МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

КОЗЛОВСЬКА В. П.

ОРГАНІЗАЦІЯ БАЗ ДАНИХ ТА ЗНАНЬ

Конспект лекцій

Одеса
Одеський державний екологічний університет
2019

УДК 681.3
К59

Рекомендовано методичною радою Одеського державного екологічного університету Міністерства освіти і науки України як конспект лекцій (протокол №3 від 31.10. 2019 р.)

Козловська В. П.
Організація баз даних та знань: конспект лекцій. Одеса, Одеський державний екологічний університет, 2019, 130с.

Основні ідеї сучасної інформаційної технології базуються на концепції баз даних (БД). Відповідно до цієї концепції, основою інформаційної технології є дані, які повинні бути рганізовані в БД із метою задоволення інформаційних потреб користувачів.

Метою дисципліни є формування у студентів теоретичних знань та практичних навичок, необхідних для роботи у будь-яких підрозділах, в яких використовують бази даних, а також здійснюють обслуговування та розробку інформаційних систем та систем баз даних.

В курсі розглядаються: призначення, організація, структура БД, моделі даних. Докладно розглядаються реляційні системи, що є найбільш розповсюдженими з 80-х років і досі. Розглядаються мова реляційних БД – SQL, та реляційна алгебра і реляційне вирахування, що є основою для мови SQL. Також розглядаються питання забезпечення нормальної роботи системи керування базами даних (СКБД): паралельна робота користувачів, захист БД, забезпечення цілісності БД.

Курс «Організація баз даних та знань» призначений для студентів III курсу денної та заочної форми навчання, що навчаються за спеціальністю 122 Компютерні науки, рівень підготовки – бакалавр.

ISBN 978-966-186-085-7

Зміст

ВСТУП

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

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

За останні 50 років в галузі теорії систем баз даних було проведено ряд продуктивних досліджень. Отримані результати цілком можна вважати найбільш важливим досягненням інформатики за цей період. Бази даних стали основою інформаційних систем і в корені змінили методи роботи багатьох організацій. Зокрема, в останні роки розвиток технології баз даних призвів до створення досить потужних і зручних в експлуатації систем. Завдяки цьому системи баз даних стали доступні широкому колу користувачів. Але, на жаль, уявна простота таких систем сприяла тому, що користувачі стали самостійно створювати бази даних і додатки до них, не маючи достатніх знань про методи проектування ефективно працюючих систем, що часто приводило до непродуктивних витрат ресурсів і неякісних результатів.

Тому метою даного є навчання студентів теоретичним і практичним основам теорії баз даних, зокрема продуктивної методології проектування баз даних. Проектування баз даних складається із трьох стадій: концептуальної, логічної й фізичної. Перша стадія передбачає створення концептуальної моделі даних, що не залежить від яких-небудь фізичних характеристик. У другій стадії, призначенням якої є створення логічної моделі даних, концептуальна модель піддається доробці за допомогою видалення елементів, які не можуть бути реалізовані в системах з обраною моделлю даних – реляційних системах у більшості випадків в цей час. Реляційна модель даних досить повно розглядається в даному курсі. У третій стадії проектування логічна модель даних перетвориться у фізичний проект, призначений для реалізації в середовищі конкретної цільової системи керування ба-

зами даних. Для придбання навичок роботи з конкретної СКБД у даному курсі розглядається робота з клієнт-серверною СУБД MS SQL Server.

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

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

1 ПРОЕКТУВАННЯ БАЗ ДАНИХ

1.1 Проблема проектування баз даних

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

В останні роки розвиваються CASE-засоби для автоматизації процесу проектування баз даних. Слід зазначити, що ці засоби, безумовно, прискорюють роботу проектувальника, але потребують від останнього досить глибоких знань по теорії баз даних, глибокого пророоблення вихідного матеріалу, системного аналізу предметної області, для якої проектується база даних.

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

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

Звичайно проектуванню ефективних баз даних заважає наступне:

- недостатньо глибокий аналіз вимог до даних (у тому числі семантики імен і взаємозв'язків даних);
- більша тривалість процесу структурування, що робить цей процес стомлюючим і важко здійсненним при ручній обробці;
- труднощі, пов'язані з обліком при проектуванні продуктивності системи;
- часові обмеження, обумовлені використовуваними технічними засобами.

1.2 Багаторівневе проектування баз даних

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

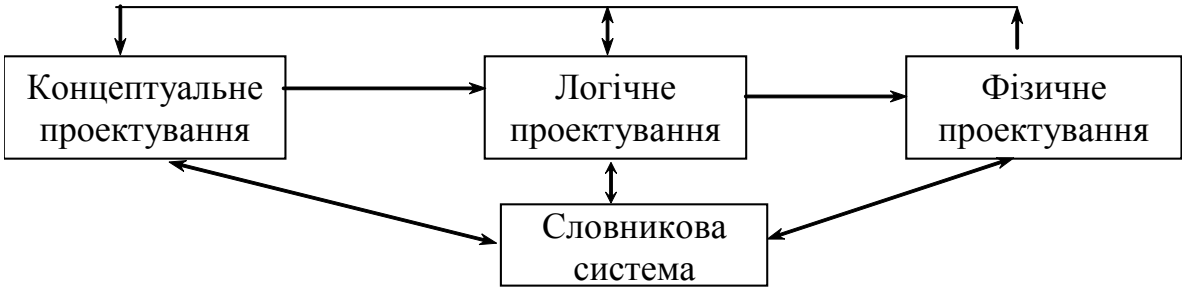


Рисунок 1.1 – Зв'язок між етапами проектування БД

На рис. 1.1 представлена сукупність процедур проектування бази даних. Процес включає три самостійних етапи.

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

Кожний етап проектування розглядається як сукупність ітеративних процедур, у результаті виконання яких одержують модель. Ми будемо говорити про створення концептуальної моделі на етапі концептуального проектування, логічної моделі на етапі логічного проектування й фізичної моделі на етапі фізичного проектування. Весь процес характеризується як ітеративний. Результати проектування, отримані на етапі логічного або фі-

зичного проектування, можуть потребувати зміни або повторення ітерацій для забезпечення необхідних властивостей системи, що проектується.

На рисунку 1.1 визначена також взаємодія між етапами проектування й словниковою системою. Однак розглянуті процедури проектування можуть використатися незалежно від словникової системи, якщо вона, наприклад, відсутня у проектувальника.

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

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

1.3 Загальні поняття. Етапи проектування.

Що таке проект? Це схема – ескіз деякого пристрою, який в подальшому буде втілений в реальність. Що таке проект реляційної бази даних? Це набір взаємозв'язаних відношень, в яких визначені всі атрибути, задані первинні ключі відношень і задані ще деякі додаткові властивості відношень, які відносяться до принципів підтримки цілісності. Чому саме взаємопов'язаних відношень? Тому що при виконанні запитів ми виробляємо об'єднання відношень, і одні і ті ж значення мають в різних відношеннях-таблицях позначатися однаково. Дійсно, якщо ми в одній таблиці оцінки будемо позначати цифрами, а в іншій – словами "відмінно", "добре", тощо, то ми не зможемо об'єднати ці таблиці по стовпчику ОЦІНКА, хоча за змістом це для нас один і той ж, але те, що інтуїтивно зрозуміло людині, зовсім не зрозуміло "розумному" комп'ютеру. Це проблема систем зі штучним інтелектом, які можуть вирішувати досить складні інтелектуальні завдання, важкі для рядового інженера, але іноді пасують перед найпростішими інтуїтивними асоціаціями, зрозумілими кожному школяреві. І це необхідно враховувати. Тому проект бази даних повинен бути дуже точний і вивірений. Фактично проект бази даних – це фундамент майбутнього програмного комплексу, який буде використовуватися досить довго і багатьма користувачами. І як в будь-якій будівлі, можна добудовувати мансарди,

переробляти дах, можна навіть міняти вікна, але замінити фундамент, не зруйнувавши всієї будівлі, неможливо. Етапи життєвого циклу бази даних зображені на рис. 1.2. Вони аналогічні, в основному, розвитку будь-якої програмної системи, проте в них є певна специфіка, що стосується тільки баз даних.

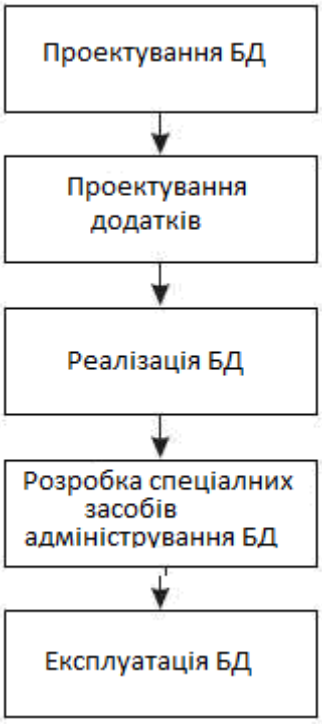


Рисунок 1.2 – Етапи життєвого циклу БД

Процес проектування БД являє собою послідовність переходів від неформального словесного опису інформаційної структури предметної області до формалізованого опису об'єктів предметної області в термінах деякої моделі. У загальному випадку можна виділити наступні етапи проектування:

1. Системний аналіз і словесний опис інформаційних об'єктів предметної області.

2. Проектування інфологічної моделі предметної області – частково формалізований опис об'єктів предметної області в термінах деякої семантичної моделі, наприклад, в термінах ER-моделі.

3. Даталогічне або логічне проектування БД, тобто опис БД в термінах прийнятої даталогічної моделі даних.

4. Фізичне проектування БД, тобто вибір ефективного розміщення БД на зовнішніх носіях для забезпечення найбільш ефективної роботи програми.

Якщо ми врахуємо, що між другим і третім етапами необхідно прийняти рішення, з використанням якої стандартної СУБД буде реалізовуватися наш проект, то умовно процес проектування БД можна уявити послідовністю виконання п'яти відповідних етапів (рис. 1.3). Розглянемо більш докладно етапи проектування БД.

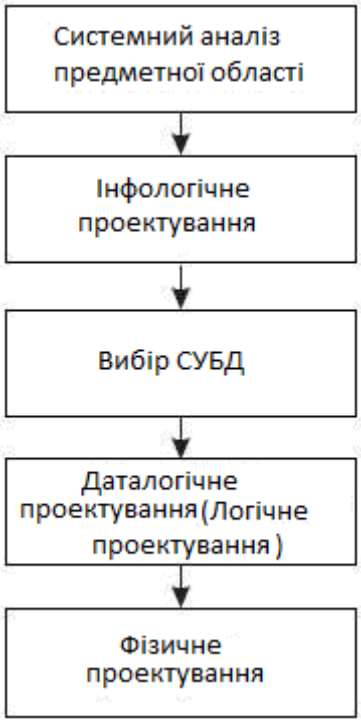


Рисунок 1.3 – Етапи проектування БД

1.4 Системний аналіз предметної області

З точки зору проектування БД в рамках системного аналізу, необхідно здійснити перший етап, тобто провести докладний словесний опис об'єктів предметної області і реальних зв'язків, які присутні між описаними об'єктами. Бажано, щоб даний опис дозволяв коректно визначити всі взаємозв'язки між об'єктами предметної області.

У загальному випадку існують два підходи до вибору складу і структури предметної області:

• Функціональний підхід – він реалізує принцип руху "від завдань" і застосовується тоді, коли заздалегідь відомі функції певної групи осіб і комплексів задач, для обслуговування інформаційних потреб яких створюється розглянута БД. У цьому випадку ми можемо чітко виділити мінімальний необхідний набір об'єктів предметної області, які повинні бути описані.

• Предметний підхід – коли інформаційні потреби майбутніх користувачів БД жорстко не фіксуються. Вони можуть бути багатоаспектний і вельми динамічними. Ми не можемо точно виділити мінімальний набір об'єктів предметної області, які необхідно описувати. В опис предметної області в цьому випадку включаються такі об'єкти і взаємозв'язки, які найбільш характерні і найбільш істотні для неї. БД, що конструюється при цьому, називається предметної, тобто вона може бути використана при вирішенні безлічі різноманітних, заздалегідь не визначених завдань. Конструювання предметної БД в деякому сенсі здається набагато більш привабливим, однак труднощі загального охоплення предметної області з неможливістю конкретизації потреб користувачів може привести до надмірно складною схемою БД, яка для конкретних завдань буде неефективною.

Найчастіше на практиці рекомендується використовувати певний компромісний варіант, який, з одного боку, орієнтований на конкретні завдання або функціональні потреби користувачів, а з іншого боку, враховує можливість нарощування нових додатків.

Системний аналіз повинен закінчуватися докладним описом інформації про об'єкти предметної області, яка потрібна для вирішення конкретних завдань, і яка повинна зберігатися в БД, формулюванням конкретних завдань, які будуть вирішуватися з використанням даної БД з коротким описом алгоритмів їх вирішення, описом вихідних документів, які повинні генеруватися в системі, описом вхідних документів, які служать підставою для заповнення даними БД.

Інфологічне проектування БД закінчується отриманням концептуальної моделі інформаційної системи, що не залежить від будь-яких фізичних аспектів подання інформації. Створена концептуальна модель є джерелом інформації для етапу логічного проектування бази даних .

Основні етапи концептуального проектування:

- інтеграція зовнішніх користувальницьких представлень;
- визначення типів сутностей;
- визначення типів зв'язків;
- визначення атрибутів і зв'язування їх з типами сутностей і зв'язків;
- визначення доменів атрибутів;
- визначення атрибутів, що є потенційними й первинними ключами;
- перевірка моделі на відсутність надмірності;

– перевірка відповідності локальної концептуальної моделі конкретним транзакціям користувачів;
– обговорення локальних концептуальних моделей даних з кінцевими користувачами.

1.4.1 Приклад опису предметної області

Нехай потрібно розробити інформаційну систему для автоматизації обліку отримання та видачі книг в бібліотеці. Система повинна передбачати режими ведення системного каталогу, що відображає перелік галузей знань, за якими є книги в бібліотеці. Усередині бібліотеки області знань в систематичному каталозі можуть мати унікальний внутрішній номер і повне найменування. Кожна книга може містити відомості з кількох областей знань. Кожна книга в бібліотеці може бути присутня в декількох примірниках. Кожна книга, що зберігається в бібліотеці, характеризується наступними параметрами:

• унікальний шифр;
• назва;
• прізвища авторів (можуть бути відсутніми);
• місце видання (місто);
• видавництво;
• рік видання;
• кількість сторінок;
• вартість книги;
• кількість примірників книги в бібліотеці.

Книги можуть мати однакові назви, але вони розрізняються по своєму унікальному шифру (ISBN).

У бібліотеці ведеться картотека читачів.

На кожного читача в картотеку заносяться такі відомості:

• прізвище ім'я по батькові;
• домашня адреса;
• телефон (будемо вважати, що у нас два телефони – робочий і домашній (або мобільний));
• дата народження.

Кожному читачеві присвоюється унікальний номер читацького квитка. Кожен читач може одночасно тримати на руках не більше 5 книг. Читач не повинен одночасно тримати більше одного примірника книги однієї на-

зви. Кожна книга в бібліотеці може бути присутнім в декількох примірниках.

Кожен екземпляр має наступні характеристики:

• унікальний інвентарний номер;

• шифр книги, який збігається з унікальним шифром з опису книг;

• місце розміщення в бібліотеці.

У разі видачі примірника книги читачеві в бібліотеці зберігається спеціальний вкладиш, в якому повинні бути записані такі відомості:

• номер квитка читача, який взяв книгу;

• дата видачі книги;

• дата повернення.

Передбачити наступні обмеження на інформацію в системі:

1. Книга може не мати жодного автора.
2. В бібліотеці повинні бути записані читачі не молодше 17 років.
3. У бібліотеці присутні книги, видані починаючи з 1960 по поточний рік.
4. Кожен читач може тримати на руках не більше 5 книг.
5. Кожен читач при реєстрації в бібліотеці повинен дати телефон для зв'язку: він може бути робочим або домашнім (мобільним).
6. Кожна галузь знань може містити посилання на безліч книг, але кожна книга може ставитися до різних областей знань.

З даної інформаційної системою повинні працювати такі групи користувачів:

• бібліотекарі;

• читачі;

• адміністрація бібліотеки.

При роботі з системою бібліотекар повинен мати можливість вирішувати такі завдання:

1. Приймати нові книги і реєструвати їх в бібліотеці.
2. Відносити книги до однієї або до кількох галузей знань.
3. Проводити каталогізацію книг, тобто призначення нових інвентарних номерів новоприйнятим книгам, і, поміщаючи їх на полиці бібліотеки, запам'ятовувати місце розміщення кожного примірника.
4. Проводити додаткову каталогізацію, якщо надійшло кілька примірників книги, яка вже є в бібліотеці, при цьому інформація про книгу в предметний каталог не вноситься, а кожному новому примірнику присво-

ється новий інвентарний номер і для нього визначається місце на полиці бібліотеки.

5. Проводити списання старих книг та книг, що не користуються попитом. Списувати можна тільки книги, жоден екземпляр яких не знаходиться у читачів. Списання проводиться за спеціальним актом списання, який затверджується адміністрацією бібліотеки.

6. Вести облік виданих книг читачам, при цьому передбачається два режими роботи: видача книг читачеві і прийом від нього книг, що повертаються їм до бібліотеки. При видачі книг фіксується, коли і який екземпляр книги був виданий даному читачеві, і до якого терміну читач повинен повернути цей екземпляр книги. При видачі книг наявність вільного екземпляра і його конкретний номер можуть визначатися по заданому унікальному шифру книги, або інвентарний номер може бути відомий заздалегідь. Не потрібно вести "історію" читання книг, тобто потрібно відображати тільки поточний стан бібліотеки. При прийомі книги, яка повертається читачем, перевіряється відповідність інвентарного номера книги, що повертається, виданому інвентарному номеру, і вона ставиться на своє старе місце на полицю бібліотеки.

7. Проводити списання загублених читачем книг за спеціальним актом списання або заміни, підписаним адміністрацією бібліотеки.

8. Проводити закриття абонемента читача, тобто знищення даних про нього, якщо читач хоче виписатися з бібліотеки і не є її боржником, тобто за ним не числиться ні однієї бібліотечної книги.

Читач повинен мати можливість вирішувати такі завдання:

1. Переглядати системний каталог, тобто перелік усіх галузей знань, книги по яким є в бібліотеці.

2. По обраній галузі знань отримати повний перелік книг, які числяться в бібліотеці.

3. Для обраної книги отримати інвентарний номер вільного примірника книги або повідомлення про те, що вільних примірників книги немає. У разі відсутності вільних примірників книги читач повинен мати можливість дізнатися дату найближчого передбачуваного повернення примірника даної книги. Читач не може дізнатися дані про те, у кого зараз екземпляри даної книги знаходяться на руках (з метою забезпечення особистої безпеки власників необхідної книги).

4. Для обраного учасника отримати список книг, які числяться в бібліотеці.

Адміністрація бібліотеки повинна мати можливість отримувати відомості про боржників-читачів бібліотеки, які не повернули вчасно взяті книги; відомості про книгах, які не є популярними, т. е. жоден екземпляр яких не знаходиться на руках у читачів; відомості про вартість конкретної книги, для того щоб встановити можливість відшкодування вартості втраченої книги або можливість заміни її іншою книгою; відомості про найбільш популярних книгах, тобто таких, всі екземпляри яких знаходяться на руках у читачів.

Цей зовсім невеликий приклад показує, що перед початком розробки необхідно мати точне уявлення про те, що ж має виконуватися в нашій системі, які користувачі в ній будуть працювати, які завдання буде вирішувати кожен користувач. На жаль, дуже часто по відношенню до баз даних вважається, що все можна визначити потім, коли проект системи вже створено. Відсутність чітких цілей створення БД може звести нанівець всі зусилля розробників, і проект БД вийде "поганим", незручним, що не відповідає ні реально модельованого об'єкту, ні завданням, які мають вирішуватися з використанням даної БД.

Розглянемо етап дата логічного (логічного) проектування. Нагадаємо, що етап даталогічного проектування відбувається вже після вибору конкретної моделі даних. І ми розглядаємо даталогічне проектування для реляційної моделі даних.

1.5 Даталогічне проектування

У реляційних БД дата логічне або логічне проектування призводить до розробки схеми БД, тобто сукупності схем відношень, які адекватно моделюють абстрактні об'єкти предметної області та семантичні зв'язки між цими об'єктами. Основою аналізу коректності схеми є так звані функціональні залежності між атрибутами БД. Деякі залежності між атрибутами відношень є небажаними через побічні ефекти і аномалії, які вони викликають при модифікації БД. При цьому під процесом модифікації БД ми розуміємо внесення нових даних в БД або видалення деяких даних з БД, а також оновлення значень деяких атрибутів.

Однак етап логічного або даталогічного проектування не закінчується проектуванням схеми відношень. У загальному випадку в результаті виконання цього етапу повинні бути отримані такі підсумкові документи:

- Опис концептуальної схеми БД в термінах обраної СУБД.
- Опис зовнішніх моделей в термінах обраної СУБД.
- Опис декларативних правил підтримки цілісності бази даних.

• Інформація про процедури підтримки семантичної цілісності бази даних.

Однак перед тим як описувати побудовану схему в термінах обраної СУБД, нам треба вибудувати цю схему. Ми повинні побудувати коректну схему БД, орієнтуючись на реляційну модель даних.

ВИЗНАЧЕННЯ: Коректної назвемо схему БД, в якій відсутні небажані залежності між атрибутами відношень.

Процес розробки коректної схеми реляційної БД називається логічним проектуванням БД.

Задачею логічного проектування бази даних є створення логічної моделі на основі обраної моделі даних, але без урахування конкретної СКБД що буде використовуватися, і інших фізичних аспектів реалізації інформаційної системи. На етапі логічного проектування отримана концептуальна модель уточнюється й перетворюється для відповідності структурам даних і зв'язкам між ними, властивих вибраної моделі даних: реляційної, об'єктно-орієнтованої, об'єктно-реляційної, і т.д.

Логічне проектування залежить від вибраної моделі даних. Для реляційної моделі можна виділити наступні етапи:

- створення й перевірка локальної логічної моделі на основі зовнішніх даних кожної групи користувачів;
- усунення особливостей локальної логічної моделі, несумісних з реляційною моделлю, наприклад, усунення зв'язків типу "багато-до-багатьох";
- визначення набору відношень, виходячи з структури локальної логічної моделі даних;
- перевірка відношень за допомогою правил нормалізації;
- перевірка відповідності відношень вимогам користувальницьких транзакцій;
- визначення вимог підтримки цілісності даних;
- створення й перевірка глобальної логічної моделі.

На останньому етапі фізичного проектування вибирається конкретна СКБД і на основі логічної моделі створюється фізична схема бази даних. Основними етапами фізичного проектування для реляційної СКБД є:

- перенос глобальної логічної моделі в середовище конкретної обраної СКБД;
- проектування базових відношень у середовищі обраної СКБД;
- проектування похідних відношень;

– реалізація обмежень цілісності предметної області;
– визначення індексів;
– розробка користувальницьких представлень;
– розробка збережених процедур та тригерів.

2 ПІДТРИМКА ЦІЛІСНОСТІ В РЕЛЯЦІЙНОЇ МОДЕЛІ ДАНИХ

Одним з основоположних понять в технології баз даних є поняття цілісності (див. розділ 3.4 [3]). У загальному випадку це поняття насамперед пов'язано з тим, що база даних відображає в інформаційному вигляді деякий об'єкт реального світу або сукупність взаємопов'язаних об'єктів реального світу. У реляційної моделі об'єкти реального світу представлені у вигляді сукупності взаємопов'язаних відношень. Під цілісністю будемо розуміти відповідність інформаційної моделі предметної області, що зберігається в базі даних, об'єктам реального світу і їх взаємозв'язкам в кожен момент часу. Будь-яка зміна в предметної області, значуща для побудованої моделі, має відображатися в базі даних, і при цьому повинна зберігатися однозначна інтерпретація інформаційної моделі в термінах предметної області.

Інфологічна модель предметної області «Бібліотека» представлена на рисунку 2.1:

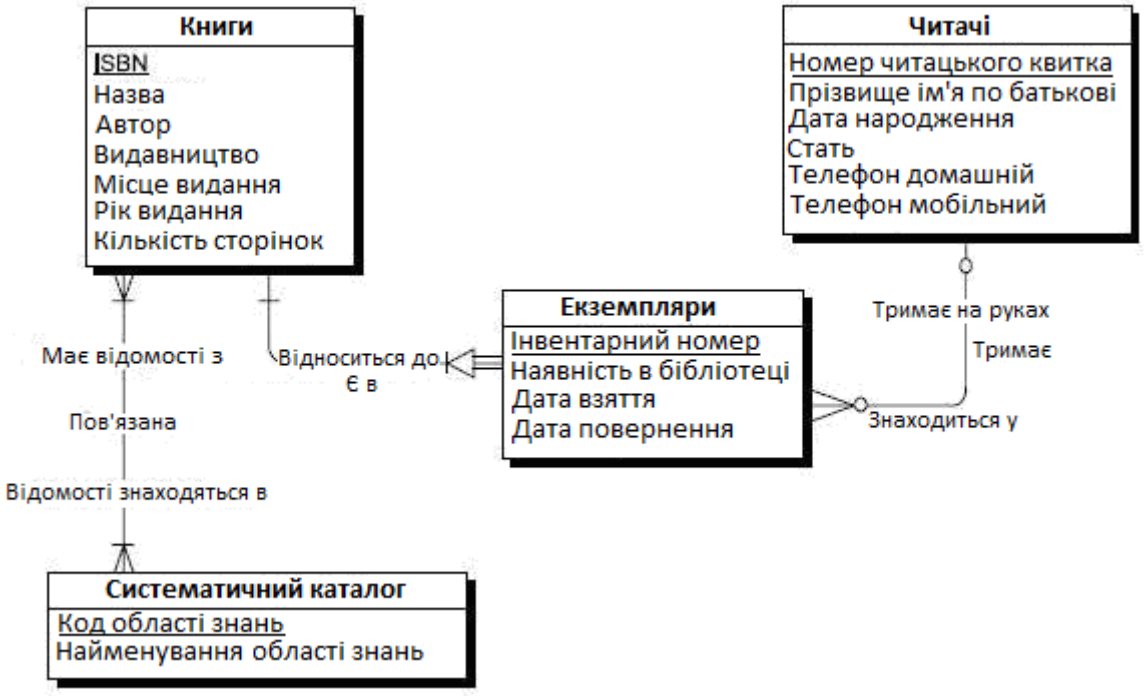


Рисунок 2.1 – Приклад інфологічної моделі предметної області

Інфологічна модель «Бібліотека», предметна область якої описана у попередньому розділі, розроблена під ті завдання, які були перераховані раніше. У цих завданнях не ставилася умова зберігання історії читання книги, наприклад, з метою пошуку того, хто раніше тримав книгу і міг

завдати їй шкоди, або забути в ній випадково велику суму грошей. Якби ми ставили перед собою завдання зберігання і цієї інформації, то наша інфологіческая модель була б іншою.

Дозвіл зв'язків типу "*багато-до-багатьох*". Так як в реляційної моделі даних підтримуються між відношенями тільки зв'язку типу "*один-до-багатьох*", а в ER-моделі (Entity-Relationship model, модель «сутність – зв'язок») допустимі зв'язки "*багато-до-багатьох*", то потрібен спеціальний механізм перетворення, який дозволить відобразити множинні зв'язки, неспецифічні для реляційної моделі, за допомогою допустимих для неї категорій. Це робиться введенням спеціального додаткового сполучного відношення, яке пов'язане з кожним вихідним зв'язком "один-ко-многим", атрибутами цього відношення є первинні ключі відношень, що зв'язуються. Так, наприклад, в схемі «Бібліотека» присутній зв'язок такого типу між сутністю «Книги» і «Системний каталог». Для вирішення цього неспецифічного зв'язку при переході до реляційної моделі має бути введено спеціальне додаткове відношення, яке має всього два атрибути: ISBN (шифр книги) і KOD (код галузі знань). При цьому кожен з атрибутів нового відношення є зовнішнім ключем (FOREIGN KEY), а разом вони утворюють первинний ключ (PRIMARY KEY) нової сполучною сутності. На рисунку 2.2 представлена реляційна (фізична) модель, що відповідає раніше інфологічної моделі «Бібліотека» (рис. 2.1).

