Державний вищий навчальний заклад
“Українська академія банківської справи
Національного банку України”
Кафедра економічної кібернетики

С.В. Вахнюк

ТЕХНОЛОГІЯ СТВОРЕННЯ ПРОГРАМНИХ ТА ІНТЕЛЕКТУАЛЬНИХ СИСТЕМ

Навчальний посібник

Для студентів денної форми навчання
освітньо-професійної програми підготовки
за напрямом 6.030501 “Економічна кібернетика”

Суми
ДВНЗ “УАБС НБУ”
2011

УДК 004.8(075.8)
ББК 32.97я73
В22

Рекомендовано Міністерством освіти і науки України як навчальний посібник для студентів вищих навчальних закладів, які навчаються за освітньо-професійною програмою бакалавра напряму підготовки "Економічна кібернетика" (Лист № 1/11-4934 від 10.06.2010)

Автор
С.В. Вахнюк – кандидат економічних наук, доцент кафедри економічної кібернетики ДВНЗ "Українська академія банківської справи Національного банку України»
Рецензенти:
Т.С. Клебанова – доктор економічних наук, професор, завідувач кафедри економічної кібернетики Харківського національного економічного університету;
В.М. Чаплига – доктор економічних наук, професор, завідувач кафедри економічної кібернетики Львівського інституту банківської справи Університету банківської справи Національного банку України;
С.О. Дмитров – доктор технічних наук, професор, начальник управління контролю за ефективністю фінансового моніторингу Департаменту з питань запобігання використанню банківської системи для легалізації кримінальних доходів та фінансування тероризму Національного банку України

Вахнюк, С.В.

В22 Технологія створення програмних та інтелектуальних систем [Текст] : навчальний посібник / С. В. Вахнюк. – Суми : ДВНЗ "УАБС НБУ", 2011. – 254 с.

ISBN 978-966-8958-69-4

У посібнику розглядаються технології та засоби створення програмних систем і систем штучного інтелекту для розв'язання економічних задач. Матеріал посібника складається з двох частин: технологія програмування інформаційних систем з базами даних і технологія створення експертних систем. Кожна з частин посібника поділена на тематичні розділи. У заключній частині кожного з розділів представлені питання, завдання і тести для самоперевірки. Теоретичний матеріал ілюструється численними прикладами. У кінці книги наводяться список літератури, предметний покажчик та додатки. За змістом посібник відповідає нормативній програмі навчальної дисципліни "Технологія створення програмних та інтелектуальних систем".

Для студентів, магістрів та аспірантів економічних спеціальностей вищих навчальних закладів.

УДК 004.8(075.8)
ББК 32.97я73

ISBN 978-966-8958-69-4

ЗМІСТ

ВСТУП

Сучасна епоха характеризується постійним підвищенням рівня значущості продуктів високотехнологічного виробництва в задоволенні потреб суспільства. Інформаційна складова у вартості таких продуктів відіграє домінуючу роль, тому суспільство, орієнтоване на їх споживання, визначається як інформаційне. Розробку інформаційних систем та їх впровадження в сучасні структури реальної економіки можуть здійснювати виключно висококваліфіковані спеціалісти, які володіють знаннями та навичками використання відповідних технологій.

Зважаючи на актуальність вивчення теорії та практики розробки інформаційних систем, що сприяють вирішенню сучасних економічних задач, до освітньо-професійної програми підготовки бакалаврів за спеціальностями напряму 0501 "Економіка і підприємництво", як обов'язкова, включена дисципліна з вивчення технологій створення програмних та інтелектуальних систем. Роль основних забезпечувальних навчальних дисциплін виконують: "Інформатика та комп'ютерна техніка", "Алгоритмізація та проектування". Дисципліни, що забезпечується нею: "Інформаційні системи в економіці", "Системи підтримки прийняття рішень", "Інформаційний бізнес".

Даний навчальний посібник підготовлений відповідно до галузевих стандартів, що містяться в освітньо-кваліфікаційній характеристиці (ОКХ) та освітньо-професійній програмі (ОПП) підготовки бакалавра даного напряму. Він присвячений формуванню теоретичних знань та практичних навичок створення програмних рішень засобами систем управління базами даних та штучного інтелекту для вирішення економічних задач.

Основні завдання навчального посібника полягають у вивченні методів та засобів розробки програмних систем архітектури "клієнт-сервер" з використанням реляційних структур даних, набуття навичок застосування моделей подання знань в процесі проектування, програмування та налагодження фрагментів програмного забезпечення експертних систем. У посібнику систематизовані результати аналізу існуючих технологій програмування алгоритмічних та інтелектуальних систем, описані способи їх застосування в тому обсязі, який, на думку автора, дозволяє створити у спеціалістів у сфері управління економікою фундамент знань, достатній для вирішення відповідних задач на практиці.

Матеріал посібника складається з двох частин: технологія програмування інформаційних систем з базами даних та технологія створення інтелектуальних систем. Теоретичні матеріали, що формують ці

частини, є логічно взаємопов'язаними й утворюють єдиний навчальний комплекс для самостійного вивчення відповідної дисципліни. Кожна з частин посібника розділена на окремі розділи, що висвітлюють конкретні питання. У заключній частині кожного з розділів представлені тестові *завдання для самоперевірки.*

У першому розділі розглядаються теоретичні основи сучасного програмного забезпечення системної обробки інформації. Другий розділ містить інформацію стосовно сучасних засобів створення програмних додатків, що використовуються для взаємодії з базами даних. Розділи з третього по шостий присвячені проектуванню структури реляційних баз даних та використанню мови структурованих запитів SQL для створення об'єктів реляційних баз даних, маніпулювання ними, процедурного програмування та адміністрування сервера баз даних.

Сьомим розділом починається друга частина посібника, в якому розглядається проблема штучного інтелекту та його практичне спрямування. У восьмому та дев'ятому розділах увага концентрується на методологічних засадах експертних систем як на найбільш практично застосовуваному різновиді штучного інтелекту. Учбовий матеріал, що міститься з десятого по тринадцятий розділи, відображає існуючі моделі та методи представлення знань в експертних системах, а також способи їх практичного використання. У чотирнадцятому розділі розглядається загальна характеристика програмних засобів створення експертних систем та даються рекомендації щодо їх ефективного застосування. П'ятнадцятий розділ присвячений базовим концепціям нейронних мереж.

Даний посібник призначений, у першу чергу, для студентів спеціальності "Економічна кібернетика", інших економічних спеціальностей, а також аспірантів, науковців, фахівців-практиків. Під час написання навчального посібника були використані матеріали періодичних та Інтернет-видань, література вітчизняних та зарубіжних авторів.

Автор висловлює подяку науково-педагогічним працівникам та співробітникам ДВНЗ "Українська академія банківської справи Національного банку України" за участь у підготовці даного навчального посібника.

Автор буде вдячний за зауваження та побажання стосовно представленого матеріалу й методики його подання, що надасть безперечну допомогу в подальшій роботі.

Зауваження й побажання щодо даного навчального посібника прохання надсилати за адресою: 40000, м. Суми, вул. Петропавлівська, 57, кафедра економічної кібернетики. E-mail: vakhnyuk@academy.sumy.ua.

Частина І
СТВОРЕННЯ ПРОГРАМНИХ СИСТЕМ З БАЗАМИ ДАНИХ

РОЗДІЛ 1. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМНОЇ ОБРОБКИ ІНФОРМАЦІЇ

1.1. Процедурне програмне забезпечення

Програмне забезпечення (ПЗ) системної обробки інформації протягом своєї еволюції пройшло складний шлях удосконалення методичних підходів створення інформаційних систем (ІС) на базі обчислювальної техніки. Цей процес сформував певні принципи програмування ІС, які можна успішно використовувати для систематизації сучасного ПЗ. У найбільш загальному вигляді принципи програмування ІС можна поділити на процедурні та декларативні. До процедурних належать наступні види ПЗ:

1) імперативне ПЗ. Мови програмування, які найчастіше застосовуються: C, Pascal, JAVA;
2) функціональне ПЗ. Мови програмування, які найчастіше застосовуються: PROLOG, LISP.

Слід зазначити, що така класифікація демонструє виключно загальний зв'язок між підходами до програмування, який не слід сприймати як суворе визначення. Деякі підходи й мови програмування мають такі особливості, що дозволяють віднести їх більш ніж до однієї категорії. Наприклад, одні фахівці розглядають функціональне програмування як принцип процедурного підходу, а інші вважають його декларативним.

Процедурним програмуванням називається реалізація алгоритму з використанням мови (мов) програмування. Алгоритм являє собою метод вирішення задачі внаслідок здійснення визначеної кількості дій. Під мовою програмування розуміють транслятор команд (операторів), записаних із застосуванням конкретного синтаксису. Поєднання мови програмування і пов'язаних з нею допоміжних програм (утиліт) утворює інструментальне середовище для розробки, налаштування й доставки користувачу програмного забезпечення.

Термін “процедурне програмування” використовується для позначення програм, виконання яких здійснюється послідовно, оператор за оператором, якщо не зустрічається команда умовного переходу. Для визначення процедурної програми часто використовуються синоніми: алгоритмічне програмування й послідовне програмування. Особливістю процедурного підходу є те, що програміст зобов’язаний точно вказувати, яким чином рішення задачі буде реалізоване в програмі.

Розглянемо принципи процедурного підходу до програмування інформаційних систем.

Імперативне ПЗ. Імперативний принцип програмування – це абстрактне представлення класичної архітектури комп’ютерів, в основі якої лежать поняття регістрів і сховища даних (пам’яті). Ця архітектура одержала назву на честь американського вченого Джона фон Неймана (von Neumann).

Машина фон Неймана – обчислювальна система, що складається з п’яти основних вузлів: пам’яті, арифметико-логічного пристрою (АЛУ), пристрою управління і пристроїв вводу-виводу. У сучасних мікропроцесорах АЛУ і пристрій управління об’єднані в одному корпусі. Арифметичні й логічні операції здійснює АЛУ на підставі двійкового коду за допомогою регістрів.

Як аналог регістрів мови імперативного програмування використовують змінні, яким привласнюються певні значення, а сховище даних є об’єктом, яким управляє програма. Для цього передбачений багатий набір команд, які дозволяють задавати структуру алгоритму.

Кожна мова імперативного програмування здійснює обробку двійкового коду, що орієнтований на певну апаратну платформу. Можливість перенесення програмного забезпечення з однієї апаратної платформи на іншу здійснюється за рахунок компілятора, який визначається мовою програмування. Компілятор реалізує мовно-орієнтовану віртуальну машину, що є програмним алгоритмом, який може виконуватися на реальних апаратних засобах під управлінням різноманітних операційних систем (ОС).

В імперативному програмуванні значення може бути позначене ім’ям, яке надалі, у разі потреби, може бути привласнене іншому значенню. Колекція імен і пов’язаних з ними значень, а також місцезнаходження в програмі оператора, якому передане управління програмою, називається її станом. Стан являє собою логічну модель пам’яті, що є асоціацією між значеннями і їх місцями знаходження в пам’яті. Програму, в ході її виконання можна розглядати як здійснення переходу з її початкового стану в кінцевий через проходження послідовності проміжних станів. Перехід від одного стану в інший визначається

операціями привласнення, командами вводу і командами управління ходом виконання програми.

Імперативні мови характеризуються тим, що в них широко використовується жорстка структура управління й, у зв'язку з цим, реалізуються спадні проекти програм. Наочно ілюстрацію таких проектів демонструє методика їх проектування за допомогою графічного представлення алгоритму у вигляді блок-схеми. Реалізація блок-схем алгоритмів програм здійснюється зверху – від блоку, що позначає початок програми, вниз – до блоку, що позначає її кінець. Між ними знаходиться складна система переходів між графічними елементами (прямокутниками, ромбами, трапеціями та ін.), які відповідають операціям: по оголошенню змінних, присвоєнню ним значень, повторному виконанню дій тощо. Усі ці операції належать до операцій низького рівня, і програмування складних систем на їх основі є надзвичайно трудомістким процесом. Тому еволюція мов програмування завжди була пов'язана з прагненням усунення необхідності безпосереднього застосування операцій низького рівня шляхом використання таких засобів, як процедури, модулі, функції, пакети і т.д.

Функціональне ПЗ. За своїм характером функціональне програмування відрізняється від імперативного програмування підходом до вирішення задач, шляхом їх розбиття на складові – задачі меншої складності. Тобто, якщо мови імперативного програмування не дозволяють застосувати достатньо складні принципи такого розбиття, то у функціональних мовах подібні обмеження багато в чому усунені.

В основі функціонального програмування лежить фундаментальна ідея об'єднання простих функцій для створення більш потужних функцій. При цьому, по суті, застосовується метод висхідного проектування, на відміну від методів спадного проектування, що використовується в імперативному підході.

Функціональне програмування зосереджене навколо поняття функції. З погляду програмування, функція – це іменований набір операторів, який може володіти наступними можливостями:

- приймати як аргумент значення (набір значень);
- виконувати певні дії з даними;
- повертати значення (набір значень);
- викликати інші функції і бути викликаною іншою функцією.

Функції дозволяють виконувати одну й ту ж операцію багато разів, не переписуючи кожного разу наново відповідний їй програмний код. Функціональна мова складається з наступних п'яти частин:

- об'єкти даних, якими повинні оперувати функції;
- примітивні функції, які мають обробляти дані;

- функціональні форми, призначені для створення нових функцій на основі існуючих функцій;
- прикладні операції з функціями, які повертають значення;
- процедури іменування, призначені для надання імен новим функціям.

Функціональні мови з метою забезпечення простоти конструювання і швидкого відгуку на введення команд користувачем, як правило, реалізуються за допомогою інтерпретаторів. Прикладом може бути мова LISP (скорочення від LISt Processing – обробка списків). У мові LISP дані є символічними виразами, які є або списками, або атомарними (неподільними) позначеннями конкретних об'єктів. Первинна версія мови LISP була виключно функціональною. Подальша її еволюція призвела до того, що LISP була деякий час провідною мовою штучного інтелекту в Сполучених Штатах.

1.2. Декларативне програмне забезпечення

Сутність декларативних підходів полягає в тому, що вони не вимагають від програміста опису точних подробиць того, як повинна бути вирішена задача. У цьому вони докорінно відрізняються від процедурних підходів, що вимагають точного визначення послідовності операторів або функцій. При використанні декларативних підходів основний обсяг роботи полягає в тому, щоб вказати, що повинно бути зроблене, і дозволити системі визначити, як це зробити.

У декларативному підході шлях досягнення мети розглядається окремо від методів, що використовуються для її досягнення. Користувач задає мету, а механізм реалізації ІС робить спробу досягти даної мети оптимальним способом. Для реалізації такої декларативної моделі було створено безліч підходів і пов'язаних з ними мов програмування. До них належать:

1. Об'єктно-орієнтоване ПЗ. Найчастіше застосовуються мови програмування: C++, Java і C#.
2. Логічне ПЗ. Найчастіше застосовується мова програмування PROLOG.
3. ПЗ експертних систем. Найчастіше застосовуються мови програмування CLIPS і OPS5.
4. ПЗ на основі індукції. Найчастіше застосовується мова програмування SQL.

Розглянемо принципи декларативного підходу до програмування інформаційних систем.

Об'єктно-орієнтоване ПЗ. Основа об'єктно-орієнтованого програмування полягає у тому, що програма розробляється за принципом представлення даних, що використовуються в програмі, у вигляді

об’єктів, з подальшою реалізацією операцій над цими об’єктами. Такий спосіб розробки програми є протилежним по відношенню до способу, що використовується при низхідному проектуванні, тому проектування за допомогою блок-схеми алгоритму тут неприйнятне. Основний спосіб, що застосовується для об’єктно-орієнтованого проектування, заснований на використанні мови UML (Unified Modeling Language – універсальна мова моделювання).

Об’єктно-орієнтовані мови програмування реалізують поняття класу, на основі якого забезпечується необов’язковість безпосереднього застосування операцій обробки інформації на низькому рівні, тобто їх інкапсуляція. Екземпляром класу є об’єкт даних, з яким можна здійснювати необхідні маніпуляції. Він містить властивості та методи управління властивостями і зв’язками з іншими об’єктами, що були визначені в його класі.

Завдяки схожості по суті властивостей і методів із змінними і функціями об’єктно-орієнтоване програмування іноді відносять до процедурних підходів. Проте це не зовсім відповідає істині. Головна якісна відмінність полягає в такому важливому понятті об’єктно-орієнтованого програмування як “спадкоємство”. За допомогою спадкоємства може бути визначений новий клас, який успадковує властивості і методи одного або декількох класів, після чого його можна збагачувати додатковими властивостями і методами.

Поняття спадкоємства може бути розширене для організації об’єктів у вигляді ієрархії, в якій вони можуть успадковувати властивості від своїх класів, а ці класи, у свою чергу, можуть успадковувати властивості від класів вищого рівня і т.д. У цьому випадку процес розробки програмного забезпечення стає вільним від необхідності програмування обробки даних на низькому рівні, яка характерна для процедурного підходу.

Об’єктно-орієнтоване програмування зводиться до визначення необхідних класів, створення об’єктів, що успадковують властивості і методи цих класів, і декларуванню їх поведінки для здійснення операцій з об’єктами даних. На даний час об’єктно-орієнтоване програмування застосовується найбільш широко, оскільки дозволяє знизити витрати на супровід ІС і забезпечити повторне використання програмного коду об’єктів.

Логічне ПЗ. Основою логічного програмування є твердження, що обчислення є окремим випадком логічної дедукції. Тобто, якщо до логічної структури:

Якщо виконується Умова зробити Висновок

застосовується зворотний логічний висновок:

Висновок зроблений Тому що виконалася Умова,

можна отримати достатньо потужний механізм програмування символічних міркувань. У цій формі умови можуть розглядатися як шаблони, з якими повинно бути знайдено відповідність. Вислови, представлені в такій формі, прийнято називати “хорнівськими виразами”, на честь Альфреда Хорна (Alfred Horn), який вперше їх дослідив. У 1972 році Ковальський (Kowalski), Колмерое (Colmerauer) і Руссел (Roussell) створили мову PROLOG для реалізації принципу логічного програмування на основі зворотного логічного висновку з використанням хорнівських виразів. У наші дні терміном “логічне програмування” зазвичай позначається програмування, що здійснюється з використанням мови PROLOG, незважаючи на те, що було розроблено багато інших мов, призначених для цієї мети.

Однією з переваг систем зворотного логічного висновку є те, що виконання закладених у них програм може здійснюватися паралельно. Це означає, що за наявності декількох процесорів існує можливість одночасного їх використання для виконання різних завдань.

ПЗ експертних систем. Експертні системи використовуються з метою проведення аналізу в умовах невизначеності. Невизначеність може бути присутньою у вхідних даних експертної системи, а також у правилах їх обробки. Цей факт робить процес створення експертних систем відмінним від розглянутих раніше підходів до програмування інформаційних систем.

Оскільки велика кількість людських знань є евристичними (приблизними), вони виявляються вірними не у всіх ситуаціях. У такому разі замість використання наявних знань доводиться застосовувати інший підхід. Крім того, вхідні дані можуть виявитися неправильними, неповними, неузгодженими, а також містити помилки різних типів. Алгоритмічні рішення не дозволяють справлятися з подібними ситуаціями, оскільки алгоритм зобов’язаний гарантувати рішення задачі за допомогою кінцевої послідовності кроків, що була визначена заздалегідь.

Залежно від вхідних даних і стану бази знань експертна система може запропонувати правильну відповідь, прийнятну відповідь, неприйнятну відповідь або взагалі не дати відповіді. У будь-якому разі це краще, ніж повна відсутність можливості отримати оцінку ситуації. Крім того, слід враховувати ще один важливий факт: гарна експертна система здатна діяти не гірше порівняно з найкращим вирішувачем задач відповідного класу – експертом-людиною.

Експертні системи можна розглядати як напрям декларативного програмування, оскільки програміст не вказує за допомогою алгоритму як програма повинна досягти заданої мети. Таким чином, оператори, які утворюють програму, не задають жорсткий порядок управління ходом її виконання.

ПЗ на основі індукції. У результаті програмування на основі індукції програма сама будує хід виконання (навчається) шляхом формування узагальнень на основі наданого зразка. Наприклад, у такий спосіб людина могла б навчитися нарощувати послідовність 2, 4, 6, "?".

Одним з напрямів, в якому був успішно застосований даний підхід – програмування запитів до реляційних баз даних. При цьому користувач може не вказувати конкретні значення атрибутів, за якими повинен бути виконаний пошук об'єктів. Для цього достатньо вибрати лише відповідні приклади атрибутів з потрібними характеристиками. Після цього програма доступу до бази даних виявляє логічним шляхом характеристики даних і виконує пошук відповідних записів у базі даних. Як наочні приклади використання розпізнавання образів при пошуку можуть служити системи управління реляційними базами даних, в яких застосовується мова SQL (Structured Query Language – мова структурованих запитів).

Деякі сучасні інструментальні засоби розробки інформаційних систем надають можливість використовувати навчання із застосуванням індукції, у процесі якого вони приймають для аналізу приклади й практичні завдання, а потім автоматично виробляють правила.

1.3. Програмні системи технології "клієнт-сервер"

Сучасні програмні системи, які застосовуються для вирішення економічних задач, розроблені за технологією "клієнт-сервер". "Клієнт-сервер" ("client-server") – це технологія створення програмних рішень, в яких виконання задач з обробки інформації розподіляється між програмою-сервером та програмою-клієнтом.

Система, спроектована за технологією "клієнт-сервер", є інформаційною системою на базі комп'ютера, в якій дані можуть спільно використовуватися й оброблятися багатьма програмними модулями. Саме в підході до організації взаємодії об'єктів даних і прикладних програмних компонентів полягає її принципова відмінність від систем-попередників, спроектованих за технологією "файл-сервер".

Від початку комп'ютерні системи управління процесами, що відбуваються в економіці, зберігають дані в спеціалізованих файлах. Такі спеціалізовані файли є набором записів, кожен з яких відповідає певному об'єкту, містить дані, що визначають атрибути об'єкта. Так, файл книг містить записи про книги, причому кожен запис про книгу складається з полів, в яких вказані номер, назва і автор.

В інформаційній системі, спроектованій за технологією "файл-сервер", множина однотипних файлів створюється й обробляється відповідним програмним рішенням. Можливості нарощування інфор-

маційного обсягу в такій системі істотно обмежені внаслідок високої схильності до збільшення надмірності даних.

Термін "надмірність даних" використовується для опису ситуації, коли одні і ті ж дані зберігаються в різних місцях зовнішньої пам'яті (файлах). Надмірність небажана з декількох причин.

Перша причина – неоднозначність. Виникає ситуація, коли один і той же елемент по-різному називається в довідниках різних програмних компонентів ІС. Ефективна робота системи вимагає відсутності подібних неоднозначностей.

Друга причина – неузгодженість. Коли кожен програмний модуль по-своєму тлумачить значення певного елемента, існує висока ймовірність неузгодженості. Наприклад, якщо вартість елемента запасів змінилася, недостатньо просто змінити її значення в програмі управління запасами. Потрібно також змінити відповідні значення в програмі обробки замовлень та інших модулях, що використовують цей елемент даних. Такий процес називається розповсюдженням оновлень і, зазвичай, вимагає значних ресурсних витрат.

Третя причина – марна праця. Створення записів з даними для підтримки окремого програмного модуля, тоді як частина цих даних вже існує в ІС, є марною витратою часу, сил і коштів.

У середовищі ІС, де окремі програмні рішення створюють і підтримують свої власні файли, ймовірність неузгодженості, неоднозначності й марних витрат праці дуже висока. Цієї проблеми можна уникнути, об'єднавши всі дані в спеціалізовану програмну систему, що управляє їхнім зберіганням і обробкою. Така система називається сервером баз даних і є ядром технології "клієнт-сервер".

В ідеалі всі дані в ІС, спроектованій за технологією "клієнт-сервер", підтримують необмежену кількість різноманітних програмних рішень і записуються тільки один раз у визначеній базі даних. Різні програмні модулі отримують можливість мати доступ до потрібних їм даних, надмірність усувається, і компоненти ІС починають працювати спільно в реальному часі. Слід зазначити, що на практиці повністю усунути надмірність неможливо. Тому база даних зобов'язана контролювати надмірність, забезпечуючи її мінімальний рівень.

Поступово переважна більшість промислових підприємств перейшла до використання клієнт-серверних систем для зберігання й обробки своїх даних. Але не тільки великі підприємства виграли від застосування такого підходу. З появою потужних і простих в обслуговуванні програмних рішень з реалізації централізованих баз даних, що здатні працювати на комп'ютерних системах загального використання, навіть дуже невеликим підприємствам вигідно використовувати ІС клієнт-серверної архітектури для організації і зберігання своїх даних.

Архітектура технології “клієнт-сервер” базується на одному з двох принципів функціонального розподілу: на основі інтелектуального клієнта або на основі тонкого клієнта. Для опису принципів архітектури технології “клієнт-сервер” використовується концепція логічних рівнів. Ці рівні являють собою помітно відокремлені функціональні області, що виконують різні задачі в рамках роботи системи. Архітектура технології “клієнт-сервер” містить наступні рівні:

1. Рівень представлень. Це графічний інтерфейс, що надається користувачам для введення даних і отримання інформації, у зручному для сприйняття вигляді.
2. Рівень обчислень. Містить алгоритми й правила обробки даних системи, відповідно до яких здійснюється інформаційний обмін між рівнем представлень і рівнем даних.
3. Рівень даних. Забезпечує реалізацію концепції інтегрованого використання даних множиною програмних компонентів ІС.

Ґрунтуючись на визначенні цих рівнів, розглянемо архітектуру клієнт-серверних ІС з інтелектуальним і тонким клієнтами.

Інтелектуальний клієнт (іноді використовується термін “дворівневий клієнт-сервер”) – це спосіб взаємодії програмного інтерфейсу користувача з сервером баз даних. Робота користувальницьких модулів ІС такого типу – програмних додатків, тобто спеціалізованих програмних розробок – зазвичай виконується на персональному комп’ютері користувача для організації взаємодії з сервером баз даних.

Програмний додаток, що забезпечує інтерфейс користувача згідно з концептуальним принципом інтелектуального клієнта, містить два вбудовані рівні архітектури “клієнт-сервер”: рівень представлень і рівень обчислень. Такий тип клієнтського ПЗ містить основну частину логічних й обчислювальних процедур, які виконує система. Це дозволяє швидко перевіряти вхідні дані і повертати повідомлення про помилки, не звертаючись до серверних компонентів системи.

Тонкий клієнт (іноді використовується термін “багаторівневий клієнт-сервер”) дозволяє провести більш чітку межу між рівнями ІС. Цей тип архітектури зумовлює розташування рівня представлень – на клієнтському комп’ютері, рівня обчислень – на сервері програмних додатків, а рівень даних – на сервері баз даних.

Архітектура з тонким клієнтом виглядає більш структурованою, ніж з інтелектуальним клієнтом. У ній кожен рівень логічно чітко відокремлений від іншого. В той же час, незважаючи на те, що рівні чітко розділені, у процесі роботи ІС вони інтенсивно взаємодіють та істотно залежать один від одного.

Яка конфігурація краща? Усе залежить від характеру вирішуваних задач. Якщо число користувачів ІС перевищує 50, причому продуктивність їх комп'ютерів різна, то, швидше за все, краще підійде архітектура з тонкими клієнтами. А якщо кількість користувачів системи не перевищує 50, пропускна спроможність обчислювальної мережі невисока, всі локальні комп'ютери однаково достатньо потужні, то слід використовувати архітектуру з інтелектуальними клієнтами.

1.4. Системи управління базами даних

На основі попереднього матеріалу можна зробити висновок, що ІС, побудовані за технологією "клієнт-сервер", завдячують своєму визнанню, здебільшого, завдяки реалізації концепції інтегрованого використання даних. Під концепцією інтегрованого використання даних множиною програмних компонент ІС, за забезпечення якої відповідає рівень даних архітектури "клієнт-сервер", мається на увазі виконання наступних положень:

- по-перше, різні програмні додатки можуть використовувати одні й ті ж дані;
- по-друге, ці дані можуть використовуватися різними програмними додатками в один і той самий час.

Процес інтегрованого використання даних називається паралельним доступом або паралелізмом (concurrency). Паралелізмом необхідно керувати, інакше дані можна легко пошкодити (наприклад, якщо один програмний додаток змінює елемент даних, який використовується в цей момент іншою програмою).

Ці обов'язки покладаються на певний клас програмного забезпечення – систему управління базою даних (СУБД). СУБД реалізовує функції сервера баз даних і виконує роль посередника між програмними додатками і даними. Саме СУБД повинна надавати можливість користувачам отримувати паралельний доступ до даних й керувати ним.

Крім паралелізму СУБД має забезпечувати гарантії безпеки й цілісності бази даних. Користувачі системи повинні мати нагоду захистити свої дані від несанкціонованого доступу, а також відновити їх у разі будь-яких системних збоїв. Централізоване управління безпекою даних – одна з найбільш важливих особливостей СУБД.

У системах, розроблених за технологією "клієнт-сервер", дані продовжують фізично зберігатися у файлах. Доступ до них здійснюється операційною системою комп'ютера. Важливо те, що різні програмні додатки можуть діставати доступ до різних частин загальної множини файлів. Кожен додаток буде використовувати необхідну підмножину даних, яка контролюється системою.

Роль СУБД полягає в тому, щоб генерувати запити, що дозволяють використовувати функціональні можливості системи управління файлами комп'ютера, на зовнішній пам'яті якого вони розташовані – хосту (host), для обслуговування різноманітних програмних додатків. Під "хостом" розуміють комп'ютер, що надає сервіси формату "клієнт-сервер" в режимі сервера через інтерфейси, на яких він є унікально визначеним. СУБД – це додатковий рівень програмного забезпечення, надбудований над системним програмним забезпеченням хосту.

Та частина інформації з бази даних, яка потрібна певному додатку, називається представленням (view). Окремі елементи даних можуть мати зовсім різний вигляд в залежності від того, за допомогою якого представлення проводиться звернення до них. Одні і ті ж дані можуть мати різні імена в різних представленнях. СУБД повинна підтримувати ці можливості різноманітного представлення частин загального обсягу даних.

Кожне представлення бази даних – це окрема логічна структура, побудована з фізичних даних, що лежать в її основі. Щоб забезпечити інтерфейс між фізичною пам'яттю бази даних та її різноманітними логічними версіями, СУБД, у свою чергу, повинна поділятись на декілька рівнів.

У будь-якій системі з базою даних є центральний, або так званий концептуальний рівень – логічне представлення даних системи. Концептуальний рівень повинен характеризуватися:

- незалежністю від того, як фізично зберігаються дані;
- повнотою, тобто він повинен містити опис усіх даних, що зберігаються в системі.

Концептуальний рівень СУБД складається з усіх об'єктів бази даних, доступних користувачам через програмні додатки. Об'єкт бази даних – це її певний логічний елемент. Залежно від типу бази даних її користувачам будуть доступні різні типи об'єктів.

Концептуальний рівень СУБД є останнім рівнем представлення даних, що доступний користувачу бази даних. Користувачі навмисно усунені від розгляду питань про те, як насправді зберігаються дані на фізичному рівні.

СУБД повинна мати змогу підтримувати різноманітні представлення одних і тих самих даних. Загальне поняття такого представлення полягає у відображенні інформації, що зберігається в базі даних у вигляді, який необхідний конкретному додатку або множині додатків.

Одна й та ж множина даних може відображатися в різні представлення. Сукупність всіх представлень утворюють так званий зовнішній рівень бази даних – інтерфейс між базою даних і її користувачами. Якщо концептуальна схема бази даних модифікується, то всі представлення, які причетні до цієї модифікації, необхідно буде змінити так,

щоб вони залишались для своїх користувачів незмінними. Логічна незалежність даних полягає в усуненні користувачів та їх додатків від зміни логічного представлення бази даних.

Існує також інша форма незалежності даних, так звана фізична незалежність даних. Вона полягає в усуненні користувачів і додатків від змін, що відбуваються у фізичному сховищі бази даних. Фізичне сховище пересічної бази даних часто піддається оновленням і змінам з метою підвищення продуктивності ІС та оперативного відображення змін, що відбуваються з реальними об'єктами, дані про які зберігаються в базі даних. На найнижчому рівні СУБД повинна встановити відповідність між представленням бази даних у вигляді концептуальної схеми та її фізичним представленням.

Це відображення називається внутрішнім рівнем системи з базою даних. Він виступає як інтерфейс між СУБД і операційною системою комп'ютера, на якому вона встановлена. Якщо фізичне сховище бази даних змінюється, то СУБД повинна на внутрішньому рівні знов встановити відповідність концептуальної схеми новому фізичному представленню. Сама концептуальна схема повинна залишитися незмінною.

Таким чином, СУБД складається з трьох рівнів: множини відображень концептуального рівня в представлення користувачів, самого концептуального рівня і відображення концептуального рівня у фізичне сховище. Ці три рівні називаються, відповідно, зовнішнім, концептуальним і внутрішнім рівнями.

Такий розподіл СУБД на рівні був запропонований як стандарт ANSI/SPARC (1978). Насправді такий чіткий розподіл рідко використовується на практиці. Зокрема, з міркувань підвищення продуктивності, СУБД часто не звертається до засобів операційної системи комп'ютера, на якому вона встановлена, а сама виконує ті операції, які стосуються управління файлами.

ПИТАННЯ ТА ЗАВДАННЯ

1. Що розуміють під мовою програмування?
2. Для позначення якого типу програм використовується термін "процедурне програмування"?
3. У чому полягає імперативний принцип програмування?
4. Зобразіть структуру фон-нейманівської машини.
5. Охарактеризуйте загальні принципи функціонального програмування.
6. З яких частин складається функціональна мова?
7. У чому полягає сутність декларативних підходів до створення програмних систем?

8. Охарактеризуйте аспекти об’єктно-орієнтованого програмування.
9. Яка властивість об’єктно-орієнтованого програмування не дозволяє віднести його до процедурних підходів?
10. В яких ситуаціях доцільно використовувати методи логічного програмування?
11. Опишіть загальне призначення експертних систем.
12. Назвіть область застосування програмного забезпечення на основі індукції.
13. Дайте визначення технології “клієнт-сервер”.
14. Проведіть порівняльний аналіз інформаційних систем, спроектованих за технологією “клієнт-сервер” та технологією “файл-сервер”.
15. Назвіть причини, за якими надмірність даних в інформаційних системах є небажаною.
16. З яких логічних рівнів складається архітектура технології “клієнт-сервер”?
17. Ґрунтуючись на концепції логічних рівнів, дайте порівняльну характеристику клієнт-серверних інформаційних систем з інтелектуальним і тонким клієнтами.
18. Виконання яких положень в організації інформаційних систем забезпечує інтегроване використання даних?
19. Яку роль виконує СУБД у системах, розроблених за технологією “клієнт-сервер”?
20. Назвіть і охарактеризуйте рівні СУБД.

ТЕСТИ

1. Що є відмінною особливістю декларативного підходу програмування інформаційних систем?
   а) у програмі необхідно точно вказувати, як повинно бути реалізоване рішення задачі;
   б) у програмі необхідно вказати, що повинно бути зроблено, і дозволити системі визначити, як це зробити;
   в) програма розробляється за принципом представлення даних у вигляді об’єктів, з подальшою реалізацією операцій над цими об’єктами.
2. Який принцип програмування інформаційних систем ґрунтується на зворотному логічному висновку?
   а) логічне програмування;
   б) імперативне програмування;
   в) об’єктно-орієнтоване програмування.
3. З якою метою застосовуються експертні системи?
   а) обробка списків;

б) проведення аналізу в умовах невизначеності;
в) управління доступом до реляційних баз даних.

4. Чим відрізняється інформаційна система, створена за технологією “клієнт-сервер”, від інформаційної системи, створеної за технологією “файл-сервер”?
   а) нічим;
   б) у клієнт-серверній системі дані не зберігаються у файлах;
   в) у клієнт-серверній системі файли не прив’язані до визначених додатків.
5. Що розуміють під надмірністю даних в інформаційних системах?
   а) ситуація, коли загальний об’єм даних перевищує 100 GB;
   б) ситуація, коли одні і ті ж дані зберігаються в декількох файлах;
   в) ситуація, коли кількість записів в одному файлі перевищує 10 0000 рядків.
6. Які проблеми не можуть виникнути через надмірність даних?
   а) марна праця;
   б) втрата зв’язків;
   в) неузгодженість;
   г) неоднозначність.
7. Який з принципів створення систем за технологією “клієнт-сервер” передбачає найбільшу кількість логічних рівнів?
   а) тонкий клієнт;
   б) інтелектуальний клієнт;
   в) кількість логічних рівнів завжди однакова.
8. Яку роль виконує СУБД у системі з базою даних?
   а) СУБД запобігає вірусним атакам;
   б) СУБД виконує роль посередника між користувальницькими додатками і даними;
   в) СУБД виконує роль диспетчера в інформаційному обміні між функціональними пристроями ІС.
9. Що таке логічна незалежність даних?
   а) усунення програмних додатків від зміни фізичного сховища даних;
   б) усунення програмних додатків від зміни представлення інформації;
   в) усунення користувачів і додатків від змін даних, створюваних іншими користувачами і додатками.
10. З яких рівнів складається СУБД?
    а) представлення, обчислень, даних;
    б) концептуального, логічного, фізичного;
    в) додатків, операційної системи хосту, даних.

РОЗДІЛ 2
ЗАСОБИ СТВОРЕННЯ ПРОГРАМНИХ ДОДАТКІВ

2.1. Сучасна ідеологія відкритого програмування

У розділі, присвяченому методологічним підходам до програмування інформаційних систем, було розглянуто класичну класифікацію принципів програмування. Згідно з нею було виділено процедурне і декларативне програмування, а також їх різновиди: імперативне, функціональне, об'єктно-орієнтоване, логічне, експертних систем та на основі індукції. Ці принципи до певного моменту застосовувалися в рамках ідеології побудови цільових інформаційних систем, призначених вирішувати певний спектр задач. Згодом невпинне зростання потреб людства в глобальному спілкуванні примусило змінити ідеологію принципів програмування.

Для програмних продуктів глобального інформаційного суспільства характерними є високі вимоги до їх комунікативних складових. Це зумовило перехід від створення монолітних рішень до створення компонентів, що допускають своє повторне використання в різних середовищах і програмних додатках.

Зміна ідеології в розробці програмних систем була відмічена провідними представниками IT-індустрії, появою якісно нового покоління програмних продуктів. Деякі виробники програмних систем інформують ринок про належність своєї продукції до відкритої ідеології, наділяючи їх характерними зовнішніми ознаками. Зокрема, для продуктів фірми Microsoft, випущених з початку XXI століття, характерним є закінчення назви на .Net (читається як Dot Net). Спираючись саме на ці рішення, далі буде проведений розгляд сутності ідеології відкритого програмування.

Однією з практичних реалізацій ідеології відкритого програмування є реалізована в останніх версіях Microsoft Visual Studio відкритість для мов програмування. Вона полягає у використанні багатомовного середовища розробки. Тобто у середовище розробки програмних додатків Visual Studio останніх версій разом із мовами програмування, включених фірмою Microsoft (Visual C++, Visual C#, J# .Net, Visual Basic .Net), можуть додаватися будь-які мови програмування, компілятори яких створюються іншими фірмами-виробниками. На сьогоднішній день таких розширень середовища Visual Studio зроблене вже

достатньо багато, практично, вони існують для всіх відомих мов (Fortran, Cobol, Component Pascal, Oberon та ін.).

Відкритість середовища не означає повної свободи. Усі розробники компіляторів при введенні нової мови в середовище розробки повинні дотримуватись установлених правил та обмежень. Головне обмеження, яке одночасно можна вважати і головною перевагою, полягає в тому, що всі мови, які включаються в середовище розробки Visual Studio, повинні використовувати єдиний каркас – Framework.Net.

Поняття каркасу додатків – Framework Applications – з'являється в літературних джерелах з другої половини 90-х років минулого сторіччя в описах застосування Visual Studio, починаючи з четвертої версії. Роль каркасу додатків Visual C++ у ранніх версіях Visual Studio виконувала бібліотека класів MFC (Microsoft Foundation Classes). Бібліотека класів MFC від початку являла собою ієрархічно організовану колекцію класів, в яку входили класи, здатні створювати архітектуру нових додатків. Вибираючи тип додатка, розробник отримував потрібну функціональну платформу, що утворювалась та підтримувалась об'єктами класів каркасу.

Наприклад, коли розробник вибирав з можливих типів додатків архітектуру "Документ-Представлення", то в його додаток автоматично вбудовувався клас Document, відповідальний за структуру документа, і клас View, відповідальний за його візуальне представлення. Клас Form, разом з іншими класами, що реалізовували елементи управління, забезпечували уніфікований інтерфейс додатків.

Упродовж подальших років роль каркаса в побудові додатків істотно зросла за рахунок розширення його можливостей до рівня Framework.NET. На сьогодні каркас Microsoft Framework.NET є платформою для створення, розгортання і запуску програмних додатків, яка надає високопродуктивне, засноване на стандартах багатомовне середовище, яке дозволяє інтегрувати існуючі додатки з додатками і сервісами наступних поколінь.

Завдяки застосуванню єдиного каркаса Framework.Net досягаються наступні переваги:

- можливість використання компонентів, розроблених різними мовами;
- можливість розробки декількох частин одного додатка різними мовами програмування;
- можливість безшовного налагодження багатомовного додатка;
- можливість створювити клас однією мовою, а його нащадків – іншими мовами.

Єдиний каркас стимулює зближення мов програмування, дозволяючи разом з тим зберігати їх індивідуальність та переваги, які вони мають. Завдяки єдиному каркасу до певної міри вирішується проблема мовного бар'єра серед програмістів.

У результаті еволюції каркаса відбувається природний процес його відокремлення від середовища розробки – Framework.Net стає надбудовою над операційною системою. У 2001 році Європейська асоціація виробників комп'ютерів (ECMA) прийняла компоненти каркаса як стандарт. Внаслідок цього каркас Framework.Net дістав можливість розвиватися для застосування на операційних платформах, відмінних від Windows.

На сьогодні каркас Framework.Net стає вільно розповсюджуваним технологічним рішенням. Це істотно розширює сферу його застосування. Виробники різних програмних продуктів вважають за краще орієнтувати свої розробки на застосування каркаса Framework.Net з метою забезпечення можливості виконання їхнього коду на різних операційних платформах.

У складі каркаса Framework.Net можна виділити два основні компоненти:

1. Статичний – FCL (Framework Class Library) – бібліотека класів каркаса.
2. Динамічний – CLR (Common Language Runtime) – загальномовне виконавче середовище.

Бібліотека класів FCL є результатом еволюції бібліотеки класів MFC, завдяки чому каркас Framework.Net став єдиним середовищем для різних мов програмування. Тому, якою б мовою програмування не велася розробка, вона використовує класи однієї загальної бібліотеки. Більшість класів бібліотеки, що утворюють загальне ядро, використовуються всіма мовами каркаса. Таким чином досягається уніфікація наступних реалізацій:

- інтерфейсу додатків, незалежно від мови, якою вони розробляються;
- взаємодії з колекціями та іншими контейнерами даних;
- доступу до різних типів зовнішніх джерел даних.

Крім того, бібліотека класів FCL містить ряд статичних компонентів, що забезпечують відкритість програмування в середовищі Visual Studio. Серед них слід виділити: вбудовані примітивні типи даних, структурні типи даних, компоненти підтримки архітектурного різноманіття додатків, простори імен.

Вбудовані примітивні типи даних. Важливою частиною бібліотеки FCL стали класи, що описують примітивні типи даних. Типи кар-

каса охоплюють усю множину типів даних, що зустрічаються в мовах програмування. Типи даних мови програмування проектуються на відповідні типи каркасу. Наприклад, тип даних, відомий у мові Visual Basic як Integer, а в мові C# як int, проектується на тип даних FCL Int32. У кожній мові програмування, разом з "рідними" для мови назвами типів даних, дозволяється використовувати імена типів, прийняті в каркасі. Як наслідок, усі мови середовища розробки можуть користуватися єдиною системою вбудованих типів даних, що забезпечує взаємодію компонентів, написаних різними мовами.

Структурні типи даних. Частиною бібліотеки стали не тільки прості вбудовані типи даних, але й структурні типи, що описують організацію складних структур даних: строки, масиви, переліки, записи. Це також сприяє уніфікації та реальному зближенню мов програмування.

Компоненти підтримки архітектурного різноманіття додатків. У середовищі розробки існує широкий набір можливих архітектурних типів додатків. Крім традиційних Windows-додатків та консольних додатків, існує можливість створення платформ для Web-додатків. Велика увага приділяється можливості створення повторно використовуваних компонентів – дозволяється будувати бібліотеки класів, бібліотеки елементів управління. Компілятори мов, що поставляються різними фірмами для створення проектів, можуть використовувати як бібліотеку FCL, так і власну бібліотеку класів.

Простори імен. Кількість класів бібліотеки FCL досягла значного рівня (декілька тисяч), тому виникла потреба у способі їх структуризації. Логічним чином класи з близькою функціональністю об'єднуються в групи, що звуться простором імен (Namespace). Основним простором імен бібліотеки FCL є простір System, що містить, поряд з класами, інші – вкладені простори імен. Наприклад, примітивний тип Int32 безпосередньо вкладений у простір імен System, і його повне ім'я, що включає ім'я простору – System.Int32. У простір System вкладений цілий ряд інших просторів імен, що використовуються при створенні додатків.

Перехід до ідеології відкритого програмування в каркасі Framework.Net реалізований багато в чому завдяки його динамічному компоненту – загальномовному виконавчому середовищу CLR. Вирішення своїх задач виконавче середовище здійснює, ґрунтуючись на наступних складових: керований модуль, віртуальна машина, метадані, збірник сміття, обробник виключень, події та загальні специфікації.

Керований модуль. За допомогою керованого модуля і керованого коду реалізується основна концепція виконавчого середовища каркасу

– двохетапна компіляція. Керований модуль – це переміщуваний виконавчий файл або PE-файл (Portable Executable). PE-файли являють собою модулі, зміст яких формується компіляторами мов програмування проміжною мовою – IL (Intermediate Language). Залежно від обраного типу проекту, PE-файл може мати розширення exe, dll, mod або mdl.

Незважаючи на те, що PE-файл має розширення exe, він виконується операційною системою не зовсім так, як звичний exe-файл. При його запуску він розпізнається як спеціальний проміжний файл, і передається виконавчому середовищу для обробки. Виконавче середовище починає працювати з кодом, в якому не залишилося жодної специфіки початкової мови програмування. Код проміжною мовою починає виконуватися під управлінням виконавчого середовища.

Віртуальна машина. Результат роботи виконавчого середовища каркасу можна розглядати як своєрідну віртуальну машину. Ця машина транслює ділянки проміжного коду, що подається на виконання, у команди реального процесора, який насправді і виконує код. Основу віртуальної машини складають транслятори JIT (Just In Time Compiler), які й виконують трансляцію проміжного коду в командний код тієї обчислювальної машини, де встановлене і функціонує виконавче середовище.

Microsoft у своїй розробці використав досвід віртуальної машини Java. Він отримав широке визнання, покращивши процес за рахунок того, що на відміну від Java, проміжний код не інтерпретується виконавчим середовищем, а компілюється з урахуванням усіх особливостей обчислювальної платформи. Завдяки цьому існує можливість створювати більш продуктивні додатки. Крім того, виконавче середовище, працюючи з проміжним кодом, здійснює достатньо ефективну оптимізацію програмного коду і, що не менш важливо, його захист.

Метадані. Переміщуваний виконавчий PE-файл є самодокументованим файлом, тобто містить разом з програмним кодом метадані, які його описують. Файл починається з маніфесту, що включає опис усіх класів, які в ньому зберігаються, їх властивостей, методів, усіх аргументів цих методів, тобто всю необхідну для CLR інформацію. Тому, крім PE-файлу, не вимагається ніяких додаткових файлів і записів у реєстр – уся необхідна інформація береться з самого файлу.

Збірник сміття (Garbage Collector). Під збором сміття розуміється звільнення оперативної пам’яті, зайнятої об’єктами, які стали зайвими і не використовуються в подальшій роботі додатка. У багатьох мовах програмування (класичним прикладом є мова C/C++) пам’ять звільняє сам програміст, в явній формі програмуючи команди як на створення, так і на видалення об’єктів. Для того, щоб запобігти неминучим

помилкам програміста при роботі з пам'яттю, видалення невживаних об'єктів, збірка сміття стала частиною виконавчого середовища.

Обробник виключень. У випадках, коли при виклику деякої функції (процедури) виявляється, що вона не може коректно виконати свою роботу, виконавче середовище викидає виключення. Викидання виключень найкращим чином погоджує процес програмування з виконавчим середовищем. У процесі розробки програмних систем організація перехоплення викинутих виключень і їх подальша обробка являє собою основний рекомендований спосіб реакції програми на нестандартні ситуації.

Події. У виконавчого середовища існує своє бачення того, що є типом кожного об'єкта. Для цього використовується формальний опис загальної системи типів CTS – Common Type System. Відповідно до цього опису кожен тип, крім методів і властивостей, може містити ще й події. При виникненні подій у процесі роботи з тим чи іншим об'єктом певного типу надсилаються повідомлення, які можуть одержувати та використовувати інші об'єкти. Механізм обміну повідомленнями заснований на делегатах – функціональному типі.

Загальні специфікації. Як вже зазначалося, каркас Framework.Net забезпечує міжмовну взаємодію. Для того, щоб класи, розроблені різними мовами, могли використовуватися в рамках одного додатка, тобто їхні різномовні нащадки мали можливість взаємодіяти, вони повинні задовольняти деякі обмеження. Ці обмеження задаються набором загальномовних специфікацій – CLS (Common Language Specification). Клас, що задовольняє специфікації CLS, називається CLS-сумісним. Він доступний для використання в інших мовах, класи яких можуть бути клієнтами або спадкоємцями сумісного класу.

Специфікації CLS точно визначають, яким набором вбудованих типів даних можна користуватися в сумісних модулях. Зрозуміло, що ці типи повинні бути загальнодоступними для всіх мов, що використовують каркас Framework.Net. У сумісних модулях повинні використовуватися керовані дані і виконуватися деякі додаткові обмеження. Слід звернути увагу на те, що обмеження торкаються тільки інтерфейсної частини класу – його відкритих властивостей і методів. Закрита частина класу може не задовольняти загальну специфікацію. Отже, класи, яким не потрібна сумісність, можуть використовувати специфічні особливості мови програмування, якою вони створені, без обмежень.

2.2. Засоби створення програмних додатків під ОС Windows

Одним з найпоширеніших видів користувальницьких додатків на сьогоднішній день є проекти, які широко використовують елементи

графічного інтерфейсу. Ці проекти розробляються і виконуються на апаратних платформах, що функціонують під управлінням ОС Windows і виконують типові задачі інтелектуального клієнта інформаційної системи клієнт-серверної архітектури. Для того, щоб визначити сутність проектів сучасних Windows-додатків, необхідно розглянути сутність пов'язаних понять: рішення (solution), проект (project), простір імен (namespace), збірка (assembly).

З погляду програміста, що працює над проектом, результатом роботи компілятора середовища розробки (Visual Studio) буде рішення. З позицій загальномовного середовища виконання CLR каркасу Framework, компілятор створює збірку, що містить переміщуваний виконавчий файл.

Рішення містить один або декілька проектів. Якщо рішення містить декілька проектів, один з них повинен бути призначений стартовим. Виконання рішення починається із стартового проекту. Проекти одного рішення можуть бути залежними або незалежними. Наприклад, усі проекти одного структурного підрозділу довільно обраного підприємства можуть бути для зручності зібрані в одному рішенні і мати загальні властивості.

Проекти Windows-додатків розробляються на основі об'єктно-орієнтованого програмування, тобто дані представляються у вигляді екземплярів класів – об'єктів, з подальшою реалізацією операцій над цими об'єктами. Стартовий проект повинен мати точку входу – клас, що містить статичну процедуру з ім'ям Main. У момент запуску рішення на виконання цій процедурі автоматично передається управління додатком. В існуюче рішення можна додавати як нові, так і існуючі проекти. Один проект може входити в декілька рішень.

Класи проекту систематизовані в одному або декількох просторах імен. Простори імен дозволяють структурувати проекти, що містять велику кількість класів, об'єднуючи в одну групу близькі класи. Якщо над проектом працюють декілька виконавців, то, як правило, кожен з них створює свій простір імен. Крім структуризації це дає можливість присвоювати класам імена, не замислюючись про їх унікальність за межами простору.

Проект являє собою основну одиницю, з якою працює програміст для створення Windows-додатка. При використанні як середовища розробки Visual Studio він вибирає відповідний тип, після чого механізм середовища створює початкову структуру проекту з точкою входу.

За замовчуванням початкова структура Windows-додатка містить клас Form1 – спадкоємець класу Form бібліотеки FCL. Цей клас містить точку входу в проект (процедуру Main), що викликає статичний

метод Run класу Application. Метод Run призначений для створення об'єкта класу Form1 і відкриття його видимого образу – форми.

Форми являють собою візуальні об'єкти, що забезпечують функціональність Windows-додатка. Зазвичай, Windows-додаток містить декілька форм, які відкриваються і закриваються в процесі роботи. У кожен конкретний момент на екрані може бути відкрито одну або декілька форм. Користувач має можливість працювати з однією формою або перемикатися в ході роботи на іншу.

Слід чітко розрізняти процес створення форми, тобто об'єкта, що належить класу Form (спадкоємцю цього класу) і процес виведення форми на екран. Для показу форми слугує метод Show того класу, який викликається відповідним об'єктом. Для заховування форми використовується метод Hide. На практиці методи Show і Hide змінюють властивість Visible об'єкта, встановлюючи його значення або true (так), або false (ні).

Між заховуванням форми та її закриттям існує різниця і реалізуються ці події різними методами: Hide і Close. Перший з них робить форму невидимою, проте сам об'єкт існує. Метод Close закриває форму, звільняючи зайняті нею ресурси. Викликати метод Show після виклику методу Close неможливо, якщо тільки не створити об'єкт наново. Відкриття і показ форми завжди означає виклик методу Show. Тобто форми мають метод Close, але не мають методу Open. Форми, як і всі об'єкти, створюються при виклику конструктора форми.

Форма, що відкривається в процедурі Main при виклику методу Run, називається головною формою проекту. Її закриття призводить до закриття решти форм і завершення Windows-додатка. Завершити додаток можна також програмно, викликавши в потрібний момент статичний метод Exit класу Application. Закриття інших форм не призводить до завершення проекту. Зазвичай головна форма проекту завжди відкрита, тоді як решта форм відкривається і закривається (ховаються).

Можна створювати форми як об'єкти класу Form. Проте такі об'єкти досить рідкісні. Найчастіше створюється спеціальний клас FormX – спадкоємець класу Form. Зокрема, так відбувається під час створення початкової платформи Windows-додатка, коли створюється клас Form1 – спадкоємець класу Form. Можлива ситуація, коли новостворювана форма багато в чому повинна бути схожою на вже існуючу, тоді клас нової форми може бути зроблений спадкоємцем класу існуючої форми.

У структурі Windows-додатків застосовуються модальні і немодальні вікна. Вікно називається модальним, якщо не можна закінчити роботу у відкритому вікні до тих пір, доки воно не буде закрите. Немода-

льні вікна допускають паралельну роботу у вікнах. Вікно стає модальним або немодальним в залежності від властивостей його форми. Метод Show відкриває форму як немодальну, а метод ShowDialog – як модальну.

Часто в структурі Windows-додатків різні форми повинні працювати з одним об'єктом, проводячи з ним різні операції. Звичайно, така схема реалізується наступним чином:

- об'єкт створюється в одній з форм (найчастіше, у головній);
- при створенні наступної форми глобальний об'єкт передається конструктору нової форми як аргумент;
- одна з властивостей нової форми повинна представляти посилання на об'єкт відповідного класу так, що конструктору залишиться тільки пов'язати посилання з переданим йому об'єктом.

Зазначений порядок дій можна характеризувати як ефективне програмування, оскільки об'єкт створюється лише один раз, а різні форми можуть його використовувати, оперуючи виключно посиланнями на нього.

Компоненти інтерфейсу форм Windows-додатків реалізуються наявними в них елементами управління. Елементи управління призначені для відображення інформації користувачам і реакції на певні події. Елементи управління Windows-додатків можна згрупувати в декілька функціональних груп.

1. Група командних об'єктів. Елементи управління Button, LinkLabel, ToolBar реагують на натискання активної зони та ініціюють певну подію. Це найбільш поширена група елементів.
2. Група текстових об'єктів. Більшість додатків надають можливість користувачу вводити текст і, у свою чергу, виводять різну інформацію у вигляді текстових записів. Елементи TextBox, RichTextBox приймають текст, а елементи Label, StatusBar виводять її. Для обробки введеного користувачем тексту, як правило, слід натиснути на один або декілька елементів із групи командних об'єктів.
3. Група перемикачів. Додаток може містити декілька визначених варіантів виконання дії або рішення задачі. Елементи управління цієї групи надають можливість користувачу здійснити вибір. Це одна з найпоширеніших груп елементів, в яку входять ComboBox, ListBox, ListView, TreeView та багато інших.
4. Група контейнерів. З елементами цієї групи дії додатку практично ніколи не пов'язуються, але вони мають велике значення для організації інших елементів управління, їх систематизації та загального дизайну форми. Як правило, елементи цієї групи використовуються як візуальна платформа для кнопок, текстових полів, списків. Еле-

менти Panel, GroupBox, TabControl реалізують розділення інтерфейсу додатку на логічні групи, забезпечуючи зручність роботи.

5. Група графічних елементів. Часто Windows-додаток містить графіку: іконки, заставку, вбудовані зображення. Для розміщення та відображення їх на формі використовуються елементи для роботи з графікою: Image List, Picture Box.
6. Діалогові вікна. Виконуючи різні операції з документом (відкриття, збереження, друк, попередній перегляд) необхідно використовувати відповідні діалогові вікна. Програмісту, що застосовує сучасне середовище розробки додатків, немає потреби займатися створенням вікон стандартних процедур, оскільки елементи OpenFileDialog, SaveFile Dialog, ColorDialog, PrintDialog містять уже готові операції.
7. Група меню. Практично в усіх Windows-додатках присутнє меню, що містить у собі доступ до функціональних можливостей додатка. Елементи MainMenu, ContextMenu являють собою готові форми для внесення заголовків і пунктів меню.

Розміщувати на формі елементи управління можна двома способами. По-перше, у режимі проектування використовувати графічний інтерфейс середовища розробки для візуального розміщення на поверхні форми необхідних елементів. При цьому як тільки на формі з'являється новий елемент управління, в тексті класу негайно з'являються відповідні рядки програмного коду. По-друге, у програмному режимі писати відповідні рядки коду, що призводить до появи елементів управління на формі.

Слід пам'ятати, що форма – це видимий образ класу Form, а елементи управління, які знаходяться на формі – це видимі образи екземплярів класів (об'єктів), спадкоємців класу Control. Отже, форма з її елементами управління є прямим віддзеркаленням програмного коду. Кожен вид елементів управління описується власним класом. Бібліотека FCL містить велику кількість класів, які реалізують елементи управління. Усі ці класи є нащадками єдиного прабатька – класу Object.

Клас Control в ієрархії класів займає досить високе положення. Він має два важливі батьківські класи: клас Component, що визначає можливість елементам управління бути компонентами, і клас MarshalByRefObject, що надає можливість передачі елементів управління по мережі. Клас Control визначає важливі властивості, методи і події, які успадковуються всіма його нащадками. Усі класи елементів управління є спадкоємцями класу Control.

Клас Form також є одним з нащадків класу Control. Таким чином, форма, по суті, це елемент управління зі спеціальними властивостями.

Будучи спадкоємцем класів ScrollableControl і ContainerControl, форма допускає прокрутку та розміщення елементів управління.

Сучасний інструментарій розробки програмних систем, реалізований у Microsoft Visual Studio, надає можливості створювати власні елементи управління і розміщувати їх на формах разом із вбудованими елементами. Значна кількість фірм спеціалізуються на створенні елементів управління як одного з видів повторно використовуваних програмних компонентів.

2.3. Базові концепції Web-програмування

Як уже зазначалося в попередніх підрозділах, концепція тонкого клієнта інформаційної системи клієнт-серверної архітектури ґрунтується на організації рівня представлення на клієнтському комп'ютері рівня обчислень на сервері програмних додатків, а рівня даних – на сервері баз даних. Рівень представлення інформації користувачу здійснюється за допомогою стандартних програм поступу до джерел інформації, організованих відповідно до Інтернет-технології – Web-браузерів. Рівень обчислень здійснюється на основі серверного програмного забезпечення, що реалізовує функціонування Інтернет-вузлів, надаючи доступ до спеціалізованих джерел інформації – Web-сторінок. Таким чином, технологія створення програмних додатків згідно з концепцією тонкого клієнта технології "клієнт-сервер" зводиться до програмування цих сторінок, яке відоме під назвою Web-програмування.

Web-стрінки створюються за допомогою мови розмітки гіпертексту HTML (Hypertext Markup Lenguage). Він складається з набору позначень – тегів, які використовуються для розмітки тексту, що знаходиться між ними. Теги обмежуються символами <> – початок розмітки та </> – кінець розмітки. Наприклад:

<B> Текст буде виділений жирним шрифтом </B>

Сторінки HTML прийнято розділяти на статичні та динамічні.

Статичні сторінки однаково відображаються для всіх користувачів і не надають можливості змінювати їх вміст. Тобто Web-браузери користувачів мають нагоду відображати тільки той HTML-код цих сторінок, який задав розробник. Статичні Web-сторінки є простими файлами, що містять текст, розмічений тегами. Їх можна створювати за допомогою практично будь-якого текстового редактора в будь-якій операційній системі. Оскільки статичний продукт не дозволяє користувачу індивідуалізуватися і змінювати дані, його створення не співвідноситься з процесом програмування, а мову HTML не прийнято вважати безпосередньою мовою програмування.

Динамічні сторінки надають користувачам інформацію, яка відрізняється від перегляду до перегляду, їх зміст залежить від того, кому вони призначені. Їх застосування дозволяє забезпечити двосторонній обмін інформацією між сервером та клієнтом. Динамічні web-сторінки проходять цикл обробки на сервері перед відправкою клієнту. Програмне забезпечення, що реалізує процес цієї обробки, є результатом Web-програмування.

Як приклад може бути наведена гіпотетична програма, яка модифікує запрошувані клієнтом статичні сторінки, використовуючи параметри одержаного запиту та сховище даних. Тобто достатньо підготувати статичні сторінки – шаблони і, перед відправкою клієнтам, програмно модифікувати їх HTML-код, підставляючи в нього значення, одержані з бази даних. Розробка подібного програмного забезпечення ґрунтується на застосуванні різних технологічних підходів, розвиток яких відбувався впродовж останнього десятиріччя.

Більшість сторінок на ранніх стадіях розвитку Інтернету були статичними. Подальший успішний розвиток “глобальної павутини” відбувся багато в чому завдяки розвитку технології динамічних сторінок. Цей розвиток був, здебільшого, стимульований вимогою користувачів Інтернету бути активними дійовими особами інформаційного простору. Наприклад, вони прагнуть замовляти товари в інтернет-магазинах, брати участь в аукціонах, одержувати інформацію про рух засобів на банківських рахунках і т.д. Динамічні сторінки задовольняють ці потреби завдяки здатності підлаштовуватись під конкретного користувача, а також реагувати на його дії в браузері.

Початок еволюції мов програмування, здатних динамічно змінювати вміст Web-сторінки, завдячує використанню так званих мов скриптів, що виконуються безпосередньо на обчислювальних ресурсах клієнта. Програмний код скриптових мов призначається для виконання тільки під управлінням відповідного програмного інтерпретатора. Найвідомішими з цих мов вважаються JavaScript і VBScript. Скрипти цими мовами вбудовуються в код HTML, який сервер посилає браузеру. Сценарії, що виконуються на стороні клієнта, виділяються тегами <SCRIPT> та </SCRIPT>. Браузер інтерпретує цей код і показує користувачу результат.

Слід відмітити, що технологія скриптового програмування не надає можливості реалізувати обчислювальний процес, складність якого перевищує мінімальний рівень. Наприклад, якщо потрібно виконати складний аналіз інформації, що міститься в базі даних, часто не уявляється можливим відправити користувачу потрібну кількість її вмісту.

Незважаючи на це, розробка скриптів досягла значного рівня популярності завдяки отриманій можливості успішного розв’язання ряду

задач за їх участю. Наприклад, скрипти можуть перевірити правильність запиту, що вводиться в інтерфейс сторінки, і тоді не доведеться перенавантажувати сервер обробкою неправильних запитів. Деякі програмісти створюють на JavaScript анімаційні ефекти. Однак можливості виконання сценаріїв на стороні клієнта недостатньо для створення повноцінних динамічних сторінок. Навіть якщо на сторінці використовується JavaScript і анімовані картинки .GIF, у наш час її прийнято вважати статичною.

Динамічна Web-сторінка повинна створюватися в потрібний момент часу програмою, що виконується на Інтернет-сервері. Для цього широко застосовується механізм шлюзів CGI (Common Gateway Interface). Спочатку користувач дістає доступ до статичної сторінки з формою, яка реалізує для нього інтерфейс, і задає адресу (URL) виконуваного додатка. Додаток розташовується на відповідній адресі серверної платформи і є одним з виконуваних файлів програм, написаних мовами прикладного програмування (наприклад, на C++/C#). Цей додаток приймає по протоколу HTTP дані з вхідного потоку, проводить необхідні обчислення і, залежно від їх результатів, записує у вихідний потік результуючу Web-сторінку. Тобто користувачу у відповідь на його запит посилається HTML-код, який Web-додаток згенерував спеціально для нього.

Web-додатки, що викликаються користувачами за технологією CGI, завантажуються в оперативну пам’ять комп’ютера, на якому встановлене серверне програмне забезпечення, а при завершенні – видаляється з пам’яті. При збільшенні кількості користувачів це негативно позначається на масштабованості. У даному випадку під масштабованістю розуміється можливість плавного зростання часу затримки відповіді програмної системи на запит у процесі різкого зростання кількості одночасно працюючих користувачів.

Для вирішення цієї проблеми Microsoft була запропонована альтернатива — ISAPI (Internet Server Application Programming Interface). Замість монолітних виконавчих файлів використовуються DLL-бібліотеки. Код DLL знаходиться в пам’яті весь час і для кожного запиту замість процесів створює нитки виконання. Усі нитки використовують єдиний програмний код, який виконується в єдиному процесі Інтернет-сервера. Це дозволяє підвищити продуктивність і масштабованість.

