**Лекція 3. Принципи функціонування систем із РБД**

1. **Фундаментальний принцип РБД**
2. **Незалежність від конфігурації**
3. **Незалежність від системних засобів**
4. **Незалежність від даних**

**3.1.Фундаментальний принцип РБД**

Фундаментальний принцип розподіленої бази даних формулюється так:

**Для користувача розподілена система повинна виглядати так само, як і нерозподілена система.**

Роботу користувачів у розподіленій системі варто організовувати в такий же спосіб, якби вона не була розподіленою. Усі зв'язані з розподіленими системами проблеми є (чи повинні бути) внутрішніми і повинні виникати тільки на внутрішньому рівні, чи на рівні розробки, а не на зовнішньому рівні чи на рівні користувачів.

У даному випадку термін "користувачі" відноситься до користувачів (чи розробників додатків), що виконують операції керування даними. Усі ці операції повинні залишатися логічно незмінними, на відміну від операцій визначення даних, що, навпаки, можуть бути розширені в розподіленій системі. Наприклад, користувач на вузлі Х може вказати, що збережене відношення можна розділити на "фрагменти" для фізичного збереження на вузлах Y і Z. Поняття "фундаментального принципу" було вперше запропоноване К.Дж.Дейтом і було названо "правилом нуль" (RuleZero). Даний принцип приводить до набору наступних правил і цілей:

|  |  |  |  |
| --- | --- | --- | --- |
| **Фундаментальний принцип (глобальна прозорість)** | | | |
| **Незалежність від конфігурації** | **Незалежність від засобів** | **Незалежність від даних** | **Режими роботи** |
| 1. Локальна автономія | 3. Незалежність від апаратного забезпечення | 7. Незалежність від фрагментації | 10. Безупинне функціонування |
| 2. Незалежність від центрального вузла | 4. Незалежність від операційної системи | 8. Незалежність від реплікації | 11. Обробка розподілених запитів |
|  | 5. Незалежність від мережі | 9. Незалежність від розташування | 12. Керування розподіленими транзакціями |
|  | 6. Незалежність від СКБД |  |  |

Ці дванадцять цілей не є незалежними одна від іншої, дотого ж не усі вони рівнозначні (різні користувачі можуть надавати різного значення різним цілям у різному оточенні), ними також не вичерпується список усіх можливих цілей. Однак вони дуже корисні для розуміння основ розподіленої технології і для загальної характеристики функціональності деякої розподіленої системи.

Необхідно розрізняти розподілені системи і системи, у яких підтримуються деякі способи віддаленого доступу до даних. У системах з віддаленим доступом до даних користувач може одночасно працювати з даними, розміщеними на декількох віддалених вузлах, але у такому випадку будуть видні з'єднання між ними. У справді розподіленій системі, навпаки, усі з'єднання сховані від користувача.

**3.2. Незалежність від конфігурації**

###### Локальна автономія

У розподіленій системі вузли потрібно робити ***автономними***. Локальна автономія означає, що операції на даному вузлі керуються цим вузлом, тобто функціонування будь-якого вузла Х не залежить від успішного виконання деяких операцій на якомусь іншому вузлі В (інакше вихід з ладу вузла В може привести до неможливості виконання операцій на вузлі X, навіть якщо з вузлом Х нічого не трапилось). Із принципу локальної автономії також випливає, що зберігання даних і керування даними здійснюється локально. Дійсно, навіть якщо доступ до даних здійснюється з інших віддалених вузлів, усі вони відносяться до деякої локальної бази даних. Такі питання, як безпека, цілісність і структура зберігання локальних даних, залишаються під контролем цього локального вузла.

Насправді мета локальної автономності досягається неповністю, тому що є багато ситуацій, у яких вузол Х повинен надавати деяку частину керування іншому вузлу У. Тому ціль досягнення локальної автономії вимагає більш точного формулювання, а саме: **вузли варто робити автономними в максимально можливій мірі**.

**Незалежність від центрального вузла**.

Під локальною автономією розуміють, що ***усі вузли повинні розглядатися як рівні***. Отже, не повинно існувати ніякої залежності і від центрального "основного" вузла з деяким централізованим обслуговуванням, наприклад, централізованою обробкою запитів чи централізованим керуванням транзакціями. Таким чином, друга мета є логічним наслідком першої. Однак, навіть якщо локальна автономія не досягається повною мірою, то досягнення незалежності від центрального вузла саме по собі дуже важливе і може розглядатися як окрема мета.

