Rozproszone i obiektowe systemy baz danych

Dr inż. Robert Wójcik

Wykład 6. Metody rozpraszania i sterowania przepływem danych

6.1. Fragmentacja i alokacja danych

6.2. Replikacja danych

6.3. Rodzaje replikacji w wybranych systemach baz danych

6.4. Standardowa replikacja migawkowa w Oracle

6.5. Replikacja zaawansowana w Oracle

6.6. Strumienie Oracle

W rozproszonych systemach baz danych pojawia się problem podziału danych pomiędzy węzły, tak aby efektywność systemu była zadowalająca.

Stosowane są techniki podziału wykorzystujące:

- Fragmentację i alokację danych,

- Replikację danych.

Wymienione techniki są implementowane w postaci różnych konfiguracji sprzętowych i programowych (np. fragmentacja pozioma, pionowa, hybrydowa; replikacja migawkowa, transakcyjna, peer-to-peer, i inne).

Celem rozproszenia danych jest lepsze odwzorowanie struktur organizacji, dla której system został zaprojektowany oraz zwiększenie kontroli nad danymi w miejscu ich wprowadzania i użytkowania.

Ponadto, rozproszenie zwiększa odporność systemu na awarie, a więc i poziom dostępności systemu.

Są jednak i wady rozproszenia danych:

- katalog systemowy bazy danych musi zawierać dodatkowe informacje o rozmieszczeniu fragmentów bazy oraz o sposobie replikacji;

- komplikuje się sposób sterowania współbieżnością oraz mechanizmy aktualizacji danych;

- komplikuje się testowanie oraz wykrywanie i usuwanie skutków awarii;

- utrudnione jest zarządzanie bezpieczeństwem systemu.

6.1. Fragmentacja i alokacja danych

Fragmentacja

W wielu przypadkach dane są z natury rozproszone geograficznie zgodnie z miejscem ich wykorzystania.

Niekiedy dane są rozpraszane po to, aby zmniejszyć rozmiar obsługiwanych tabel, co może poprawić wydajność wyszukiwania i przetwarzania informacji, np. w przypadku dużych, często używanych tabel można zmniejszyć współzawodnictwo procesów o dostęp do danych rozmieszczając je w kilku odrębnych fragmentach znajdujących się na różnych nośnikach danych. W przypadku awarii dysku z jednym fragmentem danych można korzystać z pozostałych fragmentów.

Fragmentacja umożliwia wydzielenie częściej używanych danych do osobnego fragmentu (tabeli). Taki podział danych pozwala skrócić czas odpowiedzi systemu – zwiększyć jego wydajność, dzięki równoległemu wykonywaniu operacji na różnych fragmentach tabeli.

Fragmentacja danych powinna być przeźroczysta dla użytkowników i aplikacji bazodanowych.

Alokacja

W rozproszonym systemie baz danych dane mogą być umieszczone w dowolnym jego węźle.

Problem alokacji sprowadza się do takiego rozmieszczenia danych (tabel, fragmentów), aby efektywność działania systemu w sensie zadanego kryterium oceny była optymalna.

Charakterystyka metod i celów fragmentacji tabel (partycjonowania)

Źródło:

Wykład_02 ze strony: http://wazniak.mimuw.edu.pl/index.php - systemy rozproszone, zaawansowane systemy baz danych.

Bell D., Grimson J., Distributed Database Systems, Addison Wesley, 1992.

Ozsu T. M., Valduriez P., Principles of Distributed Database Systems, Prentice Hall, 1999.

Jednym z podstawowych zagadnień projektowania RSBD jest określenie sposobu podziału danych pomiędzy węzły bazy danych, tak aby efektywność całego systemu była zadowalająca.

Fragmentacja polega na podziale obiektu przechowującego dane, najczęściej tabeli, na mniejsze części, zwane fragmentami lub partycjami.

Do najważniejszych celów fragmentacji należą:

- zwiększenie efektywności dostępu do danych;

Dzięki właściwemu doborowi kryterium podziału obiektów na fragmenty można zapewnić, że aplikacje użytkowników będą adresowały tylko wymagane partycje, w ten sposób zmniejszeniu ulegną wolumeny przeszukiwanych danych, a tym samym skrócą się czasy odpowiedzi.

- zrównoleglenie operacji dostępu do danych znajdujących się na różnych dyskach;

Ze względu na to, że każdy z fragmentów może być umieszczony na innym dysku możliwy staje się równoległy dostęp do wielu fragmentów równocześnie, co zwiększy wydajność przetwarzania danych.

- zredukowanie kosztów transmisji;

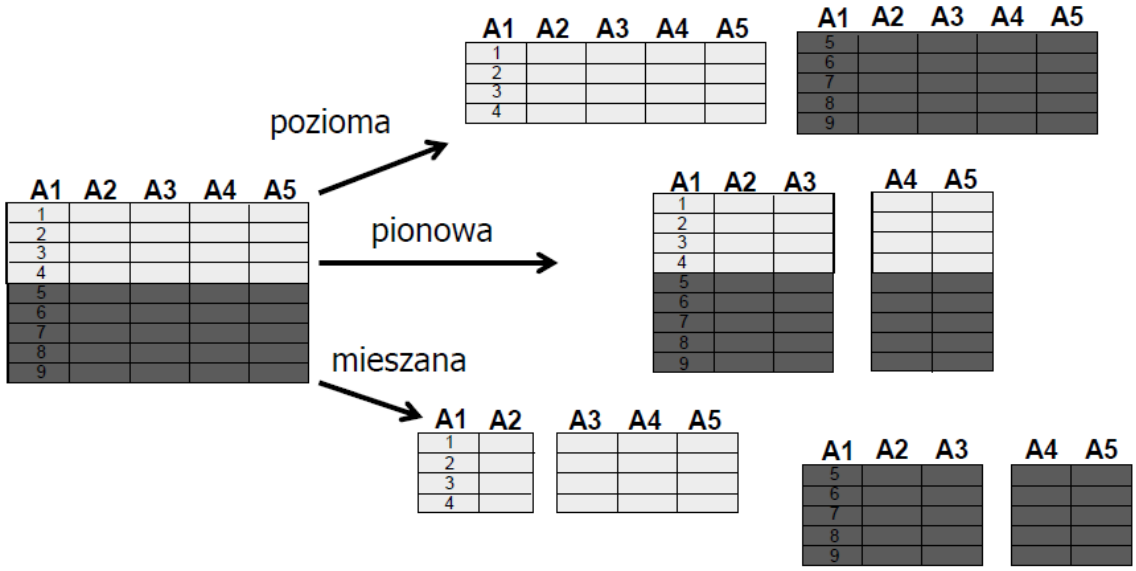
Jeżeli fragmenty zostaną umieszczone „blisko” miejsca ich wykorzystania, wówczas można zredukować koszty transmisji sieciowej w sieciach o niskiej przepustowości.

W praktyce wyróżnia się trzy metody fragmentacji:

- fragmentację poziomą,

- fragmentację pionową,

- fragmentację mieszaną.



Fragmentacja pozioma

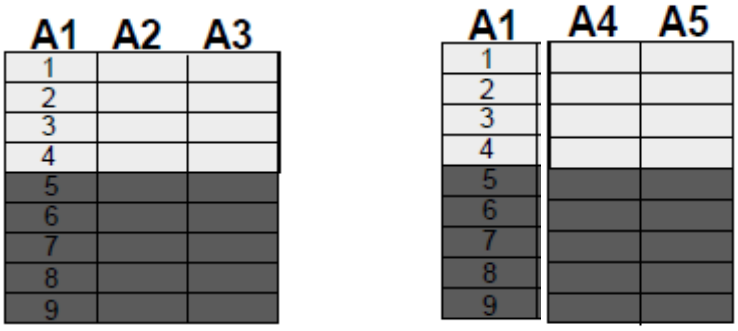
Fragmentacja pozioma (ang. horizontal fragmentation) umożliwia podział zbioru rekordów tabeli na mniejsze podzbiory, z których każdy jest opisany identyczną liczbą atrybutów (podział poziomy tabeli na odrębne fragmenty).

Wybór fragmentu (partycji) do którego trafi rekord jest realizowany na podstawie wartości jednego lub kilku wybranych atrybutów tabeli – tzw. atrybutów fragmentujących (partycjonujących, np. atrybutu wiek od … do ..).

Fragmentacja pionowa

Fragmentacja pionowa (ang. vertical fragmentation) umożliwia podział tabeli w pionie, tj. jej rozbicie na fragmenty złożone z podzbiorów atrybutów tabeli pierwotnej. Każdy fragment zawiera identyczną liczbę rekordów.

Dany atrybut A może się znaleźć tylko w jednym fragmencie. Nie dotyczy to atrybutów kluczowych, np. A1, które są dodawane do każdego fragmentu w celu umożliwienia ich łączenia.



Fragmentacja mieszana

Fragmentacja mieszana (ang. hybrid fragmentation) stanowi połączenie fragmentacji poziomej i pionowej.

Występuje w dwóch wariantach:

- w pierwszym, tabela jest najpierw dzielona poziomo, a następnie wszystkie lub wybrane jej fragmenty są dalej dzielone pionowo (poziom – pion);

- w drugim, tabela jest najpierw dzielona pionowo, a następnie wszystkie lub wybrane jej fragmenty są dalej dzielone poziomo (pion – poziom).

Kryteria poprawności fragmentacji

Poprawna fragmentacja musi spełniać trzy następujące kryteria:

- kompletności,

- rozłączności,

- rekonstrukcji.

Kryterium kompletności

Kompletność (ang. completeness) oznacza, że jeżeli tabela T została podzielona na partycje TP_1, TP_2, ..., TP_n, to każdy rekord z T lub jego fragment musi się znaleźć w jednej z partycji TP_1, TP_2, ..., lub TP_n.

W przypadku fragmentacji poziomej każdy rekord znajduje się w odrębnej partycji, natomiast w przypadku fragmentacji pionowej każdy fragment.

Kryterium to gwarantuje, że na skutek partycjonowania żadne dane z tabeli pierwotnej T nie zostaną utracone.

Kryterium rozłączności

Rozłączność (ang. disjointness) oznacza, że jeżeli tabela T została podzielona na partycje TP_1, TP_2, ..., TP_n, to każdy rekord z T lub jego fragment nie może się znaleźć w więcej niż jednej partycji.

Kryterium to gwarantuje, że na skutek partycjonowania w bazie danych nie pojawią się dane nadmiarowe.

Wyjątkiem od tej reguły jest *partycjonowanie pionowe*, w którym wartości atrybutów stanowiących klucz podstawowy tabeli występują w każdej partycji.

Kryterium rekonstrukcji

Rekonstrukcja (ang. reconstruction) oznacza, że musi istnieć możliwość zrekonstruowania pierwotnej tabeli T ze wszystkich jej partycji.

Operacja rekonstrukcji nie może doprowadzić ani do utraty żadnych danych, ani do powstania danych nadmiarowych.

- W przypadku partycjonowania poziomego, rekonstrukcji pierwotnej tabeli dokonuje się z wykorzystaniem operatora sumy zbiorów rekordów znajdujących się we wszystkich partycjach.

- W przypadku partycjonowania pionowego, rekonstrukcję realizuje się za pomocą łączenia partycji wykorzystując do tego klucz podstawowy tabeli pierwotnej.

Podstawowe algorytmy fragmentacji poziomej

Fragmentację poziomą implementuje się głównie z wykorzystaniem algorytmów:

- round-robin,

- bazujących na wartości: zakresowy i hash’owy.

Algorytm round-robin rozmieszcza rekordy cyklicznie we wszystkich węzłach systemu.

W przypadku algorytmów bazujących na wartości, tj. zakresowego i hash'owego, metoda rozmieszczania danych w węzłach sieci zależy od wartości samych danych.

Algorytm round-robin

Algorytm round-robin umożliwia równomierne rozpraszanie danych w węzłach sieci.

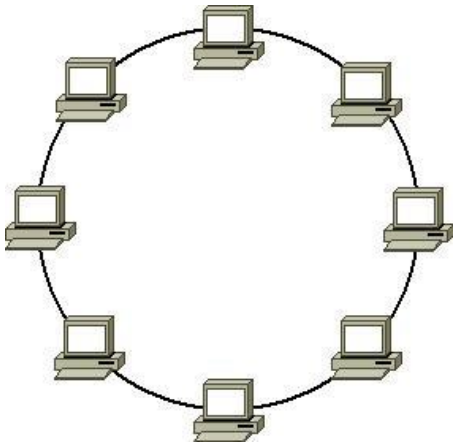
Przykładowo, jeśli w sieci znajdują się trzy węzły, to:

- pierwszy rekord tabeli zostanie umieszczony w węźle pierwszym,

- drugi - w węźle drugim,

- trzeci rekord - w węźle trzecim,

- czwarty — znów w węźle pierwszym itp.



Ponieważ dane są rozpraszane w sposób przypadkowy, więc odnalezienie żądanych rekordów wymaga przeszukania wszystkich węzłów, co jest wadą tego rozwiązania.

Algorytm zakresowy

W algorytmie zakresowym, każdy węzeł przechowuje dane o wartościach atrybutu fragmentującego z zadanego zakresu lub o zadanej wartości.

Na rysunku przedstawiono trzy węzły BD1, BD2, BD3.

Pierwszy z nich zawiera fragment tabeli faktury, przechowujący faktury z lat od 2000 do 2002.

Drugi węzeł zawiera fragment tabeli przechowujący faktury z lat 2003-2005, a trzeci - zawiera fragment tabeli przechowujący faktury z roku 2006.

Algorytm zakresowy

W przypadku algorytmu fragmentacji haszowej, dane są umieszczane w węzłach zgodnie z wartością systemowej funkcji haszowej H(x).

Argumentem wejściowym tej funkcji jest wartość atrybutu, a jej wynikiem — adres węzła, w którym zostanie umieszczony rekord, N = H(x).

W celu odnalezienia żądanych rekordów i ustalenia miejsca składowania danych system zarządzania rozproszoną bazą danych wykorzystuję tę samą funkcję haszową H(x), która została wykorzystana do fragmentacji danych.

Zaletą tej metody jest możliwość automatycznego umieszczania w tym samym węźle rekordów pochodzących z różnych, powiązanych z sobą tabel. W ten sposób zwiększa się efektywność wykonywania operacji łączenia tabel, gdyż łączone z sobą rekordy znajdują się w tym samym węźle.

Problem efektywnej alokacji – rozmieszczenia danych

Dane w systemie RBD mogą być składowane w dowolnym jego węźle, więc zachodzi konieczność takiego rozmieszczenia danych (np. tabel, fragmentów, indeksów), który zapewni największą efektywność całego systemu.

Jest to tzw. problem alokacji.

W ogólnym przypadku dane powinny być rozmieszczane blisko miejsca ich wykorzystania.

Problem alokacji można zdefiniować następująco:

- dla danego zbioru fragmentów $F = \{F_1, F_2, ..., F_n\}$

- danego zbioru węzłów systemu rozproszonej bazy danych $W = \{W_1, W_2, ..., W_n\}$, w których działa zbiór aplikacji

- użytkowników $A = \{A_1, A_2, ..., A_k\}$ należy znaleźć,

- optymalny przydział fragmentów do węzłów uwzględniający wybrane kryteria.

Jest to, więc problem optymalizacji wielokryterialnej.

Wybrane kryteria optymalizacji

Optimum przydziału może być zdefiniowane w kontekście łącznego kosztu i efektywności systemu.

Łączny koszt uwzględnia:

- koszt przechowywania fragmentu F_i w węźle W_j,

- koszt odczytu fragmentu F_i z węzła W_j,

- koszt zmodyfikowania fragmentu F_i w węźle W_j,

- koszt komunikacji z węzłem W_j.

Efektywność systemu jest natomiast mierzona przepustowością w każdym z jego węzłów.

Odpowiednie algorytmy alokacji nie będą omawiane.

6.2. Replikacja danych

Replikacja obiektów bazy danych to proces ich powielania w węzłach rozproszonej bazy danych. Obiektami bazy danych, które podlegają replikacji mogą być:

- dane zawarte w tabelach,
- instrukcje SQL,
- inne elementy schematu bazy danych (perspektywy, procedury, typy definiowane, wyzwalacze, itp.).

