Міністерство освіті і науки України
Вінницький національний технічний університет

Анатолій Михайлович Пэтух
Оксана Володимирівна Романюк,
Олександр Никифорович Романюк

БАЗИ ДАНИХ. МОВИ ЗАПИТІВ, УПРАВЛІННЯ ТРАНЗАКЦІЯМИ, РОЗПОДІЛЕНА ОБРОБКА ДАНИХ

Вінниця ВНТУ 2016

ЗМІСТ

ВСТУП

Бази даних становлять невід'ємну складову діяльності сучасних підприємств та організацій. Невпинне зростання обсягів інформації, що зберігається в базах даних, та розширення кола користувачів висувають до систем керування базами даних (СКБД) нові вимоги.

Потреба у швидкому редагуванні та отриманні інформації з бази даних постала ще в 1970-роках. Великі обсяги інформації робили неможливим ручне її опрацювання. Саме тоді в дослідницькій лабораторії компанії IBM розробляються мови запитів SQL та QBE, які сьогодні реалізовані практично в кожній системі керування базами даних. Візуальний підхід до формування запитів мовою QBE робить її незамінним інструментом для задоволення інформаційних потреб користувачів, які не володіють навикам програмування. Декларативна мова запитів SQL не вимагає вказання алгоритму отримання даних, а тому з легкістю може бути інтегрована в будь-який програмний код, що дає програмістам широкі можливості по створенню, модифікації баз даних та вибірці інформації з них в режимі реального часу.

Багатокористувацький режим роботи сучасних баз даних вимагає від СКБД здійснення управління паралельним виконанням транзакцій. Неналежне виконання цієї задачі може призводити до втрати даних, невчасного оновлення даних та формування некоректних звітів у випадках, коли декілька транзакцій, що виконуються паралельно, звертатимуться до одних і тих же даних. Тому вивчення методів управління паралельним виконанням є актуальним.

Вміст бази даних може бути зруйнованим або приведеним в неузгоджений стан, зокрема й через неправильну роботу по управлінню транзакціями. В такому випадку ключова роль віддається методам відновлення баз даних.

Сьогодні великі інтернаціональні компанії мають широку мережу дочірніх підрозділів, які збирають та обробляють необхідну їм інформацію на місцях. Тому задача централізованої обробки всієї інформації компанії є недоцільною через великі часові затримки у виконанні запитів до бази даних. Натомість, завдяки швидкому розвитку технологій мережевого зв'язку та обміну даними, спостерігається активний розвиток концепції розподілених баз даних, які вважаються одним з найбільших досягнень в галузі баз даних.

В навчальному посібнику приведені основи формування запитів мовами SQL та QBE, методи управління транзакціями та паралельним виконання, підходи до відновлення бази даних, а також розглянуто концепцію розподілених баз даних, принципи проектування розподілених баз даних та методи управління паралельним виконанням у розподіленому середовищі.

1 МОВИ ЗАПИТІВ QBE ТА SQL

Сучасні бази даних зберігають великі об'єми інформації, тому обробляти її вручну, послідовно переглядаючи і редагуючи дані в таблицях, стає досить важко. Для підвищення ефективності користування базами даних застосовують *запити*, які дозволяють виконувати множинну обробку даних, тобто одночасно вводити, редагувати та видаляти велику кількість записів, а також вибирати дані з таблиць.

Запит – це засіб отримання інформації з бази даних.

Запит являє собою спеціальним чином описану вимогу, яка визначає склад операцій, що виконуються над базою даних. До таких операцій відносять операції по вибірці, видаленню або модифікації даних.

Для підготовки запитів з допомогою різних СКБД найчастіше використовують дві основні мови опису запитів:

– мова QBE (Query By Example) – мова запитів за зразком;

– мова SQL (Structured Query Language) – структурована мова запитів.

1.1 Загальна характеристика мов запитів QBE та SQL

Мова QBE була розроблена Моше Злуфом, співробітником дослідницького центру компанії IBM, в 1970-х роках і призначалася для користувачів, зацікавлених у вибірці інформації з баз даних. Ця мова отримала в користувачів настільки широке визнання, що в даний час в тій чи іншій мірі вона реалізована практично у всіх популярних СКБД, включаючи і Microsoft Access.

В мові QBE використовується візуальний підхід до організації доступу до інформації в базі даних, який передбачає заповнення форми запиту, яку надає СКБД. Такий спосіб задання запиту забезпечує високий ступінь наочності і не потребує вказання алгоритму виконання операцій, адже достатньо лише описати зразок очікуваного результату. В кожній з сучасних СКБД є свій варіант мови QBE.

На мові QBE можна створювати запити *однотабличні* та *багатотабличні* (вибирають і обробляють дані з декількох зв'язаних таблиць).

За допомогою запитів мовою QBE можна виконати такі основні операції над даними бази даних:

– вибірку даних;

– обчислення над даними;

– додавання нових записів;

– видалення записів;

– модифікацію (зміну) даних.

У випадку перших двох операцій результатом виконання запиту є таблиця, які називають *таблиця-відповідь*, а у випадку інших операцій – *оновлена вихідна таблиця*.

Вибірка, вставка, видалення та модифікація можуть виконуватись безумовно або відповідно до певних умов, які задаються за допомогою логічних виразів. Обчислення над даними задаються за допомогою арифметичних виразів і породжують в таблицях-відповідях нові поля, які називають *обчислюваними*.

Форма запиту має вигляд таблиці, ім'я і назви полів якої співпадають з іменем і назвами полів відповідної вихідної таблиці. Щоб дізнатись імена доступних таблиць БД, в мові QBE передбачений запит на вибірку імен таблиць. Назви полів вихідної таблиці можуть вводитись в шаблон вручну або автоматично. В другому випадку виконується запит навибірку заголовків стовпців.

В сучасних СКБД, наприклад в MS Access і Visual FoxPro, більшість дій по підготовці запитів за допомогою мови QBE виконуються візуально за допомогою миші.

СУБД Microsoft Access при створенні запиту з використанням засобів QBE неявно формує еквівалентний оператор мови SQL, призначений для виконання зазначених дій.

Структурована мова запитів SQL – декларативна мова програмування для взаємодії користувача з базами даних. Мова SQL орієнтована на операції з даними, представленими у вигляді логічно взаємозв'язаних сукупностей таблиць-відношень. Найважливіша особливість структур цієї мови полягає в орієнтації на кінцевий результат обробки даних, а не на процедуру цієї обробки. SQL сама визначає, де знаходяться дані, індекси і навіть які найбільш ефективні послідовності операцій слід використовувати для отримання результату, тому не треба вказувати ці деталі в запиті до БД.

Ця мова призначена для виконання:

- операцій над таблицями (створення, видалення, зміна структури);
- операцій над даними таблиць (вибірка, зміна, додавання і видалення);
- операцій управління даними (надання і відміна привілеїв на доступ до даних, управління транзакціями та інші).

В результаті вибірки даних з однієї або декількох таблиць може бути отримана множина записів, які називають *представленням*.

SQL є непроцедурною мовою і складається з обмеженого числа команд, спеціально призначених для управління над даними. У зв'язку з цим SQL автономно не використовується, зазвичай вона занурена у середовище вбудованої мови програмування СКБД (наприклад, FoxPro СКДБ Visual FoxPro, ObjectPAL СКБД Paradox, Visual Basic for Applications СКБД Access).

Мова SQL бере свій початок в одній з дослідницьких лабораторій компанії IBM з 1970-х років. У 1986 році організація ANSI випустила офіційний стандарт під назвою SQL-86. Цей стандарт був оновлений тією

ж організацією в 1989 році і отримав назву SQL -89, а потім в 1992 році (SQL-92 або SQL2). Найостаннішою версією стандарту SQL є SQL 2003 .

Стандарт SQL визнається не лише ANSI (американським національним інститутом стандартів), а й зараз також приймається ISO (міжнародною організацією по стандартизації). Проте більшість комерційних програм БД розширюють SQL без повідомлення ANSI, додаючи різні інші особливості в цю мову, які, як вони вважають, будуть дуже корисні.

Основними перевагами мови SQL є:

– *стандартність мови SQL* – її використання в програмах стандартизоване міжнародними організаціями;

– *незалежність від конкретних СКБД* – всі поширені СКБД використовують SQL, оскільки реляційну БД і програми, які з нею працюють, можна перенести з однієї СУБД на іншу з мінімальними доопрацюваннями;

– *можливість переносу з однієї обчислювальної системи на іншу* – СУБД може бути орієнтована на різні обчислювальні системи, проте додатки, створені за допомогою SQL, допускають використання як для локальних БД, так і для великих, розрахованих на багато користувачів систем;

– *реляційна основа мови* – SQL є мовою реляційних БД, тому вона стала популярною тоді, коли популярною стала реляційна модель представлення даних. Таблична структура реляційної БД добре зрозуміла, тому мова SQL є простою і легкою для вивчення;

– *можливість створення інтерактивних запитів* – SQL забезпечує користувачам негайний доступ до даних, при цьому в інтерактивному режимі можна отримати результат запиту за дуже короткий час без написання складної програми;

– *можливість програмного доступу до БД* – мова SQL може бути легко використана в додатках, яким необхідно звертатися до БД. Одні і ті ж оператори SQL використовуються як для інтерактивного, так і для програмного доступу, тому частини програм, що містять звернення до БД, можна спочатку перевірити в інтерактивному режимі, а потім вбудовувати в програму;

– *забезпечення різного представлення даних* – за допомогою SQL можна передбачити таку структуру даних, що той або інший користувач бачитиме різні представлення даних. Крім того, дані з різних частин БД можуть бути скомбіновані і представлені користувачу у вигляді однієї простої таблиці, а значить, уявлення можна використовувати для посилення захисту БД і її настройки під конкретні вимоги окремих користувачів;

– *можливість динамічної зміни і розширення структури БД* – мова SQL навіть під час звернення до вмісту дозволяє маніпулювати структурою БД. Ця велика перевага перед мовами статичного визначення даних,

які забороняють доступ до БД під час зміни її структури. Таким чином, SQL забезпечує гнучкість з погляду пристосованості БД до вимог предметної області, що змінюються, не перериваючи при цьому роботу додатку, що її виконує в реальному масштабі часу;

– *підтримка архітектури клієнт/сервер* – SQL один з кращих засобів для реалізації додатку на платформі клієнт/сервер. При цьому SQL служить сполучною ланкою між клієнтською системою, що взаємодіє з користувачем, і серверною системою, БД, що управляє, дозволяючи кожній з них зосередитися на виконанні своїх прямих функцій.

Зважаючи на численні переваги мова SQL на сьогодні залишається найбільш поширеною мовою запитів.

1.2 Опис запитів мовою QBE

Запит QBE являє собою одну або декілька таблиць, рядки яких містять вимоги до значень рядків таблиці, одержуваної в результаті виконання запиту.

Відповідно до синтаксису запиту назву таблиці (відношення) поміщають в заголовок першого стовпця (сам стовпець залишається порожнім завжди, крім випадку короткого запису включення до результату всіх стовпців таблиці). Для кожного атрибута, що бере участь у запиті, в таблицю включається свій стовпець, в заголовоку якого міститься ім'я атрибута. Так, шаблон запиту для таблиці STUDENT, атрибутами якої є ПІБ, Група та Курс студента, матиме такий вигляд:

STUDENT	ПІБ	Група	Курс

Значення атрибутів (зразок, змінна або результат запиту) поміщаються у відповідні стовпці.

Для опису змінних в умовах відбору записів, а також для зв'язування шаблонів в запитах використовуються змінні – **елементи прикладу**. Елемент прикладу грає роль ідентифікатора змінної (як у мові програмування) і задається за допомогою символьно-цифрової послідовності. Довжина та склад елементу прикладу можуть бути довільними: головне, щоб при використанні у різних місцях шаблону він мав однаковий вигляд. Елелентом прикладу можуть виступати ідентифікатори **example**, **x** або ***n***.

Константні значення, які й складають той самий «зразок» з терміна QBE беруться в лапки.

Для вказівки шуканих атрибутів значенню відповідного стовпця – змінній, константному виразу або пустому значенню (коли необхідно включити стовпець у результат, але значення не використовується для зв'язку з іншими таблицями) – передує символ «P» (що означає «надрукувати») з крапкою. Занесення «P.» у всі стовпці шаблону можна замінити

записом P. в першому стовпці шаблону під ім'ям таблиці. Так, запит на вибірку інформації про всіх студентів матиме вигляд:

STUDENT	ПІБ	Група	Курс
P.			

Якщо потрібно вивести лише ПІБ студентів, то символ «P.» необхідно записати лише в стовбець ПІБ:

STUDENT	ПІБ	Група	Курс
	P.		

Для впорядкування виведених значень за зростанням або за спаданням, використовують конструкції «AT.» і «DO.» відповідно. Якщо потрібно виконати упорядкування за кількома стовпцями , застосовують конструкції виду: «AT (1)» (для першого стовпця упорядкування) , «AT (2)» (для другого стовпця упорядкування) і так далі.

1.2.1 Вибірка даних з умовою

Вибір записів з умовою (з кваліфікатором) в загальному випадку може бути заснований на точному співпадінні, частковому співпадінні або порівнянні.

1. *Точне* співпадіння задається введенням констант у відповідних полях шаблону.

2. *Часткове* співпадіння задається за допомогою елементів прикладу. Зокрема, для формулювання запиту про виведення списку всіх студентів, прізвища яких починаються з літери «А» , а закінчуються на «ов» , можна скористатися конструкцією P.«A»**name**«ко», записаної в поле ПІБ таблиці STUDENT, де **name** – змінна.

3 . *Умова порівняння* записується за допомогою операцій порівняння: дорівнює (=), більше (>), менше (<) , більше або дорівнює (> =) , менше або дорівнює (<=), що не дорівнює (!=), не більше (! >), що не менше (! <).

Приклад 1.1. Запит на вибірку за точним співпадінням.
Вивести ПІБ всіх студентів 3-го курсу.

STUDENT	ПІБ	Група	Курс
	P.**xx**		«3»

У наведеному шаблоні елемент прикладу замість P.**xx** можна вказати просто P., оскільки елементи прикладу обов'язково вказуються тільки при записі логічних умов, а також при зв'язуванні таблиць в запитах.

Приклад 1.2. Запит на вибірку з частковим спів падінням.

Ввести список студентів, чиї прізвища починаються на літеру «А», а закінчуються на «ко».

STUDENT	ПІБ	Група	Курс
	Р. «А»**name**«ко»		

У наведенному прикладі «А» і «ко» є константами, а **name** – елементом прикладу. Використовуючи елементи прикладу, можна задавати різноманітні варіанти часткового співпадання із значеннями даних із таблиць. Оскільки елементу приклада відповідає будь-який символ, а також пустий (відсутність символа), то умові часткового співпадання **x1**«є»**x2** відповідають слова, які мають символ “є” не тільки всередині, але й на початку та в кінці.

Приклад 1.3. Запит на вибірку з умовою порівняння і константою.

Вивести імена співробітників, які працюють у відділі «Іграшки» та отримують заробітну плату більше 4500 грн.

EMP	ПІБ	Зарплата	Керівник	Відділ
	Р.	>4500		«Іграшки»

У наведеному прикладі присутні дві умови, які об’єднуються логічною зв’язкою «І». Константою виступає назва відділу – «Іграшки».

Приклад 1.4. Запит на вибірку з умовою порівняння та елементом прикладу.

Вивести список імен та заробітних плат усіх співробітників, чия заробітна плата більша, ніж в Іванова А.О.

Оскільки значення заробітної плати Іванова А.О. не відоме, запит можна перефразувати так: «Нехай Іванов А.О. отримує зарплату в розмірі **zp**. Знайти всіх співробітників, які отримують зарплату більше, ніж **zp**, та вивести їх зарплати».

EMP	ПІБ	Зарплата
	Р.	Р.>**zp**
	«Іванов А.О.»	**zp**

У даному прикладі реалізована вибірка з порівнянням елементу прикладу. Порядок рядків у шаблоні не важливий. Кожен такий рядок визначає вимоги до результату – рядків результуючої таблиці. Відповідно до цього, набір рядків таблиці запиту можна вважати набором предикатів, об'єднаних зв'язкою «І» в логічний вираз, які становлять критерій відповідності знайдених рядків даному запиту.

Приклад 1.5. Запит з об'єднанням умов.

Знайти імена і зарплати співробітників, які отримують більше ніж Петров В.В. і працюють у відділі, де продаються олівці.

EMP	ПІБ	Зарплата	Керівник	Відділ
	P.	P.> **zp**		**department**
	«Петров В.В.»	**zp**		

SALES	Відділ	Товар
	department	«олівець»

Для виконання такого запиту необхідно зв'язати дві таблиці EMP та SALES за допомогою елемента прикладу **department**.

Приклад 1.6. Запит з двома зв'язками в одному шаблоні.

Знайти імена всіх співробітників, які отримують більше, ніж їх керівники.

Цей запит за допомогою елементів прикладів можна сформулювати таким чином: «Знайти всіх співробітників, чиї керівники є **head** та отримують зарплату, більшу, ніж **zp**, де **zp** – зарплата **head**». Шаблон відповідного запиту має вигляд:

EMP	ПІБ	Зарплата	Керівник	Відділ
	P.	> **zp**	**head**	
	head	**zp**		

Елемент **head** використовується для зв'язку керівника в першому рядку шаблону та імені в другому рядку, а елемент **zp** використовується для порівняння зарплат.

Приклад 1.7. Запит з операцією заперечення.

Вивести назви відділів, у яких не продаються товари, що виготовлені під брендом «Своя лінія».

SALES	Відділ	Товар
	P.	**i**

SYPPLY	Товар	Бренд
	!i	«Своя лінія»

У випадках , коли умови відбору записів для вибірки є великими виразами, які незручно або важко задати в шаблоні, можна використовувати блок умов. Він ззовні нагадує порожній шаблон з одним полем і ім'ям CONDITIONS. Блок умов призначений для запису логічних виразів.

Логічні вирази можуть включати операції логічного множення AND та логічного додавання OR.

Приклад 1.8. Запит з блоком умов та операцією AND.

Вивести імена співробітників, чия заробітна плата складає від 3200 грн. до 7000 грн., але не дорівнює 5000 грн.

Традиційний шаблон запиту матиме такий вигляд:

EMP	ПІБ	Зарплата
	P.**Ivan**	>=3200
	Ivan	<=7000
	Ivan	!5000

Використовуючи блок умов з явним заданням AND (&), цей запит можна сформулювати так:

EMP	ПІБ	Зарплата
	P.	**zp**

CONDITIONS
zp=(>=3200 & <=7000 & !5000)

Приклад 1.9. Запит з блоком умов та операцією OR.

Вивести імена співробітників, чия заробітна плата складає 3200 грн., 5000 грн. або 6400 грн.

Традиційний шаблон запиту матиме такий вигляд:

EMP	ПІБ	Зарплата
	P.**Ivan**	3200
	P.**Igor**	5000
	P.**Oleg**	6400

В кожному рядку шаблону використовуються різноманітні елементи прикладів і тому ці умови діють незалежно. Використовуючи блок умов з явним заданням операції OR (|), цей запит можна сформулювати так:

EMP	ПІБ	Зарплата
	P.	**zp**

CONDITIONS
zp=(3200|5000|6400)

1.2.2 Використання результуючих функцій

При запису логічних виразів на мові QBE можуть застосовуватись результуючі функції: CNT. (лічильники кількості), SUM. (сума), AVG. (се-

реднє), MIN. (мінімум), MAX. (максимум), UN. (унікальний) та ALL. (всі значення, в тому числі й ті, що повторюються). Перші п'ять із них є статистичними, а останні дві визначають характер вибірки: включати чи не включати у вибірку повторювані значення.

Функцію UN. Можна приєднати до функцій CNT., SUM., и AVG. Так, запис CNT.UN. означає кількість значень, які відрізняються. А запис CNT.ALL навпаки буде означати кількість всіх значень.

Приклад 1.10. Використання функцій з блоком умов.

Необхідно вивести назви груп, в яких навчається більше 20 студентів.

Цей запит можна розділити на три операції: згрупувати студентів за групами, підрахувати число студентів у кожній з груп і відібрати групи, в яких працює більше 20 студентів. Обчислення в групах виконуються за допомогою конструкції **GB** (Group-by).

Student	ПІБ	Група
	ALL.**Студент**	P.**GB**

Conditions
CNT.ALL.**Студент**>20

Приклад 1.11. Використання функцій зі зв'язуванням таблиць.

Вивести назви відділів, в яких продаються тільки товари зеленого кольору.

SALES	Відділ	Товар
	P.**GB**	ALL.**item**

TYPE	Товар	Колір
	ALL. **item**	«зелений»

1.2.3 Обчислення в запитах

За допомогою запитів можна вибирати дані з таблиць і проводити обчислення. Вид обчислень задається за допомогою виразу в шаблоні. У виразах, окрім звичних арифметичних операцій (+, -, *, /) та дужок, можуть використовуватися вбудовані функції: AVG., CNT., MAX., MIN. І SUM.

Приклад 1.12. Запит з обчисленням з арифметичними операціями.

Нехай є таблиця EMP1 з полями ПІБ, Зарплата та Премія. Необхідно по кожному із співробітників вивести ім'я та загальну суму зарплати та премії.

Для цього сформуємо шаблон нової таблиці OUTPUT (заповнивши в ній стрічку з ім’ям таблиці та іменами її полів) та вкажемо в ній вид обчислень. Зв’яжемо цей шаблон з шаблоном запиту до таблиці EMP1.

OUTPUT	ПІБ	СУМА
	P.**Employee**	P.(**zp**+**pr**)

EMP1	ПІБ	Зарплата	Премія
	Employee	**zp**	**pr**

Оскільки операція додавання виконується по кожній стрічці вихідної таблиці, такий тип обчислень називають ***горизонтальним***. Вбудовані функції оперують групами записів, тому можна вважати, що вони виконують ***вертикальні*** обчислення.

Приклади 1.13. Запит з обчисленням з результуючими функціями.

Для *підрахунку загальної кількості співробітників* необхідно скласти такий запит:

EMP	ПІБ	Відділ
	P.CNT.ALL.Employee	

Елемент прикладу **Employee** можна опустити.

Якщо потрібно *підрахувати кількість співробітників у відділі іграшок*, то необхідно підготувати шаблон запиту виду:

EMP	ПІБ	Відділ
	P.CNT.ALL.Employee	«Іграшки»

Якщо потрібно *підрахувати кількість співробітників у кожному відділі*, то підготуємо шаблон запиту виду:

EMP	ПІБ	Відділ
	P.CNT.ALL.Employee	P.**department**

В шаблоні запиту була застосована операція групування.

1.2.4 Операції додавання, видалення і модифікації мови QBE

На відміну від розглянутих операцій, операції додавання, видалення та модифікації призводять до зміни вихідної таблиці. Вид операції (вставка – I., видалення – D., модифікація – U.) записується в шаблоні під ім'ям таблиці, а константи і умовні вирази вказуються за тими ж правилами, що в операціях вибірки.

Приклади 1.14. Операції додавання, видалення та модифікації.

Для *додавання в таблицю* EMP нового співробітника відділу «Іграшки» з прізвищем Антонюк О.П., заробітною платою 3200 грн. і керівником Мацько А.Д. необхідно сформувати такий шаблон:

EMP	ПІБ	Зарплата	Керівник	Відділ
I.	Антонюк О.П.	3200	Мацько А.Д.	Іграшки

Для того, щоб *видалити* всю інформацію про співробітників відділу «Іграшки», сформуємо такий шаблон:

EMP	ПІБ	Зарплата	Керівник	Відділ
D.				«Іграшки»

У випадку *зміни* заробітної плати співробітника Мацька А.Д. шаблон запиту матиме вигляд:

EMP	ПІБ	Зарплата	Керівник	Відділ
U.	«Мацько А.Д.»	6400		

Порожнє поле означає, що воно не підлягає зміні. Якщо потрібно змінити деяке значення на «порожнє», використовується ключове слово NULL.

Щоб підвищити зарплату співробітникам відділу «Іграшки» на 10%, можна сформувати шаблон запиту на модифікацію такого виду:

EMP	ПІБ	Зарплата	Керівник	Відділ
U.		1,1*<u>**zp**</u>		
		<u>**zp**</u>		«Іграшки»

Реалізація цього запиту відбувається в два етапи: спочатку вибираються всі записи зі значенням «іграшки» в полі Відділ, а потім відбувається зміна поля Зарплата відібраних записів на нове значення.

1.3 Базові оператори мови SQL та особливості їх запису

Оператори мови SQL можна умовно розділити на чотири підмови:

– мова визначення даних (Data Definition Language – DDL);

– мова маніпулювання даними (Data Manipulation Language – DML);

– мова управління даними (Data Control Language – DCL);

– мова управління транзакціями (Transaction Control Language – TCL).

Базові оператори мови SQL наведено в табл. 1.1

Таблиця 1.1 – Базові оператори мови SQL

Підмова	Назва оператора	Призначення
DDL	CREATE TABLE	створення таблиці
	ALTER TABLE	зміна структури таблиці
	DROP TABLE	видалення таблиці
DML	SELECT	вибірка з вмісту таблиці
	INSERT	порядкове додавання даних в таблицю
	UPDATE	модифікація даних таблиці
	DELETE	видалення рядків таблиці
DCL	GRANT	надання прав користувачу
	DENY	явна заборона для користувача
	REVOKE	відміна заборони/дозволу користувачу
TCL	BEGIN TRANSACTION	початок транзакції
	COMMIT	збереження змін, внесений транзакцією
	ROLLBACK	відкат, повертає вміст таблиці БД в її початковий стан, зафіксований останньою командою COMMIT

Перш, ніж детально розглянути приклади використання операторів мови SQL, варто ознайомитися з особливостями їх запису.

У кожного об'єкту в БД є унікальне ім'я. Відповідно до стандарту ANSI/ISO, в SQL імена повинні містити від 1 до 18 символів, починатися з літери і не містити пропуски або спеціальні символи пунктуації. У стандарті SQL2 максимальне число символів в імені збільшене до 128, при цьому на практиці в різних СУБД дозволене використання в іменах таблиць спеціальних символів. Тому для підвищення переносності краще робити імена порівняно короткими і уникати використання в них спеціальних символів.

Стандарт ISO SQL не підтримує такі формальні терміни, як відношення, атрибут і кортеж. Замість них використовують терміни таблиця, стовпець, рядок.

Оператор SQL складається з зарезервованих слів, а також зі слів, що визначаються користувачем. Зарезервовані слова слід записувати саме так, як зазначено в стандарті, і не можна розбивати на частини для переносу з одного рядка на інший. Слова, визначені користувачем, задаються самим користувачем (у відповідності з певними синтаксичними правилами) і являють собою імена різних об'єктів бази даних – таблиць, стовпців, представлень, індексів і т.д.

Хоча в стандарті це не зазначено, однак у багатьох діалектах мови SQL вимагають завдання в кінці оператора деякого символу, що позначає закінчення його тексту; як правило, з цією метою використовується крапка з комою (;).