Залежність від центрального вузла не бажана принаймні по двох причинах. По-перше, центральний вузол може бути "вузьким" місцем усієї системи, а по-друге, більш важливо те, що система в такому випадку стане дуже *уразливою*, тобто при виході з ладу центрального вузла може відмовити вся система.

**3.3.Незалежність від системних засобів**

**Незалежність від апаратного забезпечення**

Тут важко додати що-небудь істотне, оскільки основна ідея фактично цілком викладена в назві цього розділу. Комп'ютери, які використовуються сьгодні досить різноманітні, серед них можна зустріти комп'ютери фірм ІBC, DEC, HP, а також персональні комп'ютери і робочі станції інших типів. У зв'язку з цим існує реальнана обхідність інтеграції даних на всіх цих системах і створення для користувача "уяви єдиної системи". Таким чином, дуже важливою є можливість запуску копій однієї і тієї ж СКБД на різному апаратному забезпеченні для того, щоб різні комп'ютери могли працювати в розподіленій система як рівні партнери.

**Незалежність від операційної системи**

Ця мета є наслідком попередньої, і її опис буде також небагатослівним. Очевидно, важливо запустити ту саму СКБД не тільки на різному апаратному забезпеченні, але і на різних операційних системах, причому навіть у тих випадках, коли різні операційні системи використовуються на однотипному апаратному забезпеченні. Наприклад, для того щоб версії СКБД для операційної системи MVS, а також для систем UNІX і Windows могли спільно працювати в одній і тій самій розподіленій системі.

**Незалежність від мережі**

Тут також можна обмежитися коротким зауваженням про те, що якщо система в стані підтримувати кілька вузлів з різним апаратним забезпеченням і різними операційними системами, то бажано було б, щоб у ній підтримувалися також різні типи мереж.

**Незалежність від СКБД**

Ця мета має на увазі використання трохи менш точного формулювання припущення про строгу однорідність. У новій формі це припущення означає, що ***всі екземпляри СКБД на різних вузлах підтримують той самий інтерфейс*** (типи даних, реалізація запитів, транзакції та ін.), хоча вони необов'язково повинні бути копіями того самого програмного забезпечення. Наприклад, якби системи ІNGRES і ORACLE підтримували офіційний стандарт SQL, то було б можливо організувати їхню спільну роботу в контексті розподіленої системи. Інакше кажучи, розподілена система, принаймні до деякої міри, може бути неоднорідною.

Підтримка цієї неоднорідності дуже бажана. Справа в тому, що в реальному світі робота зазвичай організована не тільки на комп'ютерах різних типів і в різних операційних системах, але і за участю різних СКБД. Було б прекрасно, якби такі СУБД могли працювати спільно в одній розподіленій системі. Інакше кажучи, в ідеальній розподіленій системі передбачається підтримка незалежності від СКБД.

Цій досить великій і дуже важливій з практичної точки зору темі присвячено окремий розділ.

**3.4 Незалежність від даних**

**Незалежність від розташування**

Основна ідея незалежності від розташування (яка також називається прозорістю розташування) досить проста: користувачам непотрібно знати, де фізично зберігаються дані, навпаки, з логічної точки зору користувачам варто було б забезпечити такий режим, при якому створюється враження, що всі дані зберігаються на їх власному локальному вузлі. Бажано забезпечувати незалежність від розташування, тому що при цьому істотно спрощуються програми користувачів і термінальна діяльність. Зокрема, це дозволяє здійснювати міграцію даних від вузла до вузла без оголошення недійсними будь-яких програм користувача і видів термінальної діяльності. Процес міграції дуже корисний, оскільки дозволяє переміщатися даним по всій мережі у відповідь на зміну вимог до продуктивності.

**Незалежність від фрагментації**

Фрагментація даних означає, що дані розділяються на частини таким чином, щоб частини (фізичні фрагменти даних - таблиці) зберігалися в тому місці де вони частіше використовуються. При такій організації багато операцій будуть чисто локальними, а обсяг даних, що пересилаються в мережі, знизиться. Фрагментація бажана для підвищення продуктивності системи. Однак з логічної точки зору користувачу забезпечується такий режим роботи при якому дані здаються нефрагменованими зовсім.

