STUDIA INFORMATICA 2009
Volume 30 Number 2A (83)

Katarzyna HARĘŻLAK
Politechnika Śląska, Instytut Informatyki

REPLIKACJA DANYCH NA URZĄDZENIA PRZENOŚNE

Streszczenie. W pracy analizowano możliwość zastosowania mobilnych serwerów baz danych do stworzenia środowiska rozproszonej bazy danych z wykorzystaniem mechanizmów replikacji danych. Szczególną uwagę zawrócono na aspekty związane z inicjalizacją repliki na urządzeniu przenośnym oraz cykliczną jej synchronizację z serwerem centralnym. Badania prowadzone były w środowisku wykorzystującym narzędzia firmy Microsoft: SQL Server 2005 oraz SQL Server Mobile.

Słowa kluczowe: replikacja danych, mobilne bazy danych, synchronizacja, rozwiązywanie konfliktów aktualizacji danych, MS SQL Server

DATA REPLICATION ON MOBILE DEVICES

Summary. The possibility of mobile devices usage for data replication was analyzed in the paper. The problems of replica initialization and periodical synchronization were considered particularly. The research was performed in environment consisting of Microsoft tools: SQL Server 2005 and SQL Server Mobile.

Keywords: data replication, mobile databases, synchronization, data modification conflict resolution, MS SQL Server

1. Wstęp

Wszędzie tam, gdzie dostęp do klasycznego komputera PC czy laptopa jest niemożliwy – swoje zastosowania znajdują urządzenia Pocket PC, zbliżające się obecnie możliwościami do swoich poprzedników. Procesor taktowany zegarem 612 MHz, 64 MB pamięci RAM i coraz lepsza grafika, a przy tym wielkość i waga takiego urządzenia porównywalne z tabliczką czekolady, to wszystko sprawia, że zwiększa się przestrzeń zastosowania tego typu urządzeń przenośnych. Producenci oprogramowania coraz częściej tworzą aplikacje mobilne będące

„przedłużeniem” aplikacji stacjonarnych, do których zaliczyć można pakiet Office czy przeglądarkę Internet Explorer. Jednak, oprócz ogólnie przydatnych aplikacji biurowych, pojawiają się także specjalizowane programy wspomagające sprzedawców, konsultantów, techników czy inżynierów spędzających większość czasu w terenie. Coraz łatwiej można nosić ze sobą spore zbiory danych, a wzrost mocy obliczeniowych powoduje, że przenośne urządzenia mogą spełniać coraz więcej funkcji zarezerwowanych dla dużych komputerów.

Również producenci serwerów baz danych nie pozostali w tyle i wielu z nich udostępnia mobilne wersje swoich produktów. Można tu wspomnieć firmę Microsoft z serwerem SQL Server Mobile, firmę Sybase z produktem UltraLite czy firmę Oracle, która dostarcza serwer OracleLite. Narzędzia te wprowadzają nową jakość przy tworzeniu rozproszonej bazy danych, w szczególności w zakresie replikacji bazy danych

W pracy przedstawione zostaną zagadnienia związane z replikacją danych na urządzenia przenośne z wykorzystaniem narzędzi firmy Microsoft. Zagadnienia te zaprezentowane zostaną na przykładzie prostego systemu obsługi kliniki lekarskiej.

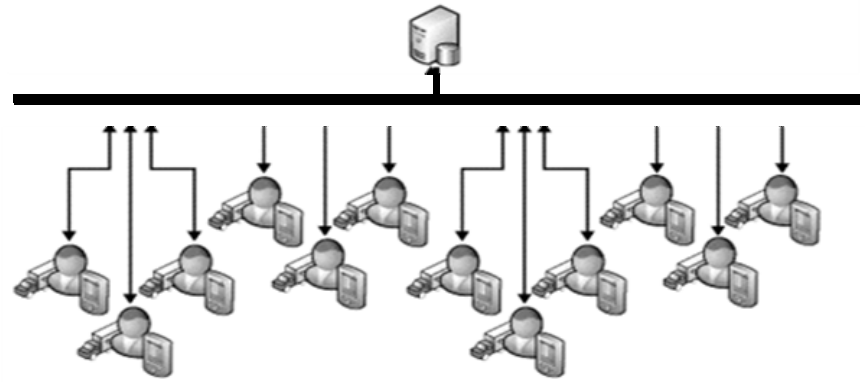
2. Replikacja danych

Replikacja danych jest jednym z mechanizmów rozproszonej bazy danych, polegającym na powielaniu danych w wielu węzłach sieci komputerowej. Uzyskuje się dzięki temu większą niezawodność systemu, gdyż, w przypadku wystąpienia awarii w jednej z lokalizacji, dane można dostarczyć z innych funkcjonujących serwerów. Ponadto, dzięki istnieniu danych w miejscu zadawania zapytania, znacznie skraca się czas oczekiwania na odpowiedź. Ma to również wpływ na zmniejszenie ruchu w sieci komputerowej. Kosztem, który trzeba zapłacić za takie rozwiązanie, jest znacznie bardziej złożona modyfikacja danych.

