Integracja systemów transakcyjnych

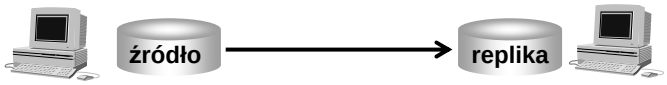
Robert Wrembel
Politechnika Poznańska
Instytut Informatyki
`Robert.Wrembel@cs.put.poznan.pl`
`www.cs.put.poznan.pl/rwrembel`

Replikacja danych

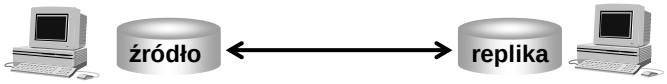
- Problematyka replikacji
- Oracle
  - migawka
  - dziennik migawki
  - grupy odświeżania
- DB2
- SQL Server

Problematyka replikacji danych

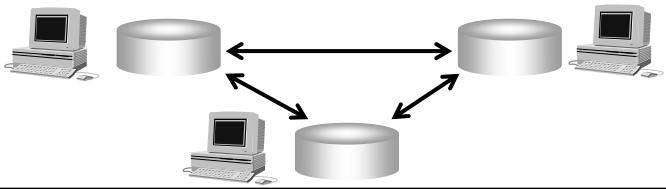
- Replikacja jednokierunkowa



- Replikacja dwukierunkowa



  - Replikacja "mult-imaster"



Problematyka replikacji danych

- Jak wykrywać zmiany?
- Kiedy odświeżać?
- Jak odświeżać?
- Co przesyłać?

Jak wykrywać zmiany?

- Rozwiązania
  - analiza zawartości dziennika transakcji (redo log)
    - okresowo (log scraping)/ na bieżąco (log sniffing)
  - kolumny audytu
    - w tabeli, data i czas operacji, rodzaj operacji
    - wypełnianie: wyzwalacze lub aplikacje
  - dziennik operacji w bazie danych
    - systemowy (np. mview log)
    - implementowany
  - porównanie poprzedniego obrazu źródła z bieżącym
    - niska efektywność
  - wyzwalacze propagujące

Kiedy odświeżać?

- Po zakończeniu transakcji w źródle
- Z opóźnieniem
  - okresowo
  - na żądanie

Jak odświeżać

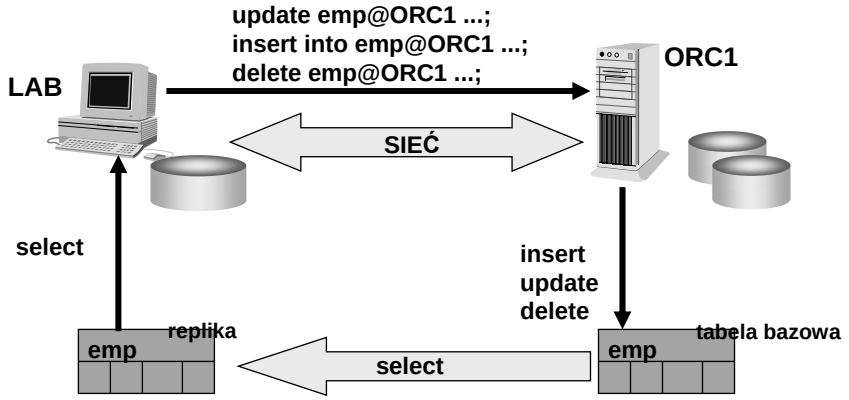
- Przyrostowo
  - problem wykrywania zmian
  - efektywny
- Całkowicie
  - łatwy w implementacji
  - nieefektywny

Co przesyłać

- Dane (data shipping)
- Polecenia (transaction shipping)

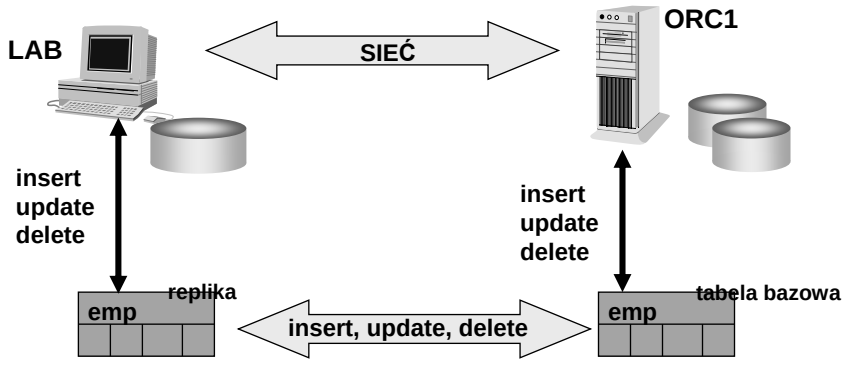
Replikacja danych - Oracle

- **standardowa**



Replikacja danych - Oracle

- **zaawansowana (Oracle Advanced Replication Option)**



Migawka (snapshot)

- kopia tabeli znajdującej się w zdalnej bazie danych
- standardowo tylko do odczytu
- przywileje:
  - CREATE SNAPSHOT, CREATE TABLE, CREATE VIEW
  - CREATE ANY SNAPSHOT
- rodzaje migawek
  - PRIMARY KEY
    - tabela master musi posiadać włączone ograniczenie PRIMARY KEY
    - klauzula SELECT musi zawierać wszystkie atrybuty wchodzące w skład klucza podstawowego tabeli master
  - ROWID
- migawka -> tabela (+ perspektywa) + indeksy

Migawka

```
create snapshot nazwa_migawki
refresh sposób_odświeżania
        start with data_pierwszego_odświeżenia
        next częstotliwość_odświeżania
        with typ_migawki
as zapytanie;
```

- migawka typu prostego
  - bazująca na jednej tabeli master
  - brak klauzul: GROUP BY, CONNECT BY, DISTINCT
  - brak funkcji, połączeń, operatorów zbiorowych
- migawka typu złożonego
- odświeżanie przyrostowe
  - migawka typu prostego
  - zapytanie z połączeniem zastąpione podzapytaniem skorelowanym

Odświeżanie migawki

- **odświeżanie przyrostowe**
  - **migawka typu prostego**
  - **zapytanie z połączeniem zastąpione podzapytaniem skorelowanym**

```
select sk.nazwa, sk.sklep_id
from scott.sklepy@lab81.ii.pp sk,
scott.sprzedaz@lab81.ii.pp sp
where sp.sklep_id=sk.sklep_id
and sp.produkt_id=100
and sp.data='23.01.2002'
and sp.l_sztuk=2;
```

```
select sk.nazwa, sk.sklep_id
from scott.sklepy@lab81.ii.pp sk
where exists
        (select sp.sklep_id
         from scott.sprzedaz@lab81.ii.pp sp
         where sp.sklep_id=sk.sklep_id
         and sp.produkt_id=100
         and sp.data='23.01.2002'
         and sp.l_sztuk=2);
```


Odświeżanie migawki

- **odświeżanie przyrostowe**
  - **migawka wyliczająca agregaty: count, sum, avg, variance, stdev**
    - **dziennik utworzony z klauzulą including new values**
    - **dziennik zawiera wszystkie atrybuty wymienione po select, również będące argumentami wywołania f. grupowych**
    - **count zawsze wyliczany w zapytaniu, gdy wyliczne sum, avg, variance, stdev**

```
create materialized view 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;
```