Розглянемо відношення співробітників ЕМР, показане на рис2. У системі, у якій підтримується фрагментація, визначені два фрагменти, показаних на мал.2

|  |  |  |
| --- | --- | --- |
| EMP# | DEPT# | SALARY |
| E1 | D1 | 4800 |
| E2 | D1 | 3700 |
| E3 | D2 | 4500 |
| E4 | D2 | 5000 |
| E5 | D3 | 6000 |

Київ

|  |  |  |
| --- | --- | --- |
| EMP# | DEPT# | SALARY |
| E1 | D1 | 4800 |
| E2 | D1 | 3700 |
| E5 | D3 | 6000 |

Одеса

|  |  |  |
| --- | --- | --- |
| **EMP#** | DEPT# | SALARY |
| E3 | D2 | 4500 |
| E4 | D2 | 5000 |

EMP

K\_EMP

O\_EMP

Сприйняття користувача

Тепер оптимізатору відомо, що доступ потрібно здійснити тільки до вузла у Києві.

Як випливає зі сказаного вище, проблема підтримки операцій для фрагментованих відношень має деяку подібність із проблемою підтримки операцій для об'єднаних і з'єднаних представлень (фактично, це різні прояви однієї і тієї ж проблеми при розгляді загальної архітектури системи з різних точок зору). Зокрема, проблема відновлення фрагментованих відносин-це проблема відновлення в об'єднаннях і з'єднаннях представлень. Звідси також випливає, що при відновленні заданий кортеж може мігрувати з одного фрагмента в іншій за умови, що він більше не задовольняє предикату відношення для фрагмента, до якого він належав до відновлення.

Існує два основних типи фрагментації - ***горизонтальна*** і ***вертикальна***, які відповідно зв'язані з реляційними операціями вибірки та проекції. (На рис.2 показана горизонтальна фрагментація.) Інакше кажучи, фрагментом може бути будь-яке довільне підлегле відношення, яке можна отримати з вихідного відношення за допомогою операцій вибірки і проекції. При цьому варто врахувати приведені нижче допущення.

* Передбачається без утрати спільності, що усі фрагменти даного відношення незалежні, тобто ні одн із фрагментів не може бути виведений з інших фрагментів або мати вибірку чи проекцію, яка може бути виведена з інших фрагментів.
* Проекції (фрагменти та їх об`єднання) не повинні допускати втрату інформації.

Реконструкцію початкового відношення на основі його фрагментів можна здійснити за допомогою операцій з'єднання (сполучення) (для вертикальних фрагментів) і об'єднання (для горизонтальних фрагментів). Зверніть увагу, що у випадку об'єднання буде потрібно виконувати операцію виключення дублікатів.

*Зауваження*. Відносно вертикальної фрагментації необхідні додаткові пояснення. Як уже згадувалося вище, при такій фрагментації не повинна допускатися втрата інформації, тому що фрагментація відношення ЕМР на проекції (EMP#, DEPT#) і (SALARY) не буде достовірною. Але у деяких системах збережені відношення мають схований атрибут TІ ("tupleІD" – ідентифікатор кортежу), тобто фізична чи логічна адреса цього кортежу. Атрибут TІ є потенційним ключем для даного кортежу, тому якби відношення ЕМР містило такий атрибут, то воно могло б бути коректно фрагментовано на проекції (TІ, EMP#, DEPT#) і (TІ, SALARY), оскільки така фрагментація відбувається без втрати інформації.

Зверніть увагу, що саме легкість виконання фрагментації і реконструкції – одна з багатьох причин, із-за яких для розподілених систем використовується реляційна модель. Нарешті, переходячи до головного, потрібно сказати, що в системі, яка підтримує фрагментацію даних, варто також передбачити підтримку ***незалежності від фрагментації*** (чи прозорість фрагментації). Іншими словами, з логічної точки зору користувачам варто забезпечити такий режим роботи, при якому дані здаються зовсім нефрагментованими. Незалежність від фрагментації (як і незалежність від розташування) досить важлива, оскільки вона дозволяє в будь-який момент здійснити дефрагментацію даних (і фрагменти можуть бути перерозподілені в будь-який момент часу) у відповідь на зміну вимог до продуктивності, причому без необхідності відключення будь-яких програм користувача і термінальної діяльності.