Replikacja danych może przebiegać na kilka sposobów. Strategie aktualizacji kopii danych, utworzonych z wykorzystaniem różnych typów replikacji, charakteryzują się różnymi cechami w zależności od funkcji, które mają realizować, i wymagań systemów, w których są implementowane [1, 2, 3, 6, 7]. Jedna ze strategii, zwana niezależną (ang. *Independent*), cechuje się możliwością realizacji transakcji w jednym węźle bez porozumienia z innymi. Wiąże się z tym możliwość występowania kolizji, z którą mamy do czynienia w przypadku, gdy dwie różne transakcje modyfikują ten sam zestaw rekordów, w tym samym okresie czasu. Ten rodzaj replikacji dedykowany jest dynamicznie rozwijającym się środowiskom mobilnych baz danych tworzonych na urządzeniach przenośnych [8, 9].

3. Architektura systemu wykorzystującego mobilne bazy danych

Architektura systemu wykorzystującego urządzenia przenośne składa się z jednego serwera centralnego zawierającego stacjonarny serwer bazy danych oraz wielu urządzeń przenośnych, na których zainstalowano mobilne serwery baz danych (rys. 1). Cechą charakterystyczną takiej architektury jest praca w ciągłym rozłączeniu z centralnym bazą danych oraz okresowe łączenie się z serwerem w celu wymiany danych.



Rys. 1. Architektura systemu wykorzystującego urządzenia przenośne [5]
Fig. 1. The architecture of the system using mobile devices [5]

Współpraca elementów takiej architektury w odniesieniu do zastosowań bazodanowych wymaga w pierwszej kolejności **zainicjowania mobilnych baz danych** rekordami z serwera stacjonarnego, a następnie **cyklicznej synchronizacji baz danych** mającej miejsce przy kolejnych ich połączeniach. W niniejszej pracy przedyskutowane zostaną wybrane zagadnienia dotyczące tych procesów.

W systemie SQL Server firmy Microsoft omawianą architekturę można zbudować z wykorzystaniem mechanizmów replikacji scalającej (ang. *Merge replication*) [10] wspieranej zarówno przez SQL Server 2005, jak i SQL Server Mobile. Centralny serwer, określany jako wydawca (ang. *publisher*), udostępnia dane do replikacji. Odpowiednie tabele zwane *artykułami* gromadzone są do powielenia w postaci *publikacji*. Serwer mobilny zwany jest *subskrybentem* (ang. *subscriber*), który zapisuje się na subskrypcję publikacji.

Dla potrzeb badań dotyczących replikacji danych na urządzenia przenośne wykorzystano środowisko rozproszone złożone z:

- komputera PC z systemem operacyjnym Windows XP oraz systemem SQL Server 2005,
- urządzenia przenośnego – Pocket PC wyposażonego w system Windows Mobile 5.0 oraz SQL Server Mobile.

W opisanej architekturze osadzono bazę danych, która w uproszczony sposób reprezentuje środowisko kliniki medycznej. Baza danych składa się z czterech tabel (rys. 2):

1. *Lekarz* – odpowiada za przechowywanie danych o lekarzu, który pełni dwie role –
   - jest lekarzem rodzinnym pacjenta,
   - przeprowadza wizyty w przychodni lekarskiej i wizyty domowe u pacjentów.
2. *Pacjent* – zawiera dane osób zapisanych do kliniki mogących korzystać z wizyt stacjonarnych i domowych.
3. *Wizyta* – przechowuje wybrane dane na temat wizyt zarówno stacjonarnych jak i domowych.
4. *Users* – zawiera zestaw użytkowników systemów przypisanych określonym lekarzom.