- Podstawowy mechanizm replikacji dotyczy powielania danych zapisanych w tabelach, co może służyć do tworzenia lokalnych kopii danych w celu zwiększenia bezpieczeństwa i odporności systemu na awarie, a także umożliwienia dostępu do wybranego fragmentu bazy danych w zdalnej lokalizacji.

- Replikacja danych może dotyczyć tylko części bazy danych.

Jest ona zazwyczaj implementowana w postaci „migawki danych”, która stanowi kopię danych, które pochodzą z wybranych fragmentów tabel źródłowych.

- W bardziej zaawansowanych systemach replikacji oprócz tabel mogą być powielane również instrukcje (DDL i DML), a także inne obiekty schematu bazy danych (np. perspektywy, procedury i funkcje, typy zdefiniowane przez użytkownika). Obiekty te są następnie implementowane w zdalnych bazach danych, tak aby w węzłach tych znajdowały się identyczne zbiory danych oraz identyczne schematy. W szczególności, operacje DDL oraz DML (insert, update, delete), jakie zostaną wykonane w węzłach źródłowych są przesyłane i aplikowane w węzłach replikowanych.

Możliwe jest tworzenie zaawansowanych systemów replikacji, w których zmiany dokonane w jednym węźle są propagowane do:

- wszystkich węzłów podrzędnych: replikacja typu master-slave;

- wszystkich pozostałych węzłów: replikacja multimaster (N-peer-to-peer).

W zależności od systemu bazodanowego (MySQL, PostgreSQL, MS SQL, Oracle) mogą być replikowane różne typy obiektów bazy danych.

Należą do nich:

- tabele,
- indeksy,
- perspektywy i migawki,
- procedury i funkcje,
- wyzwalacze,
- synonimy,
- typy zdefiniowane przez użytkowników,
- typy indeksowe,
- operatory zdefiniowane przez użytkownika,
- i inne zależne od systemu.

Sposób propagacji zmian

Zmiany dokonane w węzłach systemu replikacji mogą być propagowane w sposób:

- synchroniczny, tj. natychmiast po wykonaniu operacji;

- asynchroniczny, tj. odświeżenie zawartości węzłów następuje, co pewien określony czas.

W przypadku replikacji asynchronicznej może dochodzić do sytuacji, w których lokalne zaaplikowanie operacji z węzła zdalnego nie jest możliwe, gdyż mogłoby to spowodować utratę spójności lokalnego schematu węzła, lub nie istnieją dane w węźle, których dotyczy zdalna operacja (np. zostały one usunięte przez inną, lokalną transakcję).

Sytuacje takie prowadzą do występowania konfliktów replikacji, które pojawiają się wówczas, gdy lokalne operacje użytkowników w węzłach replikacji dotyczą tych samych obiektów bazy danych (np. wstawianie danych do tabel, usuwanie tabel lub wierszy). Operacje te po zaaplikowaniu w węzłach zdalnych mogą powodować utratę zgodności replik, tj. utratę spójności.

Możliwe sytuacje konfliktowe:

- *konflikt aktualizacji*; transakcje pochodzące z różnych węzłów środowiska, dokonują zmian tego samego rekordu w replikach, ustawiając inne wartości atrybutów;

- *konflikt unikalności*; replikacja rekordu powoduje naruszenie unikalności klucza podstawowego lub klucza unikalnego (obcego) w węzłach docelowych; np. wstawienie do tej samej tablicy różnych rekordów o tym samym id);

- *konflikt usunięcia*; jedna transakcja usuwa rekordy z replikowanej tabeli, podczas gdy w tym samym momencie są one modyfikowane lub usuwane przez transakcję z innego węzła.

W przypadku replikacji synchronicznej nie dochodzi do konfliktów, gdyż sama operacja przesyłania zmian do węzłów zdalnych jest częścią lokalnej transakcji, która te zmiany wprowadza. Problemem jest natomiast niedostępność węzła zdalnego. Wówczas następuje zawieszenie wszystkich lokalnych operacji w pozostałych węzłach środowiska replikacji.

Stosowanie replikacji:

- zmniejsza ryzyko utraty danych;

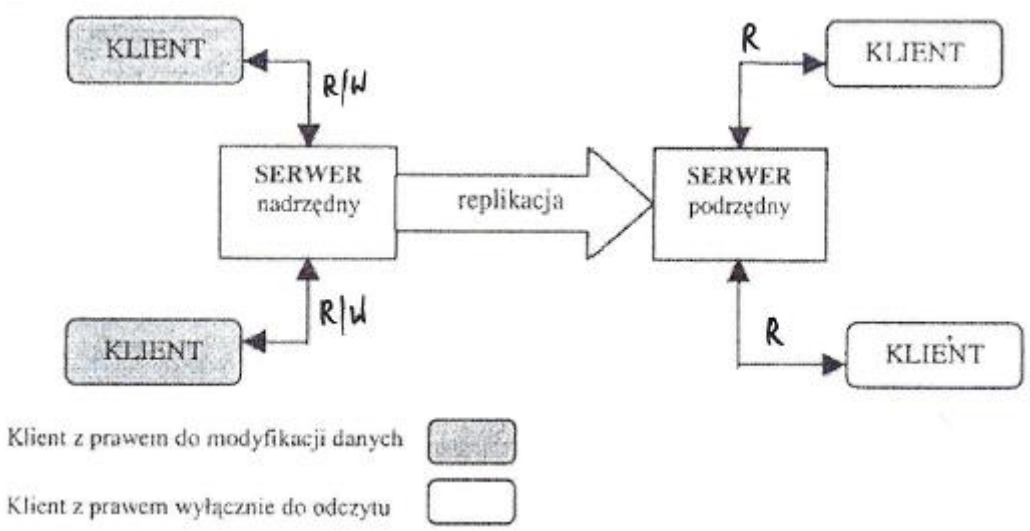
- zapewnia ciągłość pracy w przypadku wystąpienia awarii lub czasowej niedostępności jednego z węzłów dostępne są alternatywne źródła danych – zwiększenie niezawodności i dostępności systemu;

- pozwala na odciążenie łączy w przypadku, gdy zachodzi konieczność komunikacji pomiędzy oddalonymi od siebie węzłami sieci;

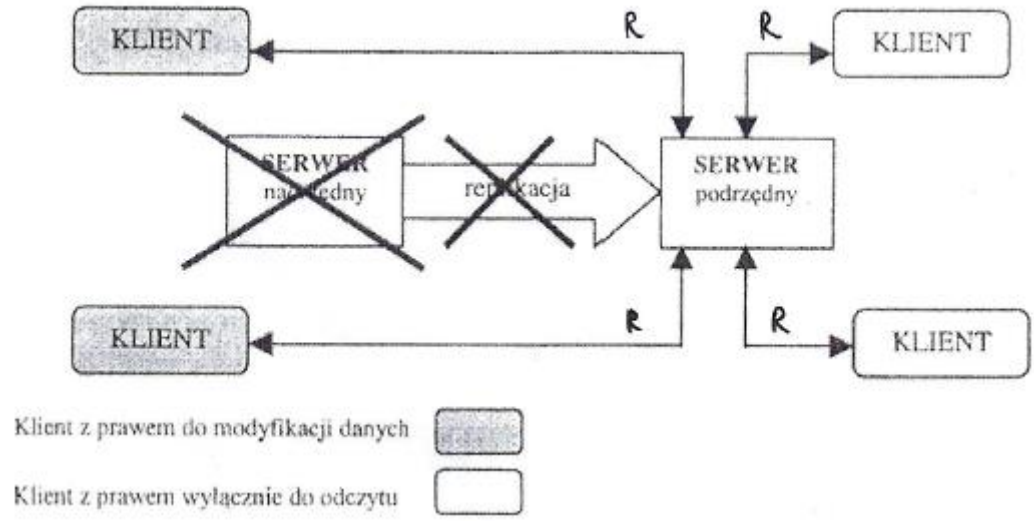
- skraca czas dostępu do danych (istnienie replik uniezależnia nas od przepustowości i aktualnego obciążenia sieci, a także wydajności zdalnych węzłów i ich aktualnego obciążenia);

- pozwala na rozłożenie obciążenia systemu bazodanowego, wynikającego z pracy użytkowników, pomiędzy poszczególne węzły środowiska replikacji.

Replikacja umożliwia realizację mechanizmu przetwarzania danych tolerującego uszkodzenia. Na przykład, w układzie dwóch serwerów, w którym dane z serwera nadrzędnego są replikowane do serwera podrzędnego, awaria serwera nadrzędnego spowoduje przejęcie operacji czytania danych przez serwer podrzędny.



Struktura systemu po awarii serwera nadrzędnego



Za pomocą mechanizmu replikacji można tworzyć w Internecie serwery lustrzane, które są zlokalizowane po stronie odległych klientów.

Rozproszone bazy danych mają również pewne wady, wynikające z rozproszenia i replikacji danych. W szczególności:

- katalog systemowy bazy danych musi zawierać dodatkowe informacje o rozmieszczeniu fragmentów bazy oraz sposobie replikacji;

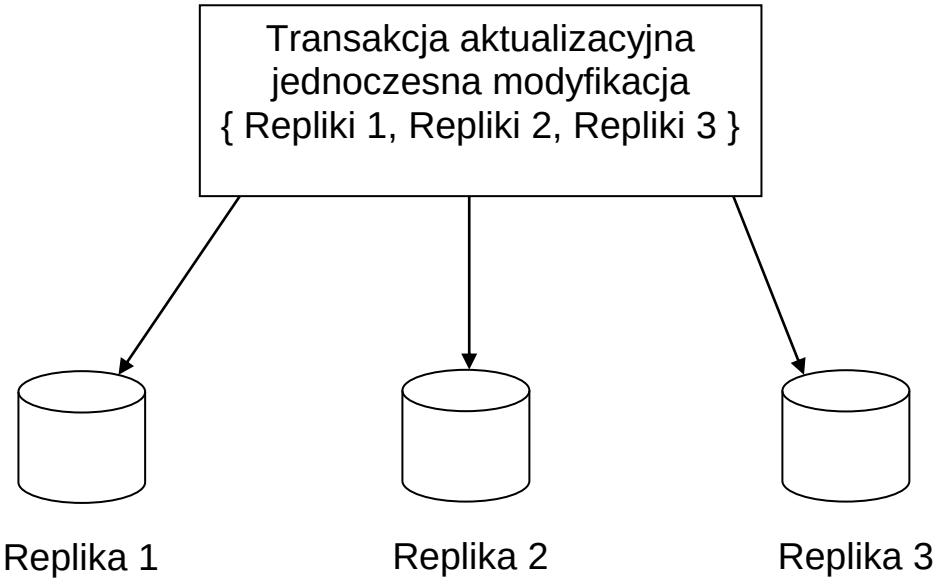
- komplikuje się sterowanie współbieżnością wykonywania operacji;

- komplikuje się testowanie bazy danych oraz wykrywanie i likwidacja awarii;

- pojawia się problem z aktualizacją replik; w idealnym przypadku aktualizacja obiektów bazy powinna następować równocześnie.

Różne warianty aktualizacji replik

W przypadku pojawienia się zmiany w dowolnym węźle systemu replikacji powinna ona w idealnym przypadku być zaaplikowana jednocześnie we wszystkich pozostałych kopiach, co w praktyce jest dość trudne.



Wymaganie jednoczesnej aktualizacji kopii w wielu przypadkach nie daje się zrealizować z kilku powodów:

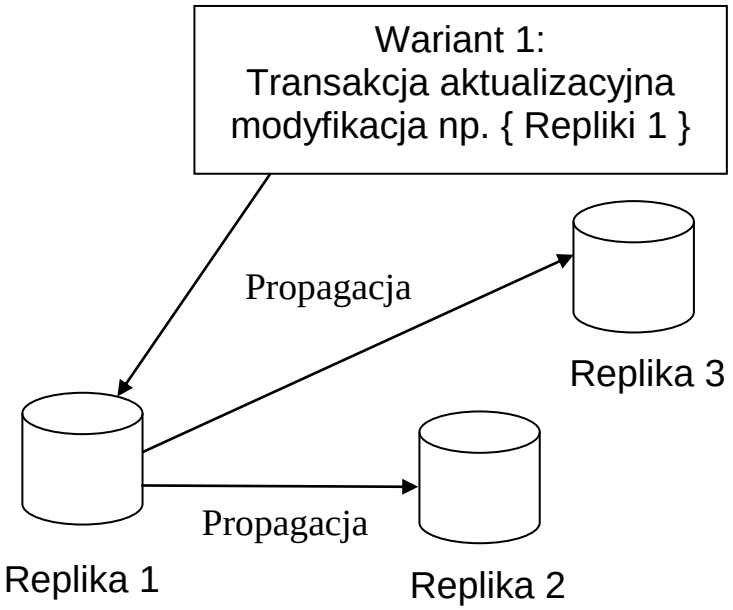
- opóźnienia wynikające z czasu transmisji zleceń aktualizacyjnych powodują, że kopie są chwilowo wzajemnie niezgodne (niespójne);

- awarie łączy transmisji danych oraz węzłów sieci mogą powodować czasową niezdolność do wykonania zleceń aktualizacyjnych;

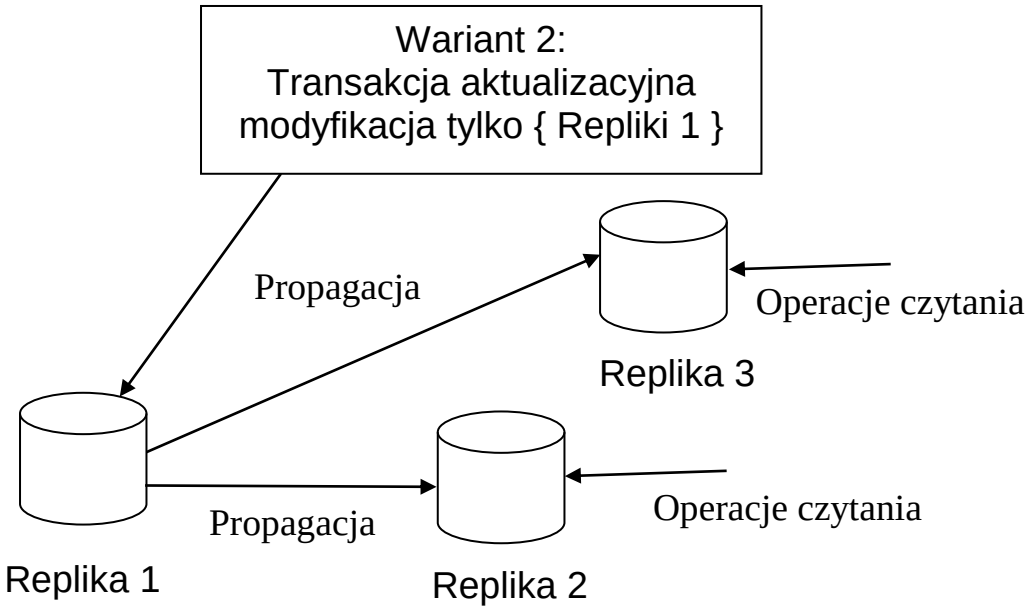
- koszt transmisji zleceń aktualizacyjnych może być zbyt wysoki w przypadku częstych zmian.

Stosuje się wówczas inne podejścia, osłabiające wymaganie jednoczesnej aktualizacji wszystkich kopii:

- wariant 1: zmiana następuje w wybranym węźle, w jednej wybranej kopii, a następnie jest ona propagowana do pozostałych replik;



- wariant 2: tylko jedna z replik może być modyfikowana, a pozostałe są tylko do czytania; aktualizacja kopii do czytania następuje, co pewien z góry określony czas (np. co godzinę, raz na dzień, itp.).



6.3. Rodzaje replikacji w wybranych systemach baz danych

Replikacja danych polega na skopiowaniu pewnego zbioru danych z jednego miejsca, tzw. źródła, do miejsca docelowego. W kontekście relacyjnych baz danych źródłem danych jest tabela, którą nazywamy **tabelą źródłową**. Obiektem docelowym jest również tabela, która jest nazywana **repliką**.