Більшість компонентів операторів SQL не відчутно до регістру. Одним важливим виключенням з цього правила є *символьні літерали* – дані, які повинні вводитися точно так, як були введені відповідні їм значення, що зберігаються в базі даних. Наприклад, якщо в базі даних зберігається значення прізвища 'ІВАНОВ', а в умові пошуку вказано символьний літерал 'Іванов', то цей запис не буде знайдений. Всі нечислові значення даних завжди повинні бути взяті в одинарні лапки, а всі числові дані не повинні бути взяті в одинарні лапки. Нижче наведено приклад використання літералів для вставки даних в таблицю.

Оскільки мова SQL має вільний формат, окремі оператори SQL і їх послідовності будуть мати більш зручний для читання вигляд при використанні відступів і вирівнювання. Рекомендується дотримуватися наступних правил:

- кожна конструкція в операторі повинна починатися з нового рядка;
- початок кожної конструкції має бути позначено таким же відступом, що і початок інших конструкцій оператора;
- якщо конструкція складається з декількох частин, кожна з них повинна починатися з нового рядка з деяким відступом відносно початку конструкції, що буде вказувати на їх підпорядкованість.

Найчастіше для визначення формату операторів SQL використовують розширену форму системи позначень BNF (Backus Naur Form – форма Бекуса-Наура), відповідно до якої

– прописні літери використовуються для запису зарезервованих слів;
– рядкові літери використовуються для запису слів, що визначаються користувачем;
– фігурні дужки визначають обов'язковий елемент, наприклад {a};
– квадратні дужки визначають необов'язковий елемент, наприклад [a].

Багато крапок (...) використовується для вказання необов'язкової можливості повторення конструкції від нуля до декількох разів.

1.3.1 Оператори мови визначення даних

Оператор створення таблиці має формат:

```
CREATE TABLE <ім'я таблиці>
   (<ім'я стовпця><тип даних>[NOT NULL]
    [,<ім'я стовпця><тип даних>[NOT NULL]]…);
```

Обов'язковими операндами оператора є ім'я створюваної таблиці та ім'я хоча б одного стопвця (поля) з вказанням типу даних, що зберігати-

муться у цьому стовпці. При створенні таблиці для окремих полів можуть вказуватись деякі додаткові правила контролю введення в них даних, наприклад конструкція NOT NULL (не пусте) означає, що в цьому стовці повинно бути визначене значення.

Приклад 1.15. Створення таблиці.

Нехай потрібно створити таблицю goods опису товарів, яка містить поля: type – вид товару, comp_id – ідентифікатор компанії-виробника, name – назва товару і price – ціна.

Оператор створення таблиці goods матиме вигляд:

```
CREATE TABLE goods
   (type SQL_CHAR(8) NOT NULL,
   comp_id SQL_CHAR(10) NOT NULL,
   name SQL_VARCHAR(20),
   price SQL_DECIMAL(8,2));
```

Формат оператора зміни структури таблиці має вигляд:

```
ALTER TABLE < ім'я таблиці >
   ({ADD,MODIFY,DROP} <і'мя стовпця> [<тип даних>][NOT NULL]
    [,{ADD,MODIFY,DROP} < і'мя стовпця > [<тип даних >][NOT NULL]]…);
```

Зміна структури таблиці може полягати в додаванні (ADD), зміні (MODIFY) або видаленні (DROP) одного чи декількох стовпців таблиці. Правила запису оператора ALTER TABLE такі ж, як і оператора CREATE TABLE. При видаленні стовпця тип даних вказувати не потрібно.

Приклад 1.16. Додавання поля таблиці.

Нехай в раніше створену таблицю необхідно додати поле number, яке відводиться для зберігання величини запасу товару.

Оператор модифікації таблиці goods матиме вигляд:

```
ALTER TABLE goods (ADD number SQL_INTEGER);
```

Оператор видалення таблиці має вигляд:

```
DROP TABLE <ім'я таблиці>;
```

Оператор дозволяє видалити наявну таблицю. Наприклад, для видалення таблиці з ім'ям goods достатньо записати оператор виду:

```
DROP TABLE goods;
```

Для прискорення виконання запитів і операцій пошуку для одного або декількох стовпців таблиці можуть створюватись індекси. Створення та видалення індексів відбувається за допомогою операторів CREATE INDEX і DROP INDEX відповідно.

1.3.2 Оператори мови маніпулювання даними

Серед операторів мови маніпулювання даними найчастіше використовується оператор формування запитів SELECT. Зважаючи на його важливість, особливості використання даного оператора розглядатимуться у підрозділі 1.4.

Для додавання нових записів в таблицю використовується оператор INSERT INTO, який має такий формат:

```
INSERT INTO <ім'я таблиці>          або    INSERT INTO <ім'я таблиці>
   [(<список стовпців>)]                      [(<список стовпців>)]
  VALUES (<список значень>)                   <вираз SELECT>
```

У першому форматі оператор INSERT призначений для введення нових записів з заданими значеннями у стовпцях. Порядок перерахування імен стовпців повинен відповідати порядку значень, перерахованих в списку операнда VALUES. Якщо <список стовпців> опущений, то у <списку значень> повинні бути перераховані всі значення в порядку стовпців структури таблиці.

У другому форматі оператор INSERT забезпечує можливість введення в задану таблицю нових рядків, які відібрані з іншої таблиці за допомогою виразу SELECT.

Приклад 1.17. Введення записів.

Ввести в таблицю EMP запис про нового співробітника.

Оператор INSERT матиме такий вигляд:

```
INSERT INTO emp
   VALUES ('Ivanov', 7500, 'Lee', 'cosmetics');
```

Для внесення змін у наявні записи використовується оператор UPDATE, який має формат виду:

```
UPDATE <ім'я таблиці>
   SET <ім'я стовпця> = {<вираз>, NULL}
   [, SET <ім'я стовпця> = {<вираз>, NULL}…]
   [WHERE <умова>];
```

Виконання оператора UPDATE полягає у зміні значення у визначених оператором SET стовпцях таблиці для тих записів, які задовільняють умові, заданій оператором WHERE. Нові значення полів в записах можуть бути порожніми (NULL), або обраховуватись у відповідності до арифметичного виразу.

Приклад 1.18. Зміна записів.

Нехай необхідно збільшити на 500 грн. заробітну плату тим співробітникам, які отримують 3200 грн. (по таблиці EMP).

Запит, сформульований за допомогою оператора UPDATE, може виглядати так:

```
UPDATE emp
   SET sal=3700
   WHERE sal=3200;
```

Видалення записів з таблиці можливо здійснити за допомогою оператора DELETE, який має формат:

```
DELETE FROM <ім'я таблиці>
   [WHERE <умова>];
```

Результатом виконання оператора DELETE є видалення з вказаної таблиці рядків, які задовольняють умові, визначеній операндом WHERE. Якщо операнд WHERE опущений, видаленню підлягають всі записи таблиці.

Приклад 1.19. Видалення записів.

У звязку з ліквідацією відділу «Іграшки» (toy), потрібно видалити з таблиці EMP всіх співробітників цього відділу.

Оператор DELETE для цієї задачі буде мати вигляд:

```
DELETE FROM emp
   WHERE dept='toy';
```

У прикладі dept – поле для введення назви відділу.

1.4 Формування запитів мовою SQL

Вибірку і відображенні даних однієї або декількох таблиць бази даних виконують за допомогою оператора SELECT, який є найбільш затребуваним оператором мови SQL. Загальний формат оператора SELECT має наступний вигляд:

```
SELECT [DISTINCT | ALL] <список даних>
  FROM <список таблиць>
  [WHERE <умова вибірки>]
  [GROUP BY <ім'я стовпця>[, <ім'я стовпця>]…]
  [HAVING <умова пошуку>]
  [ORDER BY <специфікація>[, <специфікація>]…];
```

Обробка елементів оператора SELECT виконується в наступній послідовності.

1. **FROM.** Визначаються імена використовуваної таблиці або декількох таблиць.

2. **WHERE.** Виконується фільтрація рядків об'єкта відповідно до заданими умовами.

3. **GROUP BY.** Утворюються групи рядків, що мають одне і те ж значення в зазначеному стовпці.

4. **HAVING.** Фільтруються групи рядків об'єкта відповідно до зазначеної умови.

5. **SELECT.** Встановлюється, які стовпці повинні бути присутніми у вихідних даних.

6. **ORDER BY.** Визначається впорядкованість результатів виконання оператора.

Порядок конструкцій в операторі SELECT не може бути змінений. Тільки дві конструкції оператора – SELECT і FROM – є обов'язковими, решта конструкцій можуть бути опущені.

Для того, всю інформацію про всіх співробітників компанії, необхідно перерахувати повний список стовпців таблиці в операторі **SELECT**

```
SELECT staffNo, fName, lName, position, sex, DOB, salary, branchNo
  FROM Staff;
```

або замість імен стовпців вказати символ зірочки (*)

```
SELECT *
  FROM Staff;
```

Обидва варіанти запиту є повністю еквівалентними один одному.

По замовчуванню в результуючу таблицю включаються усі рядки, навіть повторювані, що відповідає ключовому слову ALL. Для видалення з результуючої таблиці повторюваних рядків використовується ключове слово DISTINCT:

```
SELECT DISTINC propertyNo
  FROM Viewing;
```

Список даних може містити імена стовпців, що приймають участь в запиті, а також арифметичні вирази над стовпцями

1.4.1 Обчислення в запитах

У загальному випадку для створення обчислюваного поля в списку оператора SELECT слід вказати деякий вираз мови SQL. У цих виразах можуть застосовуватися операції додавання, віднімання, множення і ділення. При побудові складних виразів можуть використовуватися круглі дужки. Для отримання значення обчислюваного поля може використовуватися значення з декількох стовпців таблиці, однак тип даних стовпців, які входять в арифметичний вираз, обов'язково повинен бути цифровим.

Зазвичай стовпцям результуючої таблиці присвоюються імена відповідних їм стовпців вихідних таблиць бази даних. Однак у випадку з обчислюваними полями це правило не застосовується, оскільки в стандарті SQL не визначені правила іменування похідних стовпців. В одних діалектах мови SQL імена таким стовпцями привласнюють відповідно до порядку їх розташування в таблиця (наприклад, col4), в інших діалектах у подібного стовпець ім'я може бути зовсім відсутнім або замість нього може використовуватися вираз, що записаний в списку SELECT.

Приклад 1.20. Обчислювані поля

Створити звіт про річну заробітну плату співробітників з вказанням табельного номера, імені, прізвища та суми заробітної плати.

```
SELECT staffNo, fName, lName, salary*12
   FROM Staff;
```

Результат виконання запиту може мати такий вигляд:

staffNo	fName	lName	col4 або salary*12 або NULL
001	Сергій	Мацько	38400
002	Галина	Антонюк	42000
003	Іван	Костецький	39600

Стандарт ISO дозволяє явно задавати інші імена стовпців результуючої таблиці, для чого застосовується конструкція AS. При використанні цієї конструкції наведений вище оператор SELECT може бути переписаний таким чином чином:

```
SELECT staffNo, fName, lName, salary*12 AS yearSalary
   FROM Staff;
```

1.4.2 Вибірка рядків конструкцією WHERE

У наведених вище прикладах в результаті виконання операторів SELECT вибиралися всі рядки зазначеної таблиці. Для того, щоб обмежити набір рядків, які мають бути поміщені в результуючу таблицю запиту, використовується конструкція WHERE. Вона складається з ключового слова WHERE, за яким слідує перелік умов пошуку, що визначають ті рядки, які повинні бути обрані при виконанні запиту.

Розрізняють п'ять основних типів умов пошуку (або предикатів, якщо користуватися термінологією ISO):

– *Порівняння.* Порівнюються результати обчислення одного виразу з результатами обчислення іншого виразу.

– *Діапазон.* Перевіряється, чи потрапляє результат обчислення виразу в заданий діапазон значень.
– *Належність до множини*. Перевіряється, чи належить результат обчислення виразу до заданої множини значень.
– *Відповідність шаблону*. Перевіряється, чи відповідає деяке рядкове значення заданому шаблону.
– *Значення NULL*. Перевіряється, чи містить даний стовпець NULL (Невизначене значення).

В мові SQL можна використовувати прості оператори порівняння такі як більше (>), менше (<), дорівнює (=), не дорівнює (< > або !=), більше або дорівнює (>=), менше або дорівнює (<=).

Приклад 1.21. Умова пошуку шляхом порівняння

Скласти список співробітників з розміром заробітної плати більше 7000 грн. в місяць.

```
SELECT staffNo, fName, lName, position, salary
  FROM Staff
    WHERE salary > 10000;
```

Більш складні умови пошуку можуть бути побудовані з використанням логічних операторів AND, OR і NOT, а також за допомогою круглих дужок. Так, якщо необхідно скласти список співробітників, які отримують заробітну плату 3200 грн. або 6400 грн., то запит матиме вигляд:

```
SELECT staffNo, fName, lName, position, salary
  FROM Staff
    WHERE salary = 3200 OR salary = 6400;
```

Приклад 1.22. Використання діапазону (оператор BETWEEN/ NOT BETWEEN)

Скласти список співробітників, чия заробітна плата становить від 3200 грн. до 10000 грн.

```
SELECT staffNo, fName, lName, position, salary
  FROM Staff
    WHERE salary  BETWEEN 3200 AND 10000;
```

Конструкція NOT BETWEEN має протилежне значення і використовується у випадках, коли потрібно відібрати ті значення, що лежать за межами заданого діапазону.

Приклад 1.23. Перевірка належності до множини (оператор IN/NOT IN).

Скласти список всіх керівників та їх замісників.

```
SELECT staffNo, fName, IName, position
  FROM Staff
    WHERE position IN ('Директор', 'Замісник директора');
```

Для відбору будь-яких значень, крім тих, що вказані у списку, використовується конструкція NOT IN.

Оператори BETWEEN та IN несуттєво підвищують виразність мови SQL, тому на практиці часто їх заміняють логічними операторами. Запит з прикладу 1.22 можна записати так:

```
SELECT staffNo, fName, IName, position, salary
  FROM Staff
    WHERE salary >=3200 AND salary <=10000;
```

Запит з прикладу 1.23 може мати такий вигляд:

```
SELECT staffNo, fName, IName, position
  FROM Staff
    WHERE position= 'Директор' OR position= 'Замісник директора';
```

Використання оператора IN є більш доцільним у випадках, коли набір допустимих значень є достатньо великим.

Приклад 1.24. Умова пошуку з вказанням шаблону (LIKE/NOT LIKE).

Вивести список всіх власників квартир, що здаються в оренду у місті Вінниця.

При виконанні цього запиту необхідно організувати пошук рядків, де назва міста 'Вінниця' може знаходитись у будь-якому місці значення стовпця address таблиці PrivatOwner. В мові SQL існує два спеціальні символи шаблону, які використовуються для перевірки символьних значень:

- %. Символ відсотка замінює будь-яку послідовність з нуля і більше символів.
- _ . Символ підкреслення замінює будь-який окремий символ.
- address LIKE 'B%'. Такий шаблон означає, що перший символ значення повинен обов'язково бути символом 'B', а решта символів значення не мають і не перевіряються.
- address LIKE 'B___'. Такий шаблон означає, що значення повинно мати довжину в 4 символи, причому першим символом значення повинен обов'язково бути символом 'B'.
- address LIKE '%a'. Такий шаблон визначає будь-яку послідовність символів довжиною не менше одного символу, причому останнім символом обов'язково має бути символ 'a'.

– address LIKE ‘%Вінниця%’. Такий шаблон означає, що шукане значення має будь-яку послідовність символів, яка включає підрядок ‘Вінниця’.
– address NOT LIKE ‘B%’. Такий шаблон означає, що необхідно знайти будь-які рядки, які не починаються з символу ‘B’.

Якщо шуканий рядок повинен включати службовий символ ‘%’ або ‘ _’, то за допомогою конструкції ESCAPE визначається деякий «маскуючий» символ, який вказує, що наступний за ним символ більше не має спеціального значення. Наприклад, для перевірки значення на відповідність рядку ‘45%’ можна скористатись такою умовою:

```
LIKE ‘45#%’ ESCAPE ‘#’
```

За допомогою механізму пошуку по шаблону запит виведення списку всіх власників квартир, що здаються в оренду у місті Вінниця, матиме такий вигляд:

```
SELECT ownerNo, fName, lName, address, telNo
  FROM PrivateOwner
    WHERE address LIKE ' %Вінниця%';
```

Приклад 1.25. Використання значення NULL в умовах пошуку (IS NULL/IS NOT NULL).

Скласти список всіх відвідувань об’єкта з номером ‘КР4’, по яким не було залишено коментарів.

```
SELECT clientNo, viewDate
   FROM Viewing
     WHERE propertyNo = 'KP4' AND comment IS NULL;
```

Для перевірки наявності у стовпці значень, відмінних від NULL, може використовуватись конструкція IS NOT NULL.

1.4.3 Сортування результатів (конструкція ORDER BY)

В деяких СКБД може бути передбачено застосування за замовчуванням певного способу впорядкування рядків результуючої таблиці, наприклад за первинним ключем. Однак їх можна впорядкувати належним чином, для чого в операторі SELECT поміщають конструкцію ORDER BY.

Конструкція ORDER BY включає список розділених комами ідентифікаторів стовпців, за якими потрібно впорядкувати результуючу таблицю запиту. Ідентифікатором стовпця може бути або його ім'я, або номер, який позначає елемент списку SELECT відповідно до його позиції в цьому списку. Крайній лівий елемент списку має номер 1, наступний – 2 і т.д. Номера стовпців можуть використовуватися в тих випадках, коли стовпі,

за якими слід упорядкувати результат, є обчислюваними, а конструкція AS із зазначенням імені цього стовпця в операторі SELECT відсутня.

Конструкція ORDER BY дозволяє упорядкувати вибрані записи в порядку зростання (ASC) чи зменшення (DESC) значень будь-якого стовпця або комбінації стовпців, незалежно від того, чи присутні ці стовпці в таблиці результатів чи ні.

Приклад 1.26. Сортування за значенням одного стовпця.

Скласти звіт про середні бали студентів груп ПІ-13, розмістивши рядки за спаданням значення середнього балу.

```
SELECT name, group, averageMark
  FROM Student
    WHERE group LIKE '_ПІ-13%'
      ORDER BY averageMark DESC;
```

Конструкція ORDER BY для прикладу 1.26 може бути записана і у вигляді ORDER BY 3 DESC, де 3 означає третій стовпець у списку вибірки оператора SELECT, тобто стовпець averageMark.

Приклад 1.27. Сортування за значеннями декількох стовпців.

Скласти список квартир, що здаються в оренду, відсортувавши рядки за зростанням кількості кімнат, причому спочатку потрібно відображати найдорожчі квартири.

```
SELECT flatNo, rooms, rent
  FROM FlatForRent
    ORDER BY rooms, rent DESC;
```

Ключове слова ASC після стовпця rooms може бути пропущене, оскільки воно передбачається за замовчуванням.

Конструкція ORDER BY завжди повинна бути останнім елементом в операторі SELECT.

У стандарті ISO зазначено, що значення NULL в стовпці або виразах, для сортування яких застосовується конструкція ORDER BY, повинні розглядатися або як менші, або як більші за величиною, ніж всі непусті значення. Вибір того чи іншого варіанту залежить від розробників СКБД.

1.4.4 Використання вбудованих функцій

Стандарт ISO містить визначення таких п'яти вбудованих функції:

– COUNT – повертає кількість значень у вказаному стовпці;

– SUM – повертає суму значень в зазначеному стовпці;

– AVG – повертає середнє значення в зазначеному стовпці;

– MIN – повертає мінімальне значення в зазначеному стовпці;

– MAX – повертає максимальне значення у вказаному стовпці.

Всі ці функції оперують зі значеннями в єдиному стовпці таблиці і повертають єдине значення. Функції COUNT, MIN і MAX застосовуються як до числових, так і до нечислових полів, тоді як функції SUM і AVG – лише до числових полів. Варіант COUNT (*) є особливим випадком використання функції COUNT – його призначення полягає в підрахунку всіх рядків в таблиці, незалежно від того, містяться там порожні, повторювані або будь-які інші значення. У випадку використання всіх інших функцій спочатку виключаються всі порожні значення.

Якщо до застосування вбудованої функції необхідно виключити повторювані значення, слід перед ім'ям стовпця у визначенні функції помістити ключове слово DISTINCT. Стандарт ISO допускає використання ключового слова ALL з метою явної вказівки того, що виключення повторюваних значень проводити не потрібно, хоча це ключове слово мається на увазі за замовчуванням.

Вбудовані функції можуть використовуватися тільки в списку вибірки оператора SELECT і в конструкції HAVING (див. п.1.4.5). Якщо в операторі SELECT не використовується конструкція GROUP BY, то у його списку вибірки не може бути присутнім жодний стовпець, який не є параметром вбудованої функції. Тобто, запит, що наведений нижче є некоректним:

```
SELECT flatNo, AVG(rent)
   FROM FlatForRent;
```

Помилка полягає в тому, що в запиті відсутня конструкція GROUP BY, а звернення до стовпця flatNo відбувається без вбудованої функції.

Приклади 1.28. Використання функції COUNT.

Визначити, у скількох квартир, що здаються в оренду, орендна плата перевищує 4000 грн.

```
SELECT COUNT(*) AS count
   FROM FlatForRent
      WHERE rent > 4000;
```

Визначити, скільки різних квартир було переглянуто в січні 2017 року.

```
SELECT COUNT(DISTINCT flatNo) AS count
   FROM Viewing
      WHERE ViewDate BETWEEN ‘1-January-17’ AND ’31-January-17’;
```

Приклад 1.29. Використання функції SUM.

Визначити загальну кількість директорів компанії та загальну суму їх місячної заробітної плати.

```
SELECT COUNT(staffNo) AS count, SUM(salary) AS sum
   FROM Staff
      WHERE position = 'Директор';
```

Приклад 1.30. Використання функцій MIN, MAX і AVG.

Обчислити значення максимальної, мінімальної та середньої заробітної плати співробітників компанії.

```
SELECT MAX(salary) AS max, MIN(salary) AS min, AVG(salary) AS avg
   FROM Staff;
```

В даному прикладі необхідно опрацювати відомості про всіх співробітників компанії, тому використовувати конструкцію WHERE не потрібно.

1.4.5 Групування результатів (конструкція GROUP BY/HAVING)

Запит, в якому присутня конструкція GROUP BY, називається групуючим запит, оскільки в ньому групуються дані, отримані в результаті виконання операції SELECT, після чого для кожної окремої групи створюється єдиний підсумковий рядок.

При використанні в операторі SELECT конструкції GROUP BY кожен елемент списку в списку вибірки SELECT повинен мати єдине значення для всієї групи.

Всі імена стовпців, наведені в списку вибірки SELECT, повинні бути присутніми і в конструкції GROUP BY, за винятком випадків, коли ім'я стовпця використовується тільки у вбудованій функції. В конструкції ж GROUP BY можуть бути присутніми імена стовпців, які відсутні в списку вибірки SELECT.

Якщо спільно з конструкцією GROUP BY використовується конструкція WHERE, то вона обробляється в першу чергу, а групуванню піддаються тільки ті рядки, які задовольняють умові пошуку.

Стандарт ISO визначає, що при проведенні групування все порожні значення розглядаються як рівні. Якщо два рядки таблиці в групуючому стовпці містять значення NULL та ідентичні значення у всіх інших непустих групуючих стовпцях, вони поміщаються в одну і ту ж групу.

Приклад 1.31. Використання конструкції GROUP BY.

Визначити чисельність персоналу у кожному відділі, а також їх загальну заробітну плату.

```
SELECT branchNo, COUNT(staffNo) AS count, SUM(salary) AS sum
   FROM Staff
      GROUP BY branchNo
         ORDER BY branchNo;
```

Результат виконання такого запиту може мати вигляд:

branchNo	count	sum
Vin01	8	29000
Vin02	10	35000
Vin03	12	42000

При обробці цього запиту виконуються наступні дії.

1. Рядки таблиці Staff розподіляються в групи відповідно до значення в стовпці branchNo компанії. У межах кожної з груп виявляються дані про весь персонал одного з відділень компанії.

2. Для кожної з груп обчислюються загальна кількість рядків, рівне чисельності працівників відділення, а також сума значень в стовпці заробітної плати, яка і є шуканою сумою заробітної плати всіх працівників відділення. Потім генерується єдиний підсумковий рядок для всієї групи вихідних рядків.

3. Отримані рядки результуючої таблиці сортуються в порядку зростання номер відділення, зазначеного в стовпці branchNo.

Конструкція HAVING призначена для використання спільно з конструкцією GROUP BY для завдання обмежень, що зазначені з метою відбору тих груп, які будуть поміщені в результуючу таблицю запиту.

Стандарт ISO вимагає, щоб імена стовпців, що застосовуються в конструкції HAVING, обов'язково були присутні в списку елементів GROUP BY або використовувались у вбудованій функції. На практиці умови пошуку в конструкції HAVING завжди включають, щонайменше, одну вбудовану функцію. Інакше ці умови пошуку повинні бути поміщені в конструкції WHERE і застосовані для відбору окремих рядків.

Приклад 1.32. Використання конструкції HAVING.

Для кожного відділення компанії з чисельністю працівників більше 9 визначити чисельність персоналу, а також їх загальну заробітну плату.

```
SELECT branchNo, COUNT(staffNo) AS count, SUM(salary) AS sum
   FROM Staff
      GROUP BY branchNo
         HAVING COUNT(staffNo) > 9
            ORDER BY branchNo;
```

Результат виконання такого запиту може мати вигляд:

branchNo	count	sum
Vin02	10	35000
Vin03	12	42000

Конструкція HAVING не є обов’язковою частиною мови SQL – будь-який запит, написаний з використанням конструкції HAVING, може бути поданий в іншому вигляді, без її застосування.

1.4.6 Вкладені запити (підзапити)

Стандарт SQL допускає розміщення в операторі SELECT вкладених запитів. Зовнішній оператор SELECT використовує результат виконання внутрішнього оператора для визначення змісту остаточного результату всієї операції. Внутрішні запити
можуть знаходитися в конструкціях WHERE і HAVING, а також у списку вибірки зовнішнього оператора SELECT. В цьому випадку вони отримують назву підзапитів, або вкладених запитів. Також, внутрішні оператори SELECT можуть використовуватися в операторах INSERT, UPDATE і DELETE.

Існує три типи підзапитів:

1. *Скалярний підзапит* повертає значення, що отримується з перетину одного стовпця з одним рядком, тобто єдине значення. Варіанти використання скалярних підзапитів наведені в прикладах 1.28.
2. *Рядковий підзапит* повертає значення декількох стовпців таблиці, але у вигляді єдиного рядка. Варіант рядкового підзапиту наведено в прикладах 1.29 і 1.30.
3. *Табличний підзапит* повертає значення одного або декількох стовпців таблиці, що розміщені в більш ніж одному рядку.

Приклад 1.33. Використання скалярного підзапиту в конструкції WHERE.

