Олександр Романюк
Тамара Савчук

ОРГАНІЗАЦІЯ БАЗ ДАНИХ І ЗНАНЬ

Міністерство освіти і науки України
Вінницький національний технічний університет

О. Н. РОМАНЮК, Т. О. САВЧУК

ОРГАНІЗАЦІЯ БАЗ ДАНИХ І ЗНАНЬ

Рекомендовано Міністерством освіти і науки України як навчальний посібник для студентів вищих навчальних закладів, що навчаються за спеціальностями "Інтелектуальні системи прийняття рішень" та "Програмне забезпечення автоматизованих систем"

УНІВЕРСУМ-Вінниця
2003

УДК 681.31
Р 69

Рецензенти:
В.Г.Зайцев, доктор технічних наук
В.М.Лисогор, доктор технічних наук
О.І.Гороховський, кандидат технічних наук

Рекомендовано до видання Міністерством освіти і науки України. Лист № 14/18-2-1702 від 22.10.03р.

О.Н.Романюк, Т.О.Савчук

Р 69 **Організація баз даних і знань.** Навчальний посібник. — Вінниця: УНІВЕРСУМ-Вінниця, 2003. — 217 с.

ISBN 966-641-081-8

В навчальному посібнику розглянуто основи організації та управління базами даних та знань, викладені основні положення, наведені приклади і рекомендації по їх засвоєнню. Велика увага приділена сучасним засобам управління базами даних, приведені контрольні запитання, що можуть бути використані при проведенні як практичних, так і лабораторних занять.

Навчальний посібник призначено для студентів спеціальностей "Інтелектуальні системи прийняття рішень" та "Програмне забезпечення автоматизованих систем" денної та. заочної форм навчання.

УДК 681.31

ISBN 966-641-081-8

ЗМІСТ

ВСТУП

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

Збільшення об'єму і структурної складності збережених даних, розширення кола користувачів інформаційних систем висунуло вимогу створення зручних загальносистемних засобів інтеграції збережених даних і керування ними. Це привело до появи промислових систем керування базами даних (СКБД) - спеціалізованих програмних засобів, призначених для організації і ведення БД. Систему, яка забезпечує створення, ведення і застосування баз знань, можна розглядати як інструментальну систему або як прикладну систему з конкретною прикладною базою знань. Існує тісний взаємозв'язок між технологією БД і систем БД - з одного боку, і технологією систем баз знань з іншого. Виникла тенденція "інтелектуалізації" систем БД. На зовнішньому рівні їх архітектури реалізують різноманітні семантичні моделі даних, створюють "дружні" інтерфейси для користувачів, хоча традиційні СКБД є необхідною складовою частиною інструментарію керування даними в системах баз знань.

Наявна література з баз даних і знань не в повній мірі задовольняє потреби студентів вузів, інженерно-технічних працівників і інших спеціалістів, які займаються програмуванням та обробкою даних. Відчувається потреба в навчальному посібнику, в якому б з єдиних методологічних позицій викладались основи баз даних і знань від організації їх структур до систем керування.

В навчальномі посібнику приведені основи організації баз даних і знань на концептуальному, фізичному та логічному рівнях, а також особливості керування ними.

1 ОСНОВНІ ПОНЯТТЯ ПРО БАНКИ ДАНИХ

1.1 Визначення і класифікація інформаційних систем

Трудова діяльність людини постійно пов'язана зі сприйняттям і накопиченням інформації про навколишнє середовище, відбором і обробкою інформації при розв'язуванні різних задач, обміном нею з іншими людьми. З часом комплекс цих операцій, методи і засоби їхньої реалізації послужили основою для створення інформаційних систем, основне призначення яких - інформаційне забезпечення користувача, тобто надання йому необхідних даних із визначеної предметної області. Завдяки появі ЕОМ стало можливим створення автоматизованих інформаційних систем (АІС).

У розвитку АІС намітилися два покоління [1]:

1-е покоління - інформаційні системи, які базуються на автономних файлах. Це системи з простою архітектурою й обмеженим набором можливостей. Вони складаються із набору автономних файлів і комплексу прикладних програм, призначених для обробки цих файлів і видачі документів. Такі системи мають ряд серйозних недоліків, що обмежують їхнє широке застосування: високу надлишковість даних, складність ведення і спільної обробки файлів, залежність програм від даних і ін.

2-е покоління - банки даних. Це системи з високим ступенем інтеграції даних і автоматизації керування ними. Вони орієнтовані на колективне користування й, в основному, позбавлені недоліків, властивих АІС 1-го покоління.

Функціонування АІС пов'язано з накопиченням і обробкою інформації. Під інформацією розуміється сукупність знань про фактичні дані і залежності між ними. У ЕОМ поняття інформації і даних часто ототожнюються. Але якщо бути точними, то дані - це інформація, подана у

формі, необхідній для введення її в ЕОМ, збереження, обробки і видачі споживачам.

Інформація, яка вводиться в АІС і видається системою користувачу, представляється у вигляді документів. Документ - це матеріальний об'єкт, який містить інформацію, що має відповідно до чинного законодавства правове значення, і призначений для передачі і використання. Джерелом інформації в АІС є люди і датчики, споживачами - люди (користувачі).

Звертання користувачів до АІС здійснюється у вигляді запитів. Запит - це формалізоване повідомлення, що надходить на вхід системи. Він включає умову на пошук даних, а також вказівку про те, що необхідно проробити зі знайденими даними.

Інтерпретація введених запитів, виконання дій, зазначених у них, формування і виведення повідомлень і документів складають основні етапи роботи АІС. У цілому під автоматизованою інформаційною системою розуміється сукупність інформаційних масивів, технічних, програмних і мовних засобів, призначених для збору, збереження, пошуку, обробки і видачі даних за запитами користувачів.

Використання АІС може здійснюватися одним із двох способів, суть яких це:[1,2]:

1. Автономне функціонування системи, при якому АІС не входить до складу інших систем і використовується самостійно. Прикладом можуть служити такі АІС, як документальні (бібліотечні) інформаційно-пошукові системи, а також системи резервування авіа- і залізничних квитків типу "Сирена" і "Експрес", у яких відповіддю на запит пасажира є документ у вигляді квитка або повідомлення про відсутність вільних місць.

2. Використання АІС в якості складової частини іншої автоматизованої системи. У цьому випадку вихідні дані можуть використовуватися не тільки кінцевими користувачами, але й іншими користувачами цієї автоматизованої системи з метою подальшої обробки і застосування у виробничому процесі. Так, у навчальних системах АІС

містить досліджуваний матеріал, набір питань, задач і відповідей, у САПР - нормативно-довідкову інформацію, зведення про ДСТ і інші дані, в АСУ - всю інформацію, необхідну для керування підприємством, тобто для аналізу, оцінки, прогнозування, виробітку рішень, планування, контролю виконання.

Інформаційні системи можна класифікувати за рядом ознак. В основу класифікації, приведеної на рис.1.1, покладені найбільш істотні ознаки, що характеризують можливості й особливості сучасних АІС.

Документальні інформаційно-пошукові системи (ДІПС) призначені для збереження і обробки документальних даних - адрес збереження документів. Такі дані подаються в неструктурованому вигляді. Прикладом ДІПС є бібліотечні, бібліографічні АІС. На відміну від систем цього класу фактографічні інформаційно-пошукові системи (ФІПС) зберігають і оброблюють фактографічну інформацію - структуровані дані у вигляді чисел і текстів. Над такими даними можна виконувати різні операції. Більшість АІС являють собою системи класу ФІПС.

Друга ознака класифікації поділяє інформаційні системи на дві групи: до першої відносяться інформаційно-довідкові системи (ІДС), які виконують пошук і виведення інформації без її обробки. Автоматизовані інформаційні системи обробки даних (АІСОД, ІСОД), що відносяться до другої групи об'єднують у собі інформаційно-довідкову систему і систему обробки даних. Обробка знайдених даних виконується комплексом прикладних програм. Більшість АІС побудована за принципом ІСОД.

Ступінь інтеграції даних і автоматизації керування ними є найважливішою ознакою класифікації АІС. У ранніх системах - АІС на автономних файлах (АІС АФ) - принцип інтеграції даних практично не використовувався, а рівень автоматизації керування файлами був порівняно низьким. Такі системи застосовуються і в даний час; вони ефективні у випадку вузького, спеціалізованого використання невеликим колом осіб. Високий ступінь інтеграції мають банки даних (БнД).

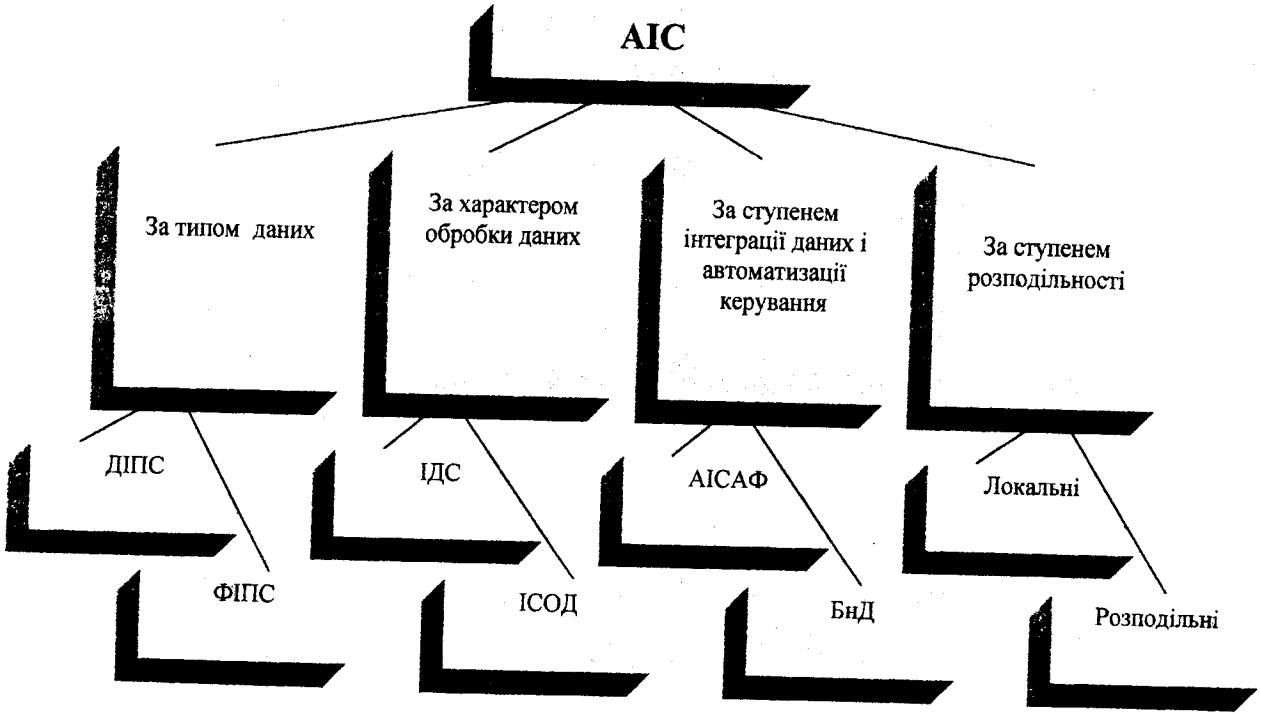


Рисунок 1.1 - Класифікація автоматизованих інформаційних систем

У порівнянні з АІС на автономних файлах у БнД збережена інформація зосереджена в єдиному інформаційному масиві - базі даних (БД), а процес маніпулювання даними автоматизований.

Останній із приведених ознак класифікації враховує розподільність компонентів АІС: локальна система розміщена на одній ЕОМ, у той час як розподілена система функціонує в середовищі обчислювальної мережі і розподілена по її вузлах (серверах і робочих станціях).

Контрольні запитання

1. Дайте визначення автоматизованої інформаційної системи.
2. За якими ознаками класифікують автоматизовані інформаційні системи?
3. Які недоліки мають автоматизовані інформаційні системи на автономних файлах?
4. Які автоматизовані інформаційні системи мають найбільшу ступінь інтеграції?

1.2 Автоматизовані банки даних

Автоматизований банк даних (БнД) - це організаційно-технічна (людино-машинна) система, що представляє собою сукупність інформаційної бази, колективу фахівців і комплексу програмних і технічних засобів забезпечення її функціонування, призначена для збереження, пошуку і видачі інформації у вигляді, зручному для користувачів. [4,23]

На загальній схемі структури автоматизованого банку даних (рис.1.2) прийняті такі позначення: БД - база даних; ПП - прикладні програми користувачів банку даних; СКБД - система керування базою даних, тобто комплекс програмних засобів, який забезпечує завантаження інформації в базу даних, реорганізацію та ведення бази, пошук та перетворення інформації для забезпечення роботи програм користувачів банку даних.

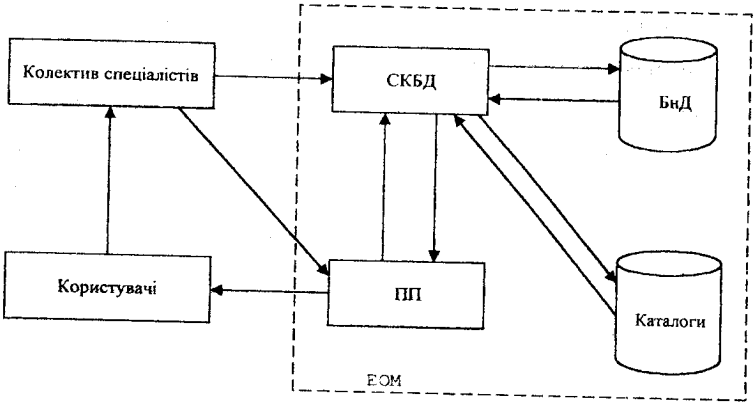


Рисунок 1.2 – Структура автоматизованого банку даних

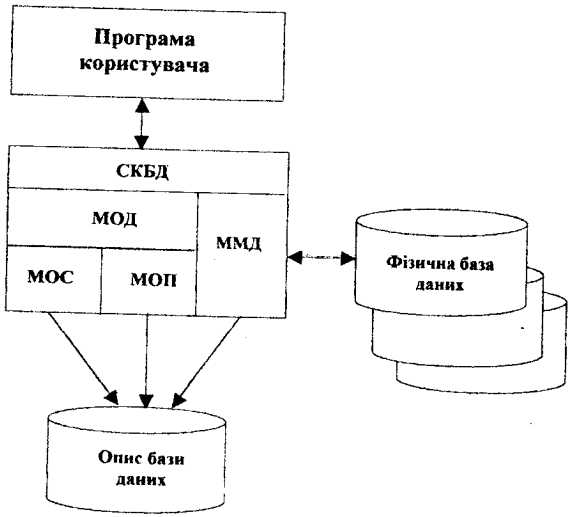


Рисунок 1.3 – Архітектура СКБД

Розглянемо більш детально структурні складові частини цього автоматизованого банку даних.

База даних. Відомі два підходи до організації інформаційних масивів: файлова організація та організація у вигляді бази даних. Файлова організація передбачає спеціалізацію та збереження інформації, орієнтованої, як правило, на одну прикладну задачу, та забезпечується прикладним програмістом. Така організація дозволяє досягнути високої швидкості обробки інформації, але характеризується рядом недоліків.

Характерна риса файлового підходу - вузька спеціалізація як обробних програм, так і файлів даних, що служить причиною великої надлишковості, тому що ті самі елементи даних зберігаються в різних системах. Поскільки керування здійснюється різними особами (групами осіб), відсутня можливість виявити порушення суперечливості збереженої інформації. Розроблені файли для спеціалізованих прикладних програм не можна використовувати для задоволення запитів користувачів, які перекривають дві і більше області. Крім того, файлова організація даних внаслідок відмінностей структури записів і форматів подання даних не забезпечує виконання багатьох інформаційних запитів навіть у тих випадках, коли всі необхідні елементи даних містяться в наявних файлах. Тому виникає необхідність відокремити дані від їхнього опису, визначити таку організацію збереження даних з обліком існуючих зв'язків між ними, яка б дозволила використовувати ці дані одночасно для багатьох застосувань. Вказані причини обумовили появу баз даних.

База даних може бути визначена як структурна сукупність даних, що підтримуються в активному стані та відображає властивості об'єктів зовнішнього (реального) світу. В базі даних містяться не тільки дані, але й описи даних, і тому інформація про форму зберігання вже не схована в сполученні "файл-програма", вона явним чином декларується в базі.

База даних орієнтована на інтегровані запити, а не на одну програму, як у випадку файлового підходу, і використовується для інформаційних

потреб багатьох користувачів. В зв'язку з цим бази даних дозволяють в значній мірі скоротити надлишковість інформації. Перехід від структури БД до потрібної структури в програмі користувача відбувається автоматично за допомогою СКБД.[3]

СКБД - це складна програмна система накопичення та з наступним маніпулюванням даними, що представляють інтерес для користувача. Кожній прикладній програмі СКБД надає інтерфейс з базою даних та має засоби безпосереднього доступу до неї. Таким чином, СКБД відіграє центральну роль в функціонуванні автоматизованого банку даних.

Архітектурно СКБД складається з двох великих компонент (рис.1.3). За допомогою мови опису даних (МОД) створюються описи елементів, груп та записів даних, а також взаємозв'язки між ними які, як правило, задаються у вигляді таблиць. В залежності від конкретної реалізації СКБД мову опису даних підрозділяються на мову опису схеми бази даних (МОС) та мову опису підсхем бази даних (МОП). Слід особливо зазначити, що МОД дозволяє створити не саму базу даних, а лише її опис.

Для виконання операцій з базою даних в прикладних програмах використовується мова маніпулювання даними (ММД). Фактична структура фізичного зберігання даних відома тільки СКБД.

З метою забезпечення зв'язків між програмами користувачів і СКБД (що особливо важливо при мультипрограмному режимі роботи операційної системи) в СКБД виділяють особливу складову - резидентний модуль системи керування базами даних. Цей модуль значно менший від всієї СКБД, тому на час функціонування автоматизованого банку інформації він може постійно знаходитись в основній пам'яті ЕОМ та забезпечувати взаємодію всіх складових СКБД і програм, які до неї звертаються.

Приведена структура притаманна усім СКБД, котрі розрізняються обмеженнями та можливостями по виконанню відповідних функцій. Отже, процес порівняння і оцінки таких систем для одного конкретного

застосування зводиться до співставлення можливостей наявних СУБД з вимогами користувачів.

Будь-який аналіз, необхідний для проведення оцінки і вибору СКБД, повинен починатись з ретельного вивчення потреб користувачів. При цьому виконується опис зв'язків між елементами даних в базі даних, а також для оцінки експлуатаційних характеристик визначаються вимоги до часу виконання кожної транзакції. Під транзакцією розуміється повідомлення, яке передається від прикладної програми до СКБД. Транзакція ініціює в останній роботу певного виду чи окрему операцію по обробці даних.

За допомогою мови опису даних адміністратор, а іноді і інші програмісти описують для СКБД вміст та структуру бази даних. При розгляданні МОД слід з'ясувати її власні характеристики (наприклад, простоту використання та наочність), а також проаналізувати обмеження СКБД на дані (наприклад, типи даних чи можливості по обмеженню доступу).

Засоби маніпулювання даними визначають методи доступу до бази даних та мову (мови), за допомогою якої відбувається цей доступ. Мова маніпулювання даними є засобом, який застосовується користувачами чи прикладними програмістами для виконання операцій над базою даних. При порівнянні можливостей СКБД з сучасними принципами обробки даних важливе значення має зв'язок ММД з існуючими мовами програмування. Простота її вивчення і використання розширює можливості розробки банку даних з конкретною СКБД. Степінь процедурності ММД визначає міру незалежності програм від даних . Чим менш процедурна мова, тим менша ймовірність зміни програм ,написаних з її використанням, при включенні нових методів доступу і навіть нових структур даних в СКБД.

Нарешті, засоби маніпулювання даними визначають можливості паралельної обробки. Для збереження цілісності бази даних в умовах її одночасного використання декількома прикладними програмами звичайно вводяться обмеження на доступ для всіх, крім одного, з процесів, які

виконуються одночасно. Наприклад, якщо програма поновлює запис бази даних, то їй може надаватись доступ до поля, що змінюється, до запису чи до всього фізичного файлу, в який цей запис входить.

Основна мета застосування баз даних – забезпечити незалежність логічної бази даних та прикладних програм від методів зберігання фізичної бази даних. При цьому способи доступу значно впливають на експлуатаційні характеристики банку даних.

Не менш важливу роль відіграють засоби копіювання та відновлення бази даних. Найбільш суттєвим тут є наявність автоматичного режиму ведення журналу фіксування роботи з базою даних. Наявність стандартних програм СКБД таких, як програма перезавантаження бази даних і програма обробки журналу, спрощують процес відновлення бази даних після апаратурних збоїв.

Серед інших стандартних програм слід відзначити програми завантаження бази даних, трасування роботи з базою даних для полегшення налагодження, а також накопичення і аналізу статистики по експлуатаційних характеристиках. Якщо база даних призначена для використання прикладними програмами, які функціонують в діалоговому режимі, то особливу роль в складі СКБД відіграють засоби передачі даних.

Концепція баз даних припускає інтеграцію даних, що раніше зберігалися окремо. Незалежно від того, охоплює чи не охоплює така інтеграція централізацію фізичних даних, вона припускає ріст спільного використання даних різними прикладними програмами та зменшення надлишковості зберігання одних і тих же даних. Для створення цим процесам сприятливих умов потрібне централізоване керування вмістом баз даних. Об'єктами централізованого керування є форма елементів даних і структури баз даних, а не власне значення самих елементів даних. Функція керування формою та вмістом бази даних називається адмініструванням даних.

Керування вмістом бази даних відбувається шляхом збору і ведення точної та повної інформації про дані. Ця інформація, яку часто називають метаданими, включає опис смислу елементів даних, методів їх використання, джерел, фізичних характеристик, а також різних правил та обмежень. Метадані дозволяють проводити аналіз запитів по нових даних, проектування і програмування нових прикладних систем, супроводження існуючих систем та документування всіх етапів розвитку бази даних.

Дані про базу даних, чи метадані, можливо розбити на три класи: семантична інформація, фізичні характеристики та інформація про використання. Засобами автоматизації формування та використання метаданих являються словники даних (системи словників-довідників даних). Перерахуємо їхні основні функції [4]:

- встановлення зв'язку між користувачами БД;
- здійснення простого та ефективного керування елементами даних при вводі в систему як нових елементів, так і при зміні опису існуючих;
- зменшення надлишковості;
- усунення протиріччя даних;
- централізація керування елементами даних з метою спрощення проектування БД та її розширення.

Для створення ефективного і зручного словника даних необхідно при зборі інформації про дані встановити правила присвоєння елементам імен, добитися однозначного тлумачення різними користувачами призначення джерел і угод по присвоєнню імен, сформулювати прийнятні для усіх користувачів описи елементів даних і виявити синоніми, усунути багатозначність (омонімію, полісемію). Вказаний процес виконується ітеративно і зв'язаний з усуненням конфліктних ситуацій.

В процесі роботи з словником даних є можливість отримати в алфавітному порядку лістинг найменувань типів всіх статей (з зазначенням частоти їх використання в системі, статусу кожної статті та числа структур кожного виду).

Словник даних використовується кінцевими користувачами при роботі з системою на мові запитів, прикладними програмістами – при написанні програм, системними програмістами – в процесі розвитку системи. Словник в умовах організації інформації у вигляді баз даних вводиться до складу опису баз даних та використовується СКБД при роботі компілювальних і інтерпретувальних програм.

З розвитком системи необхідно проводити контроль логічності та повноти даних, які оброблюються в системі. Цей контроль полягає в перевірці за допомогою словника даних відповідності потоків і елементів даних, потоків і джерел даних, процесів обробки та елементів даних.

Особливо важливий словник даних при взаємодії декількох систем обробки даних, при побудові розподілених банків даних, при використанні програм, які виконані в інших організаціях. В останньому випадку словарні статті вилучають з написаних програм, встановлюють синонімічні зв'язки їх зі статтями словника системи і переводять їх в формат, принятий в системі.

Колектив спеціалістів, який забезпечує функціонування автоматизованого банку даних, складається з адміністратора, аналітиків, системних та прикладних програмістів. Взаємодія їх між собою та кінцевими користувачами показана на рис.1.4. Розглянемо детальніше функції, покладені на кожного з перерахованих спеціалістів.

Адміністратор – це спеціаліст, який володіє інформацією про інформаційні потреби кінцевих користувачів, працює в тісному контакті з користувачами і відповідає за визначення, завантаження, захист та ефективність експлуатації баз даних.

Необхідність включення адміністратора в колектив спеціалістів стала актуальною в той період, коли виникла необхідність в централізації обробки даних, що призвело до переходу від файлових систем до інтегрованих баз даних. Оскільки користувачі повинні обслуговуватись усіма засобами автоматизованого банку даних, то адміністратор є

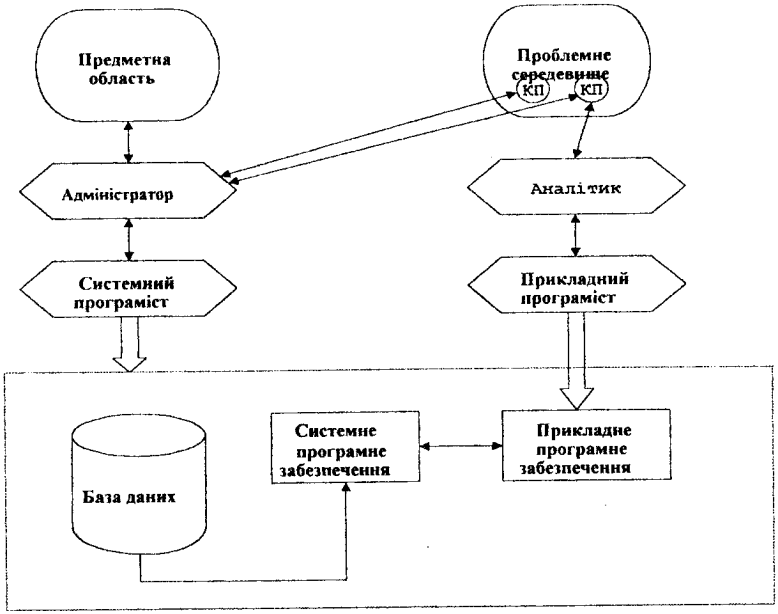


Рисунок 1.4 – Схема взаємодії колективу спеціалістів банку

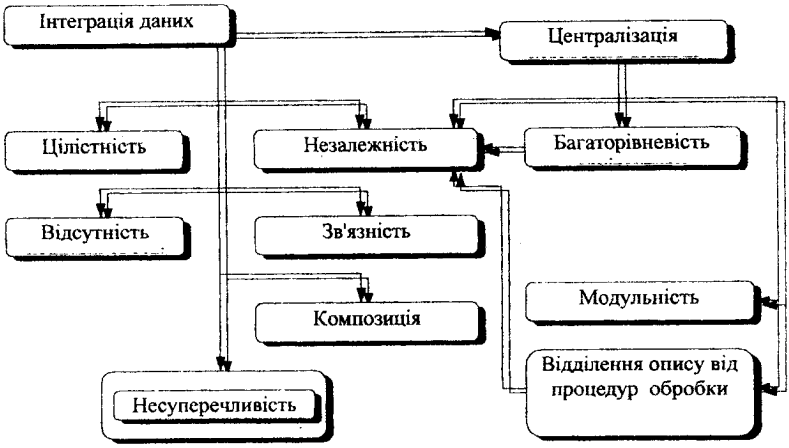


Рисунок 1.5 – Основні принципи побудови банків інформації

відповідальним за аналіз потреб користувачів, проектування бази даних, її впровадження, поновлення та реорганізацію.

Всі задачі, виконання яких покладається на адміністратора, можна розділити у відповідності з етапами розробки автоматизованих банків інформації на чотири групи: планування, проектування, експлуатація та використання.

При плануванні адміністратор бере участь у виборі програмного забезпечення, яке пов'язано з базою даних, та обладнання. Йому доводиться працювати з кінцевими користувачами, щоб встановити реальні цілі та вимоги до прикладних програм і баз даних. Адміністратор повинен гарантувати, що пріоритети розробки та експлуатації прикладних програм відповідають цілям різних категорій користувачів, приймає участь в довгостроковому плануванні, в тому числі у визначенні перспектив розширення бази даних.

При проектуванні адміністратор надає розробникам прикладних систем необхідні засоби для проектування логічної та фізичної баз даних, керує процесом логічного проектування з метою отримання повної картини ресурсів даних. При появі нових вимог до даних адміністратор банку визначає метод, за допомогою якого ці дані можна включити в склад існуючої бази даних, і керує процесом виконання необхідних змін. Він також здійснює вибір методів доступу і методів розміщення даних в фізичній пам'яті, що дозволяє забезпечити виконання вимог прикладних програм.

На етапі експлуатації в обов'язки адміністратора також входить розробка і контроль дій, які гарантують збереження цілісності бази даних, включаючи процедури її копіювання і відновлення, а також організації захисту бази даних за допомогою механізмів керування доступом і засобів СКБД.

І нарешті, адміністратору доводиться взаємодіяти з користувачами бази даних, тому він вводить стандарти на вміст і використання бази даних,

роблячи їх доступними для потенційних користувачів, супроводжує спеціальні засоби програмного забезпечення роботи з базою даних (словники даних, мови запитів). Крім того, він може консультувати користувача бази даних по застосуванню окремих елементів чи усього програмного забезпечення.

Системні програмісти займаються створенням базового математичного забезпечення для ЕОМ. Вони "генерують" операційні системи, СКБД, транслятори з різних мов, розробляють сервісні програми і інші програмні засоби, які забезпечують обробку інформації та вирішення задач на ЕОМ.

Аналітик, користуючись знаннями закономірностей певного проблемного середовища, розробляє його математичну модель, застосовує необхідні математичні методи і методи моделювання. Він оцінює альтернативні варіанти і приймає рішення в залежності від заданих вимог.

Аналітик переводить задачі кінцевого користувача (КК) в деяку вихідну формальну модель. Оскільки КК не являється математиком, то задачу він формулює на мові своєї професії, і все мистецтво аналітика полягає в умінні з цього формулювання побудувати адекватну йому математичну задачу. Результатом роботи аналітика є вихідне представлення задачі для прикладного програміста, метою котрого є перетворення продукту аналітика в програмний продукт, придатний для вводу в ЕОМ.

Контрольні запитання

1. Назвіть основні складові частини автоматизованого банку інформації.
2. Які функції виконує СКБД в банках даних?
3. Яке призначення словника даних?
4. Чим відрізняються МОД та ММД?
5. В чому проявляється інтегрований характер бази даних?
6. В чому полягає відмінність функцій прикладного програміста і аналітика?

1.3 Вимоги до банків даних

Різноманіття інформаційних потреб висуває до банків даних підвищені вимоги. До основних вимог відносяться [5]:

- Адекватність інформації стану предметної області. БнД є інформаційною моделлю предметної області і, як відзначалося вище, інформація, яка зберігається в ньому, повинна повно і точно відображати її об'єкти, їхні властивості і відношення між об'єктами. Відступ від принципу адекватності робить систему марною і навіть небезпечною, неприпустимою для використання. У свою чергу, вимога адекватності породжує ряд нових вимог до системи таких, як необхідність постійного внесення змін у дані і періодичної зміни організації даних.
- Надійність функціонування - одна з найважливіших вимог, які висуваються до будь-якої системи.
- Швидкодія і продуктивність. Ці дві близькі одна до одної вимоги відображають часові потреби користувачів. Перша з них визначається часом відповіді (реакції) системи на запит, який відраховується з моменту введення запиту до моменту початку видачі знайдених даних. Цей час залежить не тільки від швидкодії ЕОМ, але і від способів фізичної організації даних, методів доступу, способів пошуку, складності запиту й інших чинників. Друга вимога визначається кількістю запитів, які відпрацьовуються в одиницю часу.
- Простота і зручність використання. Ця вимога пред'являється до БнД з боку всіх без винятку категорій користувачів, особливо кінцевих. Складність запитів, відсутність сервісу формують у психології користувача небажання працювати з інформаційною системою.
- Масовість використання. Сучасна інформаційна система повинна забезпечувати колективний доступ користувачів, при якому вони

можуть одночасно і незалежно звертатися до баз даних для одержання необхідних даних.

- Захист інформації. Система повинна забезпечувати захист збережених у ній даних і програм як від випадкових спотворень і знищення, так і від навмисних, несанкціонованих дій користувачів.
- Можливість розширення. Архітектура системи повинна допускати розширення її можливостей шляхом модифікації або заміни існуючих програмних модулів, додаванням нових компонентів, а також шляхом реорганізації інформаційних масивів.

Контрольні запитання

1. Які вимоги до БнД відображають часові потреби користувачів?
2. Як Ви розумієте адекватність інформації стану предметної області?
3. Якими шляхами може бути розширена БнД?
4. Які типи захисту інформації повинні бути передбачені в БнД?

1.4 Принципи побудови банків даних

У основі побудови БнД лежать наукові принципи, на основі яких розроблюють високоякісні системи, які відповідають сучасним вимогам. Вибір принципів побудови БнД і їхнє втілення в конкретній системі складають основу проектування [4,5].

З множини використовуваних принципів виділимо найбільш істотні (рис.1.5): принцип інтеграції даних і принцип централізації керування ними. Обидва принципи відображають суть банку даних: інтеграція є основою організації БнД, централізація керування - основою організації і функціонування системи керування базами даних (СКБД). Інші принципи в тій або іншій мірі пов'язані з першими. Окремі з них є їхнім результатом або одним із можливих шляхів реалізації. Так, інтеграція даних припускає

взаємозалежність (зв'язність) даних; зв'язність, у свою чергу разом із принципом композиції дозволяє звести надлишковість даних до мінімуму.

Суть, принципу інтеграції даних полягає в об'єднанні окремих, взаємно не зв'язаних даних у єдине ціле, в ролі якого виступає база даних. В результаті вказаного користувачу і його прикладним програмам всі дані представляються єдиним інформаційним масивом. При цьому полегшуються пошук взаємозалежних даних і їхня спільна обробка, зменшується надлишковість даних, спрощується процес ведення БнД.

Інтеграцію даних необхідно розглядати на двох рівнях - логічному і фізичному. На логічному рівні множина структур даних відображається в єдину структуру даних, на фізичному рівні автономні файли об'єднуються в базу даних [6,7].

Принцип цілісності даних відображає вимогу адекватності збереженої в БнД інформації стану предметної області: у будь-який момент часу дані повинні в точності відповідати властивостям і характеристикам об'єктів. Порушення цілісності виникає внаслідок спотворення або навіть руйнації (стирання) усіх або частини даних, а також як результат запису в базу даних невірної інформації. Підтримка цілісності досягається за рахунок контролю вхідної інформації, періодичної перевірки збережених у БнД даних, застосуванням спеціальної системи відновлення даних, а також іншими заходами.

Під незалежністю даних будемо розуміти незалежність прикладних програм від збережених даних, при якій будь-які зміни в організації даних не вимагають корекції цих програм. Одним із шляхів досягнення незалежності є введення додаткових рівнів абстрагування даних (принцип багаторівневості). Замість двох традиційних рівнів, передбачених базовим програмним забезпеченням і стандартними мовами програмування, - логічного і фізичного - в архітектурі БнД використовується принцип трирівневої організації даних: логічний рівень ділиться на два - зовнішній

(рівень користувача) і концептуальний (загальний системний рівень даних).

Інший шлях досягнення незалежності даних - передача СКБД частини функцій, що раніше покладалися на прикладні програми. Маються на увазі функції, зв'язані з організацією доступу до БнД. При цьому прикладна програма ніяк не зв'язана ні з БнД, ні з методом доступу. Вона лише формує і передає ядру інформацію, необхідну для пошуку даних.

Незалежність даних досягається також застосуванням і дотриманням принципу відділення опису БнД від процедур обробки даних. Нарешті, істотним чинником забезпечення незалежності варто вважати реляційний підхід до побудови БнД - розробку бази даних на основі реляційної моделі даних і використання методів і засобів реляційної алгебри в процесі обробки БнД. Найбільший ефект досягається раціональним сполученням усіх зазначених шляхів.

Відсутність надлишковості - це стан даних, коли кожний елемент присутній у БнД в єдиному екземплярі. Надлишковість може мати місце як на логічному рівні, коли в структурі даних повторюються ті самі типи даних, так і на фізичному рівні, коли дані зберігаються в двох або більше екземплярах. Принцип інтеграції дозволяє звести надлишковість до мінімуму. Під несуперечливістю розуміється смислова відповідність між даними. Це такий стан бази даних, при якому збережені в ній дані не суперечать один одному. Розрізняють два аспекти несуперечливості: смислова відповідність різнотипних даних і ідентичність (рівність) дублюючих даних.

Принцип зв'язності даних полягає в тому, що дані в БнД взаємозалежні, і зв'язки відбивають відношення між об'єктами предметної області. Множина типів даних і множина зв'язків утворять логічну структуру даних. Наявність зв'язків між записами в БнД дозволяє зменшувати надлишковість, спростити і прискорити пошук даних.

Принцип централізації керування полягає в передачі усіх функцій керування даними єдиному комплексу керуючих програм - системі керування базами даних. Як було зазначено вище, всі операції, пов'язані з доступом до БнД, виконуються не прикладними програмами, а централізовано - ядром СКБД - на підставі інформації, яку отримують з цих програм. Дотримання цього принципу дозволяє автоматизувати роботу з базами даних і тим самим істотно підвищити ефект, який отримують від застосування інформаційної системи.

Відділення опису даних від процедур їхньої обробки припускає, що опис даних виключається з прикладних програм, складається і транслюється окремо від них і зберігається в базі даних (або поза нею у вигляді окремого файла). Виведення цих описів за рамки прикладної програми робить її більш незалежною від БнД, полегшує процес програмування, зменшує розміри необхідної для програми пам'яті, підвищує гнучкість маніпулювання даними.

На основі зазначених вище принципів формується архітектура БнД - концепція взаємозв'язку логічних, фізичних і програмних компонентів системи.

Контрольні запитання

1. Назвіть основні принципи побудови банків інформації.
2. Для чого необхідно забезпечити відокремлення опису даних від процедур їхньої обробки?
3. В чому полягає принцип трирівневої організації даних?
4. Покажіть взаємозв'язок основних принципів побудови банків даних.

2 ІНФОЛОГІЧНА МОДЕЛЬ

2.1 Основні поняття

Проектування бази даних треба починати з аналізу предметної області і виявлення вимог до неї окремих користувачів (співробітників організації, для яких створюється база даних). Об'єднуючи власні уявлення про зміст бази даних, отримані в результаті опитування користувачів, і свої уявлення про дані, що можуть знадобитися в майбутніх додатках, створюється узагальнений неформальний опис утворюваної бази даних. Цей опис, виконаний із використанням природної мови, математичних формул, таблиць, графіків і інших засобів, зрозумілих усім людям, що працюють над проектуванням бази даних, називають інфологічною моделлю даних (рис.2.1) [4,6].

Така модель цілком незалежна від фізичних параметрів середовища збереження даних. Інфологічна модель не повинна змінюватися доти, поки якісь зміни в реальному світі не приведуть до зміни в ній деякого визначення, щоб ця модель продовжувала відображувати предметну область. Інші моделі, показані на рис.1.3, є комп'ютерно-орієнтованими. З їхньою допомогою СКБД дає можливість програмам і користувачам здійснювати доступ до збережених даних лише за їхніми іменами, не турбуючись про фізичне розташування цих даних. Потрібні дані відшукуються СКБД на зовнішніх запам'ятовувальних пристроях по фізичній моделі даних. Оскільки зазначений доступ здійснюється за допомогою конкретної СКБД, то моделі повинні бути описані мовою опису даних цієї СКБД.

Такий опис, утворений по інфологічній моделі даних, називають

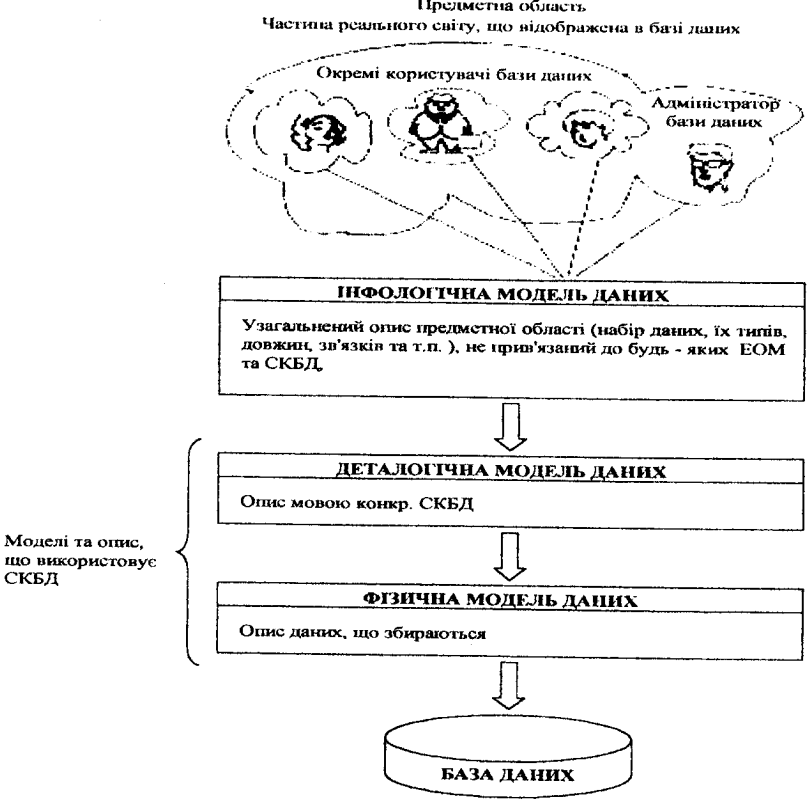


Рисунок 2.1 – Рівні моделей даних

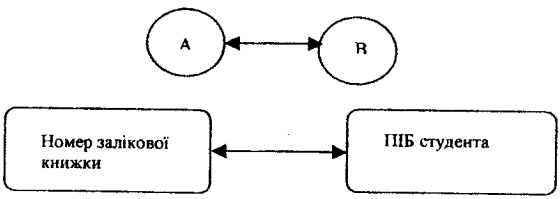


Рисунок 2.2 – Приклад відображення ОДИН-ДО-ОДНОГО

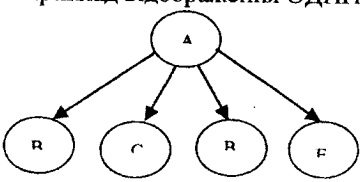


Рисунок 2.3 – Приклад відображення ОДИН-ДО-БАГАТЬОХ

Трирівнева архітектура (інфологічний, даталогічний і физичний рівні) дозволяє забезпечити незалежність збережених даних від програм, в яких вони використовуються. Можна при необхідності переписати збережені дані на інші носії інформації і (або) реорганізувати їхню фізичну структуру. Можна підключити до системи будь-яке число нових користувачів (нових додатків), доповнивши, якщо треба, даталогічну модель. Зазначені зміни фізичної і даталогічної моделей не будуть помічені користувачами системи, так само як не будуть помічені і нові користувачі. Отже, незалежність даних забезпечує можливість розвитку системи баз даних без руйнування існуючих додатків.

Таким чином, інфологічна модель відображає реальний світ у деякій зрозумілі людині концепції, цілком незалежній від параметрів середовища збереження даних. Існує безліч підходів до побудови таких моделей: графові моделі, семантичні мережі, модель "сутність-зв'язок" і т.д. Найбільш популярною серед них виявилася модель "сутність-зв'язок".

Мета інфологічного моделювання - забезпечення найбільш природних для людини засобів збору й подання інформації, що буде зберігатися в створюваній базі даних. Тому інфологічну модель даних намагаються будувати за аналогією з природною мовою (остання не може бути використаною у чистому вигляді через складність комп'ютерної обробки текстів і неоднозначності будь-якої природної мови). Основними конструктивними елементами інфологічних моделей є сутності, зв'язки між ними і їхні властивості (атрибути).

Сутність - будь-який помітний об'єкт (об'єкт, що ми можемо відрізнити від іншого), інформацію про який необхідно зберегти в базі даних. Сутностями можуть бути люди, місця, літаки, рейси, колір і т.д. Слід розрізняти такі поняття, як тип сутності і екземпляр сутності. Поняття типу сутності відноситься до набору однорідних особистостей, предметів, подій або ідей, що виступають як ціле. Екземпляр сутності відноситься до

конкретної речі в наборі. Наприклад, типом сутності може бути МІСТО, а екземпляром - Москва, Київ і т.д.

Атрибут - поіменована характеристика сутності. Його найменування повинно бути унікальним для конкретного типу сутності, але може бути однаковим для різноманітного типу сутностей (наприклад, КОЛІР може бути визначений для багатьох сутностей: СОБАКА, АВТОМОБІЛЬ, ДИМ і т.д.). Атрибути використовуються для визначення того, яка інформація повинна бути зібрана про сутність. Прикладами атрибутів для сутності АВТОМОБІЛЬ є ТИП, МАРКА, НОМЕРНИЙ ЗНАК, КОЛІР і т.д. Тут також існує розходження між типом і екземпляром. Тип атрибута КОЛІР має багато екземплярів або значень: Червоний, Синій, Банановий, Біла ніч і т.д., проте кожному екземпляру сутності присвоюється тільки одне значення атрибута [4].

Абсолютне розходження між типами сутностей і атрибутами відсутнє. Атрибут є таким тільки в зв'язку з типом сутності. У іншому контексті атрибут може виступати як самостійна сутність. Наприклад, для автомобільного заводу КОЛІР - це тільки атрибут продукту виробництва, а для лакофарбової фабрики КОЛІР - тип сутності.

Ключ - мінімальний набір атрибутів, за значеннями яких можна однозначно знайти необхідний екземпляр сутності. Мінімальність означає, що видалення із набору будь-якого атрибута не дозволяє ідентифікувати сутність по тих атрибутах, що залишились [4,6].