Replikacja jest obsługiwana przez najbardziej popularne systemy zarządzania bazami danych:

- MySQL,
- PostgreSQL,
- MS SQL Server,
- Oracle.

Środowisko Oracle dostarcza kilka mechanizmów replikacji danych, które dzielą się na następujące grupy:

- replikacja podstawowa (standardowa migawkowa);
- replikacja zaawansowana (multimaster, migawkowa, hybrydowa);
- replikacja wykorzystująca strumieniowe przetwarzanie danych.

Pierwsze dwie grupy replikacji wymagają utworzenia połączeń między węzłami systemu. Węzły komunikują się ze sobą za pomocą specjalnego obiektu schematu, nazywanego ***łącznikiem*** bazy danych.

Przed wyborem odpowiedniego mechanizmu replikacji należy zastanowić się nad charakterystycznymi elementami mechanizmu, takimi jak:

- Wielkość replikowanych danych - wskazuje jak duża jest zawartość pojedynczej repliki. Jednostkę replikacji określa się w momencie definiowania repliki.

- Zakres replikacji bazy danych. Wyróżnia się w pełni replikowane bazy danych (ang. fully replicated databases) oraz częściowo replikowane bazy danych (ang. partially replicated databases).

  W pierwszym przypadku każdy węzeł zawiera wszystkie dane z pozostałych węzłów. Bazy takie cechują się wysoką efektywnością wykonywania zapytań, ponieważ każde zapytanie może zostać wykonane lokalnie. Wadą tego typu baz danych jest duży narzut czasowy na odświeżanie replik, dlatego w praktyce stosuje się je rzadko.

  Znacznie częściej wykorzystuje się częściowo replikowane bazy danych, w których każdy węzeł zawiera wybrane dane z innych węzłów. Tego typu bazy danych cechuje mniejsza wydajność od poprzednich baz danych, ponieważ część zapytań wymaga sięgnięcia do zdalnych węzłów. Jednak narzut czasowy na odświeżanie replik jest mniejszy.

- Wybór momentu odświeżania - synchroniczne lub asynchroniczne.

  Odświeżanie synchronicznie polega na propagowaniu danych ze źródła do repliki natychmiast po dokonaniu zmian w tabelach źródłowych, z wykorzystaniem transakcji rozproszonej. Odświeżanie asynchroniczne polega na propagowaniu zmian, co pewien okres czasu.

- Typ odświeżania – pełne lub przyrostowe.

  Odświeżanie pełne polega na każdorazowym przesyłaniu ze źródła do repliki wszystkich danych, które replika udostępnia. Ta technika odświeżania charakteryzuje się łatwością implementacji i dużym kosztem odświeżania ze względu na dużą ilość przesyłanych danych.

  Odświeżanie przyrostowe polega na przesyłaniu do repliki wyłącznie zmian dokonanych w źródle od czasu ostatniego odświeżenia. Ta technika cechuje się trudniejszą implementacją i niższymi kosztami odświeżania.

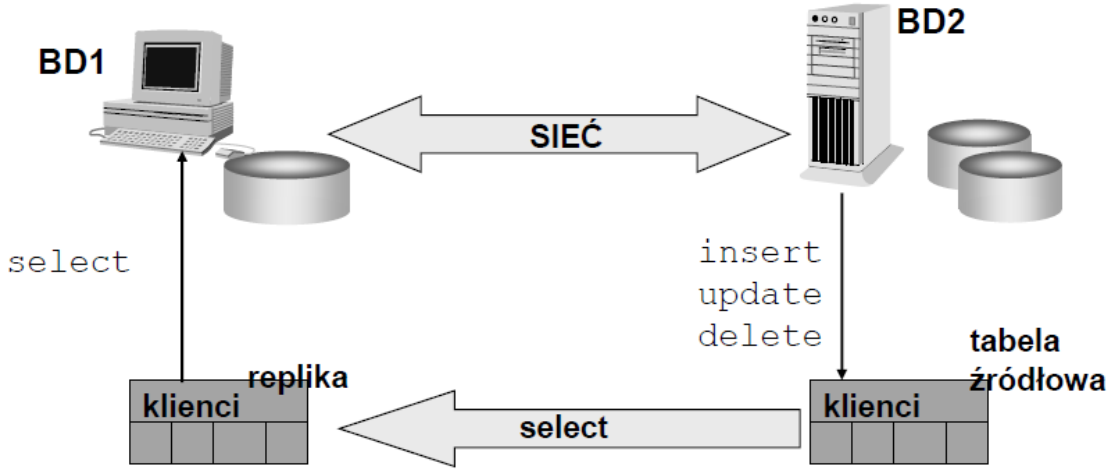
6.4. Standardowa replikacja migawkowa w Oracle

Mechanizmem wykorzystywanym w Oracle do replikacji danych jest migawka (ang. snapshot), nazywana też widokiem zmaterializowanym (ang. materialized view).

W przypadku replikacji standardowej migawka składa się z:

· Nazwy migawki;
· Momentu wypełnienia migawki danymi;
· Sposobu odświeżania;
· Typu migawki;
· Zapytania SQL określającego zakres danych w migawce.

Migawkę w systemie Oracle tworzy się poleceniami SQL *create snapshot* lub *create materialized view*.



Standardowa replikacja polega na utworzeniu kopii zdalnej tabeli (np. klienci@BD2) w bazie lokalnej BD1.

Tabela będąca kopią nazywa się repliką, a tabela na podstawie, której utworzono replikę nazywa się tabelą źródłową (nazywaną również tabelą master lub tabelą bazową).

Replika może zawierać wszystkie atrybuty i rekordy tabeli źródłowej lub ich podzbiór.

W architekturze standardowej replikacji replika jest tylko do odczytu.

Replika posiada cechę automatycznego odświeżania, tzn. zmiany zawartości tabeli źródłowej propagują się do repliki automatycznie.

Replika jako migawka

W systemie Oracle replika jest implementowana jako:

- migawka (ang. snapshot), zwana również perspektywą zmaterializowaną (ang. materialized view).

Definicja migawki zawiera zapytanie do tabeli źródłowej. Wynik tego zapytania jest dostępny w migawce.

Użytkownik tworzący migawkę musi posiadać odpowiednie uprawnienia systemowe (m.in. CREATE SNAPSHOT lub CREATE MATERIALIZED VIEW).

Migawka jest fizycznie przechowywana jako tabela z indeksem.

Z migawką jest zwykle stowarzyszony proces systemowy odpowiedzialny za jej odświeżanie.

Definicja migawki

W standardowym przypadku definicja migawki składa się z następujących elementów:

- nazwy migawki;

- momentu wypełnienia migawki danymi;

- specyfikacji sposobu odświeżania;

- specyfikacji momentu rozpoczęcia automatycznego odświeżania;

- typu migawki;

- zapytania określającego zakres danych dostępnych w migawce.

```
create snapshot nazwa_migawki

build moment_wypełnienia_danymi

refresh sposób_odświeżania

start with data_rozpoczęcia_odświeżania

next okres_odświeżania

with identyfikacja_rekordów

as zapytanie;
```

- Migawkę tworzy się poleceniem SQL **create snapshot** lub **create materialized view**, oba o identycznej składni.
- Klauzula **build** określa moment pierwszego wypełnienia migawki danymi.
- Klauzula **refresh** określa sposób odświeżania.
- Klauzula **start with** określa datę i czas pierwszego odświeżenia po utworzeniu migawki.
- Klauzula **next** określa okres odświeżania mierzony np. w dniach, godzinach, lub minutach.
- Klauzula **as** zawiera zapytanie do tabel źródłowych.

Na kolejnych slajdach podano szczegółowe objaśnienie poszczególnych klauzul.

Klauzula BUILD

Klauzula BUILD wskazuje moment wypełnienia migawki danymi, po jej utworzeniu poleceniem create snapshot (create materialized view).

Można w niej wyspecyfikować albo IMMEDIATE albo DEFERRED.

W pierwszym przypadku (IMMEDIATE) migawka jest wypełniana danymi zaraz po jej utworzeniu.

W drugim przypadku (DEFERRED), migawka jest wypełniana danymi po upłynięciu czasu (od jej utworzenia) określonego wyrażeniem w klauzuli START WITH.

Klauzula REFRESH

Klauzula REFRESH określa sposób odświeżania migawki.

Można w niej wyspecyfikować jedno z trzech słów kluczowych:

- FAST,
- COMPLETE,
- FORCE.

FAST oznacza odświeżanie przyrostowe. W tym przypadku z tabeli źródłowej do repliki są przesyłane tylko zmiany dokonane od czasu ostatniego odświeżania.

COMPLETE oznacza odświeżanie pełne. W tym przypadku z tabeli źródłowej są przesyłane wszystkie dane spełniające warunki zapytania definiującego replikę.

FORCE umożliwia automatyczne wybranie sposobu odświeżania, z preferencją odświeżania przyrostowego (jeśli takie jest możliwe).
Decyduje o tym system.

Klauzula START WITH

Klauzula START WITH określa czas po jakim migawka zostanie wypełniona danymi, po jej utworzeniu.

Klauzula ta obowiązuje (jest uwzględniana) **tylko** jeśli wyspecyfikowano klauzulę BUILD DEFERRED.

Klauzula NEXT

Klauzula NEXT określa czas po jakim replika jest ponownie odświeżana, przy czym czas ten jest mierzony od momentu zakończenia poprzedniego odświeżenia.

Jeśli nie wyspecyfikowano tej klauzuli, wówczas migawka nie jest odświeżana automatycznie. Wówczas, można ją odświeżyć ręcznie.

Migawkę można zawsze odświeżyć "ręcznie" za pomocą odpowiedniej procedury systemowej.

Procedury „ręcznego” odświeżania migawek

Migawkę można odświeżyć „ręcznie” zawsze, nawet gdy jest odświeżana automatycznie.

Jeśli jednak nie wyspecyfikowano klauzuli NEXT, wówczas jedynym sposobem odświeżenia jest odświeżenie „ręczne” za pomocą procedur systemowych REFRESH, znajdujących się w pakietach systemowych DBMS_SNAPSHOT lub DBMS_MVIEW (można je stosować zamiennie):

- DBMS_SNAPSHOT.REFRESH
- DBMS_MVIEW.REFRESH .

Składnia wywołania procedury REFRESH jest następująca:

DBMS_SNAPSHOT.REFRESH('sn_1, sn_2, ..., sn_k', 'metoda')

– sn_1, sn_2, ..., sn_k: migawki;

– metoda: metoda odświeżania
- f lub F :FAST ;
- c lub C :COMPLETE;
- ? :sposób domyślny ustalony podczas definiowania migawki.

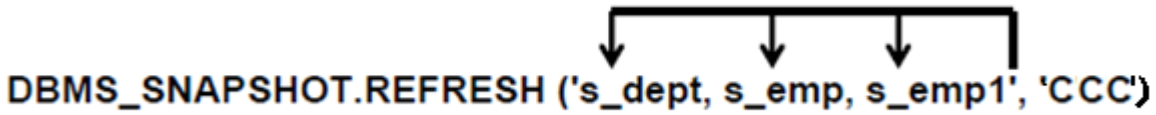
Przyjmuje ona dwa argumenty wywołania.

Pierwszy jest zbiorem nazw migawek do odświeżenia. Poszczególne nazwy są oddzielone przecinkami. Zbiór ten jest przekazywany do procedury jako ciąg znaków.

Drugim argumentem jest metoda odświeżenia. Argument ten może przyjąć wartość:
- f lub F oznaczającą odświeżanie przyrostowe,
- c lub C oznaczającą odświeżanie pełne,
- ? oznaczającą odświeżanie takie jak zdefiniowano tworząc migawkę.

Przykładowe wywołania procedury odświeżania migawek.

• Przykład 1

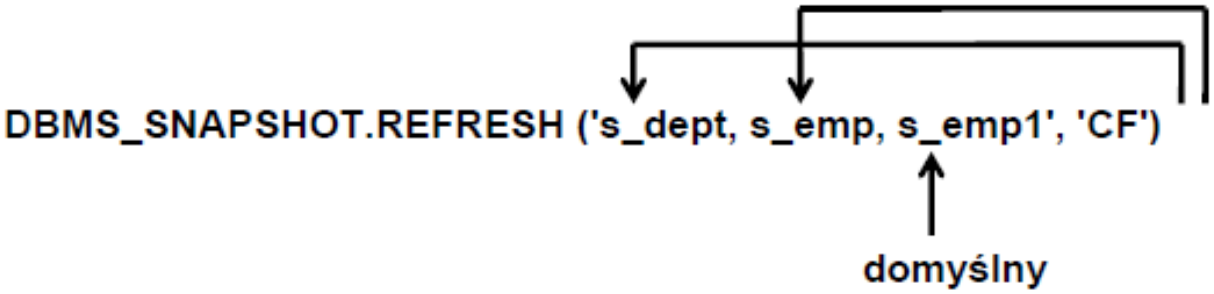


W przykładzie 1 są odświeżane trzy migawki o nazwach

s_dept, s_emp, s_emp1,

każda z nich jest odświeżana w trybie C, czyli pełnym.

• Przykład 2



W przykładzie 2 są odświeżane te same trzy migawki.

Pierwsza z nich w trybie C (pełnym), druga w trybie F (przyrostowym),
a trzecia w trybie domyślnym (określonym w momencie tworzenia migawki).

Klauzula WITH

Klauzula ta określa sposób identyfikowania rekordów w tabeli źródłowej i w migawce.

Możliwe są tu dwa sposoby identyfikacji.

- ROWID
- PRIMARY KEY.

Pierwszy z nich wykorzystuje fizyczne adresy rekordów na dysku - ROWID, gdzie ROWID zawiera w sobie adres pliku, numer bloku w ramach tego pliku i adres rekordu w bloku.

Jest on wykorzystywany tylko w przypadku odświeżania przyrostowego (F).

Drugi z nich wykorzystuje do identyfikacji rekordów źródłowych i migawki wartości klucza podstawowego tabeli źródłowej.

Począwszy od wersji Oracle (rel. 8.0) udostępnione zostały migawki typu primary key, jako domyślny typ.

Stosowanie tego typu migawek jest zalecane przez producenta oprogramowania.

Klauzula AS

Klauzula AS umożliwia zdefiniowanie zapytania sięgającego do tabel źródłowych.

Wynik tego zapytania jest materializowany w migawce.

Z punktu widzenia złożoności tego zapytania, migawki dzieli się na:

- proste,
- złożone.

Migawki proste

Migawka prosta posiada następujące cechy:

- jej zapytanie odwołuje się do jednej tabeli źródłowej;

- jej zapytanie nie wykorzystuje klauzul GROUP BY, CONNECT BY, DISTINCT;

- jej zapytanie nie wykorzystuje wyrażeń, funkcji SQL, połączeń, operatorów zbiorowych.

Migawkę prostą daje się **zawsze odświeżać przyrostowo.**

Migawki złożone

Migawka złożona to taka, która nie spełnia przynajmniej jednego warunku dotyczącego złożoności zapytania migawki prostej.

Migawki złożone **nie zawsze** dają się odświeżać przyrostowo.

Zależy to od stopnia złożoności zapytania. W wielu przypadkach jedynym sposobem odświeżenia migawki złożonej jest odświeżanie pełne.

