Rozproszone transakcje z wykorzystaniem usługi Service Broker w SQL Server 2008 R2

Andrzej Ptasznik

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI
Warszawska Wyższa Szkoła Informatyki
UNIA EUROPEJSKA
EUROPEJSKI FUNDUSZ SPOŁECZNY

Service Broker - kilka ogólników

SQL Service Broker zapewnia rozproszoną, asynchroniczną infrastrukturę pozwalającą na tworzenie rozbudowanych aplikacji korporacyjnych.

Service Broker jest mechanizmem kolejkowania komunikatów dostępnym w SQL Server 2008.

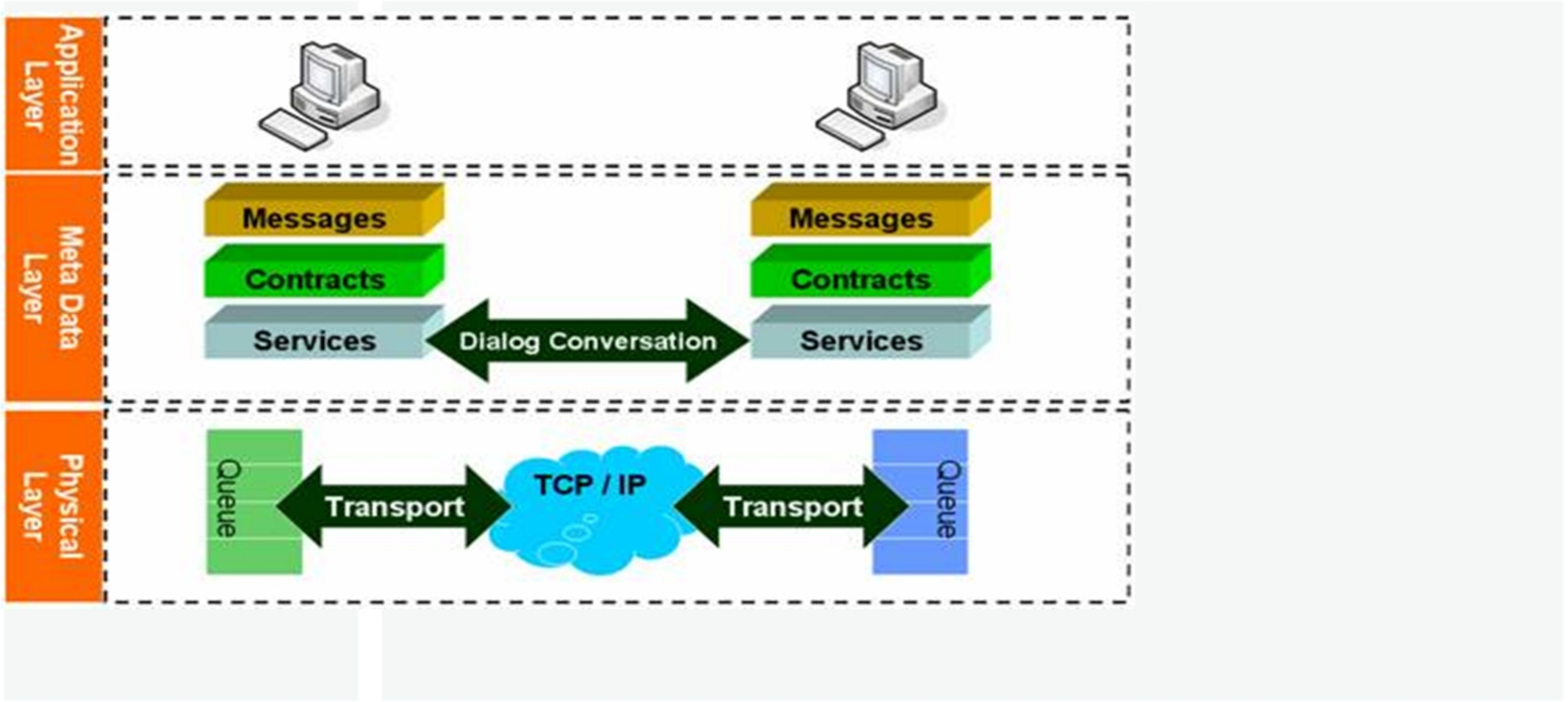
- Asynchroniczne przekazywanie danych
- Wyzwalacze rozproszone
- Poprawia wydajność
- Poprawia skalowalność

