OPTYMALIZACJA PRACY SQL SERVER PRZY WSPÓŁPRACY Z MICROSOFT DYNAMICS NAV - ZALECENIA PROGRAMISTYCZNE

Marcin WOCH, Piotr ŁEBKOWSKI

Streszczenie: Artykuł prezentuje wybrane metody optymalizacji pracy serwera baz danych SQL Server oraz 2005 przy współpracy z systemem klasy ERP Microsoft Dynamics NAV. Zwraca on szczególną uwagę na odpowiednie zaprojektowanie bazy danych oraz poprawny kod źródłowy aplikacji.

Słowa kluczowe: Microsoft Dynamics NAV, ERP, SQL Server 2005, optymalizacja, C/SIDE, C/AL.

1. Wprowadzenie

Microsoft Dynamics NAV – zwany poprzednio Navision Attain, Navision Financials jest systemem klasy ERP (Enterprise Resource Planning), który obsługuje praktycznie wszystkie możliwe obszary działalności firmy: księgowość, sprzedaż, należności, zakupy, płatności, produkcja, zarządzanie magazynem, itp.

System NAV potrafi współpracować z dwoma różnymi typami serwerów: Microsoft Navision Database Server oraz Microsoft SQL Server. Dla użytkownika w obu przypadkach system wygląda i pracuje dokładnie tak samo. Niemniej jednak istnieją pewne różnice w sferze programistycznej oraz administracyjnej. Różnice te wyrażają się m.in. w:

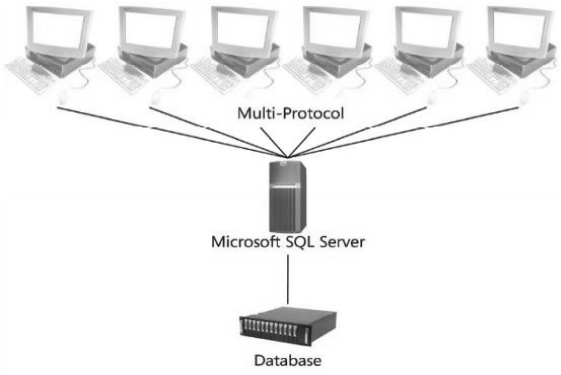
- skalowalność,
- braku limitu rozmiaru bazy w SQL Server,
- zarządzaniu RAM,
- monitorowaniu pracy serwera,
- sposobie działania SIFT (Sum Index Field Technology), itp.

W niniejszym artykule opisana jest opcja SQL Server dla Microsoft Dynamics NAV.

2. Architektura

Microsoft Dynamics NAV, który jest zintegrowany ze środowiskiem programistycznym C/SIDE (Client/Server Integrated Development Engine) pozwala na współpracę z Microsoft SQL Server. Nowsze wersje NAV współpracują zarówno z SQL 2000 jak i z SQL 2005. Niemniej jednak duże lepsze parametry pracy osiąga się przy instalacji SQL Server 2005.

Niniejszy artykuł skupia się na dwuwarstwowej architekturze systemu (rys. 1), gdzie klient łączy się bezpośrednio z serwerem bazodanowym.

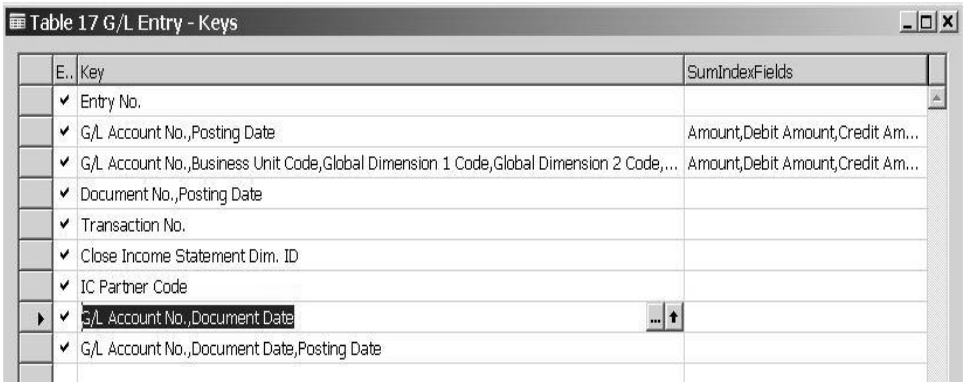


Rys. 1. Architektura opcji MS SQL Server dla MS Dynamics NAV [3]