Przykłady tworzenia migawek prostych

```
create materialized view mv_sprzedaz1
build immediate
refresh complete
next sysdate+(1/(24*60*6))
as
select * from sprzedaz@lab92
where data like '%2003';
```

Przykładowe polecenie create materialized view `mv_sprzedaz1` definiuje migawkę o nazwie mv_sprzedaz1 o następującej charakterystyce:

- wypełnienie danymi w momencie tworzenia (klauzula build immediate);

- odświeżanie pełne (klauzula refresh complete);

- okres odświeżania - 10 sekund (next sysdate+(1/(24*60*6)));

- identyfikacja rekordów za pomocą klucza podstawowego (domyślna, niewyspecyfikowana klauzula with primary key);

- udostępnia informacje o sprzedaży produktów w roku 2003 z tabeli wskazywanej łącznikiem o nazwie lab92.

```
create materialized view mv_sprzedaz2
build deferred
refresh force
start with sysdate+(1/(24*60))
next sysdate+(1/(24*30))
with rowid
as
select * from sprzedaz@lab92
where data like '%2004';
```

Przykładowe polecenie create materialized view `mv_sprzedaz2` definiuje migawkę o nazwie mv_sprzedaz2 o następującej charakterystyce:

- wypełnienie danymi z opóźnieniem (klauzula build deferred);

- odświeżanie wybierane automatycznie przez system (klauzula refresh force);

- moment rozpoczęcia odświeżania automatycznego - 1 minuta po utworzeniu migawki (klauzula start with sysdate+(1/(24*60)));

- okres odświeżania - 2 minuty (next sysdate+(1/(24*30));

- identyfikacja rekordów za pomocą fizycznych adresów (klauzula with rowid)

- udostępnia informacje o sprzedaży produktów w roku 2004 z tabeli wskazywanej łącznikiem o nazwie lab92.

Warunki odświeżania przyrostowego

Przyrostowe odświeżanie migawki jest możliwe jeśli:

- migawka jest typu prostego;

- każda z tabel źródłowych migawki posiada zdefiniowany dziennik migawki;

- lub

- migawka jest typu złożonego, ale umożliwiającego odświeżanie przyrostowe;

- każda z tabel źródłowych migawki posiada zdefiniowany dziennik migawki.

Odświeżanie przyrostowe migawki złożonej

Jeżeli migawka wylicza **agregaty**, m.in. count, sum, avg, variance, stdev, wówczas:

1. dziennik migawki musi zostać utworzony z klauzulą including new values,

2. dziennik migawki musi przechowywać wszystkie atrybuty wymienione w klauzuli select, również te, które są argumentami wywołania funkcji grupowych,

3. zapytanie musi zawierać dodatkowo funkcję count, jeśli są wyliczane wartości funkcji sum, avg, variance, stdev.

Przykład odświeżania przyrostowego migawki złożonej

```
create snapshot mv_suma_sprzedazy
build immediate
refresh fast
next sysdate+(1/(24*60*30))
as
select sklep_id, produkt_id,
sum(l_sztuk), sum(l_sztuk*cena_jedn),
count(l_sztuk),
count(l_sztuk*cena_jedn),
count(*)
from sprzedaz@lab92
group by sklep_id, produkt_id;
```

W przykładzie zapytanie definiujące migawkę zostało wzbogacone o trzy funkcje count, zapewniające możliwość przyrostowego odświeżania tej migawki.

Dziennik migawki

W celu umożliwienia odświeżania przyrostowego każda z tabel źródłowych migawki powinna posiadać zdefiniowany dziennik migawki.

Dziennik migawki (ang. snapshsot log), zwany również dziennikiem perspektywy zmaterializowanej (ang. materialized view log) jest obiektem bazy danych, którego zadaniem jest rejestrowanie operacji modyfikowania zawartości tabeli, dla której go utworzono.

Dziennik jest tworzony dla konkretnej (jednej) tabeli. Jest on wykorzystywany do przyrostowego odświeżania migawki korzystającej z tabeli z dziennikiem.

Implementacyjnie dziennik jest tabelą o nazwie:

MLOG$_nazwa_tabeli_bazowej.

Z dziennikiem są związane systemowe mechanizmy zarządzania jego zawartością, m.in. wstawianie rekordów, usuwanie rekordów wykorzystanych przez migawki.

Tworzenie dziennika migawki

```
create snapshot log
on tabela_bazowa
[with
{    PRIMARY KEY |
     ROWID |
     PRIMARY KEY, ROWID |
     ROWID (lista_kolumn_filtrujących) |
     PRIMARY KEY (lista_kolumn_filtrujących)}]
[{ including new values |
   excluding new values }];
```

Dziennik tworzy się poleceniem CREATE SNAPSHOT LOG lub CREATE MATERIALIZED VIEW LOG, którego ogólną składnię przedstawiono na slajdzie.

Parametr *tabela_bazowa* oznacza tabelę, dla której dziennik jest tworzony.

Pozostałe klauzule i parametry tego polecenia są opcjonalne.

WITH PRIMARY KEY oznacza dziennik dla migawki typu PRIMARY KEY; jest to klauzula domyślna.

WITH ROWID oznacza dziennik dla migawki typu ROWID.

WITH PRIMARY KEY, ROWID oznacza dziennik dla migawek zarówno typu PRIMARY KEY jak i ROWID.

lista_kolumn_filtrujących jest to lista zawierająca atrybuty wykorzystywane w klauzuli WHERE zapytania definiującego migawkę i atrybuty występujące jako argumenty wywołania funkcji grupowych w zapytaniu

INCLUDING NEW VALUES zapewnia przyrostowe odświeżanie migawek wyliczających agregaty.

Przykład tworzenia dziennika migawki

```
create materialized view log
on sprzedaz
with primary key,
rowid (l_sztuk, cena_jedn)
including new values;
```

Przykład zawiera polecenie tworzące dziennik migawki dla tabeli sprzedaz. Dziennik ten jest tworzony zarówno dla migawki typu PRIMARY KEY, jak i ROWID.

Dodatkowo, w dzienniku będą rejestrowane zmiany wartości atrybutów l_sztuk i cena_jedn. Zawartość tak utworzonego dziennika może być wykorzystywana do przyrostowego odświeżania migawki `mv_suma_sprzedazy` ze slajdu 32.

Modyfikowanie migawki

```
alter snapshot nazwa_migawki

refresh sposób_odświeżania

start with data_rozpoczęcia_odświeżania

next okres_odświeżania

with PRIMARY KEY;
```

Istniejącą migawkę można zmodyfikować za pomocą polecenia ALTER SNAPSHOT lub ALTER MATERIALIZED VIEW, którego podstawową składnię przedstawiono na slajdzie.

Za pomocą tego polecenia można zmienić sposób odświeżania, okres odświeżania, zamienić migawkę typu ROWID na PRIMARY KEY (ale nie odwrotnie).

Klauzula `start with` umożliwia wskazanie czasu po którym migawka zostanie odświeżona po zmianie jej definicji.

Przykład pokazuje sposób modyfikowania migawki *m_mig1*.

```
alter snapshot m_mig1

refresh complete

start with sysdate

next

sysdate+1/(24*60*10)

with primary key;
```

Modyfikowanie dziennika migawki

```
alter snapshot log

on tabela_bazowa

add

{ PRIMARY KEY |

  ROWID |

  ROWID (lista_kolumn_filtrujących) |

  primary_key (lista_kolumn_filtrujących)}

[{including new values |

  excluding new values}];
```

Istniejący dziennik migawki można zmodyfikować za pomocą polecenia ALTER SNAPSHOT LOG lub ALTER MATERIALIZED VIEW LOG, którego składnię przedstawiono na slajdzie.

Usuwanie migawki i jej dziennika

• Usuwanie migawki:

```
drop snapshot nazwa_migawki;
```

Migawkę usuwa się poleceniem DROP SNAPSHOT lub DROP MATERIALIZED VIEW LOG wskazując migawkę za pomocą jej nazwy.

• Usuwanie dziennika migawki

```
drop snapshot log on nazwa_tabeli_bazowej;
```

Dziennik migawki usuwa się poleceniem DROP SNAPSHOT LOG ON lub DROP MATERIALIZED VIEW LOG ON wskazując tabelę bazową.

Informacje o migawkach

Informacje o utworzonych przez użytkownika migawkach można uzyskać uruchamiając perspektywę słownikową o nazwie USER_SNAPSHOTS lub USER_MVIEWS.

Najczęściej wykorzystywane atrybuty perspektywy USER_SNAPSHOTS:

- NAME - nazwa migawki;
- TABLE_NAME - nazwa tabeli implementującej migawkę;
- MASTER - nazwa tabeli źródłowej migawki; w przypadku gdy migawka posiada kilka tabel źródłowych, jako wartość MASTER pojawia się nazwa tej tabeli, którą wymieniono jako ostatnią w klauzuli from zapytania definiującego migawkę;
- MASTER_LINK - nazwa łącznika bazy danych, wykorzystywanego do zrealizowania dostępu do ostatniej tabeli wymienionej w klauzuli from zapytania definiującego migawkę;
- UPDATABLE - przyjmuje wartość YES dla migawki modyfikowalnej, w przeciwnym przypadku przyjmuje NO; migawki modyfikowalne są wykorzystywane w tzw. zaawansowanej opcji replikacji;
- REFRESH_METHOD - określa sposób identyfikacji rekordów migawki; przyjmuje wartość PRIMARY KEY dla migawki odświeżanej przyrostowo, zdefiniowanej z klauzulą with primary key; przyjmuje wartość ROWID dla migawki odświeżanej przyrostowo, zdefiniowanej z klauzulą with rowid;
- TYPE - określa sposób odświeżania i może przyjąć jedną z czterech wartości: FAST, FORCE, COMPLETE, NEVER;
- NEXT - wartość wyspecyfikowana w klauzuli next;
- START_WITH - wartość wyspecyfikowana w klauzuli start with;
- REFRESH_GROUP - numer grupy odświeżania, do której należy migawka;
- QUERY - tekst zapytania (klauzula select) definiującego migawkę;

- REFRESH_MODE - przyjmuje jedną z trzech wartości: PERIODIC - jeśli wyspecyfikowano zarówno klauzulę start with, jak i next; COMMIT - jeśli wyspecyfikowano klauzulę on commit; NEVER - jeśli wyspecyfikowano klauzulę never refresh, DEMAND - w pozostałych przypadkach.

Przykładowe zapytanie do perspektywy USER_SNAPSHOTS:

```
select name, table_name, master, master_link,
refresh_method, type from user_snapshots;
```

Informacje o dziennikach migawek

Informacje o utworzonych przez użytkownika dziennikach migawek są dostępne za pomocą perspektywy słownikowej o nazwie USER_SNAPSHOT_LOGS lub USER_MVIEW_LOGS.

Do najczęściej wykorzystywanych atrybutów
perspektywy USER_SNAPSHOT_LOGS należą:

- LOG_OWNER - nazwa użytkownika będącego właścicielem dziennika migawki;
- MASTER - nazwa tabeli, dla której utworzono dziennik;
- LOG_TABLE - nazwa tabeli implementującej dziennik;
- ROWIDS - przyjmuje wartość YES dla dziennika utworzonego z klauzulą with
- ROWID, w przeciwnym przypadku przyjmuje wartość NO;
- PRIMARY_KEY - przyjmuje wartość YES dla dziennika utworzonego z klauzulą with primary key, w przeciwnym przypadku przyjmuje wartość NO;
- FILTER_COLUMNS - przyjmuje wartość YES dla dziennika zawierającego kolumny filtrujące;
- INCLUDE_NEW_VALUES - przyjmuje wartość YES dla dziennika utworzonego z klauzulą include new values, w przeciwnym przypadku przyjmuje wartość NO;

- CURRENT_SNAPTHOTS - data ostatniego odświeżenia migawki z wykorzystaniem zawartości dziennika;

- SNAPSHOT_ID - identyfikator migawki, która korzysta z dziennika.

Przykładowe zapytanie do perspektywy USER_SNAPSHOT_LOGS:

```
select log_owner, master, log_table, rowids,
primary_key, filter_columns, current_snapshots,
snapshot_id from user_snapshot_logs;
```

Parametry określają wyprowadzane kolumny.

Informacje o odświeżaniu migawek

Informacje o odświeżaniu migawek są dostępne za pomocą perspektywy słownikowej o nazwie USER_SNAPSHOT_REFRESH_TIMES lub USER_MVIEW_REFRESH_TIMES.

Poniżej przedstawiono przykładowe zapytanie do perspektywy USER_SNAPSHOT_REFRESH_TIMES.

```
select name,
to_char(last_refresh,'dd.mm.yyyy:hh24:mi:ss')
s last_refresh from user_snapshot_refresh_times;
```

Atrybut last_refresh udostępnia datę i czas ostatniego odświeżenia migawki, której nazwa jest udostępniana atrybutem name.

Grupa odświeżania migawek

Grupa odświeżania (ang. refresh group) jest to obiekt zawierający jedną lub wiele migawek, z których każda jest odświeżana w tym samym momencie.

- Grupa odświeżania zapewnia, że wszystkie umieszczone w niej migawki są odświeżane w tym samym momencie.
- Niejawnie grupa odświeżania jest tworzona przez system dla każdej migawki. Tak więc każda migawka jest umieszczona domyślnie w swojej grupie.
- Grupa odświeżania może się składać z wielu migawek, ale migawka może być umieszczona tylko w jednej grupie.
- Migawka, która nie należy do grupy odświeżania nie jest odświeżana automatycznie (może być odświeżona „ręcznie").

Grupy odświeżania mogą być również tworzone jawnie. Służy do tego celu procedura MAKE pakietu systemowego o nazwie DBMS_REFRESH.

Procedura MAKE posiada 13 argumentów wywołania, z których cztery pierwsze muszą posiadać jawnie wyspecyfikowane wartości. Pozostałe 9 argumentów może przyjąć wartości domyślne.

Definiowanie grupy odświeżania

```
DBMS_REFRESH.MAKE(

name, list, next_date, interval, implicit_destroy, ... )
```

gdzie:

`name` - nazwa tworzonej grupy odświeżania;

`list` - nazwa migawki, lub lista migawek, bądź też zmienna typu tablica PL/SQL zawierająca nazwy migawek;
migawki muszą być w tej samej bazie danych, przy czym w jednej grupie może się znaleźć ograniczona liczba migawek (np. zależy od systemu);

`next_date` - określa datę i czas pierwszego odświeżenia migawek w grupie;

`interval` - określa wspólną dla wszystkich migawek częstotliwość odświeżania;

`implicit_destroy` - pozwala określić zachowanie się grupy w przypadku usunięcia z niej ostatniej migawki; domyślnie argument ten przyjmuje wartość TRUE, co oznacza automatyczne usunięcie grupy w przypadku, gdy usunięto z niej ostatnią migawkę; z kolei wartość FALSE tego argumentu spowoduje zachowanie grupy w systemie.

Przykład definiowania grupy odświeżania

```
exec dbms_refresh.make (name=>'rg_firma',
list=>'mv_sprzedaz', next_date=>sysdate + (1/(24*60*60)),
interval=>'sysdate + (1/(24*60*10))',
implicit_destroy=>FALSE)
```

Polecenie tworzy grupę odświeżania o nazwie `rg_firma`, która zawiera jedną migawkę - `mv_sprzedaz`.

Grupa zostanie odświeżona sekundę po jej utworzeniu i będzie automatycznie odświeżana co 6 sekund.

Po usunięciu ostatniej migawki z grupy nie zostanie ona automatycznie usunięta z systemu.

Dodanie migawki do grupy odświeżania

Dodanie nowej migawki do istniejącej grupy realizuje się za pomocą procedury DBMS_REFRESH.ADD.

Jej wywołanie wymaga podania wartości przynajmniej dwóch argumentów - nazwy grupy odświeżania i listy, bądź tablicy migawek.

```
dbms_refresh.add ('nazwa_grupy','lista migawek')
```

Kolejny przykład ilustruje sposób wywołania tej procedury. Polecenie to dodaje do grupy `rg_firma` dwie migawki `mv_sklepy` i `mv_produkty`.

```
exec dbms_refresh.add(
'rg_firma','mv_sklepy, mv_produkty')
```

Usunięcie migawki z grupy odświeżania

Migawkę usuwa się z grupy za pomocą procedury DBMS_REFRESH.SUBTRACT.

Jej wywołanie wymaga podania wartości przynajmniej dwóch argumentów - nazwy grupy odświeżania i listy, bądź tablicy migawek.

```
dbms_refresh.subtract('nazwa_grupy', 'lista migawek')
```

Przykład ilustruje wywołanie tej procedury. Polecenie to usuwa z grupy `rg_firma` dwie migawki `mv_sklepy` i `mv_produkty`.

```
exec dbms_refresh.subtract(
'rg_firma','mv_sklepy, mv_produkty')
```

„Ręczne” odświeżanie grupy

Grupę można odświeżać manualnie za pomocą procedury DBMS_REFRESH.REFRESH, wymagającej podania nazwy grupy, jako argumentu wywołania.