Service Broker - elementy architektury

Elementy architektury Service Broker

- Komunikaty
- Usługi
- Kolejki
- Konwersacje
- Kontrakty
- Węzły końcowe Service Broker
- Zdalne wiązanie usług
- Trasy

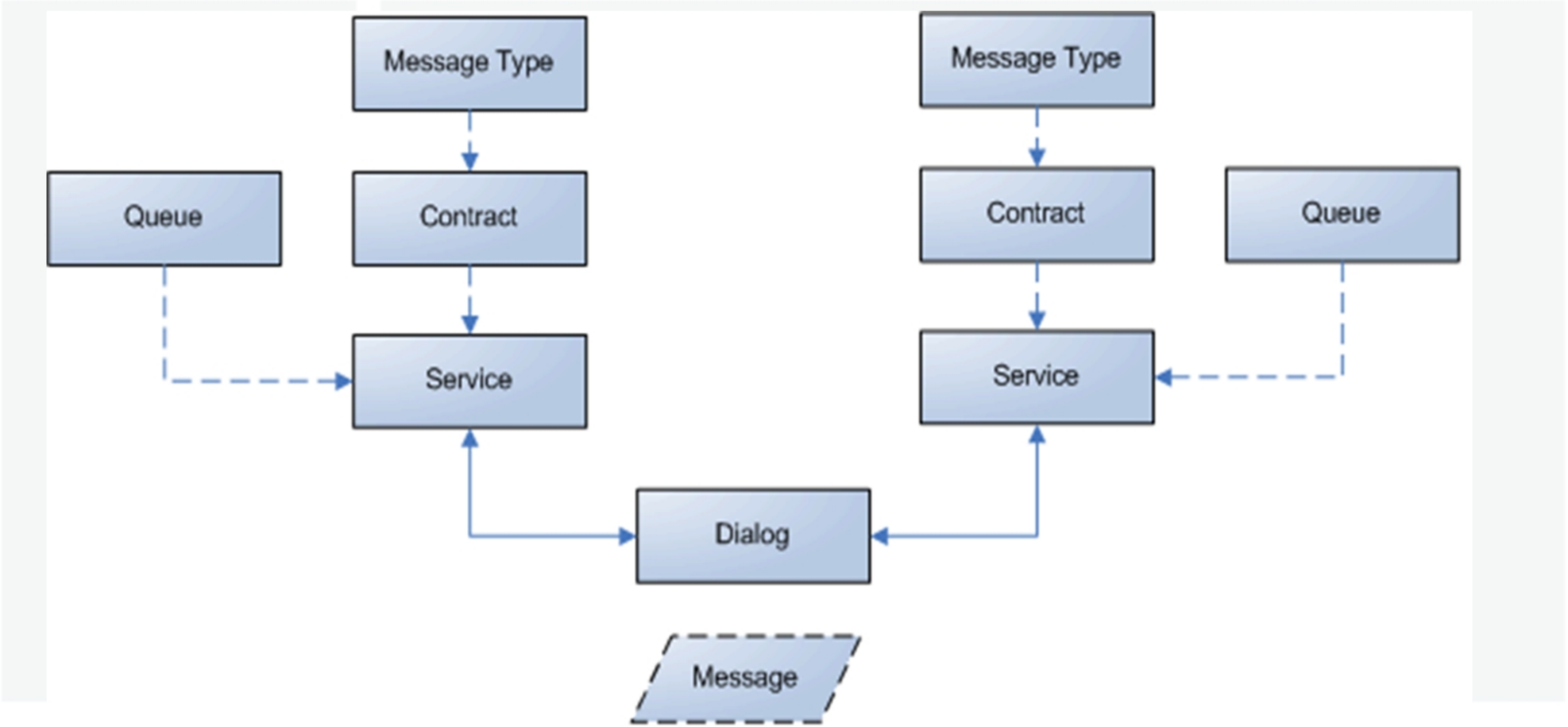
Architektura Service Broker



Źródło : sqlservercentral.com

Architektura Service Broker

Zależności pomiędzy obiektami architektury Service Broker



Źródło : diybl.com

Definiowanie elementów architektury Service Broker

Elementy architektury Service Broker – Typy komunikatów

```
CREATE MESSAGE TYPE message_type_name
[ AUTHORIZATION owner_name ]
 [ VALIDATION =
            { NONE |
              EMPTY |
            WELL_FORMED_XML |
            VALID_XML WITH SCHEMA COLLECTION schema_collection_name
} ] [ ; ]
```

Przykład :

```
CREATE MESSAGE TYPE Zmiany
     VALIDATION = WELL_FORMED_XML;
```

Definiowanie elementów architektury Service Broker

Elementy architektury Service Broker – Kontrakty

```
CREATE CONTRACT contract_name
[ AUTHORIZATION owner_name ]
 ( { { message_type_name | [ DEFAULT ] }
  SENT BY { INITIATOR | TARGET | ANY } } [ ,...n] ) [ ; ]
```

- **Kontrakt definiuje typy komunikatów, które może wysyłać uczestnik konwersacji**

- **Kontrakt musi zawierać co najmniej jeden typ komunikatu.**
- **Przykład :**

```
CREATE CONTRACT [Replikacja]
([Potwierdzenie] SENT BY TARGET,
  [Zmiany] SENT BY INITIATOR)
```

Definiowanie elementów architektury Service Broker

Elementy architektury Service Broker - kolejki

```
CREATE QUEUE <object>
  [ WITH
    [ STATUS = { ON | OFF } [ , ] ]
    [ RETENTION = { ON | OFF } [ , ] ]
    [ ACTIVATION (
       [ STATUS = { ON | OFF } , ]
        PROCEDURE_NAME = <procedure> ,
