ORDER BY money DESC ;
INTO TABLE TEMP
```

Таблиці, створені вибіркою даних з бази, звичайно називають *представленнями* (View). Не рекомендується редагувати подібні таблиці - якщо дані змінилися, то зміни повинні бути відбиті в первісних таблицях, з яких робили вибірку за допомогою запиту.

12 ЗАХИСТ І ЦІЛІСНІСТЬ ДАНИХ У СКБД

Дані є коштовним ресурсом, доступ до якого необхідно строго контролювати й регламентувати, так само як і до будь-яких інших корпоративних ресурсів. Частина або навіть всі корпоративні дані можуть мати стратегічне значення для організації й тому повинні зберігатися в секреті, під надійною охороною.

У розділі 2 були описані особливості середовища бази даних, зокрема типові функції й служби сучасної СКБД. До складу цих функцій і служб входять і засоби авторизації користувачів – механізм, за допомогою якого СКБД гарантує, що доступ до бази даних зможуть одержати тільки користувачі, яким було надано відповідне право. Інакше кажучи, будь-яка СКБД повинна гарантувати, що створена в її середовищі база даних буде надійно захищена. Термін *захист* відноситься до захищеності баз даних від несанкціонованого доступу. Однак тема захисту баз даних включає не тільки стандартні служби, які надає цільова СКБД, але й набагато більш широке коло питань, що мають відношення до захищеності самих баз даних і всього їхнього оточення.

12.1 Захист бази даних

Термін «захист» використовується для позначення засобів запобігання несанкціонованого доступу до системних ресурсів і їхнього використання. Крім засобів захисту основних ресурсів, які забезпечує операційна система, СКБД підтримує деякі спеціальні види захисту. Можна говорити про захист даних у СКБД, оскільки цей захист необов'язковий для обчислювальної системи і її операційної системи.

У цьому розділі ми познайомимося з усіма проблемами, пов'язаними із захистом баз даних, і з'ясуємо, чому організації повинні ставитися до потенційних погроз їхнім комп'ютерним системам з усією серйозністю. Крім того, ми уточнимо всі типи небезпек, які можуть загрожувати самим комп'ютерам і різним комп'ютерним системам [11].

Захист бази даних. Забезпечення захищеності бази даних проти будь-яких навмисних або ненавмисних погроз за допомогою різних засобів.

Поняття захисту застосовне не тільки до даних, що зберігаються в базі даних. Пробоїни в системі захисту можуть виникати й в інших частинах системи, що, у свою чергу, наражає на небезпеку й саму базу даних. Отже, захист бази даних повинен охоплювати встаткування, що використовується, програмне забезпечення, персонал і власне дані. Для ефективної реалізації захисту необхідні відповідні засоби контролю, які визначаються конкретними вимогами, що випливають із особливостей системи, що експлуатується. Необхідність захищати дані, яку раніше дуже часто відкидали або якою просто нехтували, у цей час вся ясніше

усвідомлюється різними організаціями. Привід для такої зміни настроїв полягає в випадках руйнування комп'ютерних сховищ корпоративних даних, які трапляються все частіше, а також в усвідомленні того, що втрата або просто тимчасова неприступність цих даних може стати причиною дійсної катастрофи.

База даних являє собою найважливіший корпоративний ресурс, що повинен бути належним чином захищений за допомогою відповідних засобів контролю. Ми обговоримо проблеми захисту бази даних з погляду таких потенційних небезпек:

- викрадення й фальсифікація даних;
- втрата конфіденційності (порушення таємниці);
- порушення недоторканності особистих даних;
- втрата цілісності;
- втрата доступності.

Зазначені ситуації відзначають основні напрямки, у яких керівництво повинне вживати заходів, що знижують ступінь ризику, тобто потенційну можливість втрати або ушкодження даних. У деяких ситуаціях всі відзначені аспекти ушкодження даних тісно зв'язані між собою, так що дії, спрямовані на порушення захищеності системи в одному напрямку, часто приводять до зниження її захищеності у всіх іншим. Крім того, деякі події, наприклад порушення недоторканності особистих даних або фальсифікація інформації, можуть виникнути внаслідок як навмисних, так і ненавмисних дій і необов'язково будуть супроводжуватися якими-небудь змінами в базі даних або системі, які можна буде виявити тим або іншим способом.

Викрадення й фальсифікація даних можуть відбуватися не тільки в середовищі бази даних – вся організація так чи інакше піддана цьому ризику. Однак дії по викраденню або фальсифікації інформації завжди відбуваються людьми, тому основна увага повинна бути зосереджене на скороченні загальної кількості зручних ситуацій для виконання подібних дій. Викрадення й фальсифікація не обов'язково пов'язані зі зміною яких-небудь даних, що справедливо й відносно втрати конфіденційності або порушення недоторканності особистих даних.

Поняття конфіденційності означає необхідність збереження даних у таємниці. Як правило, конфіденційними вважаються тільки ті дані, які є важливими для всієї організації, тоді як поняття недоторканності даних стосується вимоги захисту інформації про окремих співробітників. Наслідком порушення в системі захисту, що викликав втрату конфіденційності даних, може бути втрата надійних позицій у конкурентній боротьбі, тоді як наслідком порушення недоторканності особистих даних можуть стати судові дії, початі у відношенні організації.

Втрата цілісності даних приведе до перекручування або руйнування даних, що може мати самі серйозні наслідки для подальшої роботи організації. У цей час безліч організацій функціонує в безперервному режимі, надаючи свої послуги клієнтам 24 години на добу й 7 днів у тиждень. Втрата доступності даних буде означати, що або дані, або система, або й те й інше одночасно виявляться недоступними користувачам, а це може наразити на небезпеку фінансове становище організації. У деяких випадках ті події, які послужили причиною переходу системи в недоступний стан, можуть одночасно викликати й руйнування даних у базі.

Ціль захисту бази даних – мінімізувати втрати, викликані заздалегідь передбаченими подіями. Прийняті рішення повинні забезпечувати ефективне використання понесених витрат і виключати зайве обмеження можливостей, що надаються користувачам. Останнім часом рівень комп'ютерної злочинності істотно зріс, причому прогнози на майбутнє не втішливі й передвіщають продовження його росту й надалі.

12.1.1 Основні типи погроз

Погроза. Будь-яка ситуація (або подія), викликана навмисно або ненавмисно, яка здатна несприятливо вплинути на систему, а отже, і на роботу всієї організації.

Погроза може бути викликана ситуацією (або подією), здатної принести шкоду організації, причиною якого може служити людина, подія або збіг обставин. Збиток може бути матеріальним (наприклад, втрата встаткування, програмного забезпечення або даних) або нематеріальним (наприклад, втрата довіри партнерів або клієнтів). Перед кожною організацією постає проблема з'ясування всіх можливих небезпек, що в деяких випадках є досить непростим завданням. Тому на виявлення хоча б найважливіших погроз може знадобитися досить багато часу й зусиль, що варто враховувати.

У попередньому розділі ми вказали основні області, у яких варто очікувати втрат у випадку навмисної або ненавмисної події. Хоча деякі типи погроз можуть бути навмисні або ненавмисними, результати їхнього впливу залишаються однаковими. Навмисні погрози завжди здійснюються людьми й можуть бути зроблені користувачами, які як володіють, так і не володіють правами доступу, причому деякі з них можуть навіть не бути співробітниками організації.

Будь-яка погроза повинна розглядатися як потенційна можливість порушення системи захисту, що у випадку успішної реалізації може зробити той або інший негативний вплив. У табл. 12.1 наведені приклади різних типів погроз із вказівкою тих областей, у яких вони можуть впливати на систему.

Таблиця 12.1 Приклади можливих погроз

Небезпека	Викрадення й фальсифікація даних	Втрата конфіденційності	Порушення недоторканності особистих даних	Втрата цілісності	Втрата доступності
Використання прав доступу іншої особи	+		+		
Несанкціонована зміна або копіювання даних	+				
Зміна програм	+			+	+
Непродумані методики й процедури, що допускають змішування конфіденційних і звичайних даних в одному документі	+	+	+		
Підключення до кабельних мереж	+	+	+		
Уведення зломщиками некоректних даних	+	+	+		
Шантаж	+	+	+		
Створення "лазівок" у системі	+	+	+		
Викрадення даних, програм і устаткування	+	+	+		+
Відмова систем захисту, що викликало перевищення припустимого рівня доступу		+	+	+	
Брак персоналу й страйк				+	+
Недостатня навченість персоналу		+	+	+	+
Перегляд і розкриття засекречених даних	+	+	+		
Електронні перешкоди й радіація				+	+
Руйнування даних у результаті відключення або перенапруги в мережі електроживлення				+	+
Пожежі (через короткі замикання, ударів блискавок, підпалів), повені, диверсії				+	+
Фізичне ушкодження устаткування				+	+
Обрив або від'єднання кабелів				+	+
Впровадження комп'ютерних вірусів				+	+

Наприклад, наслідком перегляду й розкриття засекречених даних можуть стати викрадення й фальсифікація, втрата конфіденційності й порушення недоторканності особистих даних в організації.

Рівень втрат, понесених організацією в результаті реалізації деякої погрози, залежить від багатьох факторів, включаючи наявність заздалегідь продуманих контрзаходів і планів подолання непередбачених обставин. Наприклад, якщо відмова встаткування викликала руйнування вторинних сховищ даних, вся робота в системі повинна бути згорнута до повного усунення наслідків аварії. Тривалість періоду бездіяльності й швидкість відновлення бази даних залежать від декількох факторів, включаючи час створення останньої резервної копії й тривалість роботи по відбудові системи.

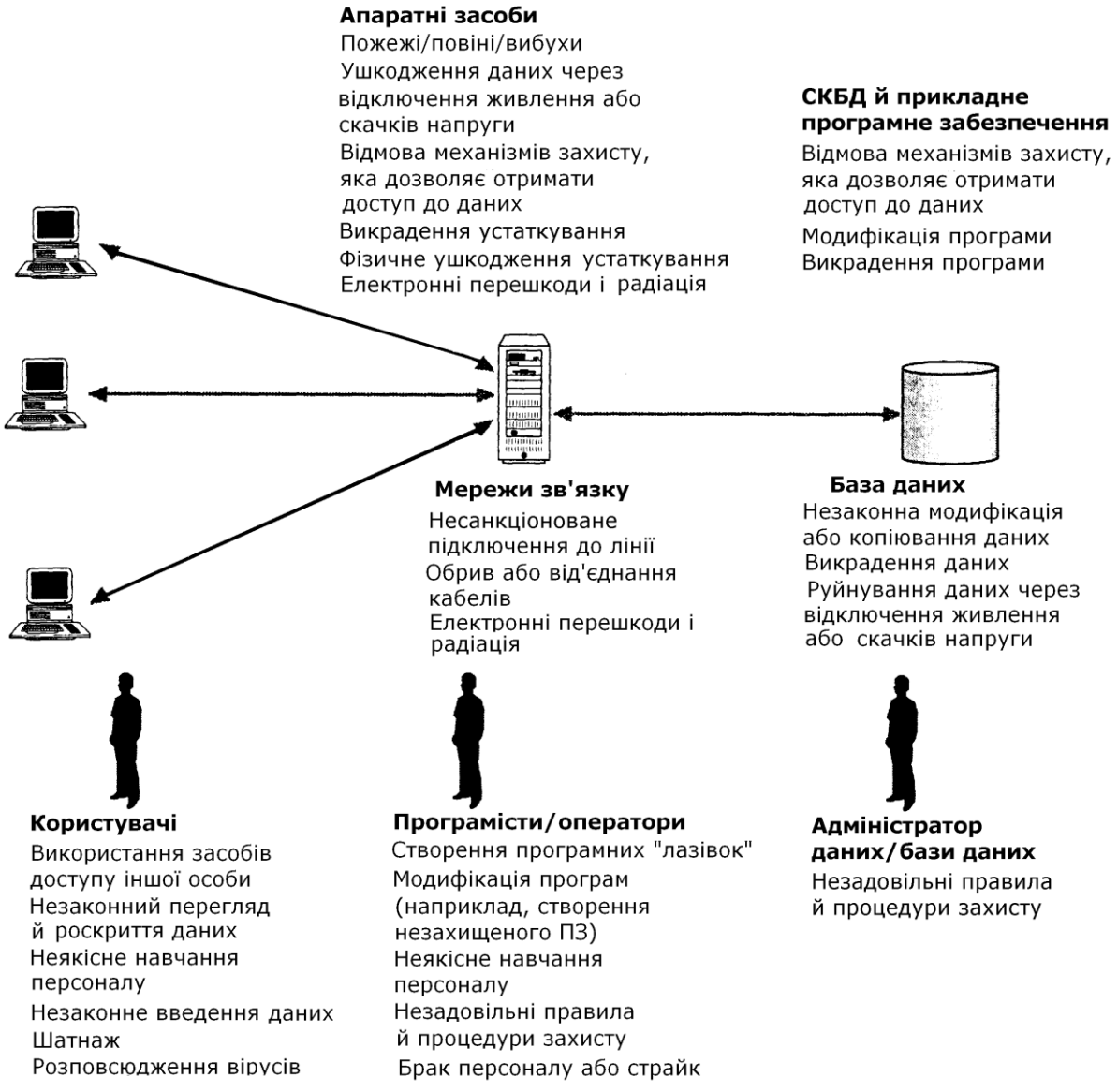


Рис. 12.1. Узагальнена схема потенційних погроз, яким піддані комп'ютерні системи

Будь-яка організація повинна встановити типи можливих погроз, яким може піддатися її комп'ютерна система, після чого розробити відповідні плани й необхідні контрзаходи, з оцінкою рівня витрат, необхідних для їхньої реалізації. Безумовно, витрати часу, зусиль і грошей навряд чи виявляться ефективними, якщо вони будуть стосуватися потенційних небезпек, здатних заподіяти організації лише незначний збиток. Ділові процеси організації можуть бути піддані таким небезпекам, які неодмінно варто враховувати, однак частина з них може мати місце у винятково рідких ситуаціях. Проте навіть настільки малоймовірні обставини повинні бути прийняті в увагу, особливо якщо їхній вплив може виявитися досить істотним.

12.2 Контрзаходи – комп'ютерні засоби контролю

Відносно погроз, які можуть зробити негативний вплив на роботу комп'ютерних систем, повинні бути прийняті контрзаходи всіляких типів, починаючи від фізичного контролю й закінчуючи адміністративно-організаційними процедурами. Незважаючи на широкий діапазон комп'ютерних засобів контролю, доступних у цей час на ринку, загальний рівень захищеності СКБД визначається можливостями операційної системи, щ використовується, оскільки робота цих двох компонентів тісно зв'язана між собою. Загальна схема типової комп'ютерної системи багатьох користувачів представлена на рис. 12.2.

У цьому розділі описані різні комп'ютерні засоби контролю, доступні в середовищі багатьох користувачів. Звичайно для персонального комп'ютера доступні не всі перераховані нижче типи засобів контролю.

- Авторизація користувачів.
- Застосування представлень.
- Резервне копіювання й відновлення.
- Підтримка цілісності.
- Шифрування.
- Застосування RAID-масивів.

