Паралельні транзакції у середовищі MsSQL

Транзакція – це послідовність операторів SQL, які використовуються для об'єднання операцій читання і запису таким чином, аби система гарантувала узгодженість даних.

Обробка транзакцій

Запити та інші команди мови управління даними групуються в транзакції (transactions) – процеси, які мають виконуватись *атомарним чином* (atomically) та *ізольовано* (in isolation) одна від одної. Зазвичай кожний окремий запит або операція зміни даних є самостійною транзакцією. Транзакція повинна мати властивість *стійкості* (durability). Це означає, що результат кожної завершеної транзакції має бути зафіксований навіть у таких ситуаціях, коли після завершення транзакції система з певних причин виходить з ладу.

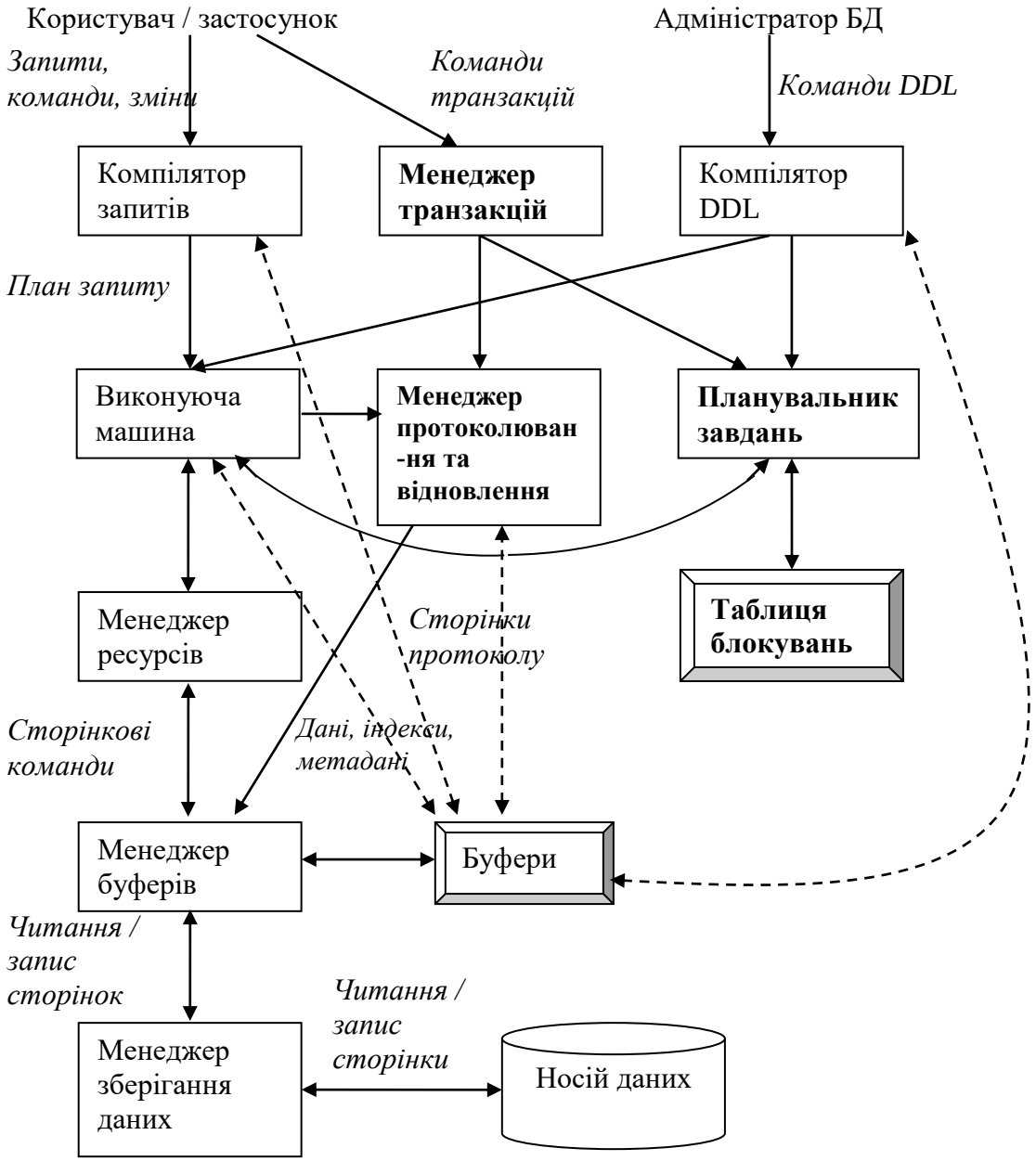


Рисунок 1 - Компоненти СУБД. Виділено те, що умовно можна віднести до менеджера транзакцій

На рис. 1 менеджер (процесор) транзакцій (transaction processor) має 2 основні компоненти:

- *Планувальник завдань* (sheduler) або *менеджер паралельних завдань* (concurrency control manager), відповідальний за забезпечення атомарності та ізольованості транзакцій.
- *Менеджер протоколювання та відновлення* (logging and recovery manager), який гарантує стійкість транзакцій.

Менеджер транзакцій (transaction manager) сприймає від застосунку *команди транзакцій* (transaction commands), які свідчать про початок і завершення транзакції (зазвичай 3 команди – почати, завершити записом на диск, відкатити). Функції *процесора транзакцій* (transaction processor):

1. *Протоколювання* (logging). Для дотримання вимоги стійкості транзакцій кожна зміна фіксується у спеціальних дискових файлах. *Менеджер протоколювання* (logging manager) керується одною з кількох стратегій, покликаних виключити шкідливі наслідки системних збоїв під час виконання транзакції. *Менеджер відновлення* (recovery manager) у таких ситуаціях здатний зчитати протокол змін і привести базу в певний коректний стан. Інформація протоколу спочатку зберігається в буферах; затим менеджер протоколювання в певні моменти часу взаємодіє з менеджером буферів, контролюючи, чи дійсно зміст буферів збережено на диску (де дані надійно захищені в разі краху системи).

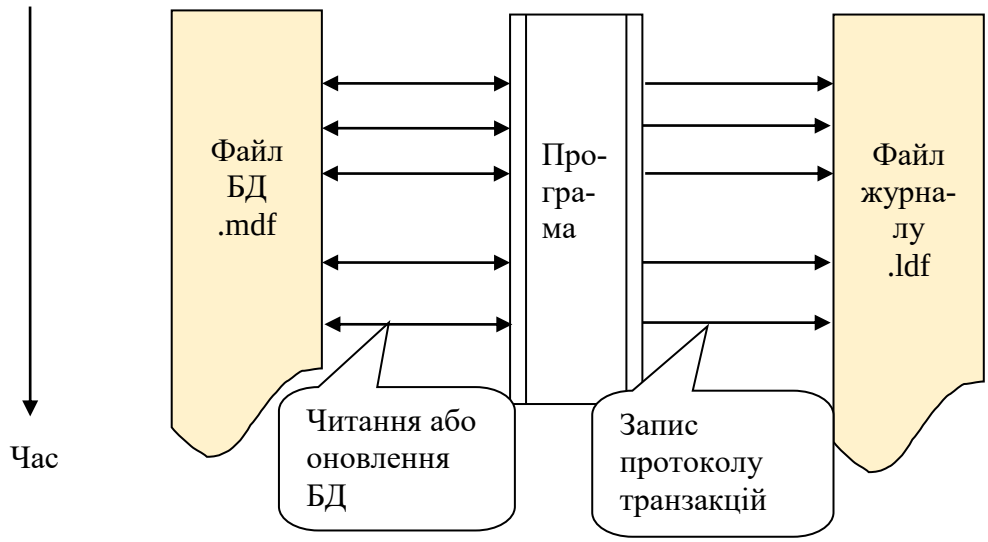


Рисунок 2 – Протоколювання

2. *Управління паралельними завданнями* (concurrency control). Транзакції повинні виконуватись в повній ізоляції одна від одної. Але насправді одночасно можуть діяти кілька процесів-транзакцій. *Планувальник завдань* (sheduler), або *менеджер паралельних завдань* (concurrency control manager), повинен забезпечити такий режим роботи системи, щоб результат окремих операцій численних транзакцій, які перетинаються в часі, виявився таким, мовби транзакції ініціювались, протікали та повністю завершувались у суворій послідовності, не „перетинаючись” одна з одною. Типовий планувальник завдань для цього встановлює *блокування* (lock) на відповідні фрагменти змісту БД. Блокування перешкоджають одночасному звертанню кількох транзакцій до порції даних способами, які погано узгоджені один з одним. Ознаки блокувань зазвичай зберігаються в *таблиці блокувань* (lock table) в ОП (рис. 1). Планувальник завдань впливає на виконання запитів та інших операцій, забороняючи виконуючій машині звертатися до заблокованих порцій даних. Зрозуміло, що наслідком цього є падіння продуктивності СУБД.