Tworzenie migawki

```
CREATE SNAPSHOT [schemat.]migawka
[ parametry_fizyczne ]
[ TABLESPACE nazwa_przestrzeni ]
[ USING INDEX
                [ parametry_fizyczne ]
                [ TABLESPACE nazwa_przestrzeni ] ]
[ REFRESH { FAST | COMPLETE | FORCE } ]
[ WITH { PRIMARY KEY | ROWID } ]
[ START WITH 'data' ]
[ NEXT 'data' ]
[ USING [ LOCAL ROLLBACK SEGMENT rbs ]
        [ MASTER ROLLBACK SEGMENT rbs ] ]
AS SELECT ...;
```

```
CREATE MATERIALIZED VIEW [schemat.]nazwa ...
```


Odświeżanie migawki

- **sposób odświeżania**
  - **REFRESH FAST -> odświeżanie przyrostowe**
    - **dla migawek prostych**
    - **musi istnieć SNAPSHOT LOG dla tabeli master**
  - **REFRESH COMPLETE -> odświeżanie pełne**
  - **REFRESH FORCE -> automatyczny wybór metody odświeżania; jeżeli możliwe to Oracle wybiera FAST**
- **okres odświeżania**
  - **START WITH -> data pierwszego odświeżenia**
  - **NEXT -> wyrażenie określające częstotliwość odświeżania**

Odświeżanie automatyczne

- musi być wyspecyfikowany parametr NEXT
- określenie częstoliwości odświeżania
  - REFRESH FAST START WITH sysdate NEXT sysdate+1
  - REFRESH FAST NEXT sysdate+1
- włączenie procesu odpowiedzialnego za odświeżanie
  - parametr konfiguracyjny JOB_QUEUE_PROCESSES -> wartość {1, ..., 36}, domyślnie 0
  - procesy drugoplanowe

Odświeżanie ręczne

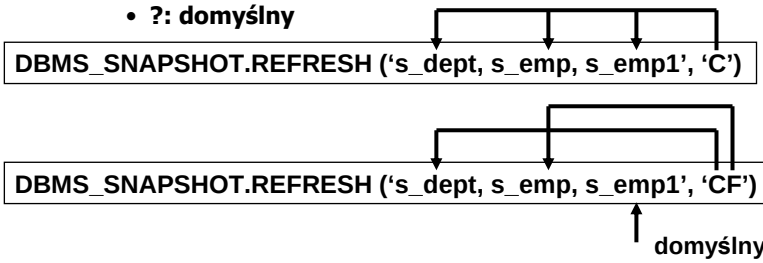
- wyłączenie procesu odpowiedzialnego za odświeżanie
  - JOB_QUEUE_PROCESSES =0
- brak parametru NEXT
  - REFRESH FAST START WITH sysdate
  - migawka odświeżona raz, w momencie jej tworzenia
- pakiet DBMS_SNAPSHOT
- pakiet DMBS_MVIEW (synonim do DBMS_SNAPSHOT)

Odświeżanie ręczne

- **procedura DBMS_SNAPSHOT.REFRESH**

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

  - **sn_1, sn_2, ..., sn_n: migawki**
  - **metoda: metoda odświeżania**
    - **f lub F: FAST**
    - **c lub C: COMPLETE**
    - **?: domyślny**



Moment odświeżania

```
refresh
    {fast | complete | force}] [{on demand | on commit}]
    [start with data_pierwszego_odświeżenia]
    [next częstotliwość_odświeżania]
```

- **on commit można stosować jedynie, gdy:**
  - **zapytanie korzysta z tabel lokalnych**
  - **migawek opartych o jedną tabelę, bez wyliczania agregatów**
  - **migawek, których zapytanie wyznacza agregaty w oparciu o pojedynczą tabelę**
  - **migawek których zapytanie wykorzystuje łączenie tabel, ale bez wyliczania agregatów**
- **brak odświeżania**

```
create materialized view mv_test
never refresh
as select * from user1.sklepy@dbl1;
```

Przykład

```
create snapshot sn_emp
pctfree 30 pctused 30
storage
    (initial 10K next 10K pctincrease 0 minextents 1 maxextents 10)
tablespace usr
refresh fast
start with sysdate+(1/(24*60))
next sysdate+(1/(24*60*6))
with rowid
using local rollback segment rb1 master rollback segment rb04
as select * from emp@lab.world;
```

odświeżanie co 10 sek.

```
SNAP$_SN_EMP    -> TABLE
I_SNAP$_SN_EMP  -> INDEX
SN_EMP          -> VIEW
```

zawiera kolumnę M_ROW$$ przechowującą ROWID rekordów tabeli emp@lab.world

na kolumnie SNAP$_SN_EMP.M_ROW$$

Moment wypełnienia danymi

- **build immediate**
- **build deferred**

```
create snapshot mv_sprzedaz_1
build deferred
refresh force
start with sysdate + (1/(24*6))
next sysdate+(1/(24*60))
with primary key
as
    select produkt_id, l_sztuk, cena_jedn, data, sklep_id
    from user1.sprzedaz@dbl1
    where sklep_id=1;
```

Modyfikowanie migawki

```
ALTER SNAPSHOT [schemat.]migawka
[ parametry_fizyczne ]
[ USING INDEX [ parametry_fizyczne ]
[ REFRESH { FAST | COMPLETE | FORCE } ]
                                    [{on demand | on commit]}
[ WITH PRIMARY KEY ]
[ START WITH 'data' ]
[ NEXT 'data' ]
[ USING  MASTER ROLLBACK SEGMENT rbs ];
```

- **parametry_fizyczne**
  - **bloku: PCTFREE, PCTUSED (nie dla indeksu), INITRANS, MAXTRANS**
  - **rozszerzeń: STORAGE**
    - **NEXT, MINEXTENTS, MAXEXTENTS, PCTINCREASE**

```
alter snapshot sn_emp1
pctfree 20 pctused 40 initrans 4
storage (next 20K pctincrease 0
         minextents 1
         maxextents 20)
refresh complete
start with sysdate
next sysdate+1/(24*60*10)
with primary key
using master rollback segment rb03;
```


Dziennik migawki (snapshot log)

- **tabela związana z tabelą bazową migawki**
- **przechowuje zmiany dokonane na danych tabeli bazowej**
- **wykorzystywany do odświeżania przyrostowego**
- **tworzenie:**

```
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 | excludign new values }];
```

Dziennik migawki (2)

- **WITH PRIMARY KEY: dla rekordów uaktualnionych wartości atrybutów wchodzących w skład klucza są rejestrowane w dzienniku**
- **WITH ROWID: ROWID rekordów uaktualnionych rejestrowane w dzienniku**
- **WITH PRIMARY KEY, ROWID: w dzienniku rejestrowane zarówno wartości atr. kluczowych, jak i ROWID**

- **kolumna filtrująca: atrybut występujacy w klauzuli WHERE zapytania definiującego migawkę**
- **including new values**
  - **konieczne dla migawek odświeżanych przyrostowo zawierających agregaty**

```
select sk.nazwa, sk.sklep_id
from scott.sklepy@lab81.ii.pp sk
where exists
        (select sp.sklep_id
         from scott.sprzedaz@lab81.ii.pp sp
         where sp.sklep_id=sk.sklep_id
         and sp.produkt_id=100
         and sp.data='23.01.2002'
         and sp.l_sztuk=2)
```