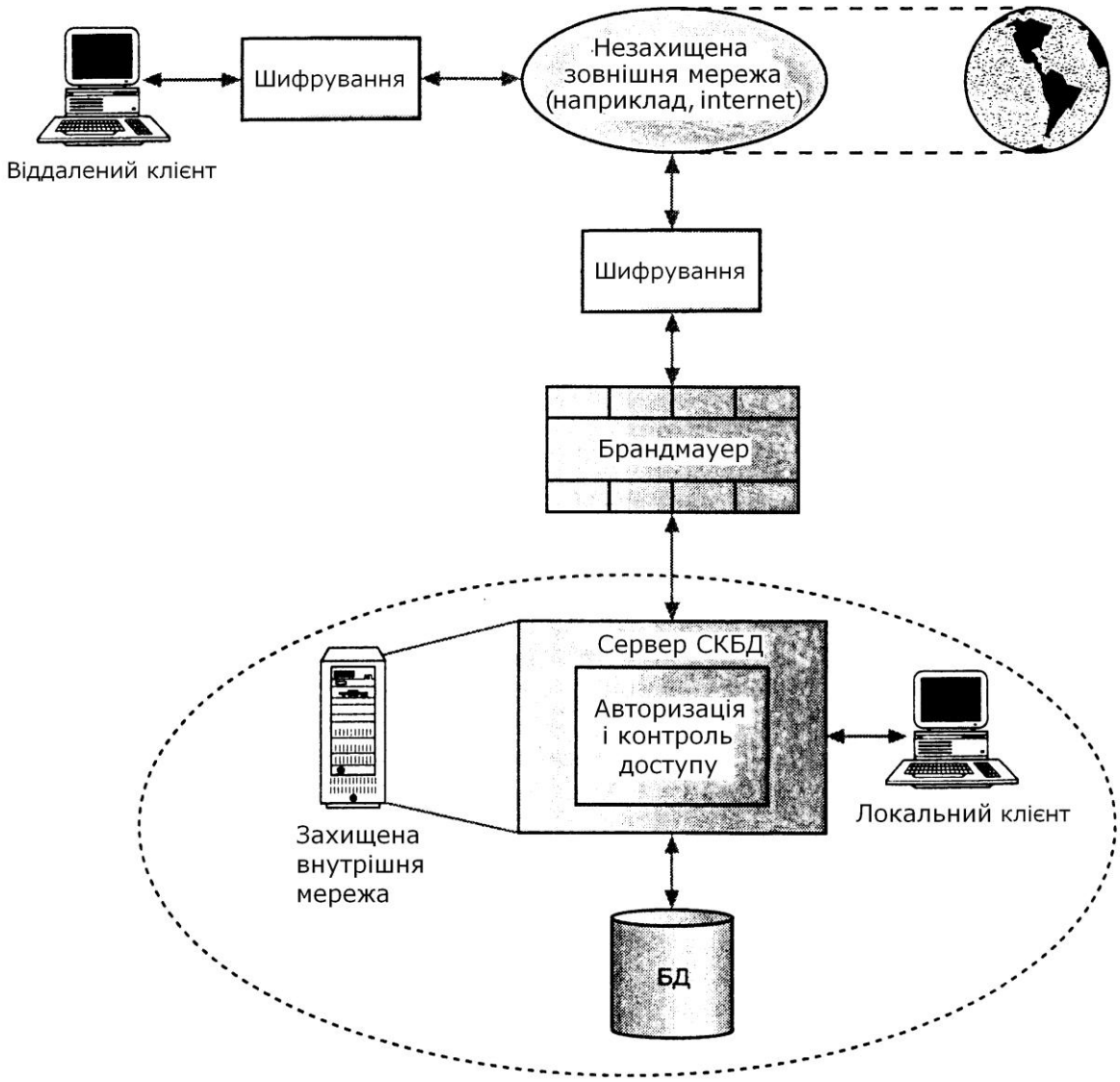


Рис.12.2 Загальна схема типової комп'ютерної системи багатьох користувачів

12.2.1. Авторизація користувачів

Авторизація. Надання прав (або привілеїв), що дозволяють їхньому власникові мати законний доступ до системи або до її об'єктів.

Засоби авторизації користувачів можуть бути убудовані безпосередньо в програмне забезпечення й керувати не тільки наданими користувачам правами доступу до системи або об'єктів, але й набором операцій, які користувачі можуть виконувати з кожним доступним йому об'єктом. Із цієї причини механізм авторизації часто називають *засобами керування доступом*. Термін "власник" у наведеному вище визначенні може позначати користувача – людину або програму. Термін "об'єкт" може позначати таблицю даних, представлення, додаток, процедуру,

тригер або будь-який інший об'єкт, що може бути створений у рамках системи.

Аутентифікація. Механізм визначення того, чи є користувач тим, за кого себе видає.

За надання користувачам доступу до комп'ютерної системи звичайно відповідає системний адміністратор, до обов'язку якого входить створення облікових записів користувачів. Кожному користувачеві привласнюється унікальний ідентифікатор, що використовується операційною системою для визначення того, хто є хто. З кожним ідентифікатором зв'язується певний пароль, обраний користувачем і відомий операційній системі. При реєстрації користувач повинен надавати системі свій пароль для виконання перевірки (аутентифікації) того, чи є він тим, за кого себе видає.

Подібна процедура дозволяє організувати контрольований доступ до комп'ютерної системи, але не обов'язково надає право доступу до СКБД або іншої прикладної програми. Для одержання користувачем права доступу до СКБД може використовуватись окрема така процедура. Відповідальність за надання прав доступу до СКБД звичайно несе адміністратор бази даних (АБД), до обов'язку якого входить створення окремих ідентифікаторів користувачів, але цього разу вже в середовищі самої СКБД.

У деяких СКБД ведеться список ідентифікаторів користувачів і пов'язаних з ними паролів, що відрізняється від аналогічного списку, що підтримується операційною системою. В інших типах СКБД ведеться список, записи якого звіряються із записами списку користувачів операційної системи з обліком поточного реєстраційного ідентифікатора користувача. Це запобігає спроби користувачів зареєструватися в середовищі СКБД під ідентифікатором, відмінним від того, котрий вони використали при реєстрації в системі.

Привілеї

Як тільки користувач одержить право доступу до СКБД, йому можуть автоматично надаватися різні привілеї, пов'язані з його ідентифікатором. Зокрема, ці привілеї можуть включати дозвіл на доступ до певних баз даних, таблицям, представленням і індексам, а також на створення цих об'єктів або ж право викликати на виконання різні утиліти СКБД. Привілеї надаються користувачам, щоб вони могли виконувати завдання, які ставляться до кола їхніх посадових обов'язків. Надання зайвих або непотрібних привілеїв може привести до порушення захисту, тому користувач повинен одержувати тільки такі привілеї, без яких він не має можливості виконувати свою роботу.

Деякі типи СКБД функціонують як *закриті системи*, тому користувачам крім дозволу на доступ до самої СКБД буде потрібно мати окремі дозволи й на доступ до конкретних її об'єктів. Ці дозволи видаються або АБД, або власниками певних об'єктів системи. На противагу цьому, *відкриті системи* за замовчуванням надають користувачам, що пройшли перевірку їхньої дійсності, повний доступ до всіх об'єктів бази даних. У цьому випадку привілею встановлюються за допомогою явного скасування тих або інших прав конкретних користувачів. Методи авторизації користувачів і надання їм привілеїв засобами мови SQL розглядаються в розділі 6.6.

Права володіння й привілеї

Деякі об'єкти в середовищі СКБД належать самої СКБД. Звичайно ця приналежність оформлена за допомогою спеціального ідентифікатора суперкористувача, наприклад адміністратора бази даних. Як правило, володіння деяким об'єктом надає його власникові весь можливий набір привілеїв відносно цього об'єкта. Це правило застосовується до всіх авторизованих користувачів, що одержують права володіння певними об'єктами. Будь-який знову створений об'єкт автоматично передається у володіння його творцеві, що і одержує весь можливий набір привілеїв для даного об'єкта. Хоча при цьому користувач може бути власником деякого представлення, єдиним привілеєм, що буде надана йому відносно цього об'єкта, може виявитися право вибірки даних з даного представлення. Причина подібних обмежень полягає в тому, що зазначений користувач має обмежений набір прав відносно базових таблиць створеного їм представлення. Приналежному власникові привілеї можуть бути передані ним іншим авторизованим користувачам. Наприклад, власник декількох таблиць бази даних може надати іншим користувачам право вибірки інформації із цих таблиць, але не дозволити їм вносити в таблиці які-небудь зміни. У мові SQL передбачено, що якщо користувач передає які-небудь привілеї, він може вказати, чи здобуває одержувач цих привілеїв право передавати ці привілеї іншим користувачам.

Найпростішою формою фіксації прав доступу в системі є таблиця, або матриця, прав доступу. Рядки таблиці відповідають різним користувачам, а стовпці – даним. Кожний елемент такої таблиці може зберігати окрему інформацію про права доступу. На рис. 12.3 показана подібна матриця, що не містить предикатів.

Дані / Користувачі	D_1	D_2	. . .	D_n
Користувач 1	Читання	Відновлення		Читання/ відновлення
Користувач 2	Не дозволено	Читання		Не дозволено
. . .			. . .	
Користувач m	Не дозволено	Читання/ відновлення		Читання

Рис. 12.3 Матриця доступу.

Стовпці D_1, ..., D_n можуть відповідати типам записів (файл, відношення, сегмент) або атрибутам. Кожний рядок матриці визначає типи записів або атрибути (поля) даного типу запису, до яких користувачеві дозволений (не дозволений) доступ (маніпулювання). При використанні предикатів рівень контролю (ступінь деталізації) може досягати окремих значень полів (або множини значень). Наприклад, користувач може не мати права доступу до атрибута ЗАРПЛАТА зі значеннями більше 30000. У більш складних ситуаціях для захисту даних можна використати механізм зовнішніх схем. Наприклад, якщо на рис. 12.3 D_1 ..., D_n позначають атрибути типів запису, то представлення користувача типу запису, оголошений у підсхемі, може бути відбите в матриці доступу. Відповідно до рис. 12.3**,** наприклад, представлення другого користувача, що збігає з його правами доступу, містить тільки атрибут D_2.

Якщо СКБД підтримує кілька різних типів ідентифікаторів авторизації, з кожним з існуючих типів можуть бути зв'язані різні пріоритети. Зокрема, якщо СКБД підтримує використання ідентифікаторів окремих користувачів і груп, тоді, як правило, ідентифікатор користувача буде мати більше високий пріоритет, чим ідентифікатор групи. Приклад визначення ідентифікаторів користувачів і груп у подібної СКБД наведений у табл. 12.2.

Таблиця 12.2. Ідентифікатори користувачів і груп

Ідентифікатор користувача	**Тип**	**Група**	**Ідентифікатор члена групи**
SG37	Користувач	Sales	SG37
SG14	Користувач	Sales	SG14
SG5	Користувач		
Sales	Група		

У стовпцях *Ідентифікатор користувача* й *Тип* наведені визначені в системі ідентифікатори й вказується їхній тип – окремий користувач і група користувачів. Стовпці із заголовками *Група* й *Ідентифікатор члена групи* містять відомості про групу, який належить користувач, і про його ідентифікатор в цій групі. З кожним конкретним ідентифікатором може бути зв'язаний конкретний набір привілеїв, що визначає тип доступу до окремих об'єктів бази даних (наприклад, для вибірки (Select), відновлення (Update), вставки (Insert), видалення (Delete) або всі типи відразу (All)). З кожним типом привілею зв'язується деяке двійкове значення, наприклад:

SELECT	UPDATE	INSERT	DELETE	ALL
0001	0010	0100	1000	1111

Окремі двійкові значення підсумовуються, і отримана сума однозначно характеризує, які саме привілеї (якщо вони передбачені) має кожний конкретний користувач або група відносно певного об'єкта бази даних. У табл. 12.3 наведений приклад матриці контролю доступу, у якій визначений набір привілеїв групи `Sales` і користувачів `SG37, SG5`.

Таблиця 12.3. Таблиця контролю доступу

Ідентифікатор користувача	**property No**	**type**	**price**	**owner No**	**staff No**	**branch No**	**Ліміт рядків, що обираються**
Sales	0001	0001	0001	0000	0000	0000	15
SG37	0101	0101	0111	0101	0111	0000	100
SG5	1111	1111	1111	1111	1111	1111	немає

Вміст таблиці показує, що групі користувачів з ідентифікатором `Sales` наданий тільки привілей `Select` (код 0001) відносно атрибутів `propertyNo`, `type` і `price`. Крім того, для неї встановлений максимальний розмір результуючого набору даних запиту, рівним 15 рядкам. Користувач `SG14` є членом цієї групи й не має яких-небудь додаткових привілеїв доступу, тому він користується тільки тими правами, які надані даній групі. З іншого боку, користувач `SG37` має власні привілеї `Select` і `Insert` (визначаються значенням 0001 + 0100 = 0101) відносно атрибутів `propertyNo`, `type` і `ownerNo`, а також привілей `Select`, `Update` і `Insert` (значення 0001 + 0010 + 0100 = 0111) відносно атрибутів `price` і `staffNo`. Крім того, для цього користувача максимальний розмір результуючого набору даних запиту встановлений рівним 100 рядкам. Користувачеві `SG5` надані привілеї `Select`, `Update`, `Insert` і `Delete` (значення 0001 + 0010 + 0100 + 1000 = 1111), тобто привілей `All` для доступу до всіх атрибутів таблиці, а розмір результуючих наборів даних запитів не обмежується.

Зазвичай для реалізації механізмів контролю доступу СКБД використовуються подібні матриці, хоча окремі деталі в різних системах можуть відрізнятися. У деяких СКБД користувачеві дозволяється вказувати, під яким ідентифікатором він має намір працювати далі, – це доцільно в тих випадках, коли той самий користувач може бути членом відразу декількох груп. Дуже важливо освоїти всі механізми авторизації й інші засоби захисту, що надаються цільовий СКБД. Особливо це важливо для тих систем, у яких існують різні типи ідентифікаторів і допускається передача права присвоєння привілеїв. Це дозволить правильно вибирати типи привілеїв, які надаються окремим користувачам, виходячи з обов'язків, які ними виконують, і набору прикладних програм, що використовуються.

Доступ до об'єкта й маніпулювання їм можуть залежати й від умови (предиката). Предикати, що використовуються в правилах доступу, можуть залежати або не залежати від даних. У першому випадку перевірка прав доступу здійснюється під час компіляції без доступу до даних. Наприклад, правила доступу, що обмежують *час доступу* й *розташування користувача* (номер термінала), можуть перевірятися без звертання до даних.

У предикатах, що залежать від даних, перевірка виконується тільки під час виконання програми з істотними витратами на організацію доступу до даних. Предикат може містити просту умову, що залежить від даних, наприклад:

ПЕРСОНАЛ: ЗАРПЛАТА < 30 000 дол.

У той же час предикати можуть визначати контекст, тобто умова може містити кілька атрибутів і/або деякий взаємозв'язок між атрибутами. Предикати з контекстною залежністю не можуть запобігти, однак, можливості доступу до даних за допомогою логічних виводів. Логічний доступ виконується за допомогою серії припустимих операцій агрегатного типу, що приводить до одержання необхідних даних.