2. Zalecenia programistyczne

Planowanie szczegółów aplikacji oraz bazy danych jest bardzo ważnym elementem projektu. Prawidłowo zaprojektowana aplikacja jest także łatwiejsza do zaimplementowania. Niniejszy rozdział nie skupia się na dobrze znanych i omawianych w literaturze metodologiach analizy, projektowania oraz implementacji, ale omawia prawidłowe zarządzanie kodem aplikacji w języku C/AL (C/SIDE Application Language), taka by uzyskać jak najlepsze efekty wydajnościowe.

Jednym z podstawowych elementów jest prawidłowa definicja kluczy oraz indeksów zdefiniowanych dla tabel. Ten element znacząco wplywa na zachowanie systemu. Należy pamiętać, że im większa liczba kluczy tym wolniej działają operacje zapisu, modyfikacji oraz usuwania danych. System musi uaktualnić dany rekord oraz wszystkie indeksy z tym związane. Wszystkie nieużywane klucze powinny zostać usunięte.



Rys. 2. Przykład zduplikowanych kluczy

W przykładzie z rysunku 2, klucz wyróżniony szarym tłem jest niepotrzebny i powinien zostać usunięty. Klucz składający się z pól “G/L Account No.” oraz “Document Date” zawiera się w “G/L Account No.”, “Document Date”, “Posting Date”. Po usunięciu tego

klucza, poniższe polecenie SETCURRENTKEY także zadziała poprawnie:

SETCURRENTKEY(“G/L Account No.”, “Document Date”);

Równie istotną sprawą jest zastosowanie prawidłowego klucza w zapytaniach używających filtrów. W przykładnie poniżej polecenie SETCURRENTKEY znacząco przyspiesza działanie zapytania. Filtry (tu SETRANGE) w tym przypadku zostaną zastosowane z użyciem indeksu zgodnego z wybranym kluczem. Zapytanie C/AL:

```
SETCURRENTKEY(Field1, Field2);
SETRANGE(Field1, Code1);
SETRANGE(Field2, Code2);
IF FIND(‘-‘) THEN
  ….
```

może zostać przetworzone na zapytanie T/SQL:

```
SELECT * FROM Table
WHERE Field1 = Code1 AND Field2 = Code2
ORDER BY FIELD1, FIELD2
```

Natomiast zapytanie bez użytego polecenia SETCURRENTKEY przyjmie postać:

```
SELECT * FROM Table
WHERE Field1 = Code1 AND Field2 = Code2
```

Klauzula ORDER BY wymusza użycie indeksu przez optymalizator zapytań.

Dobrą praktyką jest usuwanie nieużywanych pól z kluczy. Jeśli na przykład firma nie używa wariantów zapasów należy usunąć wystąpienia pola “Variant Code” z kluczy.

Każdy klucz w NAV posiada kilka parametrów, które bezpośrednio wpływają na zakładane blokady oraz wydajność zapytania. Oto typowe przykłady:

- MaintainSQLIndex – przy ustawieniu tej właściwości na FALSE SQL Server nei tworzy indeksu dla takiego klucza. Ustawienie zalecane jest dla klucza stosowanego do niewielu i rzadko używanych raportów. Opcja ustawiona na wartość TRUE wymusza stworzenie indeksu.
- SQLIndex – w przpadku, gdy wartość MaintainSQLIndex jest równa TRUE, SQL zawsze tworzy indeks dla klucza. Tak stworzony indeks niekoniecznie posiada taką samą kolejność pól jak klucz. Ustawienie parametru SQL Index wymusza identyczną kolejność pól w indeksie. Opcja taka dostępna jest w wersji 4.0 SP1 oraz wyższych.
- MaintainSIFTIndex – Sum Index Field Technology (SIFT) jest algorytmem opracowanym przez Navision służącym do obliczania sum pól wirtualnych. SQL Wówczas, gdy MaintainSIFTIndex przyjmuje wartość TRUE server przechowuje struktury SIFT w specjalnych tablicach. Gdy jest równy FALSE, wtedy sumy wirtualne są liczone na bieżąco.
- SIFTLevelsToMaintain – dzięki niemu programista może zdefiniować, które poziomy SIFT są przechowywane w tabeli SIFT a kiedy nie.
- Clustered – opcja pozwalająca zdefiniować, czy indeks jest klastrowany czy nie. Opcja ta jest dostępna w systemach 4.0 SP1 oraz wyższych.

