informatyka+

Algorytmika i programowanie

Bazy danych

Multimedia, grafika i technologie internetowe

Sieci komputerowe

Tendencje w rozwoju informatyki i jej zastosowań

Człowiek – najlepsza inwestycja

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Warszawska Wyższa Szkoła Informatyki

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

informatyka+

Kuźnia Talentów:

Optymalizacja zapytań w SQL

Andrzej Ptasznik

Człowiek – najlepsza inwestycja

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Warszawska Wyższa Szkoła Informatyki

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Optymalizacja zapytań SQL

Rodzaj zajęć: Kuźnia Talentów
Tytuł: Optymalizacja zapytań SQL
Autor: mgr inż. Andrzej Ptasznik

Zeszyt dydaktyczny opracowany w ramach projektu edukacyjnego **Informatyka+** — ponadregionalny program rozwijania kompetencji uczniów szkół ponadgimnazjalnych w zakresie technologii informacyjno-komunikacyjnych (ICT).
www.informatykaplus.edu.pl
kontakt@informatykaplus.edu.pl

Wydawca: Warszawska Wyższa Szkoła Informatyki
ul. Lewartowskiego 17, 00-169 Warszawa
www.wwsi.edu.pl
rektorat@wwsi.edu.pl

Projekt graficzny: FRYCZ I WICHA

Warszawa 2012

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Warszawska Wyższa Szkoła Informatyki

UNIA EUROPEJSKA
EUROPEJSKI FUNDUSZ SPOŁECZNY

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Optymalizacja zapytań SQL

Andrzej Ptasznik
Warszawska Wyższa Szkoła Informatyki
aptaszni@wwsi.edu.pl

Streszczenie

Kurs zapoznaje słuchaczy z problematyką wydajności i optymalizacji zapytań SQL. Omówiona zostanie fizyczna organizacja przechowywania danych i wprowadzone zostaną pojęcia indeksów zgrupowanych i niezgrupowanych. Zaprezentowane zostaną przykłady planów wykonania zapytań generowane przez optymalizator SQL. Na bazie przykładu omówione będą także problemy wyboru strategii wykonania zapytania w zależności od zawartości tabel i zdefiniowanych indeksów. Wprowadzone zostanie pojęcie statystyk indeksów i omówione będzie ich znaczenie przy wyborze strategii realizacji zapytania. Słuchacze zostaną zapoznani z problemami mechanizmu transakcyjnego i pojęciem poziomów izolacji transakcji. Kurs poświęcony problemom optymalizacji zapytań dotyka trudnych i złożonych problemów, do rozwiązywania których nie ma jednolitego sposobu działania. Rozwiązywanie problemów optymalizacyjnych wymaga dużej wiedzy i zrozumienia zjawisk i procesów działających w środowisku SQL Server.

Spis treści

1 WPROWADZENIE

1.1. WSTĘP

Problemy związane z wydajnością pojawiają się we wszystkich systemach informatycznych i nie dotyczą jedynie problemów baz danych. Jeśli problemy pojawiają się przed wdrożeniem systemu, to nie jest jeszcze tak źle. Można wtedy podjąć decyzje wiążące się z dokonywaniem zmian w projekcie, zmian w strukturze bazy danych i nie będą one się wiązać z koniecznością dbania o już istniejące dane. Gorszym wariantem jest rozwiązywanie problemów wydajnościowych, gdy system jest wdrożony do eksploatacji. Nie dość, że możliwości modyfikacji są ograniczone, to jeszcze trzeba starać się nie zakłócać normalnej pracy użytkowników. Gdy dodamy do tego presję czasu i stres – pojawia się obraz pracy nie do pozazdroszczenia. W każdym jednak przypadku istotne jest, żeby wiedzieć, jakie kroki podjąć, co sprawdzić, na co zwrócić szczególną uwagę, jakich narzędzi użyć i w jaki sposób, aby osiągnąć cel – wzrost wydajności bazy danych do akceptowalnego poziomu. Może nam się wydawać, że takie problemy dotyczą tylko dużych projektów i baz danych, więc nie ma się co martwić na zapas. Bardzo szybko jednak można natrafić na podobne problemy nawet w prostych aplikacjach.

W ramach niniejszego kursu postaramy się przedstawić podstawy wiedzy potrzebnej do poruszania się w dziedzinie zagadnień związanych z wydajnością baz danych, a dokładniej – zapytań na nich wykonywanych. Zanim zaczniemy jednak wkraczać do problematyki optymalizacji zapytań SQL, musimy poznać i zrozumieć fizyczną organizację przechowywania danych, istotę indeksów oraz mechanizm transakcyjny. Bez podstawowej wiedzy z tych zagadnień trudno zrozumieć istotę problemów wydajnościowych, a tym samym podejmować kroki poprawiające wydajność. Niejako przy okazji zaprezentowany zostanie też ogólny model optymalizacji wydajności, stosowany w praktyce przy realizacji zadań związanych z zapewnieniem wymaganego poziomu wydajności bazy danych.

1.2. PODSTAWOWE PROBLEMY OPTYMALIZACJI ZAPYTAŃ.

W świecie systemów informatycznych i komputerów od wielu lat utrzymuje się stały trend wzrostu mocy obliczeniowej, pojemności pamięci operacyjnej, pojemności i szybkości dysków twardych itp. W związku z tym, jeśli mamy do czynienia ze zbyt niską wydajnością bazy danych, to pierwszym pomysłem może być rozbudowa systemu od strony sprzętowej. Niestety nie zawsze to działa, lub przewidywane koszty rozbudowy są zdecydowanie nieakceptowalne. Osiągnięty efekt może także nie być długotrwały i po kolejnym okresie uzupełniania danych w bazie wracamy do punktu wyjścia, czyli system w dalszym ciągu nie spełnia oczekiwań pod kątem szybkości realizacji zapytań i operacji modyfikacji danych. W takiej sytuacji warto zrobić to, od czego tak naprawdę należało zacząć – przeanalizować bazę danych pod kątem możliwości optymalizacji jej wydajności. Okazuje się, że tą drogą można osiągnąć bardzo dobre rezultaty. Niestety wymaga to znacznej wiedzy i umiejętności. Istnieją sprawdzone w praktyce podejścia (modele) optymalizacji wydajności baz danych, lecz ich rola polega raczej na wyznaczeniu ogólnych ram i sekwencji czynności, których wykonanie należy wziąć pod uwagę przy prowadzeniu optymalizacji, niż na dostarczeniu gotowej recepty. Proces optymalizacji wydajności według przyjętego przez nas modelu składa się z kilku obszarów:

- Struktura (projekt) bazy danych
- Optymalizacja zapytań
- Indeksy
- Blokady
- Tuning serwera

Całość modelu jest przedstawiona na rysunku 1.

Kolejność realizacji zadań powinna przebiegać od dołu diagramu do góry. Podobnie, liczba możliwych do osiągnięcia usprawnień jest tym większa, im niżej znajdujemy się na diagramie. Sekwencja ta nie jest przypadkowa i wzajemne zależności pomiędzy blokami powodują, że założona kolejność realizacji umożliwia uzyskanie najlepszych efektów najmniejszym kosztem. W ramach kursu skupimy się na trzech blokach – optymalizacji zapytań, indeksach oraz blokadach. W ramach kolejnych faz procesu optymalizacji rozwiązujemy problemy różnego typy, ponieważ problemy wpływające na wydajność ulokowane są w różnych miejscach systemu i tylko kompleksowe podejście do ich rozwiązania gwarantuje zadawalające efekty. Zgodnie z ogólnym modelem optymalizacji, przedstawionym na rysunku 1, na poszczególnych poziomach modelu, należy zwracać uwagę na:

Rysunek 1.
Model procesu optymalizacji baz danych

- Projekt struktury bazy danych
  - Normalizacja bazy danych (doprowadzenie schematu bazy do 3 postaci normalnej)
  - Definicje widoków
  - Procedury składowane
  - Funkcje użytkownika

- Optymalizacja zapytań
  - Organizacja przetwarzania w oparciu o zbiory
  - Eliminowanie przetwarzania iteracyjnego
  - Rezygnacja z wykorzystania niewydajnych struktur danych (tabele tymczasowe, kursory)

- Indeksy
  - Definiowanie niezbędnych indeksów
  - Defragmentacja indeksów
  - Definiowanie indeksów pokrywających
  - Definiowanie widoków zmaterializowanych

- Blokady
  - Stosowanie możliwie najniższego poziomu izolacji transakcji
  - Partycjonowanie tabel i indeksów

- Sprzęt
  - Zwiększanie pamięci operacyjnej
  - Zmiana dysków twardych na wydajniejsze
  - Zwiększanie mocy procesora

Optymalizacja baz danych jest problemem bardzo złożonym i wymaga działań w różnych warstwach modelu optymalizacji.

1.3. FIZYCZNA ORGANIZACJA PRZECHOWYWANIA DANYCH W SQL SERVER.

Problemy optymalizacji zapytań są trudne do zrozumienia, jeżeli nie znamy sposobu fizycznej organizacji przechowywania danych. Jednym z istotnych zagadnień jest tu sposób, w jaki dane są fizycznie przechowywane w bazie danych. Gdy myślimy o tabeli, to od razu przedstawiamy sobie coś na kształt zbioru wierszy składających się z kolumn zawierających dane różnego typu (patrz rysunek 2).

iducznia	Nazwisko	Imie	DataUrodzenia	CzyChlopak	Pesel	idklasy	plec
1	Kotek	Katarzyna	1992-03-12	0	92031275446	2	Kobieta
2	Piesek	Jan	1992-05-15	1	92051587746	1	Mężczyzna
3	Lisek	Kasia	1992-02-22	0	92022277654	2	Kobieta
4	Kurka	Jola	1992-06-02	0	92060288788	2	Kobieta
5	Gąska	Wacek	1991-03-11	1	91031199123	1	Mężczyzna
6	Krówka	Rysio	1992-05-15	1	92051577646	1	Mężczyzna
7	Zebra	Wojtek	1993-03-13	1	93030399846	4	Mężczyzna
8	Gazela	Basia	1992-11-11	0	92111177446	2	Kobieta
9	Sarenka	Rysio	1992-12-12	0	92121278766	1	Kobieta
10	Konik	Kasia	1993-03-12	0	93031275446	2	Kobieta
11	Ryba	Jan	1993-05-15	1	93051587746	2	Mężczyzna
12	Kura	Kasia	1993-02-22	0	93022277654	2	Kobieta
13	Łoś	Jola	1993-06-02	0	93060288788	2	Kobieta
14	Miś	Wacek	1993-03-11	1	93031199123	1	Mężczyzna

Rysunek 2.
Tabela w bazie danych

Nie zastanawiamy się, jak te dane są przechowywane fizycznie na dysku ani jaki wpływ na wydajność mogą mieć nasze decyzje podjęte przy projektowaniu tabeli. Zrozumienie podstaw ułatwi później uzmysłowienie sobie, dlaczego w takiej czy innej sytuacji wykonanie zapytania czy modyfikacji danych trwa tak długo.

Najmniejszą jednostką przechowywania danych w SQL Server jest **strona** (ang. *page*). Jest to 8 KB blok składający się z nagłówka i 8 060 bajtów na dane wiersza (lub wierszy). Przy założeniu, że wiersz tabeli musi się zmieścić na stronie jasno widać, że maksymalny rozmiar wiersza to 8 060 bajtów. Część danych o rozmiarze przekraczającym 8 KB jest zapisywana na innych stronach, a w samym wierszu umieszczany jest tylko wskaźnik do pierwszej z tych stron. W SQL Server rozróżniamy kilka typów stron przechowujących informacje o rozmaitym znaczeniu:

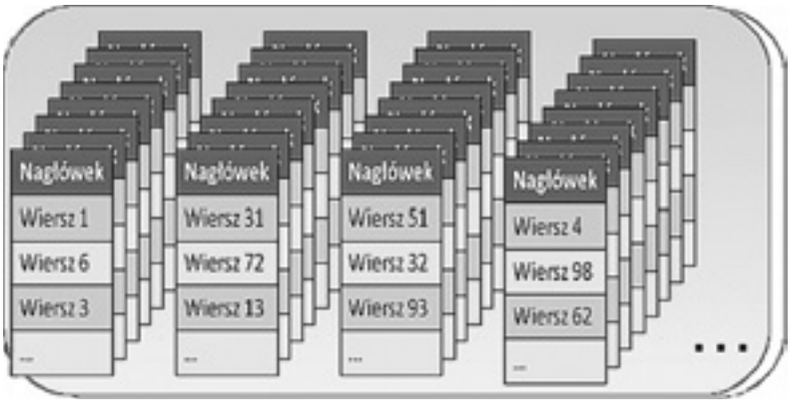
- Strony danych (data) zawierają wszystkie dane z wiersza, za wyjątkiem kolumn typów: text, ntext, image, nvarchar(max), varchar(max), varbinary(max), xml.
- Jeżeli wiersz nie mieści się w limicie długości 8 060 bajtów, to najdłuższa z kolumn jest przenoszona do tzw. **strony przepełnienia** (strona danych), a w jej miejscu w wierszu zostaje 24-bajtowy wskaźnik.
- Strony indeksów (index) zawierają poszczególne wpisy indeksu. W ich przypadku istotny jest limit długości klucza indeksu – 900 bajtów.
- Strony obiektów BLOB/CLOB (Binary/Character Large Object) (text/image) służą do przechowywania danych o rozmiarze do 2 GB.
- Strony IAM (wrócimy do nich w dalszej części kursu, gdy poznamy kolejne pojęcia dotyczące fizycznego przechowywania danych).

Wymieniliśmy tylko 5 rodzajów stron, żeby niepotrzebnie nie komplikować dalszych rozważań. Podstawową jednostką alokacji nie jest jednak w SQL Server strona, tylko zbiór ośmiu stron zwany **obszarem** (ang. *extent*) – rysunek 3.