Для реалізації динамічних web-сторінок також застосовуються технології скриптових мов, що виконуються на стороні сервера. Найвідомішими з них є: ASP (Active Server Pages) – активні серверні сторінки та PHP (Personal Home Pages) – персональні домашні сторінки.

Принцип їх роботи розглянемо на технології ASP, що була розроблена фірмою Microsoft у 90-х роках XX століття.

Виконання скриптового коду файлу ASP підтримується ISAPI-розширенням Інтернет-сервера. У конфігурації Інтернет-сервера визначаються способи обробки файлів з різними розширеннями. Для обробки файлу з розширенням ASP в установках Інтернет-сервера визначається файл Asp.dll. Файли ASP відправляються до нього на обробку: на вхід Asp.dll поступає потік коду ASP, а на виході генерується потік HTML-коду.

Цей процес реалізується за наступною схемою. Код HTML тегів ASP-файлу обробник Asp.dll поміщає у вихідний потік без змін. Код, який вміщений у тег <%...%>, сигналізує Asp.dll, що він повинен підлягати обробці. Виконується скрипт мовою, яка вказана у відповідній директиві ASP-файлу – Language. Здебільшого це – JavaScript і VBScript. Результат обробки відповідного до мови інтерпретатора – код HTML додається до вихідного потоку Інтернет-сервера і стає частиною HTML-сторінки, що відправляється на браузер користувача.

Від початку технологія ASP була обмежена в своїх можливостях, оскільки базувалася на використанні скриптових мов, які поступаються за функціональністю мовам програмування. Крім того, код ASP був вбудований у HTML у вигляді спеціальних тегів, що створювало плутанину, бо HTML-код, зазвичай, створюють дизайнери, які відповідають за оформлення сторінки, а ASP – програмісти, які реалізовують її функціональність.

У ході еволюції технології ASP ці недоліки були усунені. У 2000 році на конференції розробників як складову нової технології .NET Microsoft представив ASP+. З виходом першої версії каркасу Framework.NET вона увійшла до його складу у вигляді структурного компонента і отримала назву ASP.NET.

Вважається, що технологію ASP.NET не можна розглядати як продовження скриптової технології ASP. Розробники позиціонують її як концептуально нову технологію, що створена в рамках ідеології відкритого програмування Microsoft – .NET. У ASP.NET закладено все для того, щоб зробити весь цикл розробки Web-додатка більш швидким, а підтримку його функціонування – простішою. ASP.NET заснована на об’єктно-орієнтованій технології. Проте вона зберегла модель використання ASP: готову програму достатньо розмістити в директорії, яка прописана у Web-сервері, і вона працюватиме.

У ASP.NET з’явилося багато нових функцій, а ті, що були раніше в ASP, значно вдосконалені. Зокрема, в ASP.NET використовуються компільовані мови. Під час компіляції перевіряється синтаксична ко-

ректність початкового тексту. Скомпільований у проміжну мову код виконується швидше за некомпільований (скриптовий), незалежно від мови, яка була використана. Крім того, компільовані мови підтримують сувору типізацію.

Компіляція відбувається на сервері в момент першого звернення користувача до сторінки. Якщо програміст змінив текст сторінки, програма перекомпілюється автоматично. При написанні коду можна використовувати набір компонентів, що поставляються разом з Framework.NET.

Платформа Framework.NET надає додаткам середовище виконання і у той же час сама безпосередньо взаємодіє з операційною системою. Середовище виконання реалізує інтерфейс ASP.NET-додатків, на якому, у свою чергу, базуються Web-форми – ASP.NET-сторінки. Інтерфейс Framework.NET дозволяє стандартизувати звернення до програмної системи і надає середовище для швидшої та зручнішої її розробки. CLR забезпечує єдиний набір сервісів для всіх мов.

Завдяки наведеним якостям ASP .NET сьогодні розглядається не як мова програмування, а як технологія, що дозволяє програмувати різними мовами, компілятори яких підтримує каркас Framework.NET. Тому для її освоєння необов'язкове попереднє знання якоїсь певної мови програмування, так само як і знання ASP. Проте, оскільки мова C# була спеціально створена для платформи .NET, її використання дозволить у більш повній мірі застосувати її концепції та методи.

Для доступу до баз даних ASP.NET використовує технологію, яка забезпечує стандартний інтерфейс взаємодії. У ній реалізовані можливості автоматичного кешування: дані, одержані з бази даних, зберігаються на сервері додатків, і він не звертається до бази для обробки повторного запиту. При зміні бази даних кеш оновлює свій вміст. Сутність цієї технології та інших подібного призначення буде розглянута в наступному підрозділі.

2.4. Технологія взаємодії програмних додатків з базами даних

Для звернення до сервера реляційних баз даних мову SQL можна використовувати як в інтерактивному режимі, так і шляхом розміщення її операторів у код програмних додатків. Застосування мови SQL у додатках на практиці реалізується двома різними способами:

1. Розміщення SQL-операторів у програмному коді. Окремі SQL-оператори розміщуються в початковому тексті програми і змішуються з операторами базової мови. Цей підхід дозволяє створювати програми, що звертаються безпосередньо до бази даних. Спеціальні програ-

ми-передкомпілятори перетворять початковий текст з метою заміни SQL-операторів відповідними викликами підпрограм СУБД, потім він компілюється і збирається звичайним способом.

2. Використання прикладного інтерфейсу програмування – API (Application Programming Interface). Альтернативний варіант полягає в наданні програмісту стандартного набору функцій, до яких можна звертатися із створюваних ним програм. Функції API можуть надавати той самий набір функціональних можливостей, що й при застосуванні вбудованих операторів, але при цьому усувається необхідність передкомпіляції початкового тексту. Крім того, деякі розробники вважають, що в цьому випадку використовується більш зрозумілий інтерфейс і текст програмного коду, зручніший для супроводження.

Обидва способи припускають використання операторів як статичного SQL, так і динамічного SQL. Оператори статичного SQL характеризуються тим, що жодних змін, після їх одноразового написання, не передбачається. Динамічний SQL дає можливість програмісту запрограмувати створення запитів під час виконання програми і передавати їх СУБД. Цей спосіб часто використовується в додатках, призначених для побудови незапланованих запитів, дозволяючи оперативно формувати оператори SQL залежно від особливих вимог, що виникають у певній ситуації.

Після формування оператора SQL відповідно до потреб користувача він передається серверу баз даних для перевірки на наявність синтаксичних помилок і необхідних для його виконання дозволів, після чого відбувається його компіляція та виконання. Прикладний інтерфейс програмування (API) для виконання операторів SQL застосовує набір класів, екземпляри яких надають програмісту різноманітні можливості доступу до баз даних, а саме: підключення, виконання запитів, вибірка окремих кортежів із результуючих наборів даних тощо.

Сучасні технології створення програмних систем клієнт-серверної архітектури для взаємодії рівня даних з рівнем обчислень та рівнем представлень їх користувачам використовують технологію ADO (Active Data Objects). Технологія ADO розроблена на основі об’єктно-орієнтованої ідеології створення інформаційних систем. Тобто ADO є набором класів, організованих у спеціалізовані бібліотеки, завдяки яким забезпечується інтерфейс для доступу до баз даних. Об’єкти – екземпляри класів ADO, здійснюють з’єднання з джерелом даних, виконання запитів та обробки їх результатів.

В рамках технології ADO реалізовані наступні основні функції:

- створення незалежних об’єктів для доступу до даних;
- підтримка збережених процедур з вхідними та вихідними параметрами;

- робота з курсорами різних типів;
- пакетне виконання операторів;
- підтримка обмежень для параметрів запиту (наприклад, кількості кортежів у результуючому наборі);
- підтримка декількох наборів даних, що повертаються збереженими процедурами, або пакетними операторами.

Інтерфейс доступу до даних ADO розроблений для ширшого застосування більшості особливостей інтерфейсу OLE DB (Object Linking and Embedding for DataBases). В інтерфейс OLE DB включений механізм провайдерів, які виконують роль постачальників даних. Їх називають сервіс-провайдерами, вони допомагають об'єднувати в однотипну сукупність об'єкти, що пов'язані з різноманітними джерелами даних. На відміну від технології OLE DB, яка є більш універсальною для доступу до різних джерел даних, технологія ADO здійснює, як правило, доступ до реляційних БД, розширюючи саме ці можливості.

Основними перевагами технології ADO є простота використання, висока швидкість, невеликі потреби в оперативній пам'яті та незначні витрати зовнішньої пам'яті. Об'єктна модель ADO визначає колекцію програмованих об'єктів, які можуть застосовуватися практично в усіх сучасних мовах програмування. Крім того, посилена модель безпеки ADO дозволяє підвищити рівень захисту програмної системи від несанкціонованого доступу.

З'єднання програмного додатка з джерелом даних в ADO відбувається за допомогою різних провайдерів, наприклад, провайдера MS SQL або Oracle. Всі провайдери даних містять класи з'єднань, адаптерів та команд. У процесі програмування створюються власні класи, які успадковують можливості класів ADO і застосовують їх для вирішення певних задач додатка.

Схема типової програми, що використовує ADO, наступна:

1. Спочатку створюється з'єднання з базою даних за допомогою класу Connection, який забезпечується необхідною інформацією – рядком з'єднання.
2. Для виконання команди в потрібній СУБД застосовуються можливості класу Command і, за його підтримки, задається необхідна команда. Ця команда може бути запитом SQL або процедурою.
3. Якщо команда не повертає даних, вона просто виконується за допомогою одного з методів класу Command – Execute. Наприклад, це може бути видалення, додавання або зміна даних таблиці.
4. Якщо команда повертає вибірку даних, організовується передача даних у програму для подальшого їх використання без підтримки

постійного зв'язку з базою даних. Для цього потрібно створити спадкоємця класу DataAdapter і, з його допомогою, зберегти дані в екземпляр спадкоємця класу DataSet.

5. Клас – спадкоємець класу DataSet – використовується для створення джерела даних, над якими проводяться необхідні обчислення, і (або) які виводяться на екранний інтерфейс користувача.

Модель об'єктів ADO визначається набором класів. Ці класи інкапсулюють у собі можливості проведення операцій з базою даних, які успадковують при створенні їх екземпляри – об'єкти.

Найчастіше виконувана операція з базою даних полягає у відображенні даних, що зберігаються в її таблицях. За відображення даних відповідає об'єкт класу-спадкоємця DataSet, в який завантажуються дані з представлень або базових таблиць і потім передаються в екранний інтерфейс користувальницької програми без підтримки безперервного зв'язку з базою даних. У сучасних версіях ADO передбачений також зворотний процес передачі даних з DataSet у базові таблиці.

Центральне місце у складі DataSet займає властивість, яка являє собою колекцією об'єктів типу DataTable. Об'єкти DataTable використовуються для представлення таблиць у DataSet. Кожен об'єкт DataTable представляє одну таблицю або представлення з бази даних.

У свою чергу, DataTable складається з об'єктів DataColumn і DataRow. DataColumn – це блок для створення схеми DataTable. Кожен об'єкт DataColumn має властивість DataType, яка визначає тип даних, що містяться в кожному об'єкті DataColumn. Об'єкти DataRow складають колекцію, яка відповідає набору рядків (записів) заданої таблиці. Ця колекція використовується для вивчення результатів запиту до бази даних. Ми можемо звертатися до записів таблиці як до елементів простого масиву.

Оскільки екземпляр DataSet являє собою спеціалізований об'єкт, що містить образ бази даних, він повинен містити механізм взаємодії з джерелом даних. Такий механізм забезпечує об'єкт типу DataAdapter. Сама назва цього об'єкта – адаптер, тобто перетворювач, указує на його природу. DataAdapter містить метод Fill для отримання даних з бази і заповнення DataSet. Наприклад, використовуючи мову C#, можна створити об'єкт DataSet:

DataSet ds = new DataSet();

Та передати йому дані з представлення "Список клієнтів":

dataAdapter.Fill(ds, " Список клієнтів ");

Коли в об'єкті DataAdapter викликається метод Fill, то він сам перевіряє, чи відкрите з'єднання. Якщо з'єднання немає, DataAdapter

здатний сам його відкрити, виконати задачу і потім закрити. Проте явне управління з'єднанням є кращим підходом до роботи з базами даних, оскільки має наступні переваги:

- дає більш зручний для прочитання код;
- допомагає при налагодженні додатка;
- більш ефективне.

Для явного управління з'єднанням використовується об'єкт Connection. У його арсеналі є властивість ConnectionString символьного типу, задавши яку, можна встановити зв'язок з джерелом даних. Цей зв'язок може бути одночасно використаний декількома командними об'єктами.

Командні об'єкти є спадкоємцями класу Command. Одна з властивостей має тип Connection, якому і присвоюються, за необхідності, параметри встановленого раніше з'єднання. Наприклад:

```
String connStr = " Data Source=S4\BANKING; Initial Catalog=MyBank; Integrated Security=True ";

SqlConnection conn = new SqlConnection();

conn.ConnectionString = connStr;
```

Об'єкти Command застосовуються для виконання SQL-запитів до джерела даних. Як правило, вони слугують для виклику збережених процедур на додавання, модифікацію або видалення даних. До складу класу Command входить властивість CommandText символьного типу, в яку можна поміщати SQL-запит до джерела даних. Наприклад:

```
SqlCommand myCommand = new SqlCommand();

string comText = "EXEC УдалітьКлієнта @КлиентИдн=123";

myCommand.CommandText = comText;

myCommand.Connection = conn;

myCommand.ExecuteScalar();
```

Об'єкт Command, як і об'єкт Connection, іноді створюється неявно в момент створення об'єкта DataSet. Але бажано використовувати їх, створюючи явним чином, оскільки ефективна нормалізація структури реляційних даних припускає їх відображення за допомогою представлень, а модифікацію – за допомогою збережених процедур.

Очевидно, що об'єкти ADO забезпечують доступ до сервера баз даних, характеристики якого визначаються в програмному коді додатків. Для того, щоб не розробляти окремі версії додатка для кожного з екземплярів сервера, з яким даний додаток планується використовува-

ти, Microsoft розробив стандарт, що одержав назву Open Database Connectivity (ODBC).

Технологія ODBC передбачає застосування єдиного інтерфейсу для доступу до різних серверів реляційних баз даних. Цей інтерфейс забезпечує високий ступінь універсальності, у результаті якої програмний додаток може діставати доступ до даних, що зберігаються в базах під управлінням різних СУБД, без необхідності внесення змін у його програмний текст. Таким чином, розробники одержали інструмент для створення та розповсюдження клієнт-серверних систем, здатних працювати з широким спектром СУБД. Організовувати взаємодію додатка з цільовою СУБД можна за допомогою відповідного ODBC-драйвера.

На даний час технологія ODBC фактично отримала статус галузевого стандарту. Основною причиною її популярності є гнучкість, що надає розробникам такі переваги:

- додатки більше не пов'язані з прикладним API однієї СУБД;
- додаток здатний ігнорувати особливості використовуваних протоколів передачі даних;
- дані посилаються і доставляються в тому форматі, який є найбільш зручним для додатка;
- у даний час існують драйвери ODBC для різних типів найпоширеніших СУБД.

В інтерфейс ODBC включені наступні елементи:

- бібліотека функцій, виклик яких дозволяє додатку підключатися до бази даних, виконувати SQL-оператори та отримувати інформацію з результуючих наборів даних;
- стандартний метод підключення та реєстрації в СУБД;
- стандартне представлення для даних різних типів;
- стандартний набір кодів помилок.

Загальна архітектура програмної системи, яка використовує ODBC, включає чотири елементи:

1. Додаток. Цей компонент виконує обробку даних і виклик функцій бібліотеки ODBC для відправки SQL-операторів до СУБД та вибірки інформації, що повертається із СУБД.
2. Менеджер драйверів. Він виконує завантаження драйверів на вимогу додатка.
3. Драйвери та агенти баз даних. Ці компоненти обробляють, спрямовують SQL-запити до конкретних джерел даних та повертають одержані результати додатку. При необхідності драйвери виконують модифікацію запиту додатка з метою приведення його у відповідність синтаксичним вимогам цільової СУБД.

4. Джерела даних. При використанні в додатку засобів ODBC спочатку здійснюється звернення до певного джерела даних, а через нього – до СУБД, що представляється ним. Крім того, встановлюється загальна підсистема ODBC і визначаються пари "драйвер – база даних", яким задаються імена для встановлення з'єднання з базою даних. Відповідні пари називаються іменами джерел даних, або іменованими джерелами даних (Data Source Names, DSN). Імена DSN дозволяють додаткам звертатися до джерел даних.

Виходячи з вищесказаного, можна відзначити, що технологія ODBC пропонує єдиний інтерфейс доступу до різноманітних типів реляційних баз даних. Мова SQL використовується в ній як основний стандарт доступу до даних. Інтерфейс ODBC забезпечує високий ступінь універсальності: один додаток може звертатися до різних SQL-сумісних СУБД за допомогою загального коду. Це дозволяє розробнику створювати й поширювати додатки без урахування особливостей конкретної СУБД. Зазначені функціональні можливості технології ODBC дозволяють розробляти додатки для взаємодії з СУБД різних виробників. Для зв'язку додатків з різнотипними СУБД використовуються відповідні ODBC-драйвери.

Технологія ODBC передбачає створення додаткового рівня між програмним додатком і СУБД, з якою він взаємодіє. Служби ODBC забезпечують отримання від додатка запитів на вибірку інформації та переклад їх мовою ядра бази даних, для доступу до інформації, що зберігається в ній. Таким чином, завдяки використанню ODBC досягається абстрагування додатка від особливостей ядра серверної бази даних, тому база даних стає прозорою для клієнтських додатків.

2.5. Базові концепції мови SQL

Для того, щоб адмініструвати сервер реляційних баз даних, створювати його об'єкти й керувати ними, а також вносити, отримувати, модифікувати та видаляти дані з таблиць реляційних баз даних, використовується мова структурованих запитів – SQL. Це декларативна мова, що забезпечує програмування на основі індукції. За допомогою SQL здійснюється опис даних, які необхідно обробити за допомогою оператора. Потім цей оператор транслюється системним програмним забезпеченням у послідовність процедурних операцій, що забезпечує його виконання на комп'ютерах фон-нейманівської архітектури (практично будь-яких).

Для створення й управління реляційними базами даних на сьогоднішній день розроблено чимало програмних комплексів (ORACLE,

MS SQL SERVER, INFORMIX і т.д.), які використовують різні діалекти SQL. Зокрема, у MS SQL SERVER використовується мова Transact-SQL, яка є розширенням стандарту мови SQL, що визначений в ISO (International Organization of Standardization) та ANSI (American National Standards Institute). В цій книзі вона буде використана як основа для вивчення концепцій мови структурованих запитів.

SQL займає центральне місце у проектуванні та використанні реляційних баз даних. Будь-який програмний додаток, що взаємодіє з реляційною базою даних, незалежно від його користувальницького інтерфейсу, посилає серверу баз даних оператори SQL.

Оператор SQL складається з набору команд, що виконують певні дії над об'єктами бази даних або даними, що зберігаються в ній. Реляційні бази даних підтримують три типи операторів SQL: мову визначення даних (Data Definition Language, DDL), мову маніпулювання даними (Data Manipulation Language, DML) і мову управління даними (Data Control Language, DCL).

За допомогою операторів мови визначення даних визначається існування всіх об'єктів сервера баз даних, наприклад баз даних, базових таблиць, представлень тощо. Для кожного класу об'єктів підтримуються оператори CREATE, ALTER і DROP. Більшість операторів DDL виглядають так:

CREATE тип_об'єкта назва_об'єкта

ALTER тип_об'єкта назва_об'єкта

DROP тип_об'єкта назва_об'єкта

Оператор CREATE створює об'єкт, оператор ALTER дозволяє модифікувати об'єкт, оператор DROP видаляє об'єкт.

Мова маніпулювання даними використовується для отримання, занесення, редагування та видалення даних, що містяться в об'єктах, визначених за допомогою DDL. Як основні оператори маніпулювання даними використовуються оператори: SELECT, INSERT, UPDATE і DELETE.

Оператор SELECT здійснює вибірку інформації, що зберігається в базі даних і дозволяє вибрати один або декілька кортежів з однієї або декількох таблиць. Оператор INSERT додає в таблицю новий кортеж, оператор UPDATE служить для редагування даних, оператор DELETE видаляє кортежі з таблиці.

Мова управління даними застосовується для управління правами доступу до об'єктів бази даних. Управління правами доступу здійснюється за допомогою операторів GRANT і REVOKE. Оператор GRANT створює в системі безпеки запис, який дає можливість користувачам

бази даних працювати з інформацією, що зберігається в базі, або виконувати певні оператори SQL. Оператор REVOKE анулює раніше наданий користувачу дозвіл для роботи з базою даних.

У SQL входить багато синтаксичних елементів, які використовуються більшістю операторів, або, точніше, які впливають на хід виконання цих операторів. До таких елементів належать: ідентифікатори, змінні, функції, типи даних, вирази, мова управління ходом виконання, коментарі. Коротко розглянемо їх призначення.

Ідентифікатори. Імена об'єктів бази даних задаються ідентифікаторами. Будь-який об'єкт бази даних (таблиця, представлення, атрибут, функція, збережена процедура і т.д.) повинен мати ідентифікатор. Ідентифікатор присвоюється об'єкту при його створенні. Після цього на об'єкт посилаються за допомогою ідентифікатора. Наприклад, наступний оператор створює таблицю з ідентифікатором Клієнт:

```
CREATE TABLE Клієнт
(
КлієнтІдн INT PRIMARY KEY,
Прізвище VARCHAR(2O)

)
```

Існують два класи ідентифікаторів: звичайні та з обмежувачами. У звичайних ідентифікаторах обмежувачі не застосовуються. Ідентифікатори з обмежувачами виділяють подвійними лапками (" ") або квадратними дужками ([ ]). В операторах SQL ідентифікатори, що неповністю відповідають правилам форматування, необхідно розміщувати між обмежувачами. Наприклад, треба використовувати ідентифікатор з обмежувачами, якщо він містить пробіл або співпадає із зарезервованим – ключовим словом мови SQL.

Змінні. Змінна мови SQL є об'єктом, що використовується в наборах програмного коду, який вміщує перелік операторів для реалізації певного завдання. Імена змінних у SQL починаються з латинської літери ета – @. Після того, як змінна визначена (оголошена), один оператор SQL може присвоїти їй значення, а інший – використати це значення. Наприклад:

```
DECLARE @Идн INT
SET @Идн = 3
SELECT * FROM Клієнт
WHERE КлієнтІдн = @Идн + 1
```

Як правило, змінні в програмних наборах мови SQL (пакетах, збережених процедурах, тригерах і т.д.) виконують наступні завдання:

- слугують як лічильники для циклів;

- зберігають певне значення для подальшого аналізу операторами, що керують ходом виконання програми;
- використовуються для збереження значення, що повертає збережена процедура.

Змінні можна застосовувати тільки у виразах, вони не повинні співпадати з іменами об'єктів та ключовими словами.

Функції. Функція інкапсулює логіку програми, виконувану досить часто в підпрограму, яка складається з одного або декількох операторів SQL. Будь-яка програма, якій потрібно виконати алгоритм, реалізований функцією, може не повторювати весь алгоритм, а викликати цю функцію. SQL підтримує функції двох типів: вбудовані та користувальницькі.

Вбудовані функції – це функції, робота яких визначена в SQL і програмну логіку яких не можна модифікувати безпосередньо. Ці функції дозволяється викликати тільки в операторах SQL. Вони можуть бути трьох типів: функції отримання набору записів, агрегатні і скалярні функції.

Функції отримання набору записів повертають об'єкт, який дозволяється використовувати в операторі SQL замість посилання на таблицю. Агрегатні функції обробляють набори значень, але повертають єдине результуюче значення. Скалярні функції обробляють одиночне значення і повертають також одне значення.

Користувальницькі функції – це функції, програмна логіка яких створюється користувачем. Оператор CREATE FUNCTION дозволяє створювати користувальницькі функції, оператор ALTER FUNCTION – модифікувати їх, а оператор DROP FUNCTION – видаляти. Будь-яке повне ім'я користувальницької функції має бути унікальним.

SQL підтримує два типи користувальницьких функцій: скалярні і табличні. Скалярні функції повертають єдине значення, тип якого визначається конструкцією RETURN. Табличні функції повертають таблиці, які утворюються завдяки формування результуючого набору оператора SELECT.

У SQL розрізняють також детерміновані і недетерміновані функції. Детермінованою вважається функція, яка повертає однакові результати, якщо її викликають з одним і тим самим набором вхідних параметрів. Недетермінованою називається функція, яка при виклику з одним і тим самим набором вхідних параметрів може давати різні результати.

Типи даних. Тип даних – це синтаксичний елемент, що визначає характер інформації, яку може містити об'єкт бази даних. У всіх атрибутів, параметрів, змінних, функцій, що повертають значення та збе-

режених процедур з кодами повернення, є певні типи даних. У SQL є декілька базових типів даних, наприклад: int, char, і т.д. Усі дані, що зберігаються в реляційних базах даних, повинні бути сумісними з одним із цих базових типів. Типи даних детально розглядаються в наступних розділах.

Вирази. Вираз – це комбінація ідентифікаторів, значень і операцій, які СУБД може обробити, щоб одержати результат. Існує декілька різних способів використання виразів при зверненні до даних та під час їх модифікації. Наприклад, вираз може бути частиною запиту або визначати умову для пошуку даних, що відповідають певному набору критеріїв.

Оператори дозволяють виконувати різні операції: арифметичні, порівняння, конкатенації або присвоєння. Наприклад, можна перевірити дані, щоб з'ясувати, чи не порожнім є значення атрибуту, в якому містяться відомості про національну приналежність клієнтів банку.

Мова управління ходом виконання. Мова управління ходом виконання складається із спеціальних слів, які контролюють послідовність виконання операторів SQL. Ці слова, зазвичай, використовуються в програмних пакетах SQL та збережених процедурах. Якщо мова управління ходом виконання не використовується, окремі оператори SQL виконуються послідовно, у тому порядку, в якому вони розташовані.

Ключові слова мови управління ходом виконання застосовуються у випадку, якщо необхідно змусити програмний набір операторів SQL зробити певну дію. Наприклад, пара операторів IF…ELSE дозволяє виконати деякий блок операторів, якщо виконується певна умова, а якщо ця умова не виконується – виконати інший блок операторів.

Ключові слова мови управління ходом виконання операторів SQL наступні:

- BEGIN...END – містять у собі набір операторів SQL, дозволяючи виконувати оператори групами;
- WHILE – повторно виконує оператор або блок операторів до тих пір, доки виконується задана умова;
- BREAK – вихід з циклу WHILE;
- CONTINUE – продовжує виконання циклу WHILE без завершення поточної ітерації;
- GOTO – при виконанні програмного набору операторів SQL викликає перехід до певної мітки, пропускаючи операторів, розташованих між оператором GOTO і цією міткою;
- IF…ELSE – ставить виконання оператора SQL в залежність від зазначених умов. ELSE – визначає альтернативний набір операторів, який буде виконано, у разі невиконання умови IF;

- RETURN – безумовне завершення запиту, збереженої процедури, функції;
- WAIT FOR – припиняє роботу з'єднання з сервером баз даних протягом заданого періоду або до настання певного часу.

Коментарі. Коментарі (або примітки) – це рядки програми, що не виконуються. З їх допомогою документують первинний текст програми або тимчасово відключають частину операторів SQL під час діагностики її загальної логіки. Документування первинного тексту програми коментарями полегшує її супровід. SQL підтримує два типи коментарів:

1. Подвійний дефіс (--) – цей знак коментарію можна використовувати як всередині, так і на початку програмного рядка. Усі символи від подвійного дефіса до кінця рядка розглядаються як частина коментарію.

2. Парні символи "слеш-зірочка" (/*...*/) – ці знаки коментарів можна використовувати у будь-якому місці програмного коду. Весь вміст між знаками, починаючи (/*) і закінчуючи (*/), вважається частиною коментарію.

З попереднього матеріалу неважко зрозуміти, що в SQL передбачено декілька методів виконання операторів. Можна передавати на виконання СУБД один оператор або декілька операторів у вигляді пакета. Оператори SQL також можна виконувати у вигляді збережених процедур та тригерів.

Обробка одиночних операторів є основним способом виконання операторів SQL. Щоб визначити найбільш ефективний спосіб виконання оператора, сервер баз даних повинен провести його аналіз. Цей процес називається оптимізацією оператора, і виконує його оптимізатор запитів. Оптимізацією оператора SQL називається процес вибору одного плану виконання з декількох можливих. Оптимізатор запитів є одним з найважливіших компонентів СУБД.

Пакет операторів SQL – це група з одного або декількох операторів SQL, які програмний додаток одночасно посилає серверу баз даних на виконання. Сервер баз даних компілює оператори пакета в суцільну програмну одиницю, що має назву плану виконання. Після цього здійснюється по черзі виконання операторів цього плану. Помилка при компіляції (наприклад, синтаксична) зупиняє процес компіляції плану виконання. У цьому випадку жоден з операторів пакета виконаний не буде.

Оскільки всі оператори пакета компілюються в єдиний план виконання, пакет повинен бути логічно завершеним. План виконання, створений для одного пакета, не може посилатися на змінні, оголоше-

ні в іншому пакеті. Початок і кінець коментарію повинні знаходитися в межах одного пакета.

Оператори SQL можуть також виконуватися у складі збережених процедур і тригерів. Збережена процедура – це іменована група операторів SQL, яка зберігається на сервері баз даних, компілюється один раз і, після цього може виконуватися багато разів. Така функціональність підвищує продуктивність, оскільки відпадає необхідність у перекомпіляції операторів SQL. Тригер – це збережена процедура особливого типу, яку користувач не викликає безпосередньо. При створенні тригера визначаються умови його виконання: тригер повинен бути виконаний при певній модифікації даних у визначеній таблиці.

СУБД зберігає тільки початковий текст збережених процедур. Коли збережена процедура виконується перший раз, початковий текст компілюється у план виконання. Якщо до того, як план виконання застаріє і буде видалений з пам'яті, збережена процедура виконується знову, реляційний механізм знаходить існуючий план і використовує його повторно. Якщо план застарів – він видаляється з пам'яті, а замість нього створюється новий план виконання. Збережені процедури будуть більш детально розглянуті в наступних темах.

Оператори SQL можуть також виконуватись з використанням механізму сценаріїв. Сценарій SQL – це набір операторів SQL, який збережений у спеціалізованому файлі. Цей файл можна завантажувати в клієнтські програмні утиліти, призначені для передачі операторів SQL на виконання в СУБД та отримання результатів.

Сценарії SQL складаються з одного або декількох пакетів. Для розмежування пакетів у SQL використовується оператор GO. Тобто команда GO інтерпретується як сигнал для СУБД, що визначає число операторів SQL в пакеті. Відсутність цієї команди призведе до сприйняття СУБД набору операторів, отриманих одночасно як один пакет.

Сценарії SQL дозволяють виконувати наступні задачі:

- зберігати постійну копію команд, що створюють на сервері бази даних і таблиці та заповнюють їх первинними даними;
- передавати операторів на виконання з однієї комп'ютерної платформи на іншу;
- швидко готувати нових фахівців на основі готових програмних шаблонів, формуючи у них навички пошуку помилок і внесення змін у програми.

На цьому загальний огляд базових концепцій SQL будемо вважати закінченим. Починаючи з наступного розділу, дається більш детальний опис можливостей її використання у процесі створення інформаційних систем з базою даних.

ПИТАННЯ ТА ЗАВДАННЯ

1. У чому полягає сутність ідеології відкритого програмування?
2. Які переваги досягаються завдяки використанню єдиного каркасу Framework.Net у процесі розробки програмних продуктів?
3. Які основні компоненти утворюють каркас Framework.Net?
4. Назвіть компоненти бібліотеки класів FCL, що забезпечують відкритість програмування в середовищі Visual Studio.
5. Охарактеризуйте складові, що утворюють загальномовне виконавче середовищі CLR.
6. Розкрийте сутність наступних понять сучасних Windows-додатків: рішення (solution), проект (project), простір імен (namespace), збірка (assembly).
7. Назвіть основні об’єкти та методи тривіального Windows-додатка.
8. На які функціональні групи можна поділити елементи управління Windows-додатків?
9. Що є результатом Web-програмування?
10. Назвіть відомі технологічні підходи до створення динамічних Web-сторінок.
11. Опишіть процес застосування ASP.NET для розробки Web-додатків.
12. Які існують способи застосування мови SQL у програмних додатках?
13. Що покладено в основу технології ADO?
14. Які основні функції реалізовані в рамках технології?
15. Опишіть схему типової програми, що використовує ADO.
16. Охарактеризуйте основні об’єкти ADO.
17. Наведіть приклад застосування об’єктів ADO для встановлення зв’язку додатка з базою даних та виконання SQL-запиту.
18. У чому полягає сутність технології ODBC?
19. Які переваги надає розробникам програмних додатків технологія ODBC?
20. Опишіть загальну архітектуру програмної системи, яка використовує ODBC.

ТЕСТИ

1. Яка основна причина дозволяє вважати програмні додатки, що використовують каркас Framework.Net, результатом реалізації ідеології відкритого програмування?
   а) Framework.Net є вільно поширюваним технологічним рішенням;
   б) можливість сумісного використання різних мов програмування;

в) можливість застосовувати Framework.Net на різних операційних платформах;
г) Framework.NET є платформою для створення, розгортання і запуску програмних додатків.

2. Яка з перерахованих функціональних можливостей каркаса Framework.Net реалізується керованим модулем?
   а) двохетапна компіляція;
   б) перехоплення виключень, викинутих додатком, та їх подальша обробка;
   в) звільнення пам'яті від об'єктів, які не використовуються в подальшій роботі додатка.
3. Чим є результат роботи компілятора з погляду загальномовного середовища виконання CLR каркасу Framework?
   а) збіркою;
   б) рішенням;
   в) проектом;
   г) простором імен.
4. Який з методів відкриває форму як немодальне вікно?
   а) Run;
   б) Open;
   в) Show;
   г) ShowDialog.
5. Що є результатом Web-програмування?
   а) програмне забезпечення, що реалізовує розмітку гіпертекстових документів;
   б) програмне забезпечення, що реалізовує перегляд вмісту документів формату HTML;
   в) програмне забезпечення, що реалізовує обробку вмісту web-сторінок перед відправкою їх клієнтам.
6. На якому компоненті інформаційної системи архітектури "клієнт-сервер" можуть виконуватися скрипти?
   а) серверному;
   б) клієнтському;
   в) клієнтському та серверному.
7. Яке з наведених тверджень щодо технології ASP.NET є правильним?
   а) в ASP.NET додатки компілюються в проміжний програмний код;
   б) в основу ASP.NET покладене застосування скриптових мов;
   в) в ASP.NET додатки компілюються в програмний код процесора серверної платформи.

8. Яка основна відмінність застосування в прикладних програмах функцій API для реалізації SQL-запитів від впровадження SQL-операторів у початковий текст програми?
   а) таких відмінностей немає;
   б) не відбувається передкомпіляція початкового тексту;
   в) не передбачаються зміни запитів після їх одноразового написання;
   г) існує можливість запрограмувати створення незапланованих запитів.
9. Спадкоємець якого класу ADO застосовується для представлення даних користувачу?
   а) DataSet;
   б) DataTable;
   в) DataAdapter.
10. Фрагмент програмного коду, написаного мовою C#, виглядає наступним чином:

```
SqlCommand myCommand = new SqlCommand();
string comText = "...";
myCommand.CommandText = comText;
```

11. Що необхідно вставити замість крапок (…), щоб виконання коду могло досягти певної мети?
   а) коментар;
   б) SQL-запит;
   в) рядок з'єднання;
   г) найменування таблиці бази даних.

РОЗДІЛ 3. МОВА ВИЗНАЧЕННЯ ДАНИХ В SQL

3.1. Створення баз даних

Першим етапом практичної реалізації бази даних є створення об'єкта "база даних". Створюють цей об'єкт і визначають його характеристики на основі інформації, зібраної при визначенні вимог до інформаційної системи, яка розроблятиметься на її основі, та деталей, визначених при концептуальному проектуванні структури даних. Характеристики об'єкта "база даних" дозволяється змінювати після його створення.

Під час створення бази даних необхідно визначити її ім'я, розмір, а також файли і групи файлів, в яких вона зберігатиметься. Але перш ніж створювати базу даних, слід засвоїти декілька правил:

- право на створення бази даних за замовчуванням належить членам стандартних ролей на сервері (наприклад, в MS SQL це: sysadmin і dbcreator), проте це право може бути надане й іншим користувачам;
- користувач, що створив базу даних, стає її власником;
- на сервері може бути створена обмежена кількість баз даних, що залежить від версії СУБД;
- ім'я бази даних повинно відповідати правилам, визначеним для ідентифікаторів.

Як вже було зазначено, для збереження бази даних використовуються три типи файлів: основні, в яких знаходиться інформація для запуску; додаткові, в яких зберігаються всі дані, що не помістилися в основному файлі; файли журналу трансакцій, що містять дані журналу, необхідні для відновлення бази даних. Будь-яка база даних складається, принаймні, з двох файлів: основного і файлу журналу трансакцій. При створенні бази даних слід задати її максимальний розмір. Це дозволяє запобігти зростанню файлу при додаванні даних аж до вичерпання вільного місця на диску. Під час створення бази даних ті файли, що її визначають, заповнюються нульовими значеннями, щоб знищити всі дані, які могли залишитися на диску після файлів, видалених раніше.

Для того, щоб створити базу даних з усіма її файлами, використовують оператор CREATE DATABASE. Цей оператор дозволяє зада-

ти потрібну кількість параметрів, що визначають характеристики бази даних. Наприклад:

```
CREATE DATABASE MyBank
ON
(
NAME = MyBank_dat,
FILENAME='c:\program files\microsoft SQL server\mssql\data\bank.mdf ',
SIZE = 4,
MAXSIZE =10,
FILEGROWTH = 1
)
```

Цей сценарій створює базу даних на екземплярі сервера баз даних під назвою MyBank і задає для неї один основний файл. Крім того, автоматично створюється файл журналу транзакцій розміром 1 Мб. Оскільки в параметрі SIZE для основного файлу не задані одиниці виміру, розмір основного файлу вимірюється в мегабайтах. Якщо не задані параметри файлу журналу трансакцій, зокрема параметр MAXSIZE, цей файл може збільшуватися, доки не заповнить усе місце на диску.

У подальшому, коли необхідно усунути недоліки або прийняти рішення про внесення змін у базу даних, існує можливість переглянути визначення бази даних та параметрів її конфігурації. У сучасних СУБД для цієї мети передбачені системні збережені процедури і графічний інструментарій (у MS SQL це системна збережена процедура sphelpdb).

Після створення бази даних більшістю СУБД дозволяється модифікувати її початкові установки. Проте іноді перед внесенням змін необхідно вивести базу даних із звичного режиму роботи. Для модифікації визначених структур бази даних застосовується оператор ALTER DATABASE, за допомогою якого можна здійснити:

- зміну параметрів бази даних;
- зміну групи файлів за замовчуванням;
- зміну обмеження максимального розміру бази даних;
- зміну фізичного місця розташування бази даних;
- додавання файлу даних або файлу журналу;
- видалення файлу даних або файлу журналу;
- додавання групи файлів.

У випадку, якщо несистемна база даних більше не потрібна (або вона переміщена на інший сервер або в іншу базу даних), її можна видалити. Після видалення бази даних усі її файли та дані, що вони вміщу-

вали, фізично видаляються з диска на сервері. База даних видаляється назавжди і не може бути відновлена іншим способом ніж копіюванням із резервної копії. Видалити системні бази даних користувачам СУБД забороняє. Базу даних видаляють за допомогою оператора DROP DATABASE. Наприклад:

DROP DATABASE MyBank

3.2. Створення базових таблиць

Після створення бази даних в ній необхідно побудувати таблиці для зберігання даних. Для цього на початку необхідно визначити типи даних, які будуть задані для кожного атрибуту кожної таблиці. Нагадуємо, що тип даних визначає характер інформації, яку може містити той чи інший об'єкт бази даних. Тип даних є у наступних об'єктів:

- атрибутів таблиць та представлень;
- вхідних і вихідних параметрів збережених процедур;
- змінних;
- функцій SQL, які повертають одне або декілька значень.

Отже, один з перших етапів конструювання таблиці полягає в призначенні кожному її атрибуту типу даних. Сучасні СУБД підтримують набір системних типів даних, а також користувальницьких типів, які засновані на системних типах даних.

Системні типи даних визначають усі типи даних, що дозволені до використання в СУБД. Ці типи дозволяють забезпечувати цілісність даних, оскільки дані, що заносяться в атрибути кортежу таблиці, повинні відповідати типам, заданим в операторі CREATE TABLE при її створенні. Наприклад, не можна зберігати прізвище в атрибуті, для якого визначений тип даних datetime, оскільки в такий атрибут можна заносити тільки дати.

При призначенні об'єкта типу даних визначаються чотири властивості цього об'єкта:

1. Вид даних, які можуть міститися в об'єкті. Наприклад: символьні, цілі або двійкові числа.
2. Довжина або розмір значення, що зберігається. Довжина числового типу даних – це кількість байт, яка необхідна для зберігання кількості знаків, дозволеної для нього. Довжина символьних типів даних визначається в символах.
3. Точність числа (тільки для числових типів даних) – кількість знаків, якими представлене число. Наприклад, об'єкт типу smallint може зберігати не більше п'яти знаків, тому його точність дорівнює п'яти.

4. Точність дробової частини числа (тільки для числових типів даних) – кількість десяткових знаків після коми. Наприклад: у об'єкта типу int немає дробової частини, тому точність його дробової частини дорівнює нулю; у об'єкта типу money може бути до чотирьох знаків після коми, тому точність його дробової частини дорівнює чотирьом.

Розглянемо загальні категорії базових типів даних, які підтримуються сучасними СУБД:

- двійкові дані, що зберігають послідовності біт. Вони складаються з шістнадцятирічних чисел: binary, varbinary, image;
- символьні дані, що складаються з комбінацій букв, символів і цифр: char, varchar, text;
- дані, що складаються з допустимих комбінацій дати та часу: datetime;
- десяткові дані, які зберігають чисельні дані до найменшого значущого розряду. Цей тип даних зберігає точне представлення чисел: decimal, numeric;
- дані з плаваючою комою. Цей тип даних зберігає наближені числові дані, точність визначається можливостями двійкової системи вирахування: float;
- цілочисельні дані, що складаються з позитивних та негативних цілих чисел: int;
- грошові. Цей тип даних зберігає фінансові дані, що представляють позитивні та негативні грошові суми: money;
- спеціальні дані, що не потрапляють в жодну іншу категорію даних: bit, cursor;
- Unicode. За допомогою типів даних Unicode можна зберігати символи, визначені стандартом Unicode, куди входять усі символи, з різноманітних мовних наборів. Для зберігання типів даних Unicode потрібно в два рази більше місця порівнянно з символьними типами.

Усі дані, які зберігаються в реляційних базах, повинні бути сумісні з одним із базових типів. Тобто в основі користувальницьких типів даних лежать системні типи даних. Користувальницькі типи даних застосовуються, коли в наборі атрибутів з декількох таблиць повинні зберігатися однотипні дані, причому останнім треба гарантовано мати однаковий тип, розмір та за одним і тим самим правилом допускати або не допускати порожні значення. Наприклад, на основі типу char можна створити користувальницький тип даних під назвою "ПоштовийІндекс".

Для створення користувальницького типу даних необхідно надати наступну інформацію:

- ім’я;
- системний тип даних, що лежить в основі нового типу даних;
- можливість введення порожніх значень.

Після визначення типів даних новостворена база даних готова до створення таблиць, в яких зберігатимуться дані. У визначення таблиці повинно входити як мінімум ім’я таблиці, імена атрибутів, типи даних (з установкою розмірів, якщо їх значення за замовчуванням не підходять), а також установка допустимості в атрибуті порожніх значень. Решту властивостей можна налаштувати пізніше, хоча чим більше властивостей встановлено при створенні таблиць, тим ефективніше стає загальний процес проектування бази даних. Тому на початковому етапі розробки бази даних потрібно мати максимум відомостей, необхідних для створення базових таблиць.

В ідеалі усі необхідні параметри кожної таблиці визначаються одночасно. Виходячи із задач цього учбового курсу, ми спочатку розглянемо процес створення простих таблиць, у визначення яких входять: ім’я, атрибути, типи даних, можливість внесення порожніх значень і початкові значення атрибутів (де необхідно). В міру викладення матеріалу складність базових таблиць буде зростати з метою демонстрації можливостей реляційних баз даних. З прикладом практичного застосування сценарію SQL для створення бази даних реальної автоматизованої банківської системи можна ознайомитись у додатку А.

Для створення таблиць використовується оператор CREATE TABLE. При використанні цього оператора треба визначити, як мінімум, ім’я таблиці, атрибути і їхні типи даних. Під час визначення кожного атрибуту таблиці встановлюється можливість введення порожніх значень. Через складнощі обробки порожніх значень у визначенні всіх атрибутів бажано використовувати ключові слова NULL – якщо атрибут допускає порожні значення, або NOT NULL – якщо ні. У разі невстановлення цих значень буде використане значення за замовчуванням, хоча такий стиль програмування не є найкращим.

Наступний приклад ілюструє створення простої таблиці для збереження інформації про клієнтів у базі даних банківської установи:

```
CREATE TABLE Клієнт
(
КлієнтІдн int NOT NULL,
Прізвище varchar(10) NOT NULL,
[Ім’я] varchar(10) NOT NULL,
[По батькові] varchar(10) NULL
)
```

Слід звернути увагу на використання квадратних дужок в ідентифікаторах, що не відповідають правилам форматування. У нашому випадку – наявність апострофа та пробілу.

З матеріалу попереднього розділу зрозуміло, що атрибути, які допускають порожні значення, небажані, тому замість них краще визначати значення за замовчуванням. Наприклад, значенням за замовчуванням, яке використовується, якщо жодне значення не вказане, для числових полів, зазвичай є нуль, а для символьних — "не визначено". Для його застосування в операторі CREATE TABLE використовується ключове слово DEFAULT:

```
CREATE TABLE Клієнт
(
КлієнтІдн int NOT NULL,
Прізвище varchar(10) NOT NULL,
[Ім'я] varchar(10) NOT NULL,
[По батькові] varchar(10) NULL DEFAULT 'не визначено'
)
```

Існуюче в таблиці значення за замовчуванням можна в подальшому модифікувати або видалити.

У реляційній таблиці існує можливість визначити один атрибут з ідентифікатором, в якому будуть зберігатися генеровані системою послідовні значення, що унікально ідентифікують кожен кортеж таблиці. Наприклад, протягом додавання кортежів у таблицю "Клієнт" атрибут з ідентифікатором може автоматично генерувати унікальні номери для ідентифікації клієнта.

Атрибут з ідентифікатором можна реалізувати за допомогою визначення властивості IDENTITY, яка дозволяє розробнику бази даних задати номер ідентифікатора першого кортежу, доданого в таблицю, та інкремент, що додаватиметься до початкового значення ідентифікатора при додаванні в таблицю наступних кортежів. Наприклад:

```
CREATE TABLE Клієнт
(
КлієнтІдн int IDENTITY(101,1) NOT NULL,
...
)
```

Визначаючи атрибут з ідентифікатором за допомогою властивості IDENTITY, слід врахувати наступні моменти:

- у таблиці дозволений тільки один атрибут з властивістю IDENTITY;
- можливі типи даних для атрибуту з ідентифікатором: decimal, int, numeric, smallint, bigint або tinyint;

- можна задати початкове значення та інкремент ідентифікатора, для обох значення за замовчуванням дорівнює 1;
- атрибут з ідентифікатором не допускає порожніх значень, в ньому також заборонені визначення значення за замовчуванням DEFAULT.

Після створення таблиць в базі даних існує можливість переглянути її властивості (наприклад, ім'я або тип даних певного атрибуту). Крім того, можна переглянути залежності таблиці, щоб визначити залежні від таблиці об'єкти (наприклад, представлення, збережені процедури, тригери). Зміни, що вносяться в таблицю, впливають на залежні від неї об'єкти.

Більшість з параметрів таблиці, визначених під час її створення, дозволяється змінювати. Сучасні СУБД надають можливості: додавати, модифікувати і видаляти атрибути, зокрема, змінювати ім'я, тип даних, розмір, точність дробової частини, можливість введення порожніх значень і, також, додавати та видаляти обмеження. Для цього використовується оператор ALTER TABLE. Так, за його допомогою можна додати атрибут у таблицю:

ALTER TABLE Клієнт
ADD (Статус varchar(20) NULL)
Можна змінити визначення атрибуту:
ALTER TABLE Клієнт
MODIFY (Прізвще varchar (15))

Іноді виникають ситуації, коли необхідно видалити таблицю. Наприклад, якщо потрібно реалізувати нову структуру або звільнити місце в базі даних. При цьому структурне визначення таблиці, дані, обмеження та індекси повністю видаляються з бази даних, а місце, що було раніше зайняте таблицею та її індексами, стає доступним для інших таблиць. Видаляють таблицю з бази даних за допомогою оператора DROP TABLE. Наприклад:

DROP TABLE Клієнт

3.3. Методичні засади забезпечення цілісності реляційних даних

Для того, щоб досягти прийнятого рівня якості даних, які зберігаються в реляційній базі, необхідно забезпечити їх цілісність. Терміном "цілісність даних" позначається такий стан бази даних, коли всі дані, що зберігаються в ній, не суперечать один одному. Наприклад, припустимо, що в базі даних банку створено таблицю "Клієнт". З метою унікальної ідентифікації відомостей про кожного клієнта, що були занесені в таблицю, використовується атрибут "КлієнтІдн", параме-

три якого налаштовані відповідним чином. Цілісність даних, забезпечена в такий спосіб, гарантує, що, якщо значення “КлієнтІдн” якогось клієнта рівне 438, то для жодного іншого клієнта це значення не може бути таким самим.

Сучасні СУБД підтримують чотири типи цілісності даних: цілісність сутності, доменна цілісність, цілісність посилань та цілісність, визначена користувачем.

Цілісність сутності. Цілісність сутності визначає кортеж таблиці як унікальний екземпляр певної сутності. Вона забезпечується наявністю в таблиці атрибуту з ідентифікатором або первинного ключа таблиці.

Доменна цілісність. Доменна цілісність гарантує наявність у кожному атрибуті тільки допустимих значень. Її можна забезпечувати, обмежуючи тип, формат або діапазон значень, що можуть вміщувати атрибути.

Цілісність посилань. Цілісність посилань забезпечує збереження зв’язків між таблицями при додаванні або видаленні кортежів. У реляційних базах даних цілісність посилань ґрунтується на зв’язках між зовнішніми та первинними ключами або між зовнішніми та унікальними ключами. Ця цілісність гарантує узгодженість значень ключа у зв’язаних таблицях. Подібна узгодженість вимагає відсутність посилань на неіснуючі значення та узгодженість зміни посилань на ключ при модифікації самого ключа. Для забезпечення цілісності посилань СУБД запобігає наступним діям користувачів:

- додавання записів в таблицю, якщо немає необхідного запису в таблиці, з якою вона зв’язана;
- зміна значень у таблиці, наслідком якої у зв’язаній таблиці залишаться неузгоджені записи;
- видалення записів з таблиці за наявності відповідних записів у зв’язаній таблиці.

Цілісність, визначена користувачем. Цілісність, визначена користувачем, дозволяє встановити деякі бізнес-правила, що не потрапляють у жодну іншу категорію цілісності. Усі категорії цілісності підтримують цілісність, визначену користувачем.

Слід зазначити, що в деяких теоретичних джерелах відповідного тематичного спрямування наводиться більше (або менше) чотирьох типів цілісності даних. Проте ті типи, що розглядаються в цьому розділі, зазвичай, вважаються головними типами цілісності даних.

У таблицях реляційних баз даних існує набір інструментів, які забезпечують цілісність даних. До них належать типи даних, визначення NOT NULL, DEFAULT і IDENTITY, обмеження, правила, тригери та

індекси. Розглянемо, в який спосіб кожен з них застосовується для забезпечення цілісності реляційних структур.

Типи даних. Типи забезпечують цілісність даних, оскільки дані, що вводяться в базові таблиці, повинні відповідати типу, призначеному для об'єкта. Наприклад, не можна зберігати прізвище в атрибуті, для якого призначений тип даних datetime, оскільки атрибут з таким типом даних допускає лише введення дат. Типи даних застосовуються для забезпечення доменної цілісності.

Визначення NOT NULL. Задаючи можливість введення в атрибут порожніх значень, визначаємо можливість зберігати в цьому атрибуті таблиці порожні значення. Визначення NOT NULL застосовуються для забезпечення доменної цілісності.

Визначення DEFAULT. Встановлення для атрибуту значення "за замовчуванням" задає значення, яке буде використано, якщо при додаванні до таблиці кортежу значення атрибуту не було задане. Визначення DEFAULT застосовуються для забезпечення доменної цілісності.

Визначення IDENTITY. У таблиці реляційної бази даних дозволяється визначити єдиний атрибут, який містить послідовні значення, що генеруються системою та унікально ідентифікують кожен кортеж цієї таблиці. Це дозволяє мати впевненість у тому, що кожен кортеж таблиці є унікальним. Визначення IDENTITY застосовуються для забезпечення цілісності сутності.

Обмеження. Обмеження дозволяють визначати, яким чином СУБД автоматично забезпечує цілісність даних. Обмеження визначають шаблони, за допомогою яких система перевіряє допустимі значення атрибутів. Обмеження вважаються стандартним механізмом забезпечення всіх типів цілісності реляційних структур.

Правила. Правила виконують ряд функцій, аналогічних функціям обмежень, і застосовуються для забезпечення сумісності із застарілими версіями СУБД. Правила переважно використовуються як спосіб обмеження значень атрибутів та користувальницьких типів даних. Вони застосовуються для забезпечення доменної цілісності.

Тригери. Тригери – це особливий клас збережених процедур, що автоматично запускаються при виконанні операторів UPDATE, INSERT або DELETE, для таблиці або представлення. Тригери є потужним інструментом, який використовується для автоматичної реалізації складних бізнес-правил при модифікації даних. Тригери здатні розширити логіку перевірки цілісності, що зазвичай реалізовується обмеженнями, замовчуваннями та правилами (хоча у всіх випадках, коли обмеження здатні забезпечити необхідну функціональність, слід надавати перевагу саме їм). Тригери застосовуються для забезпечення цілісності, визначеної користувачем.

Індекси. Індекс – це структура, що упорядковує значення одного або декількох атрибутів таблиці в базі даних. Індекс містить покажчики на значення, що зберігаються в заданих атрибутах таблиці і, за їх допомогою, впорядковує кортежі таблиці відповідно до встановленого порядку сортування. Індекси застосовуються для забезпечення цілісності сутності.

3.4. Реалізація основних обмежень для забезпечення цілісності даних

Обмеження – це властивість, призначена таблиці або атрибуту таблиці, яка забороняє введення у певний атрибут (або атрибути) неприпустимих значень. Обмеження дозволяють визначати способи, за допомогою яких СУБД автоматично забезпечуватиме цілісність бази даних. Обмеження встановлюють правила визначення допустимих значень атрибутів і є стандартним механізмом, який забезпечує цілісність даних. Застосування обмежень завжди має перевагу перед застосуванням тригерів, правил та замовчувань, зважаючи на ефективність обробки.

Обмеження можливі як для атрибутів, так і для таблиць:

- обмеження для атрибуту задається як частина його визначення і застосовується тільки до цього атрибуту;
- обмеження для таблиці оголошується незалежно від визначення атрибутів і застосовується до множини атрибутів таблиці.

Отже, обмеження для таблиць слід використовувати в тих випадках, коли обмеження повинно діяти відразу для декількох атрибутів. Наприклад, якщо первинний ключ таблиці складається з двох і більше атрибутів, слід використовувати обмеження для таблиці, в яке входять усі атрибути первинного ключа.

Сучасні СУБД підтримують чотири головні класи обмежень: PRIMARY KEY, UNIQUE, FOREIGN KEY і CHECK. Розглянемо їх детальніше.

Обмеження PRIMARY KEY. Кожна базова таблиця зазвичай містить атрибут (комбінацію атрибутів), значення яких унікально ідентифікують кожен рядок таблиці. Цей атрибут (комбінація атрибутів) називається первинним ключем таблиці і забезпечує цілісність сутності таблиці. Первинний ключ можна створити, визначивши обмеження PRIMARY KEY, при створенні таблиці або завдяки модифікації її структурних компонентів.

У таблиці може бути тільки одне обмеження PRIMARY KEY, до того ж, атрибут, який бере участь в обмеженні, не повинен допускати порожніх значень. Оскільки обмеження PRIMARY KEY гарантують

унікальність даних, вони часто визначаються для атрибутів з ідентифікатором. Коли в таблиці визначено обмеження PRIMARY KEY, СУБД забезпечує унікальність даних, створюючи унікальний індекс для атрибутів первинного ключа. Цей індекс також забезпечує швидкий доступ до даних у випадку використання первинного ключа в запитах. Якщо обмеження PRIMARY KEY призначене для декількох атрибутів, то в одному з них значення можуть повторюватися, але кожна комбінація усіх їхніх значень повинна бути унікальною.

Обмеження PRIMARY KEY створюють одним з наступних способів:

- під час створення таблиці для одного атрибуту (простий ключ), наприклад при створенні таблиці для збереження інформації про клієнтів в базі даних банківської установи:

```
CREATE TABLE Клієнт
(
КлієнтІдн int PRIMARY KEY,
Прізвище varchar(30) NOT NULL,
...
)
```

- під час створення таблиці для всієї таблиці (складний ключ), наприклад, при створені таблиці для збереження інформації стосовно адрес клієнтів у базі даних банківської установи:

```
CREATE TABLE Адреса
(
КлієнтІдн int NOT NULL,
ТипАдреси int NOT NULL,
КраїнаІдн int NOT NULL,
...
PRIMARY KEY (КлієнтІдн, ТипАдреси, КраїнаІдн)
)
```

- шляхом додавання обмеження до існуючої таблиці, в якій ще не створено обмеження PRIMARY KEY, наприклад:

```
ALTER TABLE Клієнт
ADD CONSTRAINT Клієнт_ПК PRIMARY KEY (КлієнтІдн)
```

Існуюче обмеження PRIMARY KEY можна модифікувати або видалити. Однак змінити довжину типу даних атрибуту, для якого встановлено обмеження PRIMARY KEY, СУБД не дозволить.

Обмеження UNIQUE. Обмеження UNIQUE дозволяє заборонити введення повторюваних значень у ті атрибути, які не беруть участі у

формуванні первинного ключа. Незважаючи на той факт, що унікальність забезпечують обидва обмеження – PRIMARY KEY та UNIQUE, у деяких ситуаціях замість обмеження PRIMARY KEY слід використовувати обмеження UNIQUE. Так чинять у випадках, якщо:

- атрибут не є первинним ключем – для таблиці можна визначити декілька обмежень UNIQUE і лише одне обмеження PRIMARY KEY;
- атрибут повинен допускати порожні значення – обмеження UNIQUE дозволяється визначити для атрибутів, що допускають порожні значення, тоді як обмеження PRIMARY KEY можна визначити лише для атрибутів, що не допускають порожні значення.

Обмеження UNIQUE створюється так само, як PRIMARY KEY:

- при створенні таблиці (під час визначення атрибутів);
- шляхом додавання до існуючої таблиці.

Для створення обмеження UNIQUE призначений майже той самий порядок використання операторів SQL, що і для створення обмеження PRIMARY KEY. Відмінність полягає в заміні слова PRIMARY KEY словом UNIQUE. Так само, як і PRIMARY KEY, встановлене обмеження UNIQUE дозволяється модифікувати або видаляти. Коли обмеження UNIQUE додається до існуючого атрибуту (атрибутів), СУБД перевіряє дані, що знаходяться в атрибутах кортежів, щоб гарантувати унікальність всіх значень, окрім порожніх.

Обмеження FOREIGN KEY. Зовнішній ключ – це атрибут, який використовується для встановлення і забезпечення зв’язку між даними двох таблиць. Класичним шляхом встановлення зв’язку між двома таблицями є додавання до однієї з таблиць атрибуту з другої таблиці і встановлення для нього обмеження FOREIGN KEY. Цей атрибут, зазвичай, містить значення первинного ключа з іншої таблиці. У таблиці може бути декілька обмежень FOREIGN KEY.

Створити зовнішній ключ можна шляхом встановлення обмеження FOREIGN KEY під час створення таблиці або внесення змін до її структури. Окрім атрибуту з обмеженням PRIMARY KEY, обмеження FOREIGN KEY може також посилатися на атрибут з обмеженням UNIQUE в інших таблицях. Обмеження FOREIGN KEY може містити порожні значення, однак, якщо атрибут з обмеженням FOREIGN KEY їх містить, перевірка цього обмеження не виконується.

Незважаючи на те, що основне призначення обмеження FOREIGN KEY полягає в контролюванні даних, які можуть бути збережені в таблиці із зовнішнім ключем, воно також виявляє зміну даних в таблиці з первинним ключем. Це обмеження забезпечує посилальну цілісність,

гарантуючи, що в таблицю з первинним ключем неможливо внести такі зміни, які зроблять недійсним посилання на дані таблиці із зовнішнім ключем. Спроба видалення кортежу або зміни значення первинного ключа в таблиці, яка бере участь у зв'язку через первинний ключ, закінчиться невдачею, якщо значення первинного ключа, що видаляється або змінюється, не відповідатиме значенню обмеження FOREIGN KEY іншої – зв'язаної з нею таблиці.

Для того, щоб успішно змінити або видалити кортеж із зв'язаної таблиці через первинний ключ, спочатку необхідно видалити або змінити дані зовнішнього ключа з таблиці із зовнішнім ключем, внаслідок чого зовнішній ключ буде зв'язаний з іншими даними первинного ключа. Існує три можливі стратегії накладення обмежень на зовнішній ключ для дотримання цілісності посилання:

1. Заборона. Згідно з даною стратегією встановлюється заборона на всі зміни первинного ключа у разі існування зовнішніх ключів, що посилаються на нього.
2. Каскадні зміни. Дія операції по змінюванню даних зв'язаного атрибуту первинного ключа "каскадним" чином розповсюджується на всі кортежі всіх таблиць, де містяться посилання на даний атрибут.
3. Привласнення значень NULL. У цьому випадку, щоб дозволити зміну або видалення даних зв'язаного атрибуту первинного ключа і не порушити цілісність посилань, відповідні значення зовнішніх ключів замінюються на значення NULL.

Обмеження FOREIGN KEY створюють одним з наступних методів:

- під час створення таблиці (в процесі визначення її атрибутів), наприклад, при створенні таблиці для збереження інформації стосовно клієнтських клієнтів у базі даних банківської установи:

```
CREATE TABLE Рахунок
(
РахунокІдн INT PRIMARY KEY,
КлієнтІдн INT REFERENCES Клієнт(КлієнтІдн)
...
)
```

- шляхом додавання обмеження до існуючої таблиці, наприклад:

```
ALTER TABLE Рахунок
ADD CONSTRAINT Рахунок_ЗК
FOREIGN KEY (КлієнтІдн) REFERENCES Клієнт(КлієнтІдн)
```

Існуючі в базі даних обмеження FOREIGN KEY можна модифікувати або видалити. Слід зазначити, що при визначенні зовнішніх ключів необхідно також накласти обмеження, які будуть опікуватись

збереженням цілісності посилань даних у зв'язаних таблицях. Для цього в SQL можна вказувати для зовнішнього ключа обмеження RESTRICT, SET NULL та CASCADE.

RESTRICT забороняє в таблиці, на яку посилається зовнішній ключ, змінювати значення первинного ключа, якщо в таблиці, що посилається, існує кортеж, атрибут якого зв'язаний з цим значенням. SET NULL означає, що при зміні значення первинного ключа в атрибут кортежів, який посилається на нього, буде встановлене значення NULL. CASCADE означає, що всі зміни в таблиці, на яку встановлено посилання, повинні вноситися в таблицю, яка посилається на неї. При заданні посилальних обмежень необхідно вказати тип обмеження (RESTRICT, SET NULL, CASCADE) і дію, на яку розповсюджується дане обмеження (UPDATE, DELETE). Наприклад:

```
CREATE TABLE Рахунок
(
РахунокІдн INT PRIMARY KEY,
КлієнтІдн INT REFERENCES Клієнт(КлієнтІдн)
        ON UPDATE CASCADE
        ON DELETE SET NULL
)
```

Обмеження CHECK. Обмеження CHECK забезпечують доменну цілісність шляхом обмеження значень, які дозволено вносити в атрибут таблиці. Цим вони дещо схожі на обмеження FOREIGN KEY. Відмінність між ними полягає в способі визначення допустимих значень. Обмеження FOREIGN KEY одержують список допустимих значень з іншої таблиці, а обмеження CHECK визначають їх на основі логічного виразу. Наприклад, можна обмежити діапазон значень атрибуту, що містить зарплати співробітників, створивши обмеження CHECK, яке допускає тільки значення від 500 до 50000 грн. Це обмеження запобігатиме введенню значень, які виходять за межі нормального діапазону зарплати.

Створити обмеження CHECK можна за допомогою будь-якого логічного виразу, що повертає значення true (так) або false (ні), залежно від задіяного логічного оператора. Для одного атрибуту в реляційній таблиці дозволяється застосувати декілька обмежень CHECK, які перевіряються згідно з порядком їх створення.

Обмеження CHECK створюють одним з наступних способів:

- під час створення таблиці (у процесі визначення її атрибутів), наприклад, при створенні таблиці для збереження інформації стосовно співробітників у базі даних банківської установи:

```
CREATE TABLE Співробітник
(
СпівробітникІдн int PRIMARY KEY,
ЛюдинаІдн int,
...
Зарплата int
CONSTRAINT Зарплата_Мін_Макс
CHECK (Зарплата > 500 AND Зарплата < 50000)
)
```

- шляхом додавання обмеження до існуючої таблиці, наприклад:

```
ALTER TABLE Співробітник
ADD CONSTRAINT Зарплата_Мін_Макс
CHECK (Зарплата > 500 AND Зарплата < 100000)
```

Аналогічно іншими обмеженням, існуючі в базі даних обмеження CHECK можна модифікувати та видаляти. Це здійснюється загальними методами мови SQL, використовуючи найменування обмеження для його ідентифікації. Найменування присвоюється обмеженню при його створенні двома способами: неявним – СУБД надає його на розсуд власної логіки, чи явним – програміст визначає його за допомогою оператора CONSTRAINT. Спосіб явного іменування обмежень з використанням оператора CONSTRAINT, що застосований в наведених прикладах, вважається хорошим стилем програмування.