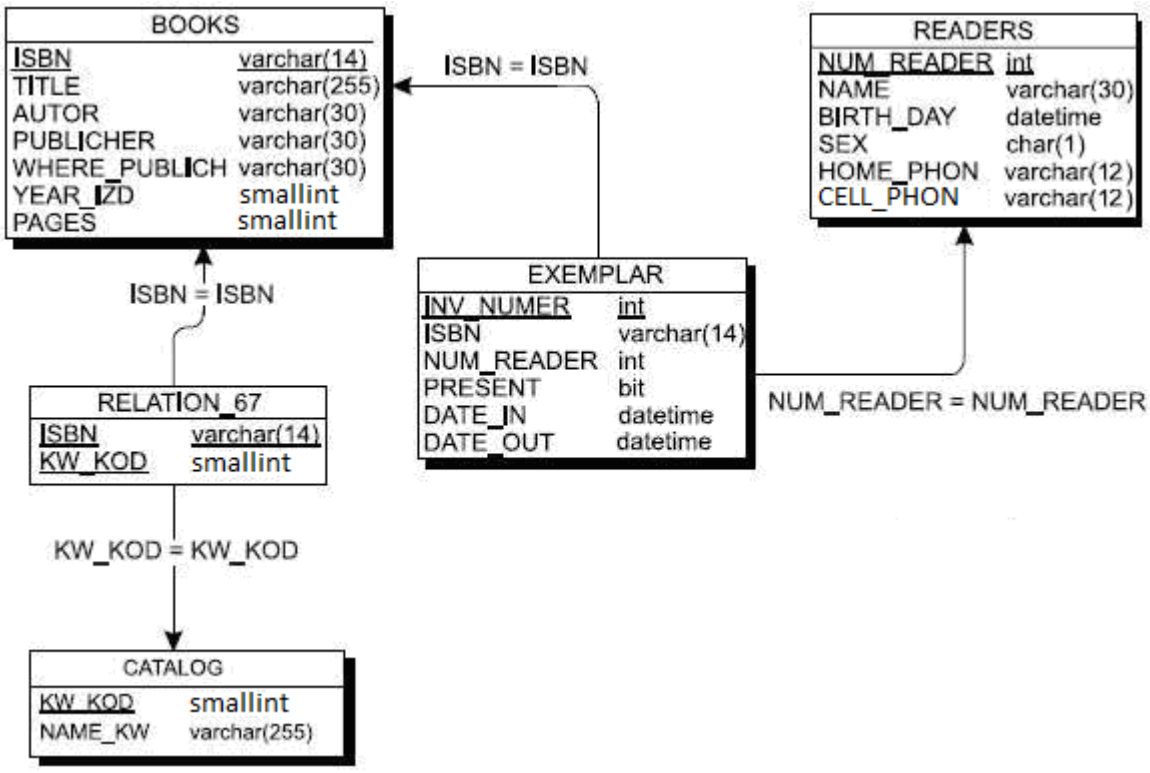


Рисунок 2.2 – Реляційна модель інформаційної системи «Бібліотека»

Було відзначено, що тільки істотні або значущі зміни предметної області повинні відслідковуватися в інформаційній моделі. Дійсно, модель завжди є деяке спрощення реального об'єкта, в моделі ми відображаємо лише те, що нам важливо для вирішення конкретного набору задач. Саме тому в інформаційній системі "Бібліотека" ми, наприклад, не відобразили місце зберігання конкретних екземплярів книг, тому що ми не ставили завдання автоматичної адресації бібліотечних стелажів. І в цьому випадку будь-яке переміщення книг з одного місця на інше не буде відображено в моделі, це переміщення несуттєво для наших завдань. З іншого боку, процес взяття книги читачем або повернення будь-якої книги в бібліотеку для нас важливий, і ми повинні його відстежувати відповідно до змін в реальній предметній області. І з цієї точки зору наявність у примірника книги покажчика на його відсутність в бібліотеці і одночасне відсутність запису про конкретний номері читацького квитка, за яким числиться цей екземпляр книги, є протиріччям, такого бути не повинно. І в моделі даних повинні бути передбачені засоби і методи, які дозволять нам забезпечувати динамічний відстеження в базі даних узгоджених дій, пов'язаних з узгодженим зміною інформації.

2.1 Загальні поняття та визначення цілісності

У першій частині конспекту лекцій [3] були розглянуті реляційні обмеження цілісності, а саме: цілісність сутностей, посилальна цілісність та корпоративні обмеження цілісності (див. розділ 3.4 [3]).

Цілісність сутностей вимагає, щоб у відношенні жоден атрибут первинного ключа не містив відсутні значення, які позначаються як NULL. Таким чином, атрибути, що складають первинний ключ відношення, при опису у команді створення таблиці повинні описуватись з властивістю NOT NULL. Не ключові атрибути, що можуть мати відсутні (порожні) значення, описуються з властивістю NULL. Наприклад, прізвище читача завжди повинне бути у кортежу відношення READERS, тому опис цього атрибуту у команді створення таблиці READERS має вигляд:

```
FIRST_NAME char(30) NOT NULL
```

Дата повернення читачем книги відсутня, доки ця книга знаходиться у читача, тому опис цього атрибуту у команді створення таблиці має вигляд:

```
DATE_OUT datetime NULL
```

Підтримка посилальної цілісності (Declarative Referential Integrity, DRI) означає забезпечення одного із заданих принципів взаємозв'язку між екземплярами кортежів взаємопов'язаних відношень:

– кортежі підлеглого відношення знищуються при видаленні кортежу основного відношення, пов'язаного з ними.
– кортежі основного відношення модифікуються при видаленні кортежу основного відношення, пов'язаного з ними, при цьому на місці ключа батьківського відношення ставиться невизначене Null значення.

Посилальна цілісність забезпечує підтримку несуперечливого стану БД в процесі модифікації даних при виконанні операцій додавання або видалення.

Крім зазначених обмежень цілісності, які в загальному вигляді не визначають семантику БД, вводиться поняття семантичної підтримки цілісності.

Структурна, мовна та довідкова цілісність визначають правила роботи СУБД з реляційними структурами даних. Вимоги підтримки цих трьох видів цілісності говорять про те, що кожна СУБД повинна вміти це робити, а розробники повинні це враховувати при побудові баз даних з використанням реляційної моделі. І ці вимоги підтримки цілісності досить абстрактні, вони визначають вимоги до форми подання та обробки інформації в реляційних базах даних. Але з іншого боку, ці аспекти ніяк не стосуються змісту бази даних. Для визначення деяких обмежень, які пов'язані з утриманням бази даних, потрібні інші методи. Саме ці методи і зведені в підтримку семантичної цілісності.

Давайте розглянемо конкретний приклад. Те, що ми можемо побудувати схему бази даних або її концептуальну модель тільки з сукупності нормалізованих таблиць, визначає структурну цілісність. І ми побудували нашу схему бібліотеки з п'яти взаємопов'язаних відношень. Але ми не можемо за допомогою перерахованих трьох методів підтримки цілісності забезпечити ряд правил, які визначені в нашій предметної області і повинні в ній дотримуватися. До таких правил можуть бути віднесені наступні:

1. В бібліотеці повинні бути записані читачі не молодше 17 років.

2. У бібліотеці присутні книги, видані починаючи з 1960 по поточний рік.

3. Кожен читач може тримати на руках не більше 5 книг.

4. Кожен читач при реєстрації в бібліотеці повинен дати телефон для зв'язку: він може бути мобільним або домашнім.

Принципи семантичної підтримки цілісності як раз і дозволяють забезпечити автоматичне виконання тих умов, які перераховані раніше.

Семантична підтримка може бути забезпечена двома шляхами: декларативним і процедурним шляхом. Декларативний шлях пов'язаний з наявністю механізмів в рамках СУБД, що забезпечують перевірку і виконання ряду декларативно заданих корпоративних обмежень цілісності, званих найчастіше "бізнес-правилами" (Business Rules) або декларативними обмеженнями цілісності.

Виділяються наступні види декларативних обмежень цілісності:

- обмеження цілісності атрибута;
- обмеження цілісності, що задаються на рівні відношень;
- обмеження цілісності, що задаються на рівні зв'язку між відношеннями.

Обмеження цілісності атрибута: значення за замовчуванням, завдання обов'язковості або необов'язковості значень (NULL), завдання умов на значення атрибутів. Завдання значення за замовчуванням означає, що кожен раз при введенні нового рядка в відношення, при відсутності даних в зазначеному стовпці цього атрибуту присвоюється саме значення за замовчуванням. Наприклад, при введенні нових книг розумно як значення за замовчуванням для року видання задати значення поточного року. Наприклад, для MS SQL Server це вираз матиме вигляд: YEAR (GETDATE ())

Тут GETDATE () – функція, яка повертає значення поточної дати, YEAR (date) – функція, яка повертає значення року зазначеної як параметр дати.

Як умова на значення для року видання треба задати вираз, яке буде істинним тільки тоді, коли рік видання буде лежати в межах від 1960 до поточного року. У конкретних СУБД це значення буде формуватися з використанням спеціальних вбудованих функцій СУБД.

Для MS SQL Server цей вираз буде виглядати наступним чином:

YEAR_PUBL Between 1960 ANDYEAR (GETDATE ())

Тут YEAR_PUBL – ім'я стовпця, що відповідає року видання.

Обмеження цілісності, що задаються на рівні доменів, за підтримки доменної структури. Ці обмеження зручні, якщо в базі даних присутні декілька стовпців різних відношень, які приймають значення з одного і того ж множини допустимих значень. Деякі СУБД підтримують подібну доменну структуру, тобто дозволяють визначати окремо домени, ставити тип даних для кожного домена і задавати відповідно обмеження у вигляді бізнес-правил для доменів. А для атрибутів задається не примітивний первинний тип даних, а їх належність того чи іншого домену. Іноді доменна структура

виражена неявно і в ряді СУБД застосовується спеціальна термінологія для цього. Так, наприклад, в MS SQL Server замість поняття домену вводиться поняття типу даних, визначеного користувачем, але сенс цього типу даних фактично еквівалентний змістом домену. У цьому випадку дійсно зручно задати обмеження на значення прямо на рівні домену, тоді воно автоматично буде виконуватися для всіх атрибутів, які приймають значення з цього домену.

Чому зручно задати це обмеження на рівні домену? А якщо поставити це обмеження для кожного атрибута, що входить в домен, хіба система буде працювати неправильно? Ні, звичайно, вона буде працювати правильно, але виникають труднощі, якщо в організації змінилися правила роботи, які виражені у вигляді декларативних обмежень на значення. У нашому випадку, наприклад, ми будемо комплектувати бібліотеку більш новими книгами і тепер будемо приймати в бібліотеку книги, видані не пізніше 1980 року. А якщо це обмеження у нас задано на декілька стовпців, то треба переглядати всі відношення і у всіх відношеннях міняти старе правило на нове. Легше замінити його один раз в домені, а всі атрибути, які приймають значення з цього домену, будуть автоматично працювати по новому правилу.

Це зробити дійсно легше, тим більше що в процесі роботи схема бази даних розростається і починає містити більше сотні відношень, і знайти всі відношення, в яких раніше встановлено це обмеження і виправити його – завдання нетривіальне.

Одним з основних правил при розробці проекту бази даних є мінімізація надмірності, а це означає, що якщо можливо інформацію про щось, в тому числі і про обмеження, зберігати в одному місці, то це треба робити обов'язково.

Обмеження цілісності, що задаються на рівні відношення. Деякі семантичні правила неможливо перетворити в виразі, який буде застосовний тільки до одного стовпцю. У прикладі з бібліотекою неможливо висловити вимогу наявності принаймні одного телефонного номера для швидкого зв'язку з читачем. Під телефони відведені два стовпці, це до певної міри штучно, але спеціально так зроблено, щоб показати інший тип обмежень. Кожен з атрибутів є в загальному випадку необов'язковим і може приймати невизначені значення: не обов'язково повинен бути заданий як мобільний, так і домашній телефон. Ми хочемо вимагати, щоб з двох принаймні один телефон був би заданий обов'язково. Спробуємо сформулювати це в термінології невизначених значень баз даних. Домашній телефон повинен бути заданий (NOT NULL) або мобільний телефон повинен бути заданий (NOT

NULL). Для MS SQL Server відповідний вираз буде виглядати наступним чином:

CELL_PHONE IS NOT NULL OR HOME_PHONE IS NOT NULL

Обмеження цілісності, що задаються на рівні зв'язку між відношеннями: завдання обов'язковості зв'язку, принципів каскадного видалення і каскадної зміни даних, завдання підтримки обмежень по потужності зв'язку. Ці види обмежень можуть бути виражені завданням обов'язковості або необов'язковості значень зовнішніх ключів у взаємозалежних відношеннях.

Декларативні обмеження цілісності відносяться до обмежень, які негайно перевіряються. Є обмеження цілісності, перевірка яких відкладається. Ці обмеження цілісності підтримуються механізмом транзакцій і тригерів.

2.1 Оператори DDL в мові SQL із завданням обмежень цілісності

Декларативні обмеження цілісності задаються на рівні операторів створення таблиць. У стандарті SQL оператор створення таблиць має наступний синтаксис:

```
CREATE TABLE
  [ database_name.[ owner ] . | owner. ] table_name
  ( { < column_definition >
     | column_name AS computed_column_expression
     | < table_constraint > ::= [ CONSTRAINT constraint_name ] }

         | [ { PRIMARY KEY | UNIQUE } [ ,...n ]
  )

[ ON { filegroup | DEFAULT } ]
[ TEXTIMAGE_ON { filegroup | DEFAULT } ]
```

Параметри команди:

database_name

Ім'я бази даних, у якій буде створена таблиця. Зазначена база даних повинна існувати. Якщо її ім'я не задається, то таблиця буде створена в поточній базі даних. Необхідно переконатися, що обліковий запис користувача має доступ до бази даних, у якій повинна бути створена таблиця, і що цей користувач має права на створення таблиць.

owner

Ім'я користувача бази даних, що буде вважатися власником створюваної таблиці. Власник таблиці може виконувати будь-які дії з нею, у т.ч. дозволяти доступ до цієї таблиці іншим користувачам. Якщо ім'я користувача не вказується, то власником таблиці є користувач, що створив її.

table_name

Ім'я, що буде привласнено таблиці. Для зберігання імені таблиці приділяється 256 байт, однак, тому що ім'я вказується у форматі Unicode, воно не повинне перевищувати 128 символів. Комбінація імені таблиці й імені її власника повинна бути унікальна в межах бази даних.

<*column_definition*>

Ця конструкція визначає властивості стовпця. Синтаксис цієї конструкції і її використання буде розглянуто далі.

column_name AS computed_column_expression

За допомогою цього аргументу можна створити стовпці, що *обчислюють* (computed). Значення таких стовпців обчислюються щораз заново при звертанні до них. Наприклад, якщо в таблиці є рядок, у якому існують стовпці із ціною товару (`price`) і його кількість (`count`), то стовпець із загальною ціною товару (`cost`) може бути обчислений автоматично. Для цього при створенні таблиці необхідно використати наступну конструкцію: `cost AS count*price`

У структурі таблиці зберігається тільки формула, по якій відбувається обчислення значень, тоді як самі значення не зберігаються. Тому неможлива вставка або зміна значень в полях, що обчислюють. При створенні стовпця, що обчислюється, необхідно вказати його ім'я (аргумент *column_name*) і після ключового слова AS вираз (аргумент *computed_column_expression*), по якому буде обчислюватися значення поля.

Крім того поля, що обчислюють, не дозволено змінювати, для них не можна встановлювати обмеження цілісності UNIQUE, PRIMARY KEY, FOREIGN KEY і DEFAULT.

<*table_constraint*>

За допомогою цієї конструкції визначаються обмеження цілісності рівня таблиці. Докладно синтаксис і застосування цієї конструкції будуть наведені далі.

[,...*n*]

Дана конструкція говорить про те, що через кому може бути зазначена множина визначень стовпців або обмежень цілісності.

Звичайні стовпці (не ті, що обчислюються) визначаються за допомогою конструкції <column_definition>, що має синтаксис:

```
< column_definition > ::= { column_name data_type }
  [ COLLATE < collation_name > ]    [ [ DEFAULT constant_expression ]
    | [ IDENTITY [ ( seed , increment ) [ NOT FOR REPLICATION ] ] ]    ]
  [ ROWGUIDCOL]
  [ < column_constraint > ] [ ...n ]
```

Розглянемо призначення й використання параметрів команди.

column_name

Ім'я, що буде мати стовпець. Ім'я повинне бути унікальним у межах таблиці. В імені стовпця допускається вказівка російських символів. Якщо в імені стовпця застосовуються заборонені символи, такі як пробіл, %, * і т.д., або ім'я стовпця збігається із зарезервованими словами, то ім'я стовпця при створенні повинне бути укладене у квадратні дужки.

data_type

Після імені стовпця через пробіл вказується тип даних, що будуть мати збережені в стовпці значення. Дозволяється застосування як стандартних типів даних SQL Server 2000, так і *користувальницьких типів даних* (UDDT, User Defined Data Type).

DEFAULT *constant_expression*

За допомогою цього аргументу можна визначити значення за замовчуванням, що буде привласнюватися відповідному полю рядка, якщо при її вставці користувач явно не вказав конкретне значення. Значення за замовчуванням не може бути застосоване до стовпців з типом даних timestamp або із установленою властивістю IDENTITY. Як значення за замовчуванням можуть застосовуватися константи, системні змінні й функції, а також будь-які вирази, побудовані на їхній основі. Використання посилань на інші стовпці заборонено.

IDENTITY [(*seed* **,** *increment*)

Зазначення цього аргументу пропонує створити стовпець із підтримкою автоматичної нумерації. При вставці нового рядка в таблицю SQL Server 2000 автоматично забезпечує вставку в поле IDENTITY унікального значення, що монотонно збільшується при вставці кожного нового рядка. Властивість IDENTITY може бути встановлено тільки для стовпців з типом даних int, smallint, tinyint, decimal (p, 0) і numeric (p, 0). У межах однієї таблиці можна створити тільки один стовпець із установленою властивістю IDENTITY.

<column_constraint>

Ця конструкція визначає обмеження цілісності на рівні стовпця. Синтаксис і використання цієї конструкції розглянуто далі.

[...*n*]

Дана конструкція говорить про те, що для одного стовпця може бути визначена декілька обмежень цілісності, які повинні бути перераховані через пробіл.

Як видно з синтаксису команди, для опису стовпця таблиці досить визначити його ім'я та тип даних. Інші аргументи є необов’язковими.

Обмеження цілісності на рівні стовпців

Як було сказано вище, обмеження цілісності для стовпця визначаються за допомогою конструкції <column_constraint>. Для одного стовпця може бути визначена множина обмежень цілісності або не визначено жодного. Конструкція <column_constraint> має наступний синтаксис:

```
< column_constraint > ::= [ CONSTRAINT constraint_name ]
  { [ NULL | NOT NULL ]
    | [ { PRIMARY KEY | UNIQUE }
       [ CLUSTERED | NONCLUSTERED ]
       [ WITH FILLFACTOR = fillfactor ]
       [ON {filegroup | DEFAULT} ] ]
    ]
    | [ [ FOREIGN KEY ]  REFERENCES ref_table [ ( ref_column ) ]
       [ ON DELETE { CASCADE | NO ACTION } ]
       [ ON UPDATE { CASCADE | NO ACTION } ]
       [ NOT FOR REPLICATION ]
    ]
    | CHECK [ NOT FOR REPLICATION ]
    ( logical_expression )
  }
```

Розглянемо призначення та використання основних параметрів:

CONSTRAINT *constraint_name*

Ключове слово CONSTRAINT вказує на те, що далі треба опис обмежень цілісності. За допомогою аргументу *constraint_name* вказується ім'я, що буде привласнено обмеженню цілісності. Ім'я обмеження цілісності повинне бути унікально в межах бази даних. Зазначення конструкції CONSTRAINT *constraint_name* не обов'язково.

NULL | NOT NULL

За допомогою цих опцій визначається, чи буде можливим зберігання в стовпці невизначених значень, чи ні. Для одного стовпця допускається застосування тільки одного з аргументів. При вказівці NULL зберігання невизначених значень дозволено, тоді як при вказівці NOT NULL забороняється. Якщо при створенні стовпця не було явно зазначене, буде він зберігати значення NULL чи ні, то для обмеження цілісності береться значення по умовчанню.

PRIMARY KEY

При створенні стовпця із цим параметром буде створений первинний ключ. У таблиці дозволяється створювати тільки один первинний ключ, у який може бути включене більше одного стовпця.

UNIQUE

Це ключове слово забезпечує створення для стовпця обмеження цілісності UNIQUE, що забезпечує унікальність, значень. Це обмеження цілісності може бути визначене на основі більш ніж одного стовпця.

Зауваження: Для того самого стовпця не можуть бути одночасно встановлені обмеження цілісності Primary Key і Unique.

FOREIGN KEY REFERENCES *ref_table* [(*ref_column*)]

За допомогою цієї конструкції визначається зовнішній ключ таблиці. Ключові слова FOREIGN KEY можуть не використовуватися. Після ключового слова REFERENCES вказується ім'я таблиці, зі стовпцем якої буде зв'язуватися обмеження цілісності FOREIGN KEY. Це обмеження цілісності зв'язується з одним стовпцем головної таблиці. Для стовпця головної таблиці, з яким зв'язується обмеження цілісності FOREIGN KEY, повинне бути встановлене обмеження цілісності PRIMARY KEY або UNIQUE.

Ім'я таблиці, з якої зв'язується зовнішній ключ, вказується за допомогою аргументу *ref_table*. За замовчуванням зовнішній ключ зв'язується з первинним ключем. Але можна зв'язати зовнішній ключ із будь-яким іншим стовпцем. Для цього за допомогою аргументу *ref_column* вказується ім'я потрібного стовпця.

ON DELETE {CASCADE | NO ACTION}

Пропонує використовувати для відповідного зовнішнього ключа обмеження цілісності CASCADE або NO ACTION на видалення рядків у головній таблиці. Таким чином, видалення рядків у головній таблиці буде приводити до видалення відповідних рядків (при вказівці CASCADE) у створюваній таблиці або скасуванні операції видалення рядка головної таблиці (при вказівці NO ACTION).

ON UPDATE {CASCADE | NO ACTION}

Пропонує використовувати для відповідного зовнішнього ключа обмеження цілісності CASCADE або NO ACTION на зміну рядків у головній таблиці. Таким чином, зміна первинного ключа в головній таблиці буде приводити до модифікації відповідних рядків (при вказівці CASCADE) у створюваній таблиці або скасуванні операції зміни рядка головної таблиці (при вказівці NO ACTION).

CHECK

Це обмеження, що забезпечує цілісність домену шляхом обмеження можливих значень, які можна вводити в стовпець або стовпці.

NOT FOR REPLICATION

Ключові слова, які використовуються для запобігання застосування обмеження CHECK під час процесу поширення, використовуваного реплікацією. Якщо таблиці є передплатниками публікації реплікації, що не оновлюйте таблицю підписки безпосередньо, а оновлюйте таблицю публікації і дозволяйте реплікації поширювати дані назад в таблицю підписки. Для таблиці підписки може бути задано обмеження CHECK, щоб користувачі не могли його змінити. Однак, якщо не додана умова NOT FOR REPLICATION, обмеження CHECK також не дозволяє процесу реплікації поширювати зміни з таблиці публікації в таблицю підписки. Пропозиція NOT FOR REPLICATION означає, що обмеження застосовується для призначених для користувача змін, але не для процесу реплікації.

Обмеження NOT FOR REPLICATION CHECK застосовується як до, так і після відображення оновленої записи, щоб запобігти додавання або видалення записів з діапазону реплікації. Все видалення і вставки перевіряються; якщо вони потрапляють в діапазон реплікації, вони відхиляються.

Коли це обмеження використовується зі стовпцем ідентифікаторів, SQL Server дозволяє таблиці не встановлювати заново значення стовпця ідентифікаторів, коли користувач реплікації оновлює стовпець ідентифікаторів.

logical_expression

Логічний вираз, яке повертає TRUE або FALSE.

При описі таблиці задається *ім'я таблиці*, яке є ідентифікатором в базовій мові СКБД і має відповідати вимогам іменування об'єктів в даній мові.

Крім імені таблиці в операторі вказується *список елементів* таблиці, кожен з яких служить або для визначення стовпчика, або для визначення обмеження цілісності таблиці, що визначається. Потрібна наявність хоча б одного визначення стовпця. Тобто таблицю, яка не має жодного стовпчика, визначити не можна. Кількість стовпців в одній таблиці не обмежена, але в конкретних СКБД зазвичай бувають обмеження на кількість атрибутів.

Оператор CREATE TABLE визначає так звану *базову таблицю*, тобто реальне *сховище даних*.

Як видно, крім обов'язкової частини, в якій задається ім'я стовпця і його *тип даних*, визначення стовпчика може містити два необов'язкових розділу: значення колонки *за замовчуванням* і розділ додаткових *обмежень цілісності* стовпчика.

У розділі значення за замовчуванням вказується *значення*, яке повинне записуватись в рядок, що заноситься в цю таблицю, якщо значення даного стовпця явно не вказано. Відповідно до стандарту мови SQL *значення за замовчуванням* може бути вказано у вигляді літеральної *константи* з типом, відповідним типу стовпчика.

Завдання в розділі обмежень цілісності стовпчика вираження NOT NULL призводить до неявному породження перевірочного обмеження цілісності для всієї таблиці "CHECK (C IS NOT NULL)" (де C – ім'я даного стовпчика). Якщо обмеження NOT NULL не вказано і розділ замовчувань відсутній, то неявно породжується розділ замовчувань DEFAULT NULL. Якщо вказана специфікація унікальності, то породжується відповідна специфікація унікальності для таблиці.

При завданні обмежень унікальності даний стовпець визначається як можливий ключ, що передбачає унікальність кожного значення, що вводиться в даний стовпець. І якщо це обмеження задано, то СКБД буде автоматично здійснювати перевірку на відсутність дублікатів значень даного стовпчика у всій таблиці.

Якщо в розділі обмежень цілісності вказано обмеження по посиланнях даного стовпця, то породжується відповідне визначення обмеження по посиланнях для таблиці: FOREIGN KEY (<ім'я стовпця>) <специфікація посилання>, що означає, що значення даного стовпчика повинні бути взяті з відповідного стовпчика батьківської таблиці. Батьківської таблицею в даному випадку називається таблиця, яка пов'язана з цією таблицею зв'язком "один-до-багатьох" (1: M). При цьому кожен рядок батьківської таблиці може бути пов'язана з кількома рядками таблиці, що визначається. Трансляція операторів SQL проводиться в режимі інтерпретації, тому важливо, щоб спочатку була описана батьківська таблиця, а потім вже все підлеглі (дочірні) таблиці, пов'язані з нею. Інакше транслятор визначить посилання на невизначений об'єкт.

Нарешті, якщо вказано *перевірочне обмеження* стовпця, то умова пошуку цього обмеження повинна посилатися тільки на даний стовпець, і неявно породжується відповідне перевірочне обмеження для всієї таблиці. У перевірочних обмеження, що накладаються на стовпець, не можна ставити порівняння зі значеннями інших стовпців цієї таблиці.

Спробуємо написати найпростіший оператор створення таблиці BOOKS з бази даних «Бібліотека».

При цьому будемо припускати наявність наступних обмежень цілісності:

- Шифр книги – послідовність символів довжиною не більше 14, однозначно визначає книгу, значить, це – фактично первинний ключ таблиці BOOKS.
- Назва книги – послідовність символів, не більше 120. Обов'язково має бути задано.
- Автор – послідовність символів, не більше 30, може бути не заданий.
- Співавтор – послідовність символів, не більше 30, може бути не заданий.
- Рік видання – ціле число, не менше 1960 і не більше поточного року. За замовчуванням ставиться поточний рік.
- Видавництво – послідовність символів, не більше 20, може бути відсутнім.
- Кількість сторінок – ціле число не менше 5 і не більше 1000.

```
CREATE TABLE BOOKS
(
 ISBN varchar(14) NOT NULL PRIMARY KEY,
 TITLE varchar(120) NOT NULL,
 AUTOR varchar(30) NULL,
 COAUTOR varchar(30) NULL,
 YEAR_PUBL smallint DEFAULT Year(GetDate()) CHECK(YEAR_PUBL between
1960 AND YEAR(GetDate())),
 PUBLICH varchar(20) NULL,
 PAGES smallint CHECK(PAGES > = 5 AND PAGES <=
      1000)
)
```

Для кількості сторінок непотрібно вказувати обмеження `NOT NULL`, тому що це є наслідком перевірочного обмеження, заданого на кількість сторінок. Кількість сторінок завжди має лежати в межах від 5 до 1000, значить, воно не може бути незаданим і система це контролює автоматично.

Тепер визначимо опис таблиці «Читачі», якій відповідає відношення READERS:

- Номер читацького квитка – це ціле число в межах 32 000 і він унікально визначає читача.
- Ім'я, прізвище читача – це послідовність символів, не більше 30.
- Адреса – це послідовність символів, не більше 50.
- Номери телефонів мобільного і домашнього – послідовність символів, не більше 12.

- Дата народження – календарна дата. У бібліотеку приймаються читачі не молодше 17 років.

```
CREATE TABLE READERS
(
 READER_ID Smallint PRIMARY KEY,
 FIRST_NAME char(30) NOT NULL,
 LAST_NAME char(30) NOT NULL,
 ADRES char(50),
 HOME_PHON char(12),
 CELL_PHON char(12),
 BIRTH_DAY datetime CHECK(DateDiff (year,BIRTH_DAY, GetDate()) >=17),
 CHECK ( HOME_PHON IS NOT NULL OR CELL_PHON IS NOT NULL)
)
```

Тут DateDiff (частина дати, початкова дата, кінцева дата) – функція MS SQL Server, яка визначає різницю між датами початку та закінчення, задану в одиницях, визначених першим параметром – частина дати. Ми задали у якості *частини дати* параметр year, тобто різниця визначається в роках.