Зв'язок - асоціювання двох або більше сутностей. Якби призначенням бази даних було тільки збереження окремих, не пов'язаних між собою даних, то її структура могла б бути дуже простою. Проте одне з основних вимог до організації бази даних - це забезпечення можливості знаходження одних сутностей за значеннями інших, для чого необхідно встановити між ними визначені зв'язки. Оскільки в реальних базах даних нерідко присутні сотні або навіть тисячі сутностей, то теоретично між ними може бути

встановлено більше мільйона зв'язків. Наявність такої множини зв'язків і визначає складність інфологічних моделей.

Контрольні питання

1. Що Ви розумієте під інфологічною моделлю даних?
2. Що таке даталогічна модель даних?
3. Що Ви розумієте під фізичною моделлю даних?
4. Що є метою інфологічного моделювання?
5. Дайте визначення основних конструктивних елементів інфологічної моделі даних (сутність, екземпляр сутності, атрибут, ключ, зв'язок).

2.2 Характеристика зв'язків

Структура даних може бути описана формально. Опис глобальної логічної структури бази даних називається схемою. Схема визначає всі типи елементів даних, які зберігаються в базі даних, а також усі зв'язки між ними. Схема бази даних, як правило, дуже складна. Конкретний користувач або прикладний програміст не повинен знати про схему в цілому. Така необізнаність часто навіть необхідна з точки зору безпеки даних. Програміст або користувач повинен бути інформований тільки про множину даних і зв'язків, які орієнтовані на його конкретну область.

Частина схеми отримала назву підсхеми. По суті, підсхема - це деяка організація файлів прикладного програміста. В функції СКБД входить побудова відповідних підсхем із загальної схеми і передача даних користувачам і системним програмістам. При цьому схема даних повина бути спроектована таким чином, щоб з неї могли бути побудовані всі підсхеми за запитами користувачів або прикладних програм. Ні схема, ні підсхема не визначають методів фізичного зберігання даних.[8,9]

Схеми і підсхеми представляють у вигляді діаграм, на яких зображують типи елементів даних і зв'язки між ними. Розрізняють чотири види зв'язків:

1) необов'язковий зв'язок: існування об'єктів не залежить від зв'язку;
2) можливий зв'язок: існування одного з об'єктів залежить від зв'язку;
3) умовний зв'язок: частковий вид можливого зв'язку, коли задається умова існування (наприклад, зв'язок між об'єктами СТУДЕНТ, СТИПЕНДІЯ можлива при умові відповідної успішності);
4) обов'язковий зв'язок: існування обох об'єктів залежить від зв'язку.

Односторонні зв'язки між парами елементів називаються асоціаціями, а двосторонні - відображеннями.

Між двома сутностями А й В можливі чотири типи зв'язків [5,9]:

- ✓ **перший тип** - зв'язок ОДИН-ДО-ОДНОГО (1:1): за допомогою такого відображения подають такий тип зв'язку, коли в кожний момент часу кожний екземпляр елемента, від якого направлений зв'язок, ідентифікує один і тільки один екземпляр елемента, до якого направлений зв'язок, при цьому ця ідентифікація є унікальною в обох напрямках. Приклад відображения 1:1 приведено на рис.2.2. Якщо відомо значення А , то однозначно визначається і значення В. І навпаки.
- ✓ **другий тип** – зв'язок ОДИН-ДО-БАГАТЬОХ (1:Б): якщо екземпляр елемента даних, від якого направлений зв'язок, ідентифікує деяке число екземплярів елементів даних, до яких направлений зв'язок, причому ідентифікація в даному напрямку не обов'язково є унікальною, то таке відображення називається ОДИН-ДО-БАГАТЬОХ (1:Б). Прикладом такого відношення (рис.2.3) може бути НОМЕР ВІДДІЛУ - ТАБЕЛЬНІ НОМЕРИ ПРАЦІВНИКІВ. У відділі працює багато службовців, але кожний працівник відноситься тільки до одного відділу.

Оскільки між двома сутностями можливі зв'язки в обох направленнях, то існує ще **два типи зв'язків** БАГАТО-ДО-ОДНОГО (Б:1) і БАГАТО-ДО-БАГАТЬОХ (Б:Б). Відображення Б:1 є аналогічним відображенням 1:Б.

Якщо екземпляр елемента даних, від якого направлений зв'язок, ідентифікує деяке число екземплярів елементів даних, до яких направлений зв'язок, і навпаки, тобто ідентифікація не є унікальною в обох напрямках, то таке відображення називається БАГАТО-ДО-БАГАТЬОХ (Б:Б).

Прикладом такого відношення (рис.2.4) є відношення ВИКЛАДАЧІ-СТУДЕНТИ. Кожний студент "пов'язаний" з багатьма викладачами і кожний викладач читає лекції різним групам студентів.

Характер зв'язків між сутностями не обмежується переліченими. Існують і більш складні зв'язки:

- множина зв'язків між одними й тими ж сутностями;

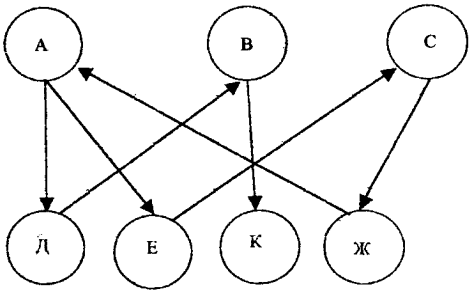


Рисунок 2.4 - Приклад відображення БАГАТО-ДО-БАГАТЬОХ

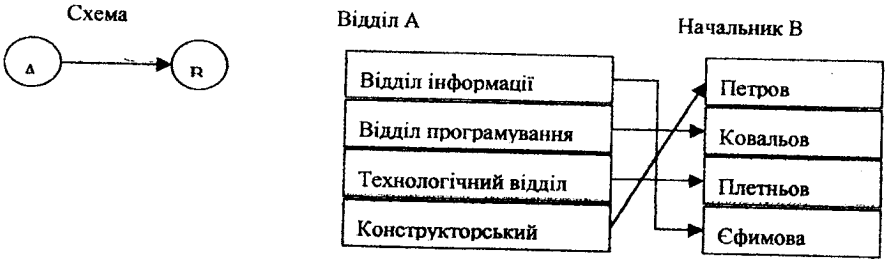


Рисунок 2.5 - Приклад простої асоціації типу 1

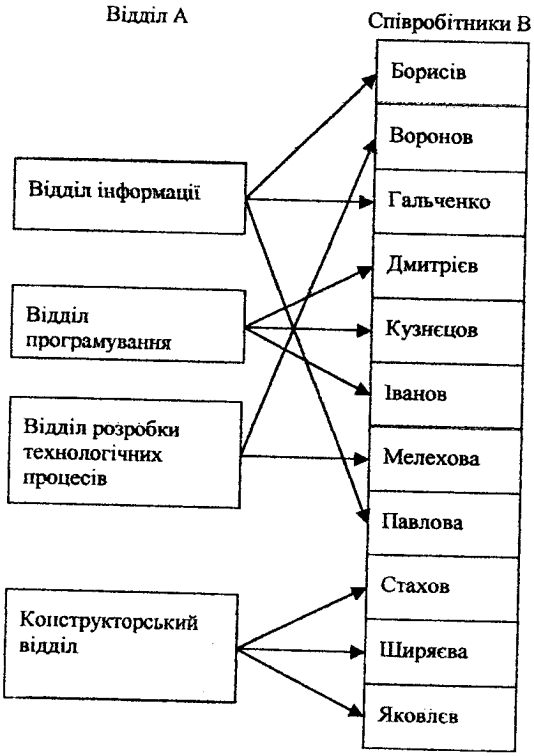


Рисунок 2.6 - Приклад складної асоціації типу M

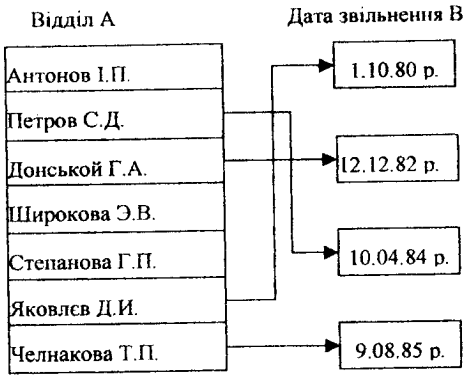


Рисунок 2.7 - Приклад умовної асоціації типу С

Наприклад, пацієнт, маючи одного лікаря, що лікує, може мати також декілька лікарів-консультантів; лікар може бути лікарем, що лікує декількох пацієнтів та може одночасно консультувати декількох інших пацієнтів.

- тренарні зв'язки;

Наприклад, лікар може призначити декільком пацієнтам декілька аналізів, аналіз може бути призначений декількома лікарями декільком пацієнтам, й пацієнту може бути призначено декілька аналізів декількома лікарями).

- зв'язки більш високих порядків, семантика (зміст) яких іноді дуже складна.

Існує три типа асоціацій [9]:

– асоціація типу 1 (проста);
– асоціація типу М (складна);
– асоціація типу С (умовна).

В простій асоціації типу 1 (рис.2.5) екземпляр елемента даних, від якого направлено зв'язок, ідентифікує один і лише один екземпляр елемента даних, до якого направлено зв'язок. Ця ідентифікація є унікальною й визначає функціональну залежність.

В складній асоціації типу М (рис.2.6) екземпляр елемента даних, від якого направлено зв'язок, ідентифікує деяке число екземплярів елементів даних, до яких направлено зв'язок. Ідентифікація є багатозначною залежністю і не обов'язково унікальною. При цьому зв'язок в зворотному напрямку не розглядається.

В умовній асоціації типу С (рис.2.7) для даного екземпляра елемента даних, від якого спрямований зв'язок, може не існувати відповідного екземпляра елемента даних, до якого спрямований зв'язок. Якщо він існує, то відноситься до єдиного екземпляра елемента даних. Наприклад, ПІБ РОБІТНИКА і ДАТА ЗВІЛЬНЕННЯ.

У реальних базах даних існує велика кількість типів елементів даних. Для зменшення кількості зв'язків елементи об'єднують у групи. Таке групування значно зменшує кількість записів. Об'єднання елементів у групи повинно бути аргументованим і продуманим.

Для підвищення ілюстративності аналізованих зв'язків застосовується мова інфологічного моделювання (МІМ), у якій сутності й асоціації подають пропозиціями виду:

СУТНІСТЬ (атрибут 1, атрибут 2 , ... , атрибут n)

АСОЦІАЦІЯ[СУТНІСТЬ S1, СУТНІСТЬ S2, ...](атрибут 1, атрибут2, ... , атрибут n)

де *S* - ступінь зв'язку, а атрибути, що входять у ключ, повинні бути відзначені підкресленням.

Для виявлення зв'язків між сутностями необхідно, як мінімум, визначити самі сутності. Але це не проста задача, тому що в різних предметних областях один і той же об'єкт може бути сутністю, атрибутом або асоціацією [10,11].

Контрольні питання

1. Що Ви розумієте під схемою бази даних?

2. Що таке підсхема бази даних?
3. Перелічіть основні типи зв'язків між елементами даних.
4. Які види зв'язків між сутностями Вам відомі?
5. Визначте основні типи асоціацій.

2.3 Класифікація сутностей

К.Дейт визначає три основні класи сутностей: стрижневі, асоціативні і характеристичні, а також підклас асоціативних сутностей – позначення[3].

Стрижнева сутність (стрижень) - це незалежна сутність.

Асоціативна сутність (*асоціація*) - це зв'язок типу Б:Б між двома або більше сутностями або екземплярами сутності. Асоціації розглядаються як повноправні сутності:

- вони можуть брати участь у других асоціаціях і позначеннях так само, як стрижневі сутності;
- можуть мати властивості, тобто мати не тільки набір ключових атрибутів, необхідних для вказівки зв'язків, але і будь-яке число інших атрибутів, що характеризують зв'язок.

Характеристична сутність (*характеристика*) - це зв'язок типу Б:1 або 1:1 між двома сутностями (окремий випадок асоціації). Єдиною метою характеристики в рамках аналізованої предметної області є опис або уточнення деякої іншої сутності. Необхідність у них виникає в зв'язку з тим, що сутності реального світу мають іноді багатозначні властивості. Наприклад, книга може мати декілька характеристик перевидання (доповнене, перероблене, ...) і т.д. Існування характеристики цілком залежить від сутності, що характеризується. Для опису характеристики використовується нова пропозиція МІМ, що має в загальному випадку вигляд:

ХАРАКТЕРИСТИКА (атрибут 1, атрибут 2, ...) {СПИСОК СУТНОСТЕЙ, ЩО ХАРАКТЕРИЗУЮТЬСЯ }.

Позначення - це зв'язок типу Б:1 або 1:1 між двома сутностями і відрізняється від характеристики тим, що не залежить від сутності, яку він позначає.

Розглянемо приклад, пов'язаний із зарахуванням співробітників у різні відділи організації. При відсутності жорстких правил (співробітник може одночасно зараховуватися в декілька відділів або не зараховуватися ні в один відділ) необхідно створити опис з асоціацією ЗАРАХУВАННЯ:

Відділи (Номер відділу, Назва відділу, ...)

Службовці (Табельний номер, Прізвище, ...)

Зарахування [Відділи M, Службовці N](Номер відділу, Табельний номер, Дата зарахування).

Проте, за умови, що кожний із співробітників повинний бути обов'язково зарахований в один із відділів, можна створити опис із позначенням СЛУЖБОВЦІ:

Відділи (Номер відділу, Назва відділу, ...)

Службовці (Табельний номер, Прізвище, ... , Номер відділу, Дата зарахування) [Відділи]

У даному прикладі службовці мають незалежне існування (якщо ліквідується відділ, то з цього не випливає, що також повинні бути скорочені службовці такого відділу). Тому вони не можуть бути характеристиками відділів і названі позначеннями. Позначення використовують для збереження повторюваних значень великих текстових атрибутів: "кодифікатори" досліджуваних студентами дисциплін, найменувань організацій і їхніх відділів, переліків товарів і т.п. Опис позначення зовнішньо відрізняється від опису характеристики тільки тим, що сутності, які позначаються, беруться не у фігурні дужки, а в квадратні:

ПОЗНАЧЕННЯ (атрибут 1, атрибут 2, ...)[СПИСОК СУТНОСТЕЙ, ЩО ПОЗНАЧАЮТЬСЯ].

Як правило, позначення не розглядаються як повноправні сутності, хоча це не призвело б до помилки.

Позначення і характеристики не є цілком незалежними сутностями, оскільки вони припускають наявність деякої іншої сутності, що буде "позначатися" або "характеризуватися". Проте вони все ж являють собою окремі випадки сутності і можуть, звичайно, мати властивості, можуть брати участь в асоціаціях, позначеннях і мати свої власні (більш низького рівня) характеристики. Підкреслимо також, що всі екземпляри характеристики повинні бути обов'язково пов'язані з яким-небудь екземпляром сутності, що характеризується. Проте припускається, що деякі екземпляри сутності, що характеризуються, не мали зв'язків. Перевизначимо тепер стрижневу сутність як сутність, що не є ні асоціацією, ні позначенням, ні характеристикою. Такі сутності мають незалежне існування, хоча вони і можуть позначати інші сутності, як, наприклад, співробітники позначають відділи.

Контрольні питання

1. Дайте визначення стрижневої сутності.
2. Що таке асоціативна сутність?
3. Що Ви розумієте під характеристичною сутністю?
4. Як використовують МІМ для опису характеристичної сутності?
5. Дайте визначення позначення.

2.4 Аналіз предметної області

Першим етапом проектування бази даних будь-якого типу є аналіз предметної області, що закінчується побудовою інформаційної структури (концептуальної схеми). На даному етапі аналізуються запити користувачів, вибираються інформаційні об'єкти та їх характеристики і на основі проведеного аналізу формується структура предметної області, яка не залежить від програмного та технічного середовища, в якому буде реалізовуватися база даних. Аналіз предметної області доцільно розбити на три фази [12-14]:

- аналіз концептуальних вимог та інформаційних потреб;
- виявлення інформаційних об'єктів та зв'язків між ними;
- побудова концептуальної моделі предметної області та проектування концептуальної схеми бази даних.

На етапі аналізу концептуальних вимог та інформаційних потреб необхідно вирішити такі задачі:

- аналіз вимог користувача до бази даних (концептуальних вимог);
- виявлення задач, що мають місце, при обробці інформації, яка повинна бути представлена у базі даних (аналіз додатків);
- виявлення перспективних задач (перспективних додатків);
- документування результатів аналізу.

Вимогами користувачів до бази даних, що розробляється є, в загальному випадку, список запитів з вказанням їх інтенсивності та об'ємів даних. Ці вказівки опрацьовуються в діалозі з майбутнім користувачем бази даних. Тут же з'ясовуються вимоги до вводу, відновлення та корегування інформації. Вимоги користувачів уточнюються та доповнюються при аналізі перспективних додатків, що мають місце.

Так, виходячи із специфіки діяльності читального залу, необхідно забезпечити облік книг, що є в наявності, виконувати швидкий пошук

творів, що входять до складу тих чи інших книг. Крім того читальному залу потрібна картотека користувачів, що дозволяло б оперативно здійснювати з ними зв'язок, а також для виявлення користувачів, які порушили строки повернення книги, повинна бути картотека творів, що є в наявності в читальному залі.Тоді в базу даних доцільно включити інформацію про: книги, що є в читальному залі; твори, що входять до складу тих чи інших книг; користувачів, що користуються послугами читального залу. При цьому, розроблювана база даних повинна забезпечити такі функції:

1.Ведення картотеки користувачів.

В інформацію про користувачів доцільно включити такі дані:

- прізвище, ім'я та по батькові;
- адреса;
- номер телефону;
- паспортні дані;
- освіта;
- професія.

2. Облік книг, що є в читальному залі чи якими в даний момент користуються повинен враховувати такі дані:

- назва книги;
- автор книги;
- рік видання;
- видавництво;
- кількість сторінок;
- предметна область.

3. Ведення картотеки творів, які містяться в даній книзі, включає таку інформацію:

- назва;
- автор;

- дата написання.

4. Виявлення боржників, що не повернули книгу протягом дня.
5. Формування даних про повернення книги.

Контрольні питання

1. Охарактеризуйте аналіз предметної області як перший етап проектування бази даних.
2. Назвіть три фази аналізу предметної області.
3. Перелічіть задачі етапу аналізу концептуальних вимог та інформаційних потреб.
4. Назвіть задачі етапу виявлення інформаційних об'єктів та зв'язків між ними.

2.5 Розробка універсального відношення

Друга фаза аналізу предметної області складається з вибору інформаційних об'єктів, задання необхідних властивостей для кожного об'єкта, виявлення зв'язків між об'єктами, виявлення обмежень, що накладаються на інформаційні об'єкти, типи зв'язків між ними, характеристики інформаційних об'єктів.

При виборі інформаційних об'єктів бажано намагатися відповісти на такі питання [15]:

1. На які класи можна розбити дані, що підлягають збереженню у базі даних?
2. Яке ім'я можна присвоїти кожному класу даних?
3. Які найбільш цікаві характеристики (з точки зору користувача) кожного класу даних можно виділити?
4. Які імена можна присвоїти вибраним наборам характеристик?

Виділення інформаційних об'єктів - процес ітеративний. Він здійснюється на основі аналізу інформаційних потоків та інтерв'ювання споживачів. Характеристики інформаційних об'єктів визначаються тими ж методами.

Введемо ряд позначень, які будуть використовуватися у ході подальшого викладення матеріалу.

R - є відношення над множинами *D1, D2,...Dn*, якщо воно є множиною упорядкованих *n*-кортежів вигляду *<d1,d2,...dn>*. *D1,D2,...Dn* – називаються доменами відношення *R*.

Відношення може бути подане у вигляді файла або таблиці, стовпцями яких є елементи доменів, а рядками - кортежі. Кожен кортеж відображає один екземпляр інформаційного об'єкта. Імена стовпців (поле запису) – називаються атрибутами, а індивідуальні значення елементів - значеннями атрибутів. Кожен атрибут відображає відповідну характеристику інформаційного об'єкта. Число стовпців у відношенні називається ступенем відношення, а число кортежів - потужністю відношення. У процесі експлуатації бази даних ступінь відношення змінюється значно рідше, ніж його потужність.

Атрибут, або набір атрибутів, який можна використати для однозначності ідентифікації конкретного кортежа, називається первинним ключем (у випадку набору атрибутів - складений ключ).

Можливі випадки, коли відношення може вміщувати декілька унікальних ключів. Тоді один з них вибирається в якості первинного, а інші отримують назву можливих ключів.

Атрибути, що представляють копії ключів інших відношень, називаються зовнішніми ключами.

Атрибут, або набір атрибутів, що використовується для більш швидкого пошуку називається другорядним індексом.

Універсальним називається відношення, що вміщує в себе всі атрибути, які будуть використовуватися в базі даних. Для невеликих баз даних універсальне відношення може служити відправною точкою при їх проектуванні.

Розглянемо порядок роботи універсального відношення при створенні бази даних для читального залу.

Враховуючи аналіз предметної області, в універсальне відношення потрібно включити атрибути, що описують такі інформаційні об'єкти:
КНИГА, ТВІР, КОРИСТУВАЧ, РОЗДІЛ.

Складемо перелік найбільш суттєвих характеристик кожного інформаційного об'єкта.

КНИГА (шифр книги, рік видання; видавництво, що надрукувало книгу; кількість сторінок, що включає книга).

ТВІР (назва твору, що входить до складу книги; автор твору; дата створення).

КОРИСТУВАЧ (код користувача, що служить для його швидкої ідентифікації; прізвище, ім'я та по батькові користувача; домашня адреса; номер телефону, домашнього або службового; паспортні дані; освіта; професія).

РОЗДІЛ (код предметної області, назва предметної області).

Перерахунок вибраних для універсального відношення атрибутів приведено в таблиці 2.1.

Оскільки всі перераховані в таблиці атрибути є незалежними, тобто значення одних з них не можуть бути обчислені за значеннями інших, то всі вони можуть бути включені в склад універсального відношення, яке при цьому приймає вигляд:

R (Cod_book , Name, Author, Cod_rozdil, Name_rozdil, Data_write, Publisher, Year_publish, Page, Cod_client, FIO, Address, Telephone, №_pasp, Culture, Profession).

Контрольні питання

1. Визначте принципи вибору інформаційних об'єктів.
2. Дайте визначення понять: домен, елемент домену, кортеж відношення.
3. У чому полягає різниця між ступенем відношення та його потужністю?
4. Що таке універсальне відношення?
5. Наведіть приклад роботи універсального відношення при створенні бази даних.

2.6 Розробка *ER*–моделі предметної області

Заключна фаза аналізу предметної області складається з розробки її інформаційної структури (або концептуальної схеми).

В звичайних випадках для побудови концептуальної схеми використовуються традиційні методи агрегації та узагальнення. При агрегації декілька інформаційних об'єктів (елементів даних) об'єднується в один у відповідності з семантичними зв'язками між об'єктами. При побудові інфологічних моделей можна використовувати мову *ER*-діаграм (від англ. Entity-Relationship, "сутність-зв'язок"). В них сутності зображуються позначеними прямокутниками, асоціації - позначеними ромбами або шестикутниками, атрибути - позначеними овалами, а зв'язки між ними - ненаправленими ребрами, над якими може проставлятися ступінь зв'язку (1 або буква, що заміняє слово "БАГАТО") і необхідне пояснення.

Розглянемо деякі риси моделі "сутність-зв'язок" (*ER*-моделі). [16]

На використанні різновидів *ER*-моделі реалізується більшість сучасних підходів до проектування баз даних. Модель була запропонована Ченом (Chen) у 1976 р. Моделювання предметної області базується на використанні графічних діаграм, що включають невелике число різнорідних компонентів. У зв'язку з наочністю уявлення концептуальних

схем баз даних *ER*-моделі знайшли широке застосування в системах CASE, що підтримують автоматизоване проектування реляційних баз даних.

Таблиця 2.1 - **Початковий перелік атрибутів для формування Універсального відношення бази даних читального залу**

Шифр книги	Cod_book	Кожна книга має унікальний шифр.
Код розділу	Cod_rozdil	Кожен предметний розділ має унікальний шифр
Назва розділу	Name_rozdil	Назва предметного розділу
Назва твору	Name	Кожен твір має назву
Автор	Author	Письменник, що написав твір.
Дата створення	Data_write	Дата, коли письменник закінчив писати твір.
Дата видання	Year_publish	Дата, коли книгу надрукували.
Видавництво	Publisher	Видавництво, що надрукувало книгу.
Кількість сторінок	Page	Кількість сторінок, що займає книга.
Код користувача	Cod_client	Код користувача, що займає книгу.
Користувач	FIO	Прізвище, ім'я та по батькові.
Адреса	Address	Адреса користувача.
Телефон	Telephone	Номер телефона користувача.
Паспорт	№_pasp	Паспортні дані користувача.
Освіта	Culture	Освіта, яку має користувач.
Професія	Profession	Професія користувача.

Головними поняттями *ER*-моделі є сутність, зв'язок і атрибут. У діаграмах *ER*-моделі сутність подається у вигляді прямокутника, що містить ім'я сутності. При цьому ім'я сутності - це ім'я типу, а не деякого конкретного екземпляра цього типу. Для кращого розуміння ім'я сутності може супроводжуватися прикладами конкретних об'єктів цього типу. Сутність АЕРОПОРТ із об'єктами Шереметьєво і Хітроу зображена на рис.2.8.

Кожний екземпляр сутності повинен відрізнятися від будь-якого іншого екземпляра тієї ж сутності (ця вимога до повної міри аналогічна вимозі відсутності кортежів-дублікатів у реляційних таблицях).

Зв'язок - це асоціація, що графічно зображується та установлюється між двома сутностями. Ця асоціація завжди є бінарною і може існувати між двома різними сутностями або між сутністю і нею ж самою (рекурсивний зв'язок). У будь-якому зв'язку виділяються два кінці (відповідно до існуючої пари сутностей, що зв'язуються), на кожному з яких вказується ім'я кінця зв'язку, ступінь кінця зв'язку (скільки екземплярів даної сутності зв'язується), обов'язковість зв'язку (тобто чи будь-який екземпляр даної сутності повинен брати участь у даному зв'язку). Зв'язок подається у вигляді лінії, що зв'язує дві сутності або веде від сутності до неї ж самої. Про це в місці "стикування" зв'язку із сутністю вказує триточковий вхід у прямокутник сутності, якщо для цієї сутності в зв'язку можуть використовуватися багато (many) екземплярів сутності, і одноточковий вхід, якщо в зв'язку може брати участь тільки один екземпляр сутності. Обов'язковий кінець зв'язку зображується суцільною лінією, а необов'язковий - переривчастою лінією.

Розглянемо приклад зв'язку між сутностями КВИТОК і ПАСАЖИР (рис.2.9). При цьому кінець сутності з ім'ям "для" дозволяє зв'язувати з одним пасажиром більше одного квитка, причому кожний квиток повинен бути пов'язаний із яким-небудь пасажиром. Кінець сутності з ім'ям "має"

означає, що кожний квиток може належати тільки одному пасажиру причому пасажир зобов'язаний мати не менше одного квитка.

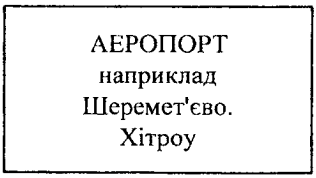
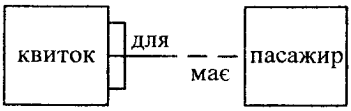


Рисунок 2.8 – Зображення сутності АЕРОПОРТ із об'єктами



Шереметьєво і Хітроу

Рисунок 2.9 - Зображення зв'язку між сутностями КВИТОК та ПАСАЖИР

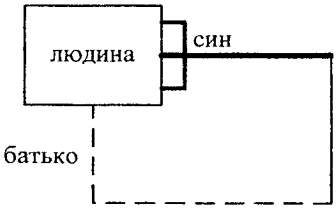


Рисунок 2.10 - Зображення рекурсивного зв'язку

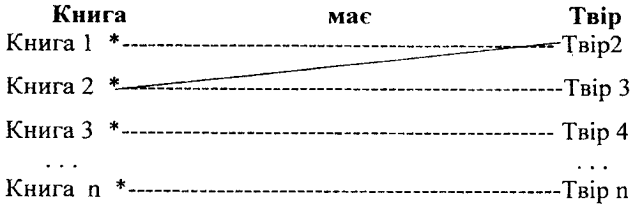


Рисунок 2.11 – Зображення зв'язку КНИГА – ТВІР

Трактування зображеної діаграми (рис.2.9) таке:

- Кожний КВИТОК призначений для одного і тільки одного ПАСАЖИРА;
- Кожний ПАСАЖИР може мати один або більше КВИТКІВ.

На рис.2.10 приведено приклад зображення рекурсивного зв'язку, що зв'язує сутність ЛЮДИНА з нею ж самою.

Кінець зв'язку з ім'ям СИН визначає той факт, що в одного батька може бути більше ніж один син. Кінець зв'язку з ім'ям БАТЬКО означає, що не в кожної людини можуть бути СИНИ.

Трактування зображеної діаграми (рис.2.10) таке:

- Кожна ЛЮДИНА є сином одного і тільки одного ЧОЛОВІКА;
- Кожна ЛЮДИНА може бути батьком для одного або більше ЛЮДЕЙ.

Атрибутом сутності є будь-яка деталь, що служить для уточнення, ідентифікації, класифікації, числової характеристики або вираження стану сутності. Імена атрибутів заносяться в прямокутник, що зображує сутність, під ім'ям сутності і зображуються малими буквами, можливо, із прикладами. Наприклад, унікальним ідентифікатором сутності є атрибут, комбінація атрибутів, комбінація зв'язків або комбінація зв'язків і атрибутів, що унікально відрізняє будь-який екземпляр сутності від інших екземплярів сутності того ж типу.

Таким чином, зв'язки можуть представлятися різними способами, з яких ми будем використовувати тільки два: діаграма *ER*-екземплярів і діаграма *ER*-типів. Діаграми *ER*-екземплярів відображають зв'язки між екземплярами сутностей. Діаграми *ER*-типів відображають зв'язки між типами сутностей.

Отже, першою задачою, яку необхідно розв'язати при розробці *ER*-моделі, є формування сутностей, що необхідні для описання предметної області. Іншими словами, необхідно вказати ті типи об'єктів (тобто набори подібних об'єктів), про які в системі повинна накопичуватися інформація.

Це означає, що за підсумками аналізу предметної області потрібно мати повну або достатньо повну уяву про реалізовані в системі запити. Разом з тим необхідно врахувати, що при кваліфікованій експлуатації системи у більшості випадків у користувача виникає бажання розширити систему запитів. У зв'язку з цим, при розробці *ER*-схеми, з одного боку зручно твердо прив'язатися до обраної в результаті аналізу предметної області системи запитів, а з іншого боку, бажано розглядати задачу в більш широкому плані, з врахуванням перспектив подальшого нарощування можливостей системи. Приведемо приклад для таких об'єктів: КНИГА, ТВІР, КОРИСТУВАЧ, РОЗДІЛ. Для кожної сутності необхідно обрати атрибут, що її однозначно ідентифікує або сукупність атрибутів, які будуть виступати ключем відповідного відношення. Необхідно врахувати, що ключ повинен не тільки виконувати задачу ідентифікації, але і по можливості, включати в свій склад мінімальне число атрибутів. У зв'язку з цим в процесі проектування вибрані в якості ключів атрибути можуть переглядатися. Наприклад, виберемо в якості ключових такі атрибути:

КНИГА --> Cod_book (шифр книги);

ТВІР --> Cod_book (шифр книги), Name_tvir (назва твору);

РОЗДІЛ --> Cod_rozdil (код предметної області).

Необхідність введення складеного ключа для сутності ТВІР обумовлено тим, що в читальному залі може бути кілька однакових творів, написаних одним автором, але входять вони до складу різних книг. Тепер виявимо залежності, що існують між визначеними нами сутностями, а також визначимо характеристики кожного зв'язку. Зв'язок КНИГА – ТВІР приведено на рис.2.11.

З рис.2.11 видно, що клас належності обох сутностей є обов'язковим, оскільки книги та твори, яких немає в читальному залі не повинні заноситись в базу даних.

Тип зв'язку – Б:Б: декілька творів можуть входити до складу однієї книги, та один і той самий твір може знаходитись в декількох книгах. Зв'язок КНИГА – РОЗДІЛ приведено на рис.2.12.

З рис.2.12 видно, що клас належності обох сутностей є обов'язковим, оскільки книги, яких немає в читальному залі не повинні заноситись в базу даних, та розділи з предметних областей, що не мають книг, не заносять до бази даних.

Тип зв'язку – Б:1: декілька творів можуть входити до складу одного предметного розділу, але одна книга не може міститися в більше ніж одному розділі.

У реальному житті часто зустрічаються ситуації, які неможливо описати бінарними зв'язками. Тобто, в цих випадках зв'язок об'єднує одночасно не дві, а більше сутностей, як правило - три. *ER*–діаграма, що ілюструє тристоронні зв'язки РОЗДІЛ–КНИГА–ТВІР приведена на рис 2.13). Розмірковуючи аналогічним чином побудуємо *ER*-модель предметної області "Читальний зал". В якості сутностей оберемо такі об'єкти предметної області (з перерахуванням ключових атрибутів кожної сутності) :

КНИГА →<Cod_book(шифр книги)>,Cod_client,Publisher,Year_publish, Page.

ТВІР → <Cod_book (шифр книги), Name_tvir (назва твору)>, Author, Data_write;

РОЗДІЛ → Cod_rozdil (код предметної області), Name_rozdil.

КОРИСТУВАЧ → Cod_client (шифр користувача), FIO, Address, Telephone, №_pasp, Culture, Profession.

Характеристики зв'язків виділених сутностей приведені в табл.2.2, а *ER*-модель предметної області "Читальний зал", що побудована на їх основі, показана на рис.2.14.

До числа більш складних елементів моделі відносяться такі:

- Підтипи і супертипи сутностей. Як у мовах програмування з розвинутими типовими системами (наприклад, у мовах об'єктно-орієнто-

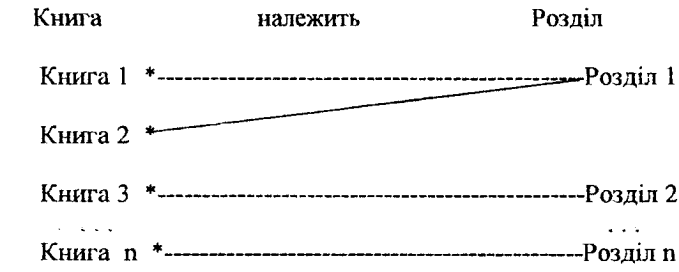


Рисунок 2.12 - Зображення зв'язку КНИГА – РОЗДІЛ

Таблиця 2.2 - **Характеристики зв'язків предметної області "Читальний зал"**

Ім'я Сутності 1	Ім'я сутності 2	Ім'я зв'язку	Тип зв'язку	Клас належності
Книга	Розділ	Належить	N : 1	Обов'язк., Обов'язк.
Книга	Твір	Має	N : M	Обов'язк., Обов'язк.
Книга	Користувач	Читає	N : 1	Необов'язк., Необов'язк.

ваного програмування), вводиться можливість спадкування типу сутності, виходячи з одного або декількох супертипів.

- Зв'язки "many-to-many". Іноді буває необхідно зв'язувати сутності таким чином, що з обох кінців зв'язку можуть бути присутніми декілька екземплярів сутності (наприклад, усі члени кооперативу

6. Назвіть основні етапи розробки *ER*-моделі.
7. Які Вам відомі складні елементи моделі?

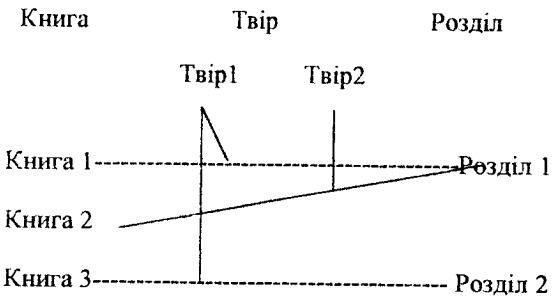


Рисунок 2.13 - *ER*–діаграма, що ілюструє тристоронні зв'язки (РОЗДІЛ – КНИГА – ТВІР)

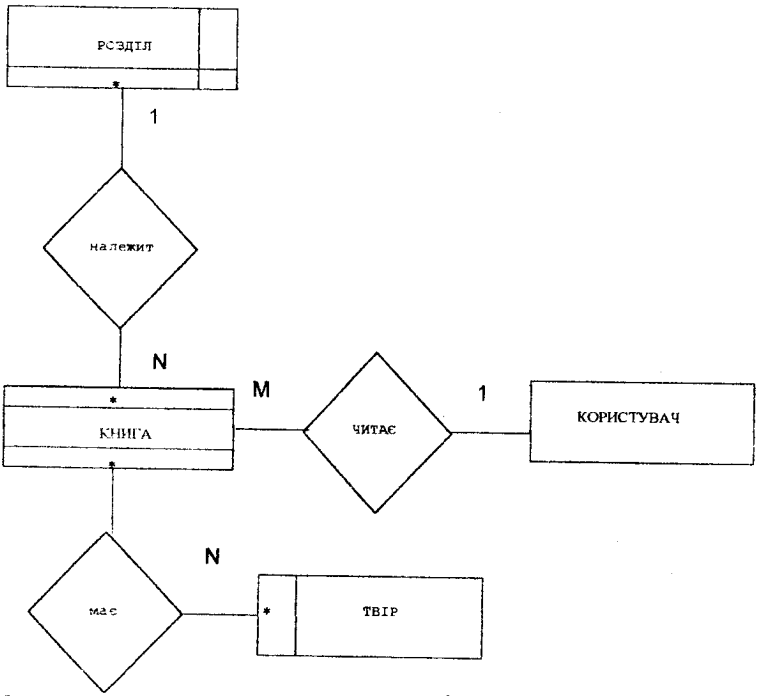


Рисунок 2.14 - *ER*-модель предметної області "Читальний зал"

спільно володіють майном кооперативу). Для цього вводиться різновид зв'язку БАГАТО-З-БАГАТЬМА.

- Ступені зв'язку, що уточнюються. Іноді буває корисно визначити можливу кількість екземплярів сутності, що беруть участь у даному зв'язку (наприклад, службовцю дозволяється брати участь не більш, ніж у трьох проектах одночасно). Для вираження цього семантичного обмеження дозволяється вказувати на кінці зв'язку її максимальний або обов'язковий ступінь.
- Каскадні видалення екземплярів сутностей. Деякі зв'язки бувають настільки сильними (звичайно, у випадку зв'язку "ОДИН-ДО-БАГАТЬОХ"), що при видаленні опорного екземпляра сутності (відповідає кінцю зв'язку "ОДИН") потрібно видалити і всі екземпляри сутності, що відповідають кінцю зв'язку "БАГАТО". Відповідну вимогу "каскадного видалення" можна сформулювати при визначенні сутності.
- Домени. Як і у випадку реляційної моделі даних буває корисна можливість визначення потенційно припустимої множини значень атрибута сутності (домена).

Ці й інші більш складні елементи моделі даних "сутність-зв'язок" роблять її більш потужною, але одночасно в певній мірі ускладнюють її використання. Звичайно, при реальному використанні *ER*-діаграм для проектування баз даних необхідно ознайомитися з усіма можливостями.

Контрольні питання

1. Як використовують мову *ER*-діаграм при побудові інфологічних моделей?
2. Як використовують *ER*-діаграми при моделюванні предметної області?
3. Визначте головні поняття *ER*-моделі (сутність, зв'язок, атрибут).
4. Наведіть приклад рекурсивного зв'язку сутностей.
5. Як Ви розумієте поняття "атрибут сутності"?

3 МОДЕЛІ ДАНИХ

Відомі три основних типи моделей даних: ієрархічна, мережна та реляційна. Перші дві з них використовують графові моделі для представлення інформації про об'єкти [16].

3.1 Ієрархічна модель даних

Деревоподібна (ієрархічна) структура (рис.3.1), або дерево - це зв'язний неорієнтований граф, що не містить циклів, тобто петель з замкнутих шляхів.

Як правило, при роботі з деревом виділяють будь-яку конкретну верхівку (початок) та визначають її як *коріння* дерева. В цю верхівку не заходить жодне ребро. В цьому випадку дерево стає орієнтованим. Орієнтація на кореневому дереві визначається або від коріння, або до коріння.

Кореневе дерево можна визначити таким чином:

1) є єдиний особливий вузол, який називається корінням, в який не заходить жодне ребро;
2) в усі інші вузли заходить тільки одне ребро, а виходить довільна (0, 1, 2,..., *n)* кількість ребер;
3) не існує циклів.

В програмуванні використовується інше визначення дерева, яке дозволяє розглядати дерево як рекурсивну структуру.

Рекурсивне дерево визначається як кінцева множина T, яка складається з одного або більше вузлів, таких, що:

1) існує один спеціально виділений вузол, який називається корінням дерева;
2) інші вузли розбиті на m>0 підмножин $T_1, T_2, \ldots, T_m$, що не перетинаються, кожна з яких в свою чергу є деревом. $T_1, T_2, \ldots, T_m$,

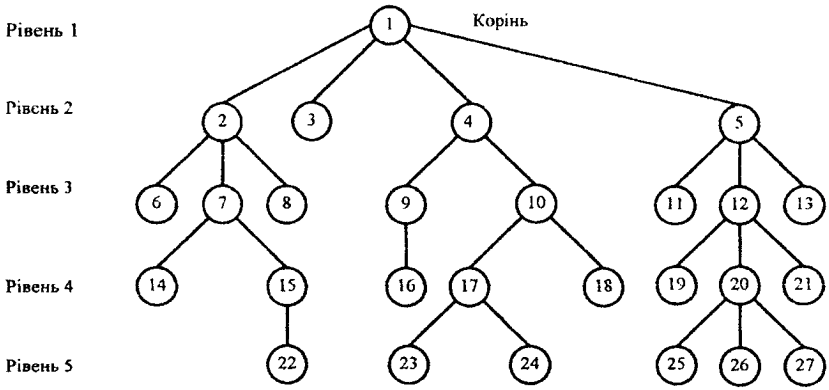
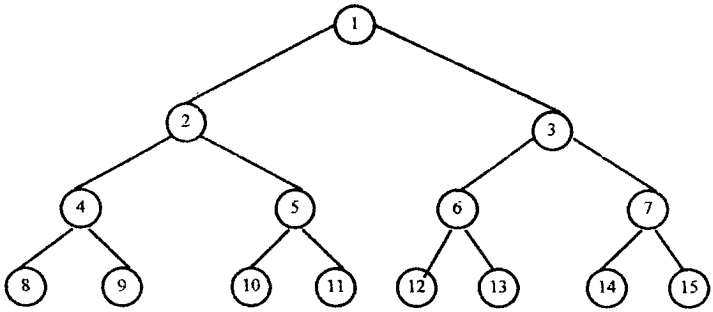
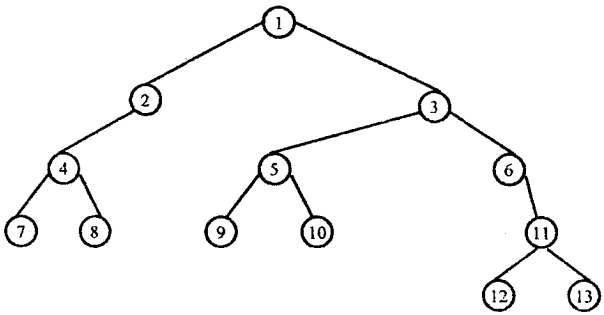


Рисунок 3.1 – Ієрархічна структура даних



а)



б)

Рисунок 3.2 – Приклади дерев

називаються піддеревами.

З визначення випливає, що будь-який вузол дерева є корінням деякого піддерева, що міститься в повному дереві. Число піддерев вузла називають ступенем вузла. Вузол називається кінцевим, якщо він має нульову ступінь. Інколи кінцеві вузли називають листками, а ребра - гілками. Кожний вузол, крім кореневого, зв'язаний з одним вузлом на більш високому рівні ієрархії і називається вихідним. Кожний вузол може бути зв'язаний з одним або декількома вузлами на більш низькому рівні і називається породженим.

Якщо кожний вузол має однакову кількість гілок, причому процес включення нових гілок іде зверху вниз, а на кожному рівні дерева - зліва направо, то таке дерево називається збалансованим (рис. 3.2,а). Для збалансованих дерев фізична організація даних суттєво спрощується. До особливої категорії дерев відносять двійкове (бінарне) дерево. Це дерево має не більше як дві гілки, які виходять з одного вузла. Двійкові дерева можуть бути як збалансованими, так і незбалансованими (рис. 3.2,б).

Прикладом простого ієрархічного подання може служити адміністративна структура вищого учбового закладу (рис.3.3.): університет - відділення – факультет – група (студентська).

Пошук даних у ієрархічній структурі виконується завжди по одній із гілок, починаючи з кореневого елемента, тобто необхідно зазначити повний шлях руху по гілках. Так, для пошуку і вибірки одного або декількох екземплярів запису типу СТУДЕНТ (див. рис.3.4) необхідно вказати кореневий елемент ФАКУЛЬТЕТ і елементи КУРС, ГРУПА. В операційній системі MS-DOS для пошуку файла використовується такий же принцип - вказуються послідовно ім'я диска, ім'я каталогу, ім'я підкаталогів, ім'я файла.