4. Inicjalizacja mobilnej bazy danych

Inicjalizacja mobilnej bazy danych odbywa się przez utworzenie tak zwanej migawki początkowej, stanowiącej pełny obraz powielanej tabeli bądź podzbioru jej wierszy, lub kolumn. Dla przykładu, rozważmy inicjalizację tabeli *Wizyta*, w sytuacji kiedy lekarz wybiera się na wizyty domowe. Nie ma potrzeby przenoszenia wszystkich danych o wizytach i pacjentach na jego urządzenie przenośne, lecz tylko o tych, które dotyczą bezpośrednio jego pacjentów. Tabela 1 zawiera przykładowy podział tabeli *Wizyta* na repliki odpowiadające poszczególnym lekarzom.

Tabela 1

Wizyta – filtracja danych względem numeru lekarza (kolumna *nrl*)

nrw	nrp	nrl	data_w	domowa?	Replika
1432	34	1	12-03-2008	tak	Pierwsza
1433	11	2	12-03-2008	nie	Druga
1434	45	1	17-03-2008	nie	Pierwsza
1435	12	3	15-03-2008	tak	Trzecia

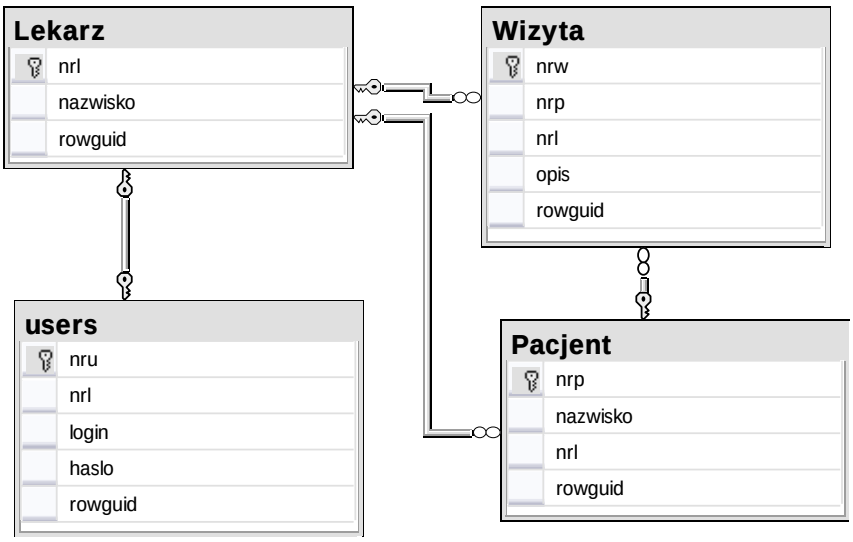
W przypadku systemu SQL Server 2005 podział taki można uzyskać odpowiednio konfigurując *artykuł*. W przykładowej bazie danych, w jego definicji, zawarto filtrację tabeli *Wizyty* względem kolumny *nrl* (numer lekarza). W tym celu zaimplementowano funkcję *dbo.get.nrl(),* która na podstawie danych o zalogowanym lekarzu (tabela *Users*) wybiera jego wizyty.

```
SELECT <published_columns> FROM [dbo].[Wizyta] WHERE [nrl] = dbo.get_nrl()
```

Utworzenie publikacji wiąże się również z rozszerzeniem schematu replikowanego obiektu o kolumnę *rowguid* typu *uniqueidentifier*, z domyślną wartością tworzoną za pomocą wbudowanej funkcji *newsequentialid()*. *Uniqueidentifier* jest to globalny unikalny identyfikator generowany między innymi z wykorzystaniem adresu karty sieciowej komputera oraz

zegara CPU. Kolumna ta używana jest do jednoznacznego identyfikowania rekordu w trakcie synchronizacji repliki ze stacjonarną bazą danych.

Schemat testowej bazy danych po utworzeniu odpowiednich publikacji przedstawiony został na rysunku rys. 2.



Rys. 2. Schemat przykładowej bazy danych po włączeniu jej w środowisko replikacji
Fig. 2. The sample replicated database schema

5. Synchronizacja danych

Ze względu na powtarzający się brak połączenia elementów prezentowanej architektury, wymagana jest okresowa synchronizacja danych pomiędzy serwerem centralnym a serwerami mobilnymi, w celu ujednolicenia zawartości wszystkich baz danych. Podstawowym problem takiego procesu jest możliwość wystąpienia kolizji wynikającej z niezależnie dokonywanych modyfikacji danych. Konflikty te dotyczyć mogą operacji dodawania, usuwania i modyfikacji rekordów, a także zmian schematów publikowanych artykułów. Każde z tych zagadnień omówione zostanie w dalszej części pracy.