Скласти список співробітників, які працюють у відділенні компанії, яке розташоване по вулиці 'Хмельницьке шосе, 90'.

```
SELECT staffNo, fName, lName, position
  FROM Staff
    WHERE branchNo = (SELECT branchNo
                        FROM Branch
                          WHERE street = 'Хмельницьке шосе, 90');
```

Внутрішній оператор SELECT призначений для визначення номера відділення компанії, розташованого по вулиці 'Хмельницьке шосе, 90'. Після отримання номера необхідного відділення виконується зовнішній підзапит, призначений для вибірки відомостей про працівників цього відділення.

В результаті зовнішній оператор SELECT набуває вигляду:

```
SELECT staffNo, fName, lName, position
  FROM Staff
    WHERE branchNo = 'Vin03';
```

Підзапит можна вказувати безпосередньо після операторів порівняння (тобто операторів =, <,>, <=,> =, <>) в конструкції WHERE або HAVING. Текст під запиту повинен бути розміщений в круглих дужках.

Приклад 1.34. Використання підзапитів з вбудованими функціями.

Скласти список всіх співробітників, які мають заробітну плату вище середньої, зазначивши, на скільки їх заробітна плата перевищує середню.

```
SELECT staffNo, fName, lName, position, salary - (SELECT AVG(salary)
                                                  FROM Staff) AS salDiff
  FROM Staff
    WHERE salary > (SELECT AVG(salary)
                    FROM Staff);
```

Необхідно відзначити, що не можна безпосередньо включити в запит вираз WHERE salary > AVG (salary), оскільки застосовувати вбудовані функції в конструкції WHERE заборонено. Для досягнення бажаного результату слід створити підзапит, що обчислює середнє значення заробітної плати, а потім використовувати його в зовнішньому операторі SELECT, призначеному для вибірки відомостей про тих працівниках компанії, чия зарплата перевищує це середнє значення.

До підзапитів застосовуються такі *правила і обмеження*.

1. У підзапитах не повинна використовуватися конструкція ORDER BY, хоча вона може бути присутньою в зовнішньому операторі SELECT.

2. За замовчуванням імена стовпців в підзапиті відносяться до таблиці, ім'я якої зазначено в конструкції FROM підзапиту. Однак дозволяється посилатися і на стовпці таблиці, зазначеної в конструкції FROM зовнішнього запиту, для чого використовуються уточнені імена стовпців (як описано нижче).

3. Якщо підзапит є одним з двох операндів, що беруть участь в операції порівняння, то підзапит повинен вказуватися в правій частині цієї операції.

З підзапитами, які повертають один стовпець чисел, можуть використовуватися ключові слова ANY і ALL. Якщо підзапиту передуватиме ключове слово ALL, умова порівняння вважається виконаною тільки в тому випадку, якщо вона виконується для всіх значень в результуючому стовпці підзапиту. Якщо підзапиту передує ключове слово ANY, то умова порівняння буде вважатися виконаною, якщо вона задовольняється хоча б для будь-якого (одного або декількох) значень в результуючому стовпці підзапиту. Якщо в результаті виконання підзапиту буде отримано порожнє значення, то для ключового слова ALL умова порівняння буде вважатися виконаною, а для ключового слова ANY – невиконаною. Відповідно до стандарту ISO додатково можна використовувати ключове слово SOME, що є синонімом ключового слова ANY.

Приклад 1.35. Використання ключового слова ANY/ SOME.

Знайти всіх співробітників, чия заробітна плата перевищує заробітну плату хоча б одного співробітника відділення компанії з номером 'Vin03'.

```
SELECT staffNo, fName, IName, position, salary
   FROM Staff
     WHERE salary > SOME (SELECT salary
                          FROM Staff
                            WHERE branchNo = 'Vin03');
```

Цей запит міг бути записаний з використанням підзапиту, що визначає мінімальну зарплату персоналу відділення під номером 'Vin03', після чого зовнішній підзапит зможе вибрати відомості про весь персонал компанії, чия зарплата перевищує це значення (див. приклад 1.34).

Приклад 1.36. Використання ключового слова ALL.

Знайти всіх співробітників, чия заробітна плата перевищує заробітну плату будь-якого співробітника відділення компанії з номером 'Vin03'.

```
SELECT staffNo, fName, INarae, position, salary
   FROM Staff
     WHERE salary > ALL(SELECT salary
                        FROM Staff
                          WHERE branchNo = 'Vin03');
```

В даному випадку можна було б використовувати підзапит, який визначає максимальне значення зарплати персоналу відділення під номером 'Vin03', після чого за допомогою зовнішнього запиту вибрати відомості про всіх працівників компанії, зарплата яких перевищує це значення.

Контрольні питання

1. Коротко охарактеризуйте мову формування запитів QBE.
2. Наведіть приклади формування простих запитів мовою QBE.
3. Що називають обчислюваним полем в мові QBE? Наведіть приклади.
4. Особливості використання вбудованих функцій в мові QBE.
5. У яких випадках в мові QBE використовується блок умов?
6. Проілюструйте використання елементів прикладу в мові QBE.
7. Як в мові QBE додавати, видаляти та модифікувати записи?
8. Охарактеризуйте переваги мови SQL.
9. Коротко охарактеризуйте структуру мови SQL.
10. Наведіть приклади використання операторів мови DDL.
11. Опишіть структуру оператора SELECT. Наведіть приклад простого запиту.
12. Які види умов пошуку ви знаєте? Наведіть приклади використання конструкції WHERE.
13. Які особливості використання конструкції ORDER BY?
14. Які вбудовані функції містить мова SQL? Наведіть приклади.
15. Наведіть приклад використання конструкції GROUP BY.
16. Що таке підзапит? Які види підзапитів в знаєте?

2 УПРАВЛІННЯ ТРАНЗАКЦІЯМИ

2.1 Поняття та властивості транзакції

Транзакція – це дія або ряд дій, що виконуються одним користувачем або прикладною програмою, які здійснюють читання або модифікацію вмісту бази даних.

Транзакція є логічною одиницею роботи, що 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 в базі даних.

Будь-яка транзакція повинна володіти чотирма властивостями. Основні властивості транзакцій прийнято позначати абревіатурою ACID (Atomacity, Consistency, Isolation, Durability – нерозривність, узгодженість, ізольованість та стійкість).

Нерозривність – це властивість, для опису якої найкраще підходить вислів «все або нічого». Будь-яка транзакція являє собою неподільну одиницю роботи, яка може бути виконана вся повністю, або невиконана взагалі. За забезпечення нерозривності відповідає підсистема відновлення СКБД.

Властивість ***узгодженості*** означає, що кожна транзакція повинна переводити базу даних з одного узгодженого стану в інший. Відповідальність за забезпечення властивості узгодженості покладається на СКБД та на розробників додатку. В СКБД узгодженість може забезпечуватись шляхом виконання всіх обмежень, заданих в схемі бази даних, таких як обмеження цілісності та обмеження предметної області. Однак подібна умова не є достатньою для забезпечення узгодженості. Наприклад, деяка транзакція призначена для переведення грошових коштів з одного банківського рахунку на інший, але програміст зробив помилку в логіці транзакції та передбачив зняття грошей з правильного рахунку, а зарахування – на неправильний рахунок. В такому випадку база даних переходить в неузгоджений стан, незважаючи на наявність правильно заданих обмежень. Відповідальність за усунення такої неузгодженості не може бути покладена на СКБД, оскільки в ній відсутні механізми виявлення подібних логічних помилок.

Ізольованість передбачає, що всі транзакції виконуються незалежно одна від одної. Іншими словами, проміжні результати незавершеної транзакції не повинні бути доступними для інших транзакцій. За забезпечення ізольованості відповідає підсистема управління паралельним виконанням.

Стійкість. Результати успішно завершеної (зафіксованої) транзакції повинні зберігатись в базі даних постійно і не повинні бути втрачені в ре-

зультаті майбутніх збоїв. За забезпечення стійкості відповідає підсистема відновлення.

2.2 Порядок виконання операцій транзакції

Будь-яка транзакція завершується одним із двох можливих способів. У випадку успішного завершення результати транзакції *фіксуються* (commit) в базі даних, і остання переходить в новий узгоджений стан. Якщо виконання транзакції не завершилось успіхом, вона відміняється. В цьому випадку в базі даних повинен бути відновлений той узгоджений стан, в якому вона знаходилась до початку даної транзакції. Цей процес називається *відкатом* (roll back), або *відміною транзакції*.

Зафіксована транзакція не може бути відмінена. Якщо виявиться, що зафіксована транзакція була помилковою, потрібно виконати іншу транзакцію, яка відмінить дії першої транзакції. Така транзакція називається *компенсуючою*. Однак, транзакція, що завершилась аварійно, і для якої був виконаний відкат, може бути викликана на виконання пізніше і, в залежності від причин попередньої відмови, цілком успішно завершена та зафіксована в базі даних.

В жодній СКБД не може бути передбачений апріорний спосіб визначення того, які саме операції оновлення можуть бути згруповані для формування єдиної логічної транзакції. Тому повинен застосовуватись метод, який дозволяє вказати границі кожної з транзакцій ззовні, тобто зі сторони користувача. В більшості мов маніпулювання даними для визначення границь окремих транзакцій використовуються оператори BEGIN TRANSACTION, COMMIT і ROLLBACK (або їх еквіваленти). Якщо ці оператори не були використані, як правило, як єдина транзакція 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 наведено на рис. 2.1.

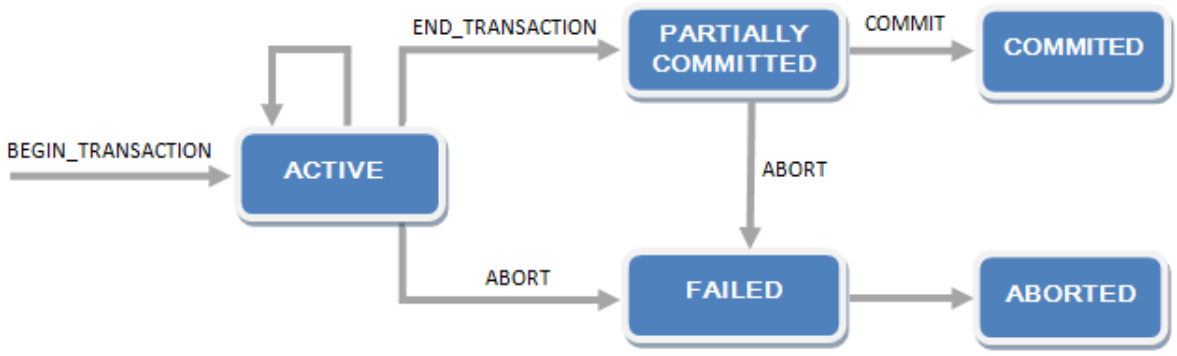

Рисунок 2.1 – Діаграма переходу транзакції

На діаграмі, окрім очевидних станів ACTIVE, COMMITTED і ABORTED присутні стани PARTIALLY COMMITTED і FAILED. Стан PARTIALLY COMMITTED виникає після виконання останнього оператора транзакції. В 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, і всі дані, оновлені в транзакції, неможливо успішно записати у зовнішню пам'ять. В подібних випадках транзакція повинна перейти у стан FAILED і завершитись аварійно. Якщо ж транзакція виконана успішно, то всі результати оновлення можуть бути успішно записані у зовнішню пам'ять і транзакція може перейти в стан COMMITTED.

2.3 Проблеми управління паралельним доступом

Управління паралельним доступом – це процес організації одночасного виконання в базі даних різних операцій доступу, який гарантує попередження їх впливу одна на одну.

Забезпечити паралельний доступ відносно нескладно, якщо всі користувачі будуть лише зчитувати інформацію з бази даних. В такому випадку робота кожного з них не впливатиме на роботу інших користувачів. Але якщо два чи більше користувачів одночасно звернуться до одного і того ж елемента даних або хоча б один з них спробує оновити збережену в базі даних інформацію, стає можливим взаємний вплив процесів доступу, який здатний привести до неузгодженості даних.

Задача управління паралельним доступом подібна до задач, які має вирішувати будь-яка багатокористувацька комп'ютерна система. Такі системи включають підсистему введення-виведення, яка здатна виконувати операції введення-виведення, в той час як процесор виконує інші операції. Система починає виконання першої транзакції і продовжує її виконання до першої операції введення-виведення. На час виконання цієї операції система призупиняє виконання першої транзакції та переходить до виконання команд другої транзакції. Коли другій транзакції потрібно буде виконати операцію введення-виведення, управління буде повернено першій транзакції та її виконання буде продовжено з тієї точки, в якій вона була призупинена. Виконання першої транзакції буде продовжено до досягнення наступної операції введення-виведення. Таким чином, виконання операцій двох транзакцій чергується та забезпечується їх паралельне виконання. Окрім того, загальна продуктивність системи (об'єм роботи, що виконується протягом заданого часового інтервалу) підвищується, оскільки процесор виконує команди іншої транзакції, замість того, щоб даремно простоювати, очікуючи завершення запущеної операції введення-виведення.

Не зважаючи на те, що кожна з транзакцій може сама по собі виконуватись коректно, подібне чергування операцій здатне привести до невірних результатів, через що цілісність і узгодженість бази даних будуть порушені.

Розрізняють такі потенційні проблеми, які можуть мати місце при паралельному виконанні транзакцій:

- проблема втраченого оновлення,
- проблема залежності від незафіксованих результатів,
- проблема аналізу неузгодженості,
- проблема неповторюваного читання,
- проблема фантомного читання.

Проблема втраченого оновлення полягає у тому, що результати цілком успішно завершеної операції оновлення однієї транзакції можуть бути перекриті результатами виконання другої транзакції.

Приклад 2.1. Транзакція ***T1*** передбачає зарахування 1000 грн. на рахунок ***bal_X***, на якому спочатку було 1000 грн., а транзакція ***T2*** полягає в знятті 100 грн. з цього ж рахунку. Якщо ж ці транзакції будуть виконуватись послідовно, одна за одною, без чергування операцій, то після завершення їх роботи на рахунку буде знаходитись сума 1900 грн., не залежно від порядку виконання транзакцій. Суть проблеми втраченого оновлення проілюстрована в табл. 2.1.

Таблиця 2.1 – Приклад проблеми втраченого оновлення

Час	Транзакція *T1*	Транзакція *T2*	Поле *bal_X*
t_1	begin_transaction		1000
t_2	read(bal_X)	begin_transaction	1000
t_3	bal_X = bal_X +1000	read(bal_X)	1000
t_4	write(bal_X)	bal_X = bal_X -100	2000
t_5	commit	write(bal_X)	900
t_6		commit	900

Припустимо, що транзакції ***T1*** і ***T2*** починаються практично одночасно і кожна з них зчитує вихідне значення суми на рахунку ***bal_X***, яке рівне 1000 грн. Потім транзакція ***T1*** збільшує суму на 1000 грн. і записує результат, що рівний 2000 грн., в базу даних – на рахунок ***bal_X***. В цей час транзакція ***T2*** зменшує на 100 грн. значення своєї копії початкової суми на рахунку ***bal_X*** і записує отримане значення, яке рівне 900 грн., в базу даних на той же рахунок, перекриваючи результат попереднього оновлення. В результаті з балансового рахунку «зникає» 1000 грн., доданих при виконанні попередньої операції.

Можна уникнути втрати результатів виконання транзакції ***T1***, заборонивши транзакції ***T2*** зчитувати початкове значення на рахунку ***bal_X*** до моменту завершення виконання транзакції ***T2***.

Проблема залежності від незафіксованих результатів або *проблема «брудного» читання* виникає в тому випадку, коли одна із транзакцій

отримає доступ до проміжних результатів виконання другої транзакції до того, як вони будуть зафіксовані в базі даних.

Приклад 2.2. Нехай транзакція ***T3*** збільшує значення суми на рахунку ***bal_X*** на 1000 грн., після чого виконання транзакції відміняється, тому СКБД повинна виконати відкат транзакції з відновленням початкового значення на рахунку, яке було рівне 1000 грн. Однак до цього моменту транзакція ***T4*** вже встигла зчитати змінене значення рахунку ***bal_X*** (2000 грн.) і використала саме це значення при виконанні операції зняття 100 грн. з рахунку, після чого зафіксувала в базі даних невірний результат, рівний 1900 грн. (замість правильного 900 грн.). Значення ***bal_X***, зчитане в транзакції ***T4***, називається брудними даними. Причина відкату транзакції не має значення. Джерело помилки полягає у тому, що при виконанні транзакції ***T4*** приймається припущення про успішне завершення операції оновлення в транзакції ***T3***, хоча в дійсності відбувся відкат цієї транзакції. Суть проблеми «брудного» читання проілюстрована в табл. 2.2.

Таблиця 2.2 – Приклад проблеми «брудного читання»

Час	Транзакція *T3*	Транзакція *T4*	Поле *bal_X*
t_1	begin_transaction		1000
t_2	read(bal_X)	begin_transaction	1000
t_3	bal_X = bal_X +1000	read(bal_X)	1000
t_4	write(bal_X)	bal_X = bal_X -100	2000
t_5	commit	write(bal_X)	900
t_6		commit	900

Проблему «брудного» читання можна усунути, заборонивши транзакції ***T4*** зчитувати значення залишку ***bal_X*** до прийняття рішення про те, чи повинна бути виконана фіксація або відкат транзакції ***T3***.

У наведених прикладах мова йшла про транзакції, які виконують оновлення в базі даних, чергування операцій яких могло призвести до приведення бази даних в неузгоджений стан. Однак транзакції, які лише зчитують інформацію з бази даних, також можуть давати невірні результати, якщо їм стануть доступні проміжні результати одночасно виконуваних і ще не завершених транзакцій, які оновлюють інформацію в базі даних.

Проблема аналізу неузгодженості виникає в тих випадках, коли транзакція зчитує декілька значень з бази даних, після чого друга транзакція оновлює деякі з цих значень безпосередньо під час виконання першої транзакції. Наприклад, транзакція, яка знаходить суму значень, вибраних з бази даних, отримає невірний результат, якщо під час її виконання друга транзакція змінить зчитані нею значення.

Приклад 2.3. Транзакція ***T6*** знаходить суму залишків на рахунках ***X*** (1000 грн.), ***Y*** (500 грн.) та ***Z*** (250 грн.). Однак в цей час транзакція ***T5*** здійснює переведення по 100 грн. з рахунків ***bal_X*** і ***bal_Z***. В результаті обчислене транзакцією ***T6*** значення виявляється невірним (більшим на 200 грн.). Приклад проблеми аналізу неузгодженості проілюстрований в табл. 2.3.

Таблиця 2.3 – Приклад проблеми аналізу неузгодженості

Час	Транзакція T5	Транзакція T6	Поле bal_X	Поле bal_Y	Поле bal_Z	Поле sum
t_1		begin_transaction	1000	500	250	
t_2	begin_transaction	sum=0	1000	500	250	0
t_3	read(bal_X)	read(bal_X)	1000	500	250	0
t_4	bal_X= bal_X-100	sum = sum+bal_X	1000	500	250	1000
t_5	write(bal_X)	read(bal_Y)	900	500	250	1000
t_6	read(bal_Z)	sum = sum+bal_Y	900	500	250	1500
t_7	bal_Z= bal_Z-100	read(bal_Z)	900	500	250	1500
t_8	write(bal_Z)	sum = sum+bal_Z	900	500	150	1750
t_9		commit	900	500	150	1750
t_{10}	commit		900	500	150	1750

Цю проблему можна вирішити, заборонивши транзакції ***T6*** зчитувати значення на рахунках ***bal_X*** та ***bal_Z*** до тих пір, поки транзакція ***T5*** не зафіксує виконані нею оновлення.

Проблема неповторюваного читання може виникнути, якщо в деякій транзакції відбувається повторне читання раніше зчитаного елемента даних, але між цими операціями читання була виконана модифікація цього елемента даних в іншій транзакції. Таким чином, в транзакції будуть отримані два різні значення одного і того ж елемента даних.

Проблема фантомного читання має місце тоді, коли деяка транзакція виконує запит на вибірку з відношення ряду рядків, які 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.

2.4 Впорядкування та відновлення транзакцій

Призначення протоколів управління паралельним доступом полягає у підготовці такого графіку виконання транзакцій, який виключить можливість їх впливу на результати роботи одна одної, що дозволить уникнути проблем, розглянутих у попередньому розділі.

Одне з очевидних рішень полягає у виконанні в кожний момент часу лише однієї транзакції – попередня транзакція обов’язково повинна бути зафіксована перш, ніж буде дозволено почати виконання наступної транзакції. Однак призначення багатокористувацьких СКБД полягає у забезпеченні максимального ступеня розпаралелення транзакцій користувачів, тому ті транзакції, які не впливають на роботу одна одної, можуть виконуватись одночасно. Виявлення таких транзакцій становить суть *процесу впорядкування транзакцій*.

Задачі впорядкування транзакції передбачають побудову графіків виконання транзакцій. *Графік виконання транзакцій* – це послідовність за-

пуску операцій декількох паралельно виконуваних транзакцій, яка зберігає черговість виконання операцій в кожній окремо взятій транзакції.

У *послідовному графіку* операції кожної з транзакцій виконуються строго послідовно і не можуть чергуватись з операціями, виконуваними в інших транзакціях. Тобто, за наявності двох транзакцій Т1 і Т2, спочатку має бути виконана транзакція Т1, а після її завершення Т2, або ж навпаки. Таким чином, при послідовному виконанні взаємовплив між транзакціями відбуватись не буде, оскільки в кожний момент часу виконується лише одна транзакція.

У *непослідовному графіку* чергуються операції з деякого набору одночасно виконуваних транзакцій.

Суть впорядкування полягає саме в пошуку таких непослідовних графіків, які дозволяють транзакціям виконуватись паралельно без впливу одна на одну. Непослідовний графік називають *впорядковуючим*, якщо він призводить до отримання таких же результатів, які досягаються при використанні деякого варіанту послідовного графіка.

Розглядаючи умови впорядкування, важливо враховувати послідовність виконання операцій читання та запису даних:

- Якщо дві транзакції тільки зчитують деякий елемент даних, вони не будуть конфліктувати між собою і послідовність їх виконання не має значення.
- Якщо дві транзакції зчитують або записують зовсім незалежні елементи даних, вони не будуть конфліктувати між собою і послідовність їх виконання не має значення.
- Якщо одна з транзакцій записує елемент даних, а інша зчитує або записує цей же елемент даних, послідовність їх виконання має суттєве значення.

Приклад 2.4. Розглянемо графік ***S1***, наведений в стовпцях *Варіанта А* табл. 2.4. Він встановлює послідовність операцій, що виконуються паралельно в транзакціях ***T7*** і ***T8***. Оскільки операція запису даних рахунку ***bal_X*** в транзакції ***T8*** не конфліктує з наступною операцією читання даних рахунку ***bal_Y*** в транзакції ***T7***, можна замінити послідовність виконання цих операцій і отримати еквівалентний графік ***S2***, наведений в стовпцях *Варіанту Б* цієї ж таблиці. Якщо змінити порядок виконання наступних не конфліктуючих між собою операцій, то можна отримати еквівалентний послідовний графік ***S3*** *(Варіант В* табл. 2.4*).*

Графік ***S3*** є послідовним, а оскільки графіки ***S1*** і ***S2*** є еквівалентними графіку ***S3,*** вони є впорядковуючими.

Такий тип впорядкування прийнято називати *конфліктним впорядкуванням*. В конфліктно впорядковуючому графіку порядок виконання будь-яких конфліктуючих операцій відповідає їх розміщенню в послідовному графіку. Згідно з *правилом запису, що підкоряється обмеженням* (яке говорить, що транзакція повинна оновлювати елемент даних виходячи з його попереднього значення, яке було прочитано транзакцією на по-

чатку), для перевірки конфліктного впорядкування можна використовувати *граф передування*, або граф впорядкування.

Таблиця 2.4 – Еквівалентні графіки: *Варіант А* – непослідовний графік ***S1***; *Варіант Б* – непослідовний графік ***S2***, еквівалентний графіку ***S1;*** *Варіант В* – послідовний графік ***S3***, еквівалентний графікам ***S1*** і ***S2***

Час	Транзакція T7	Транзакція T8
Варіант А		
t_1	begin_transaction	
t_2	read(bal_X)	
t_3	write(bal_X)	
t_4		begin_transaction
t_5		read(bal_X)
t_6		write(bal_X)
t_7	read(bal_Y)	
t_8	write(bal_Y)	
t_9	commit	
t_{10}		read(bal_Y)
t_{11}		write(bal_Y)
t_{12}		commit
Варіант Б		
t_1	begin_transaction	
t_2	read(bal_X)	
t_3	write(bal_X)	
t_4		begin_transaction
t_5		read(bal_X)
t_6	read(bal_Y)	
t_7		write(bal_X)
t_8	write(bal_Y)	
t_9	commit	
t_{10}		read(bal_Y)
t_{11}		write(bal_Y)
t_{12}		commit
Варіант В		
t_1	begin_transaction	
t_2	read(bal_X)	
t_3	write(bal_X)	
t_4	read(bal_Y)	
t_5	write(bal_Y)	
t_6	commit	
t_7		begin_transaction
t_8		read(bal_X)
t_9		write(bal_X)
t_{10}		read(bal_Y)
t_{11}		write(bal_Y)
t_{12}		commit

Для графіка ***S*** граф передування виглядає як направлений граф ***G=(N, E)***, що складається з множини вершин ***N***, множини направлених ребер ***E*** і формується таким чином.

1. Створюється вершина, що відповідає кожній із транзакцій.
2. Створюються направлені ребра T_i -> T_j, якщо транзакція T_j зчитує значення елемента, записаного транзакцією T_i.
3. Створюються направлені ребра T_i -> T_j, якщо транзакція T_j записує значення в елемент даних після того, як він був зчитаний транзакцією T_i.
4. Створюються направлені ребра T_i -> T_j, якщо транзакція T_j записує значення в елемент даних після того, як він був записаний транзакцією T_i.

Якщо в графі передування, що відповідає графіку ***S***, існує ребро T_i -> T_j, то в будь-якому послідовному графіку ***S'***, еквівалентному графіку S, транзакція T_i повинна передувати транзакції T_j. Якщо граф передування містить цикли, то відповідний йому графік не є конфліктно впорядковуючим.

Впорядковуючими називають такі графіки, які дозволяють зберегти узгодженість бази даних за умови, що жодна з транзакцій цього графіка не буде відмінена. Протилежний підхід дозволяє проаналізувати відновлюваність транзакцій, що входять у даний графік. У випадку відміни транзакції, властивість нерозривності вимагає, щоб були відмінені всі результати її виконання. Однак, якщо в деякому графіку ***S*** транзакція T_j використала елемент даних, записаний транзакцією T_i, і зафіксувала свої результати, то відповідно до властивості стійкості відміна транзакції T_i не може активізувати відміну транзакції T_j (без запуска компенсуючої транзакції). Тому такий графік не володіє властивістю відновлюваності і є недопустимим.