Для об'єднання предикатів, що залежать від даних, із правилами доступу користувальницький запит необхідно змінити. Це забезпечується додаванням до предиката запиту (умови) додаткових предикатів, що містять застосовні правила доступу. Після визначення правил доступу користувальницький запит модифікується в такий спосіб:

Вихідний запит: $\underbrace{S_1, \ldots, S_m}$: $\underbrace{Q_1, \ldots, Q_n}$

специфікація *умова*

Модифікований запит;

$S_1, \ldots, S_m$: $(Q_1, \ldots, Q_n) \wedge \underbrace{(P_1 \vee P_2 \vee \ldots P_k)}$

Предикати правила доступу

У модифікованому запиті диз'юнкція предикатів правил доступу, що застосовуються, з'єднується за допомогою конъюнкции з вихідними умовами.

12.2.2 Представлення (підсхеми)

Представлення. Динамічний результат однієї або декількох реляційних операцій з базовими відношенями 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.

Механізм представлень служить потужним і гнучким інструментом організації захисту даних, що дозволяє приховувати від окремих користувачів деякі частини бази даних. У результаті користувачі не будуть мати ніяких відомостей про існування будь-яких атрибутів або рядків даних, які не доступні через представлення, що перебувають у їхньому розпорядженні. Представлення може бути визначене на базі декількох таблиць, після чого користувачеві будуть надані необхідні привілеї доступу до цього представлення, але не до базових таблиць. У цьому випадку використання представлення встановлює більше жорсткий механізм контролю доступу, чим звичайне надання користувачеві тих або інших прав доступу до базових таблиць. Докладне обговорення призначення й методів використання представлень можна знайти в розділах 5.3.6 і 5.6.

Як уже говорилося вище, механізм представлень є складним способом обмеження доступу користувачів до бази даних. У той же час цей спосіб вимагає більш складного контролю цілісності. Це викликано можливістю виникнення побічних ефектів і помилкових операцій при модифікаціях представлення, що ілюструється пропонованим прикладом:

Приклад. Нехай задані два вихідних відношення R_1, R_2 і визначене над ними відношення представлення *V,* що є з'єднанням R_1 і R_2*:*

R_1 (*A, B)*	R_2 (C, *A, D*)	*V (C, A, B)*
a_1 b_1	c_1 a_1 d_1	c_1 a_1 b_1
a_2 b_2	c_2 a_2 d_2	c_2 a_2 b_2
	c_3 a_3 d_3	

Якщо ввести в представлення новий кортеж $\langle c_4, a_3, b_3 \rangle$, то це викличе наступні зміни:

а) Додавання кортежу $\langle a_3, b_3 \rangle$ в R_1.

б) Додавання кортежу $\langle c_4, a_3, ? \rangle$ в R_2 (? позначає невизначене значення).

в) Внаслідок а) у представлення V буде включений додатково кортеж $\langle c_3, a_3, b_3 \rangle$.

Із приклада видно, що відсутність обмежень на визначення представлення може привести до неузгодженості бази даних при модифікаціях. В System R модифікація представлень, отриманих шляхом з'єднань, просто заборонена.

12.3 Резервне копіювання й відновлення

Резервне копіювання. Періодично виконувана процедура одержання копії бази даних і її файлу журналу (а також, можливо, програм) на носії, що зберігається окремо від системи.

Будь-яка сучасна СКБД повинна надавати засоби резервного копіювання, що дозволяють відновлювати базу даних у випадку її руйнування. Крім того, рекомендується створювати резервні копії бази даних і її файлу журналу з деякою встановленою періодичністю, а також організовувати зберігання створених копій у місцях, забезпечених необхідним захистом. У випадку аварійної відмови, у результаті якого база даних стає непридатної для подальшої експлуатації, резервна копія й зафіксована у файлі журналу оперативна інформація використовуються для відновлення бази даних до останнього погодженого стану.

Ведення журналу. Процедура створення й обслуговування файлу журналу, що містить відомості про всі зміни, внесених у базу даних з моменту створення останньої резервної копії, і призначеного для забезпечення ефективного відновлення системи у випадку її відмови.

СКБД повинна надавати засоби ведення системного журналу, у якому будуть фіксуватися відомості про всі зміни стану бази даних і про хід виконання поточних транзакцій, що необхідно для ефективного відновлення бази даних у випадку відмови. Переваги використання подібного журналу полягають у тім, що у випадку порушення роботи або відмови СКБД базу даних можна буде відновити до останнього відомого погодженого стану, скориставшись останньою створеною резервною копією бази даних і оперативною інформацією, що втримується у файлі журналу. Якщо в системі, що відмовила, функція ведення системного журналу не використовулася, базу даних можна буде відновити тільки до того стану, що було зафіксовано в останній створеній резервній копії. Всі зміни, внесені в базу даних після створення останньої резервної копії, будуть загублені.

12.4 Підтримка цілісності

Засоби підтримки цілісності даних також вносять певний вклад у загальну захищеність бази даних, оскільки вони повинні запобігти

переходу даних у неузгоджений стан, а виходить, виключити погрозу одержання помилкових або неправильних результатів розрахунків. Засоби підтримки цілісності даних розглядалися в розділах 2.6.8 і 5.4.

Керування цілісністю є невід'ємною функцією СКБД і забезпечує підтримку бази даних у погодженому стані при колективному режимі роботи з паралельним маніпулюванням даними. Цілісність дуже істотна навіть у випадку однокористувальницької системи, тому що кожне відновлення бази даних повинне задовольняти її структурним і семантичним обмеженням і зберігати дані. Приведемо приклади деяких обмежень:

a) Значення атрибута ЗАРПЛАТА повинне належати множині цілих чисел у діапазоні. 500 < ЗАРПЛАТА < 5 000.
b) База даних не повинна містити однакових екземплярів запису СЛУЖБОВЕЦЬ, або ніякому іншому співробітникові не може бути привласнений номер службовця, запис про який вже є в базі даних.

Існують різні види обмежень цілісності (правила) :

1) Неявні: обмеження визначається обраною моделлю даних. До таких обмежень відносяться повна функціональна залежність в ієрархічних структурах (де підлеглий запис не може існувати без вихідного) або неприпустимість однакових кортежів у відношенні.
2) Явні: вони оголошуються подібно підтримуваним користувачем перемикаючим процедурам, які зв'язують зміну в одному екземплярі запису з іншими екземплярами зв'язаних записів. Наприклад, видалення екземпляра запису ПЕРСОНАЛ неминуче тягне видалення всіх зв'язаних екземплярів у підлеглому типі запису або відсутність даних про постачальника в типі запису ПОСТАЧАЛЬНИК приводить до неможливості зберігання інформації про поставку деталей. Останній випадок називають цілісністю посилання.
3) Область дії (ступінь деталізації) обмеження цілісності може охоплювати тип запису (відношення), екземпляр запису (кортеж) або значення поля (атрибут) з відповідним збільшенням вартості підтримки обмежень.
4) Обмеження можуть бути статичними або динамічними (наприклад, у файлі рахунків баланс не може бути підведений, перш ніж повністю виконається обробка нарахувань).

12.5. Шифрування

Шифрування. Перетворення даних с вікористанням спеціального алгоритму, в результате чого дані стають недоступними для читання будь-якою програмою, яка не має ключа дешифрування.

Якщо в системі з базою даних утримується досить важлива конфіденційна інформація, то має сенс зашифрувати її з метою попередження можливої погрози несанкціонованого доступу із зовнішньої сторони (стосовно СКБД). Деякі СКБД включають засоби шифрування, призначені для використання в подібних цілях. Підпрограми таких СКБД забезпечують санкціонований доступ до даних (після їхнього дешифрування), хоча це буде пов'язане з деяким зниженням продуктивності, викликаним необхідністю подвійного перетворення. Шифрування також може використовуватись для захисту даних при їхній передачі по лініях зв'язку. Існує безліч різних технологій шифрування даних з метою приховання переданої інформації, причому одні з них називають необоротними, а інших — оборотними. Необоротні методи, як і випливає з їхньої назви, не дозволяють установити вихідні дані, хоча останні можуть використовуватись для збору достовірної статистичної інформації. Оборотні технології використаються частіше. Для організації захищеної передачі даних по незахищених мережах повинні використовуватись *системи шифрування*, що включають наступні компоненти:

- ключ шифрування, призначений для шифрування вихідних даних (звичайного тексту);
- алгоритм шифрування, що описує, як за допомогою ключа шифрування перетворити звичайний текст у зашифрований;
- ключ дешифрування, призначений для дешифрування зашифрованого тексту;
- алгоритм дешифрування, що описує, як за допомогою ключа дешифрування перетворити зашифрований текст у звичайний вихідний.

Деякі системи шифрування, що називаються симетричними, використовують той самий ключ як для шифрування, так і для дешифрування, при цьому передбачається наявність захищених ліній зв'язку, призначених для обміну ключами. Однак більшість користувачів не мають доступу до захищених ліній зв'язку, тому для одержання надійного захисту довжина ключа повинна бути не менше довжини самого повідомлення. Проте більшість систем, що експлуатуються, побудовано на використанні ключів, які коротше самих повідомлень. Одна з розповсюджених систем шифрування називається DES (Data Encryption Standard). У ній використовується стандартний алгоритм шифрування,

розроблений фірмою IBM. У цій схемі для шифрування й дешифрування застосовується той самий ключ, що повинен зберігатися в секреті, хоча сам алгоритм шифрування не є секретним. Цей алгоритм передбачає перетворення кожного 64-бітового блоку звичайного тексту з використанням 56-бітового ключа шифрування. Система шифрування DES розцінюється як досить надійна далеко не всіма – деякі розроблювачі думають, що варто було б використати більш довге значення ключа. Так, у системі шифрування PGP (Pretty Good Privacy) використовується 128-бітовий симетричний алгоритм, застосовуваний для шифрування блоків переданих даних.

В теперішній час ключі довжиною до 64 біт розкриваються з достатнім ступенем імовірності урядовими службами розвинених країн, для чого використовується спеціальне устаткування, хоча й досить дороге. Проте протягом найближчих років ця технологія може потрапити в руки організованої злочинності, великих організацій і урядів інших держав. Хоча передбачається, що ключі довжиною до 80 біт також виявляться тими, що розкриваються, у найближчому майбутньому, є впевненість, що ключі довжиною 128 біт у доступному для огляду майбутньому залишаться надійним засобом шифрування. Терміни "сильне шифрування" і "слабке шифрування" іноді використовуються для підкреслення розходжень між алгоритмами, які не можуть бути розкриті за допомогою існуючих у цей час технологій і теоретичних методів (сильні), і тими, які допускають подібне розкриття (слабкі).

Інший тип систем шифрування передбачає використання різних ключів для шифровки й дешифрування повідомлень — подібні системи прийнята називати *асиметричними*. Прикладом такої системи є система з відкритим ключем, що передбачає використання двох ключів, один із яких є відкритим, а інший зберігається в секреті. Алгоритм шифрування також може бути відкритим, тому будь-який користувач, що бажає направити власникові ключів зашифроване повідомлення, може використати його відкритий ключ і відповідний алгоритм шифрування. Однак дешифрувати дане повідомлення зможе тільки той, хто має парний закритий ключ шифрування. Системи шифрування з відкритим ключем можуть також використовуватись для відправлення разом з повідомленням "цифрового підпису", що підтверджує, що дане повідомлення було дійсно відправлене власником відкритого ключа. Найбільш популярною асиметричною системою шифрування є RSA (це ініціали трьох розроблювачів даного алгоритму).

12.6. RAID (масив незалежних дискових накопичувачів з надмірністю)

Апаратне забезпечення, на якому експлуатується СКБД, повинне бути відмовостійким. Це означає, що СКБД повинна продовжувати

працювати навіть при відмові апаратних компонентів. Для цього необхідно мати надлишкові компоненти, які можуть бути об'єднані в систему, що зберігає свою працездатність при відмові одного або декількох компонентів. До числа основних апаратних компонентів, які повинні бути відмовостійкими, відносяться дискові накопичувачі, дискові контролери, процесори, джерела живлення й вентилятори охолодження. Дискові накопичувачі є найбільш уразливими серед всіх апаратних компонентів і характеризуються найнижчими показниками безперервної роботи між відмовами.

Одним з рішень цієї проблеми є застосування технології RAID. Спочатку ця абревіатура розшифровувалася як Redundant Array of Inexpensive Disks (Масив недорогих дискових накопичувачів з надмірністю), але надалі букву "I" у цій абревіатурі стали розглядати як скорочення від "independent" (незалежний). RAID-масив являє собою масив дискових накопичувачів великого обсягу, що складає з декількох незалежних дисків, спільне функціонування яких організовано таким чином, що при цьому підвищується надійність і разом з тим збільшується продуктивність.

Продуктивність збільшується завдяки смуговому розподілу даних. Дані на дисках розподіляються по сегментах, що представляють собою розділи дисків рівного розміру (цей розмір називається *одиницею смугового розподілу*), які розподіляються по декількох дисках і забезпечують прозорий доступ. У результаті такий масив стає аналогічним одному великому швидкодіючому диску, але фактично дані в ньому розподілені по декількох дисках меншого обсягу. Смуговий розподіл забезпечує підвищення продуктивності вводу-виводу, оскільки дозволяє одночасно виконувати кілька операцій вводу-виводу (на різних дисках). Поряд із цим смуговий розподіл даних дозволяє рівномірно розподіляти навантаження між дисками.

Підвищена надійність RAID-масиву забезпечується завдяки дублюванню даних (такі дублікати називаються *дзеркальними копіями*) і зберіганню на дисках надлишкової інформації, сформованої з використанням схем контролю парності або схем виправлення помилок, таких як код Ріда-Соломона (Reed-Solomon). У схемі контролю парності кожний байт повинен мати пов'язаний з ним біт парності, що приймає значення 0 або 1, залежно від того, чи є парним або непарним кількість бітів 1 у байті, якому відповідає цей біт контролю парності. Якщо в контрольованому байті деякі біти будуть перекручені, значення біта парності не збіжиться зі значенням, що відповідає новому складу битов 1 у цьому байті. Аналогічним образом, при перекручуванні збереженого біта контролю парності він не буде відповідати даним у байті, що дозволить виявити помилку. З іншого боку, схеми коректування помилок

передбачають зберігання двох або більше додаткових битов і дозволяють відновлювати первісні дані, якщо один з бітів буде перекручений. Схеми контролю парності й коректування помилок можуть застосовуватися при смуговому розподілі даних по дисках.

13 ОДНОЧАСНА РОБОТА Й КЕРУВАННЯ ТРАНЗАКЦІЯМИ

У розділі 2 описані загальні функціональні можливості, які повинна надавати типова СКБД. У їхнє число входять три тісно зв'язані між собою функції, призначення яких складається в гарантованій підтримці бази даних у достовірному й погодженому стані, а саме: служби підтримки транзакцій, керування паралельним доступом і засоби відновлення баз даних. Характеристики надійності, вірогідності й погодженості даних повинні зберігатися при відмовах устаткування або програмних компонентів, а також при експлуатації бази даних у середовищу багатьох користувачів. Ці три функції докладно розглядаються в даній главі.