Тепер задамо операцію створення таблиці EXEMPLAR (екземпляри книги). У цій таблиці первинним ключем є атрибут, що задає інвентарний номер примірника книги. У такій постановці ми вважаємо, що при надходженні книги в бібліотеку їй просто присвоюється відповідний порядковий номер. Для того щоб не обтяжувати бібліотекаря весь час пам'ятати, який номер був останнім, можна скористатися тим, що деякі СКБД допускають спеціальний інкрементний тип даних, тобто такий, значення якого автоматично збільшуються або зменшуються на задану величину при кожному новому введенні даних. У СКБД MS SQL Server це властивість IDENTITY, яке може бути присвоєно ряду цілочисельних типів даних. Властивість IDENTITY дозволяє рахувати з будь-яким кроком, позитивним або негативним, але обов'язково цілим. Якщо не задані додаткові параметри цій властивості, то вона починає працювати, починаючи з одиниці і додаючи при кожному введенні теж одиницю.

Крім того, таблиця EXEMPLAR є підпорядкованою двом іншим раніше визначеним таблицями: BOOKS і READERS. При цьому з таблицею BOOKS таблиця EXEMPLAR пов'язана обов'язковим зв'язком, тому що не може бути жодного примірника книги, який б не був приписаний до конкретної книжки. З таблицею READERS таблиця EXEMPLAR пов'язана необов'язковою зв'язком, тому що не кожен екземпляр в даний момент знаходиться на руках у читача. Для моделювання цих зв'язків при створенні таблиці EXEMPLAR повинні бути визначені два зовнішніх ключа (FOREIGN KEY). При цьому атрибут, відповідний шифру книги (назвемо

так само, як і в батьківській таблиці – ISBN), є обов'язковим, тобто не може приймати невизначених значень, а атрибут, який є зовнішнім ключем для зв'язку з таблицею READERS, є необов'язковим і може приймати невизначені значення.

Необов'язковими також є два інших атрибута: дата взяття і дата повернення книги, обидва вони мають тип даних, що відповідає календарній даті. У СКБД MS SQL Server для визначення атрибутів типу календарна дата є два типи: datetime та smalldatetime, – що містять упаковані дату та час. Атрибут, який містить інформацію про присутність або відсутність книги, має логічний тип. Напишемо оператор створення таблиці EXEMPLAR в синтаксисі MS SQL Server:

```
CREATE TABLE EXEMPLAR
(
 INV_NUMBER INT IDENTITY PRIMARY KEY,
 ISBN varchar(14) FOREIGN KEY references BOOKS(ISBN),
 READER_ID Smallint NULL FOREIGN KEY references READERS (READER_ID),
 DATA_IN datetime NULL,
 DATA_OUT datetime NULL,
 EXIST bit
 )
```

Не всі декларативні обмеження можуть бути задані на рівні стовпців таблиці, частина обмежень може бути задана тільки на рівні всієї таблиці. Наприклад, якщо ми маємо в якості первинного ключа не один атрибут, а послідовність атрибутів, то ми можемо визначити обмеження типу PRIMARY KEY (первинний ключ) тільки на рівні всієї таблиці.

Припустимо, що ми рахуємо екземпляри книги не підряд, а окремо для кожного видання, тоді таблиця EXEMPLAR в якості первинного ключа матиме набір з двох атрибутів: це шифр книги (ISBN) і порядковий номер примірника даної книги (EXEMPL_ID), в цьому випадку оператор створення таблиці EXEMPLAR буде виглядати наступним чином:

```
CREATE TABLE EXEMPLAR
(
 EXEMPL_ID int NOT NULL,
 ISBN varchar(14) FOREIGN KEY references BOOKS(ISBN),
 READER_ID Smallint NULL FOREIGN KEY references READERS (READER_ID),
 DATA_IN datetime NULL,
 DATA_OUT datetime NULL,
 EXIST bit,
 PRIMARY KEY(EXEMPL_ID, ISBN)
 )
```

Атрибут ISBN, з одного боку, є зовнішнім ключем (FOREIGN KEY) таблиці BOOKS, а з іншого боку, є частиною первинного ключа

(PRIMARY KEY) таблиці EXEMPLAR. І обмеження типу первинний ключ (PRIMARY KEY) таблиці EXEMPLAR задається не на рівні одного атрибута, а на рівні всієї таблиці, тому що воно містить набір атрибутів.

Те ж саме можна сказати і про перевірочні (CHECK) обмеження, якщо умови перевірки припускають порівняння значень декількох стовпців таблиці. Введемо додаткове обмеження для таблиці BOOKS, яке може бути сформульовано таким чином: співавтор не може бути заданий, якщо не заданий автор. При описі книги допустимо не ставити ні автора, ні співавтора, або задати і автора, і співавтора, або задати тільки автора. Однак завдання співавтора за відсутності завдання автора вважається помилковим. У цьому випадку оператор створення таблиці BOOKS буде виглядати наступним чином:

```
CREATE TABLE BOOKS
(
ISBN varchar(14) NOT NULL PRIMARY KEY,
TITLE varchar(120) NOT NULL,
AUTOR varchar(30) NULL,
COAUTOR varchar(30) NULL,
YEAR_PUBL smallint DEFAULT Year(GetDate()) CHECK(YEAR_PUBL >= 1960 AND
 YEAR_PUBL <= YEAR(GetDate())),
PUBLICH varchar(20) NULL,
PAGES smallint CHECK(PAGES > = 5 AND PAGES <= 1000),
CHECK (NOT (AUTOR IS NULL AND COAUTOR IS NOT NULL))
)
```

Для аналізу помилок доцільно іменувати все обмеження, особливо якщо таблиця містить кілька обмежень одного типу. Для іменування обмежень використовується ключове слово CONSTRAINT, після якого йде унікальне ім'я обмеження, потім тип обмеження і його вираження. Для ідентифікації обмежень рекомендують використовувати систему іменування, яка легко дозволить визначити при отриманні повідомлення про помилку, яке виробляє СУБД, яке саме обмеження порушено. Зазвичай ім'я обмеження складається з короткого назви типу обмеження, далі через символ підкреслення йде ім'я атрибута або таблиці, в залежності від того, до якого рівня відноситься обмеження, і, нарешті, порядковий номер обмеження даного типу, якщо до одного об'єкту задається кілька обмежень одного типу .

Скорочені позначення обмежень складаються з однієї або двох букв і можуть бути наступними:

- PK - для первинного ключа;
- FK - для зовнішнього ключа;
- CK - для перевірочного обмеження;

• U - для обмеження унікальності;

• DF - для обмеження типу значення за замовчуванням.

Наведемо приклад оператора створення таблиці BOOKS з іменованими обмеженнями:

```
CREATE TABLE BOOKS
(
 ISBN varchar(14) NOT NULL ,
 TITLE varchar(120) NOT NULL,
 AUTOR varchar (30) NULL,
 COAUTOR varchar(30) NULL,
 YEAR_PUBL smallint NOT NULL DEFAULT (YEAR(GETDATE())),
 PUBLICH varchar(20) NULL,
 PAGES smallint NOT NULL,
 CONSTRAINT PK_BOOKS PRIMARY KEY (ISBN),
 CONSTRAINT CK_YEAR_PUBL CHECK (YEAR_PUBL between 1960 AND
    YEAR(GetDate())),
 CONSTRAINT CK_PAGES CHECK (PAGES between 5 AND 1000),
 CONSTRAINT CK_BOOKS CHECK (NOT ((AUTOR IS NULL) AND
    (COAUTOR IS NOT NULL)))
)
```

Оператори мови SQL, як вказувалося раніше, транслюються в режимі *інтерпретації*, на відміну від більшості алгоритмічних мов, транслятори для яких виконані за принципом *компіляції*. У режимі *інтерпретації* кожен оператор окремо транслюється, тобто переводиться в машинні коди, і тут же виконується. У режимі *компіляції* вся програма, тобто сукупність операторів, спочатку перекладається в машинні коди, а потім може бути виконана як єдине ціле. Така особливість SQL накладає обмеження на порядок опису створюваних таблиць. Дійсно, якщо при трансляції оператора опису підпорядкованої таблиці з зазначеним зовнішнім ключем і відповідним посиланням на батьківську таблицю ця батьківська таблиці не буде виявлена, то ми отримаємо повідомлення про помилку з зазначенням посилання на неіснуючий об'єкт. Спочатку повинні бути описані всі основні таблиці, а потім підлеглі таблиці.

У нашому прикладі з бібліотекою порядок опису таблиць наступний:

1. Таблиця BOOKS

2. Таблиця READERS

3. Таблиця CATALOG (системний каталог)

4. Таблиця EXEMPLAR

5. Таблиця RELATION_1 (додаткова таблиця, що встановлює зв'язок між книгами і системним каталогом).

Набір операторів мови SQL прийнято називати не програмою, а скриптом. Тоді скрипт, який додасть набір з 5 взаємопов'язаних таблиць бази даних "Бібліотека" в існуючу базу даних, буде виглядати наступним чином:

```
CREATE TABLE BOOKS  (
 ISBN varchar(14) NOT NULL ,
 TITLE varchar(120) NOT NULL,  AUTOR varchar (30) NULL,
 COAUTOR varchar(30) NULL,
 YEAR_PUBL smallint NOT NULL DEFAULT (YEAR(GETDATE())),
 PUBLICH varchar(20) NULL,
 PAGES smallint NOT NULL,
 CONSTRAINT PK_BOOKS PRIMARY KEY (ISBN),
 CONSTRAINT CK_YEAR_PUBL CHECK (YEAR_PUBL between 1960 AND YEAR(GetDate())),
 CONSTRAINT CK_PAGES CHECK (PAGES between 5 AND 1000),
 CONSTRAINT CK_BOOKS CHECK (NOT ((AUTOR IS NULL) AND (COAUTOR IS NOT NULL)))
)

CREATE TABLE READERS  (
 READER_ID Smallint PRIMARY KEY,
 FIRST_NAME char(30) NOT NULL,
 LAST_NAME char(30) NOT NULL,
 ADRES char(50),
 HOME_PHONE char(12),
 CELL_PHONE char(12),
 BIRTH_DAY date CHECK( DateDiff(year,BIRTH_DAY, GetDate()) >=17 ),
 CONSTRAINT CK_READERS CHECK (HOME_PHONE IS NOT NULL OR CELL_PHONE IS NOT NULL)
)

CREATE TABLE CATALOG (
 ID_CATALOG Smallint PRIMARY KEY,
 KNOWELEDGE_AREA varchar(150)
)

CREATE TABLE EXEMPLAR  (
 EXEMPL_ID int NOT NULL,
 ISBN varchar(14) NOT NULL FOREIGN KEY references BOOKS(ISBN),
 READER_ID Smallint NULL FOREIGN KEY references READERS (READER_ID),
 DATA_IN date,
 DATA_OUT date,
 EXIST bit,
 PRIMARY KEY (EXEMPL_ID, ISBN)
)

CREATE TABLE RELATION_1 (
 ISBN varchar(14) NOT NULL FOREIGN KEY references BOOKS(ISBN),
 ID_CATALOG smallint NOT NULL FOREIGN KEY references CATALOG(ID_CATALOG),
 CONSTRAINT PK_RELATION_1 PRIMARY KEY (ISBN,ID_CATALOG)
)
```

При написанні скрипта в оператор створення таблиці «Читачі» додане обмеження на рівні таблиці, яке пов'язане з обов'язковою наявністю хоча б одного з двох телефонів.

3 СХЕМИ ТАБЛИЦЬ. ПРЕДСТАВЛЕННЯ

3.1 Засоби зміни опису таблиць і засоби видалення таблиць

У стандарті SQL2 є досить широкі можливості по модифікації вже існуючих схем таблиць. Для модифікації таблиць використовується оператор ALTER TABLE, який дозволяє виконати наступні операції зміни для схеми таблиці:

- додати новий стовпець в уже існуючу і заповнену таблицю;
- змінити значення за замовчуванням для будь-якого стовпця;
- видалити стовпець з існуючої таблиці;
- додати або видалити первинний ключ таблиці;
- додати новий або видалити зовнішній ключ таблиці;
- додати або видалити умову унікальності;
- додати або видалити умову перевірки для будь-якого стовпця або для таблиці в цілому.

Синтаксис оператора ALTER TABLE для модифікації обного стовпця таблиці:

```
ALTER TABLE <и'мя таблиці>
 { ADD <визначення стовпця>
 ALTER <и'мя стовпця > {SET DEFAULT <значення>
 DROP DEFAULT } |
 DROP < и'мя стовпця > {CASCADE | RESTRICT} |
 ADD { <визначення первинного ключа>|
 <визначення зовнішнього ключа> |
 <умова унікальности даних> |
 <умова перевірки> } |
 DROP CONSTRAINT и'мя умови { CASCADE |
 RESTRICT} }
```

Одним оператором ALTER TABLE можна провести лише одне з перекладних вимірів, наприклад, за один раз можна додати один стілець. Якщо вам потрібно додати два стовпчика, то потрібно замінити два оператора. Не можна змінити ім'я атрибута оператором ALTER TABLE

У СКБД MS SQL Server повний синтаксис оператора ALTER TABLE має вид:

```
ALTER TABLE table
{ [ ALTER COLUMN column_name
   { new_data_type [ ( precision [ , scale ] ) ]
      [ COLLATE < collation_name > ]    [ NULL | NOT NULL ]
      | {ADD | DROP } ROWGUIDCOL }
```

```
   ]
   | ADD
      { [ < column_definition > ]
      | column_name AS computed_column_expression    } [ ,...n ]
   | [ WITH CHECK | WITH NOCHECK ] ADD    { < table_constraint > } [ ,...n ]
   | DROP  { [ CONSTRAINT ] constraint_name
          | COLUMN column } [ ,...n ]
   | { CHECK | NOCHECK } CONSTRAINT
      { ALL | constraint_name [ ,...n ] }
   | { ENABLE | DISABLE } TRIGGER      { ALL | trigger_name [ ,...n ] }
}

< table_constraint > ::=
   [ CONSTRAINT constraint_name ]
   { [ { PRIMARY KEY | UNIQUE }
      [ CLUSTERED | NONCLUSTERED ]
      { ( column [ ,...n ] ) }
      [ WITH FILLFACTOR = fillfactor ]
      [ ON {filegroup | DEFAULT } ]
      ]
      |   FOREIGN KEY
         [ ( column [ ,...n ] ) ]
         REFERENCES ref_table [ ( ref_column [ ,...n ] ) ]
         [ ON DELETE { CASCADE | NO ACTION } ]
         [ ON UPDATE { CASCADE | NO ACTION } ]
         [ NOT FOR REPLICATION ]
      | DEFAULT constant_expression
         [ FOR column ] [ WITH VALUES ]
      |   CHECK [ NOT FOR REPLICATION ]
         ( search_conditions )
   }
```

Конструкція < column_definition > розглянута у розділі у попередньому розділі (стор. 26 – 30).

Опція ADD < table_constraint > дозволяє додати у таблицю нове обмеження цілісності на рівні таблиці.

Деякі аргументи команди ALTER TABLE:

table

Ім’я таблиці, що буде змінена.

ALTER COLUMN

Вказує, що даний стовбець повинен бути зміненим.

new_data_type

Новий тип даних для зміненого стовпця. Критеріями *new_data_type* зміненого стовпця є:

– існуючий тип даних повинен неявно перетворюватись у новий тип;

– *new_data_type* не може бути міткою часу даних (**timestamp**) .

Розглянемо кілька прикладів. Найчастіше застосовується операція додавання стовпця. Пропозиція визначення нового стовпця в операторі ALTER TABLE має точно такий же синтаксис, як і в операторі створення таблиці. Додамо стовпець EDUCATION (освіта), що містить символьний тип даних, з заданим переліком можливих значень ("початкова", "середня", "бакалавр", "магистр").

```
ALTER TABLE READERS
ADD EDUCATION varchar (30) DEFAULT NULL
CHECK (EDUCATION IS NULL OR
 EDUCATION= "начальное" OR
 EDUCATION= "среднее" OR EDUCATION= "бакалавр" OR
 EDUCATION= "магистр")
```

У таблицю READERS буде додано стовпець EDUCATION, в який за замовчуванням будуть додані всі кортежі невизначеного значення. Надалі ці значення можуть бути замінені на одне з допустимих символьних значень.

Додамо обмеження на відповідність між датами взяття і повернення книги в таблиці EXEMPLAR. Дійсно, якщо дати введені, то потрібно, щоб дата повернення книги була б більше на термін видачі книги. Вважаємо, що стандартним строком є 2 тижні. Тепер сформулюємо оператор зміни таблиці EXEMPLARE:

```
ALTER TABLE EXEMPLAR
ADD CONSTRAINT CK_EXEMPLAR CHECK ((DATA_IN IS NULL AND DATA_OUT IS
NULL) OR dateDiff(day, DATA_IN,DATA_OUT)<=14
/*Між датами 14 днів*/
)
```

Операція видалення стовпця пов'язана з перевіркою посилальної цілісності, і тому не дозволяється видаляти стовпці, пов'язані з підтримкою посилальної цілісності таблиці, тобто не можна видалити стовпці батьківської таблиці, що входять в первинний ключ таблиці, якщо на них є посилання в підлеглих таблицях.

При зміні первинного ключа таблиці слід бути уважними. По-перше, у вихідній таблиці можуть бути підлеглі, при цьому первинний ключ вихідної таблиці є зовнішнім ключем для підлеглих таблиць, і просто його видалити неможливо, СКБД контролює кількість посилань цілісність і не дозволить виконати операцію видалення первинного ключа таблиці, якщо на нього є посилання. Отже, в цьому випадку порядок зміни первинного ключа повинен бути таким, як на рис. 3.1:

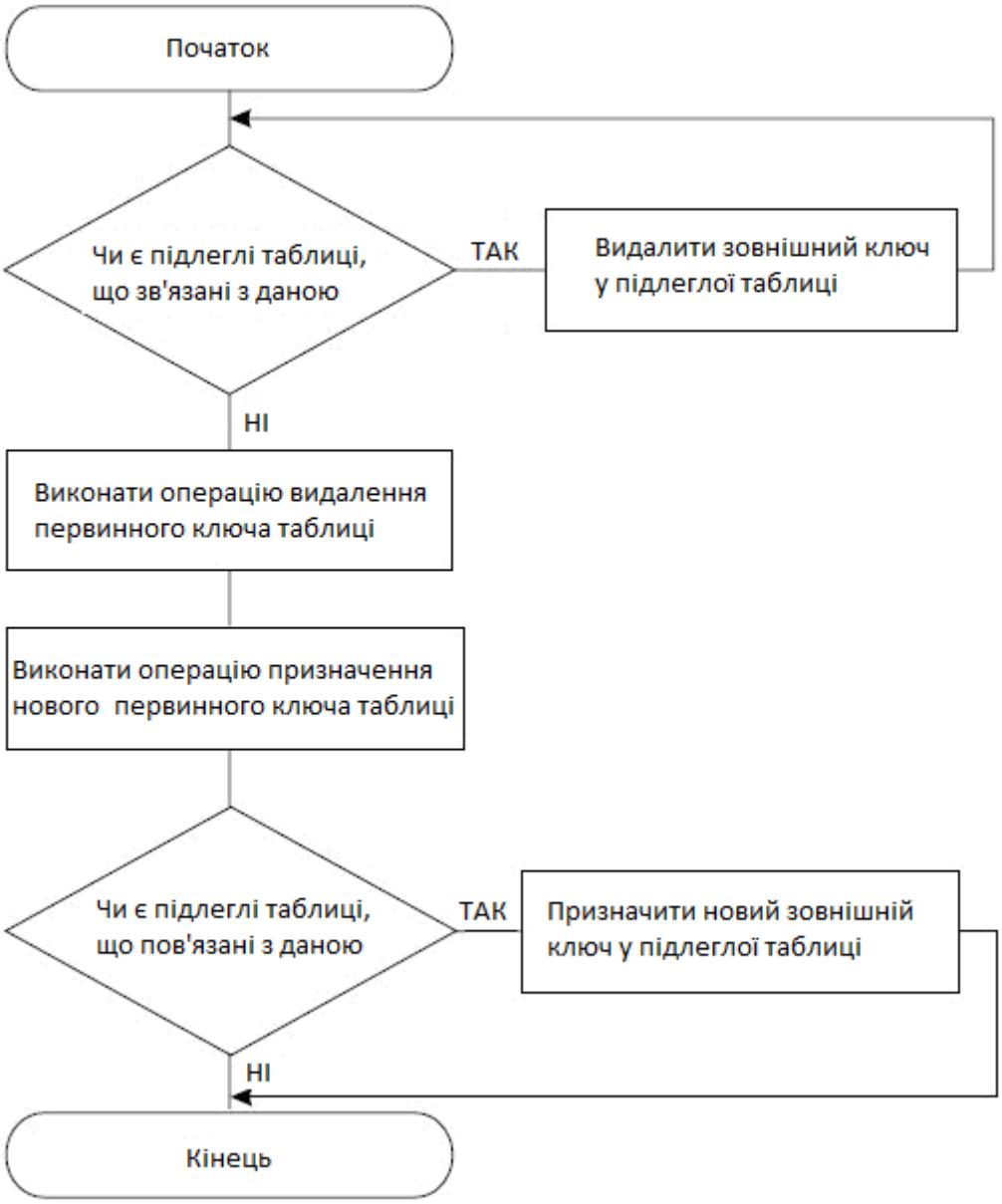


Рисунок 3.1 – Алгоритм зміни первинного ключа таблиці

Найчастіше операція ALTER TABLE застосовується в CASE-системах при автоматичній генерації скриптів створення таблиць в базі даних. У цих системах універсальний алгоритм передбачає спочатку створення всіх таблиць, які задані в даталогічній моделі, і лише після цього додаються відповідні зв'язки. І це зрозуміло – на відміну від людського розуму штучний інтелект CASE-системи буде зазнавати труднощів у визначенні ієрархічних взаємозв'язків таблиць бази даних, тому вважається за краще використовувати універсальний алгоритм, в якому спочатку всі об'єкти визначаються, а потім додаються відповідні властивості для атрибутів, які є зовнішніми ключами із зазначенням необхідних посилань. В цьому випа-

дку всі операції призначення зовнішніх ключів будуть вважатися коректними, тому що всі об'єкти були описані заздалегідь, і для такого алгоритму порядок створення таблиць не є важливим.

У мові SQL присутній і операція видалення таблиць. Синтаксис цієї операції зовсім простий:

```
DROP TABLE <им'я таблиці> [CASCADE | RESTRICT]
```

Параметр CASCADE означає, що при видаленні таблиці одночасно видаляються і всі об'єкти, пов'язані з нею. З таблицею, крім розглянутих раніше обмежень, можуть бути пов'язані також об'єкти типу тригерів і представлення. Однак операція видалення об'єктів визначається ще правами користувачів, що пов'язано з концепцією безпеки в базах даних. Це означає, що якщо ви не є власником об'єкта, то ви можете не мати прав на його видалення. І в цьому випадку синтаксично правильний оператор DROP TABLE не може бути виконаний системою в силу відсутності прав на видалення пов'язаних з видаляється таблицею об'єктів. Крім того, операція видалення таблиці не повинна порушувати цілісність бази даних, тому видаляти таблицю, на яку є посилання інших таблиць, неможливо.

Наприклад, в схемі, пов'язаній з бібліотекою, ми не можемо видалити ні таблицю BOOKS, ні таблицю READERS, ні таблицю CATALOG. У цих таблиць є зв'язок з підлеглими таблицями EXEMPLAR і RELATION_1. Тому якщо потрібно видалити деякий набір таблиць, то спочатку необхідно грамотно побудувати послідовність їх видалення, яка не порушить базових принципів підтримки цілісності схеми БД. У нашому прикладі послідовність операторів видалення таблиць може бути наступною:

```
DROP TABLE EXEMPLAR
DROP TABLE RELATION_1
DROP TABLE CATALOG
DROP TABLE READERS
DROP TABLE BOOKS
```

3.2 Поняття представлень. Операції створення представлень

Для опису зовнішніх моделей в реляційної моделі можуть використовуватися представлення. *Представлення* (подання, уявлення) (View) – це SQL-запит на вибірку, який *користувач* сприймає як деякий *віртуальне відношення*. Завдання представлень входить в опис схеми БД в СУБД. Представлення дозволяють приховати непотрібні несуттєві деталі для різних користувачів, модифікувати реальні структури даних в зручному для додатків вигляді і, нарешті, розмежувати права доступу до даних і тим самим підвищити захист даних від несанкціонованого доступу.

На відміну від реальної таблиці *представлення* у тому вигляді, як воно сконструйоване, не існує в базі даних, це дійсно тільки *віртуальне відношення*, хоча всі дані, які представлені в ньому, дійсно існують в базі даних, але в різних відносинах. Вони скомпоновані для користувача в зручному вигляді з *реальних таблиць* за допомогою деякого запиту. Однак *користувач* може цього не знати, він може звертатися з цією виставою як зі стандартною таблицею. Подання при створенні отримує деякий унікальне ім'я, його опис зберігається в описі схеми бази даних, і СКБД в будь-який момент часу при зверненні до цього подання виконує запит, відповідний його опису, тому користувач, працюючи з поданням, в кожен момент часу бачить дійсно реальні, актуальні на даний момент дані. Воно формується як би на льоту, в момент звернення.

Оператор визначення представлення має наступний вигляд:

```
CREATE VIEW <им’я представлення> [ (<список стовпців>)] AS <SQL-запит>
```

При необхідності в поданні можна задати нове ім'я для кожного стовпця віртуальної таблиці. При цьому треба пам'ятати, що якщо вказується список стовпців, то він повинен містити рівно стільки стовпців, скільки містить їх SQL-запит.

Якщо список імен стовпців в поданні не вказано, то кожен стовпець подання отримує ім'я відповідного стовпця запиту. Список стовпців у поданні обов’язково потрібно вказувати, якщо в SQL-запиті є неіменовані стовпці, що розраховуються. Наприклад, у базі даних «Розклад занять» є таблиця Groups, що описує наявні академічні групи університету:

```
CREATE TABLE Groups (
      id_f smallint NOT NULL,
      id_gr smallint NOT NULL PRIMARY KEY,
      name_gr varchar (10) UNIQUE,
      kol tinyint NULL ,
      plan_kol tinyint NULL ,
      CONSTRAINT [FK__groups_id_f] FOREIGN KEY (id_f)
      REFERENCES facultet (id_f) ON UPDATE CASCADE
)
```

У цьому відношенні атрибут kol описує кількість студентів групи у поточному семестрі, атрибут plan_kol – запланована кількість студентів у наступному семестрі. Якщо розклад занять потрібно скласти на весняний семестр, то заплановану кількість студентів не потрібно задавати – вона буде дорівнюватись kol. Якщо розклад занять потрібно скласти на осінній семестр, то заплановану кількість студентів потрібно задавати.

Тоді список груп університету з поточним та запланованим контингентом студентів можна отримати наступним запитом:

```
select name_gr, kol, case when plan_kol is null then kol else plan_kol end
from Groups
where (kol is not null and kol>0) or (plan_kol is not null and plan_kol>0)
```

Запит передбачає, що можуть бути групи, у яких немає студентів або в поточному, або в наступному семестрі. Результат запиту має вигляд (рис. 3.2):

	name_gr	kol	(No column name)
1	К-11	20	20
2	К-21	22	22
3	К-31	0	14
4	К-32	18	18
5	К-41	9	15

Рисунок 3.2 – Результат виконання запиту

Як видно з рис. 3.2, третій стовпець, що розраховується за допомогою функції CASE, не має назви (No column name). Якщо запит з таким стовпцем використовувати у операторі визначення представлення і не вказати список стовпців після імені представлення, то команда створення представлення не виконається, а буде видане повідомлення:

```
Create View or Function failed because no column name was specified
for column 3.
```

(Помилка створення Представлення або Функції, оскільки для стовпця 3 не було вказано імені стовпця)

У цьому випадку без зміни SQL-запиту можна створити представлення наступною командою:

```
create view ViewGroup (name_gr, kol, plan_count) as
select name_gr, kol, case when plan_kol is null then kol else plan_kol end
from groups
where (kol is not null and kol>0) or (plan_kol is not null and plan_kol>0)
```

4 ПРОЦЕДУРНА МОВА T-SQL

Мова SQL призначена для організації доступу до баз даних. При цьому передбачається, що доступ до БД може бути здійснений в двох режимах: в інтерактивному режимі і в режимі виконання прикладних програм (додатків).

Ця двоїстість SQL створює ряд переваг:

- Всі можливості інтерактивного мови запитів доступні і в прикладному програмуванні.
- Можна в інтерактивному режимі налагодити основні алгоритми обробки інформації, які в подальшому можуть бути готовими вставлені в працюючі додатки.

SQL дійсно є мовою по роботі з базами даних, але в явному вигляді вона не є мовою програмування. У ньому відсутні традиційні оператори, що організують цикли, що дозволяють оголосити і використовувати внутрішні змінні, організувати аналіз деяких умов і можливість зміни ходу програми залежно від виконаного умови. У загальному випадку можна назвати SQL підмовою, яка служить виключно для управління базами даних. Для створення додатків, справжніх програм необхідно використовувати інші, базові мови програмування, в які оператори мови SQL будуть вбудовуватися.

Якщо передбачається розробка додатку до бази даних, з яким будуть працювати багато користувачів, бажано обробку даних здійснювати на боці серверу, а не клієнтської програми. Така обробка можлива з використанням збережених процедур – процедур, що написані мовою Transact-SQL (T-SQL), зберігаються у базі даних, обробляються на боці серверу і викликаються з клієнтської програми.