Dziennik migawki (3)

```
create materialized view 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;
```

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

Przykład

```
create snapshot log on scott.emp
pctfree 30 pctused 30 initrans 2 maxtrans 10
storage
    (initial 10K next 10K pctincrease 0 minextents 1 maxextents 10)
tablespace lab_dane
with primary key, rowid;
```

```
MLOG$_EMP
Name                 Type
-------------------- ------------
EMPNO                NUMBER(4)
M_ROW$$              VARCHAR2(255)
SNAPTIME$$           DATE
DMLTYPE$$            VARCHAR2(1)
OLD_NEW$$            VARCHAR2(1)
CHANGE_VECTOR$$      RAW(255)
```


Przykład (2)

```
insert into emp values (1000, 'BOND', 'MANAGER', NULL, '01-JAN-99',
                        6000, 500, 30);
```

```
update emp set comm=comm+300 where empno=7839;
```

```
delete from emp where empno=7698;
```

```
select * from mlog$_emp;
EMPNO M_ROW$$              SNAPTIME$ DML    OLD_  CHANGE_
                                     TYPE$$ NEW$$ VECTOR$$
----- ------------------ --------- ------ ----- --------
1000 AAAApTAACAAAAn5AAD 01-JAN-00 I      N     FEFF
7839 AAAApTAACAAAAn5AAA 01-JAN-00 U      U     8000
7698 AAAApTAACAAAAn5AAB 01-JAN-00 D      O     0000
```

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 | excludign new values}];
```

Usuwanie migawki

```
DROP SNAPSHOT [schemat.]migawka;
```

Usuwanie dziennika migawki

```
DROP SNAPSHOT LOG ON [schemat.]tabela;
```


Informacje o migawkach

- **USER_SNAPSHOTS, ALL_SNAPSHOTS, DBA_SNAPSHOTS**

```
select name, table_name, master_owner,
       master, master_link, refresh_method,
       type, master_rollback_seg
from dba_snapshots;
```

NAME	TABLE_NAME	MASTER OWNER	MASTER	MASTER LINK	REFRESH METHOD	TYPE	MASTER RBS
SN_EMP	SNAP$_SN_EMP	SCOTT	EMP	@LAB.WORLD	ROWID	FAST	
SN_EMP1	SNAP$_SN_EMP1	SCOTT	EMP	@LAB.WORLD	PRIMARY KEY	COMPLETE	RB04

Informacje o dziennikach migawek

- **USER_SNAPSHOT_LOGS, ALL_SNAPSHOT_LOGS, DBA_SNAPSHOT_LOGS**

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

```
LOG    MASTER     LOG_TABLE    ROWIDS PRIMARY FILTER CURRENT   SNAPS.
OWNER                                 KEYS    COLS.  SNAPS.    ID
------ ---------- ------------ ------ ------- ------ --------- --------
SCOTT  EMP        MLOG$_EMP    YES    YES     NO     25-JAN-00       57
SCOTT  EMP        MLOG$_EMP    YES    YES     NO     25-JAN-00       58
```


Informacje o odświeżaniu migawek

- **USER_SNAPSHOT_REFRESH_TMIES, ALL_SNAPSHOT_REFRESH_TIMES, DBA_SNAPSHOT_REFRESH_TIMES**

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

```
OWNER NAME       MASTER_OWNER MASTER LAST_REFRESH
----- ---------- ------------ ------ -------------------
DEMO  MV_SKLEPY  USER1        SKLEPY 12.02.2002:18:05:00
```

Informacje o zarejestrowanych migawkach w bazie master

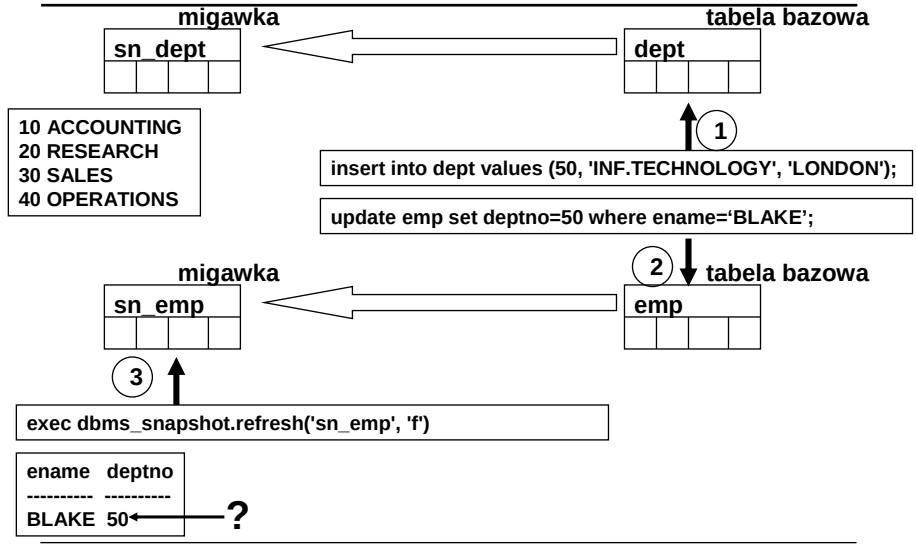
➲ **DBA_REGISTERED_SNAPSHOTS**

```
select owner, name, snapshot_site, can_use_log, updatable,
      refresh_method,snapshot_id
from user_registered_snapshots
where name='MV_SPRZEDAZ';
```

```
OWNER NAME        SNAPSHOT_SITE CAN UPD REFRESH_MET SNAPSHOT_ID
----- ----------- ------------- --- --- ----------- -----------
DEMO  MV_SPRZEDAZ DMINE.II.PP   YES NO  PRIMARY KEY          45
```

```
select sl.master "Master table", sl.log_table, rs.name as "Snp.name"
from dba_snapshot_logs sl, dba_registered_snapshots rs
where sl.snapshot_id=rs.snapshot_id;
```


Wiele niezależnych migawek - problem



Grupy odświeżania (refresh groups)

- odświeżane jednocześnie
- spójność danych migawek

Tworzenie grupy odświeżania

```
DBMS_REFRESH.MAKE
  ( name,
    list,
    next_date,
    interval,
    implicit_destroy,
    rollback_seg )
```

- name, — nazwa grupy
- list, — lista migawek przypisywanych do grupy;
- next_date, — data następnego odświeżenia
- interval, — okres odświeżania
- implicit_destroy, — TRUE: usunięcie grupy jeżeli nie zawiera migawek (zob. SUBTRACT) domyślnie FALSE
- rollback_seg) — rbs wykorzystywany do odświeżania

- lista migawek
  - migawki muszą być w tej samej bd
  - mogą być w różnych schematach
  - max. 100 migawek w grupie

Tworzenie grupy odświeżania(2)

```
exec DBMS_REFRESH.MAKE
  (name => 'orc1.rg_dept_emp', -
   list => 'orc1.sn_dept, orc1.sn_emp', -
   next_date => sysdate+(1/48), -
   interval => 'next_day(trunc(sysdate), ''FRIDAY'')+10/24', -
   implicit_destroy => TRUE, -
   rollback_seg => 'rb1')
```

Dodanie migawki do grupy