Відновлюваним графіком називається графік, в якому для кожної пари транзакцій T_i і T_j виконується правило: якщо транзакція T_j зчитує елемент даних, попередньо записаний транзакцією T_i, то фіксація результатів транзакції T_i повинна виконуватись до фіксації результатів транзакції T_j.

2.5 Методи управління паралельним доступом

На практиці графіки в СКБД не перевіряються на впорядкованість. Замість цього використовуються спеціальні методи управління паралельним доступом: ***метод блокувань*** та ***метод часових відміток***.

Вище згадані методи відносять до ***консервативних*** (або ***песимістичних***) підходів, оскільки вони відкладають виконання транзакцій, здатних в майбутньому в той чи інший момент часу вступити в конфлікт з іншими транзакціями.

2.5.1 Методи блокувань

Коли деяка транзакція отримує доступ до бази даних, механізм блокувань дозволяє заборонити спроби отримання доступу до цих же даних зі сторони інших транзакцій.

Існує декілька варіантів цього механізму, однак всі вони побудовані на фундаментальному принципі: *транзакція повинна затребувати виконати блокування для читання (поділюване) або для запису (виключне) деякого елемента даних до того, як вона зможе виконати в базі даних відповідну операцію читання чи запису.*

Блокування може бути виконане для елементів будь-якого розміру: від бази даних в цілому до окремого поля конкретного запису.

Якщо в транзакції встановлене ***поділюване*** блокування елемента даних, то в ній може виконуватись лише читання цього елемента даних, але не його оновлення.

Якщо в транзакції встановлене ***виключне*** блокування елемента даних, то в ній може виконуватись і читання цього елемента даних, і його оновлення.

Оскільки поділювані блокування не можуть стати причиною конфлікту, допускається встановлювати поділювані блокування для одного і того ж елемента даних одночасно зі сторони декількох транзакцій. Виключне блокування можна встановити лише в одній транзакції.

Встановлення блокувань відбувається за такими правилами.

1. Будь-яка транзакція, якій необхідно отримати доступ до елемента даних, повинна спочатку виконати блокування (поділюване або виключне).
2. Якщо елемент ще не заблокований іншою транзакцією , блокування елемента буде виконано успішно.
3. Якщо елемент даних в поточний момент вже заблокований, СКБД проаналізує, чи є сумісним тип отриманого запиту с типом вже існуючого блокування. У випадку поділюваного блокування запит на надання блокування виконається успішно. Інакше, транзакція переходить в стан очікування, який буде продовжуватись до тих пір, поки існуюче блокування не буде знято.
4. Транзакція повинна утримувати блокування елемента даних до тих пір, поки явним чином не вивільнить його або в ході свого виконання, або по завершенню (успішному чи неуспішному).
5. Лише після того, як з елемента даних буде знято виключне блокування, результати виконання операції запису стануть доступними для інших транзакцій.

Окрім цих правил, в деяких системах транзакціям дозволяється встановлювати поділюване блокування на деякому елементі даних, а потім підвищувати рівень цього блокування до виключного. В такому випадку спочатку слід оцінити стан оброблюваних в транзакції даних, а потім

прийняти рішення чи потрібно їх змінювати. В деяких системах навпаки, дозволено понижувати рівень блокування з виключного до поділюваного.

Для забезпечення впорядкованості слід застосовувати додатковий протокол, який визначає моменти встановлення і зняття блокувань для кожної транзакції. Найбільш популярним є ***протокол двофазного блокування (2PL)***.

Транзакція виконується по протоколу двофазного блокування, якщо в ній всі операції блокування передують першій операції розблокування.

Відповідно до протоколу двофазного блокування кожна транзакція може бути розбита на дві фази:

- *фазу розширення*, в якій встановлюються всі необхідні блокування і не вивільняється жодна з них;
- *фазу звуження*, в якій вивільняються всі раніше встановлені блокування і не встановлюється жодне нове блокування.

При цьому зовсім не обов'язково, щоб всі блокування були встановлені одночасно. Як правило, транзакція встановлює деякі блокування, виконує певну обробку, після чого знову може зажадати встановлення додаткових блокувань. Однак не може звільнити жодне з блокувань, поки не досягне стадії виконання, на якій встановлення нових блокувань вже не потрібне.

Інший варіант протокола 2PL – ***обмежений протокол двофазного блокування*** – передбачає призупинення до кінця транзакції вивільнення тільки виключних блокувань. В більшості СКБД використовується один з двох варіантів протокола 2PL.

2.5.2 Методи обробки взаємоблокувань

Використання методів блокувань може призвести до проблеми, що носить назву ***взаємоблокування***, яка є наслідком того факту, що будь-яка транзакція може бути переведена в стан очікування до вивільнення необхідного їй елемента даних. Якщо дві транзакції будуть очікувати вивільнення елементів, заблокованих другою транзакцією з цієї ж пари, то виникне стан взаємоблокування. Жодна з транзакцій не може продовжити свою роботу, оскільки кожна очікує завершення роботи іншої.

Крім того, транзакції можуть також входити в стан ***активного взаємоблокування*** , при якому вони залишаються в стані очікування невизначено довгий час і не мають можливості встановлювати будь-які нові блокування. Ця ситуація можлива у випадках, коли алгоритм переведення транзакцій в стан очікування не надає рівні можливості всім транзакціям і не враховує час, проведений транзакціями у стані очікування. Для подолання активного взаємоблокування може використовуватись система пріоритетів, в якій пріоритет транзакції тим вище, чим довше вона знаходиться в стані очікування.

Відповідальність за виявлення взаємоблокувань і усунення цієї проблеми повинна бути покладена на СКБД.

На жаль, існує лише один спосіб подолати стан взаємоблокувань: виконання однієї або декількох транзакцій повинно бути відмінено, що супроводжується відкатом усіх змін, внесених відміненими транзакціями.

Оскільки ситуація взаємоблокування повинна бути абсолютно прозорою для користувача, СКБД зобов'язана автоматично перезапустити всі відмінені нею транзакції.

Найбільш поширеними є три методи обробки зваємоблокувань:

1. Установка тайм-аутів
2. Попередження взаємоблокувань.
3. Виявлення взаємоблокувань і відновлення нормальної роботи.

Установка тайм-аутів. Це найпростіший спосіб усунення взаємоблокувань, при якому транзакція, яка запитує блокування, може очікувати його отримання протягом певного періоду часу, встановленого в системі. Якщо протягом цього часу блокування не надається, відбувається відміна запиту на блокування по тайм-ауту. В такому випадку СКБД припускає, що дана транзакція могла виявитись у стані взаємоблокування, навіть якщо це не відповідає дійсності, тому ініціює аварійне завершення і автоматичний перезапуск транзакції.

Попередження взаємоблокувань – підхід, що базується на впорядкуванні транзакцій з використанням часових відміток. Кожна транзакція позначається відміткою часу її початку, яка повинна бути унікальною. Якщо деяка транзакція А запитує блокування елемента даних, який вже заблоковано транзакцією В, то може виконуватись один із двох можливих алгоритмів:

- *„Очікування-відміна".* При такому підході лише більш старі транзакції очікують завершення більш нових. Якщо виконання транзакції А почалось раніше, ніж транзакції В, то транзакція А переходить в стан очікування; в протилежному випадку відбувається її відміна. Це означає, що відбувається відкат і перезапуск транзакції А.
- *„Відміна-очікування".* При такому підході лише більш нові транзакції очікують завершення більш старих. Якщо виконання транзакції А почалось пізніше, ніж транзакції В, то транзакція А переходить в стан очікування; в протилежному випадку відбувається відміна транзакції В, тобто відбувається відкат і перезапуск транзакції В.

Якщо транзакція має бути перезапущена, то їй після запуску присвоюється її початкова відмітка часу.

Використання будь-якого з розглянутих алгоритмів гарантує, що ситуація взаємоблокування не виникне взагалі, і жодна з транзакцій не буде

перезапускатись безкінечну кількість разів. Основним недоліком такого підходу є те, що необхідно виконувати досить багато відкатів транзакцій.

Виявлення взаємоблокувань і відновлення нормальної роботи. Виявлення взаємоблокувань, як правило, виконують за допомогою графа очікування, який відображає залежність транзакцій одна від одної. Транзакція Ti вважається залежною від транзакції Tj у тому випадку, якщо транзакція Tj заблокувала елемент даних, необхідний для продовження роботи транзакції Ti. При побудові графа створюється направлене ребро Ti->Tj, якщо тпранзакція Ti очікує вивільнення елемента даних, заблокованого в поточний момент часу транзакцією Tj. Взаємо блокування має місце лише в тому випадку, коли граф очікування містить цикл.

Перевірка на наявність циклів у графі очікування проводиться через регулярні проміжки часу. Якщо встановлений проміжок часу буде занадто коротким, то виявлення взаємоблокувань буде потребувати значних додаткових витрат часу та ресурсів. При занадто довгих проміжках часу існує ризик того, що взаємоблокування довго не будуть помічені.

Компромісним рішенням є динамічний алгоритм виявлення взаємоблокувань, при якому встановлюється деяке початкове значення проміжку часу. Після кожного сеансу виконання алгоритму контролюється наявність виявленого взаємоблокування. Якщо взаємоблокування не виявлене, то проміжок часу може бути збільшений, наприклад вдвічі, порівняно з початковим значенням. В протилежному випадку, проміжок часу скорочують наполовину.

Після того як СКБД виявила взаємоблокування необхідно виконати аварійне завершення однієї або декількох транзакцій. Для цього необхідно вирішити декілька задач.

1. Вибрати транзакції, які підлягають відміні в результаті взаємоблокування. Як правило, слід обирати ті транзакції, відміна яких потребує мінімальних витрат. При цьому враховують такі фактори:
   - тривалість виконання транзакції (може виявитись більш доцільним аварійне завершення транзакції, яка була щойно розпочата, а не та транзакція, яка вже виконувалась протягом деякого часу);
   - кількість елементів даних, які оновлюються в транзакції (може виявитись доцільнішим аварійно завершити з декількох транзакцій ту, яка внесла менше змін в базу даних);
   - кількість елементів даних, які все ще підлягають оновленню в транзакції (доцільніше відмінити ту трансакцію, яка передбачає внесення більших змін у базу даних, ніж ту, для завершення якої необхідно внести невелику кількість змін). На жаль, такі відомості не завжди містяться в СКБД.
2. Визначити ступінь відкату транзакції. Найпростішим рішенням є відміна всіх змін, внесених транзакцією. Однак для усунення взаємо блокування іноді достатньо відмінити лише частину змін.

3. Попередити виникнення ситуацію виснаження ресурсів. Виснаження ресурсів виникає, коли для відміни завжди обирається одна і та ж транзакція, тому її так і не вдається виконати. В СКБД можна уникнути ситуації виснаження ресурсів шляхом реєстрації кількості таких випадків, коли для відміни була вибрана певна транзакція, і переходу до застосування іншого критерію відбору після досягнення деякої верхньої межі кількості випадків відміни.

Враховуючи складність методу, його використання обмежене.

2.5.3 Методи управління паралельним доступом з використанням часових відміток

Методи управління паралельним доступом, які використовують часові відмітки, не потребують застосування будь-яких блокувань і не передбачають переведення транзакцій в стан очікування. Транзакції, які потрапити в конфліктну ситуацію, просто відміняються, а потім запускаються на виконання знову.

Часова відмітка – це унікальний ідентифікатор, який створюється СКБД з метою визначення відносного моменту часу запуску транзакції.

Часова відмітка може бути створена з використанням системного годинника для фіксації моменту часу запуску транзакції або шляхом збільшення значення деякого логічного лічильника при кожному запуску чергової транзакції.

Метод з використанням часових відміток – це протокол управління паралельним виконанням, основна мета якого полягає у встановленні глобальної черговості виконання транзакцій, при якій більш старі транзакції (транзакції з меншим значенням часової відмітки) мають більш високий пріоритет при вирішенні конфліктних ситуацій.

Якщо транзакція чинить спробу читання чи запису елемента даних, то ці операції виконуються тільки в тому випадку, якщо останнє оновлення затребуваного елемента даних було виконано більш старою транзакцією. В протилежному випадку, транзакція, що запросила операцію читання/запису, відміняється і пере запускається знову з присвоєнням їй нової часової відмітки. Нова часова відмітка повинна бути присвоєна перезапущеній транзакції для того, щоб попередити її потраплянні в цикл постійної відміни та перезапуску. Без отримання нової часової відмітки транзакція з більш старою часовою відміткою не зможе завершити свою роботу шляхом фіксації, оскільки більш нова транзакція вже встигла зафіксувати свої результати в базі даних.

Окрім часових відміток для транзакції в системі повинні використовуватись часові відмітки для елементів даних. Кожний елемент даних повинен мати часову відмітку читання, яка містить часову відмітку останньої транзакції, що виконала його читання, та часову відмітку запису, яка

містить часову відмітку останньої транзакції, що оновила вміст цього елемента даних.

2.5.4 Оптимістичні методи впорядкування транзакцій

На противагу песимістичним, ***оптимістичні*** підходи будуються на припущенні, що ймовірність конфлікту не висока, тому вони допускають асинхронне виконання транзакцій, а перевірка на наявність конфлікту відкладається на момент їх завершення та фіксації в базі даних. У випадку виявлення конфліктної ситуації, відбувається відкат транзакції та її повторний запуск. Застосування такої схеми доцільно лише у випадку, коли відкати транзакцій будуть відбуватись досить рідко. Такі методи дозволяють досягти більш високого рівня розпаралелення порівняно з традиційними протоколами, оскільки вони не потребують виконання механізму блокувань.

Оптимістичний протокол управління паралельним виконанням включає дві або три стадії, в залежності від того, яка операція виконується в даній транзакції – читання чи запису.

Стадія читання охоплює період від початку транзакції і до моменту, що передує фіксації результатів. Транзакція зчитує з бази даних значення всіх необхідних їй елементів даних і поміщає їх в локальні змінні. Будь-які оновлення застосовуються лише до локальної копії даних, а не до інформації, що зберігається в базі даних.

Стадія перевірки наступає за стадією читання. Виконуються перевірки, необхідні для отримання гарантії відсутності порушення впорядковуваності у випадку переносу в базу даних змін, виконаних транзакцією. Для транзакцій, що включають тільки операції читання, перевірка полягає в підтвердженні того, що використані транзакцією значення все ще залишаються актуальними значеннями відповідних елементів даних. Якщо порушень не виявлено, транзакція завершує своє виконання шляхом фіксації. Якщо знайдені змінені значення, транзакція відміняється та перезапускається. Якщо в транзакції виконувалось оновлення даних, то перевірка включає виконання контролю за тим, чи збережеться база даних в узгодженому стані після внесення в неї результатів даної транзакції, а також чи не будуть порушені умови впрорядковуваності. У випадку виявлення помилки транзакція відміняється і пере запускається.

Стадія запису виконується після успішного завершення стадії перевірки, але тільки у випадку транзакцій, що включають операції оновлення. На цій стадії всі зміни, внесені в локальні копії даних, переносять у базу даних.

Хоча застосування оптимістичних методів досить ефективне у випадку незначної кількості конфліктів у системі, вони можуть викликати відкат окремих транзакцій. Оскільки відкат стосується лише локальних копій даних, виникнення каскадного відкату виключається. Якщо відкати

виникають досить часто, це може вказувати на те, що оптимістичні методи не зовсім підходять для управління паралельним виконанням в конкретній системі.

2.6 Механізми відновлення бази даних

2.6.1 Функції СКБД по відновленню бази даних

Відновлення бази даних – це процес повернення бази даних в прийнятний стан, втрачений в результаті збою чи відмови.

Типова СКБД повинна надавати такі функції відновлення:

1. Механізм резервного копіювання, призначений для періодичного створення резервних копій бази даних.
2. Засоби ведення журналу транзакцій, в якому фіксується поточний стан транзакцій і зміни, які вони вносять у базу даних.
3. Функція створення контрольних точок, яка забезпечує перенесення виконуваних в базі даних змін у вторинну пам’ять з метою зробити їх постійними.
4. Диспетчер відновлення, що забезпечує відновлення узгодженого стану бази даних, порушеного в результаті відмови.

Резервна копія бази даних використовується у випадку пошкодження або руйнування файлів бази даних у вторинній пам’яті. Резервне копіювання може виконуватись для всієї бази даних в цілому або для зміненої її частини (тобто інкрементно). В останньому випадку в копію поміщаються відомості лише про зміни, що накопичились з моменту створення попередньої повної або інкрементної резервної копії системи.

Після того, як буде відновлена остання резервна копія, необхідно виконати накат всіх зафіксованих транзакцій, відомості про які містяться в журналі транзакцій. Безумовно, припускається, що сам журнал транзакцій не був пошкодженим. Тому рекомендується, щоб у всіх випадках, коли це можливо, файл журналу транзакцій створювався на дискових накопичувачах, відмінних від тих, де розміщені основні файли бази даних.

Якщо ж база даних не отримала фізичних пошкоджень, а лише втратила узгодженість розміщених у ній даних (наприклад, в результаті аварійної зупинки системи в процесі обробки транзакцій), то достатньо виконати відкат тих змін, які спричинили перехід бази даних в неузгоджений стан. Крім того, може виникнути необхідність виконати накат деяких транзакцій, щоб внесені в них зміни були дійсно зафіксовані у вторинній памяті. В даному випадку немає необхідності звертатись до резервної копії бази даних, оскільки повернути базу даних в узгоджений стан можна за допомогою інформації з журналу транзакцій.

Для повернення бази даних в узгоджений стан використовуються такі методи: ***метод відкладеного оновлення***, ***метод негайного оновлення*** та альтернативний ***метод тіньового сторінкового обміну***.

2.6.2 Журнал транзакцій

Журналом транзакцій називають спеціальний файл, у якому СКБД фіксує хід виконання транзакцій в базі даних. Він містить відомості про всі оновлення, що виконуються в базі даних. У файлі журналу транзакцій може міститись така інформація:

- записи про транзакції, включаючи ідентифікатор транзакції;
- тип запису журналу (початок транзакції, операція вставки, оновлення чи видалення, відміна чи фіксація транзакції);
- ідентифікатор елемента даних, над яким виконується операція обробки (операції вставки, видалення та оновлення);
- копія елемента даних до операції, тобто його значення до зміни (тільки для операцій оновлення та видалення);
- копія елемента даних після операції, тобто його значення після зміни (тільки для операцій оновлення та вставки);
- службова інформація файлу журналу транзакцій, яка включає покажчики на попередній та наступний запис журналу для цієї транзакції (для всі всіх операцій);
- записи контрольних точок.

Оскільки файл журналу транзакцій відіграє важливу роль в процесах відновлення, він може створюватися в двох і навіть трьох екземплярах, які автоматично підтримуються системою. У випадку пошкодження однієї копії при відновленні буде використовуватись інша.

Файл журнала транзакцій потенційно є вузьким місцем з точки зору продуктивності будь-яких систем, тому швидкість запису інформації у файл журналу транзакцій може виявитись одним із найважливіших факторів, що визначають загальну продуктивність системи з базою даних.

2.6.3 Створення контрольних точок

Інформація з журналу транзакцій призначена для використання в процесі відновлення системи після відмови. Але при виникненні відмови може бути відсутньою інформація про те, з якого моменту в минулому необхідно почати пошук у файлі журналу транзакцій, щоб не виконувати накат транзакцій, які вже завершились з успішною фіксацією в базі даних. Для обмеження об’єму пошуку та послідовної обробки інформації у файлі журналу транзакцій використовується ***метод створення контрольних точок***.

Контрольною точкою називається момент синхронізації між базою даних та журналом транзакцій. В цей момент всі буфера системи примусово записуються у вторинну пам’ять системи.

Контрольні точки організовуються через встановлений інтервал часу та передбачають виконання таких дій:

- перенесення всіх наявних в оперативній пам’яті записів журналу транзакцій у вторинну пам’ять;
- запис всіх модифікованих блоків у буферах бази даних у вторинну пам’ять;
- поміщення у файл журналу транзакцій записів контрольної точки. Цей запис містить ідентифікатори всіх транзакцій, які були активні в момент створення контрольної точки.

Якщо транзакції виконуються послідовно, то після виникнення відмови файл журналу транзакцій переглядається з метою виявлення останньої з транзакцій, яка почала свою роботу до моменту створення останньої контрольної точки. Будь-яка більш рання транзакція успішно зафіксована в базі даних, а це означає, що її зміни були перенесені на диск в момент створення останньої контрольної точки. Тому необхідно виконати накат тільки тих транзакцій, які були активні в момент створення контрольної точки, а також всіх наступних транзакцій, які почали свою роботу пізніше і для яких в журналі наявні записи як початку, так і їх фіксації. Та транзакція, яка була активна в момент відмови повинна бути відмінена.

Якщо транзакції виконуються паралельно, потрібно виконати накат всіх транзакцій, які були зафіксовані з часу створення контрольної точки, та відкат всіх транзакцій, які були активні в момент аварії.

Як правило, створення контрольних точок є відновно недорогою операцією, тому часто достатньо створювати три або чотири контрольні точки в годину. Таким чином, у випадку збою необхідно відновлювати роботу лише за останні 15-20 хвилин.

2.6.4 Метод відкладеного оновлення

При використанні цього методу оновлення даних не заносяться в базу даних до тих пір, поки транзакція не видасть команду фіксації виконаних змін. Якщо виконання транзакції буде припинено до досягнення цієї точки, жодних змін в базі даних виконано не буде, тому не потрібно буде їх відміняти. Однак в даному випадку може виникнути потреба у виконанні накату оновлень зафіксованих транзакцій, оскільки їх результати могли ще не досягти бази даних. При застосуванні даного методу файл журналу транзакцій використовується з метою захисту системи від аварій за таким алгоритмом:

1. При запуску транзакцій в журнал поміщається запис початку транзакції.

2. При виконанні будь-якої операції у файл журналу транзакцій заноситься уся інформація про виконану операцію за виключення значень елементів даних, які передували оновленню. При цьому реального запису змін в буфери СКБД або базу даних не відбувається.

3. Коли транзакція досягає точки фіксації, у файл журналу транзакцій заноситься запис фіксації транзакції.

4. У випадку аварійного завершення транзакції записи журналу по цій транзакції анулюються і не виводяться на диск.

Слід звернути увагу на те, що записи по виконуваній транзакції виводяться на диск до того, як її результати будуть зафіксовані. Тому якщо відмова бази даних відбудеться в процесі справжнього внесення оновлень в базу даних, занесені в журнал відомості зберігаються і необхідні оновлення можна буде виконати пізніше. У випадку відмови файл журналу транзакцій аналізується з метою виявлення транзакцій, які знаходились в процесі виконання в момент відмови. Починаючи з останнього рядка файл журналу транзакцій переглядається у зворотному напрямку, аж до запису про останню виконану контрольну точку.

5. Для будь-яких транзакцій, для яких у файлі журналу транзакцій присутні записи початку транзакції та фіксації транзакції, повинен бути виконаний накат. Процедура накату транзакцій виконує всі операції запису в базу даних, використовуючи інформацію про стан елементів даних після оновлення, яка міститься в записах журналу по даній транзакції, при чому в тому порядку, в якому вони були записані у файл журналу транзакцій. Якщо ці операції запису вже були успішно завершені до виникнення відмови, це не спричинить жодного впливу на стан елементів даних, оскільки вони не можуть бути зіпсовані, якщо будуть записані ще раз. Однак такий метод гарантує, що будуть оновлені будь-які елементи даних, які не були коректно оновлені до моменту відмови.

6. Будь-яка транзакція, для якої у файлі журналу транзакцій присутні записи початку транзакції і відміни транзакції, просто ігнорується, оскільки жодних реальних оновлень інформації в базі даних по ній не виконувалось, а тому і не потрібно виконувати їх відкат.

Якщо в процесі відновлення виникне інший системний збій, записи файлу журналу транзакцій можуть бути використані для відновлення бази даних ще раз. В такому випадку немає значення, скільки разів кожен з рядків журналу буде використаний для повторного внесення змін в базу даних.

2.6.5 Метод негайного оновлення

При використанні протоколу негайного оновлення всі зміни вносяться в базу даних одразу ж після їх виконання в транзакції, ще до досягнення моменту фіксації. Окрім необхідності виконання накату змін зафіксованих транзакцій одразу ж після аварії, може виникнути потреба виконати відкат змін, внесених транзакціями, які не були зафіксовані до моменту аварії. При застосуванні даного методу файл журналу транзакцій використовується з метою захисту системи від аварій за таким алгоритмом:

1. При запуску транзакцій в журнал поміщається запис початку транзакції.

2. При виконанні будь-якої операції у файл журналу транзакцій заноситься уся інформація про виконану операцію.

3. Щойно запис буде поміщений у файл журналу транзакцій, всі виконані оновлення заносяться в буфера бази даних.

4. Оновлення записуються в базу даних при кожному черговому скиданні буферів бази даних у вторинну пам'ять.

5. Після фіксації транзакцій у файл журналу транзакцій заноситься запис фіксації транзакції.

Важливо, щоб всі записи заносились у файл журналу транзакцій до внесення відповідних змін в базу даних. Ця вимога відома як ***протокол попереднього запису журналу***. Якщо зміни спочатку будуть внесені в базу даних і збій в системі виникне до занесення інформації про це у файл журналу транзакцій, то диспетчер відновлення не буде мати можливості виконати відкат (або накат) результатів даної операції. При використанні протоколу попереднього запису журналу диспетчер відновлення завжди може діяти, базуючись на тому, що якщо для визначеної транзакції у файлі журналу транзакцій відсутній запис фіксації транзакції, то транзакція вважається активною в момент виникнення відмови і тому для неї повинен бути виконаний відкат.

Якщо виконання транзакції завершилось аварійно, то для відміни внесених нею змін може бути використаний файл журналу транзакцій, оскільки в ньому збережені відомості про початкові значення всіх змінених елементів даних. Враховуючи те, що транзакція може виконати декілька змін одного і того ж елемента даних, відміна операції запису виконується у зворотному порядку. Незалежно від того, чи були результати виконання транзакції внесені в базу даних, наявність в записах журналу транзакцій початкових значень елементів даних гарантує, що база даних буде приведена в стан, що відповідає початку відміненої транзакції.

2.6.6 Метод тіньового сторінкового обміну

Альтернативою згаданим вище схемам відновлення, побудованим на використанні файлу журналу транзакцій, є метод ті нього сторінкового обміну. Цей метод передбачає організацію на час виконання транзакції двох таблиць сторінок – поточної та тіньової. Коли транзакція починає свою роботу, обидві таблиці сторінок є однаковими. Тіньова таблиця сторінок в подальшому не змінюється і може бути використана для відновлення бази даних у випадку відмови системи. В ході виконання транзакції поточна таблиця сторінок використовується для реєстрації всіх змін, внесених у базу даних. Після завершення транзакції поточна таблиця сторінок стає тіньовою таблицею.

Метод тіньового сторінкового обміну має ряд переваг перед методами, що використовують журнал транзакцій:

– виключаються витрати, пов'язані з веденням журналу транзакцій;

– процес відновлення відбувається значно швидше, оскільки немає необхідності виконувати операції накату і відкату.