4.1 Основні конструкції мови T-SQL

Опис змінних. Для опису змінних використовується оператор DECLARE. Змінні оголошуються в тілі пакету або процедури з оператором DECLARE і присвоюються значенням або оператором SET, або SELECT. Після оголошення всі змінні ініціалізуються як NULL:

```
DECLARE   @local_variable data_type [ ,...n]
```

Наприклад, об’явимо змінну цілого типу та змінну типу символьного рядка довжиною 50 символів. Присвоїмо першій змінній значення 25, а другій – «Приклад визначення змінних»:

```
DECLARE @x smallint, @y varchar(50)
SET @x =25
SET @y ='Приклад визначення змінних'
```

Блок операторів Transact-SQL. Блок операторів складається з групи операторів, укладених у операторні дужки – слова BEGIN та END:

```
BEGIN
      sql_statement
      ...
      sql_statement
END
```

Умовний оператор. Покладає умови на виконання оператора Transact-SQL. Оператор Transact-SQL після ключового слова IF і його умови виконується, якщо умова виконується (коли вираз Boolean повертає TRUE). Необов'язкове ключове слово ELSE вводить альтернативний оператор Transact-SQL, який виконується, коли умова IF не виконується (коли булевий вираз повертає FALSE).

Синтаксис:

```
IF Boolean_expression
    { sql_statement | statement_block }
[ ELSE
    { sql_statement | statement_block } ]
```

Приклад використання умовного оператору. Якщо в бібліотеці є вільні екземпляри підручника «Організація баз даних та знань», вивести їх кількість, якщо немає – вивести імена та прізвища читачів, що тримають на руках екземпляри цього підручника:

```
IF (SELECT COUNT(*) FROM Exemplar E JOIN Books B ON E.ISBN = B.ISBN
   WHERE TITLE = 'Організація баз даних та знань' AND READER_ID IS NULL) >0
      SELECT COUNT(*) AS "Є екземплярів"
      FROM Exemplar E JOIN Books B ON E.ISBN = B.ISBN
      WHERE TITLE = 'Організація баз даних та знань' AND READER_ID IS NULL
ELSE
     SELECT FIRST_NAME+' '+ LAST_NAME AS "Читач"
     FROM Exemplar E JOIN Books B ON E.ISBN = B.ISBN JOIN Readers R
       ON E.READER_ID = R.READER_ID
     WHERE TITLE = 'Організація баз даних та знань'
```

4.2 Оператори, пов'язані з багаторядковими запитами

Розглянемо більш складні багаторядкові запити.

Для реалізації багаторядкових запитів вводиться нове поняття – поняття курсору або покажчика набору записів. Для роботи з курсором додається кілька нових операторів SQL:

1. Оператор DECLARE CURSOR – визначає виконується запит, задає ім'я курсору і зв'язує результати запиту з заданим курсором. Цей оператор не є виконуваним для запиту, він тільки визначає структуру майбутньої множини записів і пов'язує її з унікальним ім'ям курсору. Цей оператор подібний до операторам опису даних в мовах програмування.

2. Оператор OPEN дає команду СКБД виконати описаний запит, створити віртуальний набір рядків, який відповідає заданому запиту. Оператор OPEN встановлює покажчик записів (курсор) перед першим рядком віртуального набору рядків результату.

3. Оператор FETCH просуває покажчик записів на наступну позицію в віртуальному наборі записів. У більшості комерційних СКБД оператор переміщення FETCH реалізує ширші функції переміщення, він дозволяє переміщати покажчик на довільну запис, вперед і назад, допускає як абсолютну адресацію, так і відносну адресацію, дозволяє встановити курсор на першу або останню запис віртуального набору.

4. Оператор CLOSE закриває курсор і припиняє доступ до віртуального набору записів. Він фактично ліквідує зв'язок між курсором і результатом виконання базового запиту. Однак в комерційних СУБД оператор CLOSE не завжди означає знищення віртуального набору записів.

5. Оператор DEALLOCATE. Видаляє посилання курсору. Коли відбувається розміщення останнього посилання курсору, Microsoft SQL Server вивільняє структури даних, що містять курсор.

4.2.1 Оператор визначення курсору

Стандарт визначає наступний синтаксис оператора визначення курсору:

DECLARE *cursor_name* CURSOR
FOR *select_statement*

cursor_name – це допустимий ідентифікатор в базовій мові програмування. В оголошенні курсору можуть бути використані базові змінні. Однак необхідно пам'ятати, що на момент виконання оператора OPEN значення всіх базових змінних, використовуваних в якості вхідних змінних, пов'язаних з умовами фільтрації значень в базовому запиті, повинні бути вже задані.

Визначимо курсор, який містить список всіх боржників нашої бібліотеки. Боржниками назвемо читачів, які мають на руках хоча б одну книгу, термін здачі якої вже пройшов.

```
DECLARE Debtor_reader_cursor CURSOR FOR
SELECT READERS.FIRST_NAME, READERS.LAST_NAME,
READERS.ADRES,
READERS.HOME_PHON, READERS.CELL_PHON, BOOKS.TITLE
FROM READERS R JOIN EXEMPLAR E ON R.READER_ID =
E.READER_ID JOIN BOOKS B ON B.ISBN = E.ISBN
WHERE EXEMPLAR.DATA_OUT < Getdate()
ORDER BY READERS.FIRST_NAME
```

При визначенні курсору ми знову використовували функцію Transact SQL Getdate (), яка повертає значення поточної дати. Таким чином, певний курсор буде створювати набір рядків, що містять перелік боржників, із зазначенням назв книг, які вони не повернули вчасно в бібліотеку.

Відповідно до стандарту SQL2 визначення курсору має синтаксис:

DECLARE *cursor_name* [INSENSITIVE] [SCROLL] CURSOR
FOR *select_statement*
[FOR { READ ONLY | UPDATE [OF *column_name* [**,**...*n*]] }]

Параметр INSENSITIVE (нечутливий) визначає режим створення набору рядків, відповідного визначеному курсору, при якому всі зміни у вихідних таблицях, вироблені після відкриття курсору іншими користувачами, не відображаються в ньому. Такий набір даних нечутливий до всіх змін, які можуть проводитися іншими користувачами в початкових таблицях; цей тип курсору відповідає деякому миттєвому зліпку з БД.

СКБД більш швидко та економно може обробляти такий курсор, тому якщо дійсно важливо розглянути і опрацювати стан БД на деякий конкретний момент часу, то має сенс створити "нечутливий курсор".

Ключове слово SCROLL визначає, що допустимі будь-які режими переміщення по курсору (FIRST, LAST, PRIOR, NEXT, RELATIVE, ABSOLUTE) в операторі FETCH.

Якщо не вказано ключове слово SCROLL, то вважається доступною тільки стандартне переміщення вперед: специфікація NEXT в операторі FETCH.

Якщо вказана специфікація READ ONLY (тільки для читання), то зміни та оновлення вихідних таблиць не будуть виконуватися з використанням даного курсору. Курсор з даною специфікацією може бути найшвидшим в обробці, однак якщо ви не вкажіть спеціально специфікацію READ ONLY, то СКБД буде вважати, що ви допускаєте операції модифікації з базовими таблицями, і в цьому випадку для забезпечення цілісності БД СКБД буде набагато повільніше обробляти ваші операції з курсором.

При використанні параметра UPDATE [OF <ім'я стовпця 1> [... <ім'я стовпця n>]] задається перелік стовпців, в яких допустимі зміни в процесі роботи з курсором. Таке обмеження спростить і прискорить роботу СКБД. Якщо цей параметр не вказано, то передбачається, що допустимі зміни всіх стовпців курсора.

Розширений синтаксис Transact-SQL містить розширене визначення курсору:

DECLARE *cursor_name* CURSOR
[LOCAL | GLOBAL]
[FORWARD_ONLY | SCROLL]
[STATIC | KEYSET | DYNAMIC | FAST_FORWARD]
[READ_ONLY | SCROLL_LOCKS | OPTIMISTIC]
[TYPE_WARNING]
FOR *select_statement*
[FOR UPDATE [OF *column_name* [**,***...n*]]]

Аргументи:

LOCAL

Вказує, що область дії курсору є локальною для пакету, збереженої процедури або тригера, в якому створено курсор. Ім’я курсору дійсне лише в цієї області. На курсор можна посилатися локальними змінними курсору в пакеті, збереженій процедурі або тригері, або параметром збереженої процедури з визначенням OUTPUT. Параметр OUTPUT використовується для передачі локального курсору назад в пакет, збережену процедуру або тригер, який може призначити параметр змінній курсору для посилання на курсор після закінчення збереженої процедури.

GLOBAL

Вказує, що область дії курсору є глобальною для сеансу з'єднання. На ім'я курсору можна посилатись у будь-якій збереженій процедурі або пакеті, що виконуються в сеансі. Курсор розміщений лише неявно при від-ключенні.

Примітка Якщо не вказано ані GLOBAL, ані LOCAL, за замовчуванням керується встановленням параметру бази даних для місцевих курсорів за замов-чуванням.

FORWARD_ONLY

Вказує, що курсор можна прокручувати лише з першого до остан-нього рядка. FETCH NEXT – єдиний підтримуваний варіант отримання. Якщо FORWARD_ONLY вказано без ключових слів STATIC, KEYSET або DYNAMIC, курсор працює як курсор DYNAMIC. Якщо не визначено ані FORWARD_ONLY, ані SCROLL, типовим є FORWARD_ONLY, якщо не

вказані ключові слова STATIC, KEYSET або DYNAMIC. Курсори STATIC, KEYSET і DYNAMIC за замовчуванням для SCROLL. На відміну від API баз даних, таких як ODBC та ADO, FORWARD_ONLY підтримується курсорами STATIC, KEYSET та DYNAMIC Transact-SQL. FAST_FORWARD та FORWARD_ONLY є взаємовиключними; якщо одна вказана, інша не може бути вказана.

STATIC

Визначає курсор, який робить тимчасову копію даних, які використовуються курсором. На всі запити до курсору відповідають з цієї тимчасової таблиці в tempdb; отже, модифікації, внесені до базових таблиць, не відображаються у даних, повернених вибірками, внесеними до цього курсору, і цей курсор не дозволяє змінювати.

KEYSET

Вказує, що членство та порядок рядків у курсорі фіксуються при відкритті курсору. Набір ключів, що однозначно ідентифікують рядки, вбудований у таблицю в tempdb, відому як **keyset**. Зміни неключових значень у базових таблицях, зроблені власником курсору, або здійснені іншими користувачами, є видимими, коли власник прокручує курсор. Додавання записів іншими користувачами не видимі (вставки не можна робити через курсор сервера Transact-SQL). Якщо рядок видалено, спроба отримати його повертає значення -2 для змінної @@ FETCH_STATUS.

DYNAMIC

Визначає курсор, який відображає всі зміни даних, внесені до рядків у його наборі результатів під час прокрутки курсору. Значення даних, порядок та належність рядків можуть змінюватись для кожній виборці. Опція вибірки ABSOLUTE не підтримується динамічними курсорами.

FAST_FORWARD

Вказує FORWARD_ONLY, READ_ONLY курсор з увімкненою оптимізацією продуктивності. FAST_FORWARD не можна вказати, якщо вказано також SCROLL або FOR_UPDATE. FAST_FORWARD та FORWARD_ONLY є взаємовиключними; якщо одна вказана, інша не може бути вказана.

READ_ONLY

Запобігає оновленням, здійсненим за допомогою цього курсору. На курсор не можна посилатися в опції WHERE CURRENT OF в операторі

UPDATE або DELETE. Ця опція перекриває можливість оновлення курсору за замовчуванням.

SCROLL_LOCKS

Визначає, що позиціоновані оновлення або видалення, зроблені за допомогою курсору, гарантовано матимуть успіх. SQL Server блокує рядки під час їх зчитування в курсор, щоб забезпечити їх доступність для наступних модифікацій. SCROLL_LOCKS не можна вказати, якщо також вказано FAST_FORWARD.

OPTIMISTIC

Вказує, що позиціоновані оновлення або видалення, виконані за допомогою курсору, не досягають успіху, якщо рядок було оновлено з моменту його зчитування в курсор. SQL Server не блокує рядки, коли вони читаються в курсор. Він замість цього використовує порівняння значень стовпців часових позначок або значення контрольної суми, якщо в таблиці немає стовпчика часової позначки, щоб визначити, чи змінено рядок після його зчитування в курсор. Якщо рядок було змінено, спроба позиціонованого оновлення або видалення не вдається. OPTIMISTIC не можна вказати, якщо також вказано FAST_FORWARD.

TYPE_WARNING

Вказує, що попереджувальне повідомлення надсилається клієнту, якщо курсор неявно перетворений із запитуваного типу в інший

Повернемося до нашого прикладу. Якщо ми ставимо собі за мету миттєвого зліпка БД, що дає відомості про боржників, то застосуємо всі параметри, що дозволяють прискорити роботу з нашим курсором. Тоді оператор опису курсора буде виглядати наступним чином:

```
DECLARE Debtor_reader_cursor CURSOR FAST_FORWARD FOR
SELECT READERS.FIRST_NAME, READERS.LAST_NAME,
READERS.ADRES, READERS.HOME_PHON, READERS.CELL_PHON,
BOOKS.TITLE
FROM READERS R JOIN EXEMPLAR E ON R.READER_ID =
E.READER_ID JOIN BOOKS B ON B.ISBN = E.ISBN
WHERE EXEMPLAR.DATA_OUT < Getdate()
ORDER BY READERS.FIRST_NAME
```

При опису курсору немає обмежень на вид оператора SELECT, який використовується для створення базового набору рядків, пов'язаного з курсором. В операторі SELECT можуть використовуватись угруповання і вбудовані підзапити і обчислювані поля.

4.2.2 Оператор відкриття курсору

Оператор відкриття курсору має наступний синтаксис:

OPEN { { [GLOBAL] *cursor_name* } | *cursor_variable_name* }

Аргументи:

cursor_name

Ім'я оголошеного курсору. Коли існує як глобальний, так і локальний курсор, з іменами *cursor_name*, тоді *cursor_name* посилається на глобальний курсор, якщо вказано GLOBAL; в іншому випадку *cursor_name* посилається на локальний курсор

cursor_variable_name

Ім'я змінної курсору, яка посилається на курсор

Саме оператор відкриття курсору ініціює виконання базового запиту, відповідного опису курсору, заданому в операторі DECLARE ... CURSOR. При виконанні оператора OPEN СКБД виробляє семантичну перевірку курсору, тому саме тут СУБД повертає коди помилок прикладній програмі, що повідомляють їй про результати виконання базового запиту. Помилки можуть виникнути в результаті неправильного завдання імен полів або імен вихідних таблиць або при спробі витягти дані з таблиць, до яких даний користувач не має доступу.

За стандартом СКБД повертає код завершення операції в спеціальній системній змінній SQLCODE. У прикладній програмі користувач може аналізувати цю змінну, що необхідно робити після виконання кожного оператора SQL. При невдалому виконанні операції відкриття курсору СКБД повертає від'ємне значення SQLCODE.

У разі вдалого завершення виконання оператора відкриття курсору набір даних, сформований в результаті базового запиту, залишається доступним користувачеві до моменту виконання оператора закриття курсору.

Однак треба пам'ятати, що СУБД автоматично закриває всі курсори в разі завершення транзакції (COMMIT) або відкату транзакції (ROLLBACK). Після того як курсор закритий, його можна відкрити знову, але при цьому відповідний запит виконається заново. Тому допустимо, що вміст першого курсору буде не відповідати його вмісту при повторному відкритті, тому що за цей час змінився стан БД.

Приклад оператору відкриття курсору:

```
OPEN Debtor_reader_cursor
```

4.2.3 Оператор читання чергового рядка курсору

Після відкриття покажчик поточного рядка встановлений перед першим рядком курсору. Стандартно оператор FETCH переміщує покажчик поточного рядка на наступний рядок і привласнює базовим змінним значення стовпців, відповідне поточному рядку.

Простий оператор FETCH має наступний синтаксис:

FETCH NEXT FROM *cursor_name*
INTO *@variable_name*[**,***...n*]

Аргументи:

INTO *@variable_name*[**,***...n*]

Дозволяє розміщувати дані з стовпців вибірки у локальні змінні. Кожна змінна у списку зліва направо асоціюється з відповідним стовпцем у наборі результатів курсору. Тип даних кожної змінної повинен або відповідати типу даних відповідного стовпця набору результатів, або підтримувати неявне перетворення до вказаного типу. Кількість змінних повинна відповідати кількості стовпців у списку вибору курсору.

Оператор вилучення чергового рядка з описаного вище курсору буде виглядати наступним чином:

```
FETCH Debtor_reader_cursor INTO
      @FIRST_NAME, @LAST_NAME,@ADRES,
      @HOME_PHON, @WORK_PHON, @TITLE
```

Розширений оператор FETCH має наступний синтаксис:

FETCH [[NEXT | PRIOR | FIRST | LAST | ABSOLUTE { *n* | *@nvar* }
| RELATIVE { *n* | *@nvar* }]
FROM]
{ { [GLOBAL] *cursor_name* } | *@cursor_variable_name* }
[INTO *@variable_name* [**,***...n*]]

Аргументи:

NEXT

Повертає рядок результату відразу після поточного рядка і збільшує поточний рядок до повернутого рядка. Якщо FETCH NEXT є першою вибіркою для курсору, він повертає перший рядок в наборі результатів. NEXT – опція вибірки курсору за умовчанням

PRIOR

Повертає рядок результату, що безпосередньо передує поточному рядку, і зменшує поточний рядок до повернутого рядка. Якщо FETCH

PRIOR є першим витягом з курсору, рядок не повертається, і курсор залишається розташованим перед першим рядком.

FIRST

Повертає перший рядок результату і робить його поточним рядком.

LAST

Повертає останній рядок результату і робить його поточним рядком.

Крім того, в розширеному операторі переміщення допустимо переміститися одразу на заданий рядок, при цьому допустима як абсолютна адресація, завданням параметра ABSOLUTE, так і відносна адресація, завданням параметра RELATIVE. При відносній адресації позитивне число зрушує покажчик вниз від поточного запису, негативне число зрушує вгору від поточного запису.

4.2.4 Оператор закриття курсору

Оператор закриття курсору має простий синтаксис, він виглядає наступним чином:

CLOSE *cursor_name*

Оператор закриття курсору закриває тимчасову таблицю, створену оператором відкриття курсору, і припиняє доступ прикладної програми до цього об'єкта. Єдиним параметром оператора закриття є ім'я курсору. Оператор закриття може бути виконаний в будь-який момент після оператора відкриття курсору.

У деяких комерційних СКБД крім оператора закриття курсору використовується ще оператор деактивації (знищення) курсору. Наприклад, в MS SQL Server поряд з оператором закриття курсору використовується оператор

DEALLOCATE *cursor_name*

Тут оператор закриття курсору не знищує набір даних, пов'язаний з курсором, він тільки закриває доступ до нього і звільняє всі блокування, які раніше були пов'язані з даними курсором.

При виконанні оператора DEALLOCATE SQL Server звільняє пам'ять, що розділяється, яка використовується командою опису курсору DECLARE. Після виконання цієї команди неможливе виконання команди OPEN для даного курсору.

4.2.5 Прохід по рядках вибірки курсору

Курсори часто створюють для послідовної обробки даних, які знаходяться в рядках вибірки, отриманої командою SELECT при описі курсору. Для послідовного доступу до даних з кожного рядка вибірки потрібно використати цикл.

Оператор циклу WHILE. Встановлює умову для повторного виконання оператора SQL або блоку операторів. Оператори виконуються багаторазово, поки зазначена умова виконується. Виконанням операторів в циклі WHILE можна управляти зсередини циклу за допомогою ключових слів BREAK і CONTINUE.

Синтаксис команди:

WHILE *Boolean_expression*
{ *sql_statement* | *statement_block* }

Умова продовження циклу використовує глобальну для всіх курсорів змінну @@FETCH_STATUS. Ця змінна повертає стан останнього оператора FETCH, виданого для будь-якого курсора, відкритого в даний момент з'єднанням. Змінна має наступні значення:

Значення, що повертається	Опис
0	Оператор FETCH виконався успішно
-1	Помилка оператора FETCH або рядок перевищує набір результатів
-2	Відсутній рядок вибірки

Таким чином, умова продовження циклу прокрутки рядків курсору має вигляд:

WHILE @@FETCH_STATUS = 0

4.2.6 Загальна схема обробки даних за допомогою курсору

Спочатку оголошується курсор, потім він відкривається (покажчик запису встановлюється перед першим рядком вибірки). Покажчик запису переміщується на наступну запис. Доки виконується умова подовження циклу: @@FETCH_STATUS = 0 – обробляються дані, що відповідають рядкам курсору та покажчик запису переміщується на наступну запис. Після закінчення циклу курсор закривається, а у випадку СКБД MS SQL Server – курсор ще і деактивується;

```
DECLARE cursor_name CURSOR FAST_FORWARD FOR
select_statement
OPEN cursor_name
FETCH NEXT FROM cursor_name INTO variable_list
WHILE @@FETCH_STATUS = 0
BEGIN
     select_statement
[ ... select_statement ]
     FETCH NEXT FROM cursor_name INTO variable_list
END
CLOSE cursor_name
DEALLOCATE cursor_name
```

4.3 Процедурні розширення T-SQL

Пакет (batch) – це послідовність інструкцій і процедурних розширень мови T-SQL.

Підпрограма (routine) – це збережена процедура, або обумовлена користувачем функція (ОКФ) (User Defined Function – UDF).

Кількість інструкцій в пакеті обмежується допустимим розміром скомпільованої пакетного об'єкта.

Перевага пакету над групою окремих інструкцій полягає в тому, що одночасне виконання всіх інструкцій дозволяє отримати значне поліпшення продуктивності.

Існує кілька обмежень на включення різних інструкцій мови Transact-SQL в пакет. Найбільш важливим з них є та обставина, що якщо пакет містить інструкцію опису даних CREATE VIEW, CREATE PROCEDURE або CREATE TRIGGER, то він не може містити ніяких інших інструкцій.

Інструкції мови опису даних поділяються за допомогою інструкції GO.

5 ЗБЕРЕЖЕНІ ПРОЦЕДУРИ

Збережена процедура – це спеціальний тип пакета інструкцій T-SQL, створений, з використанням мови SQL і процедурних розширень. З точки зору додатків, що працюють з БД, збережені процедури (Stored Procedure) – це підпрограми, які виконуються на сервері. По відношенню до БД – це об'єкти, які створюються і зберігаються в БД. Вони можуть бути викликані з клієнтських додатків. При цьому одна процедура може бути використана в будь-якій кількості клієнтських додатків, що дозволяє істотно заощадити трудовитрати на створення прикладного програмного забезпечення та ефективно застосовувати стратегію повторного використання коду. Так само як і будь-які процедури в стандартних мовах програмування, збережені процедури можуть мати вхідні і вихідні параметри або не мати їх зовсім.

Компонент Database Engine підтримує збережені процедури і системні процедури. Збережені процедури створюються таким же чином, як і всі інші об'єкти баз даних, тобто за допомогою мови DDL. Системні процедури надаються компонентом Database Engine і можуть застосовуватися для доступу до інформації в системному каталозі і її модифікації. Ми розглянемо збережені процедури, які визначаються користувачами.

Збережені процедури можуть бути активізовані не тільки одними додатками, але і тригерами.

Збережені процедури пишуться на спеціальній вбудованій мові програмування, вони можуть включати будь-які оператори SQL, а також включають певний набір операторів, що керують ходом виконання програм, які багато в чому схожі з подібними операторами процедурно орієнтованих мов програмування. У комерційних СУБД для написання текстів збережених процедур використовуються власні мови програмування, так, в СУБД Oracle для цього використовується мова PL/SQL, а в MS SQL Server і System11 фірми Sybase використовується мова Transact-SQL (T-SQL). У Oracle також використовується мова Java для написання збережених процедур.

Збережені процедури є об'єктами БД. Кожна збережена процедура компілюється при першому виконанні, в процесі компіляції будується оптимальний план виконання процедури. Опис процедури спільно з планом її виконання зберігається в системних таблицях БД.

Для створення збереженої процедури застосовується оператор SQL CREATE PROCEDURE.

За замовчуванням виконати збережену процедуру може тільки її власник, яким є власник БД, і творець збереженої процедури. Однак власник збереженої процедури може делегувати права на її запуск іншим користувачам.

Ім'я збереженої процедури є ідентифікатором в мові програмування, на якій вона пишеться, і має відповідати всім вимогам, які пред'являються до ідентифікаторів в даній мові.

В MS SQL Server оператор створення збереженої процедури має синтаксис:

```
  CREATE PROC [ EDURE ] procedure_name [ ; number ]
 [ { @parameter data_type }
    [ VARYING ] [ = default ] [ OUTPUT ]
 ] [ ,...n ]
[ WITH  { RECOMPILE | ENCRYPTION | RECOMPILE , ENCRYPTION } ]
[ FOR REPLICATION ]
AS sql_statement [ ...n ]
```

Або для зменшення опису аргументів команди її синтаксис можна написати так:

```
CREATE PROC[EDURE] <и'мя_процедури> [;<версія>]
[{@параметр1 тип_даних} [VARYING]
[= <значення_за_замовчуванням>] [OUTPUT]] [,параметрN…]
[ WITH {RECOMPILE, ENCRYPTION}]
[FOR REPLICATION] AS Тіло_процедури
```

Тут необов'язкове ключове слово VARYING визначає задане значення за замовчуванням для визначеного раніше параметра.

Ключове слово RECOMPILE визначає режим компіляції створюваної збереженої процедури. Якщо задано ключове слово RECOMPILE, то процедура буде перекомпілювати кожен раз, коли вона буде викликатися на виконання. Це може різко сповільнити виконання процедури. Але, з іншого боку, якщо дані, що обробляються даної збереженої процедурою, настільки динамічні, що попередній план виконання, складений при її першому виклику, може бути абсолютно неефективний при наступних викликах, то варто застосовувати даний параметр при створенні цієї процедури.

Ключове слово ENCRYPTION визначає режим, при якому вихідний текст збереженої процедури не зберігається в БД. Такий режим застосовується для того, щоб зберегти авторське право на інтелектуальну продукцію, якої і є збережені процедури. Часто такий режим застосовується, коли ви ставите готову базу замовнику і не хочете, щоб вихідні тексти розроб-

лених вами збережених процедур були б доступні адміністратору БД, що працює у замовника. Однак треба пам'ятати, що якщо ви захочете відредагувати текст збереженої процедури самі, то ви його не зможете отримати з БД теж, його треба буде зберігати окремо в деякому текстовому файлі. І це не найгірше, але от у випадку відновлення БД після серйозної аварії для перекомпіляції будуть потрібні початкові вихідні тексти всіх збережених процедур. Тому захист річ хороша, але вона ускладнює супровід і модифікацію збережених процедур.

Однак крім імені збереженої процедури всі інші параметри є необов'язковими. Процедури можуть бути процедурами або процедурами-функціями. І ці поняття тут трактуються традиційно, як в мовах програмування високого рівня. Процедура-функція повертає значення, яке присвоюється змінної, що визначає ім'я процедури. Процедура в явному вигляді не повертає значення, але в ній може бути використано ключове слово OUTPUT, яке визначає, що даний параметр є вихідним.

Розглянемо кілька прикладів найпростіших процедур.

Процедура перевірки наявності примірників заданої книги. Книга визначається параметром ISBN – шифром книги.

Процедура повертає параметр, що дорівнює кількості примірників книги. Якщо повертається нуль, то це означає, що немає вільних примірників цієї книги в бібліотеці.

```
CREATE PROCEDURE COUNT_EX (
   @ISBN varchar(12))
AS
BEGIN
     DECLARE @TEK_COUNT int
     SET @TEK_COUNT = (select count(*)
     FROM EXEMPLAR
     WHERE ISBN = @ISBN AND READER_ID Is NULL
      AND EXIST = True)
     RETURN @TEK_COUNT
END
```

У команді SELECT розраховується кількість екземплярів книги, що знаходяться не на руках читачів, а в бібліотеці. Процедура повертає нуль, якщо немає жодного вільного примірника.

Збережена процедура може бути викликана декількома способами.

Найпростіший спосіб – це використання оператора EXECUTE:

```
{ EXEC | EXECUTE }
   { procedure_name [ ;number ] | @procedure_name_var     }
     [ [ @parameter = ] { value | @variable [ OUTPUT ] | [ DEFAULT ] }    [ ,...n ]
   [ WITH RECOMPILE ]
```

Аргументи:

procedure_name

Повне або не кваліфіковане ім'я процедури для виклику. Імена процедур повинні відповідати правилам для ідентифікаторів. Імена розширених збережених процедур завжди чутливі до регістру, незалежно від кодової сторінки або порядку сортування сервера.

Процедура, створена в іншій базі даних, може бути виконана, якщо користувач, що виконує процедуру, володіє процедурою або має відповідний дозвіл на її виконання в цій базі даних. Процедуру можна виконати на іншому сервері під керуванням Microsoft® SQL Server ™, якщо користувач, що виконує процедуру, має відповідні дозволи на використання цього сервера (віддалений доступ) і виконання процедури в цій базі даних. Якщо вказано ім'я сервера, але ім'я бази даних не вказано, SQL Server шукає процедуру в базі даних користувача за замовчуванням.