```
exec DBMS_REFRESH.ADD ('orc1.rg_dept_emp', 'orc1.sn_emp1')
```

Usunięcie migawki z grupy

```
exec DBMS_REFRESH.SUBTRACT('orc1.rg_dept_emp', 'orc1.sn_emp1')
```

Zmiana parametrów grupy

```
DBMS_REFRESH.CHANGE
  ( name,
    next_date,
    interval,
    implicit_destroy,
    rollback_seg )
```

```
exec DBMS_REFRESH.CHANGE
( name => 'orc1.rg_dept_emp', -
  next_date => sysdate+(1/(48*60)), -
  interval => 'next_day(trunc(sysdate), "SATURDAY")+8/24', -
  implicit_destroy => FALSE, -
  rollback_seg => 'rb0')
```

Manualne odświeżanie grupy

```
exec DBMS_REFRESH.REFRESH('orc1.rg_dept_emp')
```

Usunięcie grupy odświeżania

- usuwa grupę z migawkami lub pustą

```
exec DBMS_REFRESH.DESTROY('orc1.rg_dept_emp')
```


Informacje nt. utworzonych grup

- USER_REFRESH, ALL_REFRESH, DBA_REFRESH

```
select rowner, rname, refgroup, implicit_destroy, rollback_seg,
       next_date, interval, broken
from user_refresh;
```

```
                          Impl.  Rollb.
ROWNER RNAME       REFGROUP  destr. segm.  NEXT_DATE INTERVAL                 Broken
------ ----------- --------- ------ ------ --------- ----------------------- ------
ORC1   RG_DEPT_EMP       96  N      RB0    26-JAN-00 next_day(trunc(sysdate),  N
                                                     'SATURDAY')+8/24
```

- jeżeli automatyczne odświeżanie stało się niemożliwe:
  - proces odświeżający wykonuje 16 prób odświeżenia w pewnych odstępach czasu
    - jeżeli 16-ta próba niepomyślna ustawiana wartość BROKEN=Y
    - po usunięciu problemu odświeżenie manualne (BROKEN=N) -> przywrócenie odświeżania automatycznego

Informacje nt. migawek w grupie

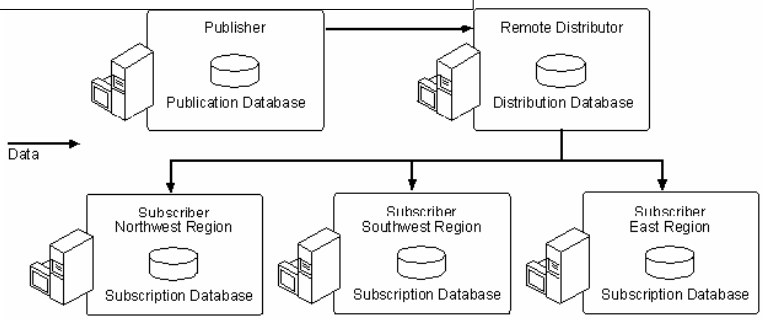
- **USER_REFRESH_CHILDREN, ALL_REFRESH_CHILDREN, DBA_REFRESH_CHILDREN**