W tym przypadku odświeżana jest grupa o nazwie `rg_firma`.

```
exec dbms_refresh.refresh('rg_firma')
```

Usunięcie grupy odświeżania

Usunięcie grupy odświeżania, bez fizycznego usuwania migawek, realizuje się za pomocą procedury DBMS_REFRESH.DESTROY.

Argumentem jej wywołania jest nazwa grupy.

```
exec dbms_refresh.destroy('rg_firma')
```

Zmiana parametrów grupy odświeżania

Możliwa jest również zmiana parametrów grupy.

Służy do tego celu procedura DBMS_REFRESH.CHANGE.

Informacje na temat utworzonych grup odświeżania

Informacje o istniejących w systemie grupach odświeżania są udostępniane za pomocą perspektywy słownikowej o nazwie USER_REFRESH.

Przykładowe zapytanie do tej perspektywy przedstawiono poniżej.

Podane parametry definiują wartości kolumn wyprowadzanych na ekranie.

```
select rname, refgroup, implicit_destroy,
to_char(next_date, 'dd.mm.yyyy:hh24:mi:ss')
next_date,
interval, broken
from user_refresh;
```

RNAME - oznacza nazwę grupy odświeżania;

REFGROUP - jest identyfikatorem grupy;

IMPLICIT_DESTROY - przyjmuje wartość Y lub N; w pierwszym przypadku grupa zostanie automatycznie usunięta po usunięciu z niej ostatniej migawki;

NEXT_DATE - przechowuje datę i czas następnego odświeżenia grupy; wartość ta zmienia się po każdym kolejnym odświeżeniu grupy;

INTERVAL - jest wyrażeniem określającym częstotliwość odświeżania grupy, przekazaną argumentem wywołania interval;

BROKEN - określa, czy grupa jest odświeżana automatycznie jeśli wyspecyfikowano wartość interval; przyjmuje wartość N lub Y; w pierwszym przypadku grupa jest odświeżana automatycznie; w drugim przypadku grupa nie jest odświeżana. Dla danej grupy zdefiniowanej jako odświeżana automatycznie, system nada atrybutowi BROKEN wartość Y jeżeli nie uda się odświeżyć grupy predefiniowaną liczbę razy, np. z powodu czasowej niedostępności źródła danych. W takim przypadku grupę należy odświeżyć manualnie i jeśli taka próba się powiedzie, system zmieni wartość BROKEN na N, co przywróci odświeżanie automatyczne.

Informacje na temat migawek w grupie odświeżania

Informacje na temat migawek umieszczonych w danej grupie odświeżania można uzyskać za pomocą perspektyw słownikowej o nazwie USER_REFRESH_CHILDREN.

Przykładowe zapytanie pokazano poniżej. Parametry podane w zapytaniu określają wyprowadzane kolumny z informacją.

```
select name, rname, refgroup from user_refresh_children;
```

NAME - oznacza nazwę migawki znajdującej się w grupie;

RNAME - oznacza nazwę grupy odświeżania;

REFGROUP - jest numerem grupy odświeżania.

Jeden rekord tej perspektywy opisuje pojedynczą migawkę. Dla migawek znajdujących się w tej samej grupie wartość RNAME i REFGROUP jest identyczna.

6.5. Replikacja zaawansowana w Oracle

Oracle dostarcza narzędzie wspomagające replikację zaawansowaną, które nosi nazwę Oracle Advanced Replication. Mechanizm zaawansowanej replikacji pozwala na replikowanie tabel, widoków, indeksów, synonimów, pakietów, procedur oraz wyzwalaczy. Aby móc korzystać z wielu serwerów typu master, konieczna jest wersja bazy danych Oracle Enterprise Edition.

W zależności od wzajemnych relacji między węzłami replikacji systemu wyróżniamy następujące typy replikacji zaawansowanych:

· Replikację multimaster,

· Replikację migawkową,

· Replikację hybrydową.

Wymienione mechanizmy wykorzystują:

- obiekty replikacji;

- grupy replikacji.

Obiekt replikacji to obiekt schematu, powielony w bazach danych tworzących środowisko replikacji. Operacje, wykonane na obiekcie replikacji w jednej bazie danych środowiska, zostają zaaplikowane kopiom tego obiektu, istniejącym w pozostałych bazach środowiska.

Możliwe jest replikowanie następujących typów obiektów: tabele (zwykłe, ale także partycjonowane, zorganizowane jako indeksy, obiektowe, zawierające kolumny bazujące na typach zdefiniowanych przez użytkownika), indeksy, perspektywy (zwykłe i obiektowe), pakiety wraz z ciałami, procedury i funkcje, typy zdefiniowane przez użytkownika oraz indeksowe, wyzwalacze, synonimy oraz operatory.

Celem ułatwienia zadań administracyjnych, logicznie powiązane ze sobą obiekty replikacji łączy się w zbiory, nazywane grupami replikacji. Jedna grupa najczęściej zawiera obiekty jednego schematu (np. wykorzystywane przez jedną aplikację), jednak nie ma żadnych przeszkód, aby grupa replikacji zawierała obiekty z różnych schematów.

Dla jednego schematu można również zdefiniować wiele grup replikacji, jednak jeden obiekt może należeć tylko do jednej grupy.

Środowisko zaawansowanej replikacji tworzą dwa rodzaje węzłów: **węzły nadrzędne** (ang. master site) oraz **węzły migawkowe** (ang. materialized view site).

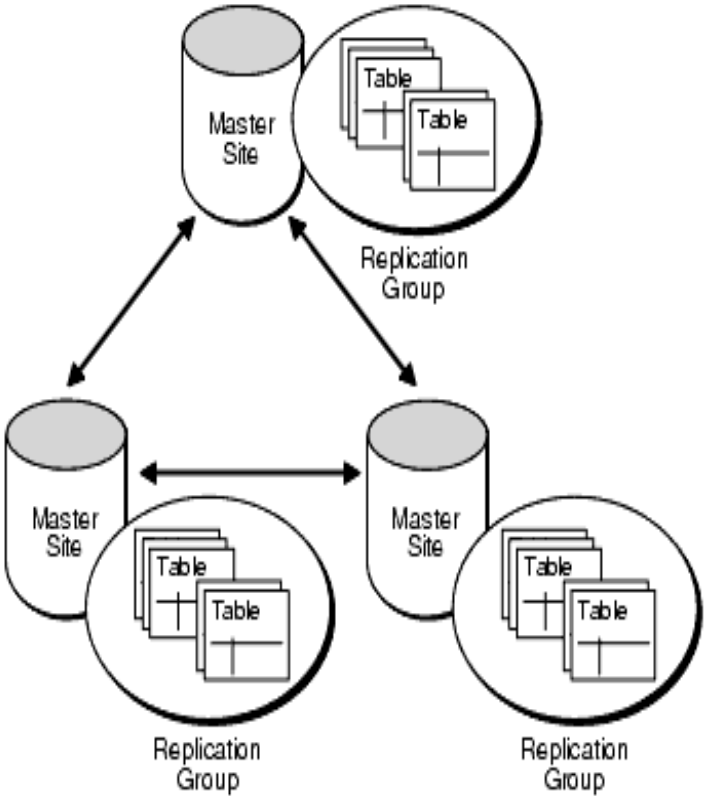
Grupy replikacji są definiowane w węzłach replikacji. Węzeł środowiska może być węzłem nadrzędnym dla jednej grupy replikacji i jednocześnie węzłem migawkowym dla innej grupy replikacji.

Jednak jeden węzeł nie może być jednocześnie węzłem nadrzędnym i węzłem migawkowym dla tej samej grupy replikacji.

Replikacja multimaster w Oracle

Najbardziej charakterystyczną cechą replikacji multimaster (nazywanej również peer-to-peer albo replikacją n-way) jest równoważność węzłów (Rysunek).

Lokalna operacja modyfikacji danych węzła zostaje przesłana i zaaplikowana we wszystkich węzłach zdalnych środowiska. Oznacza to, że każdy węzeł może być źródłem danych dla pozostałych węzłów, z tego względu przyjęto, że każdy węzeł środowiska multimaster jest węzłem nadrzędnym.



Kolejną cechą środowiska replikacji multimaster jest identyczność replikowanych obiektów we wszystkich węzłach środowiska. Wszystkie obiekty muszą być identyczne zarówno pod względem struktury jak i zawartości danych (w przypadku replikacji tabel). Nie ma możliwości ograniczenia zakresu danych, replikowanych pomiędzy węzłami.

Domyślnym trybem działania replikacji multimaster jest replikacja na poziomie rekordu. Oznacza to, że każda operacja modyfikacji rekordu musi zostać przesłana i wykonana we wszystkich zdalnych węzłach środowiska.

Ten tryb replikacji nie jest zalecany dla operacji wykonywanych na dużych zbiorach danych. Do takich zastosowań Oracle zaleca skorzystanie z alternatywnego trybu – replikacji proceduralnej. Wówczas, pomiędzy węzłami przesyłane są jedynie wywołania procedur z odpowiednio skonstruowanych pakietów, które dokonują żądanych operacji na danych.

Celem stosowania replikacji multimaster jest zwiększenie dostępności danych. W razie awarii jednego węzła system może kontynuować pracę, komunikując się z innym węzłem, który posiada taką samą strukturę jak i dane. Kolejnym zastosowaniem replikacji multimaster jest możliwość rozłożenia obciążenia, wynikającego z operacji użytkowników, pomiędzy poszczególne węzły środowiska. Użytkownicy mogą korzystać z danych dostępnych w węźle, do którego koszt dostępu jest najniższy.

Replikacja multimaster oferuje użytkownikom dwa mechanizmy odświeżania danych w replikach. Są nimi replikacja asynchroniczna i synchroniczna.

Multimaster asynchroniczna

Główną cechą replikacji asynchronicznej jest to, że operacje z lokalnej transakcji zostają zapamiętane w węźle, a następnie, co pewien okres czasu są przesyłane do węzłów zdalnych. Zatem węzły uzyskają jednakowy stan dopiero po pewnym czasie, od dokonania zmian.

Replikacja asynchroniczna jest domyślnym sposobem replikacji multimaster. Nie powoduje ona zmniejszenia wydajności wykonywanych operacji oraz pozwala na odłączanie i dołączanie węzłów systemu bez narażenia systemu na awarię. Jednak dzieje się to kosztem opóźnienia w uzyskaniu spójnego stanu replik. Inną wadą stosowania tego typu replikacji jest możliwość wystąpienia konfliktów replikacji (aktualizacji, unikalności, usunięcia).

Multimaster synchroniczna

W przypadku replikacji synchronicznej w ramach modyfikacji lokalnej repliki obiektów, zostają włączone operacje przesłania i zaaplikowania tych zmian we wszystkich pozostałych węzłach środowiska. Sam proces przesłania i zastosowania zmian jest częścią transakcji węzła lokalnego. Pomyślne zakończenie transakcji lokalnej (zatwierdzenie) oznacza, że zmiany zostały z powodzeniem przesłane i zaaplikowane we wszystkich pozostałych węzłach środowiska, dzięki czemu replikowane dane są w nich zawsze identyczne.

Co więcej przy replikacji synchronicznej nie występują żadne konflikty, więc nie są potrzebne metody ich rozwiązywania.

Z kolei wystąpienie błędów przy przesyłaniu zmian do któregokolwiek z węzłów (np. na skutek awarii sieci) powoduje niemożność dokończenia lokalnej transakcji węzła, który zapoczątkował cały proces i w konsekwencji jej wycofanie. Stąd replikacja synchroniczna wymaga stałych i pewnych połączeń pomiędzy węzłami środowiska.

Multimaster proceduralna

Replikacja proceduralna to szczególny przypadek replikacji multimaster. Odmiennie, niż w przypadku zwykłej replikacji multimaster, nie jest to replikacja na poziomie rekordu. Pomiędzy węzłami replikowane są wywołania procedury składowanej, która modyfikuje zawartość tabel. Natomiast nie są replikowane bezpośrednie operacje modyfikacji tabeli.

Replikację proceduralną stosuje się w przypadku, gdy w systemie przeważają transakcje modyfikujące duże ilości danych. W takiej sytuacji replikacja na poziomie rekordu jest mało wydajna. Przy replikacji proceduralnej jednocześnie może działać tylko jedna replikowana procedura (obowiązuje propagacja sekwencyjna). Wywołania procedury zostają natychmiast przesłane do węzłów docelowych w przypadku replikacji synchronicznej lub są składowane w lokalnej kolejce transakcji przy replikacji asynchronicznej.

Zaawansowana replikacja migawkowa w Oracle

Środowisko replikacji migawkowej tworzą dwa rodzaje węzłów: *węzły nadrzędne* (ang. master site) oraz *węzły migawkowe* (ang. materialized view site).

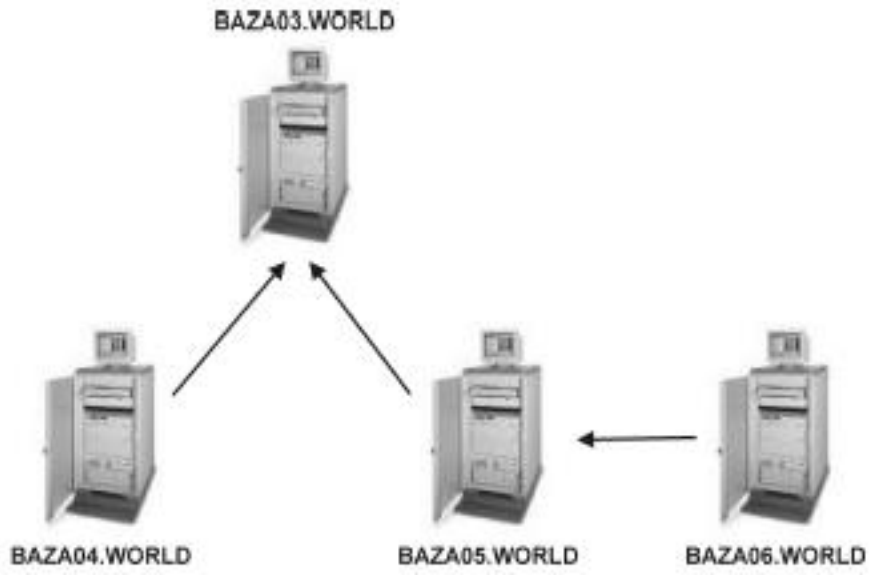
Każdy węzeł migawkowy jest połączony z dokładnie jednym węzłem nadrzędnym, natomiast węzeł nadrzędny może być połączony z wieloma węzłami migawkowymi.

Najczęściej spotykaną konfiguracją jest architektura jednowarstwowa, w której w środowisku występuje tylko jeden węzeł nadrzędny. Możliwa jest jednak również konfiguracja, w której węzeł migawkowy jest jednocześnie węzłem nadrzędnym dla innych węzłów migawkowych. Taka architektura nazywana jest wielowarstwową.