5.1. Dodawanie rekordów

Najpoważniejszym problemem podczas wprowadzania danych jest możliwość wystąpienia powtórzonej wartości klucza głównego – pola lub pól, które jednoznacznie charakteryzują rekord. W przykładowej bazie danych sytuacja taka omówiona zostanie na przykładzie dodawania nowej wizyty. Operacja ta może być wykonywana w tym samym okresie czasu zarówno w stacjonarnej, jak w mobilnych bazach danych, co zostało zamodelowane w testowym środowisku z wykorzystaniem następujących poleceń:

```
- dla stacjonarnej bazy danych:
INSERT INTO Wizyta (nrw, nrp, nrl, opis) VALUES (1, 1, 1, 'stacjonarna')

- dla mobilnej bazy danych:

INSERT INTO Wizyta (nrw, nrp, nrl, opis) VALUES (1, 1, 1, 'mobilna')
```

Po wprowadzeniu obu rekordów dokonano synchronizacji baz danych, która zakończyła się błędem wynikającym z naruszenia więzów klucza głównego. Przyczyną niepowodzenia tej operacji jest brak w systemie SQL Server procedur automatyzujących takie zadanie. Sposobów na usunięcie problemu poszukiwano w trzech obszarach.

Oddzielna pula kluczy głównych

Pierwszym z badanych rozwiązań było zastosowanie różnych zbiorów wartości dla kluczy głównych stacjonarnej i mobilnej bazy danych. Czynność tę można konfigurować ręcznie oraz z wykorzystaniem mechanizmu autoinkremenetacji (*Identity*) systemu SQL Server. Wadą takiego rozwiązania jest konieczność oszacowania liczby rekordów wprowadzanych w poszczególnych lokalizacjach oraz kontrola uzyskiwania granicznych wartości zakresu. Zbyt szybkie przydzielanie nowego zakresu może powodować niepotrzebne przerwy w numeracji, zbyt późne może zaowocować konfliktami wprowadzenia danych.

Budowanie złożonych kluczy głównych

W drugim z obszarów podjęto próbę tworzenia kluczy głównych złożonych z kilku kolumn. W przykładowej bazie danych zaproponowano rozszerzenie klucza tabeli *Wizyta* (*nrw*) o numer lekarza (*nrl*). Rozwiązanie to nie dało jednak zadowalających rezultatów, z dwóch powodów:

- nie gwarantowało pewności uniknięcia konfliktu, w przypadku tworzenia wizyty domowej na urządzeniu przenośnym oraz równoczesnej rejestracji nowej wizyty dla tego lekarza w stacjonarnej bazie danych,
- zwiększało złożoność operacji złączenia, w przypadku kiedy w schemacie bazy danych należało uwzględnić dodatkowe tabele powiązane z wizytą, takie jak – badania, recepty, czy skierowania.

Procedura obsługi konfliktu

Ze względu na to, że żadne z powyższych rozwiązań nie było satysfakcjonujące, opracowano nowy algorytm obsługi takiego konfliktu. W tym celu wykorzystano systemowe tabele systemu SQL Server, które rejestrują konflikty powstałe w drodze synchronizacji. Tabele te tworzone są w każdej bazie danych, która udostępnia dane do replikacji typu *Merge*. W przypadku operacji wstawiania rekordów do tabeli *Wizyta* jest to *MSmerge_conflict_LPWU_Pub_Wizyta*, gdzie *LPWU_Pub* jest nazwą publikacji, a *Wizyta* nazwą artykułu. Dla tabeli tej utworzono wyzwalacz (ang. *trigger*), którego zadaniem było przepisanie zarejestrowanych konfliktów do stworzonej, na potrzeby algorytmu, tabeli pomocniczej

Archiwum_conflict_LPWU_Pub_Wizyta. W prezentowanym dalej kodzie tabela *inserted* zawiera rekordy wstawiane do tabeli *MSmerge_conflict_LPWU_Pub_Wizyta.*

```
create trigger Insert_mob on MSmerge_conflict_LPWU_Pub_Wizyta after insert
as
insert into Archiwum_conflict_LPWU_Pub_Wizyta select * from inserted
```

Ponieważ w przypadku architektury wykorzystującej mobilne bazy danych synchronizacja uruchamiana jest po stronie mobilnej bazy danych, obsługa konfliktu musiała zostać zaimplemantowana na urządzeniu przenośnym. Obejmuje ona nastepujący zestaw kroków:

1. połączenie ze stacjonarną bazą danych w celu pobrania maksymalnej wartości klucza z publikowanej tabeli oraz zawartości tabeli pomocniczej, w omawianym przykładzie jest to tabela *Archiwum_conflict_LPWU_Pub_Wizyta*,
2. zmiana wartości klucza głównego w mobilnej bazie danych na podstawie pobranych w pierwszym kroku wartości; wartości klucza powodujące konflikt dostępne są w tabeli *Archiwum_conflict_LPWU_Pub_Wizyta*,
3. powtórna synchronizacja baz danych,
4. usunięcie rekordów z tabeli pomocniczej *Archiwum_conflict_LPWU_Pub_Wizyta,*

Należy podkreślić fakt, że opisana procedura uruchamiana jest tylko w przypadku wykrycia konfliktu powtórzonej wartości klucza.