;*number*

Необов'язкове ціле число, яке використовується для угруповання процедур з однаковими іменами, щоб їх можна було видалити одним оператором DROP PROCEDURE. Цей параметр не використовується для розширених збережених процедур.

Процедури, які використовуються в одному додатку, часто групуються таким чином. Наприклад, процедури, які використовуються в додатку orders, можуть називатися orderproc;1, orderproc;2, тощо. Оператор DROP PROCEDURE orderproc видаляє всю групу. Після того, як процедури були згруповані, окремі процедури в групі не можуть бути відкинуті. Наприклад, оператор DROP PROCEDURE orderproc; 2 неприпустимий.

@procedure_name_var

Ім'я локальної змінної, що представляє ім'я процедури.

@parameter

Параметр для процедури, визначений у операторі CREATE PROCEDURE. Іменам параметрів повинен передувати знак at (@). При використанні з формою @parameter_name = value імена параметрів і константи не обов’язково вказувати саме в тому порядку, в якому вони визначені в операторі CREATE PROCEDURE. Однак, якщо форма @parameter_name =

value використовується для будь-якого параметра, вона повинна використовуватися для всіх наступних параметрів.

Параметри обнуляються за замовчуванням. Якщо передається значення параметра NULL і цей параметр використовується в операторі CREATE або ALTER TABLE, в якому зазначений стовпець не допускає значення NULL (наприклад, вставка в стовпець, який не допускає значення NULL), SQL Server генерує помилку. Щоб запобігти передачі значення параметра NULL в стовпець, в якому не допускаються значення NULL, потрібно додати програмну логіку в процедуру, або використовуати значення за замовчуванням (з ключовим словом DEFAULT CREATE або ALTER TABLE) для стовпця.

value

Значення параметра для процедури. Якщо імена параметрів не вказані, значення параметрів повинні надаватися в порядку, визначеному в операторі CREATE PROCEDURE.

Якщо значення параметра є ім'ям об'єкту, символьним рядком або кваліфіковано за іменем бази даних або іменем власника, все ім'я має бути укладена в одинарні лапки. Якщо значення параметра є ключовим словом, ключове слово повинно бути укладено в подвійні лапки.

Якщо в операторі CREATE PROCEDURE визначено значення за замовчуванням, користувач може виконати процедуру без вказівки параметра. Значення за замовчуванням має бути константою і може включати символи підстановки %, _, [] і [^], якщо процедура використовує ім'я параметра з ключовим словом LIKE.

Значення за замовчуванням також може бути NULL. Зазвичай визначення процедури визначає дію, яке слід зробити, якщо значення параметра дорівнює NULL.

@variable

Змінна, в якій зберігається параметр або параметр, що повертається.

OUTPUT

Вказує, що збережена процедура повертає параметр. Відповідний параметр в збереженій процедурі також повинен бути створений з ключовим словом OUTPUT. Потрібно вказувати це ключове слово при використанні змінних курсору в якості параметрів.

Якщо використовуються параметри OUTPUT і передбачається використовувати значення, що повертаються, в інших операторах всередині пакета або процедури, значення параметра має бути передано як змінна (тобто @parameter = @variable). Не можна виконати процедуру, вказавши OUTPUT для параметра, який не визначений як параметр OUTPUT в операторі CREATE PROCEDURE. Константи не можуть бути передані в збережені процедури за допомогою OUTPUT; параметр, що повертається, вимагає ім'я змінної. Тип даних змінної повинен бути оголошений, і перед виконанням процедури змінної повинно бути присвоєно значення. Параметри, що повертається, можуть мати будь-який тип даних, крім типів даних text або image

DEFAULT

Надає значення за замовчуванням для параметра, як визначено в процедурі. Коли процедура очікує значення для параметра, який не має визначеного значення за замовчуванням, і або відсутній параметр, або задано ключове слово DEFAULT, виникає помилка.

6 ІНШІ МОДЕЛІ ДАНИХ

Все зростаюча складність додатків баз даних і обмеженість реляційної моделі привели до розвитку моделі Кодда, яка спочатку отримала назву розширеної реляційної моделі, а пізніше отримала свій розвиток в об'єктно-реляційної моделі даних. Бази даних, засновані на цих моделях, прийнято відносити до III-у поколінню.

6.1 Об'єктно-реляційна модель даних

Об'єктно-реляційна модель даних (ОРМД) реалізована за допомогою реляційних таблиць, але включає об'єкти, аналогічні об'єктів в об'єктно-орієнтованому програмуванні. У ОРМД використовуються такі об'єктно-орієнтовані компоненти, як призначені для користувача типи даних, інкапсуляція, поліморфізм, успадкування, перевизначення методів і т.п.

На жаль, до теперішнього часу розробники не прийшли до єдиної думки про те, що повинна забезпечувати ОРМД. У 1999 р був прийнятий стандарт SQL-99, а в 2003 році вийшов другий реліз цього стандарту, який отримав назву SQL-3, який визначає основні характеристики ОРМД. Але до сих пір об'єктно-реляційні моделі, підтримувані різними виробниками СКБД, істотно відрізняються одна від одної. Про перспективи цього напряму свідчить той факт, що провідні фірми-виробники СКБД, в числі яких Oracle, Informix, INGRES і ін., Розширили можливості своїх продуктів до об'єктно-реляційної СКБД (ОРСКБД).

У більшості реалізацій ОРМД об'єктами визнаються агрегат і таблиця (відношення), яка може входити до складу іншої таблиці. Методи обробки даних представлені у вигляді збережених процедур і тригерів, які є процедурними об'єктами бази даних, і пов'язані з таблицями. На концептуальному рівні всі дані об'єктно-реляційної БД представлені у вигляді відносин, і ОРСКБД підтримують мову SQL.

6.2 Об'єктно-орієнтована модель даних

Ще один підхід до побудови БД – використання об'єктно-орієнтованої моделі даних (ООМД). Моделювання даних в ООМД базується на понятті об'єкта. ООМД зазвичай застосовується в складних предметних областях, для моделювання яких не вистачає функціональності реляційної моделі (наприклад, для систем автоматизації проектування (САПР), видавничих систем і т.п.).

При створенні об'єктно-орієнтованих СУБД (ООСУБД) використовуються різні методи, а саме:

- вбудовування в об'єктно-орієнтовану мова засобів, призначених для роботи з БД;
- розширення існуючої мови роботи з базами даних об'єктно-орієнтованими функціями;
- створення об'єктно-орієнтованих бібліотек функцій для роботи з БД;
- створення нової мови і нової об'єктно-орієнтованої моделі даних.

До переваг ООМД можна віднести широкі можливості моделювання предметної області, виразну мову запитів і високу продуктивність. Кожен об'єкт в ООМД має унікальний ідентифікатор (OID – object identifier). Звернення по OID відбувається значно швидше, ніж пошук в реляційної таблиці.

7 ЗАХИСТ І ЦІЛІСНІСТЬ ДАНИХ У СКБД

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

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

7.1 Захист бази даних

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

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

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

Поняття захисту застосовне не тільки до даних, що зберігаються в базі даних. Пробоїни в системі захисту можуть виникати й в інших частинах системи, що, у свою чергу, наражає на небезпеку й саму базу даних. Отже, захист бази даних повинен охоплювати встаткування, що використовується, програмне забезпечення, персонал і власне дані. Для ефективної реалізації захисту необхідні відповідні засоби контролю, які визначаються

конкретними вимогами, що випливають із особливостей системи, що експлуатується. Необхідність захищати дані, яку раніше дуже часто відкидали або якою просто нехтували, у цей час вся ясніше усвідомлюється різними організаціями. Привід для такої зміни настроїв полягає в випадках руйнування комп'ютерних сховищ корпоративних даних, які трапляються все частіше, а також в усвідомленні того, що втрата або просто тимчасова неприступність цих даних може стати причиною дійсної катастрофи.

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

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

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

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

Поняття конфіденційності означає необхідність збереження даних у таємниці. Як правило, конфіденційними вважаються тільки ті дані, які є важливими для всієї організації, тоді як поняття недоторканності даних стосується вимоги захисту інформації про окремих співробітників. Наслідком порушення в системі захисту, що викликав втрату конфіденційності

даних, може бути втрата надійних позицій у конкурентній боротьбі, тоді як наслідком порушення недоторканності особистих даних можуть стати судові дії, початі у відношенні організації.

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

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

7.1.1 Основні типи погроз

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

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

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

ми організації. Будь-яка погроза повинна розглядатися як потенційна можливість порушення системи захисту, що у випадку успішної реалізації може зробити той або інший негативний вплив. У табл. 7.1 наведені приклади різних типів погроз із вказівкою тих областей, у яких вони можуть впливати на систему.

Таблиця 7.1 Приклади можливих погроз

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

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

На рис. 7.1 зображено, які об’єкти та суб’єкти комп’ютерних систем можуть нести погрози для БД.

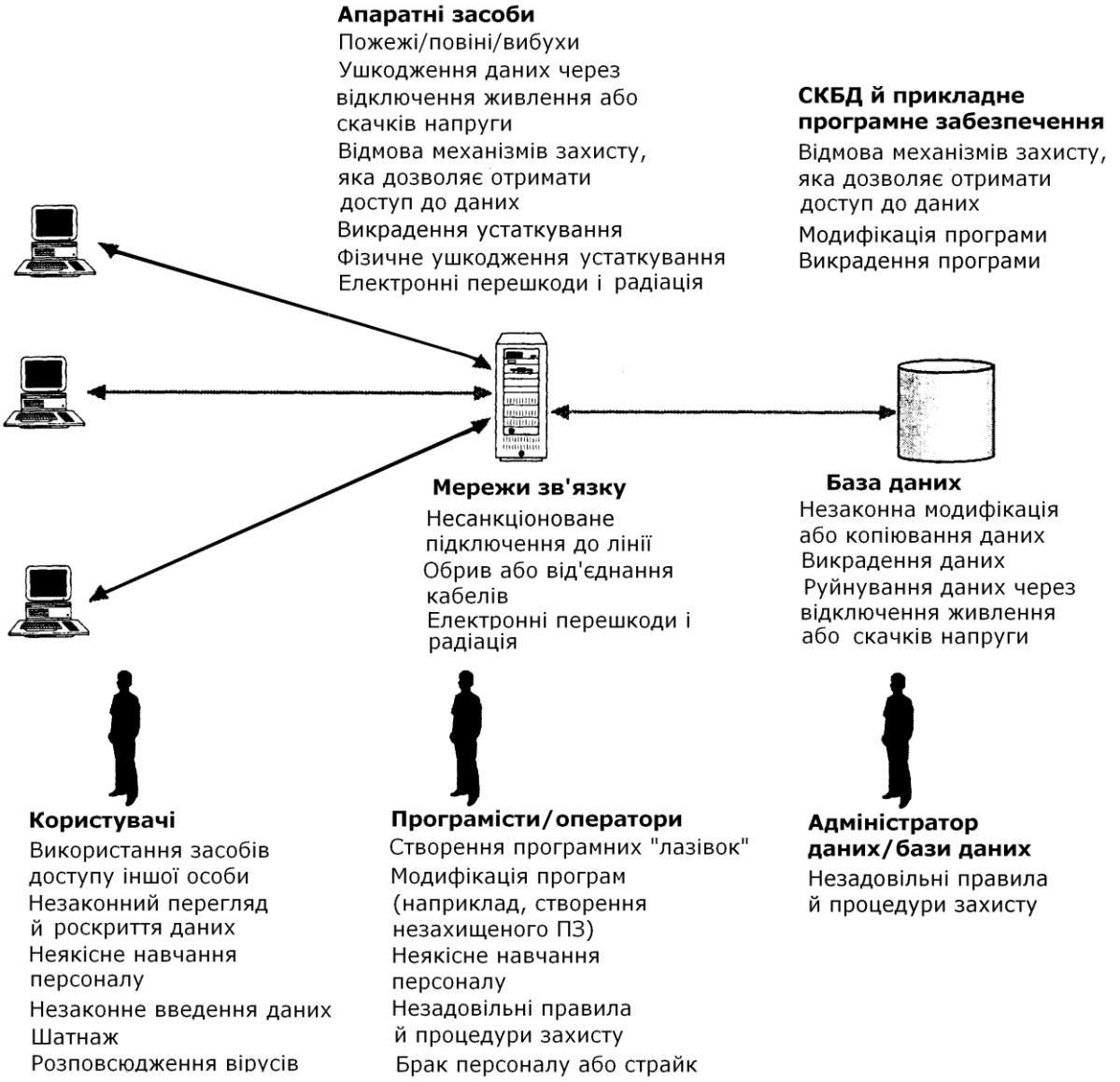


Рисунок 7.1 – Узагальнена схема потенційних погроз, яким піддані комп'ютерні системи

Рівень втрат, понесених організацією в результаті реалізації деякої погрози, залежить від багатьох факторів, включаючи наявність заздалегідь продуманих контрзаходів і планів подолання непередбачених обставин. Наприклад, якщо відмова встаткування викликала руйнування вторинних сховищ даних, вся робота в системі повинна бути згорнута до повного

усунення наслідків аварії. Тривалість періоду бездіяльності й швидкість відновлення бази даних залежать від декількох факторів, включаючи час створення останньої резервної копії й тривалість роботи по відбудові системи.

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

7.2 Контрзаходи – комп'ютерні засоби контролю

Відносно погроз, які можуть зробити негативний вплив на роботу комп'ютерних систем, повинні бути прийняті контрзаходи всіляких типів, починаючи від фізичного контролю й закінчуючи адміністративно-організаційними процедурами. Незважаючи на широкий діапазон комп'ютерних засобів контролю, доступних у цей час на ринку, загальний рівень захищеності СКБД визначається можливостями операційної системи, щ використовується, оскільки робота цих двох компонентів тісно зв'язана між собою. Загальна схема типової комп'ютерної системи багатьох користувачів представлена на рис. 7.2.

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

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

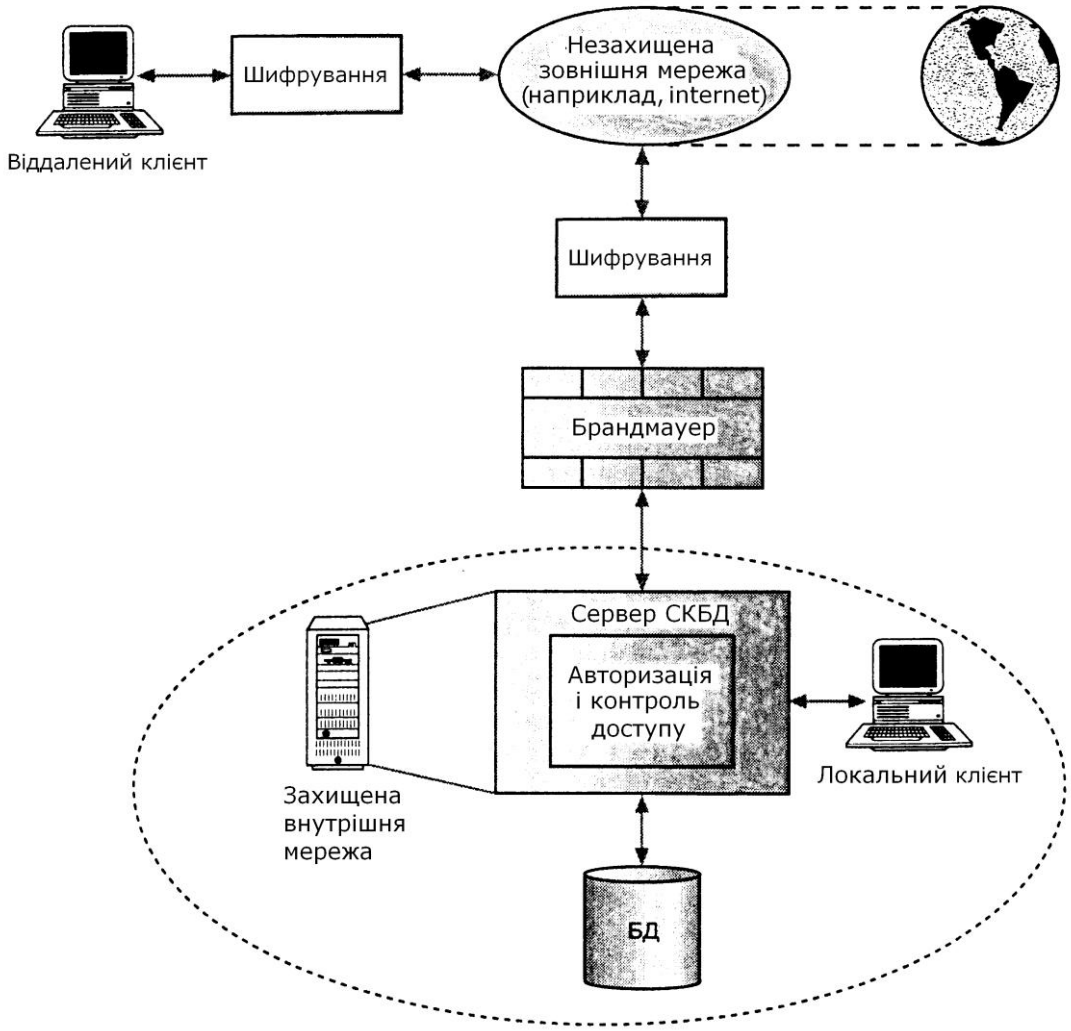


Рисунок 7.2 – Загальна схема типової комп'ютерної системи багатьох користувачів

6.2.1. Авторизація користувачів

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

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

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

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

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

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

Привілеї

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

Деякі типи СКБД функціонують як *закриті системи*, тому користувачам крім дозволу на доступ до самої СКБД буде потрібно мати окремі

дозволи й на доступ до конкретних її об'єктів. Ці дозволи видаються або АБД, або власниками певних об'єктів системи. На противагу цьому, *відкриті системи* за замовчуванням надають користувачам, що пройшли перевірку їхньої дійсності, повний доступ до всіх об'єктів бази даних. У цьому випадку привілею встановлюються за допомогою явного скасування тих або інших прав конкретних користувачів. Методи авторизації користувачів і надання їм привілеїв засобами мови SQL розглядаються в розділі 7.6.

Права володіння й привілеї

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

Найпростішою формою фіксації прав доступу в системі є таблиця, або матриця, прав доступу. Рядки таблиці відповідають різним користувачам, а стовпці – даним. Кожний елемент такої таблиці може зберігати окрему інформацію про права доступу. На рис. 7.3 показана подібна матриця, що не містить предикатів.

Стовпці D_1, ..., D_n можуть відповідати типам записів (файл, відношення, сегмент) або атрибутам. Кожний рядок матриці визначає типи записів або атрибути (поля) даного типу запису, до яких користувачеві дозволений (не дозволений) доступ (маніпулювання).

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

Рисунок 7.3 – Матриця доступу

При використанні предикатів рівень контролю (ступінь деталізації) може досягати окремих значень полів (або множини значень). Наприклад, користувач може не мати права доступу до атрибута ЗАРПЛАТА зі значеннями більше 30000. У більш складних ситуаціях для захисту даних можна використати механізм зовнішніх схем. Наприклад, якщо на рис. 7.3 D_1 ..., D_n позначають атрибути типів запису, то представлення користувача типу запису, оголошений у підсхемі, може бути відбите в матриці доступу. Відповідно до рис. 7.3**,** наприклад, представлення другого користувача, що збігає з його правами доступу, містить тільки атрибут D_2.

Якщо СКБД підтримує кілька різних типів ідентифікаторів авторизації, з кожним з існуючих типів можуть бути зв'язані різні пріоритети. Зокрема, якщо СКБД підтримує використання ідентифікаторів окремих користувачів і груп, тоді, як правило, ідентифікатор користувача буде мати більше високий пріоритет, чим ідентифікатор групи. Приклад визначення ідентифікаторів користувачів і груп у подібної СКБД наведений у табл. 7.2.

Таблиця 12.2. Ідентифікатори користувачів і груп

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

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

SELECT	UPDATE	INSERT	DELETE	ALL
0001	0010	0100	1000	1111

Окремі двійкові значення підсумовуються, і отримана сума однозначно характеризує, які саме привілеї (якщо вони передбачені) має кожний конкретний користувач або група відносно певного об'єкта бази даних. У табл. 7.3 наведений приклад матриці контролю доступу, у якій визначений набір привілеїв групи `Sales` і користувачів `SG37, SG5`.

Таблиця 7.3. Таблиця контролю доступу

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

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

0001 + 0010 + 0100 + 1000 = 1111), тобто привілей `All` для доступу до всіх атрибутів таблиці, а розмір результуючих наборів даних запитів не обмежується.

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

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

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

ПЕРСОНАЛ: ЗАРПЛАТА < 30 000 дол.

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

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

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

специфікація *умова*

Модифікований запит;

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

Предикати правила доступу

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

7.2.2 Представлення (підсхеми)

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

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

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

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

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

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

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

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

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

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

7.3 Резервне копіювання й відновлення

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

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

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

СКБД повинна надавати засоби ведення системного журналу, у якому будуть фіксуватися відомості про всі зміни стану бази даних і про хід виконання поточних транзакцій, що необхідно для ефективного відновлення бази даних у випадку відмови. Переваги використання подібного журналу полягають у тім, що у випадку порушення роботи або відмови СКБД базу даних можна буде відновити до останнього відомого погодженого стану, скориставшись останньою створеною резервною копією бази даних

і оперативною інформацією, що втримується у файлі журналу. Якщо в системі, що відмовила, функція ведення системного журналу не використовувалася, базу даних можна буде відновити тільки до того стану, що було зафіксовано в останній створеній резервній копії. Всі зміни, внесені в базу даних після створення останньої резервної копії, будуть загублені.

6.4 Підтримка цілісності

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

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

Існують різні види обмежень цілісності (правила) :

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

6.5. Шифрування

Шифрування. Перетворення даних з використанням спеціального алгоритму, в результаті чого дані стають недоступними для читання будь-якою програмою, яка не має ключа дешифрування.

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

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

Деякі системи шифрування, що називаються симетричними, використовують той самий ключ як для шифрування, так і для дешифрування, при цьому передбачається наявність захищених ліній зв'язку, призначених для обміну ключами. Однак більшість користувачів не мають доступу до захищених ліній зв'язку, тому для одержання надійного захисту довжина ключа повинна бути не менше довжини самого повідомлення. Проте більшість систем, що експлуатуються, побудовано на використанні ключів, які

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

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

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

6.6. RAID (масив незалежних дискових накопичувачів з надмірністю)

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

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

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

Підвищена надійність RAID-масиву забезпечується завдяки дублюванню даних (такі дублікати називаються *дзеркальними копіями*) і зберіганню на дисках надлишкової інформації, сформованої з використанням схем контролю парності або схем виправлення помилок, таких як код Ріда-Соломона (Reed-Solomon). У схемі контролю парності кожний байт повинен мати пов'язаний з ним біт парності, що приймає значення 0 або 1, залежно від того, чи є парним або непарним кількість бітів 1 у байті, якому відповідає цей біт контролю парності. Якщо в контрольованому байті деякі біти будуть перекручені, значення біта парності не збіжиться зі значенням,

що відповідає новому складу битов 1 у цьому байті. Аналогічним образом, при перекручуванні збереженого біта контролю парності він не буде відповідати даним у байті, що дозволить виявити помилку. З іншого боку, схеми коректування помилок передбачають зберігання двох або більше додаткових битов і дозволяють відновлювати первісні дані, якщо один з бітів буде перекручений. Схеми контролю парності й коректування помилок можуть застосовуватися при смуговому розподілі даних по дисках.

8 ОДНОЧАСНА РОБОТА Й КЕРУВАННЯ ТРАНЗАКЦІЯМИ

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

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

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

8.1 Підтримка транзакцій

Керування одночасною роботою обговорюється окремо, хоча і є складовою частиною підтримки цілісності в СКБД. У зв'язку з керуванням одночасною роботою СКБД розглядаються поняття *транзакція* й *розклад.*

Системна транзакція на відміну від логічної транзакції є елементарною частиною процесу, що включає селекцію (вибірку), модифікацію або перепис у базі даних. Логічна транзакція прикладної програми може бути розбита на кілька системних транзакцій. Ми розглянемо логічну транзакцію, що надалі називається просто транзакцією.

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

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

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

Найпростішою транзакцією, яка виконується в подібній базі даних, може бути коректування зарплати певного працівника, зазначеного його табельним номером `x`. Узагальнено подібна транзакція може бути записана, як показано в табл. 13.1 (варіант А). У цій главі ми будемо позначати операції читання або запису елемента даних `x` за допомогою виразів `read(x)` і `write(x)`. При необхідності до імені елемента даних можуть додаватися додаткові позначення. Наприклад, у стовпці Варіант А (табл. 8.1) використовується позначення `read(staffNo=x,salary)`, що вказує, що потрібно зчитати елемент даних `salary` для запису, у якій ключове значення дорівнює `x`. У даному прикладі транзакція складається із двох операцій, що виконуються у базі даних `(read i write)`, і однієї операції, що виконується поза базою даних `(salary = salary*1.1)`.

Більш складна транзакція, текст якої також показаний у табл. 8.1 (варіант Б), призначена для видалення відомостей про працівника, заданому його табельним номером `x`. У цьому випадку, крім видалення відповідного

рядка з відношення Staff, потрібно знайти всі рядки відношення PropertyForRent, що описують об'єкти нерухомості, за які відповідав даний працівник, після чого призначити їх деякому іншому працівникові, табельний номер якого, припустимо, має значення newStaffNo. Якщо всі зазначені зміни не будуть внесені до кінця, правила посилальної цілісності будуть порушена й база даних виявиться в неузгодженому стані — за об'єкт нерухомості буде відповідати неіснуючий співробітник компанії.

Таблиця 8.1. Приклад виконання транзакції

Варіант А	Варіант Б

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

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

	`end` `end`

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

Будь-яка транзакція завершується одним із двох можливих способів. У випадку успішного завершення результати транзакції *фіксуються* (commit) у базі даних, і остання переходить у новий погоджений стан. Якщо виконання транзакції не увінчалося успіхом, вона *відміняється.* У цьому випадку в базі даних повинне бути відновлене той погоджений стан, у якому вона перебувала до початку даної транзакції. Цей процес називається *відкотом* (roll back), або *скасуванням транзакції.* Зафіксована транзакція не може бути скасована. Якщо виявиться, що зафіксована транзакція була помилкової, буде потрібно виконати іншу транзакцію, що скасовує дії, виконані першою транзакцією (розділ 8.4.2). Така транзакція назива-

ється транзакцією, *що компенсує.* Але транзакція, що завершилася аварійно, для якої виконаний відкіт, може бути викликана на виконання пізніше й, залежно від причин попередньої відмови, цілком успішно завершена й зафіксована в базі даних.

У жодній СКБД не може бути передбачений апріорний спосіб визначення того, які саме операції відновлення можуть бути згруповані для формування єдиної логічної транзакції. Тому повинен застосовуватися метод, що дозволяє вказувати границі кожної із транзакцій ззовні, з боку користувача. У більшості мов маніпулювання даними для вказівки границь окремих транзакцій використовуються оператори `BEGIN TRANSACTION`, `COMMIT` і `ROLLBACK` (або їхні еквіваленти). Якщо ці обмежники не були використані, як єдина транзакція звичайно розглядається вся виконувана програма. СКБД автоматично виконає команду `COMMIT` при нормальному завершенні цієї програми. Аналогічно, у випадку аварійного завершення програми в базі даних автоматично буде виконана команда `ROLLBACK`.

Порядок виконання операцій транзакції прийнято позначати за допомогою діаграми переходів. Приклад такої діаграми наведений на рис. 8.1.

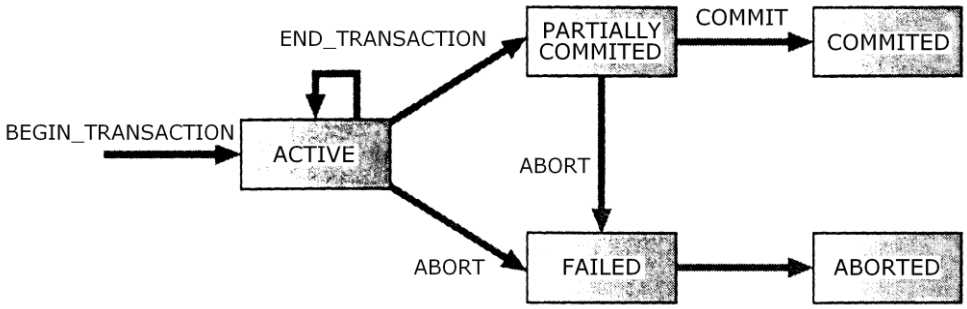


Рисунок 8.1 – Діаграма переходу для деякої транзакції

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

- `PARTIALLY COMMITTED`. Цей стан виникає після виконання останнього оператора. У цей момент може бути виявлено, що в результаті виконання транзакції порушені правила впорядкування або обмеження цілісності, тому транзакцію необхідно завершити аварійно. Ще один варіант розвитку подій полягає в тому, що в системі відбувається відмова й всі дані, обновлені в транзакції, неможливо успішно записати в зовнішню пам'ять. У подібних випадках транзакція повинна перейти в стан `FAILED` і завершитися

аварійно. А якщо транзакція виконана успішно, те всі результати відновлення можуть бути надійно записані в зовнішній пам'яті й транзакція може перейти в стан `COMMITTED`.