Rysunek 3.
Obszar

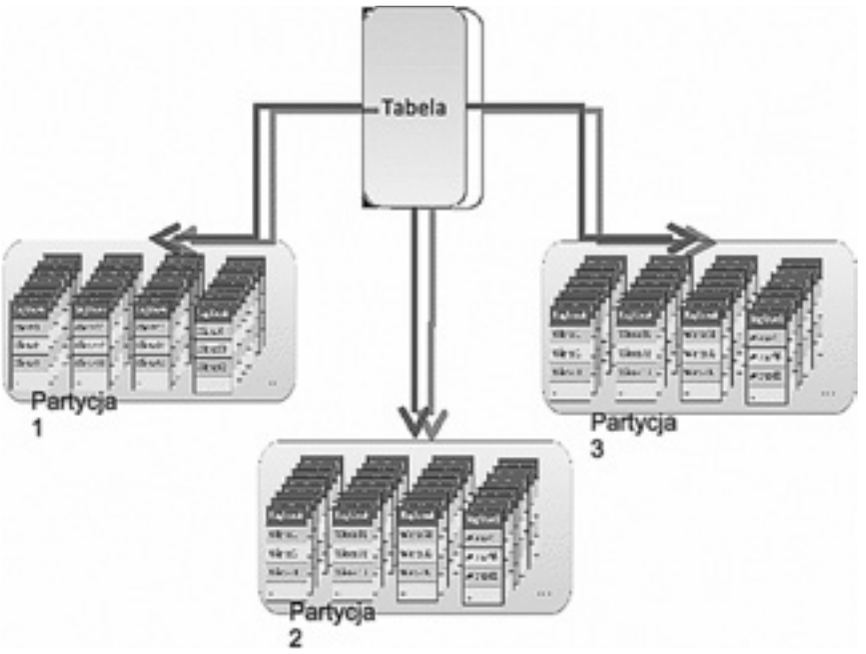
Jest tak ze względu na fakt, iż 8 KB to trochę za mało jak na operacje w systemie plików, a 64 KB to akurat jednostka alokacji w systemie plików NTFS. Obszary mogą zawierać strony należące do jednego obiektu (tabeli czy indeksu) – nazywamy je wtedy jednolitymi (uniform) – lub do wielu obiektów – stają się wtedy obszarami mieszanymi (mixed). Jeżeli SQL Server alokuje miejsce na nowe dane, to najmniejszą jednostką jest właśnie obszar.

Jeżeli tabela nie zawiera żadnego indeksu (o indeksach będziemy mówić w dalszej części kursu), to jej dane tworzą **stertę** – nieuporządkowaną listę stron należących do tej tabeli. Wszelkie operacje wyszukiwania na stercie odbywają się wolno, gdyż wymagają zawsze przejrzenia wszystkich stron. Inaczej w żaden sposób serwer nie jest w stanie stwierdzić, czy np. odnalazł już wszystkie wiersze spełniające określone kryterium wyszukiwania. Stertę można wyobrazić sobie jak pokazano na rysunku 4.



Rysunek 4.
Przykładowa sterta

Dodatkowo tabela może zostać podzielona na partycje (względy wydajnościowe – zrównoleglenie operacji wejścia wyjścia). W takim przypadku każda z partycji zawiera własną stertę. Wszystkie razem tworzą zbiór danych tabeli (rysunek 5).



Rysunek 5.
Tabela, partycja, sterty

Gdy SQL Server alokuje miejsce w plikach bazy danych, wypełnia je obszarami, które wstępnie są oznaczone jako wolne. Podobnie wszystkie strony w obszarach są oznaczone jako puste. Musimy wyjaśnić, w jaki sposób przechowywane są informacje na temat tego, czy dany obszar lub strona są wolne lub należą do jakiegoś obiektu. Służą do tego specjalne strony – GAM, SGAM i IAM. Zawierają one informacje o zajętości poszczególnych obszarów w postaci map bitowych (GAM, SGAM) lub o przynależności obszarów do obiektów (tabel, indeksów) – IAM. Dostęp do danych tabeli, w przypadku sterty, uzyskujemy na podstawie stron IAM (ang. *Index Allocation Map*). Każda tabela posiada swoje strony IAM, na podstawie których można określić, które obszary dysku zawierają dane tej tabeli. Dane IAM możemy sobie wyobrazić jako ciąg bitów (wartości 0 lub 1), gdzie każdy bit odpowiada obszarowi przestrzeni dyskowej. Jeżeli kolejna wartość bitu wynosi 1, to znaczy, że odpowiadający temu bitowi obszar dysku zawiera dane należące do tego obiektu.

1.4. ĆWICZENIA.

1.4.1. Ćwiczenie 1.Analiza wykonania zapytania do dużej tabeli.

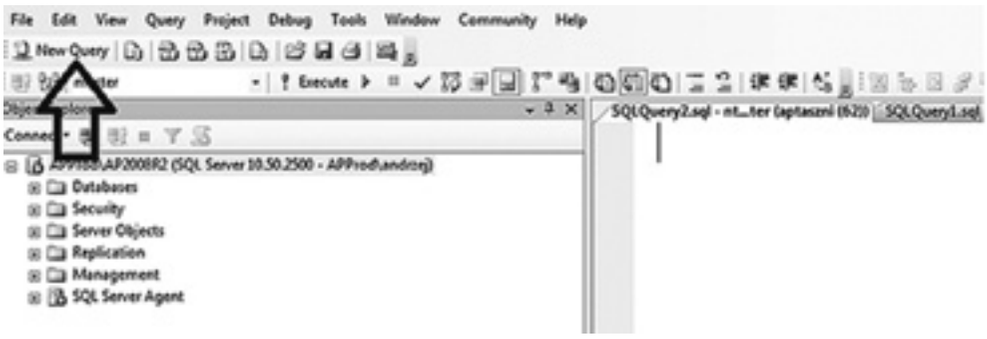
W ramach ćwiczenia sprawdzimy parametry realizacji zapytania skierowanego do tabeli zawierającej 40 mln wierszy.

W przykładowej bazie danych znajduje się tabela o nazwie Pacjenci, w której znajduje się ok. 40 mln wierszy. Strukturę tabeli pokazano na rysunku 6.

Column Name	Data Type	Allow Nulls
IdPacjenta	uniqueidentifier	☐
Nazwisko	nvarchar(128)	☐
Imie	nvarchar(64)	☐
DataUrodzenia	date	☐
Zamieszkanie	geography	☐
DaneDodatkowe	xml	☐
CzyKobieta	bit	☑
plec		☐
Aktualizacja	bit	☑
DataZgonu	date	☑
CzyZyje	bit	☑
idTeryt	int	☑
Pesel	varchar(11)	☑

Rysunek 6.
Struktura tabeli Pacjenci

Po zalogowaniu się do serwera bazy danych (parametry logowania przekaże prowadzący kurs), w środowisku MS SQL Server Management Studio otwieramy okno edycyjne (zadanie New Query) jak pokazano na rysunku 7.



Rysunek 7.
Wybieranie zadania New Query w Management Studio

W oknie edycyjnym piszemy następujące polecenia:

```
SET STATISTICS IO ON

DECLARE @data DATETIME=GETDATE()

SELECT Nazwisko, Imie, DataUrodzenia
FROM Pacjenci
WHERE Pesel='73010500109'

SELECT  DATEDIFF( s,@data,GETDATE() ) as CzasWykonania
```

Poszczególne polecenia realizują następujące zadania:

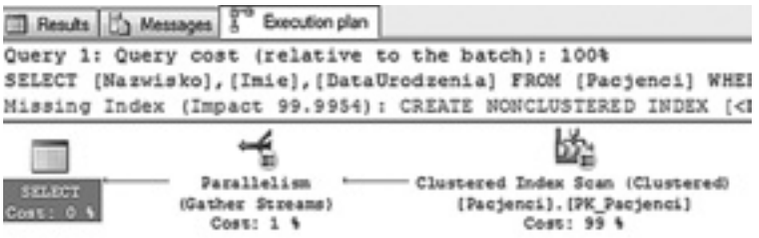
- SET STATISTICS IO ON – ustawia opcję systemową powodującą, że po wykonaniu każdego polecenia SQL, będą wyświetlane statystyki operacji wejścia-wyjścia,
- DECLARE @data DATETIME=GETDATE() – deklarujemy zmienną o nazwie @data, w której zapamiętujemy aktualny czas,
- SELECT Nazwisko, Imie, DataUrodzenia
  FROM Pacjenci
  WHERE Pesel='73010500109' – zapytanie, które wyszukuje w tabeli Pacjenci dane pacjenta o podanym numerze Pesel,
- SELECT DATEDIFF(s,@data,GETDATE()) as CzasWykonania – polecenie wyświetlajace czas (w sekundach) wykonywania zapytania.

Wykonujemy zapisane polecenia poprzez wciśnięcie klawisza F5.

Po wykonaniu poleceń, w oknie Message, wyświetlone zostaną statystyki operacji wejścia-wyjścia. W naszym przypadku otrzymujemy następujące statystyki:

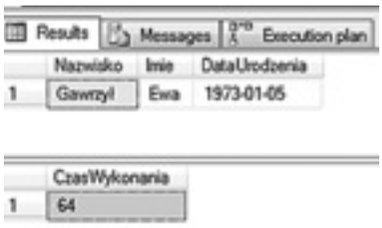
```
Table ‚Pacjenci'. Scan count 9,
        logical reads 934173,
        physical reads 4509,
        read-ahead reads 736594,
        lob logical reads 0,
        lob physical reads 0,
        lob read-ahead reads 0.
```

Nie wnikając w szczegóły widzimy, że wykonanie zapytania wymagało 934 173 odczytów (logical reads 934 173). Ponieważ jeden odczyt to jedna strona danych (8 KB) to serwer musiał przeczytać ok. 8 GB danych. Na rysunku 8 pokazano plan wykonania zapytania (plany wykonania zostaną omówione w dalszej części kursu).



Rysunek 8.
Plan wykonania zapytania

Wynik wykonanych poleceń pokazano na rysunku 9.



Rysunek 9.
Wynik wykonanych poleceń

W pierwszym zestawie danych (rysunek 8) jest wynik zapytania, a w drugim czas jego wykonania w sekundach. Zapytanie wykonywało się 64 sekundy, co w większości systemów byłoby czasem niedopuszczalnym, ponieważ trudno czekać ponad minutę na udostępnienie danych jednej osoby.

Wykonane ćwiczenie pokazuje, że gdy mamy duże ilości danych zapisane w tabeli, to wykonywanie zapytań może trwać długo, w sytuacji gdy muszą być przeczytane wszystkie dane. W dalszej części kursu poznamy indeksy, które pomagają rozwiązać takie problemy.

2 MECHANIZM TRANSAKCYJNY.

2.1. ISTOTA TRANSAKCJI W BAZIE DANYCH.

Bardzo istotnym zagadnieniem związanym z relacyjnymi bazami danych jest mechanizm transakcji. Przy tworzeniu aplikacji bazodanowych rzadko zastanawiamy się, czy baza zapewnia nam stabilność i bezpieczeństwo danych. Przyjmujemy, że tak. Warto natomiast zdawać sobie sprawę, jakie mechanizmy leżą u podstaw tego przekonania. W ramach kursu omówimy jeden z podstawowych mechanizmów – transakcje. W systemie informatycznym dane w bazie reprezentują zwykle aktualny stan biznesu (procesu produkcji, stanów magazynowych, alokacji zasobów itp.).

Ponieważ sytuacja biznesowa jest zmienna, to dane w bazie także podlegają ciągłym zmianom. Każda z takich zmian stanowi swego rodzaju całość i składa się często z wielu elementarnych modyfikacji danych. Tylko prawidłowe wykonanie wszystkich kroków w ramach takiej zmiany ma sens z punktu widzenia biznesowego. W takiej sytuacji nietrudno wyobrazić sobie, do czego może prowadzić przerwanie takiej złożonej operacji w trakcie jej realizacji – powstaje stan nieustalony. Jest to niedopuszczalna sytuacja, która może wprowadzać chaos. Na szczęście serwery baz danych potrafią sobie z nią radzić poprzez mechanizm obsługi transakcji.

Wyobraźmy sobie sytuację, gdy w pewnym systemie bankowym realizujemy operację przelewu pieniędzy z konta A na konto B. Z punktu widzenia systemu bakowego jest to jedna operacja, natomiast z punktu widzenia bazy danych możemy sobie, w uproszczeniu, wyobrazić realizację następujących działań:

- Sprawdzenie, czy na koncie A znajduje się odpowiednia kwota,
- zmniejszenia stanu konta A o kwotę realizowanego przelewu,
- zwiększenie konta B o kwotę realizowanego przelewu.

Niedopuszczalna byłaby sytuacja, gdyby udało się wykonać tylko niektóre z tych operacji.

Transakcją nazywamy sekwencję logicznie powiązanych ze sobą operacji na danych zawartych w bazie, które przeprowadzają bazę danych z jednego stanu spójnego do drugiego Na początku lat osiemdziesiątych XX wieku pojawił się akronim **ACID** (ang. *Atomicity, Consistency, Isolation, Durability*) określający w zwięzły i łatwy do zapamiętania sposób podstawowe właściwości transakcji:

- **Atomicity** (atomowość) definiuje właściwość określającą, że transakcja jest niepodzielna. Oznacza to, że albo wszystkie operacje wchodzące w jej skład wykonają się poprawnie w całości, albo wcale. Nie istnieje możliwość wykonania części transakcji.