5.2. Zależności klucza głównego i klucza obcego

Kolejnym aspektem, który należy prześledzić w trakcie synchronizacji baz danych, jest kolejność synchronizacji zmian w rekordach połączonych zależnością klucza obcego i klucza głównego. W przykładowej bazie danych problem taki może dotyczyć tabel *Pacjent* i *Wizyta*. Rozważmy następujący scenariusz. Lekarz wybiera się na wizytę domową, na której dochodzi do zapisania nowego pacjenta i równocześnie jego nowej wizyty. W trakcie synchronizacji baz danych przesłanie rekordów z tabeli *Wizyta* przed rekordami z tabeli *Pacjent* zakończy się konfliktem więzów referencyjnych.

W badaniach prowadzonych w systemie SQL Server nie wykryto problemów z przenoszeniem takich zależności. Jeśli zostanie ona raz zdefiniowana na serwerze stacjonarnym, to obsługą kolejności dokonywania zmian zajmuje się serwer centralny.

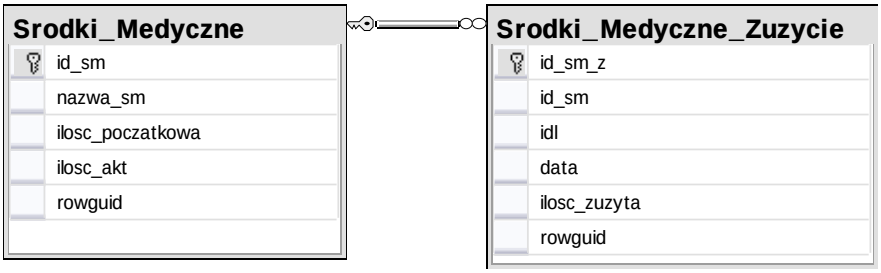
5.3. Aktualizacja danych

Replikacja danych pomiędzy serwem stacjonarnym i mobilnym, w innych celach niż operacje odczytu, wiąże się z powstawianiem konfliktów modyfikacji danych, które można rozważać w dwóch kategoriach. Konflikty operacji **UPDATE – UPDATE** oraz konflikty opera-

cji **UPDATE – DELETE**. Replikacja *Merge* dostarcza bogaty zestaw mechanizmów rozwiązywania kolizji:

- znaczniki czasowe,
- priorytety węzłów,
- wyliczanie sum, średnich, minimalnych i maksymalnych ze zmienianych wartości,
- procedury użytkownika stworzone w języku TSQL lub w innych językach programowania, np. C#, VB.

Dla potrzeb analizy konfliktów wynikających z równoczesnej aktualizacji tych samych rekordów, w bazie danych wydawcy i subskrybenta, rozszerzono schemat testowej bazy danych o nowe tabele *Srodki_Medyczne* i *Srodki_Medyczne_Zuzycie* (rys. 3).



Rys. 3. Tabele rejestrujące zużycie środków medycznych
Fig. 3. Tables of medication consumption registration

Ponieważ w zaproponowanym rozszerzeniu schematu bazy danych występuje redundancja dotycząca aktualnej liczby środków medycznych, tabelę *Srodki_Medyczne_Zuzycie* wyposażono dodatkowo w wyzwalacz zapewniający spójność danych pomiędzy tabelami:

```
create trigger uaktualnij on srodki_medyczne_zuzycie after insert
as
update srodki_medyczne set ilosc_akt = ilosc_akt - ilosc_zuzyta from inserted i
where id_sm = i.id_sm
```

Obie tabele umieszczono w postaci artykułów w jednej publikacji. Podczas pierwszej inicjalizacji subskrypcji na urządzeniu przenośnym natrafiono na ograniczenie systemu SQL Server, który nie przenosi wyzwalaczy związanych z publikowaną tabelą do mobilnej bazy danych. Należy zatem realizację logiki zawartej w wyzwalaczach zapewnić po stronie mobilnej aplikacji klienta.