- `FAILED`. Такий стан виникає, якщо транзакція не може бути зафіксована або відбулося її аварійне завершення, коли вона перебувала в стані `ACTIVE`. Це аварійне завершення могло виникнути через скасування транзакції користувачем або в результаті дії протоколу керування паралельним доступом, що викликав аварійне завершення транзакції для забезпечення впорядкованності операцій бази даних.

8.1.1 Властивості транзакцій

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

- **Нерозривність**. Це властивість, для опису якого застосовне вираження "все або нічого". Будь-яка транзакція являє собою неподільну одиницю роботи, що може бути або виконана вся цілком, або не виконана взагалі. За забезпечення нерозривності відповідає підсистема відновлення СКБД.
- **Узгодженість**. Кожна транзакція повинна переводити базу даних з одного узгодженого стану в інший. Відповідальність за забезпечення узгодженості покладає й на СКБД, і на розроблювачів додатків. У СКБД узгодженість може забезпечуватися шляхом виконання всіх обмежень, заданих у схемі бази даних, таких як обмеження цілісності й обмеження предметної області. Але подібна умова є недостатнім для забезпечення узгодженості. Наприклад, припустимо, що деяка транзакція призначена для переводу коштів з одного банківського рахунку на інший, але програміст припустився помилки в логіці транзакції й передбачив зняття грошей із правильного рахунку, а зарахування – на неправильний рахунок. У цьому випадку база даних переходить у неузгоджений стан, незважаючи на наявність правильно заданих обмежень. Проте відповідальність за усунення такої неузгодженості не може бути покладене на СКБД, оскільки в ній відсутні які-небудь засоби виявлення подібних логічних помилок.

- **Ізольованість**. Всі транзакції виконуються незалежно одна від одній. Іншими словами, проміжні результати незавершеної транзакції не повинні бути доступні для інших транзакцій. За забезпечення ізольованості відповідає підсистема керування паралельним виконанням.
- **Стійкість**. Результати успішно завершеної (зафіксованої) транзакції повинні зберігатися в базі даних постійно й не повинні бути загублені в результаті наступних збоїв. За забезпечення стійкості відповідає підсистема відновлення.

8.1.2 Архітектура бази даних

На рис. 8.2 представлений фрагмент повної структурної схеми СКБД, що включає чотири високорівневі модулі бази даних, які відповідають за обробку транзакцій, керування паралельним доступом і відновлення системи.

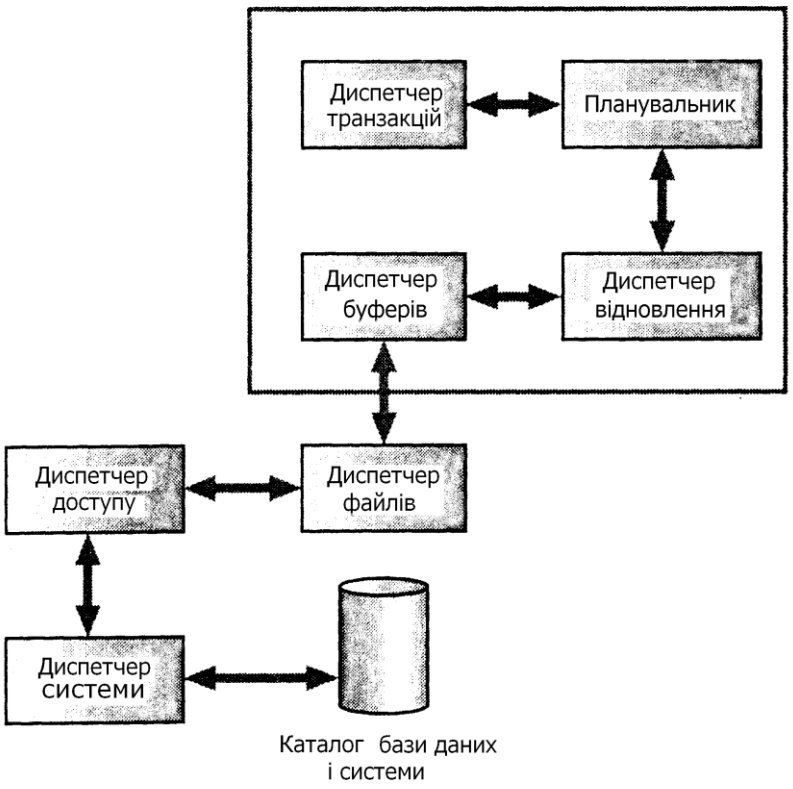


Рисунок 8.2 – Підсистема обробки транзакцій типової СКБД

Диспетчер транзакцій координує роботу транзакцій по запитах від прикладних програм. Він взаємодіє із *планувальником,* відповідальним за реалізацію обраної стратегії керування паралельним доступом. У деяких випадках планувальник називають *диспетчером блокувань,* якщо ви-

користовуваний протокол керування паралельним виконанням будується на основі системи блокувань. Ціль роботи планувальника складається в досягненні максимально можливого рівня розпаралелювання роботи, за умови виключення впливу транзакцій, що виконуються паралельно, одна на одну, оскільки це може послужити джерелом порушення цілісності або узгодженості бази даних.

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

8.2 Керування паралельним доступом

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

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

8.2.1 Необхідність керування паралельним доступом

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

Дане завдання подібне до завдань, що постають перед будь-який многопользовательской комп'ютерною системою, коли кілька програм (або транзакцій) одержують можливість одночасно виконувати операції завдяки використанню мультипрограмної організації роботи, що дозволяє двом або декільком програмам (або транзакціям) виконуватися в тей са-

мий час. Наприклад, багато систем включають підсистему введення-виводу, здатну виконувати операції введення-виводу, у той час як процесор здійснює інші операції. Подібні системи дозволяють двом або декільком транзакціям виконуватися одночасно. Система починає виконання першої транзакції й продовжує її виконання до першої операції введення-виводу. На час виконання цієї операції система припиняє виконання першої транзакції й переходить до виконання команд другої транзакції. Коли другої транзакції знадобиться виконати операцію введення-виводу, керування буде повернуто першої транзакції і її виконання буде продовжено з тієї точки, у якій вона була припинена. Виконання першої транзакції буде продовжено до досягнення наступної операції введення-виводу. Таким чином, виконання операцій двох транзакцій чергується й забезпечується їхнє паралельне виконання. Крім того, загальна продуктивність системи (обсяг роботи, виконуваної протягом заданого тимчасового інтервалу) підвищується, оскільки процесор виконує команди іншої транзакції, замість того щоб даремно простоювати, очікуючи завершення запущеної операції введення-виводу.

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

Приклад 8.1. Проблема загубленого відновлення

Результати цілком успішно завершеної операції відновлення однієї транзакції можуть бути перекриті результатами виконання іншої транзакції. Це порушення відомо як *проблема загубленого відновлення*. Суть її проілюстрована прикладом, наведеним у табл. 8.2. У цьому прикладі транзакція T_1 виконується паралельно із транзакцією T_2. Транзакція T_1 полягає в знятті 10 фунтів стерлінгів з рахунку bal_x, на якому спочатку перебуває 100 фунтів стерлінгів, а транзакція T_2 припускає зарахування 100 фунтів стерлінгів на цей же рахунок. Якщо ці транзакції будуть виконуватися послідовно, одна за іншою, без чергування їхніх операцій, то після завер-

шення роботи на рахунку буде перебувати сума в 190 фунтів стерлінгів, незалежно від порядку виконання транзакцій. Але допустимо, що транзакції T_1 і T_2 починаються практично одночасно й кожна з них зчитує вихідне значення суми на рахунку, рівним 100 фунтам стерлінгів. Потім транзакція T_2 збільшує суму на 100 фунтів стерлінгів і записує результат, рівним 200 фунтам стерлінгів, у базу даних – на рахунок bal_x. Тим часом транзакція T_1 зменшує значення своєї копії початкової суми на рахунку bal_x і записує отримане значення, рівне 90 фунтам стерлінгів, у базу даних на той же рахунок, перекриваючи результат попереднього відновлення. У результаті з балансового рахунку "зникає" 100 фунтів стерлінгів, доданих при виконанні попередньої операції. Можна уникнути втрати результатів виконання транзакції T_2, заборонивши транзакції T_1 зчитувати вихідне значення на рахунку bal_x аж до завершення виконання транзакції T_2. Ці проблеми знімаються при блокуваннях транзакцій за рахунок двофазного протоколу захвата (див. розділ 8.3).

Таблиця 8.2. Приклад проблеми загубленого відновлення

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

Приклад 8.2. Проблема залежності від незафіксованих результатів (або "брудного" читання)

Проблема залежності від незафіксованих результатів виникає в тому випадку, якщо одна із транзакцій одержить доступ до проміжних результатів виконання іншої транзакції до того, як вони будуть зафіксовані в базі даних. У табл. 13.3 наведений приклад залежності від незафіксованих результатів, що викликає появу помилки. У цьому прикладі використовуються ті ж початкові дані для залишку на рахунку bal_x, що й у попередньому прикладі. У цьому випадку транзакція T_4 збільшує значення суми на рахунку bal_x до 200 фунтів стерлінгів, після чого виконання транзакції відміняється, тому СКБД повинна виконати відкіт транзакції з відновленням початкового значення на рахунку, рівним 100 фунтам стерлінгів. Од-

нак до цього моменту транзакція T_3 уже встигла зчитати змінене значення рахунку bal_x (200 фунтів) і використала саме це значення при виконанні операції зняття 10 фунтів стерлінгів з рахунку, після чого зафіксувала в базі даних невірний результат, рівним 190 фунтам стерлінгів (замість правильного — 90 фунтів стерлінгів). Значення bal_x, зчитане в транзакції T_3, називається брудними даними. Від цього терміна походить друга назва розглянутої проблеми – проблема брудного читання.

Таблиця 8.3. Приклад проблеми залежності від незафіксованих результатів

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

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

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

Приклад 8.4. Проблема аналізу непогодженості

Проблема аналізу неузгодженості виникає в тих випадках, коли транзакція зчитує кілька значень із бази даних, після чого друга транзакція

обновляє деякі із цих значень безпосередньо під час виконання першої транзакції. Наприклад, транзакція, що підсумовує дані, обрані з бази (скажемо, що обчислює загальну суму на рахунках), одержить невірне значення, якщо під час її виконання інша транзакція змінить значення, які вона зчитала. Приклад подібної помилки наведений у табл. 13.4. Тут транзакція T_6, що обчислює підсумкове значення, виконується паралельно із транзакцією T_5. Транзакція T_6 обчислює суму залишків на рахунках x (100 фунтів стерлінгів), y (50 фунтів стерлінгів) і z (25 фунтів стерлінгів). Однак у цей же час транзакція T_5 здійснює перевод десяти фунтів стерлінгів з рахунку bal_x на рахунок bal_z. У результаті обчислене транзакцією T_6 значення виявляється невірним (більше на 10 фунтів стерлінгів). Цю проблему можна усунути, заборонивши транзакції T_6 зчитувати значення на рахунках bal_x і bal_z доти, поки транзакція T_5 не зафіксує виконані нею відновлення.

Таблиця 8.4. Приклад проблеми усунення непогодженості

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

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

двома операціями читання. Таку проблему іноді називають *фантомним читанням*.

8.3 Розклади при паралельній роботі

Розклад або *графік* при одночасній роботі представляє порядок виконання транзакцій у часі. Можна дослідити ряд розкладів, наприклад представлених на рис. 8.3, позначаючи зчитування й запис поля A відповідно R-A і W-A. (Надалі W буде інтерпретуватися як модифікація вхідними змінними).

У першому розкладі (рис. 8.3) не може виникнути неузгодженості бази даних, тому що транзакції T_1 і T_2 виконуються послідовно, одна за іншою.

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

Рисунок 8.3 – Можливі розклади.

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

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

Для виявлення розкладів, що приводяться до послідовного й завдяки цьому що зберігають узгоджений стан бази даних, можна використати ме-

тод, заснований на побудові графа залежностей. У цьому графі транзакціям відповідають вузли, а спрямована дуга графа, що зв'язує вузли T_i і T_j, позначає, що вхід T_j залежить від виходу T_i, тобто T_j використовує значення поля, вироблене транзакцією. T_i Якщо граф залежностей містить цикли, відповідний розклад не приводиться до послідовного й, отже, дає суперечливі результати. На рис. 8.4 показані три розклади з відповідними графами залежностей. Розклад S_1 – послідовне (несуперечливе), S_2 – приводиться до послідовного (несуперечливе), а S_3 – циклічне (суперечливе).

Монопольне використання значення поля в процесі роботи СКБД можна забезпечити за допомогою блокування (захвата). У цьому випадку ніяка інша транзакція не одержить доступу до поля й не зможе його використати (для читання або запису) доти, поки поле не буде звільнено. Було показано, що узгодженість бази даних можна аналізувати, досліджуючи графів передування, побудовані для послідовностей захватів і звільненні. Передбачається, що будь-яка транзакція може бути представлена у вигляді послідовності ЗАХВАТ-ДІЯ-ЗВІЛЬНЕННЯ. Для наочності будуть, однак, використовуватись тільки пари ЗАХВАТ-ЗВІЛЬНЕННЯ. Дуга графа передування, що з'єднує вузли T_i і T_j, означає, що T_j захоплює поле після його звільнення транзакцією T_i, оскільки в послідовному розкладі для виконання транзакції T_j, що вимагає захвата поля транзакція T_i, що передує, повинна звільнити це поле. Для перевірки розкладу на зведення до послідовного необхідно побудувати граф передування. Приклад представлений на рис. 8.4.

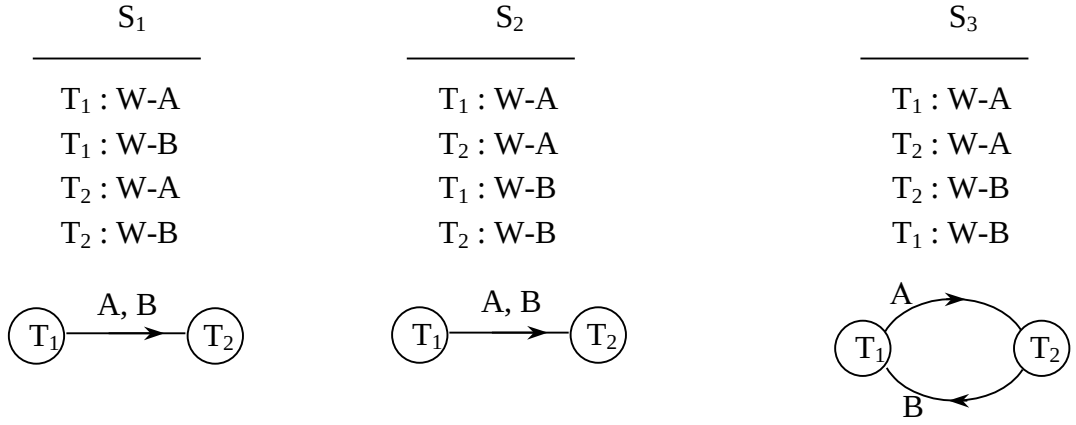


Рисунок 8.4 – Три розклади й графи їхніх залежностей

При побудові графа в розкладі вибираються вузли, що відповідають ЗВІЛЬНЕННЯМ, які попарно зв'язуються з наступними вузлами ЗАХВАТІВ (вони розташовані не обов'язково безпосередньо один за одним, що видно на прикладі пари ЗВІЛЬНИТИ С и ЗАХОПИТИ С на рис.

8.5). Граф передування так само, як граф залежностей, не повинен містити циклів.

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

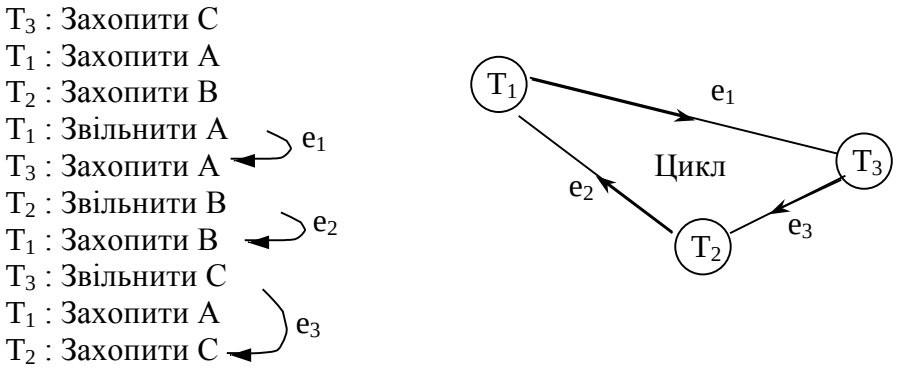


Рисунок 8.5 – Розклад і відповідний граф передування

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

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

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

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

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

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

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

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

Для подальшого розширення можливостей розкладів можна використати ідею про те, що кожний захват на запис має на увазі зчитування полів і при модифікації зчитаних полів записується нове значення. Можна припустити, що транзакції зчитують деякі поля й записують їх назад і не кожне зчитане поле модифікується, хоча воно й може використовуватись при обчисленні нових значень інших полів. У графах предшествования таких розкладі можуть використовуватись транзакції, які називаються *марними,* тобто не мають вихідних дуг. Подібні марні вузли можна видалити із графа. Однак при припущенні незалежності записів має місце наступне: нехай дан розклад, у якому спочатку T_1 записує поле, а потім T_2 зчитує його (так що можна провести дугу з T_1 в T_2). Якщо транзакція T_3 модифікує це ж поле, то вона повинна з'являтися або до T_1, або після T_2, але не між ними. Це приводить до побудови двох можливих дуг у графі, що називається *поліграфом* (тобто графом з декількома дугами між парами вершин).

(Відзначимо, що, відповідно до останнього припущення, транзакція може здійснити захвата на запис поля, значення якого не зчитувалося, якщо до захвата на запис не було захвата цього поля на читання.) Після усунення марних вузлів для перевірки на зведення до послідовного розкладу граф варто спростити шляхом видалення однієї з пари дублюючих дуг. Існує значне число можливих варіантів (2^n для n пари дуг), серед яких перебуває результат. Тут також може застосовувати двофазний протокол захвату.

9 РОЗПОДІЛЕНІ БАЗИ ДАНИХ

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

9.1 Централізовані й децентралізовані СКБД

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

- великий потік обмінів даними;
- низька надійність;
- низька загальна продуктивність;
- більші витрати на розробку.

Хоча в централізованій базі даних легше забезпечити безпека, цілісність і несуперечність інформації при відновленнях, перераховані проблеми створюють певні труднощі. Як можливе рішення цих проблем пропонується децентралізація даних. При децентралізації досягається:

- більше високий ступінь одночасності обробки внаслідок розподілу навантаження;
- поліпшене використання даних на місцях при виконанні віддалених (дистанційних) запитів;
- менші витрати;
- простота керування.

Витрати на створення мережі, у вузлах якої перебувають малі ЕОМ, набагато нижче, ніж витрати на створення аналогічної системи з використанням великий ЕОМ. На рис.9.1 наведена логічна схема розподіленої бази даних. Розглянемо її докладніше. Припустимо, що існує система обробки банківських операцій на основі мережі. Клієнт у місті А працює із системою за допомогою локальних запитів (географічний принцип). Якщо клієнт переїжджає в місто Б и хоче в банку цього міста одержати свій внесок, для перевірки його рахунку в банку міста А буде виданий віддалений запит. Крім того, більшість запитів головної установи є віддаленими. На їхній основі будується протокол роботи всієї системи. Між банками деякого

району можуть також здійснюватися обміни інформацією на основі локальних запитів, наприклад кредитні операції з фермерами цього району (функціональний принцип).

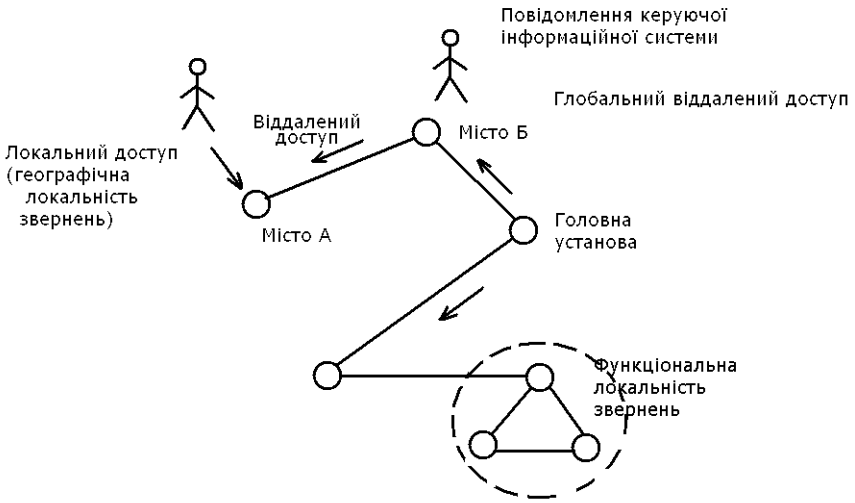


Рисунок 9.1 – Моделі доступу в розподілених базах даних.

Розподілена база даних. Це набір файлів (відношень), що зберігаються в різних вузлах інформаційної мережі й логічно зв'язаних таким чином, щоб становити єдину сукупність даних (зв'язок може бути функціональної або через копії того самого файлу).

Проблеми розподілених баз даних

Основне завдання розподіленої бази даних – розподіл даних по мережі. Існують наступні способи рішення цього завдання:

а) У кожному вузлі зберігається й використовується власний набір даних, доступний для віддалених запитів. Такий розподіл називається *розділеним.*

б) Деякі дані, що часто використовуються на віддалених вузлах, можуть дублюватися. Такий розподіл називається *частково дубльованим.*

в) Всі дані дублюються в кожному вузлі. Такий розподіл називається *повністю дубльованим.*

г) Деякі файли можуть бути розщеплені горизонтально (виділена підмножина записів) або вертикально (виділена підмножина полів (атрибутів)), при цьому виділені підмножини зберігаються в різних вузлах разом з нерозщепленими даними (п.п. а) – в)). Такий розподіл називається *розщепленим (фрагментованим*).

д) Виділені підмножини можуть дублюватися, як у пп. б) і в).

Розглянемо тепер інші проблеми розподілених баз даних. Проблеми централізованих СКБД існують і тут, однак, децентралізація додає нові:

а) Яка загальна модель даних розподіленої системи? Ми повинні мати єдину концептуальну схему всієї мережі. Це забезпечить *логічну прозорість* даних для користувача, у результаті чого він зможе формувати запит до всієї бази, перебуваючи за окремим терміналом (тобто як би працюючи із централізованою базою даних).

б) Необхідна схема, що визначає місцезнаходження даних у мережі. Це забезпечить *прозорість розміщення* даних, завдяки якій користувач може не вказувати, куди переслати запит, щоб одержати необхідні дані.

в) Розподілені бази даних можуть бути однорідними або неоднорідними в сенсі апаратних і програмних засобів (СКБД). Проблему неоднорідності порівняно легко вирішити, якщо розподілена база є неоднорідною в сенсі апаратних засобів, але однорідної в сенсі програмних засобів (однакові СКБД у вузлах). Якщо ж у вузлах розподіленої системи використовуються різні СКБД, необхідні засоби перетворення структур даних і мов. Це повинна забезпечити *прозорість перетворення* у вузлах розподіленої бази даних.

г) Керування словниками. Для забезпечення всіх видів прозорості в розподіленій базі даних потрібні програми, що управляють численними довідниками або словниками.

д) Методи виконання запитів у розподіленій базі даних відрізняються від аналогічних методів централізованих СКБД, тому що окремі частини запиту потрібно виконувати на місці розташування відповідних даних і передавати часткові результати на інші вузли; при цьому повинна бути забезпечена координація всіх процесів.

е) У розподіленій базі даних потрібний складний механізм керування одночасною обробкою, що, зокрема, повинен забезпечувати синхронізацію при відновленнях інформації, що гарантує несуперечність даних.

ж) Розвинена методологія розподілу й розміщення даних, включаючи розщеплення, є одним з основних вимог до розподіленої бази даних.

Приклад 1. Розглянемо базу даних, розподілену по трьох вузлах, схема якої складається із трьох відношень:

ЩО СЛУЖАТЬ (СЛЖ#, ВІДД#, ЗАРПЛАТА, КЕРІВНИК)

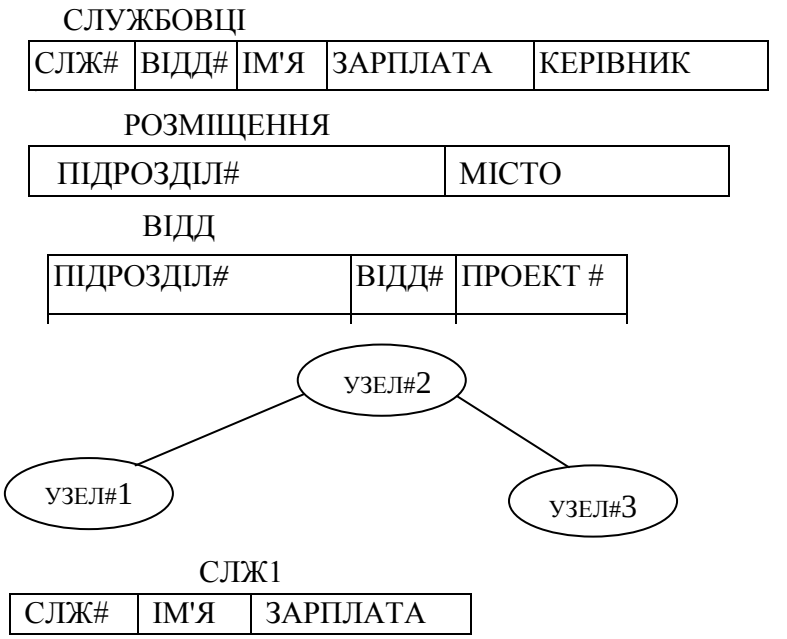
ВІДД (ПІДРОЗДІЛ#, ВІДД#, ПРОЕКТ#)

РОЗМІЩЕННЯ (ПІДРОЗДІЛ #, МІСТО)

На рис. 9.2 зображена схема бази даних, схема географічного розташування вузлів, та можлива схема розподілу бази даних.

Перша стратегія розподілу. Припустимо, що у вузлі #1 обробляються дані про зарплату й податки. Для цього потрібні атрибути СЛЖ#, ІМ'Я й ЗАРПЛАТА. У вузлі 2 перебуває головна установа й зберігаються повні копії всіх відношень СЛУЖБОВЦІ, ВІДД, РОЗМІЩЕННЯ. У вузлі #3 зосереджене виробництво й зберігаються повні дані про службовців, що заробляють менше 45 000 дол. у рік; про інших службовців зберігаються тільки відомості про номер відділу й керівника (рис. 9.2).

Відношення СЛЖ1 отримується операцією проекції, СЛЖ3 – операцією селекції, за якою виконується проекція. Для правильного відновлення повних кортежів у кожному відношенні є ключі СЛЖ#. У той час як у вузлі 2 зберігаються повні відношення, у вузлі 1 – вертикальний фрагмент відношення СЛУЖБОВЦІ а у вузлі 3 – горизонтальний і змішаний фрагменти того ж відношення. Змішаний фрагмент – результат двох процесів: горизонтального, а потім вертикального розщеплення відношення СЛУЖБОВЦІ.



Предикат: СЛЖ1← $\Pi_{\text{СЛЖ\#, ІМ'Я, ЗАРПЛАТА}}$ (СЛУЖБОВЦІ)

СЛЖ2

СЛЖ#	ВІДД#	ім'я	зарплата	керівник

Предикат: СЛЖ2← $\sigma_{\text{ЗАРПЛАТА}<45000}$ (СЛУЖБОВЦІ)

СЛЖ3

СЛЖ#	ВІДД#	КЕРІВНИК

Предикат: СЛЖ3← $\Pi_{\text{СЛЖ\#, ВІДД\#, КЕРІВНИК}}$ (СЛУЖБОВЦІ)

Рисунок 9.2 – Приклад фрагментації й розподілу даних.

Друга стратегія розподілу. Припустимо тепер, що у вузлі 1 перебуває відношення СЛЖ1, що містить відомості тільки про службовців, що працюють у підрозділі 5.

Модифікований предикат схеми вузла 1:

$$\text{СЛЖ1} \leftarrow (\Pi_{\text{СЛЖ\#, ІМ'Я, ЗАРПЛАТА}}(\text{СЛУЖБОВЦІ})) \succ (\sigma_{\text{ПІДРОЗДІЛ\# =5}}(\text{ВІДД}))$$

Відновлення в обох варіантах розміщення впливає істотно, якщо при відновленні використовуються атрибути, що становлять основу предиката схеми. Для збереження несуперечності схеми в системі потрібно зробити перетворення даних. Це перетворення здійснюється шляхом включення в потрібних місцях нових кортежів (фрагментів) і видалення старих.