На рис.3.5 приведений приклад типу набору, представленого у вигляді діаграми Бахмана. Діаграму назвали за іменем вченого, який вперше їх застосував для опису відношень між даними при розробці

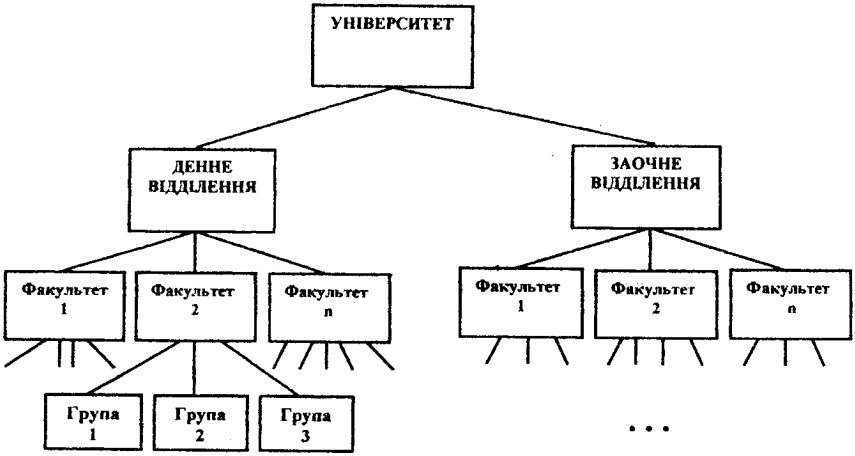


Рисунок 3.3 – Ієрархічне подання адміністративної структури вищого учбового закладу

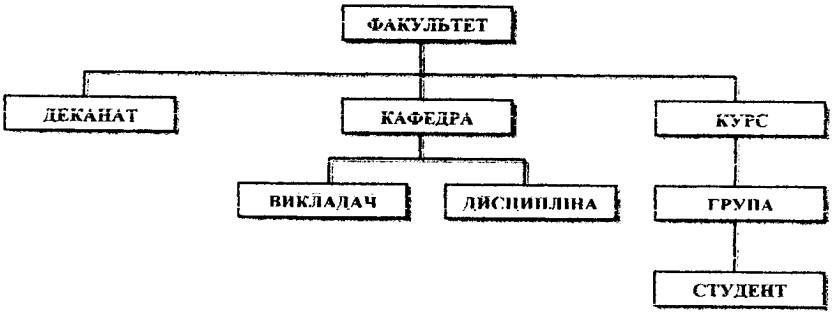


Рисунок 3.4 – Приклад ієрархічної структури

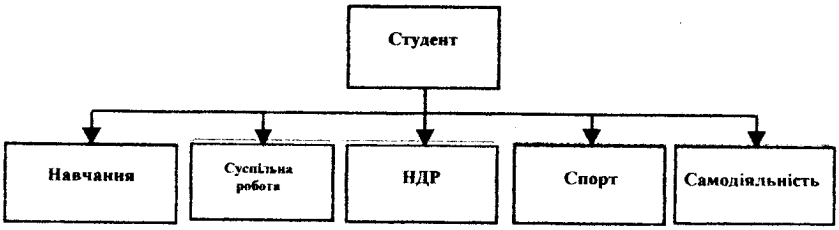


Рисунок 3.5 – Приклад типу набору у вигляді діаграми Бахмана

СКБД IDS.

На такій діаграмі кожний прямокутник представляє собою тип запису, а стрілка - відношення «один до багатьох» між типами запису. У прикладі на рис.3.5 тип запису СТУДЕНТ є записом-власником, а типи записів НАВЧАННЯ, СУСПІЛЬНА РОБОТА, НДР, СПОРТ, САМОДІЯЛЬНІСТЬ - записами-членами. Тип набору названий ім'ям СНСНСС по перших літерах імен усіх типів запису, що беруть участь у наборі (наборові можна було надати і будь-яке інше ім'я). У цілому приведений тип набору призначений для того, щоб відобразити зв'язок між загальними даними про студента, що знаходяться в типі запису СТУДЕНТ, і даними, що характеризують різні сторони діяльності студента у вузі. При традиційному підході всі ці данні можна було б помістити в один загальний запис. Оскільки не кожний студент бере участь, наприклад, у спорті або самодіяльності, то прийшлося б вибрати запис з змінною довжиною або запис із фіксованою довжиною, причому в останньому випадку частина пам'яті витрачалася б даремно. Ієрархічна структура усуває труднощі, що виникають при цьому, тому що в будь-якому екземплярі типу набору з записом-власником можна асоціювати стільки записів-членів, скільки необхідно для конкретного екземпляра.

Повна схема бази даних формується в загальному випадку з множини різних типів набору і типів запису.

Ієрархічна деревоподібна структура, що орієнтована від коріння, задовольняє такі умови [16]:

1) ієрархія завжди починається з кореневого вузла;
2) на першому рівні може знаходитися тільки один вузол - кореневий;
3) на нижніх рівнях знаходяться породжені (залежні) вузли;
4) кожний породжений вузол, який знаходиться на рівні *i*, зв'язаний тільки з одним вхідним вузлом, який знаходиться на рівні *(i-1)* ієрархії дерева;

5) кожний вхідний вузол може мати один або декілька породжених вузлів, які називаються подібними;
6) доступ до кожного породженого вузла виконується через відповідний йому вхідний вузол;
7) існує єдиний ієрархічний шлях доступу до будь-якого вузла, починаючи від кореневого вузла дерева.

Прикладами типових операторів маніпулювання ієрархічно-організованими даними можуть бути такі:

- пошук заданого дерева БД;
- перехід від одного дерева до іншого;
- перехід від одного запису до іншого в середині дерева;
- перехід від одного запису до іншого в порядку обходу ієрархії;
- установлення нового запису в зазначену позицію;
- видалення поточного запису.

Перевагами деревовидної моделі є наявність функціональних систем керування базами даних, які підтримують дану модель; простота сприйняття користувачами принципу ієрархії; забезпечення деякого рівня незалежності даних; простота оцінки операційних характеристик системи завдяки апріорно заданим взаємозв'язкам.

До недоліків ієрархічних структур відносять надлишковість зберігання інформації, так як ієрархічні структури не підтримують взамозв'язки Б:Б; строгу ієрархічну впорядкованість, яка ускладнює процедури включення та вилучення записів; вилучення вихідних вузлів призводить до вилучення відповідних їм породжених, що вимагає особливої обережності; ускладнюється доступ до даних, які лежать на більш низьких рівнях ієрархії, оскільки кореневий вузол завжди є головним, а доступ до будь-якого породженого вузла може здійснюватись через вихідний.

Розглянемо в якості прикладу задачу. Нехай необхідно побудувати ієрархічну базу даних, до якої входить інформація про викладачів, студентів та дисципліни в таких взаємозв'язках:

О - Б
ВИКЛАДАЧ ⟷ ДИСЦИПЛІНА
О - Б
СТУДЕНТ ⟷ ДИСЦИПЛІНА,

О - Б
де ⟷ - відношення ОДИН – ДО –БАГАТЬОХ.

Ця інформація в ієрархічній моделі може бути представлена різними способами. Один з варіантів показаний на рис.3.6. Кореневим вузлом є об'єкт СТУДЕНТ. Для кожного студента при даному представленні є екземпляр кореневого вузла.

Об'єкти ВИКЛАДАЧ, ДИСЦИПЛІНА об'єднані в породжуваний вузол ДИСЦИПЛІНА + ВИКЛАДАЧ. В кожний момент часу база даних, що задана моделлю рис. 3.6, може мати записи для N студентів. На рис.3.7 показаний екземпляр запису бази даних для студента Собка Н.Ю.

Побудована модель дозволяє виконати такі операції, як оперативна видача інформації про здачу іспитів студентами з різних дисциплін (отримати кількісну та якісну оцінки). В той же час, якщо необхідно отримати інформацію про те, які викладачі по дисципліні "Схемотехніка" приймають іспити, то прийдеться переглянути всі записи породжуваних вузлів всіх екземплярів кореневого вузла. При такій постановці задачі більше підходить модель, яка представлена на рис.3.8, де кореневим вузлом є об'єкт ВИКЛАДАЧ, а об'єкти СТУДЕНТ та ДИСЦИПЛІНА об'єднані в породжуваний вузол СТУДЕНТ+ ДИСЦИПЛІНА. Крім того, можна відмітити, що дані про викладачів та назви дисциплін в базі даних (див. рис.3.6) повторюються в екземплярі породжуваного вузла стільки

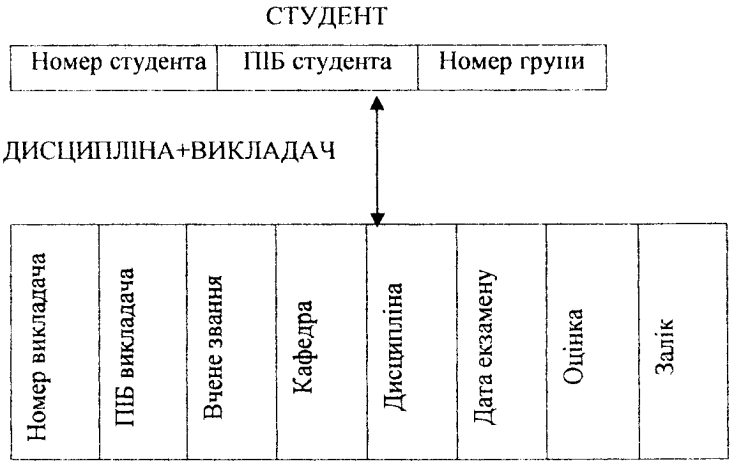
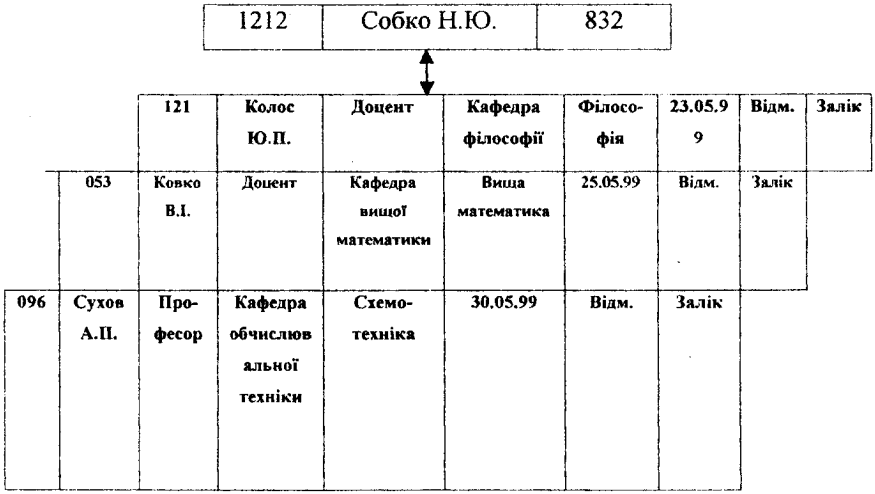


Рисунок 3.6 – Представлення бази даних (СТУДЕНТ-ДИСЦИПЛІНА) за допомогою ієрархічної моделі



		1212	Собко Н.Ю.		832				
		121	Колос Ю.П.	Доцент	Кафедра філософії	Філосо-фія	23.05.99	Відм.	Залік
	053	Ковко В.І.	Доцент	Кафедра вищої математики	Вища математика	25.05.99	Відм.	Залік	
096	Сухов А.П.	Про-фесор	Кафедра обчислювальної техніки	Схемо-техніка	30.05.99	Відм.	Залік		

Рисунок 3.7 – Екземпляр запису ієрархічної бази даних СТУДЕНТ – ДИСЦИПЛІНА

разів, скільки студентів навчаються у даного викладача. В моделі на рис.3.8 даними , що дублюються, є відомості про студентів.

На прикладах (див. рис.3.6 та рис.3.8) проілюструємо проблеми, які пов'язані з операціями включення та вилучення даних.

З принципу ієрархії випливає, що екземпляр породжуваного вузла не може існувати при відсутності відповідного йому екземпляра вихідного вузла. Таким чином, неможливо без використання будь-яких інших методів включити в базу даних, представлену на рис.3.6, відомості про викладача, який не приймає іспити або заліки. Одним з методів реалізації такого включення є формування "пустих" екземплярів, які приводять до додаткових затрат зовнішньої пам'яті ЕОМ і до необхідності строгого обліку

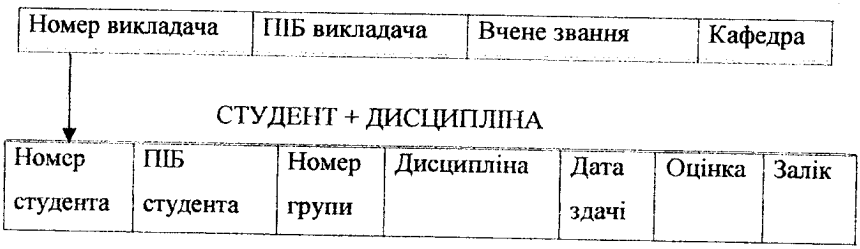


Рисунок 3.8 – Альтернативне представлення бази даних СТУДЕНТ-ДИСЦИПЛІНА

таких записів. При вилученні екземплярів вихідного вузла автоматично вилучаються всі екземпляри породжені вузлом. Наприклад, в базі даних, модель якої приведена на рис.3.8, вилучення екземпляра ВИКЛАДАЧ потягне за собою і вилучення всіх екземплярів про студентів, які навчаються в даного викладача. Вказане повністю неприпустиме при розв'язку, наприклад, задачі оцінки якості навчання студентів.

В ієрархічних базах даних проблеми, пов'язані з операціями включення нових записів і вилучення старих, а також проблеми часткового

дублювання інформації виникають в результаті того, що відношення БАГАТО-ДО-БАГАТЬОХ безпосередньо не підтримується, що і є основним недоліком ієрархічних моделей.

Контрольні запитання

1. В чому полягають особливості структурної організації ієрархічних моделей даних?
2. Чим обумовлено широке використання збалансованих дерев?
3. Перерахуйте типові оператори маніпулювання ієрархічними даними.
4. Перерахуйте основні переваги та недоліки ієрархічних моделей даних.

3.2 Мережна модель даних

Більш широкі можливості для користувача забезпечує мережна модель бази даних, яка є узагальненням ієрархічної моделі і дозволяє відображати відношення між типами записів виду «багато до багатьох». У мережній моделі кожний тип запису може бути членом більш ніж одного типу набору. В результаті можна сформувати модель бази даних із довільними зв'язками між різними типами записів. Крім того, окремі типи записів можна не включати ні в які типи набору, що забезпечує додаткові можливості для ряду задач обробки даних в СКБД. Відзначимо, що СКБД, в основі якої використовується мережна модель бази даних, називається СУБД мережного типу. В 1971 році був опублікований офіційний стандарт мережних баз даних, який відомий як модель CODASYL [16].

В основі мережної моделі даних лежать мережні структури. Розглянемо мережні структури. Припустимо, що нам необхідно графічно представити відношення між об'єктами СТУДЕНТСЬКИЙ_КОЛЕКТИВ та УЧБОВА_ГРУПА, КІМНАТА_В_ ГУРТОЖИТКУ та СТУДЕНТ. Взаємозв'язок між цими об'єктами показаний на рис.3.9, звідки можна побачити, що дана схема не є ієрархічною, оскільки породжений елемент

Номер викладача	ПІБ викладача	Вчене звання	Кафедра

СТУДЕНТ + ДИСЦИПЛІНА

Номер студента	ПІБ студента	Номер групи	Дисципліна	Дата здачі	Оцінка	Залік

Рисунок 3.8 – Альтернативне представлення бази даних СТУДЕНТ-ДИСЦИПЛІНА

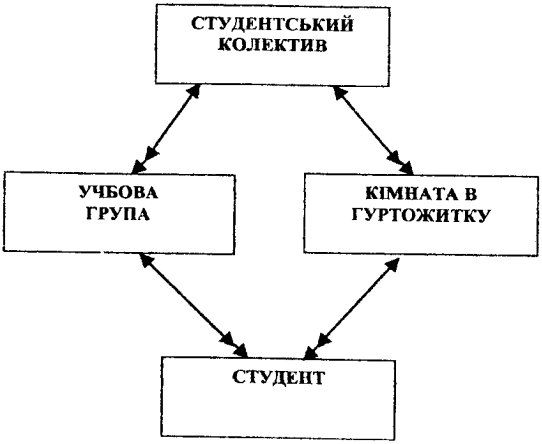


Рисунок 3. 9 – Приклад взаємозв'язків між об'єктами мережної структури

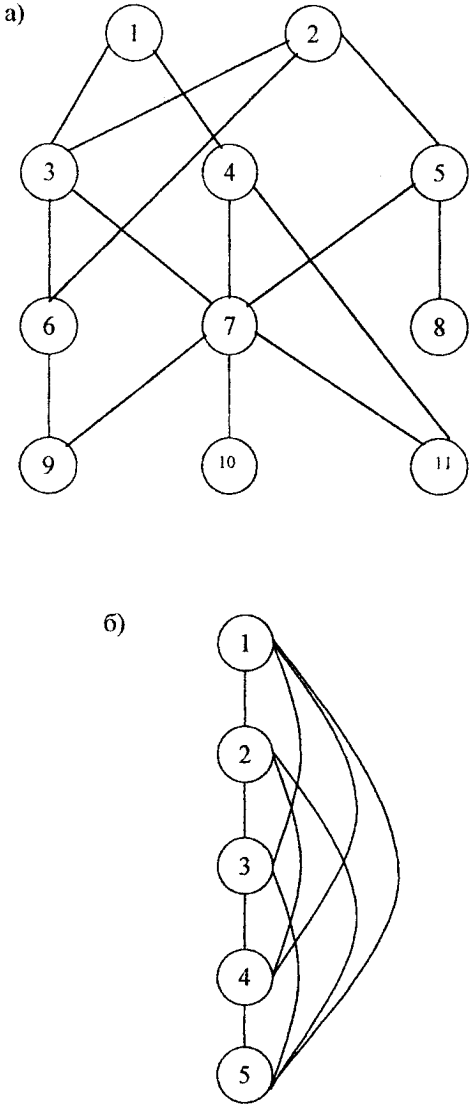


Рисунок 3.10 – Приклади мережних структур

СТУДЕНТ має два вихідних. Мережна структура відрізняється від ієрархічної тим, що в ній будь-який елемент може зв'язуватися з будь-яким іншим елементом.

На рис.3.10 наводяться приклади більш складних мережних структур, де для зручності цифрами 1,2,..., 11 позначені елементи (об'єкти). Мережну структуру можна описати за допомогою вихідних та породжених елементів та зобразити її таким чином, щоб породжені елементи знаходилися нижче вихідних, що і зроблено на рис 3.10. При розгляді деяких мережних структур природно розглядати рівні, як і при деревовидних структурах. Структура на рис.3.9 має три рівні, а мережні структури на рис.3.10 - відповідно чотири та п'ять рівнів.

Розглянемо як подаються в мережній структурі взаємозв'язки між об'єктами. На рисунку 3.9 представлена мережна структура, в якій між двома об'єктами присутні два види взаємозв'язку: ОДИН-ДО-БАГАТЬОХ (наприклад, між об'єктами НАВЧАЛЬНА_ГРУПА - СТУДЕНТ) та БАГАТО-ДО-ОДНОГО (СТУДЕНТ - КІМНАТА_В_ГУРТОЖИТКУ). Цей вид зв'язку закладено в ієрархічних структурах при умові, що дані зв'язки існують відповідно між вихідними та породженими, а зв'язок БАГАТО-ДО-ОДНОГО – між породженими та вихідними вузлами. Виконання цієї умови для відповідних вузлів мережної схеми говорить про просту мережну структуру. Складною мережною структурою називають схему, в якій присутній хоча б один зв'язок БАГАТО-ДО-БАГАТЬОХ [16].

На рис.3.11 показаний приклад складної мережної структури та можливі взаємозв'язки між об'єктами. В цьому прикладі вузол ВИКЛАДАЧ може мати декілька породжених, тому що викладач веде заняття більш ніж з одним студентом. А кожен студент навчається більш ніж у одного викладача.

Розділення мережних структур на два типи (складні та прості) необхідно хоча б тому, що структури, побудовані з використанням зв'язку БАГАТО-ДО-БАГАТЬОХ, вимагають для їх реалізації використання більш

складних методів. Крім того, деякі системи керування базами даних можуть підтримувати прості мережні структури, але не можуть підтримувати складні. Наприклад, СКБД DMS, DBMS, СЕКТОР дозволяють описувати прості мережні структури. Реалізація складних мережних структур можлива і в цих системах керування базами даних шляхом приведення їх до простішого вигляду.

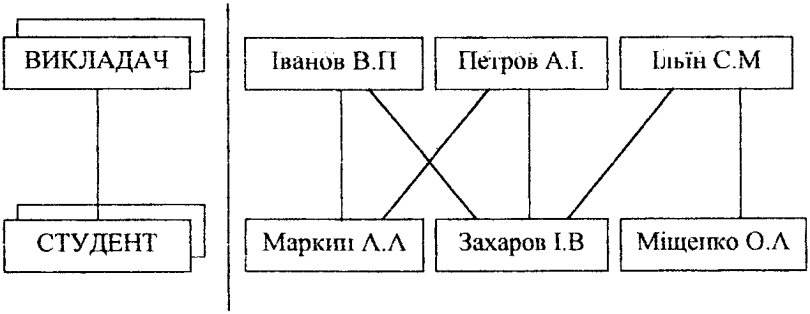


Рисунок 3.11 – Складна мережна структура

Будь-яку мережну структуру можливо привести до простішого вигляду, якщо ввести надлишковість (рис.3.12). Якщо надлишковість, яка при цьому виникає, є допустимою, то такий шлях дозволяє підтримувати мережні структури даних за допомогою СКБД, які орієнтовані на ієрархічну організацію даних.

Прикладами типових операторів маніпулювання мережними даними можуть бути:

- пошук конкретного запису в наборі однотипних записів;
- перехід від предка до першого нащадка за деяким зв'язком;
- перехід до наступного нащадка в деякому зв'язку;
- перехід від нащадка до предка за деяким зв'язком;
- створення нового запису;
- знищення запису;
- модифікація запису;
- включення в зв'язок;

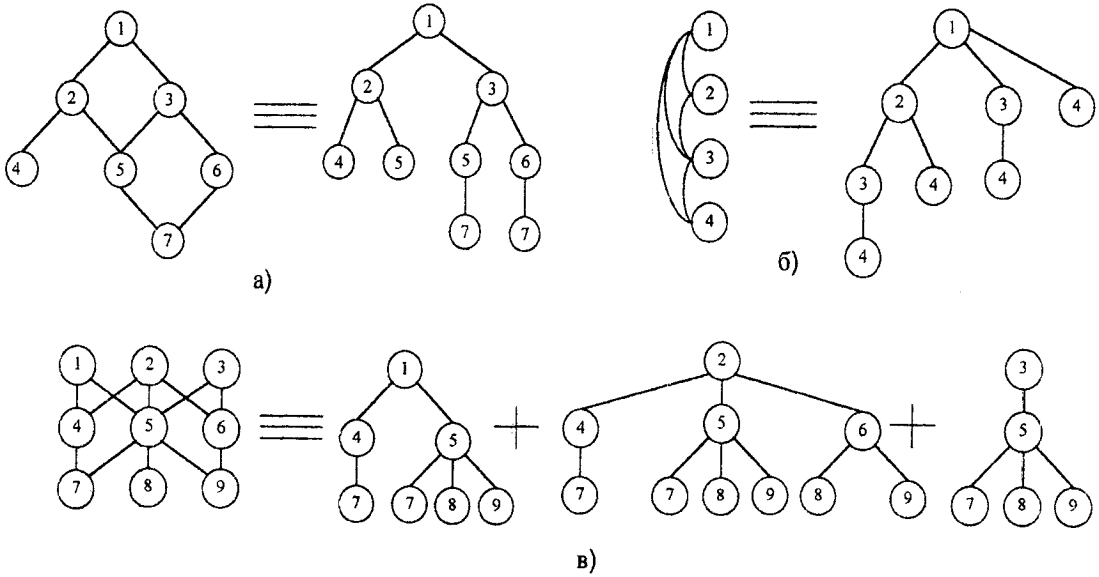


Рисунок 3.12 – Представлення простої мережної структури:
а, б — у вигляді одного дерева;
в — у вигляді суми дерев

- виключення зі зв'язку;
- перестановка в інший зв'язок і т.д.

Бази даних, які описуються мережною моделлю, складаються з декількох областей (рис.3.13). Кожна область складається з записів, які в свою чергу складаються з полів. Об'єднання записів в логічну структуру можливе не тільки по областях, але й за допомогою так званих наборів. Термін набір є основною конструкцією мови системи баз даних КОДАСІЛ. Набір – це пойменоване дворівневе дерево, яке дозволяє формувати багаторівневі дерева та прості мережні структури. Таким чином, база даних КОДАСІЛ складається з деякої кількості наборів. Використовуючи дворівневі зв'язки, спеціаліст з аналізу системи може конструювати достатньо складні структури даних. Набір – це екземпляр пойменованої сукупності записів. Для кожного типу набору тип запису може бути оголошений його власником. Кожний набір повинен мати один екземпляр запису та будь-яку кількість екземплярів кожного типу запису – членів набору. Наприклад, набір можна використовувати для об'єднання записів про студентів одної групи (див. рис. 3.10).

Перерахуємо властивості, що притаманні набору [15,16]:

а) набір – це пойменована сукупність зв'язаних записів;

б) в кожному екземплярі набору присутній тільки один екземпляр власника;

в) екземпляр набору може мати нуль, один або декілька записів-членів;

г) набір вважається пустим, якщо жодний з екземплярів запису-члена не має зв'язків з відповідним екземпляром запису-володаря;

д) екземпляр набору існує після запам'ятовування запису-володаря;

е) тип набору є логічний взаємозв'язок ОДИН-ДО-БАГАТЬОХ між володарем та членом набору (рис.3.14);

ж) кожному типу набору надається ім'я, що дозволяє одній і тій самій парі типів об'єктів брати участь у декількох взаємозв'язках.

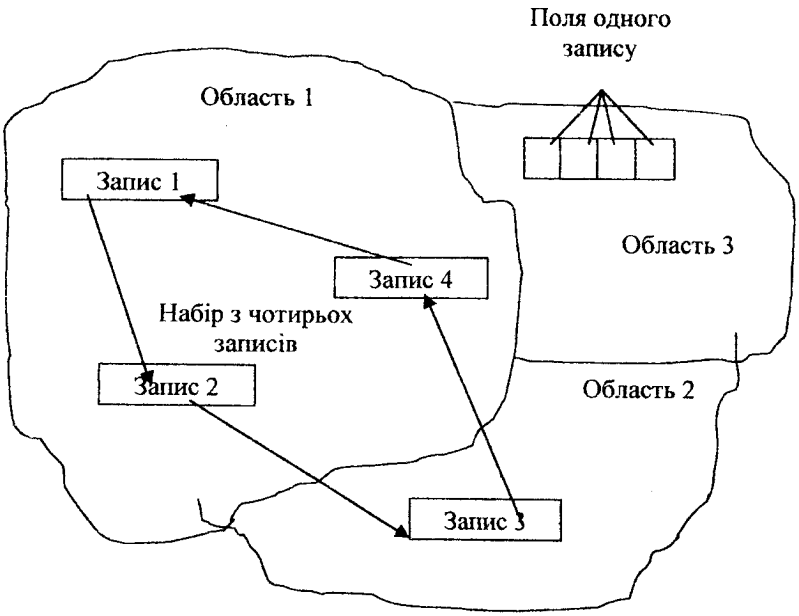


Рисунок 3.13 – Структура мережної бази даних

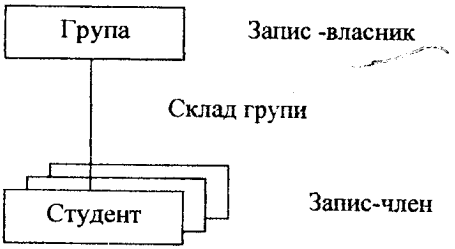


Рисунок 3.14 – Тип набору "Склад групи"

Необхідно розрізняти тип та екземпляр набору. Але спочатку пояснимо різницю між поняттями тип та екземпляр запису. СТУДЕНТ є типом запису, а рядок символів "Захаров Іван Викторович, член профкому, кімн. 638" – екземпляром типу запису СТУДЕНТ. Іншими словами, в базі даних можуть зберігатися один чи декілька екземплярів запису деякого типу. Подібне відношення існує і між типом набору та екземпляру. У прикладі на рис.3.14 наведений тип набору СКЛАД ГРУПИ, а на рис.3.15 - його екземпляр, який вміщує екземпляр типу запису-володаря ГРУПА "832 група, куратор Іванов І.П., каф. кібернетики" та N екземплярів типу запису-члена: "1013 Захаров І.П., староста групи", "201К Кирилова О.Б.", ..., "426Х Харламов О.М."

Таким чином, екземпляр типу набору складається з одного екземпляра типу запису-володаря, нуля чи більше екземплярів типу запису-членів даного типу набору. Між екземпляром типу запису-володаря та екземпляром типу запису-члена існує взаємозв'язок ОДИН-ДО-БАГАТЬОХ. Певний екземпляр типу запису-члена в екземплярі даного типу набору не може одночасно належати більш ніж одному екземпляру типу запису-володаря. Іншими словами, унікальність володаря типу набору є обов'язковою. В моделі даних, що являє собою взаємозв'язок ОДИН-ДО-БАГАТЬОХ, тип запису-володаря має від 0 до N екземплярів типу запису-члена. В свою чергу тип запису-члена в іншому наборі може мати роль типу запису-володаря. Запис-володар даного набору може грати дану роль у декількох наборах. Така структура є ієрархію. Отже, ієрархічна модель даних є окремим випадком мережної моделі.

Суттєва відмінність між мережною та ієрархічною моделями даних полягає в тому, що в мережній моделі кожний запис може брати участь у будь-якій кількості наборів. На рис.3.16 в мережній моделі, що представляється двома типами наборів ВЧИТЕЛЬ_ ВИКЛАДАЄ_ ДИСЦИПЛІНУ та СТУДЕНТ_НАВЧАЄТЬСЯ _ ДИСЦИПЛІНІ, запис-член ДИСЦИПЛІНА входить до обох типів наборів та є зв'язкою цих наборів.

832 група куратор Іванов І.П. каф. кібернетики	1013 Захаров І.П. староста групи	201К Кирилова О.Б.	426Х Харламов О.М.

Рисунок 3.15 – Екземпляр набору СКЛАД_ГРУПИ

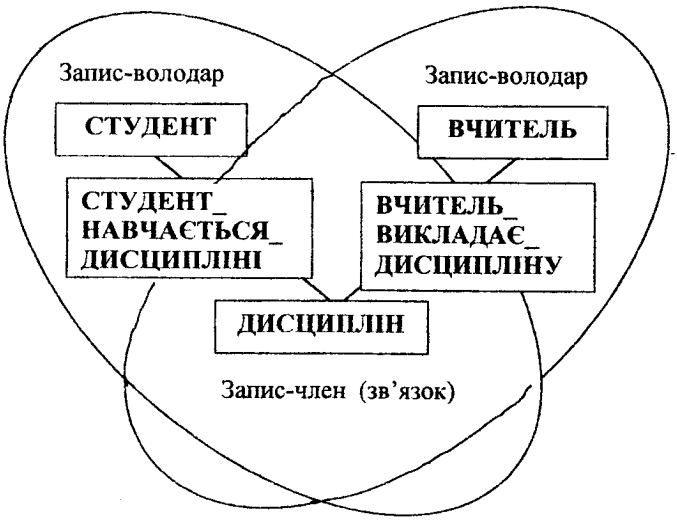


Рисунок 3.16 – Представлення відношення M:N у мережній моделі

Крім того, будь-який запис мережної моделі може мати роль як володаря, так і члену набору.

До переваг мережних структур слід віднести наявність СКБД, які успішно реалізують таку організацію, а також простоту реалізації зв'язків "багато до багатьом", які часто зустрічаються в реальному світі. Недолік таких структур полягає в їх складності по відношенню до ієрархічних структур. Прикладному програмісту часто необхідно знати логічну структуру бази даних. При реорганізації бази даних не виключається в ряді випадків втрата незалежності даних.

Представлення даних ієрархічними та мережними структурами, в загальному випадку, перешкоджає внесенню багатьох змін, пов'язаних з розширенням бази даних. Це може призвести до порушення логічного представлення даних, схем, підсхем, а, отже, до зміни в прикладних програмах.

Контрольні запитання

1. Які типи зв'язків використовують в мережних структурах?
2. В чому відмінність простої і складної мережних структур?
3. В чому полягають переваги та недоліки мережних структур?
4. Яким чином мережні структури подаються ієрархічними?

3.3 Реляційна модель даних

В деяких випадках зростання ієрархічної чи мережної бази даних може привести до порушення логічної організації даних. Такі ситуації виникають при появі нових користувачів, нових застосувань та видів запитів, при обліку інших логічних зв'язків між елементами даних [14,16].

Недоліки ієрархічної і мережної моделей привели до появи нової, реляційної моделі даних, створеної Коддом у 1970 році. Реляційна модель була спробою спростити структуру бази даних. У ній були відсутні явні покажчики на предків і нащадків, а всі дані були представлені у вигляді

простих таблиць, розбитих на рядки і стовпці.

Реляційною називається база даних, у якій усі дані, доступні користувачу, організовані у вигляді таблиць, а всі операції над даними зводяться до операцій над цими таблицями. Для представлення реляційних баз даних розроблена формальна теорія баз даних, теоретичну основу якої складає алгебра та математична логіка.

У реляційній базі даних інформація організована у вигляді таблиць, розділених на рядки і стовпці, на перетині яких містяться значення даних. У кожної таблиці є унікальне ім'я, що описує її вміст. Масив значень, що можуть міститися в стовпці, називається доменом цього стовпця.

Двовимірні таблиці в математиці отримали назву *відношення* (relation (англ.)) .

У кожного стовпця в таблиці є своє ім'я, що звичайно служить заголовком стовпця. Всі стовпці в одній таблиці повинні мати унікальні імена, однак дозволяється привласнювати однакові імена стовпцям, розташованим в різних таблицях.

Стовпці таблиці упорядковані зліва направо, і їхній порядок визначається при формуванні таблиці. У будь-якій таблиці завжди є як мінімум один стовпець.

Як правило, не вказується максимально допустиме число стовпців у таблиці, однак майже у всіх комерційних СКБД ця межа існує і, як правило, складає приблизно 255 стовпців.

На відміну від стовпців, рядки таблиці не мають визначеного порядку. Це значить, що якщо послідовно виконати два однакових запити для відображення вмісту таблиці, то немає гарантії, що обидва рази рядки будуть перераховані в тому самому порядку.

Рядки таблиці утворюють данні різного формату і різного типу, тобто можна стверджувати, що рядки таблиці є кортежами.

У таблиці може міститися будь-яка кількість рядків. Цілком припустиме існування таблиці з нульовою кількістю рядків. Така таблиця

називається порожньою. Порожня таблиця зберігає структуру, визначену її стовпцями, просто в ній не містяться дані. Стандарти реляційних баз даних не накладають обмежень на кількість рядків у таблиці, і в багатьох СКБД розмір таблиць обмежений лише вільним дисковим простором комп'ютера.

Як правило, в сучасних реляційних БД допускається збереження символьних, числових даних, бітових рядків, спеціалізованих числових даних (таких як "гроші"), а також спеціальних "темпоральных" даних (дата, час, часовий інтервал).

Найменша одиниця даних реляційної моделі – це окреме атомарне (неподільне) для даної моделі значення даних. Так, в одній предметній області прізвище, ім'я і по-батькові можуть розглядатися як єдине значення, а в інший – як три різних значення.

Опис кожного відношення складається з імені відношення (підмет), за яким в круглих дужках перераховується список атрибутів (присудок). Цей опис називають інтенсіоналом або схемою відношення. Під описом розуміють деяке заповнення кортежів відношення, яке називають екстенсіоналом.

Відношення називаються еквівалентними, якщо вони відрізняються тільки порядком чергування атрибутів.

Кодд у 1985 році сформулював 12 правил, які повинна задовольняти будь-як база даних, що претендує на тип реляційної.. З того часу дванадцять правил Кодда вважаються визначенням реляційної СУБД.

Дванадцять правил Кодда, яким повинна відповідати реляційна СКБД:

1. **Правило інформації.** Вся інформація в базі даних повинна бути надана виняткова на логічному рівні і тільки одним способом - у вигляді значень, що містяться в таблицях.
2. **Правило гарантованого доступу.** Логічний доступ до всіх і кожного елемента даних (атомарного значення) у реляційній базі даних повинний забезпечуватися шляхом використання комбінації імені

таблиці, первинного ключа та імені стовпця.

3. **Правило підтримки недійсних значень.** У реляційній базі даних повинна бути реалізована підтримка недійсних значень, що відрізняються від: рядка символів нульової довжини, рядка символів пробілів чи нуля будь-якого іншого числа і використовуватися для представлення відсутніх даних незалежно від типу цих даних.
4. **Правило динамічного каталогу, основаного на реляційній моделі.** Опис бази даних на логічному рівні необхідно представити в тому ж вигляді, що й основні дані, щоб користувачі, які мають відповідні права, могли працювати з нею за допомогою тієї ж реляційної мови, яку вони застосовують для роботи з основними даними.
5. **Правило вичерпної підмови даних.** Реляційна система може підтримувати різні мови і режими взаємодії з користувачем (наприклад, режим питань і відповідей). Однак повинна існувати принаймні одна мова, оператори якої підтримують такі елементи:
   - визначення даних;
   - визначення представлень;
   - обробку даних (інтерактивну і програмну);
   - умови цілісності;
   - ідентифікацію прав доступу;
   - границі транзакцій (початок, завершення і скасування).
6. **Правило відновлення представлень.** Усі представлення, які теоретично можна обновити, повинні бути доступні для відновлення.
7. **Правило доповнення, відновлення і вилучення.** Можливість працювати з відношенням як з одним операндом повинна існувати не тільки при читанні даних, але і при доповненні, відновленні і вилученні даних.
8. **Правило незалежності фізичних даних.** Прикладні програми й утиліти для роботи з даними повинні на логічному рівні залишатися недоторканими при будь-яких змінах методів збереження даних чи

методів доступу до них.

9. **Правило незалежності логічних даних.** Прикладні програми й утиліти для роботи з даними повинні на логічному рівні залишатися недоторканими при внесенні в базові таблиці будь-яких змін, які теоретично дозволяють зберегти недоторканими дані, що містяться в цих таблицях.
10. **Правило незалежності умов цілісності.** Повинна існувати можливість визначати умови цілісності, специфічні для конкретної реляційної бази даних, на підмові реляційної бази даних і зберігати їх у каталозі, а не в прикладній програмі.
11. **Правило незалежності поширення.** Реляційна СКБД не повинна залежати від потреб конкретного клієнта.
12. **Правило одиничності.** Якщо в реляційній системі є низькорівнева мова (що обробляє один запис за один раз), то повинна бути відсутня можливість використання її для того, щоб обійти правила й умови цілісності, виражені на реляційній мові високого рівня (що обробляє кілька записів за один раз).

Правило 2 вказує на роль первинних ключів при пошуку інформації в базі даних. Ім'я таблиці дозволяє знайти необхідну таблицю, ім'я стовпця дозволяє знайти необхідний стовпець, а первинний ключ дозволяє знайти рядок, який містить шуканий елемент даних.

Правило 3 вимагає, щоб відсутні дані можна було представити за допомогою недійсних значень (NULL).

Правило 4 говорить, що реляційна база даних повинна сама себе описувати. Іншими словами, база даних повинна містити набір системних таблиць, що описують структуру самої бази даних.

Правило 5 вимагає, щоб СКБД використовувала мову реляційної бази даних, наприклад SQL, хоча явно SQL у правилі не згадують. Така мова повинна підтримувати всі основні функції СКБД — створення бази даних, читання і введення даних, реалізацію захисту бази даних і т.д.

Правило 6 дозволяє показувати різним користувачам різні фрагменти структури бази даних.

Правило 7 акцентує увагу на тому, що бази даних за своєю природою орієнтовані на множини. Воно вимагає, щоб операції доповнення, вилучення і відновлення можна було виконувати над множинами рядків. Це правило призначене для того, щоб заборонити реалізації, у яких підтримуються тільки операції над одним рядком.

Правила 8 і 9 стверджують, що конкретні способи реалізації чи збереження доступу, які використовуються в СКБД, і навіть зміни структури таблиць бази даних не повинні впливати на можливість користувача працювати з даними.

Правило 10 вимагає, щоб мова бази даних підтримувала обмежувальні умови, які накладаються на дані, що вводяться, і дії, що можуть бути виконані над даними.

Правило 11 говорить, що мова бази даних повинна забезпечувати можливість роботи з розподіленими даними, розташованими на інших комп'ютерних системах.

І, нарешті, правило 12 запобігає використанню інших можливостей для роботи з базою даних, крім мови бази даних, оскільки це може порушити її цілісність.

Згідно Дейту реляційна модель складається з трьох частин, що описують різні аспекти реляційного підходу: структурної частини, маніпуляційної частини та цілісної частини [3].

У структурній частині моделі фіксується, що єдиною структурою даних, яка використовується в реляційних БД, є нормалізоване n-арне відношення.

В маніпуляційній частині моделі стверджується два фундаментальних механізми маніпулювання реляційними БД - реляційна алгебра і реляційне числення. Перший механізм базується в основному на класичній теорії множин (з деякими уточненнями), а другий - на класичному логічному

апараті числення предикатів першого порядку.

У цілісній частині реляційної моделі даних фіксуються дві базові вимоги цілісності, що повинні підтримуватися в будь-якій реляційній СКБД. Перша вимога називається вимогою цілісності сутностей. Об'єкту або сутності реального світу в реляційній БД відповідають кортежі відношень. Конкретно вимога полягає в тому, що будь-який кортеж будь-якого відношення відрізняється від будь-якого іншого кортежу цього відношення, тобто іншими словами, будь-яке відношення повинно мати первинний ключ. Друга вимога називається вимогою цілісності по посиланнях. Очевидно, що при дотриманні нормалізованності відношень складні сутності реального світу представляються в реляційній БД у вигляді декількох кортежів декількох відношень.Вимога цілісності по посиланнях, або вимога зовнішнього ключа полягає в тому, що для кожного значення зовнішнього ключа, який з'являється у відношенні, до якого посилаються, повинний існувати кортеж із таким же значенням первинного ключа. Так, наприклад, якщо для співробітника зазначений номер відділу, то цей відділ повинний існувати.

Степінь відношення - це число його атрибутів. Відношення степеня один називають унітарним, степеня два - бінарним, степеня три - тернарним, ... , а степеня *n* - *n*-арним.

Кардинальне число або потужність відношення - це число його кортежів. Кардинальне число відношення змінюється в часі на відміну від його степеня.

До переваг реляційної бази можна віднести незалежність від фізичного рівня представлення, зручність і розуміння організації даних користувачами, максимальна гнучкість при обробці непередбачених запитів, можливість розширення бази приєднанням нових елементів й записів без зміни при цьому існуючих підсхем та прикладних програм.

Контрольні запитання

1. Назвіть 12 правил Кодда, які повинна задовольняти реляційна БД.
2. Як визначається степінь та кардинальне число відношення?
3. Чи існують в сучасних СКБД обмеження на розміри відношень?
4. Перерахуйте переваги та недоліки реляційних моделей даних.

4 ОСНОВИ ПРОЕКТУВАННЯ РЕЛЯЦІЙНИХ БАЗ ДАНИХ

4.1 Поняття ключа, основні типи ключів

Якщо кожному стовпцю таблиці присвоїти ім'я, то розміщення стовпців буде несуттєвим. Список імен атрибутів одного відношення називається схемою заміщення; кожне відношення, як правило, має свою назву (ім'я). Якщо, наприклад, відношення (ШИФР ГРУПИ, КІЛЬКІСТЬ СТУДЕНТІВ, ФАКУЛЬТЕТ, СТАРОСТА, КУРАТОР) назвемо ГРУПА, то схема буде мати атрибути ШИФР ГРУПИ, КІЛЬКІСТЬ СТУДЕНТІВ, ФАКУЛЬТЕТ, СТАРОСТА, КУРАТОР. Формально схема відношення описується таким чином:

ІМ'Я_ВІДНОШЕННЯ (A_1, A_2, A_3, ..., A_K),

де A_1, A_2, ..., A_K – імена атрибутів.

Стовпець чи ряд стовпців, значення яких однозначно ідентифікують кожний рядок таблиці, називають ключем. Так, в розглянутому прикладі можливими ключами можуть бути атрибути: ШИФР ГРУПИ, СТАРОСТА. А такі атрибути, як КІЛЬКІСТЬ СТУДЕНТІВ, ФАКУЛЬТЕТ, не можуть бути ключами, оскільки різні студентські групи можуть мати однакову кількість студентів і на одному факультеті вчиться декілька груп.

Якщо відношення має більше одного можливого ключа, тоді має сенс виділити один ключ, який називають первинним (ШИФР ГРУПИ). І навпаки, якщо відношення не має жодного атрибута, який би повністю визначив об'єкт (рядок, кортеж), тоді приходиться мати справу зі складеним ключем, який схематично відображають подвійною горизонтальною рискою. Наприклад, в відношенні СТУДЕНТ-УСПІШНІСТЬ (табл.4.1) відсутній атрибут, який би однозначно ідентифікував кортеж, оскільки екзамени з одного навчального курсу (фізика) можуть здаватися в декількох семестрах. [12,16]

Таблиця 4.1 - **Відношення СТУДЕНТ-УСПІШНІСТЬ зі складеним ключем**

Прізвище, ім'я, по батькові студента	Назва Дисципліни	Дата здачі екзамену	Оцінка	Залік
Іванов П.В.	фізика	13.12.98	4	Залік
Іванов П.В.	фізика	17.12.98	5	Залік
Іванов П.В.	математичний аналіз	13.12.98	3	Залік
Петров А.С	фізика	3.01.99	3	Залік

Позначимо атрибути відношення СТУДЕНТ-УСПІШНІСТЬ відповідно:

ПІБ СТУДЕНТА - F;

НАЗВА ДИСЦИПЛІНИ - N;

ДАТА ЗДАЧІ ЕКЗАМЕНУ - D;

ОЦІНКА - C;

ЗАЛІК - Z.

Тоді відношення СТУДЕНТ-УСПІШНІСТЬ, що містить атрибути F, N, D, C, Z, можна представити у вигляді схеми (рис. 4.1).