Konflikty typu UPDATE – UPDATE

Rozważmy teraz sytuację, w której lekarz kliniki na swoim przenośnym urządzeniu planuje przyszłe zabiegi, rezerwując środki medyczne do ich realizacji (dla przykładu środek medyczny o identyfikatorze 1). Z punktu widzenia baz danych jest to zbiór operacji:

```
Insert into Środki_Medyczne_Zuzycie (id_sm_z, id_sm, idl, data, ilosc_zuzyta)
values (45, 1, 3, getdate(), 5)

Update Srodki_Medyczne set ilosc_akt = ilosc_akt – 5 where id_sm = 1
```

przy czym druga z nich może być wykonywana przez wyzwalacz.

Jeżeli zasymulujemy podobną operację wykonywaną przez innego lekarza w stacjonarnej bazie danych, to w momencie synchronizacji obu baz danych w tabeli *Srodki_Medyczne* zostanie wykryty konflikt.

Jak się okazało podczas badań, możliwości jego rozwiązania dla subskrypcji w mobilnych bazach danych są ograniczone. Z wbudowanych algorytmów rozwiązywania kolizji dla replikacji na urządzenia przenośne nie są dostępne rozwiązania: addytywne i wyliczające średnią. Wynika to z faktu, że są one dedykowane publikacjom, dla których śledzenie konfliktów odbywa się na poziomie kolumn. W przypadku subskrypcji dla mobilnych baz danych przewidziana jest jedynie możliwość śledzenia zmian na poziomie rekordu. Dlatego próba rozwiązania konfliktu z wykorzystaniem jednego z tych algorytmów zawsze pociąga za sobą rozwiązanie priorytetowe, co w przypadku mobilnych baz danych oznacza zapisanie zmian wprowadzonych u wydawcy.

Kolejnym ograniczeniem nakładanym na subskrypcje tworzone w mobilnych baza danych jest niemożność rozwiązywania konfliktów z wykorzystaniem procedur wbudowanych. W ramach badań podjęto próbę zniwelowania tego ograniczenia przez zastosowanie w odniesieniu do mobilnej bazy danych mechanizmu serwera przyłączonego, ułatwiającego dostęp do odległych serwerów:

```
EXEC master.dbo.sp_addlinkedserver @server = N'LPWU',
@srvproduct=N'SQLOLEDB',
@provider=N'Mcrosoft.SQLSERVER.MOBILE.OLEDB.3.0',
@datasrc=N'LPWU_Mobile.SDF',
```

jednak nie przyniosło to oczekiwanych rezultatów. Dlatego w przypadku bardziej złożonych algorytmów rozwiązywania kolizji, w których biorą udział mobilne bazy danych, jedynym rozwiązaniem jest stworzenie dynamicznej biblioteki w dowolnym z języków programowania dostępnych na platformie .NET i zarejestrowanie jej na serwerze wydawcy.

Konflikty typu UPDATE – DELETE

Badając konflikty wynikające z równoczesnego usuwania i aktualizacji danych, nie natrafiono na problemy synchronizacji takich zmian. Decyzję, jaką można podejmować w takim przypadku, jest wybór serwera, którego zmiany będą traktowane priorytetowo.

W sytuacji kiedy dochodzi do konfliktu UPDATE – DELETE, wart rozpatrzenia jest problem bezpowrotnej utraty danych z usuwanych w drodze synchronizacji rekordów. Ponieważ mechanizmy systemu SQL Server rejestrują taki konflikt w tabeli systemowej *MSmerge_conflict_<nazwa_pubublikacji_nazwa_artyklulu>*, można ją wykorzystać do uzyskania kontroli nad usuwanymi danymi. W tym celu należy zastosować w odniesieniu do tej tabeli część algorytmu zaproponowanego w kolejnym rozdziale (5.4): **Zadania stojące po stronie stacjonarnego serwera bazy danych,** Krok 1 i Krok 2.

5.4. Zmian schematu replikowanych tabel

Analizując zagadnienia synchronizacji replikowanych obiektów, należy także zbadać problem modyfikacji ich schematów. Ten rodzaj zmian możliwy jest jedynie po stronie serwera stacjonarnego. Każda zmiana struktury publikowanego artykułu umieszczana jest w tabeli systemowej *sysmergeschemachange*, skąd odpowiednie mechanizmy systemu SQL Server przenoszą ją do serwera mobilnego.