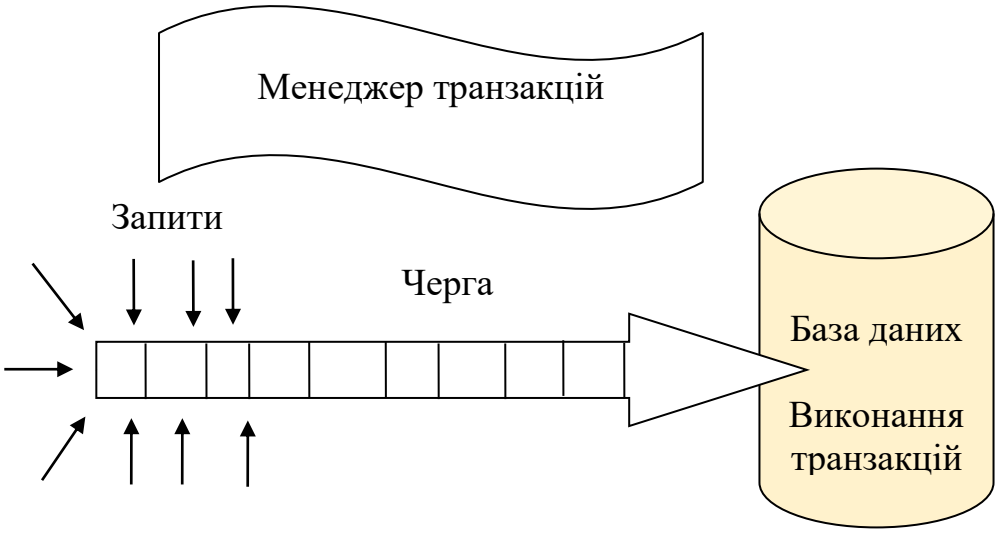


Рисунок 3 – Управління паралельними запитами шляхом ізоляції

3. *Розв'язання взаємоблокувань* (deadlock resolution). Оскільки транзакції змагаються за ресурси, які можуть бути заблоковані планувальником завдань, можливе виникнення такого стану, коли жодна транзакція не може виконати роботу, оскільки їй необхідний ресурс, заблокований іншою транзакцією. Менеджер транзакцій має право втручатися і переривати (*відкочувати* – rollback) одну або кілька транзакцій, аби дозволити іншим продовжити роботу.

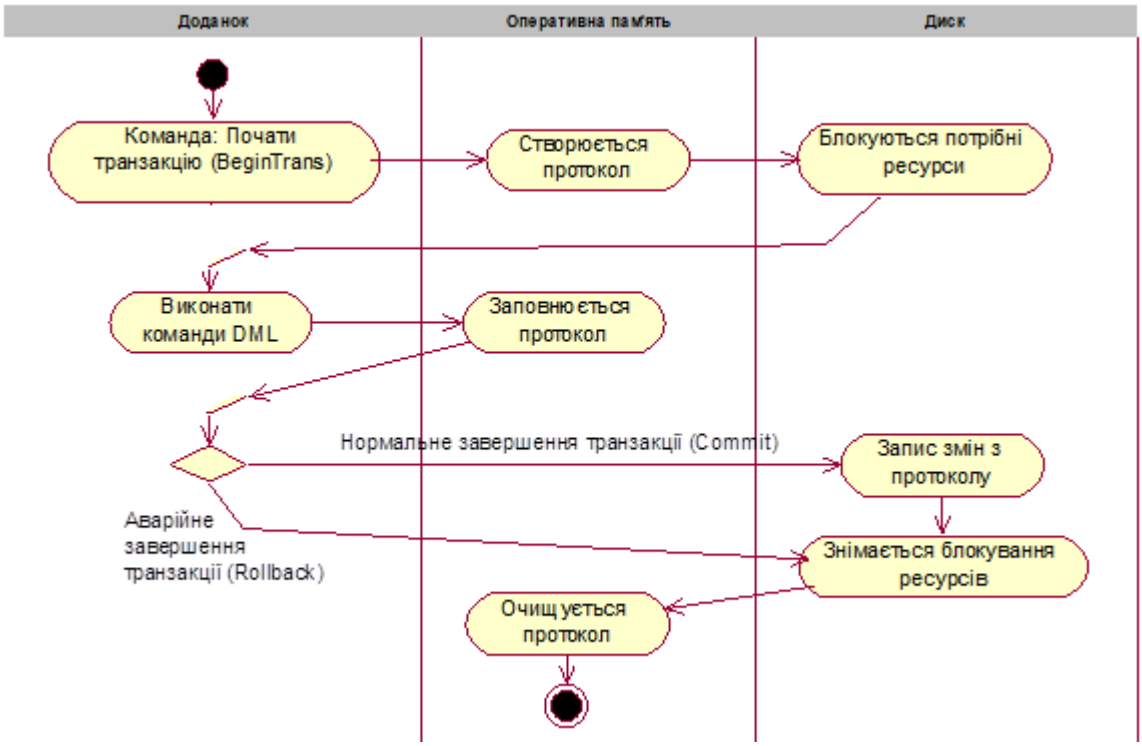


Рисунок 4 – Діаграма діяльності по виконанню транзакції

Неявна транзакція: її ініціює будь-який одиночний оператор INSERT, UPDATE або DELETE.
Явна транзакція: зазвичай це група операторів SQL, де початок і кінець цієї групи відмічається операторами початку та завершення або відміни.

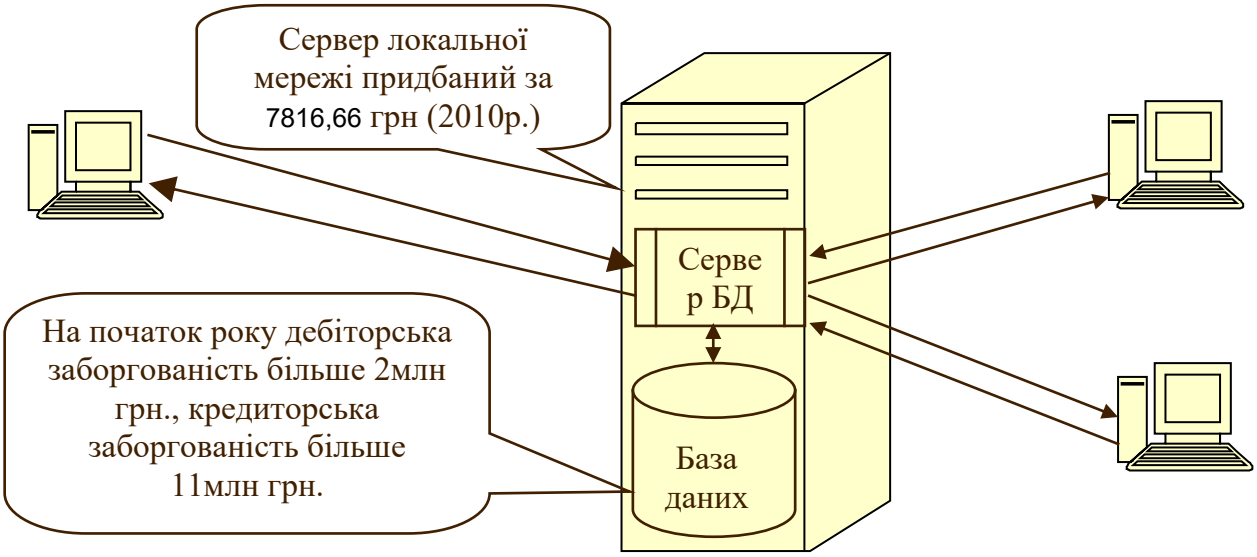
ACID - властивості транзакцій

Кажуть, що правильно реалізовані транзакції задовольняють умовам ACID, тобто:

- *A* є „atomicity” (*атомарність*) – транзакція виконується за принципом „усе або нічого”, тобто має бути або виконана вся або не виконана зовсім.
- *C* є „consistency” (*узгодженість*). Усі БД вводять ті чи інші *обмеження* (constraints), обумовлені змістом даних: особливостями елементів даних та їхньою узгодженістю (як-от: бухгалтерська проводка виконується і по рахунку дебету, і по рахунку кредиту). Транзакції мають зберігати узгодженість даних. Якщо транзакція починається, коли БД знаходиться в узгодженому стані, і протікає при відсутності інших транзакцій і системних збоїв, по завершенню транзакції стан БД також має бути узгодженим.
- *I* є „isolation” (*ізольованість*) – процес іде так, немовби інших транзакцій у цей час не існує.
- *D* є „durability” (*стійкість*) – результат виконання завершеної транзакції не повинен бути втрачений за жодних умов.

Архітектура журналу транзакцій

Вартість сервера і вартість даних



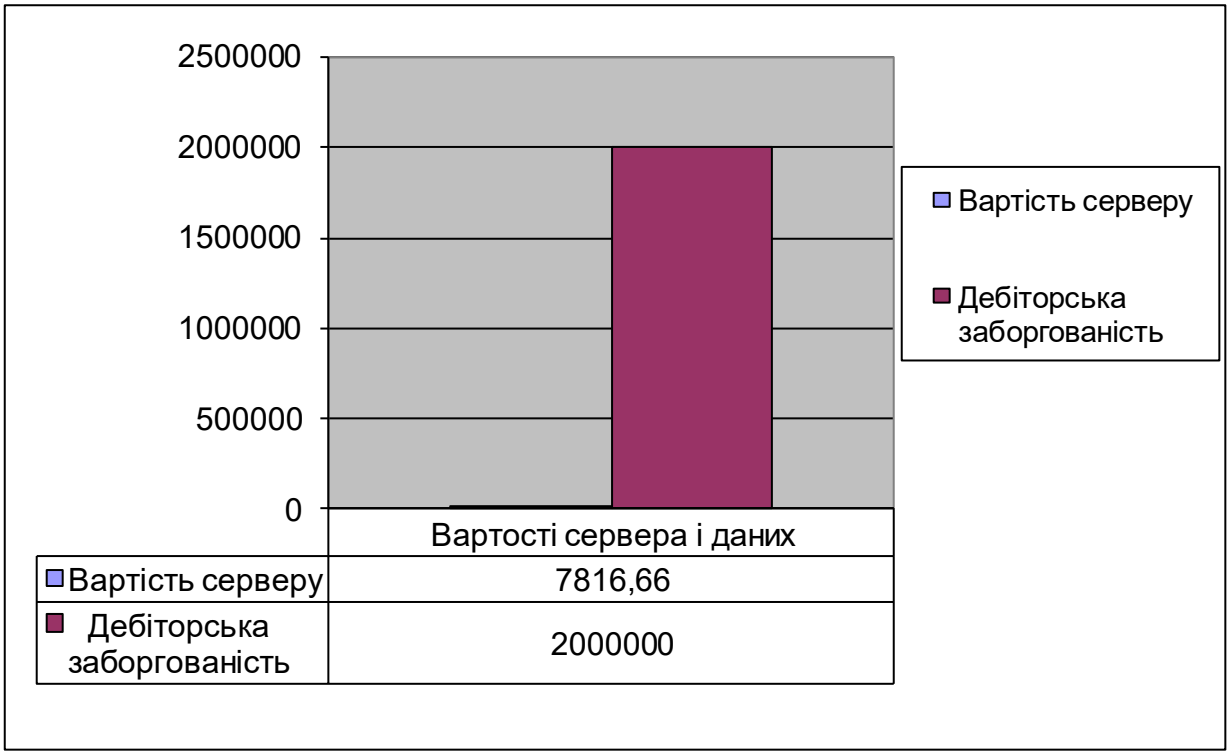


Рисунок 5 – Вартість сервера і дебіторська заборгованість підприємства

В разі втрати підприємством даних по дебіторській заборгованості буде складно в суді довести її наявність.