9.2 Виконання запитів у розподіленій базі даних

З виконанням запитів у розподіленій базі даних зв'язано кілька проблем. Оскільки дані розподілені, а також можуть бути розщеплені, запити, складені користувачем, що розглядає дані цілком (як якби база даних була централізованою), повинні бути наведені до виду, що враховує розподіл і розщеплення даних. Розподіл і розщеплення вимагають декомпозиції запиту, тому що програма запиту не може бути виконана повністю в одному вузлі, якщо там не зберігається вся необхідна інформація. Крім того, оскільки для завершення виконання програми необхідно переміщати цілі відношення (файли) або результати проміжних обчислень, для зниження витрат на передачу даних необхідно оптимізувати:

а) вирази запитів (програму);

б) методику виконання запиту з урахуванням необхідних переміщень даних.

9.3 Оптимізація запитів

Оптимізація запитів ґрунтується на перестановці операцій у межах запиту, на ідентифікації загальних підвиразів і однократному їхньому виконанні, на трансляції й оптимізації запиту щодо фрагментів.

При оптимізації запиту починають із побудови дерева розбору для виразу запиту. У цьому дереві листи відповідають відношенням, а внутрішні вузли (включаючи корінь) – операціям реляційної алгебри. Розглянемо наступний запит: *як звуть службовців, що працюють над проектом* #5, і припустимо, що відповідний вираз реляційної алгебри має вигляд:

$$\Pi_{\text{ІМ'Я}}(\text{СЛУЖБОВЦІ} \succ (\sigma_{\text{проект\# =5}}(\text{ВІДД})))$$

Порядок операцій у цьому виразі – зліва праворуч. З урахуванням дужок маємо перехід від внутрішніх до зовнішніх підвиразів. Зазначений порядок, що відповідає обходу дерева «знизу нагору», не є найкращим з погляду витрат на обробку на місцях, тобто виконання операцій у вузлах і передачі даних. Передача даних відповідає галузям зі ступенем розгалуження більше одиниці в дереві запиту для відношень, що зберігаються в різних вузлах. Розглянутий запит можна оптімизувати із урахуванням двох відношень еквівалентності:

- *Каскадність* унарних операцій показує, що задана унарна операція може бути розбита на частині, що виконуються послідовно. Наприклад, операцію вибірки з відношення кортежів, які одночасно відповідають умові типу $A\ \theta\ c$, для яка задана для двух окремих атрибутів, тобто

  $A_1\ \theta\ c_1$ AND $A_2\ \theta\ c_2$ можна розбити на дві операції вибірки з простою умовою:

  замість $\sigma_{A_1\theta\, c_1 \text{ AND } A_2\theta\, c_2}(R)$ написати

  $R_1 \leftarrow \sigma_{A_1\theta\, c_1}(R)$

  $\sigma_{A_2\theta\, c_2}(R_1)$

- *Дистрибутивність* унарних операцій щодо бінарних дозволяє поширити їх на операнди бінарних операцій. Наприклад:

$$R \rtimes_{A\,\theta\,c} S \equiv (\sigma_{A\,\theta\,c}(R)) \rtimes (\sigma_{A\,\theta\,c}(S))$$

де R і S – відношення, A – атрибути, c — константа, θ — оператор порівняння.

Можливо більш раннє застосування операцій селекції приводить до оптимального виконання запиту. Це відповідає переміщенню унарних операцій у напрямку до листів дерева. (Унарні операції – це операції селекції й проекції, що зменшують розміри відношення по горизонталі й вертикалі відповідно.)

9.4 Одночасна обробка й відновлення

У розподілених базах даних можуть одночасно виконуватися кілька транзакцій; при цьому можуть використовуватись ті самі дані, можливо, продубльовані в різних вузлах. При читанні для одержання потрібної інформації досить однієї копії, якщо система забезпечує несуперечність всіх копій. При відновленнях, однак, для збереження несуперечності копій потрібно їх вчасно модифікувати.

Як ми вже знаємо, відновлення потрібно робити відповідно до послідовних протоколів. Якщо враховувати одночасне виконання транзакцій над численними копіями, послідовність відновлень вимагає розвинених засобів синхронізації. Це приводить до компромісних рішень при керуванні відновленнями в розподіленій базі даних. Наприклад, у схемі з *основним вузлом* (або основною копією) для кожного відношення вибирається основний вузол, у якому виконується відновлення інформації. Після успішного завершення відновлення в цьому вузлі починається відновлення вторинних копій для повного завершення даної операції відновлення. У цьому випадку основна проблема при виконанні транзакції – забезпечити відповідність даних, уже обновлених і ще не обновлених. Більш того, первинний вузол, будучи центральним місцем відновлення, впливає на надійність всієї системи. Необхідно також вибрати додаткові вузли на випадок збоїв.

При запитах, що вимагають тільки зчитування з бази даних, зручним і недорогим засобом забезпечення одночасного обслуговування є «моментальні знімки» основних відносин. Їх можна продублювати там, де це необхідно. Такі знімки робляться (тобто їхні дані засилаються в різні вузли) тоді, коли зручніше за все робити відновлення бази даних (тобто в менш напружені години). До наступного відновлення ці знімки обслуговують користувачів (хоча дані в них старі), і вони не піддані впливу відновлення основних відношень. Можна обмежитися такими компромісами, однак наша мета – обговорити методологію синхронізації транзакцій, про що мова йтиме нижче.

9.5 Блокування в розподілених базах даних

При роботі із блокуваннями, як ми вже говорили, транзакція читання блокує одну копію, а транзакція відновлення повинна блокувати всі копії. Транзакція може прочитати елемент даних, якщо будь-яка копія, що містить цей елемент, блокована по читанню, і обновити його, якщо всі копії, що містять цей елемент даних, блоковані по запису. Блокування по читанню буде забезпечена доти, поки інша транзакція не встановить блокування по запису. Інакше кажучи, для того самого елемента даних не можуть існувати одночасно блокування по читанню й запису.

Як відзначалося, двофазний протокол забезпечує послідовність блокувань. Спочатку повинні бути оброблені всі блокування, а потім можна здійснити розблокування, тобто

- перед виконанням читання необхідно забезпечити блокування по читанню, щоб заборонити блокування по запису;

- перед відновленням необхідно забезпечити блокування по запису всіх копій, що містять необхідний елемент даних;
- після установки блокування його скасування не допускається до завершення транзакції.

У розподілених базах даних необхідно забезпечити передачу повідомлень про блокування: спочатку послати запити на блокування в усі вузли, потім одержати підтвердження. Після завершення операції потрібно послати запити на зняття блокувань в усі вузли, де зберігається інформація.

У схемі з основним вузлом установка й зняття блокувань для всіх транзакцій відбуваються для основного вузла. Коли надходить запит на блокування, перевіряється, чи не заблокований елемент даних, що перебуває в основному вузлі, іншою транзакцією. Якщо елемент не заблокований, він блокується. У цій схемі потрібно мати графи передування або очікування блокувань для виявлення тупикових ситуацій. Крім того, при цій схемі значно зменшується потік повідомлень про блокування, оскільки використається лише один основний вузол для установки й зняття блокувань.

У розподілених базах даних заблокувати можна один основний вузол або кілька вузлів при синхронізації розподілених відновлень. В останньому випадку потрібні графи очікування блокувань для всієї мережі. (У такому графі вузли є транзакції, а ребра – дані. Спрямоване ребро виходить із вузла, що запитує блокування деякого елемента даних, і йде до вузла, у якому ці дані в сучасний момент заблоковані. Цикл у такому графі позначає тупикову ситуацію.)

У кожному вузлі перебуває свій локальний граф очікування блокувань, у якому зв'язок з іншою частиною мережі представлений спеціальною вершиною. У цій схемі можлива глобальна тупикова ситуація, хоча жоден з локальних графів не містить циклу. Для того щоб виявити подібну ситуацію, вузли зв'язують свої локальні графи попарно через спеціальні зовнішні вершини доти, поки не буде виявлена тупикова ситуація. При синхронізації розподілених відновлень на основі блокувань у центральному вузлі може перебувати загальний граф очікування блокувань. У цьому випадку всі вузли посилають запити на установку й зняття блокувань у центральний вузол.

9.6 Часові мітки

При використанні часових міток виключається можливість тупикової ситуації. У цьому випадку одночасне виконання транзакцій еквівалентно деякому послідовному виконанню, що визначається часовими мітками.

Хоча тупикові ситуації виключаються, можуть виникнути надлишкові рестарти й відмови.

При часовому маркіруванні кожної транзакції при вході в мережу привласнюється унікальна мітка. У якості мітки може використовуватись показання годиника глобальної мережі, доповнене ідентифікатором вузла. Кожний елемент у базі даних має часові мітки останніх виконаних над ним транзакцій читання й відновлення. Ці мітки називаються мітками читання й запису відповідно. Якщо транзакція T_1 запитує операцію, що вступає в конфлікт із операцією, яка уже виконується відповідно до більш пізньої за часом транзакцією T_2, то T_1 запускається знову. Конфлікт між T_1 і T_2 виникає, якщо операція транзакції T_1 є операція читання, але об'єкт уже записаний транзакцією T_2 (випадок 1), або операція запису, а об'єкт уже був прочитаний або записаний транзакцією T_2 (випадок 2). Якщо транзакція перезапускається, їй привласнюється нова часова мітка.

Итак, якщо t - *часова* мітка транзакції, а t_r і t_w — часи читання й запису елемента даних, то необхідно:

а) виконати читання, якщо $t \geq t_w$, або запис, якщо $t \geq t_r$ і $t \geq t_w$; у першому випадку час читання встановити рівним t, якщо $t > t_r$, у другому випадку час запису встановити рівним t, якщо $t > t_w$;

б) нічого не робити, якщо транзакція виконує запис і $t_r < t < t_w$;

в) видати відмову, якщо транзакція виконує читання й $t < t_w$ або транзакція виконує запис і $t < t_r$.

Хоча система з часовими мітками має певні переваги, вона може викликати надлишкові перезапуски й відмови. Для зниження цього ефекту використовується система *консервативних часових міток,* при якій події відбуваються в строгій часовій послідовності й вузли обмінюються запитами також у строгій часовій послідовності.

9.7 Системи з голосуванням

Синхронізацію транзакцій на відновлення можна забезпечити, якщо дати можливість кожному вузлу мережі «голосувати» на користь відновлення. У деяких алгоритмах запит на відновлення виконується, якщо всі взаємодіючі вузли голосують за виконання цього запиту. Така система називається *системою синхронізації з одноголосним голосуванням.* Також використовується *мажоритарний* принцип, тобто досить, щоб за проведення відновлення проголосувало більшість вузлів. Ця система заснована на тім, що жоден вузол не може одночасно голосувати за виконання двох конфлі-

ктуючих транзакцій. При наявності запиту на відновлення вузол може видати один з наступних сигналів:

а) ОК для проведення відновлення;

б) REJ для відхилення відновлення;

в) PASS для позначення, що можливо тупикову ситуацію;

г) вузол може втриматися від голосування по даному запиті.

Якщо поточний запит суперечить запиту, по якому вже була отримана більшість голосів, вузол повинен відхилити поточний запит. Якщо запит не суперечить іншим запитам у цьому вузлі, він приймається. Може виникнути така ситуація, коли поточний запит суперечить попередньому запиту, по якому вузол проголосував ОК (так званий припинений запит). Запит називається *припиненим,* якщо результат голосування по ньому ОК, але системою він ще не прийнятий. Якщо пріоритет поточного запиту нижче, ніж припиненого, вузол видає сигнал PASS, у противному випадку він утримується від голосування, але вертається до цього запиту пізніше. Сигнал PASS означає, що можливо тупикову ситуацію. При мажоритарному голосуванні по якому-небудь запиту недостатньо одержати сигнали ОК більш ніж від половини вузлів. Потрібно також, щоб жоден вузол не голосував проти при оцінці результатів голосування. Така система досить стійка, оскільки збій в одному вузлі не впливає істотно на операцію, тому що для її виконання потрібно опитати тільки більшість вузлів.

Відомі й інші схеми. Наприклад, використовується попередній аналіз: транзакції заздалегідь класифікують і складають граф конфліктів, для того, щоб мінімізувати труднощі синхронізації на етапі виконання. Однак така система буде статичної, тому що аналіз транзакцій, їхня класифікація й побудова графів конфліктів у процесі виконання зажадали б занадто великих обчислювальних витрат.

Для всіх систем синхронізації, як тільки синхронізація виконана, і можна проводити відновлення, говорять, що транзакція вступила у фазу фіксації. Як уже говорилося, процес фіксації вимагає протоколу. Розглянутий двофазний протокол фіксації придатний також для розподілених баз даних. Запити й підтвердження проходять через вузли попарно. Відновлення відбувається тільки в тому випадку, якщо всі вузли згодні на фіксацію і якщо запит у відповідному вузлі успішно завершується після одержання підтвердження від всіх вузлів про їхню згоду.

10 ФІЗИЧНА ОРГАНІЗАЦІЯ ДАНИХ

10.1 Механізми середовища зберігання і архітектура СУБД

Механізми середовища зберігання БД служать для управління двома групами ресурсів – ресурсами збережених даних і ресурсами простору пам'яті. У завдання цього механізму входить відображення структури збережених даних в простір пам'яті, що дозволяє ефективно використовувати пам'ять і визначити місце розміщення даних при запам'ятовуванні і при пошуку даних.

З точки зору користувача робота з даними відбувається на рівні записів концептуального рівня і полягає в додаванні, пошуку, зміну і видалення записів. При цьому механізми середовища зберігання роблять наступне:

1. При запам'ятовуванні нового запису:

• визначення місця розміщення нового запису в просторі пам'яті;

• виділення необхідного ресурсу пам'яті;

• запам'ятовування цього запису (збереження в пам'яті);

• формування зв'язків з іншими записами (конкретний меха-нізм залежить від моделі даних).

2. При пошуку записи:

• пошук місця розміщення записи в просторі пам'яті по за-даними значенням атрибутів;

• вибірка записи для обробки в оперативну пам'ять (в буфер даних).

3. При зміні атрибутів записи:

• пошук запису і зчитування її в ОП;

• зміна значень атрибута (атрибутів) записи; • збереження запису на диск.

Запис поміщається на колишнє місце, якщо вона не збільшилася в обсязі або на колишньому місці досить пам'яті для неї. Якщо запис збільшилася в обсязі і не поміщається на колишньому місці, то вона або записуючиється на нове місце, або розбивається на частини, і перша частина зберігається на колишньому місці, а продовження - на новому, на яке вказується посилання з першої частини.

4. При видаленні запису:

• видалення запису зі звільненням пам'яті (фізичне видалення) або без звільнення (логічне видалення);

• руйнування зв'язків з іншими записами (конкретний механізм залежить від моделі даних).

У разі логічного видалення запис позначається як віддалена, але фактично вона залишається на колишньому місці. Фактичне видалення цієї за-

писи буде вироблено або при реорганізації БД, або спеціальної сервісної програмою, яку автоматично запускає СУБД або уручну АБД. При фізичному видаленні записи раніше зайнята ділянка звільняється і стає доступною для повторного використання.

Фізичну організацію БД ми будемо розглядати тільки для РСКБД.

Всі операції на фізичному рівні виконуються за запитами механізмів концептуального рівня СУБД. На фізичному рівні ніяких операцій безпосереднього поновлення призначених для користувача даних або перетворень уявлення збережених даних не відбувається, це завдання більш високих архітектурних рівнів. Управління пам'яттю виконується операційною системою за запитами СКБД або безпосередньо самої СКБД.

У трирівневої моделі архітектури СКБД декларується незалежність архітектурних рівнів. Але для досягнення більш високої продуктивності на рівні організації середовища зберігання часто доводиться враховувати специфіку концептуальної моделі. Аналогічно організація файлової системи не може не впливати на середовище зберігання.

10.2. Структура даних, що зберігаються

Одиницею зберігання даних в БД є **збережений запис**. Він може являти собою як повний запис концептуального рівня, так і деяку його частину. Якщо запис розбивається на частини, то ці частини представляються екземплярами збережених записів будь-яких типів. Всі частини запису зв'язуються покажчиками (посиланнями) або розміщуються за спеціальним законом так, щоб механізми міжрівневого відображення могли впізнати всі компоненти і здійснити збірку повного запису концептуальної БД за запитом механізмів концептуального рівня.

Збережені записи одного типу складаються з фіксованої сукупності полів і можуть мати формат фіксованої або змінної довжини.

Записи змінної довжини виникають, якщо допускається використання повторюваних груп полів (агрегатів) зі змінним числом повторів або полів змінної довжини. Робота з збереженими записами змінної довжини істотно ускладнює управління простором пам'яті, але може бути продиктована бажанням зменшити обсяг необхідної пам'яті або характером моделі даних концептуального рівня.

Збережена запис складається з двох частин:

1. *Службова частина*. Використовується для ідентифікації запису, завдання його типу, зберігання ознаки логічного видалення, для кодування значень елементів запису, для встановлення структурних асоціацій між за-

писами і ін. Ніякі призначені для користувача програми не мають доступу до службової частини збереженої записи.

2. *Інформаційна частина*. Містить значення елементів даних.

Поля збереженої записи можуть мати *фіксовану* або *змінну довжину*. При цьому бажано поля фіксованої довжини розміщувати на початку запису, а необов'язкові поля – в кінці. Зберігання полів змінної довжини здійснюється одним із двох способів: розміщення полів через роздільник або зберігання розміру значення поля.

Наявність полів змінної довжини дозволяє не зберігати незначні символи і знижує витрати пам'яті на зберігання даних; але при цьому збільшується час на витяг записи.

Кожному збереженому запису БД система привласнює внутрішній ідентифікатор, званий (за стандартом CODASYL) ключем бази даних (КБД). (Іноді використовується термін ідентифікатор рядка, RowID). Значення КБД формується системою при розміщенні запису і містить інформацію, що дозволяє однозначно визначити місце розміщення записи (конвертувати КБД на адресу записи). У якості КБД може виступати, наприклад, послідовний номер запису у файлі або сукупність адреси сторінки пам'яті і зміщення від початку сторінки.

Конкретні складові КБД залежать від операційної системи і від СКБД, точніше, від виду використовуваної адресації і від структуризації пам'яті, прийнятої в даній СКБД.

10.3 Управління простором пам'яті і розміщенням даних

Ресурсам простору пам'яті відповідають об'єкти зовнішньої пам'яті ЕОМ, керовані засобами операційної системи або СУБД.

Для забезпечення природної структуризації даних, що зберігаються, більш ефективного управління ресурсами і/або для технологічної зручності весь простір пам'яті БД зазвичай розділяється на частини (області, сегменти та ін.). У багатьох системах область відповідає файлу. У кожній області пам'яті, як правило, зберігаються дані одного об'єкта БД (однієї таблиці). Відомості про місце розташування даних таблиці (посилання на область зберігання) СУБД зберігає в словнику-довіднику даних (СДД). Області розбиваються на пронумеровані сторінки (блоки) фіксованого розміру. У більшості систем обробку даних на рівні сторінок веде операційна система (ОС), а обробку записів всередині сторінки забезпечує тільки СУБД.

Деякі СУБД дозволяють управляти розміром сторінки (блоку) для бази даних. У таких системах розмір сторінки визначається на основі компро-

місу між продуктивністю системи і необхідним обсягом оперативної пам'яті.

Сторінка має **заголовок** зі службовою інформацією, слідом за яким розташовуються власне дані. У більшості випадків в якості одиниці зберігання даних приймається збережена запис. На сторінці розміщується, як правило, кілька збережених записів, і є вільна ділянка для розміщення нових записів. Якщо запис не поміщається на одній сторінці, вона розбивається на фрагменти, які зберігаються на різних сторінках і посилаються один на одного. Система автоматично управляє вільним простором пам'яті на сторінках. Як правило, це забезпечується одним з двох способів:

• ведення списків вільних ділянок;

• динамічна реорганізація сторінок.

При **динамічної реорганізації сторінок** записи БД щільно розміщуються слідом за заголовком сторінки, а після них розташована вільна ділянка (рис. 10.1, а). Зсув початку вільної ділянки зберігається в заголовку сторінки. При видаленні записи залишилися записи переписуються поспіль в початок сторінки і змінюється зміщення початку вільної ділянки. При збільшенні розміру існуючої записи вона записується за колишньою адресою, а слідом йдуть записи зсуваються.

Перевага такого підходу – відсутність фрагментації. недоліки:

• Адреса запису може бути визначений з точністю до адреси сторінки, тому що всередині сторінки запис може переміщатися.

• Пошук місця розміщення нового запису може зайняти багато часу. Система буде читати сторінки одну за одною до тих пір, поки не знайде мандрівницю, на якій досить місця для розміщення нового запису.

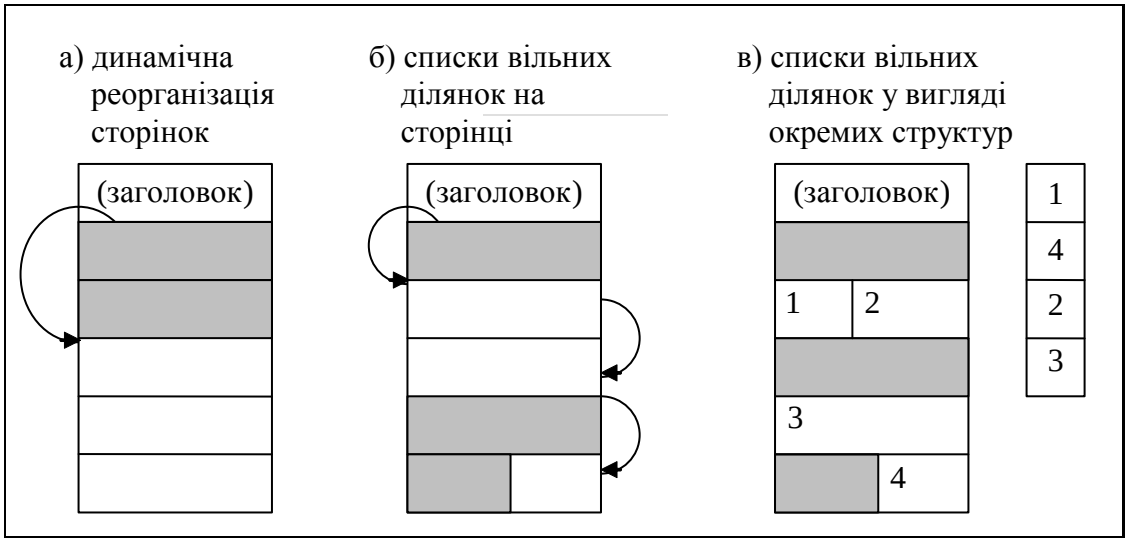


Рисунок 10.1 – Управління вільним простором пам'яті на сторінках

Для того щоб зменшити час пошуку місця для розміщення записів, при динамічної реорганізації сторінок можуть створюватися так звані *інвентарні сторінки*, на яких зберігаються розміри вільних ділянок для кожної сторінки. Пошук вільного місця для розміщення нових записів здійснюється через інвентарні сторінки, які завантажуються в оперативну пам'ять. При кожному видаленні/розміщенні даних вміст інвентарних сторінок оновлюється. Таким чином, забезпечення актуальності вмісту інвентарних сторінок займає додатковий час, але воно менше, ніж час пошуку вільної ділянки на сторінках.

Деякі СУБД управляють пам'яттю по-іншому: вони ведуть список **вільних ділянок**. Тут можна розглянути два варіанти:

1. Посилання на першу вільну ділянку на сторінці зберігається в заголовку сторінки, і кожна вільна ділянка зберігає посилання на наступну (або ознаку кінця списку) (рис. 10.1, б). Кожна ділянка, що звільняється, включається в список вільних ділянок на сторінці.

2. Списки вільних ділянок реалізуються у вигляді окремих структур (рис. 10.1, в). Ці структури також зберігаються на окремих інвентарних сторінках. Кожна інвентарна сторінка відноситься до області (або групі сторінок) пам'яті і містить інформацію про вільні ділянки в цій області. Список ведеться як стек, черга або впорядкований список. В останньому випадку впорядкування здійснюється за розміром вільної ділянки, що дозволяє при розміщенні нового запису вибирати для неї ділянку, що найбільш підходить за розміром.

Ведення списків вільних ділянок не призводить до переміщення запису, і адреса запису можна визначити з точністю до зсуву на сторінці. Це прискорює пошук даних, тому що не потрібно переглядати всі записи на сторінці для пошуку кожної конкретної записи.

При запоминании новой записи система через инвентарные страницы ищет свободный участок, достаточный для размещения этой записи. (Обычно выбирается первый подходящий участок, размер которого не меньше требуемого.) Если выбранный участок больше, чем запись, то остаток оформляется в виде свободного участка. (При динамической реорганизации страниц запись просто размещается вслед за последней записью на данной странице.) После этого система корректирует содержимое инвентарных страниц (если они есть).

При зміні запису, що має фіксований формат, вона просто перезаписується на колишнє місце. Якщо ж запис має формат змінної довжини, можливі ситуації, коли запис не поміщається на колишнє місце. Тоді запис розбивається на фрагменти, які можуть розміщуватися на різних сторінках. Ці

фрагменти пов'язані один з одним посиланнями, що дозволяє системі "збирати" запис з окремих фрагментів.

Основним недоліком, що виникають при використанні списків вільних ділянок, є фрагментація простору пам'яті, тобто поява розрізнених незаповнених ділянок пам'яті. Для того щоб зменшити фрагментацію, в подібних СУБД передбачені фонові процедури, які періодично проводять злиття суміжних вільних ділянок в один (наприклад, ділянки 1 і 2 на рис. 10.1, в).

Структура і поданні даних, що зберігаються, їх розміщення в просторі пам'яті і методи доступу, що використовуються, називаються схемою зберігання. Схема зберігання оперує в термінах типів об'єктів.

10.4. Види адресації збережених записів

У загальному випадку адреси записів БД ніде не зберігаються. При пошуку даних СУБД зі словника-довідника даних бере інформацію про те, в якій області пам'яті (наприклад, в якому файлі і/або на яких сторінках пам'яті) розташовані дані зазначеної таблиці. Але при цьому для пошуку конкретної записи (за значеннями ключових полів) система змушена буде прочитати всю таблицю. У РСУБД для прискорення пошуку даних застосовуються індекси – спеціальні структури, які встановлюють відповідність значень ключових полів записи і "адреси" цього запису (КБД). Таким чином, вид адресації збережених записів впливає на продуктивність, а також на переносимість БД з одного носія на інший.

Розглянемо три види адресації: пряму, непряму і відносну.

Пряма адресація передбачає зазначення безпосереднього місця розташування записи в просторі пам'яті. Пряма адресація використовується, наприклад, в системі ADABAS. Недоліком такої адресації є великий розмір адреси, обумовлений великим розміром простору пам'яті. Крім того, пряма адресація не дозволяє переміщати записи в пам'яті без зміни КБД. Такі зміни призвели б до необхідності корекції різних покажчиків на записи в середовищі зберігання (наприклад, в індексах), що було б надзвичайно трудомісткою процедурою. Відсутність можливості переміщати запис веде до фрагментації пам'яті.

Зазначені недоліки можна подолати, використовуючи **непряму адресацію.** Загальний принцип непрямої адресації полягає в тому, що в якості КБД виступає не сама "адреса записи", а адреса місця зберігання "адреси записи".

Існує безліч способів непрямої адресації. Один з них полягає в тому, що частина адресного простору сторінки виділяється під індекс сторінки (рис. 10.2). Число статей (слотів) в ньому однаково для всіх сторінок. У якості КБД записи виступає сукупність номера потрібної сторінки і номера

необхідного слота в індексі цієї сторінки (значення N, і на рис. 10.2). В і-м слоті на N-й сторінці зберігається власне адреса записи (зміщення від початку сторінки).

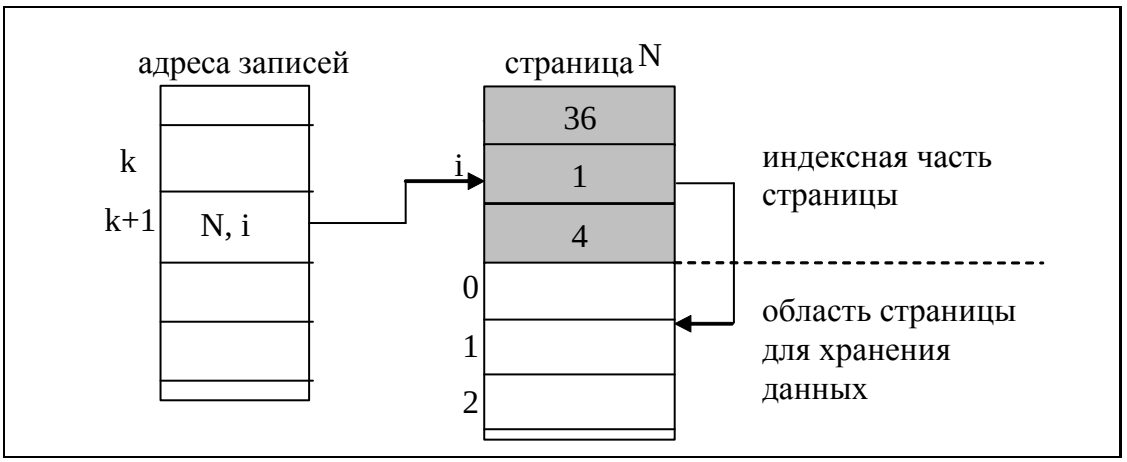


Рисунок 10.2 – Непряма адресація з використанням індексованих сторінок

При переміщенні запису він залишається на тій же сторінці, і слот як і раніше вказує на неї (змінюється його вміст, але не сам слот). Якщо запис не вміщається на сторінку, вона поміщається на спеціально відведені в даній області *сторінки переповнення*, і відповідний слот продовжує вказувати на місце її розміщення.