Kolejną techniką pozwalającą na zarządzanie kluczami i indeksami to tzw. “grupy kluczy” (key groups). Metoda ta pozwala na aktywację oraz deaktywację kluczy bez potrzeby zmiany struktury tabeli bazy danych. Podstawową zaletą grup kluczy jest możliwość deaktywacji oraz aktywacji klucza bez licencji programistycznej systemu NAV.

W celu uniknięcia zakleszczeń (deadlock) należy kontrolować kolejności blokowań tablic. Zachowanie poprawnej kolejności blokowań nie zabezpieczy całkowicie systemu przed zakleszczeniami, ale pozwala na znaczne zredukowanietej liczbę. Zakleszczenia są wynikiem sytuacji, gdy dwa procesy blokują się nawzajem oczekując na odblokowanie tabeli lub rekordów.

Dla przykładu pierwszy proces blokuje najpierw tablicę *Table1* następnie *Table2*.

```
Table1.LOCKTABLE;
Table2.LOCKTABLE;
```

Drugi proces blokuje te same tablice, ale w odwrotnej kolejności:

```
Table2.LOCKTABLE;
Table1.LOCKTABLE;
```

W przypadku, gdy oba procesy rozpoczynają sie w tym samy czasie, proces nr 1 czeka na odblokowanie tablicy *Table2*, a proces nr 2 na tablicę *Table1*. Taka sytuacja nazywa się zakleszczeniem.

Innym sposobem uniknięcia zakleszczeń jest zastosowanie polecenia LOCKTABLE. Dzięki temu poleceniu przed rozpoczęciem jakiegokolwiek przetwarzania system zablokuje całą tablicę na potrzeby danego procesu.

Użycie tablic tymczasowych także pozwala uniknąć zakleszczeń. Ich podstawową zaletą jest fakt ich przetwarzania po stronie klienta. Tabele tymczasowe pozwalają także na modyfikację danych w triggerach.

W większości przypadków system NAV przesyła dane z serwera do klienta, tam one są przetwarzane i zmienione wracają do serwera w celu aktualizacji bazy. Polecenia C/AL takie jak: INSERT, MODIFY, DELETE są przetwarzane po stronie klienta. Kod:

```
Table.SETRANGE(FieldX, ValueX);
IF Table.FIND('-') THEN
  REPEAT
    Table.FieldY := ValueY;
    Table.MODIFY;
  UNTIL Table.NEXT = 0;
```

może być zamieniony na:

```
Table.SETRANGE(FieldX, ValueX);
Table.MODIFYALL(FieldY, ValueY);
```

Dzięki temu polecenia MODIFYALL oraz DELETEALL są przetwarzane po stronie serwera. Pomaga to na uniknięcie przesyłu danych siecią do stacji klienckiej i z powrotem. Ponadto pominiętą zostaje pętla REPEAT … UNTIL, która w powiązaniu z FIND('-') zwiększa ryzyko zakleszczeń.

W NAV istnieje kilka poleceń służących do przeszukiwania tablic. Większośc z tych poleceń tworzy kursor po stronie SQL. Taką komendą jest FIND, która przeszukuje tablicę z uwzględnieniem bieżących fitlrów (SETRANGE/SETFILTER) oraz użytego klucza (SETCURRENTKEY). Komenda FIND zwraca wartość TRUE, kiedy record został znaleziony, a FALSE w przeciwnym wypadku. FIND('-') znajduje pierwszy record w zestawie, FIND('+') znajduje ostatni record wg kryteriów poszukiwania. Wadą polecenia FIND jest wspomniane tworzenie kursora oraz przeszukiwanie całej tablicy (Full Scan) SELECT * FROM nawet do znaleznienia jednego rekordu.