Призначення журналу транзакцій

Журнал транзакцій накопичує історичну інформацію про зміни в БД за відносно тривалий період (від кількох днів до кількох місяців).

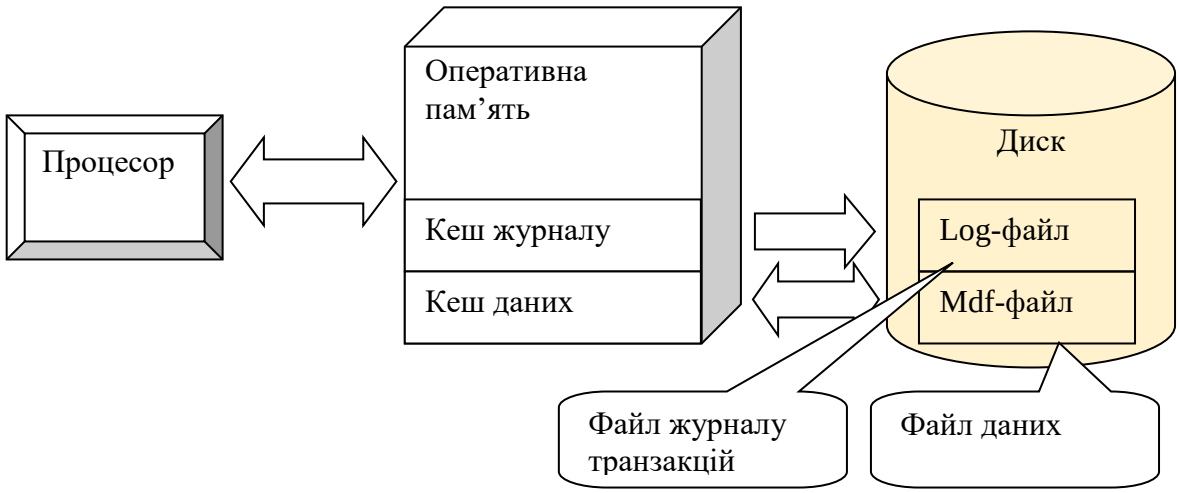


Рисунок 6 – Взаємодія файлів БД з оперативною пам'яттю

Відновлення окремих транзакцій. Якщо застосунок виконує ROLLBACK (відкат) транзакції або SQL Server виявляє помилку (наприклад, втрата зв'язку з клієнтом), записи журналу транзакцій використовуються для відкоту змін, зроблених під час транзакції.

Відновлення БД після раптової зупинки сервера.

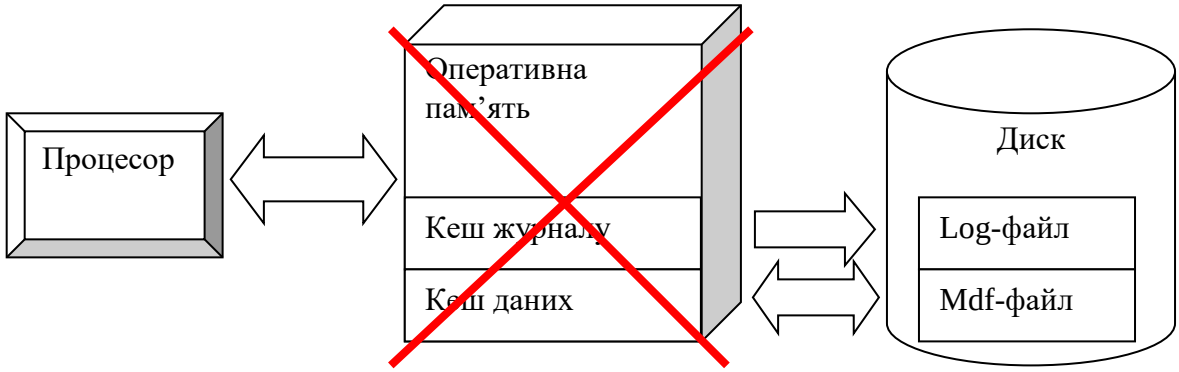


Рисунок 7 – Раптова зупинка сервера БД

В разі фізичної аварії на сервері, де працює СУБД SQL Server (як-от вимкнення живлення) у адміністратора є і журнал транзакцій, і остання версія файлів даних БД (можливо, в некоректному стані). А саме, якщо аварія трапилась під час скидання транзакції на диск, окремі зміни кеш-буферу БД можуть бути не записані у файли даних, у кеш-буфері БД з’являться зміни незавершених транзакцій. Після запуску SQL Server починає відновлення всіх баз даних по журналу. При цьому:

- Якщо транзакція відмічена в журналі як завершена, виконується повтор всіх модифікацій, зареєстрованих в журналі, але не записаних у файли даних.
- Якщо транзакція відмічена в журналі як незавершена, виконується її відкат, тобто внесені транзакцією зміни відкидаються, у файли даних нічого не записується.

Відновлення БД після втрати БД.

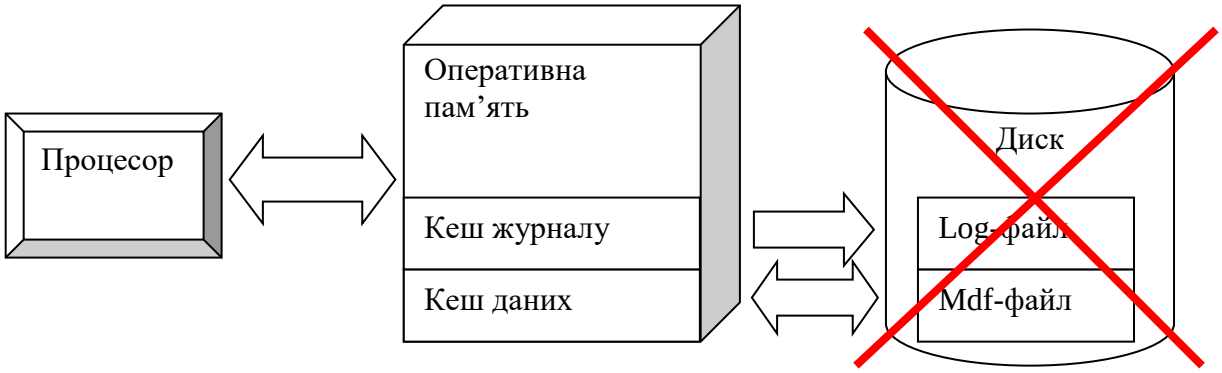


Рисунок 8 – Втрата БД з-за аварії жорсткого диску

Після втрати БД, що можливо при відмові жорсткого диску в разі відсутності на сервері RAID-масиву (а також і в разі його наявності), БД все ж удається повернути в стан, в якому вона знаходилась на момент збою. Спочатку слід відновити останню резервну копію БД, а потім — послідовність резервних копій журналів транзакцій аж до моменту збою. При відновленні кожної копії журналу SQL Server виконує „накат” (roll forward) транзакцій - повторює всі транзакції, заново виконуючи всі зареєстровані в журналі модіфікації.
Це робиться командою:

```
RESTORE LOG <Ім’я БД>
   FROM <пристрій рез.копії, м.б.шлях та ім’я файлу>
   WITH RECOVERY
```

Для такого відновлення треба:

- Постійно робити резервні копії журналу транзакцій, частіше ніж резервні копії БД,

- Зберігати резервні копії файлу даних і файлу журналу в надійному місці, не на диску сервера, а краще в іншому приміщенні!