Незважаючи на те що кожна із цих функцій обговорюється окремо, всі вони перебувають у взаємній залежності. І механізм керування паралельним доступом, і засоби відновлення призначені для захисту баз даних від втрати даних або їхнього переходу в неузгоджений стан. Багато СКБД допускають одночасне виконання декількома користувачами різних операцій у базі даних. Якщо ці операції здійснюються безконтрольно, то дії, що виконуються користувачами, можуть довільним образом впливати одна на одну, внаслідок чого база даних може перейти в неузгоджений стан. Для виключення подібних явищ у кожній СКБД реалізується протокол *керування паралельним доступом*, у завдання якого входить запобігання небажаного впливу процесів користувачів один на одного.

Відновлення бази даних являє собою процедуру перекводу бази даних у деякий припустимий стан, яка виконується після відмови. *Відмова бази даних* може виникнути в результаті аварійного останову системи, збоїв устаткування, виявлення програмних помилок або несправності носіїв інформації. Причини можуть бути різними: руйнування магнітної головки, помилки в прикладній програмі й у логіці додатку, що працює з базою даних, і т.д. Крім того, причиною може послужити ненавмисне або навмисне ушкодження або знищення даних і програмного забезпечення операторами системи або її кінцевих користувачів. Будь-яка СКБД повинна мати засобу відновлення системи (незалежно від причини відмови), а також забезпечувати повернення бази даних у погоджений стан.

13.1 Підтримка транзакцій

Керування одночасною роботою обговорюється окремо, хоча і є складовою частиною підтримки цілісності в СКБД. У зв'язку з керуванням одночасною роботою СКБД розглядаються поняття *транзакція* й *розклад.* Системна транзакція на відміну від логічної транзакції є елементарною частиною процесу, що включає селекцію (вибірку), модифікацію або

перепис у базі даних. Логічна транзакція прикладної програми може бути розбита на кілька системних транзакцій. Ми розглянемо логічну транзакцію, що надалі називається просто транзакцією.

Транзакція. Дія або ряд дій, виконуваних одним користувачем або прикладною програмою, які здійснюють читання або зміну вмісту бази даних.

Транзакція є логічною одиницею роботи, яка виконується в базі даних. Вона може бути представлена окремою програмою, частиною програми або навіть окремою командою (наприклад, командою INSERT або UPDATE мови SQL) і включати довільну кількість операцій, що виконуються у базі даних. З погляду адміністратора бази даних експлуатація будь-якого додатка може розцінюватися як ряд транзакцій, у проміжках між якими виконується обробка даних, здійснювана поза середовищем бази даних. Для ілюстрації поняття транзакції розглянемо два відношення з навчального проекту *DreamHome*, описаного в розділі 2:

```
Staff(staffNo, fName, lName, position, sex, DOB,
      salary,
      branchNo)
PropertyForRent (propertyNo, street, city, postcode,
           type, rooms, rent, ownerNo, staffNo,
           branchNo)
```

Найпростішою транзакцією, яка виконується в подібній базі даних, може бути коректування зарплати певного працівника, зазначеного його табельним номером `x`. Узагальнено подібна транзакція може бути записана, як показано в табл. 13.1 (варіант А). У цій главі ми будемо позначати операції читання або запису елемента даних `x` за допомогою вираїв `read(x)` і `write(x)`. При необхідності до імені елемента даних можуть додаватися додаткові позначники. Наприклад, у стовпці Варіант А (табл. 13.1) використовується позначення `read(staffNo=x,salary)`, що вказує, що потрібно зчитати елемент даних `salary` для запису, у якій ключове значення дорівнює `x`. У даному прикладі транзакція складається із двох операцій, що виконуються у базі даних `(read i write)`, і однієї операції, що виконується поза базою даних `(salary = salary*1.1)`.

Більш складна транзакція, текст якої також показаний у табл. 13.1 (варіант Б), призначена для видалення відомостей про працівника, заданому його табельним номером `x`. У цьому випадку, крім видалення відповідного рядка з відношення `Staff`, потрібно знайти всі рядки відношення `PropertyForRent`, що описують об'єкти нерухомості, за які відповідав даний працівник, після чого призначити їх деякому іншому працівникові, табельний номер якого, припустимо, має значення `newStaffNo`. Якщо всі зазначені зміни не будуть внесені до кінця,

правила посилальної цілісності будуть порушена й база даних виявиться в неузгодженому стані — за об'єкт нерухомості буде відповідати неіснуючий співробітник компанії.

Таблиця 13.1. Приклад виконання транзакції

Варіант А	Варіант Б

	`delete(staffNo = x)`
	`for all PropertyForRent records, pno`
	`begin`

`read(staffNo = x, salary)`
`salary = salary * 1.1`
`write(staffNo = x,`
`new_salary)`
	`read(propertyNo = pno, staffNo)`
	`if (staffNo = x) then`
	`begin`
	`staffNo = newStaffNo`
	`write(propertyNo = pno, staffNo)`
	`end`

	`end`

Будь-яка транзакція завжди повинна переводити базу даних з одного погодженого стану в інший, хоча допускається, що погодженість стану бази може порушуватися в ході виконання транзакції. Наприклад, у процесі виконання транзакції, представленої в стовпці *Варіант Б* табл. 13.1, виникає ситуація, коли один з рядків відношення `PropertyForRent` містить нове значення атрибута `staffNo` – `newStaffNo`, тоді як інші рядки усе ще містять колишнє значення `x`. Однак після завершення виконання транзакції в усі необхідні рядки повинне містити нове значення – `newstaffNo`.

Будь-яка транзакція завершується одним із двох можливих способів. У випадку успішного завершення результати транзакції *фіксуються* (commit) у базі даних, і остання переходить у новий погоджений стан. Якщо виконання транзакції не увінчалося успіхом, вона *відміняється.* У цьому випадку в базі даних повинне бути відновлене той погоджений стан, у якому вона перебувала до початку даної транзакції. Цей процес називається *відкотом* (roll back), або *скасуванням транзакції.* Зафіксована транзакція не може бути скасована. Якщо виявиться, що зафіксована транзакція була помилкової, буде потрібно виконати іншу транзакцію, що скасовує дії, виконані першою транзакцією (розділ 13.4.2). Така транзакція називається транзакцією, *що компенсує.* Але транзакція, що завершилася аварійно, для якої виконаний відкіт, може бути викликана на виконання пізніше й, залежно від причин попередньої відмови, цілком успішно завершена й зафіксована в базі даних.

У жодній СКБД не може бути передбачений апріорний спосіб визначення того, які саме операції відновлення можуть бути згруповані для формування єдиної логічної транзакції. Тому повинен застосовуватися метод, що дозволяє вказувати границі кожної із транзакцій ззовні, з боку

користувача. У більшості мов маніпулювання даними для вказівки границь окремих транзакцій використовуються оператори BEGIN TRANSACTION, COMMIT і ROLLBACK (або їхні еквіваленти). Якщо ці обмежники не були використані, як єдина транзакція звичайно розглядається вся виконувана програма. СКБД автоматично виконає команду COMMIT при нормальному завершенні цієї програми. Аналогічно, у випадку аварійного завершення програми в базі даних автоматично буде виконана команда ROLLBACK.

Порядок виконання операцій транзакції прийнято позначати за допомогою діаграми переходів. Приклад такої діаграми наведений на рис. 13.1.

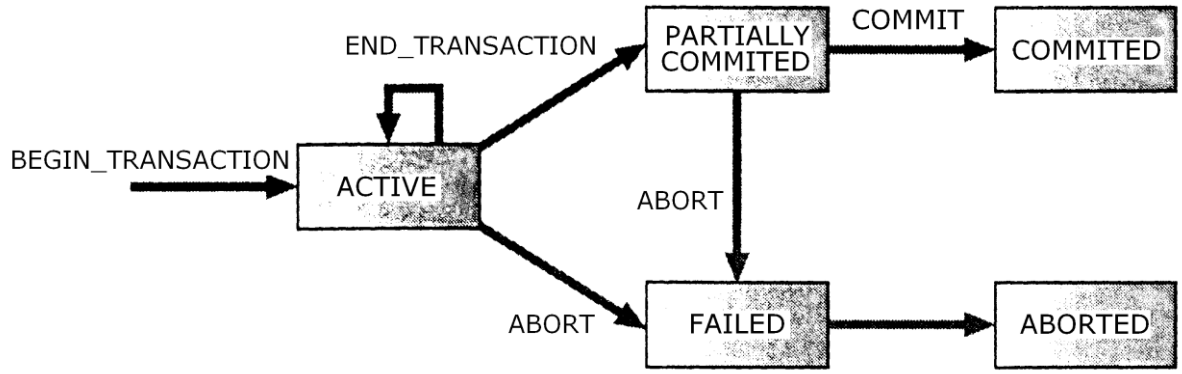


Рис. 13.1. Діаграма переходу для деякої транзакції

Слід зазначити, що на цій діаграмі, крім очевидних станів ACTIVE, COMMITTED і ABORTED, є ще два стани, описані нижче.

- PARTIALLY COMMITTED. Цей стан виникає після виконання останнього оператора. У цей момент може бути виявлено, що в результаті виконання транзакції порушені правила впорядкування або обмеження цілісності, тому транзакцію необхідно завершити аварійно. Ще один варіант розвитку подій полягає в тому, що в системі відбувається відмова й всі дані, обновлені в транзакції, неможливо успішно записати в зовнішню пам'ять. У подібних випадках транзакція повинна перейти в стан FAILED і завершитися аварійно. А якщо транзакція виконана успішно, те всі результати відновлення можуть бути надійно записані в зовнішній пам'яті й транзакція може перейти в стан COMMITTED.
- FAILED. Такий стан виникає, якщо транзакція не може бути зафіксована або відбулося її аварійне завершення, коли вона перебувала в стані ACTIVE. Це аварійне завершення могло виникнути через скасування транзакції користувачем або в результаті дії протоколу керування паралельним доступом, що

викликав аварійне завершення транзакції для забезпечення впорядкованності операцій бази даних.

13.1.1 Властивості транзакцій

Існують певні властивості, якими повинна володіти кожна із транзакцій. Нижче представлені чотири основних властивості транзакцій, які прийнято позначати абревіатурою ACID (Atomicity, Consistency, Isolation, Durability - нерозривність, узгодженість, ізольованість, стійкість), що складається з перших букв назв цих властивостей.

- **Нерозривність**. Це властивість, для опису якого застосовне вираження "все або нічого". Будь-яка транзакція являє собою неподільну одиницю роботи, що може бути або виконана вся цілком, або не виконана взагалі. За забезпечення нерозривності відповідає підсистема відновлення СКБД.
- **Узгодженість**. Кожна транзакція повинна переводити базу даних з одного узгодженого стану в інший. Відповідальність за забезпечення узгодженості покладає й на СКБД, і на розроблювачів додатків. У СКБД узгодженість може забезпечуватися шляхом виконання всіх обмежень, заданих у схемі бази даних, таких як обмеження цілісності й обмеження предметної області. Але подібна умова є недостатнім для забезпечення узгодженості. Наприклад, припустимо, що деяка транзакція призначена для переводу коштів з одного банківського рахунку на інший, але програміст припустився помилки в логіці транзакції й передбачив зняття грошей із правильного рахунку, а зарахування – на неправильний рахунок. У цьому випадку база даних переходить у неузгоджений стан, незважаючи на наявність правильно заданих обмежень. Проте відповідальність за усунення такої неузгодженості не може бути покладене на СКБД, оскільки в ній відсутні які-небудь засоби виявлення подібних логічних помилок.
- **Ізольованість**. Всі транзакції виконуються незалежно одна від одній. Іншими словами, проміжні результати незавершеної транзакції не повинні бути доступні для інших транзакцій. За забезпечення ізольованості відповідає підсистема керування паралельним виконанням.
- **Стійкість**. Результати успішно завершеної (зафіксованої) транзакції повинні зберігатися в базі даних постійно й не повинні бути загублені в результаті наступних збоїв. За забезпечення стійкості відповідає підсистема відновлення.

13.1.2 Архітектура бази даних

Архітектура типової СКБД описана в главі 2. На рис. 13.2 представлений фрагмент повної структурної схеми СКБД (див. рис. 2.7), що включає чотири високорівневі модулі бази даних, які відповідають за обробку транзакцій, керування паралельним доступом і відновлення системи. *Диспетчер транзакцій* координує роботу транзакцій по запитах від прикладних програм. Він взаємодіє із *планувальником,* відповідальним за реалізацію обраної стратегії керування паралельним доступом. У деяких випадках планувальник називають *диспетчером блокувань,* якщо використовуваний протокол керування паралельним виконанням будується на основі системи блокувань. Ціль роботи планувальника складається в досягненні максимально можливого рівня розпаралелювання роботи, за умови виключення впливу транзакцій, що виконуються паралельно, одна на одну, оскільки це може послужити джерелом порушення цілісності або узгодженості бази даних.

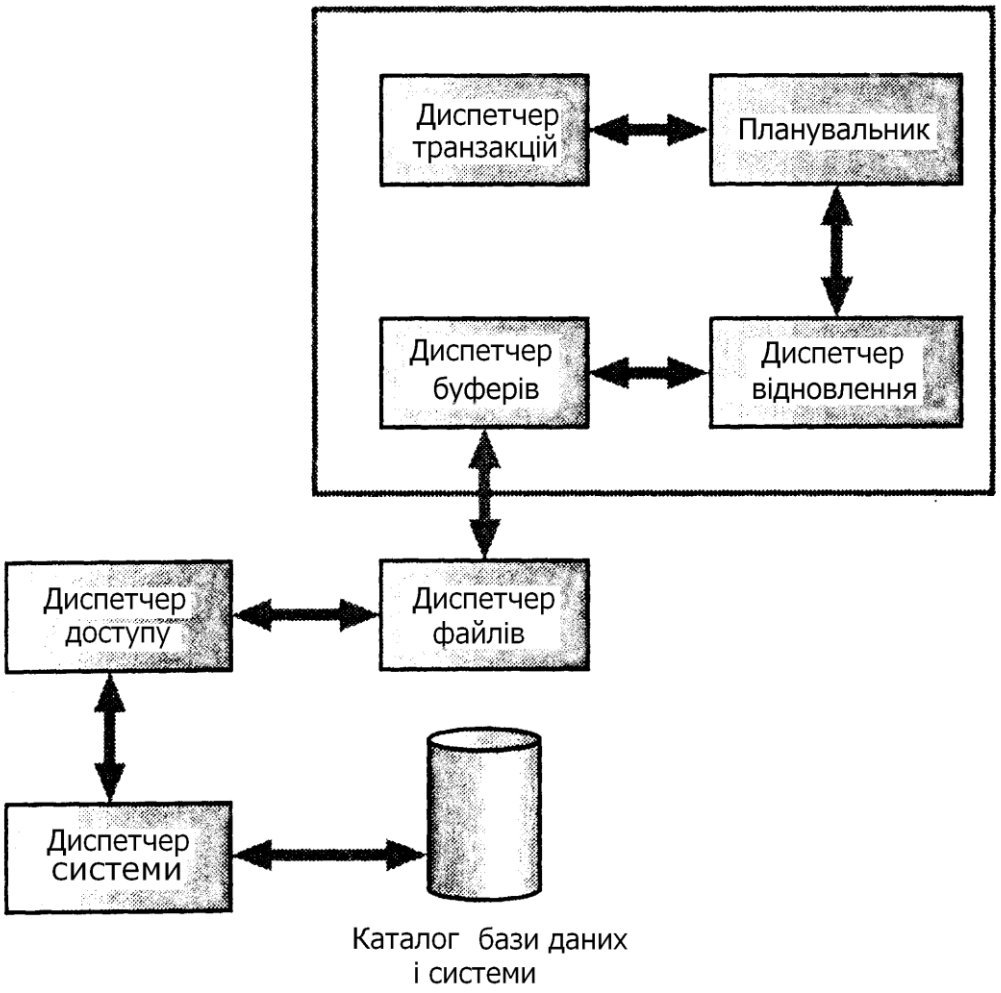


Рис. 13.2. Підсистема обробки транзакцій типової СКБД

Якщо в процесі виконання транзакції відбувається відмова, то база даних може виявитися в неузгодженому стані. Завданням *диспетчера відновлення* є забезпечення того, що в такому разі база даних буде автоматично повернута в той стан, у якому вона перебувала до початку даної транзакції, і, отже, залишиться узгодженої. Нарешті, *диспетчер буферів* відповідає за передачу даних між оперативною пам'яттю комп'ютера й зовнішньою дисковою пам'яттю.