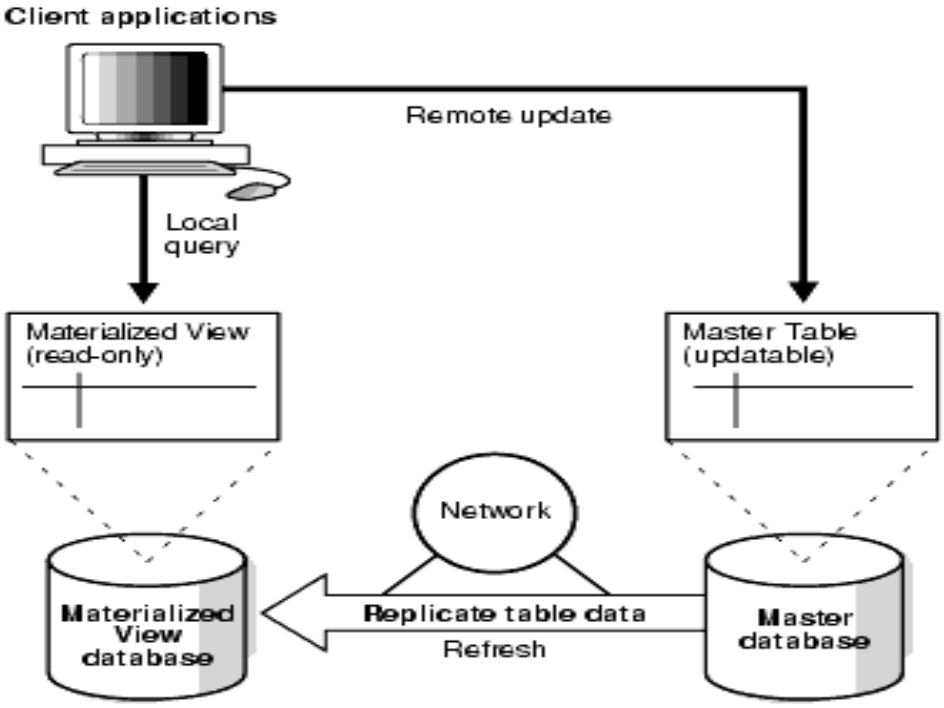
Przykład architektury wielowarstwowej z 5 węzłami.

Węzły nadrzędne - BAZA03.WORLD i BAZA05.WORLD;
Węzły migawkowe - BAZA04.WORLD, BAZA05.WORLD i BAZA06.WORLD.

Węzeł BAZA05.WORLD jest zarówno węzłem nadrzędnym, jak i migawkowym.



Replikacja migawkowa jest replikacją na poziomie rekordu. *Węzły migawkowe* zawierają migawki (perspektywy zmaterializowane). Migawka jest połączona z tabelą z *węzła nadrzędnego* i kopiuje lokalnie jej dane. W takim wypadku tabela nosi nazwę tabeli nadrzędnej (ang. master table). Gdy mamy do czynienia z architekturą wielowarstwową, wówczas migawka w węźle migawkowym kopiuje dane z migawki w węźle nadrzędnym – wówczas migawka w węźle nadrzędnym nosi nazwę nadrzędnej migawki (ang. master materialized view).



Proces propagacji zmian w replikacji migawkowej różni się znacznie od propagacji w opisanej już wcześniej replikacji multimaster. O ile w replikacji multimaster replikowane tabele węzła są w sposób ciągły uaktualniane przez operacje z innych węzłów, o tyle przy replikacji migawkowej operacje z węzła nadrzędnego są łączone w „paczki” (ang. batch updates) i uaktualniają migawkę w węźle migawkowym. Taki sposób propagacji zmian nosi nazwę odświeżania (ang refresh).

Najważniejsza cecha replikacji migawkowej:

- możliwość ograniczenia i wstępnego przetworzenia danych, kopiowanych z węzła nadrzędnego do węzła migawkowego;

- ograniczenie danych może być zarówno poziome (replikowanie rekordów z tabeli węzła nadrzędnego, spełniających określone warunki), jaki pionowe (replikowanie danych z wybranych kolumn tabel węzła nadrzędnego); jest to istotna różnica w odniesieniu do replikacji multimaster, gdzie takie ograniczenia nie mogły być stosowane, gdyż replikacji podlegała cała tabela.

Migawka może być tylko do odczytu lub może być też modyfikowalna.

Jeśli w węzłach migawkowych zastosowano migawki modyfikowalne, wówczas lokalne zmiany danych w migawkach w węźle migawkowym mogą zostać replikowane do węzła nadrzędnego, modyfikując zawartość tabeli (lub migawki jeśli węzłem nadrzędnym był węzeł migawkowy).

Wynika stąd, że mimo nierównego statusu węzłów propagacja zmian w replikacji migawkowej może mieć również charakter dwukierunkowy (podobnie jak w replikacji multimaster).

Zastosowania

Replikacja migawkowa sprawdza się w zastosowaniach wymagających replikowania tylko określonego podzbioru danych. Bardzo cenną cechą replikacji migawkowej jest możliwość pracy odłączonej. Węzłem migawkowym środowiska może być komputer przenośny, natomiast węzłem nadrzędnym – centralny serwer firmy.

Architektura systemu zaawansowanej replikacji migawkowej

Organizacja węzła nadrzędnego

Organizacja węzła nadrzędnego w replikacji migawkowej jest identyczna jak w przypadku replikacji multimaster. Jeśli tabela (migawka) w węźle nadrzędnym ma być źródłem danych dla migawki w węźle migawkowym i odświeżanie ma być realizowane w sposób przyrostowy, wówczas należy zdefiniować dziennik perspektywy zmaterializowanej.

Jeśli migawka ma być odświeżana w sposób pełny, założenie dziennika nie jest potrzebne.

Organizacja węzła migawkowego

Łączniki bazodanowe

Węzeł migawkowy wymaga utworzenia trzech łączników bazodanowych do węzła nadrzędnego.

Pierwszy łącznik jest łącznikiem prywatnym administratora węzła migawkowego i wskazuje na użytkownika – pełnomocnika administratora w węźle nadrzędnym.

Drugi łącznik prywatny jest własnością nadawcy odroczonych transakcji w węźle migawkowym i wskazuje na użytkownika – odbiorcę w węźle nadrzędnym.

Trzeci łącznik jest własnością użytkownika – właściciela perspektyw zmaterializowanych i wskazuje na pełnomocnika użytkownika odświeżającego migawki z węzła nadrzędnego.

Migawka (perspektywa zmaterializowana) lub grupa migawek

Migawka to jedyny obiekt replikacji w węźle migawkowym. Kopiuje on dane z obiektu nadrzędnego (tabeli lub innej migawki w przypadku architektury wielowarstwowej).

Rodzaje migawek

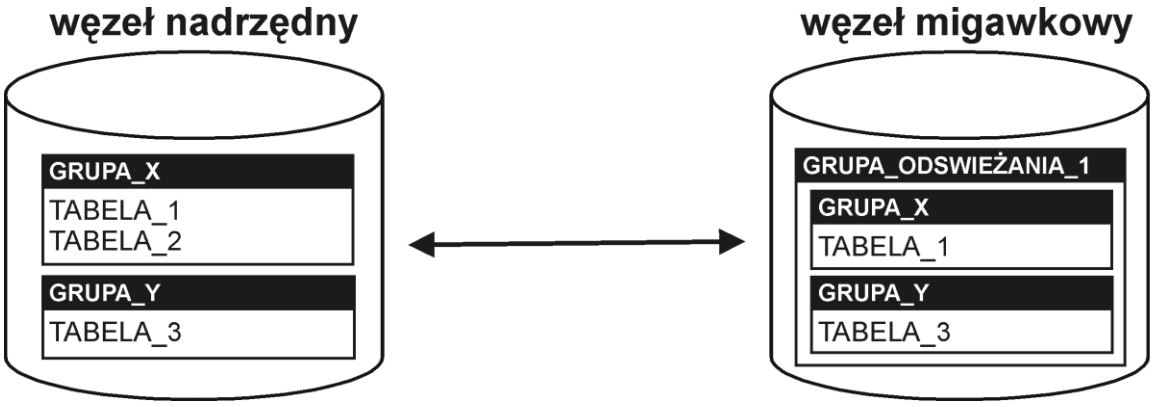
1. Tylko do odczytu – migawki tego typu mogą być odpytywane zapytaniami, jednak nie mogą być aktualizowane przez żadną z komend INSERT, UPDATE, DELETE. Jeśli dane z takiej perspektywy zmaterializowanej mają być aktualizowane należy je zmieniać bezpośrednio na tabeli źródłowej.

2. Modyfikowalna – migawka umożliwiająca użytkownikom aktualizację danych. W celu zaktualizowania tabeli źródłowej, podczas odświeżania, za pośrednictwem perspektywy zmaterializowanej, konieczne jest, aby ta perspektywa należała do grupy perspektyw zmaterializowanych (grupy migawek). Aby migawka była modyfikowalna, musi ona być oparta o jedną tabelę, mogą jednak występować referencje do innych tabel w podzapytaniach.

3. Zapisywalna - migawka tego typu może istnieć samodzielnie, tj. nie należy do grupy perspektyw zmaterializowanych, więc nie może ona powodować zmian w tabeli źródłowej, natomiast pozwala ona na wprowadzanie zmian w jej obrębie (modyfikacje danych migawki jak w tabeli). Zmiany te przepadają za każdym odświeżeniem.

Grupa migawek (perspektyw zmaterializowanych) (ang. materialized view group) to zbiór logicznie powiązanych migawek z węzła migawkowego. Musi ona być połączona z dokładnie jedną grupą replikacji z węzła nadrzędnego, *nazwy obu grup* muszą być identyczne.

Na rysunku pokazano dwie grupy migawek w węźle migawkowym, połączonych z grupami replikacji w węźle nadrzędnym.



Migawki z grupy migawek nie muszą kopiować danych ze wszystkich obiektów z grupy nadrzędnej. W tym przypadku widzimy, że migawka z grupy o nazwie GRUPA_X kopiuje tylko dane z tabeli TABELA_1, natomiast nie kopiuje danych z tabeli TABELA_2.

Grupy migawek są niezbędne jedynie wówczas, jeśli w węźle migawkowym zaimplementowane są migawki modyfikowalne. Jeśli migawka nie należy do żadnej grupy, wówczas musi to być migawka tylko do odczytu bądź zapisywalna.

Grupy odświeżania

Jeśli określone migawki powinny być odświeżane jednocześnie w celu zachowania spójności replikowanych przez nie danych, wówczas należy je dodać do grupy odświeżania (ang. refresh group). Po odświeżeniu wszystkich migawek, należących do tej samej grupy odświeżania, ich dane pochodzą z tego samego punktu w czasie. Maksymalna liczba migawek w grupie odświeżania to 400.

Na rysunku pokazano przykład grupy odświeżania o nazwie GRUPA_ODSWIEZANIA_1.

Grupa odświeżania może zawierać migawki z różnych grup migawek. Migawki z tej samej grupy migawek mogą należeć do różnych grup odświeżania. Jednak decydując się na takie rozwiązanie należy zdawać sobie sprawę, że może to prowadzić do problemów z zachowaniem spójności danych w węźle migawkowym.

Grupa odświeżania definiuje okres czasu pomiędzy kolejnymi uruchomieniami procesu odświeżenia migawek.

Proces odświeżania perspektyw zmaterializowanych

Proces odświeżania składa się z dwóch operacji:

- odświeżenie zawartości migawki, zdefiniowanej w węźle migawkowym, przez dane tabeli źródłowej z węzła nadrzędnego;

- w przypadku migawki modyfikowalnej następuje też druga operacja, tj. zawartość tabeli źródłowej węzła nadrzędnego jest modyfikowana przez operacje DML, które zostały wykonane dla migawki w węźle migawkowym.

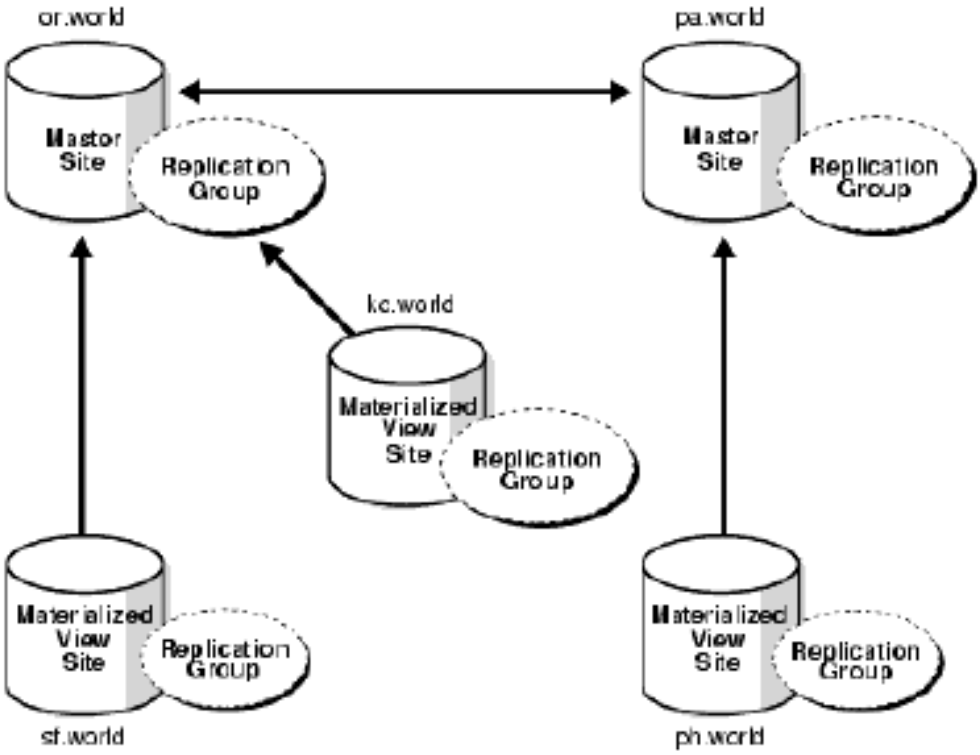
Proces odświeżania w replikacji migawkowej składa się z następujących kroków:

1. Jeśli migawka jest migawką modyfikowalną, tj. została zdefiniowana z klauzulą for update i wchodzi w skład grupy migawek, wówczas wszystkie transakcje, które dokonały modyfikacji danych migawki, zostają zapisane w kolejce odroczonych transakcji (podobnie jak przy asynchronicznej replikacji multimaster). Pierwszym krokiem procesu odświeżenia migawki jest przesłanie zawartości kolejki odroczonych transakcji do węzła nadrzędnego (lub węzła migawkowego w przypadku architektury wielowarstwowej) i zaaplikowanie ich w tabeli (migawce) źródłowej. W tym momencie mogą występować konflikty, wynikające z jednoczesnego przesłania transakcji z migawek z różnych węzłów migawkowych.

2. Właściwe odświeżenie migawki – migawka kopiuje dane z obiektu z węzła nadrzędnego. Należy pamiętać, że jest to kopia z punktu w czasie. Np. rozważmy następującą sytuację. Węzeł nadrzędny zawiera tabelę STUDENCI, która jest obiektem nadrzędnym dla migawki STUDENCI_MV w węźle migawkowym. Pomiędzy kolejnymi procesami odświeżania perspektywy zawartość tabeli nadrzędnej została poddana 10 operacjom modyfikacji danych. Jednak odświeżenie migawki będzie polegało jedynie na doprowadzeniu jej do stanu odpowiadającego bieżącemu stanowi obiektu nadrzędnego.

Replikacja hybrydowa w Oracle

Replikacja hybrydowa jest kombinacją dwóch wcześniej wymienionych typów mechanizmów replikacji. Środowisko replikacji hybrydowej zawiera zarówno węzły nadrzędne jak i migawkowe. Przykład tego typu replikacji znajduje się na rysunku poniżej.



Konflikty replikacji

W przypadku dwukierunkowych replikacji asynchronicznych (multimaster, migawkowa, hybrydowa) może dochodzić do konfliktów replikacji.

Konflikt replikacji oznacza sytuację, w której lokalne zaaplikowanie operacji z węzła zdalnego nie jest możliwe, gdyż mogłoby bądź spowodować utratę spójności lokalnego schematu węzła, bądź nie istnieją dane w węźle, których dotyczy zdalna operacja (zostały zmodyfikowane lub usunięte przez lokalną transakcję węzła). Wystąpienie konfliktu powoduje niemożność uzgodnienia stanu replikowanych obiektów pomiędzy węzłami, co w konsekwencji może doprowadzić do utraty zgodności schematów węzłów środowiska. Taka sytuacja musi zostać natychmiast wykryta i w miarę możliwości rozwiązana. SZBD Oracle próbuje automatycznie rozwiązywać konflikty określonego rodzaju, jednak nie zawsze odnosi sukces. W takim przypadku powiadamia administratora środowiska replikacji o wystąpieniu sytuacji konfliktowej - administrator musi ręcznie spróbować usunąć konflikt.