ПИТАННЯ ТА ЗАВДАННЯ

1. Які параметри реляційної бази даних необхідно визначити в процесі її створення?
2. Опишіть структуру оператора SQL, який використовується для створення бази даних.
3. Які установки існуючих баз даних можна модифікувати в сучасних СУБД та яким чином це можна здійснити?
4. Як можна видалити базу даних і що при цьому відбувається?
5. Для яких об’єктів реляційної бази даних можна задати тип даних?
6. У чому полягає різниця між системними та користувальницькими типами даних?
7. Які властивості об’єкта враховуються під час призначення йому типу даних?
8. Назвіть загальні категорії базових типів даних, які підтримуються сучасними СУБД.
9. Яку інформацію треба визначити для створення користувальницького типу даних?

10. Який мінімальний набір параметрів потрібно вказати при створенні базової таблиці?
11. Опишіть структуру оператора SQL, який використовується для створення базових таблиць.
12. Яким чином для атрибуту реляційної таблиці можна встановити значення “за замовчуванням”?
13. З якою метою встановлюється властивість IDENTITY?
14. Який оператор SQL використовується для змін властивостей базових таблиць?
15. Що означає термін “цілісність даних”?
16. Які типи цілісності даних підтримують сучасні СУБД?
17. Охарактеризуйте набір інструментів, що забезпечують цілісність даних в таблицях сучасних реляційних баз даних.
18. Назвіть головні класи обмежень, за допомогою яких СУБД автоматично забезпечує цілісність бази даних.
19. Яку властивість базової таблиці дозволяє гарантувати первинний ключ?
20. В чому полягають основні відмінності обмежень PRIMARY KEY та UNIQUE?
21. Що таке зовнішній ключ в реляційних таблицях, і для забезпечення якого виду цілісності він застосовується?
22. Назвіть можливі стратегії накладення обмежень на зовнішній ключ для дотримання цілісності посилання.

ТЕСТИ

1. Що означає параметр SIZE, який використовується в операторі CREATE DATABASE?
   а) початковий розмір файлу даних;
   б) максимальний розмір файлу даних;
   в) значення приросту розміру файлу даних.
2. Що покладено в основу визначення користувальницького типу даних?
   а) системний тип даних;
   б) характеристики сервера баз даних;
   в) спеціалізовану збережену процедуру.
3. Який процес створення базових таблиць найбільш оптимальний?
   а) встановлення максимальної кількості властивостей базових таблиць при створенні;
   б) початкове створення представлень даних і, на їх основі, налаштовування властивостей базових таблиць;
   в) встановлення мінімальної кількості властивостей базових таблиць при створенні з подальшим їх додаванням у процесі розробки бази даних.

4. Які з представлених значень може містити атрибут таблиці з ідентифікатором?
   а) негативне число;
   б) порожнє значення;
   в) визначене значення за замовчуванням.
5. Який оператор використовується для модифікації таблиці?
   а) ALTER TABLE;
   б) MODIFY TABLE ;
   в) IDENTITY TABLE.
6. Який тип цілісності забезпечує властивість IDENTITY?
   а) цілісність сутності;
   б) доменну цілісність;
   в) цілісність посилань;
   г) цілісність, визначену користувачем.
7. Для забезпечення якого типу цілісності не можуть застосовуватися обмеження?
   а) цілісність сутності;
   б) доменна цілісність;
   в) цілісність посилань;
   г) можуть застосовуватися для всіх.
8. В якому випадку можна додати до реляційної таблиці обмеження PRIMARY KEY?
   а) якщо в таблицю не внесені дані;
   б) якщо немає іншого обмеження PRIMARY KEY;
   в) якщо в таблиці присутні атрибути, для яких встановлені десяткові або цілочисельні типи даних.
9. В якому випадку не відбувається перевірка цілісності посилання для поля з обмеженням FOREIGN KEY?
   а) якщо в ньому містяться порожні значення;
   б) якщо він посилається не на ключовий атрибут;
   в) якщо в атрибут, на який він посилається, не внесені дані.
10. В якій ситуації можна змінити значення зовнішнього ключа?
   а) змінювати значення зовнішнього ключа неможливо;
   б) якщо атрибут, на який він посилається, не містить ніяких даних;
   в) якщо в атрибуті, на який він посилається, присутнє змінене значення;
   г) якщо в атрибуті, на який він посилається, задана можливість каскадних змін.

РОЗДІЛ 4. МОВА МАНІПУЛЮВАННЯ ДАНИМИ В SQL

4.1. Основні способи отримання та представлення даних в SQL

Основне призначення бази даних полягає у зберіганні даних та забезпеченні їх доступності для авторизованих програмних додатків і користувачів. Користувачі дістають змогу отримувати доступ до даних та корегувати їх, застосовуючи програмні додатки й утиліти, призначені для пересилання на сервер баз даних SQL-запитів на отримання даних та представлення їх у зручному для сприйняття вигляді.

Для того, щоб отримувати дані з реляційних баз і представляти їх користувачу у вигляді результуючих наборів, використовується оператор SELECT. Результуючий набір, що повертає цей оператор, являє собою дані в табличній формі. Більшість операторів SELECT описує чотири головні властивості результуючого набору:

- атрибути, які повинні увійти до результуючого набору;
- таблицю, в якій зберігаються дані для формування результуючого набору;
- умови, яким повинні відповідати кортежі таблиці-постачальника даних, щоб потрапити в результуючий набір;
- послідовність розташування кортежів у результуючому наборі.

Повний синтаксис оператора SELECT складний, оскільки складається із значної кількості конструкцій. У загальному вигляді головні конструкції можна записати наступний чином:

SELECT список вибору
INTO назва результуючої таблиці
FROM назва таблиці-постачальника даних
WHERE умови вибору
GROUP BY атрибут для угрупування
HAVING умови пошуку
ORDER BY атрибут для сортування кортежів ASC | DESC

Наведений перелік конструкцій не дуже часто використовується в запитах у повному обсязі, оскільки більша їх частина не належить до обов'язкових. Обов'язковими є тільки SELECT та FROM. Розглянемо детально кожну конструкцію.

Конструкція SELECT. Конструкція SELECT складається з ключового слова SELECT і списку вибору. Список вибору – це набір ви-

разів, розділених комами. Кожен вираз визначає атрибут результуючого набору. Порядок розташування атрибутів результуючого набору визначається послідовністю виразів списку вибору.

Конструкція FROM. Конструкцію FROM необхідно використовувати в кожному операторі SELECT, який запитує дані, що зберігаються в базових таблицях. Ця конструкція дозволяє задати список таблиць, на атрибути яких посилається список вибору. Наприклад, якщо в базі даних банківської установи для зберігання загальних даних про людей, що можуть бути як клієнтами, так і співробітниками, існує таблиця "Людина", створена за допомогою наступного коду:

```
CREATE TABLE Людина
(
ЛюдинаІдн int NOT NULL,
Прізвище varchar(10) NOT NULL,
[Ім'я] varchar(10) NOT NULL,
[По батькові] varchar(10) NULL,
Стать varchar(7) NOT NULL,
[Дата народження] datetime  NULL
)
```

деякі дані можна отримати з неї, скориставшись наступним запитом:

```
SELECT Прізвище, [Ім'я], [По батькові]
FROM Людина
```

Повернемося до конструкції SELECT. У списку її вибору можна вказати ключові слова, які визначають кінцевий формат результуючого набору, такі як: DISTINCT, TOP, PERCENT і т.д.

Ключове слово DISTINCT забороняє вивід у результуючому наборі кортежів, що повторюються. Наприклад, наступний запит:

```
SELECT DISTINCT Прізвище, [Ім'я], [По батькові]
FROM Людина
```

дозволяє одержати список значень "Прізвище" без дублікатів.

Ключове слово TOP *n* задає перші *n* кортежів результуючого набору, які необхідно відобразити. Наприклад, наступний запит:

```
SELECT TOP 10 Прізвище, [Ім'я], [По батькові]
FROM Людина
```

повертає перших 10 кортежів.

Якщо задане ключове слово PERCENT, то *n* – це відсоток кортежів, що будуть відображені від загального числа кортежів у результуючому наборі.

Існує можливість зробити використання оператора SELECT більш гнучким, присвоївши таблиці-постачальнику даних псевдонім. Псевдонім можна присвоїти таблиці за допомогою ключового слова AS. Наприклад:

SELECT Л. Прізвище, Л. [Ім'я], Л. [По батькові]
FROM Людина AS Л

У списку вибору дозволяється задавати різні типи інформації, такі як прості вирази або скалярні підзапити. Наприклад:

SELECT Прізвище + ' ' + [Ім'я]+ ' '+ [По батькові] AS "ПІБ"
FROM Людина

Конструкція INTO. Конструкція INTO дозволяє вказати, що для результуючого набору буде створена нова таблиця, ім'я якої задане цією конструкцією. За допомогою оператора SELECT INTO вдається об'єднати дані з декількох таблиць в одну таблицю. Використання цієї структури особливо корисне для створення нової таблиці з даними, що були отримані з іншої бази даних на віддаленому сервері. Наприклад:

SELECT Прізвище, [Ім'я], [По батькові]
INTO ПІБ
FROM Людина

Результуючий набір, що буде сформований цим запитом, створює таблицю ПІБ. У новій таблиці атрибути "Прізвище", "Ім'я", "По батькові" міститимуть значення з таблиці "Людина".

Конструкція WHERE. У операторі SELECT конструкція WHERE визначає атрибути таблиці-постачальника даних, які необхідні для побудови результуючого набору. Ця конструкція виконує роль фільтра, формуючи умови пошуку. У процесі побудови результуючого набору обираються лише ті кортежі, які відповідають умовам пошуку. Наприклад, для вибору з таблиці, що містить інформацію про співробітників, дані про зарплати, які перевищують 5000 та відповідні їм ідентифікатори співробітників, може бути використаний наступний запит:

SELECT СпівробітникІдн, Зарплата
FROM Співробітник
WHERE Зарплата > 5000

Конструкція GROUP BY. Конструкція GROUP BY використовується для отримання підсумкових значень у кожному атрибуті результуючого набору. При застосуванні оператора SELECT без конструкції GROUP BY агрегатні функції повертають лише одне підсумкове значення.

Після ключових слів GROUP BY у запит вноситься атрибут (атрибути), який називається групуючим. Конструкція GROUP BY ви-

значає кортежі результуючого набору. Для кожного конкретного значення групуючого атрибута (атрибутів) можливий тільки один кортеж. У кожному кортежі результуючого набору містяться підсумкові дані, пов'язані з деяким значенням його групуючих атрибутів. Наприклад, якщо в базі даних банківської установи для зберігання даних про клієнтські рахунки існує таблиця "Рахунок", що була створена за допомогою наступного коду:

```
CREATE TABLE Рахунок
(
РахунокІдн int NOT NULL,
КлієнтІдн int NOT NULL,
[Вид рахунку] int NOT NULL,
[Дата відкриття] datetime  NULL,
Залишок money NOT NULL
)
```

отримати сумарні залишки по кожному виду рахунку можна завдяки використанню наступного запиту:

```
SELECT [Вид рахунку], SUM(Залишок) AS ' Сумарний залишок '
FROM Рахунок
GROUP BY [Вид рахунку]
```

Конструкція HAVING. Конструкція HAVING задає додаткові фільтри, які застосовуються після завершення фільтрації, яка визначається конструкцією WHERE. Як правило, конструкція HAVING використовується з конструкцією GROUP BY (хоча її дозволяється застосовувати окремо). Наприклад, отримати з таблиці "Рахунок" сумарні залишки по 2-му та 3-му виду рахунку, які перевищують 10000, можна використовуючи наступний запит:

```
SELECT [Вид рахунку], SUM(Залишок) AS ' Сумарний залишок '
FROM Рахунок
WHERE [Вид рахунку] >1 AND [Вид рахунку] <4
GROUP BY [Вид рахунку]
HAVING SUM(Залишок) > 10000
```

Розуміння правильної послідовності, в якій застосовуються конструкції WHERE, GROUP BY і HAVING, допомагає програмувати ефективні запити. Воно полягає в наступному:

- конструкція WHERE фільтрує кортежі, які формуються в результаті установок, заданих у конструкції FROM;
- вихідна інформація конструкції WHERE групується за допомогою конструкції GROUP BY;

- кортежі згрупованого результату фільтруються засобами конструкції HAVING.

Конструкція ORDER BY. Конструкція ORDER BY сортує результат запиту по одному або декільком атрибутам. Сортування може бути як за збільшенням (ASC), так і за зменшенням (DESC). Якщо не заданий жоден з видів сортування, за замовчуванням передбачається ASC. Якщо в конструкції ORDER BY названо декілька атрибутів, виконується вкладене сортування. Наприклад, наступний оператор сортує кортежі таблиці "Співробітник" по зменшенню значень "Зарплата":

```
SELECT СпівробітникІдн, Зарплата
FROM Співробітник
ORDER BY Зарплата DESC
```

4.2. Програмування складних запитів на отримання даних з реляційних таблиць

У процесі розгляду основ роботи оператора SELECT та застосування його конструкцій були представлені прості приклади запитів на отримання даних з тієї чи іншої базової таблиці. Однак на практиці такі запити використовуються рідко тому, що у добре нормалізованій базі дані розподілені по різних базових таблицях, кількість яких може бути суттєвою. Тому для отримання потрібної інформації з бази використовуються запити відразу до декількох таблиць. Такі запити входять до розряду складних.

Один із способів програмування складних запитів полягає у використанні вкладених запитів (підзапитів). Вкладеним запитом називається оператор SELECT, вкладений в структуру іншого оператора SELECT, INSERT, UPDATE, DELETE або в інший підзапит. Вкладений запит дозволяється застосовувати в будь-якому місці, де можна використовувати вирази. У наступному прикладі підзапит вкладений в конструкцію WHERE зовнішнього оператора SELECT:

```
SELECT Прізвище, [ Ім'я], [По батькові]
FROM Людина
WHERE ЛюдинаІдн =
(
    SELECT ЛюдинаІдн
    FROM Співробітник
    WHERE Посада = ' Директор '
)
```

Мета цього запиту – отримати ім'я директора з бази даних віртуальної банківської установи. Вбудований оператор SELECT спочатку визначає значення атрибуту "ЛюдинаІдн" у таблиці "Співробітник",

атрибут “Посада” якого містить запис “Директор”. Після цього значення атрибуту “ЛюдинаІдн” використовується в зовнішньому операторі SELECT для отримання імені співробітника з однаковим значенням відповідного атрибуту в таблиці “Людина”.

Оператори, у складі яких є вкладений запит, зазвичай, використовуються в одному з наступних форматів:

WHERE вираз IN | NOT IN (підзапит)
WHERE вираз порівняння ANY | ALL (підзапит)
WHERE EXISTS | NOT EXISTS (підзапит)

Результатом вкладеного запиту з ключовим словом IN або NOT IN є список, що складається з нуля або більше значень. Результат, який повертає вкладений запит, використовується зовнішнім запитом.

В операторах порівняння, що використовуються з вкладеними запитами, дозволяється застосовувати ключові слова ALL або ANY, які порівнюють скалярне значення з набором значень одного атрибуту. Ключове слово ALL застосовується до всіх значень, а ключове слово ANY – як мінімум, до одного.

Коли вкладений запит містить ключове слово EXISTS, він функціонує як перевірка наявності тієї або іншої сутності. Конструкція WHERE зовнішнього запиту перевіряє, чи є кортежі, які повертає вкладений запит. Цей вкладений запит насправді не видає ніяких даних, натомість він повертає значення TRUE або FALSE.

Інший спосіб програмування складних запитів полягає у з’єднанні декількох таблиць з метою отримання результуючого набору, який містить атрибути й кортежі з цих таблиць. З’єднання дозволяють отримувати дані з двох або більше таблиць на основі логічних зв’язків між ними. З’єднання вказує СУБД, яким чином слід використовувати дані однієї таблиці для вибору кортежів з іншої таблиці.

Умови з’єднання можна задавати в конструкціях FROM або WHERE. Проте визначення умов з’єднання в конструкції FROM дозволяє відокремити їх від інших умов пошуку, заданих в конструкції WHERE або HAVING. Тому рекомендується задавати з’єднання саме в конструкції FROM.

Коли один запит посилається на декілька таблиць, усі посилання на атрибути слід визначати точно. Будь-яке ім’я атрибута, що повторюється у двох або більше таблицях у складі одного запиту, необхідно конкретизувати, вказавши ім’я таблиці, або її псевдонім.

Список вибору з’єднання може посилатися на всі атрибути таблиць, що з’єднуються, або на деяку їх підмножину. Необов’язково, щоб у список вибору потрапили атрибути з усіх таблиць з’єднання. Наприклад, іноді в з’єднанні, яке утворюється з трьох таблиць, одну

таблицю використовують як “міст” між двома іншими, тому атрибути цієї таблиці у список вибору можуть не потрапити.

В умові з’єднання зазвичай використовується знак рівності (=), хоча дозволяється застосовування інших реляційних операторів або операторів порівняння. У процесі обробки з’єднання механізм обробки запитів СУБД вибирає для цього найефективніший метод. Загальна логічна послідовність операцій наступна:

- застосовуються умови з’єднання з конструкції FROM;
- застосовуються умови з’єднання і умови пошуку з конструкції WHERE;
- застосовуються умови пошуку з конструкції HAVING.

Атрибутам, які використовуються в умові з’єднання, необов’язково мати однакове ім’я або тип даних. Проте, якщо типи даних не ідентичні, необхідно, щоб вони були сумісні, тобто СУБД могла виконати їх неявне перетворення. Якщо неявне перетворення типів неможливе, умова з’єднання повинна явно перетворювати типи даних за допомогою спеціальної функції – CAST.

Загальну сукупність з’єднань можна поділити на внутрішні та зовнішні. Внутрішні з’єднання повертають кортежі за умови, що в з’єднуваних таблицях існує хоча б по одному кортежу, які відповідають умові з’єднання. При цьому кортежі, для яких немає відповідних кортежів з іншої таблиці, виключаються. Зовнішні з’єднання повертають усі кортежі, принаймні з однієї, вказаної в конструкції FROM, таблиці, які відповідають умовам пошуку, що задаються конструкціями WHERE або HAVING.

Внутрішні з’єднання. У внутрішніх з’єднаннях значення атрибутів, що використовуються для його утворення, порівнюються за допомогою оператора порівняння. Наприклад, в операторі SELECT внутрішнє з’єднання використовується для отримання даних з таблиць “Людина” і “Співробітник”:

```
SELECT Л.Прізвище, Л.[Ім’я], С.Зарплата
FROM Людина AS Л INNER JOIN Співробітник AS С
ON Л.ЛюдинаІдн = С.ЛюдинаІдн
```

Цей оператор SELECT одержує дані із атрибутів “Прізвище” та “Ім’я” таблиці “Людина” (Л) та атрибута “Зарплата” таблиці “Співробітник” (С). Оскільки в цьому операторі застосовується внутрішнє з’єднання, він повертає тільки ті кортежі, для яких в атрибутах з’єднання (Л.ЛюдинаІдн і С.ЛюдинаІдн) є рівні значення.

Внутрішні з’єднання – це єдиний тип з’єднань, для яких у стандарті SQL-92 дозволяються визначення в складі конструкції WHERE.

З’єднання, визначені в такий спосіб, називаються “застарілими” внутрішніми з’єднаннями.

Зовнішні з’єднання. Сучасні СУБД підтримують три типи зовнішніх з’єднань: ліві, праві і повні.

Результуючий набір, що генерує оператор SELECT з лівим зовнішнім з’єднанням, складається з усіх кортежів таблиці, на яку посилається конструкція LEFT OUTER JOIN, і яка розташована зліва від цієї конструкції. З таблиці праворуч беруться лише ті атрибути, що відповідають умові з’єднання. У разі невідповідності в атрибут результуючого набору вставляється порожнє значення.

Результуючий набір, що генерує оператор SELECT з правим зовнішнім з’єднанням, складається з усіх кортежів таблиці, на яку посилається конструкція LEFT OUTER JOIN, і яка розташована праворуч від цієї конструкції. З таблиці ліворуч беруться лише ті атрибути, які відповідають умові з’єднання. У разі невідповідності в атрибут результуючого набору вставляється порожнє значення.

Результуючий набір, що генерує оператор SELECT, в якому є повне зовнішнє з’єднання, складається з усіх кортежів обох таблиць незалежно від того, чи присутні в атрибутах таблиць значення, що були задані умовами з’єднаннями. Повне зовнішнє з’єднання забезпечується конструкцією FULL OUTER JOIN, яка повертає таблицю заданих відповідностей з порожніми значеннями в атрибутах кортежів при відсутності умови з’єднання.

4.3. Індексування реляційних таблиць

Важливим методом, здатним суттєво підвищити швидкість виконання запитів до реляційних баз даних, вважається індексування таблиць. У результаті індексування базових таблиць та представлень утворюються індекси. Індекси – це об’єкти бази даних, що підвищують продуктивність запитів, допомагаючи СУБД знайти потрібний кортеж. Індекс нагадує за своїм призначенням книжковий зміст, по якому читач швидко відшукує потрібну інформацію. Індекс бази даних формується зі значень одного або декількох атрибутів таблиці, які у цьому випадку називаються ключем індексу, і покажчиків на відповідні кортежі. При виконанні запиту з індексом оптимізатор запитів використовує ключ індексу для пошуку потрібного кортежу.

Диспетчер індексів структурує індекс у вигляді збалансованого дерева, або B-дерева (Balanced tree, B-tree). Кожен об’єкт у структурі дерева являє собою групу відсортованих ключів індексу, яка називається сторінкою індексу. Упорядкування індексу за значеннями його ключів також підвищує продуктивність запитів. СУБД починає вико-

нання всіх запитів на пошук даних з коріння B-дерева, потім просувається далі по стовбуру, доки не досягне листового рівня.

Глибина дерева залежить від числа кортежів таблиці, а ширина дерева – від розміру ключа індексу. Дерево, створене для таблиці з великою кількістю кортежів і великим ключем індексу, є широким та глибоким. Це небажано, оскільки, чим менше дерево, тим швидше повертається результат пошуку.

Для оптимальної продуктивності запитів слід створювати індекси на тих атрибутах таблиці, які часто застосовуються в запитах на отримання даних. Небажано створювати індекс для кожного атрибута таблиці, оскільки при модифікації індексованої таблиці індекс обов'язково оновлюється. Отже, збільшення кількості індексів може негативно вплинути на продуктивність бази даних. Тому, перш ніж створювати індекс, варто переконатися, що запланований виграш у продуктивності запитів переважить додаткову витрату ресурсів системи на його супровід.

Існує два типи індексів: кластерні та некластерні. Кластерний індекс містить кортежі таблиці, а некластерний – покажчики на кортежі. Якщо для таблиці побудований кластерний індекс, то некластерний можна використовувати як допоміжний при пошуку даних. У більшості випадків для таблиці спочатку слід створювати кластерний індекс, а потім – один або декілька некластерних.

Таблиця або представлення може мати лише один кластерний індекс, оскільки ключ кластерного індексу фізично впорядковує таблицю, або представлення. Цей тип індексів особливо ефективний при виконанні запитів. Підходом до сортування кортежів кластерний індекс нагадує словник з його алфавітним порядком сортування слів і наявністю визначень після кожного слова.

При створенні обмеження PRIMARY KEY в таблиці, де ще немає кластерного індексу, СУБД використовує для створення ключа кластерного індексу атрибут з первинним ключем таблиці. Якщо в таблиці вже є кластерний індекс, то для атрибуту з обмеженням PRIMARY KEY створюється некластерний індекс. Атрибут з первинним ключем корисний для індексу, оскільки в ньому гарантовано містяться унікальні значення. У цьому випадку розмір B-дерева менше, ніж при використанні надмірних значень, отже механізм пошуку працює ефективніше.

Індекси можуть створюватися не тільки для атрибутів з обмеженнями, а й для будь-якого атрибуту або комбінації атрибутів таблиці чи представлення. Унікальність кластерного індексу забезпечується за допомогою внутрішніх механізмів. Тому при створенні неунікального кластерного індексу для атрибуту з надмірними значеннями СУБД ге-

нерує для значень атрибутів, що дублюються, унікальні ідентифікатори, які використовуються як вторинний ключ сортування. Щоб уникнути додаткової роботи з підтримки унікальних значень атрибутів з надмірними значеннями, слід віддати перевагу кластерним індексам на атрибутах з обмеженням PRIMARY KEY.

Кількість некластерних індексів кінцева (в MS SQL останніх версій можна створити до 250 некластерних індексів або 249 некластерних і 1 кластерний). Якщо в таблиці немає кластерного індексу, таблиця є неврегульованою і називається купою. Некластерний індекс, створений для купи, містить покажчики на записі таблиці. Кожен елемент сторінки індексу містить ідентифікатор запису (row ID, RID) – покажчик на табличний кортеж у купі, що містить номер елемента зовнішньої пам'яті, з якої починається сторінка. За наявності кластерного індексу сторінки некластерного індексу містять замість RID-ідентифікаторів ключі кластерного індексу.

Крім типу (кластерний або некластерний), індекс має ряд інших властивостей. Згідно з ним кожен індекс можна визначити як:

1. Унікальний індекс. В унікальному індексі ключі та відповідні їм значення атрибутів повинні бути унікальними.
2. Складний індекс. Складним називається індекс, ключ якого утворений декількома атрибутами таблиці.
3. Індекс з упорядкуванням ключів. Це індекс, в якому ключі впорядковані за збільшенням або за зменшенням значень.
4. Повнотекстовий індекс. Цей індекс дозволяє виконувати повнотекстові запити для пошуку в базі даних рядків символьних даних.

Для створення індексів використовують наступні оператори SQL: CREATE INDEX, CREATE TABLE і ALTER TABLE.

При застосуванні CREATE INDEX слід вказати ім'я індексу, таблицю або представлення, а також атрибут (атрибути), для яких створюється індекс. Також можна (але необов'язково) задати унікальність індексу, його тип (кластерний або некластерний), порядок сортування кожного атрибуту, властивості індексу та розташування групи файлів для зберігання індексу. Основні конструкції оператора CREATE INDEX мають наступний вигляд:

CREATE
UNIQUE CLUSTERED | NONCLUSTERED INDEX ім'я індексу
ON ім'я таблиці | ім'я представлення (ім'я атрибуту)
WITH властивість індексу
ON група файлів

До необов'язкових належать конструкції UNIQUE, CLUSTERED або NONCLUSTERED. Також необов'язково задавати властивості індексу за допомогою конструкції WITH і групи файлів, в якій створюється індекс (використовуючи другу конструкцію ON). Для необов'язкових параметрів встановлюється значення за замовчуванням, а саме:

- створити некластерний індекс;
- упорядкувати всі кортежі індексу за збільшенням ключового атрибуту;
- розміщувати всі результати сортування під час створення індексу в групі файлів за замовчуванням.

Наприклад, в наступному операторі:

CREATE INDEX ОпераціяРахунок ON Операція(Рахунок)

для всіх необов'язкових конструкцій застосовуються параметри за замовчуванням: у таблиці "Операція" створюється неунікальний і некластерний індекс "ОпераціяРахунок", ключем якого є атрибут "Рахунок".

У багатьох СУБД під час визначення в таблиці обмежень PRIMARY KEY та UNIQUE індекс створюється автоматично. Ці обмеження визначаються під час створення або модифікації таблиці. Оскільки оператори CREATE TABLE і ALTER TABLE дають можливість задавати ці обмеження, їх можна використовувати для створення індексів. Основні конструкції оператора CREATE TABLE, що пов'язані із створенням індексів, наступні:

CREATE TABLE ім'я таблиці
(ім'я атрибуту тип даних
CONSTRAINT ім'я обмеження
PRIMARY KEY | UNIQUE
CLUSTERED | NONCLUSTERED
WITH властивість індексу
ON група файлів)

Наприклад:

```
CREATE TABLE Клієнт
(
КлієнтІдн int PRIMARY KEY NONCLUSTERED,
ЛюдинаІдн int UNIQUE CLUSTERED,
Статус varchar(40),
)
```

Синтаксис оператора ALTER TABLE для створення чи модифікації обмежень PRIMARY KEY, або UNIQUE, аналогічний CREATE TABLE. Невелика відмінність полягає у необхідності вказувати в опе-

раторі ALTER TABLE, що буде відбуватися з обмеженням: зміна, додавання або видалення.

У випадку, якщо індекс більше непотрібен або пошкоджений, він підлягає видаленню. Зокрема, необхідно видаляти індекси для таблиць, значення ключів яких почали часто змінюватись. В іншому випадку СУБД витрачатиме зайві обчислювальні ресурси на супровід таких індексів. Синтаксис видалення індексів такий:

DROP INDEX ім'я таблиці. ім'я індексу

В операторі DROP INDEX слід вказати ім'я таблиці або представлення. Один оператор DROP INDEX може видаляти декілька індексів, розділених комами.

Шляхом послідовного використання команд DROP INDEX та CREATE INDEX можна змінювати, в разі потреби, властивості індексу. Проте для великих таблиць перебудова індексів у такий спосіб може вимагати надто багато процесорного ресурсу. Зокрема, у випадку, якщо для таблиці або представлення створений кластерний індекс, то всі некластерні індекси цієї таблиці використовують посилання на його ключ. При видаленні кластерного індексу (за допомогою оператора DROP INDEX) усі некластерні індекси перебудовуються на застосування замість ключа кластерного індексу посилання на RID-ідентифікатори. Коли згодом відновити кластерний індекс (за допомогою оператора CREATE INDEX), то всі некластерні індекси знову перебудовуються на застосування його ключа.

Зважаючи на такі незручності, на практиці застосовують більш досконалі способи реорганізації індексів. Оператор DBCC DBREINDEX перебудовує один або декілька індексів таблиці чи представлення. Для перебудови всіх індексів слід примусити DBCC DBREINDEX перебудувати кластерний індекс. Таким чином реалізується перебудова всіх індексів таблиці або представлення. Інший спосіб – запустити оператор без вказівки імені, при цьому також перебудовуються всі індекси. Оператор DBCC DBREINDEX особливо корисний для перебудови індексів, створених обмеженнями PRIMARY KEY і UNIQUE, оскільки (на відміну від DROP INDEX) перед перебудовою необов'язково видаляти обмеження.

Іноді, внаслідок зміни угод про іменування, або при невідповідності існуючих індексів угодам про іменування здійснюється їх перейменування. Перейменувати індекс можна видаливши його і створивши наново. Однак сучасні СУБД надають простіші способи. Зокрема, MS SQL дозволяє перейменувати індекс засобами системної збереженої процедури sp_rename, наприклад:

sp_rename @objname = 'Операція.ОпераціяРахунок', @newname = 'ІндексРахунок', @objtype = 'INDEX'

У значення вхідного параметра @objname вноситься ім'я таблиці. Якщо цього не зробити, процедура не знайде індекс, який потрібно перейменувати. Проте ім'я таблиці виключене з вхідного параметра @newname, оскільки воно береться з параметра @objname. Для вхідного параметра @objtype слід задати значення "INDEX", інакше процедура не знайде потрібний тип об'єкта, який потрібно перейменувати.

Тепер, після того, як було розглянуто що таке індекс, як його створити і як ним управляти, слід звернути увагу на правила, що дозволяють визначити, коли слід створювати індекс і які властивості індексу потрібно налаштувати для підвищення продуктивності бази даних. Оскільки в таблиці або представленні може бути лише один кластерний індекс, продумати його конструкцію важливіше, ніж некластерного.

Індекси створюються для підтримки різних типів запитів до бази даних, що найчастіше виконуються користувачами. При цьому оптимізатор запитів для пошуку даних застосовує один або декілька індексів. Індекси здатні підвищити ефективність виконання запитів таких типів:

1. Запити, в яких критерій пошуку, що задається конструкцією WHERE, точно відповідає вказаному значенню. Наприклад:

   SELECT Прізвище, [Ім'я], [По батькові]
   FROM Людина
   WHERE ЛюдинаІдн = 5

Якщо конструкція WHERE забезпечує повернення одного певного значення – для таких запитів варто вибрати кластерний індекс. У запитах, які повертають декілька унікальних записів, краще використовувати некластерний індекс. Наприклад, якщо запросити в базі даних відомості про людей, і як критерій вибірки застосувати тільки ім'я, запит точно відповідних значень, імовірно, поверне декілька кортежів.

2. Запити із знаками підстановки, в яких для пошуку значень застосовується конструкція LIKE і критерій пошуку, починаються із знаку відсотка (%). Наприклад:

   SELECT Прізвище, [Ім'я], [По батькові]
   FROM Людина
   WHERE LIKE '%ко'

Індекси не підвищують ефективність виконання подібних запитів, оскільки ключі індексу починаються з конкретного символьного або числового значення.

3. Запити діапазону значень – запити на пошук послідовності значень, наприклад:

SELECT Прізвище, [Ім'я], [По батькові]
FROM Людина
WHERE LIKE 'M%' AND ''H%'

Для цього типу запитів краще вибрати кластерні індекси, оскільки їхні ключі фізично впорядковані. Тому, якщо знайдений перший запис, імовірно, що решта записів діапазону розташовані поряд.

4. Запити з упорядкованими результатами, що використовують конструкцію ORDER BY. Якщо якийсь атрибут або їх комбінація часто впорядковується як ключ для такого сортування, слід подумати про реалізацію сортування за допомогою індексу.

4.4. Використання курсорів для доступу до реляційних структур

Доступ до даних, що зберігаються в реляційних структурах, можна здійснювати також за допомогою курсорів. Курсор – це об'єкт реляційної бази даних, який встановлює відповідність з результуючим набором, що повертає оператор SELECT і вказує на один з його кортежів. Після того, як позиція курсору встановлена на певному кортежі, над цим кортежем або над блоком кортежів, що починається з цієї позиції, можна виконувати дії з маніпулювання даними.

Нагадуємо, що у реляційних базах даних операції з маніпулювання даними виконуються над наборами кортежів. Оператор SELECT повертає набір, який складається з усіх кортежів, що відповідають заданій в конструкції WHERE умові. Повний набір кортежів, який повертає оператор, відповідає результуючому набору. Програмні додатки, особливо інтерактивні, не завжди можуть ефективно працювати з результуючим набором як з єдиним об'єктом. Для цих додатків необхідний механізм, що дозволить їм працювати з невеликими блоками кортежів. Такий механізм реалізований у вигляді курсорів.

Курсори розширюють можливості обробки результатів запитів на отримання даних з реляційних структур, забезпечуючи підтримку наступних функцій:

- позиціонування на певних кортежах результуючого набору;
- виділення одного кортежу або блоку кортежів, починаючи від визначеної позиції курсору в результуючому наборі;
- підтримку модифікації даних в кортежі, на якому позиціонується курсор у результуючому наборі;

- підтримку різних зон видимості під час змін, що вносять в дані результуючого набору різні користувачі;
- надання доступу до даних результуючого набору операторам SQL у складі сценаріїв, збережених процедур і тригерів.

Сучасні СУБД підтримують три типи курсорів: серверні курсори SQL, серверні курсори прикладного програмного інтерфейсу API (Application Programming Interface) та клієнтські курсори. Оскільки перші два види курсорів реалізовані на сервері, їх у сукупності називають серверними курсорами. Розглянемо їх більш детально.

Серверні курсори SQL. Курсори SQL реалізовані на сервері баз даних і управляються операторами SQL, які передаються від клієнтської компоненти. Крім того, оператори SQL можуть також міститися в пакетах, збережених процедурах і тригерах. При роботі з курсорами SQL для оголошення, заповнення й отримання даних використовується спеціальний набір операторів SQL.

Для оголошення курсору використовується оператор DECLARE CURSOR, далі слід задати оператор SELECT, який згенерує результуючий набір курсору. Наприклад:

DECLARE КурсЛюдина CURSOR FOR
SELECT Прізвище, [Ім'я], [По батькові]
FROM Людина
ORDER BY Прізвище

Для заповнення курсору використовується оператор OPEN. Він виконує оператор SELECT, що вміщений в оператор DECLARE CURSOR. Наприклад:

OPEN КурсЛюдина

Для отримання окремих кортежів з результуючого набору використовується оператор FETCH. Як правило, оператор FETCH виконується багато разів (для кожного кортежу результуючого набору). Наприклад:

FETCH NEXT FROM КурсЛюдина

За необхідності можна скористатися операторами зміни або видалення даних кортежу, на який указує курсор. Розгляд способів модифікації даних у SQL буде зроблено далі.

Для закриття курсору застосовується оператор CLOSE, наприклад:

CLOSE КурсЛюдина

Цей процес завершує активну операцію з курсором і звільняє деякі ресурси (наприклад, результуючий набір курсору і його блокування для поточного кортежу). Після закриття курсор залишається оголошеним, тому його можна повторно відкрити за допомогою оператора OPEN.

Видаляє з поточного сеансу посилання на курсор оператор DEALLOCATE. Цей процес повністю звільняє усі ресурси, виділені для курсору (у тому числі ім'я курсору). Після звільнення курсору можливе його повторне створення за допомогою оператора DECLARE.

Для використання серверних курсорів SQL безпосередньо в програмних додатках необхідне застосування оператора FETCH з подальшою прив'язкою кожного атрибута, що повернув цього оператора, до змінної в програмі. Більш ефективний спосіб полягає у використанні вбудованої функціональності курсорів API.

Серверні курсори API. Поняття API легко зрозуміти, усвідомивши, що кожна інформаційна система, у тому числі й операційна, має свій прикладний інтерфейс. Наприклад, API ОС Windows складається з набору функцій, які дозволяють при розробці власних програмних додатків використовувати системні Windows-конструкції. Отже, курсори можуть бути реалізовані на сервері й управлятися курсорними функціями API ОС Windows. Кожного разу, коли клієнтський програмний модуль викликає функцію API для управління курсорами, компонент доступу технології OLE DB або драйвер ODBC передає на сервер запит на виконання потрібної дії над курсором API сервера БД.

Хоча синтаксис курсорів API й курсорів SQL відрізняється, загалом для роботи з цими курсорами необхідно виконати однакові процедури: оголосити курсор, відкрити його, отримати з його допомогою дані, потім закрити і звільнити курсор. При використанні серверного курсору API в технологіях доступу до БД: OLE DB, ODBC або ADO за допомогою функцій API ОС Windows виконуються наступні дії:

- відкриття з'єднання;
- установка атрибутів, що визначають характеристики курсору, для якого API відповідної технології автоматично встановлює відповідність з результуючим набором;
- виконання одного або декількох операторів SQL;
- вибірка кортежів з результуючого набору.

Клієнтські курсори кешують усі кортежі клієнтського результуючого набору. При кожному виклику клієнтським додатком функції API для управління курсором драйвер ADO або ODBC виконує дії з використанням курсору над кортежами результуючого набору, що були кешовані на клієнтській платформі. При цьому не застосовуються ніякі функції підтримки серверних курсорів. Клієнтські курсори підтримують тільки статичні курсори одного напрямку дії. Тому клієнтські

курсори рекомендується застосовувати лише для того, щоб частково обійти обмеження, пов'язані з тим, що серверні курсори підтримують не всі оператори SQL.

Розглянувши основні види курсорів, перейдемо до детального вивчення процесу отримання даних з реляційних структур за допомогою курсорів.

Операція отримання кортежу з курсору називається вибіркою. При роботі з курсорами SQL вибірка кортежів з результуючого набору курсору здійснюється за допомогою оператора FETCH.

Оператор FETCH підтримує набір параметрів для вибірки певних кортежів:

- FETCH FIRST – проводить вибірку першого кортежу курсору;
- FETCH NEXT – проводить вибірку кортежу, розташованого за останнім вибраним кортежем;
- FETCH PRIOR – проводить вибірку кортежу, розташованого перед останнім вибраним кортежем;
- FETCH LAST – проводить вибірку останнього кортежу курсору;
- FETCH ABSOLUTE *n* – якщо *n* позитивне ціле число, проводить вибірку *n*-го кортежу, починаючи з першого кортежу курсору; якщо *n* – від'ємне ціле число, виконує вибірку кортежу, розташованого за *n* кортежів до кінця курсору; якщо *n* = 0, то вибірка кортежів взагалі не відбувається;
- FETCH RELATIVE *n* – проводить вибірку кортежу, розташованого через *n* кортежів після останнього вибраного кортежу – якщо *n* позитивне; якщо *n* – від'ємне, то вибирається кортеж за *n* позицій перед останнім вибраним кортежем; якщо *n* = 0, то знов вибирається поточний кортеж.

Відразу після відкриття курсору його логічна позиція завжди встановлюється на першому кортежі.

У курсорів SQL є обмеження: вони вибирають тільки по одному кортежу за раз. Серверні курсори API здатні підтримувати вибірку блоку кортежів за одну операцію вибірки. Курсор, який підтримує вибірку декількох кортежів одночасно, називається блоковим.

4.5. Можливості SQL для роботи з даними у форматі XML

У реляційній СУБД усі результати дій над таблицями баз даних відображаються в табличній формі. Клієнтські компоненти класичної "клієнт-серверної" системи обробляють результати виконаного оператора SELECT, вибираючи по черзі одиночні кортежі або блоки з табличного результуючого набору і прив'язуючи значення їхніх атрибутів до змінних у програмі. З іншого боку, програмісти Web-додатків

мають справу з ієрархічним представленням даних у форматі XML. Розширювана мова розмітки (Extensible Markup Language, XML) являє собою гіпертекстову мову програмування, яка використовується для опису вмісту сукупності даних і способу їх виведення на різні пристрої або їх відображення на Web-сторінках.

У сучасних СУБД передбачена можливість повертати результати SQL-запитів замість стандартного набору кортежів у вигляді XML. У результаті вони здатні виконувати роль сервера баз даних з підтримкою XML. Ці запити можна виконувати безпосередньо, або у складі збережених процедур. Для того, щоб одержати результати у вигляді документа XML, застосовують конструкцію FOR XML оператора SELECT. Синтаксис конструкції FOR XML наступний:

FOR XML RAW | AUTO | EXPLICIT, XMLDATA, ELEMENTS, BINARY BASE64

У конструкції FOR XML задають один з наступних режимів XML: RAW, AUTO або EXPLICIT. Режими XML визначають наступні форми результуючого набору XML:

1. Режим RAW трансформує кожен кортеж результату запиту в елемент XML з ідентифікатором, що відповідає ідентифікатору кортежу, причому кожний непорожній атрибут запиту представляється атрибутом елемента XML, ім'я якого співпадає з ім'ям атрибуту.
2. Режим AUTO повертає результат у вигляді вкладених елементів XML, в якому кожна вказана в конструкції FROM таблиця, якщо хоча б один її атрибут знаходиться в списку конструкції SELECT, представлена у вигляді елемента XML.
3. В режимі EXPLICIT форма документа XML, який повертається в результаті виконання запиту, визначається при створенні запиту. При цьому запит складають так, щоб додаткова інформація про можливі вкладені елементи задавалася як частина запиту.

Ключове слово XMLDATA вказує, що необхідно одержати схему XML-Data. Вона додається в документ як вбудована схема. Основне призначення ключового XMLDATA, заданого в запиті – отримання відомостей про типи даних XML, які можна використовувати там, де ці відомості необхідні (наприклад, при обробці числових виразів). В інших випадках весь вміст документа XML вважається текстовим виразом. При генерації схеми XML-Data потрібні додаткові ресурси, що може призвести до зниження продуктивності серверної платформи. Тому її слід застосовувати тільки тоді, коли необхідні відомості про типи даних.

Якщо заданий параметр ELEMENTS, то атрибути результуючого набору повертаються у вигляді підлеглих елементів. В інших випадках

вони відповідають атрибутам XML. Цей параметр підтримується тільки в режимі AUTO.

Якщо заданий параметр BINARY Base64, то всі двійкові дані, які повертає запит, представляються у форматі base64. Для того, щоб одержувати двійкові дані в режимах RAW і EXPLICIT, необхідно задати саме цей параметр. У режимі AUTO двійкові дані, за замовчуванням, повертаються у вигляді посилання.

Слід зазначити, що можливості SQL для роботи з даними у форматі XML не обмежуються варіаціями представлення результатів запитів на отримання даних з реляційних таблиць. У сучасних СУБД передбачена також можливість за допомогою засобів SQL діставати доступ до даних, представлених у форматі документа XML. Доступ до даних XML здійснюється за допомогою функції OPENXML.

Функція OPENXML – це ключове слово мови SQL, яке дозволяє одержати набір кортежів, сформований на основі XML-документів, що зберігаються на зовнішній пам'яті хосту. OPENXML забезпечує доступ до даних XML так, як начебто вони представлені реляційним набором кортежів. Для цього на основі внутрішньої структури документа XML формується його інтерпретація у вигляді набору кортежів. Записи такого набору кортежів можна зберігати в таблицях бази даних.

Щоб створити запит до документа XML із застосуванням функції OPENXML, необхідно викликати системну збережену процедуру, яка виконує синтаксичний аналіз документа XML і повертає його описувач. Описувач документа містить вузли документа XML, представлені у вигляді дерева. Він передається функції OPENXML, яка після цього на основі переданих параметрів генерує представлення документа у вигляді набору кортежів. Синтаксис функції OPENXML виглядає наступним чином:

OPENXML (idoc, rowpattern, flags)
WITH (Оголошення схеми | Ім'я таблиці)

Призначення аргументів функції OPENXML наступне:

- idoc – описувач внутрішнього представлення документа XML. Внутрішнє представлення документа XML створюється при виклику відповідної системної збереженої процедури;
- rowpattern – вираз XPath визначає набір вузлів документа XML. Кожен визначений цим шаблоном вузол відповідає одному кортежу з набору, що генерується функцією OPENXML;
- flags – тип відповідності між атрибутами набору кортежів і вузлами XML. Ця інформація використовується для трансформації вузлів XML в атрибути набору кортежів. Якщо відповідність не

задано, то, за замовчуванням, передбачається відповідність, орієнтована на атрибути.

Для генерації набору кортежів OPENXML необхідно надати схему їх набору. Її задають за допомогою необов'язкової конструкції WITH. У випадку, якщо конструкція WITH не використовується, функція OPENXML повертає набір кортежів у форматі таблиці зрізів. Результат називається таблицею зрізів, оскільки при використанні цього формату кожен зріз у дереві проаналізованого документа XML відповідає кортежу вихідного набору. Якщо існує таблиця, схему якої можна використовувати, то замість оголошення схеми в конструкції WITH вказують ім'я цієї таблиці.

4.6. Основні способи модифікації даних в SQL

Модифікація даних у реляційних базах полягає в передачі СУБД запитів SQL на додавання, зміну та видалення вмісту базових таблиць. Розглянемо детальні можливості виконання кожної з цих дій.

Додавання даних. Сучасні СУБД підтримують декілька методів додавання інформації в базу даних:

- за допомогою оператора INSERT;
- за допомогою оператора SELECT...INTO;
- за допомогою операцій масового копіювання.

Оператор INSERT додає в таблицю один або декілька кортежів. У простому випадку оператор INSERT має наступний вигляд:

INSERT INTO таблиця (список атрибутів) значення

Цей оператор додає в задану таблицю дані (значення) у вигляді одного або декількох кортежів. Список імен атрибутів (список атрибутів), розділених комами, задає атрибути для розміщення даних. Якщо атрибути не задані, дані заносяться в усі атрибути новоствореного кортежу таблиці. Якщо задана лише частина списку, то в усі атрибути, не зазначені в списку, буде вставлене порожнє значення або значення за замовчуванням (якщо існує визначення DEFAULT). Тому усі незазначені атрибути повинні допускати порожні значення або в них має бути визначено значення за замовчуванням. Крім того, оператор INSERT не дозволяє задавати значення для атрибутів з властивістю IDENTITY, оскільки СУБД генерує ці значення автоматично. Ключове слово INTO в операторі INSERT необов'язкове і використовується лише для того, щоб зробити текст програми більш зрозумілим.

Визначаючи оператора INSERT, можна задати значення за допомогою конструкції VALUES або підзапиту SELECT. Конструкція VALUES дозволяє задавати значення в одному кортежі таблиці. Значення вка-

зують у вигляді списку скалярних виразів, розділених комами. Тип даних, точність цілої і дробової частини цих виразів повинні співпадати з аналогічними параметрами списку атрибутів або допускати неявне перетворення. Якщо список атрибутів не заданий, значення повинні бути вказані в тій самій послідовності, що й атрибути таблиці. Наприклад:

INSERT INTO Співробітник (СпівробітникІдн, Зарплата, Посада)
VALUES (5, 3000, 'Бухгалтер')

Підзапит SELECT в операторі INSERT дозволяє додати до таблиці дані з однієї або декількох інших таблиць, причому декілька кортежів одночасно. Цей спосіб застосовується для додавання до таблиці існуючих даних, на відміну від конструкції VALUES, що використовується в операторі INSERT для додавання до таблиці нових даних. Наприклад, у базі даних банківської установи, в якій існують таблиці "Операція" (для обліку фінансових операцій) та "Рахунок" (для обліку клієнтських коштів), виникла необхідність додати дані з першої таблиці в другу:

INSERT INTO Рахунок (КлієнтІдн, ВидОперації)
SELECT DISTINCT КлієнтІдн, ВидОперації
FROM Операція
WHERE ВидОперації = 3

Додавання даних у реляційні бази за допомогою оператора SELECT...INTO полягає в можливості оператора SELECT з використанням конструкції INTO створити нову таблицю і заповнити її результатами запиту (див. п. 4.1).

Додавання даних у реляційні бази відбувається за допомогою операцій масового копіювання, що дозволяють вставляти в таблицю, а також отримувати з таблиці велику кількість кортежів. Масове копіювання здійснюється на основі використання спеціалізованих програмних утиліт та засобів СУБД і являє собою найбільш швидкий спосіб додавання кортежів до таблиць реляційних баз даних.

Модифікація даних. Після створення таблиці та її заповнення дані можна змінювати або оновлювати. Сучасні СУБД підтримують декілька методів зміни даних в існуючій таблиці:

- оператор UPDATE;
- API бази даних та курсори.

Оператор UPDATE здатний змінювати дані в одному кортежі, групі кортежів або в усіх кортежах таблиці. Оператор UPDATE може змінювати в один момент часу дані тільки в одній таблиці. Оновлення пройде успішно тільки в тому випадку, якщо нове значення сумісне з

типом даних цільового атрибуту і відповідає всім обмеженням, що накладені на нього.

Оператор UPDATE складається з наступних основних конструкцій: SET, WHERE, FROM. SET задає атрибути, які слід змінити, та їх нові значення. В усіх кортежах, які відповідають умові пошуку, заданій конструкцією WHERE, значення заданих атрибутів оновлюються значеннями, заданими в конструкції SET. Якщо конструкція WHERE не задана, оновлюються всі кортежі. Наприклад, для підвищення заробітної плати усім співробітникам на 10 відсотків можна використати наступний запит:

UPDATE Співробітник
*SET Зарплата = Price * 1.1*

У цьому операторі не використовується конструкція WHERE, тому оновлюються всі кортежі таблиці. Конструкція WHERE задає кортежі, які підлягатимуть оновленню. Наприклад, для призначення розміру заробітної плати конкретному співробітнику можна використати наступний запит:

UPDATE Співробітник
SET Зарплата = 4500
WHERE СпівробітникІдн = 5

За допомогою конструкції FROM можна замінити дані в одній таблиці в залежності від даних, що містяться в інших таблицях. Наприклад, для коригування залишків на рахунках клієнтів банку на основі сум фінансових операцій можна використати наступний запит:

UPDATE Рахунок
SET Залишок = Операція.Сума
FROM Рахунок INNER JOIN Операція
ON Рахунок.КлієнтІдн = Операція.КлієнтІдн

Цей оператор змінює значення атрибуту “Залишок” у таблиці “Рахунок” значеннями відповідного атрибуту “Сума” з таблиці “Операція”.

Для взаємодії прикладних програм з базами даних найчастіше застосовують технології ADO, OLE DB і ODBC. Програмний інтерфейс API цих технологій підтримує оновлення поточного кортежу в результуючому наборі, який отримав програмний додаток. Крім того, при використанні серверного курсору можна оновлювати поточний кортеж за допомогою оператора UPDATE, до складу якого входить конструкція WHERE CURRENT OF. Зміни будуть відноситись тільки до того кортежу, на який встановлено курсор. Наприклад, для коригу-

вання дати народження в таблиці “Людина”, для якої створено курсор “КурсЛюдина” та зроблено поточним відповідний кортеж, можна використати наступний запит:

UPDATE Людина
SET [Дата народження] = ’5-12-1986’
WHERE CURRENT OF КурсЛюдина

Видалення даних. Сучасні СУБД підтримують декілька методів видалення даних з реляційних таблиць:

- оператор DELETE;
- API і курсори;
- оператор TRUNCATE TABLE.

Оператор DELETE видаляє з таблиці один або декілька кортежів. У загальному вигляді синтаксис оператора DELETE виглядає наступним чином:

DELETE таблиця FROM похідна таблиця WHERE умова пошуку.

Ім’я таблиці, звідки потрібно видалити кортежі, вказується відразу після оператора DELETE. Видаляються всі кортежі таблиці, які відповідають умові пошуку, що задається конструкцією WHERE. Якщо конструкція WHERE не задана, то видаляються всі кортежі таблиці. Конструкція FROM задає додаткові таблиці, тобто умови сполучення кортежів, що можна використовувати в предикатах умови пошуку конструкції WHERE. Кортежі з названої в конструкції FROM таблиці не видаляються. Наприклад, якщо в таблицях “Людина” та “Співробітник” зберігаються взаємопов’язані дані, то можна видалити дані з однієї на основі даних іншої, використовуючи наступний запит:

DELETE Співробітник
FROM Людина
WHERE Співробітник.ЛюдинаІдн = Людина.ЛюдинаІдн
AND Людина.Прізвище = ’Симоненко’ AND Людина.[Ім’я] = ’Іван’

Видалення даних за допомогою API і курсорів ґрунтується на застосуванні технологій OLE DB, ADO та ODBC, що підтримують видалення поточного кортежу в результуючому наборі, що був переданий програмному додатку. Крім того, СУБД може використовувати конструкцію WHERE CURRENT OF, що була задіяна в переданому на виконання операторі DELETE, для видалення кортежу, на якому на момент видалення встановлений курсор.

Для здійснення швидкого видалення з таблиці усього набору атрибутів використовується оператор TRUNCATE TABLE. Цей метод

практично завжди працює швидше за оператор DELETE, навіть якщо у нього не задані умови. Своїй швидкості він завдячує тому факту, що оператор DELETE протоколює видалення кожного кортежу. А оператор TRUNCATE TABLE негайно звільняє все місце, зайняте даними таблиці. Наприклад:

TRUNCATE TABLE Операція

Як і у випадку оператора DELETE, після застосування оператора TRUNCATE TABLE визначення таблиці з її асоційованими об'єктами залишається в базі даних. Для фізичного видалення таблиці слід використовувати оператор DROP TABLE.

ПИТАННЯ ТА ЗАВДАННЯ

1. Яке основне призначення оператора SELECT?
2. Які головні властивості результуючого набору описує більшість операторів SELECT?
3. Напишіть загальну структуру конструкцій оператора SELECT.
4. Які конструкції оператора SELECT є обов'язковими та з якою метою вони використовуються?
5. Які конструкції оператора SELECT використовуються як фільтр для побудови результуючого набору?
6. Яка конструкція оператора SELECT застовується для сортування результуючого набору?
7. Що таке підзапити і для чого вони використовуються?
8. Напишіть, в яких форматах зазвичай використовуються оператори, у складі яких є підзапит.
9. Що таке з'єднання та в яких конструкціях їх можна задавати?
10. Охарактеризуйте основні види з'єднань та способи їх програмування.
11. Дайте визначення індексу.
12. Що таке "B-дерево" та від чого залежать його ширина та глибина?
13. Охарактеризуйте типи індексів.
14. Опишіть способи створення індексів.
15. Наведіть приклади запитів, ефективність яких дозволяє підвищити індексація.
16. Дайте визначення курсору.
17. Підтримку яких функцій забезпечують курсори?
18. Які типи курсорів підтримують сучасні СУБД?
19. Опишіть порядок використання операторів при роботі з курсорами SQL для оголошення, заповнення й отримання даних.
20. Яким чином можна отримати результати SQL-запиту у вигляді документа XML?

21. Назвіть методи додавання інформації в базу даних.
22. Опишіть призначення, структуру та порядок застосування оператора INSERT.
23. Назвіть методи модифікації даних у базових таблицях.
24. Опишіть призначення, структуру та порядок застосування оператора UPDATE.
25. Опишіть призначення, структуру та порядок застосування операторів, що видаляють дані з базових таблиць.

ТЕСТИ

1. Яка з конструкцій оператора SELECT є обов'язковою?
   а) FROM;
   б) WHERE;
   в) ORDER BY.
2. Яка з наступних конструкцій оператора SELECT не може виконувати роль фільтра?
   а) WHERE;
   б) HAVING;
   в) ORDER BY.
3. Який з наступних типів з'єднань поверне всі записи таблиць 1 і 2?
   а) Таблиця1 INNER JOIN Таблиця2;
   б) Таблиця1 LEFT OUTER JOIN Таблиця2;
   в) Таблиця1 FULL OUTER JOIN Таблиця2;
   г) Таблиця1 RIGHT OUTER JOIN Таблиця2.
4. Для якого з представлених атрибутів оператор INSERT не дозволяє задавати значення?
   а) атрибути з властивістю IDENTITY;
   б) атрибути, що допускають значення NULL;
   в) атрибути, в яких визначено значення DEFAULT.
5. В якій кількості кортежів, що додаються до реляційної таблиці, конструкція VALUES дозволяєт задавати значення атрибутів?
   а) в одній;
   б) у кількості, визначеній умовою WHERE;
   в) у кількості, визначеній підзапитом SELECT.
6. В якому випадку один оператор UPDATE оновлює всі дані таблиці?
   а) таке оновлення неможливе;
   б) якщо відсутня конструкція WHERE;
   в) якщо в конструкції SET застосовується ключове слово ALL.
7. Який із способів видалення даних з реляційної таблиці вважається найшвидшим?
   а) API і курсори;

б) оператор DELETE;
в) оператор TRUNCATE TABLE.

8. У чому полягає основне призначення курсорів?
   а) у компіляції сукупності операторів SQL в єдиний план виконання;
   б) у скороченні мережевого трафіка, пов'язаного з вибіркою кортежів з таблиць віддаленого сервера баз даних;
   в) у реалізації механізму, що дозволяє клієнтським додаткам працювати з невеликими блоками кортежів.
9. Коли відбувається заповнення серверного курсору SQL?
   а) при виконанні оператора OPEN;
   б) при виконанні оператора DEALLOCATE;
   в) при виконанні оператора DECLARE CURSOR.
10. Яка функція забезпечує доступ до даних XML?
    а) GETXML;
    б) OPENXML;
    в) XMLDATA.

РОЗДІЛ 5. МОВА ПРОЦЕДУРНОГО ПРОГРАМУВАННЯ В SQL

5.1. Програмування збережених процедур

У попередніх розділах було розглянуто питання створення та виконання програмних пакетів у вигляді сценаріїв, написаних мовою SQL. При виконанні сценаріїв СУБД обробляє команди, які вони вміщують, з метою створення результуючих наборів або маніпулювання інформацією, що зберігається в базі даних. Існує інший спосіб збереження сценарію SQL – привласнити йому ім'я і зберегти у вигляді процедури. Отже, збережена процедура – це відкомпільований сценарій послідовного виконання операторів SQL, який зберігається на сервері баз даних під унікальним ім'ям.

Збережені процедури дозволяють підвищити продуктивність бази даних, розширюють можливості програмування запитів і підтримують функції безпеки, які недоступні при використанні команд SQL, що безпосередньо передаються для обробки на сервер. Продуктивність бази даних підвищується за рахунок локального (по відношенню до бази даних) зберігання програмного сценарію, прекомпіляції початкового тексту та кешування.

У процесі передачі кожної команди (або пакета команд) SQL на сервер баз даних для обробки СУБД повинна визначити, чи є у відправника права на виконання команд і чи допустимі самі команди. Перевіривши права доступу і синтаксис команд, SQL Server будує план виконання запиту. Збережені процедури у даному випадку більш ефективні, оскільки зберігаються на сервері баз даних, і при запиті відразу передаються на виконання. Тобто один оператор, переданий на сервер баз даних, дозволяє викликати складний сценарій SQL, який міститься в збереженій процедурі, що дозволяє уникнути передачі через мережу великої кількості команд.

Перед створенням збереженої процедури її команди проходять синтаксичну перевірку. Під час першого запуску збереженої процедури створюється план її виконання, і збережена процедура компілюється. Надалі її обробка здійснюється швидше, оскільки СУБД не доводиться перевіряти синтаксис команд, створювати план виконання і компілювати текст процедури. До створення нового плану в кеші СУБД перевіряє наявність існуючого плану виконання.

Створену збережену процедуру можна, за необхідності, викликати в будь-який момент. Це забезпечує модульність і стимулює повто-

рне використання програмного коду. Останнє полегшує супровід бази даних, оскільки вона ізольована від бізнес-правил, які часто змінюються. Модифікувати збережену процедуру відповідно до нових правил можна у будь-який момент часу. Після цього усі програмні додатки, що її використовують, автоматично прийдуть у відповідність з новими бізнес-правилами без необхідності безпосередньої модифікації.

Подібно до програм, написаних іншими мовами програмування, збережені процедури здатні приймати вхідні параметри, повертати значення вихідних параметрів, підтримувати зворотний зв'язок з користувачем за допомогою повернення кодів стану і текстових повідомлень, а також викликати інші процедури. Наприклад, одна збережена процедура може повертати код стану, залежно від якого інша процедура, що її викликала, виконує ті або інші дії.

Інше важливе призначення збережених процедур полягає у підвищенні безпеки за допомогою ізоляції і шифрування програмної логіки. Користувачам можна надати право на виконання збереженої процедури без безпосереднього доступу до об'єктів бази даних, з якими вона працює. Крім того, якщо збережену процедуру зашифрувати при створенні або модифікації, користувачам не вдасться прочитати команди SQL, що її утворюють. Такі заходи безпеки дозволяють ізолювати від користувача структуру бази даних, що сприяє забезпеченню цілісності даних і надійності бази.

Збережені процедури поділяються на наступні категорії: системні, локальні, тимчасові, розширені.

Системні збережені процедури. Системні збережені процедури знаходяться в системній базі даних екземпляра сервера баз даних. У MS SQL – це база даних Master. Як правило, їх імена починаються з префікса sp_. Вони призначені для підтримки функцій СУБД. До них належать: вибірка даних з системних таблиць програмними додатками, адміністрування бази даних і управління безпекою.

Локальні збережені процедури. Локальні збережені процедури зберігаються в користувальницьких базах даних. Як правило, їх створюють для вирішення певних задач у базі даних. Локальні збережені процедури також дозволяють розширити функціональність системних збережених процедур. Для того, щоб створити на основі системної збереженої процедури локальну збережену процедуру, потрібно зробити копію системної процедури, зберегти її як локальну процедуру, а потім зробити необхідні зміни.

Тимчасові збережені процедури. Тимчасова збережена процедура схожа на локальну. Але вона існує лише до закриття з'єднання, в якому створена, або до завершення роботи сервера баз даних. Тимчасові збережені процедури знаходяться в тимчасовій базі даних, яка

створюється наново при кожному запуску сервера баз даних і видаляється з усім вмістом після завершення його роботи. Для MS SQL ім'я такої бази – TempDB. Тимчасові збережені процедури корисні при роботі з ранніми версіями СУБД, які не підтримують повторне використання планів виконання, а також у тих випадках, коли немає сенсу зберігати процедуру, оскільки значення її команд постійно змінюється.

Розширені збережені процедури. У випадку, якщо розробникам ІС необхідно використати складні програми, написані іншими мовами програмування (C++, JAVA і ін.), ці програми можна викликати із СУБД засобами збережених процедур особливого типу, які називаються розширеними збереженими процедурами. Розширені збережені процедури звертаються до зовнішніх програм, скомпільованих у вигляді 32-розрядних динамічних бібліотек програмного коду DLL.

Для створення збережених процедур використовують оператор CREATE PROCEDURE. Наприклад, для підвищення заробітної плати усім співробітникам у 2 рази можна створити збережену процедуру з використанням наступного запиту:

```
CREATE PROCEDURE ЗбільшитиЗарплатуВ2Рази
AS
UPDATE Співробітник
SET Зарплата = Зарплата*2
```

Щоб створити тимчасову збережену процедуру, слід додати до імені символ #. Цей знак повідомляє СУБД, що створити процедуру треба в тимчасовій базі даних.

Під час створення збереженої процедури СУБД перевіряє синтаксис операторів SQL, які її утворюють. При виявленні синтаксичної помилки вона генерує повідомлення про помилку і процедура не створюється. Якщо текст процедури проходить синтаксичну перевірку, то процедура зберігається. При цьому її ім'я та інша інформації (наприклад, ідентифікаційний номер, що автоматично генерується) записується в системні таблиці.

Під час першого виконання збереженої процедури план її виконання кешується за замовчуванням. Після цього кешування не виконується до перезапуску сервера баз даних або модифікації структури таблиці, що лежить в основі збереженої процедури. У деяких випадках кешування плану виконання збереженої процедури є небажаним. Наприклад, якщо параметри збереженої процедури істотно варіюються від запуску до запуску, то кешування її плану виконання дає зворотний результат замість очікуваного підвищення продуктивності. Для того, щоб викликати перекомпіляцію збереженої процедури при кожному її виконанні під час створення, слід додати до процедури ключові слова WITH RECOMPILE.

Як зазначалося раніше, шифрування збереженої процедури запобігає перегляду її вмісту. Для того, щоб зашифрувати збережену процедуру, треба створити її з ключовими словами WITH ENCRYPTION. Зашифровану збережену процедуру не можна переглядати, тому необхідно зберігати в надійному місці її програмний код в незашифрованому вигляді для забезпечення можливості її модифікації.

Під час створення збереженої процедури СУБД не перевіряє існування об'єктів, на які посилається процедура. Така можливість необхідна, оскільки іноді об'єкт (наприклад, таблиця), на який посилається збережена процедура, не існує на момент створення процедури. Ця особливість називається відкладеним дозволом імен. Об'єкт перевіряється вже при виконанні збереженої процедури. При посиланні в збереженій процедурі на об'єкт (наприклад, на таблицю), бажано указувати його власника. Більшість сучасних СУБД припускають за замовчуванням, що творець збереженої процедури одночасно є власником об'єктів, на які вона посилається.