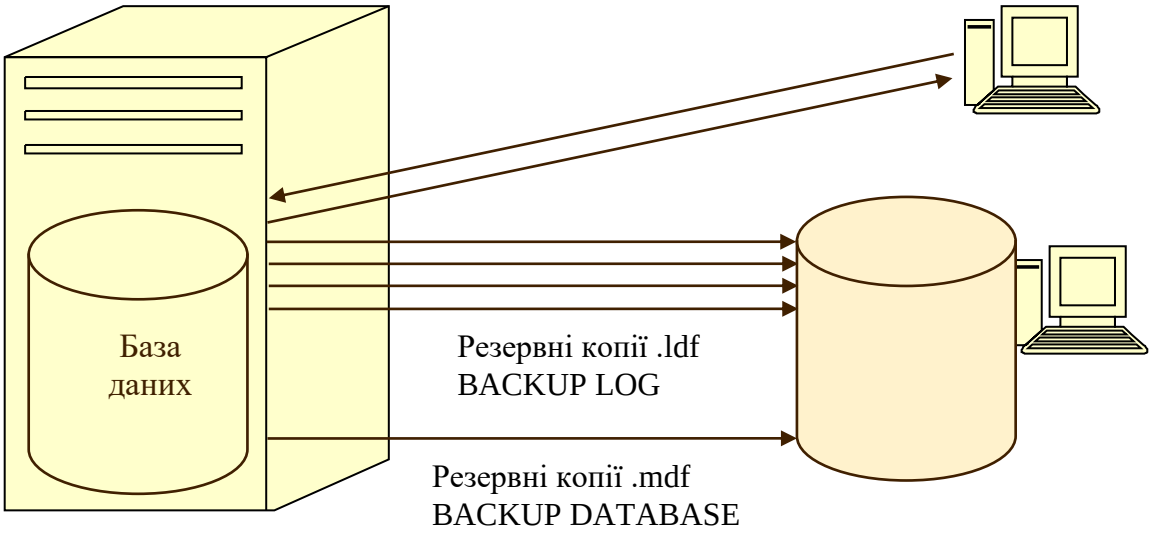


Рисунок 9 – Організація резервного копіювання для запобігання втраті БД

Випереджаюча реєстрація транзакцій

Випереджаюча реєстрація транзакцій гарантує, що модифікації файлів даних БД будуть записані на диск не раніше, ніж відповідні їм записи журналу.

Під час виконання транзакції виконується модифікація копії сторінки в кеш-буфері. Модифікації не записуються на диск до генерації в БД контрольної точки або до скидання модифікації на диск для розміщення в буфері нової сторінки. Сторінка даних, модифікована в кеші, але ще не записана на диск, називається зміненою або «брудною» сторінкою.

Якби сторінка даних скидалась на диск раніше відповідних записів журналу, внаслідок аварії під час запису таку зміну неможливо було б відмінити. Оскільки записи журналу завжди скидаються раніше відповідних сторінок даних, такий журнал називається журналом з випереджаючим записом.

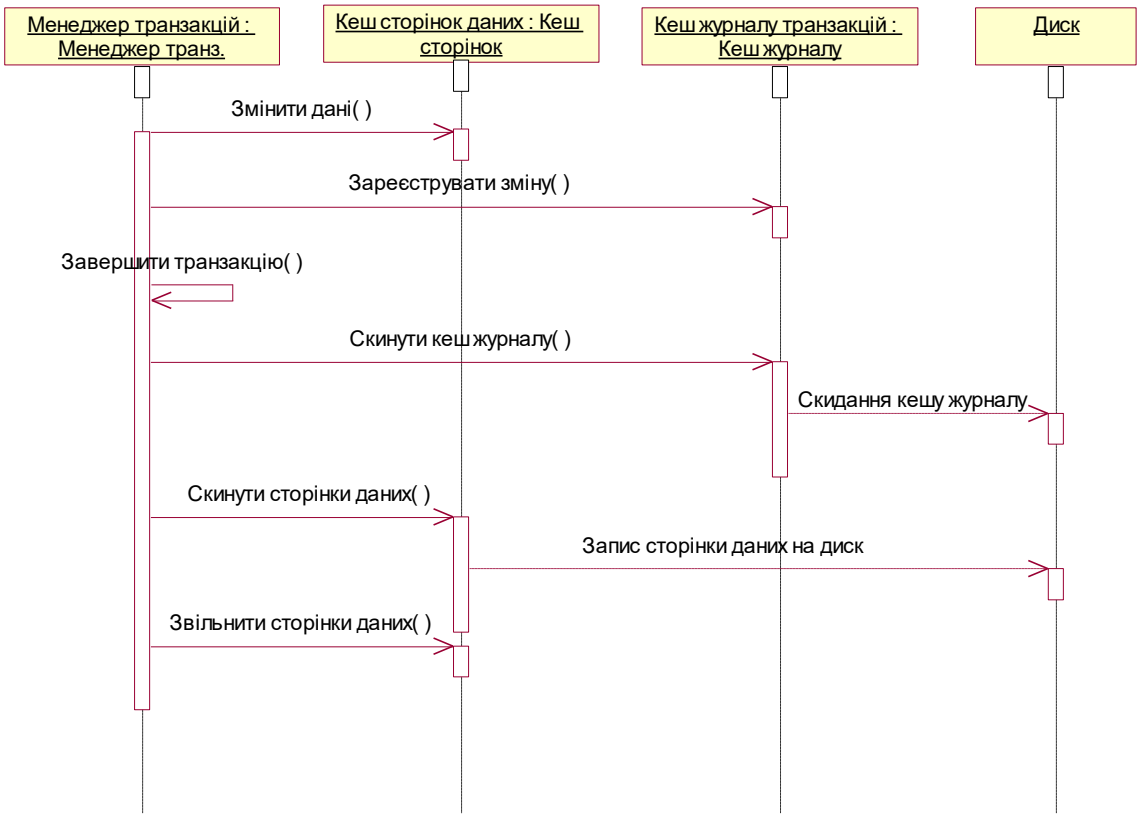


Рисунок 10 – Випереджаюча реєстрація транзакцій

Контрольні точки

У контрольних точках виконується скидання змінених сторінок даних і журналу з кеш-буфера поточної БД на диск – у .log і.mdf -файли. Контрольні точки генеруються при:

- виконанні оператора CHECKPOINT;
- зміні параметрів БД оператором ALTER DATABASE;
- зупинці екземпляра SQL Server в результаті виконання оператора SHUTDOWN або зупинці екземпляра механізму БД, що працює як сервіс, за допомогою SQL Server Service Manager;
- виконання автоматичних контрольних точок, що періодично генеруються екземпляром SQL Server в кожній БД для зменшення часу її відновлення.

SQL Server постійно генерує автоматичні контрольні точки. Інтервал часу між ними не постійний і визначається числом записів журналу. В найпростішому випадку на контрольній точці система робить наступне:

- Призупиняє приймання запитів на активізацію нових транзакцій;
- Чекає, поки всі діючі транзакції не виконають фіксацію або переривання і не збережуть у протоколі відповідні записи - <COMMIT> або <ABORT>
- “Скидає» протокол на диск ;
- Вносить у протокол запис <CKPT> («контрольна точка») і виконує його повторне «скидання»;

- Відновлює приймання транзакцій.

Усікання журналу транзакцій

Активна частина журналу транзакцій містить записи про транзакції, які не повністю скинуті на диск. Активна частина журналу необхідна для відновлення БД.

Записи журналу про завершені та скинуті на диск транзакції є „старими” і не потрібні для відновлення БД.

Якщо не видаляти записи в журналі транзакцій, журнал через деякий час заповнить усе вільне місце на диску. Процес скорочення розміру журналу шляхом видалення старих записів називається усіканням журналу. Активна частина журналу ніколи не усікається. Усікається стара частина, починаючи з найстаріших записів, до досягнення заданого розміру журналу.

Усікання журналу виконується при завершенні кожного оператора BACKUP LOG, а також під час обробки кожної контрольної точки, якщо використовується *проста модель відновлення БД*. Усікання в контрольних точках забезпечує відновлення БД лише після аварії типу зникнення електроживлення. Для відновлення БД після її втрати треба мати ланцюжок резервних копій журналу після останньої резервної копії БД (тобто зберігати резервні копії журналу на іншому диску).