Незалежність від фрагментації припускає, що дані будуть представлені для користувачів у виді логічно комбінованих фрагментів на основі відповідних об'єднань і з'єднань. При цьому системний оптимізатор відповідає за визначення фрагментів, до яких необхідно забезпечити фізичний доступ для виконання будь-якого заданого користувачем запиту. Як приклад допустимо, що для фрагментації, показаної на рис.2, користувач задає наступний запит:

EMP WHERE SALARY>40K AND DEPT#='D1'

У такому випадку оптимізатор системи буде знати з визначень фрагментів (які зберігаються в системному каталозі), що всі результати можуть бути отримані на вузлі в Одесі і немає ніякої необхідності здійснювати доступ до вузла в Києві.

Розглянемо цей же приклад більш ретельно. Відношення ЕМР сприймається користувачем як представлення базових фрагментів O\_EMP і K\_EMP: EMP=O\_EMP UNІON K\_EMP

Таким чином, оптимізатор перетворить вихідний запит користувача у наступний:

(O\_EMP UNІON K\_EMP) WHERE SALARY>40K AND DEPT#='D1'

А цей вираз, у свою чергу, можна перетворити до виду

(O\_EMP WHERE SALARY>40K AND DEPT#='D1') UNІON

(K\_EMP WHERE SALARY>40K AND DEPT#='D1')

З фрагмента L\_EMP, який зберігаються у каталозі визначення, оптимізатору відомо про те, що другий вираз дасть у результаті порожнє відношення (умова цілісності DEPT#='D1' AND DEPT#='D2' не може бути оцінена як “істина”). Таким чином, усі вирази в цілому можна звести до наступного спрощеного виду:

O\_EMP WHERE SALARY > 40K AND DEPT#='D1'

Тепер оптимізатору відомо, що доступ потрібно здійснити тільки до вузла в Одесі.

Як випливає зі сказаного вище, проблема підтримки операцій для фрагментованих відношень має деяку подібність із проблемою підтримки операцій для об'єднаних і з'єднаних представлень (фактично, це різні прояви однієї і тієї ж проблеми при розгляді загальної архітектури системи з різних точок зору). Зокрема, проблема відновлення фрагментованих відношень – це проблема відновлення в об'єднаннях і з'єднаннях представлень. Звідси також випливає, що при відновленні заданий кортеж може мігрувати з одного фрагмента в іншій за умови, що він більше не задовольняє предикату відношень для фрагмента, до якого він належав.

Незалежність від реплікації

Реплікація - **це** підтримка двох і більше ідентичних копій (реплік) даних на різних вузлах РБД. Репліка може включати всю базу даних (повна реплікація), одне або кілька взаємопов'язаних відношень або фрагмент відношень. Також можливий варіант з консолідацією даних, при якому кожен вузол володіє своєю частиною даних (наприклад, відношення) і може її оновлювати, а на одному з вузлів РБД ці частини з'єднуються (або об'єднуються) разом для утворення консолідованої копії "тільки для читання"

Реплікація даних означає, що деякий фрагмент данних (таблиця) може бути представлений у базі даних різними копіями (репліками) збереженими на декількох різних вузлах.

Реплікації корисні по двом причинам:

1. Збільшення продуктивності системи, тому що додатки можуть працювати з локальними даними без обміну інформацією з віддаленими вузлами.
2. Збільшення доступності та надійності, тому що об'єкт залишається доступним для обробки поки є хоч би одна його репліка.

Головний *недолік* реплікації полягає в тому, що при відновленні реплікованого об'єкта всі копії цього об'єкта повинні також поновлюватися. Цей недолік називається *проблемою розповсюдження поновлення*.