- **Consistency** (spójność) oznacza, że baza danych przed rozpoczęciem znajduje się w stanie stabilnym i po zakończeniu transakcji (niezależnie czy przez zatwierdzenie, czy wycofanie) też ma pozostać w stanie stabilnym.
- **Isolation** (odizolowanie) odnosi się do faktu, iż transakcje są od siebie logicznie odseparowane. Mogą oddziaływać między sobą tak, jakby były wykonywane sekwencyjnie.
- **Durability** (trwałość) jest właściwością, która gwarantuje, że jeżeli transakcja została zatwierdzona, to nawet w przypadku awarii systemu lub zasilania, nie zostanie utracona lub wycofana.

Wszystkie te cechy zostały zaimplementowane w serwerach baz danych i dzięki temu mamy zapewnioną spójność danych i gwarancję, że niezależnie od okoliczności baza danych będzie zawsze w stabilnym stanie. Ma to kluczowe znaczenie dla większości systemów informatycznych tworzonych na potrzeby biznesu i nie tylko.

2.2. TYPY I RODZAJE TRANSAKCJI.

Domyślnie SQL Server jest skonfigurowany w ten sposób, że transakcje obsługuje w trybie AUTOCOMMIT. Oznacza to, że są one automatycznie rozpoczynane po natrafieniu na polecenie modyfikujące dane lub strukturę bazy, oraz zatwierdzane zaraz po poprawnym wykonaniu tego polecenia. Można oczywiście sterować procesem tworzenia i zatwierdzania transakcji. Realizuje się to poprzez korzystanie z trybu IMPLICIT TRANSACTION – w którym transakcje rozpoczynane są tak jak w trybie AUTOCOMMIT, ale wymagają „ręcznego" zakończenia poprzez wydanie polecenia COMMIT lub ROLLBACK. Drugim wariantem jest tryb EXPLICIT TRANSACTIONS, który polega na rozpoczynaniu transakcji poleceniem BEGIN TRANSACTION i kończeniu jej poleceniem COMMIT lub ROLLBACK (tak jak w IMPLICIT).

Z obsługą transakcji związane jest kilka poleceń języka SQL:

- BEGIN TRANSACTION – rozpoczyna transakcję,
- COMMIT TRANSACTION – kończy transakcję i zatwierdza jej wyniki,
- ROLLBACK TRANSACTION – kończy transakcję i anuluje jej wyniki (usuwa zmiany wykonane na danych w ramach transakcji).

Transakcje mogą być zagnieżdżane. Przy zagnieżdżaniu transakcji warto pamiętać, że mechanizm ten nie działa do końca w intuicyjny sposób – pary poleceń BEGIN TRANSACTION i COMMMIT (lub ROLLBACK) nie pełnią roli pary nawiasów. Żeby uniknąć problemów warto dokładnie zapoznać się z mechanizmem działania transakcji zagnieżdżonych i przećwiczyć go w praktyce.

SQL Server posiada wbudowaną zmienną @@TRANCOUNT, w której przechowywany jest aktualny poziom zagnieżdżenia. Jeśli nie ma w danej chwili żadnej aktywnej transakcji – ma wartość 0. Każde polecenie BEGIN TRANSACTION zwiększa wartość zmiennej @@TRANCOUNT. Dzięki temu przy tworzeniu kodu procedur składowanych możemy zawsze łatwo sprawdzić, na jakim poziomie zagnieżdżenia jesteśmy i odpowiednio zareagować.

Przy bardziej złożonych transakcjach można budować bardziej skomplikowany przebieg transakcji. Do tego celu służy polecenie SAVE, które tworzy w ramach transakcji punkt, do którego może być ona cofnięta bez konieczności wycofywania całej transakcji. Pozwala to na budowanie alternatywnych ścieżek realizacji transakcji.

2.3. ANOMALIE ZWIĄZANE Z PRZETWARZANIEM TRANSAKCYJNYM.

Przy realizacji więcej niż jednej transakcji w tym samym czasie, mogą pojawić się różne problemy związane z uzyskiwaniem przez nie dostępu do danych i modyfikowaniem ich.

Typowe przykłady takich zjawisk to:

- Lost update (zgubiona modyfikacja). Zjawisko to można zilustrować przykładem, gdy dwie transakcje T1 i T2 odczytały wartość z bazy, następnie każda z nich próbuje zapisać tę wartość po modyfikacji. W takim przypadku transakcja, która zostanie zatwierdzona później, nadpisze modyfikacje dokonane przez transakcję zatwierdzoną wcześniej.

- Dirty read (brudny odczyt). Typowy przykład to zliczanie odwiedzin strony przez użytkowników aplikacji. Polega ono na odczytaniu aktualniej liczby odwiedzin z bazy, powiększeniu jej o 1 i zapisaniu z powrotem do bazy. Jeśli teraz transakcja T1 odczytuje dane (np. wartość 5), dokonuje ich inkrementacji i zapisuje w bazie, ale nie zostaje zatwierdzona. W tym samym czasie T2 odczytuje dane zapisane przez T1 (wartość 6), następnie inkrementuje ją. W tym momencie T1 zostaje wycofana (wartość powinna zostać ponownie zmieniona na 5). T2 zostaje zatwierdzona i w bazie ląduje wartość 7, co stanowi ewidentny błąd.
- Nonrepeatable reads (niepowtarzalny odczyt). Przykładem może być transakcja T1, w której pobierane są do modyfikacji wszystkie niezrealizowane zamówienia. W kolejnym kroku informacje te są przetwarzane. W tym czasie transakcja T2 modyfikuje status kilku zamówień (zmienia na zrealizowane) i zostaje zatwierdzona. Teraz z kolei T1 ponownie sięga do listy zamówień niezrealizowanych i... otrzymuje listę inną niż poprzednio, co może spowodować błędy w przetwarzaniu danych – załóżmy, że po pierwszym odczycie tworzone były zlecenia dla magazynierów (pobranie towarów dla zamówień), a przy drugim były generowane listy przewozowe. Efektem będzie niespójność – brak kilku listów przewozowych.
- Phantom reads (odczyt-widmo). Sytuacja może być podobna jak poprzednio. Transakcja T1 pobiera zamówienia przeznaczone do realizacji na dziś. Zaczyna je przetwarzać. W tym czasie transakcja T2 przekazuje jeszcze dwa zlecenia do realizacji na dziś i zostaje zatwierdzona. T1 ponownie sięga po dane zamówień i otrzymuje dodatkowe dwa zamówienia, dla których nie zostały wykonane czynności z pierwszego etapu przetwarzania! Podobnie jak poprzednio nie jest to dobra sytuacja i prowadzi do powstania niespójności.

Można oczekiwać, że środowisko bazy danych wyeliminuje możliwość wystąpienia omówionych anomalii. Podstawowym problemem, w sytuacji gdy chcemy uniknąć wszelkich anomalii, jest wydajność systemu. Bardzo często godzimy się na możliwość wystąpienia anomalii w celu zapewnienia większej wydajności. Technologia SQL Server umożliwia sterowanie poziomem izolacji transakcji.

2.4. POZIOMY IZOLACJI TRANSAKCJI.

Skoro istnieje problematyka spójności danych, przy jednoczesnym wykonywaniu wielu transakcji, to czy nie jest dobrym rozwiązaniem kolejkowanie transakcji i wykonywanie ich sekwencyjnie? Zapewniłoby to brak konfliktów przy modyfikacji danych z poziomu różnych transakcji. Niestety nie jest to jedyny problem. Równie istotna pozostaje kwestia wydajności, a przy zaproponowanym podejściu nie byłaby ona zadowalająca. Żeby móc podnieść wydajność i jednocześnie zachować wszystkie właściwości transakcji istnieje kilka tzw. poziomów izolacji. Każdy z nich korzysta z nieco innej strategii stosowania blokad, co pozwala na równoległe wykonywanie niektórych operacji modyfikacji danych i ich odczytu z poziomu innych transakcji.

W zależności od operacji wchodzących w skład transakcji oraz specyfiki aplikacji korzystającej z bazy danych, możemy stosować jeden z kilku poziomów izolacji transakcji. Każdy z nich cechuje się tym, że eliminuje możliwość wystąpienia kolejnego rodzaju konfliktu, przez co podnosi poziom bezpieczeństwa transakcji, ale jednocześnie powoduje obniżenie wydajności. Celem projektantów baz danych i aplikacji jest znalezienie rozsądnego kompromisu pomiędzy wydajnością a poziomem izolacji dla konkretnych transakcji przeprowadzanych w ramach działania aplikacji. Każdy rodzaj transakcji przeprowadzanej przez aplikację warto przeanalizować pod kątem wymaganego poziomu izolacji. Dąży się do tego, aby stosować możliwie najniższy poziom izolacji, aby móc zachować z jednej strony bezpieczeństwo i spójność danych, a z drugiej zadowalającą wydajność.

Poziom izolacji	Dirty read	Nonrepeatable read	Phantom read
READ UNCOMMITED	TAK	TAK	TAK
READ COMMITED	NIE	TAK	TAK
REPEATABLE READ	NIE	NIE	TAK
SERIALIZABLE	NIE	NIE	NIE

Tabela 1.
Występowanie anomalii przy różnych poziomach izolacji transakcji

Poziom izolacji transakcji jest parametrem każdej sesji otwartej w bazie danych. Stwarza to możliwości sterowania w zależności od rodzaju wykonywanych operacji. Poziom izolacji transakcji możemy traktować jako kontrakt zawierany ze środowiskiem serwera bazy danych.

2.5. ZAKLESZCZENIA.

Przy realizacji wielu transakcji jednocześnie, czasem dochodzi do zjawiska zakleszczenia (ang. *deadlock*). Jest to zjawisko zdecydowanie negatywne i nie ma uniwersalnego sposobu, żeby je na 100% wyeliminować. Do tego jest to problem nieodwracalny i jedynym rozwiązaniem jest wycofanie jednej z zakleszczonych transakcji.

Obrazowo problem zakleszczenia można przedstawić na przykładzie rysowania wykresu na tablicy. Potrzebne do tego są dwa zasoby: linijka i kreda. Przyjmijmy, że mamy dwóch chętnych do rysowania wykresu. Wiedzą oni, co jest do tego potrzebne. Pierwszy ochotnik sięga po kredę i zaczyna rozglądać się za linijką. W tym samym czasie drugi ochotnik chwycił linijkę i szuka kredy. Dochodzi do sytuacji, w której obaj blokują sobie nawzajem potrzebne zasoby, czekając na zwolnienie przez konkurenta drugiego potrzebnego do rysowania wykresu zasobu. Sytuacja jest patowa. W takiej sytuacji SQL Server korzysta z prostego i brutalnego mechanizmu. Losuje jedną z zakleszczonych transakcji (staje się ona ofiarą – deadlock victim) i wycofuje ją z odpowiednim komunikatem błędu. To pozwala drugiej z transakcji na uzyskanie wszystkich potrzebnych zasobów i zrealizowanie wszystkich zaplanowanych czynności.

Zakleszczenie jak już wspominaliśmy jest nie do uniknięcia. Jedyne co można zrobić to starać się minimalizować szanse jego wystąpienia. Wbrew pozorom nie musi to być bardzo skomplikowane. Sięganie do zasobów w tej samej kolejności pozwala na wyeliminowanie większości przypadków zakleszczeń w bazie danych. Takie podejście powoduje, że pierwsza transakcja, której uda się zdobyć pierwszy zasób ma gwarancję, że żadna inna transakcja tego samego typu nie zajmie innego zasobu potrzebnego do realizacji zaplanowanych operacji.

Przy korzystaniu z transakcji warto trzymać się kilku zasad, które pozwalają na wykonywanie transakcji w możliwie najwydajniejszy sposób.

Przede wszystkim pilnujmy długości transakcji. Im dłuższa transakcja, tym dłużej aktywne są blokady przez nią utworzone i inne transakcje nie mogą korzystać z zasobów. Pod żadnym pozorem nie należy w ramach transakcji prowadzić jakiejkolwiek interakcji z użytkownikiem! Ze względów zachowania spójności danych, transakcja powinna zawierać jedynie kod, który musi być wykonany w jej ramach. Wszelkie inne operacje mogą odbywać się przed lub po transakcji. Można tu zauważyć podobieństwo do programowania współbieżnego i sekcji krytycznej.

W ramach transakcji warto zaplanować kolejność uzyskiwania dostępu do zasobów tak, aby minimalizować ryzyko wystąpienia zakleszczeń. Zagadnienie to staje się tym bardziej skomplikowane, im więcej różnych operacji jest realizowanych z wykorzystaniem osobnych transakcji.

W specyficznych przypadkach można podpowiedzieć serwerowi, jakich rodzajów blokad ma użyć w ramach wykonywania transakcji. Jest to jednak zdecydowanie margines zastosowań. W miarę możliwości nie należy ingerować w ten mechanizm, gdyż sam z siebie działa bardzo dobrze. Nie w pełni przemyślana ingerencja może drastycznie obniżyć wydajność.

Dobranie odpowiedniego poziomu izolacji dla transakcji jest także bardzo istotne. W przypadkach gdy w ramach transakcji ryzyko wystąpienia konfliktów (dirty reads itp.) nie stanowi zagrożenia dla wykonywanej operacji warto poziom izolacji obniżać. Jedynie w przypadku krytycznych operacji sięgajmy po poziom SERIALIZABLE. Takie podejście pozwoli uzyskać lepszą wydajność bazy danych.

2.6. ĆWICZENIA.

2.6.1. Ćwiczenie 2 – Obsługa transakcji jawnych.