13.2 Керування паралельним доступом

У цьому розділі розглядаються проблеми, пов'язані з організацією паралельного доступу до даних, а також описані способи, що дозволяють вирішити пов'язані із цим проблеми. Спочатку наведене робоче визначення функції керування паралельним доступом.

Керування паралельним доступом. Процес організації одночасного виконання в базі даних різних операцій доступу, що гарантує запобігання їхнього впливу один на одного.

13.2.1 Необхідність керування паралельним доступом

Найважливішою метою створення баз даних є організація паралельного доступу багатьох користувачів до загальних даних, що використовуються ними спільно. Забезпечити паралельний доступ відносно нескладно, якщо всі користувачі будуть тільки читати дані, збережені в базі. У цьому випадку робота кожного з них не робить впливу на роботу інших користувачів. Але якщо два або декілька користувачі одночасно звертаються до бази даних і хоча б один з них бажає обновити збережену в базі інформацію, можливий взаємний вплив процесів доступу, здатний привести до неузгодженості даних.

Дане завдання подібне до завдань, що постають перед будь-який многопользовательской комп'ютерною системою, коли кілька програм (або транзакцій) одержують можливість одночасно виконувати операції завдяки використанню мультипрограмної організації роботи, що дозволяє двом або декільком програмам (або транзакціям) виконуватися в той самий час. Наприклад, багато систем включають підсистему введення-виводу, здатну виконувати операції введення-виводу, у той час як процесор здійснює інші операції. Подібні системи дозволяють двом або декільком транзакціям виконуватися одночасно. Система починає виконання першої транзакції й продовжує її виконання до першої операції введення-виводу. На час виконання цієї операції система припиняє виконання першої транзакції й переходить до виконання команд другої транзакції. Коли другої транзакції знадобиться виконати операцію введення-виводу, керування буде повернуто першої транзакції і її виконання буде продовжено з тієї точки, у якій вона була припинена.

Виконання першої транзакції буде продовжено до досягнення наступної операції введення-виводу. Таким чином, виконання операцій двох транзакцій чергується й забезпечується їхнє паралельне виконання. Крім того, загальна продуктивність системи (обсяг роботи, виконуваної протягом заданого тимчасового інтервалу) підвищується, оскільки процесор виконує команди іншої транзакції, замість того щоб даремно простоювати, очікуючи завершення запущеної операції введення-виводу.

Незважаючи на те що кожна із транзакцій може сама по собі виконуватися цілком коректно, подібне чергування операцій здатне приводити до невірних результатів, через що цілісність і узгодженість бази даних будуть порушені. Ми розглянемо три приклади потенційних проблем, які можуть мати місце при паралельному виконанні транзакцій: *проблему загубленого відновлення, проблему залежності від незафіксованих результатів* і *проблему аналізу неузгодженості.* Для ілюстрації зазначених проблем ми скористаємося відношенням з даними про банківські рахунки персоналу компанії *DreamHome.* У цьому контексті будуть розглядатися транзакції, які ми будемо вважати *об'єктами керування паралельним виконанням*.

Приклад 13.1. Проблема загубленого відновлення

Результати цілком успішно завершеної операції відновлення однієї транзакції можуть бути перекриті результатами виконання іншої транзакції. Це порушення відомо як *проблема загубленого відновлення*. Суть її проілюстрована прикладом, наведеним у табл. 13.2. У цьому прикладі транзакція T_1 виконується паралельно із транзакцією T_2. Транзакція T_1 полягає в знятті 10 фунтів стерлінгів з рахунку bal_x, на якому спочатку перебуває 100 фунтів стерлінгів, а транзакція T_2 припускає зарахування 100 фунтів стерлінгів на цей же рахунок. Якщо ці транзакції будуть виконуватися послідовно, одна за іншою, без чергування їхніх операцій, то після завершення роботи на рахунку буде перебувати сума в 190 фунтів стерлінгів, незалежно від порядку виконання транзакцій. Але допустимо, що транзакції T_1 і T_2 починаються практично одночасно й кожна з них зчитує вихідне значення суми на рахунку, рівним 100 фунтам стерлінгів. Потім транзакція T_2 збільшує суму на 100 фунтів стерлінгів і записує результат, рівним 200 фунтам стерлінгів, у базу даних – на рахунок bal_x. Тим часом транзакція T_1 зменшує значення своєї копії початкової суми на рахунку bal_x і записує отримане значення, рівне 90 фунтам стерлінгів, у базу даних на той же рахунок, перекриваючи результат попереднього відновлення. У результаті з балансового рахунку "зникає" 100 фунтів стерлінгів, доданих при виконанні попередньої операції. Можна уникнути втрати результатів виконання транзакції T_2, заборонивши транзакції T_1 зчитувати вихідне значення на рахунку bal_x аж до завершення виконання транзакції T_2. Ці

проблеми знімаються при блокуваннях транзакцій за рахунок двофазного протоколу захвата (див. розділ 13.3).

Таблиця 13.2. Приклад проблеми загубленого відновлення

Час	**Транзакція T_1**	**Транзакція T_2**	**Поле bal_x**
t_1		begin_transaction	100
t_2	begin_transaction	read (balx)	100
t_3	read (bal_x)	$bal_x = bal_x + 100$	100
t_4	$bal_x = bal_x - 10$	write (bal_x)	200
t_5	write (bal_x)	commit	90
t_6	commit		90

Приклад 13.2. Проблема залежності від незафіксованих результатів (або "брудного" читання)

Проблема залежності від незафіксованих результатів виникає в тому випадку, якщо одна із транзакцій одержить доступ до проміжних результатів виконання іншої транзакції до того, як вони будуть зафіксовані в базі даних. У табл. 13.3 наведений приклад залежності від незафіксованих результатів, що викликає появу помилки. У цьому прикладі використовуються ті ж початкові дані для залишку на рахунку bal_x, що й у попередньому прикладі. У цьому випадку транзакція T_4 збільшує значення суми на рахунку bal_x до 200 фунтів стерлінгів, після чого виконання транзакції відміняється, тому СКБД повинна виконати відкіт транзакції з відновленням початкового значення на рахунку, рівним 100 фунтам стерлінгів. Однак до цього моменту транзакція T_3 уже встигла зчитати змінене значення рахунку bal_x (200 фунтів) і використала саме це значення при виконанні операції зняття 10 фунтів стерлінгів з рахунку, після чого зафіксувала в базі даних невірний результат, рівним 190 фунтам стерлінгів (замість правильного — 90 фунтів стерлінгів). Значення bal_x, зчитане в транзакції T_3, називається брудними даними. Від цього терміна походить друга назва розглянутої проблеми – проблема брудного читання.

Таблиця 13.3. Приклад проблеми залежності від незафіксованих результатів

Час	Транзакція T_3	Транзакція T_4	Поле bal_x
t_1		begin_transaction	100
t_2		read (bal_x)	100
t_3		$bal_x = bal_x + 100$	100
t_4	begin_transact ion	write (bal_x)	200
t_5	read (bal_x)	...	200
t_6	$bal_x = bal_x$-10	rollback	100
t_7	write (bal_x)		190
t_8	commit		190

Причина відкоту транзакції не має значення; допустимо, що при її виконанні була виявлена деяка помилка (наприклад, зарахування на неправильний рахунок). Джерело помилки полягає в тім, що при виконанні транзакції T_3 приймається припущення про успішне завершення операції відновлення в транзакції T_4, хоча в дійсності відбувся відкіт цієї транзакції. Проблему можна усунути, заборонивши транзакції T_3 зчитувати значення залишку bal_x до ухвалення рішення про те, чи повинна бути виконана фіксація або відкіт транзакції T_4.

В обох наведені вище прикладах мова йшла про транзакції, що виконують відновлення даних у базі, чергування операцій яких могло привести до руйнування бази. Однак транзакції, які тільки зчитують інформацію з бази даних, також можуть давати невірні результати, якщо їм будуть доступні для читання проміжні результати транзакцій, що одночасно виконуються й ще не завершені, які обновляють інформацію в базі даних. Така ситуація розглядається в наступному прикладі.

Приклад 13.3. Проблема аналізу непогодженості

Проблема аналізу неузгодженості виникає в тих випадках, коли транзакція зчитує кілька значень із бази даних, після чого друга транзакція обновляє деякі із цих значень безпосередньо під час виконання першої транзакції. Наприклад, транзакція, що підсумовує дані, обрані з бази (скажемо, що обчислює загальну суму на рахунках), одержить невірне значення, якщо під час її виконання інша транзакція змінить значення, які вона зчитала. Приклад подібної помилки наведений у табл. 13.4. Тут транзакція T_6, що обчислює підсумкове значення, виконується паралельно із транзакцією T_5. Транзакція T_6 обчислює суму залишків на рахунках x (100 фунтів стерлінгів), y (50 фунтів стерлінгів) і z (25 фунтів

стерлінгів). Однак у цей же час транзакція T_5 здійснює перевод десяти фунтів стерлінгів з рахунку bal_x на рахунок bal_z. У результаті обчислене транзакцією T_6 значення виявляється невірним (більше на 10 фунтів стерлінгів). Цю проблему можна усунути, заборонивши транзакції T_6 зчитувати значення на рахунках bal_x і bal_z доти, поки транзакція T_5 не зафіксує виконані нею відновлення.

Таблиця 19.4. Приклад проблеми усунення непогодженості

Час	Транзакція T_5	Транзакція T_6	Поле bal_x	Поле bal_y	Поле bal_z	Поле sum
t_1		begin_transaction	100	50	25	
t_2	begin_transact ion	sum = 0	100	50	25	0
t_3	read (bal_x)	read (bal_x)	100	50	25	0
t_4	$bal_x = bal_x$-10	sum = sum + bal_x	100	50	25	100
t_5	write (bal_x)	read(bal_y)	90	50	25	100
t_6	read (bal_z)	sum = sum + bal_y	90	50	25	150
t_7	$bal_z = bal_z$ + 10		90	50	25	150
t_8	write (bal_z)		90	50	35	150
t_9	commit	read(bal_z)	90	50	35	150
t_{10}		sum = sum + bal_z	90	50	35	185
t_{11}		commit	90	50	35	185

Ще одна проблема може виникнути, якщо в деякій транзакції `T` відбувається повторне читання раніше зчитаного елемента даних, але між цими операціями читання була виконана модифікація цього елемента даних в іншій транзакції. Таким чином, у транзакції `T` будуть отримані два різних значення того самого елемента даних. Таку ситуацію іноді характеризують як проблему *неповторювального* (або *нечіткого*) *читання*. Аналогічна проблема може відбутися, якщо транзакція `T` виконує запит, у якому відбувається вибірка з відношення ряду рядків, що задовольняють деякому предикату, а при повторному виконанні цього запиту в більш пізній момент часу виявляється, що отримана множина рядків містить додаткові (фантомні) рядки, які були вставлені іншою транзакцією в період між двома операціями читання. Таку проблему іноді називають *фантомним читанням*.

13.3 Розклади при паралельній роботі

Розклад або *графік* при одночасній роботі представляє порядок виконання транзакцій у часі. Можна дослідити ряд розкладів, наприклад представлених на рис. 13.3, позначаючи зчитування й запис поля A відповідно R-A і W-A. (Надалі W буде інтерпретуватися як модифікація вхідними змінними).

У першому розкладі (рис. 13.3) не може виникнути неузгодженості бази даних, тому що транзакції T_1 і T_2 виконуються послідовно, одна за іншою.

Розклад 1	Розклад 2	Розклад 3	Розклад 4	Розклад 5
T_1:R-A	T_1:R-A	T_1:W-A	T_1:W-A	T_2:R-A
T_1:W-A	T_2:R-A	T_2:W-A	T_2:R-A	T_1:W-A
T_2:R-A	T_1:W-A	T_1:Скасувати	T_1:Перервати	T_2:R-A
T_2:W-A	T_2:W-A			

Рис.13.3. Можливі розклади.

У другому розкладі модифікація транзакції T_1 затирається транзакцією T_2, отже, модифікація транзакції T_1 губиться. У розкладі 3 також відбувається втрата модифікації транзакції T_2, викликана тим, що після запису T_2 треба скасування транзакції T_1. Ситуація розкладу 4 відома за назвою *неправильне зчитування,* оскільки обране транзакцією T_2 значення згодом видаляється з бази даних. Дві вибірки в розкладі 5 дадуть різні значення, тому що транзакція T_2 переривалася записом. Для обмеження порядку виконання транзакцій необхідні протоколи, складність яких залежить від того, на якому рівні потрібно підтримувати цілісність при роботі з базою даних.

Узгоджені стани бази даних забезпечують послідовні розклади або розклади, що приводяться до послідовного. Розклад називається *послідовним,* якщо всі дії транзакції виконуються один за одним. Розклад називається тим, *що приводиться до послідовного,* якщо результат застосування цього розкладу до бази даних еквівалентний результату послідовного розкладу. Також можливі *циклічні* розклади, які повинні бути заборонені при одночасній роботі з базою даних, тому що вони можуть викликати неузгодженість бази даних..

Для виявлення розкладів, що приводяться до послідовного й завдяки цьому що зберігають узгоджений стан бази даних, можна використати метод, заснований на побудові графа залежностей. У цьому графі транзакціям відповідають вузли, а спрямована дуга графа, що зв'язує вузли T_i і T_j, позначає, що вхід T_j залежить від виходу T_i, тобто T_j використовує значення поля, вироблене транзакцією. T_i Якщо граф залежностей містить цикли, відповідний розклад не приводиться до послідовного й, отже, дає суперечливі результати. На рис. 13.4 показані три розклади з відповідними графами залежностей. Розклад S_1 – послідовне (несуперечливе), S_2 – приводиться до послідовного (несуперечливе), а S_3 – циклічне (суперечливе).

Монопольне використання значення поля в процесі роботи СКБД можна забезпечити за допомогою блокування (захвата). У цьому випадку ніяка інша транзакція не одержить доступу до поля й не зможе його використати (для читання або запису) доти, поки поле не буде звільнено. Було показано, що узгодженість бази даних можна аналізувати, досліджуючи графів передування, побудовані для послідовностей захватів

і звільненні. Передбачається, що будь-яка транзакція може бути представлена у вигляді послідовності ЗАХВАТ-ДІЯ-ЗВІЛЬНЕННЯ. Для наочності будуть, однак, використовуватись тільки пари ЗАХВАТ-ЗВІЛЬНЕННЯ. Дуга графа передування, що з'єднує вузли T_i і T_j, означає, що T_j захоплює поле після його звільнення транзакцією T_i, оскільки в послідовному розкладі для виконання транзакції T_j, що вимагає захвата поля транзакція T_i, що передує, повинна звільнити це поле. Для перевірки розкладу на зведення до послідовного необхідно побудувати граф передування. Приклад представлений на рис. 13.4.