Запуск на виконання збереженої процедури здійснюється або вручну або автоматично (наприклад, при запуску сервера). Перед виконанням процедури слід задати значення всіх необхідних її параметрів. Для виконання служить ключове слово EXECUTE (або його скорочена версія – EXEC), яке має передувати її повному імені у форматі:

EXEC ім'я бази даних. власник. ім'я процедури

У випадку, коли база даних, в якій міститься збережена процедура, поточна, то для виклику процедури достатньо вказати частину імені: власника та ім'я процедури. Якщо ім'я процедури унікальне в активній базі даних, то можна використовувати просто ім'я процедури.

Якщо при створенні збереженої процедури були задані необхідні вхідні параметри, то при виконанні процедури необхідно задати їх значення. Визначення вхідних і вихідних параметрів починаються із знака "at" (@), після якого йде ім'я параметра, опис його типу даних. Під час виклику процедури потрібно задати значення параметра і (необов'язково) його ім'я.

На практиці часто доводиться модифікувати збережені процедури. Наприклад, щоб додати параметр або змінити ту чи іншу команду. Модифікація процедури, замість видалення і повторного створення "з нуля", дозволяє заощадити час, оскільки при модифікації зберігається багато властивостей збережених процедур. Для модифікації процедур використовуються ключові слова ALTER PROCEDURE.

Для видалення збережених процедур використовують оператора DROP PROCEDURE або його скорочену версію DROP PROC. Наприклад:

DROP PROCEDURE ЗбільшитиЗарплатуВ2Рази

Не слід видаляти збережену процедуру до тих пір, доки всі залежні від процедури об'єкти не будуть видалені або модифіковані (щоб ліквідувати залежність).

Збережені процедури істотно розширюють можливості програмування мовою SQL за рахунок їх динамічності. Динамічність збережених процедур досягається, здебільшого, завдяки використанню вхідних та вихідних параметрів. Вхідні параметри дозволяють користувачу передавати процедурі різні значення та одержувати у кожному випадку відповідні результати. Вихідні параметри дозволяють одержувати не тільки результуючий набір даних, але і додаткові відомості. Під час виконання процедури значення вихідних параметрів зберігаються в пам'яті. Для того, щоб отримати значення вихідного параметра, необхідно створити змінну для його зберігання. Це значення в подальшому можна відобразити за допомогою команд SELECT і PRINT або використати його для виконання інших команд процедури.

Вхідний параметр визначають при оголошенні збереженої процедури, а його значення задають при описі логіки її виконання. Вихідний параметр збереженої процедури визначається за допомогою ключового слова OUTPUT. Під час виконання процедури значення вихідного параметра зберігається в пам'яті. Як правило, вихідні значення використовуються після завершення виконання процедури. Застосування вхідних і вихідних параметрів ілюструє текст процедури "ЗарплатаСпівробітника", яка повертає у вигляді вихідного параметра значення зарплати співробітника відповідно введеним як вхідним параметрам персональним даним:

```
CREATE PROCEDURE ЗарплатаСпівробітника
@Парам1 varchar(10),
@Парам2 varchar(10),
@Парам3 varchar(10),
@Зарплата int OUTPUT
AS
SET @Зарплата =
        (SELECT Зарплата
        FROM Співробітник
        WHERE СпівробітникІдн =
                ( SELECT СпівробітникІдн
                FROM Людина
                WHERE Прізвище = @Парам1 AND
                  [Ім'я] = @Парам2 AND
                  [По батькові] = @Парам3))
```

Часто основні зусилля при програмуванні якісних збережених процедур (власне, як і програм іншого спрямування), витрачаються на реалізацію обробки помилок. Сучасні СУБД надають програмістам функції і операторів для обробки помилок, що виникають під час виконання процедури. Помилки можна розділити на дві основні категорії: пов'язані з комп'ютерами (наприклад, недоступний сервер баз даних) і користувальницькі. Для обробки помилок, що виникають під час виконання процедур, використовуються коди повернення і функція @@ERROR.

Коди повернення можуть бути використані не тільки для обробки помилок. Оператор RETURN застосовується для генерації кодів повернення і виходу з процедури, відповідно до ситуації. Він може повернути програмі, що викликала процедуру, встановлене числове значення. Оператор RETURN використовується головним чином для обробки помилок, оскільки при виконанні цього оператора виконується безумовний вихід з процедури.

Щоб переконатися, чи задане значення вхідного параметра, необхідно встановити для нього значення за замовчуванням. Встановлення значень за замовчуванням для вхідних параметрів збережених процедур являє собою потужний засіб, що дозволяє уникати помилок. Для цього часто застосовується невизначене значення NULL. Наприклад, у процедурі "ЗарплатаСпівробітника" можна здійснити попереджувальні заходи від передачі некоректних вхідних параметрів:

```
CREATE PROCEDURE ЗарплатаСпівробітника
@Парам1 varchar(10) = NULL,
@Парам2 varchar(10) = NULL,
@Парам3 varchar(10) = NULL,
@Зарплата int OUTPUT
AS
IF @Парам1 IS NULL OR @Парам2 IS NULL
RETURN(1)
SET @Зарплата =
      (SELECT Зарплата
      FROM Співробітник
      WHERE СпівробітникІдн =
            ( SELECT СпівробітникІдн
            FROM Людина
            WHERE Прізвище = @Парам1 AND
             [Ім'я] = @Парам2 AND
             [По батькові] = @Парам3)
```

Інша важлива категорія помилок, які необхідно обробляти – помилки бази даних. Функція @@ERROR дозволяє виявити значну кількість різних помилок бази даних. Вона фіксує номери помилок упродовж виконання процедури. Кожен номер, який повертає функція, можна використовувати для виведення повідомлення, що повідомить користувача про помилку та розкриває її причини. В операторах INSERT і UPDATE часто припускаються користувальницьких помилок, які призводять до виникнення помилок бази даних. Як правило, у цих операторах користувач намагається ввести інформацію, що порушує цілісність даних (наприклад, недійсний ідентифікаційний номер). Якщо процедура виконана успішно, значення функції @@ERROR дорівнює 0. При виникненні помилки функція @@ERROR повертає номер помилки, відмінний від 0. Значення функції @@ERROR змінюється при виконанні кожного оператора SQL.

Під час програмування збережених процедур особливу увагу слід звернути на їх вкладеність. Під вкладеністю збережених процедур розуміють виклик однієї процедури з іншої. Одна збережена процедура здатна виконувати декілька задач, проте краще створювати простіші процедури, які для виконання додаткових задач викликають інші процедури. Глибина вкладеності процедур кінцева (наприклад, у MS SQL вона досягає 32 рівні). Проте кількість процедур, що викликаються однією процедурою, зазвичай, необмежена. Процедура також здатна рекурсивно викликати сама себе. Для виклику однієї процедури з іншої служить оператор EXECUTE.

У складі збережених процедур дозволяється використовувати курсори. Такий підхід може бути застосований для реалізації результуючого набору, що містить більше одного кортежу. У цьому випадку створюється вихідний параметр, для якого визначений тип даних cursor. Після ключового слова AS цьому параметру присвоюється значення курсора. У результаті цей параметр одержує усі кортежі, які повернув оператор SELECT у складі курсору. Але цього слід уникати, якщо є можливість виконати ту ж саму задачу за допомогою результуючих наборів. Процедури, що використовують результуючі набори, більш ефективні, і, як правило, їх простіше писати, ніж процедури, що використовують курсори. Для реалізації результуючого набору, що містить більше одного кортежу, потрібно використовувати оператор SELECT без вихідних параметрів, оскільки вихідний параметр оголошується як змінна і може містити лише одне значення.

5.2. Програмування тригерів

При оцінці бази даних, як правило, звертають увагу на узгодженість і точність інформації, яку вона вміщує. Для підтримки узгодже-

ності і точності використовуються декларативна та процедурна цілісності даних. Методи декларативної цілісності даних були розглянуті в минулих розділах. Наразі настав час розглянути один з найпоширеніших методів забезпечення процедурної цілісності реляційних даних, що забезпечується за рахунок використання тригерів.

Тригери – це особливий клас збережених процедур, що автоматично виконуються під час модифікації даних або після неї. Технологія тригерів дозволяє програмувати процедури, що викликаються при модифікації табличних даних у реляційних базах. Вони дозволяють реалізувати в базі даних складні процедурні методи підтримки цілісності даних та бізнес-логіку.

Завжди потрібно пам'ятати, що перш ніж реалізовувати тригер, слід з'ясувати, чи не можна одержати аналогічні результати за рахунок використання обмежень або правил, оскільки використання великої кількості тригерів здатне понизити продуктивність бази даних. Тригери корисні тоді, коли необхідно описувати значно складнішу логіку в порівнянні з тією, що доступна методам декларативної цілісності. Традиційно тригери застосовуються в наступних випадках:

- якщо використання методів декларативної цілісності даних не відповідає функціональним потребам ІС;
- якщо необхідна каскадна зміна в непов'язаних таблицях бази даних;
- якщо база даних ненормалізована і потрібен спосіб автоматизованого оновлення надмірних даних у декількох таблицях;
- якщо необхідно звірити значення в одній таблиці з неоднорідними значеннями в іншій таблиці;
- якщо потрібне виведення призначених для користувача повідомлень і складна обробка помилок.

Застосування каскадної посилальної цілісності здатне розширити можливості забезпечення точності й узгодженості даних, змінюючи та видаляючи дані, що містяться в атрибутах із зовнішніми ключами в базових таблицях. Тригери збільшують ці можливості, забезпечуючи захист цілісності даних в атрибутах будь-якої таблиці бази даних і навіть в об'єктах, розташованих за межами поточної бази даних.

Один тригер здатний виконувати декілька дій і реагувати на декілька різних подій. Автоматичне спрацьовування тригера викликають три події: INSERT, UPDATE і DELETE, які відбуваються в таблиці. Тригери не можна запустити вручну. У синтаксисі тригерів перед фрагментом програми, що формує виконувану тригером задачу, завжди визначається одна або декілька таких подій. Типи тригерів відповідають цим подіям. Наприклад, можна створити тригер на оновлення, що спрацьовує при зміні даних таблиці. Один тригер дозволяється за-

програмувати для реакції на декілька подій, тому нескладно створити процедуру, яка одночасно є тригером на зміну та додавання даних. Порядок переліку цих подій у визначенні тригера є довільним.

Існують певні обставини, за яких подія модифікації або видалення даних не викликає спрацьовування відповідного тригера. Наприклад, при виконанні оператора TRUNCATE TABLE не відбувається спрацьовування DELETE-тригера. Головною особливістю тригерів є автоматичний відкат невдалих трансакцій. Оскільки TRUNCATE TABLE – непротокольована подія, її відкіт неможливий і тому вона не викликає спрацьовування DELETE-тригера.

У сучасних СУБД реалізовані два класи тригерів: INSTEAD OF та AFTER. Перші виконуються в обхід дій, що викликали їх спрацьовування, замінюючи ці дії. Наприклад, оновлення таблиці, в якій є тригер INSTEAD OF, викличе спрацьовування цього тригера. У результаті замість оператора UPDATE виконується код тригера. Це дозволяє розміщувати в тригері складні оператори обробки даних, які доповнюють дії оператора, що модифікує таблицю. AFTER-тригери виконуються після дії, що викликала спрацьовування тригера. Вони вважаються класом тригерів за замовчуванням.

До таблиці дозволено прив’язувати тригери обох класів. Якщо в таблиці визначені обмеження і тригери обох класів, то першим з них спрацьовує тригер INSTEAD OF, потім обробляються обмеження, і останнім спрацьовує AFTER-тригер. При порушенні обмеження виконується відкат дій INSTEAD OF-тригера. Якщо порушуються обмеження або відбуваються інші події, що не дозволяють модифікувати таблицю, AFTER-тригер не виконується.

Для створення тригерів використовують оператор CREATE TRIGGER. Основні конструкції оператора CREATE TRIGGER наступні:

CREATE TRIGGER назва тригера
ON назва таблиці
FOR | INSTEAD OF клас тригера та тип(и) тригера
AS оператори SQL

Перед створенням тригера необхідно вибрати потрібну базу даних (за допомогою конструкції USE, ім’я бази даних і ключового слова GO). Право на створення тригерів за замовчуванням належить власнику таблиці, до якої він буде прикріплений. При створенні тригера для його ідентифікації після слів CREATE TRIGGER необхідно вказати його унікальне найменування. Імена тригерів мають відповідати правилам, визначеним для ідентифікаторів. Наприклад, щоб створити тригер “Перевірка Співробітника”, слід помістити його ім’я в квадратні дужки:

CREATE TRIGGER [Перевірка Співробітника]

Конструкція ON використовується для прив'язки тригера до таблиці. Таблиця після прив'язки називається таблицею тригера. Наприклад, наступний код визначає прив'язку тригера "Перевірка Співробітника" до таблиці "Співробітник":

CREATE TRIGGER [Перевірка Співробітника]
ON Співробітник

Тригер можна прив'язати тільки до однієї таблиці. Для того, щоб прив'язати до іншої таблиці тригер, який виконує аналогічну задачу, слід створити новий тригер з іншим ім'ям, але з тією ж самою логікою і прив'язати його до іншої таблиці. AFTER-тригери (цей клас заданий за замовчуванням) дозволено прив'язувати тільки до таблиць, а тригери класу INSTEAD OF – як до таблиць, так і до представлень.

Під час створення тригера слід задати тип події, що викликає його спрацьовування. Як зазначалося вище, існує три типи подій: INSERT, UPDATE і DELETE. Один тригер може спрацювати на одну, дві або всі три події. У разі, якщо необхідно, щоб він спрацьовував на всі події, після конструкцій FOR, AFTER або INSTEAD OF слід помістити всі три ключові слова: INSERT, UPDATE і DELETE у будь-якому порядку. Наприклад:

CREATE TRIGGER [Перевірка Співробітника]
ON Співробітник
FOR INSERT, UPDATE, DELETE

Конструкція FOR – синонім AFTER. Тому приведений вище код створює AFTER-тригер. Для створення тригера INSTEAD OF необхідно замість конструкції FOR застосувати конструкцію INSTEAD OF.

Конструкція AS і розміщені за нею оператори SQL визначають задачу, для виконання якої тригер призначений. Наприклад, послати по електронній пошті повідомлення адміністратору, якщо в таблиці "Співробітник" відбуваються події INSERT, UPDATE або DELETE:

CREATE TRIGGER [Перевірка Співробітника]
ON Співробітник
FOR INSERT, UPDATE, DELETE
AS
EXEC master.. xp_sendmail 'admin@academy.sumy.ua',
'У таблиці Співробітник відбулись зміни '
GO

Для модифікації вмісту тригера його можна видалити і відтворити з новим набором операторів. Інший спосіб, без видалення, полягає

у використанні оператора ALTER TRIGGER. Синтаксис цього оператора аналогічний синтаксису оператора CREATE TRIGGER.

Іноді нові угоди про іменування вимагають перейменувати тригер. Це також буває необхідно при розробці для таблиці декількох тригерів. MS SQL для перейменування тригера застосовують системну збережену процедуру sp_rename, наприклад:

sp_rename @objname = [Перевірка Співробітника], @newname = Перевірка

Для перегляду вмісту тригерів застосовуються запити до відповідних атрибутів системних таблиць. В MS SQL для того, щоб побачити вміст тригера, необхідно зробити запит до атрибуту "Text" системної таблиці "SysComments".

Для видалення з бази даних одного або декількох тригерів застосовують оператор DROP TRIGGER. Наприклад:

DROP CREATE TRIGGER Перевірка

При видаленні тригера інформація про нього видаляється із системних таблиць. Якщо видаляється таблиця тригера, то всі прив'язані до них тригери також видаляються.

Можливо, під час виправлення помилок у базі даних, при перевірці внесених змін або в процесі створення нової процедури доведеться відключити один або декілька прив'язаних до таблиці тригерів. Для того, щоб відключити тригер, застосовується оператор ALTER TABLE. Наприклад, для відключення тригера "Перевірка" в таблиці "Співробітник" можна застосувати наступну команду:

ALTER TABLE Співробітник DISABLE TRIGGER Перевірка

Якщо потрібно відключити всі прив'язані до таблиці тригери, слід після конструкції DISABLE TRIGGER вказати ключове слово ALL. Для того, щоб включити один або декілька тригерів, слід замінити ключове слово DISABLE в операторі ALTER TABLE на ENABLE.

При спрацьовуванні тригера на події INSERT, UPDATE або DELETE створюється одна або декілька тимчасових таблиць (також відомих як логічні таблиці). Тимчасові таблиці можна розглядати як журнали трансакцій для події. Існують два типи тимчасових таблиць: Inserted і Deleted. Inserted створюється в результаті події додавання або зміни даних. У ній знаходиться набір кортежів, що додаються до таблиці або підлягають зміні. UPDATE-тригер створює також тимчасову таблицю Deleted. У ній знаходиться початковий набір кортежів таблиці в тому стані, в якому він був до операції зміни. Використовуючи ці дані, тригер може відмінити зміни і повернути початкові дані. При спрацьовуванні тригера на зміну даних набір кортежів, що вида-

ляються, поміщається в логічну таблицю Deleted. Нові дані заносяться в таблицю Inserted. Наприклад, наступний код створює тригер, що перевіряє наявність клієнта в таблиці співробітників (мається на увазі, що таке співпадіння є забороненим бізнес-правилами):

```
CREATE TRIGGER ПеревіркаКлієнта
ON Клієнт
FOR INSERT, UPDATE
AS
BEGIN TRANSACTION
DECLARE @ЛюдинаІдн INT
DECLARE @СпівробітникІдн INT
SELECT @ЛюдинаІдн = і.ЛюдинаІдн
FROM inserted і
SELECT @СпівробітникІдн = СпівробітникІдн
FROM Співробітник С
WHERE С.ЛюдинаІдн = @ЛюдинаІдн
IF (@СпівробітникІдн IS NOT NULL) AND (@СпівробітникІдн > 0)
BEGIN
RAISERROR (’Спроба вставити в таблицю Клієнт об’єкт,
        що присутній в таблиці Співробітник.’, 16, 1)
ROLLBACK TRANSACTION
END
ELSE
BEGIN
COMMIT TRANSACTION
END
```

Для того, щоб ідентифікувати в тригері атрибут, в якому відбулася зміна даних, використовується логічна конструкція:

IF UPDATE (найменування атрибуту)

Ця конструкція повертає true (або false), якщо в атрибуті, вказаному в дужках, відбулась (або не відбулась) подія INSERT або UPDATE. Якщо потрібно задати декілька атрибутів, слід розділити їх кількома такими конструкціями, наприклад:

```
IF UPDATE (Посада) OR UPDATE (Зарплата)
BEGIN
-- Якщо дані цих атрибутів змінюються,
-- виконати потрібні дії.
END
```

На момент прив’язки тригера до таблиці особлива властивість реляційних баз даних, що має назву “відкладений дозвіл імен”, допускає відсутність у таблиці атрибута, заданого параметром “наймену-

вання атрибута". Проте необхідно, щоб цей атрибут існував під час спрацьовування тригера.

У тригерах часто використовується функція @@ROWCOUNT. Вона повертає кількість кортежів, на яких вплинуло виконання попереднього оператора SQL. Тригер спрацьовує на подію INSERT, UPDATE або DELETE, навіть, якщо при цьому не змінюється жоден кортеж. Для завершення роботи тригера за відсутності модифікацій даних таблиці використовується системна команда RETURN. У разі виникнення помилки іноді потрібно вивести повідомлення з описом її причини. Для виведення повідомлень про помилки використовується системна команда RAISERROR.

Під час програмування тригерів можуть бути застосовані оператори SELECT для надання змінним потрібних значень. Але не рекомендується використовувати ці оператори для повернення результуючих наборів та вставлення значень у повідомлення, що виводяться. Зазвичай, спрацьовування тригера прозоре для користувача або програмного додатка. Одак, якщо в додатку не запрограмована обробка значень, що повертаються, наприклад результуючого набору оператора SELECT, то в роботі додатка може виникнути збій. Проте цілком допустиме використання оператора SELECT як постачальника значення для оператора перевірки умови. Можна застосувати оператор SELECT для перевірки існування деякого значення та передавання цього значення оператору IF EXISTS для подальшої обробки.

Існує певний перелік операторів SQL, використання яких під час програмування тригерів забороняється. Зокрема, у тригерах недопустимі наступні оператори: ALTER, CREATE, DROP, RESTORE, LOAD DATABASE.

На завершення цього розділу, узагальнюючи викладений матеріал, доречним буде систематизувати найпоширеніші задачі, що вирішуються за допомогою тригерів. Найчастіше тригери застосовують у наступних ситуаціях:

1. Розрахунок проміжних результатів та інших обчислюваних значень. База даних постійно змінюється при додаванні, видаленні та модифікації даних в її таблицях. У деяких випадках дані атрибута однієї таблиці можна розраховувати на основі даних іншої таблиці, що модифікуються. Тригери є ідеальним засобом реалізації обчислюваних атрибутів.
2. Створення записів аудиту. Для забезпечення безпеки або просто для відстеження операцій, що виконуються над даними таблиці, у базі даних можна створити тригер. Він збирає у спеціальній таблиці дані, що були додані, модифіковані або видалені з іншої таблиці.

3. Виклик зовнішніх дій. У тригері дозволяється задавати дії, що виходять за межі стандартної обробки бази даних. Наприклад, відправка поштового повідомлення при спрацьовуванні тригера.
4. Реалізація складного захисту цілісності даних. Інколи стандартних заходів захисту цілісності даних недостатньо. Наприклад, за допомогою операції каскадного видалення можна видалити записи з таблиць, тоді як просте видалення порушить посилальну цілісність між ними. Однак, каскадне видалення може виявитися небажаним. У цьому випадку замість нього для видалення записів із зв'язаних таблиць використовують INSTEAD OF-тригер, а видалені записи розміщують в іншій таблиці для подальшої перевірки.

5.3. Програмування представлень реляційних даних

Один з основних методів підвищення ефективності роботи інформаційних систем з базами даних полягає в застосуванні для доступу до реляційних даних спеціальних об'єктів – представлень. Представлення являють собою віртуальні таблиці, вміст яких визначається SQL-запитом. Подібно до базової таблиці представлення складається з атрибутів та кортежів, що містять дані. Проте ці дані не належать йому безпосередньо. У момент звернення до представлення відбувається виконання запиту, що покладений у його основу при створенні. Завдяки цьому запиту відбувається динамічне формування атрибутів і кортежів представлення.

Неважко здогадатися, що представлення формується з результуючого набору, що повертає оператор SELECT. Тому представлення можна розглядати як збережений запит, що створений з використанням цього оператора. Цей запит може посилатися на одну або декілька таблиць з поточної, або з інших баз даних. Представлення функціонує як з'єднувач та фільтр для таблиць, що лежать у його основі. В операторах SQL допустиме посилання на представлення як на базову таблицю. Тобто через представлення можна отримувати та модифікувати дані. Крім того, представлення може посилатися на інше (інші) представлення.

Для підвищення ефективності програмування складних SQL запитів та сценаріїв представлення забезпечують виконання декількох особливо корисних дій, а саме:

- обмеження доступної користувачу області таблиці визначеними кортежами та атрибутами;
- об'єднання атрибутів з декількох таблиць у вигляді єдиної таблиці;
- заміна детальних відомостей агрегованими.

На додачу представлення дозволяють секціонувати дані і розподіляти їх між декількома базами даних, що можуть знаходитися під управлінням різних екземплярів серверів баз даних. За допомогою секціонованих представлень навантаження по обробці даних розподіляється між декількома серверами, що складають одну групу. Сучасні СУБД також підтримують індексацію представлень, що дозволяє значно підвищити продуктивність складних представлень.

Існує безліч способів використання представлень для вибірки даних з реляційних таблиць. Крім того, що представлення допомагають користувачам відібрати потрібні дані, вони дозволяють також підвищити захист самих даних, оскільки користувачам видно лише дані представлення, а не самої таблиці, що лежить в його основі.

Представлення сприяють спрощенню маніпулювання даними. Визначивши часто використовувані з'єднання, об'єднання і SELECT-запити у вигляді представлень, користувачі позбавляються необхідності заново задавати всі умови та параметри при кожній операції над даними.

Представлення спрощує доступ до даних, оскільки при генерації кожного звіту не доводиться наново вводити запит, що визначає представлення, і відправляти його на сервер. Натомість, виконується запит до самого представлення.

Представлення дозволяють різним користувачам по-різному відображати одні й ті ж дані, навіть при одночасному зверненні. Перевага представлень особливо відчутна, коли до бази даних звертається декілька користувачів, що мають різні інтереси та досвід роботи. Наприклад, для менеджера можна створити представлення, яке надає відомості лише про тих клієнтів, з якими він безпосередньо має справу.

Представлення можна створювати шляхом об'єднання результатів двох та більше запитів до різних таблиць в єдиний результуючий набір, за допомогою оператора UNION. Користувач сприймає одержаний результат як одну таблицю, яка називається в цьому випадку секціонованим представленням. Уявімо, що в одній таблиці містяться дані з продажу в Донецьку, а в другій, аналогічній по структурі таблиці, – дані у Львові. Оператор UNION дозволяє створити на основі об'єднання цих таблиць представлення, що відображає обсяги продажів в обох регіонах.

Представлення створюються так само, як і інші об'єкти бази даних і, так само, їх потім дозволяється модифікувати та видаляти. Представлення створюють тільки в поточній базі даних. Проте якщо нове представлення визначене на основі розподілених запитів, воно може посилатися на таблиці і представлення з інших баз даних. При

створенні представлення необхідно, щоб його ім’я відповідало правилам для ідентифікаторів і було унікальним для кожного користувача.

Під час створення представлень діють наступні обмеження:

1. З представленнями не можна пов’язувати правила і визначення DEFAULT.
2. У запитах, що визначають представлення, не повинно бути конструкцій ORDER BY і ключових слів INTO.
3. Необхідно явно задати ім’я кожного атрибута представлення, якщо хоч один з них формується на основі арифметичного виразу, функції, константи, а також, якщо є загроза, що декілька атрибутів одержать однакові імена (останнє можливе при використанні у представленні з’єднання декількох таблиць, в яких імена атрибутів співпадають).

Створення стандартних представлень здійснюється засобами оператора CREATE VIEW. Наприклад:

```
CREATE VIEW СписокСпівробітників
AS
SELECT Л.Прізвище, Л.[Ім’я], С.Зарплата
FROM Людина AS Л INNER JOIN Співробітник AS С
ON Л.ЛюдинаІдн = С.ЛюдинаІдн
```

Процес побудови результуючого набору представлення називається матеріалізацією представлення. У стандартному представленні витрати системи на динамічну побудову результуючого набору для кожного запиту, що посилається на представлення, зростають при використанні складної обробки великої кількості кортежів (наприклад, агрегації множини даних). Якщо в запитах часто зустрічаються посилання на такі представлення, то для підвищення продуктивності можна створити для представлення індекс. Індекси будуть розглянуті детально в наступному підрозділі.

Підвищити ефективність доступу до реляційних структур можна шляхом створення секціонованих представлень. Секціоноване представлення сполучає набір горизонтальних фрагментів даних з таблиць-учасників, розташованих на одному або декількох серверах, що дозволяє представити їх у вигляді єдиної таблиці.

Сучасні СУБД підтримують два види секціонованих представлень: локальні та розподілені. У локальному секціонованому представленні всі базові таблиці і представлення розташовані в одному екземплярі сервера баз даних. У розподіленому секціонованому представленні вони розташовані на різних серверах. Наприклад, на сервері баз даних багатофіліального банку знаходяться бази даних філіалів. Можна створити представлення, яке відображає всю клієнтську базу:

```
CREATE VIEW СписокКлієнтів AS
SELECT *
FROM Department1.Owner.Клієнт
UNION ALL
SELECT *
FROM Department2.Owner.Клієнт
UNION ALL
SELECT *
FROM Department3.Owner.Клієнт
```

У разі потреби представлення можна змінити, не вдаючись до видалення і повторного створення, що призводить до втрати відповідних прав доступу. Зміна представлення не впливає на залежні від нього об'єкти (наприклад, на збережені процедури або тригери). Модифікувати представлення можна змінивши засобами оператора ALTER VIEW запит SELECT, що визначає представлення. Наприклад:

```
ALTER VIEW СписокСпівробітників
AS
SELECT Л.Прізвище, Л.[Ім'я], Л.[По батькові], С.Зарплата
FROM Людина AS Л INNER JOIN Співробітник AS С
ON Л.ЛюдинаІдн = С.ЛюдинаІдн
```

Якщо представлення більше непотрібне, або виникла необхідність відмінити визначення представлення і пов'язані з ним права доступу, його можна видалити. Видалення представлень не впливає на базові таблиці та їх дані. Видаляють представлення засобами оператора DROP VIEW, наприклад:

```
DROP VIEW СписокСпівробітників
```

Запрошувати дані через представлення можна без будь-яких обмежень. Представлення використовуються для отримання даних за допомогою оператора SELECT практично так само, як звичні таблиці. Наприклад:

```
SELECT *
FROM СписокСпівробітників
```

Практично в усіх сучасних СУБД за допомогою представлень дозволяється змінювати дані в базових таблицях. Тобто представлення можуть бути цільовими об'єктами операторів UPDATE, DELETE або INSERT. Модифікувати дані через представлення дозволяється з деякими обмеженнями. Тому перед модифікацією даних через представлення слід врахувати наступні правила:

1. Не можна модифікувати дані кортежів так, щоб вони зникали з представлення. Будь-яка модифікація, що породжує таку ситуацію, відміняється, і виводиться повідомлення про помилку.
2. Не можна використовувати запити, що модифікують дані декількох базових таблиць одночасно. Тому атрибути представлення, вказані в операторах UPDATE або INSERT, повинні належати до однієї базової таблиці, що є для нього постачальником даних.
3. Значення всіх атрибутів базової таблиці, яка не допускає порожніх значень, слід задавати оператором INSERT. Це гарантує наявність значень у всіх атрибутах базової таблиці, для яких вони необхідні.
4. Усі модифікації атрибутів представлення мають відповідати визначеним для відповідних атрибутів базової таблиці інтервалам допустимих значень, можливостям введення порожніх значень, обмеженням, визначенням DEFAULT тощо.
5. Не можна обновляти розподілене секціоноване представлення за допомогою курсора.

5.4. Програмування ефективних SQL-трансакцій

Сучасні СУБД при виникненні в системі помилок забезпечують цілісність інформації, що зберігається в базі даних, використовуючи механізм трансакцій. Трансакція – це логічна одиниця роботи, що складається з набору операторів SQL і обробляється СУБД як єдине ціле. У складі трансакцій дозволяється виконання практично всіх операторів мови SQL. Якщо при виконанні трансакції не виникає ніяких помилок, то всі модифікації даних стають постійними. В іншому випадку для них виконується відкіт, тобто повернення до початкового стану.

Чотири властивості логічної одиниці роботи дозволяють віднести її до трансакцій: атомарність, узгодженість, ізольованість та стійкість. Ці властивості називаються ACID-властивостями (від англ. atomicity, consistency, isolation, durability). Їх сутність полягає в наступному:

1. Атомарність – трансакція повинна бути атомарною (неподільною) одиницею роботи, в якій виконуються або всі модифікації, з яких складається трансакція, або жодна модифікація не виконується.
2. Узгодженість – після закінчення трансакції всі дані повинні залишитися в узгодженому стані. Усі внутрішні структури даних мають зберегти цілісність.
3. Ізоляція – модифікації даних, що виконуються однією трансакцією, слід ізолювати від модифікацій, які паралельно виконуються іншими трансакціями. Дані доступні для трансакції в тому стані, в якому вони перебували до зміни їх іншою трансакцією, або після

її завершення. Дані в проміжному стані недоступні для жодної трансакції.

4. Стійкість – після успішного закінчення трансакції її результат повинен зберегтися в системі. Модифікації не повинні зникнути навіть у випадку аварії системи.

Програмні додатки управляють трансакціями, головним чином, визначаючи моменти їх початку та закінчення. Система повинна коректно обробляти помилки, які переривають трансакцію до її завершення, тобто автоматично відкатують її і звільняють усі ресурси, які вона утримувала. Якщо розривається з'єднання внаслідок збою в роботі клієнтського додатка або відключення клієнтського комп'ютера, СУБД робить відкіт всіх перерваних трансакцій. Якщо клієнт завершує додаток, усі незавершені трансакції також підлягають відкоту.

Сучасні СУБД підтримують три типи трансакцій: явні, з автоматичною фіксацією і неявні. Розглянемо детально їх сутність.

Явна трансакція. Це трансакція, початок і кінець якої визначені явно. Для визначення явних трансакцій в додатках і сценаріях використовуються оператори:

- BEGIN TRANSACTION – задає початкову точку явної трансакції для з'єднання;
- COMMIT TRANSACTION – використовується для успішного завершення трансакції, внаслідок чого усі виконані трансакцією модифікації даних фізично зберігаються в базі даних, а задіяні в трансакції ресурси звільняються;
- ROLLBACK TRANSACTION – використовується для відміни трансакції, внаслідок чого усі дані, що були модифіковані під час трансакції, повертаються в початковий стан, а задіяні в трансакції ресурси звільняються.

Як приклад трансакції можна розглянути трансакцію по переведенню коштів з одного рахунку на інший в таблиці "Рахунок" бази даних банківської установи, яка реалізується за допомогою наступного коду:

```
BEGIN TRANSACTION
UPDATE Рахунок
SET Залишок = Залишок – 1000
WHERE НомерРахунку = 2206221401
UPDATE Рахунок
SET Залишок = Залишок + 1000
WHERE НомерРахунку = 220611424
ROLLBACK TRANSACTION
```

У цій трансакції оператор ROLLBACK TRANSACTION робить відкіт всіх змін, зроблених оператором UPDATE. Якщо в даному прикладі замість оператора ROLLBACK TRANSACTION використати оператор COMMIT TRANSACTION, усі оновлення запишуться в базу даних.

Режим явних трансакцій діє тільки на проміжку програми, що визначається операторами BEGIN TRANSACTION трансакції. Після завершення трансакції програма повертається в режим, що був заданий до її початку – у режим неявних трансакцій або трансакцій з автоматичною фіксацією.

Трансакція з автоматичною фіксацією. У режимі трансакції з автоматичною фіксацією кожен завершений оператор SQL або фіксується, якщо він виконався успішно, або відкатується, якщо при його виконанні виникла помилка. У багатьох СУБД режим автоматичної фіксації трансакцій є режимом управління трансакціями за замовчуванням. З'єднання програмного додатка з базою даних продовжує працювати в режимі автоматичної фіксації до тих пір, доки не почнеться явна трансакція або не буде активований режим неявних трансакцій. Після фіксації чи відкоту явної трансакції, або виключення режиму неявних трансакцій, сервер баз даних повертається для зазначеного з'єднання в режим трансакцій з автоматичною фіксацією.

Іноді в режимі автоматичної фіксації сервер баз даних робить відкіт не одного оператора SQL, а всього пакета. Подібна ситуація спостерігається тільки при виникненні помилки компіляції. Помилка компіляції не дає серверу баз даних побудувати план виконання, тому жоден оператор пакета не буде виконаний. При цьому здійснюється відкіт усіх операторів, що були виконані перед збійним оператором.

Неявна трансакція. У режимі неявних трансакцій з'єднання програмного додатка з базою даних після фіксації або відкоту поточної трансакції автоматично починає нову трансакцію. При використанні цього режиму програмісту не потрібно піклуватися про визначення меж трансакцій – тільки про фіксацію або відкіт модифікацій. У такому режимі трансакції генеруються безперервно, одна за одною, утворюючи ланцюжок трансакцій.

Після включення режиму неявної трансакції СУБД автоматично починає трансакцію при першому виконанні будь-якого з операторів: SELECT, INSERT, UPDATE, DELETE, ALTER TABLE, TRUNCATE TABLE, CREATE, DROP, OPEN, FETCH, REVOKE, GRANT. Трансакція триває до виконання оператора COMMIT або ROLLBACK. СУБД безперервно генерує неявні трансакції до тих пір, доки цей режим не буде вимкнений. Для запуску режиму неявних трансакцій використовують оператор SET IMPLICIT TRANSACTIONS ON. Для від-

ключення цього режиму застосовується оператор SET IMPLICIT TRANSACTIONS OFF.

Програмування ефективних трансакцій ґрунтується на дотриманні наступних принципів:

- не можна вимагати від користувачів введення інформації під час трансакції;
- не варто починати трансакцію під час перегляду даних;
- трансакції повинні бути максимально короткими;
- необхідно мінімізувати обсяг даних, до яких має доступ трансакція.

Особливої обережності необхідно дотримуватися при управлінні неявними трансакціями, оскільки саме цей режим найбільш схильний до виникнення проблем у процесі паралельного виконання запитів. Оскільки при використанні неявних трансакцій будь-який оператор SQL, наступний за COMMIT і ROLLBACK, автоматично запускає нову трансакцію, то вона може початися під час перегляду інформації, або навіть під час очікування введення даних користувачем. Тому після виконання останньої необхідної трансакції слід відключити режим неявних трансакцій до моменту, доки він знову не стане необхідним. Цей процес дозволяє СУБД використовувати режим автоматичного підтвердження трансакцій під час перегляду інформації або введення даних користувачем. Саме програміст несе відповідальність за вибір додатком правильної дії (COMMIT або ROLLBACK) у випадку виникнення помилки періоду виконання або помилки компіляції.

Компонентом СУБД, від якого суттєво залежить реалізація трансакційності SQL-сценаріїв, є журнал трансакцій. Журнал трансакцій присутній у кожній реляційній базі даних і в ньому реєструються модифікації її даних, що були виконані кожною трансакцією. У сучасних базах даних журнал трансакцій підтримує три види операцій:

1. Відкат трансакцій. Якщо додаток виконує оператор ROLLBACK, або СУБД знаходить помилку (наприклад, втрату зв’язку бази даних з додатком, ініціювавши трансакцію), записи журналу трансакцій використовуються для відкоту будь-яких змін, зроблених під час виконання трансакції.
2. Відновлення всіх незавершених трансакцій в процесі запуску сервера баз даних. У випадку аварії сервера у деяких базах даних не вистачає часу на запис окремих модифікацій у файли даних. У такій ситуації, у файлах даних можлива поява модифікацій, виконаних незавершеними трансакціями. Після запуску сервера починається відновлення всіх баз даних. При цьому виконується повтор усіх модифікацій, зареєстрованих в журналі, але не записаних у

файли даних, і відкіт всіх незавершених трансакцій, знайдених у журналі. Так забезпечується цілісність баз даних.

3. Відновлення баз даних шляхом повтору всіх зареєстрованих до моменту аварії дій. Після втрати бази даних, яка можлива, наприклад, у випадку відмови жорсткого диска, її вдається повернути у стан, в якому вона знаходилася на момент збою. Спочатку слід відновити останню копію бази даних, а потім виконати послідовність копій трансакцій аж до моменту збою. При відновленні кожної копії журналу повторюються всі трансакції, наново виконуючи всі зареєстровані в журналі модифікації. Для забезпечення цієї можливості СУБД використовують попереджувальну реєстрацію трансакцій, яка гарантує, що модифікації будуть записані у файли не раніше, ніж створені асоційовані з ними записи журналу.

У процесі, коли множина користувальницьких додатків одночасно намагається модифікувати дані, необхідне існування системи управління, яка в змозі захищати модифікації, виконані одним додатком, від негативної дії модифікацій, зроблених іншими. Цей процес називається управлінням паралельним виконанням трансакцій. Сучасні СУБД реалізують управління паралельним виконанням при одночасній роботі декількох користувачів, що модифікують вміст бази даних за допомогою механізму блокувань.

Блокування – це об’єкт, за допомогою якого СУБД встановлює залежність користувача від ресурсу. Програми забороняють іншим користувачам виконувати над ресурсами операції, які негативно впливають на роботу користувача, що володіє блокуванням цього ресурсу. Блокування управляються внутрішніми системними механізмами. Вони встановлюються і знімаються залежно від дій користувача.

Блокування застосовуються в базах даних на різних рівнях. Їх встановлюють для таблиць, кортежів, атрибутів і т.д. СУБД динамічно визначає рівень блокування для кожного оператора SQL. Рівень, на якому задається блокування, може варіюватися для різних об’єктів у межах одного запиту. Наприклад, для однієї таблиці невеликого розміру блокування встановлюється на рівні таблиці, тоді як у більшій таблиці воно задається для окремих кортежів.

Якщо з’єднання намагається встановити блокування, що конфліктує з блокуванням, яке утримується іншим з’єднанням, йому не вдасться це зробити, доки не відбудеться одна з наступних подій:

- звільниться конфліктне блокування;
- закінчиться тайм-аут з’єднання.

За замовчуванням, тайм-аут для з’єднання не заданий, але для деяких додатків він встановлюється, щоб не чекати невизначено довго. У випадку, якщо декілька з’єднань блокуються в очікуванні конфлікт-

ного блокування на одному і тому ж ресурсі, то звільнене попереднім з'єднанням блокування надається за принципом черги: першим прийшов – першим обслужений.

Хоча сучасні СУБД управляють блокуваннями автоматично, існують способи збільшення ефективності програмних додатків, що знаходяться в стадії розробки. Налаштування блокувань у додатках можна здійснювати наступними способами:

1. Задаючи обробку взаємних блокувань та установку пріоритетів при взаємних блокуваннях. Взаємні блокування виникають, коли дві (або більше) трансакції, що конкурують за ресурс, стають взаємозалежними. Жодна трансакція не може звільнити задіяні ресурси, поки здійснені нею зміни не будуть зафіксовані або відмінені, а цього не трапляється, оскільки обидві трансакції чекають звільнення ресурсу іншою трансакцією.
2. Задаючи обробку тайм-аутів і визначаючи тривалість тайм-ауту блокувань. Якщо встановити значення LOCKTIMEOUT, додаток зможе орієнтуватись на максимальний час, протягом якого оператор чекає заблокований ресурс. Якщо оператор чекає довше, ніж визначено параметром LOCKTIMEOUT, заблокований оператор автоматично відміняється, а додатку повертається повідомлення про помилку 1222 "Lock request time-out period exceeded" (Перевищення тайм-ауту при очікуванні блокування).
3. Встановлюючи рівень ізоляції трансакції. Наприклад, MS SQL Server за замовчуванням працює на рівні ізоляції READ COMMITTED (підтверджене читання). Для того, щоб використовувати в додатках інші рівні ізоляції, слід визначити блокування для поточного сеансу, встановивши рівень ізоляції сеансу за допомогою оператора SET TRANSACTION ISOLATION LEVEL.
4. Призначаючи блокування на рівні таблиці для операторів SELECT, INSERT, UPDATE і DELETE. Для операторів SELECT, INSERT, UPDATE і DELETE дозволяється задавати ряд вказівок блокування (locking hints) на рівні таблиці. Вони потрібні для тонкої настройки типів блокувань об'єкта.

ПИТАННЯ ТА ЗАВДАННЯ

1. Дайте визначення збереженої процедури.
2. Які переваги дозволяють отримати збережені процедури при виконанні складних програмних сценаріїв?
3. Яким чином збережені процедури сприяють підвищенню безпеки баз даних?
4. Охарактеризуйте категорії збережених процедур.
5. Опишіть порядок створення збережених процедур.

6. Який порядок створення та зміни плану виконання операторів збереженої процедури?
7. Опишіть порядок запуску на виконання збереженої процедури.
8. Які оператори дозволять модифікувати та видаляти збережені процедури?
9. Завдяки чому реалізується динамічність збережених процедур?
10. Опишіть порядок використання вхідних та вихідних параметрів збережених процедур.
11. Як здійснюється обробка помилок, що виникають під час виконання збережених процедур?
12. Що розуміють під вкладеністю збережених процедур?
13. У яких випадках у збережених процедурах доцільне використання курсорів?
14. Дайте визначення тригера.
15. У яких випадках доцільне використання тригерів для забезпечення цілісності реляційних даних?
16. Які події викликають спрацьовування тригерів?
17. Охарактеризуйте класи тригерів, що реалізовані в сучасних СУБД.
18. Опишіть порядок створення тригерів.
19. Які методи використовуються для модифікації та видалення тригерів?
20. Яким чином можна заблокувати, а потім відновити спрацьовування тригера?
21. Що таке псевдотаблиці та як вони використовуються при спрацьовуванні тригерів?
22. З якою метою в тригерах використовується функція @@ROWCOUNT?
23. Назвіть оператори, використання яких під час програмування тригерів забороняється.
24. Перерахуйте найпоширеніші задачі, що вирішуються за допомогою тригерів.
25. Дайте визначення представлення.
26. За рахунок яких можливостей представлень можна забезпечити підвищення ефективності програмування складних SQL запитів та сценаріїв?
27. Яким чином представлення дозволяють спростити доступ до даних та маніпулювання ними?
28. Які обмеження діють під час створення представлень?
29. Опишіть порядок застосування оператора SQL для створення представлень.
30. Охарактеризуйте матеріалізацію представлень.
31. Наведіть приклад створення секціонованого представлення.
32. Назвіть способи модифікації та видалення представлень.

33. Яким чином здійснюється доступ до даних за допомогою представлень?
34. Назвіть правила, які слід врахувати перед модифікацією даних через представлення.
35. Дайте визначення трансакції.
36. Назвіть властивості трансакцій.
37. Охарактеризуйте типи трансакцій.
38. Що в реляційних базах даних розуміють під блокуванням?
39. Назвіть способи налаштування блокувань.

ТЕСТИ

1. Коли створюється план виконання збереженої процедури?
   а) при створенні збереженої процедури;
   б) при першому запуску збереженої процедури;
   в) при зміні структури базової таблиці, до якої прив'язана збережена процедура.
2. Які збережені процедури називаються розширеними?
   а) збережені процедури, які звертаються до зовнішніх програм;
   б) збережені процедури, які викликають інші збережені процедури;
   в) збережені процедури, які можуть приймати і повертати значення.
3. У якій базі даних буде створена збережена процедура після виконання наступного сценарію на сервері MS SQL?
   USE Bank
   GO
   CREATE PROCEDURE #ShowEmpl
   AS
   SELECT EmpName FROM Employee
   а) Bank;
   б) Master;
   в) TempDB.
4. Що означають ключові слова WITH RECOMPILE?
   а) компіляцію збереженої процедури при її створенні;
   б) компіляцію збереженої процедури при кожному її виконанні;
   в) компіляцію збереженої процедури при першому її виконанні.
5. Яка особливість збережених процедур називається відкладеним дозволом імен?
   а) при виконанні системних збережених процедур необов'язково використовувати повні імена;
   б) під час створення збережених процедур не відбувається перевірка існування об'єктів, на які посилається процедура;

в) якщо база даних, в якій міститься збережена процедура, поточна, то для виклику процедури достатньо вказати частину її імені.

6. Після якого ключового слова починаються оператори, які утворюють виконавчу частину збереженої процедури?
   а) AS;
   б) RETURN;
   в) CREATE PROCEDURE.
7. Який з перерахованих операторів може викликати спрацьовування тригера?
   а) DROP;
   б) SELECT;
   в) UPDATE.
8. Як впливає на продуктивність системи збільшення кількості тригерів у базі даних?
   а) знижує продуктивність;
   б) підвищує продуктивність;
   в) не впливає на продуктивність.
9. Яким способом можна запустити тригер вручну?
   а) таких способів не існує;
   б) виконавши оператор *EXEC ім'я тригера*;
   в) виконавши оператора *TRUNCATE TABLE ім'я тригера*.
10. Для деякої таблиці визначені обмеження і прив'язані AFTER-тригер і INSTEAD OF-тригер. У якій послідовності вони спрацьовуватимуть?
    а) INSTEAD OF-тригер, обмеження, AFTER-тригер;
    б) Обмеження, INSTEAD OF-тригер, AFTER-тригер;
    в) INSTEAD OF-тригер, AFTER-тригер, обмеження.

РОЗДІЛ 6. ЗАСОБИ АДМІНІСТРУВАННЯ SQL-СЕРВЕРА

6.1. Модель системи безпеки інформаційної системи з базою даних

Система безпеки є невід'ємною частиною правильно спроектованої інформаційної системи з базою даних. У загальному вигляді систему безпеки можна представити у вигляді моделі, яку складають шість відносно незалежних рівнів: фізична безпека, безпека мережевого доступу, доменна безпека, безпека локального комп'ютера, безпека сервера баз даних, безпека програмних додатків. Розглянемо детально кожен з цих рівнів.

Фізична безпека. Перший рівень моделі безпеки інформаційної системи з базою даних представлений фізичною безпекою. Фізична безпека забезпечує захист доступу до інфраструктури, яку утворюють внутрішні мережеві компоненти та апаратне обладнання, що підтримує роботу серверних компонент системи.

Фізична безпека має на меті захист життєво важливого серверного та мережевого обладнання шляхом розміщення їх у приміщеннях з обмеженим доступом. Для захисту від стихійного лиха необхідно ретельно планувати і регулярно створювати резервні копії бази даних, які слід зберігати окремо від місця експлуатації сервера.

Безпека мережевого доступу. На другому рівні – безпека мережевого доступу. Вона включає такі компоненти, як шифрування пакетів та ізоляція транспортного протоколу.

Шифрування пакетів реалізоване на проміжному рівні між клієнтом і сервером баз даних за допомогою протоколу SSL (Secure Socket Layer) або шифрування при виклику віддалених процедур (RPC). При обміні даними між комп'ютерами, під управлінням операційної системи Windows версій, випущених після 2000 року, шифрування пакетів реалізується за допомогою IP безпеки (IP Security, IPSec). IPSec підтримується також багатьма сучасними маршрутизаторами.

Інша дієва методика забезпечення безпеки мережевого доступу полягає у застосуванні брандмауера або спеціалізованого обладнання для розмежування комп'ютерних мереж. Найпростіше рішення розмежування мережевого доступу – використання різних протоколів у різних сегментах комп'ютерної мережі. Наприклад, у локальній мережі можна використовувати стек протоколів IPX/SPX, а в зовнішній (Інтернет) – TCP/IP. У сучасних реалізаціях локальних обчислювальних мереж потребу в розділенні доступу між сегментами мережі ви-

рішують застосовуючи технологію віртуальних приватних мереж (virtual private network, VPN).

Доменна безпека. Третій рівень забезпечує доменну безпеку. Вона реалізується за допомогою служб каталогу облікових записів області мережевих імен. Наприклад, у мережах Microsoft цю роль виконують служби Active Directory. Якщо комп'ютер, на якому працює сервер баз даних, є членом мережевого домена, можна наділити обліковий запис користувача або групи користувачів привілеями для доступу до сервера баз даних і виконання на ньому певних дій.

Служби обліку доменних записів користувачів комп'ютерної мережі забезпечують надійну перевірку прав користувачів на рівні доступу до мережевих компонентів. Доменні паролі шифруються з метою уникнення їх перехоплення під час передачі по мережі. Зазвичай, вони чутливі до регістра символів, що ускладнює їх підбір. Крім того, широко застосовуються такі прийоми, як визначення мінімальної довжини пароля та терміну його дії.

Безпека локального комп'ютера. На четвертому рівні забезпечується безпека локального комп'ютера. Вона зумовлює проведення аудиту засобами операційної системи, верифікацію прав доступу до файлів та реєстру, а також функціонування служб шифрування. Сервери сучасних баз даних, зазвичай, інсталюються на комп'ютерах, які працюють під управлінням операційних систем сімейств Windows або Unix. Ці операційні системи підтримують аудит системи безпеки, дозволяючи відстежувати такі події, як вхід користувачів у систему, спроби звернення до файлових об'єктів і, у тому числі, до файлів баз даних.

Безпека сервера баз даних. П'ятий рівень забезпечує систему безпеки сервера баз даних, яка включає чотири категорії безпеки: аутентифікацію, авторизацію, шифрування і служби аудиту.

Процес надання доступу до бази даних складається з двох фаз: спочатку виконується підключення до сервера баз даних, а потім відкривається доступ до бази даних з усіма її об'єктами (авторизація). Дозволи на роботу з об'єктами дозволяють або забороняють користувачу виконувати дії над об'єктами бази даних, наприклад, таблицями та представленнями. Дозволи на виконання SQL-операторів дозволяють або забороняють користувачам створювати об'єкти бази даних, переглядати та маніпулювати даними, робити їх копії.

Для зменшення кількості адміністративних операцій, які необхідно виконати для надання (позбавлення) дозволів користувачам бази даних, сучасні СУБД підтримують групування користувачів за ролями. Ролі схожі на групи операційних систем, але створюються і супроводжуються в рамках сервера бази даних. Права доступу можна надати ролі, так само, як і окремому користувачу. Якщо права доступу

призначаються ролі, то кожен користувач, включений у цю роль, набуває її права доступу. Спадкоємство – це здатність об'єкта (у даному випадку користувача) приймати всі властивості іншого об'єкта (у цьому випадку ролі). Ролі мають властивість вкладеності одна в одну, що дозволяє будувати ієрархію ролей.

Існує два типи ролей: стандартні та прикладні. Стандартні ролі від початку присутні в сервері баз даних та наділені певними правами, які можуть успадковуватися користувачами, що одержали в них членство. Прикладні ролі створюються розробниками інформаційних систем з базами даних з метою вирішення пов'язаних з ними задач безпеки.

Усі дії, що виконуються в базі даних, відстежуються за допомогою аудиту СУБД. Деякі об'єкти бази даних, наприклад, збережені процедури, дозволяється зашифровувати, щоб захистити їхній вміст від несанкціонованого читання.

Безпека програмних додатків. Шостий рівень організовує безпеку програмних додатків. Додаток може розширювати можливості системи безпеки баз даних, доповнюючи її власними функціями безпеки.

Програма, що звертається до бази даних, викликає спеціальну системну збережену процедуру (у SQL Server це sp_setapprole) з метою активізації ролі програмних додатків. Крім того, у додатку іноді реалізована власна система безпеки, непідконтрольна СУБД. Для ізоляції додатків від деталей механізму доступу до даних застосовуються функції API доступу до даних, що підтримуються технологіями, такими як: ADO, OLE DB та ODBC.

Згідно з моделлю безпеки бази даних, відповідальність за кожен її рівень розподіляється між визначеними посадовими особами. За реалізацію перших чотирьох рівнів системи безпеки бази даних відповідають мережеві та системні адміністратори, адміністратори та розробники бази даних відповідають за п'ятий рівень, а розробники додатків – за шостий. Фахівці з інформаційної безпеки, як правило, спостерігають за проектуванням системи безпеки в загальному обсязі.

6.2. Планування системи безпеки бази даних

Реалізація ефективної системи безпеки бази даних потребує її ретельного планування, при якому основна увага приділяється системі безпеки сервера баз даних, що займає п'ятий рівень моделі безпеки. Цей процес полягає у визначенні користувачів бази даних, доступних їм об'єктів та дій, які їм дозволено виконувати в базі даних. Потрібна інформація може бути сформована на основі вимог до інформаційної системи, основу якої утворює база даних.

Після складання списку об'єктів бази даних, що потребують захисту та дій, що доступні не всім користувачам, слід створити список

користувачів, які повинні мати доступ до бази даних. На основі цієї інформації готують список відповідностей “дія – користувач” у вигляді таблиці з перехресними посиланнями на елементи списків окремих дій і користувачів. Цей список визначає, яким користувачам доступні захищені об’єкти, а також, які захищені дії користувачі можуть виконувати в базі даних.

Існують загальні правила реалізації системи безпеки бази даних, які визначають, кому необхідно призначати дозволи: ролям або окремим користувачам. Вибір залежить від рівня персоніфікації цих дозволів. Наприклад, якщо одному користувачу потрібен унікальний набір дозволів на доступ та маніпулювання даними, слід призначити відповідні дозволи окремому обліковому імені. Проте таких дій, по можливості, слід уникати. Користувачів бази даних краще включати у визначені ролі, а не призначати їм окремі дозволи.

Для призначення дозволів на роботу з об’єктами і виконання SQL-операторів краще використовувати стандартні ролі рівня сервера баз даних, уникаючи безпосереднього призначення дозволів прикладним ролям і окремим користувачам. У системних ролей є певні особливості, які не можна змінити. Тому слід бути обережними, приписуючи користувачів до системних ролей.

У разі необхідності виділення прикладних ролей і окремих користувачів спочатку слід призначати дозволи ролям і тільки потім окремим користувачам. Нагорі ієрархії системи безпеки потрібно розташовувати мінімальні дозволи, які можна застосовувати до всіх користувачів, а нижче – розширені, для обмеженої кількості користувачів.

Гарною практикою забезпечення контрольованого доступу до даних є створення представлень та збережених процедур, що використовують ланцюжки володіння. У випадку, коли право на використання об’єктів, що звертаються послідовно один до одного, належить одному користувачу (ролі) і розташовується в одній базі даних, говорять про створення ланцюжка володіння. Тобто, якщо у користувача (ролі) бази даних є дозвіл на роботу з першим об’єктом ланцюжка володіння (збереженою процедурою або представленням), то по ланцюжку він зможе діставати доступ до даних і виконувати потрібні дії.

Ланцюжок володіння існує завдяки тому, що представлення може викликати дані базових таблиць або інших представлень. Крім того, збережені процедури здатні звертатися до таблиць, представлень та інших збережених процедур. Об’єкт, що звертається до інших об’єктів, знаходиться в залежності від цих об’єктів. Наприклад, представлення, яке звертається до базової таблиці, залежить від результуючого набору, що повертається з таблиці. Ланцюжки володіння можна

застосовувати лише до операторів SELECT, INSERT, UPDATE і DELETE.

6.3. Реалізація та адміністрування системи безпеки сервера бази даних

Для того, щоб користувач зміг підключитися до сервера баз даних, тобто виконати аутентифікацію, для нього необхідно створити на сервері облікове ім'я і відкрити для нього доступ. У сучасних СУБД цю задачу дозволяють вирішити інтерактивні утиліти і спеціалізовані системні збережені процедури.

Для того, щоб отримати доступ до певної бази даних, облікові записи системи безпеки проходять авторизацію, тобто перевірку на приналежність до ролі, якій дозволений доступ до цієї бази даних. Так само, як для аутентифікації, для управління авторизацією використовуються інтерактивні утиліти і системні збережені процедури. Крім того, для гнучкого управління дозволами на роботу з об'єктами бази даних і виконання SQL-операторів застосовуються оператори: GRANT, REVOKE.

Для застосування оператора GRANT необхідно вказати дозвіл (або дозволи), які потрібно призначити, а також ім'я, для якого це робиться. Основні конструкції оператора GRANT для призначення дозволів на виконання SQL-операторів наступні:

GRANT ALL | список SQL-операторів
TO ім'я облікового запису(ів)

Дозволи на виконання SQL-операторів відділяються один від одного комами так само, як і облікові імена, наприклад:

GRANT CREATE TABLE, CREATE VIEW
TO User01, User02

Внаслідок виконання цього оператора користувачі "User01" та "User02" одержують два дозволи на виконання SQL-операторів: CREATE TABLE, CREATE VIEW.

У випадку, якщо потрібно надати всі допустимі дозволи на роботу з об'єктами і виконання SQL-операторів, застосовується ключове словом ALL, як показано в наступному прикладі:

GRANT ALL TO Public

Внаслідок виконання цього оператора ролі "Public" надаються всі можливі дозволи на виконання SQL-операторів.

При призначенні дозволів на роботу з об'єктом необхідно також вказати, до якого об'єкта вони належать. Це не потрібно було робити для дозволів на виконання SQL-операторів, оскільки вони належать до

бази даних в цілому. Основні конструкції оператора GRANT для призначення дозволів на роботу з об'єктами наступні:

GRANT ALL | список SQL-операторів
(атрибут(ти)) ON ім'я таблиці | ім'я збереженої процедури
TO ім'я облікового запису(ів)
WITH GRANT OPTION
AS ім'я облікового запису(ів)

Використовуючи ці конструкції, можна визначати дозволи як для всієї таблиці, так і для окремих її атрибутів. Наприклад:

GRANT INSERT, DELETE, UPDATE, SELECT
(Прізвище, [Ім'я], [По батькові]) ON Людина
TO User01

Для збереженої процедури дійсний лише один з дозволів на роботу з об'єктом – EXECUTE. Наступний приклад демонструє призначення дозволу EXECUTE ролі "Бухгалтерія" для процедури "ЗбільшитиЗарплатуВ2Рази", що зберігається:

GRANT EXEC
ON ЗбільшитиЗарплатуВ2Рази
TO Бухгалтерія

Існують ситуації, коли виникає необхідність делегувати право певному користувачу надавати дозволи на виконання SQL-операторів та роботу з об'єктами бази даних іншим обліковим іменам. Для цього застосовується конструкція WITH GRANT OPTION. Ключове слово AS дозволяє вказати ім'я облікового запису або роль у поточній базі даних, яка матиме повноваження на виконання оператора GRANT.

Для зняття дозволів, раніше наданих обліковому імені, використовується оператор REVOKE. Формат оператора аналогічний формату оператора GRANT за деякими виключеннями. Ключове слово TO оператора GRANT при знятті дозволів на виконання SQL-операторів замінюється на ключове слово FROM, а при знятті дозволів на роботу з об'єктами використовуються як ключове слово TO, так і FROM.

Синтаксис зняття дозволів на роботу з об'єктами включає необов'язкові конструкції GRANT OPTION FOR і CASCADE. Конструкція GRANT OPTION FOR використовується для зняття дозволу WITH GRANT OPTION, призначеного обліковому імені. Ключове слово CASCADE служить для зняття дозволів із заданого облікового імені разом з усіма дозволами, наданими від цього імені іншим обліковим іменам. Наступний приклад знімає дозвіл EXEC з користувачів, що входять до ролі "Бухгалтерія" та всіх інших користувачів, яких вони наділили дозволом EXEC:

REVOKE EXEC
ON УвеличитьЗарплатуВ2Раза
FROM Бухгалтерія
CASCADE

Створену систему безпеки слід випробувати, підключаючись до БД різними обліковими іменами, перевіряючи роботу дозволів та виконуючи аудит безпеки.

6.4. Імпорт і експорт реляційних даних

Імпорт реляційних даних являє собою процес отримання даних із зовнішніх по відношенню до бази даних джерел (наприклад, з текстового файлу ASCII) та розміщення їх у реляційних таблицях. Зворотний процес називається експортом реляційних даних і являє собою передачу даних з екземпляра реляційної бази даних у певному, заданому користувачем, форматі (наприклад, копіювання вмісту таблиці MS SQL в таблицю Microsoft Office Excel). Для імпорту та експорту реляційних даних використовуються спеціалізовані програмні утиліти та оператори SQL.

Принцип роботи утиліт розглянемо на прикладі утиліти командного рядка bcp, яка використовується для передачі даних між базами даних MS SQL Server та зовнішніми файлами. При використанні утиліти bcp спочатку виконується експорт даних із таблиці-джерела у файл, а потім, при необхідності, імпорт даних із файлу в таблицю бази даних. Крім того, bcp дозволяє переносити дані з таблиці бази даних у файл, призначений для використання іншими програмами, наприклад, Microsoft Office Excel.

Для того, щоб імпортувати інформацію в базу даних цим способом, дані, що містяться у файлі, слід розбити на рядки і стовпці. Дані, що імпортуються, додаються до вмісту таблиці. Навпаки, при копіюванні даних з бази даних у файл, вони записуються поверх попереднього вмісту файлу. При застосуванні утиліти bcp потрібно пам’ятати про наступні правила:

- дані дозволяється імпортувати тільки в існуючу таблицю, а при експорті даних у файл, що не існує, утиліта bcp створює новий файл;
- файл повинен містити дані в символьному форматі або у форматі, що був раніше встановлений утилітою (наприклад, у власному форматі bcp);
- необхідно, щоб кожне поле таблиці було сумісне з відповідним полем файлу, з якого імпортуються дані (наприклад, не можна скопіювати данні типу char в атрибут типу int);

- для копіювання даних необхідні відповідні права доступу до їхнього джерела та цільового місця зберігання: для копіювання з файлу в базову таблицю необхідно мати права доступу INSERT і SELECT для таблиці, а для копіювання з базової таблиці у файл – право доступу SELECT для таблиці.

Наприклад, масове копіювання даних із таблиці "Людина" в базі даних "MyBank" у файл person.txt, можна здійснити, застосовуючи команду bcp наступним чином:

bcp MyBank..Людина out person.txt -c -T

Ключове слово out задає режим експорту даних із таблиці "Людина" у файл person.txt. Якщо в цій команді замість out використати ключове слово in, то можна вставити в таблицю "Людина" всі дані з файлу person.txt. Параметр -c задає символьний (char) формат даних, а параметр -T визначає спосіб підключення до сервера баз даних (у цьому випадку – довірене з'єднання).

Інший спосіб перенесення даних із файлу в таблицю бази даних та навпаки – за допомогою оператора BULK INSERT. Попередній приклад за допомогою цього оператора може бути реалізований у наступному вигляді:

BULK INSERT Людина
FROM 'C:\person. txt'
WITH (DATAFILETYPE = 'CHAR')

Зміст цього коду видається інтуїтивно зрозумілим, тому немає сенсу наводити його детальний опис. Крім того, на сьогодні адміністратори баз даних для виконання операцій масового копіювання вважають за краще використовувати більш прогресивні програмні інструменти.

У сучасних СУБД для отримання, трансформації й об'єднання даних з різних об'єктів у єдиний цільовий набір використовуються різні графічні інструменти і програмовані об'єкти. Прикладом можуть служити сервіси трансформації даних сервера MS SQL – DTS (Data Transformation Services). Засобами DTS генерують пакети DTS, які дозволяють створювати рішення для перенесення даних, що відповідають бізнес-вимогам конкретної організації. Зокрема, DTS дозволяє імпортувати та експортувати дані.

Пакет DTS – це організований набір з'єднань, задач, перетворень і обмежень потоку робіт, зібраний за допомогою програмних методів і збережений у структурованому файлі. У кожному пакеті міститься опис завдання, яке реалізовується під час виконання пакета. У цей час пакет з'єднується з необхідними джерелами даних, копіює дані та об'єкти бази даних, трансформує дані і повідомляє про події інших

користувачів або процеси. Пакети можна редагувати, захищати паролем і регламентувати їх виконання.

Пакети DTS складаються із задач. Задача DTS – це дискретний набір функціональності, виконуваний як єдиний етап у складі пакета. Кожна задача визначає дію, яка є частиною процесу переміщення або трансформації даних у рамках виконуваного завдання. За з'єднання пакетів DTS з різними джерелами та цільовими сховищами даних відповідає служба, заснована на технології OLE DB, яка дозволяє копіювати й трансформувати дані з різноманітних джерел.