```
select owner, name, type, rowner, rname, refgroup
from user_refresh_children;
```

OWNER	NAME	TYPE	ROWNER	RNAME	REFGROUP
ORC1	SN_DEPT	SNAPSHOT	ORC1	RG_DEPT_EMP	96
ORC1	SN_EMP	SNAPSHOT	ORC1	RG_DEPT_EMP	96

Replikacja danych w SQL Server - Model replikacji

- **Publisher - wydawca/publikator**
- **Distributor - dystrybutor**
- **Subscriber - subskrybent**
- **Publication - publikacja**



Model replikacji

- **Publikator**
  - **Serwer udostępniający dane do replikacji innym serwerom**
  - **Każdy publikator może udostępniać jedną lub więcej publikacji**
  - **Zawiera informacje o wszystkich publikacjach w danym węźle**
- **Dystrybutor**
  - **Serwer z „dystrybucyjną" bazą danych (distribution database), danymi do replikacji, informacjami o transakcjach i metadanymi**

Publikacja

- **Zawiera przynajmniej jeden artykuł**
- **Jest replikowana jako całość**

Artykuł

- Obiekt bazy danych przeznaczony do replikacji
- Artykuł może być
  - całą tabelą
  - podzbiorem kolumn tabeli
  - podzbiorem wierszy tabeli
  - procedurą składowaną
  - definicją perspektywy

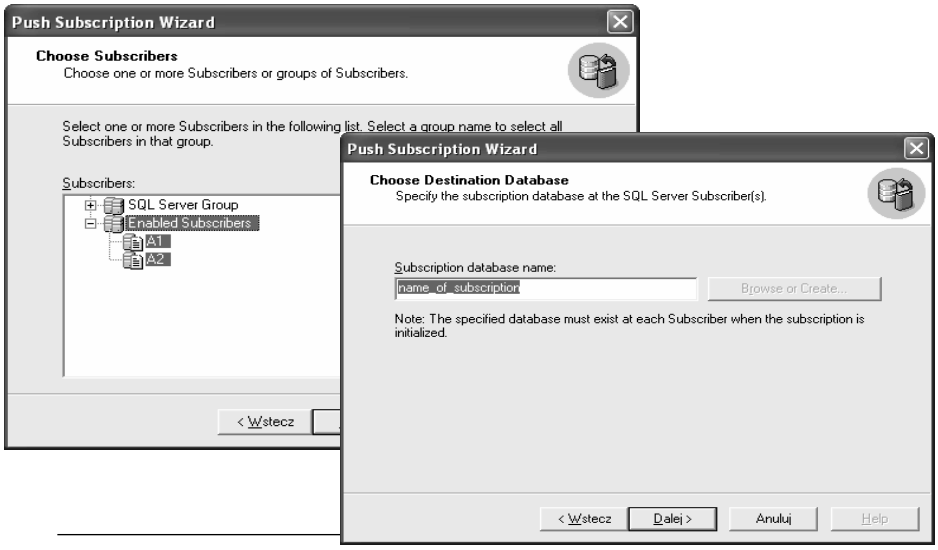
Subskrybent

- Subskrybent
  - Węzeł, do którego są replikowane dane
  - Subskrybenci zapisują się na całe publikacje, a nie na pojedyncze artykuły
  - Mogą propagować zmiany w danych z powrotem do publikatora lub udostępniać dane innym subskrybentom

Subskrypcja

- Definicja zawiera
  - replikowane publikacje
  - moment replikowania
  - miejsce docelowe replikowania - wskazania do subskrybentów
- Rodzaje subskrypcji
  - push subscription
  - pull subscription
- Ta sama publikacja może być wykorzystana w obu rodzajach subskrypcji

Subskrypcja



Subskrypcja

Agenci

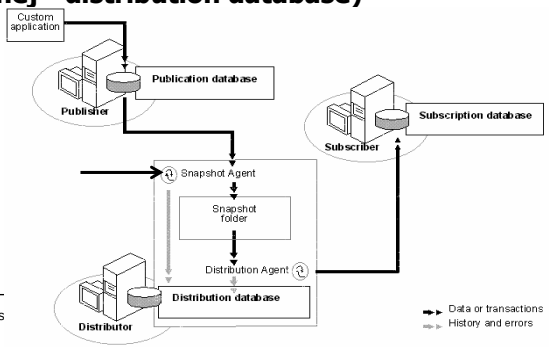
- Wykonują zadania związane z
  - zarządzaniem replikacją
  - kopiowaniem
  - dystrybuowaniem danych
- Rodzaje
  - SQL Server Agent
  - Snapshot Agent
  - Log Reader Agent
  - Distribution Agent
  - Merge Agent

SQL Server Agent

- Zarządza agentami używanymi w replikacji
- Kontroluje i monitoruje inne operacje wykonywane poza replikacją
  - utrzymywanie logów błędów
  - uruchamianie innych procesów

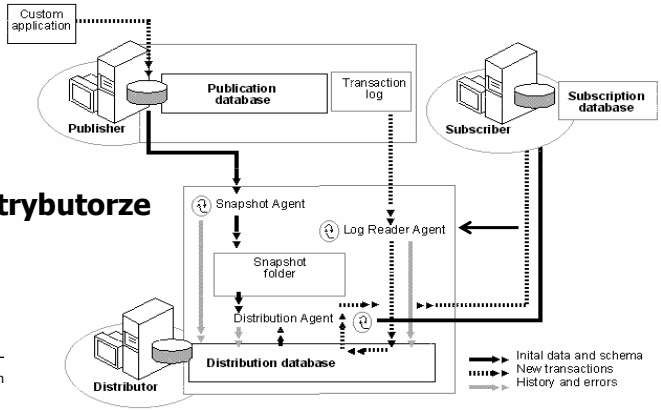
Snapshot Agent

- Używany z każdym typem replikacji
- Działa na węźle dystrybutora
- Uruchamiany przez SQL Server Agent
- Zadania
  - przygotowuje pliki migawek
  - zapisuje informacje o synchronizacji (przechowywane w bazie dystrybucyjnej - distribution database)



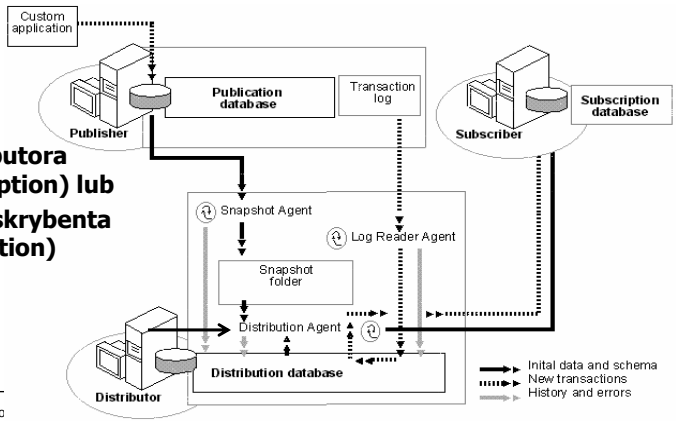
Log Reader Agent

- Używany w replikacji transakcyjnej
- Przesyła informacje o transakcjach do replikacji z pliku logu z węzła publikatora do bazy dystrybucyjnej
- Działa na dystrybutorze



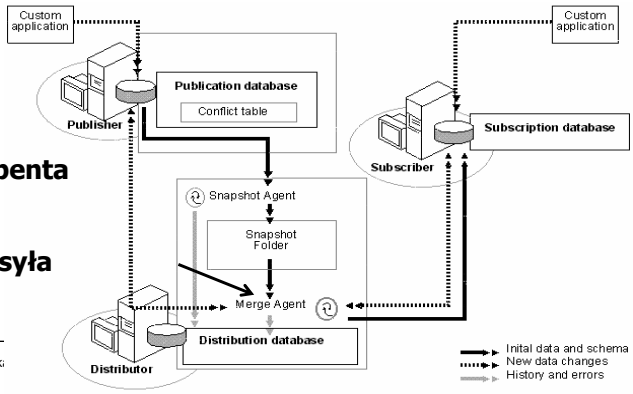
Distribution Agent