W ramach ćwiczenia sprawdzimy działanie poleceń sterujących wykonaniem transakcji. W tym celu wykonujemy następujące czynności:

- W Management Studio otwieramy okno edycyjne w bzie danych ElektronicznyDziennikOcen (zadanie New Query),
- W oknie edycyjnym wykonujemy kolejno następujące polecenia:
  - BEGIN TRANSACTION – polecenie rozpoczyna transakcję,
  - INSERT INTO Uczniowie (Nazwisko, Imie, DataUrodzenia, Pesel, CzyChlopak, idklasy)
    VALUES(‘Nowak’ , ‘Zbigniew’ , ‘1967-12-22’, ‘67122245436’,0,5) – polecenie zapisujące do tabeli Uczniowie nowy wiersz (w tym miejscu należy wstawić swoje dane),
  - SELECT * FROM Uczniowie WHERE Pesel=’67122245436’ – odczytanie z tabeli zapisanego wiersza – wynik zapytania pokazano na rysunku 10,

iducznia	Nazwisko	Imie	DataUrodzenia	CzyChlopak	Pesel	idklasy	plec
612	Nowak	Zbigniew	1967-12-22	0	67122245436	5	Kobieta

Rysunek 10.
Wynik zapytania

  - ROLLBACK TRANSACTION – kończymy transakcję z wycofaniem jej skutków,
  - Powtarzamy zapytanie SELECT * FROM Uczniowie WHERE Pesel=’67122245436’ – wynik zapytania pokazano na rysunku 11.

Rysunek 11.
Wynik zapytania po anulowaniu transakcji (polecenie ROLLBACK)

Polecenie INSERT zapisało nowy wiersz do tabeli Uczniowie i w ramach transakcji widzieliśmy tego wynik na rysunku 10. Po anulowaniu transakcji zapisane dane zostały usunięte, ponieważ polecenie ROLLBACK przywraca stan bazy danych do początku transakcji – wynik zapytania nie zwraca żadnych danych (rysunek 11).

- wykonujemy ponownie sekwencje poleceń, zastępując polecenie ROLLBACK poleceniem COMMIT:

```
BEGIN TRANSACTION

INSERT INTO Uczniowie(Nazwisko, Imie,DataUrodzenia,Pesel,CzyChlopak,idklasy)
VALUES(,Nowak’ , ,Zbigniew’ , ,1967-12-22’, ,67122245436’,0,5 )

SELECT * FROM Uczniowie
WHERE Pesel=’67122245436’

COMMIT TRANSACTION

SELECT * FROM Uczniowie
WHERE Pesel=’67122245436’
```

- po wykonaniu sekwencji poleceń widzimy, że dane zapisane poleceniem INSERT zostały zatwierdzone i zpytanie wykonane po poleceniu COMMIT zwrace dane zapisanego wiersza.

Problemy i uwagi do ćwiczenia omówić z prowadzącym kurs.

2.6.2. Ćwiczenie 3 – Analiza poziomu izolacji READ COMMITTED.

W ramach ćwiczenia sprawdzimy działanie transakcji odwołujących się do tych samych danych przy poziomie izolacji transakcji READ COMMITTED. Ponieważ poziom ten jest poziomem domyślnym, to nie musimy wydawać poleceń zmieniających poziom izolacji transakcji.

W celu zademonstrowania współdziałania transakcji przy poziomie izolacji READ COMMITTED realizujemy następujące czynności:

- słuchacze otwierają nowe okno edycyjne w bazie danych ElektronicznyDzinnikOcen i wykonują w nim polecenie SELECT * FROM UCZNIOWIE – w wyniku polecenia wyświetlona zostaje zawartość tabeli Uczniowie jak pokazano na rysunku 12.

iducznia	Nazwisko	Imie	DataUrodzenia	CzyChlopak	Pesel	idklasy	plec
1	Kotek	Katarzyna	1992-03-12	0	92031275446	2	Kobieta
2	Piesek	Jan	1992-05-15	1	92051587746	1	Mężczyzna
3	Lisek	Kasia	1992-02-22	0	92022277654	2	Kobieta
4	Kurka	Jola	1992-06-02	0	92060288788	2	Kobieta
5	Gąska	Wacek	1991-03-11	1	91031199123	1	Mężczyzna
6	Krówka	Rysio	1992-05-15	1	92051577646	1	Mężczyzna
7	Zebra	Wojtek	1993-03-13	1	93030399846	4	Mężczyzna
8	Gazela	Basia	1992-11-11	0	92111177446	2	Kobieta
9	Sarenka	Rysio	1992-12-12	0	92121278766	1	Kobieta
10	Konik	Kasia	1993-03-12	0	93031275446	2	Kobieta
11	Ryba	Jan	1993-05-15	1	93051587746	2	Mężczyzna
12	Kura	Kasia	1993-02-22	0	93022277654	2	Kobieta
13	Łoś	Jola	1993-06-02	0	93060288788	2	Kobieta
14	Miś	Wacek	1993-03-11	1	93031199123	1	Mężczyzna
15	Okoń	Rysio	1993-05-15	1	93051577646	2	Mężczyzna

Rysunek 12.
Zawartość tabeli Uczniowie

- prowadzący kurs otwiera nowe okno edycyjne w bazie danych ElektronicznyDzinnikOcen i wykonuje polecenie BEGIN TRANSACTION, a następnie polecenie UPDATE Uczniowie SET Nazwisko=’Pomyłka!!!’ – w każdym wierszu nazwisko ucznia przyjmie wartość ‘Pomyłka!!!’,
- prowadzący kurs wykonuje i demonstruje wynik zapytania SELECT * FROM Uczniowie – wynik zapytania pokazano na rysunku 13,

iducznia	Nazwisko	Imie	DataUrodzenia	CzyChlopak	Pesel	idklasy	plec
1	Pomyłka!!!	Katarzyna	1992-03-12	0	92031275446	2	Kobieta
2	Pomyłka!!!	Jan	1992-05-15	1	92051587746	1	Mężczyzna
3	Pomyłka!!!	Kasia	1992-02-22	0	92022277654	2	Kobieta
4	Pomyłka!!!	Jola	1992-06-02	0	92060288788	2	Kobieta
5	Pomyłka!!!	Wacek	1991-03-11	1	91031199123	1	Mężczyzna
6	Pomyłka!!!	Rysio	1992-05-15	1	92051577646	1	Mężczyzna
7	Pomyłka!!!	Wojtek	1993-03-13	1	93030399846	4	Mężczyzna
8	Pomyłka!!!	Basia	1992-11-11	0	92111177446	2	Kobieta
9	Pomyłka!!!	Rysio	1992-12-12	0	92121278766	1	Kobieta
10	Pomyłka!!!	Kasia	1993-03-12	0	93031275446	2	Kobieta
11	Pomyłka!!!	Jan	1993-05-15	1	93051587746	2	Mężczyzna
12	Pomyłka!!!	Kasia	1993-02-22	0	93022277654	2	Kobieta
13	Pomyłka!!!	Jola	1993-06-02	0	93060288788	2	Kobieta
14	Pomyłka!!!	Wacek	1993-03-11	1	93031199123	1	Mężczyzna
15	Pomyłka!!!	Rysio	1993-05-15	1	93051577646	2	Mężczyzna

Rysunek 13.
Zawartość tabeli Uczniowie po wykonaniu modyfikacji danych

- słuchacze ponownie wykonują zapytanie SELECT * FROM Uczniowie – zapytanie nie zwraca wyników, ponieważ próbuje odczytać dane zmodyfikowane w transakcji zrealizowanej przez prowadzącego kurs (transakcja ta jest jeszcze niezakończona),

- prowadzący anuluje swoją transakcję wykonując polecenie ROLLBACK TRANSACTION,
- po anulowaniu transakcji, uczestnicy otrzymują wynik swoich zapytań taki jak na rysunku 12.

W trakcie ćwiczenia zauważyliśmy następujące problemy:

- słuchacze nie otrzymali w ramach realizowanych zapytań danych, które były zmienione w ramach niezakończonej transakcji,
- zapytania skierowane do danych modyfikowanych, w ramach niedokończonej transakcji, oczekiwały na jej zakończenie (SQL Server zastosował odpowiednie blokady danych),
- po zakończeniu transakcji zapytania słuchaczy zostały niezwłocznie wykonane zwracając wynik jak na rysunku 12, ponieważ prowadzący anulował swoją transakcję (polecenie ROLLBACK).

UWAGA: Ponieważ w ramach tego ćwiczenia wykonywane są operacje na tych samych danych, należy postępować uważnie i nie wykonywać tych poleceń, które demonstruje prowadzący kurs, a jedynie te polecenia, które mają wykonać słuchacze.

Problemy i uwagi do ćwiczenia omówić z prowadzącym kurs.

2.6.3. Ćwiczenie 4. – Analiza poziomu izolacji READ UNCOMMITTED.

W ramach ćwiczenia sprawdzimy działanie transakcji odwołujących się do tych samych danych przy poziomie izolacji transakcji READ UNCOMMITTED. Ponieważ poziom ten nie jest poziomem domyślnym, to będziemy musieli wydawać poleceń zmieniających poziom izolacji transakcji.

W celu zademonstrowania współdziałania transakcji przy poziomie izolacji READ UNCOMMITTED realizujemy następujące czynności:

- słuchacze otwierają nowe okno edycyjne w bazie danych ElektronicznyDzinnikOcen i wykonują w nim polecenie SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED, a następnie polecenie SELECT * FROM UCZNIOWIE – w wyniku polecenia wyświetlona zostaje zawartość tabeli Uczniowie jak pokazano na rysunku 12,
- prowadzący kurs otwiera nowe okno edycyjne w bazie danych ElektronicznyDzinnikOcen i wykonuje polecenie BEGIN TRANSACTION, a następnie polecenie UPDATE Uczniowie SET Nazwisko='Pomyłka!!!' – w każdym wierszu nazwisko ucznia przyjmie wartość 'Pomyłka!!!',
- prowadzący kurs wykonuje i demonstruje wynik zapytania SELECT * FROM Uczniowie – wynik zapytania pokazano na rysunku 13,
- słuchacze ponownie wykonują zapytanie SELECT * FROM Uczniowie – zapytanie zwraca wyniki takie same jak w oknie prowadzącego (rysunek 13), chociaż transakcja nie została jeszcze zakończona,
- prowadzący anuluje swoją transakcję, wykonując polecenie ROLLBACK TRANSACTION,
- po anulowaniu transakcji, uczestnicy ponawiają zapytanie i otrzymują wynik swoich zapytań taki jak na rysunku 12.

W trakcie ćwiczenia zauważyliśmy następujące problemy:

- słuchacze mogli odczytać dane z niedokończonej transakcji,
- zapytania skierowane do danych modyfikowanych, w ramach niedokończonej transakcji, nie oczekiwały na jej zakończenie (SQL Server nie zastosował odpowiedniej blokady danych),
- po zakończeniu transakcji ponowione zapytania słuchaczy zwróciły wynik jak na rysunku 12, ponieważ prowadzący anulował swoją transakcję (polecenie ROLLBACK),
- można uznać, że słuchacze odczytali dane, których nigdy nie było w bazie danych (brudny odczyt), gdyż transakcja, która je modyfikowała została anulowana.

UWAGA: Ponieważ w ramach tego ćwiczenia wykonywane są operacje na tych samych danych, należy postępować uważnie i nie wykonywać tych poleceń, które demonstruje prowadzący kurs, a jedynie te polecenia, które maja wykonać słuchacze.

Problemy i uwagi do ćwiczenia omówić z prowadzącym kurs.

2.6.4. Ćwiczenie 5 – Analiza poziomu izolacji REPEATABLE READ.

W ramach ćwiczenia sprawdzimy działanie transakcji odwołujących się do tych samych danych przy poziomie izolacji transakcji REPEATABLE READ. Ponieważ poziom ten nie jest poziomem domyślnym to nie będziemy musieli wydawać poleceń zmieniających poziom izolacji transakcji. Należy zwrócić uwagę na fakt, że po raz pierwszy blokady danych będzie wprowadzało polecenie SELECT.

W celu zademonstrowania współdziałania transakcji przy poziomie izolacji REPEATABLE READ realizujemy następujące czynności:

- słuchacze otwierają nowe okno edycyjne w bazie danych ElektronicznyDzinnikOcen i wykonują w nim polecenie SET TRANSACTION ISOLATION LEVEL READ COMMITTED w celu przywrócenia domyślnego poziomu izolacji transakcji,
- prowadzący kurs otwiera nowe okno edycyjne w bazie danych ElektronicznyDzinnikOcen i wykonuje polecenia:
  - BEGIN TRANSACTION – rozpoczyna transakcję,
  - SET TRANSACTION ISOLATION LEVEL REPEATABLE READ – zmienia poziom izolacji transakcji,
  - SELECT * FROM Uczniowie WHERE IdKlasy=4 – odczytuje zawartość tabeli Uczniowie (uczniowie przypisani do klasy o wartości IdKlasy=4),
  - prowadzący w ramach rozpoczętej transakcji nie dokonał żadnej modyfikacji danych, a jedynie odczytał zawartość tabeli.
- słuchacze wykonują polecenie UPDATE Uczniowie SET Idklasy=5 WHERE IdKlasy=4,
- polecenia oczekują na wykonanie, ponieważ są blokowane przez transakcję rozpoczętą przez prowadzącego – poziom izolacji REPEATABLE READ gwarantuje, że dane odczytane w ramach transakcji nie ulegną zmianie do jej zakończenia,
- prowadzący zatwierdza swoja transakcję wykonując polecenie COMMIT i polecenia oczekujące u słuchaczy zostają wykonane.

W dalszej części ćwiczenia realizujemy następujące czynności:

- prowadzący kurs otwiera nowe okno edycyjne w bazie danych ElektronicznyDzinnikOcen i wykonuje polecenia:
  - BEGIN TRANSACTION – rozpoczyna transakcję,
  - SET TRANSACTION ISOLATION LEVEL REPEATABLE READ – zmienia poziom izolacji transakcji,
  - SELECT * FROM Uczniowie WHERE IdKlasy=4 – odczytuje zawartość tabeli Uczniowie (uczniowie przypisani do klasy o wartości IdKlasy=4),
  - prowadzący w ramach rozpoczętej transakcji nie dokonał żadnej modyfikacji danych, a jedynie odczytał zawartość tabeli.
- słuchacze wykonują polecenie:
  - INSERT INTO Uczniowie(Nazwisko, Imie,DataUrodzenia,Pesel,CzyChlopak,idklasy) VALUES(‘Nowak’ , ‘Zbigniew’ , ‘1967-12-22’, ‘67122245436’,0,4) (w klauzuli VALUES można zapisać innej dane)
- polecenia zostają niezwłocznie wykonane, ponieważ nie modyfikują danych odczytanych w ramach niedokończonej transakcji,
- prowadzący ponawia zapytanie SELECT * FROM Uczniowie WHERE IdKlasy=4 i niezwłocznie otrzymuje wynik, w którym są nowe wiersze danych wprowadzone poleceniami INSERT wykonanymi przez słuchaczy,
- prowadzący zatwierdza swoją transakcję wykonując polecenie COMMIT.