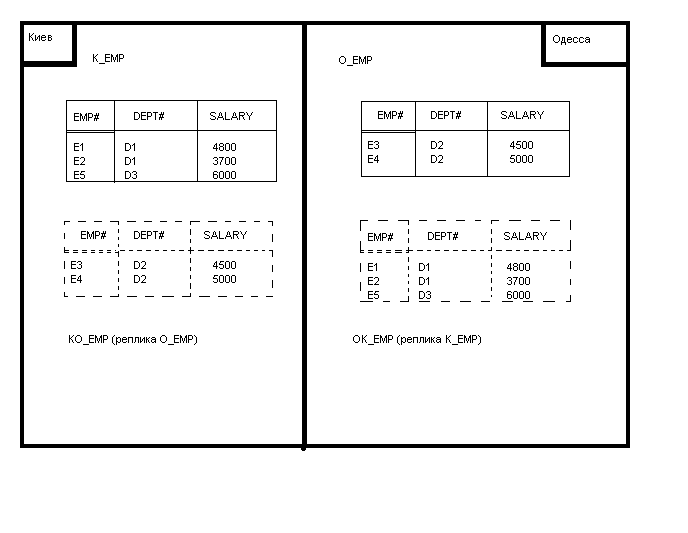
З логічної точки зору користувачі повинні праювати так ніби дані не є репліковані.

Існує декілька механізмів виконання реплікації, це - множинна публікація, централізована публікація і централізована підписка.

Нижче наведено приклад реплікації, схематично показаний на рис.3.

REPLICATE К\_ЕМР ОК\_ЕМР АТ SIТЕ 'Одеса'; REPLICATE О\_ЕМР КО\_ЕМР АТ SIТЕ 'Київ';

Внутрішні імена реплік ОК\_ЕМР і КО\_ЕМР.



Київ

K\_EMP

|  |  |  |
| --- | --- | --- |
| EMP# | DEPT# | SALARY |
| E1 | D1 | 4800 |
| E2 | D1 | 3700 |
| E5 | D3 | 6000 |

KO\_EMP (репліка O\_EMP)

|  |  |  |
| --- | --- | --- |
| EMP# | DEPT# | SALARY |
| E3 | D2 | 4500 |
| E4 | D2 | 5000 |

Одеса

O\_EMP

|  |  |  |
| --- | --- | --- |
| EMP# | DEPT# | SALARY |
| E3 | D2 | 4500 |
| E4 | D2 | 5000 |

OK\_EMP (репліка K\_EMP)

|  |  |  |
| --- | --- | --- |
| EMP# | DEPT# | SALARY |
| E1 | D1 | 4800 |
| E2 | D1 | 3700 |
| E5 | D3 | 6000 |

Рис.3. Приклад реплікації

Слід зазначити, що реплікація в розподіленій системі являє собою особливий вид втілення ідеї керованої надлишковості. В ідеальному випадку реплікація, як і фрагментація, повинна бути "прозорою для користувача". Інакше кажучи, у системі, у якій підтримується реплікація даних, повинна також підтримуватися незалежність від реплікації (чи прозорість реплікації), тобто користувачі, принаймні, з логічної точки зору, повинні працювати в такому режимі, наче б то дані нерепліковані зовсім. Незалежність від реплікації (так само як незалежність від розташування і незалежність від фрагментації) дуже бажана, оскільки дозволяє істотно спростити програми користувача і термін їх дії. Зокрема, при цьому дозволяється у будь-який момент створювати і видаляти репліки у відповідь на зміни вимог без відключення програм користувача.

Під незалежністю від реплікації мається на увазі, що системний оптимізатор відповідає за визначення фізичного доступу саме до тих реплік, які необхідні для виконання заданого запиту користувача.

До основних достоїнств механізму реплікації можна віднести підвищення доступності та надійності даних і підвищення локалізації посилань на репліковані дані. Основним недоліком реплікації є складність підтримання ідентичності реплік: якщо в одну копію даних вносяться зміни, то ці зміни також повинні бути внесені в інші копії. Це називається поширення змін і реалізується службою тиражування.  
Служба тиражування повинна виконувати такі функції:  
1. Забезпечення масштабованості, тобто ефективна обробки великих і малих об'ємів даних.  
2. Перетворення типів і моделей даних (для гетерогенних РБД).  
3. Реплікація об'єктів БД, наприклад, індексів, тригерів і т.п.  
4. Ініціалізація новостворюваної репліки.  
5. Забезпечення можливості "підписатися" на існуючі репліки, щоб отримувати їх в певною періодичністю.