        MAX_QUEUE_READERS = max_readers ,
        EXECUTE AS { SELF | 'user_name' | OWNER }
        ) [ , ] ]
    [ POISON_MESSAGE_HANDLING (
      [ STATUS = { ON | OFF } )
    ]
    [ ON { filegroup | [ DEFAULT ] } ]
[ ; ]
```

Definiowanie elementów architektury Service Broker

Elementy architektury Service Broker – zawartość kolejki

Column name	Data type	Description
status	**tinyint**	**0**=Received message **1**=Ready **2**=Not yet complete **3**=Retained sent message
priority	**tinyint**	Reserved for future use.
queuing_order	**bigint**	Message order number within the queue.
conversation_group_id	**uniqueidentifier**	Identifier for the conversation group that this message belongs to.
conversation_handle	**uniqueidentifier**	Handle for the conversation that this message is part of.
message_sequence_number	**bigint**	Sequence number of the message within the conversation.
service_name	**nvarchar(512)**	Name of the service that the conversation is to.
service_id	**int**	SQL Server object identifier of the service that the conversation is to.
service_contract_name	**nvarchar(256)**	Name of the contract that the conversation follows.
service_contract_id	**int**	SQL Server object identifier of the contract that the conversation follows.
message_type_name	**nvarchar(256)**	Name of the message type that describes the message.
message_type_id	**int**	SQL Server object identifier of the message type that describes the message.
validation	**nchar(2)**	Validation used for the message. **E**=Empty **N**=None **X**=XML
message_body	**varbinary(MAX)**	Content of the message.
message_id	**uniqueidentifier**	Unique identifier for the message.