СТУДЕНТ-УСПІШНІСТЬ

F	D	N	C	Z

Рисунок 4.1 – Схематичне представлення відношення СТУДЕНТ-УСПІШНІСТЬ

Щоб виділити вказаним способом складений ключ відношення СТУДЕНТ-УСПІШНІСТЬ, потрібно представити домени відношення таким чином, щоб мінімальна кількість атрибутів, які однозначно визначають кортеж, були розміщені на початку схеми. Відношення СТУДЕНТ-УСПІШНІСТЬ прийме вигляд (рис.4.2):

СТУДЕНТ-УСПІШНІСТЬ

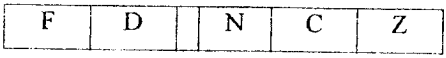


Рисунок 4.2 – Складений ключ

Атрибути F і D визначають кожний рядок даної таблиці. Якщо припустити, що можлива здача двох і більше екзаменів одним студентом в один день, тоді атрибути F, D вже не будуть однозначно ідентифікувати кортеж, оскільки даним екземплярам атрибутів F, D може відповідати в таких випадках два і більше кортежі. При цих умовах складений ключ буде містити три атрибути, що схематично зображено на рис 4.3.

СТУДЕНТ-УСПІШНІСТЬ

Рисунок 4.3 – Складений ключ, який включає три атрибути

Можливі випадки, коли складений ключ включає всі атрибути відношення.

Розрізняють наступні типи ключів: простий ключ, повністю складений ключ, напівскладений ключ.

Простим називається ключ, що складається з одного атрибута, причому атрибут повинен бути атомарним, а екземпляри даного атрибута – унікальні.

Атомарним є ключ, значення якого сприймається програмою (СКБД) як неподільний елемент даних, навіть якщо він створений з декількох об'єктів. Наприклад, атомарність атрибута ДАТА ЗДАЧІ ЕКЗАМЕНА в відношенні означає неможливість виділення з дати окремо числа, місяця та року.

Повністю складеним називається ключ, що містить декілька атрибутів, причому між цими атрибутами існує відображення (залежність) БАГАТО-ДО-БАГАТЬОХ. Атрибути, що складають такий ключ, не залежать один від одного. Приклад повністю складеного ключа в відношенні СТУДЕНТ-ВИКЛАДАЧ приведено в табл. 4.2.

Таблиця 4.2 - **Приклад відношення з повністю складеним ключем СТУДЕНТ-ВИКЛАДАЧ**

Студент	Викладач	Дисципліна
Іванов	Харитонов	фізика
Петров	Харитонов	фізика
Іванов	Міщенко	інформатика
Петров	Борисов	інформатика

Напівскладеним називається ключ, що містить декілька атрибутів, між якими на відміну від повністю складеного ключа існує залежність Б: 1(БАГАТО-ДО-ОДНОГО). Тут атрибути в ключі впорядковуються за принципом: кожний наступний уточнює попередній. У відношенні СТУДЕНТ-УСПІШНІСТЬ використовувався напівскладений ключ F, D .

Взаємозв'язок таблиць є найважливішим елементом реляційної моделі даних. Вона підтримується зовнішніми ключами (*foreign key*). Розглянемо приклад, у якому база даних зберігає інформацію про рядових співробітників (таблиця СПІВРОБІТНИК) і керівників (таблиця КЕРІВНИК) у деякій організації (рис.4.4.). Первинний ключ таблиці КЕРІВНИК - стовпець НОМЕР (наприклад, табельний номер). Стовпець ПРІЗВИЩЕ не може виконувати роль первинного ключа, тому що в одній організації можуть працювати два керівники з однаковими прізвищами. Будь-який співробітник підпорядковується єдиному керівнику, що

повинно бути відображено в базі даних. Таблиця СПІВРОБІТНИК містить стовпець НОМЕР керівника, і значення в цьому стовпці вибираються зі стовпця НОМЕР таблиці КЕРІВНИК (див. рис.4.4). Стовпець НОМЕР КЕРІВНИКА є зовнішнім ключем для таблиці СПІВРОБІТНИК.

Контрольні запитання

1. Які основні вимоги висуваються до ключів?
2. Які типи відношень між атрибутами характерні для різних типів ключів?
3. В яких випадках застосовується зовнішні ключі?
4. Коли ключ включає всі атрибути відношення?

4.2 Основи реляційної алгебри

Реляційна модель баз даних надає можливість маніпулювати над доменами відношень. Для цих цілей існує два види апаратів маніпулювання відношеннями: реляційна алгебра (алгебра відношень) і реляційне обчислення (обчислення відношень). Алгеброю відношень називають систему операцій маніпулювання відношеннями, кожний оператор якого в якості операнда (операндів) використовує одне чи більше відношень і утворює нове відношення за попередньо обумовленим правилом.

Реляційне обчислення дозволяє шляхом використання обчислення предикатів та кванторів змінних описувати відношення та операції над ними в вигляді аналітичного виразу або формули.

В реляційній алгебрі використовують п'ять основних операцій: об'єднання, різниця, декартовий добуток, проекція і селекція. [14,16]

6. **Проекція.** Суть цієї операції полягає в тому, що береться відношення R, видаляються деякі з його компонентів і перевпорядковуються компоненти, що залишились. Якщо в результаті

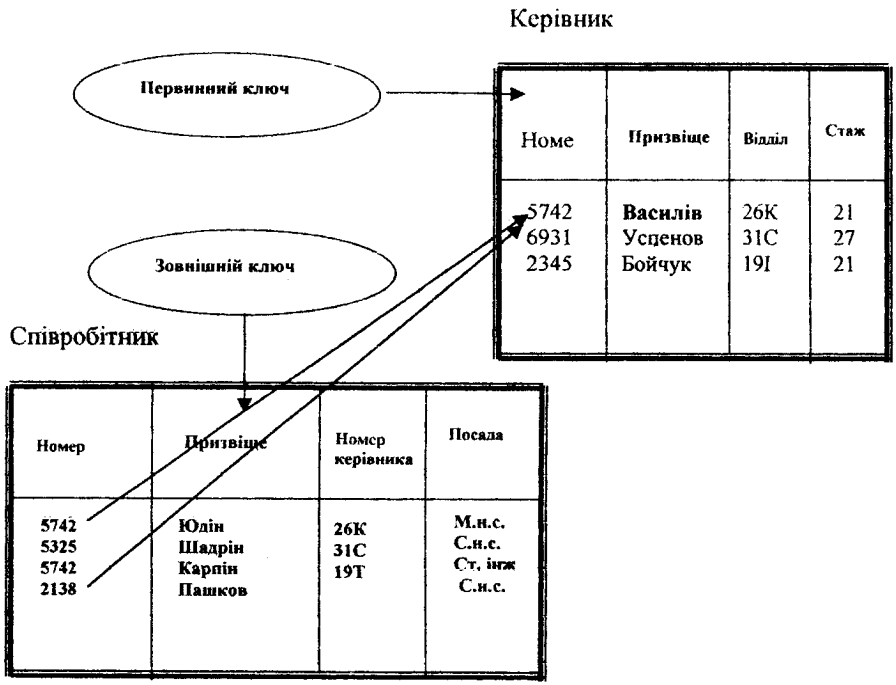


Рисунок 4.4 – Схематичне представлення зовнішнього ключа

проекції з'являються однакові кортежі, то вони з результуючого відношення вилучаються.

Операція проекції полягає в виділенні необхідних стовпців (доменів) з відношення. Нехай дано відношення СТУДЕНТ-УСПІШНІСТЬ.

Таблиця 4.3 - **СТУДЕНТ-УСПІШНІСТЬ**

ПІБ студента	Назва Дисципліни	Дата здачі екзамену	Оцінка
Іванов П.В.	Фізика	13.01.2000	4
Іванов П.В.	Фізика	14.06.2000	5
Іванов П.В.	Мат.аналіз	10.01.2000	3
Петров А.С.	Фізика	3.01.2000	3

В результаті виконання операції проекції отримуємо нове відношення, яке представлено в табл.4.4.

6. **Об'єднання.** Об'єднання відношень R і S (позначається R∪S) представляє собою множину кортежів, які належать R чи S або їм обом. Оператор об'єднання застосовується тільки до відношень однакової арності. Якщо в результаті об'єднання відношень мають місце однакові кортежі, то вони заміняються одним.

Таблиця 4.4 - **Приклад операції "проекція"**

ПІБ Студента	Назва дисципліни	Оцінка
Іванов П.В.	Фізика	4
Іванов П.В.	Фізика	5
Іванов П.В.	Мат. аналіз	3
Петров А.С.	Фізика	3

Нехай задано два відношення, представлені таблицями 4.5, 4.6. Виконаємо над ними операцію об'єднання. В результаті об'єднання відношень отримуємо результуюче відношення, яке представлено в табл.4.7.

Таблиця 4.5 – **ВИКЛАДАЧІ – ДИСЦИПЛІНИ**

Викладачі	Дисципліни
Майданюк В.П.	Мікропроцесорні системи
Власюк В.Х.	Основи програмування
Романюк О.Н.	Засоби машинної графіки

Таблиця 4.6 – **ВИКЛАДАЧІ – ДИСЦИПЛІНИ**

Викладачі	Дисципліни
Арапов С.М.	Експертні системи
Войтко В.В.	Мережі ЕОМ
Обідник Д.Т.	Схемотехніка ЕОМ

Таблиця 4.7 – **ВИКЛАДАЧІ – ДИСЦИПЛІНИ**

Викладачі	Дисципліни
Арапов С.М.	Експертні системи
Войтко В.В.	Мережі ЕОМ
Обідник Д.Т.	Схемотехніка ЕОМ
Майданюк В.П.	Мікропроцесорні системи
Власюк В.Х.	Основи програмування
Романюк О.Н.	Засоби машинної графіки

6. **Різниця.** Різницею відношень R і S (позначається як R-S), називається множина кортежів, які належать R, але не належать S (рис.4.5). При реалізації різниці необхідно, щоб R і S мали одну і ту ж саму арність.

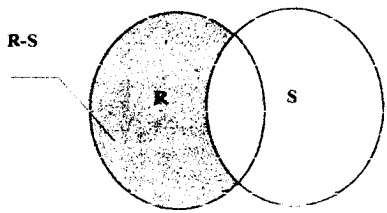


Рисунок 4.5 – Графічна ілюстрація операції віднімання відношень

Якщо A - відношення про жителів мікрорайону, B- відношення про тих, хто пройшов медичний огляд, то відношення (A-B) буде містити дані про тих жителів мікрорайону, хто не пройшов медичний огляд.

6. **Декартовий добуток.** Нехай R і S – відношення арності k_1 і k_2 відповідно. Тоді декартовим добутком відношень R і S називається множина кортежів довжини $(k_1 + k_2)$, перші k_1 компонентів яких утворюють кортежі, які належать R, а останні k_2 – кортежі, що належать S. Наприклад :

$$\begin{bmatrix} X & A \\ Y & A \\ Z & A \\ W & B \end{bmatrix} \otimes \begin{bmatrix} 5 & A \\ 1 & B \end{bmatrix} = \begin{bmatrix} X & A & 5 & A \\ X & A & 1 & B \\ Y & A & 5 & A \\ Y & A & 1 & B \\ Z & A & 5 & A \\ Z & A & 1 & B \\ W & B & 5 & A \\ W & B & 1 & B \end{bmatrix}$$

Результатом декартового добутку відношень СТУДЕНТИ (Табл. 4.8) та ЕКЗАМЕНИ (табл.4.9) буде відношення ЕКЗАМЕНАЦІЙНА ВІДОМІСТЬ (табл.4.10).

Таблиця 4.8 – **СТУДЕНТИ**

Студенти
Гаврилюк
Піддубчак
Скрипник
Ящук

Таблиця 4.9 – **ЕКЗАМЕНИ**

Дисципліна	Дата	Оцінка
Математика	10.03.2000	
Фізика	15.03.2000	

Таблиця 4.10 – **ЕКЗАМЕНАЦІЙНА ВІДОМІСТЬ**

Студенти	Дисципліна	Дата	Оцінка
Гаврилюк	Математика	10.03.2000	
Гаврилюк	Фізика	15.03.2000	
Піддубчак	Математика	10.03.2000	
Піддубчак	Фізика	15.03.2000	
Скрипник	Математика	10.03.2000	
Скрипник	Фізика	15.03.2000	
Ящук	Математика	10.03.2000	
Ящук	Фізика	15.03.2000	

5. **Селекція.** Нехай F – формула, яка може бути утворена такими засобами: а) операндами, які є константами чи номерами компонентів; б) арифметичними операторами порівняння $<, =, >, \neq, \geq, \leq$; в) логічними операторами $\wedge$(І), $\vee$(АБО), $\neg$(НІ).

В цьому випадку δ_f (R) є множина кортежів t, які належать R, таких, що при підстановці *i*–го компонента t замість будь-якого входження номера *i* в формулу F для всіх *i* вона стане істиною. Наприклад, $\delta_{2>3}$ (R) означає множину кортежів, що належать R, другий компонент яких більше третього компонента.

При реалізації селекції відношення, приведеного в табл.4.11, згідно з

Таблиця 4.11 - **Відношення**

ПІБ студента	Вік
Гаврилюк	21
Піддубчак	20
Скрипник	23
Ящук	19

ознакою (вік>20), отримуємо відношення (табл.4.12):

Таблиця 4.12 – **Відношення, над яким виконано операцію селекції**

ПІБ студента	Вік
Гаврилюк	21
Скрипник	23

Крім перерахованих операцій існують і інші, але їх можна отримати з п'яти основних.

Розглянемо неосновні операції перетину та ділення.

Перетин R∩S двох відношень R та S знаходиться згідно формул (R-(R-S)). Нехай R та S є відношення арності r і s відповідно, де (r > s) і $S \neq \{\otimes\}$. Тоді **частка** (R÷S) є множина кортежів *t* довжини (r-s) таких, що для всіх кортежів *u* довжини s, які належать S, кортеж *tu* належить R. Виконаємо операцію ділення над відношеннями, які представлені відповідно таблицями 4.13 та 4.14. В результаті отримуємо частку (табл.4.15).

Схематично операції реляційної алгебри представлені на рис.4.6.

Розроблені мови маніпулювання даними, що дозволяють реалізувати всі операції реляційної алгебри і практично будь-які їх сполучення. Серед

Таблиця 4.13 - **Екзаменаційна відомість**

Студенти	Дисципліна	Дата	Оцінка
Гаврилюк	Математика	10.03.2000	3
Гаврилюк	Фізика	15.03.2000	5
Піддубчак	Математика	10.03.2000	4
Піддубчак	Фізика	15.03.2000	5
Скрипник	Математика	10.03.2000	5
Скрипник	Фізика	15.03.2000	4
Ящук	Математика	10.03.2000	4
Ящук	Фізика	15.03.2000	5

Таблиця 4.14 – **Відношення**

Дисципліна	Оцінка
Математика	4
Фізика	5

Таблиця 4.15 – **Відношення – частка**

Студенти
Піддубчак
Яшук

них найбільше поширені SQL (Structured Query Language - структуризована мова запитів) і QBE (Quere-By-Example - запити за зразком) Обидві відносяться до мов дуже високого рівня, за допомогою яких користувач вказує, які дані необхідно одержати, не уточнюючи процедуру їхнього формування. За допомогою єдиного запиту на будь-якій із цих мов можна з'єднати декілька таблиць у тимчасову таблицю і вирізувати з неї необхідні рядки і стовпці (селекція і проекція).

Контрольні запитання

1. Скільки основних операції використовується в реляційній алгебрі?
2. Чи може відношення мати два однакових кортежі після виконання операцій проекції чи об'єднання?
3. Які основні операції використовуються при знаходженні частки?
4. Як змінюється потужність відношень при виконанні операцій реляційної алгебри?

4.3 Нормалізація схем баз даних

Нормалізація - це розбивка таблиці на дві або більше, які характеризуються кращими властивостями при доповненні, зміні і вилученні даних. Кінцева мета нормалізації зводиться до отримання такого проекту бази даних, у якому кожний факт з'являється лише в одному місці, тобто виключена надлишковість інформації. Це робиться не стільки з метою економії пам'яті, скільки для виключення можливої суперечливості збережених даних. [11,12,14]

Кожна таблиця в реляційній БД задовольняє умову, у відповідності з якою у позиції на перетині кожного рядка і стовпця таблиці завжди знаходиться єдине атомарне значення і ніколи не може бути множини таких значень. Будь-яка таблиця, що задовольняє цю умову, називається нормалізованою.

Кожній нормальній формі відповідає деякий визначений набір обмежень. Відношення знаходиться в деякій нормальній формі, якщо задовольняється властивий їй набір обмежень.

Кожна нормальна форма є більш обмеженою і більш бажаною, ніж попередня. Це пов'язано з тим, що в (N+1)-ій нормальній формі вилучаються деякі небажані властивості, які характерні N-ій нормальній формі. Теорія нормалізації грунтується на наявності тієї або іншої залежності між полями таблиці.

Основні властивості нормальних форм:

- кожна наступна нормальна форма в деякому змісті краща попередньої;
- при переході до наступної нормальної форми властивості попередніх нормальних властивостей зберігаються.

Найбільше важливі нормальні форми відношень грунтуються на фундаментальному в теорії реляційних баз даних понятті функціональної

залежності.

Визначення 1. Функціональна залежність.

У відношенні R атрибут Y функціонально залежить від атрибута X (X і Y можуть бути складовими) у тому і тільки в тому випадку, якщо кожному значенню X відповідає в точності одне значення Y: $X \rightarrow Y$.

Визначення 2. Повна функціональна залежність.

Функціональна залежність $X \rightarrow Y$ називається повною, якщо атрибут Y не залежить функціонально від будь-якої підмножини X.

Визначення 3. Транзитивна функціональна залежність .

Функціональна залежність називається транзитивною, якщо з функціональних залежностей $X \rightarrow Y$ та $Y \rightarrow Z$ випливає, що $X \rightarrow Z$.

Наприклад, Вінниця входить до Поділля, а Поділля - до України. Для даного прикладу має місце транзитивна залежність ВІННИЦЯ $\rightarrow$ УКРАЇНА.

Визначення 4. Неключовий атрибут .

Неключовим атрибутом називається будь-який атрибут відношення, що не входить до складу первинного ключа .

Визначення 5. Взаємно незалежні атрибути .

Два або більше атрибути взаємно незалежні, якщо жодний із цих атрибутів не є функціонально залежним від інших.

Відношення R задано в першій нормальній формі, якщо воно задано у вигляді множини своїх кортежів, які не повторюються.

Для того, щоб представити відношення в першій нормальній формі необхідно над його кортежами виконати операцію проекції для видалення рядків, які повторюються.

Відношення R задано в другій нормальній формі, якщо воно, по-перше, є відношенням у першій нормальній формі і, по-друге, кожний його атрибут, який не є основним атрибутом, функціонально повно залежить від будь-якого можливого ключа цього відношення.

Нехай задано відношення ФАКУЛЬТЕТ (НАЙМЕНУВАННЯ, ПІБ ДЕКАНА, ТЕЛЕФОН).

Відношення ФАКУЛЬТЕТ задано в другій нормальній формі, тому що у відношенні ФАКУЛЬТЕТ атрибут ТЕЛЕФОН, який не є основним, повністю залежить від будь-якого можливого ключа: НАЙМЕНУВАННЯ, ПІБ ДЕКАНА.

У загальному випадку, якщо всі можливі ключі відношення містять по одному атрибуту, то це відношення задане в другій нормальній формі, тому що в цьому випадку всі атрибути, які не є основними, функціонально повно залежать від можливих ключів. Однак це твердження не завжди справедливе, якщо ключ відношення R є складовим.

Розглянемо відношення:

СТУДЕНТ – КУРС_ПРОЕКТ (НОМЕР_ЗАЛІКОВОЇ_КНИЖКИ, КОД_ ПРЕДМЕТУ, ПРІЗВИЩЕ_СТУДЕНТА, НОМЕР_ГРУПИ, ВИКЛАДАЧ, ПРОЦЕНТ_ВИКОНАННЯ).

Припустимо, що в одній групі можуть навчаються однофамільці. Тоді для цього відношення можливий тільки один ключ: НОМЕР_ЗАЛІКОВОЇ КНИЖКИ, КОД_ПРЕДМЕТУ. Виходячи з прийнятого припущення, атрибут ПРІЗВИЩЕ_СТУДЕНТА не входить у ключ. Тоді атрибут НОМЕР_ ЗАЛІКОВОЇ_КНИЖКИ не визначається значенням атрибута ПРІЗВИЩЕ_ СТУДЕНТА, тобто атрибути ПРІЗВИЩЕ_СТУДЕНТА і НОМЕР_ГРУПИ не є основними, але функціонально залежать від основного атрибута НОМЕР_ЗАЛІКОВОЇ_КНИЖКИ, що входить у складовий ключ. Функціональні залежності між атрибутами цього відношення показані на рис. 4.7.

Розщепивши вихідне відношення на два нових у другій нормальній формі, можна усунути надлишковість (рис.4.8). При виконанні цієї операції розбивки на два відношення враховано те, що атрибути, які функціонально залежать від одного основного атрибута разом із ним

утворять одне відношення з єдиним ключем НОМЕР_ЗАЛІКОВОЇ_КНИЖКИ, а інші атрибути, які функціонально повно залежать від складового ключа, залишено у вихідній схемі.

Розглянемо приклад схеми відношення:

СПІВРОБІТНИКИ - ВІДДІЛИ - ПРОЕКТИ

(СПІВРОБ_НОМЕР,СПІВРОБ_ЗАРП, ВІДДІЛ_НОМЕР, ПРО_НОМЕР, СПІВРОБ_ЗАВДАННЯ).

У відношенні використані скорочення: СПІВРОБ - співробітник, ЗАРП - зарплата, ПРО - проект.

Первинний ключ:

СПІВРОБ_НОМЕР, ПРО_НОМЕР.

Функціональні залежності:

СПІВРОБ_НОМЕР -> СПІВРОБ_ЗАРП

СПІВРОБ_НОМЕР -> ВІДДІЛ_НОМЕР

ВІДДІЛ_НОМЕР -> СПІВРОБ_ЗАРП

СПІВРОБ_НОМЕР, ПРО_НОМЕР -> СПІВРОБ_ЗАВДАННЯ.

Хоча первинним ключем є складовий атрибут СПІВРОБ_НОМЕР, ПРО_НОМЕР, атрибути СПІВРОБ_ЗАРП і ВІДДІЛ_НОМЕР функціонально залежать від частини первинного ключа, тобто атрибута СПІВРОБ_ НОМЕР. В результаті, неможливо вставити у відношення СПІВРОБІТ- НИКИ-ВІДДІЛИ-ПРОЕКТИ кортеж, що описує співробітника, який ще не виконує ніякого проекту (первинний ключ не може містити невизначене значення). При видаленні кортежу не тільки руйнується зв'язок даного співробітника з даним проектом, але втрачається інформація про те, в якому відділі він працює. При переводі співробітника в інший відділ необхідно модифікувати всі кортежі, які описують цього співробітника, або одержимо неузгоджений результат. Такі неприємні явища називаються аномаліями схеми відношення. Вони усуваються шляхом нормалізації.

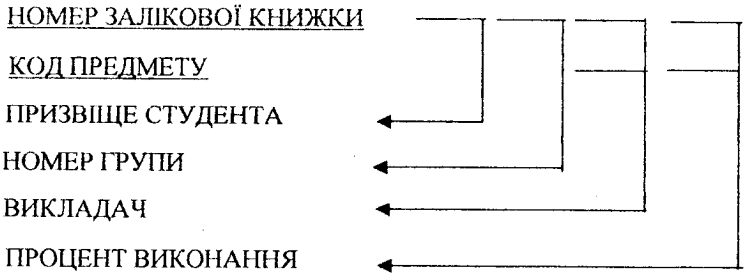


Рисунок 4.7 – Функціональна залежність

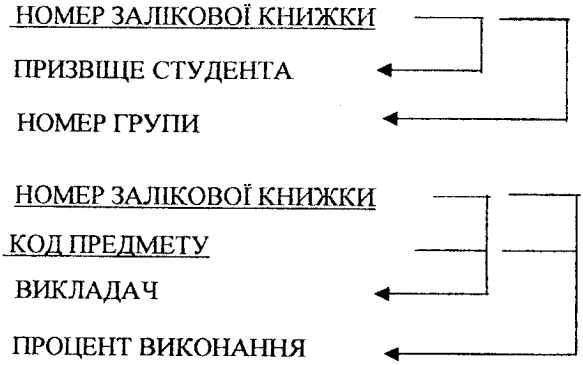


Рисунок 4.8 – Приклад усунення функціональної залежності

Виконаємо декомпозицію відношення СПІВРОБІТНИКИ-ВІДДІЛИ в два відношення СПІВРОБІТНИКИ-ВІДДІЛИ і СПІВРОБІТНИКИ-ПРОЕКТИ:

СПІВРОБІТНИКИ-ВІДДІЛИ (СПІВРОБ_НОМЕР, СПІВРОБ_ЗАРП, ВІДДІЛ_НОМЕР)

Первинний ключ:

СПІВРОБ_НОМЕР

Функціональні залежності:

СПІВРОБ_НОМЕР -> СПІВРОБ_ЗАРП

СПІВРОБ НОМЕР -> ВІДДІЛ НОМЕР

ВІДДІЛ НОМЕР -> СПІВРОБ ЗАРП

СПІВРОБІТНИКИ-ПРОЕКТИ (СПІВРОБ_НОМЕР, ПРО_НОМЕР, СПІВРОБ_ЗАВДАННЯ)

Первинний ключ:

СПІВРОБ_НОМЕР, ПРО_НОМЕР

Функціональна залежність:

СПІВРОБ_НОМЕР, ПРО_НОМЕР -> СПІВРОБ_ЗАВДАННЯ

Кожне з цих двох відношень знаходиться в 2НФ і в них усунуті відзначені вище аномалії .

Відношення R знаходиться в третій нормальній формі (3НФ) у тому і тільки в тому випадку, якщо знаходиться в 2НФ і кожний неключовий атрибут нетранзитивно залежить від первинного ключа.

Наприклад, відношення:

ГУРТОЖИТОК (ПІБ_СТУДЕНТА, НОМЕР_ГРУПИ, НОМЕР_КІМНАТИ, СТАРОСТА_КІМНАТИ) знаходиться в другій нормальній формі, але не в третій, тому що атрибут СТАРОСТА_КІМНАТИ залежить від атрибута НОМЕР_КІМНАТИ, який у свою чергу залежить від атрибута ПІБ_СТУДЕНТА і, отже, СТАРОСТА_КІМНАТИ транзитивно залежить від ПІБ_СТУДЕНТА. Це відношення можна привести до необхідної форми шляхом його розщеплення на два:

СТУДЕНТ-ГУРТОЖИТОК (ПІБ_СТУДЕНТА, НОМЕР_ГРУПИ, НОМЕР_КІМНАТИ) та КІМНАТА-ГУРТОЖИТОК (НОМЕР_КІМНАТИ, СТАРОСТА_КІМНАТИ).

Залежності між атрибутами вихідного й отриманих відношень подані на рис.4.9, звідки видно, що отримані відношення більш доцільніші від вихідного. Так, інформація про старосту кімнати може знадобитися незалежно від інформації про студентів, що проживають у цій кімнаті.

Нехай маємо відношення (ФІРМА, СКЛАД, ОБ'ЄМ).

Для даного відношення характерні такі аномалії:

1. Якщо в даний момент відсутня фірма, яка отримує товар зі складу, то в базу даних неможливо ввести інформацію про об'єм складу.
2. Якщо фірма перестає отримувати товар зі складу, то данні про склад та його об'єм не можна зберігати в базі даних.
3. Якщо об'єм складу змінився, то необхідно переглянути всі рядки відношення і змінити кортежі для форм, пов'язаних зі складом.

Причиною аномалій для даного відношення є наявність транзитивного зв'язку між атрибутами.

Для усунення аномалій розіб'ємо вихідне відношення на два: ЗБЕРІГАННЯ (ФІРМА, СКЛАД) та ОБ'ЄМ (СКЛАД, ОБ'ЄМ).

На практиці в більшості випадків три нормальні форми схем відношень є достатніми і приведенням до третьої нормальної форми процес проектування реляційної бази даних, як правило, закінчується. Однак іноді корисно продовжити процес нормалізації.

Розглянемо приклад схеми відношення:

СПІВРОБІТНИКИ-ПРОЕКТИ (СПІВРОБІТНИКА_НОМЕР, СПІВРОБІТНИКА_ПРІЗВИЩЕ, ПРОЕКТУ_НОМЕР, СПІВРОБІТНИКА_ЗАВДАННЯ).

Можливі ключі (важливо, що на цій стадії нормалізації до уваги приймається існування можливих ключів):

СПІВРОБІТНИКА_НОМЕР, ПРОЕКТУ_НОМЕР;

СПІВРОБІТНИКА_ПРІЗВИЩЕ, ПРОЕКТУ_НОМЕР.

Функціональні залежності:

СПІВРОБІТНИКА_НОМЕР -> СПІВРОБІТНИКА_ПРІЗВИЩЕ;

СПІВРОБІТНИКА_НОМЕР -> ПРОЕКТУ_НОМЕР;

СПІВРОБІТНИКА_ПРІЗВИЩЕ -> СПІВРОБІТНИКА_НОМЕР;

СПІВРОБІТНИКА_ПРІЗВИЩЕ -> ПРОЕКТУ_НОМЕР;

СПІВРОБІТНИКА_НОМЕР, ПРОЕКТУ_НОМЕР -> СПІВРОБІТНИКА_ ЗАВДАННЯ;

СПІВРОБІТНИКА_ПРІЗВИЩЕ, ПРОЕКТУ_НОМЕР -> СПІВРОБІТНИ - КА_ ЗАВДАННЯ.

Рисунок 4.9 – Приклад усунення транзитивної залежності

У цьому прикладі припускаємо, що особа співробітника повністю визначається як його номером, так і прізвищем.

Незалежно від того, який із можливих ключів обраний в якості первинного ключа, ця схема знаходиться в ЗНФ. Однак той факт, що є функціональні залежності атрибутів відношення від атрибута, що є частиною первинного ключа, приводить до аномалій. Наприклад, для того, щоб змінити ПРІЗВИЩЕ співробітника з даним номером погодженим способом, буде потрібно модифікувати всі кортежі, які включають його номер. Введемо визначення.

Детермінантом називається будь-який атрибут, від котрого функціонально повно залежить деякий інший атрибут.

Нормальна форма Бойса-Кодда. Відношення R знаходиться в нормальній формі Бойса-Кодда (БКНФ) у тому і тільки в тому випадку, якщо кожний детермінант є можливим ключем.

Зауважимо, що якщо у відношенні є тільки один можливий ключ (який є первинним ключем), то це визначення стає сквівалентним визначенню третьої нормальної форми.

Очевидно, що ця вимога не виконана для відношення СПІВРОБІТНИКИ-ПРОЕКТИ. Можна виконати його декомпозицію до відношень СПІВРОБІТНИКИ і СПІВРОБІТНИКИ-ПРОЕКТИ:

СПІВРОБІТНИКИ (СПІВРОБІТНИКА_НОМЕР, СПІВРОБІТНИКА_ ПРІЗВИЩЕ) .

Можливі ключі:

СПІВРОБІТНИКА_НОМЕР;

СПІВРОБІТНИКА_ПРІЗВИЩЕ.

Функціональні залежності:

СПІВРОБІТНИКА_НОМЕР -> СПІВРОБІТНИКА_ПРІЗВИЩЕ;

СПІВРОБІТНИКА_ПРІЗВИЩЕ -> СПІВРОБІТНИКА_НОМЕР.

СПІВРОБІТНИКИ-ПРОЕКТИ (СПІВРОБІТНИКА_НОМЕР, ПРОЕКТУ_ НОМЕР, СПІВРОБІТНИКА_ЗАВДАННЯ).

Можливий ключ:

СПІВРОБІТНИКА_НОМЕР, ПРОЕКТУ_НОМЕР.

Функціональні залежності:

СПІВРОБІТНИКА_НОМЕР, ПРОЕКТУ_НОМЕР -> СПІВРОБІТНИКА_ ЗАВДАННЯ.

Можлива альтернативна декомпозиція, якщо вибрати за основу СПІВРОБІТНИКА_ПРІЗВИЩЕ. В обох випадках отримані відношення СПІВРОБІТНИКИ і СПІВРОБІТНИКИ-ПРОЕКТИ знаходяться в БКНФ, і їм не властиві відзначені аномалії.

Розглянемо відношення *R* (МІСТО, АДРЕСА, ІНДЕКС). Атрибут ІНДЕКС визначає індекс відділення зв'язку, яке обслуговує адресатів деякої вулиці міста, АДРЕСА - назву вулиці і номеру будинку. При цьому будемо припускати, що кортеж *(C, S, Z)* належить деякому відношенню зі схемою відношення R, якщо тільки в місті *C* є будинок за адресою S і Z є відповідним поштовим індексом. У цьому випадку мають місце такі функціональні залежності:

МІСТО, АДРЕСА → ІНДЕКС;

ІНДЕКС → МІСТО.

Іншими словами, повна адреса (назва міста і адреса в місті) визначає поштовий індекс, а поштовий індекс, у свою чергу, визначає назву міста, але не визначає адресу, тому що одне відділення зв'язку обслуговує багато будинків на різних вулицях. Таким чином, в якості основного ключа можна вибрати одне з двох множин атрибутів:

МІСТО, АДРЕСА і АДРЕСА, ІНДЕКС .

Схема відношення *R* (МІСТО, АДРЕСА, ІНДЕКС) не знаходиться в нормальній формі Бойса-Кодда, так як має місце залежність ІНДЕКС→МІСТО. Декомпозицією відношення його можна привести до нормальної форми Бойса-Кодда.

Розглянемо приклад схеми відношення:

ПРОЕКТИ (ПРОЕКТУ_НОМЕР, ПРОЕКТУ_СПІВРОБ, ПРОЕКТУ_ЗАВДАННЯ) .

Відношення ПРОЕКТИ містить номера проектів, кожний проект - список співробітників, які можуть виконувати проект, і список завдань, які передбачаються проектом. Співробітники можуть брати участь у декількох проектах, і різні проекти можуть включати однакові завдання.

Кожний кортеж відношення зв'язує деякий проект із співробітником, які беруть участь у цьому проекті, і з завданням, яке співробітник виконує в рамках даного проекту (припускаємо, що будь-який співробітник, який

бере участь у проекті, виконує всі завдання, передбачені цим проектом). Через сформульовані вище умови єдиним можливим ключем відношення є складовий атрибут ПРОЕКТ_НОМЕР, ПРОЕКТ_СПІВРОБ, ПРОЕКТ_ЗАВДАННЯ, і немає ніяких інших детермінантів. Отже, відношення ПРОЕКТИ знаходиться в БКНФ. Але при цьому воно має аномалії: якщо, наприклад, деякий співробітник приєднується до даного проекту, необхідно вставити у відношення ПРОЕКТИ стільки кортежів, скільки завдань у ньому передбачено.

У відношенні R (A, B, C) існує багатозначна залежність (multi-valued dependence - MVD) (R. A $\rightarrow\!\!\rightarrow$ R. B) в тому і тільки в тому випадку, якщо множина значень B, що відповідає парі значень A і C, залежить тільки від A і не залежить від C.

У відношенні ПРОЕКТИ існують дві багатозначні залежності:

ПРОЕКТ_НОМЕР $\rightarrow\!\!\rightarrow$ ПРОЕКТ_СПІВРОБ;

ПРОЕКТ_НОМЕР $\rightarrow\!\!\rightarrow$ ПРОЕКТУ_ЗАВДАННЯ.

Неважко показати, що в загальному випадку у відношенні R (A, B, C) існує багатозначна залежність A $\rightarrow\!\!\rightarrow$B у тому і тільки в тому випадку, коли існує багатозначна залежність A $\rightarrow\!\!\rightarrow$ C.

Відношення R знаходиться в четвертій нормальній формі (4НФ) у тому і тільки в тому випадку, якщо у випадку існування багатозначної залежності A$\rightarrow\!\!\rightarrow$B всі інші атрибуги R функціонально залежать від A.

У нашому прикладі можна виконати декомпозицію відношення ПРОЕКТИ на два відношення ПРОЕКТИ-СПІВРОБІТ і ПРОЕКТИ-ЗАВДАННЯ:

ПРОЕКТИ-СПІВРОБІТ (ПРОЕКТ_НОМЕР, ПРОЕКТ_СПІВРОБІТ);

ПРОЕКТИ-ЗАВДАННЯ (ПРОЕКТ_НОМЕР, ПРОЕКТ_ЗАВДАННЯ).

Обидва ці відношення знаходяться в 4НФ .

Розглянемо ще один приклад. Нехай задано відношення R(СТУДЕНТ, ТОВАРИСТВО, СУСПІЛЬНА_РОБОТА, РІК).

Атрибут ТОВАРИСТВО визначає назву товариств, членом яких є студент; атрибути СУСПІЛЬНА_РОБОТА і РІК - найменування суспільних доручень, виконуваних студентом, і рік їх призначення. Передбачається, що те саме суспільне навантаження не може бути призначене двічі протягом одного року тому ж самому студенту, але заміна одного навантаження на інше протягом року допускається. У табл. 4.16 приведений фрагмент відношення.

Таблиця 4.16 – **Приклад відношення з нетривіальними залежностями**

Студент	Член товариства	Суспільна робота	Рік
Іванов	ВТВР	Староста групи	1999
Іванов	ДТСААФ	Староста групи	1999
Петров	ВТВР	Член інформаційного Прожектору	1999
Петров	ДТСААФ	Член інформаційного "Прожектору"	1999
Петров	ВТВР	Профорг групи	2000
Петров	ДТСААФ	Профорг групи	2000

Виділимо нетривіальні багатозначні залежності:

СТУДЕНТ→→ ТОВАРИСТВО;

СТУДЕНТ →→(СУСПІЛЬНА_РОБОТА, РІК).

Вважаємо, що атрибут ТОВАРИСТВО не залежить від того, яку суспільну роботу веде студент. Атрибути СУСПІЛЬНА_РОБОТА і РІК взаємозалежні.

Наявність подібних нетривіальних багатозначних залежностей у схемі одного відношення і незалежність їхніх правих частин в остаточному підсумку приводять до комбінації значень правих частин, що ілюструється в табл. 4.16. Відомості про те, що студент Іванов є старостою групи у 1999 р. повторюються двічі в силу того, що він є членом двох товариств (ВТВР,

ДТСААФ). Для студента Петрова в зв'язку зі зміною суспільної роботи (1999 - 2000 р.) доводиться вводити додаткові кортежі, у яких буде повторюватися інформація про членство студента в товариствах ВТВР і ДТСААФ. Незважаючи на надлишковість відношення *R*, що виникає при цьому воно подано в третій нормальній формі, тому що в цьому відношенні відсутні функціональні залежності. З метою усунення надлишковості у відношенні, поданій в третій нормальній формі, необхідно виконати розкладання по багатозначній залежності даного відношення (див.табл.4.17, табл.4.18).

В усіх розглянутих до цього часу нормалізаціях здійснювалась декомпозиція одного відношення на два. Іноді це зробити не вдається, але можлива декомпозиція на більше число відношень, кожне з який має кращі властивості.

Таблиця 4.17 – **Приклад відношення в четвертій нормальній формі**

Студент	Товариство
Іванов	ВТЗВ
Іванов	ДТСААФ
Петров	ВТРВ
Петров	ДТСААФ

Таблиця 4.18 – **Приклад відношення в четвертій нормальній формі**

Студент	Суспільна робота	Рік
Іванов	Староста групи	1999
Петров	Член інформаційного прожектору	1999
Петров	Профорг групи	2000

Розглянемо відношення СПІВРОБІТНИКИ-ВІДДІЛИ-ПРОЕКТИ (СПІВРОБ_НОМЕР, ВІДДІЛУ_НОМЕР, ПРОЕКТУ_НОМЕР).

Припустимо, що той самий співробітник може працювати в декількох відділах і в кожному відділі брати участь у декількох проектах. Первинним ключем цього відношення є повна сукупність його атрибутів. Відсутні функціональні і багатозначні залежності. Тому відношення знаходиться в 4НФ. Однак у ньому можуть існувати аномалії, які можна усунути шляхом декомпозиції на три відношення.

Відношення R (X, Y, ... , Z) задовольняє залежності з'єднання * (X, Y, ... , Z) у тому і тільки в тому випадку, коли R відновлюється без втрат шляхом з'єднання своїх проекцій на X, Y, ... , Z.

Відношення R знаходиться в п'ятій нормальній формі у тому і тільки в тому випадку, коли будь-яка залежність з'єднання в R випливає з існування деякого можливого ключа в R.

Введемо наступні імена складових атрибутів:

З = {СПІВРОБ_НОМЕР, ВІДДІЛУ_НОМЕР};

СП = { СПІВРОБ _НОМЕР, ПРОЕКТУ_НОМЕР};

ВП = {ВІДДІЛУ_НОМЕР, ПРОЕКТУ_НОМЕР}.

Припустимо, що у відношенні СПІВРОБІТНИКИ-ВІДДІЛИ-ПРОЕКТИ існує залежність з'єднання: * (З, СП, ОП)

На прикладах можна легко показати, що при вставках і видаленнях кортежів можуть виникнути проблема. Їх можна усунути шляхом декомпозиції вихідного відношення на три нових відношення:

СПІВРОБІТНИКИ-ВІДДІЛИ (СПІВРОБ_НОМЕР, ВІДДІЛУ_НОМЕР),

СПІВРОБІТНИКИ-ПРОЕКТИ (СПІВРОБ_ НОМЕР, ПРОЕКТУ_НОМЕР),

ВІДДІЛИ-ПРОЕКТИ (ВІДДІЛУ НОМЕР, ПРОЕКТУ НОМЕР).

П'ята нормальна форма - це остання нормальна форма, яку можна одержати шляхом декомпозиції.

На закінчення приведемо послідовність етапів нормалізації:

1. Перехід від структурної моделі даних до плоских двовимірних відношень (таблицям).
2. Усунення всіх неповних залежностей атрибутів, які не є основними, від усіх ймовірних ключів.
3. Усунення всіх транзитивних залежностей атрибутів, які не є основними, від усіх ймовірних ключів.
4. Усунення всіх нетривіальних багатозначних залежностей атрибутів, які не є основними, від усіх ймовірних ключів.

Після того як визначені елементи даних і залежності між ними, ці етапи в принципі можуть бути виконані автоматично за приведеним алгоритмом.

Контрольні запитання

1. Для чого виконується нормалізація відношень?
2. Які залежності між атрибутами називають транзитивними та функціонально повними?
3. Приведіть приклади відношень в різних нормальних формах.
4. Приведіть послідовність етапів нормалізації.

5 Фізична організація баз даних

5.1 Спискові структури

У системах обробки даних у якості даних виступають описи фактів і понять предметної області на точній і формалізованій вхідній мові системи - мові опису даних. За допомогою вхідної мови при описі фактів і понять предметної області між елементами даними формуються логічні структурні відношення. У якості логічних структур використовуються або таблиці, що зображуються як двовимірний або *n*-мірний масив даних, або деревовидні ієрархічні структури, або мережні структури, що зображають складну багатозв'язну структуру з великою кількістю взаємних з'єднань і т.п. Щоб правильно використовувати обчислювальну машину, необхідно добре представляти собі структурні відношення між даними, знати засоби представлення таких структур у пам'яті машини і методи роботи з ними. Структура даних типу «дерево» може бути подана в пам'яті ЕОМ декількома різноманітними способами. [14]

Таким чином, будь-яке представлення структури даних у пам'яті ЕОМ повинно містити в собі як самі дані, так і задані взаємозв'язки, що і визначають структурування.

Форма представлення структур даних у пам'яті ЕОМ залежить від очікуваного використання даних, оскільки для різноманітних типів структур ефективність виконання тих або інших операцій опрацювання даних різноманітна. Основна розбіжність форм уявлення структур даних у пам'яті ЕОМ визначається в першу чергу тим, як адресуються елементи структури даних у пам'яті машини - по місцю або по вмісту. У першому випадку вказуються логічні або фізичні адреси даних, визначальне місце розташування даних і їхня вибірка здійснюються по відомому значенню ключа, тобто визначаються вмістом самих даних. Цей випадок реалізується в спеціальній - асоціативній пам'яті ЕОМ. Деякий аналог асоціативної

пам'яті може бути реалізований засобами спеціального програмного забезпечення в звичайній пам'яті ЕОМ.

Найбільш простою формою збереження даних у пам'яті ЕОМ є одномірний лінійний список. *Лінійний список* - це множина $n \geq 0$ об'єктів (вузлів) X[1], X[2], ..., X[n], структурні властивості якого пов'язані лише з лінійним розташуванням вузлів. Якщо $n > 0$, те X[1] є першим вузлом; для $1<i<n$ вузол X[i-1] передує вузлу X[i], а вузол X[i+1] іде за ним, X[n] є останнім вузлом, тобто лінійний список реалізує структуру, що можна визначити як лінійне впорядкування елементів даних.

Лінійний список X розглядають як послідовність X[1], X[2], ..., X[i], ..., X[n], компоненти якої ідентифіковані порядковим номером, що вказує їхнє відносне розташування в X.

Одномірний лінійний список, що використовується для збереження даних у пам'яті машини, називають ще вектором даних або фізичною структурою збереження даних. Використання лінійного списку в якості фізичної структури збереження даних визначається властивостями пам'яті обчислювальної машини.

Проблема представлення логічних структур даних у пам'яті ЕОМ полягає в знаходженні ефективних методів відображення логічної структури даних на фізичну структуру збереження. Таке відображення називають *адресною функцією.*

При реалізації адресної функції використовують два основних методи: послідовний розподіл пам'яті; зв'язний розподіл пам'яті.

Послідовний розподіл пам'яті. Послідовний розподіл - простий і природний засіб збереження лінійного списку. У цьому випадку вузли списку розміщаються в послідовних елементах пам'яті (рис.5.1).