W trakcie ćwiczenia zauważyliśmy następujące problemy:

- słuchacze mogli dopisywać nowe dane do tabeli,
- ponowione, w ramach transakcji zapytanie zwracało inną ilość wierszy.

UWAGA: Ponieważ w ramach tego ćwiczenia wykonywane są operacje na tych samych danych, należy postępować uważnie i nie wykonywać tych poleceń, które demonstruje prowadzący kurs, a jedynie te polecenia, które maja wykonać słuchacze.

Problemy i uwagi do ćwiczenia omówić z prowadzącym kurs.

2.6.5. Ćwiczenie 6 – Analiza poziomu izolacji SERIALIZABLE.

W ramach ćwiczenia sprawdzimy działanie transakcji odwołujących się do tych samych danych przy poziomie izolacji transakcji SERIALIZABLE. Ponieważ poziom ten nie jest poziomem domyślnym, to będziemy musieli wydawać poleceń zmieniających poziom izolacji transakcji. Poziom SERIALIZABLE gwarantuje niezmienność wyników polecenia SELECT powtarzanego w ramach transakcji.

W celu zademonstrowania współdziałania transakcji przy poziomie izolacji SERIALIZABLE realizujemy następujące czynności:

- prowadzący kurs otwiera nowe okno edycyjne w bazie danych ElektronicznyDzinnikOcen i wykonuje polecenia:
  - BEGIN TRANSACTION – rozpoczyna transakcję,
  - SET TRANSACTION ISOLATION LEVEL SERIALIZABE – zmienia poziom izolacji transakcji,
  - SELECT * FROM Uczniowie – odczytuje zawartość tabeli),
  - prowadzący w ramach rozpoczętej transakcji nie dokonał żadnej modyfikacji danych, a jedynie odczytał zawartość tabeli.
- słuchacze wykonują polecenie UPDATE Uczniowie SET Idklasy=4 WHERE IdKlasy=5,
- polecenia oczekują na wykonanie, ponieważ są blokowane przez transakcję rozpoczętą przez prowadzącego – poziom izolacji SERILIZABLE gwarantuje, że dane odczytane w ramach transakcji nie ulegną zmianie do jej zakończenia,
- prowadzący zatwierdza swoja transakcję wykonując polecenie COMMIT i polecenia oczekujące u słuchaczy zostają wykonane.

W dalszej części ćwiczenia realizujemy następujące czynności:

- prowadzący kurs otwiera nowe okno edycyjne w bazie danych ElektronicznyDzinnikOcen i wykonuje polecenia:
  - BEGIN TRANSACTION – rozpoczyna transakcję,
  - SET TRANSACTION ISOLATION LEVEL SERIALIZABLE – zmienia poziom izolacji transakcji,
  - SELECT * FROM Uczniowie – odczytuje zawartość tabeli Uczniowie,
  - prowadzący w ramach rozpoczętej transakcji nie dokonał żadnej modyfikacji danych, a jedynie odczytał zawartość tabeli.
- słuchacze wykonują polecenie:
  - INSERT INTO Uczniowie(Nazwisko, Imie,DataUrodzenia,Pesel,CzyChlopak,idklasy) VALUES(‘Nowak’ , ‘Zbigniew’ , ‘1967-12-22’, ‘67122245436’,0,4) (w klauzuli VALUES można zapisać innej dane)
- polecenia nie zostają wykonane, ponieważ zmieniłyby ilość wierszy zwracanych przez zapytanie wykonane w ramach transakcji z poziomem izolacji,
- prowadzący ponawia zapytanie SELECT * FROM Uczniowie WHERE IdKlasy=4 i niezwłocznie otrzymuje wynik identyczny z tym, który otrzymał wcześniej.
- prowadzący zatwierdza swoją transakcję wykonując polecenie COMMIT,
- polecenia INSERT słuchaczy zostają niezwłocznie wykonane – SQL Server anulował blokady na odczytanych danych w ramach transakcji realizowanej w oknie prowadzącego kurs.

W trakcie ćwiczenia zauważyliśmy następujące problemy:

- słuchacze nie mogli dopisywać nowych dane do tabeli,
- ponowione, w ramach transakcji zapytanie zwracało identyczne wyniki.

UWAGA: Ponieważ w ramach tego ćwiczenia wykonywane są operacje na tych samych danych, należy postępować uważnie i nie wykonywać tych poleceń, które demonstruje prowadzący kurs, a jedynie te polecenia, które maja wykonać słuchacze.

Problemy i uwagi do ćwiczenia omówić z prowadzącym kurs.

2.6.6. Ćwiczenie 7 – Zakleszczenia

W ramach ćwiczenia prowadzący zademonstruje sytuację, w której wystąpi zjawisko zakleszczenia. Zadanie może być powtórzone, po pokazie, przez słuchaczy.

W ramach pokazu, prowadzący kurs wykonuje następujące czynności:

- otwiera okno edycyjne (Okno Nr 1), w którym wykonuje polecenie BEGIN TRANSACTION, a następnie polecenie UPDATE Uczniowie SET Idklasy=4 WHERE IDklasy=5 – modyfikacja danych zostaje wykonana niezwłocznie,
- otwiera drugie okno edycyjne (Okno Nr 2), w którym wykonuje polecenie BEGIN TRANSACTION, a następnie polecenie UPDATE Klasy SET Nazwa='VI D' WHERE IdKlasy=7 – polecenie zostaje wykonane niezwłocznie,
- po wykonaniu pierwszych punktów pokazu są dwie nizakonczone transacje, z których jedna dokonała modyfikacji danych w tabeli Uczniowie, a druga w tabeli Klasy,
- w oknie Nr 1 wykonuje polecenie SELECT * From Klasy – polecenie nie zwraca wyników, ponieważ oczekuje na zakończenie transakcji rozpoczętej w oknie Nr 2 – nie mamy do czynienia z zakleszczeniem, ponieważ w oknie Nr 2 nie oczykujemy na wyniki innych transakcji,
- w oknie Nr 2 wykonuje polecenie SELECt * FROM Uczniowie – polecenie nie zwraca wyników, ponieważ oczekuje na zakończenie transakcji rozpoczętej w oknie Nr 1 – mamy do czynienia z zakleszczeniem, ponieważ obie transakcje oczekuje na zakonczenie poleceń, kótre są blokowane przez dwie niedokończone transakcję,
- w ciągu kilku sekund w jednym z okien pojawi się komunikat: „Transaction (Process ID 58) was deadlocked on lock resources with another process and has been chosen as the deadlock victim. Rerun the transaction" informujący, że jedna z transakcji została przerwana przez serwer, a druga dokończyła swoje działanie.

Po pokazie, każdy słuchacz próbuje samodzielnie wywołać zakleszczenie.

Problemy i uwagi do ćwiczenia omówić z prowadzącym kurs.

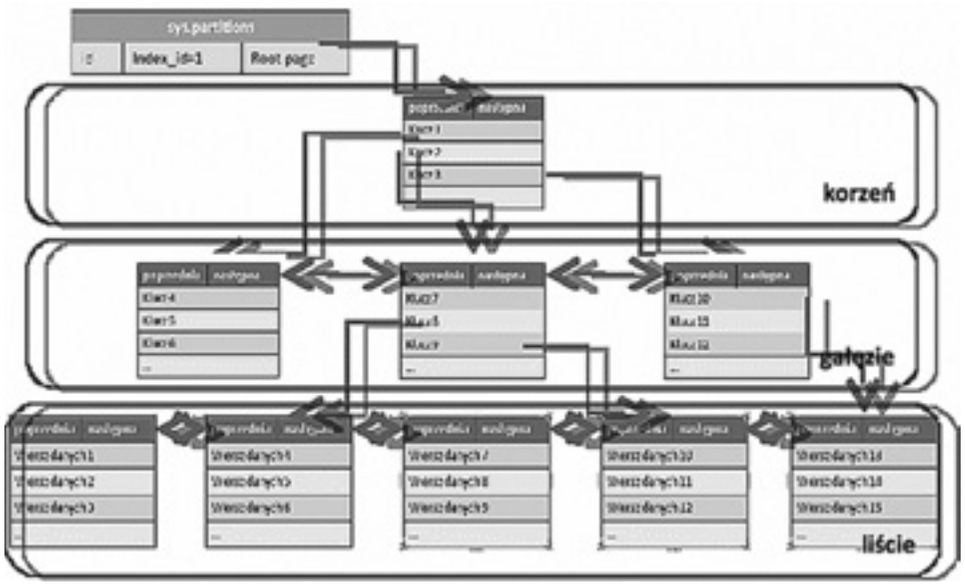
3 INDEKSY.

3.1. INDEKS ZGRUPOWANY.

W rozdziale 1.3 omawialiśmy fizyczną organizację przechowywana danych i w ramach ćwiczenia 1 doszliśmy do wniosku, że dla tabeli zorganizowanej jako sterta, jedyną możliwością wykonania zapytania, jest odczytanie pełnej zawartości tabeli, co przy dużych ilościach danych jest bardzo czasochłonne. Taka operacja nosi nazwę **skanowania tabeli** (ang. *table scan*). Jest ona bardzo kosztowna (w sensie zasobów) i wymaga częstego sięgania do danych z dysku, tym częściej im więcej danych znajduje się w tabeli. Taki mechanizm jest skrajnie nieefektywny, więc muszą istnieć inne, bardziej efektywne mechanizmy wyszukiwania. Mechanizmem zwiększającym wydajność operacji wyszukiwania danych, a tym samym realizacji zapytań są **indeksy**. Indeksy są dodatkową strukturą danych, przechowywanych przez SQL Server. Indeksy występują w dwóch podstawowych wariantach jako indeksy: **zgrupowane** (ang. *clustered*) i **niezgrupowane** (ang. *nonclustered*).

Indeks zgrupowany ma postać drzewa zrównoważonego (*B-tree*). Na poziomie korzenia i gałęzi znajdują się strony indeksu zawierające kolejne wartości klucza indeksu uporządkowane rosnąco. Na poziomie liści znajdują się podobnie uporządkowane strony z danymi tabeli. To właśnie jest cechą charakterystyczną indeksu zgrupowanego – powoduje on fizyczne uporządkowanie wierszy w tabeli, rosnąco według wartości klucza indeksu (wskazanej kolumny). Z tego względu oczywiste jest ograniczenie do jednego indeksu zgrupowanego dla tabeli. Schemat indeksu zgrupowanego pokazano na rysunku 14.

Specyfika indeksu zgrupowanego polega na fizycznym porządkowaniu danych z tabeli według wartości klucza indeksu. W związku z tym jasne jest, że indeks ten będzie szczególnie przydatny przy zapytaniach operujących na zakresach danych, grupujących dane oraz korzystających z danych z wielu kolumn. W takich przypadkach indeks zgrupowany zapewnia znaczny wzrost wydajności w stosunku do sterty lub indeksu niezgrupowanego.



Rysunek 14.
Indeks zgrupowany

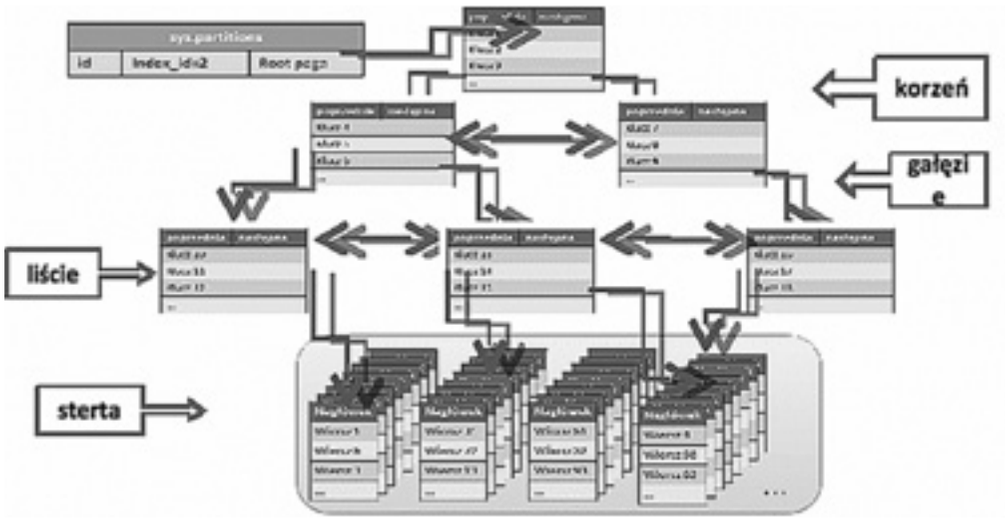
Istotną rzeczą przy podejmowaniu decyzji o utworzeniu indeksu zgrupowanego jest wybranie właściwej kolumny (kolumn). Długość klucza powinna być jak najmniejsza, co umożliwia zmieszczenie większej ilości wpisów indeksu na jednej stronie, co z kolei przenosi się na zmniejszenie liczby stron całości indeksu i w efekcie mniej operacji wejścia/wyjścia do wykonania przez serwer. Żeby indeks zgrupowany korzystnie wpływał na wydajność przy dodawaniu nowych wierszy do mocno wykorzystywanej tabeli, klucz powinien przyjmować dla kolejnych wpisów wartości rosnące (zwykle stosowana jest tu kolumna z cechą IDENTITY). Indeks daje duży zysk wydajności, gdy jego klucz jest możliwie wysoko selektywny (co oznacza mniejszą liczbę kluczy o tej samej wartości – duplikatów). Istotny jest także fakt, że kolumny klucza indeksu zgrupowanego nie powinny być raczej modyfikowane, gdyż pociąga to za sobą konieczność modyfikowania nie tylko stron indeksu, ale także porządkowania stron danych.