W wersji NAV 4.0 SP1 wprowadzono kilka nowych poleceń, które pomagają uniknąć wad komendy FIND. Są to: FINDLAST, FINDFIRST oraz FINDSET, nie skanują one

całej tabeli tylko wybrane rekordy. FINDFIRST oraz FINDLAST są przetwarzane do zapytania SELECT TOP 1. Nie tworzą one już kursorów tymczasowych, za wyjątkiem FINDSET użytego w powiązaniu z transakcją. Kursor nie jest tworzony w przypadku, gdy FINDSET jest użyty tylko do czytania rekordów.

Kolejną bardzo użyteczną komendą jest ISEMPTY, która sprawdza czy tabela wg danego kryterium jest pusta czy nie. Komenda ISEMPTY także nie tworzy kursora i przetwarzana jest do SELECT TOP 1.

3. Wnioski

Obecne wersje Microsoft Dynamics NAV oprócz serwera natywnego pozwalają na współpracę z MS SQL Server 2000 oraz MS SQL Server 2005. Prawidłowa administracja serwerem oraz systemem Navision pozwala na optymalizację zapytań oraz manipulacji danymi.

Płaszczyzny administracji można podzielić na 3 grupy:

- zarządzanie sprzętem,
- administracja SQL Server,
- administracja systemem NAV.

Sama decyzja zakupu sprzętu powinna uwzględniać planowane wdrożenie, w szczególności obszar systemu ERP, który będzie używany, liczbę wszystkich użytkowników systemu oraz przewidywaną liczbę równoległych, konkurencyjnych sesji. Należy także brać pod uwagę możliwy przyrost bazy danych [9].

Rutynowe prace administratora serwera oraz systemu Navision obejmują między innymi śledzenie logów, pilnowanie blokad, zmianę parametrów wraz ze wzrostem bazy danych. Istnieje szereg parametrów oraz narzędzi, które pozwalają monitorować pracę systemu [9].

Bardzo ważnym elementem wpływającym na wydajność systemu jest odpowiednio zaprojektowana i napisana baza danych. Prawidłowe nawyki programistyczne takie jak: używanie tabel tymczasowych, zachowanie odpowiedniej kolejności dostępu do tabel, pozwalają przyspieszyć pracę systemu oraz uniknąć zbędnych blokad i zakleszczeń. Istotnym elementem jest odpowiednie zaprojektowanie kluczy, indeksów, przy czym pamiętać należy, że zbyt ich duża liczba spowalnia usuwanie, dodawanie i modyfikowanie rekordów. Konieczne jest także użycie w zapytaniach odpowiednich filtrów i kluczy sortujących.

Literatura

1. Bieniek D., Dyess R., Hotek M., Loria J., Machanic A., Soto A., Wiernik A.: Microsoft SQL Server 2005 Implementation and Maintenance, Self Paced Training Kit. MSPress, Redmont, 2006.
2. Microsoft Business Solutions Navision 4.0 Course: 8404B Installation and Configuration Training. Microsoft Corporation, 2004.
3. Muhlbaher H.: Optimizing Dynamics NAV on SQL Server – Application. 2007.
4. Nielsen P.: SQL Server 2005 Bible. Wiley Publishing, Inc., Indianapolis, 2007.
5. Rankins R., Jensen P., Bertucci P.: Microsoft SQL Server 2000. Księga eksperta. Helion, Gliwice, 2003.
6. Raheem M., Sonkin D., D'Hers T., LeMonds K.: Inside SQL Server 2005 Tools. Addison Wesley Professional, Boston, 2006.

7. Thomas O., McLean I.: Optimizing and Maintaining a Database Administration Solution by Using Microsoft SQL Server 2005. Microsoft Press, Redmont, 2006.
8. Woch M.: Microsoft SQL Server Optimization for Microsoft Dynamics NAV. Studia Informatica vol. 28, number 2(71), s. 17-29, Gliwice, 2007.
9. Woody B.: Administrator's Guide to SQL Server 2005. Addison Wesley Professional, Boston, 2006.

Mgr inż. Marcin WOCH
Dr hab. inż. Piotr ŁEBKOWSKI, prof. AGH
Wydział Zarządzania, Akademia Górniczo-Hutnicza
30-067 Kraków, ul. Gramatyka 10
tel./fax.: (0-12) 617 42 80
e-mail: marcinwoch@wp.pl
plebkows@zarz.agh.edu.pl