Для виконання цих функцій в мові, підтримуваному СУБД, передбачається наявність засобів визначення схеми реплікації, механізму передплати та механізму ініціалізації реплік (створення і заповнення даними).  
Існують різні підходи до організації реплікації:  
**1 Реплікація з основною копією.**

Існують наступні варіанти: **2.**1. Класичний підхід полягає в наявності однієї основної копії, в яку можна вносити зміни, інші копії створюються з визначенням тільки для читання.  
2. 2. Асиметрична реплікація: основна копія фрагментована й розподілена по різних вузлах РБД, та інші вузли можуть бути передплатниками окремих фрагментів (тільки для читання).  
2.3. Робочий потік. При використанні цього підходу право поновлення не належить постійно одній копії, а переходить від однієї копії в інший відповідно до потоком операцій. У кожен момент часу оновлюватися може тільки одна копія. Наприклад, при послідовній обробці замовлень на поставку товарів спочатку у відділі прийому замовлення приймається, потім на складі визначають, чи може воно бути виконано і скільки це буде коштувати, потім ці дані передаються у фінансовий відділ, а після оплати - у відділ доставки. При реплікації даних з вузла Si на вузол Si +1 разом з новими даними передається право на оновлення репліки.  
**2. Симетрична реплікація (без основної копії).**

Усі копії набору що підлягає реплікації можуть оновлюватися одночасно і незалежно один від одного, але всі зміни однієї копії повинні потрапити в усі інші копії.  
Існує два основних механізми поширення змін при симетричній реплікації:  
**- Синхронний: зміни у всі копії вносяться в рамках однієї транзакції;**

Достоїнствами синхронного поширення змін є повна узгодженість копій і відсутність конфліктів поновлення. До недоліків слід віднести трудомісткість і велику тривалість модифікації даних і низьку надійність роботи системи: при виході з ладу одного вузла всі копії стають недоступні для внесення змін.  
**- Асинхронний: мається на увазі відкладений характер внесення змін до віддалені копії.**

При використанні асинхронного режиму розповсюдження змін можуть виникати конфлікти поновлення.

Можна виділити наступні конфліктні ситуації:  
1. Додавання двох записів з однаковими первинними або унікальними ключами. Для запобігання таких ситуацій зазвичай кожному вузлу РБД виділяється свій діапазон значень ключових (унікальних) полів.  
2. Конфлікти видалення: одна транзакція намагається видалити запис, який в іншій копії вже видалена інший транзакцією. Якщо така ситуація вважається конфліктом, то вона дозволяються вручну.  
3. Конфлікти поновлення: дві транзакції в різних копіях оновили одну і ту ж запис, можливо, по-різному, і намагаються розповсюдити свої зміни. Для ідентифікації конфліктів поновлення необхідно передавати з транзакцією додаткову інформацію: старе і нове вміст запису. Якщо стара запис не може бути виявлена, наявності конфлікт поновлення.

Для вирішення конфліктів поновлення застосовуються різні методи, наприклад:  
1) Дозвіл по пріоритету вузлів: для кожного вузла призначається пріоритет, і до запису застосовується оновлення, яке надійшло з вузла з максимальним пріоритетом.  
2) Дозвіл по тимчасовій відмітці: всі транзакції мають тимчасову відмітку, і до запису застосовується оновлення з мінімальної або максимальної відміткою. Чи використовувати для цього мінімальну або максимальну позначку - залежить від предметної області і, звичайно, може регулюватися.  
3) Адитивний метод (додати - додати): може застосовуватися в тих випадках, коли зміни засновані на попередньому значенні поля, наприклад, зарплата = оклад + X. При цьому до значення поля послідовно застосовуються всі оновлення.  
4) Використання користувальницьких процедур.  
5) Вирішення конфліктів вручну. Відомості про конфлікт записуються в журнал помилок для подальшого аналізу та усунення адміністратором.  
Для налаштування реакції на конфлікти деякі СУБД дозволяють виділити в кожній таблиці стовпець (групу стовпців), і описати для них спосіб вирішення конфліктів (т.зв. група поновлення).  
Існують різні варіанти реалізації розповсюдження змін. Один з них полягає у використанні тригерів. Нагадаємо, що тригери - це процедури, які спрацьовують при настанні певних подій (наприклад, вставка, видалення, оновлення). Всередину тригера містяться команди, які проводять на інших копіях поновлення, аналогічні тим, які викликали виконання тригера. Цей підхід досить гнучкий, але він має низку недоліків:  
- Тригери створюють додаткове навантаження на систему;  
- Тригери не можуть виконуватися за графіком (час спрацьовування тригера не визначено);  
- За допомогою тригерів складніше організувати групове оновлення зв'язаних таблиць (через проблеми мутують таблиць).  
Інший спосіб реалізації механізму поширення змін - підтримка журналів змін для даних що підлягають реплікації. Розсилка цих змін входить у завдання сервера СУБД або сервера тиражування (входить до складу СУБД). Основні принципи, яких необхідно дотримуватися при цьому:  
1. Для збереження узгодженості даних повинен дотримуватися порядок внесення змін.  
2. Інформація про зміни повинна зберігатися в журналі до тих пір, поки не будуть оновлені всі копії цих даних.