Однак йому властиві деякі недоліки: фрагментація даних та необхідність періодичного виконання процедури збору сміття для повернення в систему не використовуваних блоків пам’яті.

Контрольні питання

1. Дайте визначення поняття транзакції.
2. Наведіть властивості транзакції.
3. Опишіть порядок виконання операцій транзакції та стани, в яких може перебувати транзакція.
4. Охарактеризуйте та наведіть приклади проблем, що виникають при паралельному виконанні транзакцій.
5. Поясніть яким чином і з якою метою будують впорядковувальні графіки?
6. Охарактеризуйте метод управління паралельним виконанням транзакцій з використанням блокувань.
7. Які методи усунення проблеми взаємоблокування Вам відомі? Коротко охарактеризуйте їх.
8. Яким чином використовуються часові відмітки для управління паралельним виконанням?
9. Яка принципова відмінність між оптимістичними та консервативними методами управління паралельним виконанням транзакцій?
10. Які функції повинна виконувати типова СКБД для відновлення бази даних?
11. Яку інформацію реєструють у журналі транзакцій? Яким чином журнал транзакцій використовують для відновлення бази даних?
12. Що називають контрольною точкою? З якою метою їх створюють?
13. Опишіть процес відновлення бази даних за методом відкладеного оновлення.
14. Охарактеризуйте процес відновлення бази даних за методом негайного оновлення.
15. Назвіть переваги та недоліки використання методу сторінкового обміну для відновлення бази даних.

3. РОЗПОДІЛЕНІ БАЗИ ДАНИХ

3.1 Концепція розподілених баз даних

3.1.1 Основні поняття

Основною передумовою розробки систем, що використовують бази даних, є прагнення об'єднати всі дані, які обробляються на підприємстві, в єдине ціле та забезпечити контрольований доступ до них. Сучасні потужні підприємства часто мають велику кількість підрозділів, які можуть фізично розташовуватись за сотні та навіть тисячі кілометрів один від одного. Кожний з цих підрозділів має свою локальну базу даних. Якщо ці бази мають різні архітектури та використовують різні протоколи зв'язку, то їх розглядають як *інформаційні острови*, важкодоступні для інших. В такому випадку виникає необхідність об'єднати розрізнені бази даних в одне логічне ціле, тобто створити розподілену базу даних.

Розподілена база даних – це сукупність логічно зв'язаних баз даних або частин однієї бази, які розпаралелені між декількома територіально-розподіленими ПЕОМ і забезпечені відповідними можливостями для управління цими базами або їх частинами. Тобто, розподілена база даних реалізується на різних просторово розосереджених обчислювальних засобах, разом з організаційними, технічними і програмними засобами її створення і ведення.

В дійсності розподілена база даних є *віртуальною* базою даних, компоненти якої фізично зберігаються на декількох різних *реальних* базах даних на декількох різних вузлах.

На рис.3.1 наведено приклад типової топології розподіленої бази даних.

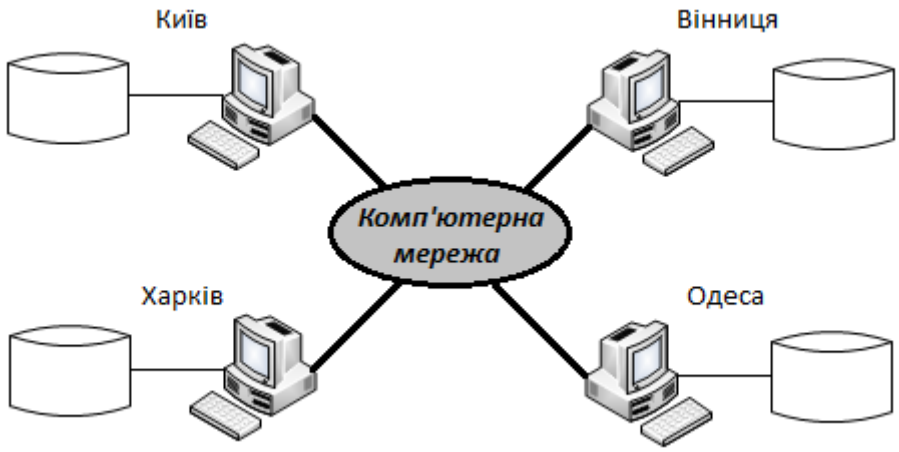

Рисунок 3.1 – Приклад типової топології розподіленої бази даних

Використовувати розподілені бази даних ефективно і доцільно в предметних областях, які характеризуються:

– занадто великими обсягами даних, які зберігаються і обробляються;
– фізичною розосередженістю місць збирання, зберігання і використання даних;
– наявністю розвинутих засобів обчислювальної техніки і мереж передачі даних;
– можливістю обробки більшої частини інформації в місцях, де вона виникає або зберігається;
– необхідністю одночасного виконання масової обробки інформації тощо.

Розподілена система керування базою даних (СКБД) – це програмний комплекс, призначений для управління розподіленими базами даних і забезпечує прозорий доступ користувачів до розподіленої інформації.

Прозорий доступ означає непомітний для користувача, тобто у користувача має складатися враження ніби він працює з єдиною базою даних, яка розміщена на його власному комп'ютері.

Розподілена СКБД, порівняно з централізованою СКБД, повинна мати додатковий набір функціональних можливостей:
– розширені служби встановлення з'єднань повинні забезпечувати доступ до віддалених вузлів і дозволяти передавати запити і дані між вузлами, які входять у мережу;
– розширені засоби ведення каталогу, що дозволяють зберігати відомості про розподіл даних в мережі;
– засоби обробки розподілених запитів, включаючи механізми оптимізації запитів і організації віддаленого доступу до даних;
– розширені функції управління захистом, що дозволяють забезпечити дотримання правил авторизації та прав доступ до розподілених даних;
– розширені функції управління паралельним виконанням, які дозволяють підтримувати цілісність копіювання;
– розширені функції відновлення, що враховують ймовірність відмов в роботі окремих вузлів і відмов ліній зв'язку.

Всі функції розподіленої СКБД мають виконуватись прозоро для користувача.

До складу розподіленої СКБД повинні входити такі компоненти:
– *комп'ютерні робочі станції* (сайти або вузли), що формують мережеву систему;
– *компоненти мережевого обладнання та програмного забезпечення* кожної робочої станції, які дозволяють усім вузлам взаємодіяти один з одним та обмінюватися даними;
– *комунікаційні пристрої*, які переносять дані з однієї робочої станції на іншу;
– *процесор транзакцій (transaction processor, TP)* – програмний компонент, що знаходиться на кожному комп'ютері, де виконується

запит даних. Процесор транзакцій отримує і обробляє запити (віддалені та локальні);

– *процесор даних (data processor, DP)* – програмний компонент, розташований на кожному комп'ютері, де зберігаються і обробляються дані, розташовані на даному вузлі. Процесор даних може навіть являти собою централізовану СКБД.

На рис.3.2 показано розміщення і взаємодію усіх компонентів розподіленої СКБД.

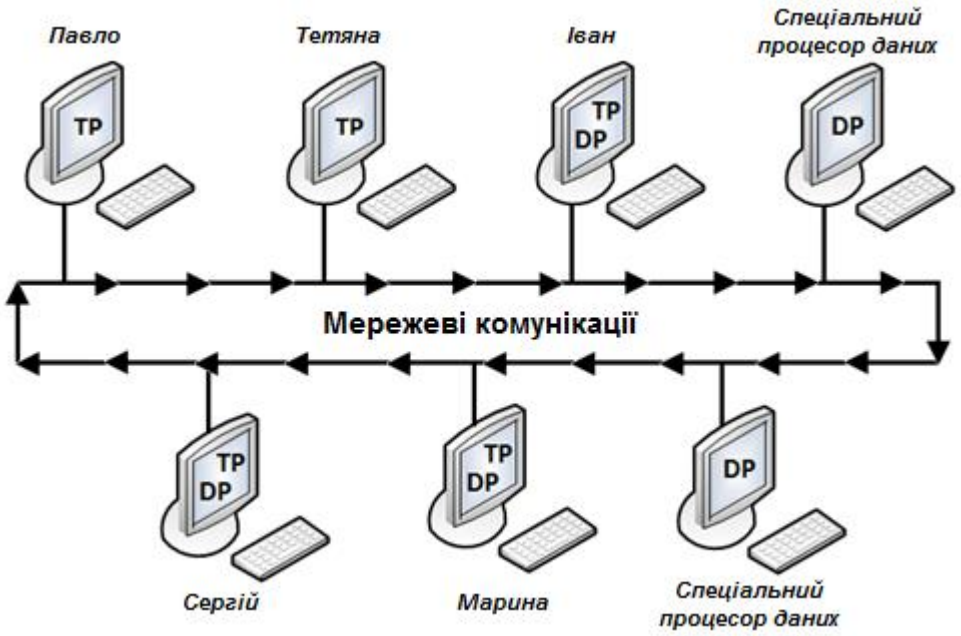

Рисунок 3.2 – Взаємодія компонентів системи розподіленої БД

Кожний TP може отримати доступ до даних на будь-якому DP і кожний DP обробляє всі запити локальних даних від будь-якого TP.

Зв'язок між TP і DP здійснюється відповідно до специфічних правил або *протоколів*, які визначають, як система розподіленої бази даних:

– організовує інтерфейс з мережею для передачі даних і команд між процесорами даних (DP) і процесорами транзакцій (TP);
– синхронізує всі дані, отримані від DP, і передає отримані дані на відповідні TP;
– забезпечує функції загального управління БД в розподіленій системі (безпека, управління паралельним виконанням, створення резервних копій і відновлення).

Процесори TP і DP можуть бути розміщені на одному і тому ж комп’ютері, дозволяючи користувачам отримувати доступ до локальних і віддалених даних, не турбуючись про їх місцезнаходження. Теоретично DP може представляти собою незалежну централізовану СКБД з відповідним інтерфейсом для підтримки віддаленого доступу до інших незалежних СКБД в мережі.

Одним із різновидів розподіленої СКБД є *мультибазова система* – розподілена система управління базами даних, в якій управління кожним вузлом здійснюється автономно.

Мультибазові системи дозволяють кінцевим користувачам різних вузлів отримувати доступ і спільно використовувати дані без необхідності фізичної інтеграції існуючих баз даних. Вони забезпечують користувачам можливість управляти базами даних їх власних вузлів без будь-якого централізованого контролю, який обов'язково присутній у звичайних типах розподілених СКБД. Адміністратор локальної бази даних може дозволити доступ до певної частини своєї бази даних за допомогою створення схеми експорту, яка визначає, до яких елементів локальної бази даних зможуть отримувати доступ зовнішні користувачі.

Існують так звані *необ'єднані* (не мають локальних користувачів) і *об'єднані* мультибазові системи. Об'єднана система являє собою деякий гібрид розподіленої та централізованої систем, оскільки вона виглядає як розподілена система для віддалених користувачів і як централізована система – для локальних.

Іншими словами, мультибазова СКБД непомітно для користувача розміщується над існуючими системами баз даних і файловими системами і розглядається користувачами як єдина база даних. Мультибазова СКБД підтримує глобальну схему, на основі якої користувачі можуть формувати запити і модифікувати дані. Мультибазова СКБД працює тільки з глобальною схемою, тоді як локальні СКБД власними силами забезпечують підтримку даних всіх своїх користувачів. Глобальна схема створюється шляхом об'єднання схем локальних баз даних. Програмне забезпечення мультибазової СКБД попередньо транслює глобальні запити і перетворює їх на запити та оператори модифікації даних відповідних локальних СКБД. Потім отримані після виконання локальних запитів результати зливаються в єдиний глобальний результуючий набір, що надається користувачеві. Крім того, мультибазова СКБД здійснює контроль за виконанням фіксації або відкату окремих операцій глобальних транзакцій локальних СКБД, а також забезпечує збереження цілісності даних в кожній з локальних баз даних. Програми мультибазової СКБД управляють різними шлюзами, за допомогою яких вони контролюють роботу локальних СКБД.

Одним із прикладів мультибазової системи є система UniSQL компанії Cincom Corporation. Вона дозволяє розробляти програми за допомогою єдиного глобального уявлення і єдиної мови доступу до бази даних для роботи з багатьма різнорідними реляційними і об'єктно-орієнтованими СКБД.

3.1.2 Відмінності між розподіленими системами баз даних, засобами розподіленої обробки даних та паралельними системами баз даних

Дуже важливо розуміти відмінність між розподіленими системами баз даних і засобами розподіленої обробки даних.

Розподілена обробка даних (distributed processing) використовує централізовану базу даних, доступ до якої може здійснюватись з різних комп'ютерів мережі. Вузол, на якому розміщена база даних, називають *сервером бази даних*. Ключовим моментом у визначенні розподіленої СКБД є твердження, що система працює з даними, які фізично розподілені в мережі, тобто *фрагменти розподіленої бази* даних розміщені на різних вузлах. Топологію системи розподіленої обробки даних зображено на рис. 3.3. Як видно з рисунка 3.3, база даних розміщена лише на одному вузлі, в той час як на рис. 3.1 можна побачити, що в розподіленій СКБД кожен вузол може містити свій фрагмент розподіленої бази даних.

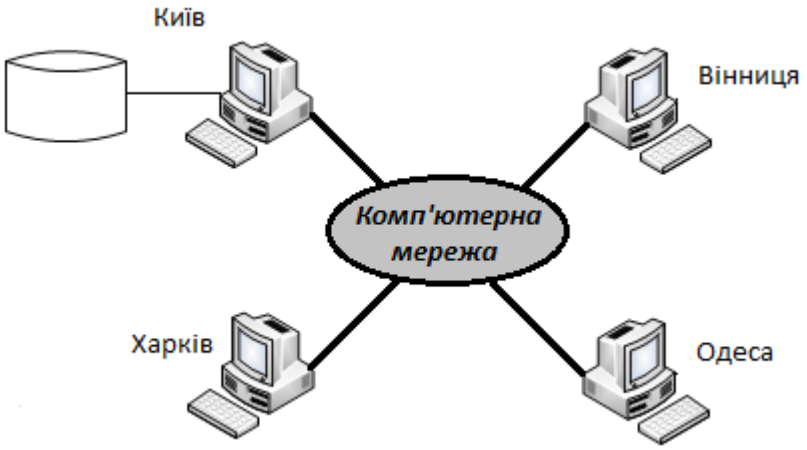

Рисунок 3.3 – Топологія середовища розподіленої обробки даних

При розподіленій обробці даних логічні процеси бази даних розподіляються серед двох або більше фізично незалежних вузлів, об'єднаних в мережу. Наприклад, розподілена обробка може виконувати введення/виведення даних, вибірку даних і перевірку даних на одному комп'ютері, а на іншому комп'ютері випускати звіт на основі отриманої інформації. Кожен вузол може мати доступ до даних та оновлювати базу даних.

Розподілена обробка даних не вимагає розподіленої бази даних, але розподілена база даних обов'язково вимагає розподіленої обробки інформації.

Спільною рисою розподіленої обробки даних і розподіленої бази даних є необхідність в локальній мережі, яка зв'язує всі компоненти між собою.

Також варто розуміти відмінності, які існують між розподіленими і паралельними СКБД. *Паралельна СКБД* функціонує з використанням декількох процесорів і жорстких дисків, що дозволяє їй розпаралелити виконання деяких операцій з метою підвищення загальної продуктивності обробки.

Застосування паралельних СКБД дозволяє об’єднати декілька малопотужних машин для отримання такого ж рівня продуктивності, що й у випадку однієї більш потужної машини, але при цьому досягається вищий рівень масштабованості та надійності системи в порівнянні з однопроцесорною СКБД.

Для надання декільком процесорам спільного доступу до однієї і тієї ж бази даних паралельна СКБД повинна забезпечувати управління сумісним доступом до ресурсів. Залежно від того, які саме ресурси поділяються, розрізняють три види паралельних СКБД:

– системи з поділом пам'яті;
– системи з поділом дисків;
– системи без поділу обчислювальних ресурсів.

Паралельна *система без поділу обчислювальних ресурсів* іноді розглядається як розподілена СКБД, однак в такій системі розподіл даних зумовлений лише прагненням до підвищення продуктивності. Крім того, вузли розподіленої СКБД зазвичай розділені географічно, знаходяться під управлінням різних адміністраторів і з'єднані між собою відносно повільними мережевими з'єднаннями, тоді як вузли паралельної СКБД найчастіше розташовуються на одному і тому ж комп'ютері або в межах одного і того ж виробничого майданчика. Архітектура системи баз даних з паралельною обробкою зображена без поділу обчислювальних ресурсів на рис.3.4.

Системи з поділом пам'яті складаються з тісно пов'язаних між собою компонентів, до числа яких входить кілька процесорів, які поділяють загальну системну пам'ять. Ця архітектура, яку ще називають архітектурою з симетричною багатопроцесорною обробкою (SMP), на даний час отримала широке поширення і застосовується для найрізноманітніших обчислювальних платформ, від персональних робочих станцій, що містять кілька паралельно працюючих мікропроцесорів, великих RISC-систем і до найбільших мейнфреймів. Ця архітектура забезпечує швидкий доступ до даних для обмеженого набору процесорів, кількість яких звичайно не перевершує 64. В іншому випадку взаємодія по мережі стає вузьким місцем всієї системи.

Системи з поділом дисків створюються з менш тісно пов'язаних між собою компонентів. Вони є оптимальним варіантом для додатків, які успадкували високу централізацію обробки і повинні забезпечувати найвищі показники доступності та продуктивності. Кожен з процесорів має безпосередній доступ до всіх спільно використовуваних дискових пристроїв, але має власну оперативну пам'ять. Як і у випадку архітектури без поділу обчислювальних ресурсів, архітектура з поділом дисків виключає вузькі місця, пов'язані з спільно використовуваною пам'яттю. Однак, на відміну від архітектури без поділу обчислювальних ресурсів, дана архітектура виключає згадані вузькі місця без внесення додаткових витрат, по-

в'язаних з фізичним розподілом даних по окремим пристроям. Колективні дискові системи іноді називають *кластерами.*

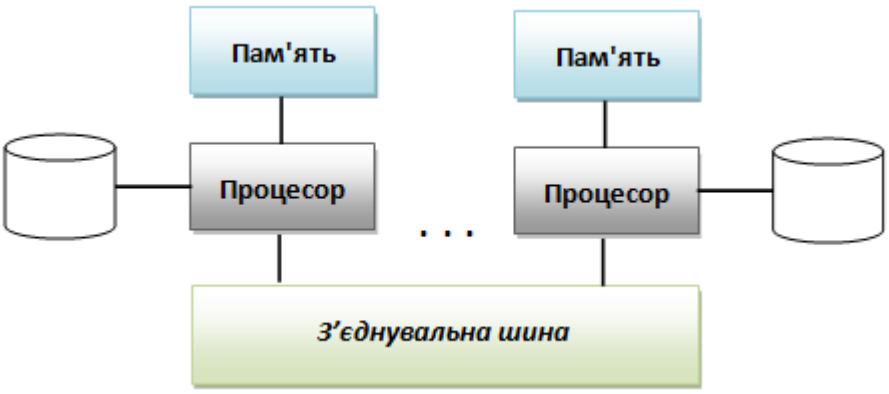

Рисунок 3.4 – Архітектура системи баз даних з паралельною обробкою без поділу обчислювальних ресурсів

Системи без поділу обчислювальних ресурсів (цю архітектуру інакше називають архітектурою з масовою паралельною обробкою) використовують схему, в якій кожен процесор, який є частиною системи, має свою власну оперативну і дискову пам'ять. База даних розподілена між всіма дисковими пристроями, які підключені до окремих, пов'язаних з цією базою даних обчислювальних підсистем, в результаті чого всі дані прозоро доступні користувачам кожної з цих підсистем. Така архітектура забезпечує більш високий рівень масштабованості, ніж системи з поділом пам'яті, і дозволяє легко підтримувати велику кількість процесорів. Однак оптимальна продуктивність буде досягнута тільки в тому випадку, якщо необхідні дані зберігаються локально.

Паралельні технології зазвичай використовуються у випадку виключно великих баз даних, розміри яких можуть досягати декількох терабайт (1012 байт), або в системах, які забезпечують виконання тисяч транзакцій в секунду. Подібні системи потребують доступу до великого обсягу даних і повинні забезпечувати прийнятний час реакції на запит.

3.1.3 Класифікація розподілених баз даних

За способом розміщення розподілені бази даних ділять на *зосереджені* і *розосереджені*.

Зосереджені (або централізовані) розподілені бази даних фізично розміщені в одному місці. Для обміну інформацією між окремими (локальними) півбазами використовуються канали зв'язку прямого доступу. Обмін даними між взаємозв'язаними півбазами здійснюється без помітних обмежень на обсяги і характер інформації, що передається.

Такі бази даних мають ряд переваг:

– просту побудову бази даних,

– зведення до мінімуму дублювання інформації,
– максимальну уніфікацію методів зберігання, коригування і пошуку інформації.

Проте зосереджені бази даних в одному місці – вузлі мережі – мають цілий ряд недоліків: при централізації зберігання значно збільшується час на передачу інформації і за рахунок цього зростає час реакції системи; централізована система обмежена обсягами пам'яті ЕОМ тощо.

Розосереджені (або децентралізовані) розподілені бази даних фізично розміщені в різних місцях – вузлах обчислювальної мережі. Обмін інформацією між підбазами здійснюються з використанням каналів зв'язку. Як підбази розподіленої бази даних можуть використовуватись зосереджені (централізовані) бази даних і окремі (локальні) підбази.

Обмін інформацією між взаємозв'язаними підбазами здійснюється головним чином результатною (обробленою, узагальненою) інформацією. При виконанні запиту в таких системах використовується декомпозиція запиту на підзапити до локальних підбаз і паралельне виконання виділених підзапитів у різних вузлах обчислювальної мережі.

Ці бази даних мають безперечні переваги у порівнянні з централізованими:

– обсяги пам'яті обмежені пам'ятю не однієї ЕОМ, а сумарною пам'ятю ЕОМ, які знаходяться в усіх вузлах мережі;
– зменшуються затрати на передавання інформації, так як у кожному вузлі знаходиться та інформація, яка необхідна конкретному користувачу і по можливості забезпечує всі його інформаційні потреби.

Однак розосереджена база даних призводить до неминучого дублювання деякої інформації, безконтрольності її зростання, а також значно ускладнюється проблема зберігання несуперечності інформації.

Залежно від рівня підтримки різних типів локальних СКБД, розподілені СКБД поділяються на *гомогенні* (*однорідні)* та *гетерогенні* (*різнорідні)*.

В гомогенних системах усі вузли використовують один і той же тип СКБД. У різнорідних системах на вузлах можуть функціонувати різні типи СКБД, що використовують різні моделі даних, тобто різнорідна система може включати вузли з реляційними, мережевими, ієрархічними або об'єктно-орієнтованими СКБД.

Однорідні системи значно простіше проектувати і супроводжувати. Крім того, подібний підхід дозволяє поетапно нарощувати розміри системи, послідовно додаючи нові вузли до вже існуючої розподіленої системи. Додатково з'являється можливість підвищувати продуктивність системи за рахунок організації на різних вузлах паралельної обробки інформації.

Різнорідні системи зазвичай виникають в тих випадках, коли незалежні вузли, які вже експлуатують свої власні системи з базами даних, з ча-

сом інтегруються у новостворювану розподілену систему. У різнорідних системах для організації взаємодії між різними типами СКБД потрібно забезпечити перетворення переданих повідомлень. Для забезпечення прозорості щодо типу використовуваної СКБД користувачі кожного з вузлів повинні мати можливість формулювати запити мовою тієї СКБД, яка використовується на їх локальному вузлі. Система повинна взяти на себе пошук необхідних даних і виконання всіх необхідних перетворень переданих повідомлень. У загальному випадку дані можуть бути затребувані з іншого вузла, який характеризується такими особливостями:

- інший тип використовуваного обладнання;
- інший тип використовуваної СКБД;
- інший тип застосовуваних обладнання та СКБД.

Якщо використовується інший тип обладнання, але на вузлах застосовуються однакові СКБД, методи виконання перетворень цілком очевидні і включають заміну кодів і зміну довжини машинного слова. Якщо типи використовуваних на вузлах СКБД різні, процедура перетворення ускладнюється тим, що необхідно перетворювати структури даних однієї моделі даних в еквівалентні структури даних іншої моделі даних. Наприклад, відношення в реляційній моделі даних повинні бути перетворені в записи і набори, характерні для мережевої моделі даних. Крім того, доводиться транслювати текст запитів з однієї мови в іншу (наприклад, запити з оператором SELECT мови SQL може знадобитися перетворити в запити з операторами GET і FIND мови маніпулювання даними мережевої СКБД). Якщо відрізняються і тип використовуваного обладнання, і тип програмного забезпечення, то необхідно виконати обидва види трансляції. Все викладене вище надзвичайно ускладнює обробку даних в різнорідних розподілених СКБД.

В гетерогенних системах може виникати проблема семантичної неоднорідності. Наприклад, атрибути, що мають у різних схемах одне і те ж ім’я, фактично можуть представляти абсолютно різні поняття. Аналогічно, атрибути з різними іменами фактично можуть представляти одну і ту ж характеристику.

Типове рішення, що застосовується в деяких реляційних системах, полягає в тому, що окремі частини різнорідних розподілених систем повинні використовувати шлюзи, призначені для перетворення мови та моделі даних кожного з використовуваних типів СКБД в мову і модель даних реляційної системи. Однак підходу із застосуванням шлюзів властиві деякі серйозні обмеження. По-перше, шлюзи не дозволяють організувати систему управління транзакціями навіть для окремих пар систем. Іншими словами, шлюз між двома системами представляє собою не більш ніж транслятор запитів. Також шлюзи не дозволяють системі координувати управління паралельним виконанням та процедурами відновлення транзакцій, які включають оновлення даних в обох базах. По-друге, використання шлюзів дозволяє вирішити лише завдання трансляції запитів з мови

однієї СКБД на мову іншої. Тому вони, як правило, не дозволяють вирішити проблему створення однорідної структури і усунути відмінності між представленнями даних у різних схемах.

Для подолання проблем гетерогенних систем комітет Open Group організував робочу групу (Specification Working Group – SWG), покликану підготувати специфікації, що регламентують інфраструктуру середовища бази даних, яка включає такі елементи:

- уніфікований і досить потужний інтерфейс мови SQL (SQL API), який дозволяє створювати клієнтські програми таким чином, щоб вони не були прив'язані до конкретного типу використовуваної СКБД;
- уніфікований протокол доступу до бази даних, що дозволяє безпосередньо взаємодіяти СКБД різних типів без необхідності використання будь-якого шлюзу;
- уніфікований мережевий протокол, що дозволяє здійснювати взаємодію СКБД різних типів.

Найважливішим завданням цієї групи слід вважати пошук способу, який дозволяє в одній транзакції виконувати обробку даних, що містяться в декількох базах, керованих СКБД різних типів, причому без необхідності застосування будь-яких шлюзів.