Definiowanie elementów architektury Service Broker

Elementy architektury Service Broker - usługi

- Usługa jest węzłem końcowym dla konwersacji
  - Usługa inicjująca
  - Usługa docelowa
- Każda usługa jest powiązana z jedną kolejką

```
CREATE SERVICE service_name
[ AUTHORIZATION owner_name ]
ON QUEUE [ schema_name. ]queue_name
[ ( contract_name | [DEFAULT] [ ,...n ] ) ] [ ; ]
```

Definiowanie elementów architektury Service Broker

Elementy architektury Service Broker – obsługa dialogu
Inicjowanie dialogu

```
DECLARE @dialog_handle  uniqueidentifier
BEGIN DIALOG [ CONVERSATION ] @dialog_handle
     FROM SERVICE initiator_service_name
     TO SERVICE 'target_service_name' [ , { 'service_broker_guid' | 'CURRENT
                                                                  DATABASE' } ]
 [   ON CONTRACT contract_name ]
    [ WITH [ { RELATED_CONVERSATION = related_conversation_handle |
         RELATED_CONVERSATION_GROUP = related_conversation_group_id } ] [ [ ,
]
         LIFETIME = dialog_lifetime ] [ [ , ]
         ENCRYPTION = { ON | OFF } ] ] [ ; ]
```

Definiowanie elementów architektury Service Broker

Elementy architektury Service Broker – obsługa dialogu
Wysłanie komunikatu do kolejki

SEND ON CONVERSATION *conversation_handle*
[**MESSAGE TYPE** *message_type_name*]
[**(** *message_body_expression* **)**] [**;**]

Definiowanie elementów architektury Service Broker

Elementy architektury Service Broker – obsługa dialogu
Odczytanie komunikatu z kolejki

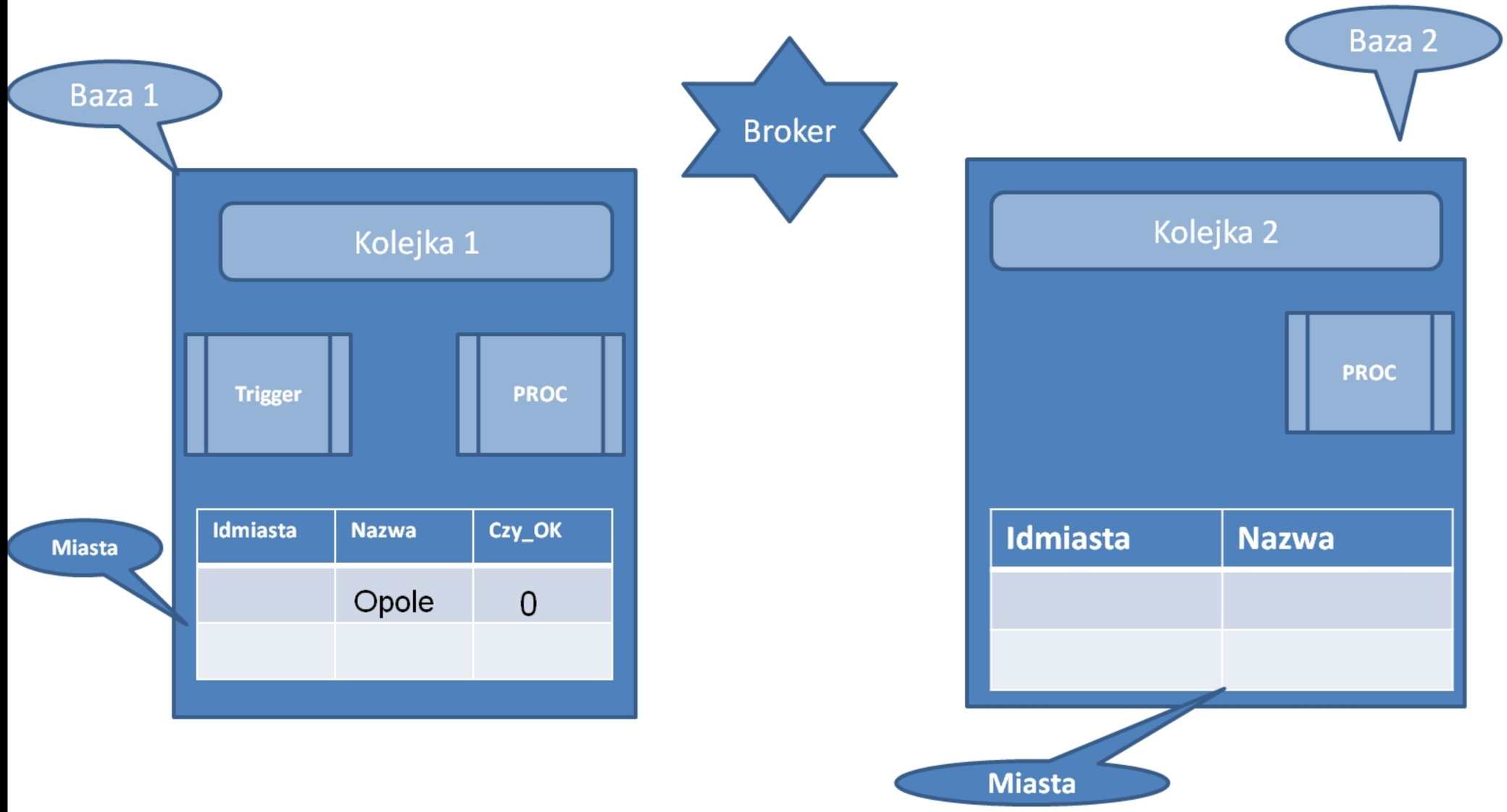
```
[ WAITFOR (]
    RECEIVE [ TOP ( n ) ] <column_specifier> [ ,...n ]
    FROM <queue> [ INTO table_variable ]
    WHERE { conversation_handle = conversation_handle |
                conversation_group_id = conversation_group_id } ] [
               ) ] [ , TIMEOUT timeout ]
[ ; ]
```