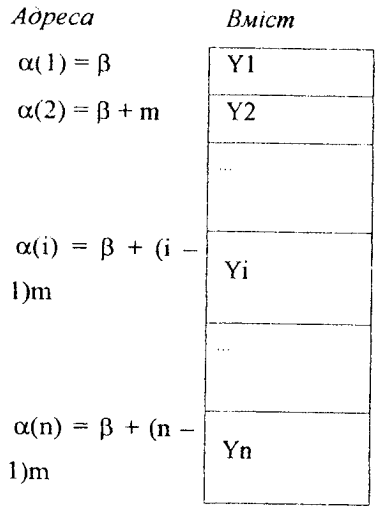


Рис. 5.1 Приклад послідовного розподілу пам'яті для представлення лінійного списку

При послідовному розподілі вектор даних логічно відділений від опису структури збережених даних. Наприклад, якщо структура даних являє собою лінійний список (наприклад, файл записів фіксованої довжини), то опис структури зберігається в окремому записі і містить:
а) n - розмір вектора даних, тобто кількість елементів списків-записів;
б) m - розмір елемента списку, тобто розмір запису, наприклад у байтах; в) β - адреса бази, що вказує на початок вектора даних у пам'яті.

У цьому випадку адреса кожного запису можна обчислити за допомогою адресної функції, що відображає логічний індекс, що ідентифікує запис у структурі, на адресу фізичної пам'яті
$\alpha(i) = \beta + (i - 1)m$.

У випадку лінійного списку адресна функція складається з операцій зсуву і маштабування. Будь-які відношення, що можна висловити на мові

цілих чисел, можна витлумачити як відношення між елементами пам'яті, одержуючи при цьому усілякі варіанти структур.

Як приклад розглянемо реалізацію за допомогою лінійного списку при послідовному розподілі пам'яті для логічної структури типу регулярного двійкового дерева (рис 5.2).

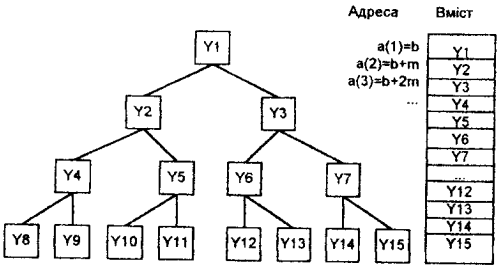


Рис.5.2 Приклад реалізації структури типу регулярного двійкового дерева за допомогою лінійного списку

Ідея способу полягає в тому, що, починаючи з елемента пам'яті a(1), роблять його коренем дерева, розміщають там дані, що відповідають вузлу Y1. У елементах пам'яті a(2) і a(3) розміщають безпосередніх нащадків вузла Y1 - Y2, Y3 і т.д. У загальному випадку, безпосередні нащадки вузла Yк розміщаються по адресах: a(2к) і a(2к+1). Адресна функція має вид

$a(к)=b+(к-1)*m$,

де k - номер вузла деревоподібної структури; b - базова адреса; m - розмір елемента пам'яті, що потрібно для збереження даних вузлів дерева.

По дереву, що при цьому утворюється, можна рухатися в обох напрямках, тому що від вузла Y_k можна перейти до його нащадків, подвоївши k (або подвоївши k і збільшивши на одиницю). Можна рухатися до вузла, що є вихідним для вузла Y_k, розділивши k навпіл і відкинувши дробову частину. Адреса відповідного вузла елемента пам'яті визначається по адресній функції.

Розглянемо ще один спосіб реалізації, що використовується тільки для двійкових дерев. Якщо для уявлення двійкового дерева використовується вектор пам'яті від елемента i до елемента j включно, то корінь дерева розміщається в елементі пам'яті з адресою

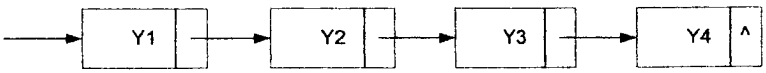
$$\gamma = \lfloor (i+j)/2 \rfloor,$$

де $\lfloor\ \rfloor$ - знак округлення до найближчого меншого цілого.

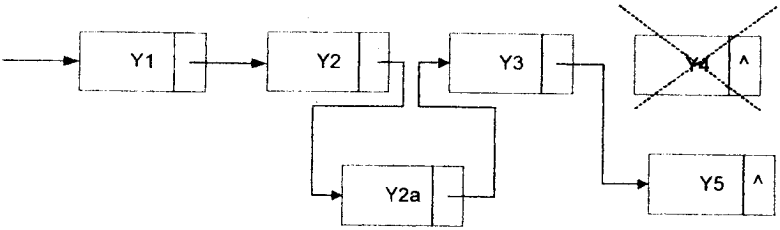
Корінь дерева розміщається в середину вектора. У елементах пам'яті від i -го до (m-1)-го включно розміщається ліве піддерево. У елементах пам'яті те (m+1)-го до j -го включно розміщається праве піддерево. Аналогічно процес повторюється для розміщення кожного піддерева. Приведений засіб дозволяє реалізувати двійкове збалансоване дерево.

Зв'язаний розподіл пам'яті. Зв'язане представлення лінійного списку називається *зв'язаним списком*. При зв'язаному розподілі пам'яті для побудови структури необхідно задати відношення прямування і передування елементів за допомогою покажчиків. Покажчиками служать адреси, збережені в записах даних. На відміну від послідовного розподілу пам'яті, при якому за допомогою адресної функції обчислюється адреса наступного елемента, при зв'язному розподілі пам'яті значення адресної функції можна одержати тільки шляхом перегляду покажчиків, що зберігаються.

Зв'язаний розподіл - більш складний, але і більш гнучкий спосіб збереження лінійного списку. Кожний вузол містить покажчик на наступний вузол списку, тобто адреса наступного вузла списку (рис.5.3)



Мал.5.3.а Приклад зв'язного лінійного списка

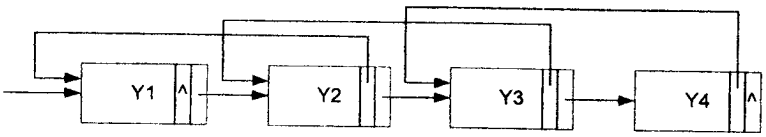


Мал.5.3.6 Приклади видалення і додавання елементів списка

При зв'язаному розподілі не потрібно, щоб список зберігався в послідовних елементах пам'яті. Наявність адрес зв'язку в даному способі збереження дозволяє розміщати вузли списку довільно в будь-якій вільній ділянці пам'яті. При цьому лінійна структура списку забезпечується покажчиками.

Структура лінійного списку, подана за допомогою зв'язаного розподілу, називається також ланцюговою структурою або ланцюгом.

Для досягнення більшої гнучкості при роботі з лінійними списками в кожний вузол X[i] вводиться два покажчики. Один із покажчиків реалізує зв'язок вузла, що розглядається з вузлом X[i+1], а інший - із вузлом X[i-1] (рис.5.4).

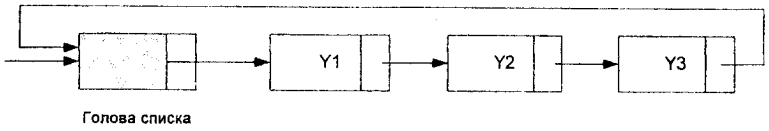


Мал.5.4 Приклад двонаправленого лінійного списка

Для зв'язаних однонаправленого або двонаправленого списків у ряді випадків доцільно створити спеціальний вузол списку - голову списку - і берегти його в спеціальному фіксованому осередку пам'яті машини за адресою β.

У цей вузол поміщається покажчик на перший елемент списку. У голові списку можна берегти різноманітну інформацію, необхідну при обробці списку (ідентифікатор списку, кількість вузлів у списку і т.п.).

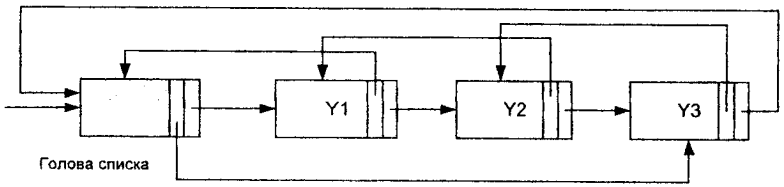
Важливим різновидом представлення в пам'яті лінійного списку є *циклічний список*. Циклічно зв'язаний лінійний список характеризується тією особливістю, що зв'язок від останнього вузла йде до першого вузла списку (рис.5.5).



Мал.5.5 Приклад однонаправленого лінійного списка

Циклічний список дозволяє одержати доступ до будь-якого вузла списку, відправляючись від будь-якого заданого вузла. Циклічні списки називаються також кільцевими структурами або кільцями.

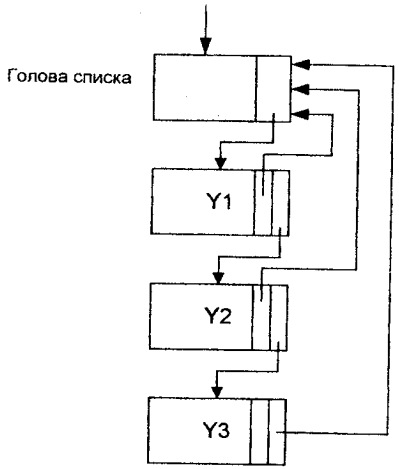
Поряд з однонаправленими використовуються двонаправлені циклічні списки (рис.5.6).



Мал.5.6 Приклад двонаправленого циклічного списка

У ряді випадків зручно використовувати циклічний список із покажчиками на голову списку з кожного вузла (рис.5.7), за винятком

останнього вузла - оскільки використовується прямий покажчик на голову списку.



Мал.5.7 Приклад однонаправленого циклічного списка із вказівниками на голову списка

З погляду організації структури даних розрізняють два типи покажчиків: вбудовані покажчики і довідник покажчиків. Якщо покажчики утворюють частину запису, то вони називаються *вбудованими*. Якщо покажчики зберігаються окремо від записів, то вони утворюють *довідник*.

Покажчики мають наступні можливі шляхи використання:

- визначають напрямок доступу (можна рухатися тільки в тих напрямках, що задані покажчиками);
- з'єднують разом зв'язані за змістом дані;
- відображають орієнтовані ребра в деревовидних або мережних структурах;
- зв'язують пам'ять на дисках і організують ланцюжки дискових сторінок і т.п.

Застосування багатозв'язних списків - це основний механізм, що дозволяє розроблювачам СУБД реалізувати складні нелінійні структури. Проте варто уникати занадто великої кількості покажчиків, оскільки на них

витрачається пам'ять і час на переходи по покажчиках. Крім того, при великій кількості покажчиків основна структура втрачає чіткість і можуть виникнути зв'язки, що у відображеній структурі відсутні.

Контрольні питання

1. Чим визначається форма представлення даних в ЕОМ ?
2. Які основні недоліки характерні лінійному списку ?
3. Поясніть призначення адресних покажчиків в циклічному списку.
4. В чому полягає різниця між вбудованими покажчиками і довідником ?

5.2 Нелінійні зв'язкові структури

Однозв'язний список завжди лінійний. Двозв'язний список може і не бути лінійним, якщо другий покажчик кожного елемента списку задає порядок довільного виду, який не є зворотним стосовно порядку, встановлюваному першим покажчиком елемента. Кожний елемент такого узагальненого двозв'язного списку міститься одночасно в двох різних однозв'язних списках, як показано на рис. 5.8. [14,15]

На цьому рисунку змінні S1 і S2 є покажчиками початку двох різних однозв'язних списків, у які одночасно входить кожний із п'яти елементів, а P1 і P2- позначення зв'язків у першому і другому однозв'язних списках відповідно. Покажчики S1 і S2 є компонентами двох різних дескрипторів однозв'язних списків.

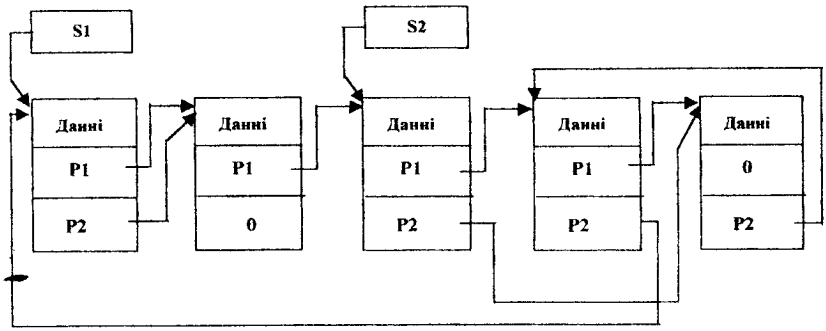


Рисунок 5.8 - Приклад логічної структури узагальнюючого двозв'язного списку

У ще більш загальному випадку кожний елемент зв'язкового списку може містити довільне кінцеве число зв'язків , однакове або різне в різних елементах. У результаті такого узагальнення утворюється багатозв'язний список, кожний елемент якого входить одночасно в стільки різних однозв'язних списків, скільки є зв'язків у відповідному елементі. При цьому не обов'язково, щоб кожний елемент неодмінно входив у всі однозв'язні списки. Такий багатозв'язний список як би прошитий в різних напрямках багатьма зв'язками. Тому багатозв'язні списки іноді називають *прошитими* списками. Використовується також ще одна назва багатозв'язних списків - *плекси* .

Проілюструємо сутність багатозв'язного списку на прикладі організації даних про абітурієнтів вузу під час здачі вступних іспитів. Припустимо, що крім усіх відомостей про кожного студента інтерес представляє і те, які студенти з загального списку характеризуються визначеними даними. Наприклад, приймальну комісію вузу можуть цікавити крім загальних відомостей про кожного абітурієнта часткові дані:

1. Абітурієнти, що проживають у Вінниці та Вінницькій області.
2. Абітурієнти медалісти.
3. Абітурієнти, що здали вступні іспити на «5».

У цьому випадку дані про абітурієнтів можна організувати у виді чотиризв'язного списку, кожний елемент якого містить усі необхідні відомості про якого-небудь одного студента, а також чотири покажчики. Перший покажчик необхідний для зв'язування елементів у загальний список відомостей про всіх студентів, другий – для зв'язування в один список тих елементів, у яких містяться відомості про абітурієнтів із Вінниці і Вінницької області. Третій покажчик адресує список абітурієнтів-медалістів, а четвертий - список абітурієнтів, що здали

вступні іспити на «5». Для кожного з цих чотирьох списків повинний бути окремий дескриптор. У загальному випадку кожний елемент чотиризв'язного списку входить не в усі однозв'язні списки одночасно. Цей приклад ілюструється на рис. 5.9 , причому в кожному дескрипторі для спрощення приведені лише поле покажчика початку, розмір відповідного

списку і номер покажчика, який використовується в елементах багатозв'язного списку для утворення відповідному даному дескриптору однозв'язного списку. З рисунка видно, що загальний список абітурієнтів містить 100 чоловік, список абітурієнтів із Вінниці і Вінницької області-34 чоловік, список абітурієнті - медалістів - 12 чоловік, а список абитурієнтів-відмінників-15 чоловік. З прикладу видно також, що кожний елемент багатозв'язного списку містить у точності чотири поля покажчиків, хоча в деяких елементах використовуються не всі чотири поля. Така організація структури елемента—не є єдиною можливою. Наприклад, шляхом ускладнення алгоритму обробки списку можна відводити в кожному елементі стільки полів покажчиків, скільки фактично використовується.

Найбільш загальний вид багатозв'язної структури - багатозв'язна структура , яка характеризується наступними властивостями.

1. Кожний елемент структури включає довільну кількість однонаправлених зв'язків з другими елементами.
2. З кожним елементом може зв'язуватися довільна кількість інших елементів.
3. Кожний зв'язок в структурі має не тільки напрямок , але і вагу.

При обробці багатозв'язних структур важливе місце має процедура виключення вузлів. Існує два основних метода: метод лічильника та метод "збору сміття".

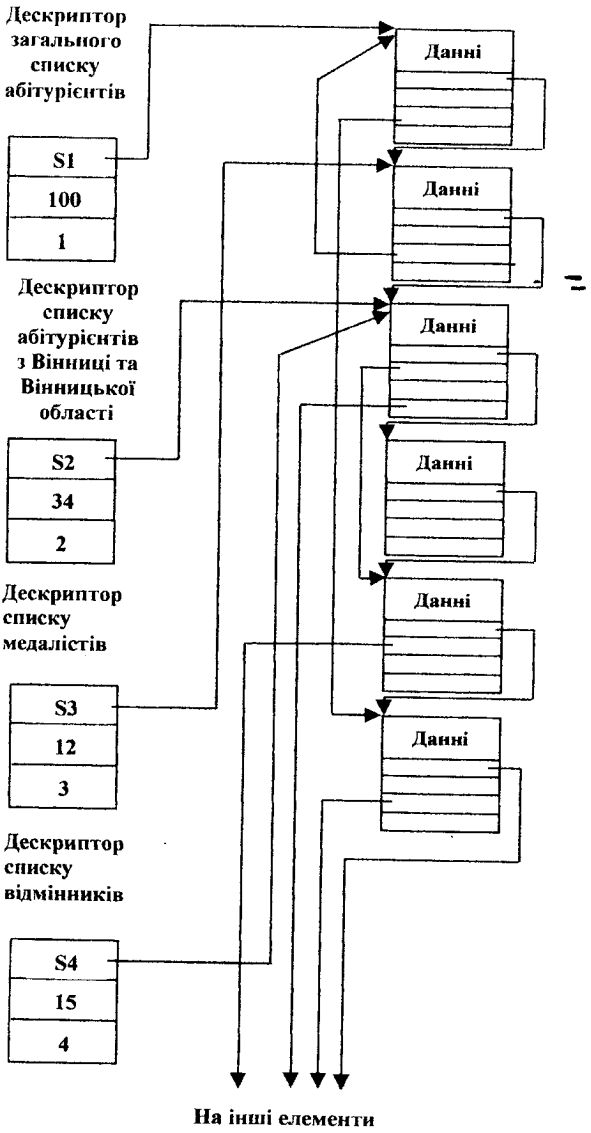


Рисунок 5.9 - Фрагмент чотиризв'язного списку

Метод лічильника зводиться до того, що після кожної операції виключення елементу з того чи іншого списку виконується перевірка, чи залишився даний елемент хоча б в якому-небудь одному функціональному списку.

Згідно методу в кожному елементі багатозв'язної структури вводиться спеціальне поле. Вміст вказаного поля збільшується на одиницю при формуванні кожного зв'язку і зменшується на одиницю при кожному розриві зв'язку. Якщо значення додатково введеного поля включає нульове, то елемент може бути виключений, оскільки він не входить до складу жодного списку.

В методі "збору сміття" не вимагається при розриві кожного зв'язку , який направлений до елементу, виконувати перевірку на предмет виключення елементу із списку та переводити його у вільний список. Для цього кожному елементу даних відводять однорозрядне поле для маркера.

Елемент, який вивільняється як би "провисає" до тих пір, поки не буде вичерпано вільний список. Тільки після цього запускається процедура "збору сміття".

Суть методу полягає в тому, що сканується виділена ділянка пам'яті і обнуляються всі маркерні поля. Під час другого проходу всі маркери всіх елементів, які входять в списку , встановлюються в одиничний стан. При третьому скануванні виділеної ділянки пам'яті всі елементи, маркери яких мають нульове значення, вилучаються.

Контрольні питання.

1. Скільки дескрипторів мають плекси ?
2. Які переваги мають плекси по відношенню до лінійних зв'язних списків ?
3. В якому випадку плекс стає лінійним з вязним списком ?
4. Дайте порівняльну характеристику методів "Лічильника " і "Збору сміття".

5.3 Представлення рядкових даних

У сучасних мовах програмування та інформаційних системах важливу роль відіграють рядкові дані. *Рядком* називається кінцева лінійно упорядкована послідовність простих даних символьного типу, яку розглядають як єдине ціле. Таке визначення рядка в загальному випадку припускає, що задана множина різних символів, або літер і набір правил, що дозволяє створювати бажані ланцюжки символів. [15,16]

У загальному випадку рядок у процесі виконання операцій над ним змінюється як по складу вхідних у нього символів, так і по довжині, тобто по числу символів. Тому логічна структура рядка може бути в найпростішому випадку представлена у вйгляді вектору з перемінною довжиною. Метою доступу до рядка може бути не один його елемент (символ, або літера), а деякий ланцюжок символів, у тому числі і др. всього рядка в цілому.

Часто для рядка виділяється вектор деякої фіксованої довжини, причому всі зміни розміру рядка повинні відбуватися в межах цієї довжини. При такому підході до організації рядка його варто вважати напівстатичною структурою даних. Існує і більш загальне трактування рядка як динамічної структури. Але така структура організується, як правило, за допомогою зв'язкового списку.

При векторному представленні фізичної структури рядки можуть бути організовані за допомогою дескриптора або граничних маркерів. При дескрипторному методі (рис.5.10) для кожного рядка відводять дескриптор, який включає інформацію про ім'я рядка, його довжині і про вказівник початку рядка в пам'яті . В деяких випадках може бути корисною інформація про максимальну довжину рядка.

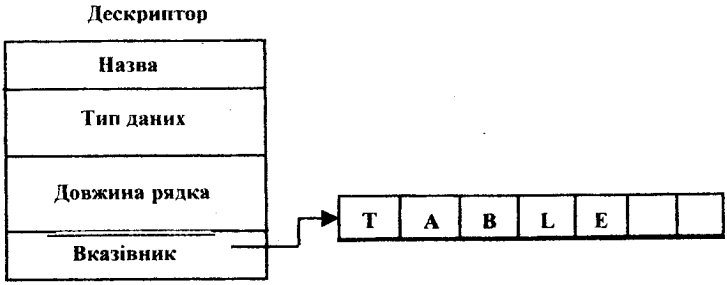


Рисунок 5.10 - Приклад векторного представлення фізичної структури рядка з використанням дескриптора

В методі граничних маркерів границі кожного рядка задаються в явному виді з використанням спеціального маркера, який , звичайно, не повинен бути елементом ніякого рядка. Приклад представлення рядка з маркером приведений на рис.5.11 , на якому в якості маркера використано спеціальний символ #.

#	С	Т	Р	У	К	Т	У	Р	А	#

Рисунок 5.11 - Приклад векторного представлення рядка з використанням граничного маркеру

В методі маркерів відпадає необхідність зберігати в дескрипторі інформацію про довжину рядка, і, відповідно , дескриптор спрощується. Однак маркери самі вимагають додаткової пам'яті для їх зберігання. Крім того, в цьому методі довжина рядка в випадку необхідності повинна

визначатися шляхом підрахунку її елементів, що приводить до збільшення обробки відповідного рядка. У випадку, коли для запису довжини рядка і маркера необхідна однакова ділянка пам'яті, метод маркерів навряд чи має перевагу перед дескрипторним методом.

Розглянемо представлення рядків в машинній пам'яті з використанням звязних списків. В цьому методі кожному елементу рядка відповідає вузол однозв'язного списку, як показано на рис. 5.12 .

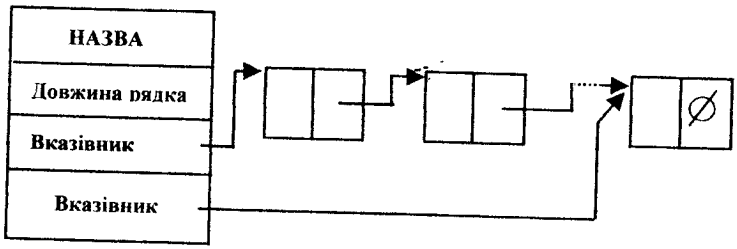


Рисунок 5.12 - Представлення рядка за допомогою однозв'язного списку

Спискове представлення рядків забезпечує максимальну гнучкість в виконанні різних операцій над рядками. Однак пам'ять при цьому використовується дуже неефективно , оскільки елемент і вказівник може займати один і то й же об'єм пам'яті. Другий недолік спискового представлення рядка полягає в тому, що логічно сусідні елементи рядка не є фізично сусідніми в пам'яті.

В ряді випадків рядок розбивається на групи, для зберігання яких використовується зв'язний список з елементами однакової довжини (рис 5.13) . В кінці кожного елемента розміщують вказівник, який зберігає адресу наступного елементу списку.

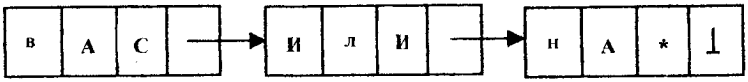


Рисунок 5.13 - Приклад представлення рядка, який розділений на групи символів , у вигляді зв'язного списку з елементами одного і того ж розміру

Якщо в зв'язному списку використовуються елементи різного розміру(рис.5.14), то перед адресним вказівником необхідно розміщувати спеціальний символ, який вказує, що наступний елемент є вказівником. Зрозуміло, що такий символ не повинен входити до алфавіту, який використовується для формування рядків.

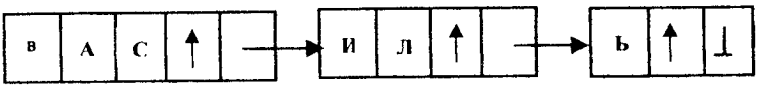


Рисунок 5.14 - Приклад представлення рядка, який розділений на групи символів , у вигляді зв'язного списку з елементами різного розміру

Контрольні питання

1. Яку інформацію розміщують в дескрипторі у випадку розміщення в пам'яті алфавітно-цифрової інформації ?
2. Які переваги і недоліки властиві списковому представленню рядків ?
3. Які особливості представлення рядка при розділені рядка на групи різного розміру ?
4. Поясніть використання в якості маркера використано спеціальний символ # в методі граничних маркерів .

5.4 Індексні методи

Існує множина індексних методів доступу, в основі яких лежить принцип формування окремого файлу або в списку з статей значень дійсного ключа. Стаття дійсного ключа називається *статтею індексу*, а весь файл дійсних ключів — *індексом*. Індексний файл значно менший відповідної йому бази даних. [14-16]

Широке застосування метод індексування знайшов в документальних інформаційно-пошукових системах і в системах ручної обробки, таких, як словники, алфавітно-предметні покажчики, систематичні покажчики в бібліотеках. В силу того що індексування припускає як альтернативу прямому доступу ще один підхід до довільної обробки даних, воно має безпосереднє відношення до автоматизованих систем, що закликані забезпечити ефективне зберігання і вибірку даних.

В індексно-послідовному методі доступу індексний файл завжди упорядкований по так званому первинному ключу. *Первинний ключ* — головний атрибут фізичного запису. По його значенню ідентифікується фізичний запис. Як правило, записи зберігаються в тій же логічній послідовності, що і індекс (звідси і назва індексно-послідовний метод доступу), тому індекс повинен містити посилання не на кожний запис бази даних по ключу, а на групу записів в фізичному блоці по діапазону ключів. Наприклад, якщо в блоку зберігається 10 записів, то для нього в індексному файлі буде одна стаття, а не десять, і розмір індексного файлу зменшиться в 10 раз.

Послідовна організація індексного файлу допускає індексацію його вмісту. Записи індексу групуються в блоки, які можна також

індексувати. При роботі з більшими файлами така організація дозволяє покращити характеристики доступу.

На практиці ключі звичайно розподілені нерівномірно, із-за чого попереднє закріплення значень ключів за блоками небажано, тому при створенні файлу даних виконується початкове завантаження в нього фізичних записів, упорядкованих по первинному ключу.

Найбільше або найменше значення ключа останнього запису, що міститься в блоку (більшість методів доступу забезпечує завантаження файлу даних по зростанню значення первинного ключа), заноситься в індексний файл (рис.).

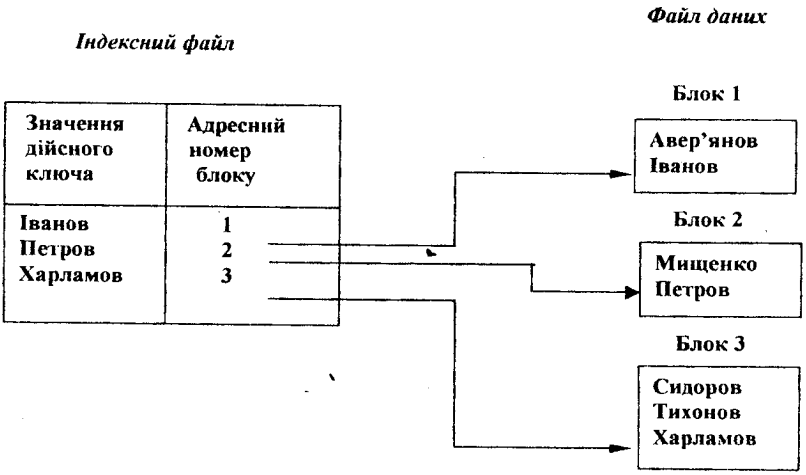


Рисунок 5.15 - Індексно – послідовний метод доступу

В цьому прикладі індексний файл і файл даних організовані послідовно. Індексний файл містить тільки максимальні значення первинних ключів записів кожного блоку. Поєднання кодів літер слова ТИХОНОВ більше в числовий інтерпретації поєднання кодів літер слова СИДОРОВ, але менше поєднання кодів літер слова ХАРЛАМОВ.

Існує два способи включення нових записів:

1. Запис запам'ятовується в окремій області, що називається *областю переповнення*. Блок в цій області зв'язується в ланцюжок з блоком, якому логічно належить запис. Звернення до записів в області переповнення значно знижують ефективність доступу.

2. При неможливості введення запису в блок він ділиться на два нових блоки, в перший з яких включається половина або дві третини записів, що містяться в вхідному блоці, в другій — частина записів ,що залишилася у вхідному блоці. Після цього в індексному файлі створюється нова стаття. У порівнянні з організацією ланцюжка записів в області переповнення при використанні даного методу ефективність доступу в більшості випадків виявляється вищою. Однак при багатократному розчепленні одних і тих же блоків потрібно розвантажити і реорганізувати файл, а також створити новий індексний файл.

Розглянемо питання організації індексних файлів. Якщо файл даних не дуже великий, то індексний файл формується, як показано на рис. . Для дуже великих файлів доцільно будувати декілька індексних файлів, тобто. ієрархію індексних файлів. При цьому кожний індексний файл наступного рівня містить покажчики на індексний файл попереднього рівня. Побудова такої ієрархічної системи індексації може забезпечити прийнятні розміри індексного файлу вищого рівня і

можливість зберігання його в пам'яті процесора. На рис. 5.16 для ілюстрації наведена дворівнева система індексних файлів, що реалізує індексно-послідовний метод доступу до файлів даних достатньо великого обсягу.

Записи індексного файлу 1-го рівня (рис.) групуються в блоки (101, 102,..., 150(тут блоки індексного файлу містять по два записи) таким чином, щоб отримані блоки могли індексуватися. Наприклад, перший запис індексного файлу 2-го рівня (блок 151) містить адресу (номер блоку 101) індексного файлу 1-го рівня. В цьому блоці знаходяться адреси записів, значення ключа яких менше або рівне значенню ключа першого запису блоку 151.

Ефективність доступу. По мірі розширення бази даних індекс набуває все більшої кількості елементів. Як показано на прикладі (рис.5.16), результатом цього є підвищення рівня індексації. Розміри індексного файлу і рівень індексації величини взаємозв'язанні: зменшення розміру першого призводить до збільшення другого. Як використання індексного файлу великого розміру, так і велике число рівнів індексації знижують ефективність доступу. В значній мірі зменшити час вибірки первинних ключів при використанні цього методу дозволяє розміщення індексу вищого рівня в оперативній пам'яті ЕОМ, що, однак, тягне за собою необхідність резервувати достатньо великі об'єми пам'яті процесора.

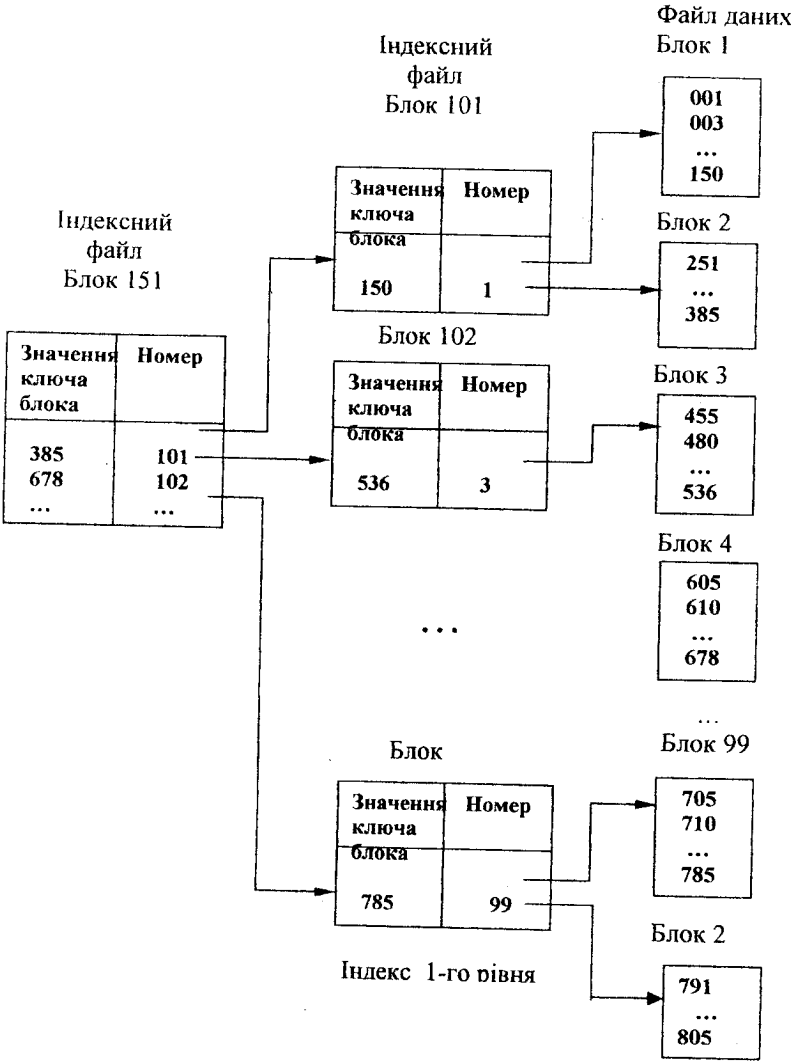


Рисунок 5.16- Дворівнева організація індексно – послідовного методу

Ефективність зберігання. Для розміщення записів, що додаються в базу даних після початкового завантаження, використовується вільна область, що резервується під час початкової завантаження. Ефективність зберігання залежить від того, яку частину пам'яті, що відводиться базі даних, займає ця вільна область. Оскільки записи повинні бути організовані логічно послідовно, ефективність зберігання залежить і від того, в який частині бази даних частота включення і вилучення записів бази даних найбільш висока. Визначення числа вільних фізичних блоків і їхніх частин вимагає проведення відповідного аналізу.

Особливу увагу необхідно звернути на розмір блоків, що використаються як для зберігання даних, так і індексів. Цей чинник впливає як на ефективність зберігання, так і на ефективність доступу. Середній розмір блоку безпосередньо зв'язаний з загальним числом блоків в базі даних. Чим більше розмір блоку, тим менше число їх зберігається в базі даних і тим менше елементів в індексі. Із зростанням числа блоків необхідно більше число рівнів індексації.

В деяких випадках вимагається обробка записів бази даних не в їх логічній послідовності, а в довільному порядку. Цю можливість забезпечує організація даних на основі індексно – довільного методу , який , як і попередній базується на використанні індексу. Як видно з назви методу , в даному випадку записи зберігаються в довільному порядку Рис. 5.17), але додатково створюється окремий файл статей , що включає значення дійсного ключа та фізичні адреси записів що зберігаються.

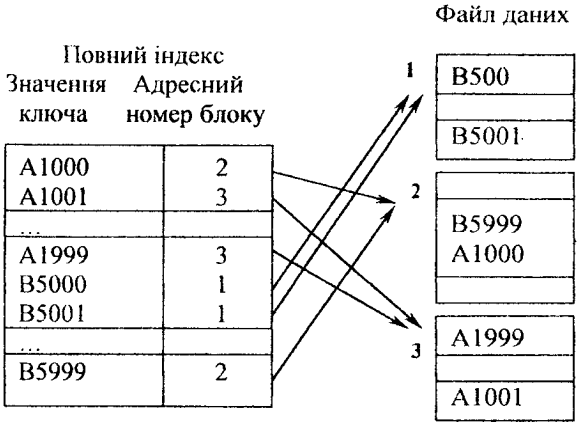


Рисунок 5.17 - Індексно – послідовний метод доступу з впорядкованим масивом

Стаття, що містить дійсний ключ і адресу, називається *статтею індексу*, а весь файл — *індексом*. Кожному запису бази даних відповідає стаття індексу.

Можливо застосування для ведення індексу схеми упорядкування, що дозволяє застосувати до індексу, наприклад, бінарний пошук

Оскільки для кожного запису відповідає стаття індексного файлу, останній може мати досить великий обсяг.

Можна відмітити, що індексно-послідовний метод доступу вказаного недоліку не має.

Розглянемо ефективність доступу. При запитах типу **отримати унікальний запис** ефективність доступу дуже висока і рівна одиниці, бо індекс містить відповідну **статтю** для кожного запису. Очевидно це справедливо тільки для ключових атрибутів.

При запитах типу **отримати всі записи** ефективність послідовної обробки завбачити неможливо, оскільки записи бази даних зберігаються в довільному порядку і необов'язково в логічній послідовності.

Ефективність зберігання. Так як кожний запис бази даних представлена відповідною статтею індексу, індексний файл може мати у порівнянні з індексно-послідовним засобом досить великий об'єм, що вимагає створення декількох рівнів індексації і призводить відповідно до збільшення розмірів бази даних.

Контрольні питання

1. В чому полягає відмінність індексно-довільного та індексно-послідовного методів доступу ?
2. Охарактеризуйте способи включення нових записів для індексно – послідовного методу доступу.
3. Як застосувати до індексу бінарний пошук ?
4. Коли доцільно застосування індексно – довільного методу ?

5.5 Адресні методи

Термін «методи обчислення адресу», відомий як перетворення ключа на адрес, охоплює велику групу розроблених до цього часу методів пошуку, що починаються з обчислення адресу по імені. Такі методи найбільше повно використовують властивості пам'яті з довільним доступом.

У методах обчислення адресу використовується деяка обчислювальна операція, що перетворить значення ключа К запису даних у відповідний йому адрес *пам'яті, тобто реалізується адресна* функція К - >М.

Методи обчислення адресу розділяють на дві групи [14]:

1) методи, у яких адресна функція реалізує взаємно однозначну відповідність адресів та ключів;

2) методи перемішування, у яких адресна функція реалізує тільки однозначне перетворення ключа на адресу; обернене перетворення частіше не має місця.

Методи першої групи.

Методи адресації за допомогою ключа, еквівалентного адресу. Метод використовується у випадках, коли таке перетворення можливо. Нехай у запис при його надходженні в систему в якості атрибута включається адрес пам'яті, із якого він розміщається. Надалі цей атрибут використовується в якості ключа. Наприклад, адрес запису рахунку буде друкуватися в розрахунковій книжці клієнта ощадкаси і тоді при наступних звертаннях до системи пошук запису рахунку клієнта в БД здійснюється безпосередньо по ключу-адресу.

Метод адресації з використанням алгоритму перетворення ключа на адрес. Метод припускає лінійну упорядкованість записів у файлі, у якому виконується адресація, і використання записів фіксованої довжини. Відповідно до розміру записів і характеру їхньої упорядкованості у файлі складають рівняння адресної функції, по якому перетворяться значення ключа запису в його адрес у файлі. Наприклад, організується файл для збереження даних про продаж авіаквитків. Схема запису файла: (НОМЕР - РЕЙСА, ДАТА-ВИЛЬОТУ, НАЯВНІСТЬ-СВОБОДНИХ-МІСЦЬ). Ключем є атрибути НОМЕР-РЕЙСА і ДАТА-ВИЛЬОТУ. Шлях атрибут НОМЕР-РЕЙСА приймає значення від 01 до 50, атрибут ДАТА-ВИЛЬОТУ - від 001 до 365. Розмір запису - 20 байт. У цьому випадку адреса запису визначається: Аотн і Амаш - відносний і машинний адреси; НР - значення атрибута НОМЕР - РЕЙСА; ДВ - значення атрибута ДАТА - ВИЛЬОТУ; З - базова адреса, починаючи з який файл розташовується в пам'яті.

При своїй простоті даний метод має серйозний недолік - мале заповнення пам'яті, відведеної для збереження файла. У файлі залишаються вільні ділянки пам'яті, якщо ключі не перетворяться в неперервну множину адресів. У прикладі що розглядається це буде у випадку, якщо не всі рейси виконуються щодня. У цьому випадку по цілому ряді адреса запису будуть відсутні. Розглянуті методи адресації

прості і забезпечують найбільш високу швидкість пошуку даних. Тому їх частіше усього застосовують у діалогових системах, де час пошуку критичний.

У розглянутих методах адресація виконується в два етапи: спочатку ключ перетвориться у відносну адресу (відносно початку файла), а потім відносна адреса перетвориться в машинну. Використання відносних адрес робить адресацію даних більш гнучкої.

Методи другої групи. Методи перемішування відомі також за назвою методів кешування, методів розсіяної пам'яті, іноді їх називають методами рандомизації.

Для розгляду кешування введемо поняття адресного простору пам'яті.

Адресний простір - множина адрес його блоків. Потужність такої множини дорівнює числу блоків у пам'яті, включаючи і "порожні" блоки.

Простір ключів - множина можливих значень ключів.

Коефіцієнт завантаження файла - відношення числа зайнятих блоків до загального числа блоків у файлі.

Коефіцієнт завантаження - не є постійним величиною . На початку, коли у файл не записаний жодний блок даних, цей коефіцієнт дорівнює 0, а з часом, у міру додавання у файл нових блоків, він може наближатися до одиниці.

Основна ідея кеш-адресації полягає в тому, що кожний примірник збереженого запису розміщається в пам'яті за адресою, що обчисляється за допомогою деякої адресної функції - кеш-функції - за значенням первинного ключа запису. У разі потреби не обов'язково організовувати запису у відповідності зі значеннями саме первинного ключа. Формально можна виконувати організацію записів за значеннями любого з полів, що входять до складу запису. Усе визначається конкретною прикладною програмою.

Таким чином, щоб спочатку запам'ятати екземпляр запису, необхідно

обчислити адрес збереження в пам'яті і помістити екземпляр по цьому адресу. При пошуку екземпляру запису виконуються ті ж самі обчислення і запис зчитується з пам'яті по отриманому адресу.

Розглянуті вище методи першої групи, що використовують числове значення первинного ключа в якості адресу збереженого запису можна використовувати тільки для обмеженого кола прикладних програм. У загальному випадку ці методи не використовують тому, що діапазон значень первинного ключа звичайно багато ширше діапазону адрес пам'яті, виділених для збереження записів.

Якщо в розглянутому вище прикладі припустити, що номера рейсів приймають значення від 0001 до 9999, а аеропорт, для якого створюється інформаційна система, обслуговує не більш 50 рейсів, то застосування методу адресації з використанням алгоритму перетворення ключа на адрес неможливо через колосальні витрати пам'яті.

Оскільки для збереження записів про рейси потрібно не більш 50x365=18250 ділянок пам'яті по 20 байт (вважаємо, що рейси виконуються щодня), те необхідно розробити спеціальну функцію адресації - хеш-функцію, яка б перетворювала б будь-яке значення первинного ключа в діапазоні від 00010001 до 9999365 у значення з діапазону 0-18250. Щоб врахувати можливе розширення в майбутньому, останній діапазон звичайно збільшують на 20%.

Цей приклад також ілюструє і недоліки кеш-адресації. Отримана за допомогою кешування послідовність розташування в пам'яті екземплярів збережених записів звичайно не збігається з послідовністю, обумовленої первинним ключем. Фактично збережений файл із хеш-адресацією звичайно розглядають як не упорядкований за значеннями ключа. Інший недолік кеш-адресації - можливість колізій.

Вибір функції хешування - досить важка задача і повинна виконуватися програмістом з урахуванням наступних вимог.

1. Функція кешування повинна відображати значення ключів в адреси блоків таким чином, щоб забезпечувався досить рівномірний розкид значень ключів по адресах блоків.
2. Функція кешування повинна як можна рідше приводити до звернення декількох ключів по тій самій адресі.
3. Функція кешування повинна бути простою, щоб не було потрібно тривалого часу на обчислення адреси блока по заданому значенню ключів.

Метод квадратів - це один із перших методів для побудови хеш-функції. Для багатьох практичних випадків він працює задовільно. Проте, якщо в реальному файлі існують численні групи ключів, значення яких закінчуються декількома нулями, то метод може виявитися неприйнятним через велике число випадків колізій. Зведення в квадрат удвічі збільшує кількість нулів і цих нулів можуть поширюватися на середню частину квадрата ключа, що інтерпретується як адрес.

Метод середини квадрата. Значення ключа перетвориться в число. Це число потім зводиться в квадрат, із нього вибираються декілька середніх цифр, і інтерпретуються як адреса запису.

Метод, оснований на діленні. Метод ділення простий і дає непогані результати для реальних файлів, що виявлено на численних перевірках. Кеш-функция обчислюється по формулі, де її значення G(k) дорівнює залишку від ділення значення ключа на число *M*. Вибір числа *M* є вирішальним у цьому методі. Якщо *M-парне* число, то значення G(k) завжди буде парним при парному k і непарним при непарному k. Якщо *M* дорівнює степені основи системи числення, застосовуваної в ЕОМ, то G(k) залежить тільки від самих правих розрядів k і не залежить від інших. Якщо *M* кратно 3, то багато значень буквених ключів, що відрізняються один від одного лише порядком букв, можуть дати значення функції, різниця між який кратна 3. Подібні властивості створюють серйозну

потенційну небезпеку виникнення великого числа колізій. Тому в якості *M* рекомендується вибирати просте число, рівне або близьке до розміру ділянки пам'яті.