Wśród przeprowadzonych testów dotyczących tego zakresu zmian znalazły się operacje dodawania i usuwania kolumny *Specjalizacja* do i z tabeli *Lekarz*. Z punktu widzenia poprawności przeprowadzania modyfikacji można powiedzieć, że w obu przypadkach operacje przebiegały bez zakłóceń. Budzi wątpliwość jednak fakt, że w przypadku usuwania kolumny z tabeli zmiana ta przenoszona jest do mobilnej bazy danych nawet w przypadku, kiedy znajdują się w niej dane.

Z tej też przyczyny opracowano i zaimplementowano mechanizmy, które pozwalają pobrać dane z usuwanej w mobilnej bazie danych kolumny, by można było świadomie podjąć decyzję o ich dalszym losie. Niezbędne kroki postępowania prezentuje przedstawiony dalej algorytm.

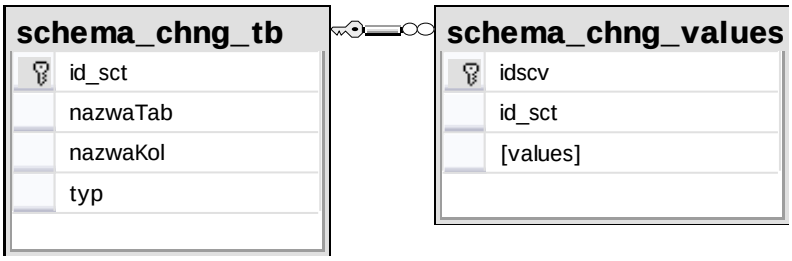
Zadania stojące po stronie stacjonarnego serwera bazy danych wykonywane jednorazowo w trakcie implementacji algorytmu.

Krok 1. Rozszerzenie bazy danych o pomocniczą tabelę, w której rejestrowane są zmiany dotyczące schematu bazy danych. W omawianym środowisku tabeli tej nadano nazwę *SchemaChanges.* Ma ona taką samą strukturę jak tabela *sysmergeschemachange*, poszerzoną o kolumnę reprezentującą nazwę modyfikowanej tabeli.

Krok 2. Utworzenie dla tabeli *sysmergeschemachange* wyzwalacza, którego zdaniem jest przenoszenie rejestrowanych w niej zmian schematu bazy danych do tabeli pomocniczej *SchemaChanges*

```
create trigger schema_chanage on sysmergeschemachange after insert
as
insert into SchemaChanges select i.*, ma.name as nazwaTab from inserted i,
sysmergearticles ma nwhere i.artid=ma.artid
```

Krok 3. Rozszerzenie baz danych o dwie tabele, które mają przechowywać dane z usuwanych kolumn (rys. 4). Tabela *schema_chng_tb* zawiera dane dotyczące tabeli, z której ma zostać usunięta kolumna oraz nazwę i typ tej kolumny. Natomiast do tabeli *schema_chng_values* wprowadzane są wartości usuwane z kolumny.



Rys. 4 Tabele rejestrujące dane dotyczące usuwanych kolumn
Fig. 4. Tables of system changes registration

Zadania stojące po stronie aplikacji zarządzającej mobilną bazą danych, które powinny zostać wykonane każdorazowo przed rozpoczęciem synchronizacji baz danych, a po nawiązaniu połączenia ze stacjonarnym serwerem.

Krok 1. Sprawdzenie, czy w tabeli *SchemaChanges* znajdują się rekordy świadczące o zmianach w schemacie publikowanych artykułów, co realizowane jest poleceniem języka SQL *select * from SchemaChanges* uruchamianym z poziomu obiektów aplikacji.

Krok 2. Ustalenie rodzaju operacji ADD (dodanie) lub DROP COLUMN (usunięcie kolumn). W przypadku drugiej operacji wypełnienie tabel *schema_chng_tb* oraz *schema_chng_values* odpowiednimi danymi. Wprowadzenie wartości do tabeli *schema_chng_values* wymaga połączenia z mobilną bazą danych; nazwy tabeli i kolumny, których dotyczy operacja, dostępne są w tabeli *SchemaChanges*.

Krok 3. Uruchomienie synchronizacji, w trakcie której dochodzi do usunięcia odpowiedniej kolumny z mobilnej bazy danych. Zawartość kolumny już znajduje się w tabeli *schema_chng_values*.

Krok 4. Usunięcie rekordów z tabeli *SchemaChanges*.