Jak już wspomniano konflikty występują tylko przy asynchronicznej propagacji zmian. Przy propagacji synchronicznej wystąpienie konfliktu nie jest możliwe, gdyż sama operacja przesłania zmian do węzłów zdalnych jest częścią lokalnej transakcji, która te zmiany wprowadza. W tym przypadku moduł zarządzania współbieżnością systemu zarządzania bazą danych nie dopuszcza do wystąpienia sytuacji błędnych (czyli niepożądanego wpływu jednoczesnych operacji z innych transakcji). Odbywa się to m.in. przez zakładanie blokad na modyfikowanych rekordach.

W przypadku stosowania *replikacji proceduralnej* procedura, która replikuje dane pomiędzy węzłami, jest odpowiedzialna za utrzymywanie spójności danych. Powinna ona zostać tak zaprojektowana, aby unikała konfliktów lub je wykrywała i usuwała.

Rodzaje konfliktów w Oracle

W środowisku zaawansowanej replikacji mogą występować następujące rodzaje konfliktów:

• konflikt aktualizacji – występuje wtedy, gdy transakcje, pochodzące z różnych węzłów środowiska, dokonują w tym samym czasie operacji aktualizacji tego samego rekordu,

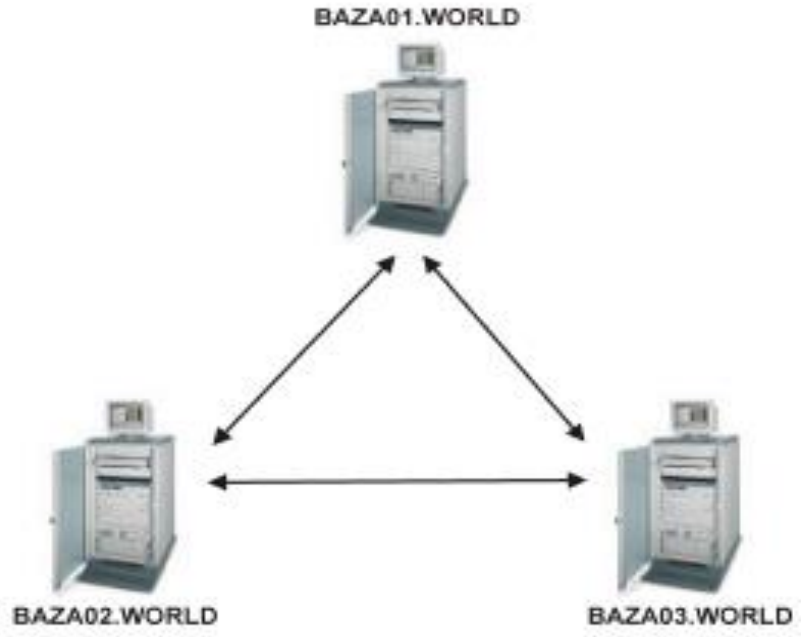
• konflikt unikalności – występuje wtedy, gdy replikacja rekordu powoduje naruszenie ograniczeń typu klucz podstawowy lub klucz unikalny w węzłach docelowych,

• konflikt usunięcia – występuje w przypadku, gdy jedna transakcja usuwa rekordy z replikowanej tabeli, które są w tym samym momencie modyfikowane lub usuwane przez transakcję z innego węzła.

• konflikt kolejnościowy (ang. ordering conflict), powstający w wyniku różnej kolejności propagacji transakcji pomiędzy węzłami. Występuje on tylko w środowisku, które tworzy trzy lub więcej węzłów.

W przypadku systemu asynchronicznej replikacji multimaster - pokazanego na rysunku, załóżmy, że w wyniku zajścia pewnych okoliczności węzeł BAZA01.WORLD nie jest dostępny, czyli uaktualnienia (tworzące transakcje), które w międzyczasie zostały wykonane w replikowanych obiektach w węzłach BAZA02.WORLD i BAZA03.WORLD nie mogą zostać zaaplikowane w tym węźle. W systemie replikacji multimaster odroczone transakcje w przypadku niedostępności węzła docelowego są przechowywane w węzłach

wysyłających. Kiedy węzeł BAZA01.WORLD stanie się znowu dostępny, odroczone transakcje zostaną przesłane do tego węzła, jednak ich kolejność może być różna od kolejności, w jakiej zostały one zaaplikowane na pozostałych węzłach środowiska. Przez to uaktualnienia, które tworzą te transakcje mogą spowodować konflikty.



Wykrywanie konfliktów

Każdy węzeł posiada wbudowane mechanizmy, umożliwiające automatyczne wykrycie sytuacji konfliktowej w momencie aplikowania odroczonej transakcji, przesłanej z węzła zdalnego.

SZBD Oracle, przesyłając pomiędzy węzłami informacje o operacji modyfikacji rekordu dokonanej w sposób asynchroniczny, przesyła nie tylko „nowe” wartości kolumn z replikowanego rekordu (czyli postać rekordu po wykonaniu modyfikacji), ale także „stare” wartości kolumn (czyli sprzed wykonania modyfikacji). Jeśli węzeł docelowy wykryje, że istnieje różnica pomiędzy „starą” wartością replikowanego rekordu, przesłaną wraz z operacją modyfikującą z węzła zdalnego, a wartością aktualną, odpowiadającego mu lokalnego rekordu, zgłasza wystąpienie konfliktu aktualizacji (węzeł zdalny i lokalnie replikowany nie są spójne).

Rozwiązywanie konfliktów

Wykrycie konfliktu powoduje uruchomienie odpowiednich procedur, mających na celu rozwiązanie konfliktu i zapewnienie zgodności replikowanych schematów węzłów środowiska.

System Oracle dostarcza zestaw predefiniowanych metod, próbujących rozwiązać *konflikty aktualizacji i unikalności*.

Nie ma żadnych wbudowanych metod rozwiązywania *konfliktów usunięcia i kolejnościowych*. Administrator może zaimplementować swoje własne metody rozwiązywania konfliktów i używać ich zamiast metod predefiniowanych (w przypadku konfliktów aktualizacji i usunięcia) lub w miejsce ich braku (dla konfliktów usunięcia i kolejnościowych).

Predefiniowane metody rozwiązywania konfliktów aktualizacji

- Metoda najpóźniejszego znacznika (ang. latest timestamp) bazuje na znaczniku czasowym, który dla każdej operacji aktualizacji danych określa moment jej wystąpienia. W przypadku konfliktu dwóch operacji update wykonana zostaje ta, której znacznik czasowy jest późniejszy. Dobrym rozwiązaniem jest połączenie metody najpóźniejszego znacznika z inną metodą, która przejmie rozwiązanie konfliktu w przypadku, gdy konfliktowe operacje mają identyczne znaczniki czasowe. Metoda wymaga zaimplementowania w każdym z węzłów mechanizmu oznaczania czasu wykonania każdej operacji update danych replikowanej tabeli (np. odpowiedniego wyzwalacza).

- Metoda nadpisania (ang. overwrite) polega na zastępowaniu, w przypadku wystąpienia konfliktu, bieżącej wartości kolumny w węźle docelowym nową wartością kolumny z węzła wysyłającego. Metoda gwarantuje zbieżność schematów (stanu) węzłów w środowiskach z jednym węzłem nadrzędnym i wieloma węzłami migawkowymi. Można ją również zastosować w środowisku z wieloma węzłami nadrzędnymi, jednak zbieżność węzłów nie jest wtedy zagwarantowana.

- Metoda addytywna (ang. additive) służy do rozwiązywania konfliktów w grupach kolumn, zawierających pojedyncze kolumny numeryczne. W przypadku wystąpienia konfliktu nową wartością kolumny jest suma wartości bieżącej i różnicy pomiędzy wartością nową a wartością poprzednią. Metoda zapewnia zbieżność schematów węzłów dla środowisk z dowolną liczbą węzłów nadrzędnych i migawkowych. Przeznaczona jest szczególnie dla systemów finansowych, z dużą liczbą operacji składania i wycofywania depozytów.

Predefiniowane metody rozwiązywania konfliktów unikalności

Predefiniowane metody rozwiązywania konfliktów unikalności nie gwarantują zbieżności schematów węzłów. Ich działanie sprowadza się tylko do usunięcia sytuacji naruszającej klucz podstawowy lub unikalny w replikowanej tabeli, powinien jednak zostać zaimplementowany dodatkowy mechanizm, który spowoduje zawiadomienie administratora o wystąpieniu konfliktu. Wówczas, administrator musi ręcznie doprowadzić replikowane schematy do zgodności.

- Metoda dodania nazwy węzła (ang. append site name) działa w sposób następujący. Jeśli operacja, wchodząca w skład zdalnej transakcji, modyfikując wartość określonej kolumny replikowanej tabeli w węźle docelowym, spowoduje naruszenie jej unikalności (naruszenie klucza podstawowego lub unikalnego), wówczas SZBD Oracle do nowej wartości modyfikowanej kolumny dodaje część globalnej nazwy węzła (do pierwszej kropki), z którego pochodzi transakcja. Czyli np. w przypadku konfliktu, spowodowanego przez operację z transakcji z węzła TEST.WORLD, zmieniającą wartość kolumny x na ‘ABC’, metoda spowoduje zapisanie w kolumnie x wartości ‘ABCTEST’. Oczywiście metoda dodania nazwy węzła może zostać zastosowana wyłącznie dla kolumn tekstowych.

- Metoda dodania sekwencji (ang. append sequence) działa podobnie jak metoda poprzednia, jednak zamiast nazwy węzła dodaje do wartości kolumny, powodującej konflikt, automatycznie wygenerowany numer sekwencyjny.

- W metodzie zignorowania (ang. discard), jeśli rekord, przesłany z węzła zdalnego, narusza w węźle docelowym klucz podstawowy lub unikalny, wówczas SZBD Oracle po prostu ignoruje rekord i nie aplikuje go w węźle docelowym.

Nie wszystkie wbudowane metody rozwiązywania konfliktów doprowadzają replikowane schematy do zbieżności. Przykładem jest metoda nadpisania – bieżąca wartość w węźle docelowym zostaje zastąpiona wartością z węzła zdalnego. W środowisku, składającym się z więcej niż dwóch węzłów nadrzędnych mechanizm taki doprowadza do utraty zgodności schematów węzłów.

W przypadku metod nie gwarantujących zbieżności węzłów musi zostać zaimplementowany dodatkowy mechanizm, mający za zadanie poinformowanie administratora środowiska o wystąpieniu konfliktu i „połowicznym” jego rozwiązaniu – na administratora spada wówczas zadanie doprowadzenia schematów do stanu zgodnego.

Informacja o konfliktach

Informacja o konflikcie, którego rozwiązanie się nie powiodło (zawiodły wszystkie metody) lub też nie zdefiniowano żadnych metod rozwiązywania konfliktów, trafia do kolejki błędów węzła, gdzie konflikt wystąpił. Kolejka błędów jest implementowana przez perspektywę DEFERROR. Umożliwia ona dostęp do informacji o transakcjach, których zaimplementowanie w węźle nie było możliwe (czyli spowodowały konflikt). W perspektywie DEFERROR nie ma informacji o konfliktach, których rozwiązanie zakończyło się powodzeniem w wyniku zastosowania zdefiniowanych metod. Dlatego, należy zaimplementować samodzielnie mechanizm powiadamiania administratora o konfliktach rozwiązanych przez metody, które nie gwarantują zbieżności schematów.

6.6. Strumienie Oracle

Mechanizm Oracle Streams umożliwia współdzielenie danych pomiędzy bazami poprzez przekazywanie informacji od producenta do szeregu subskrybentów.

Oracle Streams pozwala między innymi na budowę środowisk replikacyjnych o topologii jeden-do-jednego, jeden-do-wielu, wiele-do-jednego, wiele-do-wielu, środowisk klasy single-master i multimaster w oparciu o mechanizm kolejek komunikatów.

Kolejki komunikatów zapewniają, że informacje o zdarzeniach dotrą do wszystkich odbiorców dzięki spełnieniu następujących własności:

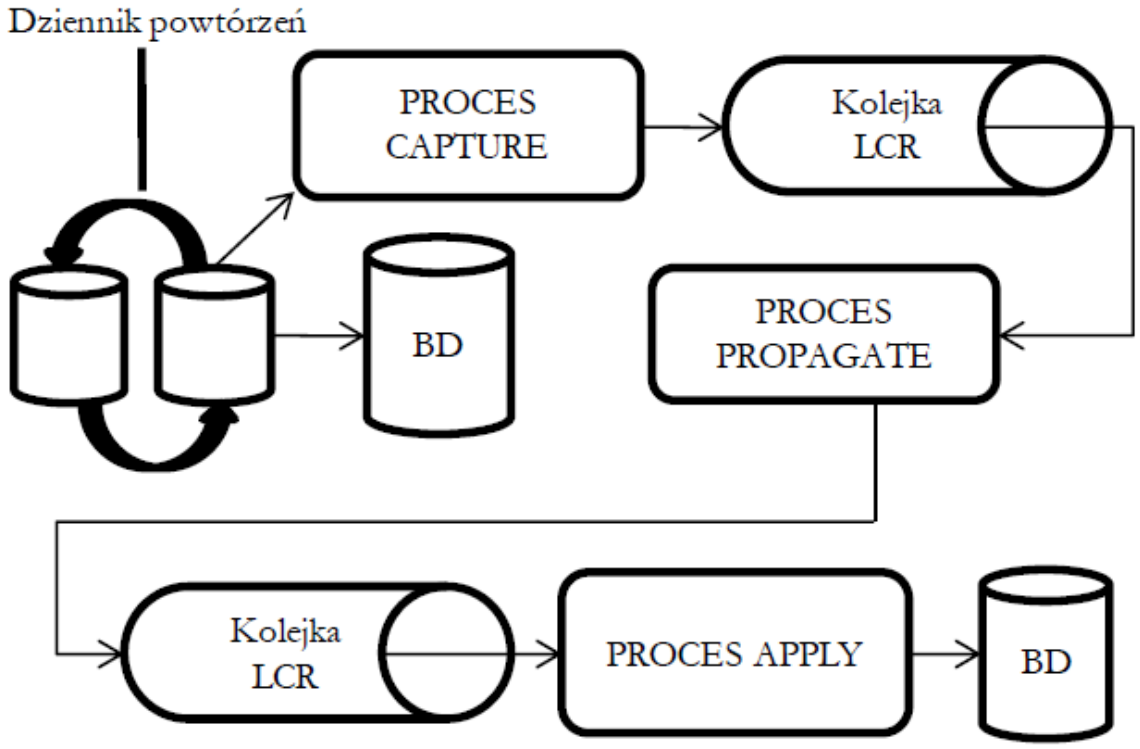
- zdarzenia związane z wykonywanymi operacjami są propagowane do kolejek docelowych zawsze dokładnie raz;
- kolejka źródłowa jest informowana o dotarciu informacji do kolejki docelowej;
- informacja o zdarzeniu przebywa w kolejce źródłowej tak długo, aż nie zostanie przesłana do wszystkich kolejek docelowych.

Podstawowa idea działania technologii Oracle Streams opiera się na użyciu trzech typów zadań bazodanowych:

- Capture,

- Propagate,

- Apply.



Capture

Proces Capture prowadzi nasłuch operacji DML i DDL wykonywanych przez użytkowników w systemie źródłowym, co pozwala na wydobywanie informacji o wykonywanych operacjach na rekordach bazy danych w tym także informacji o zmianach w strukturze samej bazy. Źródłem tych informacji może być dziennik powtórzeń lub sama aplikacja korzystająca z odpowiedniego API.

Do tymczasowego przechowywania wychwyconych operacji służą struktury kolejkowe Oracle Advanced Queueing. Zmiany w bazie danych zapisywane są w obiektach LCR (Logical Change Record).

Proces Capture posługuje się w systemie operacyjnym identyfikatorami Cnnn. Proces ten może być wykorzystywany w dwóch konfiguracjach:

- detekcji lokalnej (Local Capture),

- detekcji zdalnej (Downstream Capture).

Z detekcją lokalną mamy do czynienia wtedy, kiedy proces Capture pracuje na tym samym komputerze, na którym znajduje się system zarządzania źródłową bazą danych.