ПИТАННЯ ТА ЗАВДАННЯ

1. Чим опікується рівень фізичної безпеки інформаційної системи з базою даних?
2. Охарактеризуйте компоненти безпеки мережевого доступу.
3. За допомогою яких служб забезпечується доменна безпека?
4. Що зумовлює безпека локального комп'ютера?
5. Охарактеризуйте категорії рівня безпеки сервера баз даних.
6. Як організується рівень безпеки програмних додатків?
7. Плануванню якого з рівнів моделі безпеки слід приділяти основну увагу?
8. Охарактеризуйте загальні правила призначення дозволів на роботу з об'єктами і виконання SQL-операторів.
9. Що розуміють під ланцюжком володіння?
10. До яких операторів можна застосовувати ланцюжки володіння?
11. Які дії потребує організація аутентифікації клієнта бази даних?
12. Які SQL-оператори здатні забезпечити авторизацію клієнта бази даних?
13. Назвіть основні конструкції оператора GRANT для призначення дозволів на виконання SQL-операторів.
14. Наведіть приклад застосування оператора GRANT для призначення дозволів на роботу з об'єктом бази даних.
15. Яким чином можна делегувати право певному користувачу надавати дозволи на виконання SQL-операторів іншим обліковим іменам?
16. Назвіть призначення та опишіть синтаксис оператора REVOKE.
17. Що являє собою імпорт реляційних даних?
18. Назвіть правила використання утиліти командного рядка bcp для передачі даних між базами даних MS SQL Server та зовнішніми файлами.
19. Наведіть приклад перенесення даних з зовнішнього файлу в таблицю бази даних за допомогою оператора BULK INSERT.
20. Охарактеризуйте сервіси трансформації даних, які реалізовані в сучасних версіях MS SQL.

ТЕСТИ

1. Що покладено в основу організації доменної безпеки баз даних?
   а) технологія розділення доступу на основі створення віртуальних приватних мереж (VPN);
   б) служби каталогу облікових записів області мережевих імен локальної обчислювальної мережі;
   в) шифрування пакетів на проміжному рівні між клієнтом і сервером баз даних за допомогою протоколу SSL.
2. На якому рівні моделі системи безпеки баз даних розглядаються категорії "авторизація" і "аутентифікація"?
   а) безпека додатків;
   б) доменна безпека;
   в) безпека серверу баз даних;
   г) безпека мережевого доступу.
3. Який спосіб є найбільш раціональним для вирішення задач безпеки, пов'язаних із підключенням до бази даних нової клієнтської програми?
   а) визначення прав користувачів за допомогою стандартних ролей;
   б) визначення прав користувачів за допомогою прикладних ролей;
   в) визначення прав користувачів за допомогою персональних дозволів.
4. Що використовує додаток, який звертається до бази даних для активізації ролі прикладних додатків?
   а) службу IP безпеки IPSecure;
   б) системну збережену процедуру;
   в) каталог облікових записів Active Directory.
5. Яка необхідна умова повинна бути виконана для того, щоб створити ланцюжок володіння з організації створення некластерних індексів базових таблиць?
   а) такі ланцюжки володіння неможливі;
   б) об'єкт, що викликає ланцюжок володіння, повинен бути збереженою процедурою;
   в) набір усіх об'єктів ланцюжка володіння та об'єкт, що їх викликає, повинні належати одному користувачу.
6. Який із запропонованих варіантів відповідей найбільш коректно характеризує результат виконання наступного запиту:

   GRANT INSERT
   ON ДодатиКлієнта
   TO ФронтОфіс

   якщо "ДодатиКлієнта" – збережена процедура, а "ФронтОфіс" – роль?

а) відбудеться фатальна помилка;
б) роль “ФронтОфіс” одержить дозвіл виконувати збережену процедуру “ДодатиКлієнта”;
в) ролі “ФронтОфіс” буде заборонено виконувати збережену процедуру “ДодатиКлієнта”;
г) роль “ФронтОфіс” одержить дозвіл вносити дані в базові таблиці за допомогою збереженої процедури “ДодатиКлієнта”.

7. З якою метою застосовується конструкція GRANT OPTION FOR?
а) для зняття дозволу клієнту надавати дозвіл іншим обліковим іменам;
б) для надання дозволу клієнту надавати дозвіл іншим обліковим іменам;
в) для надання дозволу на роботу з об’єктами для облікового запису, вказаного після конструкції;
г) для надання дозволу на виконання SQL-операторов для облікового запису, вказаного після конструкції.

8. Який із запропонованих варіантів правильно характеризує роботу утиліти bcp?
а) при імпорті дані додаються до вмісту таблиці, при експорті – у файл, дані записуються поверх попереднього вмісту файлу;
б) при імпорті дані записуються поверх попереднього вмісту таблиці, при експорті – у файл, дані додаються до вмісту файлу;
в) при імпорті дані записуються поверх попереднього вмісту таблиці, при експорті – у файл, дані записуються поверх попереднього вмісту файлу;
г) при імпорті дані додаються до вмісту таблиці, при експорті – у файл, дані додаються до вмісту файлу.

9. Яка з перерахованих вимог є обов’язковою при експорті даних з таблиці бази даних у текстовий файл?
а) існування вказаного файлу;
б) право SELECT для таблиці;
в) право на виконання операції копіювання в базі даних.

10. Що є результатом застосування засобів DTS для реалізації імпорту/експорту реляційних даних?
а) програмні пакети, збережені у вигляді спеціалізованих файлів;
б) збережені процедури, що зберігаються в системній базі даних master;
в) користувальницькі функції, призначені для отримання, трансформації й об’єднання даних із різних об’єктів в єдиний цільовий набір.

Частина ІІ
СТВОРЕННЯ ІНТЕЛЕКТУАЛЬНИХ СИСТЕМ

РОЗДІЛ 7. ВСТУП ДО ПРОБЛЕМИ ШТУЧНОГО ІНТЕЛЕКТУ

7.1. Загальне поняття штучного інтелекту

У другій половині XX століття було сформульовано поняття штучного інтелекту (artificial intelligence) і запропоновано декілька його визначень. Одним з перших визначень, яке, незважаючи на значну широту трактування, дотепер не втратило своєї актуальності, полягає в представленні штучного інтелекту як спосіб примусити обчислювальну машину думати, як людина.

Актуальність інтелектуалізації обчислювальних систем зумовлена потребою людини знаходити рішення в таких реаліях сучасного світу, як неточність, двозначність, невизначеність, нечіткість і необґрунтованість інформації. Необхідність підвищення швидкості й адекватності даного процесу стимулює створення обчислювальних систем, які через взаємодію з реальним світом засобами робототехніки, виробничого устаткування, приладів та інших апаратних засобів можуть сприяти його здійсненню.

Обчислювальні системи, в основу роботи яких покладена виключно класична логіка, тобто алгоритми рішення відомих задач, стикаються з проблемами, зустрічаючи невизначені ситуації. На відміну від них, живі істоти хоча й програють у швидкості, здатні приймати успішні рішення в подібних ситуаціях.

Як приклад можна навести обвал фондового ринку 1987 року, коли комп'ютерні програми продавали акції на сотні мільйонів доларів з метою одержати прибуток у декілька сотень доларів, що власне і створило передумови для обвалу. Положення було виправлене після переходу повного контролю за біржовими торгами до протоплазмових інтелектуальних систем, тобто до людей.

Визначаючи поняття інтелекту як наукову категорію, його слід розуміти як придатність системи до навчання. Таким чином, одне з найбільш конкретизованих, на наш погляд, визначень штучного інтелекту трактується як здатність автоматизованих систем здобувати,

адаптувати, модифікувати та поповнювати знання з метою пошуку рішень задач, формалізація яких ускладнена.

Знання в теорії штучного інтелекту, базах знань та експертних системах являють собою сукупність даних, фактів і правил висновку (у індивідуума, суспільства або у системи штучного інтелекту) про світ, що включають інформацію про властивості об'єктів, закономірності процесів і явищ, а також правила використання цієї інформації для ухвалення рішень. Правила використання включають систему причинно-наслідкових зв'язків. Головна відмінність знань від даних полягає в їх активності, тобто поява в базі нових фактів або встановлення нових зв'язків може стати джерелом змін в ухваленні рішень.

У цьому визначенні термін "знання" має якісну відмінність від поняття інформації. Цю відмінність добре відображає представлення цих понять у вигляді інформаційної піраміди на рис. 7.1.

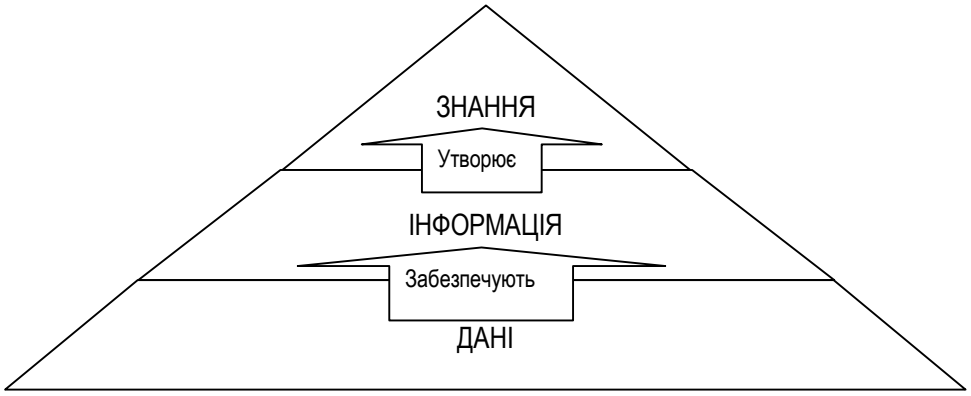

Рис. 7.1. Інформаційна піраміда

В її основі знаходяться дані, наступний рівень займає інформація, завершує піраміду рівень знань. В міру просування вгору по інформаційній піраміді обсяги даних переходять у цінність інформації, і далі – у цінність знань. Тобто інформація виникає у момент взаємодії суб'єктивних даних та об'єктивних методів їх обробки. Знання формуються на основі утворення розподілених взаємозв'язків між різнорідною інформацією, створюючи при цьому формальну систему – спосіб відображення їх у точних поняттях або твердженнях.

Класична класифікація знань виглядає наступним чином:

1. За своєю природою знання можуть бути декларативними та процедурними. Декларативні знання містять в собі лише уявлення про структуру певних понять. Ці знання наближені до даних, фактів. Наприклад, вищий навчальний заклад є сукупністю факультетів, а кожен факультет, у свою чергу, є сукупністю кафедр. Процедурні знання мають активну природу. Вони визначають уявлення про засоби і шляхи

отримання нових знань, перевірки знань. Це алгоритми різного роду. Наприклад: метод мозкового штурму для пошуку нових ідей.

2. За ступенем науковості знання можуть бути науковими і ненауковими. Наукові знання поділяються на: емпіричні – створені на основі досвіду або спостереження; теоретичні – створені на основі аналізу абстрактних моделей. Ненаукові знання можуть поділятися на: паранаукові – несумісні з наявним гносеологічним стандартом; лженаукові – які свідомо експлуатують домисли і забобони; квазінаукові – шукають собі прихильників, спираючись на методи насильства і примусу; антинаукові – утопічні і які свідомо спотворюють уявлення про дійсність; псевдонаукові – інтелектуальна активність, що спекулює на сукупності популярних теорій; повсякденно-практичні – елементарні відомості про природу і навколишню дійсність; власні – які залежать від здібностей того або іншого суб'єкта і від особливостей його інтелектуальної пізнавальної діяльності; "народна наука" – особлива форма позанаукового та ірраціонального знання, яка в даний час стала справою окремих груп або окремих суб'єктів: знахарів, цілителів, екстрасенсів, а раніше – шаманів, жерців, старійшин роду.

3. За місцем знаходження виділяють: неявні знання і формалізовані знання. До неявних знань належать особові (приховані) знання людей. До формалізованих (явних) знань належать: знання в документах, знання в персональних комп'ютерах; знання в Інтернеті.

Підтримка системи знань у такому актуальному стані, який дозволяє будувати програми дій для пошуку рішень поставлених перед ними задач, зважаючи на конкретні ситуації, що утворюються на певний момент часу в навколишньому середовищі, є основною задачею штучного інтелекту. Таким чином, штучний інтелект можна також представити як універсальний надалгоритм, здатний створювати алгоритми рішення нових задач.

7.2. Можливість та доцільність створення штучного інтелекту

Основна філософська проблема у сфері штучного інтелекту полягає в доведенні можливості та доцільності моделювання процесу мислення людини. Існує небезпека витрачати час на вивчення того, що неможливо створити, зокрема, на сучасному етапі розвитку людства. Прикладом подібного марнування часу може бути зайняття науковим комунізмом – наукою, що впродовж десятиріч вивчала те, чого немає, і в досяжному майбутньому бути не може. Розглянемо ряд доказів, які підводять до позитивної відповіді на питання можливості створення штучного інтелекту.

Перший доказ виходить із сфери схоластики і говорить про непротиріччя штучного інтелекту і Біблії. Про це говорять слова священного писання: “І створив Господь Бог людину за своїм образом і подобою”. Виходячи з цього, можна стверджувати, що оскільки людина за своєю сутністю подібна до Творця, то вона цілком може штучним шляхом створити дещо за власним образом і подобою.

Другий доказ витікає з успіхів людства, досягнутих у сфері створення нового розуму біологічним шляхом. У 90-х роках минулого сторіччя з’явилася можливість клонування ссавців, починаючи з ягнички Доллі. Подальші досягнуті успіхи в даному напрямі полягають у створенні форм штучного життя, які не мають жодного природного екземпляру, до якого б вони були подібні. Наприклад, кролики з додатковим геном, що створює ефект світляка. На відміну від клонів, ці форми повною мірою являють собою штучне життя. Разом з тим, такі створіння можна вважати інтелектуальними, зважаючи на їх спроможність до елементарного навчання. Тому вони можуть називатися системами штучного інтелекту, хоча нествореними на основі використання засобів обчислювальної техніки, які становлять найбільший інтерес для людства.

Третій доказ – це доказ можливості самовідтворення об’єктів, що складаються з неживої матерії. Здатність до самовідтворення як ознака наявності інтелекту довгий час вважалася прерогативою живих організмів. Проте деякі явища, що відбуваються в неживій природі, наприклад, зростання кристалів, синтез складних молекул через копіювання, багато в чому ідентичні самовідтворенню.

На початку 50-х років минулого сторіччя Дж. фон Нейман зайнявся ґрунтовним вивченням самовідтворення і заклав основи математичної теорії автоматів, що “самовідтворюються”. Він також довів теоретичну можливість керованої ініціалізації самовідтворення. На сьогодні існує багато різних неформальних доказів можливості самовідтворення об’єктів, але для програмістів найбільш суттєвий доказ полягає в існуванні комп’ютерних вірусів.

Четвертий доказ – це існування принципової можливості автоматизації рішення інтелектуальних задач за допомогою обчислювальної техніки. Вона забезпечується її властивістю алгоритмічної універсальності. Алгоритмічна універсальність обчислювальних машин означає, що на них можна програмно реалізовувати будь-які алгоритми перетворення інформації: обчислювальні алгоритми, алгоритми управління, пошуку доказу теорем тощо. При цьому мається на увазі, що процеси, породжувані цими алгоритмами, є потенційно здійсненними,

тобто вони здійснюються в результаті проведення визначеної кількості елементарних операцій.

Практична реалізація алгоритмів залежить від існуючих обчислювальних потужностей, що змінюються з розвитком техніки. Зокрема, внаслідок появи швидкодіючих комп'ютерів стало практично можливим створення програмних систем, здатних реалізовувати такі алгоритми, які раніше вважались лише потенційно здійсненними.

Для позначення програмних систем, в яких використовується штучний інтелект, склався загальний термін – інтелектуальна система. Доцільність створення інтелектуальних систем полягає в потребі вирішення задач, які не вирішуються на достатньому рівні ефективності програмними системами, створеними на жорсткій алгоритмічній основі. До таких задач належать задачі, що мають, як правило, наступні особливості:

- у них невідомий алгоритм рішення – такі задачі носять назву інтелектуальних задач;
- у них використовується, крім традиційних форматів даних, інформація у вигляді графічних зображень, малюнків, звуків;
- у них передбачається наявність свободи вибору, тобто відсутність єдиного алгоритму рішення задачі зумовлює необхідність зробити вибір між варіантами дій в умовах невизначеності.

Наведений перелік задач формує особливості інтелектуальних систем, призначених для їх вирішення. Джерелом такого визначення особливостей фактично є відомий тест Тьюрінга, запропонований британським математиком й одним з перших дослідників у сфері комп'ютерних наук Аланом Тьюрінгом (Alan Turing). У даному тесті експериментатор, обмінюючись повідомленнями з піддослідним об'єктом, намагається визначити, ким він є насправді – людиною чи комп'ютерною програмою.

Інтелектуальна система, що успішно пройшла такий тест, вважається сильним штучним інтелектом. Термін "сильний штучний інтелект" пропагується фахівцями, які вважають, що штучний інтелект повинен базуватися на суворій логічній підставі. На відміну від сильного, слабкий штучний інтелект, на їх думку, базується виключно на одному з методів рішення інтелектуальних задач (штучних нейронних мережах, генетичних алгоритмах, еволюційних методах). У наші дні стало очевидним, що жоден із методів штучного інтелекту не дозволяє успішно вирішити прийнятну кількість задач – краще проявляє себе використання комбінації методів.

Перша програма, що пройшла тест Тьюрінга, була написана в ході проведення психологічних експериментів Стівеном Вейценбаумом

(Steven Weizenbaum) у 1967 році. З тих пір рівень знань у цій сфері значно зріс, а способи взаємодії експериментатора з об'єктом дослідження стали набагато досконалішими. У наші часи проводяться окремі змагання з призовим фондом у сотні тисяч доларів США під назвою: "Змагання за приз Лебнера", у ході яких визначається краща програма.

Не слід думати, що інтелектуальні системи можуть вирішувати будь-які задачі. Математиками було доведене існування таких типів задач, для яких неможливий єдиний алгоритм, щоб відтворював їхні ефективні рішення. У цьому контексті визначається неможливість рішення задач такого типу за допомогою інтелектуальних систем, розроблених для обчислювальних машин. Крім того, твердження про алгоритмічну неможливість розв'язання деякого класу задач є одночасно й прогнозом на майбутні часи, згідно з яким алгоритми їх вирішення не будуть знайдені ніколи.

Цей факт сприяє кращому розумінню того, де в сучасному світі можуть знайти своє практичне застосування системи штучного інтелекту. Зокрема, для вирішення задачі, що не має універсального алгоритму рішення, доцільним буде її звуження до рівня, коли вона розв'язується тільки для певної підмножини початкових умов. Такі рішення під силу інтелектуальним системам, а їхній результат здатний звузити для людини область варіантів інтуїтивного вибору.

7.3. Сфери практичного застосування систем штучного інтелекту

Серед найважливіших класів задач, які ставилися перед розробниками інтелектуальних систем з моменту визначення штучного інтелекту як наукового напряму (з середини 50-х років XX століття), слід виділити наступні задачі, рішення яких погано піддаються формалізації: доказ теорем, розпізнавання зображень, машинний переклад і розуміння людської мови, ігрові програми, машинна творчість, експертні системи. Коротко розглянемо їхню сутність.

Доказ теорем. Вивчення прийомів доказу теорем відіграло важливу роль у розвитку штучного інтелекту. Багато неформальних задач, наприклад, медична діагностика, застосовують при вирішенні методичні підходи, що використовувались під час автоматизації доказу теорем. Пошук доведення математичної теореми вимагає не тільки провести дедукцію, виходячи з гіпотез, але також створити інтуїтивні припущення про те, які проміжні твердження слід навести для загального доведення основної теореми.

Розпізнавання зображень. Застосування штучного інтелекту для розпізнаванні образів дозволило створювати практично працюючі системи ідентифікації графічних об'єктів на основі аналогічних ознак. Як ознаки можуть розглядатися будь-які характеристики об'єктів, що підлягають розпізнаванню. Ознаки повинні бути інваріантні до орієнтації, розміру та форми об'єктів. Алфавіт ознак формується розробником системи. Якість розпізнавання багато в чому залежить від того, наскільки вдало сформований алфавіт ознак. Розпізнавання полягає в апріорному отриманні вектора ознак для виділеного на зображенні окремого об'єктаі, потім, у визначенні, якому з еталонів алфавіту ознак цей вектор відповідає.

Машинний переклад і розуміння людської мови. Задача аналізу речень людської мови із застосуванням словника є типовою задачею систем штучного інтелекту. Для її вирішення була створена мова-посередник, що полегшує зіставлення фраз із різних мов. Надалі ця мова-посередник перетворилася на семантичну модель представлення значень текстів, що підлягають перекладу. Еволюція семантичної моделі призвела до створення мови для внутрішнього представлення знань. У результаті сучасні системи здійснюють аналіз текстів та фраз у чотири основні етапи: морфологічний аналіз, синтаксичний, семантичний та прагматичний аналіз.

Ігрові програми. В основу більшості ігрових програм покладені декілька базових ідей штучного інтелекту, таких як перебір варіантів і самонавчання. Одна з найбільш цікавих задач у сфері ігрових програм, що використовують методи штучного інтелекту, полягає в навчанні комп'ютера грі в шахи. Вона була започаткована ще на зорі обчислювальної техніки, у кінці 50-х років. У шахах існують певні рівні майстерності, ступені якості гри, які можуть дати чіткі критерії оцінки інтелектуального зростання системи. Тому комп'ютерними шахами активно займалися вчені з усього світу, а результати їх досягнень застосовуються в інших інтелектуальних розробках, що мають реальне практичне значення.

У 1974 році вперше пройшов чемпіонат світу серед шахових програм у рамках чергового конгресу IFIP (International Federation of Information Processing) у Стокгольмі. Переможцем цього змагання стала шахова програма "Каїсса". Вона була створена в Москві, в Інституті проблем управління Академії наук СРСР.

Машинна творчість. До однієї зі сфер застосувань штучного інтелекту можна віднести програмні системи, здатні самостійно створювати музику, вірші, оповідання, статті, дипломи і навіть дисертації. Сьогодні існує цілий клас музичних мов програмування (наприклад,

мова C-Sound). Для різних музичних задач було створене спеціальне програмне забезпечення: системи обробки звуку, синтезу звуку, системи інтерактивної композиції, програми алгоритмічної композиції.

Експертні системи. Методи штучного інтелекту знайшли застосування у створенні автоматизованих консультуючих систем або експертних систем. Перші експертні системи були розроблені як науково-дослідні інструментальні засоби в 1960-х роках минулого сторіччя. Вони були системами штучного інтелекту, спеціально призначеними для вирішення складних задач у вузькій предметній області, такій, наприклад, як медична діагностика захворювань. Класичною метою цього напряму від початку було створення системи штучного інтелекту загального призначення, яка була б здатна розв'язати будь-яку проблему без конкретних знань у предметній області. Зважаючи на обмеженість можливостей обчислювальних ресурсів, ця задача виявилася дуже складною для вирішення з прийнятним результатом.

Комерційне впровадження експертних систем відбулося на початку 1980-х років, і з того часу експертні системи набули значного поширення. Вони використовуються в бізнесі, науці, техніці, на виробництві, а також у багатьох інших сферах, де існує цілком визначена предметна область. Основне значення виразу "цілком визначене" полягає у тому, що експерт-людина здатна визначити етапи міркувань, за допомогою яких може бути вирішена будь-яка задача з даної предметної області. Це означає, що аналогічні дії можуть бути виконані комп'ютерною програмою.

Сьогодні експертні системи є одним з найуспішніших застосувань технології штучного інтелекту. Тому подальше вивчення штучного інтелекту буде здійснюватись у рамках методів, які використовуються в технології створення експертних систем.

ПИТАННЯ ТА ЗАВДАННЯ

1. Дайте визначення штучного інтелекту.
2. Чим зумовлена актуальність інтелектуалізації обчислювальних систем?
3. Побудуйте інформаційну піраміду.
4. У чому полягає якісна відмінність знань від інформації?
5. Доведіть непротиріччя штучного інтелекту і Біблії.
6. У чому полягають успіхи людства, досягнуті у сфері створення нового розуму біологічним шляхом?
7. Назвіть приклади можливості самовідтворення об'єктів, що складаються з неживої матерії.
8. Що означає алгоритмічна універсальність обчислювальних машин?

9. Що розуміють під інтелектуальною системою?
10. Які особливості мають задачі, що належать до компетенції інтелектуальних систем?
11. В чому полягає сутність тесту Тьюрінга?
12. Чи здатні інтелектуальні системи вирішувати (в принципі) будь-які задачі та чому?
13. Яку роль у розвитку штучного інтелекту відіграло вивчення прийомів доказу теорем?
14. Опишіть практичну користь від застосування штучного інтелекту для розпізнавання образів.
15. Яким чином застосовуються методи штучного інтелекту для аналізу речень людської мови?
16. Які ідеї штучного інтелекту покладені в основу більшості ігрових програм?
17. Що розуміють під машинною творчістю?
18. Яка з технологій штучного інтелекту знайшла найбільше практичне застосування?

ТЕСТИ

1. Вкажіть найбільш правильне трактування поняття інтелекту як наукової категорії.
   а) здатність системи до навчання;
   б) здатність системи вирішувати задачі;
   в) здатність системи обробляти інформацію;
   г) здатність системи функціонувати в умовах невизначеності.
2. У чому полягає якісна відмінність знань від інформації?
   а) знання формуються на основі утворення розподілених взаємозв'язків між різнорідною інформацією;
   б) знання виникають в момент взаємодії суб'єктивної інформації з об'єктивними методами її обробки;
   в) інформація виникає в момент взаємодії суб'єктивних знань з об'єктивними методами їх обробки;
   г) інформація формується на основі утворення розподілених взаємозв'язків між різнорідними знаннями.
3. Яка з перерахованих можливостей характеризує штучний інтелект?
   а) незалежність від розробника в знаходженні рішень;
   б) можливість створювати алгоритми рішення нових задач;
   в) здатність реагувати на зміни в навколишньому середовищі;
   г) можливість вибору найбільш відповідного алгоритму рішення задач.

4. Яку з перерахованих розробок не можна віднести до доказу можливості створення штучного інтелекту?
   а) комп'ютерні віруси;
   б) штучні форми біологічного життя;
   в) мікропроцесори на кремнієвій основі.
5. Яка задача є інтелектуальною?
   а) для вирішення якої не існує алгоритмів;
   б) для вирішення якої невідомий алгоритм;
   в) для вирішення якої існує декілька алгоритмів;
   г) для вирішення якої існує єдиний універсальний алгоритм.
6. Що визначає тест Тьюрінга?
   а) чи належить система до штучного інтелекту;
   б) чи володіє система сильним штучним інтелектом;
   в) чи має система можливість використовувати інформацію у вигляді зображень, малюнків, звуків.
7. Який результат очікується від використання інтелектуальної системи?
   а) усунення невизначеності в задачах управління;
   б) виключення людського чинника з процесу знаходження рішень;
   в) звуження для людини області варіантів інтуїтивного вибору.
8. Який з прийомів доказу теорем відіграв важливу роль у розвитку штучного інтелекту?
   а) створення мови для внутрішнього представлення знань;
   б) застосування алфавіту ознак для ідентифікації вектора ознак;
   в) створення в процесі рішення задачі проміжних інтуїтивних гіпотез.
9. Якою була первинна задача, з якої розвинувся напрям експертних систем?
   а) створення системи штучного інтелекту, здатної розв'язати будь-яку проблему на основі знань у предметній області;
   б) створення системи штучного інтелекту, здатної розв'язати будь-яку проблему без наявності знань у предметній області;
   в) створення системи штучного інтелекту, здатної розв'язати конкретну проблему без наявності знань у предметній області.
10. У чому полягає значення виразу "цілком визначений"?
    а) у визначенні практично працюючої системи класифікації об'єктів за аналогічними ознаками;
    б) в алгоритмізації процесу дедуктивної побудови наукових теорій, виходячи з гіпотез;
    в) у визначенні етапів міркувань, за допомогою яких може бути вирішена задача з даної предметній області.

РОЗДІЛ 8. ЕКСПЕРТНІ СИСТЕМИ ЯК РІЗНОВИД СИСТЕМ ШТУЧНОГО ІНТЕЛЕКТУ

8.1. Сутність експертного аналізу

Сучасні дослідження у сфері експертного аналізу сконцентровані на розробці та впровадженні програмних систем, в основу яких покладені методи штучного інтелекту. Для того, щоб програмна система мала можливості експерта, вона повинна відповідати наступним умовам:

- програмна система повинна володіти знаннями, тобто мати до них доступ та уміти їх використовувати;
- знання, якими володіє програмна система, повинні бути спрямовані на певну предметну область;
- на основі цих знань програмна система має бути здатна знаходити способи вирішення проблем;
- програмна система повинна мати змогу поповнювати та оновлювати знання.

Як зазначалось у попередньому розділі, не слід плутати знання з інформацією, і, тим більше, з даними (див. інформаційну піраміду). Наприклад, дані про певний обчислювальний комплекс, систематизовані в посібник з експлуатації, являють собою інформацію. Проте її наявність не дає можливість особі, яка не є фахівцем, відразу виправити ситуацію при збої в роботі комплексу. Лише після детального вивчення посібника і визначення відповідно до цієї проблеми логічних взаємозв'язків між окремими частинами одержаної інформації, можна сподіватися на усунення проблеми. У процесі її усунення відбувається набуття знань, і дилетант робить перший крок до перетворення в експерта.

Підсумок наведених міркувань дає змогу сформулювати визначення експертної системи в наступному вигляді. Експертна система – це програмна система, яка оперує знаннями в певній предметній області з метою вироблення рекомендацій для вирішення проблем. Експертна система може повністю взяти на себе функції, виконання яких вимагає залучення досвіду людини-фахівця, або виконувати роль асистента для людини, що приймає рішення. Людина-фахівець предметної області, що співпрацює з експертною системою, може добитися, з її допомогою, результатів найвищого ґатунку. При цьому правильний розподіл функцій між інтелектом людини та штучним інтелектом є

однією з ключових умов ефективності практичного використання експертних систем.

Перелік типових задач, для вирішення яких призначені експертні системи, включає:

- формування інформації з первинних даних (наприклад, із сигналів, що надходять від гідролокатора);
- діагностика несправностей (як у технічних системах, так і в людському організмі);
- структурний аналіз складних об'єктів (наприклад, хімічних з'єднань);
- вибір конфігурації складних багатокомпонентних систем (наприклад, розподілених комп'ютерних систем);
- планування послідовності виконання операцій, що призводять до заданої мети (у фінансуванні проектів різного рівня ризикованості).

Найбільш відомі експертні системи та області їх практичного застосування наведені в додатку Б.

Слід зазначити, що для вирішення перерахованих задач можуть застосовуватися програмні системи, які необов'язково належать до класу експертних систем. Це можуть бути як традиційні прикладні системи, так і системи штучного інтелекту. Для того, щоб виділити експертні системи в окремий, чітко визначений клас програмних систем, необхідно окреслити набір ознак, які їм притаманні в тій чи іншій мірі. Ці ознаки визначаються в результаті аналізу відмінних характеристик експертних систем.

8.2. Характеристики експертних систем

Експертна система відрізняється від програмних систем із чітко визначеним алгоритмічним спрямуванням завдяки наявності наступних ознак:

1. Експертна система моделює не стільки фізичну (або іншу) природу певної предметної області, скільки механізм мислення людини стосовно рішення задач у цій предметній області. Це істотно відрізняє експертні системи від систем математичного або імітаційного моделювання. Не можна стверджувати, що програма повністю відтворює психологічну модель фахівця певної області (експерта). Важливо те, що в ній основна увага приділяється відтворенню засобами комп'ютерної техніки способу вирішення проблем, що застосовується експертом, тобто реалізації міркувань так само, як це робить експерт.

2. Експертна система, крім виконання обчислювальних операцій, формує певні висновки, ґрунтуючись на тих знаннях, якими вона володіє. Знання в системі представлені, як правило, за допомогою спеці-

альної мови і зберігаються окремо від програмного коду, що формує висновки та міркування. Цей компонент програми прийнято називати базою знань.

3. Під час рішення задач експертною системою, в основному, використовуються евристичні методи, які, на відміну від алгоритмічних, не завжди гарантують успіх. Евристика, по своїй суті, є приблизним правилом, яке в програмному вигляді представляє знання експерта, набуте в міру накопичення практичного досвіду, внаслідок вирішення схожих проблем.

Експертні системи також відрізняються від інших видів систем штучного інтелекту. Ці відмінності полягають у наступному:

1. Експертні системи мають яскраво виражену практичну спрямованість у науковій або господарській діяльності. На відміну від них, інші програми з області штучного інтелекту є суто дослідницькими, і основна увага в них приділяється абстрактним математичним проблемам або спрощеним варіантам реальних проблем (іноді їх називають “іграшковими” проблемами).

2. По-друге, для експертної системи критичною характеристикою є її продуктивність, тобто швидкість отримання результату та рівень його достовірності. Дослідницькі програми штучного інтелекту можуть бути не дуже швидкими, і в них допускаються, в окремих ситуаціях, відмови. Натомість, експертна система повинна за прийнятний час знайти рішення, яке було б не гіршим за те, яке може запропонувати фахівець у відповідній предметній галузі.

3. Експертна система повинна володіти здатністю пояснити, чому запропоноване саме таке рішення, і довести його обґрунтованість. А дослідницькі програми штучного інтелекту надають результат тільки своєму творцю, який і без того (швидше за все) знає, на чому він ґрунтується. Експертна система проектується з розрахунку на взаємодію з різними користувачами, для яких її робота повинна бути, по можливості, прозорою.

Підсумовуючи наведені положення, можна зробити висновок, що експертна система містить знання в певній предметній області, накопичені в результаті практичної діяльності людини, і використовує їх для вирішення проблем, специфічних для цієї області. Цим експертні системи відрізняються від інших систем штучного інтелекту, в яких перевага надається більш загальним і менш пов’язаним з предметними областями теоретичним методам. Процес створення експертної системи часто називають інженерією знань і він розглядається як прикладне застосування методів штучного інтелекту.

8.3. Базові принципи функціонування експертних систем

В основу експертних систем покладені принципи функціонування систем, заснованих на знаннях. До систем, заснованих на знаннях, належать системи, процес роботи яких базується на застосуванні правил відношень до символічного представлення знань.

Принципи роботи експертної системи, заснованої на знаннях, ілюструє схема, представлена на рис. 8.1. Згідно з цією схемою користувач передає в експертну систему факти або іншу інформацію і одержує у результаті поради у вигляді експертних знань.

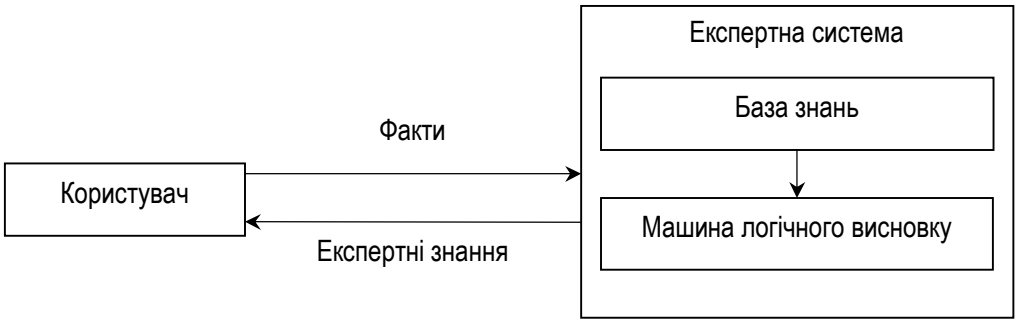

Рис. 8.1. Принципи роботи системи, заснованої на знаннях

За своєю структурою експертна система розподіляється на два основні компоненти: базу знань і машину логічного висновку. База знань містить знання, на підставі яких машина логічного висновку формує висновки. Ці висновки є відповідями експертної системи на запити користувача, що бажає одержати експертні знання.

Знання в експертній системі можуть бути представлені багатьма способами. Одним з широко вживаних методів представлення знань є правила у формі IF THEN. Його сутність полягає у тому, що в разі виявлення факту, який узгоджується з шаблоном правила, повинна бути виконана обумовлена правилом дія. Наприклад, стосовно банківської діяльності з кредитування приватних осіб можна створити наступне правило:

IF людина не має постійного доходу
THEN кредит не видавати

У даному прикладі шаблоном є фраза “людина не має постійної роботи”. У випадку, коли факт відповідає шаблону – рекомендується виконати дію “кредит не видавати”.

Застосовуючи підхід до представлення знань експертів у вигляді правил, було розроблено декілька успішних експертних систем. До них можна віднести систему XCON/R1, створену в компанії Digital

Equipment Corporation (DEC), яка здатна проектувати конфігурації комп'ютерних систем, не поступаючись окремим експертам-людям.

Даний підхід виявився також прийнятним для створення значної кількості порівняно невеликих систем з метою вирішення спеціалізованих задач на основі декількох сотень правил. Такі системи не можуть функціонувати на рівні експерта, проте дозволяють застосовувати інтелектуальні технології для вирішення задач, які не вимагають великих обсягів знань. Знання, що необхідні для подібних невеликих систем, можуть бути отримані з книг, журналів або з іншого джерела загальнодоступної інформації.

На відміну від таких систем, класична експертна система втілює в собі недруковані знання, які повинні бути одержані від експерта за допомогою розширених інтерв'ю, що проводяться інженером по знаннях протягом тривалого періоду часу. Такий процес називається інженерією знань. Інженерія знань – це отримання знань від експерта-людини або з інших джерел з метою подальшого їх представлення в експертній системі. Графічно основні етапи трансформації знань та даних при їх обробці на ЕОМ можна представити у вигляді схеми, показаної на рис. 8.2.

Рис. 8.2. Процес розробки бази знань експертної системи

Спочатку інженер по знанням ініціює діалог з експертом-людиною, щоб виявити знання експерта. Цей етап аналогічний етапу роботи, який виконує розробник традиційної програмної системи в ході обговорення вимог до системи з клієнтом, для якого створюється система. Потім інженер по знанням представляє знання в явному вигляді для внесення в базу знань. Після цього експерт проводить оцінку експертної системи і передає критичні зауваження інженеру по знанням. Такий процес повторюється знову і знову до тих пір, доки експерт не оцінить результати роботи системи як прийнятні.

Розглянутий процес створення експертної системи вважається на сьогоднішній день загальновживаним. Найбільш трудомістким його аспектом є циклічна задача перенесення знань експерта-людини в базу знань експертної системи. Насправді, ця проблема є настільки важли-

вою, що одержала назву “вузького місця” у процесі набуття знань. Цей термін використовується як описовий, оскільки “вузьке місце” у процесі набуття знань впливає на процес створення експертної системи і підкреслює важливість ролі інженера по знанням.

Ще одна проблема, з якою стикаються експертні системи, полягає в тому, що представлені в них експертні знання обмежуються областю знань. Областю знань вважаються знання експерта, що стосуються рішення конкретних задач. Наприклад, банківська експертна система, призначення якої спрямоване на надання рекомендацій щодо видачі кредитів, повинна володіти великим обсягом знань про фізичних та юридичних осіб, охочих отримати кредит. У цьому випадку областю знань є банківська діяльність, а самі знання складаються з відомостей про клієнтів та методів оцінки їх лояльності. Зв’язок між предметною областю і областю знань показаний на рис. 8.3.

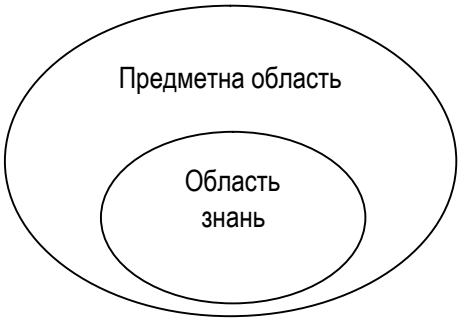

Рис. 8.3. Зв’язок між предметною областю і областю знань

Слід звернути увагу на те, що на рисунку 8.3 область знань повністю включена в предметну область. Частина предметної області, що виходить за межі області знань, символізує область, в якій відсутні знання про рішення задач, що належать до даного сегмента предметної області. Така обмеженість області знань призводить до того, що типові експертні системи недостатньо пристосовані використовувати свої знання для мислення в нових ситуаціях.

Незважаючи на вказані обмеження, експертні системи показали свою здатність успішно вирішувати практичні задачі, які неможливо було розв’язати за допомогою традиційного програмування, особливо в тих умовах, коли доводиться користуватися невизначеною або неповною інформацією. Це означає, що для ефективного використання будь-якої технології важливо знати, які саме переваги можна отримати в результаті.

8.4. Переваги експертних систем

Експертні системи мають наступну низку привабливих особливостей, які дають переваги у разі їхнього практичного використання:

1. Доступність. Для забезпечення доступу до експертних знань можуть застосовуватися тривіальні комп'ютерні апаратні засоби загального користування. У певному значенні цілком виправданим є твердження, що експертна система – це засіб масового виробництва експертних знань.
2. Незначні витрати. Вартість надання експертних знань з розрахунку на одного користувача при застосуванні експертної системи істотно знижується порівняно з експертом-людиною.
3. Зменшена небезпека. Експертні системи можуть використовуватися в такому навколишньому середовищі, яке може виявитися небезпечним для людини.
4. Постійність. Експертні знання об'єктивно нікуди не зникають. На відміну від експертів-людей, які можуть піти на пенсію, звільнитися з роботи або померти, знання експертної системи зберігаються протягом невизначено довгого часу.
5. Можливість отримання експертних знань із багатьох джерел. За допомогою експертних систем можуть бути зібрані знання багатьох експертів, які були залучені до роботи над вирішенням задачі. Рівень експертних знань, скомбінованих шляхом об'єднання досвідів декількох експертів, може перевищувати рівень знань окремо взятого експерта-людини.
6. Надійність. Застосування експертних систем дозволяє підвищити ступінь довіри до того чи іншого висновку шляхом надання додаткової точки зору експерта-посередника в разі непогодження висновків між декількома експертами-людьми. Зрозуміло, такий метод вибору рішень не може використовуватися, якщо експертна система запрограмована одним з експертів, що бере участь у зіткненні думок.
7. Пояснення. Експертна система здатна детально пояснити свої міркування, які привели до певного висновку. На відміну від неї, експерт-людина може виявитися втомленою, не схильною до пояснень або нездатною робити це постійно. Можливість одержати пояснення сприяє підвищенню довіри до того, що системою було запропоноване правильне рішення.
8. Швидка реакція. У деяких випадках вирішення проблеми може вимагати швидких дій або реагування в реальному часі. Залежно від апаратного та програмного забезпечень, які використовує експертна система, вона реагуватиме швидше і буде більш готовою до роботи, ніж експерт-людина. Такі ситуації можуть потребувати більш швидкої реакції, ніж є у пересічної людини. У такому разі прийнятним варіантом стає застосування експертної системи, що здатна діяти в реальному часі.

9. Об’єктивний результат за будь-яких обставин. Ця властивість може виявитися дуже важливою в екстремальних ситуаціях, коли експерт-людина буде нездатна діяти з максимальною ефективністю через стрес або втому.
10. Можливість застосування як навчальної програми. Експертна система може діяти як інтелектуальна навчальна програма, демонструючи учням як приклади рішення задач та пояснюючи, на чому засновані міркування системи.
11. Можливість інтелектуального аналізу баз даних. Експертні системи можуть використовуватися для доступу до баз даних з метою проведення інтелектуального аналізу їх вмісту. Як приклад можна привести пошук прихованих закономірностей у даних.

Крім наведеного списку переваг, процес розробки експертної системи надає також непряму перевагу. Вона витікає із необхідності перетворення знань експертів-людей в явну форму для введення в комп’ютер. Оскільки знання унаслідок цих дій стають явно відомими, а не присутніми неявно в голові експерта, з’являється можливість перевіряти знання на правильність, несуперечливість та логіку. Після такої перевірки, можливо, виникне необхідність відкоригувати ці знання.

ПИТАННЯ ТА ЗАВДАННЯ

1. Яким умовам повинна відповідати програмна система для того, щоб мати можливості експерта?
2. Дайте визначення експертної системи.
3. Назвіть типові задачі, для вирішення яких призначені експертні системи.
4. Наявність яких ознак відрізняє експертні системи від програмних систем із чітко визначеним алгоритмічним спрямуванням?
5. Назвіть ознаки, які відрізняють експертні системи від інших видів систем штучного інтелекту.
6. Який принцип покладений в основу експертних систем?
7. Намалюйте загальну структуру експертної системи.
8. Який спосіб представлення знань в експертних системах вважається класичним?
9. Чи можна вважати програмні системи, що оперують знаннями, отриманими із загальнодоступних джерел, експертними системами і чому?
10. Що таке інженерія знань?
11. Представте процес розробки бази знань експертної системи в графічному вигляді.
12. У чому полягає “вузьке місце” в процесі набуття знань?

13. Опишіть зв’язок між предметною областю та областю знань.
14. Перерахуйте всі переваги застосування експертних систем.
15. У чому полягає доступність експертних систем?
16. Чому однією з переваг експертних систем вважають їх постійність?
17. Чому експертні системи вважаються надійними?

ТЕСТИ

1. Яким умовам повинна відповідати програмна система, щоб володіти здібностями експерта?
   а) програмна система повинна поповнювати та поновлювати інформацію;
   б) система повинна мати доступ до даних та уміти їх використовувати;
   в) знання, якими володіє програмна система, повинні бути спрямовані на певну предметну область;
   г) програмна система має бути здатна знаходити способи рішення задач на основі відомих алгоритмів.
2. У рішенні якої з перерахованих типових задач експертних систем потрібна найменша участь людини?
   а) діагностика несправностей;
   б) структурний аналіз складних об’єктів;
   в) отримання інформації з первинних даних;
   г) планування послідовності виконання операцій, що приводять до заданої мети.
3. Що є основною відмінністю експертних систем від програмних систем імітаційного моделювання?
   а) експертні системи моделюють фізичну природу певної проблемної області;
   б) експертним системам надається перевага при моделюванні складних багатокомпонентних систем;
   в) експертні системи моделюють механізм мислення людини в процесі рішення задач проблемної області;
   г) експертні системи моделюють поведінку об’єктів проблемної області, використовуючи графічні елементи та анімацію.
4. У якому вигляді прийнято представляти знання в експертних системах?
   а) у вигляді програмного коду, написаного спеціальною мовою;
   б) у вигляді взаємопов’язаних структур даних, що зберігаються у спеціалізованих базах даних;
   в) у вигляді евристик, які вбудовуються в програмний код системи, що формує висновки і міркування.

5. Яка з наведених ознак систем штучного інтелекту не належить до відмінних ознак експертних систем?
   а) застосування евристичних методів;
   б) здатність пояснювати запропоновані рішення;
   в) підвищені вимоги до швидкості отримання результату та рівня його достовірності;
   г) явно виражена практична спрямованість у науковій або господарській сферах.
6. Який з принципових елементів системи, заснованої на знаннях, є джерелом фактів?
   а) база знань;
   б) користувач;
   в) машина логічного висновку.
7. Як машина логічного висновку використовує шаблони правил, представлених у формі IF THEN?
   а) зіставляє з ними факти;
   б) вибирає дії, які в них містяться;
   в) зберігає в них типові експертні висновки.
8. Що є основним джерелом знань класичної експертної системи?
   а) бази даних;
   б) експерт-людина;
   в) наукові публікації;
   г) документація з експлуатації системи.
9. Що може бути наслідком значного відставання обсягу області знань експертної системи від предметної області?
   а) неможливість пояснити свої міркування, які призвели до певного висновку;
   б) нездатність використовувати аналогії для міркувань у нових ситуаціях;
   в) виникнення циклічності в процесі перенесення знань експерта-людини в базу знань експертної системи.
10. Яка перевага експертних систем здатна підвищити ступінь довіри до суперечного рішення?
    а) експертна система здатна надати додаткову обґрунтовану версію при зіткненні думок між декількома експертами-людьми;
    б) для забезпечення доступу до експертних знань можуть застосовуватися тривіальні комп'ютерні апаратні засоби;
    в) експертна система може використовуватися в таких умовах середовища, які можуть виявитися небезпечними для людини;
    г) експертна система може реагувати швидше та бути більш готовою до роботи, ніж експерт-людина.

РОЗДІЛ 9. МЕТОДОЛОГІЯ ПРОЕКТУВАННЯ ЕКСПЕРТНИХ СИСТЕМ

9.1. Методологія формалізації знань

У другій половині XX століття очевидним став факт, що ключем до створення програмних вирішувачів задач, здатних функціонувати на рівні експерта-людини, є знання в певній предметній області. Тобто в ході вирішення задач експертом міркування, насправді, виконують незначну роль. Натомість, експерт покладається на обсяг знань, які сформовані на основі досвіду, набутого протягом значного проміжку часу. У випадку, якщо з'являється задача, яку експерт не може вирішити на основі свого досвіду, у нього виникає необхідність здійснити процес міркування, починаючи від загальних принципів та теоретичних положень.

Отже, здібності експерта з позиції техніки міркувань нічим не кращі в порівнянні зі звичною людиною, якій доводиться діяти в абсолютно незнайомій ситуації. Саме тому перші спроби створення універсальних програм, здатних вирішувати будь-які задачі в будь-якій предметній області, закінчились невдачею, оскільки в їх основу була покладена методологія, орієнтована виключно на процес міркування.

Сучасні експертні системи виробляють рішення, базуючись на формальному представленні знань. У методології формалізації знань експертної системи широко застосовується представлення в базі знань їх поверхневого обсягу. Основна причина такого підходу полягає у складності процесу представлення знань будь-якої предметної області, для якої він здійснюються вперше. У знаннях експерта можуть бути несумісності, двозначності, повтори або інші недоліки. Ці недоліки не стають очевидними до тих пір, доки не будуть зроблені спроби формально представити ці знання в експертній системі.

Поверхневі знання мають евристичний характер і, по суті, є емпіричними правилами, набутими на підставі досвіду, які можуть сприяти досягненню рішення з приблизною точністю. В той же час евристики здатні забезпечувати пошук оптимальних шляхів вирішення багатьох задач. Крім того, методологія формалізації знань, що призвела до успішної розробки експертних систем, дозволила створити інтелектуальні системи, які не обов'язково містять експертні знання.

Експертними знаннями вважаються спеціалізовані знання, відомі лише обмеженій кількості осіб. На відміну від них, звичайні знання –

це знання, які можна отримати із загальнодоступних інформаційних ресурсів (книг, Web-сторінок, періодичних видань і т.д.). Наприклад, до звичайних знань можна віднести знання про те, як вирішувати квадратні рівняння або здійснювати диференціювання.

У даний час широко застосовуються програмні системи, засновані на звичайних знаннях, які дозволяють виконувати математичні операції як з числовими, так і з символічними даними. Наприклад: MATLAB, Mathematica тощо. Такі системи не можна віднести до класу експертних, оскільки методи, які лежать в їх основі, не належать до евристичних. Прикладом експертної системи може бути програма, що застосовує експертні знання для оцінки ризиків або інвестиційної привабливості проектів із комерціалізації наукових відкриттів.

Незважаючи на це, в наші дні терміни "система, заснована на знаннях" та "експертна система" часто сприймаються як синоніми. Фактично, останнім часом експертні системи стали розглядатися як підхід до програмування, альтернативний по відношенню до традиційного алгоритмічного програмування.

Вивчення проблеми формалізації знань започатковано в багатьох наукових дисциплінах. Наука про знання називається епістемологією. У рамках цієї науки розглядаються характер, структура і походження знань.

З практичної точки зору, особливо слід виділити область психології, яка присвячена дослідженню процесів сприйняття інформації людиною – когнітологію або науку про пізнання. Вивчення процесу пізнання являє собою вивчення способу, в який люди засвоюють або обробляють інформацію. Для розробки програмних систем, здатних ефективно емітувати роботу експертів-людей, необхідне використання когнітивних методик.

Програмні емулятори експертів призначаються для задач, що не мають задовільного алгоритмічного рішення. Для отримання їх прийнятного рішення використовуються логічні висновки. Ключовою вимогою до експертних систем є реальність висновків та недопущення їх двозначності.

Формування несуперечливих логічних висновків здійснюється на основі методів формальної логіки. Вони базуються на твердженні, що за наявності необхідної кількості істинних фактів висновок завжди повинен бути істинним. На відміну від них, неформальна логіка ґрунтується на методах переконання, що не базуються на фактах, а побудовані на непідтверджених відомостях, які допускають суперечливості.

В основу експертних систем покладені методи формальної логіки. Це визначає ключове значення способу представлення знань в експертних системах, оскільки від нього залежить повнота і точність аналізу фактів. Крім того, від правильного вибору способу такого

представлення залежить весь хід розробки, а також ефективність, швидкодія та зручність супроводу системи.

З тематикою представлення знань тісно пов'язана не менш важлива тематика представлення даних, яка розглядається в проектуванні баз даних. Безумовно, бази даних, в основному, розглядаються як репозитарії поточних даних, а не знань. Тим не менш, вони можуть бути базовим матеріалом для аналізу прихованих закономірностей з метою формування знань.

Ще один важливий науковий підхід, який повинен бути втілений у технології представлення знань, полягає в семантичних розробках, що дозволяють систематизувати значення символів, що їх описують. Крім того, експертні системи повинні володіти здатністю розпізнавати знаки. Тобто вони як представники систем штучного інтелекту мають розуміти основні значення як простих – статичних знаків (наприклад, знаку Stop), так і знаків, що характеризують типові елементи поведінки (наприклад, людина почервоніла – хвилюється). Проведення досліджень у цьому напрямку є основою наукової області, що має назву семіотика.

9.2. Моделювання процесу рішення задач людиною

Найбільш адекватна модель рішення задач людиною була запропонована Ньюеллом і Саймоном в 60-х роках минулого століття під час роботи над програмою GPS – однією із спроб створення універсального вирішувача задач. Незважаючи на те, що проект GPS не досяг поставлених перед ним цілей, дана модель до сьогодні вважається базовою при проектуванні експертних систем. Модель Ньюелла-Саймона складається з наступних компонентів: довготривала пам'ять, короткочасна пам'ять, когнітивний процесор. Розглянемо їх детально.

Довготривала пам'ять. Один із найбільш значущих результатів, отриманих Ньюеллом і Саймоном, полягає в доведенні твердження, що значну частину людського пізнання можна представити у вигляді правил форми IF-THEN. Правило являє собою невелику модульну форму знань, що відповідає певному фрагменту знань у предметній області. Фрагменти організовуються в довільній формі і забезпечуються зв'язками. Нижче наведений приклад правила, що представляє фрагмент знань:

IF людина має постійну роботу і володіє нерухомістю
THEN можна видавати кредит

Ньюелл і Саймон популяризували використання правил для представлення людських знань і показали, як можуть бути виконані стандартні міркування за допомогою правил. Основуючись на цих досягненнях, психологи, що спеціалізуються у області когнітології, ви-

користовували правила як модель для пояснення процесу обробки інформації людиною.

Основна ідея полягає у тому, що мозок людини виділяє стимули із сенсорної вхідної інформації. Стимули активізують у довготривалій пам'яті відповідні правила, за допомогою яких формується відповідна реакція. Довготривала пам'ять виконує роль місця зберігання постійних знань людини. Наприклад, працівники банку засвоїли приблизно такі правила по відношенню до клієнтів:

IF клієнт подав заявку на закриття рахунку THEN банк втратив клієнта
IF клієнт зняв всі гроші з рахунку THEN банк може втратити клієнта
IF клієнт відкрив рахунок в іншому банку THEN банк ймовірно втратить клієнта

З прикладу видно, що останні два правила не виражені з остаточною визначеністю. У даний момент клієнту можуть знадобитися всі гроші, а надалі він знову використовуватиме свій рахунок. Аналогічним чином, відкриття ним в іншому банку рахунку ще не означає, що він не користуватиметься послугами даного банку. Стимули, що виникають, коли співробітник банку дізнається, що клієнт відкрив рахунок в іншому банку, зняв усі гроші з рахунку і подав заявку на закриття рахунку, можуть активізувати також інші правила подібного типу. Наприклад, банк-конкурент знизив тарифи за послуги.

Короткочасна пам'ять. Довготривала пам'ять людини може вміщувати багато правил, що мають просту структуру IF-THEN. На відміну від довготривалої пам'яті, короткочасна пам'ять використовується для тимчасового зберігання знань у період рішення задачі. Незважаючи на те, що довготривала пам'ять може зберігати сотні тисяч (або навіть більше) фрагментів, місткість короткочасної, або робочої пам'яті, значно менша – від чотирьох до семи фрагментів. Як простий приклад, що підтверджує цей факт, можна навести здатність людей одночасно бачити внутрішнім поглядом від чотирьох до семи цифр. Безумовно, люди здатні запам'ятовувати набагато більшу кількість цифр, але ці цифри, поряд із багатьма іншими відомостями, зберігаються в довготривалій пам'яті. Більше того, для забезпечення їх запису на постійне зберігання потрібен час.

Існує теорія, що короткочасна пам'ять здатна вміщувати тільки фрагменти, які можуть бути активними одночасно. Згідно з цією теорією процес розв'язання задачі людиною розглядається як перерозподіл таких активізованих фрагментів у мозку. Зрештою, фрагменти можуть бути активізовані таким чином, що буде вироблена свідома дум-

ка. Наприклад, працівник банку, що розмовляє з потенційним клієнтом, може відзначити про себе: “Він надав про себе суперечливі відомості – можливо, він шахрай”.

Когнітивний процесор. Ще одним елементом, необхідним для вирішення задач людиною, є когнітивний процесор. Когнітивний процесор робить спроби знайти правила, які повинні бути активізовані відповідно до стимулів. У випадку, коли існує багато правил, які можуть бути активізовані одночасно, когнітивний процесор повинен виконати операцію вирішення конфліктів, щоб визначити, яке правило має найвищий пріоритет. Після цього має бути виконане це правило. Наприклад, активізовані два наступні правила:

IF клієнт банку хоче одержати кредит THEN скласти договір
IF клієнт банку хоче одержати кредит THEN ознайомити його з умовами

У такому випадку необхідно спочатку виконати дію другого правила, а потім – першого.

У сучасних експертних системах когнітивному процесору відповідає машина логічного висновку. Важливим чинником при проектуванні експертної системи є обсяг знань або ступінь деталізації правил. У випадку, коли ступінь деталізації дуже малий, то розуміння окремого правила без вивчення інших правил стає ускладненим. Навпаки, якщо ступінь деталізації дуже великий, то виникають проблеми при необхідності модифікації експертної системи, оскільки в одному правилі можуть змішуватися декілька фрагментів знань.

9.3. Методологічні засади створення експертних систем

Виходячи з попереднього матеріалу, стає очевидно, що методологія створення експертних систем відрізняється від методології створення алгоритмічних систем. Це цілком пояснює існування функціональної відмінності між мовами експертних систем та алгоритмічними мовами. Суть відмінності полягає у тому, що покладено в основу представлення.

Алгоритмічні мови програмування націлені на забезпечення підтримки гнучких та надійних методів представлення даних. Наприклад, вони дозволяють легко створювати, а також маніпулювати такими структурами даних, як: масиви, записи, зв’язані списки, стеки, черги, дерева. Сучасні мови, такі як Java і C#, розроблені для надання програмісту істотної допомоги в досягненні абстракції даних. Для цього передбачені структури, що забезпечують інкапсуляцію даних – об’єкти. Тим самим досягається необхідний рівень абстракції, який

потім реалізується в програмі. Дані та методи маніпулювання ними тісно переплітаються в об'єктах.

Розробники мов експертних систем зосереджують свою увагу на підтримці гнучких і надійних способів представлення знань. Підхід до розробки експертних систем передбачає застосування двох рівнів абстракції – абстракції даних і абстракції знань. У мовах експертних систем спеціально передбачене відокремлення даних від методів маніпулювання ними. Прикладом подібного відокремлення є застосування фактів (абстракції даних) і правил (абстракції знань) у мовах експертних систем, орієнтованих на застосування правил.

Цей приклад наглядно ілюструє аналогія з класичним виразом, який оприлюднив Дональд Кнут у своїй книзі з програмування алгоритмічних задач:

Програма = Структура даних + Алгоритми

Стосовно експертних систем цей вираз виглядає таким чином:

Експерта система = Знання + Логічний висновок

Сформульована відмінність у спрямованості між мовами двох типів пояснює різницю в методологіях програмування. З одного боку, в алгоритмічних мовах дані та знання тісно переплітаються, тому програмісти зобов'язані ретельно описувати послідовність виконання операторів програми. З іншого боку, у мовах експертних систем дані явно відокремлені від знань, чим нівелюється необхідність жорсткого контролю над послідовністю виконання коду програми. Як правило, для застосування знань відносно даних використовується повністю відокремлений компонент експертної системи – машина логічного висновку. Таке розділення знань і даних сприяє досягненню високого рівня розпаралелювання та модульності обчислень.

Таким чином, у вигляді загального визначення мова програмування експертних систем представляє собою сукупність команд, записаних із застосуванням визначеного синтаксису та машини логічного висновку, призначеної для їх виконання. Залежно від реалізації, машина логічного висновку може забезпечувати прямий логічний висновок, зворотний логічний висновок або обидва варіанти розробки висновків.

Прямий логічний висновок є методом формування міркувань у напрямку від фактів до висновків, які виходять з цих фактів. Наприклад, якщо перед виходом із будинку виявиться, що йде дощ (факт), то ви повинні взяти із собою парасольку (висновок).

Зворотний логічний висновок передбачає формування міркувань у зворотному напрямі – від гіпотези (потенційного висновку, який повинен бути доведений) до фактів, які підтверджують гіпотезу. Напри-

клад, якщо не виглядати у вікно, але побачити, що хтось увійшов до будинку з вологими черевиками і парасолькою, то можна створити гіпотезу, що йде дощ. Щоб підтвердити цю гіпотезу, достатньо запитати цю людину, чи йде дощ. У разі позитивної відповіді буде доведено, що гіпотеза істинна, тому вона стає фактом. При застосуванні зворотного логічного висновку гіпотеза може розглядатися як факт, істинність якого викликає сумнів, і повинна бути встановлена. У такому разі підтвердження гіпотези (або її спростування) може інтерпретуватися як мета, що має бути досягнута.

Слід відмітити, що створювати експертні системи можна також мовами програмування, які за наведеним визначенням не належать до мов експертних систем, наприклад: C++, Паскаль і навіть асемблер. Доцільність використання тієї чи іншої мови програмування залежить від таких показників, як час розробки, зручність експлуатації, зручність супроводу, ефективність та швидкодія.

Для розробки систем штучного інтелекту були створені спеціалізовані мови, орієнтовані на роботу зі знаннями: LISP, PROLOG, FRL, KRL, SMALLTALK, OPS5, PLANNER, QA4, MACSYMA, REDUCE, CLIPS. Подальша їх еволюція призвела до можливості застосування ідеології об’єктно-орієнтованого підходу до розробки експертних систем.

Поєднання мови програмування та пов’язаних із нею допоміжних програм (утиліт), що дозволяють спростити розробку, налагодження та доставку користувачу програмних продуктів, утворює інструментальний засіб. Сучасні інструментальні засоби розробки експертних систем допускають використання в одній розробці різних підходів, включаючи прямий та зворотний логічні висновки.

Інструментальний засіб експертних систем спеціального призначення, що вимагає від розробника лише представлення бази знань, називається командним інтерпретатором. Командний інтерпретатор надає можливість повторного використання в експертних системах таких програмних компонентів, як машина логічного висновку та користувальницький інтерфейс. Технології командних інтерпретаторів позбавляють від необхідності створювати експертну систему для кожної предметної області, починаючи “з нуля”.

ПИТАННЯ ТА ЗАВДАННЯ

1. Чому спроби створення універсальних програм, здатних вирішувати будь-які задачі в будь-якій предметній області, закінчились невдачею?
2. На чому основується процес пошуку рішень експертної системи?
3. Який характер мають знання експертної системи?
4. Як називається наука про знання?

5. Яка область психології знайшла практичне застосування в експертних системах?
6. У чому полягає різниця між формальною та неформальною логікою?
7. Яким чином методи семіотики використовуються в експертних системах?
8. Назвіть компоненти моделі процесу рішення задач людиною, запропонованої Ньюеллом і Саймоном.
9. В якому вигляді можна представити значну частину людського пізнання згідно з моделлю Ньюелла-Саймона?
10. Як працює довготривала пам'ять?
11. У чому полягає відмінність довготривалої та короткотривалої пам'яті?
12. Яка частина експертної системи виконує роль когнітивного процесора?
13. Наведіть приклад відокремлення даних від методів маніпулювання ними в експертних системах.
14. У чому полягає відмінність між алгоритмічними мовами програмування та мовами розробки експертних систем?
15. Охарактеризуйте методи прямого та зворотного логічного висновків.
16. Чи можна застосовувати для розробки експертних систем мови програмування, які для цього не призначені?
17. Наведіть приклади спеціалізованих мов, орієнтованих на роботу зі знаннями.
18. Які компоненти утворюють інструментальний засіб?
19. Що в експертних системах називається командним інтерпретатором?

ТЕСТИ

1. Що виконує головну роль у процесі рішення нових задач експертною системою?
   а) міркування;
   б) точні знання;
   в) поверхневі знання;
   г) універсальний вирішувач задач.
2. Які знання вважаються експертними?
   а) евристичні знання;
   б) спеціалізовані знання професіоналів;
   в) знання, що містяться у спеціалізованих базах даних;
   г) знання, що містяться в загальнодоступних інформаційних ресурсах.

3. Яка з перерахованих вимог є ключовою стосовно логічних висновків експертних систем?
   а) недопущення двозначності;
   б) недопущення використання звичних знань;
   в) недопущення посилань на суперечливі відомості;
   г) недопущення застосування символічних обчислень.
4. Яка логіка ґрунтується на переконаннях?
   а) машинна логіка;
   б) формальна логіка;
   в) неформальна логіка.
5. В рамках якої науки розглядаються характер, структура та походження знань?
   а) семіотика;
   б) когнітологія;
   в) епістемологія.
6. Який із перерахованих компонентів моделі рішення задач людиною, запропонованої Ньюеллом і Саймоном, є місцем зберігання знань?
   а) короткочасна пам'ять;
   б) довготривала пам'ять;
   в) когнітивний процесор.
7. Що розуміється під правилами, які визначають людське знання?
   а) фрагменти пам'яті;
   б) керівництво до дій;
   в) елементи знайдених рішень;
   г) стимули із сенсорної вхідної інформації.
8. Результатом якого процесу є свідома думка?
   а) деталізація правил у когнітивному процесорі;
   б) послідовне розміщення стимулів у довготривалій пам'яті;
   в) розподіл активованих фрагментів у короткочасній пам'яті.
9. Який підхід є характерним для мов розробки експертних систем?
   а) відділення даних від методів маніпулювання даними;
   б) об'єднання даних і методів маніпулювання даними в об'єктах;
   в) представлення даних у таких структурах, як: масиви, записи, зв'язні списки, стеки, черги, дерева.
10. Яка з перерахованих мов не підходить для створення експертних систем?
    а) CLIPS;
    б) Паскаль;
    в) PROLOG;
    г) усі підходять.