Метод, оснований на множенні. Метод також простий і дає гарні результати. Він заснований на такий властивості: якщо значення k рівномірно розподілені у відрізку *[А, Б]*, то і значення G(k) рівномірно розподілені у відрізку [С-А, *С-Б]*. Відрізок *[С-А, С-Б]* відображається на просторі адрес. У даному методі дуже важливий правильний добір значення множника *с*.

Метод згортки. Цифрове представлення ключа розбивається на частини, кожна з який має довжину, рівну довжині необхідної адреси. Над частинами робляться якісь арифметичні або порозрядні логічні операції, результат яких інтерпретується як адреса. Наприклад , для порівняно невеликих таблиць із ключами - символьними рядками непогані результати дає функція хешування, у якій адреса запису утворюється в результаті додавання кодів символів, що складають рядок-ключ.

Розглянуті вище кеш-функції припускають, що значення ключів записів і всі проміжні обчислення не перевищують машинних форматів представлення числових даних для даної ЕОМ. Є і інші типи хеш-функції.

Методам другої групи властиве виникнення колізій. Хеш-функції можуть породжувати в деяких випадках той самий адрес для декількох різноманітних ключів. Поки пам'ять не занадто заповнена, наприклад на 50%, колізії виникають рідко і продуктивність методу хешування визначається в основному часом обчислення хеш-функції. Проте в міру заповнення пам'яті все частіше виникають колізії і в підсумку продуктивності методу все в більшій мірі залежить від продуктивності схеми усунення колізій.

Існує два підходи побудови схеми усунення колізій. Перший базується

на послідовному розподілі списків і називається методом відкритої адресації. Другий базується на звязному представленні списків і називається методом ланцюгів.

Метод відкритої адресації. Полягає в тому, що при виникненні колізії, проглядаються одна за іншою ділянки пам'яті, відведені для організації даних, до тих пір, поки не буде знайдена вільна ділянка, куди і поміщається запис. При виконанні пошуку запису послідовність дій аналогічна. Ділянки пам'яті проглядаються до тих пір , поки не буде знайдено запис із заданим значенням ключа, або поки не буде досягнута вільна позиція, що в даному методі є ознакою відсутності у файлі запису з зазначеним значенням ключа.

У даному методі задається правило, відповідно якого кожний *ключ h* визначає послідовність адрес у пам'яті, які необхідно переглядати щоразу при вставці або пошуку запису зі значенням ключа, рівним *h*. Якщо при виконанні операції пошуку, використовуючи ключ *h* послідовність адрес, виявиться вільна позиція, то це означає відсутність запису у файлі.

Розглянемо конкретний приклад.

Алгоритм роботи хеш - функції - складаємо десяткові коди обох символів, що утворять ключ, а потім результат ділимо на 10 і виділяємо цілочисельний залишок від ділення.

Згідно отриманих даних будуємо кеш-таблицю , в якій при наявності синонімів елементи даних розміщуються в вільних областях, які є найближчими до обчисленої за кеш-функцією.

Табл.5.1 - Розрахунок адреси

№	Ключ	Коды	Адреса
1	MO	12 14	6
2	CM	02 12	4
3	AX	00 23	3
4	BI	01 08	9
5	BJ	01 09	0
6	AN	00 13	3
7	JR	09 17	6
8	BC	01 02	3

Табл. 5.2. Розміщення даних у пам'яті

Адреса	Данні
0	BJ
1	
2	
3	AX
4	CM
5	AN
6	MO
7	JR
8	BC
9	BI

Основний недолік методу відкритої адресації - повторне звертання до таблиці . Повторне звертання виникає, коли таблиця містить багато ключів з однаковими кеш-адресами. Досить часто воно виникає тоді , коли імена з різноманітними хеш-адресами мають майже однакові послідовності адрес. Щоб уникнути повторного підгортання, приймають спеціальні міри.

Метод ланцюжків. Метод достатньо простий і полягає в тому, щоб організувати M зв'язаних лінійних списків по одному на кожну можливу хеш-адресу .

Після хешування ключа, якщо ділянка пам'яті по обчисленій адресі вільна, виконується розміщення запису по цій адресі. Якщо ж ділянка пам'яті по обчисленій адресі зайнята, то реалізується звертання по покажчику до іншої ділянки пам'яті (елементу списку), і так до кінця списку. Після цього запис поміщається на вільну ділянку пам'яті і за допомогою покажчиків приєднується до кінця свого списку. При пошуку записів дії виконуються в тієї ж послідовності. Спочатку перевіряється ділянка пам'яті по обчисленій адресі. Якщо там знаходиться запис з іншим значенням ключа, то по покажчику звертаються до іншого запису, і так доти, поки не буде знайдено необхідний запис або досягнутий кінець списку. Пам'ять, що виділяється для організації списків, *називають областю переповнювання*, а ділянка пам'яті з хеш-адресами - *основною областю.*

При організації даних на зовнішніх пристроях із прямим доступом намагаються мінімізувати число звертань до пристрою, тому запису групують у блоки, щоб за один раз зчитувати з зовнішньої пам'яті декілька записів. З цією метою весь адресний простір пам'яті, виділений для збереження файла, розбивається на блоки.

Розглянемо конкретний приклад (табл. 5.3.)

Табл. 5.3

Вихідні ключі	Ключі після перетворення	Основна область зберігання			Область переповнення		
		Адреса	Вміст запису	Вказівник ланцюга	Адреса	Вміст запису	Вказівник ланцюга
Іванов	101	101	Іванов	0	850		900
Мищенко	213	213	Мищенко	0	852	Захаров	
Луков	311	311	Луков	0	900	Шешиков	0
Курлов	416	416	Курлов	0			
Голікова	415	415	Голімова	423			
Петров	420	420	Петров	852			
Ванін	415	423	Ванін				
Потон			Потон				

В табл. значення вхідного ключа ПЕТРОВ перетворюється в адресу першого запису ланцюжка синонімів — 415. Запис ПЕТРОВ містить показник на адресу 423, по якому міститься запис ПОТОН. В свою чергу запис ПОТОН містить показник на адресу 852, по якій міститься запис ЗАХАРОВ. Запис ЗАХАРОВ містить показник на адресу 900, по якій міститься запис ШЕШИКОВ. Цей останній запис в ланцюжку містить нульовий показник. Таким чином будується найпростіший механізм пошуку синонімів по так званому ланцюжку синонімів. В загальному випадку процес ліквідації колізій синонімів складається з двох кроків. На першому кроку виконується перегляд первинної області з метою виявлення в ньому поруч з фрагментом першого запису вільного

простору для нового запису. За наявності вільного простору для нового запису перегляд припиняється. В противному випадку повинен бути здійснений другий крок — обробка переповнення.

На ефективність доступу даного методу впливають наступні три чинника:

1. *Розподіл вхідних ключів.* В більшості випадків розподіл вхідних ключів відрізняється від чисто випадкового (рівномірного). Чим більше розробнику відомо про розподіл, тим вище імовірність правильного вибору числа блоків і числа записів в блоках. Оптимальний вибір цих значень дозволяє розробнику скоротити середню довжину ланцюжка синонімів. Ефективність методу доступу шляхом кешування в значному ступені залежить від довжини ланцюжка синонімів. В більшості систем керування базами даних функція хешування забезпечує прийнятні параметри.

2. *Розподіл пам'яті.* Для ефективності доступу першорядне значення має рівномірність розподілу дійсних ключів на множині блоків (тобто у виділеному полі пам'яті). Якщо функція хешування призначає велике число ключів в одній області, то число синонімів зростає. Із збільшенням обсягу розподіленої пам'яті збільшується число адрес, що можуть формуватися підпрограмою рандомизації.

3. *Функція хешування.* Правильно підібрана функція хешування повинна забезпечувати рівномірний розподіл дійсних ключів у виділеному полі пам'яті, знижуючи таким чином середню довжину ланцюжка синонімів.

Контрольні питання

1. Які основні вимоги висуваються до хеш-функції ?
2. Що таке колізії ? Які причини їх виникнення ?
3. Які принципи побудови хеш-функції Вам відомі ?
4. Які методи вилучення колізій найбільш поширені ?

5.6 Інвертований метод

Серед *мультисписковых* методів доступу особливе місце займає інвертований метод. Описані вище методи доступу не дозволяють проводити вибірку записів по більше, ніж одному типу атрибуту або полю запису. Для реалізації такий можливості треба використати **інвертований метод доступу,** оснований на застосуванні індексів для різних атрибутів. Цей метод частіше використовується для пошуку і (в деяких випадках) для поновлення (модифікації). [14,15]

Зберігання найчастіше здійснюється з допомогою інших методів доступу. Завантаження бази даних може, наприклад, виконуватися будь-яким методом .

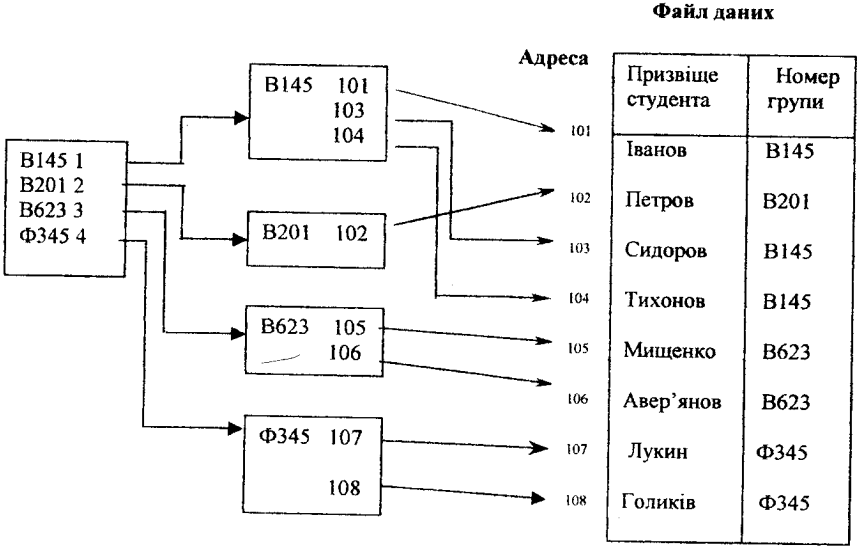


Рис 5.18. - Інвертований метод доступу

Кожному "інвертованому" полю файлу даних відповідає стаття в індексі. Стаття включає в себе ім'я поля, його значення і адресу запису. Після початкової завантаження бази даних здійснюється упорядкування статей по імені поля, а груп статей з загальним ім'ям поля — по значенню поля. Для кожного імені поля можна побудувати окремий індекс. Записи з одним і тим же значенням поля групуються, а загальне значення для всіх записів групи використовується в якості покажчика цієї групи. По мірі додавання нових записів створюються статті у відповідних індексах для відповідних значень. При виключені записів стаття індексу у відповідних індексах для відповідних значень знищуються.

На рис. 5.18 наведений приклад організації інвертованого індексу для номера групи файлу даних СТУДЕНТ, який інвертований відносно атрибуту запису "номер групи". Кожному інвертованому полю відповідає стаття в індексі (файл 1 рівня на рис.5.19). Стаття звичайно включає в себе ім'я поля, його значення і адресу запису. Якщо здійснюється пошук по іншому атрибуту, тобто необхідно побудувати окремий індекс, тобто інвертувати файл відносно цього поля. На рис.5.19 наведений приклад такої побудови для файлу СТУДЕНТ бази даних ФАКУЛЬТЕТ.

Організація даних в файлі (рівень 4) диктується вимогами так званого первинного методу доступу (певного на рівні 3) або задачами оптимізації операції читання всього файлу даних.

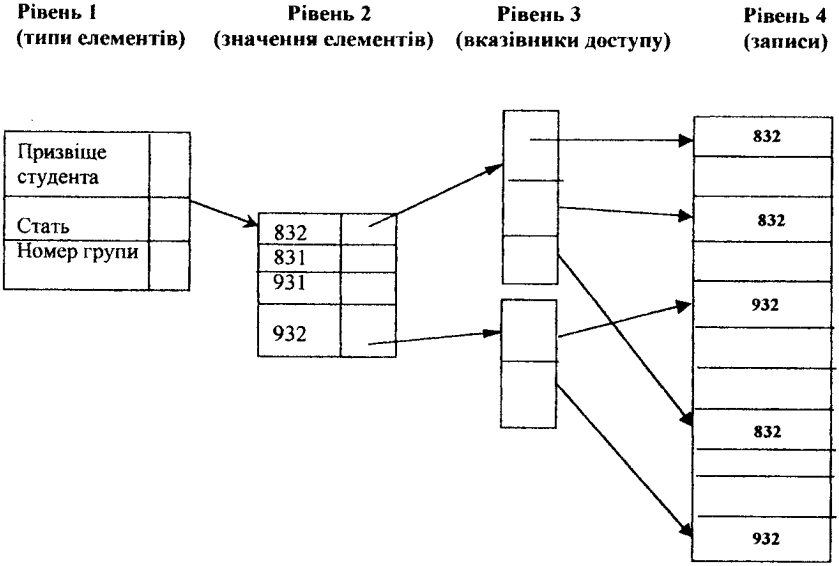


Рис. 5.19- Структура інвертованого файлу з дворівневим індексом для файлу СТУДЕНТ.

Контрольні питання

1. В яких випадках доцільно використовувати інвертований метод ?
2. Які атрибути відношення доцільно інвертувати ?
3. Як здійснюється завантаження даних для інвертованого методу ?

6.Організація баз знань

6.1 Поняття експертних систем

Система, яка основана на знаннях - система програмного забезпечення, основними структурними елементами якої є бази знань і механізм логічних висновків. У першу чергу сюди відносяться експертні системи, що є першим етапом на шляху реалізації досягнень у штучному інтелекті. [17]

Експертні системи (ЕС) - це комп'ютерні програми, розроблені для виконання тих видів діяльності, які під силу людині-експерту. Вони працюють таким чином, що імітують дії людини-експерта, і істотно відрізняються від точних, добре аргументованих алгоритмів і не схожі на математичні процедури більшості традиційних розробок.

Якщо при традиційному процедурному програмуванні комп'ютеру необхідно повідомити, що і як він повинний робити, то загальним для експертних систем є те, що вони мають справу зі складними проблемами:

- які недостатньо добре висвітлені або вивчені;
- для яких немає чітко заданих алгоритмічних рішень;
- які можуть бути досліджені за допомогою механізму символічних міркувань.

Специфіка ЕС полягає є в тому, що вони використовують:

- механізм автоматичного міркування (висновку);
- «слабкі методи», такі як пошук або евристика.

Експерти - це кваліфіковані фахівці у своїх областях діяльності, які мають загальні якості:

- мають великий об'єм знань про конкретну предметну область;
- мають великий досвід роботи в цій області;
- уміють точно сформулювати і правильно вирішити задачу.

ЕС покликані замінити фахівців у конкретній предметній області, тобто дозволити вирішити задачу без експерта.

Спрощена базова структура ЕС має наступний вид (рис. 6.1):

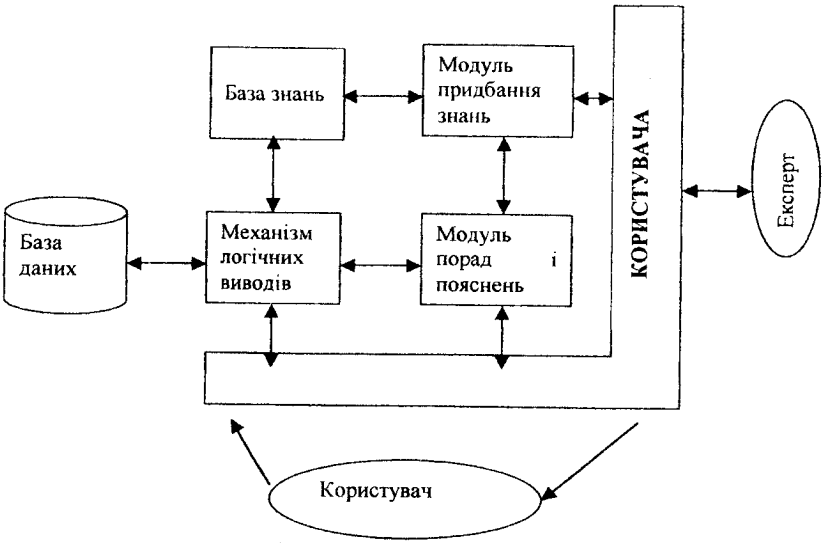


Рис. 6.1. - Базова структура експертної системи.

Для успішного виконання функцій, покладених на **ЕС**, необхідні:

- механізм формування знань про конкретну предметну область і керування ними (БД і БЗ);
- механізм, який на підставі наявних у БЗ знань спроможний робити висновки;
- інтерфейс для одержання і модифікації знань експерта, а також для правильної передачі відповідей користувачу (інтерфейс користувача);
- механізм одержання знань від експерта, підтримки БЗ і при необхідності її доповнення (модуль придбання знань);
- механізм формування різних коментарів .

Слід особливо підкреслити важливість механізму пояснень у складі **ЕС**:

- без них користувачу важко буде зрозуміти висновок, отриманий при консультації або рішенні якогось питання;

- цей механізм важливий для експерта - він дозволяє визначити, як працює система і з'ясувати, як використовуються надані їм знання.

Мова формування знань, яка використовується для розробки ЕС, називається ***мовою розробки*** ЕС, а система програмного забезпечення, що включає зазначені вище функції, називається ***інструментом для розробки*** ЕС або ***оболонкою*** ЕС. [17,18]

База знань містить факти і правила. ***Факти*** - це фрази без умов, вони містять твердження, які завжди абсолютно вірні. ***Правила*** містять твердження, істинність яких залежить від деяких умов, які утворять тіло правила.

Факти містять короткострокову інформацію в тому сенсі, що вони можуть змінюватися, наприклад, під час консультації.

Правила – це довгострокова інформація про те, як породжувати нові факти або гіпотези з того, що зараз відомо.

Чим такий підхід відрізняється від звичайної методики використання БД?

Основна відмінність полягає в тому, що БЗ має великі «творчі» можливості.

Факти в БД звичайно пасивні: вони там або є, або їх немає. БЗ, з іншого боку, активно намагається поповнити відсутню інформацію.

Однієї з основних проблем, характерних для СОЗ, є проблема представлення знань. Це пояснюється тим, що форма представлення знань має істотний вплив на характеристики і властивості системи.

Для можливості оперування знаннями з реального світу за допомогою ПК, необхідно здійснити їхнє моделювання. При цьому необхідно відрізняти знання, призначені для обробки комп'ютером, від знань, які використовуються людиною.

При проектуванні моделі представлення знань варто враховувати такі чинники, як: однорідність представлення і простота розуміння

Однорідність представлення приводить до спрощення механізму керування логічним виводом і керуванням знаннями. Простота розуміння припускає доступність розуміння представлення знань як експертами , так і користувачами системи. У противному випадку необхідним є придбання знань і їхня оцінка.

Однак виконати ці вимоги в однаковій мірі, як для простих, так і складних задач досить важко. В даний час для представлення знань використовують наступні види моделей [19]:

- модель на базі логіки;
- продуктивна модель;
- модель семантичної мережі;
- модель, основана на використанні фреймів і ін.

Основна ідея логічного підходу полягає в тому, щоб розглядати всю систему знань ,необхідну для рішення прикладних задач, як сукупність фактів (тверджень). Факти представляються як формули в деякому логічному зв'язку (першого або вищого порядку, багатозначної, нечіткої або ін.) Система знань відображається сукупністю таких формул і, подана в ЕОМ, вона утворить БЗ.

Формули неподільні і при модифікації БЗ можуть лише добавлятися або віддалитися. Логічні методи забезпечують розвитий апарат виводу нових фактів із тих, що явно подані в БЗ. Основним примітивом маніпуляції знаннями є операція виводу.

Контрольні питання

1. В чому відмінність баз даних від баз знань ?
2. Які методи використовують бази знань ?
3. Які чинники враховують при проектуванні моделей представлення знань ?

6.2. Отримання і формалізація знань

Важливим етапом при створенні БЗ є етап набуття знань. На цьому етапі різноманітний набір фактів про деякий предмет повинен бути поданий у вигляді деякої узагальненої структури. Однією з таких є структура, що одержала назву "дерево рішень". Це один із найпростіших способів подання фактів і його застосування обмежено. Водночас, використання "дерева рішень" може бути ефективно там, де знання подаються у вигляді правил. [19,20]

Даний підхід буде розглянутий з єдиною метою – показати, як знання про конкретну предметну область можуть бути формалізовані до рівня структури БЗ деякої експертної системи.

Структура дерева рішень ілюструє відношення, що повинні бути встановлені між правилами в добре організованій БЗ. Даний підхід можна реалізувати в системі Мікроексперт для IBM PC і багатьох інших, більш сучасних оболонках ЕС.

Формалізація задачі .Уявимо собі, що ми присутні при бесіді, коли фахівця в області ботаніки по телефону просять визначити тип деякої рослини. Тому що він не бачить конкретний екземпляр, а той, що запитує, не є фахівцем в області ботаніки, то консультуючий задає ряд питань, щоб одержати знання, необхідні для рішення задачі.

Один із варіантів такої консультації може бути поданий у графічному вигляді (рис. 6.2).

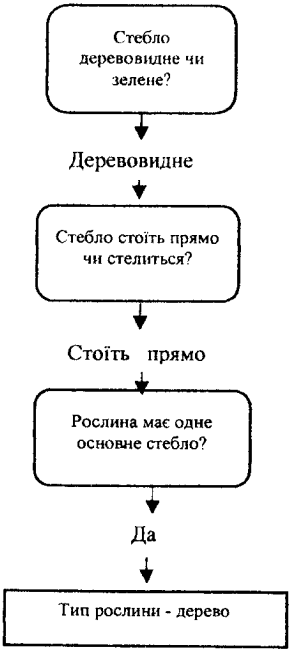


Рис. 6.2.- Телефонна консультація

Результатом такої консультації буде висновок: «Базуючись на Вашій відповіді, можна припустити, що тип рослини - дерево».

Однак дана діаграма ілюструє тільки один із можливих варіантів питань і відповідей (тобто експертизи). Аналогічним чином можна подати всі інші варіанти відповідей і хід консультації.

Що ж дозволяє фахівцю провести таку консультацію і визначити як послідовність питань, так і їхній зміст в залежності від відповідей опитуваного? Відповідь одна: його знання в конкретній предметній області, в якій він є фахівцем, (експертом).

Базуючись на знаннях експерта графічно діаграму всіх можливих висновків даної консультації можна подати у вигляді рис. 6.3.

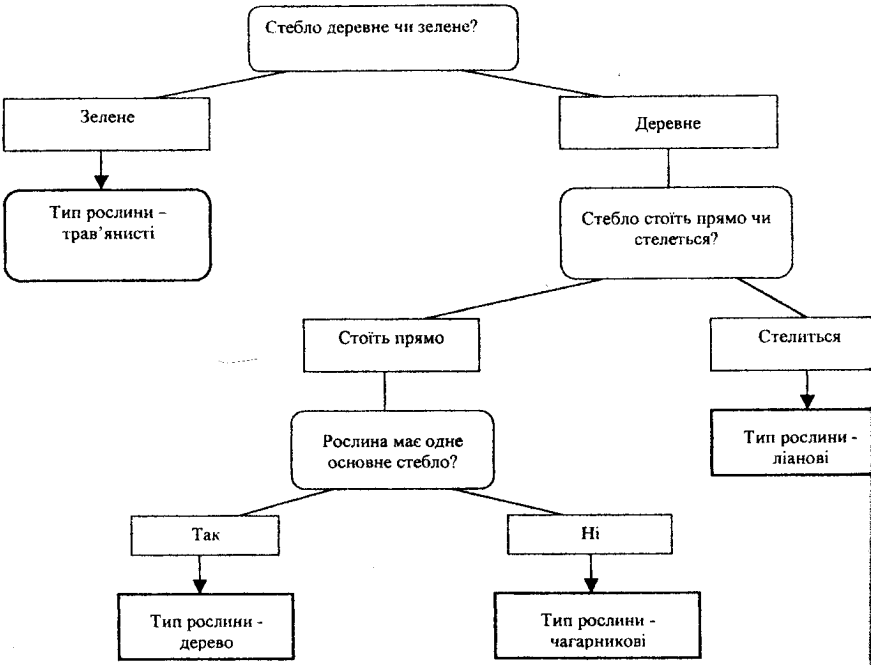


Рис. 6.3. Дерево рішень задачі

Це графічне подання моделі даних називається "деревом рішення", що об'єднує всі гілки пошуку типу невідома рослина.

Якщо консультація ЕС повинна бути більш глибокою і визначати, наприклад, клас рослини, то в цьому випадку для «типу рослини - дерево» повинно бути побудоване своє дерево рішень. Після одержання знань від фахівця його можна подати у вигляді рис. 6.4. Як видно, нова частина буде "піддеревом" вихідного "дерева рішень".

Існує декілька причин, за якими "дерево рішень" розбивається на секції:

- "дерево рішень" швидко стає довгим, важкооглядовим;
- розподіл "дерева рішень" на секції спрощує запам'ятовування мети, що переслідується в процесі отримання знань.

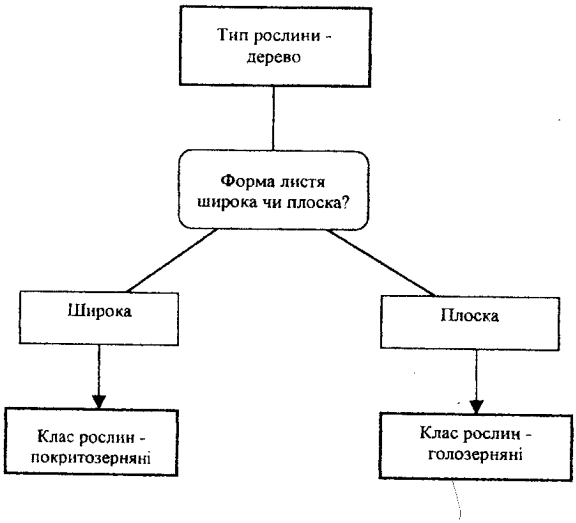


Рис. 6.4.- "Піддерево-1" рішення задачі

Існує декілька причин, за якими "дерево рішень" розбивається на секції:

- "дерево рішень" швидко стає довгим, важкооглядовим;
- розподіл "дерева рішень" на секції спрощує запам'ятовування мети, що переслідується в процесі отримання знань.

Коли "піддерево" створено, заключна його частина може бути скопійована в корінь знов створюваної гілки дерева рішень, і для неї на основі знань, що були одержані від експерта, може бути побудоване своє "піддерево рішень" (рис. 6.5) і т.д.

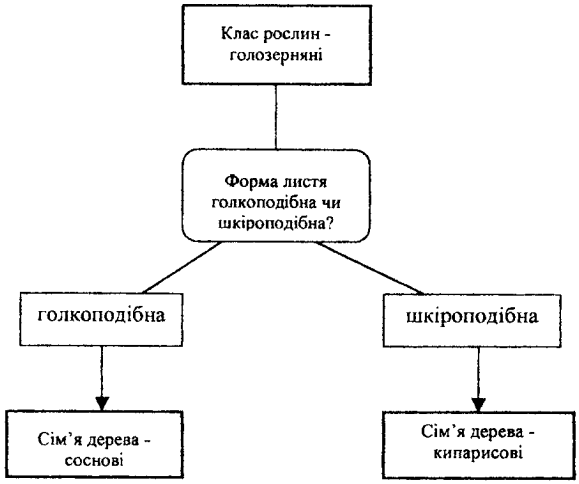


Рис. 6.5- "Піддерево-2" розв'язок задачі

Діаграми, приведені на рис.6.3 - 6.5 - це модель незакінченої ботанічної БЗ, що вирішує тільки вузьку частину загальної задачі.

На основі викладеного можна зробити висновок, що при розробці моделі БЗ будь-якої предметної області на основі дерева рішень необхідно:

- загальну задачу розбити на ряд підзадач;
- для кожної з підзадач розробити своє дерево рішень (це спростить створення і налагодження БЗ).

На прикладі задачі про ідентифікацію типу рослини розглянемо, як дерево рішень можна подати у вигляді правил.

Першим етапом формування правил є переклад дерева рішень із питань-відповідей в твердження-факти. Графічно для рис.6.3. це буде відповідати поданню у вигляді рис.6.6.

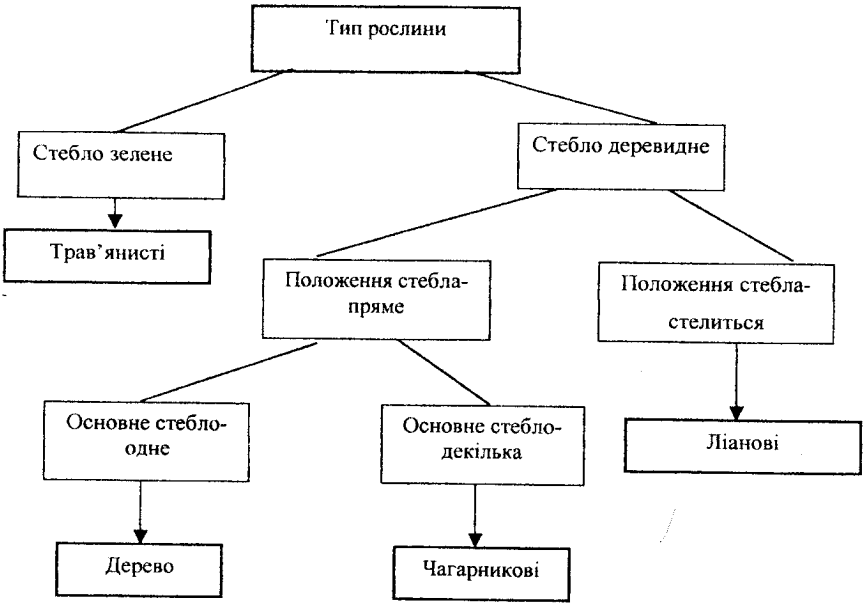


Рис. 6.6. Формування правил

Аналогічні перетворення можуть бути виконані для всіх "піддерев" "дерева рішень".

Загальним для них є те, що всі твердження подаються пропозиціями які включають **<атрибут>, <предикат>, <значення>.**

Наприклад:

<тип рослини> <є> <дерево>

<стебло> <є > <зелене>

У даному трактуванні:

Атрибут - це ключове слово або фраза, що описує деяку якість, про яку ми намагаємося знайти інформацію.

Значення - це опис, призначений атрибуту.

Предикат - це елемент, що вказує на відношення між атрибутом і його значенням.

Більш конкретно це поняття буде визначене в наступних розділах. На цьому етапі ми припускаємо, що між атрибутом і його значенням існує тільки одне відношення - *являтися (належати, бути)*, яке будемо позначати *IS*.

Слід зазначити, що в найпростіших ЕС, у тому числі й у Мікроексперт для РС, використовується тільки предикат IS.

Тоді всю множину фактів для дерева рішень можна подати набором пропозицій (таблиця 6.1).

Таблиця 6.1- Факти

Атрибут	Предикат	Значення
Тип рослини	Is	Трав'янисті Дерево Чагарникові Ліанові
Стебло	Is	Зелений Деревний
Положення стебла	Is	Пряме Стелиться
Основне стебло	Is	Один Декілька

Безсуперечна підмножина цих фактів буде складати БД експертної системи.

Розв'язок будь-якої задачі за допомогою цієї БД можна одержати за допомогою правил, з яких формується БЗ. Набір правил на основі дерева рішень формується виходячи з:

- кожна гілка дерева рішень від її початку до кінця утворює правило;
- пропозиції в дереві рішень, що розташовані по стрільці, утворюють умовну частину правила, а після стрілки - висновок правила;
- гілка в якій відсутній висновку, не може бути подана у вигляді правила;
- кожна пропозиція умовної частини правила являє собою кон'юнктивний член, тобто пропозиції з'єднуються логічною кон'юнкцією («і»).

Приклад:

ЯКЩО	стебло	IS	деревний
І	положення	IS	пряме
І	основне стебло	IS	один
ТО	тип рослини	IS	дерево

В деяких мовах програмування і оболонках експертних систем використовується більш короткий запис правил:

- виключається прийменник «то»;
- замість логічного «і» використовується кома;
- предикат записується в вигляді з використанням дужок.

В цьому випадку правило, що приведене вище можна записати у вигляді:

IS (type_of_plant, tree) if IS (stream, woody),

IS (position, upright),

IS (main_treak, one).

До складу БЗ можуть додатково включатися допоміжні інформаційні елементи, що полегшують роботу користувача з ЕС і роблять діалог користувача з ЕОМ більш докладним і осмисленим. Одним з таких елементів можуть бути підказки.

Підказка - це питання, що з'являється на екрані для одержання від користувача інформації про деякий атрибут, значення якого на даний момент невідомо.

Підказка -це рядок символів .Кожний атрибут повинний мати тільки одну підказку, що асоціюється з ним. Якщо немає запиту про деякий атрибут, то в нього немає підказки.

Приклад підказки в БЗ:

PROMT положення	Promt <атрибут>
Положення стебла - коштує прямо або стелиться ?	<рядки підказки>

Правила, записані в стиснутій формі, можуть бути незрозумілі для непрофесійного користувача. Атрибути, які вводяться у вигляді коротких фраз, дозволяють мінімізувати об'єм пам'яті, необхідний для збереження правила, і прискорити його введення й обробку.

Для того щоб зробити правило більш зрозумілим і інформативним, кожному атрибуту можна поставити у відповідність його переклад.

Переклад - це рядок символів, що застосовується для пояснення атрибута.

Приклад:

Пропозиція: стебло є зелене	trans <атрибут>
Атрибут - «стебло»	<рядки перекладу>
Можливі значення - «зелений», «деревний»	
Переклад для атрибута - «стебло рослини»	

В цьому прикладі як тільки з'явиться в правилі атрибута «стебло» він буде замінений перекладом «стебло рослини», після чого для закінчення пропозиції система сама додасть предикат «є» і значення, що відповідає даному атрибуту.

Розглянутий синтаксис підказок і правил типовий для системи Мікроексперт. Вони разом із набором правил складають БЗ Мікроексперта і можуть бути приміщені в БЗ у будь-якому порядку.

Контрольні питання

1. Що називається "деревом рішень" ?
2. Для чого дерево рішень розбивають на секції ?
3. Дайте означення атрибуту та предикату .
4. Для чого вводять підсказки ?

6.3 Представлення знань із використанням логіки предикатів

Одним із способів представлення знань є мова математичної логіки, що дозволяє формально описувати поняття предметної області і зв'язки між ними. [18-20]

На відміну від природної мови, яка дуже складна, мова логіки предикатів використовує тільки такі конструкції природної мови, які легко формалізуються. Тобто логіка предикатів - це мовна система, що оперує з реченнями на природній мові в межах синтаксичних правил цієї мови.

Мова логіки предикатів використовує *слова*, що описують:

- поняття та об'єкти предметної області;
- властивості цих об'єктів і понять, а також відношення між ними.

У термінах логіки предикатів перший тип слів називається *термами*, а другий - *предикатами*.

Терми - це засоби для позначення індивідуумів, а ***предикати*** виражають відношення між індивідуумами.

Логічна модель - це множина пропозицій, що виражають різні логічні властивості іменованих відношень.

При логічному програмуванні користувач описує предметну область сукупністю пропозицій у вигляді логічних формул, а ЕОМ, маніпулюючи цими пропозиціями, будує необхідний для рішення задач висновок.

Найпростіші конструкції мови предикатів.

Терм - це знак (символ) або комбінація знаків (символів), що є найменшим значимим елементом мови. До термів відносяться константи, змінні і функції.

Константа застосовується для позначення конкретних об'єктів реального світу. Приклад: ластівка, птиця, один, 2 і т.д.

Змінні використовуються для позначення деякого з можливих об'єктів реального світу або їхньої сукупності (у Пролозі починаються з заголовної літери). Приклад: Хтось, X, Who, Річ і т.д.

Функції (структури) - послідовність із декількох констант або змінних, які поміщаються у круглі дужки. Приклад: сума (1,2); +(1,2); подвоїти (X).

Функтори позначають оператори, яким після впливу на об'єкт повертають деяке значення.

Предикат - це логічна функція, що виражає відношення між своїми аргументами і приймає значення «дійсно», якщо це відношення має місце, або «недійсно», якщо воно відсутнє.

Розміщена в дужки послідовність із n термів, перед яким стоїть предикатний символ, називається *n-місцевим* (або *n-арним*) *предикатом*. Він приймає значення «дійсно» або «недійсно» у відповідності зі значенням термів, які є його аргументами.

Приклад:

є (ластівка, птиця)

батько (Х, Джон)

Такого типу предикати одержали назву *атомарних предикатів* і відповідають найбільш простим пропозиціям нашої розмовної мови - нерозповсюдженим пропозиціям.

У звичайній мові з *нерозповсюджених* пропозиції за допомогою сполучних займенників, союзів, і інших частин мови будують більш складні конструкції - *складні* пропозиції.

Предикатні формули

У логіці предикатів складними пропозиціями природної мови відповідають предикатні формули. ***Предикатні формули*** форимують з атомарних предикатів і логічних зв'язків (таблиця 6.2):

Таблиця 6.2 Логічні зв'язки

,	V	¬	←	↔
«і»	«або»	«ні»	«якщо»	«тоді і тільки»

Логічні зв'язки мають наступний пріоритет використання:

1. ¬
2. , V
3. ←, ↔

Найбільше часто в логічному програмуванні використовуються низки «І», «НІ» «ЯКЩО».

Приклад предикатної формули, що відповідає складній пропозиції:

є (ластівка, птиця) ← має (ластівка, крила),

володіє (ластівка, гніздо).

де є (_,_); має (_,_); володіє (_ _) - атомарні предикати; «,» і «←» - логічні зв'язки.

Приведена конструкція предикатної формули дозволяє робити твердження не тільки про конкретного індивідуума яким є ластівка, але і

про всіх індивідуумів із класу птиць, використовуючи замість констант змінні:

є (X, птиця) ← має (X, крила), володіє (X, гніздо)

Таким чином, ставлячи змінні замість конкретних імен, приходимо до більш загальних понять кортежу довжини n, предиката і логічної формули.

Предикат, що містить змінні, наприклад,

має (X, крила)

не може бути оцінений, тобто не можна визначити дійсність, тому що його значення визначається після підстановки в змінну деякої константи.

Однак іноді можна визначити значення предиката не роблячи підстановок використовуючи *квантори спільності* ( ) і *існування* ( ), що позначають «для всіх» і «існує принаймні одне».

Тоді приведена вище логічна формула буде записана у вигляді:

(X) [бути (X, птиця) ← має (X, крила), володіє (X, гніздо)]

і відповідає пропозиції, що може читатися як: «будь-яке X є птицею ,якщо це X має крила і володіє гніздом».

Квантори і можуть використовуватися і для будь-якого числа змінних. Розглянемо їхнє різне використання на прикладі двомісного предиката

любить (X, У),

що описує відношення «X любить Y»:

- (X)(Y) люблять (X, Y) - усі люди люблять усіх людей;
- (X)(Y) любить (X, Y) - існує людина, що любить усіх;
- (X) (Y) любить (X, Y) - для кожної людини існує той, що його любить;
- (X) (Y) любить (X, Y) - існує людина, що кого-небудь любить.

Визначення правильно побудованої формули

Комбінуючи логічні зв'язки і квантори можна рекурсивно визначити складову формулу логіки предикатів, яку називають *правильно побудованою формулою* (далі просто ППФ або логічна формула).

Якщо говорити стосовно до природної мови, те ППФ описує звичайну пропозиція загального виду.

1. Термом є або константа, або змінна, або кортеж із n термів, перед якими стоїть функтор.
2. Предикат - це кортеж із n термів, перед якими стоїть предикатний символ.
3. Атомарний предикат є логічною формулою.
4. Якщо F і G - логічні формули, то (F); F,G; F v G; ¬ F; F→ G; F↔ G - також є логічними формулами.
5. Якщо F(X) - логічна формула, те обидва вираження (X) F(X), (X) F(X) є логічними формулами.
6. Всі результати, одержувані повторенням кінцевого числа n1 - n6, є логічними формулами.

Множина всіх пропозицій, побудованих відповідно до даних правил, утворять мову логіки предикатів першого порядку. Скориставшись цими визначеннями, можна, наприклад, пропозицію «усі люди смертні» записати у вигляді:

(X) [людина (X) ← смертний]

Логічне виведення

Логічний вивід - це процес одержання з множини правильно побудованих формул (S) деякої ППФ (s) шляхом застосування одного або декількох правил виводу.

Правило резолюції для простих пропозицій

Найбільше простий метод логічного виводу використовує тільки одне правило виводу, яке називається *резолюцією*. Її застосовують до логічних формул виду:

факт: A

заперечення: $\neg (A_1, \ldots, A_n)$

імплікація: $A \leftarrow B_1, \ldots, B_m,$

де A_i (i = 1, n) і B_j (j = 1, m) - довільні предикати.

Розглянемо найбільше просту з форм резолюції для випадку всього лише двох S = { S_1, S_2 } ППФ виду:

S_1 (заперечення): ¬ **A**

S_2 (імплікація): **A ← У ,**

у якій предикат A з S_1 збігається з предикатом A лівої частини S_2.

У результаті одного кроку виводу з S_1 і S_2 буде отримана нова ППФ виду:

S: ¬ B.

На цьому кроку виводу ППФ S_1 і S_2 називаються *батьківськими пропозиціями*, а S - *резольвентою*, що утворюється в результаті застосування резолюції до S_1 і S_2.

Резолюція в цьому найпростішому випадку відповідає правилу виводу modus tollens, що записується у виді:

¬ A A← B

¬ B

і відповідає наступній умові:

допускаючи, що: **НЕ А і А ЯКІЩО У,**

виводимо: **НЕ У**

У ще більш простому випадку, коли S_1 - заперечення, а S_2 - факт, тобто:

S_1: ¬ **A**

S_2: **A,**

застосування правила резолюції дасть резольвенту - *порожнє заперечення*

S: □

і означає протиріччя. Даний крок виводу може бути записаний у виді:

¬ A, A

. □

і резолюцією є міркування типу

допускаючи, що: **НЕ А і А**

виводимо протиріччя.

Правило резолюції для складних пропозицій

Реальні логічні моделі містять значно більш складні пропозиції. Так заперечення можуть містити декілька предикатів, так само як і праві частини імплікацій. Тому більш загальним є випадок, коли батьківські пропозиції мають вид:

S_1: $\neg (A_1, \ldots, A_k, \ldots, A_n)$

S_2: $A_k \leftarrow (B_1, \ldots, B_m)$ **(де 1<k<n).**

Тут деякий предикат A_k із заперечення S1 збігається з предикатом із лівої частини S_2. У цьому випадку крок виводу заміняє A_k у S_1 на праву частину S_2 і в якості резольвенти отримують

$$S: \neg (A_1, \ldots, A_{k-1}, B_1, \ldots, B_m, A_{k+1}, \ldots, A_n).$$

Приклад.

допускаючи, що **Не (темно і зима і холодно)**

і що **Зима якщо Січень**

виводимо, що **НЕ (темно і січень і холодно)**

У тому випадку, якщо S_1 має той же вид, а S_2 - факт

$$S_2: A_k,$$

причому A_k є одним із предикатів із S_1, крок виводу тільки викреслює A_k із S_1 і утворюється резольвента

$$S: \neg (A_1, \ldots, A_{k-1}, A_{k+1}, \ldots, A_n).$$

Даний крок виводу можна ілюструвати наступним прикладом

допускаючи, що НЕ (темно і зима і холодно)

і що Зима

виводимо, що: **НЕ (темно і холодно)**

Проста резолюція зверху вниз

Розглянуті вище правила застосовуються на кожному кроку виводу тільки до двох батьківських речень. Водночас опис будь-якої області

знання містить множину ППФ. Розглянемо процес логічного виводу для приклада, коли знання виражаються двома пропозиціями.

S_2: одержує (студент, стипендію) ← здає (успішно, сесію, студент)

S_3: здає (успішно, сесію, студент).

Задача полягає в тому, щоб відповісти на запит

чи одержує студент стипендію.

Коли використовується звичайна система логічного виводу, то таке питання представляється у виді заперечення

S: ¬ одержує (студент, стипендію)

і система повинна відкинути це заперечення за допомогою інших пропозицій, демонструючи, що дане допущення веде до протиріччя. Цей підхід часто застосовується в математиці і називається *доказом від противного*.

Тепер уявимо собі, що вихідна логічна модель, складена з трьох пропозицій S_1, S_2 ,S_3 , надходить на вхід системи логічного виводу ЕОМ.

КРОК 1

Система на першому кроку застосує правило (*) до батьківських пропозицій S_1 і S_2 і одержить резольвенту:

S: ¬ здає (успішно, сесію, студент).

КРОК 2

Використовуючи правило (**) до S і S3 система виводить протиріччя:

S': □.

Таким чином, для доказу суперечливості S_1 ,S_2, S_3 виявилося досить двох кроків виводу. Якщо вважати, що S_2 і S_3 не суперечать один одному , то вони спільно суперечать S_1, тобто

підтверджують заперечення S: ¬ S_1: ¬ (¬ одержує (студент, стипендію))

або іншими словами підтверджують пропозицію:

одержує (студент, стипендію)

і відповіддю на вихідну задачу є «**ТАК**».

Логічний вивід, що породжує послідовність заперечень, таких як (S1 , S , S') у розглянутому прикладі, називається *резолюцією зверху вниз*.

Загальна резолюція зверху вниз

У загальному випадку предикати і ППФ в якості термів містять не тільки константи, але і змінні і функції. Розглянемо дві батьківських пропозиції:

S_1: **¬ одержує (студент, У)**

S_2: **одержує (X, стипендію) ← здає (успішно, Z, X)**

До них безпосередньо не можна застосувати правило резолюції, тому що вони не містять однакових предикатів у лівій частині імплікації та у запереченні. Дані пропозиції містять три змінні X, Y, Z, що неявно універсально квантировані.

Розглянемо першу пропозицію, S_1 , яка стверджує, що:

для усіх У студент не отримує У.