Структура журналу транзакцій

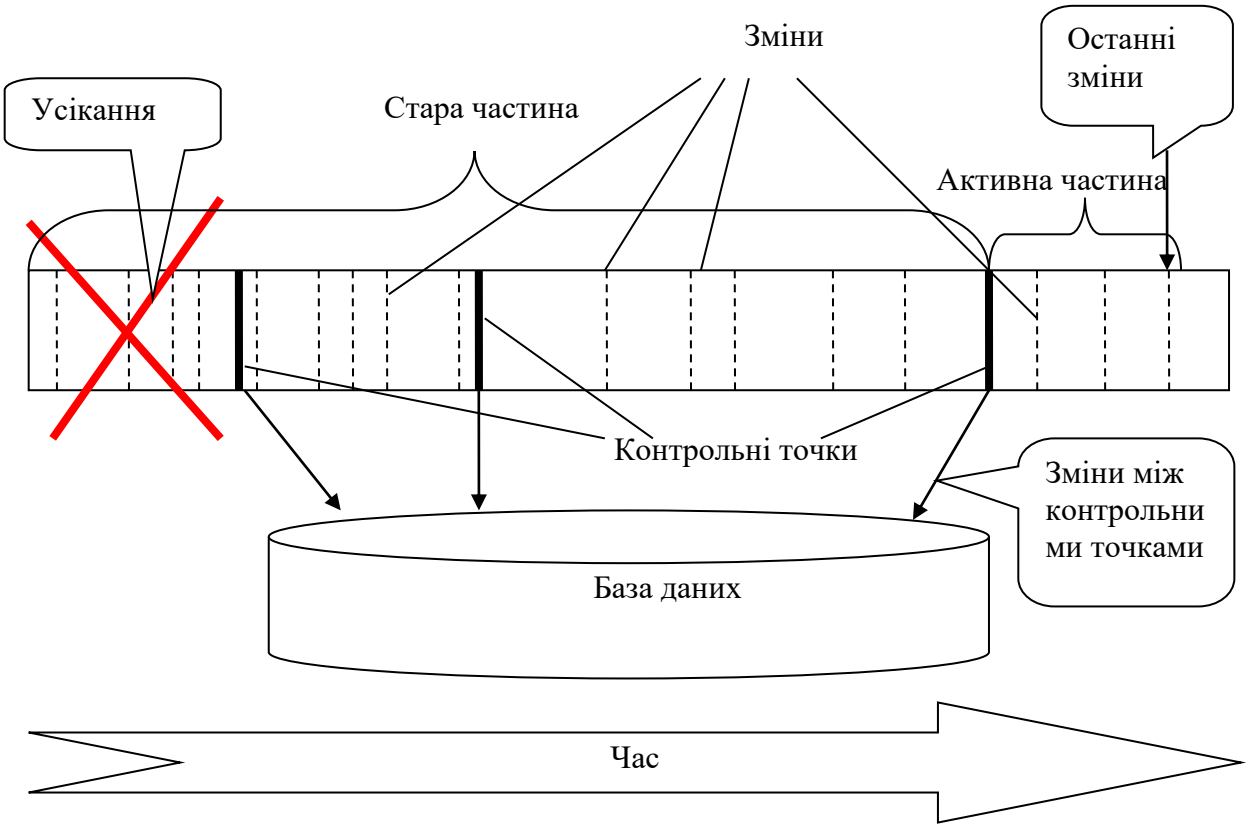


Рисунок 11 – Структура журналу транзакцій

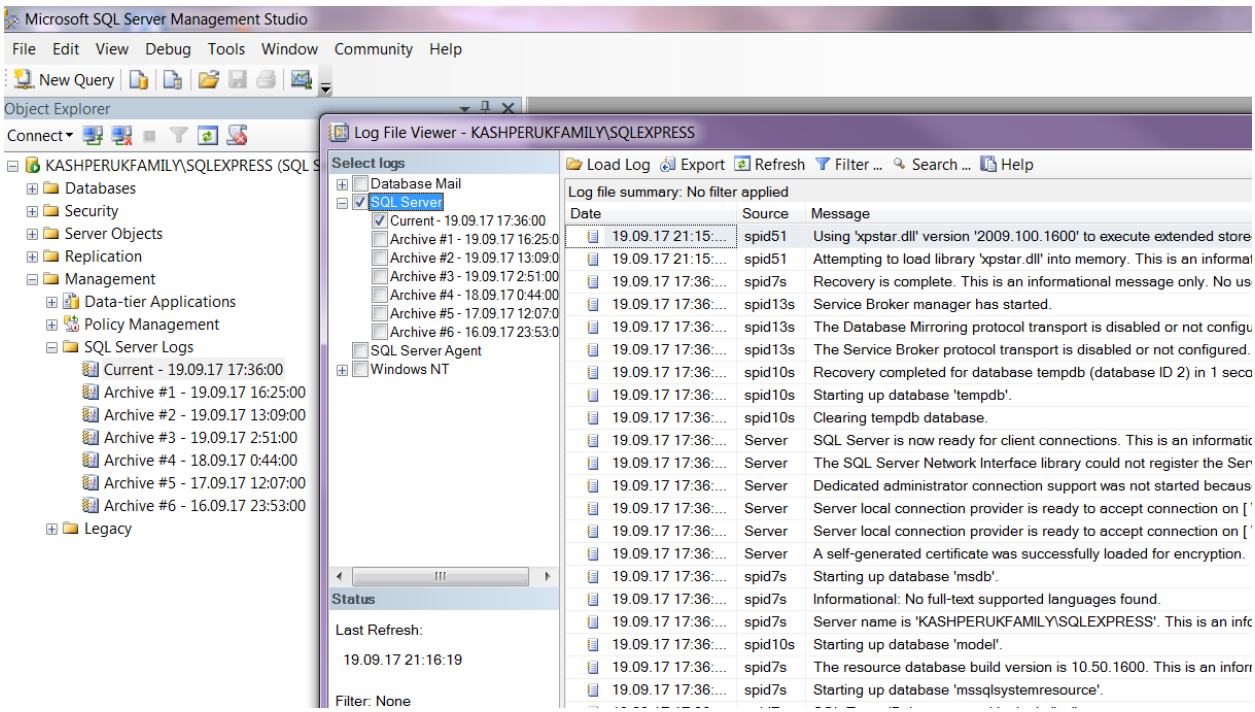


Рисунок 12 – Проглядання журналу транзакцій в SQL Server Management Studio (2008 R2)

Проблеми паралелізму

При одночасному доступі кількох (мільйонів) користувачів до одних і тих же даних в разі відсутності або невірного використання механізму транзакцій можуть виникати проблеми паралелізму.

Загублені оновлення

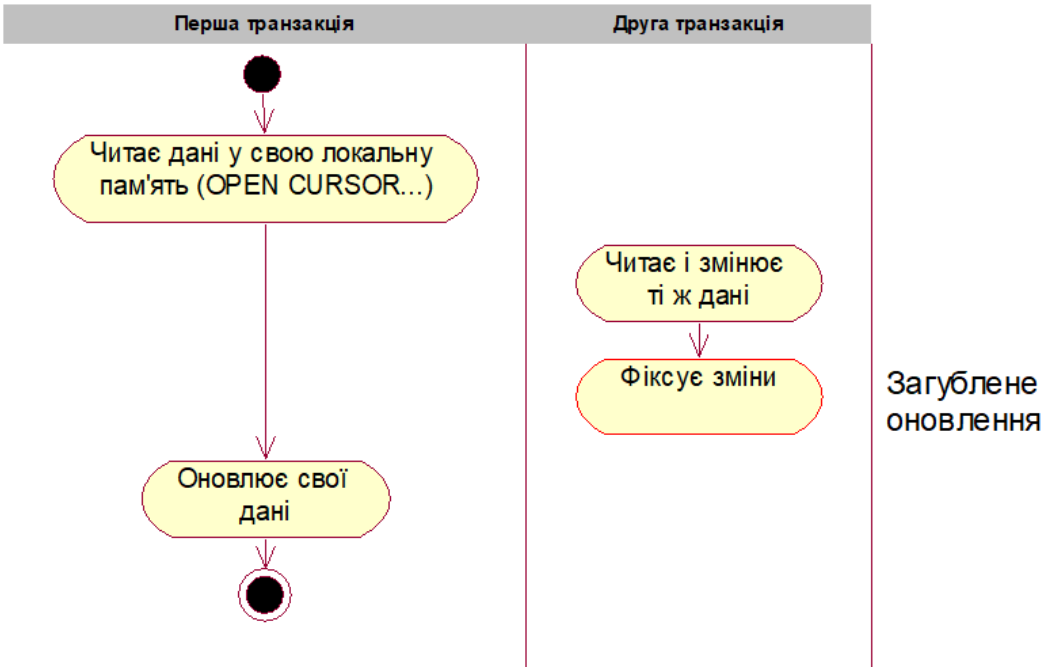


Рисунок 13 – Утворення загубленого оновлення

„Брудне" читання

Якщо до даних, щойно змінених, але не зафіксованих однією транзакцією, отримує доступ інша транзакція, має місце „брудне" читання (dirty read). Ці дані логічно ще не існують або ніколи не будуть існувати, якщо транзакція буде відмінена.

Приклад: в базі університету відбулась зміна прізвища студентки під час видачі звіту по поточній успішності. Можлива поява студентки в різних частинах звіту під різними прізвищами, якщо невірно встановити параметри транзакцій.

Неповторюване читання

В межах однієї транзакції два запити, виконані послідовно, повертають різні результати, оскільки в період між запитами інша транзакція змінила дані. Порушується принцип ізоляції.

Фантомне читання

Це особливий випадок неповторюваного читання. В одній транзакції 2 запити до одної таблиці використовують одне речення WHERE, але запит, що був виконаний останнім, повертає більше рядків, ніж перший (додались phantom rows).

Рівні ізоляції транзакцій

Усі названі проблеми пов'язані з порушенням ізоляції транзакцій. Згідно стандарту SQL-99 програміст може визначити 4 рівні ізоляції транзакцій:

Таблиця 1. Рівні ізоляції транзакцій

1	READ COMMITTED	Читаємо лише зафіксовані дані. Якщо дані заблоковані іншою транзакцією, чекаємо. Запобігає „брудному" читанню.
2	READ UNCOMMITTED	Припускаємо читання „брудних" рядків. Це дозволяє підвищити продуктивність. Для уникнення неповторюваного читання варто використовувати цей рівень для таблиць, які підлягають лише читанню.
3	REPEATABLE READ	Захищає від оновлення і видалення прочитаних рядків. Прочитані дані (але не додані дані) блокуються і утримуються упродовж транзакції, що не дозволяє іншим транзакціям змінювати прочитані дані. Існують ризики падіння продуктивності та взаємних блокувань. Захищає від неповторюваного читання, але не від фантомного.
4	SERIALIZABLE	Запобігає усім проблемам паралелизму, в тому числі фантомному читанню. Для запобігання додаванню даних у прочитану таблицю можливе її блокування. Ще вищі ризики падіння продуктивності та взаємних блокувань.
На додачу: Рівні ізоляції, які існують у SQL Server з версії 2005		
5	SNAPSHOT	Миттєвий знімок. Версії рядків на момент початку транзакції зберігаються в системній базі TempDb в оперативній пам'яті. Таким чином, транзакція виконується в „альтернативній реальності". Завдяки цьому загальні блокування не встановлюються на операції читання.
6	READ COMMITTED SNAPSHOT	Це не є окремий рівень ізоляції. Якщо встановити параметр бази READ_COMMITTED_SNAPSHOT в ON, рівень блокування READ COMMITTED буде вести себе як SNAPSHOT.