Цей підхід дозволяє переміщати записи на сторінці, виключати фрагментацію, повертати пам'ять, що звільнилася, для повторного використання.

Третій спосіб адресації – **відносна адресація**. Найпростіший варіант відносної адресації може використовуватися, наприклад, в ситуації, коли дані одного об'єкта БД (таблиці) зберігаються в окремому файлі і збережена запис має формат фіксованої довжини. Тоді як значення КБД береться порядковий номер запису, за яким можна обчислити зміщення від початку файлу

Загальний принцип відносної адресації полягає в тому, що адреса відраховується від початку тієї області пам'яті, яку займають дані об'єкта БД. Якщо пам'ять розбита на сторінки (блоки), то адресою може виступати номер сторінки (блоку) та номер запису на сторінці (або зсув від початку сторінки). У разі відносної адресації переміщення записи призведе до зміни КБД і необхідності коригування індексів, якщо вони є.

10.5. Способы размещения данных и доступа к данным в РБД

При створенні нового запису в багатьох випадках істотно розміщення цього запису в пам'яті, тому що це має великий вплив на час вибірки. Найпростіша стратегія розміщення даних полягає в тому, що новий запис роз-

міщується на першому вільному ділянці (якщо ведеться облік вільного простору) або слідом за останньою з раніше розміщених записів. Серед більш складних методів розміщення даних відзначимо хешування і кластеризацію.

Хешування полягає в тому, що спеціально підібрана хеш-функція перетворює значення ключа запису на адресу блоку (сторінки) пам'яті, в якому цей запис буде розміщуватися. Під ключем записи тут мається на увазі поле або набір полів, що дозволяють класифікувати запис. Наприклад, для таблиці СПІВРОБІТНИКИ як ключ запису може виступати поле Номер паспорта або набір полів (Прізвище, Ім'я, Дата народження).

Кластеризація – це спосіб зберігання в одній області пам'яті таблиць, пов'язаних зовнішніми ключами (одна батьківська таблиця, одна або кілька підлеглих таблиць). Для розміщення записів використовується значення зовнішнього ключа таким чином, щоб всі дані, що мають однакове значення зовнішнього ключа, розміщувалися в одному блоці даних. Наприклад, для таблиць СПІВРОБІТНИКИ, ДІТИ СПІВРОБІТНИКІВ, ТРУДОВА КНИЖКА, ВІДПУСТКИ як зовнішній ключ підлеглих таблиць виступає первинний ключ Ідентифікатор співробітника таблиці СПІВРОБІТНИКИ, і тоді при кластеризації всі дані про кожного співробітника будуть зберігатися в одному блоці даних

10.5.1. Способи доступу до даних

Розглянемо основні способи доступу до даних:

• **Послідовна обробка області БД**. Областю БД може бути файл або інша множина сторінок (блоків) пам'яті. Послідовна обробка передбачає, що система послідовно переглядає сторінки, пропускає порожні ділянки та видає записи в фізичної послідовності їх зберігання.

• **Доступ по ключу бази даних** (КБД). КБД визначає місце розташування записи в пам'яті ЕОМ. Знаючи його, система може отримати потрібний запис за одне звернення до пам'яті.

• **Доступ по ключу (зокрема, первинному**). Якщо система забезпечує доступ по ключу, то цей ключ також може використовуватися при запам'ятовуванні запису (для визначення місця розміщення записи в пам'яті). У базах даних застосовуються такі способи доступу по ключу, як індексування, хешування і кластеризація.

10.5.2. Індексування даних

Визначимо індексування як спосіб доступу до даних в реляційної таблиці за допомогою спеціальної структури – індексу.

Індекс – це структура, яка визначає відповідність значення ключа запису (атрибута або групи атрибутів) і розташування цього запису – КБД.

Кожен індекс пов'язаний з певною таблицею, але є зовнішнім по відношенню до таблиці і зазвичай зберігається окремо від неї.

Індекс зазвичай зберігається в окремому файлі або окремої області пам'яті. Порожні значення атрибутів (NULL) не індексуються.

Індексування використовується для прискорення доступу до записів за значенням ключа і не впливає на розміщення даних цієї таблиці. Прискорення пошуку даних через індекс забезпечується за рахунок:

1) упорядкування значень індексованого атрибута. Це дозволяє переглядати в середньому половину індексу при лінійному пошуку;

2) індекс займає менше сторінок пам'яті, ніж сама таблиця, тому система витрачає менше часу на читання індексу, ніж на читання таблиці.

Індекси підтримуються динамічно, тобто після поновлення таблиці – додавання або видалення записів, а також модифікації індексованих полів, – індекс приводиться у відповідність з останньою версією даних таблиці. Оновлення індексу, природно, займає деякий час (іноді, дуже великий), тому існування багатьох індексів може уповільнити роботу БД.

Звернення до запису таблиці через індекси здійснюється в два етапи: спочатку СУБД зчитує індекс в оперативну пам'ять (ОП) і знаходить в ньому необхідне значення атрибута і відповідний адресу записи (КБД), потім за цією адресою відбувається звернення до зовнішнього пристрою зберігання даних. Індекс завантажується в ОП цілком або зберігається в ній постійно під час роботи з таблицею БД, якщо вистачає обсягу ОП.

Індекс називається первинним, якщо кожному значенню індексу відповідає унікальне значення ключа. Індекс по ключу, що допускає дублікати значень, називається вторинним. Більшість СУБД автоматично будують індекс по первинному ключу і по унікальним стовпчиках. Ці індекси використовуються для перевірки обмеження цілісності unique (унікальність).

Для кожної таблиці можна одночасно мати кілька первинних і вторинних індексів, що також відноситься до переваг індексування.

10.5.3. Хешування

При асоціативному доступі до збережених записів, який передбачає визначення місця розташування запису за значеннями даних, що містяться в ньому, використовуються більш складні механізми розміщення. Для цієї мети використовуються різні методи відображення значення ключа на адресу, наприклад, методи хешування (перемішування).

Принцип хешування полягає в тому, що для визначення адреси запису в області зберігання до значення ключового поля цього запису застосовується так звана хеш-функція h (K). Вона перетворює значення ключа K на адресу ділянки пам'яті (це називається згорткою ключа). Новий запис буде

розміщуватися за тією адресою, який видасть хеш-функція для ключа цього запису. При пошуку записи за значенням ключа K хеш-функція видасть адресу, який вказує на початок тої ділянки пам'яті, в якій треба шукати цей запис.

Хеш-функція h (K) повинна володіти двома основними властивостями:

1) видавати такі значення адрес, щоб забезпечити рівномірний розподіл записів в пам'яті, зокрема, для близьких значень ключа значення адрес повинні сильно відрізнятися, щоб уникати перекосів в розміщенні даних:

$$K_1 \approx K_2 \Rightarrow h(K_1) >> h(K_2) \vee h(K_2) >> h(K_1),$$

2) для різних значень ключа видавати різні адреси:

$$K_1 \neq K_2 \Rightarrow h(K_1) \neq h(K_2).$$

Другу вимогу складно здійсненити. Важко підібрати таку хеш-функцію, яка для будь-якого розподілу значень ключа завжди видавала б різні адреси для різних значень. Для реальних функцій хешування допускається збіг значень функції h (K) для різних ключів. Для вирішення невизначеності при збігу адрес (колізії) після обчислення h (K) використовуються спеціальні методи.

Недолік методів підбору хеш-функцій полягає в тому, що кількість даних і розподіл значень ключа повинні бути відомі заздалегідь. Також методи хешування незручні тим, що записи зазвичай невпорядковані за значенням ключа, що призводить до додаткових витрат, наприклад, при виконанні сортування. До переваг хешування відноситься те, що прискорюється доступ до даних за значенням ключа. Звернення до даних відбувається за одну операцію введення/виводу, тому що значення ключа за допомогою хеш-функції безпосередньо перетворюється на адресу відповідного запису (або адреси блоку пам'яті, в якому зберігається цей запис). При цьому не потрібно створювати ніяких додаткових структур (типу індексу) і витрачати пам'ять на їх зберігання.

11 СИСТЕМИ КЕРУВАННЯ БАЗАМИ ЗНАНЬ

11.1 Термінологія

Існують різні визначення для баз знань і систем керування базами знань. Ми розуміємо СБЗ широко як систему, що дає можливість використовувати представлені підходящим способом знання за допомогою обчислювальної машини.

Термін "система баз знань" не є загальноприйнятим; існують інші близькі за змістом і більше розповсюджені терміни. Цей термін утворений за аналогією з терміном "система баз даних", що є калькою англомовного терміна data base system. *Під системою баз даних* (СБД) розуміють як інструментальну систему, що забезпечує створення й використання баз даних, так і систему, що забезпечує функціонування конкретної прикладної бази даних або декількох баз, тобто прикладну систему. У першому значенні в україномовній літературі звичайно вживається термін "система керування базами даних" (СКБД).

Ключовим поняттям у системах баз даних, що виражають їхню специфіку, є *база даних (БД),* а для систем баз знань – поняття *бази знань (БЗ).* Аналогічно СБД *системою баз знань (СБЗ)* називають систему, що забезпечує створення й використання баз знань. Її розглядають як інструментальну систему, яку також називають *системою керування базами знань (СКБЗ),* або як прикладну систему з конкретною прикладною базою знань.

Однак в англомовній літературі замість knowledge base system (система баз знань) воліють вживати термін knowledge based system (KBS), що означає *система, заснована на знаннях (СЗЗ),* або *система, що базується на знаннях (СБЗ).* Кращим, імовірно, є в якості українського технічного терміна останній, що дає, до речі, колишнє скорочення **СБЗ.**

До цього терміна близький за змістом термін *"експертна система"* (ЕС). У ньому акцент робиться на знання експертів, тобто фахівців у певній області. У літературі можна зустріти кілька іноді досить багатослівних і досить екзотичних визначень ЕС, але суть їх полягає в тому, що ЕС – це система, що забезпечує створення й використання за допомогою обчислювальної машини баз знань експертів, тобто по суті те ж, що СБЗ.

11.2 Принципи, структура й функції систем баз знань (СБЗ)

Аналізуючи схеми типових ЕС або СБЗ, можна виділити в них три основні частини – базу знань (БЗ), механізм одержання рішень (МР) і інтерфейс (ІФ), як показано на рисунку 11.1.

Кожна із цих частин може бути влаштована по-різному в різних системах, причому відмінності можуть бути як у деталях, так і в принципах. Слід також зазначити що між цими частинами немає абсолютної границі, їхнє розмежування досить умовно, вони можуть "перетинатися".

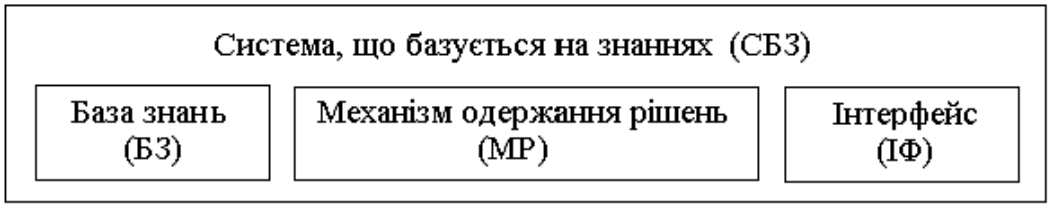


Рисунок 11.1– Основні частини СБЗ

Сама характерна риса СБЗ – наявність і використання БЗ. Термін *"база знань"* не має однозначного тлумачення. Так, у доповіді "Технологія конструювання баз знань" А.С.Нариньяни, керівник розробки технологічного комплексу побудови баз знань, сказав: "Що таке база знань - ніхто не знає. Це поняття широке і розмите, існує два основних різночитання: 1) БЗ як пакет знань, що занурений у систему; 2) БЗ як інтегрована система. Я розумію БЗ у другому сенсі". У такому ж сенсі трактує цей термін С.С.Лавров, що зображує структуру БЗ так, як показано на наступному рисунку:

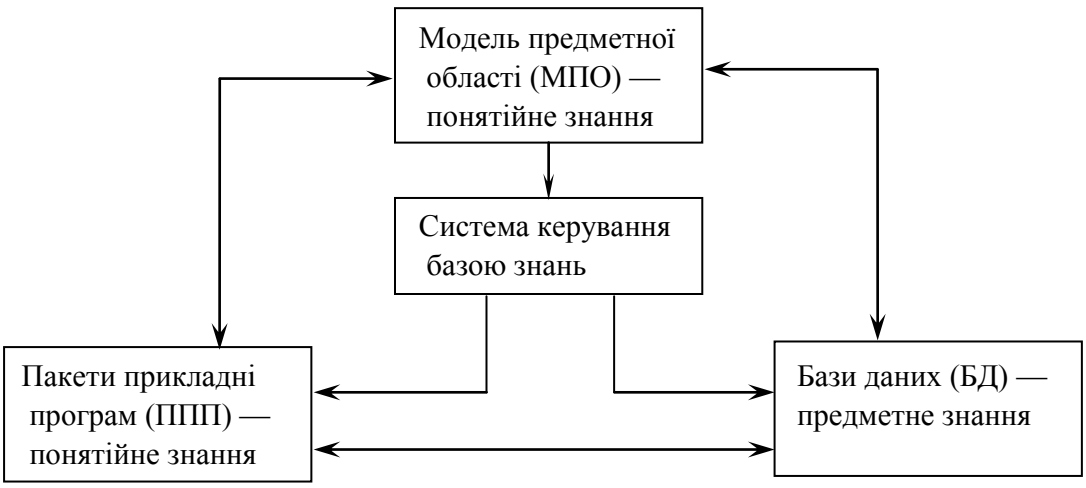


Рисунок 11.2 – Структура СУБЗ

У контексті СБЗ і ЕС, зв'язаному в остаточному підсумку з "програмуванням знань", корисно розрізняти алгоритмічні й неалгоритмічні знання. *Алгоритмічні* (або *процедурні) знання* — це алгоритми (програми, процедури), що обчислюють функції, виконують перетворення, вирішують точно визначені конкретні завдання. Базою алгоритмічних знань можна вважати будь-яке зібрання (бібліотеку) програм. Кожна система програмування й операційна система містять у собі базу алгоритмічних знань (особливо яскравий приклад – операційна система UNIX). Основою кожного пакета прикладних програм (ППП) або проблемно-орієнтованої системи (ПОС) є база алгоритмічних знань у конкретній прикладній області.

Неалгоритмічні знання охарактеризувати набагато сутужніше, ніж алгоритмічні. Вони складаються насамперед з уявних об'єктів, що називаються поняттями. Поняття звичайно має ім'я (можливо, кілька імен-синонімів), визначення, структуру (частини, елементи й т.д.), воно пов'язане з іншими поняттями й входить у якусь систему понять. Іншого роду неалгоритмічні знання – це зв'язки між поняттями або твердження про властивості понять і зв'язки між ними. Можна виділити два види неалгоритмічних знань: концептуальне (понятійне) і фактуальне (предметне). При цьому концептуальну частину бази знань називають моделлю предметної області (МПО), алгоритмічну – пакетом прикладних програм (ППП), фактуальну – базою даних (БД), а ту частину, представлену на рисунку вище системи рішення завдань, що має справу з концептуальним знанням, можна називати системою штучного інтелекту (СШІ).

Однак знання, втілені в поняттях, не зводяться до моделей предметних областей. Математичні знання, що складаються з математичних понять, зв'язків між ними й тверджень про них, принципово відрізняються від знань у предметних областях – фізиці, економіці, техніці й т.д., у яких уявні об'єкти (поняття) співвіднесені з реальними об'єктами (а в математиці тільки один з одним).

У багатьох ЕС і СБЗ уміст бази знань розділяють на “факти” і “правила”, причому факти відіграють роль елементарних “одиниць знання” (простих тверджень про характеристики об'єктів), а правила служать для вираження зв'язків, залежностей між фактами і їхніми комбінаціями. У системах баз даних цьому розподілу відповідає розподіл на об'єкти й зв'язки, прийняте в багатьох концептуальних моделях даних. База даних містить не тільки "факти" (записи об'єктів з їхніми атрибутами), але й зв'язки, тобто вона не обмежується "фактуальними знаннями". Отже, традиційну БД можна розглядати як своєрідну базу неалгоритмічних знань (про поточну си-

туацію в тій реальній обстановці, що ця БД моделює). Розвиток так званих семантичних моделей даних і мов запитів до БД збільшує подібність СБД зі СБЗ.

Таким чином, ми приходимо до наступної практично корисної класифікації знань, що застосовується в реальних СБЗ: 1) поняття (математичні й нематематичні); 2) факти; 3) правила, залежності, закони, зв'язки; 4) алгоритми, процедури.

Системи, що зараховуються їхніми авторами до СБЗ**,** відрізняються від традиційних СБД способами подання неалгоритмічних знань, з відповідними механізмами одержання рішень, і характером знань, що поміщають у БЗ, – знань фахівців (експертів), у яких відбиваються розуміння конкретної прикладної області й прийоми рішення виникаючих у ній типових завдань.

Можна відзначити ще два класи прикладних систем, у яких легко розгледіти БЗ. Це навчальні системи, що включають у себе "машинний підручник" і засоби керування навчанням; і системи підтримки прийняття рішень, що містять інформацію й засоби для моделювання й оцінки ситуації, що полегшують прийняття рішень.

Звичайну бібліотеку теж можна вважати БЗ, але в контексті СБЗ мова йде про знання, збережені у машинній пам'яті й більшою мірою "готові до вживання", чим знання, що перебувають на полках бібліотеки. Доступність знань у машинній формі й можливість прямого їхнього використання для рішення конкретних завдань є характерною рисою БЗ у СБЗ.

Пряме використання знань із БЗ для рішення завдань забезпечується *механізмом одержання рішень (МР)* – другою основною частиною СБЗ, яка називається також механізмом, або машиною виводу (перший термін — з логіки, другий — буквальний переклад inference engine), процедурою пошуку, планування, рішення й т.д. МР дає можливість витягати із БЗ відповіді на питання, одержувати рішення завдань, які формуються у термінах понять, що зберігаються в БЗ. Звичайно це "часткові" завдання й питання такого роду: "Дане те-те, знайти те-те"; "Знайти об'єкти, що задовольняють такий-те умові"; "Які дії виконати в такий-те ситуації".

Принципи роботи МР тісно зв'язані зі способами подання знань у БЗ. Для знань, представлених у БЗ рівняннями, МР є процедурою рішення рівнянь, для знань, представлених логічними формулами або правилами спеціального виду, це деякий механізм виводу й т.д. У СБД і ППП, що не претендують на звання СБЗ, теж є механізм одержання рішення. У СБД він

забезпечує, зокрема, переходи по зв'язках між об'єктами й пошук об'єктів, що задовольняють заданій умові. У ППП він установлює зв'язки між модулями, необхідні для одержання рішення, і настроює модулі на параметри конкретного завдання. Однак левова частина одержання рішення в ППП припадає на самі модулі – програми рішення прикладних завдань.

У кожному разі МР містить алгоритм одержання рішення, тобто спеціальне алгоритмічне знання. Тому що в МР утримується принаймні частина семантики БЗ, обумовлена в процедурній формі, це підтверджує відносність межі між МР і БЗ.

Третя основна частина СБЗ, що ми назвали *інтерфейсом (ІФ),* забезпечує роботу із БЗ і МР мовою досить високого рівня, наближеною до професійної мови фахівців у тій прикладній області, до якої відноситься СБЗ. Для цього в ІФ включається відповідний мовний процесор. Крім того, у функції ІФ входить підтримка діалогу користувача із системою, що дає можливість користувачеві одержувати пояснення дій системи, брати участь у пошуку рішення, поповнювати й коректувати БЗ. Іноді інтерфейс трактують дуже широко, розуміючи під "системою інтелектуального інтерфейсу" те, що ми називаємо СБЗ.

Розходження в поглядах на поняття СБЗ (БЗ, ЕС і т.д.) нерідко зводяться до різних розміщень акцентів на трьох згаданих частинах СБЗ, оцінкам їхньої відносної значимості й внеску в загальний ефект системи. Суть СБЗ полягає в тому, що знання, так чи інакше представлені в машинній пам'яті, "працюють" на користувача, дають ефект, що виправдує витрати на побудову або придбання СБЗ. Таким чином, ми приходимо до прагматичного визначення СБЗ і ЕС як прикладної системи, що забезпечує працездатність закладених у машинну пам'ять знань фахівців (експертів). Це дає критерій, по якому варто оцінювати СБЗ або ЕС незалежно від того, яка із трьох згаданих частин у ній переважає, яке в ній співвідношення алгоритмічного й неалгоритмічного знання й наскільки примітивні або витончені засоби й методи в ній використовуються.

11.3 Експертні системи і бази знань

Останнім часом з'явилася необхідність зберігання та використання слабоструктурованих даних, якими є, наприклад, людські знання в експертних системах.

Експертна система – система штучного інтелекту, що включає знання про певну слабо структуровану вузьку предметну область, яка важко формалізується, і ця система здатна пропонувати і пояснювати користува-

чеві розумні рішення. Експертна система складається з бази знань, механізму логічного висновку і підсистеми пояснень.

База знань – семантична модель, що описує предметну область і дозволяє відповідати на такі питання з цієї предметної області, відповіді на які в явному вигляді не присутні в базі. База знань є основним компонентом інтелектуальних та експертних систем.

Для зберігання баз знань в сучасних експертних системах використовуються або промислові СУБД і спеціалізоване проміжне програмне забезпечення (ПЗ), або спеціалізоване ПЗ.

12 ПЕРСПЕКТИВИ РОЗВИТКУ ТЕХНОЛОГІЇ БАЗ ДАНИХ

Ось уже більше 30-и років бази даних є однією з однією з найбільш широко затребуваних інформаційних технологій. Деякі автори стверджують [1], що поява баз даних стало найважливішим досягненням в області програмного забезпечення.

Системи баз даних докорінно змінили роботу багатьох організацій, і практично немає такої сфери діяльності, яку вони не торкнулися.

До числа найбільш важливих і перспективних напрямків розвитку БД слід віднести наступні:

- Сховища даних і OLAP-обробка. Сховище даних – це предметно-орієнтований, інтегрований, прив'язаний до часу і незмінний набір даних, призначений для підтримки прийняття рішень. Сховище даних дозволяють зберігати історичні дані з метою аналізу та прогнозування розвитку ситуацій. При правильному проектуванні сховище даних дає високу віддачу за рахунок більш якісного управління роботою організації (підприємства). Дані в сховищі даних обробляються за допомогою OLAP (online analytical processing) - інструментів оперативної аналітичної обробки даних. OLAP дозволяє швидко проводити розрахунки над величезними обсягами даних, в тому числі, з метою виявлення динаміки зміни різних параметрів (параметри задаються аналітиком).
- Робота з неточними даними. Інформація в базах даних часто містить помилки або є неповною. Результати запиту по такій БД можуть сильно відрізнятися від реального положення справ. Процесор запитів, що працює з можливостями, коефіцієнтами довіри, коефіцієнтами повноти і т.д. дозволив би враховувати ступінь достовірності даних при прийнятті рішень на основі цих даних.
- Нові призначені для користувача інтерфейси. Це одне з найбільш актуальних напрямків сучасних інформаційних технологій. Кінцеві користувачі не знають мову запитів (SQL), і для отримання інформації з БД змушені користуватися інтерфейс, які для них створюють програмісти. У додатки зазвичай включають певний набір готових запитів і можливість сформулювати довільний запит за допомогою якогось конструктора. Але для того, щоб скористатися конструктором, користувач повинен знати структуру бази даних і добре розбиратися в запропонованому йому формалізмі ПО.

Найбільш природним виглядом є запит до БД, сформульований на природній мові (ПМ). Але для таких запитів характерні неточності і неоднозначність. Вирішення цієї задачі неможливе без використання знань про предметну область і про структуру мови.

Одним з варіантів вирішення цієї проблеми є онтології.

Під онтологією розуміється певним чином формалізована система знань про предметну область, що описує, класифікує і погоджує між собою поняття цієї ПО. Інтеграція онтологій і баз даних дозволить користувачам робити запити у власній термінології з використанням обмеженого природної мови. Це спростить створення і супровід додатків і підвищить ефективність використання БД.

- Проблеми оптимізації запитів. Крім завдання пошуку нових способів оптимізації, що залишається актуальним, можна виділити ще дві серйозні проблеми оптимізації: обробка неструктурованих запитів (можливо, на обмеженій природній мові), і оптимізація групи запитів. Робота з неструктурованими запитами особливо актуальна в світлі використання баз даних в пошукових системах (в тому числі, при пошуку в Internet). А оптимізація групи запитів, що одночасно виконуються, дозволить поліпшити характеристики СКБД з точки зору швидкодії.
- Інтеграція різнорідних і слабо формалізованих даних. Спочатку бази даних призначалися для зберігання і обробки фактографічних добре структурованих даних. Але величезна кількість даних представлено в різних графічних і мультимедійних форматах. Включення в СУБД способів обробки подібних даних дозволяє використовувати технології баз даних в таких сферах, як, наприклад, ГІС (геоінформаційні системи), видавничі системи (з підтримкою верстки номерів видання), САПР (системи автоматизації проектування) і т.д.
- Організація доступу до баз даних через Internet. Багато web-сайтів містять динамічну інформацію, наприклад, про товари і ціни в Internet-магазинах. У локальних системах така інформація традиційно зберігається в базах даних. Інтеграція СКБД в web-середовище дозволяє зберегти всі переваги баз даних для використання в web-додатках. Основними задачами тут є:
    a. організація ефективного інтерфейсу, розрахованого на непідготовленого користувача;

b. оптимізація запитів, спрямована на зменшення мережевого трафіку;

c. підвищення продуктивності СКБД в розрахованому на багато користувачів режимі роботи.

- Самоадаптація. Сучасні СКБД мають широкі можливості по налаштуванню баз даних під конкретну предметну область і апаратні засоби. Але використання цих можливостей – досить складне завдання, яке вимагає наявності висококваліфікованого адміністратора БД. Для спрощення налаштування і супроводу БД СКБД повинна брати на себе більшість функцій настройки і виконувати їх в автоматичному або автоматизованому режимі.
- Використання GRID. GRID – це концепція об'єднання обчислювальних ресурсів в єдину мережу. В якості аналогії тут можна привести електричні мережі: при виникненні потреби користувач просто підключається до мережі і отримує електрику. Точно так же при виникненні потреби в обчисленнях користувач повинен просто підключатися до GRID і отримувати обчислювальні ресурси. Переваги цього підходу очевидні: можливість вирішувати більш ресурсомісткі завдання і перерозподіляти навантаження на вузли мережі. Але і невирішених проблем тут теж досить, тому це завдання майбутнього.

  Проте, вже існують промислові GRID-системи, які підтримують бази даних: це системи Oracle 10G і Oracle 11G (G - це скорочення від GRID). Вони динамічно виділяють ресурси для виконання завдань користувача з доступу до БД Oracle і перерозподіляють навантаження на вузли мережі з метою оптимізації використання обчислювальних ресурсів і підвищення загальної продуктивності системи.
- Збереження даних. Кількість накопичених цифрових даних в світі величезна. Але з часом застарівають і формати зберігання даних, і засоби доступу до них. Відбувається також старіння носіїв: розмагнічуються магнітні стрічки і диски, змінюються оптичні та фізичні властивості носія. Тому навіть заархівовані дані можуть стати недоступними, особливо якщо немає пристрою для читання застарілого носія або відсутня можливість запустити додаток, яке може читати застарілий формат. Вирішити цю проблему можуть засоби, що забезпечують міграцію даних в нові формати із збереженням їх опису (тобто метаданих).

- Технології розробки даних і знань (data mining і knowledge mining). Технології розробки даних призначені для пошуку неочевидних тенденцій і прихованих закономірностей у великих обсягах даних. А knowledge mining – це витяг знань з баз даних (або зі сховища даних). Тут використовуються як формальні методи (регресійний, кореляційний і інші види статистичного аналізу), так і методи інтелектуальної обробки даних, засновані на моделюванні пізнавальних механізмів - індукції, дедукції, абдукції.

СПИСОК ПОСИЛАНЬ

1. Конноли, Томас, Бегг, Каролин. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. 3-е издание.: Пер. с англ.– М.: Издательский дом “Вильям”, 2003. – 1440 с.
2. Дейт К. Дж.. Введение в системы баз данных, 6-е издание: Пер. с англ. – К.; М.; СПб: Издательский дом «Вильяме», 2000 – 848 с.
3. Козловська В.П. Організація баз даних та знань: Конспект лекцій, електронна версія. – Одеса, ОДЕКУ, 2013 – 125 с.

Навчальне електронне видання

КОЗЛОВСЬКА ВАЛЕНТИНА ПЕТРІВНА

ОРГАНІЗАЦІЯ БАЗ ДАНИХ ТА ЗНАНЬ

Конспект лекцій

Видавець і виготовлювач
Одеський державний екологічний університет
вул. Львівська, 15, м. Одеса, 65016
тел./факс: (0482) 32-67-35
E-mail: info@odeku.edu.ua
Свідоцтво суб'єкта видавничої справи
ДК № 5242 від 08.11.2016