Oprócz faktu, że indeks zgrupowany jest formą fizycznej organizacji tabeli (zamiast stery mamy do czynienia z listą dwukierunkową, a dane są fizycznie uporządkowane według wartości klucza) otrzymujemy dodatkowo strukturę drzewa zrównoważonego, dzięki której możemy wyszukiwać wiersze danych dla zadanej wartości klucza indeksu. Korzeń indeksu zawiera zbiór wartości klucza tworząc przedziały wartości. W przypadku wyszukiwania konkretnej wartości klucza odnajdujemy na korzeniu drzewa (jeden odczyt danych) przedział, w którym poszukiwana wartości się znajduje. Oprócz wartości klucza, dla każdego przedziału, zapamiętany jest adres strony na poziomie niższym, w którym znajduje się zbiór wartości klucza z odnalezionego przedziału, tworząc nowe, bardziej precyzyjne przedziały wartości. Proces odczytywania kolejnych gałęzi drzewa jest kontynuowany aż do odnalezienia konkretnej, poszukiwanej wartości klucza. Warto sobie uzmysłowić fakt, że jeżeli na stronie korzenia drzewa będzie się znajdowało 100 przedziałów wartości klucza to kolejny poziom gałęzi będzie zawierał 100 stron po 100 wartości klucza, czyli 10 000 przedziałów, a kolejny 10 000 stron po 100 wartości klucza, czyli 1 000 000 przedziałów. W systemach produkcyjnych, nawet dla bardzo dużych tabeli zawierających setki milionów wierszy, odszukanie konkretnej wartości klucza wymaga nie więcej niż 10 operacji odczytu danych (w ćwiczeniu 1 zapytanie wymagało ok. miliona operacji odczytu).

3.2. INDEKS NIEZGRUPOWANY.

Indeksy zgrupowane nie wyczerpują możliwości budowania tego typu struktur w SQL Server 2008. Drugim typem indeksów są **indeksy niezgrupowane**. Ich budowa odbiega nieco od budowy indeksu zgrupowanego, a do tego indeksy niezgrupowane mogą być tworzone w oparciu o stertę lub istniejący indeks zgrupowany. Dla jednej tabeli można utworzyć do 248 indeksów niezgrupowanych. Indeks niezgrupowany różni się od zgrupowanego przede wszystkim tym, że w swojej strukturze na poziomie liści ma także strony indeksu (a nie strony danych).

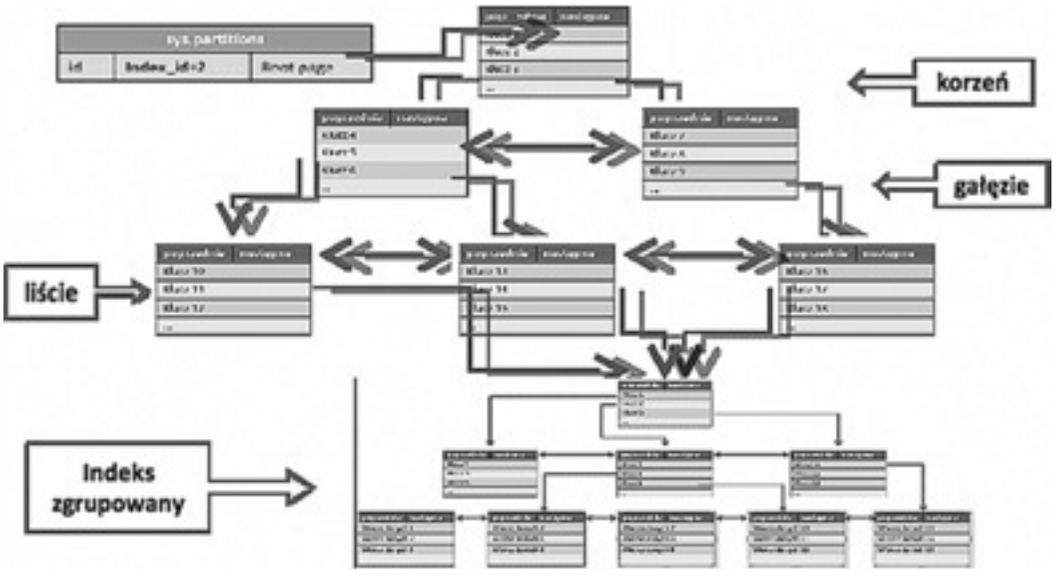
W przypadku budowania indeksu niezgrupowanego na stercie, strony te oprócz wartości klucza indeksu zawierają wskaźniki do konkretnych stron na stercie, które dopiero zawierają odpowiednie dane. Schemat budowy indeksu niezgrupowanego dla sterty pokazano na rysunku 15.



Rysunek 15.
Indeksy niezgrupowane dla tabeli zbudowanej jako sterta

Indeksy niezgrupowane mają strukturę zbliżoną do zgrupowanych. Zasadnicza różnica polega na zawartości liści indeksu. O ile indeksy zgrupowane mają w tym miejscu strony danych, to indeksy niezgrupowane – strony indeksu. Strony te zależnie od wariantu indeksu niezgrupowanego zawierają oprócz klucza różne informacje. Indeksy niezgrupowane mogą być tworzone w oparciu o stertę. Jest to możliwe tylko w przypadku, gdy tabela nie posiada indeksu zgrupowanego. W takim przypadku liście indeksu zawierają wskaźniki do konkretnych stron na stercie.

Indeks niezgrupowany tworzony na tabeli zawierającej już indeks zgrupowany, jest tworzony nieco inaczej. Korzeń, gałęzie i liście zawierają strony indeksu, ale liście zamiast wskaźników do stron na stercie zawierają wartości klucza indeksu zgrupowanego. Każde wyszukanie w oparciu o indeks niezgrupowany, po dojściu do poziomu liści, zaczyna dalsze przetwarzanie od korzenia indeksu zgrupowanego (wyszukiwany jest klucz zawarty w liściu).



Rysunek 16.
Indeksy niezgrupowane – dla tabeli zawierającej indeks zgrupowany

W przypadku budowania indeksów niezgrupowanych, szczególnie na dużych tabelach, warto dobrze zaplanować tę czynność, szczególnie gdy planowane jest też utworzenie indeksu zgrupowanego. Niewzięcie tego pod uwagę może powodować konieczność przebudowywania indeksów niezgrupowanych w związku z dodaniem lub usunięciem indeksu zgrupowanego.

W sporym uproszczeniu rola indeksów sprowadza się do ograniczenia liczby operacji wejścia/wyjścia niezbędnych do realizacji zapytania. SQL Server nie odczytuje poszczególnych obszarów potrzebnych do realizacji zapytania z dysku za każdym razem. Zawiera rozbudowany bufor pamięci podręcznej, do której trafiają kolejne odczytywane z dysku obszary. Ze względu na ograniczony rozmiar bufora, strony nieużywane lub używane rzadziej są zastępowane tymi, z których zapytania korzystają częściej.

3.3. INDEKSY POKRYWAJĄCE.

Przy korzystaniu z indeksów niezgrupowanych istnieje jeszcze jedna możliwość dalszego ograniczania liczby operacji wejścia/wyjścia. Polega ona na tym, że do indeksu (dokładnie do stron liści indeksu) dodawane są dodatkowe kolumny. Jeżeli liście indeksu niezgrupowanego zawierają wszystkie kolumny zwracane przez zapytanie, to nie ma w ogóle konieczności sięgania do stron z danymi. W takim przypadku mamy do czynienia z tak zwanym **indeksem pokrywającym**. Dodawanie kolumn do indeksu niezgrupowanego może polegać na dodawaniu kolejnych kolumn do klucza (występuje tu ograniczenie do 16 kolumn w kluczu i 900 bajtów długości klucza) albo na dodawaniu kolumn „niekluczowych" do indeksu (nie wliczają się one do długości klucza). Trzeba jednak pamiętać, że tworzenie indeksów pokrywających dla kolejnych zapytań nie prowadzi do niczego dobrego, gdyż po pierwsze rośnie ilość danych (wartości kolumn są przecież kopiowane do stron indeksu), a po drugie drastycznie spada wydajność modyfikowania danych (pociąga za sobą konieczność naniesienia zmian we wszystkich indeksach).

Mechanizm indeksów pokrywających jest pomocny przy realizacji wielu zapytań i nie jest trudny w zastosowaniu. Jest jednak druga strona medalu. Zwykle zapytań jest więcej niż jedno i zwracają więcej kolumn. Rozbudowywanie indeksów (zarówno ich liczba, jak i liczba kolumn w nich zawartych) prowadzi do znacznego wzrostu rozmiaru bazy danych oraz spadku wydajności przy modyfikowaniu danych z tej tabeli. W skrajnych przypadkach tworzymy przecież kopie poszczególnych wierszy na stronach indeksu, a co za tym idzie ilość operacji wejścia/wyjścia staje się zbliżona do tej potrzebnej do skanowania tabeli czy indeksu zgrupowanego.

3.4. PLANY WYKONANIA ZAPYTAŃ.

Język SQL jest językiem deklaratywnym, czyli pisząc polecenie w tym języku określamy, co chcemy osiągną, a nie jak to zrobić. Gdy przekazujemy serwerowi wykonanie zapytania rozpoczyna się dość złożony proces prowadzący do określenia sposobu realizacji zapytania. Zależnie od konstrukcji samego zapytania, rozmiarów tabel, istniejących indeksów, statystyk itp. serwer tworzy kilka planów wykonania zapytania. Następnie spośród nich wybierany jest ten, który cechuje się najniższym kosztem wykonania (wyrażanym przez koszt operacji wejścia/wyjścia oraz czasu procesora). Tak wybrany plan jest następnie kompilowany (przetwarzany na postać gotową do wykonania przez silnik bazodanowy) i przechowywany w buforze w razie gdyby mógłby się przydać przy kolejnym wykonaniu podobnego zapytania. W ramach tego punktu zajmiemy się nieco dokładniej procesem wykonania zapytania przez SQL Server.

Cały proces, przebiegający od momentu przekazania zapytania do wykonania do odebrania jego rezultatów, jest dość złożony i może stanowić temat niejednego wykładu. Postaramy się choć z grubsza zasygnalizować najistotniejsze etapy tego procesu.

- **Parsowanie zapytania** – polega na zweryfikowaniu składni polecenia, wychwyceniu błędów i nieprawidłowości w jego strukturze. Jeżeli takie błędy nie występują, to efektem parsowania jest tak zwane **drzewo zapytania** (postać przeznaczona do dalszej obróbki).
- **Standaryzacja zapytania**. Na tym etapie drzewo zapytania jest doprowadzane do postaci standardowej – usuwana jest ewentualna nadmiarowość, standaryzowana jest postać podzapytań itp. Efektem tego etapu jest ustandaryzowane drzewo zapytania.
- **Optymalizacja zapytania**. Polega na wygenerowaniu kilku planów wykonania zapytania oraz przeprowadzeniu ich analizy kosztowej zakończonej wybraniem najtańszego planu wykonania.
- **Kompilacja** polega na przetłumaczeniu wybranego planu wykonania do postaci kodu wykonywalnego przez silnik bazodanowy.

- **Określenie metod fizycznego dostępu do danych** (skanowanie tabel, skanowanie indeksów, wyszukiwanie w indeksach itp.).

Proces optymalizacji zapytania składa się z kilku etapów. W ich skład wchodzą: analizowanie zapytania pod kątem kryteriów wyszukiwania i złączeń, dobranie indeksów mogących wspomóc wykonanie zapytania, oraz określenie sposobów realizacji złączeń. W ramach realizacji poszczególnych etapów optymalizator zapytań może korzystać z istniejących statystyk indeksów, generować je dla wybranych indeksów lub wręcz tworzyć nowe indeksy na potrzeby wykonania zapytania. Efektem tego procesu jest plan wykonania o najniższym koszcie, który jest następnie przekazywany do kompilacji i wykonania. Plan wykonania dla zapytania można podejrzeć w formie tekstowej, XML bądź zbioru wierszy. Realizuje się to za pomocą ustawienia na „ON" jednej z opcji SHOWPLAN_TEXT, SHOWPLAN_XML, SHOWPLAN_ALL. SQL Server, a właściwie narzędzie SQL Server Management Studio, umożliwia podejrzenie graficznej reprezentacji planu wykonania dla zapytania

Opcja prezentacji graficznej postaci planu wykonania dla zapytania jest dostępna w dwóch wariantach: *Estimated Execution Plan* oraz *Actual Execution Plan*. Pierwszy z nich polega na wygenerowaniu planu wykonania dla zapytania bez jego wykonywania. Powoduje to, że część informacji w planie wykonania jest szacunkowa lub jej brakuje (np. liczba wierszy poddanych operacjom, liczba wątków zaangażowanych w wykonanie itp.). Zaletą tego wariantu jest na pewno szybkość działania. Jest to szczególnie odczuwalne przy zapytaniach, które wykonują się dłużej niż kilkanaście sekund...

Drugi wariant zawiera pełne dane na temat wykonania zapytania. Jest on zawsze wiarygodny i mamy gwarancję, że dokładnie tak zostało wykonane zapytanie. W praktyce lepiej jest pracować z faktycznymi planami wykonania, chyba że czas potrzebny ich uzyskanie jest przeszkodą.