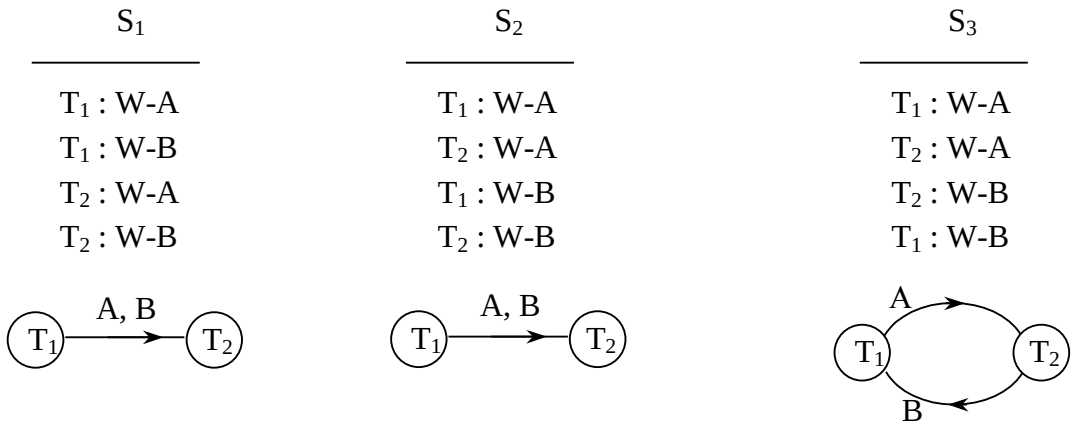


Рис. 13.4 Три розклади й графи їхніх залежностей

При побудові графа в розкладі вибираються вузли, що відповідають ЗВІЛЬНЕННЯМ, які попарно зв'язуються з наступними вузлами ЗАХВАТІВ (вони розташовані не обов'язково безпосередньо один за одним, що видно на прикладі пари ЗВІЛЬНИТИ С и ЗАХОПИТИ С на рис. 13.5). Граф передування так само, як граф залежностей, не повинен містити циклів.

Двофазний протокол *захвата* гарантує приведення до послідовного розкладу й, отже, збереження цілісності бази даних. Двофазний протокол містить просте правило, що полягає в тому, що ніякий захват не може йти відразу за звільненням. Під час першої, висхідної, фази всі транзакції здійснюють захвати (якщо це можливо, тому що протокол може бути тупиковим). Тупикова ситуація виникає при захвату у випадку двох взаємно зв'язаних транзакцій, що нескінченно очікують один одного. Наприклад, розклад захватів (T_1 : Захват А, T_2 : Захват В, T_1 : Захват В, T_2 : Захват А) приведе до нескінченного очікування (тупикові) транзакцій T_1 і T_2. За першою фазою треба друга, спадна, фаза, під час якої в міру завершення транзакцій виконуються всі звільнення.

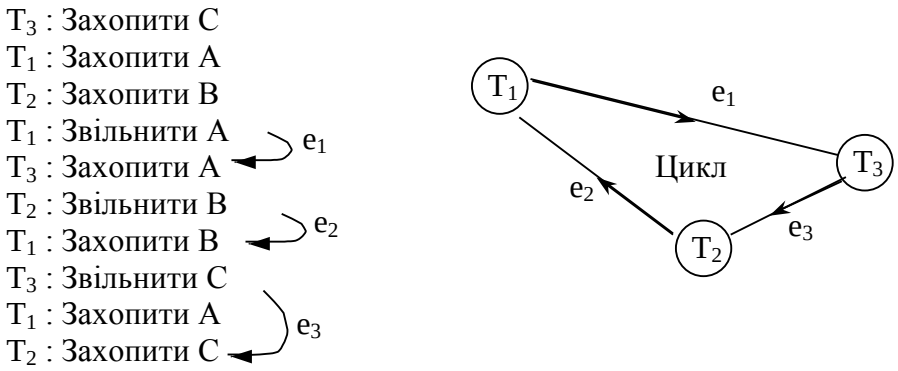


Рис.13.5 Розклад і відповідний граф передування

Дотепер ми розглядали зв'язані модифікації транзакцій T_1 і T_2, тобто такі ситуації, у яких модифікації транзакції T_2 залежать від значення поля, що модифікувалося транзакцією T_1. Практично не всі запити (транзакції) пов'язані з відновленнями бази даних; вони можуть вимагати тільки вибірку даних. У цих випадках можна розрізняти захвати на читання й запис (останній включає також і зчитування інформації), причому *захвати на читання* можуть бути сумісними на відміну від схеми, що обговорювалася вище, монопольного захвата. Для розглянутого протоколу можна сформулювати наступні вимоги:

а) Якщо транзакція здійснила захвата поля на читання, то припустимі захвати на читання цього поля іншими транзакціями; ніяка транзакція, однак, не може захопити поле на запис, поки існує хоча б один захват на читання.

б) Якщо транзакція здійснила захвата поля на запис, те ніяка інша транзакція не може захопити це ж поле на читання або запис.

При побудові графа передування для розкладу, що містить захвати на читання й запис, не можна не брати до уваги послідовність цих захватів, тобто порядок залежностей між захватами на читання й запис повинен бути таким самим, як якби всі захвати були тільки на запис. У розкладі (T_1: ЗВІЛЬНЕННЯ А, T_2.: ЗАХВАТ-ЧИТАННЯ А, T_3: ЗАХВАТ-ЧИТАННЯ А, T_4: ЗАХВАТ-ЗАПИС А), де ЗАХВАТ-ЧИТАННЯ й ЗАХВАТ-ЗАПИС означають відповідно захвати па читання й запис, повинне виконуватися наступне:

транзакція T_2 повинна бути з'єднана дугами із транзакціями T_2, T_3 і T_4. Розглянутий протокол допускає одночасний захват на читання транзакцій T_2 і T_3, але T_4 не може початися, перш ніж і T_2 і T_3 не звільнять А; отже, розклад повинен бути таким:

(T_1: ЗВІЛЬНЕННЯ А, T_2: ЗАХВАТ-ЧИТАННЯ А, T_3: ЗАХВАТ-ЧИТАННЯ А, ... T_2: ЗВІЛЬНЕННЯ А, T_3: ЗВІЛЬНЕННЯ А, T_4: ЗАХВАТ-ЗАПИС А).

Відносний порядок T_2 і T_3 при звільненні несуттєвий, якщо припустити, що до другої появи T_2 у розкладі не відбувається захвата A. Як і колись, для приведення до послідовного розкладу граф передування не повинен містити циклів.

Тут також застосуємо двофазний протокол, тому всі захвати на запис і читання повинні передувати всім звільненням. Такий розклад гарантує цілісність бази даних, оскільки воно забезпечує зведення до послідовного розкладу.

Для подальшого розширення можливостей розкладів можна використати ідею про те, що кожний захват на запис має на увазі зчитування полів і при модифікації зчитаних полів записується нове значення. Можна припустити, що транзакції зчитують деякі поля й записують їх назад і не кожне зчитане поле модифікується, хоча воно й може використовуватись при обчисленні нових значень інших полів. У графах предшествования таких розкладі можуть використовуватись транзакції, які називаються *марними,* тобто не мають вихідних дуг. Подібні марні вузли можна видалити із графа. Однак при припущенні незалежності записів має місце наступне: нехай дан розклад, у якому спочатку T_1 записує поле, а потім T_2 зчитує його (так що можна провести дугу з T_1 в T_2). Якщо транзакція T_3 модифікує це ж поле, то вона повинна з'являтися або до T_1, або після T_2, але не між ними. Це приводить до побудови двох можливих дуг у графі, що називається *поліграфом* (тобто графом з декількома дугами між парами вершин). (Відзначимо, що, відповідно до останнього припущення, транзакція може здійснити захвата на запис поля, значення якого не зчитувалося, якщо до захвата на запис не було захвата цього поля на читання.) Після усунення марних вузлів для перевірки на зведення до послідовного розкладу граф варто спростити шляхом видалення однієї з пари дублюючих дуг. Існує значне число можливих варіантів (2^n для n пари дуг), серед яких перебуває результат. Тут також може застосовувати двофазний протокол захвату.

14 РОЗПОДІЛЕНІ БАЗИ ДАНИХ

Розглянемо деякі важливі поняття розподілених баз даних: різні форми прозорості, виконання запитів, паралельні відновлення інформації.

14.1 Централізовані й децентралізовані СКБД

СКБД і централізація обробки інформації дозволили усунути такі недоліки традиційних файлових систем, як незв'язаність, непогодженість і надмірність даних. У міру росту баз даних і особливо при їхньому використанні в територіально розділених організаціях з'являються інші проблеми. Так, для централізованої СКБД, що перебувають у вузлі телекомунікаційної мережі, за допомогою якої різні підрозділи організації одержують доступ до даних, з ростом обсягу інформації й кількості транзакцій виникають наступні труднощі:

- великий потік обмінів даними;
- низька надійність;
- низька загальна продуктивність;
- більші витрати на розробку.

Хоча в централізованій базі даних легше забезпечити безпека, цілісність і несуперечність інформації при відновленнях, перераховані проблеми створюють певні труднощі. Як можливе рішення цих проблем пропонується децентралізація даних. При децентралізації досягається:

- більше високий ступінь одночасності обробки внаслідок розподілу навантаження;
- поліпшене використання даних на місцях при виконанні віддалених (дистанційних) запитів;
- менші витрати;
- простота керування.

Витрати на створення мережі, у вузлах якої перебувають малі ЕОМ, набагато нижче, ніж витрати на створення аналогічної системи з використанням великий ЕОМ. На рис.14.1 наведена логічна схема розподіленої бази даних. Розглянемо її докладніше. Припустимо, що існує система обробки банківських операцій на основі мережі. Клієнт у місті А працює із системою за допомогою локальних запитів (географічний принцип). Якщо клієнт переїжджає в місто Б и хоче в банку цього міста одержати свій внесок, для перевірки його рахунку в банку міста А буде виданий віддалений запит. Крім того, більшість запитів головної установи є віддаленими. На їхній основі будується протокол роботи всієї системи. Між банками деякого району можуть також здійснюватися обміни

інформацією на основі локальних запитів, наприклад кредитні операції з фермерами цього району (функціональний принцип).

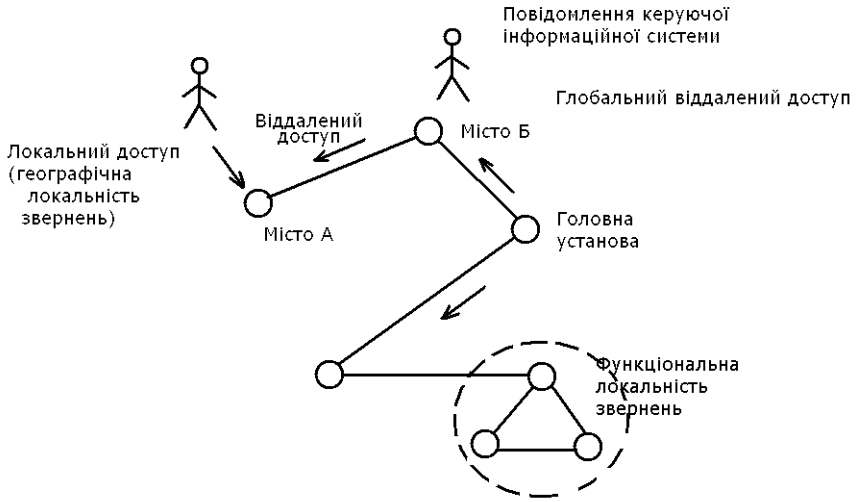


Рис. 14.1 Моделі доступу в розподілених базах даних.

Розподілена база даних. Це набір файлів (відношень), що зберігаються в різних вузлах інформаційної мережі й логічно зв'язаних таким чином, щоб становити єдину сукупність даних (зв'язок може бути функціональної або через копії того самого файлу).

Проблеми розподілених баз даних

Основне завдання розподіленої бази даних – розподіл даних по мережі. Існують наступні способи рішення цього завдання:

а) У кожному вузлі зберігається й використовується власний набір даних, доступний для віддалених запитів. Такий розподіл називається *розділеним.*

б) Деякі дані, що часто використовуються на віддалених вузлах, можуть дублюватися. Такий розподіл називається *частково дубльованим.*

в) Всі дані дублюються в кожному вузлі. Такий розподіл називається *повністю дубльованим.*

г) Деякі файли можуть бути розщеплені горизонтально (виділена підмножина записів) або вертикально (виділена підмножина полів (атрибутів)), при цьому виділені підмножини зберігаються в різних вузлах разом з нерозщепленими даними (п.п. а) – в)). Такий розподіл називається *розщепленим (фрагментованим*).

д) Виділені підмножини можуть дублюватися, як у пп. б) і в).

Розглянемо тепер інші проблеми розподілених баз даних. Проблеми централізованих СКБД існують і тут, однак, децентралізація додає нові:

а) Яка загальна модель даних розподіленої системи? Ми повинні мати єдину концептуальну схему всієї мережі. Це забезпечить *логічну прозорість* даних для користувача, у результаті чого він зможе формувати запит до всієї бази, перебуваючи за окремим терміналом (тобто як би працюючи із централізованою базою даних).

б) Необхідна схема, що визначає місцезнаходження даних у мережі. Це забезпечить *прозорість розміщення* даних, завдяки якій користувач може не вказувати, куди переслати запит, щоб одержати необхідні дані.

в) Розподілені бази даних можуть бути однорідними або неоднорідними в сенсі апаратних і програмних засобів (СКБД). Проблему неоднорідності порівняно легко вирішити, якщо розподілена база є неоднорідною в сенсі апаратних засобів, але однорідної в сенсі програмних засобів (однакові СКБД у вузлах). Якщо ж у вузлах розподіленої системи використовуються різні СКБД, необхідні засоби перетворення структур даних і мов. Це повинна забезпечити *прозорість перетворення* у вузлах розподіленої бази даних.

г) Керування словниками. Для забезпечення всіх видів прозорості в розподіленій базі даних потрібні програми, що управляють численними довідниками або словниками.

д) Методи виконання запитів у розподіленій базі даних відрізняються від аналогічних методів централізованих СКБД, тому що окремі частини запиту потрібно виконувати на місці розташування відповідних даних і передавати часткові результати на інші вузли; при цьому повинна бути забезпечена координація всіх процесів.

е) У розподіленій базі даних потрібний складний механізм керування одночасною обробкою, що, зокрема, повинен забезпечувати синхронізацію при відновленнях інформації, що гарантує несуперечність даних.

ж) Розвинена методологія розподілу й розміщення даних, включаючи розщеплення, є одним з основних вимог до розподіленої бази даних.