При інтепретації S_1 і S_2 принаймні один індивідуум **У** буде зв'язаний з іменем «**стипендія**» і тому безпосереднім слідством S_1 є більш конкретна пропозиція:

S_1': **¬ одержує (студент, стипендію).**

Аналогічно розглядається S_2 на області інтепретації S_1 і S_2 і, обираючи для **X** індивідуум з ім'ям «**студент**», одержуємо більш конкретну пропозицію:

S_2': **одержує (студент, стипендію) ← здає (успішно, Z, студент).**

Тепер маємо дві пропозиції S_1' і S_2', що задовольняють умові придатності правила резолюції. Тому резольвентою S_1' і S_2' буде:

S: **¬ здає (успішно, Z, студент)**

Предикат

одержує (студент, стипендію)

називається *загальним прикладом батьківських предикатів*

одержує (X, стипендію)

одержує (студент, У)

і отриманий за допомогою *уніфікатора* виду

Θ = { X:= студент, У:= стипендія}.

Уніфікатори і приклади уніфікації

Уніфікатором називається множина присвоювань виду

$$\Theta = \{X_1 := t, \ldots, X_n := t_n\},$$

де X_i - змінна, а t_i - терм, застосування яких до двом висловам дає однакові загальні приклади. На практиці уніфікатори визначають, порівнюючи по черзі відповідні аргументи предикатів і виписуючи ті присвоювання термів змінним, які б зробили б ці аргументи однаковими.

Розглянемо ряд прикладів уніфікації (таблиця 6.3):

Таблиця 6.3 Приклади уніфікації

Батьківські пропозиції	**Уніфікатор**	**Загальний приклад**
p(5), p(5)	**Θ** - порожня множина (не заміняється жодна змінна)	p(5)
p (x), p(5)	**Θ** = {x:=5}	p(x)**Θ** = p(5)**Θ** = p(5)
p(x), p(y)	**Θ** = {x:=y}	p(y)
p(x, y), p(5, x)	**Θ** = {x:=5, y=x} = = {x:=5, y:=5}	p(x,y)**Θ** = p(5,x)**Θ** = p(5,5)
¬ p(5, x) p(x, y) ← q(x)	**Θ** = {x:=5, y:=5}	p(5,5) резольвента: s: ¬ q(x)**Θ** = ¬ q(s)

Розв'язування задачі

Рішення задачі з використанням логічного програмування розбивається на 3 етапи:

На першому етапі необхідно сформулювати наші знання і допущення про предметну область у виді множини ПФ

На другому етапі потрібно висловити конкретну задачу, поставлену на предметній області, як запит про один або декількох відношеннях. Звичайно запит ставиться як вихідне заперечення.

Упорядкуванням допущень і вихідного запиту завершується робота програміста, ціль якої - сформулювати задачу.

КРОК 3 виконується комп'ютером, який намагається вирішити задачу, будуючи доказ від противного. Він будує вивід зверху вниз, починаючи з вихідного заперечення, і породжує послідовність заперечень

$$D_1, D_2, \ldots, D.$$

Якщо може бути побудований вивід, що закінчується запереченням

Dn = □ (протиріччя) ,

те цей вивід, називаний *успішним виводом*, оскільки відразу дає рішення поставленої задачі.

Контрольні питання

1. Який приорітет мають логічні зв'язки ?
2. Що називають правильно побудованою формулою ?
3. На які етапи розбивається рішення задачі з використанням логічного програмування на ?

6.4 Семантичні мережі

Семантичною мережею є структура даних, що має визначений зміст як мережа. Стандартного визначення семантичної мережі не існує, але звичайно під ним розуміють наступне [18-20]:

Семантична мережа - це система знань, що має визначений зміст у виді цілісного образу мережі, вузли якої відповідають поняттям і об'єктам, а дуги - відношенням між об'єктами.

Отже, різні мережі можна розглядати як мережі, що входять до складу семантичної мережі. У тому числі до них можуть бути віднесені і мережні структури моделей БД. Сама по собі семантична мережа є моделлю пам'яті і не розкриває, яким чином здійснюється представлення знань. Тому в контексті знайомства із СОЗ семантичні мережі повинні розглядатися як метод представлення знань із можливостями структурування цих знань, процедурами їхнього використання і механізмом виводу.

У ієрархічній структурі понять існують відношення, принаймні, двох типів: відношення включення або співпадання (IS - A); відношення «ціле - частина» (PART - OF).

Наприклад, у пропозиції

«людина» IS - A «савець»

основною думкою є те, що людина належить до класу ссавців. Це означає, що має місце відношення, включення або співпадання. Для цих відношень характерним є те, що екземпляри понять нижнього рівня містять всі атрибути понять верхнього рівня. Ця властивість називається у*спадкуванням атрибутів* між рівнями ієрархії IS - A. .

Відношення «ціле - частина» можна ілюструвати пропозицією

«ніс» PART - OF «тіло»,

що характеризує те, що екземпляри поняття «ніс» є частиною будь-якого екземпляра поняття «тіло».Найбільше часто використовується графічне представлення семантичних мереж у виді діаграми. Таке речення

«усі ластівки - птахи»

можна представити графом, що містить дві вершини ,які відповідають поняттям, і дугу, що вказує відношення між ними (рис.6.7).

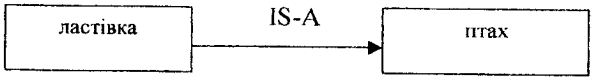


Рис. 6.7 – Семантична мережа - 1

Якщо ластівка має конкретне ім'я, наприклад, Ласта, то семантична мережа може бути розширена (рис. 6.8).

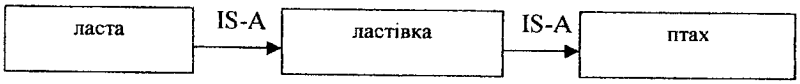


Рис.6.8.- Семантична мережа - 2

Поряд із тим, що за допомогою даної мережі описані два факти

«Ласта - ластівка»

«ластівка – птаха»,

із неї можна зробити висновок

«Ласта - птах».

Цей факт показує, що спосіб представлення семантичною мережею дозволяє легко робити висновки завдяки ієрархії спадкування.

Семантичними мережами можна також представляти знання, що стосуються атрибутів об'єкта. Наприклад, факт «Птахи мають крила» можна відобразити у вигляді рис. 6.9.

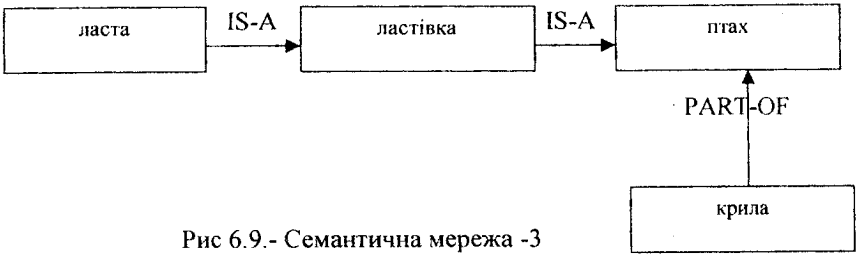


Рис 6.9.- Семантична мережа -3

Це означає, що, використовуючи відношення «IS - A» і «PART - OF» можна вивести факт «Ласта має крила».

Вершини в семантичній мережі звичайно показують об'єкт проблемної області, концепт, ситуацію і т.п., а дуги - це відношення між ними. При розширенні семантичної мережі в ній виникають додаткові відношення. Наприклад, якщо розглянуту мережу доповнити фактами «Ласта володіє гніздом» і «Ласта володіє гніздом із весни по осінь», те одержимо семантичну мережу, зображену на рис. 6.10. Тут гніздо - це конкретне гніздо, яким володіє Ласта, а для вершини ситуації (володіє j) визначено декілька зв'язків. Така вершина називається надійною рамкою і визначає різні аргументи предиката ситуації.

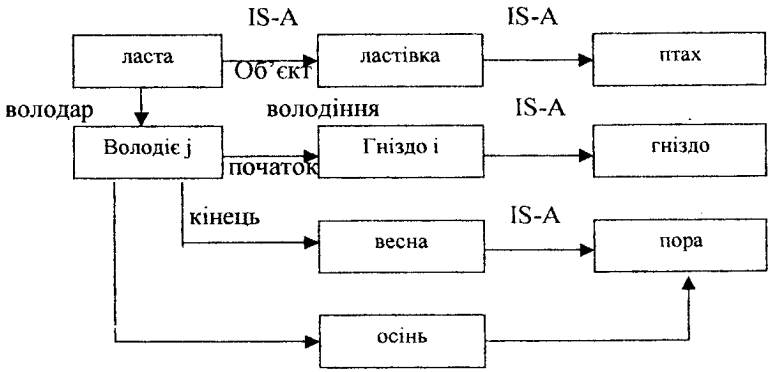


Рис.6.10 - Семантична мережа – 4

Найважливішою концепцією формалізму семантичних мереж є ієрархія понять і зв'язане з ній успадкування атрибутів між рівнями ієрархії IS - A. Якщо семантичну мережу розглядати як опис відношень, що підтримуються між поняттями, то її безпосередньо можна реалізувати на мові Пролог.

На рис.6.11 подана структура мережі, аналогічна прикладу попереднього розділу.

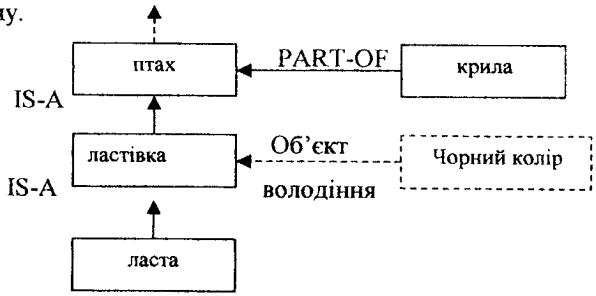


Рис. 6.11- Семантична мережа – 5

Ця мережа може бути реалізована в Пролог - програмі.

є (ластівка, птиця)

є (Ласта, ластівка)

має (крила, птиця)

має (X,Y): - є (Y,Z), має (X, Z).

/* враховує ієрархію спадкування */.

Для врахуванні в моделі знань, поданих семантичною мережею, такого властивості всіх ластівок, що вони чорного кольору, у програму досить додати факт:

Має (чорний_колір, ластівка).

Якщо модель знань буде доповнена загальною властивістю для всього класу птахів, такою, що вони літають, те це приведе до додавання в програму не тільки факту

літає (птиця),

але і правила, що повинно реалізувати ієрархію спадкування, тобто

літає (X):- є (X, Y), літає (Y).

Семантична мережа представляє собою орієнтований граф із позначеними (пойменованими) дугами і вершинами. Основними елементами мережі є вершини і дуги. При цьому вершинам семантичної мережі відповідають поняття, події і властивості (рис. 6.12).

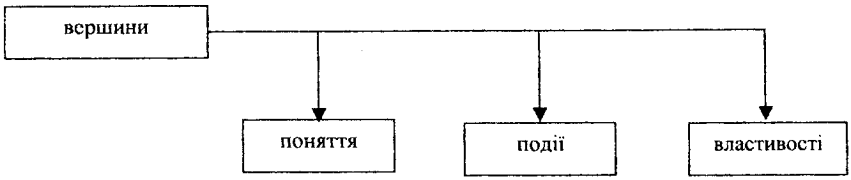


Рис.6.12- Вершини семантичної мережі

Поняття - це зведення про абстрактні або фізичні об'єкти предметної області (реального світу).

Події – це дії, що відбуваються в реальному світі і визначають:

- вказівка типу дії;
- вказівка ролей, що відіграють об'єкти в цій дії.

Властивості використовуються для уточнення понять і подій. Стосовно до понять вони описують їхньої особливості і характеристики (колір, розмір, якість), а стосовно до подій - тривалість, час, місце.

Дуги графа семантичної мережі відображають різноманіття семантичних відношень, що умовно можна розділити на чотири класи (рис.6.13).

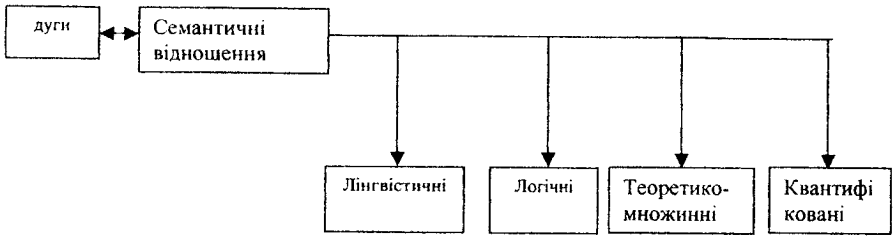


Рис.6.13 - Класифікація семантичних відношень

Лінгвістичні відношення відображають взаємозв'язок між подіями, між подіями і поняттями або властивостями. Лінгвістичні відношення бувають:

- дієслівні (час, вид, рід, застава, нахилення);
- атрибутивні (колір, розмір, форма);
- відмінковими (див. нижче).

Логічні відношення - це операції алгебра логіки: диз'юнкція, кон'юнкція, інверсія, імплікація.

Теоретико-множинні - це відношення підмножини, відношення частини цілого, відношення множини та елемента. Прикладами таких відношень є IS-A, PART-OF.

Квантифікування відношення - це логічні квантори спільності та існування. Вони використовуються для представлення таких знань як «Будь-який верстат треба ремонтувати», «Існує працівник А, який обслуговує склад Б».

Розглянуті вище приклади семантичних мереж відображали знання про структуру понять і їхніх взаємозв'язків. Далі розглянемо використання семантичних мереж для уявлення подій і дій.

Основою для визначення будь-якого поняття є множина його відношень з іншими поняттями. Обов'язковими відношеннями є:

- клас, якому належить дане поняття;

- властивості, що виділяють поняття з усіх понять даного класу;
- приклади (екземпляри) даного поняття.

Оскільки терми, які використовують у визначенні поняття, самі є поняттями, то вони організується по тієї ж схемі. У підсумку зв'язки понять утворять структуру, в загальному випадку мережну, в якій використовується як мінімум два типи зв'язків (IS - A і PART - OF).

Приклад: Семантична мережа, що відображає зв'язки понять при описі знань про структуру "юридична особа " має вигляд (рис.6.14):

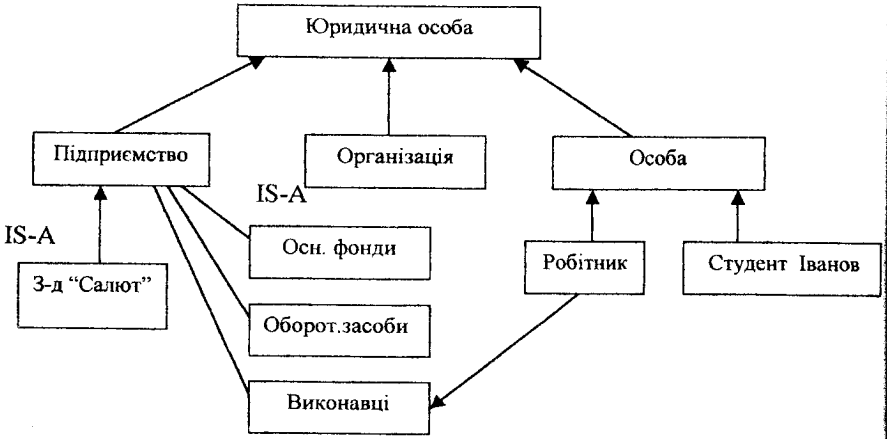


Рис.6.14 - Приклад семантичної мережі

При представленні подій попередньо виділяються прості відношення, що характеризують основні компоненти події. У першу чергу з подій виділяється *дія*, що описуються *дієсловом*. Далі визначаються: об'єкти, що діють; об'єкти, над котрими ці дії виконуються.

Всі зв'язки понять, подій і властивостей із дією (дієсловом) називають ***відмінками*** або ***відмінковими відношеннями***

Таблиця 6.4 Основні відмінки

Відмінок	**Лінгвістичне (відмінкове) відношення, що визначає зв'язок дії з:**
Агент	• предметом, що є ініціатором дії
об'єкт	• предметом, що піддається дії
Джерело	• розміщенням предмета перед дією
Приймач	• розміщенням предмета після дії
Час	• моментом виконання дії
Місце	• місцем проведення дії
Ціль	• дією іншої події

Так, наприклад, семантична структура знання про подію **«Директор заводу «Салют» зупинив 30.03.96 цех №4, щоб замінити устаткування»** буде подана у виді рис. 6.15..

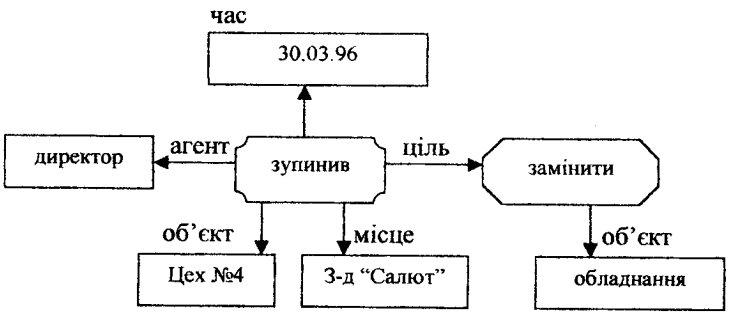


Рис. 6.15 - Приклад семантичної структури

Особливість семантичної мережі як моделі знань полягає в єдності БЗ і механізму виводу. При формуванні запиту до БЗ:

4. будується семантична мережа, що відображає структуру запиту;

5. вивід забезпечується за рахунок зіставлення загальної мережі БЗ і мережі для запиту.

Розглянемо речення: **«Якщо верстат закінчив обробку, робот вантажить касету з деталями на робокар, що перевозить їх на склад».**

Виділимо основні факти цих знань, що відповідають діям:

F1 - верстат закінчив обробку,

F2 - працівник вантажить,

F3 - робокар перевозить,

F4 - касета містить деталі.

Схема семантичної мережі буде наступною (рис. 6.16):

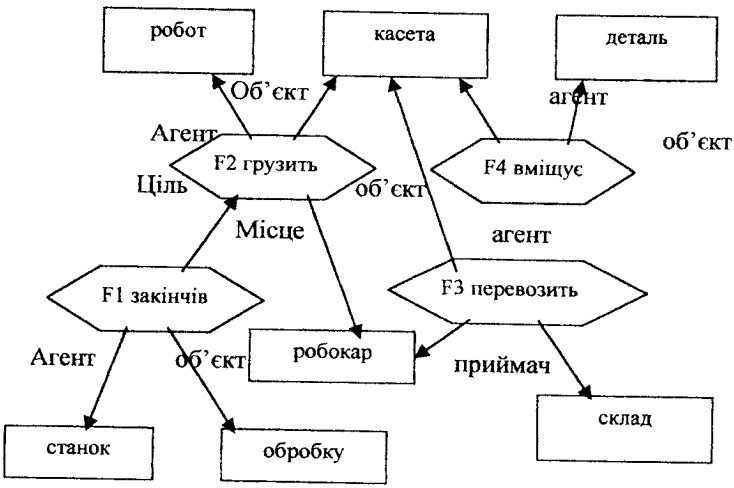


Рис. 6.16- Побудова семантичної мережі

Необхідно відзначити ряд переваг семантичної мережі:

- опис понять і подій робиться на рівні, дуже близькому до природної мови; забезпечується можливість зчеплення різних фрагментів мережі;
- відношення між поняттями і подіями утворять досить невелике і добре формалізовану множину;
- для кожної операції над даними і знаннями можна виділити з повної мережі, що представляє всю семантику (або всі знання), деякий її ділянка, що охоплює необхідні в даному запиті значеннєві характеристики.

Контрольні питання

1. Назвіть основні елементи семантичної мережі .
2. Які відношення являються обов'язковими ?
3. Які дії відносять до підмета ?
4. Приведіть приклад семантичної мережі.

6.5 Продукційні моделі

Продукційні моделі - це найбільш поширені на даний день моделі, в яких знання представляються за допомогою правил вигляду [20]:

ЯКЩО - ТО (явище - реакція)

При використанні таких моделей у системах, заснованих на знаннях є можливість:

- застосування простого і точного механізму використання знань;
- представлення знань з високою однорідністю, що описуються по єдиному синтаксису.

Ці дві відмінні риси і визначили широке поширення методів подання знань правилами.

Програмні засоби, що оперують із знаннями, поданими за допомогою правил, одержали назву ***продукційних систем*** (або ***систем продукції***) і вперше були запропоновані Постом у 1941 році.

Продукція в системі Поста має таку схему:

$$\frac{t_1, t_2, \ldots, t_n}{t}, \qquad (*)$$

де $t_1, t_2, \ldots, t_n$ - посилки;

t - висновок.

Застосування схеми Поста (*) ґрунтується на підстановці ланцюжків знаків замість змінних, причому замість входження однієї і тієї ж змінної подається той самий ланцюжок.

Системи продукції складаються із трьох елементів:

а) набір правил, що використовуються як БЗ, його ще називають базою правил;

б) робоча пам'ять, де зберігаються передумови, що стосуються окремих задач, а також результати висновків, одержаних на основі цих передумов (динамічна база даних - ДБД);

с) механізм логічного висновку, що використовує правила відповідно до вмісту робочої пам'яті.

Конфігурацію систем продукції спрощено можна подати в наступному вигляді рис. 6.17.

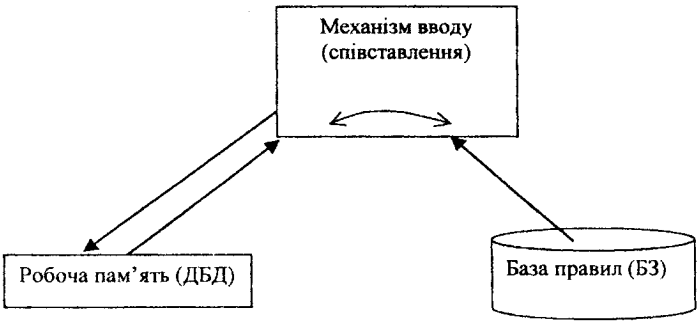


Рис. 6.17 - Конфігурація продукційної системи

На простому прикладі розглянемо спрощено механізм функціонування систем продукції. Нехай дані, записані в робочу область, являють собою зразки у вигляді набору символів:

«намір - відпочинок»

«місце відпочинку - гори»

Ці зразки відповідають фактам «намір IS відпочинок» і «місце відпочинку IS гори».

Правила віддзеркалюють вміст робочої пам'яті. В їхній умовній частині знаходяться або одиночні зразки, або декілька умов, з'єднаних «І», а в заключній частині - зразки, що додатково реєструються в пам'яті:

Правило №1:

ЯКЩО «намір - відпочинок» І «дорога - вибоїста»,

ТО «використовувати - джип».

Правило №2:

ЯКЩО «місце відпочинку - гори»,

ТО «дорога - вибоїста».

Тоді після того, як у робочу пам'ять записуються зразки і в базу - правила, розглядається можливість застосування цих правил. Для цього механізм висновку порівнює зразки з умовної частини правила зі зразками, що зберігаються в робочій пам'яті. Якщо всі зразки є в робочій пам'яті, умовна частина вважається правдивою, у протилежному випадку - помилковою.

Для розглянутого приклада робоча пам'ять і база правил буде заповнена, як це зображено на рис. 6.18.

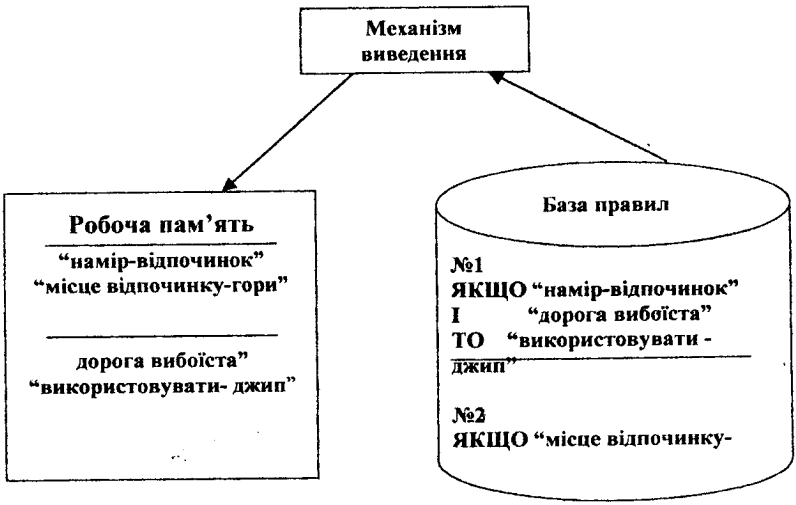


Рис. 6.18- Прямий ланцюжок міркувань

Для розглянутого прикладу послідовність логічного висновку буде такою:

1.Механізм виводу аналізує правила, починаючи з першого, визначає наявність зразка «намір - відпочинок» у робочій пам'яті і відсутність в ній зразка «дорога - вибоїста».

2.Умовна частина правила №1 вважається помилковою, і механізм висновку переходить до наступного правила (в нашому випадку до правила №2).

3.Умовна частина правила №2 признається правдивою, тому що зразок «місце відпочинку - гори» є присутнім у робочій пам'яті і механізм висновку переходить до виконання його заключної частини.

4.Заключна частина правила №2 «дорога - вибоїста» заноситься в робочу пам'ять.

5.Після перегляду всіх правил відбувається вторинне їхнє застосування, починаючи з першого правила, за винятком тих, що вже були застосовані (у прикладі це правило №2).

6.При повторному співставленні правила №1 його умовна частина стає правдивою через довизначення робочої пам'яті, і механізм висновку виконує його заключну частину.

6. Заключна частина «використовувати - джип» переноситься в робочу пам'ять, а правило №1 виключається з подальшого узгодження.

7. Правил для співставлення не залишається, і система зупиняється.

Якщо тепер звернутися до робочої пам'яті, то виходячи з посилок що

«наміри - відпочинок» і

«місце відпочинку - гори»

результатом висновку є рекомендація

«використовувати - джип»

із поясненням причин даного висновку, що визначається тим, що

«дорога - вибоїста»

В даному прикладі для одержання висновку проводилася робота:

- багатократного перегляду вмісту бази правил;
- послідовного застосування правил на основі попередньо записаного вмісту робочої пам'яті;
- доповнення даних, що поміщаються в робочу пам'ять.

Такі висновки називаються *прямими* (прямий ланцюжок міркувань). Навпаки, спосіб, при якому на підставі фактів досліджується можливість застосування правила, придатного для підтвердження, називається *зворотним висновком* (зворотний ланцюжок міркувань).

Для пояснення цього способу звернемося до знайомого прикладу. Метою запиту до системи є факт встановлення доцільності використання джипу при відпочинку в горах. Вважаючи, що робоча пам'ять містить зразки «наміри - відпочинок» і «місце відпочинку - гори», а база містить

обидва правила, метою упорядкування є доказ факту «використовувати - джип». Тобто в цьому випадку робоча пам'ять має вихідний вигляд (рис. 6.19).

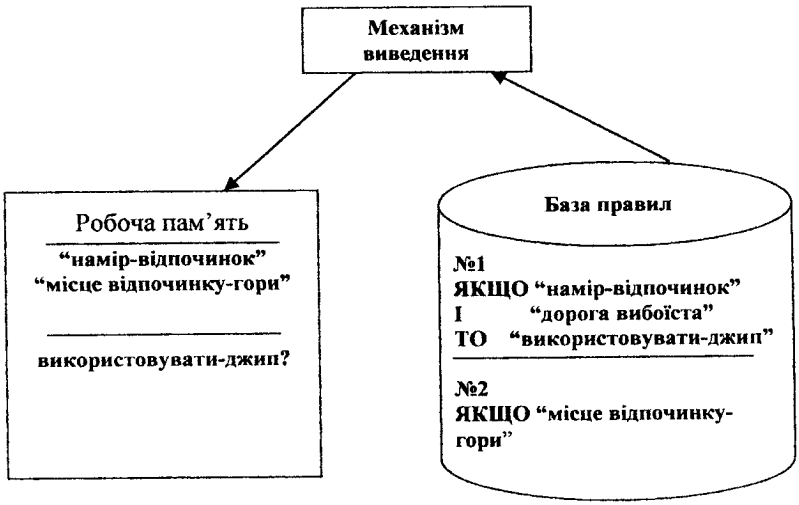


Рис. 6.19- Зворотний ланцюжок міркувань

Послідовність впорядкування системою продукції така:

1. Визначається правило, в якому в заключній частині міститься цільовий факт.
2. Досліджується можливість застосування першого правила для підтвердження вихідного факту.
3. Оскільки зразок «намір - відпочинок» з умовної частини правила №1 занесений у робочу пам'ять, то для досягнення мети досить підтвердити факт «дорога - вибоїста».
4. Зразок «дорога - вибоїста» приймається за нову мету, і необхідно знайти правило, що підтверджує цей факт.

5. Досліджується можливість застосування правила №2. Умовна частина цього правила є правдивою, тому що зразок «місце відпочинку - гори» є в робочій пам'яті.
6. Беручи до уваги можливість застосування правила №2, робоча пам'ять поповниться зразком «дорога - вибоїста» і з'явиться можливість застосування правила №1 для підтвердження мети «використовувати - джип».

Таким чином, результатом висновку є підтвердження мети «використовувати - джип» за умови «дорога - вибоїста».

Контрольні питання

1. Які елементи входять до продукційної моделі ?
2. Приведіть конфігурацію продукційної моделі .
3. Приведіть приклад продукційної моделі.
4. В чому полягає зворотній ланцюг міркувань ?

6.6 Подання знань із застосуванням фреймів

У складних семантичних мережах, що включають множину понять, процес відновлення вузлів і контроль зв'язків між ними стає досить складним. При цьому кількість опосередкованих родовидних зв'язків між поняттями різко зростає. [19,20]

Основна ідея фреймового підходу до подання знань полягає в тому, що все, що стосується поняття або ситуації, не «розмивається по мережі», а представляється у фреймі.

Фреймом називається структура для опису поняття або ситуації, що складається з характеристик цієї ситуації і їхніх значень. Фрейм можна розглядати як фрагмент семантичної мережі, призначений для опису

понять із усією сукупністю характерних їм властивостей. Поняття про діловий звіт у системі, заснованій на фреймах, може мати такий вигляд (рис.6.20).

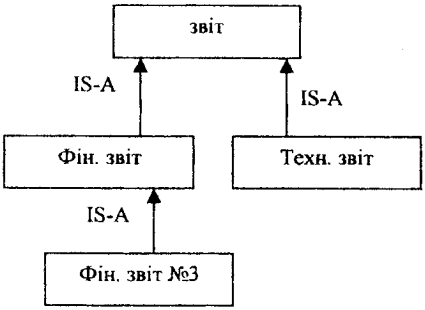


Рис. 6.20- Діловий звіт у системі, заснованій на фреймах

Графічно це виглядає аналогічно семантичній мережі, але принципова відмінність є в тому, що кожний вузол у фреймовій системі має вигляд рис.6.21.

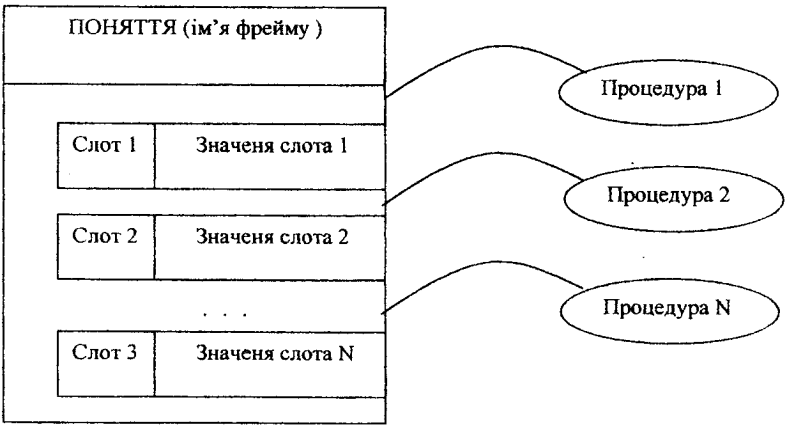


Рис. 6.21- Фрейм

Автор ідеї фреймового підходу Марвін Мінський дав таке визначення: «Фрейм - це структура даних, що являє собою стереотипну ситуацію, начебто присутності в середині житлової кімнати. До кожному фрейму приєднується декілька видів інформації. Частина інформації про те, як використовувати фрейм; частина про те, що очікується далі; частина про те, що варто робити, якщо сподівання не підтвердяться».

У кожному вузлі поняття визначаються набором атрибутів і їхніми значеннями, що містяться в слотах фрейму. *Слот* - це атрибут, пов'язаний із вузлом у системі, заснованій на фреймах.

Слот є складовою фрейму. Ім'я слоту відповідає типу атрибута, значенням слоту може бути екземпляр атрибута, інший фрейм або фасет. З кожним слотом може бути зв'язана одна або декілька процедур, що виконуються, коли змінюються значення слотів. Частіше усього зі слотами зв'язують процедури:

1.Якщо - додано (виконується, коли нова інформація міститься в слоті);
2. Якщо - віддалено (виконується при видаленні інформації із слоту);
3. Якщо - потрібно (виконується, коли запитується інформація із слоту, а він порожній).

Ці процедури можуть стежити за приписуванням інформації до даного вузла і перевіряти, як при зміні значення робляться відповідні дії.

Для ілюстрації роботи цього класу СОЗ розглянемо ієрархію поняття звіту, структура якого визначена вище, але вже з усіма слотами, їхніми значеннями і процедурами (рис. 6.22).

Нехай деякі слоти мають значення по замовченню. Яким способом можна використовувати так організовані дані? Якщо системі заданий запит: «Мені потрібний фінансовий звіт про виконання проекту по новій технології», то інтерфейсна програма аналізує його :

1.Вносить «проект по новій технології» у слот «ТЕМА» наступного порожнього вузла «Фінзвіт» (у нашому прикладі це вузол №3). Далі все відбувається автоматично:

2.Процедура «якщо - додана», зв'язана зі слотом «ТЕМА», здійснює пошук керівника цього проекту. Нехай його прізвище Іванов. Процедура вписує його прізвище в слот «АВТОР» фінансового звіту №3. Якщо керівник цієї теми не буде знайдений, у слот «АВТОР» буде успадковане значення класу, а саме текст «КЕРІВНИК ПРОЕКТУ».

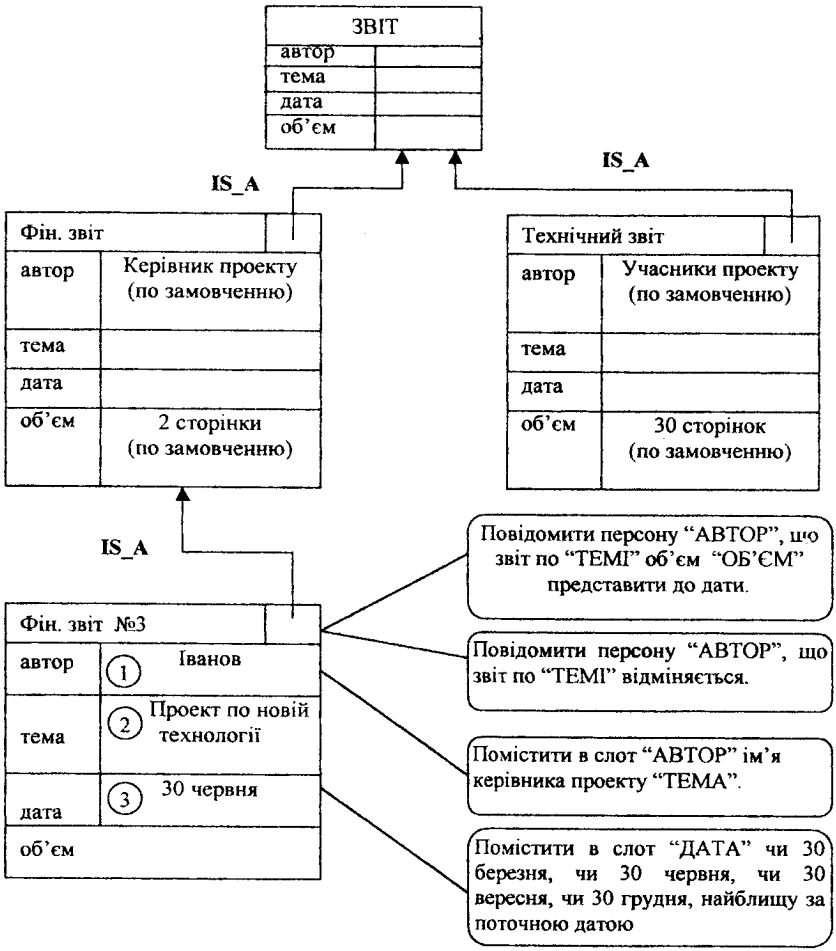


Рис. 6.22- Функціонування фреймової системи

3.Процедура «якщо - додана», продивляючись слот «ДАТА» і виявивши його порожнім, активізує процедуру «якщо - потрібно», зв'язану з цим слотом, який, аналізуючи поточну дату (наприклад 12.04.96), вирішить, що «30 червня» найближче до неї і впише цю дату в слот «ДАТА».

4.Тепер процедура «якщо - додана», зв'язана зі слотом «АВТОР», знайде, що ще одне значення, що потрібно включити в повідомлення, тобто об'єм звіту, а саме - відсутніх. Слот «ОБ'ЄМ» не зв'язаний із процедурами і нічим допомогти не може. Однак вище вузла № 5 існує вузол загальної концепції фінансового звіту, що містить значення об'єму. Процедура, використовуючи концепцію спадкування властивостей класу, використовує значення об'єму і складає наступне повідомлення: «Пан Іванов, підготуйте фінансовий звіт по проекту нової технології до 30 червня об'ємом 2 сторінки».

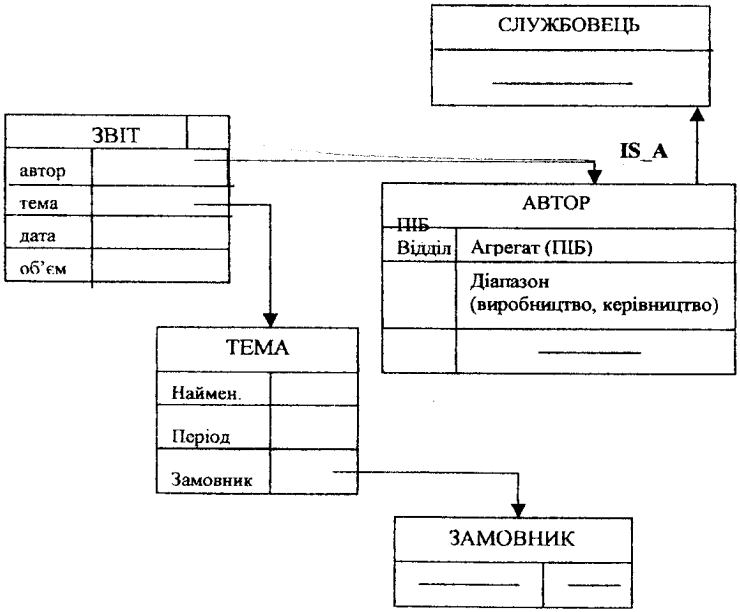


Рис. 6.23- Функціонування фреймової системи

Якщо в певний момент прізвище Іванов буде вилучено зі слоту «АВТОР», то система автоматично відправить йому повідомлення, що його звіт не потрібен. У розглянутому прикладі заповнювачами слугували конкретні екземпляри атрибутів (що заповнюються по замовченню). Водночас, заповнювачами слотів можуть бути:

- імена інших фреймів системи, на які робиться посилання;
- фасети («агрегат», «діапазон», «по замовченню» і ін.).

Розширимо розглянутий вище приклад для ілюстрації цих понять (рис. 6.23.):

При визначенні фреймів фасет «агрегат» вказує на те, що повинні бути задані необхідні об'єкти, а «інтервал» - на те, що повинний бути обраний один із множини об'єктів. Ця інформація використовується інтерфейсною програмою при введенні інформації.

У загальному випадку фрейм являє собою таблицю, структура і принципи організації якої є розвитком поняття відношення в реляційній моделі даних. Узагальнена структура фрейму має вигляд таблиці 6.5.

Слотом фрейму називається елемент даних для фіксації знань про об'єкт, якому відведений даний фрейм.

Таблиця 6.5. – Структура фрейму

ІМ'Я ФРЕЙМУ			
Ім'я слоту	Покажчик спадкування	Покажчик типу	Значення слоту
слот 1			
слот 2			

слот N			

Перерахуємо параметри слотів:

Ім'я слоту. Кожний слот повинен мати унікальне ім'я у фреймі, до якого він належить. Ім'я слоту в деяких випадках може бути службовим. Серед службових імен можуть бути:

- ім'я користувача, що визначає фрейм;
- дата визначення або модифікації фрейму;
- коментар.

Покажчик_спадкування. Він показує, яку інформацію про атрибути слотів у фреймі верхнього рівня успадковують слоти з тими ж іменами у фреймі нижнього рівня. Типові покажчики спадкування:

- **S** (той же) - слот успадковується з тими ж значеннями даних;
- **U** (унікальний) - слот успадковується, але дані в кожному фреймі можуть приймати будь-яке значення;
- **I** (незалежний) - слот не успадковується.

Покажчик типу даних. Типом даних, що включаються в слот, можуть бути:

- FRAME (покажчик) - вказує ім'я фрейму верхнього рівня.
- ATOM (перемінна);
- TEXT (текстова інформація);
- LIST (список);

LISP (приєднана процедура).

За допомогою механізму керування дослідженням із відношень IS_A здійснюється автоматичний пошук і визначення значень слотів фрейму верхнього рівня і приєднаних процедур.

Контрольні питання

1. В чому полягає основна ідея фреймового підходу ?
2. Приведіть графічне представлення конфігурації фрейма.
3. Для чого в фреймові структури вводять слоти ?

6.7 Стратегії пошуку в СОЗ

СОЗ, в тому числі і експертні системи, мають специфічну організацію. Суть якої полягає в тому, що вони здійснюють пошук деякої мети (тобто кінцевого стану) на основі деяких вихідних посилок і набору фактів (тобто початкового стану) (рис. 6.24). [18-20]

Рис. 6.24-Пошук в СОЗ

При чому пошук кінцевого стану виконується автоматично на основі реалізованої в СОЗ стратегії пошуку, що:

- реалізує можливість вибору;
- дозволяє виконувати кроки від початкового стану до нових станів, більш-менш близьких до мети.

Таким чином, реалізовані в СОЗ стратегії пошуку відшукують мету, «крокуючи» від одного стану системи до іншого. При цьому вони забезпечують розпізнавання ситуації, коли вони знаходять мету або потрапляють в тупик.

Як правило, на проміжних стадіях обчислюється деяке число (критерій), за допомогою якого програма пошуку оцінює свій хід і визначає подальший напрямок пошуку необхідного кінцевого стану. При цьому мета може бути одна або ж може бути деякий набір прийнятних цілей (кінцевих станів). Більшість СОЗ працюють по цій моделі.

Розглянемо процес пошуку на конкретному прикладі. Нехай маємо карту доріг (рис. 6.25), по якій ми хочемо вивчити маршрут руху з одного конкретного міста в інший.

Система, побудована на основі знань про цю карту доріг, повинна попрацювати з картою і спланувати нашу подорож, наприклад із міста «А» у місто «F».

При рішенні поданої задачі комп'ютер перетворить вихідну карту і подасть її у вигляді дерева пошуку, що без врахування циклів буде мати вигляд рис. 6.26.

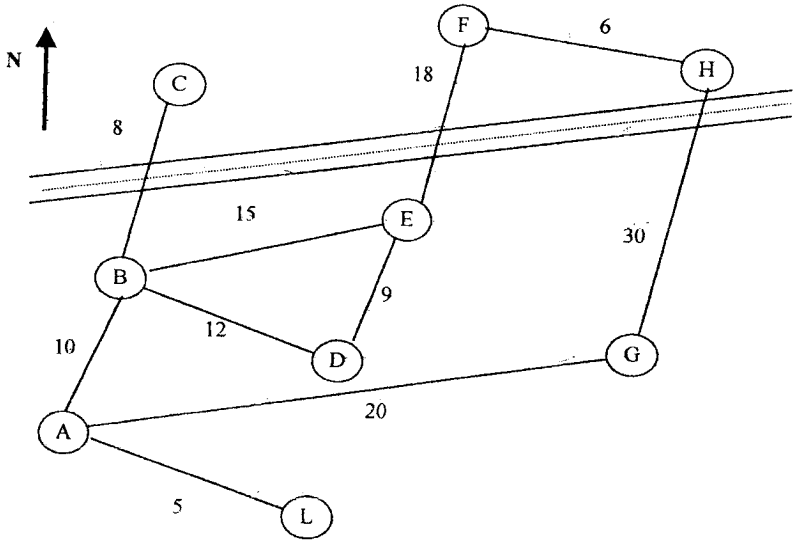


Рис. 6.25- Карта доріг

Дерево пошуку показує всі можливі варіанти вибору при руху з «А» (початковий стан), а також і усі варіанти вибору в кожному з проміжних пунктів (станів).

Місце, звідки починається пошук, знаходиться в вершині дерева пошуку. Усі дороги, що починаються у «А», або приводять у тупик, або закінчуються в «F».

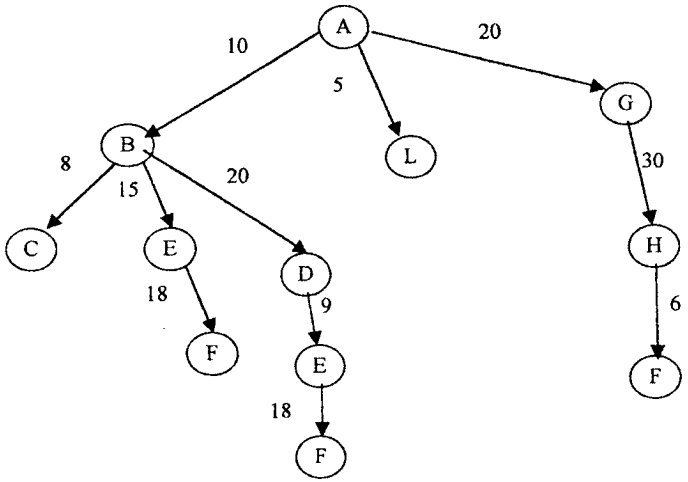


Рис. 6.26- Дерево пошуку

Задача комп'ютера полягає в тому, щоб знайти придатну дорогу з вершини дерева до кінцевого стану (мети) у його нижній частині.