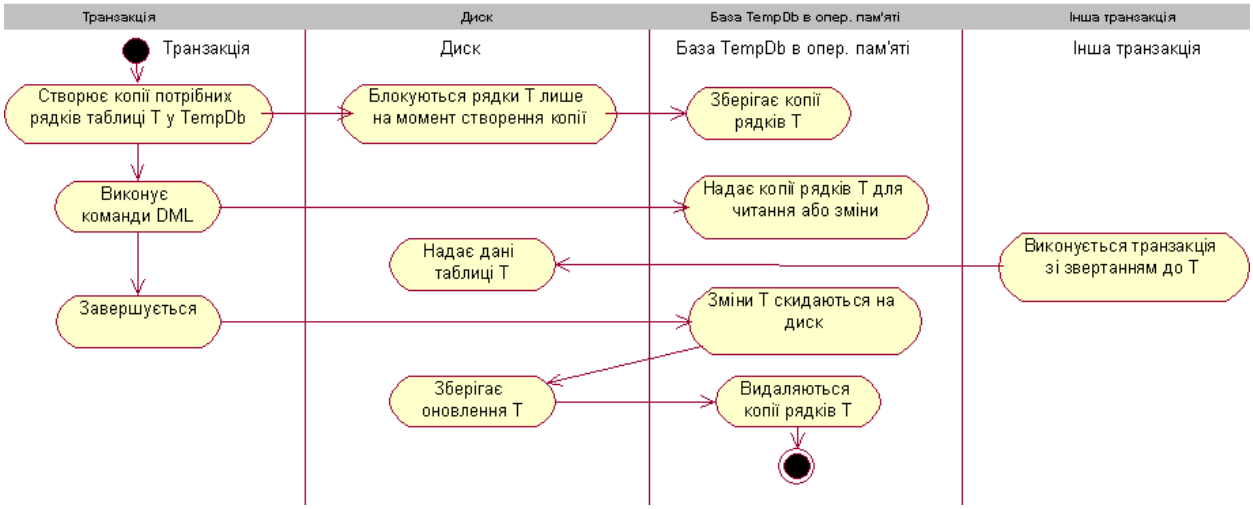


Рисунок 14 – Виконання транзакції з рівнем ізоляції SNAPSHOT

При зачиненні транзакції з рівнем SNAPSHOT можуть бути додаткові конфлікти оновлення, якщо змінені рядки видалені або змінені іншою транзакцією. Тому цей рівень ізоляції доцільно використовувати, якщо дані переважно лише читаються.

Таблиця 2. Які проблеми паралелізму існують та не існують при відповідних рівнях ізоляції

	Загублені оновлення	„Брудне" читання	Неповторюване читання	Фантомне читання
READ UNCOMMITTED	+	+	+	+
READ COMMITTED	-	-	+	+
REPEATABLE READ	-	-	-	+
SERIALIZABLE	-	-	-	-
SNAPSHOT	-	-	-	-

Програміст може задати рівень ізоляції транзакцій для поточного з'єднання з БД:
SET TRANSACTION ISOLATION LEVEL
{ READ COMMITTED
| READ UNCOMMITTED
| REPEATABLE READ
| SERIALIZABLE
}
Оскільки високий рівень ізоляції тягне падіння продуктивності СУБД, адміністратор або програміст мають в різних ситуаціях обирати доцільний рівень ізоляції.

Вказівка блокування на рівні оператора SQL і таблиці

Для операторів SELECT, INSERT, UPDATE і DELETE дозволяється задавати ряд вказівок блокування (locking hints) на рівні таблиці. Вказівки блокування на рівні таблиці служать для тонкого налаштування типів блокувань для певного об'єкту, як-от таблиці. Вказівки блокування підміняють поточний рівень ізоляції транзакцій сеансу.

Таблиця 3. Деякі вказівки блокування на рівні операторів SQL

Вказівка	*Призначення*
NOLOCK	Забороняє генерацію блокувань, що розділяються, і не враховує монопольні блокування. Можна читати непідтверджені транзакції і набори сторінок, для яких виконаний відкат під час операції читання («брудне» читання).
ROWLOCK	Використовує блокування на рівні рядків замість грубіших

	блокувань рівня сторінки і таблиці
PAGLOCK	В операторі, де використовується одиночна таблиця, може бути використане блокування сторінки
READCOMMITTED	Семантика блокування аналогічна рівню ізоляції READ COMMITTED. SQL Server працює на цьому рівні ізоляції за замовчанням
TABLOCK	Використовує блокування таблиць замість тонших блокувань рівня рядків і сторінок. SQL Server утримує це блокування до завершення оператора.
TABLOCKX	Використовує монопольне блокування таблиці. Це блокування забороняє іншим транзакціям читання і оновлення таблиці і утримується до завершення оператора або транзакції
XLOCK	Монопольне блокування, яке утримується до завершення транзакції для всіх даних, що обробляються оператором. Це блокування задають за допомогою вказівок PAGLOCK або TABLOCK, в цьому випадку монопольне блокування встановлюється на відповідному рівні

Приклад: встановлений рівень ізоляції транзакцій SERIALIZABLE, а також використовується вказівка блокування на рівні таблиці з оператором SELECT

```
USE Pubs
GO
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
GO
SELECT Au_lname FROM Authors WITH (NOLOCK)
GO
```

(Приклад взято з [8], також він є в Books Online, стаття Locking Hints)

Ескалація блокувань

Якщо, наприклад, у стрічкову форму читається невелика кількість рядків таблиці, по замовчанню діє вказівка блокування ROWLOCK. Якщо кількість прочитаних рядків велика, виконується блокування TABLOCK. І тоді може виявитись, що користувачі не можуть працювати.

Приклад 8.1. Папка первинних документів у системі „Регістри”
В наведеній папці первинних документів (ПД) містяться видаткові документи з АЗС за три роки. Користувач хоче бачити їх усі в одній папці. Усього ПД у таблиці 55 тис., з них у папці виводиться 2637.

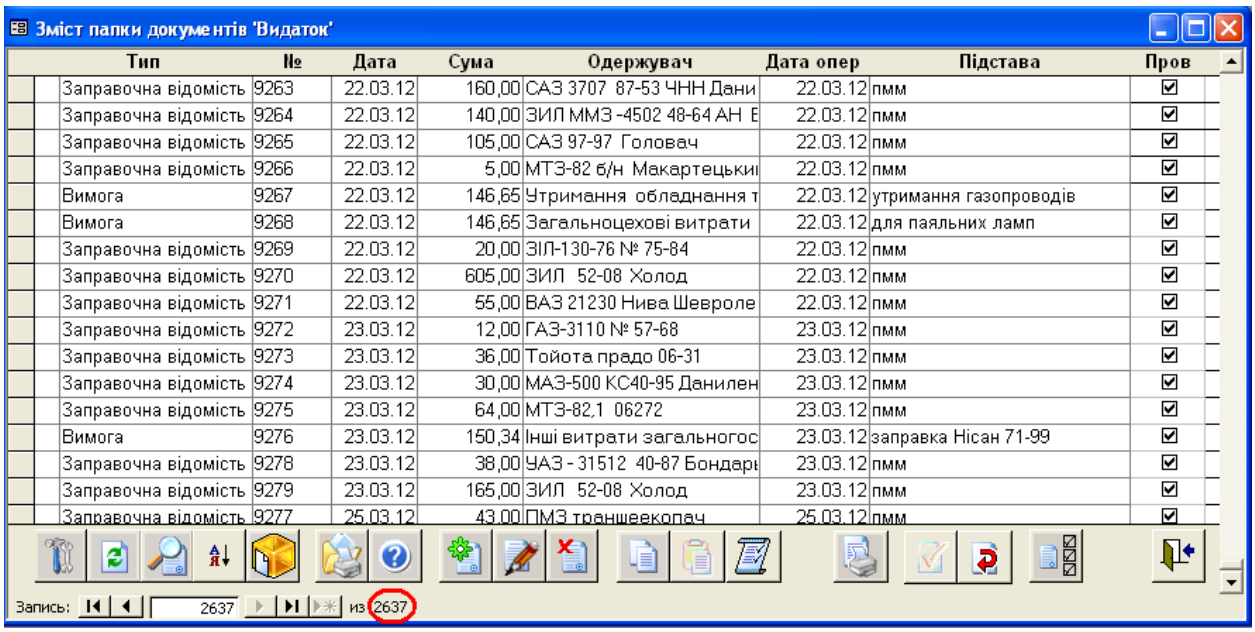


Рисунок 15 – Папка ПД видатку з АЗС

До певного часу при відчиненні папки блокування виконувалось по замовчанню. Але з певного моменту упродовж двох днів стало неможливо користувачам відчиняти папки: доводилось довго чекати. Програмісту довелось при визначенні джерела папки дати вказівку NOLOCK.

```
CodeDb.QueryDefs(sQdfName).connect = RegDatabase.TableDefs("kAnal").connect
CodeDb.QueryDefs(sQdfName).SQL = "SELECT * FROM qryCardPd WITH (NOLOCK)" _
    & " WHERE " & Me.sFilter & " ORDER BY " & sOrderBy
'Без блокувань відчиняємо qryCardPd, _
без цього папка блокує навіть проведення ПД- ВП 13.11.05
DoCmd.OpenForm sFormName                'Джерело форми визначимо при відчиненні
```

Рисунок 16 – Фрагмент коду визначення джерела для папки ПД

Взаємоблокування

Якщо одна транзакція заблокувала ресурс, необхідний другій транзакції, то друга транзакція очікує зняття блокування першою транзакцією. По замовчанню у транзакцій SQL Server нема тайм-ауту (якщо лиш не встановлено параметр LOCK_TIMEOUT). В цьому випадку друга транзакція просто блокується, але взаємоблокування не виникає.