РОЗДІЛ 10. ТЕХНОЛОГІЯ ПРОЕКТУВАННЯ ЕКСПЕРТНИХ СИСТЕМ НА ОСНОВІ ПРОДУКЦІЙНОЇ МОДЕЛІ

10.1. Продукційна модель експертних систем

Як було зазначено в попередньому розділі, експертними системами найбільш поширеного типу є системи, засновані на правилах. Правила, що організовані у вигляді IF-THEN структур, називаються продукційними правилами. Продукційні правила, разом з інтерпретатором, який управляє їх активізацією в залежності від наявних фактів, складають продукційну модель представлення та використання знань в експертних системах. Такі системи носять назву продукційних.

У продукційних системах знання представлені у формі множини правил, на основі яких формуються висновки, що повинні бути зроблені (або не зроблені) в різних ситуаціях. Висновки робляться на основі методів прямого або зворотного логічного висновку. Залежно від методу логічного висновку розрізняють два види продукційних систем: системи з прямим логічним висновком та системи із зворотним логічним висновком.

Загальна стратегія вирішення задач полягає в розбитті їх на фрагменти, які можна легше довести. При цьому системи з прямим логічним висновком знаходяться під управлінням фактів. Вони починають свою роботу з відомих початкових фактів і продовжують, використовуючи правила для створення висновків або виконання певних дій. Системи із зворотним логічним висновком керуються гіпотезами. Вони починають свою роботу з гіпотези, або мети, яку користувач намагається довести і продовжують, відшукуючи правила, які дозволять довести правдивість гіпотези.

Широке застосування систем, заснованих на продукційних правилах, обумовлене наявністю в них наступних особливостей:

1. Модульна організація. Завдяки модульній організації спрощується представлення знань та розширення експертної системи, нарощуючи її можливості крок за кроком.
2. Наявність засобів пояснення. Продукційні експертні системи за допомогою правил дозволяють легко створювати засоби пояснення. Засіб пояснення відстежує послідовність активованих правил і, на цій основі, дає можливість відновити хід міркувань, які привели до певного висновку.
3. Наявність аналогії з пізнавальним процесом людини. Згідно з результатами, одержаними Ньюеллом і Саймоном, правила є приро-

дним способом моделювання процесу рішення задач людиною. Тому в процесі виявлення експертних знань не виникають зайві складнощі в поясненні експертам структури представлення знань, оскільки застосовується просте їх представлення у вигляді правил IF-THEN.

Загальну структуру експертної системи, заснованої на правилах, можна представити у вигляді схеми, зображеній на рис. 10.1.

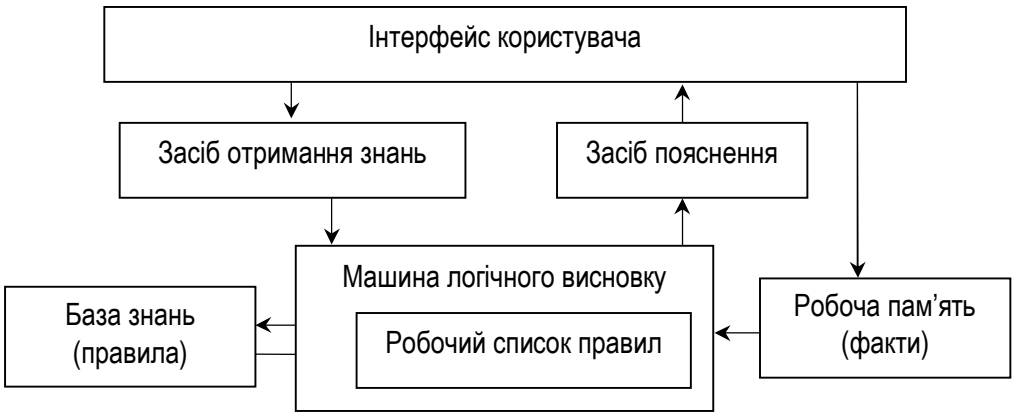

Рис. 10.1. Загальна структура експертної системи, заснованої на правилах

Наведена структура наглядно ілюструє основні аспекти продукційної моделі експертних систем, тому розглянемо детально сутність та призначення її компонентів.

Інтерфейс користувача. Інтерфейс користувача – це механізм, за допомогою якого відбувається спілкування користувача з експертною системою. Залежно від призначення системи інтерфейс користувача може використовувати простий текстовий дисплей або складний растровий дисплей із високою роздільною здатністю. Дисплеї з високою роздільною здатністю, зазвичай, застосовуються для задач моделювання, які вирішує експертна система.

Засіб отримання знань. Засіб отримання знань являє собою автоматизований спосіб, який дозволяє користувачу вводити знання в систему, не застосовуючи явного кодування знань за допомогою інженера по знанням. Цей інструментальний засіб у деяких експертних системах здатний навчатися, здійснюючи автоматичне формування правил на підставі прикладів. Для формування правил у машинному навчанні застосовуються такі методи й алгоритми, як ID3, C4.5, C5.1, штучні генетичні алгоритми, нейронні мережі.

База знань. Системи, що відповідають продукційній моделі, зберігають знання, необхідні для вирішення задач, у певній проблемній

області в базі знань. Базу знань експертної системи, в якій знання закодовані у формі правил, називають продукційною пам'яттю. У ній правила виражені у форматі продукційного псевдокоду IF-THEN.

Кожне правило позначається ім'ям. Після нього починається IF-частина правила. Ця частина продукційного правила розташована між ключовими словами IF і THEN та носить назву антецедента або лівої частини (LHS – left-hand-side) правила. На практиці застосовуються також назви "умовний елемент" та "шаблон".

За частиною IF починається частина THEN правила. Вона містить висновки або список дій, які повинні бути виконані згідно з правилом. Ця частина правила називається консеквентом або правою частиною (RHS – Right-Hand Side). До складу дій консеквентів правил, зазвичай, входять додавання або видалення фактів із робочої пам'яті, або формування результатів. Формат опису цих дій залежить від синтаксису мови експертної системи.

Машина логічного висновку. Машина логічного висновку є програмним компонентом, який визначає антецеденти правил (якщо такі існують), що виконуються згідно з фактами. Для цього машина логічного висновку виконує наступні дії:

- обирає правила, яким відповідають факти;
- розподіляє обрані правила за пріоритетами;
- виконує правило з найвищим пріоритетом.

Як класичні стратегії рішення задач в експертних системах використовуються два загальні методи логічного висновку: прямий логічний висновок і зворотний логічний висновок. У число інших методів, що застовуються для виконання конкретизованих завдань, можуть входити: аналіз цілей та засобів, спрощення задачі, перебір з поверненнями, метод "запланувати-виробити-перевірити", ієрархічне планування.

Робоча пам'ять. Робоча пам'ять призначена для розміщення фактів, що стосуються поточного стану об'єкта досліджень. Факти, що знаходяться в робочій пам'яті, не взаємодіють один з одним, на відміну від правил, що зберігаються в базі знань. Якщо в робочій пам'яті є факт, який відповідає умовній частині правила, машина логічного висновку розміщує це правило в робочому списку правил.

У випадку, якщо правило має декілька шаблонів, то для того, щоб правило можна було розмістити в робочому списку правил, усі ці шаблони повинні бути розпізнані як відповідні. Як умову відповідності деяких шаблонів можна назвати відсутність певних фактів у робочій пам'яті. Машина логічного висновку працює в режимі здійснення циклів "розпізнавання-дія".

Робочий список правил. Робочий список правил являє собою створений машиною логічного висновку та розташований за пріоритетами список правил, шаблони яких відповідають фактам, що знаходяться в робочій пам'яті. Правило, всі шаблони якого розпізнані як відповідні, називається активізованим або реалізованим. У робочому списку правил можуть бути одночасно присутні декілька активізованих правил. У цьому випадку машина логічного висновку повинна вибрати, залежно від пріоритету, одне з правил для запуску дії.

Після завершення виконання всіх правил управління повертається до інтерпретатора команд верхнього рівня, щоб користувач міг надати командному інтерпретатору експертної системи додаткові інструкції. Роль верхнього рівня системи виконує інтерфейс користувача, який, по суті, є механізмом інтерпретації команд користувача.

Засіб пояснення. Головною особливістю експертної системи є передбачений у ній засіб пояснення, який відображає інформацію про те, як система дійшла певного висновку. У системах, заснованих на правилах, нескладно організувати пояснення, яким чином був отриманий певний висновок, оскільки хронологія активізації правил і вміст робочої пам'яті можна зберігати в стеку. Розвинені засоби пояснення можуть дати користувачу можливість ставити питання на зразок "що?", "якщо?" і вивчати альтернативні шляхи формування висновків за принципом гіпотетичних міркувань.

10.2. Методи практичної реалізації концепції продукційних правил

Методи реалізації продукційних правил пройшли еволюційний процес із середини XX століття. Оскільки спосіб представлення знань на основі продукційних правил є класичним засобом створення експертних систем, їх розробникам необхідно володіти системною інформацією щодо методичного змісту етапів розвитку концепції правил.

Продукційні системи були вперше використані в символічній логіці американським логіком Емілем Постом (Emil Post), тому ім'я цього вченого увійшло до назви вказаних систем. Основна ідея Поста полягала в тому, що на основі логічної та математичної систем можна представити набір правил, який встановлює порядок перетворення рядка символів в інший послідовний набір символів. Це означає, що продукційне правило, після отримання вхідного рядка (антецедента), здатне виробити новий рядок (консеквент), наприклад:

Клієнт має постійний дохід → клієнт кредитоспроможний

У цьому правилі стрілка означає, що один символьний рядок повинен бути перетворений в інший. Вказане правило можна інтерпретувати за допомогою системи позначень IF-THEN наступним чином:

IF клієнт має постійний дохід THEN клієнт кредитоспроможний

Слід зазначити, що маніпуляції з рядками засновані на синтаксисі, а не на семантиці. Іншими словами, продукційна система Поста застосовується лише як спосіб перетворення одного рядка в інший, без розуміння значення слів "постійний дохід" та "клієнт кредитоспроможний".

Продукційні правила також можуть мати декілька антецедентів, що розділяються зв'язкою AND, як показано в наступному прикладі:

Клієнт має постійний дохід AND він володіє нерухомістю вартістю 1000000$ → видати кредит

Безумовно, продукційні правила Поста були необхідні для формування певної частини фундаменту експертних систем, але вони не були достатніми для програмування продуктів, що мають практичну цінність. Основним обмеженням продукційних правил Поста, з погляду програмування, є відсутність стратегії управління, яка дозволяла б регламентувати застосування правил. Зрозуміти це обмеження можна легко, уявивши гіпотетичну ситуацію з візитом в бібліотеку. Якщо для пошуку потрібної книги не використовувати жодної системи управління процесом, можна згаяти безліч часу, переглядаючи усі можливі варіанти.

Наступним значним кроком у розробці методів застосування продукційних правил стало відкриття, зроблене Марковим, яке дозволило визначити структуру управління для продукційних систем. Марківський алгоритм – це впорядкована група продукцій, які застосовуються згідно з пріоритетами до вхідного символьного рядка. Якщо правило з найвищим пріоритетом є непридатним, то використовується наступне правило з нижчим пріоритетом і т.д. Марківський алгоритм завершує свою роботу після виявлення однієї з наступних умов: по-перше, остання продукція не була застосовна до рядка, або, по-друге, була застосована продукція, яка закінчується крапкою.

Марківські алгоритми можуть також застосовуватися до сегментів символьних рядків, починаючи зліва. Наприклад, продукційна система складається з одного правила:

ЗА → ВИ

Після її застосування вхідний рядок "ЗАЛУЧИТИ" перетворюється на новий рядок "ВИЛУЧИТИ". Оскільки тепер продукція засто-

совується до нового рядка, остаточним результатом стає рядок “ВИЛУЧИТИ”.

Марківські алгоритми застосовують спеціальні символи. Зокрема, спеціальний символ ^ позначає порожній рядок, що не містить символів. Наступна продукція видаляє всі входження символу A у вхідному рядку:

$$A \rightarrow \text{^}$$

Інші спеціальні символи Марківського алгоритму можуть представляти інші набори символів та позначаються буквами a, b, c, …, y, z. Ці символи є односимвольними змінними і являють собою важливу частину сучасних мов експертних систем. Наприклад, наступне правило змінює місцями символи A та B в рядку, якщо між ними знаходиться будь-який одиничний символ:

$$AxB \rightarrow BxA$$

Як спеціальні знаки пунктуації у Марківському алгоритмі використовуються грецькі літери б, β і т.д. Вони застосовуються тому, що їх можна легко відрізнити від звичайних літер алфавіту.

Марківський алгоритм може застосовуватися як основа експертної системи, але він не є достатньо ефективним способом створення систем із великою кількістю правил. Такі системи потребують алгоритму, який має повну інформацію про всі правила та може застосувати будь-яке з них, не роблячи спроби послідовно перевіряти кожне.

Рішенням цієї проблеми є rete-алгоритм, розроблений Чарльзом Л. Форгі (Charles L. Forgy) в 1979 році. Термін “rete-алгоритм” походить від латинського слова rete, яке означає мережу. Згідно з назвою rete-алгоритм функціонує як мережа, призначена для зберігання великого обсягу знань. Він заснований на використанні динамічної структури даних, яка автоматично реорганізується з метою оптимізації пошуку аналогічно B-дерева, що застосовується при індексації структур реляційних баз даних.

Rete-алгоритм є високошвидкісним засобом порівняння фактів із шаблонами, швидкодія якого досягається завдяки зберіганню в оперативній пам’яті інформації про правила, що знаходяться в мережі. В rete-алгоритмі втілені два емпіричні спостереження, на підставі яких була запропонована структура даних, покладених у його основу:

1. Тимчасова надмірність. Дія, що виникає в результаті запуску одного з правил, зазвичай, пов’язана з декількома фактами.
2. Структурна подібність. Один і той самий шаблон часто знаходиться в лівій частині більш ніж одного правила.

В rete-алгоритмі в циклах “розпізнавання-дія” контролюються тільки зміни в узгодженнях, тому в кожному циклі немає потреби погоджувати факти з кожним правилом. Завдяки цьому істотно підвищується швидкість узгодження фактів з антецедентами, оскільки статичні дані, які не змінюються від циклу до циклу, можуть бути проігноровані.

Необхідною умовою практичної реалізації концепції продукційних правил є використання формальної системи визначення продукцій. У загальному вигляді під формальною системою визначення розуміють систему позначень, яка виконує роль метамови для визначення синтаксису інших мов. Мови поділяються на декілька типів: природні мови, логічні мови, мови математики, комп’ютерні мови тощо. Синтаксис мови визначає її форму, а семантика – значення її слів.

Однією із формальних систем позначень, що використовуються для визначення продукцій, є нормальна форма Бекуса-Наура (Backus-Naur form – BNF). Для її детального розгляду скористаємося простим правилом граматики англійської мови, представленим у вигляді продукційного правила:

<sentence>::= <subject> <verb> <end-mark>

Згідно з системою позначень форми Бекуса-Наура речення (sentence) складається з підмета (subject) та присудка (verb), за якими йде знак пунктуації, що позначає кінець речення (end-mark). У цьому правилі кутові дужки (<>) та символ ”: : =” є символами метамови. Символ ”::=” означає “визначено як”. Він використовується замість символу “→”, який застосовувався в Марківських алгоритмах продукційних правил.

Інтуїтивно визначений вираз метамови, що являє собою формальний ідентифікатор об’єкта, носить назву терма. Терми, поміщені в кутові дужки, прийнято називати нетермінальними символами. Нетермінальний символ – це змінна, що представляє інший терм. У свою чергу, в якості такого “іншого” терму може використовуватися нетермінальний або термінальний символ.

Термінальний символ не може бути замінений іншим символом і тому є константою. З його допомогою реалізується остання ланка ланцюга утворень. Наприклад, наступні правила дозволяють розкривати значення нетермінальних символів, оскільки вказують на термінальні символи, якими вони можуть бути замінені:

<subject> ::= Гривня | Долар | Євро

<verb> ::= дорожчає | дешевшає

<end-mark> ::= . | ! | ?

У даній метамові вертикальна риска означає “або”.

Усі можливі речення мови, тобто продукції, можуть бути створені шляхом послідовної заміни кожного нетермінального символу відповідними йому нетермінальними або термінальними символами, взятими з правої частини правил, до тих пір, доки не будуть усунені всі нетермінальні символи. Як приклад застосування продукцій даної мови можна навести такі речення:

Гривня дорожчає. Долар дешевшає! Євро дешевшає?

Послідовність термінальних символів називається рядком символів мови. Якщо рядок символів може бути одержаний з початкового символу шляхом заміни нетермінальних символів із застосуванням правил, що їх визначає, то такий рядок називається дійсним реченням. В іншому випадку рядок не вважається дійсним реченням, наприклад:

Євр одешевшаєдешевшає

Повний набір продукційних правил, що однозначно визначає мову, називається її граматикою. Розглянуті в прикладі правила визначають деяку граматику, але вона є дуже обмеженою, оскільки забезпечує створення невеликої кількості можливих продукций. Складність граматики збільшується за рахунок різних доповнень, наприклад:

<sentence>::= <subject> <verb> <object> <end-mark>

<object> ::=на 1% | на 5% | на 10%

Однак, незважаючи на рівень досягнутої складності граматики, принцип системи визначення продукцій залишається незмінним.

Як приклад практичної реалізації концепції продукційних правил наведемо фрагмент програми управління надзвичайними ситуаціями на підприємстві, яка реалізована в системі CLIPS. Експертна система в системі CLIPS може виконувати змістовну роботу тільки в тому випадку, якщо в ній присутні факти і правила. В нашому прикладі розглянемо, які типи фактів і правил можуть виявитися корисними в тій ситуації, коли доводиться здійснювати поточний контроль і реагувати на декілька можливих надзвичайних ситуацій. Однією з таких надзвичайних ситуацій може виявитися пожежа, а іншою – повінь. Нижче наведений псевдокод одного з можливих правил в експертній системі поточного контролю на підприємстві.

IF the emergency is a fire

THEN the response is to activate the sprinkler system

Перш ніж перетворити цей псевдокод в правило, необхідно визначити конструкції deftemplate для фактів такого типу, які згадуються в цьому правилі. Надзвичайна ситуація може бути представлена за допомогою наступної конструкції deftemplate:

(deftemplate emergency (slot type))

У цій конструкції поле *type* факту *emergency* повинно містити такі символи, як *fire* (пожежа), *flood* (повінь) і *power outage* (припинення подачі електроенергії). Аналогічним чином, відповідь експертної системи *(response)*, може бути представлена наступною конструкцією *deftemplate*:

(deftemplate response (slot action))

У даній конструкції поле *action* факту *response* указує на те, яка відповідь повинна бути вироблена системою.

Правило, виражене за допомогою синтаксису CLIPS, наведене нижче, в ньому використовуються коментарі, відповідні окремим частинам правила. Коментарі починаються з крапки з комою і продовжуються до кінця рядка. Коментарі ігноруються системою CLIPS. Правило можна ввести, набираючи його текст у вбудованому редакторі системи CLIPS:

```
; Заголовок правила
(defrule fire-emergency “An example rule”
; Шаблони
(emergency (type fire))
; Стрілка THEN
=>
; Дії
(assert (response
(action activate-sprinkler-system))))
```

У правилі присутній один шаблон і одна дія. Заголовок правила складається з трьох частин. Правило повинно починатися з ключового слова defrule, за яким йде ім’я правила – *fire-emergency*. За заголовком правила слідує від нуля і більше умовних елементів. Простим типом умовного елемента є умовний елемент шаблону, або просто шаблон. Кожен шаблон складається з одного або декількох обмежень, які призначені для зіставлення з полями факту, заданого за допомогою конструкції deftemplate. У правилі *fire-emergency* шаблоном є (*emergency* (*type fire*)). Обмеження для поля *type* указує на те, що це правило задовольняється тільки для фактів *emergency*, що містять в своєму полі *type* символ *fire*. Система CLIPS робить спроби зіставити шаблони правил з фактами в списку фактів. Якщо всі шаблони правила узгоджуються з фактами, правило активізується і потрапляє в робочий список правил – в колекцію активізованих правил. У робочому списку правил може знаходитися від нуля і більше правил.

Символ =>, який іде за шаблонами в правилах, називається стрілкою. Стрілка – це символ, що представляє початок частини THEN правила IF-THEN. Частина правила, що знаходиться перед стрілкою, називається лівою частиною (Left-Hand Side — LHS), а частина, що знаходиться за стрілкою, називається правою частиною (Right-Hand Side — RHS).

Останньою частиною правила є список дій, що виконуються після запуску правила. У даному прикладі одна з дій передбачає внесення факту (*response*(*action activate-sprinkler-system*)) в список фактів. Термін "запуск" означає, що система CLIPS виконує дії одного з правил, що знаходяться в робочому списку правил. Звичайно програма за відсутності правил в робочому списку правил припиняє своє виконання. Якщо ж в робочому списку правил є декілька правил, то система CLIPS автоматично визначає, яке з правил є відповідним для запуску. Система CLIPS упорядковує правила, що знаходяться в робочому списку правил, з урахуванням зростання пріоритету і запускає правило з найвищим пріоритетом.

ПИТАННЯ ТА ЗАВДАННЯ

1. Які системи мають назву продукційних?
2. Охарактеризуйте загальну стратегію вирішення задач в експертних системах, побудованих за продукційною моделлю.
3. Які особливості систем, заснованих на продукційних правилах, обумовили їх широке застосування?
4. Намалюйте схему загальної структури продукційної експертної системи.
5. Що являє собою засіб отримання знань продукційної експертної системи?
6. Опишіть базу знань продукційної експертної системи.
7. Які дії виконує машина логічного висновку продукційної системи?
8. Як працює робоча пам'ять продукційної системи?
9. З якою метою в продукційній системі формується робочий список правил?
10. Охарактеризуйте продукційні системи Поста.
11. Що являє собою Марківський алгоритм?
12. Назвіть символи, які найчастіше використовуються в Марківських алгоритмах.
13. Для вирішення якої проблеми призначений rete-алгоритм?
14. Назвіть емпіричні спостереження, на підставі яких була запропонована структура даних, покладена в основу rete-алгоритму.

15. Для чого призначена нормальна форма Бекуса-Наура?
16. Наведіть приклади позначень форми Бекуса-Наура.
17. У чому полягає відмінність термальних та нетермальних символів форми Бекуса-Наура?
18. Як називається повний набір продукційних правил, який однозначно визначає мову?

ТЕСТИ

1. Яке з перерахованих тверджень правильно характеризує продукційну систему?
   а) продукційна система використовує продукційні правила спільно з інтерпретатором, який управляє їх активізацією, залежно від наявних фактів;
   б) в продукційних системах знання представлені у формі продукційних правил, які здійснюють обробку фактів в процесі підготовки рішення;
   в) продукційна система використовує факти, організовані у вигляді IF-THEN структур, для набуття знань – продукцій.
2. Чим управляється система із зворотним логічним висновком?
   а) фактами;
   б) знаннями;
   в) гіпотезами;
   г) правилами.
3. Який компонент продукційної моделі експертних систем може бути здатним здійснювати автоматичне формування правил на основі прикладів?
   а) робоча пам’ять;
   б) засіб пояснення;
   в) засіб отримання знань;
   г) машина логічного висновку.
4. Як називається частина продукційного псевдокоду, що містить список дій, які повинні бути виконані відповідно до правила?
   а) база знань;
   б) консеквент;
   в) антецедент;
   г) ліва частина.
5. Яким чином у продукційній експертній системі організована взаємодія фактів?
   а) вони не взаємодіють;

б) вони поміщаються в стек і застосовуються як засіб пояснення;
в) вони поміщаються в робочу пам'ять і розподіляються за пріоритетами;
г) вони взаємодіють, використовуючи загальні шаблони правил, що знаходяться в робочому списку.

6. На чому засновані маніпуляції із рядками символів у продукційних системах Поста?
   а) граматиці;
   б) семантиці;
   в) синтаксисі;
   г) прямому та зворотному логічному висновках.
7. Що розуміють під Марківським алгоритмом?
   а) високошвидкісний засіб порівняння фактів із шаблонами правил;
   б) впорядковану групу продукційних правил, які застосовуються до символьного рядка;
   в) систему математичних та логічних дій для визначення правила з найвищим пріоритетом.
8. Яке емпіричне спостереження, втілене в реалізацію rete-алгоритму, носить назву "структурна подібність"?
   а) один і той самий шаблон може знаходитись у лівій частині більш ніж одного правила;
   б) зміни, що виникають у результаті запуску одного з правил, зазвичай, стосуються лише декількох фактів;
   в) кожна із змін, що виникає в результаті запуску одного з правил, впливає тільки на декілька правил.
9. Що розуміють під нормальною формою Бекуса-Наура?
   а) метод управління надмірністю продукційних правил у базі знань;
   б) формальну систему позначень, що використовується для визначення продукцій;
   в) метамову, що застосовується для опису синтаксису мов розробки експертних систем.
10. Який символ форми Бекуса-Наура називається нетермальним?
    а) змінна;
    б) константа;
    в) компонент, що не є термом.

РОЗДІЛ 11. ТЕХНОЛОГІЯ ПРОЕКТУВАННЯ ЕКСПЕРТНИХ СИСТЕМ НА ОСНОВІ ЛОГІЧНОЇ МОДЕЛІ ПОДАННЯ ЗНАНЬ

11.1. Представлення знань на основі формальної логіки

В експертних системах для представлення знань, окрім правил, можуть використовуватися символи формальної логіки. Нагадаємо, що логіка – це наука, яка вивчає правила формування обґрунтованих міркувань. Термін "формальний" означає, що логіка, до якої він належить, розповсюджується тільки на форму логічних тверджень, але не враховує їх значення. Іншими словами, у формальній логіці розглядається тільки синтаксис тверджень і не розглядається їх семантика. В результаті відділення форми від семантики з'являється можливість об'єктивно оцінювати правильність доказу, не піддаючись дії упереджень, викликаних семантикою.

Аналогією по відношенню до формальної логіки може розглядатися алгебра, в якій правильність таких виразів, як X + X = 2X залишається незаперечною, незалежно від того, що позначає X: кількість яблук або аеропланів. Така властивість формальної логіки є корисною при створенні експертних систем, оскільки дозволяє відділити знання від міркувань. Тобто вислови, які, на перший погляд, виглядають, як міркування, можуть, у дійсності, виконувати роль знань.

Важливою частиною процесу проведення міркувань є логічне виведення висновків із посилань. На цьому ґрунтується найбільш рання система формальної логіки, яка була розроблена давньогрецьким філософом Аристотелем у IV ст. до н.е. Ключовим поняттям Аристотелевої логіки є силогізм. У силогізмі посилання виконують роль свідоцтв, з яких повинен обов'язково бути сформований висновок. Силогізми мають два посилання і один висновок, який з них витікає. Класичний приклад силогізму може мати наступний вигляд:

Посилання 1: Усі люди смертні

Посилання 2: Викладач – людина

Висновок: Викладач смертний

Для графічного способу представлення силогізмів добре підходять діаграми Венна. Приклад представлення знання, приведеного в силогізмі, у вигляді діаграми Венна зображений на рис. 11.1.

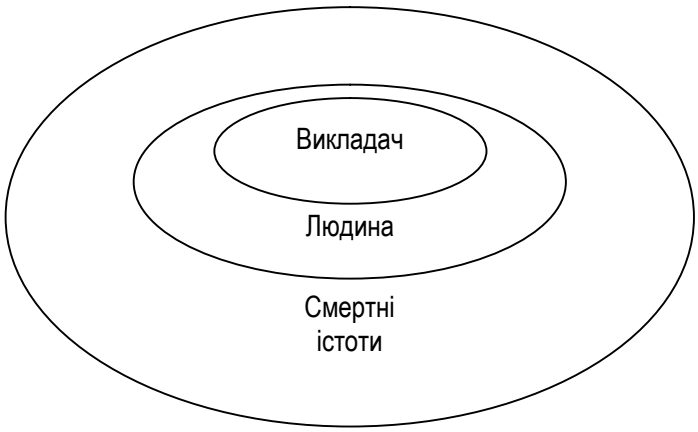

Рис. 11.1. Представлення силогізму у вигляді діаграми Венна

З погляду математики, кожен еліпс на діаграмі Венна представляє певну множину, тобто колекцію об'єктів. Таким чином, механізм теорії множин може бути успішно застосований для формального представлення знань у логічній моделі. Діаграми Венна, що зображують основні операції над множинами, представлені на рисунках 11.2, 11.3, 11.4.

У теорії множин певну зручність надає практика вважати досліджувані множини підмножинами однієї універсальної множини – універсуму. Графічно універсум зображується як прямокутник, що оточує свої підмножини. Найбільш уживані символи в теорії множин наступні:

- символом у вигляді грецької букви епсилон ($\in$) позначається приналежність об'єкта або множини до іншої множини;
- символом $\cap$ – перетин двох множин;
- символом $\cup$ – об'єднання двох множин;
- символом штрих ($'$) – доповнення множини.

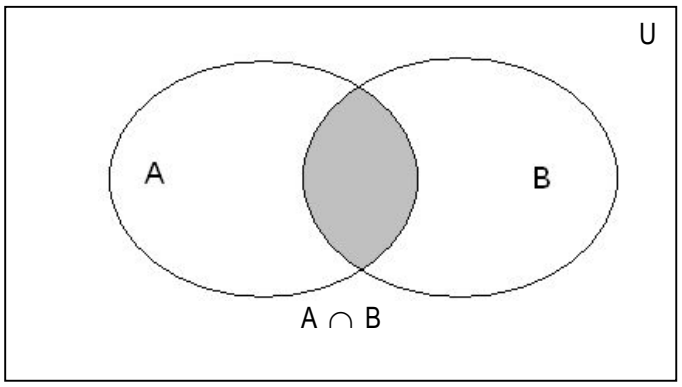

Рис. 11.2. Перетин множин

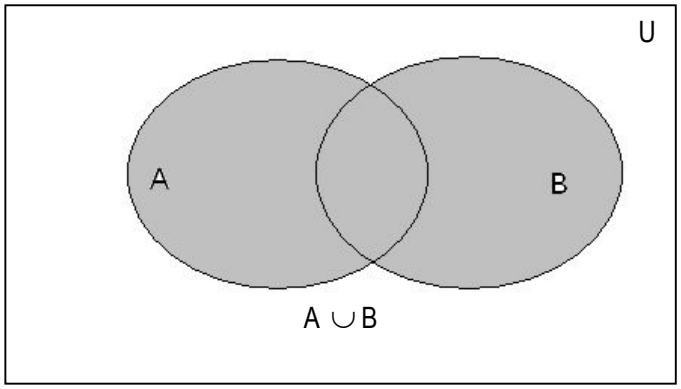

Рис. 11.3. Об’єднання множин

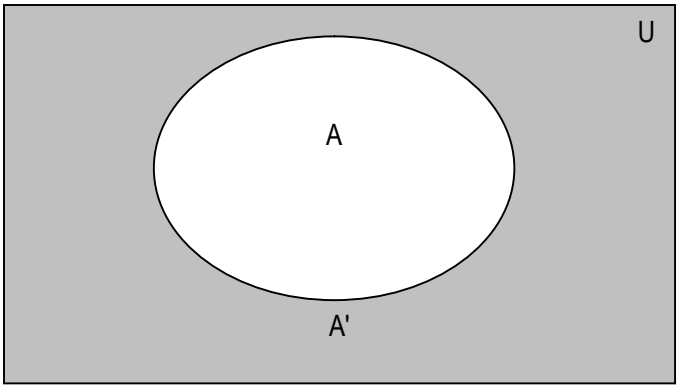

Рис. 11.4. Доповнення множини

Аристотелеві силогізми залишалися фундаментом логіки до 1847 року, коли англійський математик Джордж Буль (George Boole) опублікував першу книгу з описом символічної логіки. Одним з нових понять, запропонованих Булем, з’явилася модифікація представлення Аристотеля, що має назву “екзистенціальне значення”, згідно з яким суб’єкт міркувань повинен існувати. Наприклад, відповідно до класичних Аристотелевих поглядів такий вислів, як “Дід Мороз приносить подарунки 31 грудня”, не може використовуватися як посилання або наслідок, оскільки Діда Мороза насправді не існує.

Булеві представлення, що мають наразі статус сучасних представлень, дозволяють міркувати про порожні класи об’єктів. Тобто дозволяють створювати логічні конструкції для прогнозування наслідків використання результатів послідовності дій, які планується отримати вперше.

Ще один внесок у розвиток символічної логіки, зроблений Булем, полягав у тому, що вчений дав визначення поняття формулювання аксіом. Формулювання аксіом, згідно з Булем, складається із символів,

які використовуються для представлення об'єктів і класів, а також операцій алгебри – для маніпулювання цими символами.

11.2. Застосування елементів пропозиціональної логіки в представленні знань

Для маніпулювання висловами використовується символічна логіка, що має назву "пропозиціональна логіка". Зокрема, пропозиціональна логіка може використовуватися для маніпулювання логічними змінними, що позначають вислови. Модель представлення знань із застосуванням пропозиціональної логіки часто представляється в науковій літературі як обчислення висловів.

У пропозиціональній логіці розглядається певний тип речень природної мови. Усі речення природної мови поділяються на чотири основні типи: наказові, питальні, окличні, оповідні. Пропозиціональна логіка розглядає оповідні речення. Вони можуть розділятися на істинні та помилкові. Речення, істинне значення якого може бути визначене, називається твердженням або висновком. Твердження прийнято називати також закритим реченням, оскільки його значення не підлягає обговоренню, зважаючи на його істинність.

Речення, про істинність яких неможливо дати однозначну відповідь, називаються відкритими реченнями. Наприклад, відкритим є речення "Якість позичальника визначається наявністю постійного місця роботи", оскільки воно є істинним для одних людей і помилковим для інших. Неоднозначністю такого роду неможливо ефективно оперувати за допомогою пропозиціональної логіки. Подібні проблеми розв'язуються з використанням нечіткої логіки.

Обчислення висловів здійснюється на основі логічних зв'язок до окремих висловів, у результаті застосування яких формуються складні вислови. Для цього застосовуються логічні зв'язки, найбільш поширені з яких наступні:

- AND (Λ) – кон'юнкція;
- OR (V) – диз'юнкція;
- якщо... то... (→) – умовна операція;
- якщо і тільки якщо (↔) – двостороння умовна операція;
- NOT (~) – заперечення.

Кон'юнкція здійснюється завдяки застосуванню логічного "І" між двома висловами. Її результат – складний вислів буде істинним, якщо істинні обидва вислови. Диз'юнкція здійснюється завдяки застосуванню логічного "АБО" між двома висловами. Її результат буде істинним, якщо істинний хоча б один із цих висловів.

Умовна операція аналогічна формі IF-THEN продукційних правил. Наприклад, наступний вираз:

IF клієнт має постійний дохід THEN видати кредит

може бути представлений у такій формі:

$p \rightarrow q,$

де застосовуються наступні позначення: p – клієнт має постійний дохід, q – видати кредит.

Двостороння умовна операція $p \leftrightarrow q$ еквівалентна виразу:

$(p \rightarrow q)\ \Lambda\ (q \rightarrow p)$

Вона є істинною тільки тоді, коли p і q мають одночасно істинне або помилкове значення.

Усі операції пропозіціональної логіки, що були розглянуті, належать до бінарних. На відміну від них, операція заперечення є унарною, оскільки застосовується до одного виразу. Вона має вищий пріоритет у порівнянні з іншими операціями.

Сутність логічних зв'язок добре демонструє їх застосування для формалізованого опису таких логічних понять, як тавтологія та суперечність. Тавтологія – це складний вислів, який завжди є істинним, незалежно від того, істинні чи помилкові окремі вислови, з яких він складається. Прикладом тавтології може бути вислів: "Начальник завжди має рацію, оскільки він ніколи не може не мати рації".

На відміну від фактів, які можуть бути в реальному світі істинними або ні, тавтологія завжди, в чисто логічному значенні, істинна, оскільки посилається для доказу на саму себе. А суперечність являє собою складний вислів, який завжди є помилковим. Вислови, які не є ні тавтологією, ні суперечністю, у пропозиціональній логіці називаються контингентними.

Тавтології та суперечності називаються, відповідно, аналітично істинними і аналітично помилковими висловами, оскільки істинність їхніх значень може бути визначена виключно на підставі аналізу форми. Зокрема, істинне значення для виразу:

$p\ V \sim p$

показує, що це – тавтологія, а для виразу:

$p\ \Lambda \sim p,$

що це – суперечність.

Множина логічних зв'язок називається адекватною, якщо за допомогою зв'язок, узятих виключно з цієї множини, можна представи-

ти будь-яку функцію істини. До прикладів адекватних множин можна віднести множини: {~, Λ, V}, {~, Λ }, {~, V} .

Незважаючи на те, що пропозиціональна логіка є дуже корисною, в представленні знань вона має певні обмеження. Основна проблема полягає у тому, що пропозиціональна логіка може застосовуватися тільки з повним висловом. Це означає, що з її допомогою не можна досліджувати внутрішню структуру вислову. Наприклад, пропозіциональная логіка не дозволяє довести обґрунтованість наведеного раніше силогізму стосовно смертності викладача.

11.3. Логіка предикатів в експертних системах

З метою забезпечення вирішення проблем пропозиціональної логіки була розроблена логіка предикатів. Причому пропозиціональна логіка розглядається сьогодні як підмножина логіки предикатів. Предикат (лат. predicatum – заявлене, сказане) – це термін логіки та мовознавства, що означає конститутивний член вислову. Тобто предикат являє собою дещо, що стверджується або заперечується про суб'єкт.

Логіка предикатів дозволяє розглядати внутрішню структуру висловів. У ній допускається використання таких спеціальних слів, як: "все", "деякі", "жоден". Ці слова називаються кванторами. Квантори дозволяють надавати явні кількісні оцінки іншим словам і точніше формулювати вислови. Всі квантори відповідають на питання "скільки", і тому дозволяють застосовувати ширший круг виразів порівняно з пропозиціональною логікою. Існують різні види кванторів, і для детального розгляду їх сутності візьмемо найпоширеніші з них – квантор загальності та квантор існування.

Квантор загальності. Квантор загальності забезпечує організацію універсальних висловів. Вислів з квантором загальності приймає однаково істинне значення при підстановці всіх об'єктів з однієї області визначення.

Квантор загальності представляється за допомогою символу $\forall$, за яким йде один або декілька параметрів, що відповідають змінним області визначення. Символ $\forall$ інтерпретується як "для кожного" або "для всіх". Наприклад, вираз:

$(\forall x)(x + x = 2x)$

свідчить про те, що для кожного x (де x — число) вираз $x + x = 2x$ є істинним. Якщо цей вираз буде позначений символом p, то наведене вище твердження може бути представлене більш стисло наступним чином:

$(\forall x)(p)$.

Слід зазначити, що разом з фіктивними змінними x і p можна використовувати інші змінні, вислови та функції. Припустимо, що H – предикативна функція, що позначає людей, а M – функція, що позначає смертних істот. У такому разі твердження, згідно з яким всі люди смертні, можна записати наступним чином:

$(\forall x)(H(x) \rightarrow M(x))$.

Це твердження читається так: для всіх x, якщо x – людина, то x – смертна. Ця пропозиція логіки предикатів може бути також виражена в термінах продукційних правил:

IF x – людина THEN x – смертна

Квантор загальності може інтерпретуватися як кон'юнкція предикатів, що належать до окремих екземплярів. Під екземпляром тут розуміється конкретний випадок. Наприклад, припустимо, що людина з прізвищем Петренко Р.М. є конкретним екземпляром класу клієнтів банку. Тоді цю думку можна виразити через предикативну функцію "Клієнт", для якої Петренко Р.М. буде аргументом:

Клієнт(Петренко Р.М.).

Використання кон'юнкції дозволяє вислів логіки предикатів, представлений у вигляді:

$(\forall x)P(x)$

інтерпретувати в термінах екземплярів a_i:

$P(a_1) \wedge P(a_2) \wedge P(a_3) \wedge \ldots \wedge P(a_N)$

У цьому випадку послідовність крапок (…) вказує, що дія предиката розповсюджується на всі елементи даного класу. Таким чином, у наведеному виразі сказано, що предикат застосовується до всіх екземплярів класу.

У виразах може бути декілька кванторів. Наприклад, для формулювання закону комутативності суми чисел потрібні два квантори:

$(\forall x)(\forall y)(x + y = y + x)$

У цьому виразі стверджується, що "для кожного x і для кожного y сума x та y дорівнює сумі y та x".

Квантор існування. Квантор існування визначає твердження як істинне стосовно, принаймні, одного елемента області визначення. Він є обмеженою формою квантора загальності. Квантор існування записується як символ $\exists$, за яким йде одна або декілька змінних, наприклад:

(∃x)(x • x = 1),

(∃x)(Клієнт (x) Л Прізвище(Петренко Р. М.))

У першому з виразів вказано, що існує число *x*, результат множення якого на самого себе дорівнює одиниці. У другому виразі сказано, що існує клієнт на прізвище Петренко Р. М.

Квантор існування можна прочитати декількома способами, зокрема: "існує", "деякий", "щонайменше один". Так само, як квантор загальності може бути виражений за допомогою кон'юнкції, квантор існування може бути виражений за допомогою диз'юнкції екземплярів, a_i:

$P(a_1)\ V\ P(a_2)\ V\ P(a_3)\ V\ \ldots\ V\ P(a_N)$.

Як приклад практичного застосування логіки предикатів в експертних системах розглянемо реалізацію зв'язування значення та результату в програмі CLIPS, що виконує спрощений аналіз заробітної плати співробітників.

```
CLIPS> (deftemplate person
                (slot name)
                (slot post)
                (slot salary))
CLIPS> (defrule flnd-salary
          (person (name ?name) (salary 1000$))
          =>
          (printout t ?name " has salary 1000$." crlf))
CLIPS> (deffacts people
          (person (name Jane) (salary 1000$) (post manager))
          (person (name Joe) (salary 500$) (post cashier))
          (person (name Jack) (salary 1000$) (post programmer))
          (person (name Jeff) (salary 1000$) (post guard)))
CLIPS> (reset)
CLIPS> (run)
Jack has salary 1000$.
Jane has salary 1000$.
CLIPS>
```

У наведеному прикладі для реалізації зв'язування використовуються змінні, що зберігають значення в лівій частині правила, які надалі застосовуються в правій частині правила. Змінні в мові CLIPS завжди записуються із застосуванням синтаксису, що передбачає використання знаку питання, за яким іде ім'я символічного поля. В даному випадку і Джейн, і Джек мають зарплату 1000$, тому правило *find-salary* активізується двічі, по одному разу для кожного з фактів,

що описують Джейн і Джека. Після запуску правила слот *name* перевіряється на наявність факту, що активізував правило, після чого це значення використовується в операторі *printout*.

Багато типів висловів можна представити на основі логіки предикатів із використанням кванторів загальності та існування. Проте вона має деякі обмеження для представлення знань в експертних системах. Наприклад, у логіці предикатів неможливо виразити наступний вислів: “Більшість клієнтів банку взяли кредит у доларах”. У ньому квантор “більшість” означає “більше половини”.

Квантор “більшість” не може бути виражений у термінах квантора загальності та квантора існування. Для реалізації квантора “більшість” у логіці повинні бути передбачені предикати, які забезпечують підрахунок кількості елементів (що можливо при використанні нечіткої логіки).

Ще одне обмеження логіки предикатів полягає у тому, що вона не дозволяє виражати залежності, які можуть бути істинними іноді, але не завжди. Вказана проблема також може бути розв’язана за допомогою нечіткої логіки. Проте впровадження в логічну систему засобів проведення обчислень спричиняє появу додаткових ускладнень.

11.4. Нечітка логіка в експертних системах

Міркування, що спирається виключно на точні факти та точні висновки, які виходять із цих фактів, називаються строгими міркуваннями. У випадках, коли для прийняття рішень потрібно використовувати невизначені факти, строгі міркування стають непридатними. Тому однією з найсильніших сторін будь-якої експертної системи вважається її здатність формувати міркування в умовах невизначеності так само успішно, як це роблять експерти-люди. Такі міркування мають характер нестрогих.

Невизначеність може розглядатися як недостатність адекватної інформації для прийняття рішення. Невизначеність стає проблемою, оскільки може перешкоджати створенню найкращого рішення і навіть стати причиною того, що буде знайдене неякісне рішення. Слід відмітити, що якісне рішення, знайдене в реальному часі, часто вважається більш прийнятним, ніж найкраще рішення, для обчислення якого потрібна велика кількість часу. Наприклад, затримка в наданні лікування з метою проведення додаткових аналізів може привести до того, що пацієнт помре, не дочекавшись допомоги.

Причиною невизначеності стає наявність в інформації різноманітних помилок. Спрощена класифікація цих помилок може бути представлена в їхньому розділенні на наступні типи:

- неоднозначність інформації, виникнення якої пов’язано з тим, що деяка інформація може інтерпретуватися різними способами;
- неповнота інформації, яка пов’язана з відсутністю деяких даних;
- неадекватність інформації, обумовлена застосуванням даних, що не відповідають реальній ситуації (можливими причинами є суб’єктивні помилки: брехня, дезінформація, несправність устаткування);
- погрішності вимірювання, що виникають через недотримання вимог правильності та точності критеріїв кількісного представлення даних;
- випадкові помилки, проявом яких є випадкові коливання даних щодо середнього їх значення (причиною можуть бути: ненадійність устаткування, броунівський рух, теплові ефекти тощо).

На сьогодні розроблена значна кількість теорій невизначеності, в яких робиться спроба усунення деяких або навіть усіх помилок та забезпечення надійного логічного висновку в умовах невизначеності. До найбільш уживаних на практиці належать теорії, засновані на класичному визначенні ймовірності та на апостеріорній імовірності.

Одним з найстаріших і найважливіших інструментальних засобів рішення задач штучного інтелекту є ймовірність. Імовірність – це кількісний спосіб урахування невизначеності. Класична ймовірність бере початок із теорії, яка була вперше запропонована Паскалем і Ферма в 1654 році. З того часу була проведена велика робота у сфері вивчення ймовірності, здійснені численні застосування ймовірності в науці, техніці, бізнесі, економіці та інших областях.

Класичну ймовірність називають також апріорною ймовірністю, оскільки її визначення належить до ідеальних систем. Термін “апріорна” позначає ймовірність, що визначається “до подій”, без урахування багатьох чинників, що існують у реальному світі. Поняття апріорної ймовірності розповсюджується на події, що відбуваються в ідеальних системах, несхильних до зношення або впливу інших систем. В ідеальній системі поява будь-якої з подій відбувається однаково, завдяки чому їх аналіз стає набагато простішим.

Фундаментальна формула класичної імовірності (P) визначена наступним чином:

$$P = \frac{W}{N}$$

У цій формулі W – кількість очікуваних подій, а N – загальна кількість подій з рівними ймовірностями, які є можливими результатами експерименту або випробування. Наприклад, імовірність випадан-

ня будь-якої грані шестигранної гральної кістки дорівнює 1/6, а витягання будь-якої карти з колоди, що містить 52 різні карти – 1/52.

Формальна теорія ймовірності може бути створена на основі трьох аксіом:

1. Аксіома 1. У цій аксіомі стверджується, що областю визначення ймовірності події (E) є дійсні числа від 0 до 1. Від'ємні значення ймовірності не допускаються. Достовірній події привласнюється ймовірність 1, а неможливій події – імовірність 0:

$$0 \le P(E) \le 1$$

2. Аксіома 2. У даній аксіомі стверджується, що сума ймовірності всіх подій, незалежних одна від одної, що називаються взаємовиключними подіями, дорівнює 1:

$$\sum_i P(E_i) = 1$$

3. Аксіома 3. У цій аксіомі вказано, що якщо події E_1 і E_2 не можуть виникати одночасно (тобто є взаємовиключними подіями), то ймовірність виникнення тієї або іншої події дорівнює сумі ймовірностей цих подій:

$$P(E_1 \cup E_2) = P(E_1) + P(E_2)$$

Наведені аксіоми дозволили закласти фундамент теорії ймовірності, проте в них не розглядається імовірність подій, що відбуваються в реальних – неідеальних системах. На відміну від апріорного підходу, в реальних системах для визначення ймовірності деякої події $P(E)$ застосовується спосіб визначення експериментальної ймовірності як ліміту розподілу частот:

$$P(E) = \lim_{N \to \infty} \frac{f(E)}{N}$$

У цій формулі $f(E)$ позначає частоту появи деякої події поміж N-ї кількості спостережених загальних результатів. Імовірність такого типу називається також апостеріорною ймовірністю, тобто ймовірністю, що визначається "після подій". В основу визначення апостеріорної ймовірності покладене вимірювання частоти, з якою виникає деяка подія під час проведення великої кількості випробувань. Наприклад, визначення соціального типу кредитоспроможного клієнта банку на основі емпіричного досвіду.

Події, що не належать до взаємовиключних, можуть впливати одна на одну. Такі події входять до класу складних. Імовірність складних подій може бути обчислена шляхом аналізу відповідних їм вибі-

ркових просторів. Ці вибіркові простори можуть бути представлені за допомогою діаграм Венна, як показано на рис. 11.5.

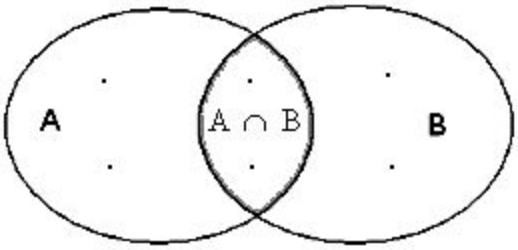

Рис. 11.5. Вибірковий простір для двох невзаємовиключних подій

Імовірність виникнення події А, яка визначається з урахуванням того, що відбулася подія В, називається умовною ймовірністю і позначається: *P(A |B)*. Умовна ймовірність визначається наступним чином:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

У цій формулі ймовірність *P(B)* не повинна дорівнювати нулю, і являє собою апріорну ймовірність, що визначається до того, як стане відома інша додаткова інформація. Апріорну ймовірність, що застосовується у зв'язку з використанням умовної ймовірності, іноді називають абсолютною ймовірністю.

Існує задача, яка є по суті протилежною задачі обчислення умовної ймовірності. Вона полягає у визначенні зворотної імовірності, яка показує ймовірність попередньої події з урахуванням тих подій, що відбулись у подальшому. На практиці з імовірністю такого типу доводиться зустрічатися досить часто, наприклад, при проведенні медичної діагностики або діагностики устаткування, при якій виявляються певні симптоми, а задача полягає в тому, щоб знайти ймовірну причину.

Для вирішення цієї задачі застосовується теорема Байєса, названа на честь британського математика XVIII століття Томаса Байєса. Байєсівська теорія в наші дні широко використовується для аналізу дерев рішень в економіці та суспільних науках. Метод байєсівського пошуку рішень застосовується також в експертній системі PROSPECTOR при визначенні перспективних майданчиків для розвідки корисних копалин. Система PROSPECTOR набула широкої популярності як перша експертна система, за допомогою якої було відкрито цінне родовище молібдену, що має вартість 100 мільйонів доларів.

Загальна форма теореми Байєса може бути записана в термінах подій (*E*) та гіпотез (*H*) у наступному вигляді:

$$P(H \mid E) = \frac{P(E \mid H)P(H)}{P(E)}$$

При визначенні ймовірності події застосовується ще один тип імовірності, що має назву суб'єктивної ймовірності. Поняття суб'єктивної ймовірності поширюються на події, які не є відтворними і не мають історичної основи, за допомогою якої можна було б здійснювати екстраполяцію. Таку ситуацію можна порівняти з бурінням нафтової свердловини на новому майданчику. Проте оцінка суб'єктивної імовірності, зроблена експертом, краща, порівняно з повною відсутністю оцінки.

Суб'єктивна ймовірність, фактично, являє собою переконання або думку, виражені у вигляді ймовірності, а не об'єктивне значення імовірності, засноване на аксіомах та емпіричних вимірюваннях. Переконання та думки експертів виконують важливу роль в експертних системах.

ПИТАННЯ ТА ЗАВДАННЯ

1. Яка властивість формальної логіки є особливо корисною в представленні знань експертних систем?
2. Які дії вважаються найбільш важливою частиною процесу проведення міркувань?
3. Наведіть приклад силогізму.
4. Сформулюйте зв'язок між Аристотелевою логікою та теорією множин.
5. У чому полягає основна відмінність Булевої логіки від Аристотелевої?
6. Яке основне призначення пропозиціональної логіки?
7. Наведіть приклад відкритого речення.
8. Назвіть найбільш поширені логічні зв'язки, які застосовуються для обчислення висловів.
9. Дайте визначення тавтології, суперечності та контингентних виразів.
10. Яке основне обмеження має пропозиціональна логіка в представленні знань?
11. Назвіть призначення логіки предикатів.
12. Що таке квантори?
13. Дайте визначення квантора загальності.
14. Наведіть приклад квантора загальності.
15. Дайте визначення квантора існування.
16. Наведіть приклад квантора існування.

17. Які міркування називаються нестрогими?
18. Назвіть найбільш відомі типи помилок, що стають причиною невизначеності.
19. Напишіть фундаментальну формулу класичної ймовірності.
20. Які аксіоми утворюють фундамент класичної теорії ймовірності?
21. У чому полягає відмінність між апріорною та апостеріорною ймовірностями?
22. Напишіть загальну форму теореми Байєса.

ТЕСТИ

1. Яка властивість формальної логіки вважається найбільш корисною при створенні експертних систем?
   а) формальна логіка розглядає виключно форму логічних тверджень;
   б) формальна логіка вивчає форму і значення логічних тверджень, заснованих на фактах;
   в) формальна логіка використовує строгий механізм представлення семантики тверджень.
2. Що розуміють під Аристотелевим представленням, яке має назву "екзистенціальне значення"?
   а) суб'єкт логічних міркувань повинен існувати;
   б) у силогізмі посилання слугують свідоцтвами, з яких повинен обов'язково виходити висновок;
   в) для графічного способу представлення силогізмів можна використовувати діаграми Венна;
   г) механізм теорії множин може бути застосований для формального представлення знань у логічній моделі.
3. Який тип речень природної мови розглядається в пропозиціональній логіці?
   а) усі типи речень;
   б) окличні речення;
   в) наказові речення;
   г) оповідні речення;
   д) питальні речення.
4. Яка умовна операція пропозиціональної логіки еквівалентна виразу $(p \rightarrow q) \Lambda (q \rightarrow p)$?
   а) pVq;
   б) $p \leftrightarrow q$;
   в) $\sim pV \sim q$;
   г) $\sim(p \rightarrow q)$.

5. Яку назву має складний вислів, який для доказу посилається на самого себе?
   а) тавтологія;
   б) суперечність;
   в) контингентний вислів.
6. З якою метою в логіці предикатів застосовуються квантори?
   а) для виведення одного висновку з двох посилань;
   б) для надання кількісної оцінки словам у реченні;
   в) для обчислення висловів на основі логічних зв'язків.
7. Який із запропонованих варіантів прочитання не підходить до квантора: $\exists X$?
   а) існує;
   б) для кожного;
   в) щонайменше один.
8. Причиною якого типу помилок, що породжують невизначеність, стає використання інформації, яка не відображає реальну ситуацію?
   а) випадкові помилки;
   б) неповнота інформації;
   в) неадекватність інформації;
   г) неоднозначність інформації.
9. Які події розглядаються в аксіомі теорії ймовірності, яка представлена формулою: $\sum_i P(E_i) = 1$?
   а) залежні;
   б) рівноімовірні;
   в) взаємовиключні.
10. Що являє собою ймовірність P(B) у формулі розрахунку умовної ймовірності: $P(A \mid B) = \frac{P(A \cap B)}{P(B)}$?
   а) апріорну ймовірність;
   б) суб'єктивну ймовірність;
   в) апостеріорну ймовірність.

РОЗДІЛ 12. ПОНЯТТЯ СЕМАНТИЧНОЇ МЕРЕЖІ ТА ЇЇ ВИКОРИСТАННЯ В ЕКСПЕРТНИХ СИСТЕМАХ

12.1. Сутність та призначення семантичних мереж

Як було зазначено в попередньому розділі, пропозиціональна логіка використовується для маніпулювання висловами – реченнями людської мови, істинне або помилкове значення яких може бути визначене. Наприклад: “усі собаки – ссавці” або “трикутник має три сторони”. Для більш наочного – візуалізованого способу представлення пропозіциональної інформації в процесі проектування експертних систем використовуються семантичні мережі.

Вислови мають форму декларативних знань, оскільки в них стверджуються факти. З погляду математики, семантична мережа є поміченим орієнтованим графом. При цьому вислів завжди вважається атомарним, оскільки його істинне значення не підлягає подальшій декомпозиції.

Семантичні мережі вперше були розроблені для досліджень у сфері штучного інтелекту як спосіб опису людської пам’яті Квілліаном (Quillian) у 1968 році. Квілліан використовував семантичні мережі для аналізу значення слів у реченнях. У подальші часи семантичні мережі успішно застосовувалися для вирішення багатьох задач, пов’язаних із представленням знань. Їх цінність полягає в можливості враховувати в базі знань, окрім форм тверджень, їх семантику. Такі знання дозволяють експертним системам дійти прийнятного логічного висновку при роботі з неоднозначними фактами.

Структура семантичної мережі відображається графічно за допомогою вузлів та дуг, що їх з’єднують. Вузли називаються об’єктами, а дуги – зв’язками або ребрами. Зв’язки в семантичній мережі застосовуються для представлення відносин, а вузли, як правило, – для представлення фізичних об’єктів, концепцій або ситуацій.

Як приклад можна навести семантичну мережу (рис. 12.1), зв’язки якої визначають відносини між членами родини.

Для семантичних мереж відносини мають особливо важливе значення, оскільки утворюють базову структуру для організації знань. Знання, задані без урахування відносин, перетворюються просто на колекцію непов’язаних фактів. Лише при визначенні відносин знання набувають вигляду зв’язаної структури, дослідження якої дозволяє логічним шляхом створювати інші знання. На підставі наведеного прикладу можна зробити висновок, що Ганна та Борис – бабуся й ді-

дусь Івана, незважаючи на те, що на рисунку не присутній явний зв'язок, позначений як "онук".

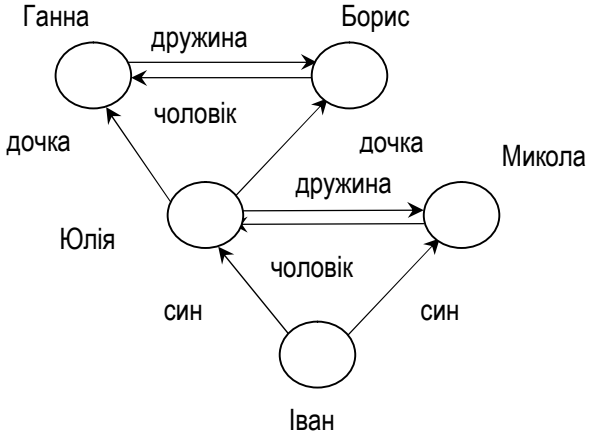

Рис. 12.1. Семантична мережа родинних зв'язків

Семантичні мережі іноді називають асоціативними мережами, оскільки вузли таких мереж зв'язані, тобто асоційовані між собою. В наукових дослідженнях Квілліана людська пам'ять від початку моделювалася як асоціативна мережа, в якій поняття були представлені у вигляді вузлів, а зв'язки показували, як ці поняття сполучаються один з одним.

Згідно з вказаною моделлю, якщо відбувається стимуляція одного вузла, як в результаті читання слів у реченні, то йде активізація зв'язків цього вузла з іншими вузлами. Надалі ця активність розповсюджується по мережі. Як тільки вузол отримує достатню активізацію, у свідомому розумі виникає концепція, представлена цим вузлом. Наприклад, відомо, що людина знає тисячі слів, але в процесі читання речення в її свідомості відбиваються лише ті слова, які вона читає.

Як показала практика, в багатьох способах представлення знань особливо корисним є застосування відносин однакових типів. Тому при побудові семантичних мереж для представлення знань у різних предметних областях замість того, щоб кожного разу визначати нові відносини, прийнято використовувати саме стандартизовані типи.

До найбільш широко використовуваних зв'язків у семантичних мережах належить зв'язок типу IS-A. Він означає, що окремий об'єкт "є екземпляром" певного класу. Прикладом такого типу зв'язків може бути віднесення клієнтів банківських установ до певного класу, що зображені на рис. 12.2. Зв'язок IS-A використовується для позначення відносин між окремими об'єктами через приналежність їх до загального класу завдяки тотожності атрибутів.

Іншим, широко застовуваним типом зв'язку, є тип A-KIND-OF (записується як АКО). На відміну від зв'язку IS-A, який визначає від-

ношення між окремими об'єктами та родовими класами, зв'язок АКО визначає відношення поміж самими родовими класами (рис. 12.3). Слід зазначити, що загальний клас, на який указує стрілка АКО, називається суперкласом. У випадку, якщо суперклас має зв'язок АКО, що вказує на інший вузол, то він, разом із тим, є класом суперкласу.

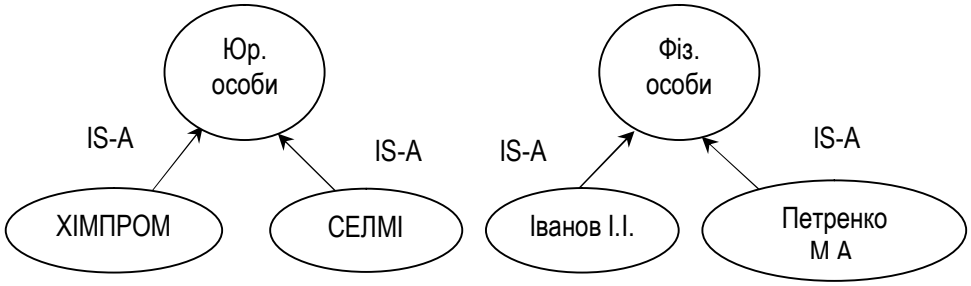

Рис. 12.2. Зв'язок типу IS-A в семантичній мережі

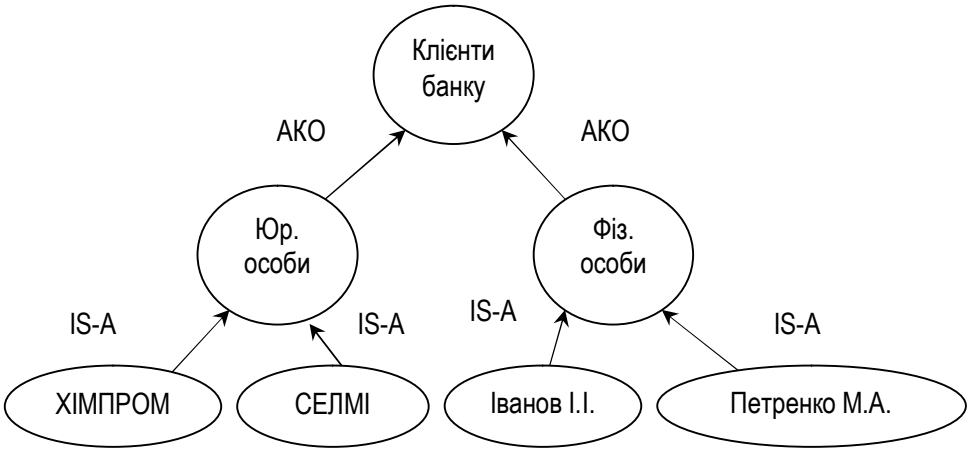

Рис. 12.3. Зв'язки типу АКО та IS-A в семантичній мережі

Повторення характеристик вузла в його нащадках називається спадкоємством. Якщо немає доказів, що дозволяють стверджувати зворотне, то важається, що всі елементи деякого класу успадковують усі властивості суперкласів цього класу. Зв'язки та спадкоємство є основою ефективних способів представлення знань, оскільки дають можливість представляти значну кількість складних відносин за допомогою декількох вузлів і зв'язків.

У семантичних мережах використовуються також зв'язки інших типів. Зокрема, до них належить зв'язок CAUSE, який виражає причинні знання. Наприклад, гаряче повітря CAUSE (стає причиною) того, що повітряна куля підіймається вгору.

Ще одним важливим зв'язком семантичних мереж є зв'язок HAS-A, що встановлює відношення між класом і підкласом. Спрямованість зв'язку HAS-A протилежна по відношенню до зв'язку АКО. Цей тип

часто використовується для позначення відношення між одним об'єктом та його складовою частиною, наприклад:

банк HAS-A касу
банк HAS-A бухгалтерію
банк IS-A Райффайзен

Тобто можна сказати, що якщо зв'язок IS-A встановлює відношення між значенням та атрибутом, то зв'язок HAS-A – між об'єктом та атрибутом.

Усі об'єкти одного класу повинні мати один або декілька загальних атрибутів. Комбінація атрибуту та значення називається властивістю. Такі три поняття, як об'єкт, атрибут та значення, зустрічаються разом настільки часто, що з'являється можливість створити спрощену семантичну мережу з використанням тільки цих понять. Для того, щоб охарактеризувати всі знання, представлені в семантичній мережі, можна скористатися триплетом "об'єкт-атрибут-значення" (object-attribute-value — OAV).

Саме такі триплети були використані під час створення експертної системи MYCIN, що призначена для діагностики інфекційних захворювань. У ній на основі триплетів "об'єкт-атрибут-значення" була реалізована система узгодження фактів та антецедентів продукційних правил.

12.2. Використання семантичної мережі в експертних системах

Семантичні мережі можуть бути легко перетворені в програму за допомогою мови PROLOG. Програми мовою PROLOG складаються з фактів і правил, заданих у загальній формі цілей:

$p: p_1,p_2,\ldots,p_n$

У цьому операторі термін p є головою виразу, а терміни p_i виконують ролі складових частин загальної цілі – підцілей. Як правило, вираз, який представлений у формі такого оператора, стверджує, що ціль p, задана в голові виразу, виконується тоді і тільки тоді, коли виконані всі підцілі. Виняток з цього правила виникає тільки в тому випадку, коли використовується спеціальний вираз, що означає відмову.

У мові PROLOG вирази представляються у вигляді предикатів, а оператори для їх представлення, відповідно, ґрунтуються на предикативній логіці. Оператори мови PROLOG складаються з імені предиката, такого як IS-A і HAS-A, за яким вказуються параметри (кількістю від нуля і більше), поміщені в круглі дужки та розділені комами. Наприклад:

HAS-A (Клієнти, Юридичні особи). ; клас "Клієнти банку" містить підклас "Юридичні особи"

IS-A (Селмі, Юридичні особи). ; об'єкт "Селмі" належить до класу "Юридичні особи"

Кінець оператора визначається крапкою, коментарі позначаються крапками з комою та ігноруються системою PROLOG. Загальна структура системи PROLOG зображена на рис. 12.4.