3.1.4 Переваги та недоліки систем керування розподіленими базами даних

Система керування розподіленою БД порівняно з традиційними системами має ряд *переваг*:

- *дані розташовуються близько до найзатребуванішого вузла*. Дані в розподіленій БД розміщуються на тому вузлі, де в них є найбільша потреба;
- *високий ступінь локальної автономності*. Користувачі, які найчастіше працюють з даними на деякому локальному вузлі можуть здійснювати локальний контроль над необхідними їм даними, встановлювати або регулювати локальне обмеження на їх використання. Адміністратор глобальної бази даних (АБД) відповідає за систему в цілому. Як правило, частина цієї відповідальності делегується на локальний рівень, завдяки чому АБД локального рівня отримує можливість управляти локальною СКБД;
- *швидкий доступ до даних*. Кінцеві користувачі часто працюють тільки з деякою підмножиною даних компанії. При цьому підмножина може зберігатися локально, і система бази даних забезпечить більш оперативний доступ до даних, ніж централізована БД, де дані зберігаються віддалено;

- *підвищення продуктивності.* В розподіленій БД має місце розпаралелення процесів обробки даних. Оскільки кожен вузол працює лише з частиною бази даних, то ступінь використання центрального процесора та служб введення-виведення може виявитися нижчим, ніж у випадку централізованої БД;
- *підвищення надійності та доступності даних.* У централізованих СКБД відмова центрального комп'ютера приведе до припинення функціонування всієї СКБД. На відміну від них, розподілена СКБД дозволяє перемісити операції з вузла, що вийшов з ладу, на інший. Навантаження системи розподіляється між іншими робочими станціями, оскільки, завдяки реплікації, дані зберігаються на декількох вузлах;
- *модульність системи.* Розширення розподіленої бази даних відбувається шляхом додавання до мережі нового вузла, який не впливає на функціонування вже існуючих вузлів. В централізованих СКБД розширення бази даних може вимагати заміни обладнання та програмного забезпечення;
- *зменшення вартості організації і витрат на експлуатацію баз даних*. У 1960-ті роки потужність обчислювальних засобів зростала пропорційно квадрату вартості її устаткування, тому система, вартість якої була втричі вища за вартість даної, перевершувала її за потужністю у дев'ять разів. Ця залежність отримала назву закону Гроша. Однак сьогодні вважається загальноприйнятим положення, згідно з яким набагато дешевше зібрати з невеликих комп'ютерів систему, потужність якої буде еквівалента потужності одного великого комп'ютера (мейнфрейма). Виявляється, що значно вигідніше встановлювати в підрозділах організації власні малопотужні комп'ютери та додавати в мережу нові робочі станції, ніж модернізувати систему з мейнфреймом. В багатьох корпораціях на мейнфреймах виконують лише вузькоспеціалізовані задачі, а всі інші – на персональних комп’ютерах;
- *зменшення вартості передачі даних*. Вартість передачі даних через комп’ютерну мережу є відносно високою порівняно з локальним доступом до даних. Вона визначається не лише кількістю переданих бітів інформації, а й витратами на підтримку багаторівневого мережевого протоколу, який управляє підготовкою інформаційних пакетів до передачі і збором отриманих пакетів на вузлі-приймачі. Реалізація кожного рівня протоколу вимагає значних обчислювальних витрат. У розподіленій СКБД забезпечується можливість розмістити дані на тому вузлі, де в них є найбільша потреба і обробляти їх локально;
- *незалежність від вузла обробки даних.* Кінцевий користувач отримує доступ до будь-якої наявної копії даних, а запит кінцево-

го користувача обробляється будь-яким вузлом в місці розташування даних. Іншими словами, запит не залежать від конкретного вузла обробки даних: будь-який доступний вузол може обробити запит користувача;

– *збільшення обсягу збережених і доступних для обробки даних*. Обсяг даних не обмежується об’ємом пам’яті мейнфрейма. Перевантаження через збільшення розміру бази даних зазвичай усуваються шляхом додавання до мережі нових обчислювальних потужностей і пристроїв зовнішньої пам'яті;
– *зменшення обсягів даних, які пересилаються*. Досягається завдяки тому, що частина даних зберігається і обробляється локально.

На жаль, сучасні розподілені СКБД мають ряд *недоліків*:

– *складність управління і контролю*. Управління розподіленими даними – більш важке завдання, ніж управління централізованими даними. Додатки повинні визначати місцезнаходження даних і вміти потім зв'язувати в єдине ціле інформацію, отриману з різних місць. Адміністратори БД повинні мати можливість координувати дії бази даних, щоб запобігти погіршенню якості БД через аномалії даних. Особливу увагу слід приділяти управлінню транзакціями, паралельному виконанню, безпеці, резервному копіюванню, реплікації даних, відновленню процесів, оптимізації запитів, вибору шляхів доступу і т. д;
– *ускладнення процедури розробки розподіленої БД.* Розробка розподілених баз даних, вимагає прийняття рішення про фрагментацію даних, розподіл фрагментів по окремим вузлам і реплікацію даних;
– *безпека.* В централізованих системах доступ до даних легко контролюється. В розподілених системах відповідальність за управління даними розподіляється між різними людьми, що знаходяться в різних місцях, а локальні мережі все ще залишаються уразливими у відношенні безпеки;
– *недостатня стандартизація.* Незважаючи на те, що розподілені бази даних залежать від ефективності комунікацій, поки не існує стандартного протоколу обміну інформацією на рівні баз даних (хоча TCP/IP є фактично стандартним протоколом на рівні мереж, відсутній стандарт протоколу на рівні додатків). Відсутність стандартів істотно обмежує потенційні можливості розподілених СКБД. Крім того, не існує інструментальних засобів і методологій, здатних допомогти користувачам перетворити централізовані системи в розподілені;
– *складність управління середовищем даних.* Організувати доступ до диска і зберігання даних у розподіленому середовищі складніше, ніж в централізованій системі, тому управління таким середови-

щем даних стає більш складним у відношенні як людських ресурсів, так і програмного забезпечення;

– *підвищення вартості навчання.* Вартість навчання в розподіленій моделі, як правило, вище, ніж в централізованій, і часто порівнюється з вартістю обладнання;
– *нестача досвіду.* Ще не накопичено достатньо досвіду промислової експлуатації розподілених систем. Таке становище є значним стримуючим фактором для потенційних прихильників розподіленої технології;
– *підвищені вимоги до умов зберігання даних*. У розподілених БД декілька копій даних необхідно розміщувати на декількох сайтах, що вимагає додаткових ресурсів (місце на жорсткому диску). Цей недолік не є дуже істотним, оскільки вартість зберігання інформації на жорстких дисках швидко зменшується.

Сьогодні розподілені БД з успіхом експлуатуються, однак гнучкість і потужність, якими вони теоретично володіють, використовується не в повній мірі. Властива розподіленим базам даних складність вимагає невідкладної розробки стандартів на протоколи управління транзакціями, паралельного виконання, безпеки, резервного копіювання і відновлення, оптимізації, вибору шляхів доступу і т. д.

3.1.5 Принципи створення розподілених БД

Фундаментальний принцип створення розподілених баз даних (*«правило 0»*) передбачає їх організацію таким чином, щоб для користувача розподілені системи виглядали так само, як і нерозподілені.

З фундаментального принципу витікають додаткові правила, запропоновані К.Дейтом:

1. *Незалежність локального вузла*. Кожний локальний вузол може працювати як незалежна, або автономна СКБД. Всі операції на вузлі контролюються цим вузлом: безпека, управління паралельним виконанням, резервне копіювання і відновлення даних.
2. *Незалежність від центрального вузла.* Всі вузли мають рівні можливості. Жодний вузол в мережі не повинен звертатись до «центрального» вузла з метою отримання певного централізованого сервісу.
3. *Незалежність від збоїв.* Функціонування системи не залежить від збою на якомусь вузлі. Система продовжує виконання операцій навіть при несправності вузла або при розширенні мережі.
4. *Прозорість місця розташування.* Користувачу не потрібно знати фізичне місце розташування даних, щоб здійснити їх пошук. Користувач працює з даними так, ніби вони розташовані на його локальному вузлі.

5. *Прозорість фрагментації.* Користувачу не потрібно знати імена фрагментів БД, щоб отримати доступ до них. Користувач бачить єдину логічну базу даних.
6. *Прозорість реплікації.* Для користувачів повинно бути створено таке середовище, щоб вони могли вважати, що в дійсності дані не дублюються. Тобто, користувачу не потрібно мати відомості про існуючі копії фрагментів.
7. *Розподілена обробка запитів.* Для обробки запиту може знадобитись звернення до декількох вузлів. В розподіленій системі може існувати багато способів пересилання даних для виконання цього запиту. СУРБД повинна виконати оптимізацію запиту прозоро для користувача.
8. *Управління розподіленими транзакціями.* Транзакції можуть оновлювати дані на декількох різних вузлах. Виконання транзакцій відбувається прозоро для користувача.
9. *Апаратна незалежність.* Система повинна виконуватись на різних апаратних платформах.
10. *Незалежність від операційної системи.* Система повинна виконуватись в будь-якій операційній системі.
11. *Незалежність від мережі.* Можливість підтримувати багато типів комунікаційних мереж.
12. *Незалежність від типу бази даних.* Необхідно, щоб екземпляри БД на різних вузлах всі разом підтримували один і той же інтерфейс, і зовсім необов'язково, щоб це були копії однієї і тієї ж версії БД. Іншими словами, розподілені бази даних можуть бути неоднорідними.

На сьогодні жодна з розподілених баз даних не відповідає *усім* наведеним правилам, однак правила Дейта повністю описують розподілені бази даних і відіграють важливу роль при їх розробці.

3.2 Проектування розподіленої БД

При проектуванні розподіленої БД виникають три нові проблеми.

– Як розбивати базу даних на фрагменти?

– Які фрагменти необхідно дублювати (реплікувати або тиражувати)?

– Де розташувати ці фрагменти і репліки?

Фрагментація даних і реплікація даних відносяться до перших двох зазначених проблем, а розміщення даних – до третьої.

Визначення та розміщення фрагментів повинно відбуватись з урахуванням особливостей використання бази даних. Зокрема, необхідно проводити аналіз транзакцій та запитів. Згідно з емпіричним правилом Парето доцільно проводити аналіз лише 20% найбільш активних запитів користувачів, оскільки вони створюють 80% навантаження на базу даних. Для визначення того, які з транзакцій підлягають детальному аналізу, можна

скористатися *таблицею відповідності транзакцій і відношень*, в якій показані відношення, доступ до яких відбувається при виконанні кожної транзакції, а також *діаграмою частоти виконання транзакцій*, яка схематично показує відношення, вірогідність використання яких в транзакціях є найбільшою.

Проектування повинно проводитись на основі як кількісних, так і якісних показників. Кількісна інформація використовується як основа для розміщення, а якісна слугує базою при створенні схем фрагментації. Кількісна інформація включає такі показники:

– частота виконання транзакції;

– вузол, на якому виконується транзакція;

– вимоги до продуктивності транзакцій.

Якісна інформація може містити відомості про виконувані транзакції:

- використовувані відношення, атрибути та рядки;
- тип доступу (читання або запис);
- предикати операцій читання.

Визначення та розміщення фрагментів по вузлах виконується для досягнення п’яти основних стратегічних цілей.

1. **Локалізація посилань.** Всюди, де це можливо, дані повинні зберігатись якомога ближче до місць їх використання. Якщо фрагмент використовується на декількох вузлах, то може виявити доцільним розмістити на них його копії (репліки).
2. **Підвищення надійності та доступності.** Надійність та доступність даних підвищується за рахунок використання механізму реплікації. У випадку відмови одного з вузлів завжди буде існувати копія фрагмента, яка зберігається на іншому вузлі.
3. **Задовільний рівень продуктивності.** Невірне розміщення фрагментів може призвести до виникнення вузьких місць в системі. У цьому випадку деякий вузол буде перевантажений запитами від інших вузлів, що суттєво може знизити продуктивність всієї системи. В той же час неправильне розміщення може призвести до неефективного використання ресурсів системи.
4. **Компроміс між ємністю та вартістю зовнішньої пам’яті.** Всюди, де це можливо, рекомендовано використовувати дешеві пристрої масової пам’яті. Ця вимога повинна бути узгоджена з вимогою забезпечення локалізації посилань.
5. **Мінімізація витрат на передачу даних.** Варто ретельно враховувати вартість виконання в системі віддалених запитів. Витрати на вибірку будуть мінімальними при забезпеченні максимальної локалізації посилань, тобто тоді, коли кожен вузол буде мати свою власну копію необхідних йому даних. Однак при оновленні даних реплікованих даних внесені зміни необхідно буде поширити на всі

вузли, які мають копію оновленого відношення, що збільшить витрати на передачу даних.

3.2.1 Фрагментація даних

Фрагментація даних допускає розбиття одного об'єкта на два або більше сегментів чи фрагментів. Об'єкт може являти собою користувацьку базу даних, системну базу даних або таблицю. Кожен фрагмент може зберігатися на будь-якому вузлі комп'ютерної мережі. Інформація про фрагментацію даних зберігається в каталозі розподілених даних (distributed data catalog, DDC), до якого процесор транзакцій (TP) може отримати доступ при обробці запитів користувача.

Необхідність у фрагментації зумовлена рядом причин:

– *умови використання.* Найчастіше програми працюють з деякими представленнями, а не з повними базовими відношеннями. Тому з точки зору розміщення даних доцільніше організувати роботу додатків з певними підмножинами відношень, які розглядаються як мінімальна одиниця розміщення;

– *ефективність.* Дані зберігаються в тих місцях, де вони найчастіше використовуються. Крім того, виключається необхідність зберігання даних, які не використовуються локальними додатками;

– *паралельність.* Оскільки фрагменти є мінімальними одиницями розміщення, транзакції можуть бути розділені на декілька підзапитів, що звертаються до різних фрагментів. Такий підхід дає можливість підвищити рівень паралельності обробки в системі, тобто дозволяє транзакціям, які допускають це, ефективно виконуватися в паралельному режимі.

– *захищеність.* Дані, які не використовуються локальними додатками, не зберігаються на вузлах, і відповідно, користувачі, які не володіють відповідними правами, не зможуть отримати до них доступ.

В розподіленій системі можливі три види фрагментації даних: *горизонтальна*, *вертикальна* і *змішана*. Зазначені види фрагментації даних діють на рівні таблиць і полягають у розбитті таблиці на логічні фрагменти.

При проведенні фрагментації необхідно дотримуватись трьох правил.

1.Повнота. Якщо екземпляр відношення R розбивається на фрагменти, наприклад R_1, R_2, … R_n, то кожний елемент даних, присутній у відношенні R, має міститися, принаймні, в одному зі створених фрагментів. Виконння цього правила гарантує, що жодні дані не будуть втрачені в результаті виконання фрагментації.

2.Відновлюваність. Повинна існувати операція реляційної алгебри, яка дозволила б відновити відношення R з його фрагментів. Це правило гарантує збереження функціональних залежностей.

3.Відсутність перетинань. Якщо елемент даних міститься у фрагменті R_1, то він не повинен одночасно бути присутнім в будь-якому іншому фрагменті. Виключенням з цього правила є операція вертикальної фрагментації, оскільки в цьому випадку в кожному фрагменті повинні бути присутні атрибути первинного ключа, необхідні для відновлення вихідного відношення. Це правило гарантує мінімальну надлишковість даних у фрагментах.

У випадку горизонтальної фрагментації елементом є рядок, а у випадку вертикальної фрагментації – атрибут.

Горизонтальна фрагментація. При *горизонтальній фрагментації* розбиття таблиці (відношення) відбувається за рахунок розміщення в окремій таблиці з однаковою структурою унікальних (не перекриваються) груп рядків (кортежів). Фактично здійснюється зберігання рядків однієї логічної таблиці в декількох ідентичних фізичних таблицях на різних вузлах. Кожен фрагмент зберігається на окремому вузлі і кожен фрагмент має унікальні рядки. Однак всі унікальні рядки мають однакові атрибути (стовпці). Інакше кажучи, кожен фрагмент еквівалентний оператору SELECT з модифікуючим виразом WHERE по єдиному атрибуту.

Нехай компанія «Геліос» має представництва у Вінниці (VIN), Хмельницькому (KHM) та Тернополі (TER). Керівництву необхідна інформація про клієнтів по всім трьом містам, але кожному підрозділу компанії необхідна інформація лише по своїм локальним клієнтам. Враховуючи це, доцільним є розподіл даних по містах з використанням горизонтальної фрагментації (табл. 3.1).

Таблиця 3.1 – Горизонтальна фрагментація таблиці CUSTOMERS по містах

Фрагмент	Місце розташування	Умова	Вузол	Номера клієнтів	Кількість рядків
CUST_1H	Вінниця	CUS_REGION='VIN'	BIR	24	1
CUST_2H	Хмельницький	CUS_REGION='KHM'	KTB	21,23	2
CUST_3H	Тернопіль	CUS_REGION='TER'	DAS	20,22,25	3

Кожний горизонтальний фрагмент може мати довільну кількість рядків, але повинен мати такі ж атрибути, що й інші фрагменти. Після фрагментації будуть створені три таблиці, показані на рис. 3.5.

В одних випадках доцільність використання горизонтальної фрагментації цілком очевидна, а в інших випадках потрібне виконання детального аналізу додатків. Цей аналіз повинен включати перевірку *предикатів* (або умов) пошуку, які використовуються у транзакціях або запитах, що виконуються в додатку. Предикати можуть бути простими (включають один атрибут) та складними (включають декілька атрибутів). Для кожного з використовуваних атрибутів предикат може містити єдине значення або

декілька значень. В останньому випадку значення можуть бути дискретними або входити в діапазон значень.

Фрагмент: CUST_1H Місце розташування: Вінниця Вузол: BIR

	CUS_NO	CUS_NAME	CUS_ADDRESS	CUS_REGION	CUS_KRED	CUS_DEB
	24	Сінтек, ПП	вул.Чорновола,12	VIN	12 000,00₴	7 000,00₴

Фрагмент: CUST_2H Місце розташування: Хмельницький Вузол: KTB

	CUS_NO	CUS_NAME	CUS_ADDRESS	CUS_REGION	CUS_KRED	CUS_DEB
	21	Агронік, ТОВ	вул.Мазепи,46	KHM	35 000,00₴	2 000,00₴
	23	Рондос, ТОВ	вул.Зелена,120	KHM	24 000,00₴	10 000,00₴

Фрагмент: CUST_3H Місце розташування: Тернопіль Вузол: DAS

	CUS_NO	CUS_NAME	CUS_ADDRESS	CUS_REGION	CUS_KRED	CUS_DEB
	20	Агросоюз, ТОВ	вул.Литвиненка,1	TER	46 000,00₴	8 500,00₴
	22	Берег, ПП	вул.Зарічна,130	TER	21 000,00₴	5 000,00₴
	25	Іванченко, ФОП	вул.Весела,28	TER	7 000,00₴	1 000,00₴

Рисунок 3.5 – Фрагменти таблиці по місцю розташування

Стратегія визначення типу фрагментації передбачає пошук набору *мінімальних* (тобто повних і релевантних) *предикатів*, які можна використовувати як основу для створення схеми фрагментації. Набір предикатів є повним тоді і тільки тоді, коли ймовірність звернення до будь-яких двох рядків одного і того ж фрагмента з боку будь-якої транзакції однакова. Предикат є релевантним, якщо існує, принаймні, одна транзакція, яка порізному звертається до виділених за допомогою цього предиката фрагментів.

Вертикальна фрагментація. Такий тип фрагментації передбачає поділ таблиці (відношення) на підмножини стовпців (атрибутів). Кожний фрагмент зберігається на окремому вузлі і має унікальні стовпці – за винятком ключового стовпця, який є у всіх фрагментах. Це еквівалентно застосуванню оператора PROJECT. Нехай у компанії є два відділи: відділ обслуговування і відділ прийому платежів. Кожний відділ розташований в різних будівлях і кожному відділу потрібна інформація лише по декільком атрибутам таблиці CUSTOMERS. У цьому випадку доцільно виконати вертикальну фрагментацію (табл.3.2).

Таблиця 3.2 – Вертикальна фрагментація таблиці CUSTOMERS по атрибутах

Фрагмент	**Місце розташування**	**Вузол**	**Атрибути**
CUST_1V	Приміщення обслуговування	IML	CUS_NO, CUS_NAME, CUS_ADDRESS, CUS_REGION
CUST_2V	Приміщення прийому платежів	SNT	CUS_NO, CUS_KRED, CUS_DEB

Кожний вертикальний фрагмент має однакову кількість рядків, але включає в себе різні атрибути і спільний для усіх фрагментів ключовий атрибут.

Результати вертикальної фрагментації наведено на рис.3.6.

Місце розташування: Приміщення обслуговування

Фрагмент: CUST_1V Вузол: IML

	CUS_NO	CUS_NAME	CUS_ADDRESS	CUS_REGION
	20	Агросоюз, ТОВ	вул.Литвиненка,1	TER
	21	Агронік, ТОВ	вул.Мазепи,46	KHM
	22	Берег, ПП	вул.Зарічна,130	TER
	23	Рондос, ТОВ	вул.Зелена,120	KHM
	24	Сінтек, ПП	вул.Чорновола,12	VIN
	25	Іванченко, ФОП	вул.Весела,28	TER

Місце розташування: Приміщення прийому платежів

Фрагмент: CUST_2V Вузол: SNT

	CUS_NO	CUS_KRED	CUS_DEB
	20	46 000,00₴	8 500,00₴
	21	35 000,00₴	2 000,00₴
	22	21 000,00₴	5 000,00₴
	23	24 000,00₴	10 000,00₴
	24	12 000,00₴	7 000,00₴
	25	7 000,00₴	1 000,00₴

Рисунок 3.6 – Вміст таблиць при вертикальній фрагментації

При формуванні вертикальних фрагментів необхідно враховувати сполучуваність атрибутів один з одним. Один із способів визначення сполучуваності атрибутів полягає у створенні *матриці*, яка містить кількість звернень до кожної пари атрибутів. Наприклад, транзакція, яка здійснює доступ до атрибутів a_1, a_2 і a_4 відношення R, що складається з набору атрибутів (a_1, a_2, a_3, a_4), може бути представлена такою матрицею:

$$\begin{array}{c} \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{array} \begin{array}{c} \begin{array}{cccc} a_1 & a_2 & a_3 & a_4 \end{array} \\ \begin{bmatrix} & 1 & 0 & 1 \\ & & 0 & 1 \\ & & & 0 \\ & & & \end{bmatrix} \end{array}$$

Ця матриця є трикутною; її діагональ не заповнюється, а нижня частина є дзеркальним відображенням верхньої частини і тому може не розглядатися. Одиниці в матриці означають наявність доступу із зверненням до відповідної пари атрибутів і, в кінцевому рахунку, повинні бути замінені числами, що відображають частоту виконання транзакції. Подібна матриця складається для кожної транзакції, після чого створюється загальна матриця, яка містить суми всіх показників доступу до кожної з пар

атрибутів. Пари атрибутів з високим показником сполучуваності повинні бути присутніми в одному і тому ж вертикальному фрагменті. Пари з невисоким показником сполучуваності можуть бути розподілені по різним вертикальним фрагментами. Очевидно, що обробка відомостей про окремі атрибути для всіх найважливіших транзакцій може потребувати багато часу і обчислень. Тому, якщо дані про сполучуваність певних атрибутів заздалегідь накопичені, може виявитися доцільним обробляти відомості відразу про групи атрибутів.

Подібний підхід носить назву *розщеплення* (splitting) і вперше був запропонований в 1984 році. Він дозволяє виділити набір фрагментів, які гарантовано будуть відповідати правилу відсутності перетинань. Фактично вимога відсутності перетинань стосується лише атрибутів, що не входять в первинний ключ відношення. Атрибути первинного ключа повинні бути присутніми в кожному з виділених вертикальних фрагментів, тому можуть не розглядатися при аналізі.

Змішана фрагментація. Змішана фрагментація являє собою комбінацію вертикальної і горизонтальної фрагментацій. Іншими словами, таблиця може поділятися на кілька горизонтальних множин (рядків), кожна з яких поділяється на безліч атрибутів (стовпців).

Таблиця 3.3 – Змішана фрагментація таблиці CUSTOMERS

Фрагмент	Місце розташування	Умова горизонтальної фрагментації	Вузол	Номера клієнтів	Атрибути вертикальної фрагментації на кожному фрагменті
CUST_1M	Обслуговування у Вінниці	CUS_REGION='VIN'	BIR_S	24	CUS_NO, CUS_NAME, CUS_ADDRESS, CUS_REGION
CUST_2M	Прийом платежів у Вінниці	CUS_REGION='VIN'	BIR_P	24	CUS_NO, CUS_KRED, CUS_DEB
CUST_3M	Обслуговування в Хмельницькому	CUS_REGION='KHM'	KTB_S	21,23	CUS_NO, CUS_NAME, CUS_ADDRESS, CUS_REGION
CUST_4M	Прийом платежів в Хмельницькому	CUS_REGION='KHM'	KTB_P	21,23	CUS_NO, CUS_KRED, CUS_DEB
CUST_5M	Обслуговування в Тернополі	CUS_REGION='TER'	DAS_S	20,22, 25	CUS_NO, CUS_NAME, CUS_ADDRESS, CUS_REGION
CUST_6M	Прийом платежів в Тернополі	CUS_REGION='TER'	DAS_P	20,22, 25	CUS_NO, CUS_KRED, CUS_DEB

Припустимо, що структура компанії «Геліос» вимагає, щоб дані таблиці CUSTOMERS були фрагментовані горизонтально, відображаючи поділ компанії на представництва, і в той же час в межах представництва дані необхідно фрагментувати вертикально, щоб виділити відділи обслуговування і прийому платежів. В цьому випадку необхідно виконати змі-

шану фрагментацію. Спочатку потрібно провести горизонтальну фрагментацію на кожному вузлі на основі розбиття компанії по представництвам (CUS_REGION). В результаті отримаємо набір кортежів клієнтів (горизонтальні фрагменти), які розташовані на різних вузлах. Оскільки відділи розташовані в різних приміщеннях, необхідно виконати вертикальну фрагментацію в межах кожного горизонтального фрагмента для розбиття атрибутів, що забезпечить кожне представництво необхідною йому інформацією. Результати змішаної фрагментації наведено в талб.3.3.

Кожний фрагмент в табл.3.3 містить дані клієнтів по регіону, в межах кожного регіону – по локальним відділам. Таблиці, які утворено внаслідок змішаної фрагментації, показано на рис.3.7.