Na diagramach reprezentujących plany wykonania zapytań może znajdować się kilkadziesiąt różnych symboli graficznych reprezentujących różne operatory (logiczne i fizyczne) oraz przebieg wykonania zapytania. Nie sposób omówić ich choćby pobieżnie w ramach tego kursu. Wśród całej gamy informacji wyświetlanych w szczegółach wybranego operatora, dla nas najistotniejsze są te, związane z kosztem wykonania danego etapu. Przy realizacji ćwiczeń będziemy się na nich opierać prezentując zmiany kosztu wykonania zapytania w zależności od podjętych kroków przy optymalizacji zapytania.

3.5. STATYSTYKI INDEKSÓW.

Sam fakt istnienia takiego czy innego indeksu nie powoduje, że od razu staje się on kandydatem do skorzystania w ramach realizacji zapytania. W trakcie optymalizacji zapytania potrzebne są dodatkowe informacje na temat indeksów – **statystyki indeksów**. Sensowność skorzystania z indeksu można ocenić tylko w połączeniu z informacjami o liczbie wierszy w tabeli oraz o rozkładzie wystąpień poszczególnych wartości lub zakresów wartości w danych zawartych w kolumnie. Przykładowo mamy tabele klientów, w której 80% klientów ma siedzibę w Warszawie, a jedynie dwóch ma siedzibę w Sopocie. Na podstawie samego faktu istnienia indeksu na kolumnie określającej miejscowość trudno ocenić, czy sensownie jest go wykorzystać przy wyszukiwaniu klientów z Warszawy lub Sopotu. Po przejrzeniu statystyk może okazać się, że dla Warszawy nie ma co zaprzątać sobie głowy indeksami, natomiast w przypadku Sopotu może to znacznie poprawić wydajność. Ponieważ dane zawarte w tabelach zwykle się zmieniają (pojawiają się nowe, istniejące są modyfikowane lub usuwane), istotne jest także aktualizowanie statystyk. Optymalizator zapytań podejmujący decyzje na podstawie nieaktualnych statystyk działa jak pilot samolotu, któremu przyrządy pokładowe pokazują wskazania sprzed pięciu minut. Skutki mogą być opłakane. Z tego powodu, jeżeli mamy do czynienia z sytuacją, gdy do tej pory zapytanie wykonywało się zadowalająco szybko, a nagle wydajność spadła, pierwszym krokiem do wykonania jest właśnie uaktualnienie statystyk. Warto o tym pamiętać, bo może to nam oszczędzić sporo czasu.

3.6. ĆWICZENIA

3.6.1. Ćwiczenie 8. Analiza wydajności zapytania – dla różnych ilości danych.

W ramach ćwiczenia poddamy analizie przykładowe zapytanie i sprawdzimy strategie jego wykonania w zależności od ilości zwracanych wierszy. W przykładowej bazie danych istnieje tabela o nazwie DostawyPrasy o strukturze pokazanej na rysunku 17.

Column Name	Data Type	Allow Nulls
IdDostawy	int	☐
IdTytułu	int	☐
NumerWydania	char(16)	☐
DataDostawy	smalldatetime	☐
Ilosc	int	☐
InformacjeDodatkowe	xml	☐
Cena	money	☐
uwagi	char(2000)	☑
czy_przeniesiono	bit	☐

Rysunek 17.
Struktura tabeli DostawyPrasy

Przeznaczenie tej tabeli nie jest istotne z punktu widzenia naszych ćwiczeń, a istotnym jest fakt, że jest w niej zapisanych ok. 100 000 wierszy.

Tabela jest zorganizowana w formie indeksu zgrupowanego dla kolumny IdDostawy i jest utworzony indeks niezgrupowany dla kolumny DataDostawy.

Słuchacze wykonują następujące czynności:

- Otwieramy nowe okno edycyjne dla bazy danych ElektronicznyDziennikOcen,
- Piszemy polecenie SET STATISTICS IO ON – które spowoduje wyświetlanie, po wykonaniu zapytania, statystyk operacji wejścia-wyjścia,
- W menu głównym wybieramy opcję Incude Actual Excecution Plan, jak pokazano na rysunku 18, która spowoduje, że po wykonaniu zapytania zostanie wyświetlony plan jego wykonania,

Rysunek 18.
Wybór opcji Incude Actual Excecution Plan

- W oknie edycyjnym piszemy i wykonujemy następujące zapytanie:

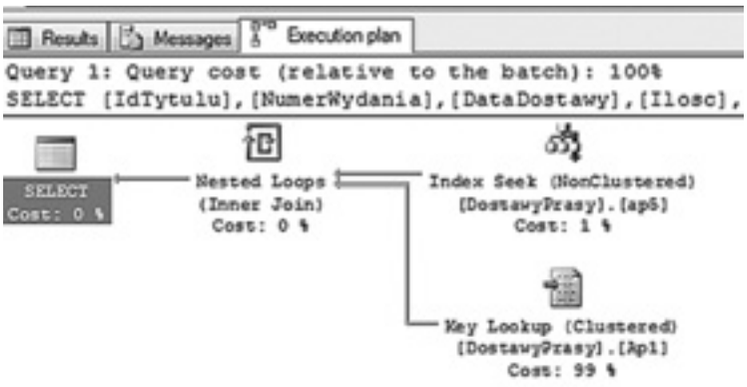
```
SELECT  IdTytulu,
        NumerWydania,
        DataDostawy,
        Ilosc,
        Cena
FROM DostawyPrasy
WHERE  DataDostawy BETWEEN ‚20010201’ AND ‚20010202’ ,
```

- Przykładowy fragment wyniku zapytania pokazano na rysunku 19 – dla przykładowych danych zapytanie zwraca 55 wierszy,

IdTytulu	NumerWydania	DataDostawy	Ilosc	Cena
1849	2	2001-02-01 00:00:00	26722	1,00
1908	3	2001-02-01 00:00:00	21648	6,00
1158	3	2001-02-01 00:00:00	40182	8,00
10	3	2001-02-01 00:00:00	44768	7,00
198	2	2001-02-01 00:00:00	27794	6,00
90	2	2001-02-01 00:00:00	41706	8,00
1139	1	2001-02-01 00:00:00	10876	4,00
746	1	2001-02-01 00:00:00	6961	5,00
687	2	2001-02-01 00:00:00	26744	9,00
336	3	2001-02-01 00:00:00	17435	5,00

Rysunek 19.
Przykładowy fragment wyniku zapytania

- W oknie Message zwrócony został komunikat ze statystykami operacji wejścia-wyjścia: „Table ‚DostawyPrasy’. Scan count 1, logical reads 264, physical reads 0, read-ahead reads 24, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.”, z którego wynika, że przy realizacji zapytania wykonane zostały 264 operacje odczytu danych,
- W oknie Execution Plan pokazany został plan wykonania zapytania, którego postać pokazano na rysunku 20.



Rysunek 20.
Plan wykonania zapytania

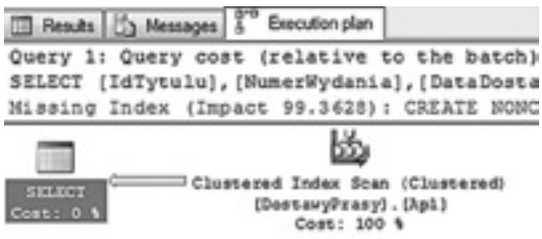
Postać planu zapytania możemy zinterpretować następująco:

- wyszukujemy w indeksie (dla kolumny DataDostawy) wartości klucza z przedziału podanego w klauzuli WHERE zapytania – operator Index Seek,
- dla każdego wyszukanego wiersza (operator Nested Loop) odnajdujemy odpowiedni wiersz danych w indeksie zgrupowanym (operator Key Lookup),
- dla odnalezionych danych zwracamy wynik zapytania (operator SELECT)

Działanie, na podstawie tego planu wykonania wydaje się logiczne, ponieważ istnieje indeks niezgrupowany dla kolumny DataDostawy, to korzystamy z tego indeksu w celu szybkiego odnalezienia odpowiednich wierszu, a następnie musimy ze stron, na których zapisane są dane tabeli odczytać wartości z tych kolumn, które są zwracane w poleceniu SELECT.

W dalszej części ćwiczenia realizujemy następujące czynności:

- modyfikujemy postać zapytania, zmieniając datę końcową na '20011202' – zapytanie będzie miało postać:
  SELECT IdTytulu,
  NumerWydania,
  DataDostawy,
  Ilosc,
  Cena
  FROM DostawyPrasy
  WHERE DataDostawy BETWEEN ,20010201' AND ,20010202' ,
- wykonujemy zapytanie,
- w oknie Message otrzymujemy komunikat: „Table ,DostawyPrasy'. Scan count 1, logical reads 34 030, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0." – czyli do wykonania zapytanie potrzebna była realizacja ok. 34 000 operacji odczytu danych,
- w oknie Execution Plan pokazany zostanie plan wykonania zapytania, pokazany na rysunku 21.



Rysunek 21.
Plan wykonania zapytania po modyfikacji granicy przedziału

Wnioski:
- SQL Server uzależnił strategie wykonania zapytania od wielkości przedziału dat, z którego pobieraliśmy dane,
- Decyzja o zmianie strategii została podjęta po analizie, w procesie budowy planu wykonania, statystyk indeksy DataDostawy,
- Optymalizator uznał, że gdy zwracana ilość wierszy jest duża, to konieczność wykonywania operacji Key Lookup jest zbyt kosztowna i zdecydował na operację odczytania całej tabeli (operator Clustered Index Scan).

Problemy i uwagi do ćwiczenia omówić z prowadzącym kurs.

3.6.2. Ćwiczenie 9. Analiza wydajności zapytania – dla indeksu pokrywającego.

W ramach ćwiczenia poddamy analizie przykładowe zapytanie i sprawdzimy strategie jego wykonania w zależności od istnienia indeksu pokrywającego dla tego zapytania. Zapytanie kierujemy do tabeli DostawyPrasy (wykorzystywana w ramach ćwiczenia 8).

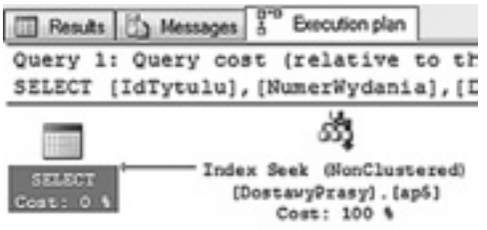
Tabela jest zorganizowana w formie indeksu zgrupowanego dla kolumny IdDostawy i jest utworzony indeks niezgrupowany dla kolumny DataDostawy.

Słuchacze wykonują następujące czynności:
- Otwieramy nowe okno edycyjne dla bazy danych ElektronicznyDziennikOcen,
- Piszemy polecenie SET STATISTICS IO ON – które spowoduje wyświetlanie, po wykonaniu zapytania, statystyk operacji wejścia-wyjścia,

- W menu głównym wybieramy opcję Incude Actual Excecution Plan, jak pokazano na rysunku 18, która spowoduje, że po wykonaniu zapytania zostanie wyświetlony plan jego wykonania,
- W oknie edycyjnym piszemy i wykonujemy następujące zapytanie:

```
SELECT  IdTytulu,
        NumerWydania,
        DataDostawy,
        Ilosc,
        Cena
FROM DostawyPrasy
WHERE  DataDostawy BETWEEN ‚20010201’ AND ‚20010202’ ,
```

- W oknie Message zwrócony został komunikat ze statystykami operacji wejścia-wyjścia: „Table ‚DostawyPrasy’. Scan count 1, logical reads 264, physical reads 0, read-ahead reads 24, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.”, z którego wynika, że przy realizacji zapytania wykonane zostały 264 operacje odczytu danych,
- W oknie Execution Plan, pokazany został plan wykonania zapytania, którego postać pokazano na rysunku 20 (analogicznie jak w ramach ćwiczenia 8),
- Modyfikujemy istniejący indeks dodając do niego dodatkowe kolumny (IdTytulu, NumerWydania, Ilość i Cena) tworząc indeks pokrywający, gdyż wszystkie dane potrzebne do wykonania zapytania będą znajdować się w indeksie,
- Wykonujemy ponownie zapytanie,
- W oknie Message otrzymujemy komunikat: „Table ‚DostawyPrasy’. Scan count 1, logical reads 5, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.”, z którego wynika, że do realizacji zapytania niezbędne było tylko 5 operacji odczytu danych,
- W oknie Execution Plan wyświetlony zostanie plan wykonania pokazany na rysunku 22.



Rysunek 22.
Plan wykonania dla indeksu pokrywającego

Działanie, na podstawie tego planu wykonania wydaje się logiczne, ponieważ istnieje indeks niezgrupowany dla kolumny DataDostawy i dodatkowo indeks ten zawiera wszystkie kolumny potrzebne do realizacji zapytania, więc nie ma konieczności przeprowadzania operacji KeyLookup (jak w pierwszej części ćwiczenia).

W dalszej części ćwiczenia realizujemy następujące czynności:

- modyfikujemy postać zapytania, zmieniając datę końcową na ‘20011202’ – zapytanie będzie miało postać:

```
SELECT  IdTytulu,
        NumerWydania,
        DataDostawy,
        Ilosc,
        Cena
FROM DostawyPrasy
WHERE  DataDostawy BETWEEN ‚20010201’ AND ‚20010202’ ,
```

- wykonujemy zapytanie,
- W oknie Message otrzymujemy komunikat: „Table ‚DostawyPrasy’. Scan count 1, logical reads 60, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.”, z którego wynika, że do realizacji zapytania niezbędne było tylko 60 operacji odczytu danych,
- W oknie Execution Plan, otrzymujemy taki sam plan wykonania (rysunek 20),

Wnioski:

- W przypadku istnienia indeksu pokrywającego serwer nie zmienia strategii wykonania dla dużego przedziału wartości klucza, ponieważ nie musi już wykonywać operacji dodatkowego odczytu danych (Key Lookup)
- Widać, że indeksy pokrywające bardzo dobrze pomagają przy optymalizacji zapytań.