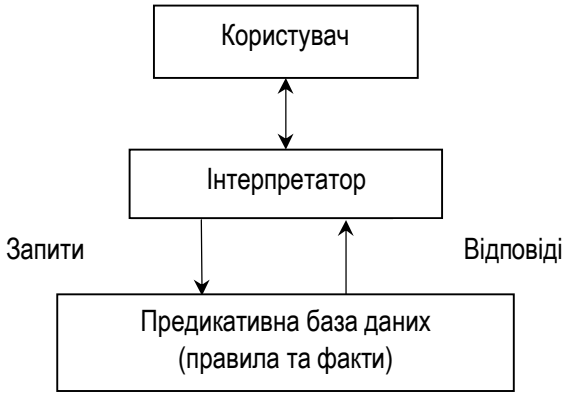

Рис. 12.4. Загальна структура організації експертної системи мовою PROLOG

Користувач через інтерпретатор передає на вхід системи PROLOG запит. Після того, як система PROLOG приймає вхідний запит, починається пошук виразу, голова якого узгоджується з вхідним шаблоном запиту. Така процедура називається зіставленням із шаблоном. Вона аналогічна зіставленню, в якому беруть участь факти та антецеденти продукційного правила.

Якщо голова предиката узгоджена, система PROLOG робить спробу погоджувати тіло правила, що складається з підцілей, із запитом. Пошук починається з першого введеного оператора, і може бути організований як у глибину, так і в ширину семантичної мережі, яку утворюють підцілі. Графічно цей процес зображений на рис. 12.5. та 12.6.

У випадку, якщо всі цілі, що підлягають узгодженню, досягаються, система визначає запит як істину. Якщо ні, система повертається до пошуку наступного виразу, голова якого узгоджується з шаблоном вхідного запиту. Тобто відбувається реалізація методу перебору з поверненнями для визначення істинності запиту.

З погляду розробників експертних систем, особливо важливими можливостями PROLOG є перебір з поверненнями та зіставлення із шаблонами. До того ж, важливим є те, що мова PROLOG має декларативний характер, оскільки в тілі виконавчої програми безпосередньо застосовується специфікація програмного коду.

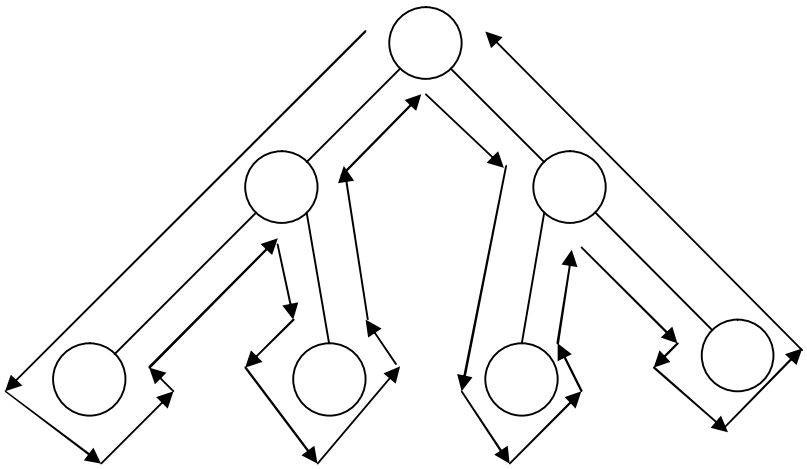

Рис. 12.5. Пошук у глибину семантичної мережі системи PROLOG

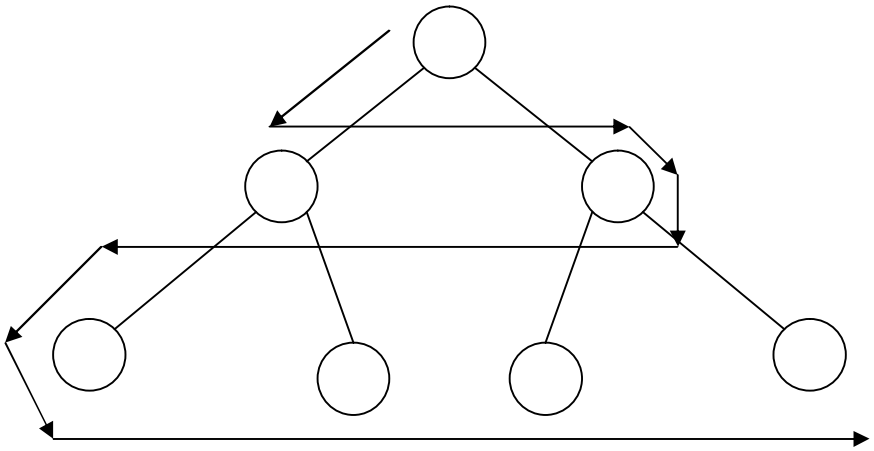

Рис. 12.6. Пошук у ширину семантичної мережі системи PROLOG

На основі викладеного матеріалу можна дійти висновку, що семантичні мережі можуть бути достатньо ефективним засобом представлення знань. Проте вони мають суттєві обмеження. Одне з них полягає у відсутності єдиних стандартів іменування вузлів, що породжує проблему відхилення від стандартів іменування зв'язків. Наприклад, якщо в мережі є вузол, позначений як "рахунок", то залежно від встановлених зв'язків він може позначати: банківський рахунок конкретного клієнта; поняття рахунку як категорії; рахунок, який обслуговується певним працівником банку, тощо. Тобто семантична мережа може застосовуватися для представлення категоричних знань (знань, визначених однозначно) лише за умови ретельної розробки системи іменування вузлів та зв'язків.

Ще одна проблема використання семантичних мереж полягає у тому, що при здійсненні пошуку вузлів виникає ймовірність комбінаторного вибуху. Вона зростає, якщо відповідь на запит, по суті, є негативною і це може змусити систему виконати пошук по всіх зв'язках

у мережі. Такий пошук потребуватиме значних обчислювальних ресурсів, особливо в повнозв'язній мережі, кількість зв'язків якої дорівнює факторіалу від кількості вузлів за мінусом одиниці.

На відміну від такої системи, пересічній людині не потрібно багато часу для відповіді на запитання: "Чи є представництво банку Райффайзен на Місяці?", незважаючи на те, що в мозку людини є приблизно 10^{11} нейронів та приблизно 10^{15} зв'язків. Якби всі його знання були сформовані виключно за допомогою семантичної мережі, то для формулювання негативних відповідей на подібні питання потрібно було б витрачати надзвичайно тривалі проміжки часу, оскільки довелося б застосовувати всю наявну кількість зв'язків.

Крім того, існує суттєве обмеження семантичних мереж для використання в експертних системах, пов'язане з тим фактом, що вони залишаються непридатними для евристичних методів. Нагадаємо, що евристика – це емпіричне правило, яке може допомогти знайти рішення за відсутності алгоритму. Оскільки єдиною вбудованою в семантичні мережі стратегією управління, яка може допомогти у вирішенні задач, є спадкоємство, досі не знайдено можливості представити евристичну інформацію за його допомогою.

Результати аналізу наведених обмежень показують, що семантичні мережі повинні використовуватися в тих областях, для яких вони пристосовані найбільше. Наприклад, для представлення бінарних відносин. Фахівці з експертних систем не рекомендують намагатися використовувати семантичні мережі як універсальний засіб представлення знань.

ПИТАННЯ ТА ЗАВДАННЯ

1. Що являють собою семантичні мережі по суті?
2. Наведіть приклад семантичної мережі.
3. Який тип зв'язку між елементами семантичної мережі позначається АКО?
4. Як позначається тип зв'язку семантичної мережі, який виражає причинні знання?
5. Направлення якого типу зв'язку протилежне по відношенню до зв'язку АКО?
6. Наведіть приклад операторів мовою PROLOG.
7. Намалюйте загальну структуру організації експертної системи мовою PROLOG.
8. Опишіть процес обробки запиту в системі PROLOG.
9. Які обмеження мають семантичні мережі в представленні знань?
10. У чому полягає якісна відмінність схеми від семантичної мережі?

ТЕСТИ

1. Що являє собою семантична мережа?
   а) візуальний спосіб представлення декларативних знань;
   б) помічений орієнтований граф, що використовується для декомпозиції висловів;
   в) графічне зображення набору вузлів, сполучених дугами, яке надає базову структуру для організації семантичних зв'язків між елементами знань різних предметних областей.
2. Яку головну перевагу дозволяє одержати представлення знань за допомогою семантичних мереж?
   а) дозволяє виводити логічним шляхом інші знання;
   б) дозволяє встановлювати відносини між атомарними висловами;
   в) дозволяє представляти знання у вигляді системи зв'язаних однозначних фактів.
3. Які три поняття семантичних мереж достатньо використати для їх спрощеної організації?
   а) клас-об'єкт-атрибут;
   б) об'єкт-зв'язок-вираз;
   в) вираз-зв'язок-значення;
   г) об'єкт-атрибут-значення.
4. Яке рішення приймає система PROLOG, якщо вхідний шаблон запиту узгоджується з головою виразу?
   а) система визначає, що запит є істинним;
   б) система поміщує вираз у предикативну базу даних;
   в) система робить спробу погодити тіло виразу із запитом.
5. У якому з перерахованих варіантів використання семантичних мереж виникає якнайменша ймовірність комбінаторного вибуху в процесі пошуку вузлів?
   а) відповідь на запит є негативною;
   б) відповідь на запит є позитивною;
   в) семантична мережа має повнозв'язну структуру.
6. Структура якого типу баз даних відповідає організації семантичних мереж?
   а) реляційні бази даних;
   б) ієрархічні бази даних;
   в) об'єктно-орієнтовані бази даних.

РОЗДІЛ 13. ТЕХНОЛОГІЯ ПРОЕКТУВАННЯ ЕКСПЕРТНИХ СИСТЕМ НА ОСНОВІ ФРЕЙМОВОЇ МОДЕЛІ ПОДАННЯ ЗНАНЬ

13.1. Застосування схем для представлення складних структур знань

Структура семантичної мережі, розглянута в попередньому розділі, надто проста, щоб з її допомогою ефективно представляти знання багатьох типів, що існують у реальному світі. У штучному інтелекті для опису більш складних структур знань, ніж семантичні мережі, використовується термін “схема”. Схеми, на відміну від семантичних мереж, мають структуру, внутрішню по відношенню до вузлів, що в них реалізовані. Тобто в них здійснюється розширення інформативності про кожен вузол. Нагадаємо, що всі знання про певний вузол в семантичній мережі зведені до напису на ньому – його імені.

Семантична мережа аналогічна інформаційно-технологічній структурі даних, в якій ключ пошуку одночасно є, по суті, елементом даних, що визначає вузол. Такий принцип організації даних застосовується в базах даних ієрархічного та мережевого типів. На відміну від них, схема аналогічна структурі даних, в якій вузли містять записи. Така організація, у свою чергу, є характерною для реляційних баз даних. Аналогічно реляційним структурам, у схемах кожен вузол може містити дані, записи або покажчики на інші вузли.

Термін “схема” запозичений із психології, де він визначає реакції живої істоти, які виробляються у відповідь на стимули. Це означає, що живі істоти, вивчаючи причинні відносини між стимулом та результатом, прагнутимуть повторно отримати приємні стимули та уникнути неприємних. Наприклад, при здійсненні такої дії, як їзда на велосипеді, складається певна сенсорно-рухова схема. Вона забезпечує координацію інформації, одержаної від органів, що сприймають відчуття, з необхідними мускульними рухами. В результаті людина не замислюється над знаннями стосовно виконання цих дій.

До найбільш важливих типів схем можна віднести концептуальні схеми, за допомогою яких у мозку людини складаються концепції щодо об’єктів та явищ. Тобто всі люди у своїй свідомості мають певні стереотипи, що складаються з концепцій. Концептуальна схема являє собою абстракцію, в якій конкретні об’єкти класифікуються згідно з

їхніми загальними властивостями. Зосередження на загальних властивостях об'єкта дозволяє спростити проведення міркувань про всі його властивості, оскільки, при цьому не доводиться відволікатися на незначущі подробиці. Це підвищує ефективність застосування знань, сформованих таким чином. У штучному інтелекті термін "стереотип" позначає типовий приклад.

У багатьох практичних реалізаціях штучного інтелекту використовується різновид схем, що мають назву фрейм. Фрейми розглядаються в контексті процесу розуміння різних способів отримання інформації людиною, наприклад: зображень органами зору, сприйняття мовних конструкцій і т.д. Вони являють собою зручну структуру для опису об'єктів, типових для конкретної ситуації, тобто її стереотипів. Власне, саме фрейми, послідовність яких є впорядкованою в часі, утворюють сценарії – тип схем, що застосовуються для формалізації знань в експертних системах.

Фрейми є особливо корисним засобом моделювання знань, які вважаються заснованими на здоровому глузді. Під знаннями, заснованими на здоровому глузді, розуміють знання, сформовані на загальновідомих фактах. Здоровий глузд застосовується для знаходження рішень у випадках, коли знання, які точно відповідають конкретній ситуації, є недоступними.

На відміну від семантичних мереж, які по суті можна розглядати як двовимірне представлення знань, фрейми додають третє вимірювання, оскільки дозволяють використовувати вузли, що мають внутрішню структуру. В ролі таких структур можуть застосовуватися прості значення або інші фрейми.

Основна характерна особливість фрейму полягає у тому, що він представляє взаємопов'язані знання щодо певної вузької тематики, значна частина яких – це знання, задані за замовчуванням. Для представлення складної структури знань потрібно створювати системи фреймів.

Наприклад, система фреймів непогано підходить для опису такого об'єкта, як автомобіль. Хоча автомобілі різних моделей відрізняються один від одного, вони, як правило, мають аналогічні компоненти: двигун, корпус, ходова частина тощо. Додаткові відомості про компоненти можуть бути оримані за допомогою вивчення структури фреймів, що входять до складових частин фрейму автомобіля.

Системи фреймів проектуються так, щоб більш універсальні фрейми знаходилися ближче до вершини ієрархії. Передбачається, що спеціалізація фреймів стосовно конкретних випадків може здійснюватися

шляхом модифікації наявних значень за замовчуванням та створення більш конкретних фреймів, з додаванням до них нових елементів.

За допомогою фреймів може бути зроблена спроба моделювати об'єкти реального світу, використовуючи універсальні знання для опису більшості атрибутів певного об'єкта та більш конкретні знання – для опису окремих випадків. Об'єкт, що володіє всіма типовими характеристиками, прийнято називати прототипом. Цей термін буквально означає "первинний тип".

Для роботи з фреймами в процесі розвитку засобів програмування систем штучного інтелекту були створені мови спеціального призначення. Серед найбільш відомих можна виділити: FRL, SRL, KRL, KEE, HP-RL. Крім того, в мові LISP передбачені вдосконалені засоби роботи з фреймами.

13.2. Практичні аспекти використання фреймів в експертних системах

З точки зору інструментарію розробки інформаційних систем, фрейми можна порівняти з такою складною структурою мов програмування високого рівня (наприклад, C#), як записи. Зокрема, полям та значенням записів відповідають такі компоненти фрейму, як слоти та заповнювачі слотів. Фрейм, по суті, являє собою групу слотів та заповнювачів, які визначають об'єкт.

Фрейми, зазвичай, застосовуються для представлення або універсальних, або спеціальних знань. Наприклад, універсальний фрейм, призначений для представлення концепції власності, якою може володіти людина, матиме вигляд, наведений в таблиці 13.1.

Таблиця 13.1

Слоти	Заповнювачі
ім'я	власність
спеціалізація	a_kind_of об'єкт
тип	(будинок, квартира, автомобіль) if-added: процедура ADD_PROPERTY
характеристика	(рухома, нерухома)
стан	(хороший, середній, поганий)
під заставою	(так, ні)

Заповнювачами можуть бути значення, такі як назва власності в слоті "ім'я", або низка значень, наприклад, як у слоті "тип". Заповню-

вачі можуть також містити процедури, які закріплені за слотами та називаються процедурними вкладеннями. Процедури, як правило, поділяються на три типи: if-needed, if-added, if-removal.

Процедура if-needed виконується у двох випадках:

- якщо потрібне значення заповнювача, яке від початку не було визначене;
- значення заповнювача, передбачене за замовчуванням (default), стало непридатним.

Значення фреймів, передбачені за замовчуванням, дозволяють на підставі досвіду моделювати очікування людини відносно деякої ситуації. При появі нової ситуації здійснюється модифікація найбільш відповідного фрейму, що дозволяє простіше до неї пристосуватися. Саме задані за замовчуванням значення використовуються для представлення знань, заснованих на здоровому глузді.

Процедура if-added викликається на виконання, якщо в слот має бути введене додаткове значення. Наприклад, у слоті "тип" процедура if-added викликає на виконання (у разі потреби) процедуру ADD-PROPERTY для додавання власності нового типу. Ця процедура може бути викликана після придбання коштовностей, яхти або власності іншого типу, оскільки вказані значення не містяться в слоті "тип".

Процедура if-removal викликається на виконання кожного разу, коли виникає необхідність видалити зі слоту якесь значення. Зокрема, процедура такого типу виконується, якщо дані застарівають.

Заповнювачі слотів можуть також містити відносини, призначені для створення ієрархічних зв'язків між фреймами. Найбільш використовувані відносини:

- a-kind-of – підклас-клас;
- is-a – екземпляр-клас.

Створюючи такі фрейми, розробники керуються угодою, що відносини a-kind-of є універсальними, а відносини is-a – конкретними. Наприклад, фрейм автомобіля, що представлений у таблиці 13.2, належить до універсальних субфреймів із описом типу власності. Тоді екземпляр фрейму з описом конкретного автомобіля виглядатиме як показано в таблиці 13.3.

Класифікація фреймів проводиться залежно від області застосування. Загалом виділяють три види фреймів:

1. Ситуативні фрейми – містять знання про те, чого можна очікувати в конкретній ситуації. Наприклад, оцінки за демонстрацію студентом того чи іншого рівня знань на іспиті.
2. Фрейми дії – визначають дії, що підлягають виконанню в конкретній ситуації. Це означає, що заповнювачами слотів цих фреймів є процедури, призначені для виконання певних дій. Наприклад, ви-

ведення студента з іспиту через спробу списування. Фрейм дії представляє процедурні знання.

3. Фрейми причинних знань – призначені для опису причинно-наслідкових відносин і є поєднанням ситуативних фреймів та фреймів дії.

Таблиця 13.2

Слоти	Заповнювачі
ім'я	автомобіль
спеціалізація	a_kind_of власність
тип	(седан, універсал, кабріолет)
характеристика	рухома
виробник	(GM, Ford, Toyota)
двигун	(бензиновий, дизельний)
трансмісія	(ручна, автоматична)
стан	(хороший, середній, поганий)
під заставою	(так, ні)

Таблиця 13.3

Слоти	Заповнювачі
ім'я	антилопа гну
спеціалізація	IS_A автомобіль
тип	кабріолет
виробник	GM
власник	Козлєвіч К.
двигун	бензиновий
трансмісія	ручна
стан	хороший
під заставою	ні

Від початку застосування фрейми призначалися для представлення стереотипних знань. Важливою особливістю будь-якого стереотипу є те, що він має цілком визначені характеристики, тому дозволяє надати багатьом своїм слотам значення, задані за замовчуванням. Тому застосування фреймів має інтуїтивну привабливість, оскільки з їх допомогою забезпечується впорядковане представлення знань, більш доступне для розуміння в порівнянні з логічними або продукційними системами, в яких з тією ж метою застосовується велика кількість правил.

Фрейми отримують свою значущість при побудові ієрархічних систем та в умовах застосування спадкоємства. Використання фреймів у вигляді заповнювачів слотів та введення в дію зв'язків спадкоємства дозволяє створювати дуже потужні системи представлення знань. Зокрема, експертні системи на основі фреймів є надзвичайно корисним засобом представлення причинних знань, оскільки інформація, що зберігається в них, організована з урахуванням причин та наслідків. На відміну від них, експертні системи, що засновані на правилах, зазвичай, спираються на неорганізовані знання, які не належать до причинних.

Деякі інструментальні засоби, що базуються на фреймах, такі як KEE, дозволяють зберігати в слотах найрізноманітніші елементи. Слоти фреймів можуть зберігати правила, графіку, коментарі, допоміжну інформацію, питання користувачів, гіпотези або інші фрейми.

На практиці були створені дуже складні системи фреймів, призначені для вирішення різних задач. Однією з найбільш вражаючих систем, що продемонструвала широкі можливості застосування фреймів для відкриття нових математичних понять, стала програма AM (Automated Mathematician – автоматизований математик) Дуга Лената (Doug Lenat). У класичній системі AM Лената, на основі загальновідомих, створювалися нові поняття, а потім досліджувалися їх поєднання. Ця система запропонувала деякі абсолютно нові математичні докази для значної кількості теорем.

Проте при використанні фреймових моделей подання знань виявляються певні недоліки. В основному, вони пов'язані з тим, що у фреймових системах допускається необмежена модифікація або знищення слотів. Тобто в більшості таких систем непередбачені способи визначення незмінних слотів. А оскільки у зв'язку з цим може піддатися змінам будь-який слот, то властивості, успадковані іншими фреймами, можуть бути змінені або видалені на будь-якому рівні ієрархії. Проблеми подібного типу виникають також при застосуванні семантичних мереж, в яких дозволено вносити зміни у властивості будь-якого вузла.

Однак слід зазначити, що існує інший спосіб застосування фреймів, який має велику практичну цінність. Якщо поняття фрейму буде розширене так, щоб воно охоплювало властивості об'єктів, то з'являється можливість розглядати будь-який об'єкт як фрейм. Тобто завдяки застосуванню об'єктів, заснованих на фреймах, стає простіше створювати, експлуатувати і супроводжувати великі бази знань, у порівнянні з такими системами, в яких робляться спроби представити всі знання в тисячах окремих правил та фактів.

Для реалізації цієї переваги в мову експертних систем CLIPS вбудована повна об'єктно-орієнтована мова, що має назву COOL. Після цього CLIPS став розглядатися як мова програмування, заснована на фреймах, і якій доступні всі переваги об'єктно-орієнтованого програмування. Сьогодні мовою CLIPS можна створювати об'єктно-орієнтовані експертні системи, маючи змогу використовувати правила, що розглядаються як невеликі фрагменти знань і, разом з тим, надають можливість організовувати більш масштабні фрагменти знань.

Для демонстрації різних методів застосування фреймів та шаблонів в мові CLIPS використаємо програму, що реалізує гру з двома гравцями під назвою Sticks. Мета гри Sticks полягає в тому, щоб уникнути необхідності узяти останню паличку з купи паличок. Кожен гравець повинен брати з купи по черзі 1, 2 або 3 палички. Весь секрет виграшу в цій грі полягає у тому, що можна змусити супротивника програти, якщо на Вашій черзі ходу залишаться 2, 3 або 4 палички. Таким чином, гравець, на ході якого залишилося 5 паличок, програв. Для того, щоб вимусити іншого гравця потрапити в ситуацію, в якій для нього залишається 5 паличок, слід завжди залишати після свого ходу 5 паличок плюс деяку кількість паличок, кратну 4.

Перш, ніж програма зможе приступити до ведення гри Sticks, вона повинна визначити деяку інформацію. Оскільки програма повинна грати проти супротивника-людини, необхідно в першу чергу з'ясувати, хто ходить першим. Крім того, слід визначити початковий розмір купи паличок. Ця інформація може бути поміщена в конструкцію deffacts. Наступний приклад показує, як застосовується функція read для введення даних:

```
(deffacts initial-phase (phase choose-player))
(defrule player-select
     (phase choose-player)
=>
     (printout t "Who moves first (Computer: з " "Human: h)? ")
     (assert (player-select (read))))
(defrule good-player-choice
     ?phase <- (phase choose-player)
     ?choice <- (player-select ?player&c | h)
=>
     (retract ?phase ?choice)
     (assert (player-move ?player))
     (assert (phase select-pile-size)))
```

В обох правилах використовується шаблон (phase choose-player) для вказівки на те, що ці правила застосовні тільки за наявно-

сті конкретного факту в списку фактів. Такий шаблон називається управляючим шаблоном, оскільки він спеціально призначений для управління тим, чи є правило застосовним чи ні. А для запуску управляючого шаблону використовується управляючий факт. В управляючому шаблоні для цих правил використовуються тільки літеральні поля, тому управляючий факт повинен точно узгоджуватися з шаблоном. У даному випадку управляючим фактом, вживаним для запуску цих правил, повинен бути факт (phase choose-player). Крім всього іншого, такий управляючий факт дозволяє виправити помилку, якщо вхідні дані, одержані за допомогою функції read, не співпадають з очікуваними значеннями, “c” або “h” (скорочення від “computer” – комп’ютер і “human” – людина). Повторну активізацію правила player-select можна забезпечити, вводячи в список фактів цей управляючий факт.

Нижче наведений висновок, який показує, як працюють правила player-select і good-player-choice при визначенні того, хто повинен ходити першим.

CLIPS> (unwatch all)
CLIPS> (watch facts)
CLIPS> (reset)
==> f-0 (initial-fact)
==> f-1 (phase choose-player)
CLIPS> (run)
Who moves first (Computer: c Human: h)? c
==> f-2 (player-select c)
<== f-1 (phase choose-player)
<== f-2 (player-select c)
==> f-3 (player-move c)
CLIPS>

Ці правила функціонують належним чином, якщо вводиться передбачена в них відповідь c або h, але якщо введена відповідь відрізняється від того, що вимагається, то перевірка на наявність помилок не здійснюється. Це – один з прикладів численних ситуацій, в яких необхідно повторити запит на введення, щоб виправити помилку при введенні. У наведеному нижче коді показаний зручний спосіб реалізації в програмі циклічного повтору запиту на введення. Правило, приведене в цьому коді, в поєднанні з правилами player-select і good-player-choice забезпечує перевірку на наявність помилок.

(defrule bad-player-choice
?phase <- (phase choose-player)

```
?choice <- (player-select ?player&~c&~h)
=>
(retract ?phase ?choice)
(assert (phase choose-player))
(printout t "Choose з or h." crlf))
```

В цьому випадку слід зазначити, що використовується управляючий шаблон (phase choose-player). Він забезпечує загальне управління циклом введення, а також запобігає запуску цієї групи правил при здійсненні інших етапів виконання програми.

Припустимо, що черговість ходу належить гравцю-людині. Якщо в купі залишилася тільки одна паличка, то людина програла. А якщо паличок більше за одну, то комп'ютер повинен запитати людину, скільки паличок слід прибрати з купи. У тому правилі, в якому формулюється питання до людини про те, скільки паличок слід прибрати з купи, необхідно передбачити перевірку, що кількість паличок в купі більше за одну. Після того, як людина вкаже кількість паличок, що підлягають видаленню, її відповідь необхідно перевірити, щоб переконатися у тому, що вона є допустимою. Нижче наведені правила, в яких умовні елементи test застосовуються для перевірки того, чи є допустимими дані про кількість паличок, узятих з купи гравцем-людиною. У цих правилах для зберігання інформації про кількість паличок, що залишилися в купі, використовується факт pile-size.

```
(defrule get-human-move
        (player-move h)
        (pile-size ?size)
; У людини-гравця є вибір, тільки якщо в купі
; залишилося більше однієї палички
        (test (> ?size 1))
=>
        (printout t "How many sticks do you wish to take?")
        (assert (human-takes (read))))
(defrule good-human-move
        ?pile <- (pile-size ?size)
        ?move <- (human-takes ?choice)
        ?whose-turn <- (player-move h)
        (test (and (integerp ?choice)
             (>= ?choice 1)
             (<= ?choice 3)
             (< ?choice ?size)))
=>
```

```
        (retract ?pile ?move ?whose-turn)
        (bind ?new-size (- ?size ?choice))
        (assert (pile-size ?new-size))
        (printout t ?new-size " stick(s) left in the pile." crlf)
        (assert (player-move з)))
(defrule bad-human-move
        (pile-size ?size)
        ?move <- (human-takes ?choice)
        ?whose-turn <- (player-move h)
        (test (or (not (integerp ?choice))
        (< ?choice 1)
        (> ?choice 3)
        (>= ?choice ?size)))
=>
        (printout t "Number of sticks must be between 1 and 3," crlf
        "and you must be forced to take the last stick." crlf)
        (retract ?move ?whose-turn)
        (assert (player-move h)))
```

Виконання даної програми завершується тільки після того, як буде введене допустиме значення кількості узятих з купи паличок. І в цьому випадку для повторної активізації правила, що дозволяє ще раз ввести відповідь після введення в попередній спробі неприпустимої відповіді, застосовується управляючий факт. Повторне введення управляючого факту (player-move h) для ще однієї активізації правила bad-human-move здійснюється за допомогою правила get-human-move. Після цього гравець-людина дістає можливість ще раз вказати кількість паличок, які повинні бути узяті з купи.

ПИТАННЯ ТА ЗАВДАННЯ

1. Що розуміють під концептуальною схемою?
2. Дайте визначення фрейму.
3. Для вирішення яких задач доцільно використовувати знання, засновані на здоровому глузді?
4. Опишіть загальну структуру фреймів.
5. Які типи процедур використовуються в складі фреймів?
6. Наведіть приклад застосування системи фреймів для представлення знань стосовно об'єктів або явищ.
7. Охарактеризуйте класифікацію фреймів залежно від області застосування.

8. У побудові яких систем знань фрейми виявляють свою найбільшу практичну значущість?
9. У чому полягають недоліки використання фреймових моделей подання знань?

ТЕСТИ

1. Структура якого типу баз даних відповідає організації семантичних мереж?
   а) реляційні бази даних;
   б) ієрархічні бази даних;
   в) об'єктно-орієнтовані бази даних.
2. Яка з перерахованих систем дозволяє моделювати найбільш складні структури знань?
   а) фрейм;
   б) сценарій;
   в) семантична мережа.
3. Якого правила слід дотримуватися при проектуванні системи фреймів?
   а) ближче до вершини ієрархії слід розміщувати фрейми, що містять загальну інформацію про об'єкти;
   б) ближче до вершини ієрархії слід розміщувати фрейми, що містять конкретну інформацію про об'єкти;
   в) ближче до вершини ієрархії слід розміщувати фрейми, утворені сукупністю слотів та заповнювачів, які визначають об'єкт.
4. Які з перерахованих заповнювачів фреймів застосовуються для представлення знань, заснованих на здоровому глузді?
   а) відносини is-a;
   б) значення default;
   в) відносини a-kind-of;
   г) процедури типу if-added;
   д) процедури типу if-needed.
5. Що є головним недоліком застосування систем фреймів при моделюванні знань?
   а) в цих системах допускається необмежена модифікація або знищення слотів фреймів;
   б) в цих системах не дозволяється вносити один фрейм у заповнювач слота іншого фрейма;
   в) в цих системах необхідно дотримуватися жорсткої ієрархічної залежності між фреймами.

РОЗДІЛ 14. ХАРАКТЕРИСТИКА ПРОГРАМНИХ ЗАСОБІВ СТВОРЕННЯ ЕКСПЕРТНИХ СИСТЕМ

14.1. Категорії сучасних програмних засобів розробки експертних систем

Як уже зазначалось у попередніх розділах, сучасні мови програмування використовуються в поєднанні з набором допоміжних програм, формуючи таким чином інструментальний засіб розробки програмних систем. Нагадаємо, що експертна система – це, по суті, різновид програмної системи, яка оперує із знаннями в певній предметній області з метою вироблення рекомендацій для вирішення задач.

Практично всі інструментальні засоби, що використовуються в процесі розробки експертних систем, застосовують методологію автоматизації проектування на основі прототипів. По відношенню до програмного забезпечення термін “прототип” означає працюючу модель програми, яка функціонально еквівалентна підмножині кінцевого продукту.

Ідея використання прототипів полягає в розробці на ранній стадії роботи проекту спрощеної версії кінцевої програми, яка була б спроможна послужити доказом продуктивності основних ідей, покладених в основу проекту. Тобто прототип має бути здатний вирішувати одну з характерних задач для заданої області застосування. На основі аналізу досвіду роботи з прототипом розробники можуть уточнити вимоги до основних функціональних характеристик експертної системи. Працездатність прототипу може послужити наочним доказом можливості рішення проблем за допомогою створюваної системи ще до того, як на її розробку будуть витрачені значні засоби.

Процес розробки експертної системи, як правило, складається з послідовності окремих етапів, упродовж яких нарощуються можливості системи, причому кожен з етапів поділяється на фази: проектування, реалізації, компонування та тестування. В результаті, після завершення чергового етапу утворюється система, здатна впоратися з більшими за складністю варіантами проблеми.

На відміну від експертних систем, при створенні більшості програмних продуктів інших видів використовується інша модель процесу: спочатку розробляється специфікація продукту, потім виконується планування, проектування компонентів, їх реалізація, компонування комплексу та тестування кінцевого варіанта. Той факт, що при розро-

бці експертних систем існує можливість спочатку побудувати та всебічно випробувати прототип, дозволяє уникнути безлічі переробок у процесі створення робочої версії системи.

Однак слід зазначити, що технологія послідовного нарощування функціональних можливостей містить у собі проблему інтеграції нових функцій системи з функціями, що були реалізовані в попередніх варіантах. Тому інструментальні засоби розробки експертних систем від початку створювалися на основі модульного представлення знань з урахуванням необхідності подолання виникаючих при цьому ускладнень.

За своїм призначенням та функціональними можливостями інструментальні засоби, що використовуються в програмуванні експертних систем, можна розділити на чотири досить великі категорії:

1. Оболонки експертних систем.
2. Мови програмування високого рівня.
3. Середовище програмування, що підтримує декілька парадигм.
4. Додаткові модулі.

Розглянемо детально кожну з цих категорій.

Оболонки експертних систем. Системи типу оболонки експертних систем створюються, як правило, на основі експертних систем, які достатньо добре зарекомендували себе на практиці. В процесі створення оболонки, з системи-прототипу видаляються компоненти, які є специфічними для області її безпосереднього застосування, а залишаються ті, що не мають вузької спеціалізації.

Прикладом може слугувати система EMYCIN, створена на основі системи MYCIN. У структурі EMYCIN був збережений інтерпретатор, усі базові структури знань та пов’язаний з ними механізм індексації. Оболонка була доповнена засобами підтримки бібліотеки типових випадків та висновків, зроблених по ним експертною системою. Внаслідок подальшого розвитку оболонки EMYCIN з’явилися системи S.1 і M.4, в яких механізм побудови ланцюжка зворотних міркувань, запозичений в EMYCIN, був об’єднаний із фреймоподібною структурою даних та додатковими засобами управління ходом міркувань.

У загальному сенсі оболонки експертних систем створюються з метою дозволити непрограмістам скористатися результатами роботи програмістів, що вирішували аналогічні проблеми. Клас цих інструментальних засобів орієнтований на достатньо вузький клас задач, хоча й ширший, ніж та програма, на основі якої була створена оболонка.

Більшість комерційних продуктів цього типу підходить виключно для тих проблем, в яких простір пошуку невеликий. Як правило, в них застосовується метод пошуку рішення з побудовою ланцюжка зворотного логічного висновку та обмежені можливості управління проце-

сом. Цей підхід погано підходить для вирішення проблем конструювання, тобто об'єднання окремих елементів в єдиний комплекс з урахуванням заданих обмежень.

Проте, незважаючи на обмеження, цей тип експертних систем прогресує. У порівнянні з першими розробками, сучасні оболонки більш гнучкі, принаймні в тому, що без особливих зусиль можуть бути інтегровані в більшість операційних середовищ та оздоблені достатньо розвиненими засобами користувальницького інтерфейсу. Наприклад, оболонка M.4 може функціонувати під управлінням будь-якої з операційних систем персональних комп'ютерів, підключатися до баз даних, включати фрагменти програмного коду мовами Visual BASIC та Visual C++. Оболонка підтримує індивідуальну настройку користувальницького інтерфейсу та можливість формування пояснень у відповідях на питання "чому?".

Мови програмування високого рівня. Мови програмування високого рівня можуть бути ефективним засобом швидкого створення прототипів експертних систем. Вони дозволяють забезпечити гнучкість процесу розробки, мінімізацію матеріальних витрат і термінів виконання проекту. Інструментальні засоби цієї категорії позбавляють розробника необхідності заглиблюватися в деталі реалізації системи, такі як способи ефективного розподілу пам'яті, низькорівневі процедури доступу до даних та маніпулювання ними. Як правило, середовище розробки таких мов дозволяє суміщати вставку, редагування та тестування фрагментів програмного коду.

Досвідчений програміст, застосовуючи для проектування експертних систем мови високого рівня, одержує значно більшу свободу дій, ніж при використанні оболонки. Особливо це стосується програмування процедур управління та обробки невизначеності. Ці процедури дуже корисні для створення експериментальних систем, в яких не видається можливим заздалегідь обрати оптимальний режим управління.

Інструментальні засоби експертних систем, зазвичай, використовують тип мов програмування високого рівня, відомий як мова опису продукційних правил. Одним із найвідоміших представників таких мов є OPS5. Для цієї мови характерний порівняно простий синтаксис та механізм активізації правил. У ньому використовуються різні версії rete-алгоритму для оптимізації процесів узгодження фактів із правилами. Нагадаємо, rete-алгоритм позбавляє машину логічного висновку необхідності погоджувати факти з кожним правилом.

Для мови OPS5 характерною ознакою є труднощі при реалізації деяких типів структур управління ходом виконання. Наприклад, до них можна віднести рекурсивні та ітераційні цикли, оскільки вони ви-

магають серйозного ускладнення опису процесу обробки правил. Розробники мов, подібних OPS, завжди вимушені шукати компроміс між наочністю засобів мови програмування та ефективністю виконання програмного коду.

На думку фахівців, найбільш раціональний шлях подолання недоліків програмування продукційних правил полягає в об'єднанні їх з іншими парадигмами програмування. Прикладом такого об'єднання може бути комбінування продукційних правил та фреймів, що дозволяє зіставляти умови, специфіковані в правилах, із вмістом слотів фреймів.

Іншим типом мов програмування високого рівня, що застосовується в інструментальних засобах експертних систем, є об'єктно-орієнтовані мови. В цьому контексті мовами об'єктно-орієнтованого програмування створюється програмне середовище для організації знань у термінах декларативного представлення об'єктів предметної області. Усі дії, пов'язані з процедурною стороною рішення проблем, розподіляються між цими об'єктами, які мають у своєму розпорядженні власні процедури та можуть спілкуватися один з одним за допомогою інтерфейсів передачі повідомлень.

До ще одного корисного аспекту об'єктно-орієнтованого програмування належить можливість інтеграції символічних обчислень в операційне середовище, яке базується на засобах графічного інтерфейсу. Оснащення експертної системи цими засобами дозволяє краще представити користувачу процеси, що відбуваються в системі.

Основна причина складності використання об'єктно-орієнтованого стилю в програмуванні експертних систем полягає в організації співвідношення програмних об'єктів з абстрактними поняттями та категоріями предметної області. Тобто в експертних системах об'єкти повинні представляти факти та цілі, набори правил або окремі гіпотези, а не моделі елементів реального світу, як у класичних задачах. Тому схеми відображення цих понять та категорій на програмні об'єкти, а також повідомлення, якими вони повинні обмінюватися, мають бути ретельно продумані.

Окрім розглянутих мов програмування високого рівня, в інструментальних засобах експертних систем також застосовуються мови логічного програмування. Типовою мовою логічної розробки експертних систем є PROLOG. Для цього PROLOG володіє достатньо корисними можливостями, а саме:

- вбудований у PROLOG режим управління приблизно відповідає стратегії зворотного логічного висновку;

- індексовану базу даних фраз мови PROLOG можна використовувати для представлення правил;
- рекурсивні структури даних (графи та дерева) можна організувати за допомогою фраз мови PROLOG;
- універсальний механізм зіставлення мови PROLOG дозволяє виконувати зіставлення даних та шаблонів, що включають змінні;
- мовні засоби PROLOG дозволять програмісту розробити власний механізм обробки невизначеності.

Проте практика застосування ідей логічного програмування в експертних системах не позбавлена недоліків. Зокрема, наявність синтаксичних та семантичних обмежень, що присутні в стандартних версіях PROLOG, не були подолані ані в системах MECHO та PLANNER, ані в інших системах, що базуються на аналогічній ідеології.

Середовище розробки експертних систем, що підтримує декілька парадигм. Засоби цієї категорії включають декілька програмних модулів, що дозволяє комбінувати в процесі розробки експертної системи різні стилі програмування, вибираючи відповідні поєднання різних методів. Причиною їх створення стали результати роботи експертних систем із різними схемами представлення знань та логічного висновку. Виявилося, що кожна з них має свої слабкі сторони.

Зокрема, продукційні правила дозволяють представити в програмі емпірично виявлені зв'язки між умовами та діями. Проте вони значно гірше підходять для представлення відносин між об'єктами предметної області, включаючи такі важливі відносини, як "множина-елемент" або "множина-підмножина". З іншого боку, структуровані об'єкти, наприклад фрейми, виявляються зручним засобом для зберігання та маніпулювання описами об'єктів предметної області. Але застосування таких знань вимагає включення в програму фрагментів програмного коду, які потім важко аналізувати.

У результаті аналізу наведених реалій логічним чином була сформована ідея об'єднання методик в єдине середовище, в якому переваги одних компенсують недоліки інших. Одним із перших багатофункціональних середовищ штучного інтелекту став відповідний продукт, що має назву LOOPS (Bobrow and Steflk, 1983). У ньому в рамках єдиної архітектури обміну повідомленнями були об'єднані чотири парадигми програмування:

1. Процедурно-орієнтоване програмування. Ця парадигма була представлена мовою LISP, в якій активним компонентом є процедури, а пасивним – дані. В рамках єдиного середовища процедури можуть бути також використані для обробки зовнішніх даних.

2. Програмування, орієнтоване на правила. Ця парадигма аналогічна попередній, але роль процедур виконують правила “умова-дія”. В середовищі LOOPS набори правил за своєю суттю є об’єктами, які можна рекурсивно вкладати один у другий. Таким чином, частина “дія” одного правила, у свою чергу, може активізувати підлеглий набір правил.
3. Об’єктно-орієнтоване програмування. Згідно з цією парадигмою структуровані об’єкти володіють властивостями як процедур, так і даних. Обробка вхідних повідомлень призводить до передачі даних або зміни їх значень. Усі маніпуляції даними виконуються під управлінням того компонента, який звернувся до об’єкта. При цьому зовнішні об’єкти не інформуються про те, яким чином зберігаються дані та як вони модифікуються усередині об’єкта.
4. Програмування, орієнтоване на дані. В ньому процеси доступу та оновлення даних запускають певні процедури. Зі змінними, в яких зберігаються значення даних, пов’язані певні процедури, подібно до того, як це робиться у слотах фреймів.

Основу системи складає об’єктно-орієнтована парадигма. В рамках її модулів можна комбінувати модулі середовища, що підтримують різні стилі програмування. Зазвичай, умови в продукційних правилах пов’язуються із значеннями слотів структурованих об’єктів, а правила модифікують значення цих слотів. Саме такий стиль об’єднання парадигм реалізований у мові CLIPS.

У системах KEE і LOOPS поведінка об’єктів описується в термінах множини продукційних правил. У середовищі Knowledge Craft до перерахованих вище парадигм додане логічне програмування в стилі мови PROLOG. Одна з наступних версій KEE, відома під назвою KAPPA-PC, надає в розпорядження програміста ще більш розширений набір стилів для комбінування правил, об’єктів та процедур.

Додаткові модулі розробки експертних систем. Засоби цієї категорії є автономними програмними модулями, які призначені для виконання специфічних задач у рамках вибраної архітектури експертної системи. Під “додатковими модулями” розуміються корисні програмні розробки, які можна виконувати разом із основним додатком. Як правило, такі програми реалізують деякі спеціальні функції, підключаючи їх, начебто, з поза меж системи. Причому звернення до таких функцій не потребує додаткового програмування в основному додатку.

Прикладом додаткового модуля може бути модуль роботи з семантичною мережею, використаний в системі VT. Цей модуль дозволяє в процесі роботи над проектом відстежувати зв’язки між значеннями раніше встановлених та нових параметрів проектування. Подібні модулі управління семантичною мережею можна застосовувати для

розповсюдження внесених змін на всі компоненти системи. Іншим прикладом додаткового модуля може слугувати програмний пакет Simkit із комплекту середовища KEE. Цей пакет дозволяє доповнити експертну систему методами моделювання.

Тенденція використання додаткових модулів найімовірніше розвиватиметься, оскільки користувачі експертних систем часто мають потребу в різних додаткових функціональних можливостях, специфічних для конкретного додатка. Більше того, часто виникає необхідність інтегрувати експертну систему з програмними продуктами інших класів. На практиці експертна система часто використовується разом з базою даних або системою управління, яка одержує інформацію від пакетів статистичної обробки або систем обробки сигналів.

14.2. Характерні складнощі розробки експертних систем та способи їх уникнення

Процес розробки та впровадження експертної системи має характерні складнощі. Їх систематизація дозволяє виробити практичні рекомендації, що дозволяють їх уникати. Один із найбільш об'єктивних способів систематизації цих складнощів та способів їх подолання запропонував у своїх працях вчений Уотерман (Waterman). Скористаємося його результатами.

Перша складність з'являється тоді, коли знання, що стосуються предметної області, дуже тісно переплетені з іншими частинами системи. Зокрема, може бути важко відокремити ці знання від знань загального застосування, що стосуються способів пошуку в просторі рішень. Уотерман припускає, що цього можна досягти, поклавши в основу організації бази знань набір правил. Проте у спеціалізованій літературі існують думки, що така організація не завжди гарантує досягнення бажаного результату.

Друга складність полягає у тому, що база знань, яка сформувалася після отримання та представлення множини правил у процесі опитування експертів, виявляється неповною настільки, що не дозволяє вирішувати потрібні задачі. Причиною цього є відсутність у ній фундаментальних концепцій предметної області або ці концепції представлені з помилками. Для її вирішення рекомендується послідовно нарощувати обсяг бази знань, починаючи із фундаментальних понять. Це дозволить ще на ранніх стадіях розробки виявити вказану проблему. Крім того, корисно виконувати тестування на кожному етапі розробки, використовуючи для цього відповідні інструментальні засоби інженерії знань.

Третя складність виникає, коли середовище розробки не має у своєму розпорядженні вбудованих засобів формування функцій пояснення експертної системи. Додавання таких функцій у вже спроектовану систему – задача не з легких. Для її запобігання слід піклуватися про прозорість експертної системи з перших кроків її розробки. Крім того, без засобів моніторингу навіть її творець не може бути до кінця впевнений, що вона працює належним чином.

Четверта складність полягає у тому, що система може містити надмірну кількість дуже специфічних правил. Це, по-перше, призводить до уповільнення роботи системи, а по-друге – ускладнює управління нею. Для її запобігання рекомендується об'єднувати, де тільки можливо, дрібні правила в більш загальні. Це виглядає як прагнення знайти компроміс між ефективністю правил та їх зрозумілістю.

Подальші питання, з необхідністю вирішення яких стикаються розробники експертних систем, полягають у виборі відповідного інструментарію, оцінці наочності вибраних засобів і визначенні "хорошого стилю програмування" у вибраному середовищі розробки. Подальший матеріал буде сконцентрований саме на цих питаннях.

14.3. Методика вибору оптимального інструментарію для розробки експертної системи

В основу рекомендацій щодо вибору інструментальних засобів для побудови експертної системи у відповідних наукових працях покладене зіставлення характеристик проблем, що має вирішувати експертна система, та необхідних функціональних можливостей інструментального комплексу. При цьому рекомендується дотримуватись наступних загальних правил:

- слід вибирати інструмент зі ступенем сервісної насиченості, який не перевищує необхідного рівня для вирішення даної задачі;
- вибір інструментарію повинен визначатися, в першу чергу, характеристиками задачі, яку вирішуватиме експертна система, а не сторонніми обставинами (наприклад тим, що якийсь інструмент вже є в наявності або знайомий краще за інших);
- якщо успіх проекту залежить від терміну розробки, то слід вибирати інструментальне середовище із вбудованими засобами формування пояснень та елементів користувальницького інтерфейсу, оскільки їх розробка найбільш трудомістка;
- необхідно якнайшвидше провести випробування нового інструментального середовища на реальних даних.

Процес вирішення питання вибору можна представити схематично як показано на рис. 14.1.

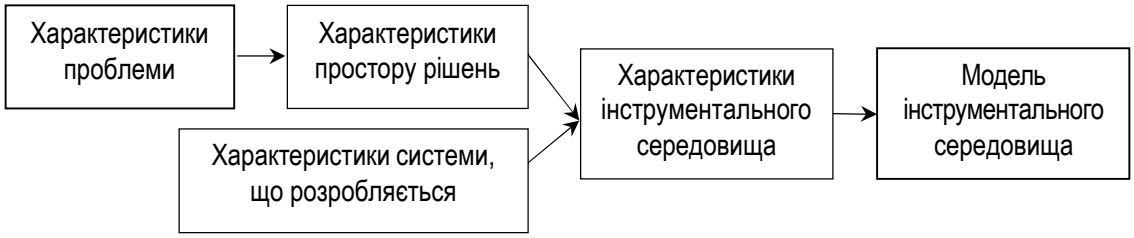

Рис. 14.1. Схема вибору інструментальних засобів для розробки експертної системи

Найважливішим питанням у процесі вибору інструментального середовища є питання способу визначення характеристик проблеми, для вирішення якої призначається експертна система. Ці характеристики можна звести до 4-х основних категорій.

1. Малий простір рішень, надійні дані та знання. Передбачається, що кількість альтернатив, які слід брати до уваги при пошуку рішення, є невеликою, всі дані є достовірними, а істинність правил не викликає сумнівів. Для вирішення проблем цієї категорії можна скористатися готовими рішеннями, тобто раніше створеною оболонкою на базі експертної системи, що вирішувала аналогічну проблему в іншій предметній області.
2. Ненадійні дані або знання. Якщо дані та(або) знання ненадійні, то існує небезпека, що дані, які вводяться в систему, не є достовірними, а правила в базі знань не мають однозначності. У цьому випадку в експертній системі потрібно комбінувати інформацію від декількох джерел та використовувати логіку нечітких міркувань.
3. Великий факторизований простір рішень. Простір пошуку можна назвати факторизованим, якщо існує можливість розділити його на декілька незалежних підпросторів, які можна обробляти окремо. Причому для різних підпросторів можуть бути використані різні множини правил або окремі підмножини однієї й тієї ж множини правил. Зазвичай, таке розбиття виконується на рівні проблеми, тобто велика загальна проблема розбивається на декілька дрібніших. Успіх у досягненні головної мети оцінюється за сукупністю успіхів у досягненні незалежних цілей.
4. Великий нефакторизований простір рішень. Простір рішень може виявитися нефакторизованим, якщо задача допускає вироблення приватного рішення будь-якого компонента тільки в контексті всього проекту. Загальний підхід до роботи у великому просторі пошуку полягає в тому, щоб послідовно розглядати його на різних

рівнях абстракції. Тобто потрібно використовувати варіанти опису простору з різним рівнем урахування деталей. Рішення проблеми таким методом часто називають спадним уточненням.

З'ясувавши характеристики проблеми, для вирішення якої розробляється експертна система, можна визначитися з властивостями простору рішень. Потім вони розглядаються спільно з передбачуваними характеристиками системи: моделлю подання знань, напрямком логічного висновку, способом формування пояснень. У результаті виявляються бажані характеристики інструментального середовища, які дозволяють підібрати потрібну модель інструментального середовища. Як показує практика, більшість розробників явно або неявно використовують саме такий підхід при створенні експертних систем.

У процесі вибору інструментального середовища важливе значення відіграють також наступні аспекти:

- наскільки просте середовище у використанні;
- як швидко розробники експертної системи зможуть оволодіти методикою роботи в цьому середовищі;
- яку підтримку готова надавати фірма-розробник середовища;
- яка буде загальна вартість середовища з урахуванням прямих і непрямих витрат.

У матеріалах, що описують програмні засоби з розробки експертних систем, можна зустріти твердження, що даним інструментом "може успішно користуватися програміст, мало знайомий із технологіями штучного інтелекту", або навіть непрограміст. Проте практика показує, що це не так. Практичне оволодіння типовими інструментальними засобами проектування експертних систем не поступається за складністю оволодінню новою мовою програмування.

Як правило, типове середовище розробки експертних систем підтримує чотири режими роботи:

- підготовка та редагування бази знань;
- використання бази знань для виконання консультацій, тобто прогін програми;
- виявлення та усунення помилок на стадії компіляції;
- виявлення та усунення помилок на стадії виконання.

Як показав досвід, навіть досвідчені програмісти важко засвоюють методику сумісного використання цих режимів у процесі проектування експертної системи. Це пов'язано, перш за все, з тим, що стандартна стратегія розробки бази знань передбачає постійне нарощування її обсягу. Тому інженеру по знанням доводиться виконувати ітеративні процедури поповнення бази знань значно частіше, ніж звичайному програмісту виконувати розширення функцій програми.

В міру збільшення складності проектованої системи відбувається збільшення обсягу бази знань, додавання до розгляду різного роду невизначеностей, включення в роботу системи додаткових режимів. Тому стратегія проектування вимагає від розробників дедалі більш ретельної попередньої підготовки. Крім того, можна виділити інші характерні причини складності вибору інструментального середовища розробки експертних систем:

- більшість розвинених середовищ розробки надто дорогі для того, щоб купувати їх для проведення порівняльного аналізу;
- час, необхідний для засвоєння навиків роботи з системою та виявлення її сильних та слабких сторін, дуже великий, тому складно проводити порівняння конкуруючих моделей на практиці;
- термінологія, яку застосовують у документації розробники різних систем, істотно відрізняється, тому проводити їх порівняння на основі технічної документації достатньо важко.

Останнє зауваження справедливе відносно більшості програмних продуктів, що пропонуються на ринку. Коли ж йдеться про програмні засоби, пов'язані з областю штучного інтелекту, то новизна та незвичність термінології ще більш посилює проблему. Вже давно в середовищі фахівців існує думка, що порівняння конкуруючих систем одного класу можна виконувати тільки після ретельного вивчення їх на практиці.

На завершення слід зазначити, що будь-які інструментальні засоби потребують адекватної методології користування ними. В літературних джерелах, присвячених програмуванню, для виразу рівня адекватності часто застосовується поняття "стиль програмування". Дотримуючись загальноприйнятого стилю програмування, можна уникнути небезпеки зробити програмну систему нежиттєздатною ще до закінчення етапу розробки проекту. Тому доречно буде розглянути загальні рекомендації, які визначають стиль розробки експертних систем:

1. Задача, яку передбачається вирішувати за допомогою експертної системи, повинна бути повністю під силу експерту-людині.
2. Задача повинна бути чітко сформульована. Краще створити систему, яка зможе надійно вирішувати обмежену задачу, ніж систему, що претендує на рішення широкого класу задач, проте дає правильне рішення лише час від часу.
3. Починаючи з першої стадії роботи над системою, необхідно визначити, як вона буде вдосконалюватися, та окреслити межі, яких вона повинна досягти в процесі еволюції.
4. Слід ретельно відпрацювати поведінку системи на наборі окремих випадків та організувати бібліотеку таких випадків. Тобто прикла-

ди, які застосовувались на етапі проектування, повинні бути репрезентовані.

5. Потрібно відділити ті знання, які є специфічними для певної предметної області, від знань, які стосуються загальної методики рішення проблем. Бажано, наскільки це можливо, спростити машину логічного висновку в системі.
6. Необхідно на перших стадіях проектування системи розробити однозначні угоди про оформлення програм. Це надасть їй одноманітний вигляд.
7. Бажано поступатися продуктивністю програми, якщо це зробить її більш зрозумілою та спростить її супровід. Це необхідно, оскільки в роботі інтерактивних експертних систем багато часу йде на діалог з користувачем та звернення до баз знань.
8. Як тільки постане питання про розробку нового прототипу системи, від попереднього необхідно відмовитися. Багато проектів зазнали невдачі лише тому, що їх автори не змогли позбавитися прихильності до першого варіанта реалізації власних ідей. Звичайно, у процесі розробки нового прототипу потрібно враховувати досвід створення попереднього, але тільки досвід, а не програмний код.
9. Розробка експертної системи, яка буде здатна успішно працювати, вимагає наполегливості та терпіння професійного програміста, залучення до роботи досвідченого експерта у відповідній області та певного рівня примусу з боку керівництва.

Наведений перелік рекомендацій, звичайно, не можна вважати вичерпним, зважаючи хоча б на той факт, що інструментальні засоби постійно знаходяться в процесі розвитку. Тим не менше, вони можуть претендувати на звання фундаментальних у процесі розробки експертних систем. Програміст, який буде дотримуватись цих рекомендацій, відповідно, буде дотримуватися такого стилю програмування, що дозволить максимізувати ймовірність успіху в розробці програмного продукту з ознаками штучного інтелекту.

ПИТАННЯ ТА ЗАВДАННЯ

1. Яку методологію застосовують практично всі інструментальні засоби, що використовуються в процесі розробки експертних систем?
2. Охарактеризуйте відмінності в організації процесів розробки експертної системи від розробки програмних продуктів інших видів.
3. Яку проблему містить у собі технологія послідовного нарощування функціональних можливостей експертних систем?

4. Яким чином створюються системи типу оболонки експертних систем?
5. В чому полягає привабливість застосування оболонок експертних систем?
6. Які переваги можна отримати завдяки використанню мов програмування високого рівня для створення прототипів експертних систем?
7. У чому полягає основна причина складності використання об'єктно-орієнтованого стилю у програмуванні експертних систем?
8. Перерахуйте можливості типової мови логічного програмування, які дозволяють використовувати її для створення експертних систем.
9. Що являє собою середовище розробки експертних систем, яке підтримує декілька парадигм?
10. Назвіть чотири парадигми програмування, які були об'єднані в середовищі LOOPS.
11. Що розуміють під додатковими модулями розробки експертних систем?
12. Наведіть приклад додаткового модуля експертних систем.
13. Перерахуйте найбільш характерні складнощі розробки експертних систем.
14. Яких загальних правил потрібно дотримуватись у процесі вибору інструментарію для розробки експертної системи?
15. Намалюйте схему вибору інструментальних засобів для розробки експертної системи.
16. До яких категорій можна звести характеристики проблеми, для вирішення якої призначається експертна система?
17. Назвіть додаткові аспекти, які потрібно враховувати в процесі вибору інструментального середовища для розробки експертної системи.
18. Які режими роботи підтримує тривіальне середовище розробки експертних систем?
19. Назвіть характерні причини, що зумовлюють складність вибору інструментального середовища розробки експертних систем.
20. Перерахуйте загальні рекомендації, які визначають стиль розробки експертних систем.

ТЕСТИ

1. Яке з перерахованих визначень найбільш точно характеризує термін "прототип" по відношенню до програмного забезпечення?
   а) модель програми, яка формується із специфікацій кінцевого продукту;

б) модель програми, яка функціонально еквівалентна підмножині кінцевого продукту;
в) модель програми, яка створюється в результаті планування та проектування компонентів кінцевого продукту.

2. Яка з категорій засобів розробки експертних систем найбільш підходить для програмування процедур управління?
   а) оболонки експертних систем;
   б) мови програмування високого рівня;
   в) середовище програмування, що підтримує декілька парадигм;
   г) додаткові модулі.
3. Які дії є характерними для процесу створення оболонки експертних систем?
   а) з існуючої експертної системи видаляються компоненти, які не мають вузької спеціалізації;
   б) з існуючої експертної системи видаляються компоненти, специфічні для області її застосування;
   в) спочатку розробляється специфікація продукту, потім виконується планування, проектування компонентів, їх реалізація, компоновка комплексу і тестування кінцевого варіанта.
4. Які недоліки характерні для мов опису продукційних правил?
   а) наявність синтаксичних та семантичних обмежень;
   б) труднощі при реалізації деяких типів структур управління ходом виконання;
   в) труднощі при співвідношенні програмних об’єктів з абстрактними поняттями та категоріями предметної області.
5. В якій з парадигм програмування інструментального середовища LOOPS структуровані об’єкти мають властивості процедур і даних?
   а) процедурно-орієнтоване програмування;
   б) програмування, орієнтоване на правила;
   в) об’єктно-орієнтоване програмування;
   г) програмування, орієнтоване на дані.
6. Який з перерахованих чинників стимулює застосування додаткових модулів експертних систем?
   а) підвищення швидкості розробки продуктів;
   б) необхідність знаходити рішення в умовах невизначеності;
   в) позбавлення розробника необхідності зважати на деякі деталі реалізації системи;
   г) необхідність інтегрувати експертну систему з програмними продуктами інших класів.

7. Яка з характерних складностей розробки експертних систем призводить до уповільнення роботи системи та ускладнює управління нею?
   а) база знань не має достатнього обсягу;
   б) система містить велику кількість дуже специфічних правил;
   в) знання надто тісно переплетені з іншими частинами системи.
8. Для якої категорії характеристик проблем, що вирішуються експертними системами, характерне застосування методу спадного уточнення?
   а) малий простір рішень, надійні дані та знання;
   б) ненадійні дані або знання;
   в) великий факторизований простір рішень;
   г) великий нефакторизований простір рішень.
9. В чому полягає основна складність, притаманна програмуванню експертних систем?
   а) необхідність постійного нарощування бази знань;
   б) необхідність використання бази знань для пошуку рішень;
   в) необхідність виявлення та усунення помилок на стадії компіляції;
   г) необхідність виявлення та усунення помилок на стадії виконання.
10. Яка з перерахованих рекомендацій не відповідає стилю розробки експертних систем?
   а) як тільки постане питання про розробку нового прототипу системи, від попереднього необхідно відмовитися;
   б) не можна жертвувати продуктивністю системи, навіть якщо це робить її більш зрозумілою та спрощує супровід;
   в) задача, яку передбачається вирішувати за допомогою проектованої системи, повинна бути цілком під силу експерту-людині;
   г) потрібно відокремити ті знання, які специфічні для певної предметної області, від знань, що стосуються загальної методики рішення проблем.

РОЗДІЛ 15. БАЗОВІ КОНЦЕПЦІЇ НЕЙРОННИХ МЕРЕЖ

15.1. Фундаментальне поняття нейронних мереж

У 80-х роках XX століття до числа відомих принципів програмування увійшов новий напрям розробок інформаційних систем, який одержав назву штучних нейронних систем. Цей напрям був заснований на відкритті одного із способів обробки інформації мозком людини, згідно з яким процеси рішення задач моделюються шляхом навчання моделей нейронів, що утворюють мережу. Зазначений підхід іноді згадується під назвою коннекціонізму (від англ.. *connection* – з'єднання).

Фундаментальне поняття нейронних мереж обумовлює структура системи обробки інформації, що складається з великої кількості елементів обробки даних – нейронів, які сполучаються зв'язками – синапсами. Конфігурація нейронної мережі налаштовується на реалізацію конкретного додатка за допомогою процесу засвоєння знань, тобто навчання. При цьому аналогічно біологічним системам навчання передбачає внесення змін у характеристики синаптичних з'єднань між нейронами.

Передбачено багато способів класифікації штучних нейронних систем. Найбільш корисна, з практичної точки зору, класифікаційна ознака полягає в наявності або відсутності потреби застосовувати для такої системи навчальну сукупність даних. Якщо навчальна сукупність передбачена, то штучна нейронна система називається заснованою на контрольованій моделі. В іншому випадку модель вважається неконтрольованою.

Прикладом контрольованої штучної нейронної системи є система, яка використовується для розпізнавання образів. На відміну від неї, у випадках, коли точно не відомо якими повинні бути вихідні дані, штучна нейронна система може бути застосована як класифікатор для групування вхідних даних. Наприклад, розпізнавання хворих та здорових людей у разі виявлення спалахів захворювання.

Нейронні мережі також можуть класифікуватися згідно з наступними характеристиками:

- спосіб з'єднання нейронів;
- застосування обчислень, що виконують нейрони;
- спосіб передачі шаблонів активності по мережі;
- спосіб та швидкість навчання.

Нейронні мережі застосовувалися для вирішення практичних задач усіх типів. Основною перевагою нейронних мереж є те, що вони дозволяють вирішувати задачі, які виявляються надто складними для звичайних технологій. Маються на увазі задачі, які не мають алгоритмічного рішення, або для яких алгоритмічне рішення є дуже складним, щоб його можна було визначити аналітично. Власне кажучи, нейронні мережі добре підходять для вирішення задач, які успішно вирішують люди, але не можуть пояснити, як вони це роблять.

До числа задач нейронних систем належать задачі розпізнавання образів та прогнозування подій, зокрема, знаходження тенденції зміни даних. У теперішні часи штучний інтелект на основі нейронних мереж широко використовується під час аналізу прихованих закономірностей для виявлення в історичних даних таких шаблонів, якими можна керуватися в майбутньому. Наприклад, аналіз прихованих закономірностей у даних може застосовуватися в банківській установі для визначення характеристик потенційного зловмисника серед позичальників.

Крім того, нейронні мережі використовуються як інтерфейсна частина в експертних системах, які обробляють великі обсяги вхідних даних від датчиків, і від яких вимагається реакція в реальному часі. Зокрема, у 1980-х роках нейронна мережа, яка експлуатувалася на звичайному мікрокомп'ютері, дозволила одержати дуже якісне рішення задачі комівояжера за 0,1 секунди. Цей результат набагато перевищував час отримання оптимального рішення із застосуванням традиційної алгоритмічної системи на аналогічному устаткуванні, який складав на той час одну годину.

Слід зазначити, що задача комівояжера має важливе практичне значення, оскільки є класичною задачею, яку доводиться вирішувати під час маршрутизації пакетів в системі передачі даних. Задача пошуку оптимальних маршрутів є важливим засобом мінімізації часу, оскільки від цього залежать ефективність та швидкодія, як при маршрутизації пакетів даних через Інтернет, так і при доставці посилок поштою численним адресатам.

15.2. Характеристики штучної нейронної системи

Штучна нейронна система може розглядатися як аналоговий обчислювальний комплекс, в якому використовуються прості елементи обробки даних, які, здебільшого, паралельно сполучені один з одним. Елементи обробки даних виконують дуже прості логічні або арифметичні операції над своїми вхідними даними. Основою функціонування штучної нейронної системи є те, що з кожним елементом такої системи пов'язані вагові коефіцієнти. Ці вагові коефіцієнти представляють

інформацію, що зберігається в системі. Схема типового штучного нейрону зображена на рис. 15.1.