Місце розташування: Приміщення обслуговування у Вінниці

Фрагмент: CUST_1M — Вузол: BIR_S

	CUS_NO	CUS_NAME	CUS_ADDRESS	CUS_REGION
	24	Сінтек, ПП	вул.Чорновола,12	VIN

Місце розташування: Приміщення прийому платежів у Вінниці

Фрагмент: CUST_2M — Вузол: BIR_P

	CUS_NO	CUS_KRED	CUS_DEB
	24	12 000,00₴	7 000,00₴

Місце розташування: Приміщення обслуговування у Хмельницькому

Фрагмент: CUST_3M — Вузол: KTB_S

	CUS_NO	CUS_NAME	CUS_ADDRESS	CUS_REGION
	21	Агронік, ТОВ	вул.Мазепи,46	KHM
	23	Рондос, ТОВ	вул.Зелена,120	KHM

Місце розташування: Приміщення прийому платежів у Хмельницькому

Фрагмент: CUST_4M — Вузол: KTB_P

	CUS_NO	CUS_KRED	CUS_DEB
	21	35 000,00₴	2 000,00₴
	23	24 000,00₴	10 000,00₴

Місце розташування: Приміщення обслуговування у Тернополі

Фрагмент: CUST_5M — Вузол: DAS_S

	CUS_NO	CUS_NAME	CUS_ADDRESS	CUS_REGION
	20	Агросоюз, ТОВ	вул.Литвиненка,1	TER
	22	Берег, ПП	вул.Зарічна,130	TER
	25	Іванченко, ФОП	вул.Весела,28	TER

Місце розташування: Приміщення прийому платежів у Тернополі

Фрагмент: CUST_6M — Вузол: DAS_P

	CUS_NO	CUS_KRED	CUS_DEB
	20	46 000,00₴	8 500,00₴
	22	21 000,00₴	5 000,00₴
	25	7 000,00₴	1 000,00₴

Рисунок 3.7 – Вміст таблиць після змішаної фрагментації

Механізму фрагментації притаманні два основних недоліки:

– *продуктивність.* Продуктивність глобальних додатків, які потребують доступу до даних з декількох фрагментів, розташованих на різних вузлах, може виявитися нижче, ніж локальних;
– *цілісність даних.* Підтримка цілісності даних може істотно ускладнюватися, оскільки функціонально залежні дані можуть зробитися фрагментованими і розміщуватися на різних вузлах.

Від фрагментації варто відмовитись у випадку, коли відношення містить невелику кількість рядків, які оновлюються відносно рідко. В такому випадку краще розмістити копію такого відношення на кожному з вузлів мережі.

3.2.2 Реплікація даних

Реплікація (тиражування) даних (Data Replication) – це механізм синхронізації вмісту декількох копій фрагментів розподіленої БД. Під реплікацією слід розуміти *асинхронний* переніс змін об'єктів вихідної бази даних (source database) в бази даних, що належать різним вузлам розподіленої системи.

Можливі три варіанти реплікації БД:

– *повністю репліковані* БД*:* зберігає копії одного й того ж фрагмента БД на всіх вузлах мережі. В даному випадку всі фрагменти БД репліковані. Така БД може виявитися не зручною у використанні через великі витрати.
– *частково репліковані* БД*:* зберігає копії одного й того ж фрагмента БД на декількох вузлах мережі (рис.3.8). Більшість СУРБД допускають роботу саме з частково реплікованою БД.
– *нереплікована* БД: зберігає кожний фрагмент БД на окремому вузлі. В цьому випадку дубльовані фрагменти БД відсутні.

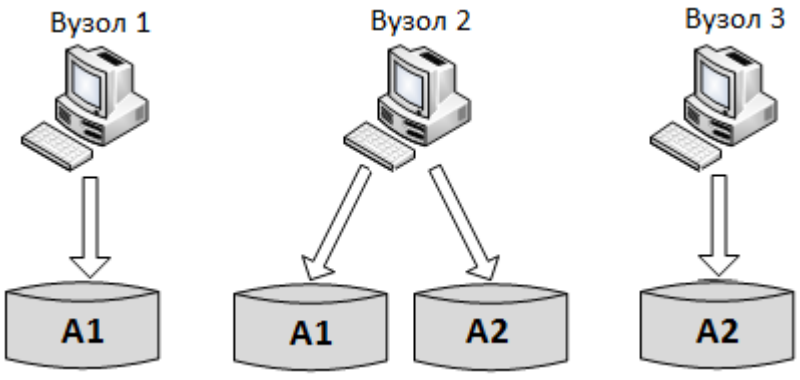

Рисунок 3.8 – Часткова реплікація даних

На реплікацію вливають декілька факторів:

– розмір БД;

– частота використання БД;
– витрати (ефективність, непродуктивні витрати програмного забезпечення та управління), пов'язані з синхронізацією транзакцій та їх частин при забезпеченні належної відмовостійкості, пов'язаної з реплікацією даних.

Якщо частота звернення до віддалених даних дуже висока, а база даних велика, то реплікація даних може зменшити витрати на обробку запитів. Інформація про реплікацію даних зберігається в каталозі розподілу даних (DDC), вміст якого ТР використовує, щоб прийняти рішення – до якої копії фрагмента БД необхідно забезпечити доступ. Реплікація даних дозволяє відновити загублені дані.

При реплікації даних потрібно дотримуватись *правила взаємної несуперечності*, згідно з яким всі копії фрагменту повинні бути ідентичні (визначення та склад даних у всіх копіях повинні бути однаковими). Для забезпечення несуперечності даних в репліках СУРБД повинна гарантувати своєчасне оновлення БД на всіх вузлах, де є репліки даних. Ці функції виконує спеціальний модуль СУРБД – сервер тиражування даних, який називають реплікатором (replicator).

Реплікатор повинен вирішувати чотири основні задачі:
– Що тиражувати?
– Де тиражувати?
– Коли тиражувати?
– Як вирішувати можливі конфлікти?

Розглянемо детальніше кожну із зазначених задач.

Що тиражувати?

Ключовим поняттям у процесі тиражування є поняття узгодженого розподіленого набору даних (consistent distributed data set - CDDS). Це набір даних в базі (або база повністю), ідентичність яких підтримується реплікатором у всіх вузлах, залучених до процесу тиражування.

CDDS може бути представлений такими конфігураціями даних:
– вся база даних;
– обрані об’єкти бази даних: таблиці або представлення;
– вертикальні фрагменти БД;
– горизонтальні фрагменти БД;
– поєднання наборів 2-4.

Набір даних буде вважатись CDDS лише у випадку дотримання таких мов:

1. Набір даних повинен розташовуватися в декількох базах даних в ідентичних копіях. Ідентичність тут розуміється як однаковість визначення та складу даних.

2. Дані з різних CDDS повинні бути взаємно ортогональні, тобто одні й ті ж дані не можуть входити до різних CDDS.

3. CDDS повинен мати властивість повноти, тобто включати всі ідентичні копії даних, що існують в розподіленій системі.

Узгодженість CDDS автоматично підтримується реплікатором і проявляється в тому, що:

– будь-яка зміна будь-якої копії CDDS автоматично поширюється на всі інші копії;
– усередині CDDS жодна копія даних не має переваги перед іншою копією – вони абсолютно рівноправні.
– всі об'єкти, що становлять кожну копію CDDS, мають однакові імена.

Принципово є те, що всі вузли є рівноправними, а тому можуть бути як джерелами, так і приймачами змін, хоча за необхідності всередині CDDS визначається пріоритет даних кожного вузла, який використовується при автоматичному вирішенні конфліктів в тиражованих даних.

Де тиражувати?

Ще одним важливим елементом тиражування є *шлях перенесення змін* (data propagation path - DPP), відповідно до якого реплікатор передає дані з кожної тиражованої бази даних в інші БД. Гнучкість тиражування в значній мірі забезпечується великим вибором способів передачі даних між вузлами розподіленої системи.

Розрізняють такі схеми тиражування даних, реалізовані реплікатором:

1. *«Центр-філії»*, зміни до БД філій переносяться в центральну БД, та/або навпаки.

2. *Рівноправне*, кілька БД поділяють загальний набір змінюваних і тиражованих даних.

3. *Каскадне*, зміни в одній БД переносяться в іншу БД, звідки у свою чергу в третю БД і т. д., ця схема дозволяє оптимізувати баланс завантаження серверів баз даних, розташованих на різних вузлах.

4. *Через шлюзи* зміни в базі даних можуть переноситися в БД іншої СКБД.

5. Різні комбінації всіх перерахованих вище схем.

В основі описаних схем лежать механізми, що регулюють взаємини між залученими в процес тиражування вузлами (з точки зору приймаючого вузла):

1. «Рівний-з-рівним» (peer-to-peer або full peer): всі зміни, виконані з CDDS на першому вузлі, потраплять на другий вузол і навпаки, виконується контроль можливих колізій.

2. Доступ з виявленням і розв'язанням конфліктів (protected read): зміни з першого вузла потрапляють на другий, проводиться контроль мо-

жливих конфліктів (наприклад, якщо джерел декілька); зміни CDDS на другому вузлі незаконні, ігноруються і на перший вузол не передаються.

3. Доступ з читанням без попередження конфліктів: те ж саме, що і 2, але конфлікти не виявляються і не вирішуються.

4. Доступ через шлюз: те ж саме, що і 3, але другий вузол містить дані, одержувані через шлюз до БД іншої СКБД, при цьому використовується мова запитів OpenSQL.

Коли тиражувати?

Елементарною зміною, яка викликає реакцію реплікатора, є транзакція. Тиражувати кожну транзакцію по одинці було б не завжди зручно. Тому, для досягнення максимальної гнучкості, реплікатор надає такі можливості:

– тиражування починається після завершення певного числа транзакцій, в тому числі і після кожної транзакції;
– тиражування відбувається через рівні проміжки або до певного моменту часу;
– процес тиражування контролюється вручну адміністратором системи або створеним користувачем монітором.

Як вирішувати можливі конфлікти?

Конфлікти, що виникають в деяких ситуаціях, наприклад, при зустрічному тиражуванні або при відновленні бази даних за допомогою реплікатора з реплікованої копії, можна віднести до розряду планованих проблем. Реплікатор за необхідності самостійно виявляє протиріччя в тиражованих даних і надає адміністратору можливість вирішення конфлікта (суперечливі дані обов'язково реєструються в журналі), або робить це автоматично.

Можливі такі варіанти:

– вирішення конфлікту на користь більш ранньої або більш пізньої зміни;
– вирішення конфлікту на користь найвищого пріоритету тиражованого запису.

Загалом при роботі з базою даних СКРБД повинна виконувати такі процеси:

– якщо БД фрагментована, то СКРБД для отримання доступу до відповідного фрагмента повинна розбивати запит на підзапити;
– якщо БД реплікована, то СКРБД повинна прийняти рішення, до якої копії необхідно забезпечити доступ. Операція читання (Read) обирає *найближчу* копію, яка придатна для даної транзакції. Операція

запису (Write) у відповідності з правилом взаємної несуперечності повинна вибирати і оновлювати всі копії даних;

– процесор транзакцій (TP) надсилає дані на обробку на кожний обраний процесор даних (DP);

– DP отримує дані, обробляє запит і відсилає дані назад на TP;

– TP об’єднує всі відповіді DP.

Технологія реплікації даних дозволяє зменшити загальні витрати на комунікації і виконання запитів, оскільки копії фрагментів підвищують рівень доступності даних та зменшують час відгуку. До основних переваг цієї технології можна віднести:

1. Дані завжди розташовані там, де вони обробляються. Це дозволяє істотно збільшити швидкість доступу до них.

2. Передаються лише операції, які змінюють дані, що в асинхронному режимі дозволяє значно зменшити трафік.

3. З боку вихідної БД для приймаючих БД реплікатор постає як процес, ініційований одним користувачем, тоді як у фізично розподіленому середовищі з кожним локальним сервером працюють всі користувачі розподіленої системи, конкуруючи за ресурси один з одним.

4. Жоден тривалий збій зв'язку неспроможний порушити передачу змін. Це пояснюється тим, що тиражування передбачає буферизацію потоку змін (транзакцій) і після відновлення зв'язку передача відновлюється з тих транзакцій, на яких тиражування було перервано.

Серед недоліків технології тиражування даних можна виділити неможливість цілковитого виключення конфліктів між двома версіями одного й того ж запису. Вони виникатимуть, коли внаслідок асинхронності передачі два користувача на різних вузлах виправлять один і той самий запис в той момент, коли зміни даних із першої бази даних ще не були перенесені на другу. Отже, під час проектування розподіленого середовища за допомогою технології тиражування даних слід передбачити конфліктні ситуації та запрограмувати реплікатор на будь-який варіант їх вирішення.

3.2.3 Розміщення даних

Розміщення даних (data allocation) – це процес прийняття рішення про місце зберігання даних. Виділяють чотири альтернативні стратегії розміщення даних.

Централізоване розміщення. Дана стратегія передбачає створення на одному з вузлів єдиної бази даних під управлінням СКБД, доступ до якої будуть мати всі користувачі мережі (ця стратегія відома під назвою «розподілена обробка»). В цьому випадку локалізація посилань мінімальна для всіх вузлів, за виключенням центрального, оскільки для отримання будь-якого доступу до даних необхідне встановлення мережевого з’єднання. Тому рівень витрат на передачу даних досить високий. Рівень

надійності та доступності в системі низький, оскільки відмова на центральному вузлі призведе до порушення роботи всієї системи.

Секційне (роздільне, фрагментоване) розміщення. База даних розбивається на декілька фрагментів, кожний з яких зберігається на одному з вузлів системи. Якщо кожний елемент даних буде розміщений на тому вузлі, на якому він найчастіше використовується, то рівень локалізації посилань буде високий. За відсутності реплікації вартість зберігання даних буде мінімальна, але при цьому рівень надійності та доступності даних в системі буде невисокий. Однак він вищий, ніж в попередньої стратегії, оскільки відмова на будь-якому з вузлів спричинить припинення доступу лише до тих даних, які на ньому зберігались. За правильного способу розміщення даних можна досягти високого рівня продуктивності та низького рівня витрат на передачу даних.

Розміщення з повною реплікацією. Згідно з цією стратегією на кожному з вузлів системи розміщується повна копія всієї бази даних. Локалізація посилань, надійність та доступність даних, рівень продуктивності системи будуть максимальними. Однак високими будуть вартість пристроїв зберігання інформації та витрати на передачу інформації про оновлення. Для подолання частини цих проблем в деяких випадках використовують *технологію знімків*. Під знімком розуміють копію бази даних в деякий момент часу. Ці копії оновлюються через деякий встановлений інтервал часу, наприклад один раз в годину або в тиждень, тому вони не завжди актуальні в поточний момент. Іноді в розподілених системах знімки використовуються для реалізації представлень, що дозволяє покращити час виконання в базі даних операцій з представленнями. (дет. 23.6 Конноли)

Розміщення з вибірковою реплікацією. Дана стратегія є комбінацією методів фрагментації, реплікації та централізації. Одні масиви даних розбиваються на фрагмента, що дозволяє досягти для них високого рівня локалізації посилань. В той же час, інші масиви даних, які не підлягають частим оновленням, але використовуються на багатьох вузлах, реплікуються. Всі інші дані зберігаються централізовано. Дана стратегія об’єднує переваги перших трьох стратегій та виключає їх недоліки. Завдяки своїй гнучкості ця стратегія використовується найчастіше.

В табл. 3.4 наведено зведені характеристики розглянутих стратегій розміщення даних.

Інформація про схеми фрагментації, реплікації та розміщення міститься *в глобальному системному каталозі*. Каталог може бути організований у вигляді розподіленої бази даних і тому також може підлягати фрагментації та розміщуватися на різних вузлах, повністю реплікуватися або розміщуватися централізовано.

Таблиця 3.4 – Порівняльна характеристика стратегій розміщення даних

	Локалізація посилань	**Надійність та доступність**	**Продуктивність**	**Вартість зберігання**	**Витрати на передачу даних**
Централізоване розміщення	Найнижча	Найнижча	Незадовільна	Найнижча	Найвищі
Секційне розміщення	Висока	Низька для окремих елементів; висока для системи в цілому	Задовільна	Найнижча	Низькі
Повна реплікація	Найвища	Найвища	Висока при виконанні операцій читання	Найвища	Високі при виконанні операцій оновлення; низькі при виконанні операцій читання
Часткова реплікація	Висока	Низька для окремих елементів; висока для системи	Задовільна	Середня	Низькі

3.2.4 Методологія проектування розподілених БД

Методологія проектування розподілених баз даних включає шість основних етапів.

1. Розробити проект для глобальних відносин.

2. Провести дослідження топології системи. Наприклад, визначити, чи повинно бути передбачено застосування бази даних в кожному підрозділі компанії, в кожному місті або регіоні. У першому варіанті може виявитися найбільш доцільною фрагментація відношень за номером підрозділу. А в останніх двох варіантах найбільш прийнятне рішення може передбачати фрагментацію відношень з урахуванням того, чи відносяться дані до певного міста чи регіону.

3. Проаналізувати найбільш важливі транзакції в системі і визначити, за яких умов може знадобитися горизонтальна або вертикальна фрагментація.

4. Прийняти рішення про те, які відношення не слід фрагментувати; копії таких відношень повинні бути розподілені по всіх вузлах. З глобальної ER-діаграми видалити відношення, які не повинні бути фрагментовані, а також зв’язки, в яких беруть участь виконувані на них транзакції.

5. Провести дослідження відношень, які знаходяться на стороні «один» зв’язку «один до багатьох», і вибрати найбільш прийнятну схему фрагментації для цих відношень з урахуванням топології системи. Відношення, які перебувають на стороні «багато» зв’язку «один до багатьох», можуть виявитися найбільш підходящими для похідної фрагментації.

6. За результатами виконання попереднього етапу визначити необхідність введення додаткової вертикальної або змішаної фрагментації (тобто визначити, чи є такі транзакції, для яких потрібен доступ до підмножини атрибутів відношення).

3.3 Управління паралельним доступом в розподіленому середовищі

В розподілених СКБД передбачена можливість одночасного виконання декількома користувачами різних операцій в базі даних. Якщо ці операції здійснюються безконтрольно, то дії користувачів можуть довільним чином впливати один на одного, внаслідок чого база даних може перейти в неузгоджений стан. Якщо виконуються складні запити, пов'язані з пошуком по багатьох файлах бази даних, користувач може почати операції пошуку при одному стані бази даних, а закінчити зовсім при іншому. Це може виникнути в результаті внесення змін у файли бази даних іншими користувачами. Наприклад, при паралельному внесенні змін в один і той самий запис один користувач може стерти зміни іншого користувача.

До основних проблем, які мають місце в розподілених СКБД, відносять:

– проблема втраченого оновлення,

– проблема залежності від проміжних результатів,

– проблема неузгодженості обробки,

– проблема узгодженості багатьох копій даних.

Перші три проблеми характерні і для централізованих СКБД. Остання проблема притаманна саме розподіленим СКБД і виникає в тих випадках, коли існує декілька копій одного елемента даних, розміщених у різних місцях. Очевидно, що для підтримки узгодженості глобальної бази даних при оновленні реплікованого елемента даних на одному з вузлів необхідно відобразити цю зміну і у всіх інших копіях даного елемента. Якщо оновлення не буде відображено у всіх копіях, база даних перейде в неузгоджений стан.

Щоб запобігти подібним явищам у розподіленій СКБД здійснюють управління паралельним доступом, яке засноване на підходах з використанням механізмів *блокування* і *тимчасових відміток.*

3.3.1 Протоколи блокування для управління паралельним виконанням у розподіленій базі даних

Для управління паралельним доступом використовують такі методи блокування: централізований протокол двофазного блокування, двофазне блокування з первинними копіями, розподілений протокол двофазного блокування та блокування більшості копій.

Централізований протокол двофазного блокування. При використанні цього протоколу вся інформація про блокування елементів даних у системі зберігається на єдиному вузлі. Тому у всій розподіленій СКБД існує тільки один *планувальник*, або *диспетчер блокувань*, здатний встановлювати і знімати блокування з елементів даних. При запуску глобальної транзакції на вузлі S_1 централізований протокол двофазного блокування працює таким чином.

1. Координатор транзакцій на вузлі S_1 поділяє глобальну транзакцію на кілька субтранзакцій, використовуючи інформацію, що зберігається в глобальному системному каталозі. Координатор відповідає за дотримання узгодженості бази даних. Якщо транзакція передбачає оновлення реплікованого елемента даних, координатор повинен забезпечити оновлення всіх існуючих копій цього елемента даних. Тому координатор повинен затребувати встановити виключні блокування на всіх копіях оновлюваного елемента даних, а потім звільнити ці блокування. Для читання оновлюваного елемента даних координатор може вибрати будь-яку з існуючих копій. Зазвичай зчитується локальна копія, якщо така існує.

2. Локальні диспетчери транзакцій, які приймають участь у виконанні глобальної транзакції, запитують і знімають блокування елементів даних під управлінням центрального диспетчера блокувань, керуючись при цьому звичайними правилами протоколу двофазного блокування.

3. Центральний диспетчер блокувань перевіряє допустимість вхідних запитів на блокування елементів даних з урахуванням поточного стану блокування цих елементів. Якщо блокування є допустимим, диспетчер блокувань направляє на вихідний вузол повідомлення про те, що необхідне блокування елемента даних надане. У протилежному випадку запит поміщається в чергу, де і знаходиться аж до того моменту, коли необхідне блокування може бути надане.

Варіантом цієї схеми є випадок, коли координатор транзакцій виконує всі запити на блокування від імені локальних диспетчерів транзакцій. У цьому випадку диспетчер блокувань взаємодіє тільки з координатором транзакцій, а не з окремими локальними диспетчерами транзакцій.

Перевага централізованого протоколу двофазного блокування полягає в тому, що його можна відносно просто реалізувати. Виявлення взаємоблокувань може виконуватися тими ж методами, що і в централізованих СКБД, оскільки вся інформація про блокування елементів знаходиться в розпорядженні єдиного диспетчера блокувань. Основний недолік цієї схеми пов'язаний з тим, що будь-яка централізація в розподіленій СКБД автоматично призводить до появи вузьких місць в системі і різко знижує рівень її надійності та стійкості. Оскільки всі запити на блокування елементів даних спрямовуються на єдиний центральний вузол, його швидкодію обмежують можливості всієї системи. Крім того, відмова цього вузла викликає порушення роботи всієї розподіленої системи, тому вона стає менш надійною. Однак цій схемі притаманний відносно невисокий рівень

витрат на передачу даних. Наприклад, глобальна операція оновлення за участю агентів (субтранзакцій) на *n* вузлах при наявності центрального диспетчера блокувань може зажадати відправки йому не менш *2n+3* повідомлень, у тому числі: один запит на блокування; одне повідомлення про надання блокування; *n* повідомлень з вимогою оновлення; *n* підтверджень про виконані оновлення; один запит на зняття блокування.

Двофазне блокування з первинними копіями. У цьому варіанті протоколу спроба подолати недоліки централізованого протоколу двофазного блокування виконується за рахунок розподілу функцій диспетчера блокувань по декільком вузлам. У даному випадку кожен локальний диспетчер відповідає за управління блокуванням деякого набору елементів даних. У процесі реплікації для кожного копійованого елемента даних одна з копій обирається в якості первинної копії (primary copy), а всі інші розглядаються як вторинні (slave copy). Вибір первинного вузла може здійснюватися за різними правилами, причому вузол, який обраний для управління блокуванням первинної копії даних, не обов'язково повинен містити саму цю копію.

Даний протокол є простим розширенням централізованого протоколу двофазного блокування. Основна відмінність полягає в тому, що при оновленні елемента даних координатор транзакцій повинен визначити, де знаходиться його первинна копія, і послати запит на блокування елемента відповідному диспетчеру блокувань. При оновленні елемента даних досить встановити виняткове блокування тільки його первинної копії. Після того як первинна копія буде оновлена, внесені зміни можуть бути поширені на всі вторинні копії. Це розповсюдження повинно бути виконано з максимально можливою швидкістю, щоб запобігти читання іншими транзакціями застарілих значень даних. Однак немає необхідності виконувати всі оновлення у вигляді однієї елементарної операції. Даний протокол гарантує актуальність значень тільки первинної копії даних.

Подібний підхід може використовуватися в тих випадках, коли дані підлягають вибірковій реплікації, їх оновлення відбувається відносно рідко, а вузли не потребують використання найновішої копії всіх елементів даних. Недоліками цього підходу є ускладнення методів виявлення взаємних блокувань у зв'язку з наявністю декількох диспетчерів блокувань, а також збереження в системі певною мірою централізації, оскільки запити на блокування первинної копії елемента можуть бути виконані лише на одному вузлі. Останній недолік може бути частково компенсований за рахунок застосування резервних вузлів, що містять копію інформації про блокування елементів. Даний протокол характеризується меншими витратами на передачу повідомлень і більш високим рівнем продуктивності, ніж централізований протокол двофазного блокування. Це досягається в основному за рахунок менш частого використання віддалених блокувань.

Механізм двофазного блокування з первинними копіями запропонований в розподіленій СКБД Ingres.

Розподілений протокол двофазного блокування. У цьому протоколі також робиться спроба подолати недоліки, властиві централізованому протоколу двофазного блокування, але вже за рахунок розміщення диспетчерів блокувань на кожному вузлі системи. У даному випадку кожен диспетчер блокувань відповідає за управління блокуванням даних, що знаходяться на його вузлі. Якщо дані не піддаються реплікації, цей протокол функціонує аналогічно протоколу двофазного блокування з первинними копіями. В протилежному випадку розподілений протокол двофазного блокування використовує особливий протокол управління реплікацією, що отримав назву «читання однієї копії та оновлення всіх копій» (Read-One-Write-All – ROWA). У цьому випадку для операцій читання може використовуватися будь-яка копія копійованого елемента, але перш ніж можна буде оновити значення елемента, повинні бути встановлені винятькові блокування на всіх копіях. У такій схемі управління блокуванням здійснюється децентралізованим способом, що дозволяє позбутися недоліків, властивих централізованому управлінню. Однак даному підходу притаманні свої недоліки, пов'язані з суттєвим ускладненням методів виявлення взаємних блокувань (через наявність багатьох диспетчерів блокувань) і зростанням витрат на передачу даних (у порівнянні з протоколом двофазного блокування з первинними копіями), які викликані необхідністю блокувати всі копії кожного оновлюваного елемента. У цьому протоколі виконання глобальної операції оновлення, що має агентів на n вузлах, вимагатиме передачі не менше $5n$ повідомлень, в тому числі:

– n повідомлень із запитами на блокування;

– n повідомлень з наданням блокування;

– n повідомлень з вимогою оновлення елемента;

– n повідомлень з підтвердженням виконаного оновлення;

– n повідомлень із запитами на зняття блокування.

Ця кількість повідомлень може бути скорочена до $4n$, якщо не передавати запити на зняття блокування, яке в цьому випадку буде виконуватися при обробці операцій остаточної фіксації розподіленої транзакції. Розподіленний протокол двофазного блокування реалізований в системі *System R**.