У РБД застосовується сценарій централізованої підписки. Адміністратор призначає сервер реплікації та сервер підписки, визначає час, через який повинна відбуватися реплікація і набір таблиць, що беруть участь у ній. Також визначається кілька додаткових параметрів.

У нашому випадку реплікація відбувається відразу після завершення транзакції та поновлення таблиці. Адміністратор з клієнтського місця вносить зміни в таблицю, використовуючи для цього структуровану мову запитів SQL. Після виконання команди COMMІТ WORK, що означає, що транзакція успішно завершена, сервер публікації/реплікації встановлює з'єднання із сервером підписки і перевіряє дані з таблиць, які беруть участь у реплікації, на ідентичність. І у випадку виявлення нових даних сервер публікації репліккує нові записи у відповідні таблиці сервера підписки. У випадку виникнення помилки в процесі реплікації (наприклад, у випадку розриву мережі) відбувається відкат (откат) виконання реплікації, і після відновлення працездатності мережі, система автоматично переходить у нормальний стан і процес реплікації повторюється.

**Тиражування** – окремий випадок реплікації (на прикладі, ІNFORMіX-OnLіne DS)

Тиражування - підтримка на іншій обчислювальній машинні всіх копій об'єктів бази даних. При цьому реалізується прозоре тиражування даних з основного сервера баз даних на вторинний (чи підтримуючий) сервер, до якого дозволено доступ тільки на читання і який може знаходитися в іншому географічному пункті. У цій термінології сервер, що не бере участь у тиражуванні, називається стандартним.

*Головна мета тиражування* –це забезпечення високої готовності (Hіgh Avaіlabіlіty Data Replіcatіon - HDR). У випадку відмови основного сервера вторинному серверу автоматично чи вручну надаєтся статус стандартного, з доступом на читання і запис. Прозоре перенаправлення клієнтів при відмові основного сервера не підтримується, але воно може бути реалізоване в рамках додатків. Після відновлення основного сервера, у залежності від значення параметра конфігурації, вибирається один і здвох можливих сценаріїв:

* відновленому серверу знову надаєтся статус основного. Вторинний сервер, перед поверненням у режим доступу тільки на читання, зупиняється, щоб забезпечити від'єднання клієнтів, які здійснювали доступ до запису;
* відновлений серверста є вторинним, а колишньому вторинному, котрий уже функціонує в режимі читання-запису, надається статус основного; клієнти, що підключені до нього, продовжують роботу. Цей сценарій забезпечує безперервний доступ до баз даних.

Тиражування здійснюється шляхом передачі інформації з журналу транзакцій (логічного журналу) у буфер тиражування основного сервера, звідки вона пересилається в буфер тиражування вторинного сервера. Таке пересилання може відбуватися або в синхронному, або в асинхронному режимі. *Синхронний режим* гарантує повну погодженість баз даних – жодна транзакція, зафіксована на основному сервері, не залишиться незафіксованою на вторинному, навіть у випадку збою основного сервера. *Асинхронний режим* не забезпечує абсолютної погодженості, але поліпшує робочі характеристики системи.

*Дзеркальне відображеня*, що також є прозорим засобом підтримки високої готовності, забезпечує тільки копіювання дискових областей у межах однієї установки сервера ІNFORMіX-OnLіne DS і захищає тільки від дискових збоїв. Механізм тиражування забезпечує підтримку повної віддаленої копії баз даних і захищає від усіх видів відмовлень, включаючи повний крах однієї із установок.

Крім забезпечення безвідмовної роботи, тиражування дає наступні переваги:

* більш оперативний доступ до даних для локальних клієнтів вторинного сервера;
* можливість винести додатки користувача переважно на вторинний сервер, де вони виконуються з максимальним використанням даних, не подавляючи додатків, які виконуються на основному сервері.