Problemy i uwagi do ćwiczenia omówić z prowadzącym kurs.

3.6.3. Ćwiczenie 10. Optymalizacja przykładowego zapytania.

W ramach ćwiczenia porównamy koszty wykonania takiego samego zapytania dla różnych indeksów. Porównywać będziemy ilość operacji wejścia-wyjścia i całkowity koszt realizacji zapytania (w celu określenia stopnia optymalizacji).

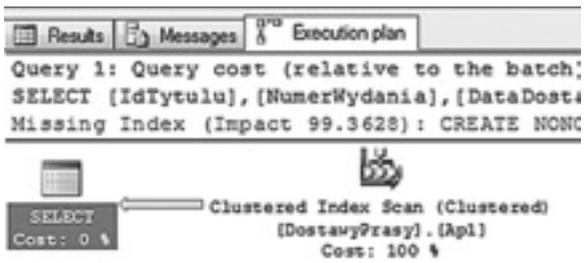
Tabela DostawyPrasy (rysunek 17) jest zorganizowana w formie indeksu zgrupowanego dla kolumny IdDostawy i nie ma innych indeksów zdefiniowanej dla tej tabeli.

Słuchacze wykonują następujące czynności:

- Otwieramy nowe okno edycyjne dla bazy danych ElektronicznyDziennikOcen,
- Piszemy polecenie SET STATISTICS IO ON – które spowoduje wyświetlanie, po wykonaniu zapytania, statystyk operacji wejścia-wyjścia,
- W menu głównym wybieramy opcję Incude Actual Excecution Plan, jak pokazano na rysunku 18, która spowoduje, że po wykonaniu zapytania zostanie wyświetlony plan jego wykonania,
- W oknie edycyjnym piszemy i wykonujemy następujące zapytanie:

```
SELECT  IdTytulu,
        NumerWydania,
        DataDostawy,
        Ilosc,
        Cena
FROM DostawyPrasy
WHERE  DataDostawy BETWEEN ‚20010201’ AND ‚20010202’ ,
```

- W oknie Message zwrócony został komunikat ze statystykami operacji wejścia-wyjścia: „Table ‚DostawyPrasy’. Scan count 1, logical reads 34030, physical reads 0, read-ahead reads 24, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.”, z którego wynika, że przy realizacji zapytania wykonane zostało 34030 operacji odczytu danych,
- W oknie Execution Plan, pokazany został plan wykonania zapytania, którego postać pokazano na rysunku 23,



Rysunek 23.
Plan wykonania przy braku indeksów

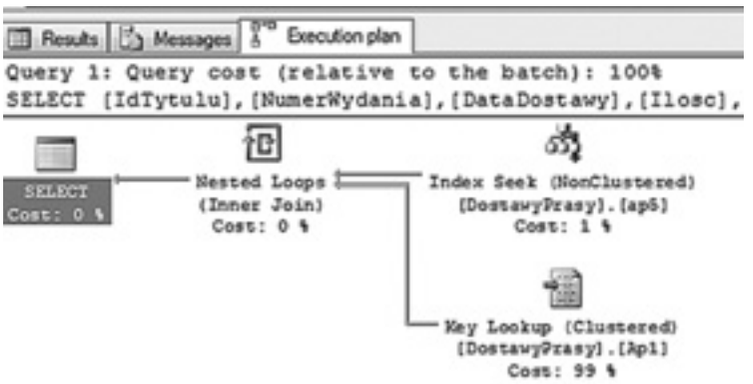
- Odczytujemy koszt wykonania zapytania, w tym celu ustawiamy wskaźnik myszy na operatorze SELECT, pojawi się dodatkowe okno ze szczegółami, jak pokazano na rysunku 24,

- Z okna, pokazanego na rysunku 24, odczytujemy wartość Estimated Subtree Cost, która w tym wypadku określa całkowity koszt wykonania zapytania.

Rysunek 24.
Okno szczegółów operatora SELECT

Wynik 1 – przy braku indeksów:
Ilość odczytów – 34030
Koszt zapytania – 25,2798.

- Prowadzący tworzy, w tabeli DostawyPrasy, indeks niezgrupowany dla kolumny DataDostawy,
- Wykonujemy ponownie zapytanie i w analogiczny sposób odczytujemy parametry kosztu zapytania,
- Plan wykonania zapytania, po zdefiniowaniu indeksu dla kolumny DataDostawy, pokazano na rysunku 25,

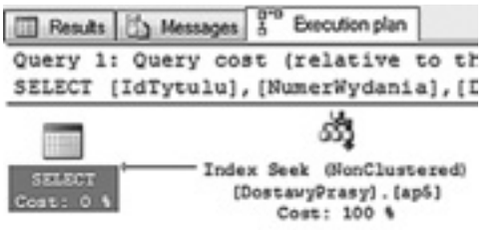


Rysunek 25.
Plan wykonaniu po zdefiniowaniu indeksu dla kolumny DataDostawy

Wynik 2 – dla indeksu niezgrupowanego dla kolumny DataDostawy:
Ilość odczytów – 179
Koszt zapytania – 0,42642

Porównanie zestawu wyników 1 i 2 pokazuje, że utworzenie indeksu przyspieszyło wykonanie zapytania ponad 50 razy (koszt z wartości 25,2798 zmniejszył się do 0,42642).

- W kolejnym kroku prowadzący tworzy indeks pokrywający dla przykładowego zapytania, dodając do istniejącego indeksu dodatkowe kolumny.
- Wykonujemy ponownie zapytanie i w analogiczny sposób odczytujemy parametry kosztu zapytania,
- Plan wykonania zapytania, po zdefiniowaniu indeksu pokrywającego, pokazano na rysunku 26,



Rysunek 26.
Plan wykonaniu po zdefiniowaniu indeksu pokrywającego

Wynik 3 – dla indeksu pokrywającego:
Ilość odczytów – 4
Koszt zapytania – 0,0034248

Porównanie zestawu wyników 1 i 3 pokazuje, że utworzenie indeksu przyspieszyło wykonanie zapytania ponad 8000 razy (koszt z wartości 25,2798 zmniejszył się do 0,0034248).

Problemy i uwagi do ćwiczenia omówić z prowadzącym kurs.

4 OPTYMALIZACJA PRZETWARZANIA DANYCH.

4.1. ĆWICZENIA.

4.1.1. Ćwiczenie 11 – Analiza porównawcza różnych sposobów przetwarzania.

Ważnym elementem w procesie optymalizacji jest sposób, w jaki wykonujemy przetwarzanie danych. SQL Server optymalizuje każde polecenie i buduje plany wykonania. W wielu problemach rozwiązywanych w bazie danych pojawia się pokusa przetwarzania iteracyjnego (wiersz po wierszu). W ramach ćwiczenia porównamy przetwarzanie realizowane wiersz po wierszy z przetwarzaniem (realizującym to samo zadanie) wykonanym za pomocą jednego polecenia SQL.

Opis problemu:

- W przykładowej bazie danych znajduje się tabela o nazwie Osoby o strukturze pokazanej na rysunku 27.

Column Name	Data Type	Allow Nulls
Nazwisko	nvarchar(128)	☐
imie	nvarchar(64)	☐
DataUrodzenia	date	☐
CzyKobieta	bit	☑
id	int	☐
Pesel	char(11)	☑

Rysunek 27.
Struktura tabeli Osoby

- Tabela zawiera dane 495260 osób urodzonych w roku 1953 (dane są fikcyjne),
- Kolumna Pesel, w tabeli Osoby, zawiera wartości null,
- Naszym zadaniem jest utworzenie unikalnych numerów Pesel dla każdej osoby zapisanej w tabeli pamiętając o tym, że pierwsze sześć cyfr numeru Pesel odwzorowuje datę urodzenia, przedostatnia cyfra określa płeć (parzysta kobiety, nieparzysta mężczyźni), a ostatnia cyfra jest wynikiem algorytmu obliczania sumy kontrolnej,
- W pierwszym podejściu napiszemy skrypt realizujący następujący algorytm:
  - Odczytujemy wiersz z tabeli osoby,
  - Budujemy numer Pesel,

- Modyfikujemy odczytany wiersz wstawiając do kolumny Pesel zbudowany numer,
- Powtarzamy czynności dla każdego wiersza z tabeli Osoby.

- Otwieramy okno edycyjne,
- W oknie edycyjnym piszemy polecenia:

```
DECLARE @licznik int=1,
            @Data date
WHILE @licznik‹495260
Begin
 SELECT @Data=DataUrodzenia,
          @Plec=CzyKobieta
  FROM Osoby
  WHERE ID=@licznik
   UPDATE Osoby SET Pesel=CONVERT (varchar(6), @data,12) WHERE id=@licznik
  SET @licznik=@licznik
End
```

Powyzszy program realizuje nasze zadanie w ograniczonym zakresie (numer Pesel został ograniczony tylko do sześciu cyfr odwzorowujacych date urodzenia).

- Wykonujemy program zapisany w oknie edycyjnym,
- W trakcie testowania tego programu wykonywał się około 6 minut (budowa numeru Pesel jest niepełna),
- W kolejnym kroku wykonamy to samo zadanie, ale w pełnym zakresie za pomocą polecenia SQL,
- Otwieramy nowe okno edycyjne,
- W oknie edycyjnym piszemy następujące polecenie:

```
DECLARE @DataOd date='19530101', @DataDO date='19531231';

WITH CT1 as
(
select
     NTILE(5) OVER (PARTITION BY DataUrodzenia,CzyKobieta ORDER BY id ) AS gr, pesel,
     id,
     DataUrodzenia,
     CzyKobieta
FROM Osoby
WHERE DataUrodzenia BETWEEN @dataOd AND @DataDo
), CT2 AS
(
SELECT ROW_NUMBER() OVER ( PARTITION  BY DataUrodzenia,CzyKobieta,gr ORDER BY id ) as Lp,
     id,
     DataUrodzenia,
     CzyKobieta,
     gr,
    CASE
      WHEN CzyKobieta=1 THEN (gr-1)*2
      ELSE (gr-1)*2+1
     END AS Post

FROM CT1
), CT3 AS
(
SELECT *, CASE
         WHEN DataUrodzenia‹'20000101'  THEN   CONVERT(varchar(6),CAST(DataUrodzenia AS date) ,12)
         ELSE RIGHT(,00'+cast(YEAR(DataUrodzenia)-2000 AS varchar(2)),2)+
             CAST(MONTH(DataUrodzenia)+20 as varchar(2))+
```

```
            RIGHT(,0'+CAST(DAY(DataUrodzenia) as varchar(2)),2)
        END
       +RIGHT(,00'+CAST(lp as varchar(3)),3)
       +CAST(post as varchar(1)) as PeselP
FROM CT2
), CT4 AS
(
SELECT id,PeselP+dbo.SumaKontrolnaPesel(PeselP) AS tmp
FROM CT3
)
UPDATE OSOBY
SET Pesel=tmp
FROM  ct4
WHERE ct4.Id=Osoby.idPodsumowanie
```

- Wykonujemy polecenie zapisane w oknie edycyjnym,
- W trakcie testowania tego polecenia wykonywało się około 15 sekund (pełna budowa numeru pesel łącznie z obliczeniem sumy kontrolnej).

Podsumowanie ćwiczenia:

- Zaprezentowane zostały dwa podejścia do przetwarzania danych,
- Przetwarzanie iteracyjne (pierwszy program) wykonywało się długo (około 6 minut) pomimo tego, że nie realizowało całego algorytmu,
- Przetwarzanie oparte na zbiorach zostało zrealizowane w ramach jednego polecenia SQL z wykorzystaniem wyrażeń CTE realizowało pełny zakres budowy numeru Pesel – wykonywało się bardzo szybko (około 15 sekund).

Zadanie dla słuchaczy:

- Omówić z prowadzącym kursu szczegóły obu rozwiązań,
- Uzupełnić przetwarzanie iteracyjne tak, żeby wykonywało pełny zakres budowy numeru Pesel i przetestować czas wykonania.
- Wszelkie problemu i uwagi do ćwiczenia omówić z prowadzącym kurs.

6 PODSUMOWANIE

W ramach tego kursu zaledwie rozpoczęliśmy omawianie zagadnień związanych z optymalizacją zapytań i optymalizacją wydajności SQL Servera jako taką. Celem było przedstawienie pewnych podstawowych zagadnień i mechanizmów niezbędnych do zrozumienia podstawowych zasad w dziedzinie sposobów realizacji zapytań i ich optymalizacji przez SQL Server.

7 LITERATURA

1. Ben-Gan I., Kollar L., Sarka D., *MS SQL Server 2005 od środka: Zapytania w języku T-SQL,* APN PROMISE, Warszawa 2006
2. Coburn R., *SQL dla każdego*, Helion, Gliwice 2001
3. Rizzo T., Machanic A., Dewson R., Walters R., Sack J., Skin J., *SQL Server 2005*, WNT, Warszawa 2008
4. Szeliga M., *ABC języka SQL*, Helion, Gliwice 2002
5. Vieira R., *SQL Server 2005. Programowanie. Od Podstaw*, Helion, Gliwice 2007

W projekcie **Informatyka +**, poza wykładami i warsztatami, przewidziano następujące działania:

- 24-godzinne kursy dla uczniów w ramach modułów tematycznych
- 24-godzinne kursy metodyczne dla nauczycieli, przygotowujące do pracy z uczniem zdolnym
- nagrania 60 wykładów informatycznych, prowadzonych przez wybitnych specjalistów i nauczycieli akademickich
- konkursy dla uczniów, trzy w ciągu roku
- udział uczniów w pracach kół naukowych
- udział uczniów w konferencjach naukowych
- obozy wypoczynkowo-naukowe.

Szczegółowe informacje znajdują się na stronie projektu

www.informatykaplus.edu.pl