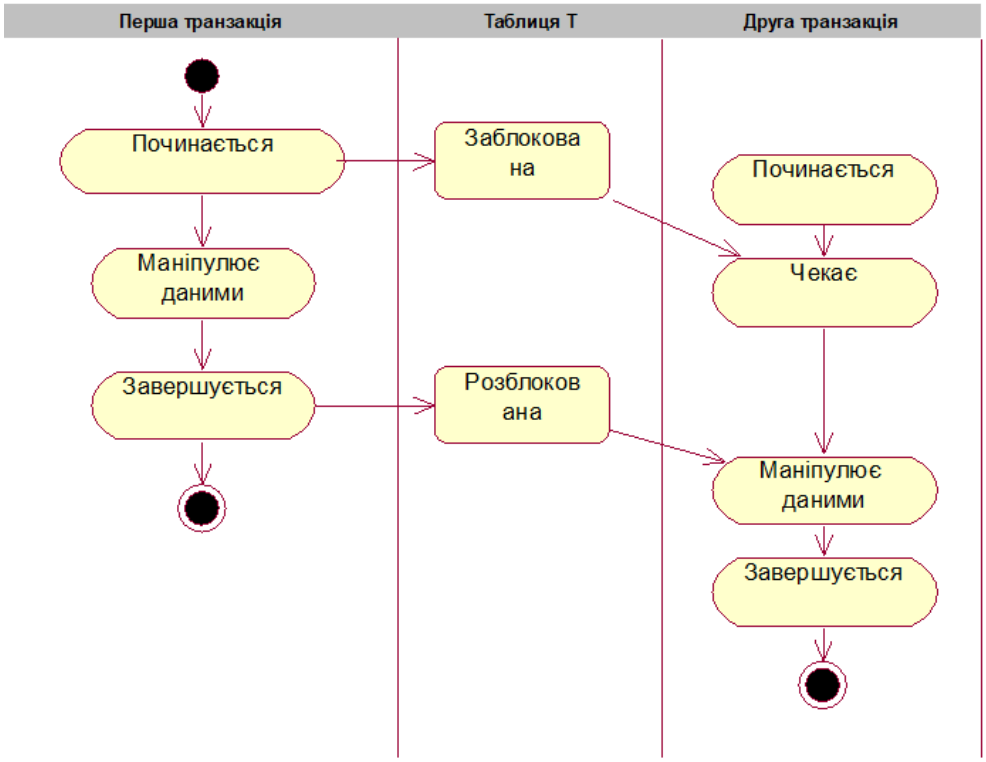


Рисунок 17 – Блокування другої транзакції першою

Якщо друга транзакція спочатку заблокує інші дані, потрібні першій транзакції, виникає взаємоблокування.

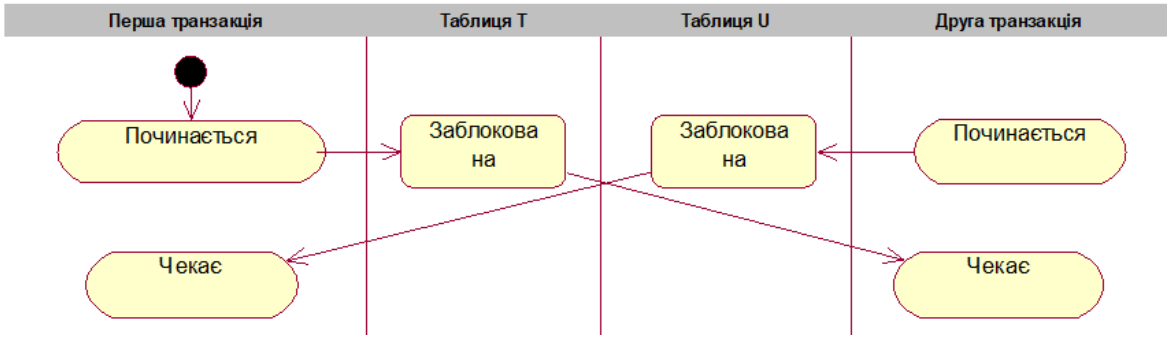


Рисунок 18 – Взаємоблокування першої та другої транзакцій

В даному прикладі обом транзакціям потрібні одні й ті ж таблиці T і U для монопольного блокування таблиць або одні й ті ж рядки цих таблиць. Але транзакція 2 не може отримати блокування таблиці T, тому що її вже заблокувала транзакція 1. Транзакція 1, у свою чергу, намагається заблокувати таблицю U, але її вже заблокувала транзакція 2. Жодна з транзакцій не зможе звільнити утримувані блокування до фіксації або відкоту. Але транзакціям не удасться зафіксуватися або виконати відкот, оскільки для продовження роботи необхідне блокування таблиці, що утримується іншою транзакцією.

Для зняття взаємоблокувань у СУБД передбачено автоматичний пошук і припинення блокуючих процесів, але покладатись на це не варто.

Мінімізація числа взаємоблокувань

Для мінімізації взаємоблокувань треба дотримуватись наступних правил:

- Уникати взаємодії з користувачем під час транзакції.
- Слідкувати, аби транзакції були короткими і не виходили за межі пакету.
- Використовувати низькі (мінімальні) рівні ізоляції.
- Оператор SET Lock_timeout дозволяє задати максимальний період чекання оператором звільнення ресурсу (в мілісекундах). Якщо чекання продовжилося довше встановленого LOCK_TIMEOUT, заблокований оператор автоматично відміняється і застосунку повертається повідомлення про помилку.

Розподілені транзакції

Приклад 8.2
База даних залізниці з даними про пасажирські перевезення є розподіленою.

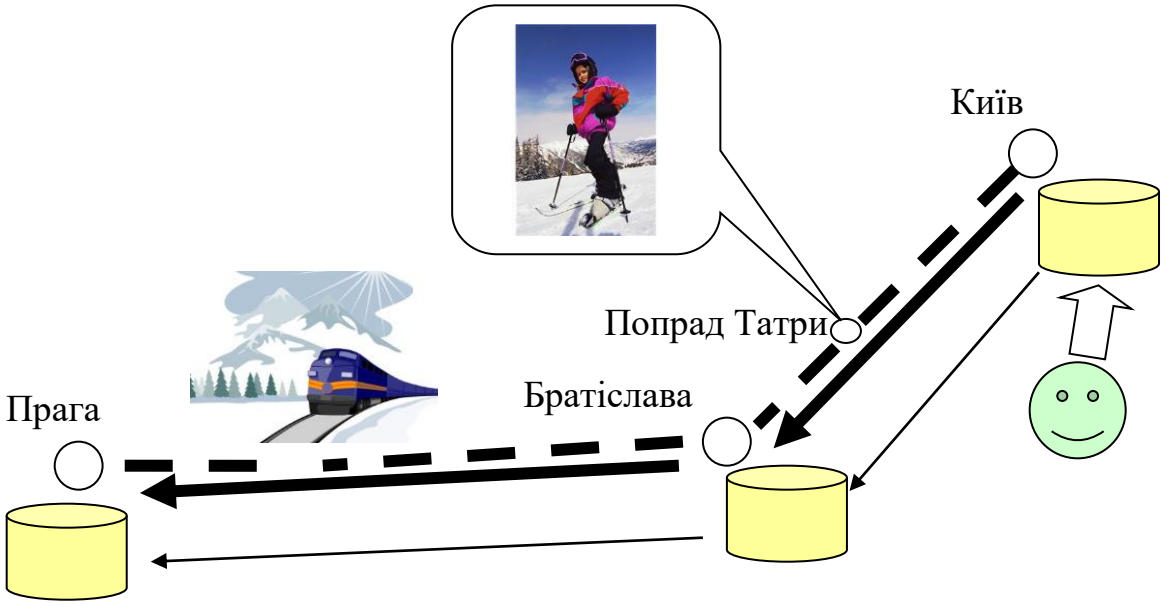


Рисунок 21 – Не вдаючись до деталей структури БД, можна стверджувати, що інформація про проданий квиток у міжнародний безпересадковий вагон має бути відображена як мінімум у БД залізниць трьох держав.

У виконанні розподілених транзакцій беруть участь два або більше серверів, які в цьому випадку називаються *диспетчерами ресурсів*. Серверний компонент під назвою *диспетчер транзакцій* координує управління транзакціями, здійснюване диспетчерами ресурсів.