Приклад 1. Розглянемо базу даних, розподілену по трьох вузлах, схема якої складається із трьох відношень:

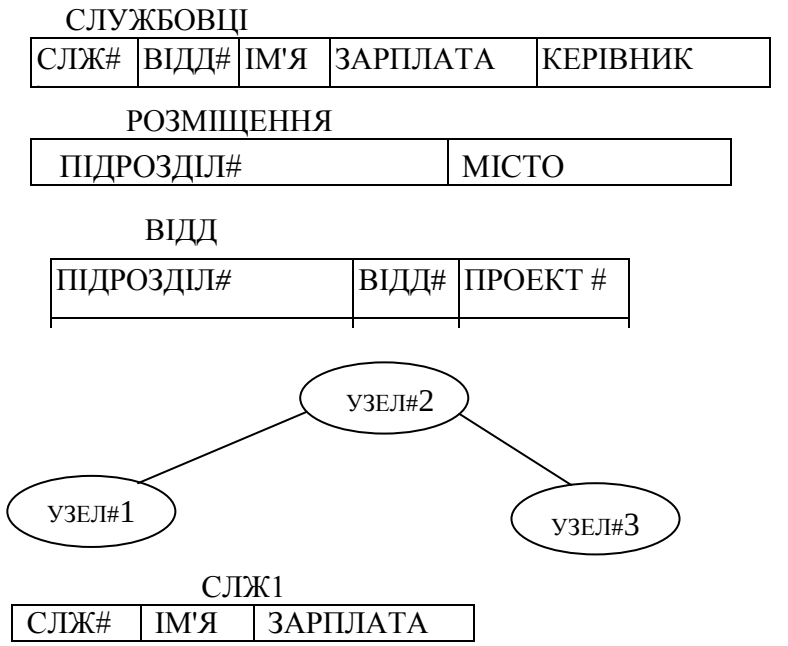
ЩО СЛУЖАТЬ (СЛЖ#, ВІДД#, ЗАРПЛАТА, КЕРІВНИК)
ВІДД (ПІДРОЗДІЛ#, ВІДД#, ПРОЕКТ#)
РОЗМІЩЕННЯ (ПІДРОЗДІЛ #, МІСТО)

На рис. 14.2 зображена схема бази даних, схема географічного розтатування вузлів, та можлива схема розподілу бази даних.

Перша стратегія розподілу. Припустимо, що у вузлі #1 обробляються дані про зарплату й податки. Для цього потрібні атрибути СЛЖ#, ІМ'Я й ЗАРПЛАТА. У вузлі 2 перебуває головна установа й зберігаються повні копії всіх відношень СЛУЖБОВЦІ, ВІДД,

РОЗМІЩЕННЯ. У вузлі #3 зосереджене виробництво й зберігаються повні дані про службовців, що заробляють менше 45 000 дол. у рік; про інших службовців зберігаються тільки відомості про номер відділу й керівника (рис. 14.2).

Відношення СЛЖ1 отримується операцією проекції, СЛЖ3 – операцією селекції, за якою виконується проекція. Для правильного відновлення повних кортежів у кожному відношенні є ключі СЛЖ#. У той час як у вузлі 2 зберігаються повні відношення, у вузлі 1 – вертикальний фрагмент відношення СЛУЖБОВЦІ а у вузлі 3 – горизонтальний і змішаний фрагменти того ж відношення. Змішаний фрагмент – результат двох процесів: горизонтального, а потім вертикального розщеплення відношення СЛУЖБОВЦІ.



Предикат: СЛЖ1← $\Pi_{\text{СЛЖ\#, ІМ'Я, ЗАРПЛАТА}}$ (СЛУЖБОВЦІ)

СЛЖ2

СЛЖ#	ВІДД#	ім'я	зарплата	керівник

Предикат: СЛЖ2← $\sigma_{\text{ЗАРПЛАТА} < 45000}$ (СЛУЖБОВЦІ)

СЛЖ3

СЛЖ#	ВІДД#	КЕРІВНИК

Предикат: СЛЖ3← $\Pi_{\text{СЛЖ\#, ВІДД\#, КЕРІВНИК}}$ (СЛУЖБОВЦІ)

Рис.14.2 Приклад фрагментації й розподілу даних.

Друга стратегія розподілу. Припустимо тепер, що у вузлі 1 перебуває відношення СЛЖ1, що містить відомості тільки про службовців, що працюють у підрозділі 5.

Модифікований предикат схеми вузла 1:

$СЛЖ1 \leftarrow (\Pi_{СЛЖ\#, ІМ'Я, ЗАРПЛАТА}(СЛУЖБОВЦІ)) \bowtie (\sigma_{ПІДРОЗДІЛ\# = 5}(ВІДД))$

Відновлення в обох варіантах розміщення впливає істотно, якщо при відновленні використовуються атрибути, що становлять основу предиката схеми. Для збереження несуперечності схеми в системі потрібно зробити перетворення даних. Це перетворення здійснюється шляхом включення в потрібних місцях нових кортежів (фрагментів) і видалення старих.

14.2 Виконання запитів у розподіленій базі даних

З виконанням запитів у розподіленій базі даних зв'язано кілька проблем. Оскільки дані розподілені, а також можуть бути розщеплені, запити, складені користувачем, що розглядає дані цілком (як якби база даних була централізованою), повинні бути наведені до виду, що враховує розподіл і розщеплення даних. Розподіл і розщеплення вимагають декомпозиції запиту, тому що програма запиту не може бути виконана повністю в одному вузлі, якщо там не зберігається вся необхідна інформація. Крім того, оскільки для завершення виконання програми необхідно переміщати цілі відношення (файли) або результати проміжних обчислень, для зниження витрат на передачу даних необхідно оптимізувати:

а) вирази запитів (програму);

б) методику виконання запиту з урахуванням необхідних переміщень даних.

14.3 Оптимізація запитів

Оптимізація запитів ґрунтується на перестановці операцій у межах запиту, на ідентифікації загальних підвиразів і однократному їхньому виконанні, на трансляції й оптимізації запиту щодо фрагментів.

При оптимізації запиту починають із побудови дерева розбору для виразу запиту. У цьому дереві листи відповідають відношенням, а внутрішні вузли (включаючи корінь) – операціям реляційної алгебри. Розглянемо наступний запит: *як звуть службовців, що працюють над проектом* #5, і припустимо, що відповідний вираз реляційної алгебри має вигляд:

$\Pi_{ІМ'Я}(СЛУЖБОВЦІ \bowtie (\sigma_{проект\# = 5}(ВІДД)))$

Порядок операцій у цьому виразі – зліва праворуч. З урахуванням дужок маємо перехід від внутрішніх до зовнішніх підвиразів. Зазначений порядок, що відповідає обходу дерева «знизу нагору», не є найкращим з погляду витрат на обробку на місцях, тобто виконання операцій у вузлах і передачі даних. Передача даних відповідає галузям зі ступенем розгалуження більше одиниці в дереві запиту для відношень, що

зберігаються в різних вузлах. Розглянутий запит можна оптімизувати із урахуванням двох відношень еквівалентності:

- *Каскадність* унарних операцій показує, що задана унарна операція може бути розбита на частині, що виконуються послідовно. Наприклад, операцію вибірки з відношення кортежів, які одночасно відповідають умові типу $A\ \theta\ c$, для яка задана для двух окремих атрибутів, тобто $A_1\ \theta\ c_1$ AND $A_2\ \theta\ c_2$ можна розбити на дві операції вибірки з простою умовою:

  замість $\sigma_{A_1\theta\, c_1 \text{ AND } A_2\theta\, c_2}(R)$ написати

  $R_1 \leftarrow \sigma_{A_1\theta\, c_1}(R)$

  $\sigma_{A_2\theta\, c_2}(R_1)$

- *Дистрибутивність* унарних операцій щодо бінарних дозволяє поширити їх на операнди бінарних операцій. Наприклад:

$$R \bowtie_{A\,\theta\,c} S \equiv (\sigma_{A\,\theta\,c}(R)) \bowtie (\sigma_{A\,\theta\,c}(S))$$

де R і S – відношення, A – атрибути, c — константа, θ — оператор порівняння.

Можливо більш раннє застосування операцій селекції приводить до оптимального виконання запиту. Це відповідає переміщенню унарних операцій у напрямку до листів дерева. (Унарні операції – це операції селекції й проекції, що зменшують розміри відношення по горизонталі й вертикалі відповідно.)

14.4 Одночасна обробка й відновлення

У розподілених базах даних можуть одночасно виконуватися кілька транзакцій; при цьому можуть використовуватись ті самі дані, можливо, продубльовані в різних вузлах. При читанні для одержання потрібної інформації досить однієї копії, якщо система забезпечує несуперечність всіх копій. При відновленнях, однак, для збереження несуперечності копій потрібно їх вчасно модифікувати.

Як ми вже знаємо, відновлення потрібно робити відповідно до послідовних протоколів. Якщо враховувати одночасне виконання транзакцій над численними копіями, послідовність відновлень вимагає розвинених засобів синхронізації. Це приводить до компромісних рішень при керуванні відновленнями в розподіленій базі даних. Наприклад, у схемі з *основним вузлом* (або основною копією) для кожного відношення вибирається основний вузол, у якому виконується відновлення інформації. Після успішного завершення відновлення в цьому вузлі починається відновлення вторинних копій для повного завершення даної операції відновлення. У цьому випадку основна проблема при виконанні транзакції

– забезпечити відповідність даних, уже обновлених і ще не обновлених. Більш того, первинний вузол, будучи центральним місцем відновлення, впливає на надійність всієї системи. Необхідно також вибрати додаткові вузли на випадок збоїв.

При запитах, що вимагають тільки зчитування з бази даних, зручним і недорогим засобом забезпечення одночасного обслуговування є «моментальні знімки» основних відносин. Їх можна продублювати там, де це необхідно. Такі знімки робляться (тобто їхні дані засилаються в різні вузли) тоді, коли зручніше за все робити відновлення бази даних (тобто в менш напружені години). До наступного відновлення ці знімки обслуговують користувачів (хоча дані в них старі), і вони не піддані впливу відновлення основних відношень. Можна обмежитися такими компромісами, однак наша мета – обговорити методологію синхронізації транзакцій, про що мова йтиме нижче.

14.5 Блокування в розподілених базах даних

При роботі із блокуваннями, як ми вже говорили, транзакція читання блокує одну копію, а транзакція відновлення повинна блокувати всі копії. Транзакція може прочитати елемент даних, якщо будь-яка копія, що містить цей елемент, блокована по читанню, і обновити його, якщо всі копії, що містять цей елемент даних, блоковані по запису. Блокування по читанню буде забезпечена доти, поки інша транзакція не встановить блокування по запису. Інакше кажучи, для того самого елемента даних не можуть існувати одночасно блокування по читанню й запису.

Як відзначалося, двофазний протокол забезпечує послідовність блокувань. Спочатку повинні бути оброблені всі блокування, а потім можна здійснити розблокування, тобто

- перед виконанням читання необхідно забезпечити блокування по читанню, щоб заборонити блокування по запису;
- перед відновленням необхідно забезпечити блокування по запису всіх копій, що містять необхідний елемент даних;
- після установки блокування його скасування не допускається до завершення транзакції.

У розподілених базах даних необхідно забезпечити передачу повідомлень про блокування: спочатку послати запити на блокування в усі вузли, потім одержати підтвердження. Після завершення операції потрібно послати запити на зняття блокувань в усі вузли, де зберігається інформація.

У схемі з основним вузлом установка й зняття блокувань для всіх транзакцій відбуваються для основного вузла. Коли надходить запит на

блокування, перевіряється, чи не заблокований елемент даних, що перебуває в основному вузлі, іншою транзакцією. Якщо елемент не заблокований, він блокується. У цій схемі потрібно мати графи передування або очікування блокувань для виявлення тупикових ситуацій. Крім того, при цій схемі значно зменшується потік повідомлень про блокування, оскільки використається лише один основний вузол для установки й зняття блокувань.

У розподілених базах даних заблокувати можна один основний вузол або кілька вузлів при синхронізації розподілених відновлень. В останньому випадку потрібні графи очікування блокувань для всієї мережі. (У такому графі вузли є транзакції, а ребра – дані. Спрямоване ребро виходить із вузла, що запитує блокування деякого елемента даних, і йде до вузла, у якому ці дані в сучасний момент заблоковані. Цикл у такому графі позначає тупикову ситуацію.)

У кожному вузлі перебуває свій локальний граф очікування блокувань, у якому зв'язок з іншою частиною мережі представлений спеціальною вершиною. У цій схемі можлива глобальна тупикова ситуація, хоча жоден з локальних графів не містить циклу. Для того щоб виявити подібну ситуацію, вузли зв'язують свої локальні графи попарно через спеціальні зовнішні вершини доти, поки не буде виявлена тупикова ситуація. При синхронізації розподілених відновлень на основі блокувань у центральному вузлі може перебувати загальний граф очікування блокувань. У цьому випадку всі вузли посилають запити на установку й зняття блокувань у центральний вузол.

14.6 Часові мітки

При використанні часових міток виключається можливість тупикової ситуації. У цьому випадку одночасне виконання транзакцій еквівалентно деякому послідовному виконанню, що визначається часовими мітками. Хоча тупикові ситуації виключаються, можуть виникнути надлишкові рестарти й відмови.

При часовому маркіруванні кожної транзакції при вході в мережу привласнюється унікальна мітка. У якості мітки може використовуватись показання годиника глобальної мережі, доповнене ідентифікатором вузла. Кожний елемент у базі даних має часові мітки останніх виконаних над ним транзакцій читання й відновлення. Ці мітки називаються мітками читання й запису відповідно. Якщо транзакція T_1 запитує операцію, що вступає в конфлікт із операцією, яка уже виконується відповідно до більш пізньої за часом транзакцією T_2, то T_1 запускається знову. Конфлікт між T_1 і T_2 виникає, якщо операція транзакції T_1 є операція читання, але об'єкт уже записаний транзакцією T_2 (випадок 1), або операція запису, а об'єкт

уже був прочитаний або записаний транзакцією T_2 (випадок 2). Якщо транзакція перезапускається, їй привласнюється нова часова мітка.

Итак, якщо t - *часова* мітка транзакції, а t_r і t_w — часи читання й запису елемента даних, то необхідно:

а) виконати читання, якщо $t \geq t_w$, або запис, якщо $t \geq t_r$ і $t \geq t_w$; у першому випадку час читання встановити рівним t, якщо $t > t_r$, у другому випадку час запису встановити рівним t, якщо $t > t_w$;

б) нічого не робити, якщо транзакція виконує запис і $t_r < t < t_w$;

в) видати відмову, якщо транзакція виконує читання й $t < t_w$ або транзакція виконує запис і $t < t_r$.

Хоча система з часовими мітками має певні переваги, вона може викликати надлишкові перезапуски й відмови. Для зниження цього ефекту використовується система *консервативних часових міток,* при якій події відбуваються в строгій часовій послідовності й вузли обмінюються запитами також у строгій часовій послідовності.

14.7 Системи з голосуванням

Синхронізацію транзакцій на відновлення можна забезпечити, якщо дати можливість кожному вузлу мережі «голосувати» на користь відновлення. У деяких алгоритмах запит на відновлення виконується, якщо всі взаємодіючі вузли голосують за виконання цього запиту. Така система називається *системою синхронізації з одноголосним голосуванням.* Також використовується *мажоритарний* принцип, тобто досить, щоб за проведення відновлення проголосувало більшість вузлів. Ця система заснована на тім, що жоден вузол не може одночасно голосувати за виконання двох конфліктуючих транзакцій. При наявності запиту на відновлення вузол може видати один з наступних сигналів:

а) ОК для проведення відновлення;

б) REJ для відхилення відновлення;

в) PASS для позначення, що можливо тупикову ситуацію;

г) вузол може втриматися від голосування по даному запиті.