- Używany w replikacji migawkowej i transakcyjnej
- Odpowiedzialny za przekazanie migawek i transakcji z bazy dystrybucyjnej do subskrybentów
- Działa na
  - węźle dystrybutora (push subscription) lub
  - na węźle subskrybenta (pull subscription)



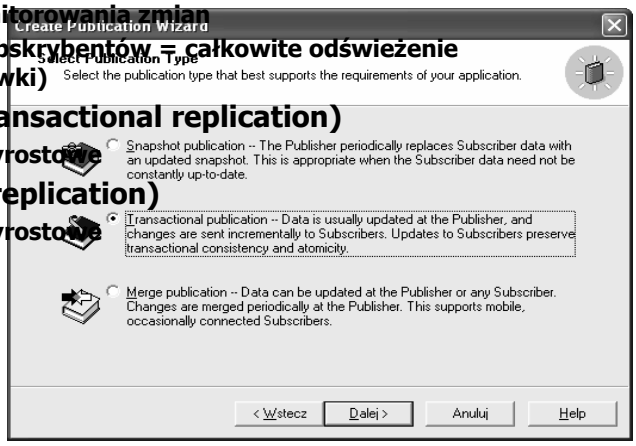
Merge Agent

- Używany w replikacji scalanej
  - Przekazuje początkową migawkę do subskrybentów
  - Łączy przyrostowe zmiany, które wystąpiły po tym przekazaniu
- Każda subskrypcja ma swojego agenta, który łączy się z subskrybentem i publikatorem uaktualniając obie BD
- Pobiera i przesyła zmiany z subskrybenta do publikatora
- Pobiera zmiany z publikatora i przesyła do subskrybenta



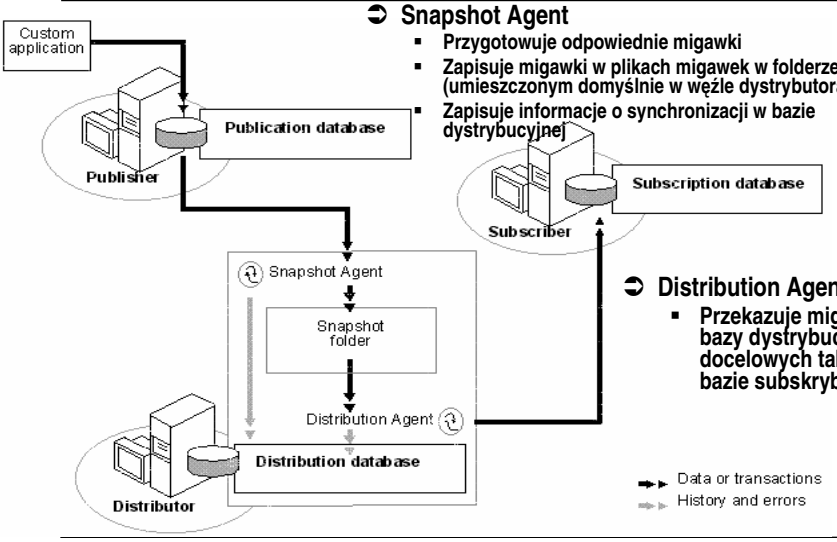
Typy replikacji

- Migawkowa (snapshot replication)
  - Dane są replikowane „z danej chwili”
  - Nie wymaga monitorowania zmian
  - Uaktualnienie subskrybentów = całkowite odświeżenie (przesłanie migawki)
- Transakcyjna (transactional replication)
  - Odświeżanie przyrostowe
- Scalana (merge replication)
  - Odświeżanie przyrostowe



Replikacja migawkowa

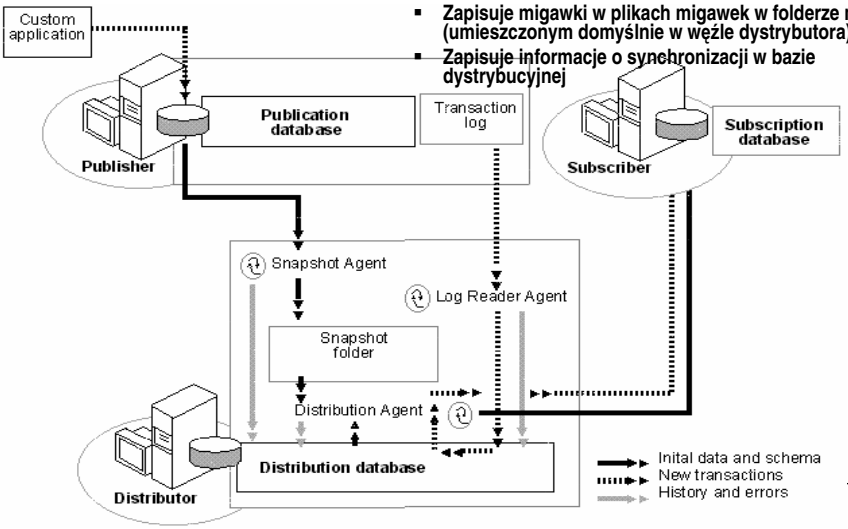
- **Snapshot Agent**
  - Przygotowuje odpowiednie migawki
  - Zapisuje migawki w plikach migawek w folderze migawek (umieszczonym domyślnie w węźle dystrybutora)
  - Zapisuje informacje o synchronizacji w bazie dystrybucyjnej



- **Distribution Agent**
  - Przekazuje migawkę z bazy dystrybucyjnej do docelowych tabel w bazie subskrybenta

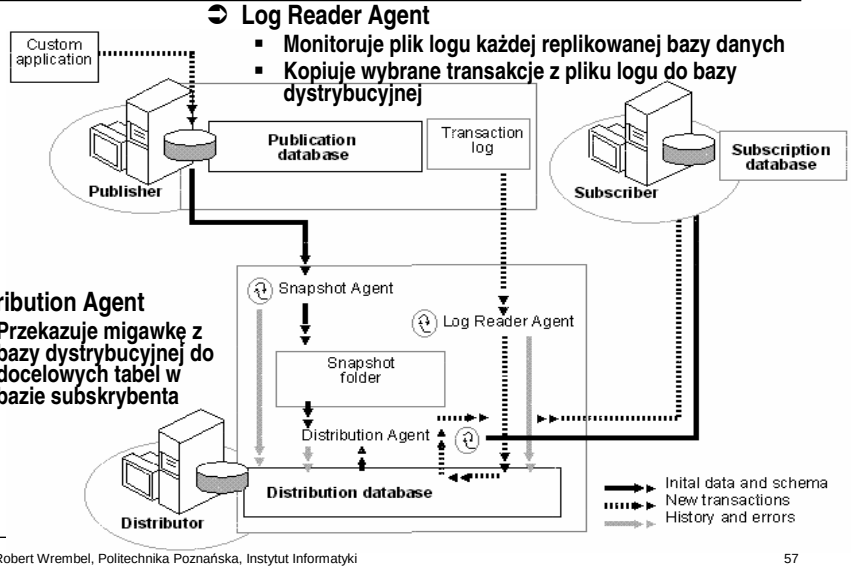
Replikacja transakcyjna

- **Snapshot Agent**
  - Przygotowuje odpowiednie migawki
  - Zapisuje migawki w plikach migawek w folderze migawek (umieszczonym domyślnie w węźle dystrybutora)
  - Zapisuje informacje o synchronizacji w bazie dystrybucyjnej



Replikacja transakcyjna

- Log Reader Agent
  - Monitoruje plik logu każdej replikowanej bazy danych
  - Kopiuje wybrane transakcje z pliku logu do bazy dystrybucyjnej



- Distribution Agent
  - Przekazuje migawkę z bazy dystrybucyjnej do docelowych tabel w bazie subskrybenta

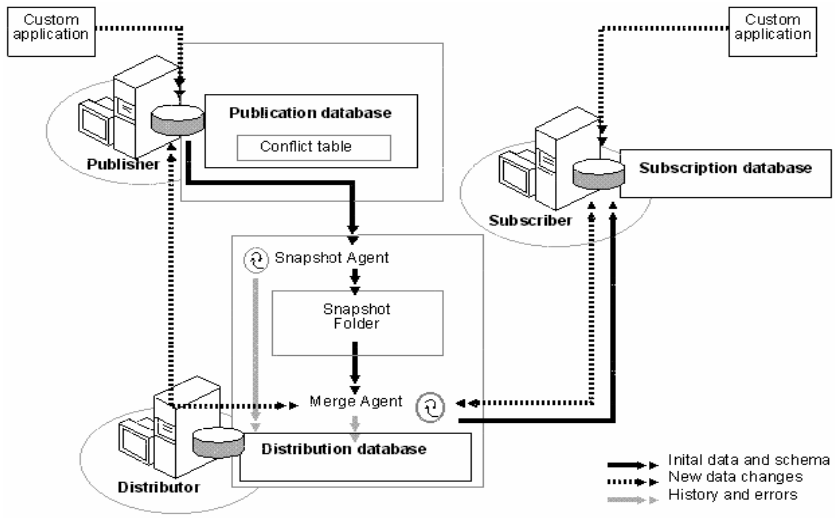
Replikacja scalana

- Początkowo migawka danych przesyłana jest do subskrybentów
- System śledzi zmiany w publikatorze i w subskrybentach
- Dane są synchronizowane na żądanie lub w określonym czasie
- Modyfikacje są dokonywane niezależnie na wielu serwerach
- Możliwe konflikty podczas łączenia modyfikacji
- W razie konfliktu Merge Agent inicjuje proces wybierania ostatecznej wersji danych

Replikacja scalana

- **Snapshot Agent**
  - **Przygotowuje odpowiednie migawki**
  - **Zapisuje migawki w plikach migawek w folderze migawek (umieszczonym domyślnie w węźle dystrybutora)**
  - **Zapisuje informacje o synchronizacji w dystrybucyjnej bd**