Detekcja zdalna polega na osadzeniu procesu Capture po stronie docelowego systemu zarządzania bazą danych i na wykorzystaniu mechanizmów transportu plików dziennika powtórzeń. W ramach tego rozwiązania pliki dziennika powtórzeń, powstające po stronie źródłowej bazy danych, są w niezmienionej postaci transferowane do systemu docelowego i dopiero tam podlegają analizie, w wyniku której są generowane i kolejkowane obiekty LCR.

Zaletami wykorzystywania zdalnej konfiguracji procesu detekcji zmian są m.in.: odciążenie systemu źródłowego, gdyż przetwarzanie plików dziennika powtórzeń odbywa się w środowisku docelowym, a także prostota architektury w przypadku, gdy replikacja następuje z wielu systemów źródłowych do jednego systemu docelowego.

Propagate i Apply

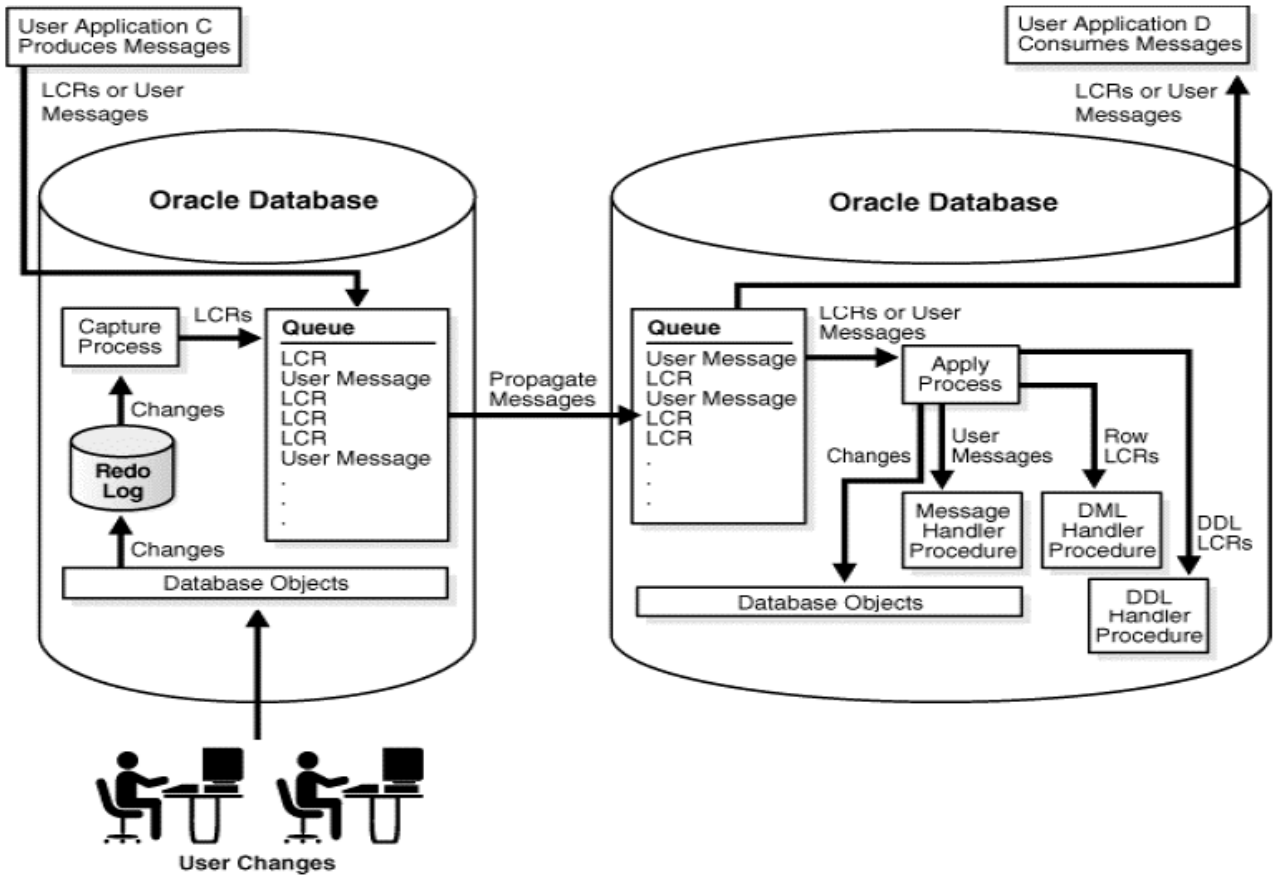
Proces Propagate przesyła wychwycone operacje, w ramach jednej lub wielu baz danych, do systemu docelowego.

Proces Apply pozwala na wykonanie rozpoznanych operacji zmian rekordów bazy danych na systemie docelowym lub przekazanie informacji o takich operacjach poprzez specjalizowane API do dowolnej aplikacji.

Propagacja do określonej lokalizacji następuje po spełnieniu przez komunikat określonych, definiowanych przez użytkownika reguł. Informacja taka jest uznawana za skonsumowaną, jeżeli uda się poprawnie zaaplikować taką informację lub umieścić w kolejce błędów informację o niemożności zaaplikowania zmiany.

Zmiany mogą być aplikowane bezpośrednio przez proces Apply, bądź też przesyłane jako parametr do programu zdefiniowanego przez użytkownika (PL/SQL, Java, C++, C).

Dla każdego źródła danych konieczny jest osobny proces Apply.



Przechwytywane operacje mogą być automatycznie filtrowane i transformowane w każdym punkcie architektury systemu. Definiuje się w tym celu odpowiednie reguły.

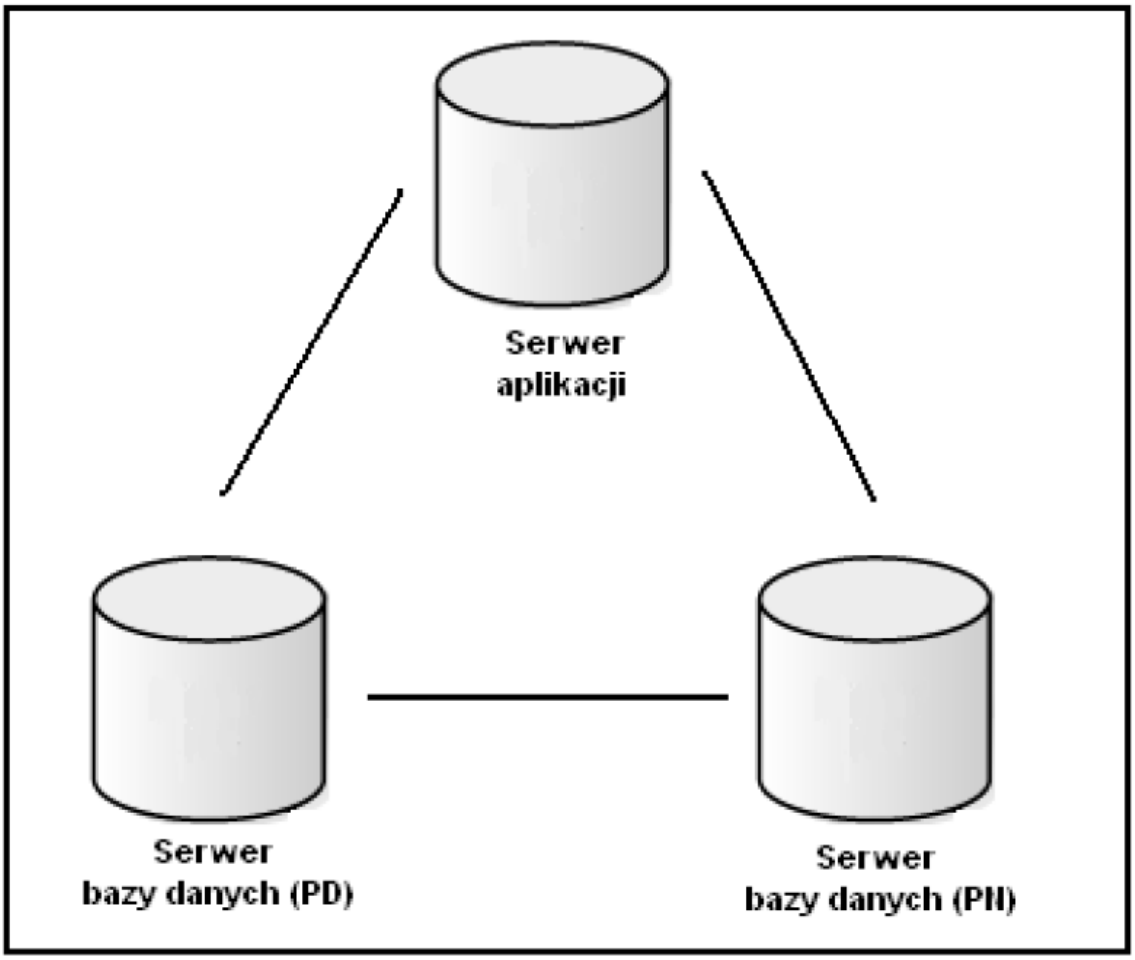
Wykorzystywane w mechanizmie Oracle Streams reguły, są to obiekty bazodanowe, których akcja jest wykonywana po zajściu określonego zdarzenia i spełnieniu zdefiniowanych warunków.

Reguły mogą zawierać w sobie oprócz warunku także kontekst rozpatrywania tego warunku oraz informacje o kontekście wykonywania zdefiniowanej akcji. Aby reguła była rozpatrywana przez system musi wchodzić w skład tak zwanego zestawu reguł. System może samodzielnie tworzyć reguły na poziomie globalnym, schematu użytkownika i pojedynczych tabel. Takie reguły są generowane dla zdarzeń związanych ze zmianą rekordów bazy danych lub struktur bazy danych (jak tabele) lub obu tych typów jednocześnie.

Konfiguracja baz danych implementująca Oracle Streams

Zbiór nazw usług baz danych (lokalnych i zdalnych), które mogą być wykorzystywane przez aplikacje użytkownika jest umieszczony w pliku *tnsnames.ora*.

W przykładzie na jednej maszynie (localhost) uruchomione zostały dwie instancje bazy danych. Instancje te zostały nazwane: PN i PD.



Komunikacja między bazami przebiega przy wykorzystaniu protokołu TCP/IP, natomiast *listener* słucha na porcie 1521.

Listing przedstawia zawartość pliku tnsnames.ora:

```
PD =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = arek)(PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = pd)
    )
  )

PN =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = arek)(PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = pn)
    )
  )
```

W celu umożliwienia zdalnego dostępu aplikacjom do bazy danych niezbędne jest skonfigurowanie sieciowego procesu komunikacyjnego listener.

Proces taki jest uruchomiony na tej samej maszynie, co baza danych. Konfiguracja listenera sprowadza się do edycji pliku *listener.ora*, umieszczonego w tym samym katalogu, co plik tnsnames.ora.

Listing przedstawia zawartość pliku listener.ora:

```
LISTENER =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS = (PROTOCOL = TCP)(HOST = arek)(PORT = 1521))
      (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC1521))
    )
)
```

Definiowanie łączników do bazy

Dostęp do zdalnej bazy danych w systemie Oracle realizowany jest za pomocą łączników do bazy danych.

Łącznik (ang. database link) umożliwia odwoływanie się do tabel lub perspektyw znajdujących się w zdalnej bazie danych z wykorzystaniem takich poleceń SQL jak: select, update, insert, delete oraz lock table.

Sposób utworzenia przykładowych łączników PD i PN dla przykładowego użytkownika CRM został przedstawiony poniżej:

create database link PD connect to CRM identified by crm using 'PD';

create database link PN connect to CRM identified by crm using 'PN';

Łącznik do bazy PD (z bazy PN) oraz do bazy PN (z bazy PD).

Konfigurowanie strumieni

Aby można było wykorzystać mechanizm Oracle streams należy utworzyć specjalnego użytkownika STRADMIN, który będzie zarządzał mechanizmem replikacji.

Definicja użytkownika zarządzającego replikacją Oracle streams

W tym celu przydzielamy mu nowy obszar na dysku przeznaczony na wszystkie obiekty jego schematu oraz nadajemy mu prawa wymagane do zarządzania replikacją.

```
CREATE  TABLESPACE streams_tbs DATAFILE
' D:\\app\\JJ\\oradata\\arek\\streams_tbs.dbf '
SIZE 100M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;

CREATE USER stradmin IDENTIFIED BY stradmin DEFAULT
TABLESPACE streams_tbs QUOTA UNLIMITED ON streams_tbs;

GRANT DBA TO stradmin;

GRANT EXECUTE ON DBMS_FILE_TRANSFER TO stradmin;
```

```
BEGIN
  DBMS_STREAMS_AUTH.GRANT_ADMIN_PRIVILEGE(
    grantee          => 'stradmin',
    grant_privileges  => TRUE);
END;
```

Łącznik do zdalnej bazy danych

Dla tak utworzonego użytkownika należy zdefiniować łącznik PD do bazy zdalnej PD.

```
CREATE DATABASE LINK PD CONNECT TO STRADMIN IDENTIFIED BY
STRADMIN USING 'PD';
```

Tworzenie kolejek replikacji

W obu bazach danych zostały utworzone dwie kolejki. Pierwsza z nich (związana z procesem CAPTURE) odpowiedzialna jest za przechowywanie zmian dokonanych na lokalnej bazie danych. Druga (związana z procesem APPLY) przechowuje wszystkie operacje, które zostały wykonane w zdalnej bazie danych i przesłane do bazy lokalnej w celu zaaplikowania.

```
DBMS_STREAMS_ADM.SET_UP_QUEUE(
  queue_name=>'capture_queue',
  queue_user=>'STRADMIN');

DBMS_STREAMS_ADM.SET_UP_QUEUE(
  queue_name=>'apply_queue',
  queue_user=>'STRADMIN');
```

Definiowanie procesów replikacji

Kolejnym krokiem jest zdefiniowanie procesów odpowiedzialnych za przechwytywanie zmian w bazie danych PN i umieszczanie ich w kolejce CAPTURE, procesu odpowiedzialnego za propagowanie tych zmian do zdalnej kolejki APPLY bazy danych PD, a także procesu odpowiedzialnego za dokonywanie zmian w lokalnej bazie danych na podstawie informacji zawartych w lokalnej kolejce APPLY.

```
--zdefiniowanie procesów CAPTURE

DBMS_STREAMS_ADM.ADD_TABLE_RULES(
  table_name      =>'crm.program',
  streams_type    =>'sync_capture',
  streams_name    =>'sync_capture',
  queue_name      =>'stradmin.capture_queue');

--propagacja zmian z PN do PD

DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES(
  table_name             =>'crm.program',
  streams_name           =>'send_crm_program',
  source_queue_name      =>'stradmin.capture_queue',
  destination_queue_name  =>'stradmin.apply_queue@PD',
  source_database        =>'PN',
  queue_to_queue         =>TRUE);

--definicja procesów APPLY

DBMS_APPLY_ADM.CREATE_APPLY(
  queue_name      =>'stradmin.apply_queue',
  apply_name      =>'apply_crm_program',
  apply_captured  => FALSE);

--wprowadzanie zmian z kolejki APPLY

DBMS_STREAMS_ADM.ADD_TABLE_RULES(
  table_name      =>'crm.program',
  streams_type    =>'apply',
  streams_name    =>'apply_crm_program',
  queue_name      =>'stradmin.apply_queue',
  source_database =>'PD');
```

Ostatnim krokiem, który należy wykonać jest uruchomienie procesu APPLY odpowiedzialnego za odbieranie danych ze zdalnych baz danych, dzięki czemu nasze środowisko będzie w pełni funkcjonalne. Przedstawiono to na poniższym listingu.

```
DBMS_APPLY_ADM.START_APPLY(
  apply_name  =>'apply_crm_program');
```

Po wykonaniu tych kroków wszystkie operacje wykonane na bazie danych PN są natychmiastowo propagowane do kolejek zdalnej bazy danych PD.

W przypadku gdy jeden z węzłów nie odpowiada operacja pozostaje w kolejce CAPTURE, aż do momentu, w którym węzeł jest dostępny i zmiany zostaną przesłane.