Якщо поточний запит суперечить запиту, по якому вже була отримана більшість голосів, вузол повинен відхилити поточний запит. Якщо запит не суперечить іншим запитам у цьому вузлі, він приймається. Може виникнути така ситуація, коли поточний запит суперечить попередньому запиту, по якому вузол проголосував ОК (так званий припинений запит). Запит називається *припиненим,* якщо результат голосування по ньому ОК, але системою він ще не прийнятий. Якщо пріоритет поточного запиту нижче, ніж припиненого, вузол видає сигнал

PASS, у противному випадку він утримується від голосування, але вертається до цього запиту пізніше. Сигнал PASS означає, що можливо тупикову ситуацію. При мажоритарному голосуванні по якому-небудь запиту недостатньо одержати сигнали ОК більш ніж від половини вузлів. Потрібно також, щоб жоден вузол не голосував проти при оцінці результатів голосування. Така система досить стійка, оскільки збій в одному вузлі не впливає істотно на операцію, тому що для її виконання потрібно опитати тільки більшість вузлів.

Відомі й інші схеми. Наприклад, використовується попередній аналіз: транзакції заздалегідь класифікують і складають граф конфліктів, для того, щоб мінімізувати труднощі синхронізації на етапі виконання. Однак така система буде статичної, тому що аналіз транзакцій, їхня класифікація й побудова графів конфліктів у процесі виконання зажадали б занадто великих обчислювальних витрат.

Для всіх систем синхронізації, як тільки синхронізація виконана, і можна проводити відновлення, говорять, що транзакція вступила у фазу фіксації. Як уже говорилося, процес фіксації вимагає протоколу. Розглянутий двофазний протокол фіксації придатний також для розподілених баз даних. Запити й підтвердження проходять через вузли попарно. Відновлення відбувається тільки в тому випадку, якщо всі вузли згодні на фіксацію і якщо запит у відповідному вузлі успішно завершується після одержання підтвердження від всіх вузлів про їхню згоду.

15 СИСТЕМИ КЕРУВАННЯ БАЗАМИ ЗНАНЬ

15.1 Термінологія

Існують різні визначення для баз знань і систем керування базами знань. Ми розуміємо СБЗ широко як систему, що дає можливість використовувати представлені підходящим способом знання за допомогою обчислювальної машини.

Термін "система баз знань" не є загальноприйнятим; існують інші близькі за змістом і більше розповсюджені терміни. Цей термін утворений за аналогією з терміном "система баз даних", що є калькою англомовного терміна data base system. *Під системою баз даних* (СБД) розуміють як інструментальну систему, що забезпечує створення й використання баз даних, так і систему, що забезпечує функціонування конкретної прикладної бази даних або декількох баз, тобто прикладну систему. У першому значенні в україномовній літературі звичайно вживається термін "система керування базами даних" (СКБД).

Ключовим поняттям у системах баз даних, що виражають їхню специфіку, є *база даних (БД),* а для систем баз знань – поняття *бази знань (БЗ)*. Аналогічно СБД *системою баз знань (СБЗ)* називають систему, що забезпечує створення й використання баз знань. Її розглядають як інструментальну систему, яку також називають *системою керування базами знань (СКБЗ),* або як прикладну систему з конкретною прикладною базою знань.

Однак в англомовній літературі замість knowledge base system (система баз знань) воліють вживати термін knowledge based system (KBS), що означає *система, заснована на знаннях (СЗЗ),* або *система, що базується на знаннях (СБЗ)*. Кращим, імовірно, є в якості українського технічного терміна останній, що дає, до речі, колишнє скорочення **СБЗ.**

До цього терміна близький за змістом термін *"експертна система"* (ЕС). У ньому акцент робиться на знання експертів, тобто фахівців у певній області. У літературі можна зустріти кілька іноді досить багатослівних і досить екзотичних визначень ЕС, але суть їх полягає в тому, що ЕС – це система, що забезпечує створення й використання за допомогою обчислювальної машини баз знань експертів, тобто по суті те ж, що СБЗ.

15.2 Принципи, структура й функції систем баз знань (СБЗ)

Аналізуючи схеми типових ЕС або СБЗ, можна виділити в них три основні частини – базу знань (БЗ), механізм одержання рішень (МР) і інтерфейс (ІФ), як показано на рисунку 15.1.

Кожна із цих частин може бути влаштована по-різному в різних системах, причому відмінності можуть бути як у деталях, так і в

принципах. Слід також зазначити що між цими частинами немає абсолютної границі, їхнє розмежування досить умовно, вони можуть "перетинатися".

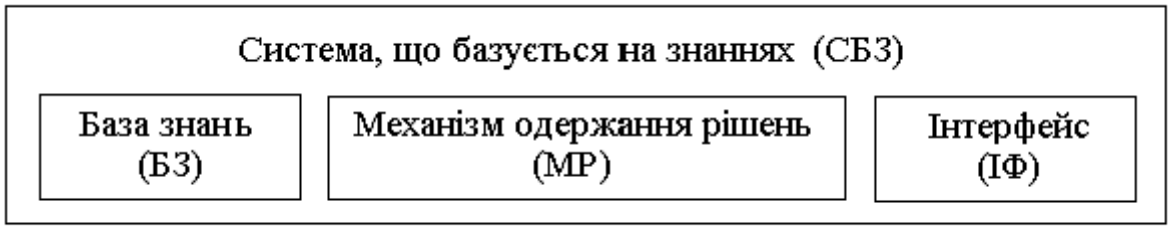


Рис. 15.1 Основні частини СБЗ

Сама характерна риса СБЗ – наявність і використання БЗ. Термін *"база знань"* не має однозначного тлумачення. Так, у доповіді "Технологія конструювання баз знань" А.С.Нариньяни, керівник розробки технологічного комплексу побудови баз знань, сказав: "Що таке база знань - ніхто не знає. Це поняття широке і розмите, існує два основних різночитання: 1) БЗ як пакет знань, що занурений у систему; 2) БЗ як інтегрована система. Я розумію БЗ у другому сенсі". У такому ж сенсі трактує цей термін С.С.Лавров, що зображує структуру БЗ так, як показано на наступному рисунку:

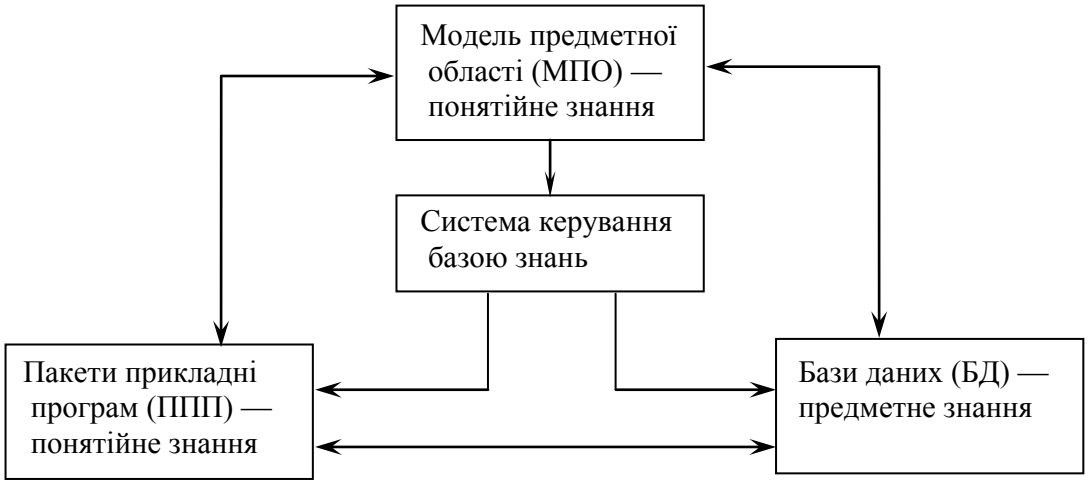


Рис.15.2 Структура СУБЗ

У контексті СБЗ і ЕС, зв'язаному в остаточному підсумку з "програмуванням знань", корисно розрізняти алгоритмічні й неалгоритмічні знання. *Алгоритмічні* (або *процедурні) знання* — це алгоритми (програми, процедури), що обчислюють функції, виконують перетворення, вирішують точно визначені конкретні завдання. Базою алгоритмічних знань можна вважати будь-яке зібрання (бібліотеку) програм. Кожна система програмування й операційна система містять у

собі базу алгоритмічних знань (особливо яскравий приклад - операційна система UNIX). Основою кожного пакета прикладних програм (ППП) або проблемно-орієнтованої системи (ПОС) є база алгоритмічних знань у конкретній прикладній області.

Неалгоритмічні знання охарактеризувати набагато сутужніше, ніж алгоритмічні. Вони складаються насамперед з уявних об'єктів, що називаються поняттями. Поняття звичайно має ім'я (можливо, кілька імен-синонімів), визначення, структуру (частини, елементи й т.д.), воно пов'язане з іншими поняттями й входить у якусь систему понять. Іншого роду неалгоритмічні знання – це зв'язки між поняттями або твердження про властивості понять і зв'язки між ними. Можна виділити два види неалгоритмічних знань: концептуальне (понятійне) і фактуальне (предметне). При цьому концептуальну частину бази знань називають моделлю предметної області (МПО), алгоритмічну – пакетом прикладних програм (ППП), фактуальну – базою даних (БД), а ту частину, представлену на рисунку вище системи рішення завдань, що має справу з концептуальним знанням, можна називати системою штучного інтелекту (СШІ).

Однак знання, втілені в поняттях, не зводяться до моделей предметних областей. Математичні знання, що складаються з математичних понять, зв'язків між ними й тверджень про них, принципово відрізняються від знань у предметних областях – фізиці, економіці, техніці й т.д., у яких уявні об'єкти (поняття) співвіднесені з реальними об'єктами (а в математиці тільки один з одним).

У багатьох ЕС і СБЗ уміст бази знань розділяють на “факти” і “правила”, причому факти відіграють роль елементарних “одиниць знання” (простих тверджень про характеристики об'єктів), а правила служать для вираження зв'язків, залежностей між фактами і їхніми комбінаціями. У системах баз даних цьому розподілу відповідає розподіл на об'єкти й зв'язки, прийняте в багатьох концептуальних моделях даних. База даних містить не тільки "факти" (записи об'єктів з їхніми атрибутами), але й зв'язки, тобто вона не обмежується "фактуальними знаннями". Отже, традиційну БД можна розглядати як своєрідну базу неалгоритмічних знань (про поточну ситуацію в тій реальній обстановці, що ця БД моделює). Розвиток так званих семантичних моделей даних і мов запитів до БД збільшує подібність СБД зі СБЗ.

Таким чином, ми приходимо до наступної практично корисної класифікації знань, що застосовується в реальних СБЗ: 1) поняття (математичні й нематематичні); 2) факти; 3) правила, залежності, закони, зв'язки; 4) алгоритми, процедури.

Системи, що зараховуються їхніми авторами до СБЗ**,** відрізняються від традиційних СБД способами подання неалгоритмічних знань, з відповідними механізмами одержання рішень, і характером знань, що поміщають у БЗ, – знань фахівців (експертів), у яких відбиваються розуміння конкретної прикладної області й прийоми рішення виникаючих у ній типових завдань.

Можна відзначити ще два класи прикладних систем, у яких легко розгледіти БЗ. Це навчальні системи, що включають у себе "машинний підручник" і засоби керування навчанням; і системи підтримки прийняття рішень, що містять інформацію й засоби для моделювання й оцінки ситуації, що полегшують прийняття рішень.

Звичайну бібліотеку теж можна вважати БЗ, але в контексті СБЗ мова йде про знання, збережені у машинній пам'яті й більшою мірою "готові до вживання", чим знання, що перебувають на полках бібліотеки. Доступність знань у машинній формі й можливість прямого їхнього використання для рішення конкретних завдань є характерною рисою БЗ у СБЗ.

Пряме використання знань із БЗ для рішення завдань забезпечується *механізмом одержання рішень (МР)* – другою основною частиною СБЗ, яка називається також механізмом, або машиною виводу (перший термін — з логіки, другий — буквальний переклад inference engine), процедурою пошуку, планування, рішення й т.д. МР дає можливість витягати із БЗ відповіді на питання, одержувати рішення завдань, які формируються у термінах понять, що зберігаються в БЗ. Звичайно це "часткові" завдання й питання такого роду: "Дане те-те, знайти те-те"; "Знайти об'єкти, що задовольняють такий-те умові"; "Які дії виконати в такий-те ситуації".

Принципи роботи МР тісно зв'язані зі способами подання знань у БЗ. Для знань, представлених у БЗ рівняннями, МР є процедурою рішення рівнянь, для знань, представлених логічними формулами або правилами спеціального виду, це деякий механізм виводу й т.д. У СБД і ППП, що не претендують на звання СБЗ, теж є механізм одержання рішення. У СБД він забезпечує, зокрема, переходи по зв'язках між об'єктами й пошук об'єктів, що задовольняють заданій умові. У ППП він установлює зв'язки між модулями, необхідні для одержання рішення, і настроює модулі на параметри конкретного завдання. Однак левова частина одержання рішення в ППП припадає на самі модулі – програми рішення прикладних завдань.

У кожному разі МР містить алгоритм одержання рішення, тобто спеціальне алгоритмічне знання. Тому що в МР утримується принаймні частина семантики БЗ, обумовлена в процедурній формі, це підтверджує відносність межі між МР і БЗ.

Третя основна частина СБЗ, що ми назвали *інтерфейсом (ІФ),* забезпечує роботу із БЗ і МР мовою досить високого рівня, наближеною до професійної мови фахівців у тій прикладній області, до якої відноситься СБЗ. Для цього в ІФ включається відповідний мовний процесор. Крім того, у функції ІФ входить підтримка діалогу користувача із системою, що дає можливість користувачеві одержувати пояснення дій системи, брати участь у пошуку рішення, поповнювати й коректувати БЗ. Іноді інтерфейс трактують дуже широко, розуміючи під "системою інтелектуального інтерфейсу" те, що ми називаємо СБЗ.

Розходження в поглядах на поняття СБЗ (БЗ, ЕС і т.д.) нерідко зводяться до різних розміщень акцентів на трьох згаданих частинах СБЗ, оцінкам їхньої відносної значимості й внеску в загальний ефект системи. Суть СБЗ полягає в тому, що знання, так чи інакше представлені в машинній пам'яті, "працюють" на користувача, дають ефект, що виправдує витрати на побудову або придбання СБЗ. Таким чином, ми приходимо до прагматичного визначення СБЗ і ЕС як прикладної системи, що забезпечує працездатність закладених у машинну пам'ять знань фахівців (експертів). Це дає критерій, по якому варто оцінювати СБЗ або ЕС незалежно від того, яка із трьох згаданих частин у ній переважає, яке в ній співвідношення алгоритмічного й неалгоритмічного знання й наскільки примітивні або витончені засоби й методи в ній використовуються.

СПИСОК 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..