Rozpatrując zagadnienia dotyczące zmiany schematu publikowanego artykułu, należy wspomnieć jeszcze o zmianie typu określonej kolumny oraz usuwaniu tabeli subskrybowanej w innych bazach danych. Nie jest to jednak kwestia ważna z punktu widzenia synchronizacji danych, ponieważ serwer nie dopuszcza do wykonania takiej operacji do czasu, kiedy tabela ta nie zostanie usunięta z publikacji.

6. Podsumowanie

W pracy przeanalizowano możliwości replikacji danych na urządzenia mobilne. Omówione zostały różne aspekty tego procesu wynikające z cech architektury wykorzystującej tego typu urządzenia, charakteryzującej się pracą w permanentnym rozłączeniu jej elementów. Szczególną uwagę zwrócono na inicjalizację mobilnej bazy danych oraz jej cykliczną synchronizację. Jednym z wyników przeprowadzonych badań jest ocena istniejących mechanizmów zarządzania procesem replikacji na urządzenia mobilne. Analizowane rozwiązania

w większości przypadków realizują powierzone im funkcje, choć dostrzeżono pewne ograniczenia nakładane na subskrypcje w mobilnych bazach danych. Drugim z efektów badań jest opracowanie algorytmów uzupełniających funkcjonalność istniejących mechanizmów synchronizacji.

Zaproponowane rozwiązania testowane były w środowisku stworzonym za pomocą narzędzi firmy Microsoft: SQL Server 2005 znajdujący się na serwerze centralnym oraz SQL Server Mobile pracujący na urządzeniu przenośnym. W takim środowisku zbudowano przykładową bazę danych kliniki medycznej oraz aplikację zaimplementowaną na urządzeniu przenośnym [4]. Uniwersalność opracowanych algorytmów pozwala jednak stosować je w dowolnej dziedzinie przedmiotowej.

BIBLIOGRAFIA

1. Acosta–Elias J., Navarro–Moldes L.: A Demand based Algorithm for Rapid Updating of Replicas. 22nd International Conference on Distributed Computing Systems. Austria, 2002.
2. Bersnstein P.A., Hadzilacos V., Goodman N.: Concurrency Control and Recovery in Database Systems. Addison–Wesley 1987.
3. Ceri S., Houtsma M.A.W., Keller A.M., Samarati P.: A Classification of Update Methods for Replicated Databases. Internal Report CS Dept. Stanford University, STAN-CS-91-1392, October 1991, s. 1÷17.
4. Creating a Mobile Application with SQL Server Compact Edition, [@:] http://technet.microsoft.com/en-us/library/ms171908.aspx, 2009.
5. Exchanging Data with Mobile Users. [@:] http://msdn.microsoft.com/en-us/library/ms151323.aspx, 2009
6. Harężlak K.: Asynchroniczna metoda aktualizacji kopii danych. Współczesne problemy sieci komputerowych. Nowe technologie. WNT, 2004.
7. Malkhi D., Reiter M..K., Rodeh O., Sella Y.: Efficient Update Diffusion in Byzantine Environments. Proceeding of the 20th Symposium Reliable Distributed Systems, 2001.
8. Preguica N., Sharpio M., Martins J.L: Automating semantics-based reconciliation for mobile databases. Proceedings of 3ème Conférence Française sur les Systèmes 'Exploitation, 2003.
9. Tancred L., XML Threeway Merge as a Reconciliation Engine for Mobile Data. Proceedings of the 3rd ACM international workshop on Data engineering for wireless and mobile access, San Diego, CA, USA, 2003

10. Using Merge Replication, SQL Server CE Books Online, [@:] http://msdn2.microsoft.com/en-us/library/ms172367.aspx, 2008.

Recenzent: Prof. dr hab. inż. Mieczysław Muraszkiewicz

Wpłynęło do Redakcji 4 lutego 2009 r.

Abstract

The possibility of mobile devices usage for data replication was analyzed in the paper. Problems of replica initialization and periodical synchronization were considered particularly. The research was performed in architecture consisting of Microsoft tools: SQL Server 2005 and SQL Server Mobile. Its effects cover the assessment of existing methods of mobile database management and work out new mechanisms of replicated data synchronization. The proposed solutions connected with constraints of Primary Key violations, update conflicts and database schema changes were tested in sample database dedicated for medical clinic. The schema of this database is presented on figures Fig. 1., Fig. 2, Fig. 3. and Fig. 4.

The new environment was prepared for research. This environment consists of stationary database server and mobile device application using data stored on mobile database server. Although proposed solutions were tested in sample environment, their universality made them capable of being used in other areas of life.

Adres

Katarzyna HARĘŻLAK: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-100 Gliwice, Polska, katarzyna.harezlak@polsl.pl.