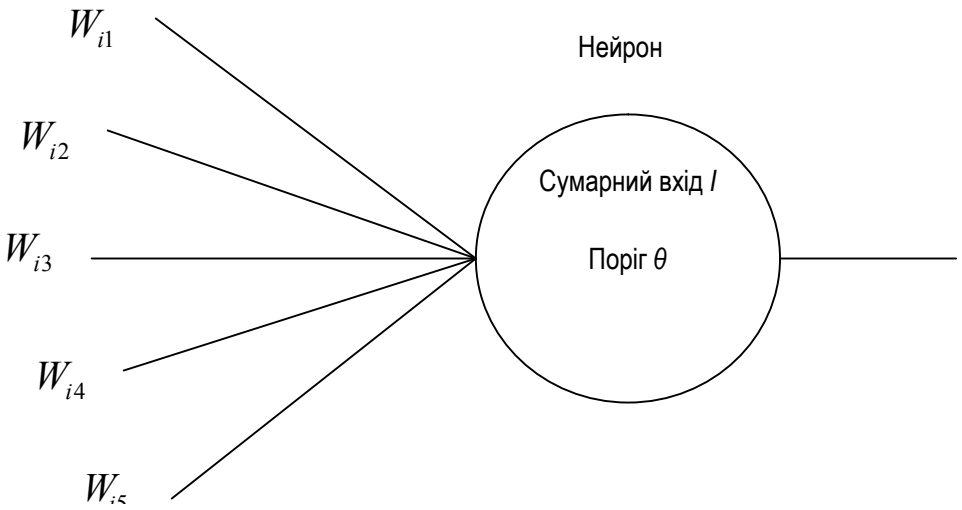

Рис. 15.1.Типовий штучний нейрон

Нейрон може мати багато входів, але тільки один вихід. Людський мозок містить приблизно 10^{11} нейронів, і кожен нейрон може мати тисячі з'єднань з іншими. Вхідні сигнали I_j нейрона помножуються на вагові коефіцієнти W_{ij} та складаються для отримання сумарного входу нейрона – *I*:

$$I = \sum_j W_{ij} I_j$$

Функція, яка зв'язує вихід нейрона з його входами, називається функцією активізації. Вона має вигляд сигмоідальної функції $(1+e^{-x})^{-1}$. Формалізація реакції нейрона полягає у тому, що вихідний сигнал прямує до однієї з меж при отриманні дуже малих та дуже великих вхідних сигналів. Крім того, з кожним нейроном пов'язане порогове значення – θ, яке у формулі обчислення вихідного сигналу віднімається від загального вхідного сигналу. В результаті, вихідний сигнал нейрона – *O* часто описується наступним чином:

$$O = \frac{1}{1+e^{-(I-\theta)}}$$

Як приклад можна навести штучну нейронну мережу, яка здатна обчислювати значення логічної операції "виключне АБО" (XOR) від її входів із використанням способу, названого зворотним розповсюдженням. Структура мережі із зворотним розповсюдженням (відома також як мережа, заснована на узагальненому дельта-правилі) представлена на рис. 15.2.

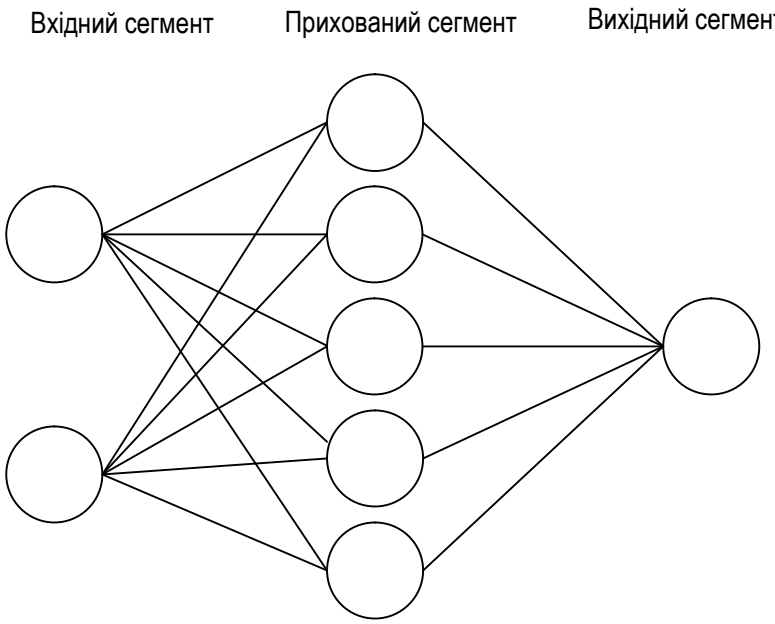

Рис. 15.2. Мережа із зворотним розповсюдженням

Мережа із зворотним розповсюдженням, зазвичай, поділяється на три сегменти, хоча можуть бути сформовані також додаткові сегменти. Сегменти (сегмент), що знаходяться між вхідним та вихідним сегментами, називаються прихованими сегментами, оскільки зовнішній світ сприймає наочно тільки вхідний і вихідний сегменти. Мережа, що обчислює значення логічної операції "виключне АБО", видає на вихід істинне значення, тільки у випадках, коли не на всіх її входах є істинні значення або не на всіх входах є помилкові значення. Кількість вузлів у прихованому секторі може змінюватись, залежно від мети проекту.

Слід зазначити, що нейронні мережі не вимагають програмування в звичному значенні цього слова. Для навчання нейронних мереж застосовуються спеціальні алгоритми навчання нейронних мереж, такі як зустрічне розповсюдження та зворотне розповсюдження. Програміст "програмує" мережу, задаючи вхідні дані та відповідні вихідні дані. Мережа навчається, автоматично корегуючи вагові коефіцієнти для синаптичних з'єднань між нейронами.

Вагові коефіцієнти, разом із пороговими значеннями нейронів, визначають характер розповсюдження даних по мережі і, тим самим, задають правильний відгук на дані, що використовуються в процесі навчання. Навчання мережі з метою отримання правильних відповідей може потребувати багато часу. Наскільки багато – залежить від того, яка кількість образів повинна бути засвоєна в ході навчання мережі, а також від можливостей застосовуваних апаратних та допоміжних про-

грамних засобів. Проте після завершення такого навчання мережа здатна надавати відповіді з високою швидкістю.

За своєю архітектурою штучна нейронна система відрізняється від інших обчислювальних систем. У класичній інформаційній системі реалізується можливість з'єднання дискретної інформації з елементами пам'яті. Наприклад, зазвичай, інформаційна система зберігає дані про конкретний об'єкт у групі суміжних елементів пам'яті. Отже, можливість доступу та маніпулювання даними досягається за рахунок створення взаємно однозначного зв'язку між атрибутами об'єкта та адресами елементів пам'яті, в яких вони записані.

На відміну від таких систем, моделі штучних нейронних систем розробляються на основі сучасних теорій функціонування мозку, згідно з якими інформація представлена в мозку за допомогою вагових коефіцієнтів. При цьому безпосередньої кореляції між конкретним значенням вагового коефіцієнта і конкретним елементом збереженої інформації не існує.

Таке розподілене представлення інформації аналогічне технології збереження та представлення зображень, яка використовується в голограмах. Згідно з цією технологією лінії голограми діють як дифракційні решітки. За їх допомогою під час проходження лазерного променя відтворюється збережене зображення, проте самі дані не піддаються безпосередній інтерпретації.

Нейронна мережа виступає в ролі прийнятного засобу рішення задачі, коли присутня велика кількість емпіричних даних, але немає алгоритму, який був би спроможний забезпечити отримання достатньо точного рішення з необхідною швидкістю. В даному контексті технологія представлення даних штучної нейронної системи має суттєві переваги перед іншими інформаційними технологіями. Ці переваги можна сформулювати таким чином:

1. Пам'ять нейронної мережі є відмовостійкою. При видаленні окремих частин нейронної мережі відбувається лише зниження якості інформації, що в ній зберігається, але не повне її зникнення. Це відбувається тому, що інформація зберігається в розподіленій формі.
2. Якість інформації в нейронній мережі, яка підлягає скороченню, знижується поступово, пропорційно тій частині мережі, що була видалена. Катастрофічної втрати інформації не відбувається.
3. Дані в нейронній мережі зберігаються природним чином за допомогою асоціативної пам'яті. Асоціативною пам'яттю називають таку пам'ять, в якій достатньо виконати пошук частково представлених даних, щоб повністю відновити всю інформацію. У цьому полягає відмінність асоціативної пам'яті від звичайної пам'яті, в якій

отримання даних здійснюється шляхом вказівки точної адреси відповідних елементів пам'яті.

4. Нейронні мережі дозволяють виконувати екстраполяцію та інтерполяцію на основі інформації, що зберігається в них. Тобто навчання дозволяє додати мережі здатності здійснювати пошук важливих особливостей або зв'язків в даних. Після цього мережа в змозі екстраполювати та виявляти зв'язки в нових даних, що до неї надходять. Наприклад, в одному експерименті було проведене навчання нейронної мережі на гіпотетичному прикладі. Після закінчення навчання мережа набула здатності правильно відповідати на питання, відносно яких навчання не проводилося.
5. Нейронні мережі – пластичні. Навіть після видалення певної кількості нейронів може бути проведене повторне навчання мережі до її первинного рівня (звісно, якщо в ній залишилася достатня кількість нейронів). Така особливість є також характерною для мозку людини, в якому можуть бути пошкоджені деякі частини, але з часом, за допомогою навчання, досягнуто первинного рівня навичок та знань.

Завдяки таким особливостям штучні нейронні системи стають дуже привабливими для застосування в роботизованих космічних апаратах, устаткуванні нафтопромисловості, підводних апаратах, засобах управління технологічними процесами та в інших технічних пристроях, які повинні функціонувати тривалий час без ремонту в несприятливому середовищі. Штучні нейронні системи не тільки дозволяють розв'язати проблему надійності, але й надають можливість зменшити експлуатаційні витрати завдяки своїй пластичності.

Проте в цілому штучні нейронні системи не зовсім підходять для створення додатків, в яких потрібні складні математичні розрахунки або пошук оптимального рішення. Крім того, застосування штучної нейронної системи не буде найкращим варіантом у випадку, якщо існує алгоритмічне рішення, яке вже надало позитивний результат внаслідок практичного застосування для рішення подібних задач.

15.3. Практичне спрямування нейронних мереж

Розробка штучних нейронних мереж почалася з досліджень математичного моделювання нейронів, проведених вченими Маккаллохом та Піттсом у 1943 році. Пояснення процесу навчання за допомогою нейронів, запропоноване вченим Хеббом (Hebb) у 1949 році. У хеббівському навчанні ефективність активізації одних нейронів іншими зростає відповідно до збільшення кількості запусків. Термін "за-

пуск" означає, що нейрон випускає електрохімічний імпульс, здатний стимулювати інші, пов'язані з ним нейрони.

Підвищення ефективності активізації свідчить про те, що провідність з'єднань між нейронами на ділянках їх сполучення – синапсів – зростає із збільшенням кількості запусків. У штучній нейронній системі для моделювання змін провідності синапсів природних нейронів застосовується спосіб коректування вагових коефіцієнтів з'єднань між нейронами.

У 1961 році Розенблатт оприлюднив свою книгу, що значно вплинула на подальший хід досліджень нейронних систем. У ній розглядався різновид штучної нейронної системи, що отримав назву "персептрон". Персептрон складається з двох сегментів нейронів і дає можливість використовувати простий алгоритм навчання. Вагові коефіцієнти в ньому повинні встановлюватися вручну, на відміну від сучасних штучних нейронних систем, які здатні самостійно встановлювати власні вагові коефіцієнти на основі навчання.

Рання епоха дослідження персептронів завершилася в 1969 році, коли вчені Мінський (Minsky) і Пейперт (Papert) оприлюднили книгу "Perceptrons", в якій показали теоретичні обмеження персептрона, що досі розглядався як обчислювальна машина загального призначення. Автори роботи підкреслили той факт, що персептрони здатні обчислювати тільки 14 з 16 основних логічних функцій, тому мають критичні недоліки. Це означало, що персептрон не можна вважати обчислювальним пристроєм загального призначення.

Через ці відкриття бюджетне фінансування досліджень у сфері штучних нейронних систем було скорочене на користь символічного підходу до розробки штучного інтелекту з використанням таких мов, як LISP. У 1970-х роках набувають широкого поширення нові способи представлення символічної інформації в розробках штучного інтелекту на основі фреймів, запропоновані Мінським. Надалі, на основі фреймів, був створений сучасний підхід, в якому застосовуються сценарії.

На початку 1980-х років вчені, які складали групу з дослідження можливостей проведення паралельних обчислень, виявили зацікавленість до теорій пізнання, заснованих на нейронних мережах. У результаті, Хопфілд (Hopfield) розробив надійну теоретичну підставу застосування штучних нейронних систем на прикладі так званої мережі Хопфілда. З її допомогою вчений довів, що штучні нейронні системи можуть успішно вирішувати найрізноманітніші задачі. Зокрема, Хопфілд показав, що за допомогою штучної нейронної системи задачу комівояжера можна вирішувати за одну годину, тоді як у звичних ал-

горитмічних рішеннях відбувається комбінаторний вибух. Загальна структура мережі Хопфілда наведена на рис. 15.3.

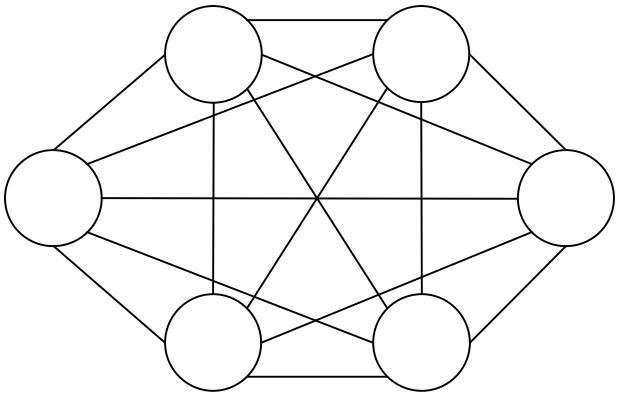

Рис. 15.3. Штучна нейронна мережа Хопфілда

Ще одним широко відомим типом штучної нейронної системи є мережа із зустрічним розповсюдженням, запропонована Хехтом-Нілсеном (Hecht-Nielsen) у 1986 році. З її допомогою було встановлено, що один з важливих теоретичних результатів у математиці – теорему Колмогорова – можна інтерпретувати як доказ того, що трисегментна мережа з n входами та 2^{n+1} нейронами в прихованому сегменті дозволяє представити будь-яку безперервну функцію.

Важливий приклад ефективного навчання за допомогою зворотного розповсюдження був продемонстрований за допомогою нейронної мережі, яка навчалася правильній вимові слів, представлених у вигляді тексту. Ця штучна нейронна система навчалася шляхом коректування свого виходу за допомогою пристрою перетворення тексту в мову, що має назву DECTalk.

Для розробки правил грамотної вимови, які були застосовані в пристрої DECTalk, знадобилося близько двадцяти років лінгвістичних досліджень. На відміну від нього, штучна нейронна система засвоїла еквівалентні навички вимови за одну добу, після прослуховування правильної вимови словосполучень під час читання тексту. При цьому в цій штучній нейронній системі не було закладено шляхом програмування жодних первинних лінгвістичних навичок.

Штучні нейронні системи були також застосовані для розпізнавання радарних цілей за допомогою комп’ютерів. Існує перспектива створення на основі нових реалізацій нейронних мереж із застосуванням оптичних компонентів, нового покоління обчислювальної техніки – оптичних комп’ютерів. За прогнозами, їх швидкодія буде здатна перевищити швидкодію сучасних електронних комп’ютерів у мільйони разів.

Привабливість реалізацій штучних нейронних систем на основі оптичних компонентів обумовлена тим, що світло характеризується властивістю паралельного розповсюдження. Це означає, що не відбувається взаємного впливу між променями світла, що спільно розповсюджується. Крім того, такі оптичні компоненти, як дзеркала, лінзи, високошвидкісні програмовані модулятори світла, масиви оптичних пристроїв та дифракційні решітки дозволяють достатньо легко створювати та здійснювати маніпуляції з великими кількостями фотонів.

Штучні нейронні системи можуть бути також покладені в основу експертних систем. Зокрема, в одній із відомих експертних систем штучна нейронна система є базою знань у сфері діагностики захворювань, що була сформована шляхом навчання на основі прикладів, взятих із практичної медицини. Ця експертна система робить спроби розпізнати захворювання, використовуючи як вхідні дані його симптоми.

Для того, щоб використовувати базу знань на основі штучної нейронної системи, була спроектована машина логічного висновку, яка має назву MACIE (Matrix Controlled Inference Engine). У цій системі використовується прямий логічний висновок – для формування логічних висновків, та зворотний логічний висновок – для передачі користувачу запитів на надання додаткових даних, потрібних для знаходження рішень. Машина MACIE здатна інтерпретувати відповіді штучної нейронної системи і виробляти правила IF-THEN, що застосовуються для пояснення її знань.

У такій експертній системі на основі штучної нейронної системи використовується індуктивне навчання. Це означає, що система логічно формує, за допомогою прикладів, інформацію, що міститься в її базі знань. Нагадаємо, що індукція – це процес логічного виведення загального висновку із проміжних.

Існує значна кількість комерційних програмних засобів знаходження рішень, що дозволяють явно виробляти правила на основі прикладів. Індуктивне навчання використовується для усунення вузьких місць, пов’язаних із набуттям знань. Все навантаження набуття знань покладається на експертну систему. Завдяки цьому з’являється можливість скорочувати час розробки та підвищувати надійність інформаційної системи, здатної логічним шляхом виводити правила, які до тих пір не були відомі.

ПИТАННЯ ТА ЗАВДАННЯ

1. Дайте визначення нейронній мережі.
2. Чим відрізняються контрольовані нейронні мережі від неконтрольованих?
3. Наведіть приклад контрольованої штучної нейронної системи.

4. Згідно з якими характеристиками можуть класифікуватися нейронні мережі?
5. Охарактеризуйте основні задачі, для вирішення яких використовується технологія нейронних мереж.
6. Які компоненти штучної нейронної системи представляють інформацію, що в ній зберігається?
7. Намалюйте схему типового штучного нейрону.
8. За якою формулою знаходиться сумарний вхід нейрона?
9. За допомогою якої формули можна обчислити вихідний сигнал нейрона?
10. Як можна використати штучну нейронну систему для обчислення значення логічної операції "виключне АБО"?
11. Опишіть у загальному вигляді процес програмування нейронних систем.
12. Які переваги має технологія представлення даних штучної нейронної системи перед іншими інформаційними технологіями?
13. Назвіть придатні та непридатні області застосування технології штучних нейронних систем.
14. Яка система має назву персептрон?
15. Наведіть приклади вдалого практичного застосування технології нейронних систем.

ТЕСТИ

1. Які дії передбачає процес навчання штучних нейронних систем?
   а) зміна загальної кількості нейронів;
   б) внесення змін у характеристики з'єднань між нейронами;
   в) внесення змін у структуру мережі шляхом перенаправлення з'єднань між нейронами.
2. До якого типу штучних нейронних систем належить система, яка використовується як класифікатор для групування вхідних даних?
   а) контрольована нейронна система;
   б) неконтрольована нейронна система;
   в) нейронна система з ваговими коефіцієнтами.
3. Який тип з'єднань між елементами переважає в штучній нейронній системі?
   а) ієрархічний;
   б) послідовний;
   в) паралельний.
4. Які елементи нейронної системи представляють інформацію, яка в ній зберігається?
   а) вагові коефіцієнти;
   б) порогові значення нейронів;

в) функції, які зв’язують входи та виходи нейронів.

5. Скільки входів та виходів має нейрон?
   а) один вхід, один вихід;
   б) один вхід, множину виходів;
   в) множину входів, один вихід;
   г) множину входів, множину виходів.
6. Як використовується порогове значення нейрона при обчисленні вихідного сигналу?
   а) порогове значення додається до значення вхідного сигналу;
   б) порогове значення віднімається від значення вхідного сигналу;
   в) порогове значення помножується на значення вхідного сигналу;
   г) значення вхідного сигналу зводиться до ступеня порогового значення.
7. В якому випадку нейронна мережа зі зворотним розповсюдженням, яка обчислює значення логічної операції XOR, видає істинний вихід?
   а) на всіх її входах є істинні значення;
   б) на всіх входах є помилкові значення;
   в) не на всіх входах є помилкові значення.
8. Які набори даних необхідно надати нейронній мережі для її навчання?
   а) вхідні;
   б) вихідні;
   в) вхідні та вихідні.
9. Що розуміють під “пластичністю” нейронної мережі?
   а) можливість зберігати дані природним чином за допомогою асоціативної пам’яті;
   б) можливість відновлення первинного рівня навиків після видалення деякої кількості нейронів;
   в) поступове зниження якості збереженої інформації, після видалення окремих частин нейронної мережі.
10. Для вирішення якого типу задач доцільним є застосування нейронних мереж?
    а) для вирішення задач, де потрібен пошук оптимального рішення;
    б) для вирішення задач, для яких існують ефективні алгоритмічні рішення;
    в) для вирішення задач, в яких потрібні складні математичні розрахунки;
    г) для вирішення задач, в яких потрібна обробка великої кількості емпіричних даних за короткий проміжок часу.

СПИСОК ЛІТЕРАТУРИ

1. Астахова И. Ф. SQL в примерах и задачах [Текст] : учебное пособие / И. Ф. Астахова, А. П. Толстобров, В. М. Мельников. – Минск : Новое знание, 2002. – 176 с.
2. Арсеньєв Ю. Н. Принятие решений. Интегрированные ителлектуальные системы [Текст] : учебное пособие / Ю. Н. Арсеньев, С. И. Шелобаев, Т. Ю. Давыдова. – М. : ЮНИТИ-ДАНА, 2003. – 270 с.
3. Ахо А. В. Структуры данных и алгоритмы [Текст] / А. В. Ахо, Д. Э. Хопкрофт, Д. Д. Ульман. – М.-СПб.-К. : Вильямс, 2003.
4. Бєгун А. В. Технологія програмування: об`єктно-орієнтований підхід [Текст] : навчально-методичний посібник / А. В. Бєгун ; КНЕУ, Мін-во освіти України. – К. : КНЕУ, 2000. – 200 с.
5. Братко И. Алгоритмы искусственного интеллекта на языке PROLOG. [Текст] : учебное пособие / И. Братко. – М. : Вильямс, 2006. – 968 с.
6. Гаврилова Т. А. Базы знаний интеллектуальных систем [Текст] : учебное пособие / Т. А. Гаврилова, В. Ф. Хорошевский. – СПб. : Питер, 2001. – 384 с.
7. Галузинський Г. П. Перспективні технологічні засоби оброблення інформації [Текст] : навчально-методичний посібник / Г. П. Галузинський, І. В. Гордієнко ; Мін-во освіти і науки України, КНЕУ. – К. : КНЕУ, 2002. – 280 с.
8. Глибовець М. М. Штучний інтелект [Текст] : підручник / М. М. Глибовець, О. В. Олецький. – К. : КМ Академія, 2002. – 366 с.
9. Дейт К. Дж. Введение в системы баз данных [Текст] : : пер. с англ / К. Дж. Дейт. – 6-е изд. – К.-М. : Диалектика, 1998. – 784 с.
10. Джераратано Д. Экспертные системы [Текст] : пер. с англ / Джераратано Д., Райли Г. – М. : Вильямс, 2007. – 1152 с.
11. Кнут Д. Э. Искусство программирования: В 3 т. Т. 1: Основные алгоритмы [Текст] : учебное пособие : пер. с англ / Д. Э. Кнут. – 3-е изд. – М.-СПб.-К. : Вильямс, 2000. – 720 с.
12. Кнут Д. Э. Искусство программирования : В 3 т. Т. 2: Получисленные алгоритмы [Текст] : учебное пособие : пер. с англ / Д. Э. Кнут. – 3-изд. – М.-СПб.-К. : Вильямс, 2000. – 832 с.
13. Кнут Д. Э. Искусство программирования: В 3 т. Т. 3: Сортировка и поиск [Текст] : учебное пособие : пер. с англ / Д. Э. Кнут. – 2-е изд. – М.-СПб.-К. : Вильямс, 2000. – 877 с.
14. Ковалюк Т. В. Основи програмування [Текст] : підручник / Т. В. Ковалюк ; ред. М. З. Згуровський. – К. : BHV, 2005. – 384 с.

15. Лафоре Р. Объектно-ориентированное программирование в С++ [Текст] / Р. Лафоре. – 4-е изд. – СПб. : Питер, 2004. – 924 с.
16. Люгер Д. Искусственный интеллект: структуры и стратегии решений сложных прблем [Текст] : пер. с англ. / Д. Люгер. – М.: Вильямс, 2006. – 610 с.
17. Маслов В. П. Інформаційні системи і технології в економіці [Текст] : навчальний посібник / В. П. Маслов ; Мін-во освіти і науки України. – К. : Слово, 2003. – 264 с.
18. Проектирование и реализация баз данных Microsoft SQL Server 2000: Учебный курс MCAD/MCSE, MCDBA [Текст] : учебный курс. – 2-е изд., испр. – М. : Русская редакция, 2003. – 512 с.
19. Пасічник В. В. Організація баз даних та знань [Текст] : підручник / В. В. Пасічник, В. А. Резніченко. – К. : Видавнича група BHV, 2006. – 384 с.
20. Романов В. П. Интеллектуальные информационные системы в экономике [Текст] : учебное пособие / В. П. Романов ; ред. Н. П. Тихомиров ; Российская эконом. академия им. Г.В. Плеханова. – М. : Экзамен, 2003. – 496 с.
21. Роланд Ф. Д. Основные концепции баз данных [Текст] : учебное пособие / Ф. Д. Роланд. – М.: Вильямс, 2001. – 248 с.
22. Ситник В. Ф. Інтелектуальний аналіз даних (дейтамайнінг) [Текст] : навчальний посібник / В. Ф. Ситник, М. Т. Краснюк ; Мін-во освіти і науки України, ДВНЗ “КНЕУ ім. Вадима Гетьмана”. – К. : КНЕУ, 2007. – 376 с.
23. Ситник Н. В. Проектування баз і сховищ даних [Текст] : навчально-методичний посібник для самост. вивч. дисц. / Н. В. Ситник, М. Т. Краснюк. – Мін-во освіти і науки України, КНЕУ. – [Б. м. : б. и.], 2005. – 264 с.
24. Рассел С. Искусственный интеллект. Современный подход [Текст] : пер. с англ. / С. Рассел, П. Норвиг. – М. : Вильямс, 2006. – 1408 с.
25. Хайкин С. Нейронные сети полный курс [Текст] : пер. с англ / С. Хайкин. – М. : Вильямс, 2006. – 465 с.
26. Хоторн Р. Разработка баз данных Microsoft SQL Server 2000 на примерах [Текст] : пер. с англ. / Р. Хоторн. – М. : Вильямс, 2001. – 464 с.
27. Хювёнен Є. Мир Лиспа. Т. 1. Введение в язык Лисп и функциональное программирование (електронна версия) [Текст] / Є. Хювёнен, И. Сеппянен. – [Б. м. : б. и.]. – 458 с.
28. Шкарина Л. Язык SQL [Текст] : учебный курс / Л. Шкарина. – СПб. : Питер, 2001. – 592 с.

ПРЕДМЕТНИЙ ПОКАЖЧИК

Авторизація – етап перевірки прав доступу клієнта до серверної компоненти системи "клієнт-сервер", який визначає набір дій, що йому дозволені.

Атрибут – поіменований стовпець реляційної таблиці.

Аутентифікація – етап перевірки прав доступу клієнта до серверної компоненти системи "клієнт-сервер", упродовж якого відбувається його ідентифікація.

База даних – спеціальний програмний продукт, призначений для збору, зберігання й обробки інформації.

Бібліотека класів MFC – ієрархічно організована колекція класів, які здатні створювати архітектуру програмних додатків.

Бібліотека класів FCL – статична компонента каркасу Framework.Net, результат еволюції бібліотеки класів MFC.

Блокування – об'єкт бази даних, за допомогою якого СУБД встановлює рівень залежності користувача від ресурсу.

Ймовірність – кількісний спосіб урахування невизначеності.

Віртуальна машина – результат роботи виконавчого середовища каркасу Framework.Net з трансляції проміжного коду в командний код процесора тієї обчислювальної машини, на якій встановлене і функціонує виконавче середовище.

Декларативне програмування – спосіб досягнення певної мети в обробці інформації, шлях досягнення якої розглядається окремо від методів, що використовуються.

Динамічна Web-сторінка – Web-сторінка, яка надає користувачам інформацію, що відрізняється від перегляду до перегляду і зміст якої залежить від того, кому вона призначена.

Домен – область значень, які може приймати атрибут реляційної таблиці.

Евристика – приблизне правило, яке в програмному вигляді представляє знання людини, набуте в міру накопичення практичного досвіду внаслідок вирішення схожих проблем.

Експертна система – програмна система, яка оперує знаннями в певній предметній області з метою вироблення рекомендацій для вирішення проблем.

Експорт реляційних даних – процес передачі даних з екземпляра реляційної бази даних в певному, заданому користувачем, форматі.

Загальномовне виконавче середовище (CLR) – динамічна компонента каркасу Framework.Net, яку утворюють керований модуль, віртуальна машина, метадані, збірник сміття, обробник виключень, події, загальні специфікації.

Загальні специфікації – компонент CLR, що формує обмеження для забезпечення взаємодії різномовних компонентів одного додатка.

Збережена процедура – це відкомпільований сценарій послідовного виконання операторів SQL, який зберігається на сервері баз даних під унікальним ім'ям.

Збірник сміття – компонент CLR, відповідальний за звільнення оперативної пам'яті, зайнятої об'єктами, які не використовуються в подальшій роботі додатка.

Зворотний логічний висновок – формування міркувань у напрямі від гіпотези до фактів, які підтверджують гіпотезу.

Зовнішній ключ – атрибут реляційної таблиці, що забезпечує зв'язок з іншою таблицею, посилаючись на значення відповідного атрибуту.

Імпорт реляційних даних – процес отримання даних із зовнішніх, по відношенню до бази даних, джерел та розміщення їх у реляційних таблицях.

Індекси – об'єкти бази даних, що підвищують продуктивність запитів, допомагаючи СУБД знайти потрібний кортеж.

Інженерія знань – отримання знань від експерта-людини або з інших джерел з метою подальшого їх представлення в експертній системі.

Інтелектуальна система – програмна система, в якій використовується штучний інтелект.

Інструментальний засіб – поєднання мови програмування та пов'язаних з нею допоміжних програм, що дозволяють спростити розробку, налагодження та доставку користувачу програмних продуктів.

Каркас (Framework.NET) – платформа для створення, розгортання і запуску програмних додатків.

Квантор – поняття, що дозволяє надавати явні кількісні оцінки іншим поняттям з метою більш точного формулювання висловів.

Керований модуль – це переміщуваний виконавчий файл (Portable Executable, PE-файл), зміст якого формується компіляторами мов програмування каркасу Framework.Net проміжною мовою – IL (Intermediate Language).

Клієнт-сервер – технологія створення програмних рішень, в яких виконання задач з обробки інформації розподіляється між програмою-сервером і програмою-клієнтом.

Концептуальна модель бази даних – логічна модель бази даних представлена на вищому рівні абстракції за допомогою методів семантичного моделювання.

Концептуальна схема – абстракція, в якій конкретні об'єкти класифікуються згідно з їх загальними властивостями.

Кортеж – рядок реляційної таблиці.

Курсор – об'єкт реляційної бази даних, який встановлює відповідність із результуючим набором, що повертає оператор SELECT і вказує на один з його кортежів.

Ланцюжок володіння – приналежність права на використання об'єктів, що звертаються послідовно один до одного і розташовані в одній базі даних, до одного облікового запису користувача або ролі.

Метадані – маніфест керованого модуля, що включає всю необхідну для CLR інформацію: опис класів, які в ньому зберігаються, їх властивостей, методів, аргументів цих методів.

Мова програмування експертних систем – сукупність команд, записаних із застосуванням визначеного синтаксису та машини логічного висновку, призначеної для їх виконання.

Мова розмітки гіпертексту (HTML) – мова форматування вмісту текстових документів, яка використовується для створення Web-стрінок на основі фіксованого набору позначень – тегів.

Мова структурованих запитів (SQL) – декларативна мова, що дозволяє адмініструвати сервер реляційних баз даних, створювати його об'єкти та управляти ними, а також вносити, отримувати, модифікувати і видаляти дані з таблиць реляційних баз даних.

Модальне вікно – форма Windows-додатка, яка не допускає паралельну роботу з іншими формами за її межами.

Надмірність даних – ситуація, коли одні й ті ж дані зберігаються в різних місцях зовнішньої пам'яті (файлах).

Нейронна мережа – система обробки інформації, що складається з елементів обробки даних – нейронів, які сполучаються зв'язками – синапсами.

Немодальне вікно – форма Windows-додатка, яка допускає паралельну роботу з іншими формами за її межами.

Нормалізація – процес, здійснення якого дозволяє гарантувати ефективність структур даних у реляційній базі даних.

Область знань – знання експерта, що стосуються рішення конкретних задач.

Обробник виключень – компонент CLR, що здійснює перехоплення викинутих виключень програми внаслідок реакції на нестандартні ситуації та їх подальшу обробку.

Операційна система – набір програм, які забезпечують інтерфейс між прикладними програмами та апаратним забезпеченням комп'ютера.

Пакет SQL – група з одного або декількох операторів SQL, одночасно переданих серверу баз даних на виконання, з яких формується суцільна програмна одиниця – план виконання.

Первинний ключ – атрибут реляційної таблиці, що однозначно визначає кожен її кортеж (запис).

Правило – невелика, модульна форма знань, що відповідає певному фрагменту знань у предметній області.

Представлення – віртуальна таблиця, вміст якої визначається SQL-запитом, утвореним з використанням оператора SELECT.

Предикат – термін логіки та мовознавства, що означає конститутивний член вислову.

Пропозиціональна логіка – символічна логіка, що використовується для маніпулювання висловами.

Процедурне програмування – спосіб досягнення певної мети в обробці інформації шляхом реалізації алгоритму її досягнення з використанням мови (мов) програмування.

Прямий логічний висновок – метод формування міркувань у напрямку від фактів до висновків, які виходять із цих фактів.

Реляційна база даних – база даних, в якій дані на концептуальному рівні представляються у вигляді двомірних таблиць, зв'язок між якими організується на основі значень визначених стовпців.

Розширювана мова розмітки (XML) – гіпертекстова мова програмування, яка використовується для опису вмісту сукупності даних і способу виведення даних на різні пристрої або їх відображення на Web-сторінках.

Семантична мережа – візуалізований спосіб представлення пропозіціональної інформації, що використовується при проектуванні експертних систем.

Система управління базами даних (СУБД) – спеціалізований клас програмного забезпечення, що виконує роль посередника між програмними додатками і фізичним сховищем даних, реалізуючи можливості паралельного доступу й управління.

Спадкоємство – здатність об'єкта приймати властивості іншого об'єкта.

Статична Web-сторінка – Web-сторінка, вміст якої однаково відображається для всіх користувачів, не надаючи можливості змінювати її вміст.

Сценарій SQL – збережений у спеціалізованому файлі набір операторів SQL, який можна завантажувати в клієнтські програмні утиліти, призначені для передачі операторів SQL на виконання в СУБД та отримання результатів.

Трансакція – логічна одиниця роботи, що складається з набору операторів SQL і обробляється СУБД як єдине ціле.

Тригер – особлива збережена процедура, що автоматично виконується під час модифікації табличних даних у реляційних базах або після неї.

Форми – візуальні об'єкти, що забезпечують функціональність Windows-додатка.

Функціональна залежність – семантичне поняття, яке відображає певний семантичний зв'язок між атрибутами таблиці.

Хост – комп'ютер, що надає сервіси формату "клієнт-сервер" у режимі сервера через інтерфейси, на яких він є унікально визначеним.

Цілісність даних – стан бази даних, коли всі дані, що зберігаються в ній, не суперечать один одному.

Штучний інтелект – здатність автоматизованих систем здобувати, адаптувати, модифікувати та поповнювати знання з метою пошуку рішень задач, формалізація яких ускладнена.

Додаток А

Приклад практичного застосування сценарію SQL

```
-- 0000000000000000000000000000000000000000000000000000000000000000000000000000000
-- RestoreDB.sql – SQL сценарій відновлення бази даних автоматизованої банківської системи --
                    "СКАРБ" з BACKUP копії.
-- 0000000000000000000000000000000000000000000000000000000000000000000000000000000
set nocount on declare @namDB sysname, @pathDB varchar(100), @restoreDB varchar(255),
                 @bacData sysname, @namenLang sysname, @collate varchar(100),
                         @crEX varchar(2000), @bacLog sysname, @DB_CreOwn sysname,
                         @pathLOG varchar(100), @i int, @Cr_US tinyint, @namUS sysname,
                         @namDB_BC_AR sysname, @namDB_BC sysname, @e int
-- SETUP__SETUP__SETUP__SETUP__SETUP__SETUP__SETUP__SETUP__SETUP__SETUP__
select @namDB=N'ScarbDB', -- Ім'я: – відновлюваної БД
      @namUS=N'PolySysDBO', -- – користувача SQL Server (dbo БД)
      @pathDB='d:\Mssql2000\Data\', -- Шлях до файлів: – даних
      @pathLOG='d:\Mssql2000\LOG\', -- – журналу трансакцій
      @restoreDB='d:\Mssql2000\ScarbDB.bac', -- – копії
      @Cr_US=0
--   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
--   !! Адміністратор, що проводить відновлення повинен відредагувати
--   !! значення змінних: – @pathDB
--   !!                   – @restoreDB
  -- !! так, щоб їх значення відповідали розміщенню файлів даних БД, її журналу трансакцій
--   !! та місцю знаходження копії.
-- REM__REM__REM__REM__REM__REM__REM__REM__REM__REM__REM__REM__REM__
-- 1. Ім'я БД (змінна @namDB) може бути будь-яким, відмінним від 'ScarbDB'.
-- 2. Ім'я користувача (змінна @namUS) може бути будь-яким, відмінним від 'PolySysDBO'.
-- 3. Якщо БД немає на SQL Server, тобто відновлюється вперше, користувача @namUS або не
--        повинно бути на сервері – тоді він буде створений сценарієм, або бути раніше створеним з
--        роллю 'dbcreator', оскільки він буде переведений у статус 'dbo' цієї БД.
-- 4. Якщо БД є на SQL Server, її 'dbo' та всі підключені до неї користувачі будуть
--        "запам'ятовані" в тимчасовій таблиці і, після відновлення БД, перепідключені до неї:
--        ті які не 'dbo' – з роллю 'public'. Завдання @namUS необов'язково, а якщо ім'я вказане
--        - то за відсутності на сервері, буде створений користувач з таким реєстраційним ім'ям
--        і підключений до БД з роллю 'db_owner'. Усі користувачі, зареєстровані в копії але
-- відсутні в БД, що заміщається нею, видаляються.
.-- 5. Створення "нового" користувача на SQL Server вимагає певних привілеїв. Тому,
--        у цьому випадку, слід запускати даний скрипт під управлінням користувача,
--        який володіє ними, наприклад 'sa' – "системному адміністратору" таке доступне.
--        Без створення користувача досить прав 'dbo' БД.
=================================================================
if isnull(datalength(rtrim(@namDB)),0) = 0
begin select @crEX='Не визначена відновлювана БД – @namDB is NULL'
goto RestoreError
end
if isnull(datalength(rtrim(@restoreDB)),0) = 0
begin select @crEX='Не визначена копія – @restoreDB is NULL'
goto RestoreError
end
if isnull(datalength(rtrim(@pathDB)),0) = 0
begin select @crEX='Не визначені файли даних – @pathDB is NULL'
goto RestoreError
end
if isnull(datalength(rtrim(@pathLOG)),0) = 0
begin select @crEX='Не визначені файли журналу трансакцій – @pathLOG is NULL'
goto RestoreError
end
```

```
-- ==========< ПЕРЕВІРКА НА НАЯВНІСТЬ РЕЗЕРВНОЇ КОПІЇ > ==================
create table [#Yf] ([y] [int],[d] [int],[p] [int])
insert into [#Yf] exec [master].[dbo].[xp_fileexist] @restoreDB
if not exists(select '*' from [#Yf] where y = 1 and d = 0) -- Файл НЕ ЗНАЙДЕНИЙ
begin select @crEX='Файл '''+@restoreDB+''' відсутній!' drop table [#Yf]
goto RestoreError
end
drop table [#Yf]
-- ======== < ПЕРЕВІРКА НА ВІДПОВІДНІСТЬ ФОРМАТУ РЕЗЕРВНОЇ КОПІЇ >=========
select @crEX='restore verifyonly from disk='''+@restoreDB+''''
exec(@crEX)
if @@ERROR != 0
    begin select @crEX='Файл '''+@restoreDB+''' не є резервною копією!'
    goto RestoreError
end
-- ======== < ОТРИМАННЯ ІНФОРМАЦІЇ СТОСОВНО РЕЗЕРВНОЇ КОПІЇ >  ==========
create table [#HeaderBac] (BackupName sysname collate database_default NULL,
            BackupDescription varchar(100) collate database_default NULL,
            BackupType smallint NULL, ExpirationDate datetime NULL,
            Compressed smallint NULL, Position smallint NULL, DeviceType smallint NULL,
            UserName sysname collate database_default NULL,
            ServerName sysname collate database_default NULL,
            DatabaseName sysname collate database_default NULL, DatabaseVersion smallint NULL,
            DatabaseCreationDate datetime NULL, BackupSize bigint NULL,
            FirstLsn varchar(100) collate database_default NULL,
            LastLsn varchar(100) collate database_default NULL,
            CheckpointLsn varchar(100) collate database_default NULL,
            DifferentialBaseLsn varchar(100) collate database_default NULL,
            BackupStartDate datetime NULL, BackupFinishDate datetime NULL,
            SortOrder int NULL, CodePage int NULL, UnicodeLocaleId int NULL,
            UnicodeComparisonStyle int NULL, CompatibilityLeve int NULL,
            SoftwareVendorId int NULL, SoftwareVersionMajor int NULL,
            SoftwareVersionMinor int NULL, SoftwareVersionBuild int NULL,
            MachineName sysname collate database_default NULL, Flags int NULL,
            BindingId varchar(100) collate database_default NULL,
            RecoveryForkId varchar(100) collate database_default NULL,
            Collation varchar(100) collate database_default NULL)
select @crEX='restore headeronly from disk='''+@restoreDB+'''' insert into [#HeaderBac] ex-
ec(@crEX)
if @@ERROR != 0
    begin select @crEX='Помилка виконання – '+@crEX
    goto RestoreError
end
-- =====< ОТРИМАННЯ ІНФОРМАЦІЇ СТОСОВНО ФАЙЛІВ БД РЕЗЕРВНОЇ КОПІЇ >======
create table [#ListFile] (LogicalName sysname collate database_default NULL,
            PhysicalName varchar(300) collate database_default NULL,
            Type char(1) collate database_default NULL,
            FileGroupName sysname collate database_default NULL,
            Size bigint NULL, MaxSize bigint NULL)
select @crEX='restore filelistonly from disk='''+@restoreDB+''''
insert into [#ListFile]
exec(@crEX)
if @@ERROR != 0
   begin select @crEX='Помилка виконання – '+@crEX
   goto RestoreError
end
```

```
-- ========< ПЕРЕВІРКА НА НАЯВНІСТЬ ПРОЦЕСІВ, ПОВ'ЯЗАНИХ З БД > ============
use master
select @crEX=NULL
select @crEX=space(12)+'logname – '''+rtrim(loginame)+''', spid – '+convert(varchar(6),spid)+',
program_name – ''' +rtrim(program_name)+'''' collate database_default from master.dbo.sysprocesses
(nolock)
where dbid = db_id(@namDB) and spid != @@SPID
    if @@ERROR != 0
    begin
select @crEX='Помилка виконання – select logname, spid, program_name from
master.dbo.sysprocesses'+ ' where dbid = db_id('''+@namDB+''') and
spid != @@SPID'
goto RestoreError
    end
    if @crEX is not NULL -- Процеси знайдені
    begin
select @crEX='З БД '''+@namDB+''' взаємодіє процес:' + char(13) + char(10) + @crEX +
char(13) + char(10)+ space(12)+ подальша робота можлива тільки після його
завершення або знищення!'
        goto RestoreError
   end
-- ============== === < ВИЯВЛЕННЯ КОРИСТУВАЧІВ БД >=================
create table [#ListUs]
(uid int NOT NULL, namu sysname collate database_default NULL, sid varbinary(85) NULL,
ton tinyint NOT NULL, Primary Key(uid,ton))
if @@ERROR != 0
begin
select @crEX='Помилка створення – #ListUs'
goto RestoreError
end
if db_id(@namDB) is not NULL
begin
    select @crEX='use '+@namDB+' select uid,name,sid,0 from sysusers where status > 0'
    insert into [#ListUs]
    exec(@crEX)
    if @@ERROR != 0
    begin
select @crEX='Помилка виконання – '+@crEX
goto RestoreError
   end
   select @DB_CreOwn=loginname collate database_default from master.dbo.syslogins -- dbo БД
   where sid=(select sid from [#ListUs] where namu = 'dbo')
   if @@ERROR != 0
   begin
select @crEX='Помилка визначення dbo БД'
goto RestoreError
   end
end
if @DB_CreOwn is NULL and @namUS is NULL
    begin select @crEX='Не визначений користувач – @namUS is NULL'
    goto RestoreError
end
select @crEX='** 1 – перевірена копія '''+@restoreDB+''''
print '*************************************************************************'
print @crEX
print '*************************************************************************'
```

```
-- ======================= < ВИДАЛЕННЯ БД > ================================
if db_id(@namDB) is not NULL
begin
    select @crEX='drop database '+@namDB
    exec(@crEX)
    if @@ERROR != 0
    begin
    select @crEX='Не видаляється поточна версія БД '''+@namDB+''''
goto RestoreError
   end
       select @crEX='** 2 – видалена поточна БД '''+@namDB+''''
    print ' ********************************************************************************'
    print @crEX
    print ' ********************************************************************************'
end
-- ================== < СТВОРЕННЯ "ПОРОЖНЬОЇ" БД > ======================
if db_id(@namDB) is not NULL
    begin select @crEX='Не видалена поточна версія БД '''+@namDB+''''
    goto RestoreError
end
select @namDB_BC=DatabaseName from [#HeaderBac] -- Ім'я БД в копії
select @collate=Collation from [#HeaderBac]                    -- Колація
select @crEX='use master'+' create database '+@namDB+
' ON PRIMARY (name='''+@namDB+'_Data'', fileame='''+@pathDB+@namDB+'_Data.MDF'','+
        ' size=1MB, maxsize=UNLIMITED, filegrowth=10%)'+
        ' LOG ON (name='''+@namDB+'_Log'', filename='''+@pathLOG+@namDB+'_Log.LDF'','+
        ' size=1MB, maxsize=UNLIMITED, filegrowth=10%) collate '+
        isnull(@collate,'Cyrillic_General_CI_AS')
exec(@crEX)
if @@ERROR != 0
begin
select @crEX='Не створюється "порожня" БД '''+@namDB+''''
goto RestoreError
end
    select @crEX='** 3 – сворена БД '''+@namDB+''''
print '********************************************************************************'
print @crEX
print '********************************************************************************'
if db_id(@namDB) is NULL
begin
    select @crEX='Не створена "порожня" БД '''+@namDB+''''
    goto RestoreError
end
-- ======================= < ВІДНОВЛЕННЯ > ================================
select @bacData=LogicalName from [#ListFile] where Type = 'D' -- Файли: даних
select @bacLog= LogicalName from [#ListFile] where Type = 'L' --   журналу трансакцій
select @crEX='use master restore database '+@namDB+
        ' from disk='''+@restoreDB+''' with recovery,replace'+
        ',move '''+@bacData+''' to '''+@pathDB+@namDB+ '_Data.MDF'''+
        ',move '''+@bacLog+ ''' to '''+@pathLog+@namDB+'_Log.LDF'''
exec(@crEX)
if @@ERROR != 0
begin select
    @crEX='Помилка виконання – restore database '+@namDB
    goto RestoreError
end
    select @crEX='** 4 – відновлена БД '''+@namDB+''''
print '********************************************************************************'
```

```
print @crEX
print '*********************************************************************************'
if db_id(@namDB) is NULL
begin
    select @crEX='Не відновлена БД '''+@namDB+''''
    goto RestoreError
end
-- ================ < УСТАНОВКА ПАРАМЕТРІВ БД > ========================
if DataBaseProperty(@namDB,'IsRecursiveTriggersEnabled') = 0
begin
    exec sp_dboption @namDB,'recursive triggers','TRUE'
    if @@ERROR != 0
    begin
        select @crEX='Помилка виконання – sp_dboption '''+@namDB+''','''recursive trig-
        gers''','''TRUE'''
        goto RestoreError
    end
    print 'recursive triggers'
end
if DataBaseProperty(@namDB,'IsBulkCopy') = 0
begin
     exec sp_dboption @namDB,'select into/bulkcopy','TRUE'
     if @@ERROR != 0
     begin
         select @crEX='Помилка виконання – sp_dboption '''+@namDB+''','''select into/bulkcopy'', 'TRUE'''
         goto RestoreError
    end
    print 'select into/bulkcopy'
end
if DataBaseProperty(@namDB,'IsTruncLog') = 0
begin
    exec sp_dboption @namDB,'trunc. log on chkpt.','TRUE'
    if @@ERROR != 0
    begin
        select @crEX='Помилка виконання – sp_dboption '''+@namDB+''','''trunc. log on chkpt.''','''TRUE'''
        goto RestoreError
    end
     print 'trunc. log on chkpt.'
end
if DataBaseProperty(@namDB,'IsAutoClose') = 0
begin
     exec sp_dboption @namDB,'autoclose','TRUE'
     if @@ERROR != 0
     begin
         select @crEX='Помилка виконання – sp_dboption '''+@namDB+''','''autoclose''','''TRUE'''
         goto RestoreError
     end
     print 'autoclose'
end
if DataBaseProperty(@namDB,'IsAutoCreateStatistics') = 0
begin
    exec sp_dboption @namDB,'auto create statistics','TRUE'
    if @@ERROR != 0
    begin
        select @crEX='Помилка виконання – sp_dboption '''+@namDB+''','''auto create statistics''','''TRUE'''
        goto RestoreError
    end
    print 'auto create statistics'
end
```

```
if DataBaseProperty(@namDB,'IsAutoUpdateStatistics') = 0
begin
    exec sp_dboption @namDB,'auto update statistics','TRUE'
    if @@ERROR != 0
    begin
        select @crEX='Помилка виконання – sp_dboption '''+@namDB+''','''auto update statistics''','''TRUE'''
        goto RestoreError
    end
    print 'auto update statistics'
end
select @crEX='** 5 – встановлені параметри БД '''+@namDB+''''
print
'*******************************************************************************'
print @crEX
print
'*******************************************************************************'
-- ============== < ПЕРЕУСТАНОВКА КОРИСТУВАЧА > ======================
select @crEX='use '+@namDB+' select uid,name,sid,1 from sysusers where status > 0' --Користувачі
БД
insert into [#ListUs]
exec(@crEX)
if @@ERROR != 0
begin
    select @crEX='Помилка виконання – '+@crEX
    goto RestoreError
end
if @namUS is not NULL
begin
    if not exists (select '*' from syslogins where loginname = @namUS)
    begin
        select @Cr_US=1
        select @i=UnicodeLocaleId from [#HeaderBac] -- Ідентифікатор мови
        if exists (select '*' from syslanguages where msglangid = @i)
        select @namenLang=name collate database_default from syslanguages -- Назва мови
        where msglangid = @i
        exec sp_addlogin @loginame = @namUS, @defdb = @namDB, -- Логічне ім'я
                @deflanguage = @namenLang                    -- входу в SQL Server
        if not exists(select * from syslogins where loginname collate database_default = @namUS)
        begin
            select @crEX='Не створюється логічне ім'я для користувача SQL Server '''+@namUS+''''
            goto RestoreError
        end
    end
end
-- ..........
if DataBaseProperty(@namDB,'IsDboOnly') = 1
begin
    exec sp_dboption @namDB,'dbo use only','FALSE'
    if @@ERROR != 0
    begin
        select @crEX='Помилка виконання – sp_dboption '''+@namDB+''', ''dbo use only, FALSE'''
        goto RestoreError
    end
end
-- ..........
if @DB_CreOwn is NULL
begin
-- ..........
```

```
    if @Cr_US = 1
    begin
        exec sp_addsrvrolemember @namUS,'dbcreator' -- "Творець БД"
        if @@ERROR != 0
        begin select
             @crEX='Помилка виконання – sp_addsrvrolemember '''+@namUS+''','' dbcreator'''
             goto RestoreError
         end
         exec sp_addsrvrolemember @namUS,'bulkadmin' -- "Bulk insert"
         if @@ERROR != 0
         begin
             select @crEX='Помилка виконання – sp_addsrvrolemember '''+@namUS+''','' bulkadmin'''
             goto RestoreError
         end
         exec sp_adduser @namUS,@namUS,'public' -- 'master'
         select @crEX='Користувач '''+@namUS+''' підключений до БД ''master'' в ролі ''public'''
         print @crEX
         use msdb
         exec sp_adduser @namUS,@namUS,'db_owner' -- 'msdb'
         select @crEX='Користувач '''+@namUS+''' підключений до БД ''msdb'' в ролі ''db_owner'''
         print @crEX
         use tempdb
         exec sp_adduser @namUS,@namUS,'public' -- 'tempdb'
         select @crEX=Користувач '''+@namUS+''' підключений до БД ''tempdb'' в ролі ''public'''
         use master
         select @crEX='grant execute on xp_cmdshell to '+@namUS+
              ' grant execute on xp_readerrorlog to '+@namUS
         exec(@crEX)
         if @@ERROR != 0
         begin
              select @crEX='Помилка виконання – '+@crEX
              goto RestoreError
         end
         select @crEX=' користувачу '''+@namUS+''' надане право запуску: '+
              '''xp_cmdshell, xp_readerrorlog'''
         print @crEX
     end -- dbo
     select @crEX='use '+@namDB+' exec sp_changedbowner '''+ @namUS+''''
     exec(@crEX)
     if @@ERROR != 0
     begin
         select @crEX='Помилка виконання – '+@crEX
         goto RestoreError
     end
     select @crEX=' користувач '''+@namUS+''' підключений до БД '''+@namDB+''' в якості ''dbo'''
     print @crEX
end
-- ..........
else begin
-- ..........
     if not exists(select '*' from [#ListUs] o,[#ListUs] n
          where o.uid = n.uid and o.ton != n.ton and o.sid = n.sid and o.namu = 'dbo')
     begin                              -- dbo
         select @crEX='use '+@namDB+' exec sp_changedbowner '''+ @DB_CreOwn+''''
         exec(@crEX)
         if @@ERROR != 0
```

```
        begin
            select @crEX='Помилка виконання – '+@crEX
            goto RestoreError
        end
    select @crEX=' користувач '''+@DB_CreOwn+''' підключений до БД '''+@namDB+
                ''' в якості''dbo'''
    print @crEX
end
-- .....
if @namUS is not NULL and @namUS != @DB_CreOwn and
                not exists(select '*' from [#ListUs] o,#ListUs n
                        where o.uid = n.uid and o.ton != n.ton and o.sid = n.sid and n.namu = @namUS)
begin
    select @crEX='use '+@namDB+' declare @X varchar(2000) select @X='''''+
      ' select @X=@X+'' exec sp_dropuser ''''''+name+''''''''' from sysusers where name ='''+@namUS+''''+
          ' if @X is not NULL begin select @X=''use ''+db_name()+@X exec (@X) end'
    exec(@crEX)
    if @@ERROR != 0
    begin
      select @crEX='Помилка виконання – '+@crEX
      goto RestoreError
    end
    select @crEX='use '+@namDB+' exec sp_adduser
    '''+@namUS+''','''+@namUS+''','db_owner'''
    exec(@crEX)
    if @@ERROR != 0
    begin
        select @crEX='Помилка виконання – '+@crEX
        goto RestoreError
    end
        select @crEX=' користувач '''+@namUS+''' підключений до БД '''+@namDB+
                ''' в ролі ''db_owner'''
        print @crEX
    end
end
select @crEX='** 6 – забезпечена взаємодія користувача '''+isnull(@namUS,@DB_CreOwn)+
                ''' з БД '''+@namDB
print '*************************************************************************'
print @crEX
print
'************************************************************************* '
--==< ВИДАЛЕННЯ СТОРОННІХ КОРИСТУВАЧІВ І ВІДНОВЛЕННЯ ЗАРЕЄСТРОВАНИХ >==
delete [#ListUs] where namu = 'dbo' or namu = isnull(@namUS,@DB_CreOwn)
select @i=-1
while 1 = 1
begin
    select @namUs=NULL
    select top 1 @namUs=namu,@i=uid from [#ListUs] n
    where n.uid > @i and n.ton > 0 and not exists(select '*' from [#ListUs] o where o.ton = 0 and
                                o.namu = n.namu)
    if @namUs is NULL
    break
    select @crEX='use '+@namDB+' exec sp_dropuser '''+@namUs+''''
    exec(@crEX)
    if @@ERROR != 0
    begin
        select @crEX='Помилка виконання – '+@crEX
        goto RestoreError
    end
```

```
    select @crEX=' з БД '''+@namDB+''' видалений користувач '''+@namUS
    print @crEX
    delete [#ListUs] where uid = @i and ton > 0
end
-- ..........
select @i=-1
while 1 = 1
begin
    select @namUs=NULL
    select top 1 @namUs=namu,@i=uid from [#ListUs] o
    where o.uid > @i and o.ton = 0 and not exists(select '*' from #ListUs n
            where n.uid = o.uid and n.ton > 0 and n.sid = o.sid and n.namu = o.namu)
    if @namUs is NULL
        break
    if exists(select '*' from [#ListUs] where ton > 0 and namu = @namUs)
    begin
        select @crEX='use '+@namDB+' exec sp_dropuser '''+@namUs+''''
        exec(@crEX)
    if @@ERROR != 0
    begin
        select @crEX='Помилка виконання – '+@crEX
        goto RestoreError
    end
end
select @crEX='use '+@namDB+' exec sp_adduser '''+@namUS+''','''+@namUS+''','public''' ex-
ec(@crEX)
if @@ERROR != 0
begin
    select @crEX='Помилка виконання – '+@crEX
    goto RestoreError
end
select @crEX=' до БД '''+@namDB+''' підключений користувач '''+@namUS+''' в ролі ''public'''
print @crEX
delete [#ListUs] where uid = @i and ton = 0
end
select @crEX='** 7 – из БД'''+@namDB+''' видалені сторонні користувачі'
print '**********************************************************************************'
print @crEX
print '**********************************************************************************'
select @crEX='use '+@namDB+' declare @x int select @x=1 from sysobjects (nolock)'+
                ' where name = ''SETUP_BANK'' and xtype = ''U''' exec(@crEX)
select @e = @@ERROR, @i = @@ROWCOUNT
if @e != 0
begin
    select @crEX='Помилка виконання – '+@crEX
    goto RestoreError
end
if @i != 0
begin
    create table [#DB_AR] (NameDb varchar(40) collate database_default NULL,
        NameServer varchar(40) collate database_default NULL,
        RemoteUser varchar(40) collate database_default NULL)
    select @crEX='use '+@namDB+' select NameDB,NameServer,RemoteUser from [AdditDB] (nolock)'+
        ' where AliasDB = ''DB_AR'''
    insert into [#DB_AR]
    exec(@crEX)
    if @@ERROR != 0
```

```
  begin
     select @crEX='Помилка виконання – '+@crEX
     goto RestoreError
  end
  select top 1 @namDB_BC_AR=NameDb from [#DB_AR]
  select @namDB_BC_AR=case when @namDB_BC_AR is NULL then '.'
       when datalength(@namDB_BC_AR) = 0 or @namDB_BC_AR = '.' then '.'
       when @namDB_BC_AR = @namDB_BC then '.' else @namDB_BC_AR end
  if @namDB_BC_AR != '.'
  begin
     select @crEX='АРХІВНА БД '''+@namDB_BC_AR+''''
     print 'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA'
     print @crEX
     print 'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA'
  end
end
-- ..........................................................................................................................................
RestoreNormal: print ''
  print ''
  print '        @@@@@@@@@@@@@@@@@@@@@@@@@@@@'
  print '        @                              @'
  print '        @     ВІДНОВЛЕННЯ УСПІШНО ЗАВЕРШЕНЕ      @'
  print '        @                              @'
  print '        @@@@@@@@@@@@@@@@@@@@@@@@@@@@'
  print ''
select @crEX=' З копії БД '''+@namDB_BC+''' ('+ltrim(str(BackupSize/1048576.0,16,2))+'Мб.)'+
         case when datalength(isnull(ltrim(rtrim(BackupDescription)),'')) > 0 then ' /'+
         ltrim(rtrim(BackupDescription))+'/,' else ',' end+char(13)+char(10)+
         створеної на сервері'''+ServerName+''' '+convert(varchar(10),BackupFinishDate,104)+
         ' '+convert(varchar(10),BackupFinishDate,8)+
         ' користувачем '''+UserName+'''' from [#HeaderBac]
print @crEX
print '================================================================'
goto DropTab
-- NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
RestoreError: print ''
  print ''
  print '        @@@@@@@@@@@@@@@@@@@@@@@@@@@@@'
  print '        @                               @'
  print '        @       ВІДНОВЛЕННЯ АВАРІЙНО ЗАВЕРШЕНЕ       @'
  print '        @                               @'
  print '        @@@@@@@@@@@@@@@@@@@@@@@@@@@@@'
  print ''
if @crEX is not NULL
begin
  select @crEX=' ПРИЧИНА: '+@crEX
  print @crEX
end
print
'!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!'
DropTab: if Object_Id('tempdb..#HeaderBac') is not NULL drop table [#HeaderBac]
         if Object_Id('tempdb..#ListUs') is not NULL drop table [#ListUs]
         if Object_Id('tempdb..#ListFile') is not NULL drop table [#ListFile]
         if Object_Id('tempdb..#DB_AR') is not NULL drop table [#DB_AR]
return
-- EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
go
```

Додаток Б

Найбільш відомі експертні системи та сфери їх практичного застосування

Таблиця Б.1

Експертні системи в хімічній галузі

Назва	Призначення
CRYSALIS	Інтерпретація тривимірної структури білка
DENDRAL	Інтерпретація молекулярної структури
TQMSTUNE	Усунення порушень в роботі потрійного квадрупольного мас-спектрометра (підтримка в налагодженому стані)
CLONER	Проектування біологічних молекул нового типу
MOLGEN	Проектування експериментів по клонуванню генів
SECS	Проектування складних органічних молекул
SPEX	Планування експериментів у сфері молекулярної біології

Таблиця Б.2

Експертні системи в електроніці

Назва	Призначення
ACI	Діагностика причин збоїв в телефонних мережах
IN-ATE	Діагностика причин помилок осцилографів
NDS	Діагностика державної мережі зв'язку
EURISKO	Проектування тривимірних мікроелектронних приладів
PALLADIO	Проектування і перевірка нових інтегральних схем
REDESIGN	Перепроектування цифрових схем на підставі нових принципів
CADHELP	Підготовка інструктивних вказівок для учасників машинного проектування
SOPHIE	Підготовка інструктивних вказівок по діагностуванню причин збоїв в електронних мережах

Таблиця Б.3

Експертні системи в медицині

Назва	Призначення
PUFF	Діагностика легеневих захворювань
VM	Поточний контроль за станом пацієнтів в палатах інтенсивної терапії
ABEL	Діагностика характеристик кислотних та лужних електролітів
AI/COAG	Діагностика захворювань крові
AI/RHEUM	Діагностика ревматичних захворювань
CADUCEUS	Діагностика захворювань внутрішніх органів
ANNA	Поточний контроль за терапевтичним лікуванням із застосуванням препаратів наперстянки
BLUE BOX	Діагностика і лікування депресії
MYCIN	Діагностика і лікування захворювань, викликаних бактерійними інфекціями
ONCOCIN	Лікування та спостереження за пацієнтами, що проходять курс хіміотерапії
ATTENDING	Підготовка інструктивних вказівок по проведенню анестезії
GUIDON	Підготовка інструктивних вказівок по боротьбі з бактерійними інфекціями

Таблиця Б.4

Технічні експертні системи

Назва	Призначення
REACTOR	Діагностика та усунення збоїв в роботі реактора
DELTA	Діагностика та усунення несправностей в локомотивах General Electric
STEAMER	Підготовка інструктивних вказівок по експлуатації парової силової установки

Таблиця Б.5

Експертні системи в геології

Назва	Призначення
DIPMETER	Інтерпретація даних журналів роботи пластового наклономіра
LITHO	Інтерпретація даних журналів роботи нафтової свердловини
MUD	Діагностика проблем, що виникають в процесі буріння
PROSPECTOR	Інтерпретація геологічних даних з метою виявлення наявності корисної копалини

Таблиця Б.6

Комп'ютерні експертні системи

Назва	Призначення
PTRANS	Видача прогнозів стосовно успішного управління комп'ютерами DEC
XCON	Настройка конфігурації комп'ютерних систем DEC
BDS	Діагностика несправних компонентів в комутованій мережі передачі даних
XSEL	Підготовка замовлень на комп'ютери DEC
XSITE	Визначення вимог до платформи замовника, призначеної для установки комп'ютерів DEC
YES/MVS	Поточний контроль та управління операційною системою MVS компанії IBM
TIMM	Діагностика комп'ютерів DEC

Навчальне видання

Вахнюк Сергій Валерійович

ТЕХНОЛОГІЯ СТВОРЕННЯ ПРОГРАМНИХ ТА ІНТЕЛЕКТУАЛЬНИХ СИСТЕМ

Навчальний посібник

Редактор *Н.І. Козьменко*
Комп'ютерна верстка *Н.А. Височанська*

Підписано до друку 22.03.2011. Формат 60х90/16. Гарнітура Times.
Обл.-вид. арк. 13,0. Умов. друк. арк. 15,8. Тираж 300 пр. Зам. № 1016

Державний вищий навчальний заклад
“Українська академія банківської справи Національного банку України”
40000, м. Суми, вул. Петропавлівська, 57
Свідоцтво про внесення до Державного реєстру видавців, виготівників
і розповсюджувачів видавничої продукції: серія ДК № 3160 від 10.04.2008

Надруковано на обладнанні Державного вищого навчального закладу
“Українська академія банківської справи Національного банку України”
40000, м. Суми, вул. Петропавлівська, 57