Definiowanie elementów architektury Service Broker

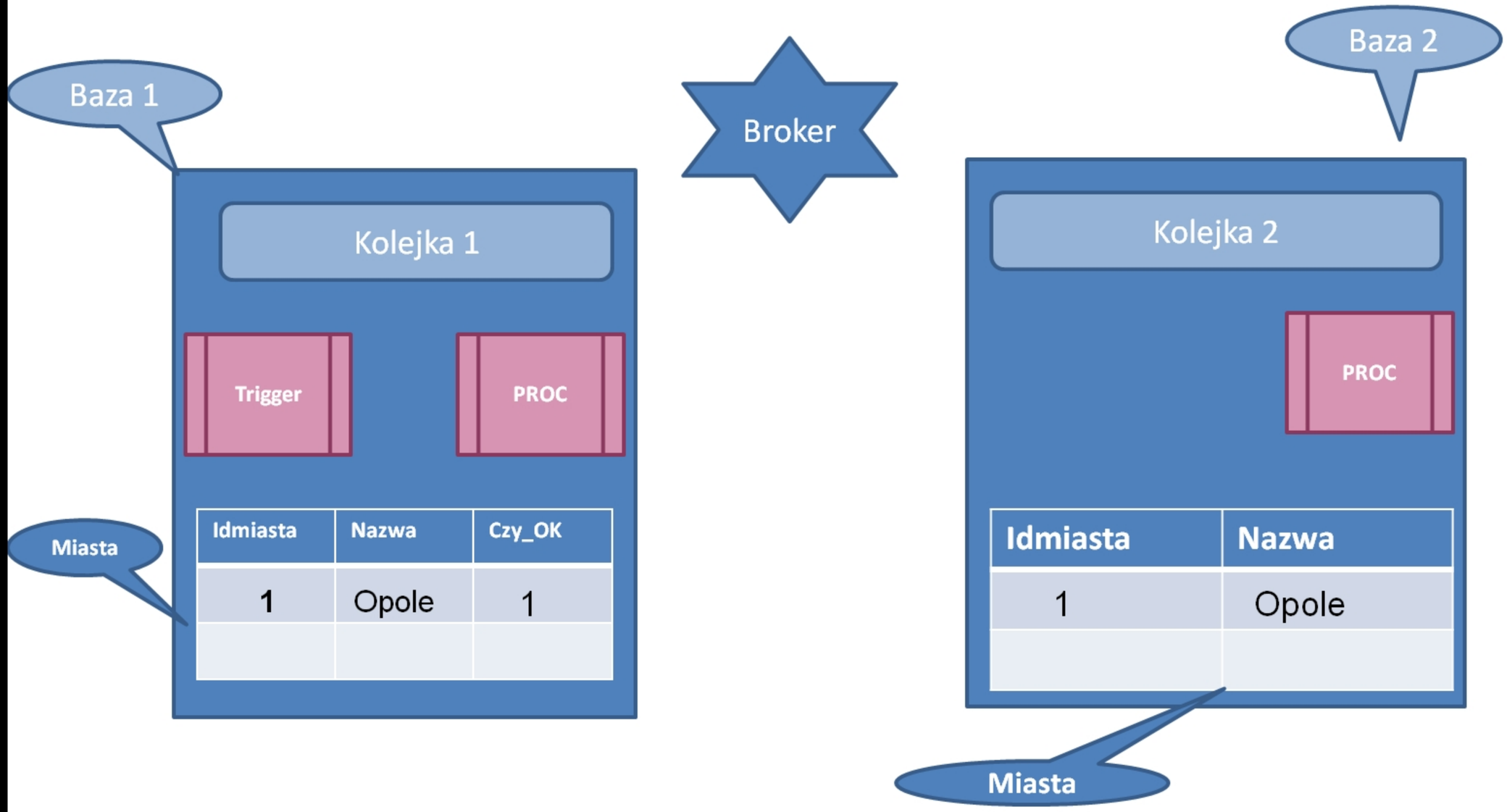
Elementy architektury Service Broker – obsługa dialogu
Zakończenie dialogu

```
END CONVERSATION conversation_handle
    [ [ WITH ERROR = failure_code
                DESCRIPTION = 'failure_text' ] |
    [ WITH CLEANUP ] ] [ ; ]
```