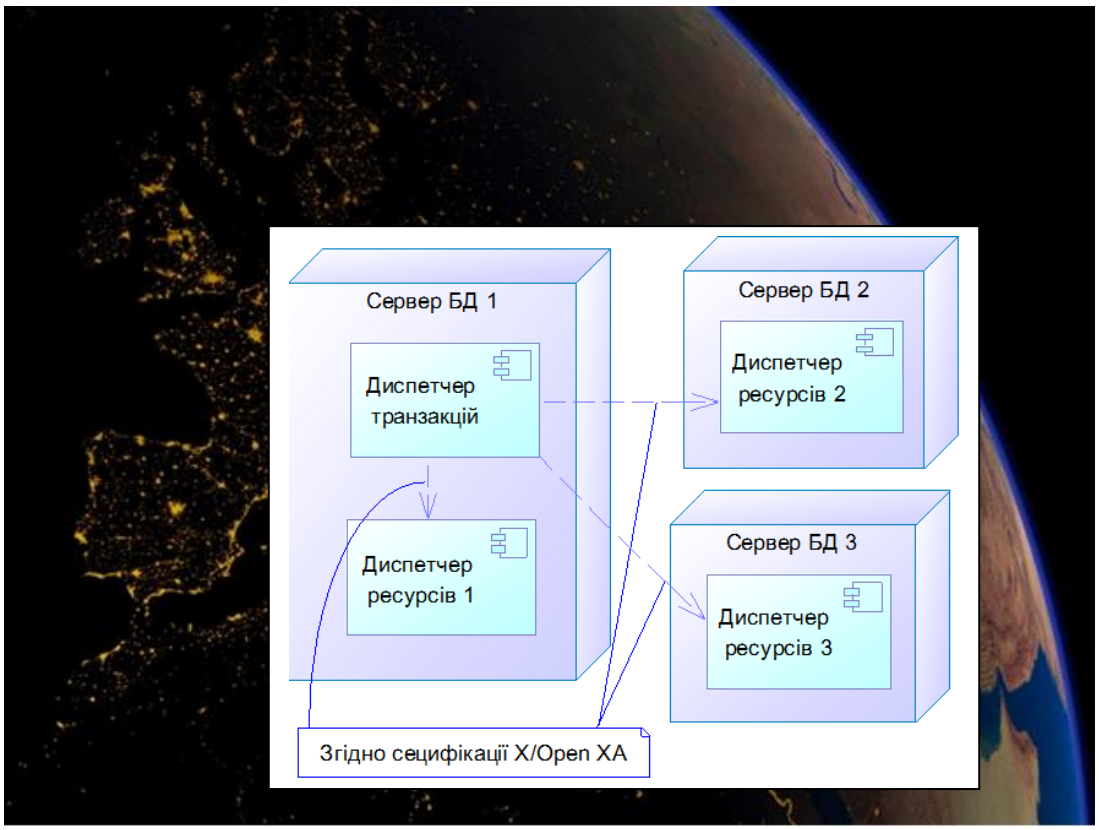


Рисунок 22 – Координація розподіленої транзакції

MsSQL Server здатний працювати як диспетчер ресурсів під час виконання розподілених транзакцій, що координуються такими диспетчерами транзакцій, як Microsoft Distributed Transaction Coordinator (MS DTC – компонента MsSQL), або іншими, що підтримують специфікацію обробки розподілених транзакцій X/Open XA.

XA транзакції вперше були описані в 1992 році. Не дивлячись на настільки поважний вік, специфікація обов'язково реалізується в менеджерах розподілених транзакцій (в цьому разі – XA-сумісних) для реляційних баз даних. XA означає "eXtended Architecture" (розширена архітектура) та є стандартом X/Open group для виконання «глобальної» транзакції, яка звертається до більше ніж одного сховища даних. Для цього клієнту надається процедурний API.

Вимоги ACID є одним з ключових елементів баз даних, але, як правило, гарантії ACID забезпечуються для діяльності, які відбуваються всередині однієї бази даних. Координація XA передбачає дотримання вимог ACID в розподіленій транзакції. Натомість в розподіленій мережі реляційних БД ми втрачаємо доступність (Availability), що випливає з CAP-теореми.

Виконання розподіленої транзакції

Рішення про закріплення (збереження, фіксацію) або відкат розподіленої транзакції лягає на координатора – Диспетчера транзакцій, прикладна програма визначає лише межі транзакцій. Диспетчер транзакцій ділить одну глобальну транзакцію на декілька гілок (branch).

Розглянемо виконання розподіленої транзакції, коли диспетчер транзакцій керує двома диспетчерами ресурсів.

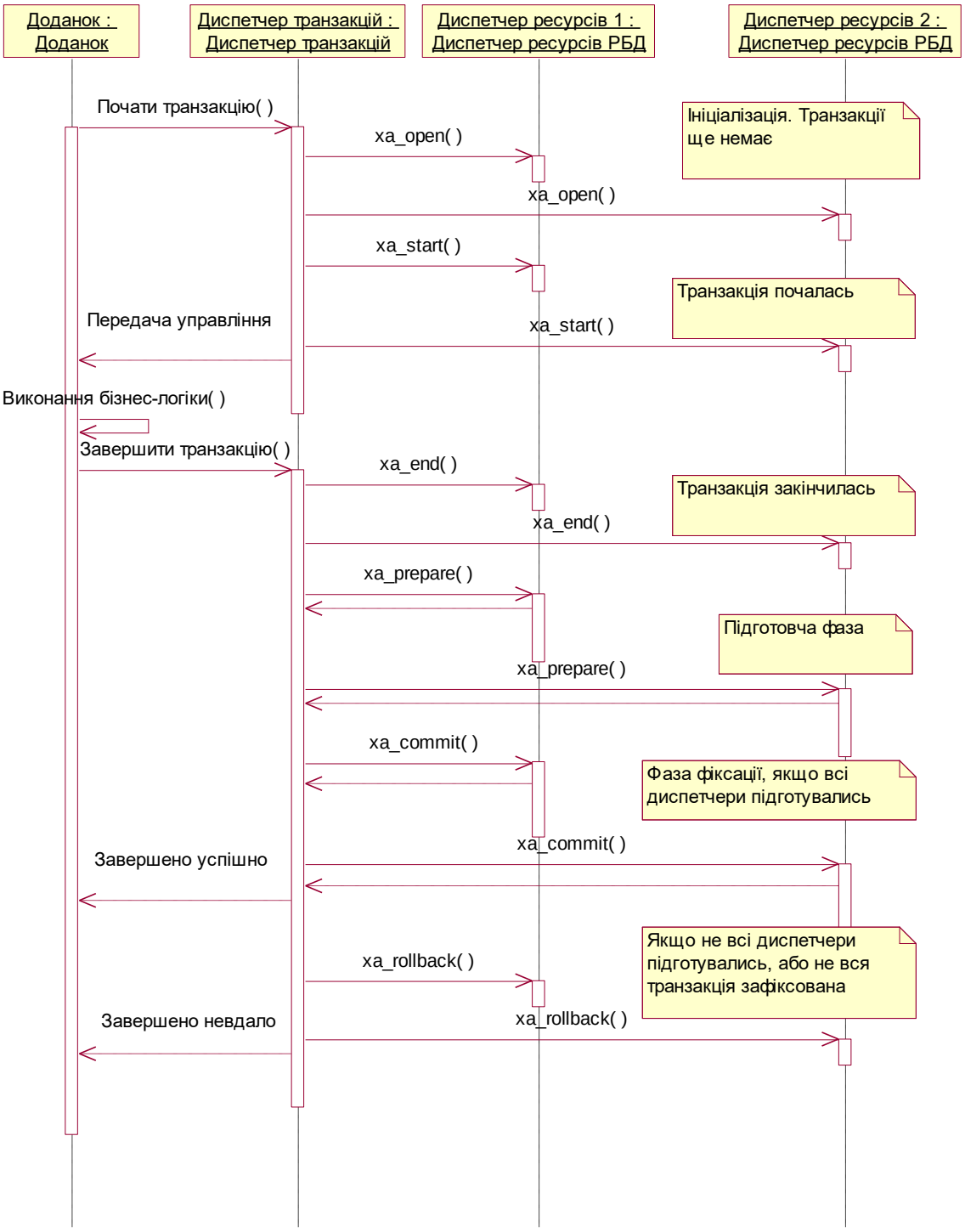


Рисунок 23 – Виконання розподіленої транзакції

Під час *підготовчої фази* (prepare) при завершенні транзакції кожний диспетчер ресурсів:

- Скидає на диск кеш журналу транзакцій.
- Повертає диспетчеру транзакцій повідомлення про успішне або невдале завершення підготовчої фази.

Під час *фази фіксації* (commit) при завершенні транзакції :

- Якщо всі диспетчери ресурсів повідомляють диспетчер транзакцій про успішне завершення підготовчої фази, то він посилає їм команду «зафіксувати».
- Диспетчери ресурсів фіксують транзакцію.
- Якщо всі диспетчери ресурсів повідомляють про успішну фіксацію, диспетчер транзакцій повідомляє застосунок про успішне завершення транзакції.
- Якщо хоч один з диспетчерів ресурсів посилає повідомлення про збій підготовчої фази, диспетчер транзакцій посилає всім диспетчерам ресурсів команду ROLLBACK і повідомляє застосунок про збій фіксації.

Що відбудеться в разі відмови сторони Диспетчера транзакцій (ДТ) або сторони СУБД під час тої чи іншої фази?

Відмова під час фази	*Відмовила сторона*	*Що буде*
До фази prepare	СУБД	В разі втрати з'єднання або по спливанню певного часу (timeout) ДТ відкатить транзакцію
	ДТ	СУБД по timeout відкатить транзакцію
prepare	СУБД	СУБД, яка відмовила, відкатить транзакцію, оскільки для неї не було фази prepare. ДТ та інші СУБД відкатять транзакцію по timeout
	ДТ	Після рестарту ДТ він, спираючись на log транзакції, виконує функцію xa_recover() для кожної СУБД. Для тих гілок, які пройшли фазу prepare, буде виконано відкат (виклик xa_rollback()). Інші гілки будуть забуті (виклик xa_forget()).
commit	СУБД	Після відновлення СУБД ДТ викличе xa_recover() та завершить транзакцію.
	ДТ	Після старту ДТ прочитає з log'у інформацію про незавершені транзакції та завершить фазу commit - зафіксує транзакцію.

Виклик розподіленої транзакції

MS DTC (Distributed Transaction Coordinator) викликається, коли транзакція стає розподіленою, якщо ми в її межах запускаємо розподілений запит, який звертається до віддалених даних, або викликаємо віддалену збережену процедуру.

Для організації розподіленої транзакції можна викликати стандартні оператори Transact-SQL:
BEGIN TRANSACTION
BEGIN DISTRIBUTED TRANSACTION
COMMIT TRANSACTION
ROLLBACK TRANSACTION

Таким чином, застосування, що використовують OLEDB, ODBC, ADO, можуть використовувати розподілені транзакції, використовуючи TRANSACT-SQL. Окрім цього, OLE DB і ODBC також підтримують управління розподіленими транзакціями на рівні API. За допомогою функцій API застосування, що використовують OLE DB і ODBC, дістають можливість управляти розподіленими транзакціями, в яких задіяні інші СУБД, відмінні від SQL Server.