Людина при погляді на дерево пошуку відразу ж вибере придатну дорогу, але комп'ютер на це не здатен. Яку стратегію пошуку застосує комп'ютер? Розглянемо приклад, що змусить нас думати як комп'ютер.

Ми знаходимося в темній кімнаті, на стіні якої дерево пошуку, а в нас ліхтарик, що висвітлює усього два сусідніх пункти.

Нам відомо де розташована вершина дерева, звідки треба почати пошук. Як найбільш ефективно переміщати промінь ліхтарика, щоб знайти

маршрут до «F»?

При відсутності якоїсь додаткової інформації комп'ютер для пошуку використовує стратегію «грубої сили». При цьому програма пошуку здійснює пошук мети, йдучи від стана до стана, і систематично досліджуючи всі можливі шляхи.

Для визначення кожного наступного кроку застосовується деяка проста механічна стратегія, що має два різновиди: пошук у глибину і пошук у ширину.

Пошук у глибину заснований на повному дослідженні одного варіанта до вивчення інших варіантів. Пошук виконується по нескладній методиці.

Наприклад, ЕОМ завжди першою досліджує невідому ліву гілку дерева. Коли процес пошуку заходить у тупик, він повертається вверх в останній пункт вибору, де є невивчені альтернативні варіанти руху, і потім здійснюється наступний варіант вибору (рис. 6.27).

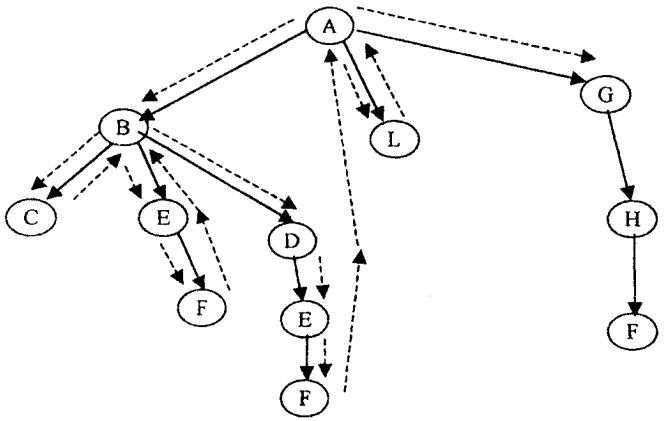


Рис. 6.27- Пошук у глибину

У середньому пошук у глибину підходить для рішення проблеми, де всі шляхи пошуку від вершини дерева до його основи мають однакову довжину.

Стратегія пошуку в ширину передбачає перехід у першу чергу до вершин, найближчих до стартової вершини (тобто таким, що відстають від неї на один зв'язок), потім до вершин, що відстають на два зв'язки, потім на трьох і т.д., поки не буде знайдена цільова вершина (рис. 6.28).

Програма пошуку просувається по дереву рішень зліва праворуч, розширяючи кожний із маршрутів, і відкидаючи ті з них, що є тупиковими.

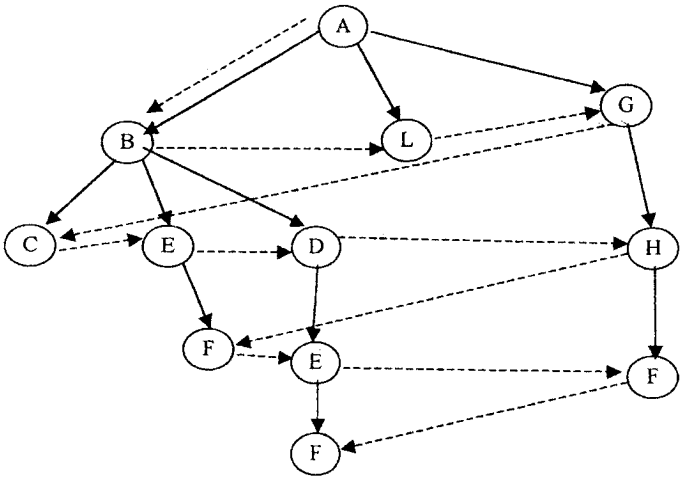


Рис. 6.28- Пошук у ширину

Таким чином пошук у ширину придатний при рішенні проблем, де гілки пошуку в дереві (від вершини до підстави) мають неоднакову довжину, і не існує вказівок на те, яка саме дорога приводить до мети. Обидві ці стратегії припускають послідовний перебір можливих варіантів.

Пошук буде більш ефективним, якщо деякий механізм у пунктах вибору самий зможе робити найбільш бажаний вибір. Це так звана евристика пошуку. Евристика - це емпіричне правило, за допомогою якого людина-експерт при відсутності формули або алгоритму намагається здійснити свої наміри.

Пошук у графах при рішенні задач, як правило, неможливий без рішення проблеми комбінаторної складності, що виникає через швидкий ріст числа альтернатив. Ефективним засобом боротьби з цим є евристичний пошук.

Один із шляхів використання евристичної інформації про задачу - це одержання чисельних евристичних оцінок для вершин простору стану. Оцінка вершини вказує, наскільки дана вершина перспективна з погляду досягнення мети. Ідея перебуває в тому, щоб завжди продовжувати пошук, починаючи з найбільш перспективної вершини, обраної з усієї множини кандидатів.

Існує цілий ряд евристичних методів. Найпростіший з яких зветься «підіймання на гору». Його принцип:

- якщо є прийнятні варіанти вибору, вибирайте найкращий з них, використовуючи будь-який критерій міркувань;
- якщо потрапили в тупик, повертайтесь в останнє місце, де є альтернативні варіанти вибору, і зробіть наступний найкращий вибір.

Щоб програма пошуку маршруту виконувала евристичний пошук, вона повинна мати у своєму розпорядженні базу для здійснення вибору. Припустимо, в якості бази програма прийме відповідний напрямок кожної дороги визначений азимутом: перевага буде віддаватися тій дорозі, що в найбільшій мірі збігається по напрямку з прямою, що з'єднує стартову і цільову вершини (рис. 6.29). Такий підхід не завжди діючий, але це добра стратегія. У нашому прикладі програма дослідить маршрут A □ B □ C і

знайде тупик, потім повернеться в те місце, де існує альтернатива вибору, тобто в «B», а потім відразу відправиться по маршруті B □ E □ F, що приведе до мети.

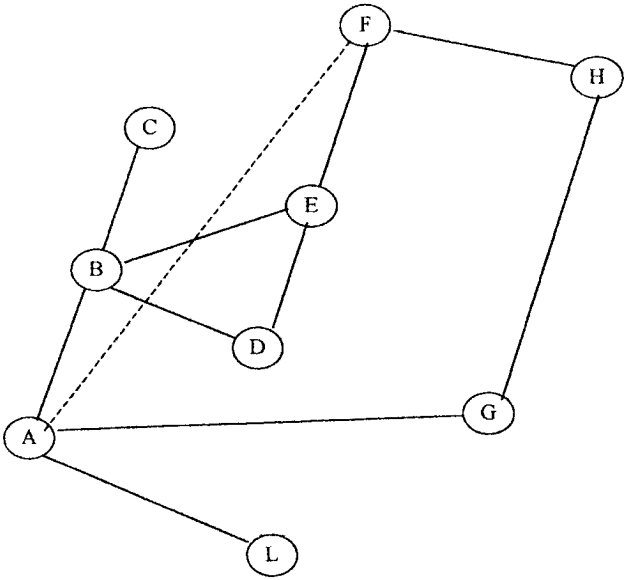


Рис. 6.29- Метод «підіймання на гору»

Ми розглянули різні варіанти пошуку, але виникає питання: звідки взялося саме дерево пошуку? Звичайно машина створює дерево пошуку сама на основі запропонованих їй початкових знань, причому робить це поступово в процесі самого пошуку.

Процес рішення задачі, як правило, включає два етапи: подання задачі і реалізація стратегії пошуку.

- Повне подання задачі в просторі станів включає: опис всіх або тільки початкових станів; завдання операторів, що відображують одні стани в інші;

- завдання цільового стану.

Пошук рішення задачі в просторі станів зводиться до визначення послідовності операторів, що відображують початковий стан у цільовий.

Таким чином, подання задачі в просторі станів визначається сукупністю трьох складових - трійкою

$$< S_0, F, G >,$$

де S_0 - множина початкових станів (може включати один елемент);

F - множина операторів, що відображують один стан в інший;

G - множина цільових станів.

Для опису станів можуть використовуватися вектори, матриці, графи, списки і т.д. Вибір тієї або іншої форми опису станів визначається застосуванням найпростішого оператора, що перетворить один стан в інший. При цьому оператори відображення (перетворення) часто подаються у вигляді набору правил, що задають можливість переходу з одного стану в інший (перетворення одних станів в інші).

Так, для розглянутого раніше приклада пошуку шляху на карті доріг, будь-який поточний стан можна описати у вигляді списку міст, пройдених до даного моменту (тобто включеного в маршрут пошуку на даний момент). Тоді початковий стан буде описуватися списком, що складається тільки з одного елемента, що відповідає початковому пункту шляху: **[A]**. Кінцевим станам будуть відповідати будь-які списки, перші елементи яких відповідають цільовій вершині, тобто кінцевому пункту маршруту: **[F : □]**. Для розглянутого вище приклада кінцевому стану будуть відповідати списки:

[F : [E,B,A]],

[F : [H,G,A]] і т.д.

Оператори перетворення для цього приклада можуть бути подані у вигляді: набору фактів, що визначають дороги між містами; набору правил, що визначають можливість переміщення з одного міста в інший.
Для розглянутого приклада набір фактів може бути таким:

дорога (a, b)
дорога (a, g)
дорога (a, e)
дорога (b, c)
.....
дорога (g, h)

Для випадку найпростішої стратегії пошуку для одноорієнтованого графа доріг набір правил може бути таким:

маршрут (Кін_місто,Кін_місто,[Кін_місто]).
маршрут (Місто,Кін_місто,[Місто|Шлях_до_кінця]):-
дорога(місто,місто_к),
маршрут (Місто_до,Кін_місто,Шлях_до_кінця)

Процедури пошуку рішень у просторі станів зручно розглядати, використовуючи граф станів, на якому будується дерево рішень.

Простір станів - це граф, вершини якого відповідають ситуаціям, що зустрічаються в задачі (проблемні ситуації), а рішення задачі зводиться до пошуку шляху в цьому графі. Процес рішення задачі містить у собі пошук у графі. При цьому виникає проблема, як обробляти альтернативні шляхи пошуку.

Розглянемо приклад найпростішої задачі для інтелектуального робота по перевпорядковуванню кубиків, поставлених один на одного, як показано на рис. 6.30.

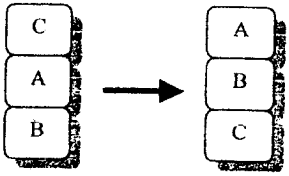


Рис. 6.30- Задача

Умови задачі:

1. На кожному кроку можна переставити тільки один кубик;
2. Кубик можна взяти тільки тоді, коли верхня поверхня вільна;
3. Кубик можна поставити або на стіл, або на інший кубик.

Для того, щоб побудувати необхідний план, інтелектуальний робот повинен відшукати послідовність ходів, що реалізують задану трансформацію. Цю задачу можна подати як задачу вибору серед множини можливих альтернатив.

У вихідній ситуації альтернатива одна - поставити кубик С на стіл.

Після того як кубик С поставлений на стіл, ми маємо три альтернативи:

1) поставити А на стіл,

або

2) поставити А на С,

або

3) поставити С на А.

Як показує розглянутий приклад з задачами такого роду пов'язано два типи понять:

- проблемні ситуації.
- дозволені ходи або дії, що перетворять одні проблемні ситуації в інші.

Проблемні ситуації разом із можливими ходами утворять спрямований граф, що називається простором станів. Простір станів для

цієї найпростішої задачі може бути зображений у вигляді повного графа подання задачі маніпулювання кубиками (рис. 6.31). Вершини графа відповідають проблемним ситуаціям, а дуги - дозволеним переходам з одних станів в інші. Задача знаходження плану переміщення еквівалентна задачі побудови шляху між початковою ситуацією («стартової» вершиною) і деякою зазначеною заздалегідь кінцевою ситуацією, що називається також цільовою вершиною.

Для розглянутої задачі рішенням є виділений на графі шлях.

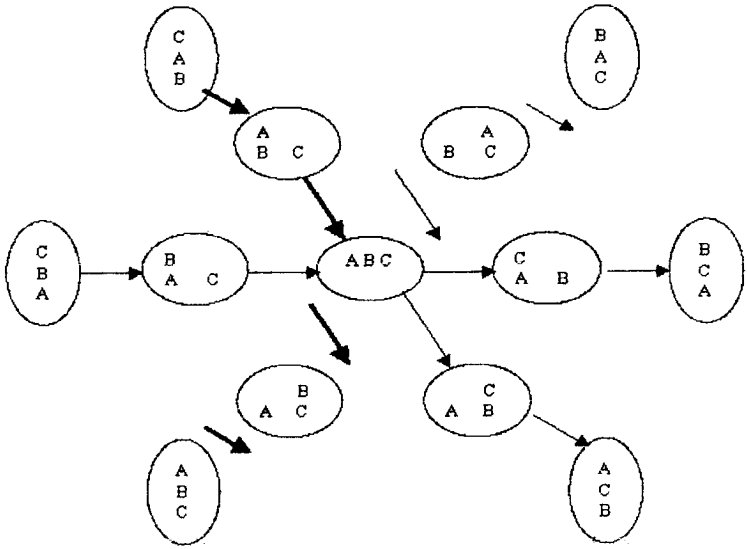


Рис. 6.31- Граф (простір станів)

Природно, що для більш складних задач графічне подання простору станів не представляється можливим через його складність і велику розмірність.

Кожному дозволеному ходу або дії можна приписати його вартість. В

розглянутій задачі, це буде вказувати на те, що один кубик переміщати складніше, ніж інші. У тих випадках, коли кожний хід має вартість, ми зацікавлені в знаходженні рішення мінімальної вартості.

Вартість рішення - це сума вартостей дуг, з яких складається шлях із стартової вершини в цільову. І вона може бути іншою, ніж у випадку, коли вартості не задані. Навіть якщо вартості не задані, може виникнути оптимізаційна задача: знайти найкоротше рішення.

Але перед тим, як говорити про можливі стратегії пошуку в просторі станів, давайте з'ясуємо, як можна представити простір станів у БЗ, використовуючи для приклада мову Пролог.

Будемо представляти простір станів за допомогою відношення предиката

після (X,Y)

яке правдиве тоді, коли в просторі станів існує дозволений хід із вершини X в вершину Y. Будемо говорити, що Y - це спадкоємець вершини X. Якщо з ходом зв'язані їхні вартості, ми додамо третій аргумент, вартість ходу:

після (X,Y,S).

Ці відношення можна задавати в програмі явно за допомогою набору відповідних фактів. Однак такий підхід непрактичний і нереальний для тих типових випадків, коли простір станів досить складний.

Тому відношення проходження «після ( )» звичайно визначається за допомогою правил обчислення вершин-приймачів деякої заданої вершини.

Іншим питанням є спосіб подання станів, тобто самих вершин. Це подання повинно бути компактне, але в той же час забезпечувати ефективне виконання необхідних операцій: операцій обчислення вершин – приймачів, а можливо і вартостей відповідних ходів. Для розглянутого приклада проблемну ситуацію (стан) можна представити як список стовпців. При цьому кожний стовпець представляється списком кубиків, з яких він складений. Кубики упорядковані в списку так, що самий верхній

знаходиться на чолі списку. «Порожні» стовпчики зображуються як порожні списки. Тоді вихідну ситуацію (стан) можна записати як терм вигляду:

[[c,a,b],[ ],[ ]]

Цільова ситуація - це будь-яка конфігурація кубиків, що містить стовпчик, складений із усіх кубиків у потрібному порядку. Таких станів три:

[[a,b,c],[ ],[ ]],
[[ ],[a,b,c],[ ]],
[[ ],[ ],[a,b,c]].

Відношення проходження може бути описане на Пролозі, виходячи з такого правила:

«Стан 2» є приймач «стану 1», якщо в «стані 1» є два стовпчики «стовп 1» і «стовпчик 2», такі, що верхній кубик із «стовпа 1» можна поставити зверху на «стовп 2» і одержати при цьому «стан 2».

При цьому цільова умова може бути також подана у вигляді правила:

Мета (стан):-належить([a,b,c],Стан)

Тобто нашою метою є пошук такої проблемної ситуації, що представляє собою список стовпчиків, в якій один із стовпчиків містить список кубиків у призначеному нами порядку.

Контрольні питання

1. Приведіть порівняльну характеристику стратегій пошуку в глибину і ширину.
2. Приведіть приклад стратегій евристичного пошуку.
3. Які етапи включає процес розв'язку задачі в системах баз знань?
4. Охарактеризуйте представлення простору станів у вигляді бази знань.

6.8 Нечіткі множини в системах баз знань

При роботі з економічними і статистичними даними як у системах прийняття рішень, так і в базах даних і знань, виникають ситуації коли ми не можемо з певною точністю кількісно описати будь-які об'єкти або явища. Наприклад, на питання «Який попит у Києві на меблі для кухні вітчизняних виробників?» можна одержати лише якісну відповідь - «високий», «низький», «нижче, чим на італійські кухні» т.п. Навіть якщо ми добудемо цифру 254 кухні на місяць, то, мабуть, довіра в нас буде досить низьке, тому що це значення може не враховувати ряд дрібних виробників, продавців, сезонність і т.д. [20]

Тому для успішного ухвалення рішення ми повинні навчитися працювати з якісними даними і нечітко визначеними поняттями. В зв'язку з цим виникають питання:

- як скласти два якісних описи або знайти їх середнє (наприклад, попит на вітчизняні кухні в Донецьку і Києві);
- як у базі даних знайти всі міста, де попит на кухні між «низьким» і «дуже низьким», тобто як виконати фільтрацію при таких даних.

Для відповіді на подібні питання використовується спеціальний математичний апарат, який називається *теорією нечітких множин*, що, як показали два останніх десятиліття, виявився досить життєвим, широко й успішно застосовуваним в самих різних областях. Розглянемо основні визначення і положення теорії нечітких множин.

Нечіткою множиною $\tilde{A}$ на множині U називається сукупність пар $\tilde{A} = < \mu_A(u), u >$, де $\mu_A(u)$ - функція належності нечіткої множини $\tilde{A}$, а u - носій нечіткої множини $\tilde{A}$.

Розглянемо приклад. Нехай потрібно визначити нечітке поняття «Студент вчиться добре». Позначимо це поняття $\tilde{A}$. При визначенні цього поняття цілком очевидно, що якщо в студента середній бал - 4, то його

можна назвати таким, що добре вчиться. Однак якщо середній бал 4.4 , то студент вчиться теж «добре», але вже з деяким сумнівом: а може «відмінно» $\tilde{A}$. При середньому балі 3.7, його теж можна віднести до

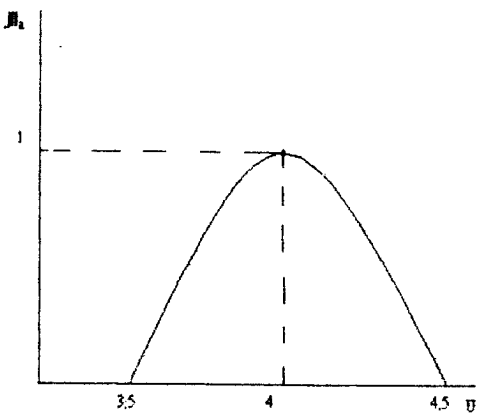


Рис. 2-32- Функція належності

категорії учнівських на «добре», але з ще більшим сумнівом: а може «задовільно» ? Наше уявлення (знання) про поняття «Студент учиться добре» можна подати графічно (рис.2-32). Графік показує наш сумнів або впевненість у тому, що студента з деяким середнім балом можна віднести до таких, що «добре вчиться». Тут середній бал <u> є *носієм нечіткої множини.*

А ступенем впевненості в тому, як значення <u> відповідає поняттю «добре» і є *функція належності*. Таким чином, нечітка множина $\tilde{A}$ є не що інше, як спроба кількісно описати якісне вираження поняття «добре».

Чому функція належності $\mu_A(u) \in [0,1]$? Це зв'язано з тим, що нам зручно виражати своє відношення у відсотках. Наприклад, на 90% довіряю. А якщо є абсолютна впевненість, то це 100% довіра або 1. Тому й область зміни значень функції належності від 0 до 100% або [0, 1].

Розглянемо тепер поняття *лінгвістична змінна*. Це змінна, що може приймати нечіткі значення. Пояснимо це поняття на прикладі. Визначимо лінгвістичну змінну X, що характеризує успішність студентів. Ця змінна може приймати одне з чотирьох можливих нечітких значень

$$X = \{\tilde{A}_1, \tilde{A}_2, \tilde{A}_3, \tilde{A}_4\},$$

де $\tilde{A}_1$ = «незадовільно»; $\tilde{A}_2$ = «задовільно»; $\tilde{A}_3$ = «добре»; $\tilde{A}_4$ = «відмінно». Для того, щоб працювати з лінгвістичною змінною X ми повинні кількісно визначити всі нечіткі множини $\tilde{A}_i$ (рис. 6.33).

З цього рисунка видно, що графіки функцій належності перетинаються. Це говорить про те, що нам важко сказати точно, до якого нечіткого значення віднести визначений середній бал. У таких випадках звичайно використовують максимальне значення $\mu_{A_{iu}}$, наприклад, середній бал 2,75 більш підходить до значення «задовільно», чим «незадовільно».

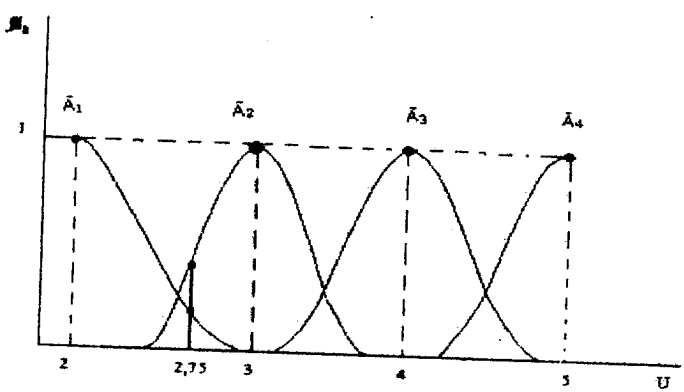


Рис. 6.33- Нечіткі множини $\tilde{A}_i$

Слід зазначити, що в більшості випадків функції належності будуються суб'єктивно за результатами опиту експертів, тому вони є в деякій мірі «наближеними», тобто не абсолютно адекватно відображують явище або об'єкт. Власне кажучи, із суб'єктивності випливає, що абсолютної адекватності в принципі і не існує. Тому, мабуть, потрібно

вибирати таку функцію, з якою можна було б як можна простіше вести розрахунки. Такими функціями є *трапецієподібні* (рис. 6.34) функції.

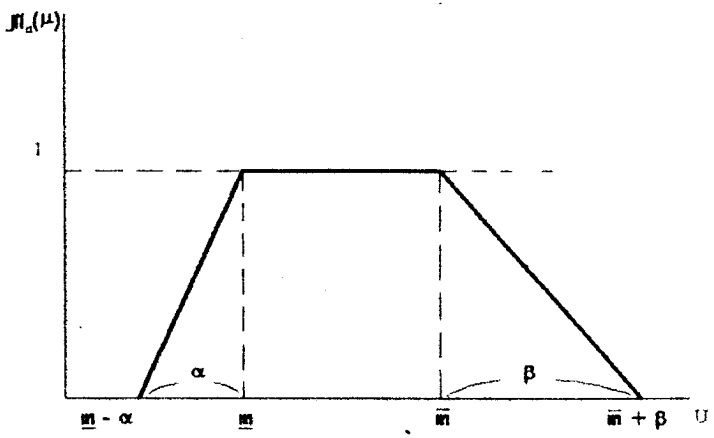


Рис. 6.34 – Трапецієподібна функція належності

Тоді $\mu_A(u)$ характеризується четвіркою ($\underline{m}$, $\overline{m}$, α, β). Як окремий випадок при $\underline{m} = \overline{m}$ маємо трикутну форму функції.

Перед тим як визначити найпростіші арифметичні операції над нечіткими змінними розглянемо приклад. Нехай у рамках упорядкування проекту бюджету розглядаються різні джерела фінансування. Причому деякі з них характеризуються неточністю оцінки грошових сум на день оцінювання, а інші малою надійністю. Крім того, із бюджету необхідно віддати борги, кількість яких також неточна, тому що залежить від того, чи зажадає кредитор усі або тільки частину в наступному фінансовому періоді.

- *Джерело А*: фінансування забезпечується, його сума може змінюватися від 40 до 100 млн. в залежності від кон'юнктури, але з найбільшою імовірністю очікується надходження в сумі від 50 до 70 млн.

- *Джерело B*: джерело надійне і можна думати, що фінансування буде надано і складе суму 100 - 110 млн.
- *Джерело C*: джерело ненадійне, а якщо і дасть, то не більш ніж 20 млн.
- *Борг D*: плата за кредити 50 - 100 млн., але найбільше ймовірна виплата 80 млн.

Таким чином, маємо три джерела надходжень і одне джерело витрат. Побудуємо на основі їх описів трапецієподібні функції належності для кожної з чотирьох нечітких змінних (рис.6.35). Після завдання всіх нечітких змінних, постає задача визначення суми всього бюджету, що також буде нечіткою величиною. А для цього треба вміти виконувати найпростіші арифметичні операції над нечіткими змінними.

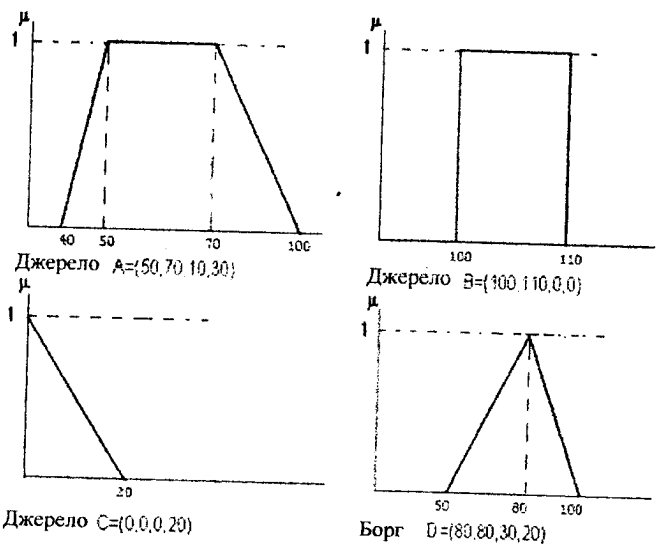


Рис. 6.35 – Проект бюджету

Визначення цих операцій розглянемо для випадку двох нечітких змінних $\tilde{A}_1$ і $\tilde{A}_2$, що задані своїми трапецієподібними функціями належності вигляду

$$\tilde{A}_1 = (\underline{m1}, \overline{m1}, \alpha_1, \beta_1),$$

$$\tilde{A}_2 = (\underline{m2}, \overline{m2}, \alpha_2, \beta_2).$$

Результатом операції буде також нечітка змінна A = ($\underline{m}$, $\overline{m}$, α, β), що також має трапецієподібну функцію належності, параметри якої визначаються в залежності від виду арифметичної операції (табл.6.6)

Таблиця 6.6

Тип операції	Залежності параметрів функцій належності
$A = \tilde{A}_1 (+) \tilde{A}_2$	$\underline{m} = \underline{m1} + \underline{m2}$, $\overline{m} = \overline{m1} + \overline{m2}$, $\alpha = \alpha_1 + \alpha_2$, $\beta = \beta_1 + \beta_2$.
$A = \tilde{A}_1 (-) \tilde{A}_2$	$\underline{m} = \underline{m1} - \overline{m2}$, $\overline{m} = \overline{m1} - \underline{m2}$, $\alpha = \alpha_1 + \beta_2$, $\beta = \beta_1 + \alpha_2$.
$A = \tilde{A}_1 (x) \tilde{A}_2$	$\underline{m} = \underline{m1} * \underline{m2}$, $\overline{m} = \overline{m1} * \overline{m2}$, $\alpha = \underline{m1} * \underline{m2} - (\underline{m1} - \alpha_1)(\underline{m2} - \alpha_2)$, $\beta = (\overline{m1} + \beta_1)*(\overline{m2} + \beta_2) - \overline{m1} * \overline{m2}$
$A = \tilde{A}_1 (/) \tilde{A}_2$	$\underline{m} = \underline{m1} / m2$, $\overline{m} = \overline{m1} / m2$, $\alpha = (\underline{m1} * \beta_2 + \overline{m2} * \alpha_1) / (\overline{m}2^2 + \overline{m2} * \beta_2)$ $\beta = (\underline{m2} * \beta_1 + \overline{m1} * \alpha_2) / (\underline{m}2^2 * \alpha_2)$

На основі приведених вище описів арифметичних операцій можна для розглянутого приклада визначити оцінку бюджету без обліку боргів (Ф) як суму трьох джерел фінансування. Причому результат буде також нечіткою змінною:

Ф = A (+) B (+) C = (50+100+0, 70+110+0, 10+0+0, 30+0+20) = (150,180,10,50)

з трапецієподібною функцією належності, приведеною на рис.9.5. Для одержання повної оцінки передбачуваного бюджету необхідно з отриманого результату відняти передбачувані плати по кредитах. При цьому бюджет з обліком боргів (Фд) також буде нечіткою змінною (рис.6.36):

Фд = Ф (-) D = (150 - 80, 180 - 80, 10 + 20, 50 + 30) = (70, 100, 30, 80),

функція належності якої також має трапецієподібний вид і приведена на рис.6.37.

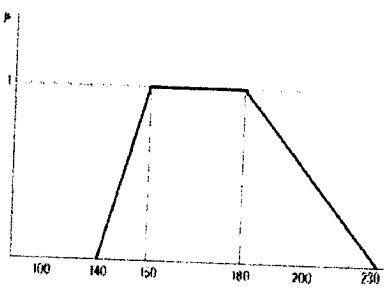


Рис. 2-36- Бюджет без обліку боргів

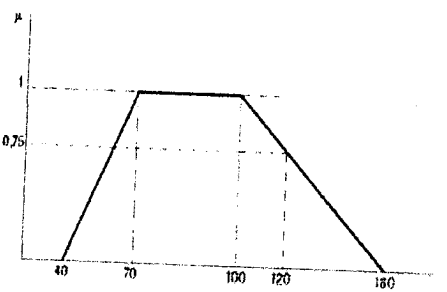


Рис. 2-37- Бюджет з обліком боргів

Таким чином, у бюджеті може бути сума від 40 до 180 млн., але з найбільшою імовірністю можна говорити про суми від 70 до 100млн.

Операції нечіткої фільтрації і вибору .Визначення цих операцій також почнемо з розгляду приклада. Нехай створюється база даних по бюджетах різних фірм або підрозділів. При цьому будь-який з бюджетів може бути оцінено як «малий» (A_1), «середній» (A_2) або «великий» (A_3). Тобто ми маємо справу з лінгвістичною змінною «об'єм бюджету» ($\tilde{A}$), що може приймати одне з трьох нечітких значень (A_1, A_2, A_3).

Припустимо, що функція належності для кожній із змінних були оцінені в такий спосіб

A_1 = «малі» = (0, 50, 0, 50);

A_2 = «середні» = (80, 150, 20, 20);

A_3 = «великі» = (200, 250, 20, 20).

Тепер постає задача - до якого з класів бюджетів можна віднести значення бюджету, отриманого в попередньому прикладі. Тобто необхідно визначити чи є наш бюджет «малим», «середнім» або «великим»

Ця задача відноситься до класу задач нечіткої фільтрації в базах даних або θ -відбору. Для її рішення вводиться два показники:

- $\Pi (Ai/ Фд) = \sup \min (\mu_{ф}(u), \mu_{Ai}(u))$ - це *можливість*, що нечітка множина Ф належить значенню Ai атрибута $\tilde{A}$.
- $N (Ai/Фд) = \inf \max (1 - \mu_{ф}(u), \mu_{Ai}(u))$ - це *необхідність*, що нечітка множина Фд належить значенню Ai атрибута $\tilde{A}$.

Для всіх значень атрибута $\tilde{A}$ ми знаходимо пари (П, N), а потім по максимальному значенню цієї пари знаходимо залежність нечіткої множини до того або іншого значення атрибута. На рис.6.38 показана суть цих показників. Розглянемо докладніше ці два показники.

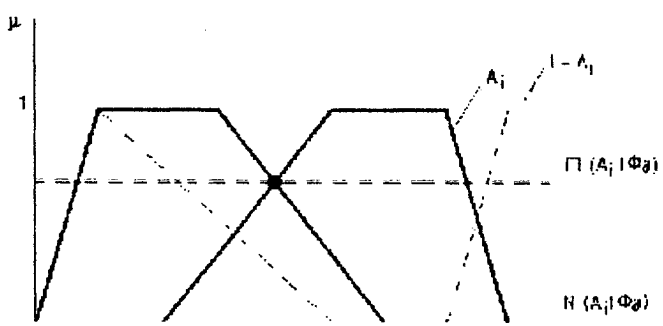


Рис. 6.38- Графічна інтерпретація можливості і необхідності

Показник П(Ai/Фд) характеризує *можливість збігу* (м'яке рішення або м'який вивід - звідси англійське soft computing). Фактично, будь-яку функцію належності можна розглядати як розподіл можливостей. Наприклад, із рис.9.6 видно, що можливість одержання суми 120 млн. у бюджеті Фд дорівнює 0.75. На відміну від даної можливості, можливість П(Ai/Фд) є умовної і обчислюється на основі принципу узагальнення Заде (приведені вище формули).

Нехай $\mu_{A1}(u)$, $\mu_{A2}(u)$, і $\mu_{A3}(u)$ мають вигляд, приведений на рис.6.39. На цьому ж рисунку зображена функція належності для Фд. Розглянемо геометричну інтепретацію визначення $П(A_1|Фд)$:

T

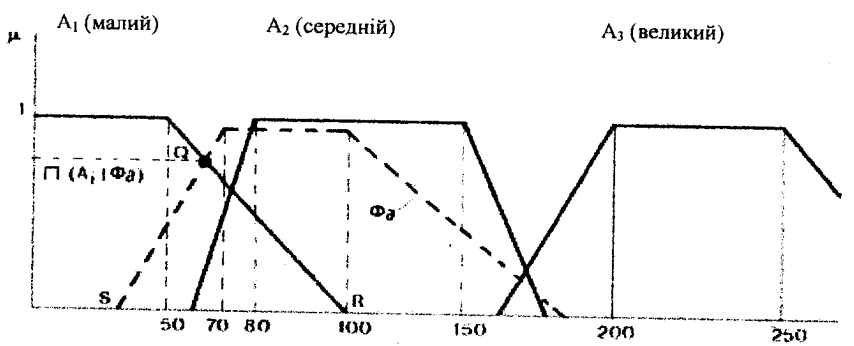


Рис. 6.39- Геометричне визначення $П(\tilde{A_1}/Фд)$

- min ($\mu_ф(u)$, $\mu_{A1}(u)$) - являє собою трикутник SQR, тобто це нечітка множина з функцією належності обмеженої точками SQR. Ми для кожної точки на осі взяли найменше значення з двох $\mu_ф(u)$ і $\mu_{A1}(u)$.
- sup min ($\mu_ф(u)$, $\mu_{A1}(u)$) - це точка Q, тобто найбільше значення MSQR в усіх точках U=[0, ∞). Це приблизно 0.75, тобто $П(A_1|Фд) = 0.75$.

Значення $\Pi(A_1|\ \Phi\partial)$ може бути обчислене і аналітично на основі приведених вище формул принципу узагальнення Заде. Нехай

$$\Phi\partial = (\underline{m1},\ \overline{m1},\ \alpha_1,\ \beta_1),$$

$$A_1 = (\underline{m2},\ \overline{m2},\ \alpha_2,\ \beta_2).$$

Тоді

$$\Pi(A_1|\ \Phi\partial) = \min\{\max(0, \min(1, 1 + (\overline{m1} - \underline{m2}) / (\beta_1 + \alpha_2))),$$
$$\max(0, \min(1, 1 + (\overline{m2} - \underline{m1}) / (\beta_2 + \alpha_1)))\}.$$

Для розглянутого приклада маємо:

$$\Phi\partial = (70, 100, 30, 80);$$

$$A_1 = (0, 50, 0, 80).$$

Використовуючи вищенаведене вираження отримаємо

$$\Pi(A_1|\ \Phi\partial) = \min\{\max(0, \min(1, 1 + (100\text{-}0) / (80 + 0))),$$
$$\max(0, \min(1, 1 + (50\text{ -}70) / (30 + 50)))\} = \min\{\max(0, \min(1, 1 + 100 / 80)),$$
$$\max(0,\min(1,1\text{-}20/80))\} = \min\{\max(0, 1),\max(0,1\text{-}0{,}25)\} = \min\{1, 0{,}75)\} = 0{,}75,$$

що відповідає графічному розрахунку. Аналогічно можна знайти $\Pi(A_1/\Phi\partial)$ для A_2 і A_3. Навіть з рисунка видно, що $\Pi(A_2/\Phi\partial) = 1$; $\Pi(A_3/\Phi\partial) \approx 0.2$.

Показник $N(Ai/\Phi\partial)$ характеризує *необхідність збігу*, тобто значення бюджету Ai обов'язково належать $\Phi\partial$ (жорстке ухвалення рішення, навіть зверхжорстке). Цей розмір використовується в двох випадках:

- коли $\Pi(Ai\ |\ \Phi\partial) = \Pi(Aj/\ \Phi\partial)$, $i \neq j$, тобто можливості однакові і тому нерозрізнених;
- при більш жорсткому доборі.

Розглянемо суть цього показника на нашому прикладі при обчисленні $N(A_2/\Phi\partial)$. Як і раніше почнемо з геометричної інтерпретації визначення $\Pi(\tilde{A_1}\Phi\partial)$:

- Спочатку знайдемо $1 - \mu_{\phi}(u)$
- Потім $\max(1 - \mu_{\phi}(u), \mu_{A2}(u))$. Це ломана ABCDEFG (на рис. 6.40 вона позначена точками).

- Обчислимо inf max (1 - $\mu_{\phi}(u)$, $\mu_{Ai}(u)$). Це буде нижня межа ломаної ABCDEFG. Очевидно, що це точка С.

- Тоді $N(A_2 | \Phi\partial)$ буде відповідати точці. Це приблизно 0.20.

Виконуючи аналогічні дії, із рис. 6.40 можна одержати, що $N(A_1 | \Phi\partial) = 0$ і $N(A_3 | \Phi\partial) = 0$

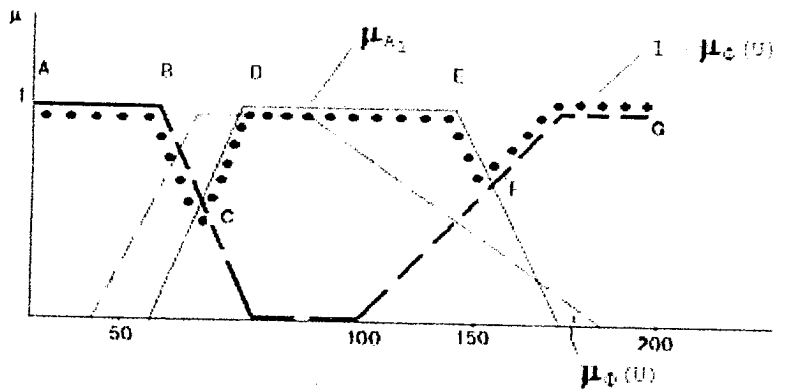


Рис. 6.40- Геометричне визначення $N(A_1 | \Phi\partial)$

Використовуючи формули, що випливають із принципу узагальнення Заде можна провести аналітичний розрахунок показника необхідності. Якщо прийняти, що $A_2 = (\underline{m2}, \overline{m2}, \alpha_2, \beta_2)$, тоді

$$N(A_2 | \Phi\partial) = \min \{\max (0, \min (1, (\underline{m1} - \underline{m2} + \alpha_2) / (\alpha_1 + \alpha_2))),$$
$$\max (0, \min (1, (\overline{m2} - \overline{m1} + \beta_2) / (\beta_1 + \beta_2)))\}.$$

Для даного приклада:

$$\Phi\partial = (70, 100, 30, 80).$$
$$A_2 = (80, 150, 20, 20).$$

Тоді

$$N(A_2 | \Phi\partial) = \min \{\max (0, \min (1, (70-80+20)/(30+20)),$$
$$\max (0, \min(1,(150-100+20)/(100)) = \min\{\max (0, \min (1, 0,2),$$
$$\max (0, \min (1,0.7) = 0.2,$$

що відповідає графічним перетворенням, отриманим з рис.9.9. Таким чином, після проведення всіх обчислень ми маємо три значення показників можливості і необхідності (П, N) для кожного зі значень A_1, A_2, A_3 лінгвістичної змінної «об'єм бюджету» ($\tilde{A}$):

для A_1="малий" → (0,75; 0,0)

для A_2="середній" → (1,00; 0,2)

для A_3="великий" → (0,20; 0,0)

з яких очевидно, що розрахований нами бюджет необхідно віднести до «середнього» із можливістю 1 і необхідністю 0.2

Контрольні питання

1. В яких випадках в базах знань використовують нечіткі множини?
2. Які арифметичні операції виконують над нечіткими операндами ?
3. Охарактеризуйте операції нечіткої фільтрації і відбору ?

СПИСОК ЛІТЕРАТУРИ

1. Атре Ш. Структурный подход к организации баз данных. – М.: Финансы и статистика, 1983. – 320 с.
2. Бойко В.В., Савинков В.М. Проектирование баз данных информационных систем. – М.: Финансы и статистика, 1989. – 351 с.
3. Дейт К. Руководство по реляционной СУБД DB2. – М.: Финансы и статистика, 1988. – 320 с.
4. Джексон Г. Проектирование реляционных баз данных для использования с микроЭВМ. -М.: Мир, 1991. – 252 с.
5. Костин А.Е., Шальгин В.Ф. Организация и обработка структур данных в вычислительных системах Учебное пособие для вузов. -М.: Высш. Шк.,1987.-248 с.
6. Кириллов В.В. Структуризованный язык запросов (SQL). – СПб.: ИТМО, 1994. – 80 с.
7. Мартин Дж. Планирование развития автоматизированных систем. – М.: Финансы и статистика, 1984. – 196 с.
8. Месюра В.І., Романюк О.Н., Денисюк А.В. Методичні вказівки до виконання лабораторних робіт з дисципліни "Організація та управління базами даних для студентів інженерної спеціальності 7.080403" Програмне забезпечення обчислювальної техніки та автоматизованих систем". - Вінниця, :ВДТУ, 1995.
9. Мейер М. Теория реляционных баз данных. – М.: Мир, 1987. – 608 с.
10. Романюк О.Н., Місюра В.І., Денисюк А.В. Методичні вказівки до курсового проектування по дисципліні "Організація та управління базами даних" для студентів інженерії спеціальності 7.080403 "Програмне забезпечення обчислювальної техніки та автоматизованих систем ".- Вінниця, :ВДТУ, 1994.
11. Системы управления базами данных и знаний: Справочное издание / А.Н. Наумов, А.М. Вендров, В.К. Иванов и др.; Под ред. А.Н.Наумова. – М.: Финансы и статистика, 1991. – 352 с.

12.Тиори Т., Фрай Дж. Проектирование структур баз данных. В 2 кн., – М.: Мир, 1985. Кн. 1. – 287 с.: Кн. 2. – 320 с.

13.Ульман Дж. Базы данных на Паскале. – М.: Машиностроение, 1990. – 386 с.

14.Хаббард Дж. Автоматизированное проектирование баз данных. – М.: Мир, 1984. – 294 с.

15.Четвериков В.Н., Ревунков Г.И., Самохвалов Э.Н. Базы и бланки данных. – М.: Высшая школа, 1993.

16.Цикритизис Д., Лоховски Ф. Модели данных. – М.: Финансы и статистика, 1985. – 344 с.

17.Н. Д. Нильсон. Искусственный интеллект. Методы поиска решений.- М.: Мир, 1973.

18.К. Таунсенд, Д. Фохт. Проектирование и программная реализация экспертных систем на персональных ЭВМ.- М.: Финансы и статистика, 1990.

19.Д. Элти, М. Кумбс. Экспертные системы: концепции и примеры.- М.: Финансы и статистика, 1987.

20.Хабаров С. П. Лекции по курсу "Информационные технологии ", http://firm.trade.spb.ru/serp/main_es.htm.

Навчальне видання

Романюк О.Н., Савчук Т.О.

ОРГАНІЗАЦІЯ БАЗ ДАНИХ І ЗНАНЬ

Навчальний посібник

Оригінал-макет підготовлено авторами

Редактор С.А.Малішевська

Видавництво ВНТУ «УНІВЕРСУМ- Вінниця»
Свідоцтво Держкомінформу України
серія ДК № 746 від 25.12.2001
21021, м. Вінниця, Хмельницьке шосе, 95, ВНТУ

Підписано до друку 20.11.2003.
Формат 29,7х42 ¼
Друк різографічний
Тираж 200 прим.
Зам. № 2003-177

Гарнітура Times New Roman
Папір офсетний
Ум.друк.арк. 12,3

Віддруковано в комп'ютерному інформаційно-видавничому центрі
Вінницького національного технічного університету
Свідоцтво Держкомінформу України
серія ДК № 746 від 25.12.2001
21021, м.Вінниця, Хмельницьке шосе, 95,
ВНТУ, головний корпус, к. 114
Тел.: (0432) 44-01-59