Блокування більшості копій. Цей протокол можна вважати розширенням розподіленого протоколу двохфазного блокування, в якому усувається необхідність блокування всіх копій реплікованого елемента даних перед його оновленням. У цьому випадку диспетчер блокувань також є на кожному з вузлів системи, де він управляє блокуваннями всіх даних, що розміщуються на цьому вузлі. Коли транзакції потрібно зчитати або записати елемент даних, копії якого є на n вузлах системи, вона повинна відправити запит на блокування цього елемента більш ніж на половину з усіх n вузлів, де є його копії. Транзакція не має права продовжувати своє виконання, поки не встановить блокування на більшості копій елемента даних. Якщо їй не вдасться це зробити за деякий встановлений проміжок

часу, вона скасовує свої запити та інформує всі вузли про скасування її виконання. Якщо більшість підтверджень буде отримано, всі вузли інформуються про те, що необхідний рівень блокування досягнутий. Розподілене блокування на більшості копій може бути встановлене одночасно для будь-якої кількості транзакцій, а виключне блокування на більшості копій може бути встановлене тільки для однієї транзакції.

У цьому випадку також усуваються недоліки, властиві централізованому підходу. Але даному протоколу властиві власні недоліки, які полягають в підвищеній складності алгоритму, ускладненні процедур виявлення взаємних блокувань, а також необхідності відправки не менше *[(n + 1) / 2]* повідомлень із запитами на встановлення блокування і *[(n + 1) / 2]* повідомлень із запитами на скасування блокування.

3.3.2 Усунення взаємних блокувань в розподіленому середовищі

Існують три основні методи виявлення взаємних блокувань в розподіленій СКБД: *централізований*, *ієрархічний* і *розподілений*.

Централізований метод виявлення взаємних блокувань. При централізованому виявленні взаємних блокувань один з вузлів системи призначається координатором виявлення взаємних блокувань (Deadlock Detection Coordinator – DDC). Вузол DDC відповідає за побудову та обробку глобального графа очікування. З певним інтервалом кожен диспетчер блокувань в системі направляє на адресу DDC свій локальний граф очікування. Вузол DDC виконує побудову глобального графа очікування і перевіряє його на наявність циклів. Якщо граф очікування містить один або кілька циклів, DDC повинен розірвати кожен цикл, вибравши ті транзакції, які підлягають скасуванню з виконанням відкату, а потім перезапуску. В обов'язки DDC входить інформування всіх вузлів, що приймають участь в обробці транзакцій, які відміняються, про те, що для останніх необхідно виконати відкат і перезапуск.

Щоб звести до мінімуму кількість даних, що пересилаються, кожен диспетчер блокування посилає на адресу DDC тільки відомості про зміни в локальному графі очікування, що відбулися з моменту попередньої відправки цих відомостей. Передані відомості включають лише інформацію про додавання або видалення ребер в локальному графі очікування. Недоліком централізованого підходу є те, що він знижує надійність всієї системи, оскільки відмова центрального вузла може викликати великі проблеми у функціонуванні всієї системи.

Ієрархічний метод виявлення взаємних блокувань. При ієрархічному методі виявлення взаємних блокувань вузли в мережі створюють деяку ієрархію. На рис. 3.9 зображена ієрархія з восьми вузлів, від S_1 до S_8.

Кожен з вузлів для виявлення наявності взаємних блокувань посилає свій локальний граф очікування на вузол, розташований в ієрархії на рівень вище його.

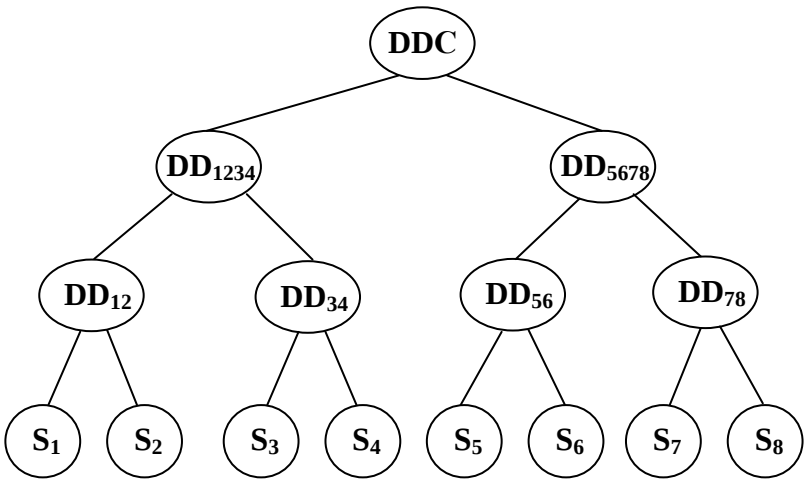

Рисунок 3.9 – Схема побудови ієрархічного механізму виявлення взаємних блокувань

Перший рівень ієрархії утворюють всі вісім вузлів, кожний з яких виконує локальний контроль наявності взаємних блокувань. Другий рівень утворюють вузли DD_{ij}, що забезпечують виявлення взаємних блокувань для сусідньої пари вузлів S_i і S_j. Третій рівень утворюють вузли, що виконують контроль взаємних блокувань на чотирьох сусідніх вузлах. Коренем дерева є глобальний детектор взаємних блокувань, здатний виявити взаємні блокування між будь-якими вузлами системи, наприклад між вузлами S_1 і S_6.

Ієрархічний підхід знижує залежність виявлення взаємних блокувань від центрального вузла, а також сприяє зниженню витрат на передачу даних. Проте його реалізація набагато складніша, особливо з урахуванням можливості відмов окремих вузлів і ліній зв'язку.

Розподілений метод виявлення взаємних блокувань. Найбільш широко використовується розподілений метод виявлення взаємних блокувань, запропонований Обермарком. У цьому методі для кожного вузла будується локальний графу очікування із додаванням зовнішнього вузла T_{ext}, що відображає наявність агента на віддаленому вузлі. Якщо локальний граф очікування містить цикл, що не включає вузол T_{ext}, то на вузлі і в розподіленій СКБД існує локальне взаємоблокування. Якщо цикл на локальному графі очікування включає вузол T_{ext}, то в системі може існувати глобальне взаємоблокування. Однак існування подібного циклу не обов'язково означає наявність глобального взаємоблокування, оскільки вузол T_{ext} може представляти кілька різних агентів. Для уточнення, чи дійсно має місце глобальне взаємоблокування, локальні графи повинні бути об'єднані у глобальний граф очікування.

Розподілений метод виявлення взаємоблокувань потенційно більш надійний, ніж ієрархічний та централізований методи, але оскільки жоден з вузлів не містить всієї інформації, необхідної для виявлення взаємобло-

кування, це може стати причиною високої інтенсивності обміну інформацією між вузлами.

3.3.3 Особливості використання часових відміток для управління паралельним виконанням у розподіленій базі даних

В розподілених базах даних використовуються такі ж протоколи з часовими відмітками, що й в централізованих базах даних. Завданням подібних протоколів є глобальне впорядкування транзакцій таким чином, що більш старі транзакції (ті, що мають меншу часову відмітку) у разі конфлікту отримують пріоритет. У розподіленому середовищі необхідно також виробляти унікальні значення часових відміток, причому як локально, так і глобально. Очевидно, що використання на кожному вузлі системних годинників або накопичувального лічильника подій вже не є прийнятним рішенням. Годинники на кожному вузлі можуть бути недостатньо синхронізовані, а при використанні лічильників подій ніщо не перешкоджає виробленню одних і тих же значень лічильника одночасно на різних вузлах.

Загальним підходом у розподілених СКБД є конкатенація локальної часової відмітки з унікальним ідентифікатором вузла у форматі <*локальна_відмітка, ідентифікатор_вузла*>. Значення ідентифікатора вузла має менший ваговий коефіцієнт, що гарантує впорядкування подій у відповідності з моментом їх виникнення і лише потім у відповідності з місцем їх появи. Щоб запобігти виробленню більш завантаженими вузлами великих значень часових відміток в порівнянні з недовантаженими вузлами, необхідно використовувати певний механізм синхронізації значень часових відміток між вузлами. Кожен вузол поміщає свою поточну часову позначку в повідомлення, що передаються на інші вузли. При отриманні повідомлення вузол-одержувач порівнює поточне значення його часової відмітки з отриманим і, якщо його поточна часова відмітка виявляється меншою, змінює її значення на деяке інше, що перевищує те значення часової відмітки, яке було одержане ним у повідомленні. Наприклад, якщо вузол 1 з поточною часовою відміткою <10, 1> передає повідомлення на вузол 2 з поточною часовою відміткою <15, 2>, то вузол 2 не змінює свою часову відмітку. І навпаки, якщо поточна часова відмітка на вузлі 2 дорівнює <5, 2>, то вузол 2 повинен змінити свою часову відмітку на <11, 2>.

3.4 Відновлення розподілених баз даних

3.4.1 Особливості відновлення розподілених баз даних

Для розподілених СКБД характерні чотири типи відмов, а саме:
- втрата повідомлення;
- відмова лінії зв’язку;

– аварійна зупинка одного з вузлів;

– поділ мережі на окремі підмережі.

Втрата повідомлень або порушення порядку слідування повідомлень можна усунути з використанням відповідного мережевого протоколу.

Для досягнення нерозривності глобальної транзакції розподілена СКБД повинна забезпечувати або повну фіксацію всіх субтранзакцій глобальної транзакції, або аварійне завершення всіх субтранзакцій. Якщо в розподіленій СКБД буде виявлено, що деякий вузол відмовив або став недоступний, в ній необхідно виконати такі дії:

1. Аварійно завершити всі транзакції, які мають відношення до даної відмови.
2. Відзначити вузол як відмовив, щоб запобігти будь-яким спробам його використання іншими вузлами.
3. Періодично перевіряти стан вузла, що відмовив, для відновлення його функціонування або чекати надходження від цього вузла повідомлення із зазначенням про відновлення його нормальної роботи.
4. При перезапуску вузла після відмови на ньому повинна виконуватися процедура відновлення, призначена для відкату будь-яких транзакцій, виконаних на момент відмови лише частково.
5. Після завершення процедури локального відновлення вузол, що відмовив,повинен оновити свою копію бази даних, щоб привести її у відповідність з іншою частиною системи.

Якщо має місце поділ мережі розподілена СКБД повинна гарантувати, що якщо агенти деякої глобальної транзакції виявилися активними в різних фрагментах, то вузол S_1 та інші вузли одного фрагмента повинні бути позбавлені можливості зафіксувати результати цієї глобальної транзакції, оскільки вузол S_2 та інші вузли іншого фрагмента можуть прийняти рішення виконати її відкат.

Процеси відновлення в розподілених СКБД ускладнюються тим, що дотримуватись властивості нерозривність потрібно і до локальних субтранзакцій, і до всієї глобальної транзакції в цілому. З цієї причини в процедури фіксації і відкату транзакцій необхідно внести такі зміни, які не дозволять глобальній транзакції зафіксувати або відмінити результати її виконання, поки всі її субтранзакції не будуть успішно зафіксовані або відмінені. Крім того, модифіковані протоколи повинні реагувати на ситуації відмови вузла або лінії зв'язку таким чином, щоб можна було гарантувати, що відмова одного з вузлів не вплине на обробку даних на іншому вузлі.

Протоколи, що забезпечують виконання останньої вимоги, називаються *неблокуючими*, найбільш поширеними з яких є протокол двофазної фіксації транзакцій (2PC) і неблокуючий протокол трифазної фіксації транзакцій (3PC).

Тут передбачається, що кожна глобальна транзакція пов'язана з деяким вузлом, що функціонує як *координатор* (або *диспетчер*) виконання

цієї транзакції. Зазвичай координатором є той вузол, на якому транзакція була ініційована. Вузли, на яких глобальна транзакція створила агенти, називаються *учасниками* (або *диспетчерами ресурсів*). Передбачається, що координатор транзакції має інформацію про ідентифікатори всіх учасників, а кожен учасник знає ідентифікатор координатора, але не зобов'язаний знати ідентифікатори інших учасників.

3.4.2 Протокол двофазної фіксації транзакцій (2PC)

Як випливає з назви даного протоколу, фіксація результатів транзакцій виконується в два етапи: голосування (фаза голосування) і прийняття рішення (фаза прийняття рішення). Основна ідея полягає в тому, що координатор повинен опитати всіх учасників, чи готові вони до фіксації транзакції. Якщо хоча б один з учасників зажадає відкату або не відповість на запит протягом встановленого тайм ауту, координатор вкаже всім учасникам на необхідність виконати відкат даної транзакції. Глобальне рішення повинно бути прийнято усіма учасниками. Якщо деякий учасник вимагає відкату транзакції, то він має право виконати його негайно. Фактично будь-який вузол має право виконати відкат своєї субтранзакції в будь-який час, аж до того моменту, поки він не відправить згоду на її фіксацію. Подібний тип відкату називається *одностороннім відкатом*.

Якщо учасник проголосував за фіксацію транзакції, то він повинен очікувати до тих пір, поки координатор не розішле повідомлення або про глобальну фіксацію, або про глобальний відкат цієї транзакції.

Даний протокол передбачає, що кожен вузол має свій власний локальний журнал і з його допомогою може надійно виконати відкат або фіксацію транзакції.

Двофазний протокол фіксації транзакцій включає етап очікування повідомлення від інших вузлів. Щоб уникнути небажаних блокувань процесів система використовує механізм контролю тайм-ауту. Для фіксації глобальної транзакції координатор виконує таку послідовність дій:

Eman1

1. Занести запис *begin_commit* в системний журнал і забезпечити його примусовий запис з буфера у вторинну пам'ять. Надіслати всім учасникам команду PREPARE. Очікувати надходження відповідей від усіх учасників протягом встановленого тайм-ауту.

Eman 2

2. Якщо учасник повертає повідомлення ABORT, помістити в системний журнал запис аварійного припинення та забезпечити його примусовий запис з буфера у вторинну пам'ять. Надіслати всім учасникам команду GLOBAL_ABORT. Очікувати відповіді від всіх учасників протягом встановленого тайм-ауту.

3. Якщо один з учасників повертає повідомлення READY_COMMIT, оновити список учасників, що надіслали свої відповіді. Якщо повідом-

лення про готовність фіксації надіслали всі учасники, помістити в системному журналі запис *commit* і забезпечити його примусовий запис з буфера у вторинну пам'ять. Надіслати всім учасникам команду GLOBAL_COMMIT. Очікувати відповіді від всіх учасників протягом встановленого тайм-ауту.

4. Після надходження всіх підтверджень про фіксацію транзакції помістити в системному журналі запис *end_transaction*. Якщо деякі вузли не надіслали підтвердження про фіксацію, знову направити на ці вузли повідомлення про прийняте глобальне рішення і діяти за цією схемою до отримання всіх необхідних підтверджень.

При фіксації транзакції учасник виконує наступну процедуру.

1. При отриманні учасником команди PREPARE він виконує одну з таких дії:

а) поміщає запис *ready_commit* в файл журналу і переносить з буфера у вторинну пам'ять всі записи, що відноситься до даної транзакції. Надсилає координатору повідомлення READY_COMMIT;

б) поміщає запис *abort* в файл журналу і переносить його з буфера у вторинну пам'ять. Відправляє координатору повідомлення ABORT. Виконує односторонній відкат транзакції.

2. Очікування відповіді координатора триває протягом встановленого тайм-аут.

3. Якщо учасник отримує команду GLOBAL_ABORT, то він поміщає запис *abort* у файл журналу і переносить його з буфера у вторинну пам'яті. Виконує відкат транзакції і по його завершенні надсилає координатору відповідне підтвердження.

Якщо учасник отримує команду GLOBAL_COMMIT, то він поміщає запис *commit* у файл журналу і переносить її з буфера у вторинну пам'ять. Потім виконується фіксація транзакції, звільнення всіх встановлених блокувань, після чого координатору надсилається відповідне підтвердження.

Якщо учаснику не вдалось отримати від координатора команди про проведення голосування, то по закінченню встановленого тайм-ауту він просто виконує відкат даної транзакції. Тому ще до надходження команди про проведення голосування будь-який учасник може прийняти рішення про відкат субтранзакції і виконати його локально.

Учасник повинен очікувати надходження від координатора команди GLOBAL_COMMIT або GLOBAL_ABORT. Якщо протягом встановленого тайм-аут він не отримає ніякої команди або координатор не отримає відповідь від учасника, то передбачається, що на відповідному вузлі відбулася відмова і в роботі запускається *протокол аварійного завершення* (протокол завершення). Протокол аварійного завершення виконують тільки функціонуючі вузли, а вузли, що відмовили, після перезапуску виконують *протокол відновлення*.

3.4.3 Неблокуючий протокол трифазної фіксації транзакцій (3PC)

Протокол двофазного блокування не може усунути можливість блокування, оскільки при його використанні виникають ситуації, коли деякий вузол залишається заблокованим. Наприклад, процес, який виявив закінчення тайм-ауту після відправки своєї згоди на фіксацію транзакції, але так і не отримав глобального підтвердження від координатора, залишається заблокованим, якщо може взаємодіяти тільки з вузлами, які також не мають відомостей про прийняте глобальне рішення. На практиці ймовірність блокування процесу досить мала, тому в більшості існуючих розподілених СКБД використовується саме протокол двофазної фіксації транзакцій. Проте був запропонований альтернативний неблокуючий протокол трифазної фіксації транзакцій.

Трифазна фіксація є неблокуючою в умовах відмови вузлів, за винятком випадку одночасної відмови всіх вузлів. Однак відмови ліній зв'язку можуть призвести до того, що на різних вузлах будуть прийняті різні рішення, що спричиняє спотворення нерозривності глобальної транзакції.

Для використання цього протоколу, слід дотримуватись таких умов.

– Поділ мережі є неприпустимим.
– Принаймні один вузол завжди повинен бути доступний.
– Може відмовити одночасно найбільше К вузлів мережі (тому такі системи називають К-стійкими).

Основна перевага протоколу трифазної фіксації полягає в усуненні періоду очікування в стані невизначеності, в яке переходять учасники з моменту підтвердження своєї згоди на фіксацію транзакції і до моменту отримання від координатора сповіщення про глобальну фіксацію або глобальний відкат. У трифазному протоколі фіксації між етапами голосування і прийняття глобального рішення вводиться третій етап попередньої фіксації. Після отримання результатів голосування від всіх учасників координатор розсилає глобальне повідомлення PRE-COMMIT. Учасник, який отримав глобальне повідомлення про попередню фіксацію, знає, що всі інші учасники проголосували за фіксацію результатів транзакції і що з часом сам цей учасник безумовно виконає фіксацію транзакції, якщо не відбудеться відмови. Кожен учасник підтверджує отримання повідомлення про попередню фіксацію. Після того як координатор отримає всі ці підтвердження, він розсилає команду глобальної фіксації транзакції. Якщо деякий учасник зажадав відкату транзакції, то обробка цієї ситуації виконується так, як в протоколі двофазної фіксації.

Як координатор, так і учасник як і раніше переходять на деякий час в стан очікування, однак головна особливість полягає в тому, що всі функціонуючі процеси отримують інформацію про глобальне рішення зафіксувати транзакцію за допомогою відправки повідомлення PRE-COMMIT ще до того, як перший процес виконає фіксацію результатів транзакції, що дозволяє учасникам діяти незалежно один від одного в разі відмови.

3.4.4 Протоколи відновлення

Дії, які виконуються при перезапуску системи після відмови, залежать від того, на якому етапі обробки глобальної транзакції знаходились координатор або учасник до моменту відмови.

Відмова координатора

Розглянемо три різні варіанти відмови вузла-координатора.

1. Відмова в стані INITIAL. Координатор ще не почав процедуру фіксації транзакції. В даному випадку відновлення полягає в запуску цієї процедури фіксації.

2. Відмова в стані WAITING. Координатор вже направив команди PREPARE вузлам учасникам, і, хоча отримані ще не всі відповіді, жодної пропозиції про відкат отримано не було. У цьому випадку відновлення полягає у повторному запуску процедури фіксації транзакції.

3. Відмова в стані DECIDED. Координатор вже направив учасникам транзакції вказівки про її глобальну фіксацію або глобальний відкат. Якщо після перезапуску координатор отримає всі необхідні підтвердження, завершення транзакції можна вважати успішним. В іншому випадку потрібно вдатися до використання протоколу аварійного завершення.

Відмова учасника

Метою застосування протоколу відновлення на вузлі-учаснику є отримання гарантій, що після перезапуску системи учасник виконає у відношенні транзакції ті ж дії, що і всі інші учасники її виконання, і ці дії можуть бути виконані незалежно (тобто без необхідності проведення консультацій з координатором або іншими учасниками). Розглянемо три різні варіанти відмови вузла-учасника.

1. Відмова в стані INITIAL. Учасник ще не встиг проголосувати за спосіб завершення транзакції. Тому в процесі відновлення він може виконати відкат транзакції в односторонньому порядку, оскільки без отримання відомостей від даного вузла-учасника координатор не може прийняти рішення про глобальну фіксацію цієї транзакції.

2. Відмова в стані PREPARED. Учасник вже направив відомості про своє рішення на адресу координатора. У цьому випадку відновлення полягає в застосуванні протоколу аварійного завершення.

3. Відмова в стані ABORTED / COMMITTED. Учасник вже завершив обробку транзакції. Тому після перезапуску подальших дій не потрібно.

3.4.5 Протоколи аварійного завершення

Протокол аварійного завершення виконується кожен раз, коли координатор або учасник не отримує очікуваного повідомлення до закінчення тайм-ауту. Дії залежать від того, хто не отримав повідомлення, координатор або учасник, і яке саме повідомлення не було отримано.

Координатор

В процесі виконання фіксації транзакції координатор може перебувати в одному з чотирьох станів: INITIAL, WAITING, DECIDED і COMPLETED. У подібних випадках робляться такі дії.

– Тайм-аут в стані WAITING. Координатор очікує надходження від усіх учасники повідомлень про рішення, яке вони прийняли щодо фіксації або відкату даної транзакції. У подібній ситуації координатор не може зафіксувати транзакцію, оскільки він не отримав усі необхідні підтвердження, тому він організовує дії по глобальному відкату даної транзакції.

– Тайм-аут в стані DECIDED. Координатор очікує надходження від усіх учасники повідомлень про успішний відкат або фіксацію даної транзакції. У подібній ситуації координатор просто повторно розсилає відомості про прийняте глобальне рішення на всі вузли, які не надіслали очікуваного підтвердження.

Учасник

Найпростіший протокол аварійного завершення для учасника полягає у збереженні процесу на стороні учасника заблокованим до тих пір, поки з'єднання з координатором не буде поновлено. Після цього учасник зможе отримати інформацію про прийняте глобальне рішення і відновити обробку транзакції відопвідним чином. Однак з міркувань підвищення продуктивності системи на стороні учасника можуть бути вжиті й інші дії.

В процесі виконання фіксації транзакції учасник може перебувати в одному з чотирьох станів: INITIAL, PREPARED, ABORTED і COMMITTED.

Але учасник може завершити роботу по тайм-ауту тільки в перших двох станах, як описано нижче.

– Тайм-аут в стані INITIAL. Учасник очікує від координатора надходження команди PREPARE. Її відсутність може свідчити про те, що вузол координатора відмовив, коли процес фіксації транзакції знаходився в стані INITIAL. У цьому випадку учасник може виконати відкат транзакції в односторонньому порядку. При надходженні згодом команди PREPARE він зможе або її ігнорувати (в результаті чого координатор організовує відкат глобальної транзакції по тайм-ауту), або відправити координатору повідомлення ABORT.

– Тайм-аут в стані PREPARED. Учасник очікує надходження від координатора вказівок про глобальну фіксації або глобальний відкат даних транзакції. Учасник вже сповістив координатора про своє рішення зафіксувати транзакцію, тому не має права змінити своє рішення і виконати її відкат. Але він також не може продовжити роботу і зафіксувати транзакцію, оскільки глобальне рішення може виявитися вимогою її відкату. До отримання додаткової інформації учасник виявляється заблокованим. Але учасник може звернутися до кожного з решти учасників транзакції, щоб отримати від одного з них відомості про прийняте глобальне рішення. Цей варіант відомий як *кооперативний протокол аварійного завершення*. Найпростіший спосіб повідомити учасникам про те, хто ще бере участь у

виконанні транзакції, полягає в приєднанні до команди про проведення голосування список вузли-учасники.

Хоча кооперативний протокол аварійного завершення знижує ймовірність виникнення блокування, ця ситуація як і раніше залишається можливою, і для заблокованого процесу не залишається іншого виходу, крім повторних спроб зняти блокування до отримання інформації про усунення відмови. Якщо відмова сталася тільки на вузлі координатора (всі інші учасники можуть встановити це в результаті виконання протоколу аварійного завершення), вони зможуть вибрати нового координатора і таким чином зняти блокування.

Контрольні питання

1. Що називають розподіленою базою даних? Яка принципова відмінність розподіленої бази даних від централізованої?
2. Які основні функції повинна виконувати система керування розподіленою базою даних?
3. Чим відрізняється розподілена база даних від середовища розподіленої обробки даних?
4. Назвіть принципи проектування розподіленої бази даних.
5. Які основні задачі необхідно вирішити при проектуванні розподіленої бази даних?
6. У чому полягає сутність механізму реплікації?
7. Які види фрагментації Ви знаєте?
8. Охарактеризуйте стратегії розміщення даних.
9. Які протоколи блокування використовуються для управління паралельним виконанням у розподіленій базі даних?
10. У чому полягає особливість використання часових відміток для управління паралельним виконанням у розподіленій базі даних?
11. Які механізми усунення взаємоблокувань використовують у розподіленому середовищі?
12. У чому полягає особливість відновлення розподіленої бази даних?
13. Яку проблему протоколу двофазної фіксації транзакцій дозволяє усунути трифазний протокол фіксації транзакції?
14. Опишіть процес відновлення розподіленої бази даних.
15. В яких випадках використовують протоколи аварійного завершення? Що називають кооперативним протоколом аварійного завершення?

ЛІТЕРАТУРА:

1. Гарсиа-Молина Г. Системы баз данных. Полный курс / Г. Гарсиа-Молина, Дж. Ульман, Дж. Уидом. - М.: «Вильямс», 2003. – 1088 с.
2. Грабер М. Введение в SQL / Мартин Грабер. – М.: Лори, 2010. – 227 с.
3. Дейт К. Дж. Введение в системы баз данных, 8-е издание / К. Дж. Дейт. – М.: «Вильямс», 2005. – 1328 с.
4. Коннолли Т. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. 3-е издание / Т. Коннолли, К. Бегг. – М.: «Вильямс», 2003. – 1440 с.
5. Пасічник В. В. Організація баз даних та знань / В.В. Пасічник, В.А. Резніченко. – К.: Видавнича група BHV, 2006. – 384 с.
6. Роб П. Системы баз данных: проектирование, реализация и управление / П. Роб, К. Коронел. – СПб.: БХВ-Петербург, 2004. – 1040 с.
7. Романюк О.Н. Організація баз даних та знань / О.Н. Романюк, Т.О. Савчук. – Вінниця: ВДТУ, 2001.
8. Ульман Л. MySQL / Лари Ульман. – СПб.: Питер, 2004. – 352 с.
9. Хомоненко А.Д. Базы данных: Учебник для высших учебных заведений / А.Д. Хомоненко, В.М. Цыганков, М.Г.Мальцев. – СПб.: КОРОНА принт, 2004. – 736 с.