  - **Tworzy specyficzne dla tej replikacji procedury, wyzwalacze i tabele systemowe**
- **Merge Agent**
  - **Przesyła początkową migawkę do subskrybentów**
  - **Łączy zmiany przyrostowe w danych, które wystąpiły od czasu przesłania migawki**
  - **Rozwiązuje konflikty (niespójność danych)**

Replikacja scalana

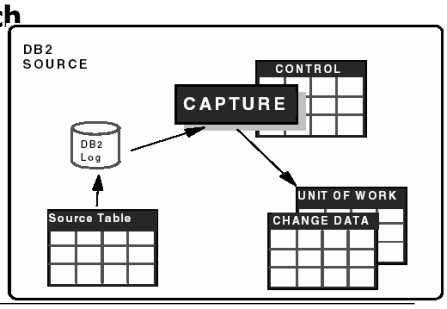


DB2 - Replikacja SQL

- CAPTURE – proces wykrywający zmiany w tabelach źrodłowych
- APPLY – proces wprowadzający wykryte zmiany w replikach
- MONITOR – proces wykorzystywany do monitorowania replikacji

CAPTURE

- Capture wykrywa/pobiera zmiany z logu używanego do odtwarzania i zarządzania transakcjami
- Każdej tabeli źródłowej odpowiada tabela zmian CHANGE DATA (CD) tworzona przy rejestracji tabeli źródłowej
- Tabela UNIT OF WORK przechowuje informacje o zatwierdzonych transakcjach

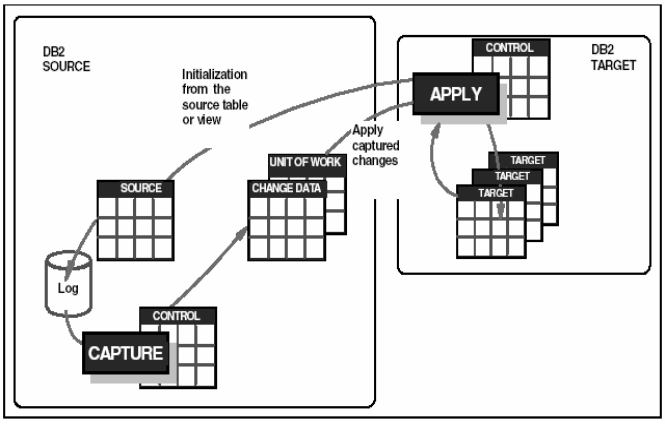


CAPTURE

- Tabela kontrolna
  - informacje nt. które tabele źródłowe monitorować (replikować)
  - informacje do komunikacji z procesem APPLY
- Tabele CHANGE DATA, UNIT OF WORK i kontrolne muszą się znajdować w tej samej bazie danych co replikowana tabela

APPLY

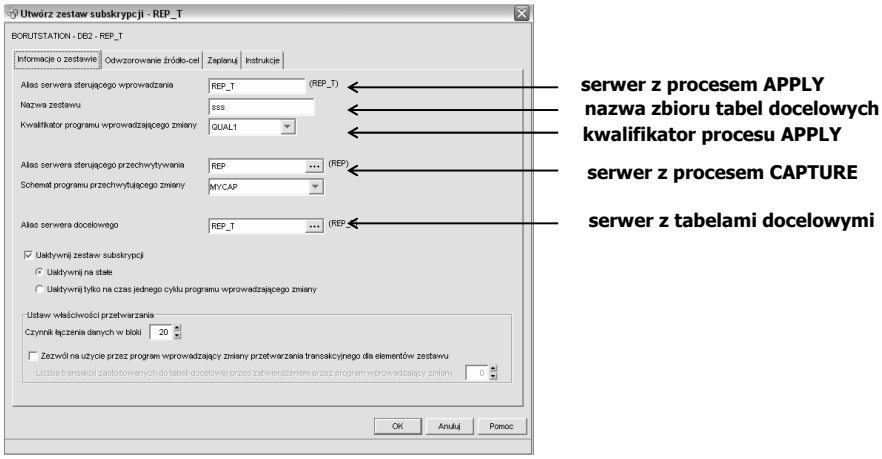
- Może działać na dowolnym serwerze, ale musi mieć dostęp do serwerów źródłowych i docelowych (zalecane na serwerze docelowym)
- Wszystkie tabele kontrolne procesu APPLY znajdują się w <u>jednym</u> schemacie – schemat ASN



APPLY

- Na początku replikacji odświeżanie pełne
- APPLY czyta zmiany zgodnie z ustalonym harmonogramem (np. co 10s) lub w reakcji na zdefiniowane zdarzenia w tabeli kontrolnej ASN.IBMSNAP_SUBS_EVENT
- Definiuje się zbiory tabel docelowych (subscription set) na których działa APPLY i każdemu zbiorowi przypisuje się kwalifikator (apply qualifier)
- Każdy zbiór tabel docelowych (subscription set) składa się z mapowań tabel źródłowych na docelowe
- Dla każdego zbioru możemy zdefiniować dodatkowe transformacje (procedury), które zostaną wykonane na replikowanych danych przed wczytaniem ich do tabel docelowych

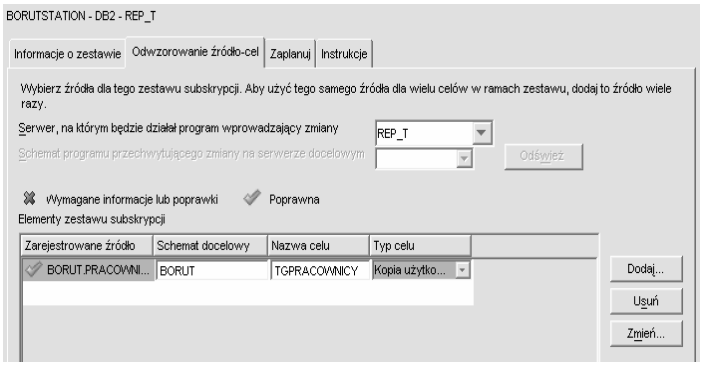
Definiowanie zbiorów subskrypcji



Powiązanie serwerów źródłowych/docelowych oraz instancji procesów CAPTURE (schemat) i APPLY (kwalifikator)

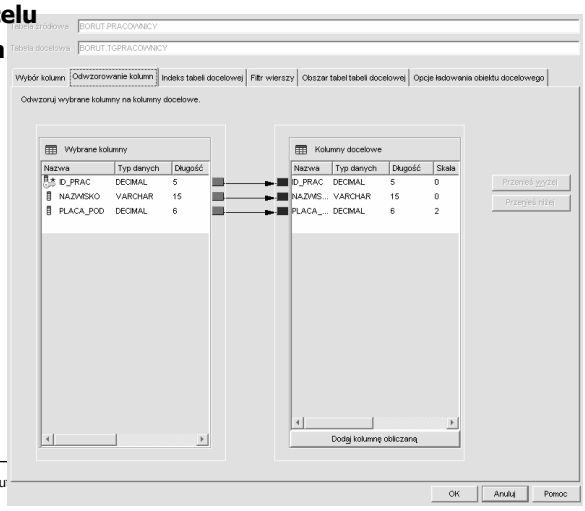
Definiowanie odwzorowań źródło-cel

- Dla zarejestrowanej tabeli źródłowej tworzymy tabelę docelową i wybieramy dla niej schemat
- Typy celów:
  - user copy
  - aggregate tables
  - point in time
  - replica



Definiowanie odwzorowań źródło-cel

- Wybór kolumn ze źródeł oraz celów i ich odwzorowanie
- Utworzenie indeksu na celu
- Możliwość zdefiniowania klauzuli where



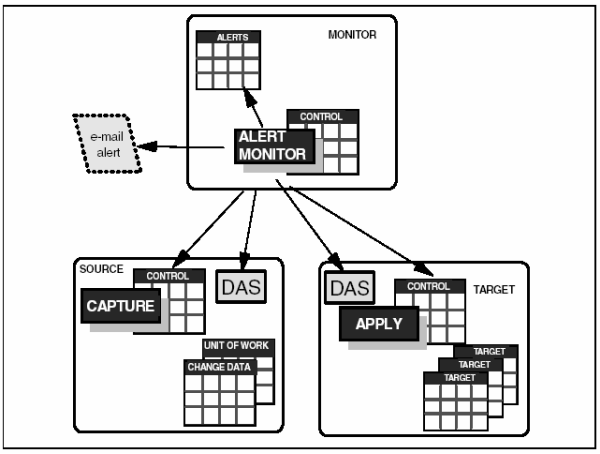
Plan wykonania replikacji

- Określenie w czasie punktu uruchomienia