Mini replikacja - przykład



Mini replikacja - przykład



Prezentacja replikacji dla wielu subskrybentów z dystrybutorem

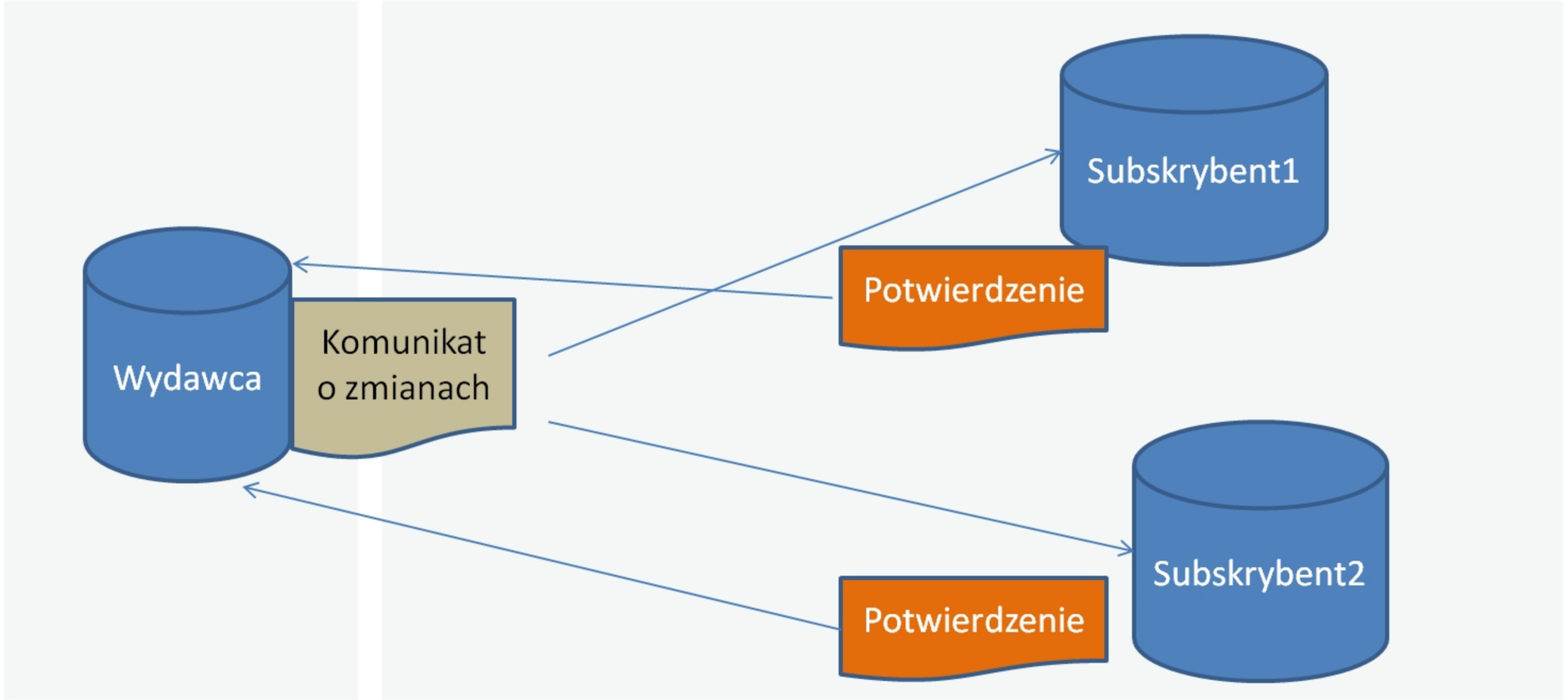
Założenia:

1. Jeden wydawca i wielu subskrybentów

1. Wprowadzony wiersz w bazie wydawcy – przekazywany jest do pozostałych uczestników

1. Wydawca rejestruje potwierdzenie dokonania zmian u subskrybentów

Prezentacja replikacji dla wielu subskrybentów



Definiowanie modelu elementy Service Broker

```
CREATE MESSAGE TYPE [Potwierdzenie]  ] VALIDATION = WELL_FORMED_XML

CREATE MESSAGE TYPE [Zmiany]  ] VALIDATION = WELL_FORMED_XML

CREATE CONTRACT [Replikacja]
([Potwierdzenie] SENT BY ANY,
[Zmiany] SENT BY ANY)

CREATE QUEUE [dbo].[Kolejka]
WITH STATUS = ON ,
RETENTION = OFF ,
ACTIVATION
 (  STATUS = ON , PROCEDURE_NAME = [dbo].[ObslugaKolejki] ,
MAX_QUEUE_READERS = 5 , EXECUTE AS OWNER  ),
POISON_MESSAGE_HANDLING (STATUS = OFF)
 ON [PRIMARY]

CREATE SERVICE [ObslRepl]
ON QUEUE [dbo].[Kolejka] ([Replikacja])
```

Definiowanie modelu - tabela

W bazie danych Wydawcy jest tabela o nazwie Miasta o podanej strukturze

Column Name	Data Type	Allow Nulls
id	uniqueidentifier	☐
Nazwa	varchar(50)	☐
OK1	bit	☐
OK2	bit	☐

Definiowanie modelu - wyzwalacz

Dla tabeli Miasta zdefiniowany został wyzwalacz :

```
Create trigger [dbo].[TR_Repl]
on [dbo].[Miasta]
after insert, update,delete
as
declare @operacja char(1)=

case
   when not exists (select * from inserted) and not exists (select * from deleted)
then 'P'
   when not exists (select * from inserted)  then 'D'
  when not exists (select * from deleted)  then 'I'
 else 'U'
end

declare @Guid  uniqueidentifier
```

Definiowanie modelu – wyzwalacz cd.