- Interwały replikacji lub działanie ciągłe
- Replikacja uruchamiana zdarzeniami
- Możliwość określenia procedur składowanych transformujących dane
  - wykonywane przed/po każdym cyklu replikacji

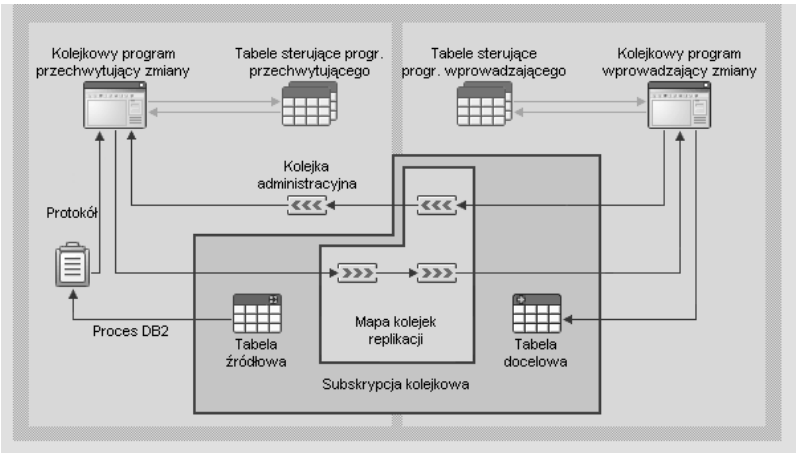
ALERT MONITOR

- Jego zadaniem jest monitorowanie tabel kontrolnych procesów CAPTURE i APPLY
  - Status procesów CAPTURE i APPLY
  - Komunikaty o błędach z tabel kontrolnych
  - Progi opóźnień replikacji
  - Zużycie pamięci
  - Odrzucone transakcje
- Możliwość zdefiniowania przez administratora zdarzeń i progów generujących alarmy
- Wynik monitorowania zapisywany w tabeli kontrolnej ASN.IBMSNAP_ALERTS

ALERT MONITOR



Replikacja kolejkowa



- **System IBM WebSphere MQ**

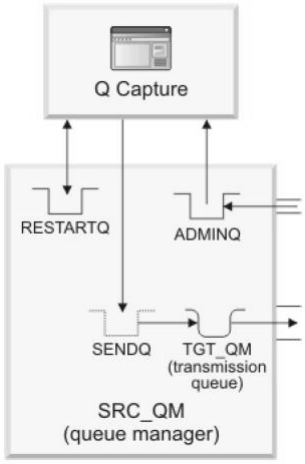
Replikacja kolejkowa

- Połączenia pomiędzy bazami danych nie są utrzymywane przez cały czas
- Trwałość komunikatów nawet przy awarii serwerów
- Minimalizacja opóźnienia (dane przepisywane natychmiast po odczytaniu z logu)

Replikacja kolejkowa

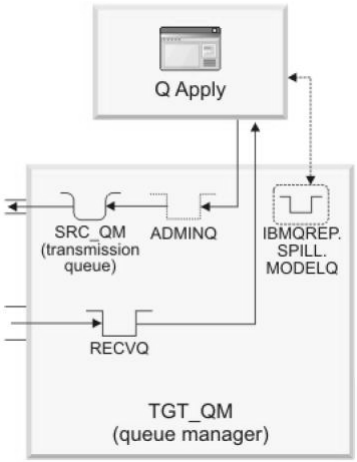
- Zmiany umieszczane są i propagowane przez system kolejek
- Kolejka administracyjna - przesyła sygnały sterujące
- Reguła przechwytywania wierszy:
  - tylko pełne odświeżanie – brak przechwytywania zmian
  - przechwytywanie zmian we wszystkich kolumnach
  - przechwytywanie zmian w wybranych (zarejestrowanych) kolumnach
  - filtrowanie operacji (np. ignorowanie operacji delete)
- Replikowane porcje danych przesyłane są jako jedna transakcja WebSphere MQ (w źrodle jest to wiele transakcji)

Szczegóły komunikacji – Q Capture



- Send queue – wysyłanie replikowanych danych
- Admin queue – odbiór komunikatów kontrolnych od Q Apply
- Restart queue – przechowuje pojedynczy komunikat o miejscu rozpoczęcia analizy logu

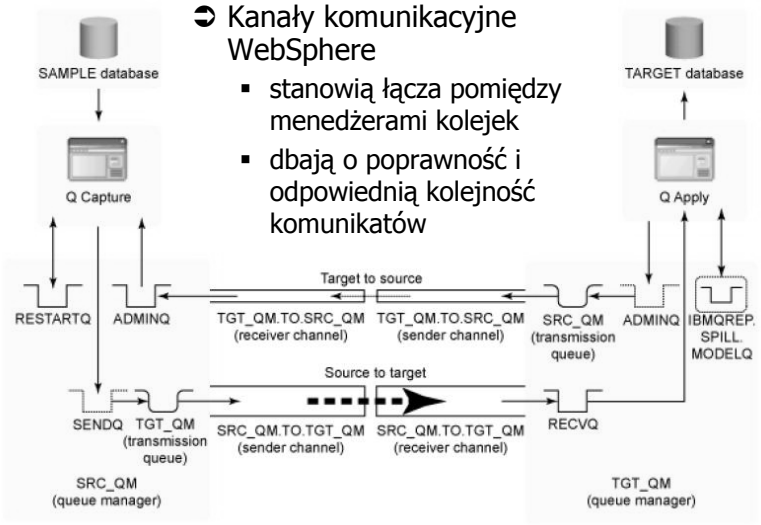
Szczegóły komunikacji – Q Apply



- Recieve queue – odbiór replikowanych danych
- Admin queue – wysyłanie komunikatów kontrolnych do Q Capture
- Spill queue – bufor przechowujący te wiadomości od Q Capture, które przychodzą podczas ładowania docelowej bazy danych

Szczegóły komunikacji – WebSphere MQ

- Kanały komunikacyjne WebSphere
  - stanowią łącza pomiędzy menedżerami kolejek
  - dbają o poprawność i odpowiednią kolejność komunikatów



Rodzaje replikacji kolejkowej

- **Jednokierunkowa**
  - **jedno źródło, wiele serwerów docelowych**
  - **możliwość filtrowania**
- **Dwukierunkowa**
  - **tylko dwa serwery (primary i secondary)**
  - **jeden z pary konfigurowany jako consistent winner**
  - **stosowana w przy małym prawdopodobieństwie konfliktów**
- **Peer-to-peer**
  - **wiele serwerów na równorzędnych prawach**
  - **konflikty roztrzygane na podstawie znaczników czasowych**
- **Replikacja do obcej bazy danych (np. Oracle)**