Dla tabeli Miasta zdefiniowany został wyzwalacz :

```
if @operacja!='P'
begin
    if @operacja in ('I','D') or
    (
       @operacja='U' and exists (select *
                                 from inserted as i join deleted as d
                                    on i.id=d.id
                                   where i.nazwa!=d.nazwa )
    )
    begin

                        declare  @komunikat xml=
                        (
                         Select @operacja as Operacja,
                                 (Select  coalesce(i.id,d.id) as Id,
                                          coalesce(i.nazwa,d.nazwa) as Nazwa
                                 from inserted as i full join deleted as d
                                 on i.id=d.id
                                for xml path('Wiersz'),root('Wiersze'),type  )
                         For xml path('Zmiany')
                        );

                        insert into logx(baza,dane)
                        select DB_NAME(), @komunikat;
```

Definiowanie modelu – wyzwalacz cd.

```
begin dialog  @Guid
                    from service ObslRepl
                    to service 'S1ObslRepl'
                    on contract Replikacja
                    WITH
    ENCRYPTION = OFF;

                    send on conversation @guid
                    message type Zmiany
                    (@komunikat);

                    begin dialog  @Guid
                    from service ObslRepl
                    to service 'S2ObslRepl'
                    on contract Replikacja
                    WITH
    ENCRYPTION = OFF;

                    send on conversation @guid
                    message type Zmiany
                    (@komunikat);
        end
end
```

Definiowanie modelu – procedura u sybskrybentów

```
Create procedure [dbo].[ObslugaKolejki]
as
declare @guid uniqueidentifier,
        @typ sysname,
        @komunikat xml
begin try
  begin transaction
 ;waitfor(
   receive top (1)  @guid=conversation_handle,
                    @typ= message_type_name,
                   @komunikat=cast(message_body as xml)

   from kolejka
  ) ,timeout 6000;
```

Definiowanie modelu – procedura u sybskrybentów cd.

```
if @typ='Zmiany'
   begin
      insert into wydawca.dbo.logx(baza,dane)
     select db_name(), @komunikat  ;
      with tmp as
     ( Select k.value('../../Operacja[1]', 'char(1)') as operacja,
                k.value('Nazwa[1]', 'varchar(50)') as nazwa,
                k.value('Id[1]', 'uniqueidentifier') as Id
       from @komunikat.nodes('Zmiany/Wiersze/Wiersz') as t(k)
     )
     merge miasta
     using  tmp
     on miasta.id=tmp.id
     when matched and Operacja='D' then delete
     when matched and Operacja='U' then update set nazwa=tmp.nazwa
     when not matched then insert(id,nazwa) values(tmp.id,tmp.nazwa);
```

Definiowanie modelu – procedura u sybskrybentów cd.

```
set @komunikat.modify('insert <Baza>Subskrybent1</Baza>
                       as first into (Zmiany)[1]');

    send on conversation      @guid
    message type    Potwierdzenie
    (@komunikat)


  end
   commit

end try

begin catch
 if @@trancount>0
   rollback
end catch
```

Definiowanie modelu – procedura u Wydawcy

```
Create procedure [dbo].[ObslugaKolejki]
as
declare @guid uniqueidentifier,
        @typ sysname,
        @komunikat xml
begin try
  begin transaction

   ;waitfor
  (  receive top (1)  @guid=conversation_handle,
                      @typ= message_type_name,
                     @komunikat=cast(message_body as xml)

   from kolejka ), Timeout 6000;

   insert into logx(baza,dane)
   select 'Wydek', @komunikat;
```

Definiowanie modelu – procedura u Wydawcy cd.

```
if @typ='Potwierdzenie'
   begin
      with tmp as
      ( Select k.value('../../Baza[1]', 'varchar(128)') as Baza,
                k.value('Id[1]', 'uniqueidentifier') as Id
        from @komunikat.nodes('Zmiany/Wiersze/Wiersz') as t(k)
      )


      update Miasta set
      Ok1=case
             when  tmp.baza='Subskrybent1' then 1 else Ok1
            end,
       Ok2=case
             when tmp.baza='Subskrybent2' then 1 else Ok2
            end
      from tmp
      where tmp.id=miasta.id
      ;
 end conversation      @guid      WITH CLEANUP
```

Definiowanie modelu – procedura u Wydawcy cd.

```
 end
  commit tran

end try

begin catch
 if @@trancount>0
   rollback
end catch
```

Przejdźmy do pokazu ☺